

Identify Instrumentor User Guide

March 2015

<https://solvnet.synopsys.com>

SYNOPSYS[®]

Copyright Notice and Proprietary Information

© 2015 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only.

Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

“This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____.”

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.

Registered Trademarks (®)

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

Trademarks (™)

AFGen, Apollo, ARC, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIMplus, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, Total-Recall, TSUPREM-4, VCSi, VHDL Compiler, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Service Marks (sm)

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

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A
March 2015

Contents

Chapter 1: Design Setup

Instrumenting and Saving a Design	7
---	---

Chapter 2: IICE Configuration

Multiple IICE Units	10
Adding an IICE Unit	10
Deleting an IICE Unit	10
Common IICE Parameters	11
Device Family	11
Communication Port	12
Board Type	12
Use Skew-Resistant Hardware	12
Prepare Incremental	13
Individual IICE Parameters	14
IICE Sampler Tab	14
IICE Controller Tab	19
HAPS Settings Tab	21

Chapter 3: HAPS Deep Trace Debug

External Memory Instrumentation and Configuration	23
Running Deep Trace Debug with DDR3 Memory	26
Running Deep Trace Debug with SRAM Memory	27
Hardware Configuration Verification	30

Chapter 4: Support for Instrumenting HDL

VHDL Instrumentation Limitations	32
Verilog Instrumentation Limitations	34
SystemVerilog Instrumentation Limitations	37

Chapter 5: Using the Instrumentor

Instrumentor Windows	41
Instrumentation Window	42
Project Window	45
Console Window	46
Commands and Procedures	47
Opening Designs	47
Executing Script Files	48
Selecting Signals for Data Sampling	49
Instrumenting Buses	51
Partial Instrumentation	53
Multiplexed Groups	55
Sampling Signals in a Folded Hierarchy	56
Instrumenting the Verdi Signal Database	58
Instrumenting Signals Directly in the idc File	58
Selecting Breakpoints	60
Selecting Breakpoints Residing in Folded Hierarchy	61
Configuring the IIICE	63
Real-time Debugging	63
Writing the Instrumented Design	66
Synthesizing Instrumented Designs	68
Listing Signals	69
Searching for Design Objects	70
Console Text	72

CHAPTER 1

Design Setup

After your HDL is successfully created, the instrumentor is used to define the specific signals to be monitored. Saving the instrumented design generates an *instrumentation design constraints* (idc) file and adds constraint files to the HDL source for the instrumented signals and break points. The design is synthesized and then run through the remainder of the process. After the device is programmed with the debuggable design, the debugger is launched to debug the design while it is running in the target system. For information on using the debugger, see the *Identify Debugger User Guide*.

The information required to instrument a design includes references to the HDL design source, the user-selected instrumentation, the settings used to create the Intelligent In-Circuit Emulator (IICE™), and other system settings. Additionally, you can save the original design, in either an encrypted or non-encrypted format. This information is used to reproduce the exact state of the design.

Instrumenting and Saving a Design

After setting up the IICE as described in [Chapter 2, IICE Configuration](#) and after defining the instrumentation (selecting the signals for sampling, and setting breakpoints) as described in [Chapter 5, Using the Instrumentor](#), the design is instrumented and saved.



To save your instrumented design, select File->Save project instrumentation from the menu or click on the Save project's activated instrumentation icon.

Saving a design generates an *instrumentation design constraints* (.idc) file and adds compiler pragmas in the form of constraint files to the design RTL for the instrumented signals and break points. This information is then used to incorporate the instrumentation logic (IICE and COMM blocks) into the synthesized netlist.

CHAPTER 2

IICE Configuration

An important part preparing a design for debugging is setting the parameters for the Intelligent In-Circuit Emulator (IICE) in the instrumentor. The IICE parameters determine the implementation of one or more IICE units and configure the units so that proper communication can be established with the debugger. The IICE parameters common to all IICE units are set in the instrumentor window and apply to all IICE units defined for that design; the IICE parameters unique to each IICE definition in a multi-IICE configuration are interactively set on one of two IICE Configuration dialog box tabs. Additional IICE tabs are available to support the deep-trace debug feature.

- **IICE Sampler Tab** – includes sample depth selection and clock specification and also defines the external memory configuration for the HAPS[®] Deep Trace Debug feature.
- **IICE Controller Tab** – includes complex counter trigger width specification, selection of state machine triggering, and import/export of the IICE trigger signal.

This chapter describes how to configure one or more IICE units.

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

Multiple IICE Units

Multiple IICE units allow triggering and sampling of signals from different clock domains within a design. Each IICE unit is independent and can have unique IICE parameter settings including sample depth, sampling/triggering options, and sample clock and clock edge. During the subsequent debugging phase, individual or multiple IICE units can be armed.

Adding an IICE Unit

A New IICE button is included in the instrumentor graphical window to allow an additional IICE to be defined for the current design. When you click the New IICE button, the window is reduced to a tab and a new window is opened with the HDL source code redisplayed without any signals instrumented.

Note: When you create a new IICE from the graphical window, the name of the IICE unit is formed by adding an `_n` suffix to IICE (for example, IICE_0, IICE_1, etc.). If you create a new IICE from the command line using the instrumentor `iice new` command, you can optionally include a name for the IICE.

Right-clicking on the IICE tab and selecting Configure IICE from the popup menu brings up the Configure IICE dialog box which allows you to define the parameters unique to the selected IICE (see [Individual IICE Parameters, on page 14](#)).

Note: Communication port selection, which is common to all IICE units for an instrumentation, is defined in the instrumentor graphical window (see [Common IICE Parameters, on page 11](#)).

Deleting an IICE Unit

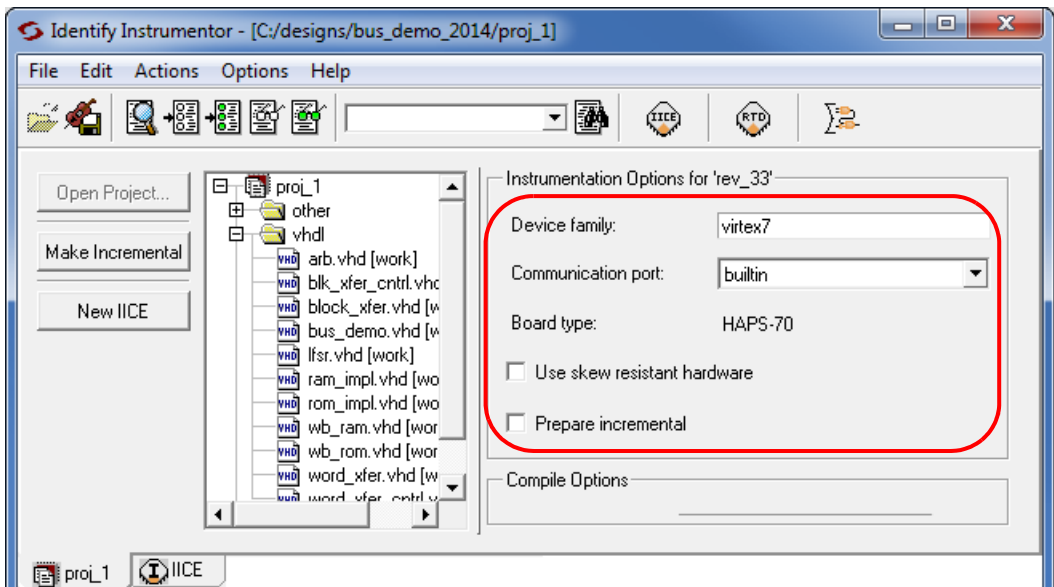
To delete an IICE unit from your instrumentor design, right-click on the tab of the IICE to be deleted and select Delete IICE from the popup menu.

Common IICE Parameters

The IICE parameters common to all IICE units defined for an instrumentation include:

- the IICE device family as defined by the synthesis tool
- the communication port
- if optional skew-resistant hardware is to be used
- if incremental updating is enabled (Xilinx technologies only)

These parameters are set/displayed in the instrumentor project window for the currently-active instrumentation. All IICE units in a multi-IICE configuration share these same parameter values.



Device Family

The device family selected in the synthesis tool is reported in the Device family field (the field is read-only) and cannot be changed.

Note: If the device family specified in the synthesis tool is not supported, an error message is issued and you are prompted to exit the instrumentor.

Communication Port

The Communication port parameter specifies the type of connection to be used to communicate with the on-chip IICE. The connection types are:

- **builtin** – indicates that the IICE is connected to the JTAG Tap controller available on the target device.
- **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 on a HAPS-6 or HAPS-7 series system through the UMRBus.

See [Chapter 5, *Connecting to the Target System*](#) in the *Debugger User Guide*, for a description of the communication interface.

Board Type

The Board Type read-only field is only present when one of the supported Synopsys HAPS device technologies is selected in the synthesis tool and indicates the targeted HAPS board type.

Use Skew-Resistant Hardware

The Use skew-resistant hardware check box, when checked, incorporates skew-resistant master/slave hardware to allow the instrumentation logic to operate without requiring an additional global clock buffer resource for the JTAG clock.

When no global clock resources are available for the JTAG clock, this option causes the IICE to be built using skew-resistant hardware consisting of master-slave flip-flops on the JTAG chain which prevents clock skew from affecting the logic. Enabling this option also causes the instrumentor to NOT explicitly define the JTAG clock as requiring global clock resources.

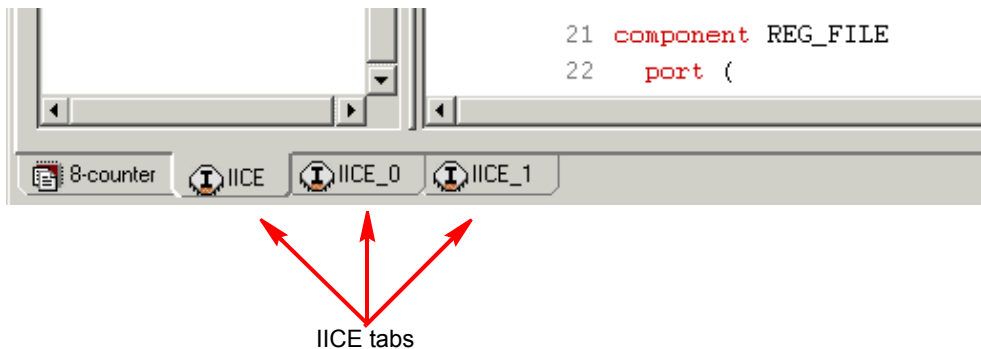
Prepare Incremental

The Prepare incremental check box is only enabled when one of the supported Xilinx Virtex technologies is selected in the synthesis tool. Checking this box enables the multi-pass, incremental flow feature available for specific Xilinx technologies (see [Chapter 3, *Incremental Flow*](#) in the *Debugger User Guide*).

Individual IICE Parameters



The individual parameters for each IICE are defined on two tabs of the Configure IICE dialog box. To display this dialog box, first select the active IICE by clicking the appropriate IICE tab in the instrumentor window and then select Actions->Configure IICE from the menu or click on the Edit IICE settings icon. You can also right-click directly on the IICE tab and select Configure IICE from the popup menu.

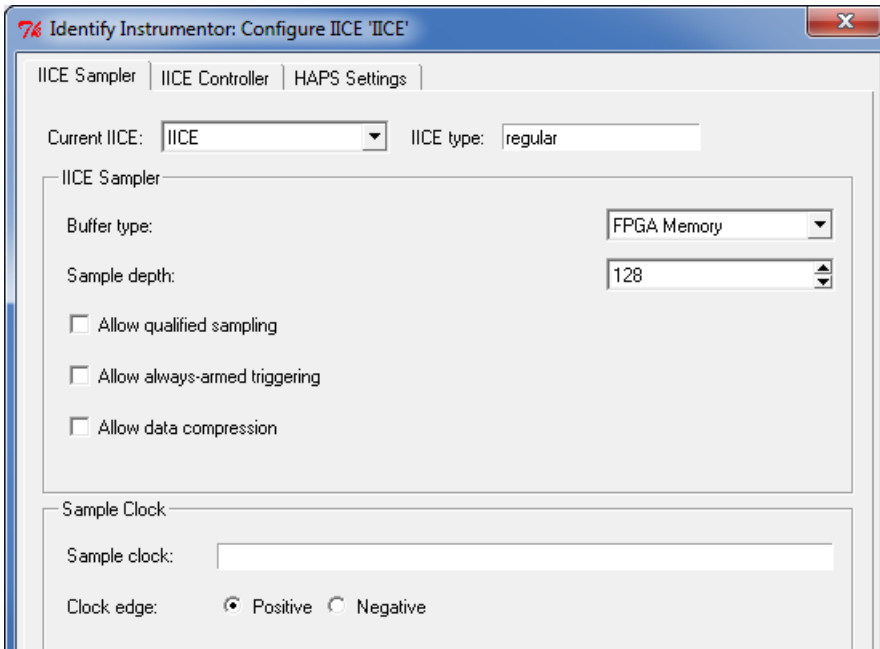


IICE Sampler Tab

The IICE Sampler tab defines:

- IICE unit for multi-IICE configurations
- IICE type (regular or real-time debugging)
- Buffer type
- Sample depth
- Sampling/triggering options
- Data compression use
- Sample clock and clock edge

Note: The IICE Sampler tab is redefined when the Buffer type is set to haps_DTD.



Current IICE

The Current IICE field identifies the target IICE when there are multiple IICE units defined within an implementation. The IICE is selected from the drop-down menu.

IICE type

The IICE type parameter is a read-only field that specifies the type of IICE unit currently selected – regular (the default) or rtd (real-time debugging). The IICE type is set from the project view in the user interface when a new IICE is defined or by an `iice sampler` Tcl command with a `-type` option. For information on the real-time debugging feature, see [Real-time Debugging, on page 63](#).

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 `internal_memory`; the `hapssram` setting configures the IICE to use external HAPSRAM memory. For more information, see [Chapter 3, HAPS Deep Trace Debug](#).

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.

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.

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

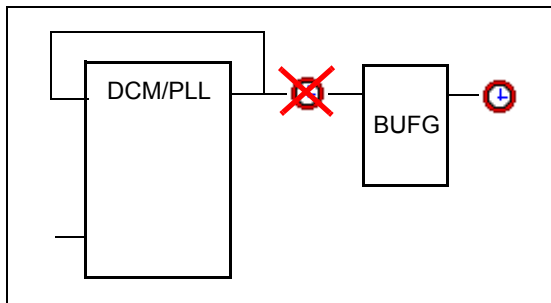
The Allow data compression check box, when checked, adds compression logic to the IICE to support sample data compression in the debugger (see [Sampled Data Compression, on page 28](#)). When unchecked (the default), compression

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.





Sample Clock

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 that the sampled signals are synchronous with or a multiple of that 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 instrumentation 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.

```
54
55     always @(posedge clk or negedge clr)
56     begin
57     if (clr == 1'b0)
58         current state = s_RESET; /* 4'b0000 */
59     else begin
```

Sample Clock Icon

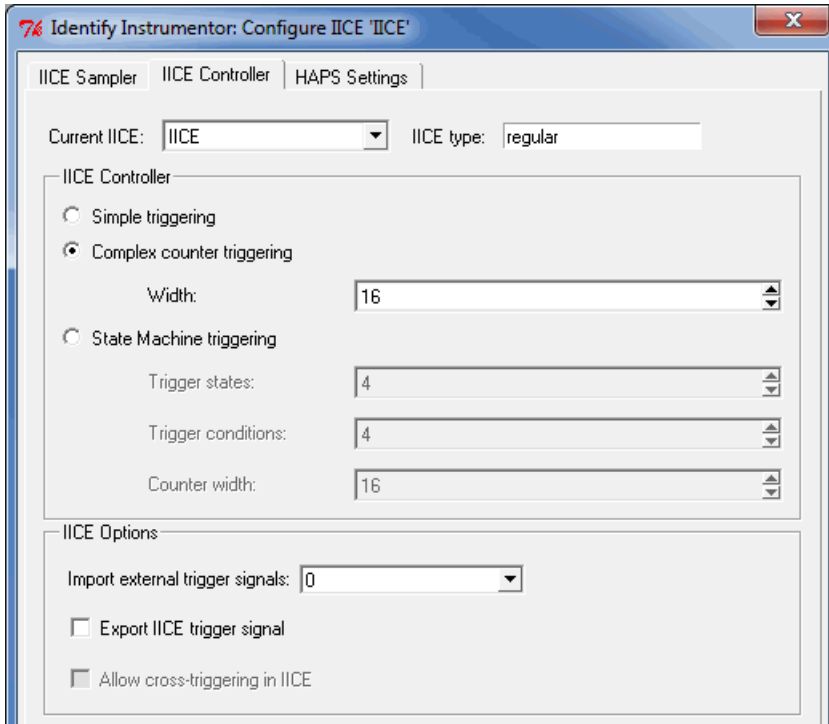
Note: You must set the other individual IICE parameters from the Configure IICE dialog box including the sample clock edge.

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.



Current IICE

The Current IICE field is used to identify the target IICE when there are multiple IICE units defined within an implementation. The IICE is selected from the drop-down menu.

IICE type

The IICE type parameter is a read-only field that specifies the type of IICE unit currently selected – regular (the default) or rtd (real-time debugging). The IICE type is set from the project view in the user interface when a new IICE is defined or by an iice sampler Tcl command with a -type option. For information on the real-time debugging feature, see [Real-time Debugging, on page 63](#).

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. For more information on state-machine triggering, see [State Machine Triggering, on page 88](#) in the *Debugger User Guide*.

Import External Trigger Signals

External triggering allows the 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. 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 [Cross Triggering, on page 36](#) in the *Debugger User Guide*.

HAPS Settings Tab

The HAPS Settings tab configures the deep-trace debug feature. For deep-trace debug configuration and setup information, see [Chapter 3, HAPS Deep Trace Debug](#).

CHAPTER 3

HAPS Deep Trace Debug

The HAPS Deep Trace Debug feature uses external SRAM as sample memory which allows Identify to use both the internal block RAM of the FPGA as well as external HAPSRAM. With the added external memory, a much deeper, signal-trace buffer is available.

Identify enables the use of HAPS DDR3 or SRAM daughter boards as an external buffering memory for designs being targeted for the HAPS platform in a single-FPGA debugging mode.

This chapter provides detailed descriptions of the feature and its use. A step-by-step tutorial using an example design is also available. The HAPS Deep Trace Debug feature is only available with Synopsys HAPS-60 and HAPS-70 series prototyping boards using a HAPS SRAM_1x1_HTII daughter board (HAPS-60) or a HAPS SRAM_HT3, DDR3_SODIMM_2R (8GB), or DDR3_SODIMM (4GB) daughter board (HAPS-70).

External Memory Instrumentation and Configuration

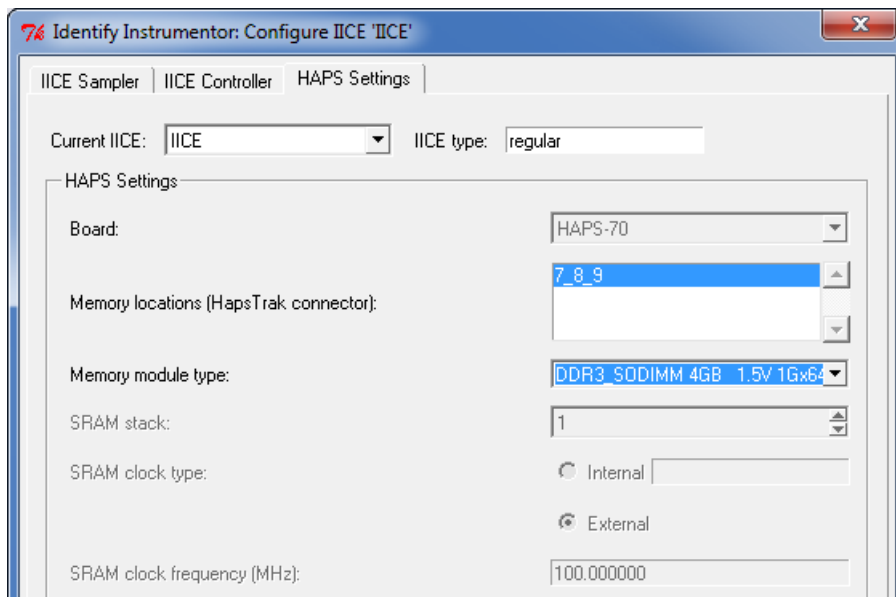
With the HAPS Deep Trace Debug mode, the Synplify/Identify flow remains unchanged. The only difference is in the configuration of a HAPS memory as the external sample buffer using IICE parameters.

The HAPS Deep Trace Debug feature includes the capability to control the configured sample depth. This depth can be dynamically varied using the Sample depth option available on the IICE Sampler tab. The depth can be varied between the minimum depth to the maximum configured depth, but cannot exceed the maximum configured depth.

The following figure shows the HAPS Settings dialog box for the HAPS Deep Trace Debug mode when the buffer type on the IICE Sampler tab is set to hapssram.

Note: You can also select the hapssram buffer type using the iice sampler hapssram Tcl command.

The individual HAPS Deep Trace Debug parameters are described in the ensuing table.




Note: Sample depth can only be set from the instrumentor and cannot be changed from within the debugger.

The parameters can also be set directly from the command line using the iice sampler command (see [Instrumentor iice sampler Options](#), on page 57 in the *Reference Manual* for complete syntax).

Parameter	Description
Board	The HAPS Deep Trace Debug feature is available only with specific HAPS boards. The instrumentor allows selection of hapssram buffer type only when the device used on the HAPS board is set in the implementation options. For example, if a Synopsys HAPS-70 device is specified in the synthesis tool, HAPS-70 is reported in the read-only Board field.
Memory locations	The buffering of the instrumented samples is performed using an external daughter card connected to any or all the HapsTrak® II or HapsTrak 3 connectors of a single FPGA. The selection of the connectors where the daughter cards are physically connected is done by selecting one or more HapsTrak locations (locations 1 through 6 for HAPS-60 or a matrix location for HAPS-70) of the daughter cards for the FPGA under debug.
Memory module type	The HapsTrak daughter card type is selected using this drop-down option. The daughter card type is determined by the Board type.
SRAM stack	The depth of SRAM on each daughter card is 4M locations of 72-bit words for HTII SRAM cards and 8M locations of 90-bit words for HT3 SRAM cards. To increase the external SRAM memory depth beyond 4M x 72 or 8M x 90, the daughter cards can be stacked. For the HTII type SRAM, 1, 2, or 4 daughter cards can only be stacked for the selected SRAM locations and for HT3 type SRAM cards, 1, 2, 3, or 4 cards can be stacked. The stack number specified applies to all connector locations specified by SRAM locations.
SRAM clock type	The clock to the SRAM daughter card can be either fed from the clock used within the design (Internal) or from an external clock source present on the HAPS board (see SRAM Clocks, on page 28).
SRAM clock frequency	Specifies the frequency of the clock source to the SRAM. The supported SRAM operating frequency ranges for various HAPS board and SRAM card stacks using the FPGA internal PLL output as the SRAM clock are listed in SRAM Clocks, on page 28 .

Running Deep Trace Debug with DDR3 Memory

The following steps provide an overview of running deep trace debug using the memory on either the DDR3_SODIMM_HT3 or DDR3_SODIMM2R_HT3 daughter board. Using external SRAM memory with the ProtoCompiler is described in [Running Deep Trace Debug with SRAM Memory, on page 27](#).

1. In the synthesis software, compile the design, highlight the implementation, and select New Identify Implementation from the drop-down menu.
2. Highlight the Identify implementation and select Launch Instrumentor from the drop-down menu.
3. In the instrumentor, complete the instrumentation of the design and click the Edit IICE settings icon (); the IICE Sampler tab opens by default.
4. Set the Buffer type to hapsram and make sure that the sample depth is set to the intended depth (once set in the instrumentor, the sample depth setting cannot be changed in the debugger).
5. Select the HAPS Settings tab and set the Memory module type of the DDR3 daughter board from the drop-down menu.
6. Save the IICE settings and save the instrumented design. You can close the instrumentor.
7. Run synthesis on the instrumented design, run place and route, and generate the bit file. If you intend to debug the design on a different system, copy the required files to the target host (see [Debugging on a Different Machine, on page 44](#) in the *Debugger User Guide*).


When you debug the design later, the tool automatically calculates the sample depth and source clock based on the configuration settings you supplied. The configured sample depth can be varied dynamically in the debugger from the minimum depth to the maximum configured depth.

To maximize performance when using DDR3 memory, use the guidelines in the table below to determine the sample frequency based on the number of sample bits being instrumented. The maximum number of instrumented bits that can be sampled with DDR3 memory is 2042.

Instrumented Bits	Max Sample Frequency	Max Sample Depth (4GB single rank)	Max Sample Depth (8GB dual rank)
1 to 250	140 MHz	134,217,727	268,435,455
251 to 506	70 MHz	67,108,863	134,217,727
507 to 1018	35 MHz	33,554,431	67,108,863
1019 to 2042	17.5 MHz	16,777,215	33,554,431

Running Deep Trace Debug with SRAM Memory

To configure the SRAM for deep trace debug:

1. In the synthesis software, compile the design, highlight the implementation, and select New Identify Implementation from the drop-down menu.
2. Highlight the Identify implementation and select Launch Identify Instrumentor from the drop-down menu.
3. In the instrumentor, complete the instrumentation of the design and click the Edit IICE Settings icon (); the IICE Sampler tab opens by default.
4. Set the Buffer type to hapssram and make sure that the sample depth is set to the intended depth (once set in the instrumentor, the sample depth setting cannot be changed in the debugger).
5. Select the HAPS Setting tab and set the following parameters:
 - Memory locations – select the HapsTrak connector location or location matrix where the SRAM daughter card is physically connected.
 - Memory module type – select the HapsTrak entry for SRAM.
 - SRAM stack – specify the number of daughter cards stacked at the connector location.
 - SRAM clock type – select either Internal or External. If Internal is selected, enter the name of the internal clock in the adjacent field. For more information on clocks, see [SRAM Clocks, on page 28](#).
 - SRAM clock frequency – specify the SRAM clock frequency. For better performance, it is recommended that you use FPGA internal PLL output as the source of the SRAM clock.

6. Save the IICE settings and save the instrumented design. You can close the instrumentor.
7. From the synthesis tool, synthesize the instrumented design.

When you debug the design later, the tool automatically calculates the sample depth and source clock based on the configuration settings you supplied. The configured sample depth can be varied dynamically from the minimum depth to the maximum configured depth.

Note: For every parameter/option that is set, as described above, an equivalent Tcl command is also available.

SRAM Clocks

When the clock source is internal to the design, any clock signal within the design at any hierarchy level can be instrumented as the SRAM clock. When the clock source is external, specify a suitable pin-lock constraint in the synthesis constraint file for the `deepbuf_sclk_iiceName_p` and `deepbuf_sclk_iiceName_n` ports (these ports are created automatically in the instrumented design). Provide the external clock source to this FPGA port.

Because of performance considerations, users are recommended to use FPGA internal PLL output as the source of the SRAM clock. The supported SRAM operating frequency ranges for SRAM card stacks using the internal PLL output as the SRAM clock are:

SRAM	SRAM_1x1_HTII	SRAM_HT3 (1.8V)
Board	HAPS-60 Series	HAPS-70 Series
1 SRAM stack	96 to 155 MHz	150 MHz
2 SRAM card stack	92 to 116 MHz	100 MHz
3 SRAM card stack	Not Supported	Not Supported
4 SRAM card stack	80to 110 MHz	Not Supported

Sample Depth Calculation

For a given, user-defined SRAM configuration setting, the maximum allowed depth can be calculated based on the formula described below.

- Number of HapsTrak slots used: N_{slot}
- Number of SRAM cards stacked: $NSRAM$
- Number of 72-bit or 90-bit words per SRAM card: N_{word}
(4194304 for HapsTrak II; 8388608 for HapsTrak 3)
- Number of signals to be sampled (instrumented): N_{signal}

$$K_{sample} \leq \left\lceil \frac{N_{word} N_{SRAM}}{\frac{N_{signal} + 6}{72 N_{slot}}} \right\rceil$$

HapsTrak II

$$K_{sample} \leq \left\lceil \frac{N_{word} N_{SRAM}}{\frac{N_{signal} + 6}{90 N_{slot}}} \right\rceil$$

HapsTrak 3

For example, if $N_{slot} = 1$, $NSRAM = 1$, $N_{word} = 8M$ (8388608) and $N_{signal} = 1900$, the maximum sampling depth for K samples for the SRAM card is 396k.

Sample Clock Calculation

For a given set of user-defined external memory configuration settings, the sample clock frequency can be estimated using the formula described below.

$$f_{sampling} \leq \left\lceil \frac{f_{SRAM}}{\frac{N_{signal} + 6}{72 N_{slot}}} \right\rceil + 2$$

HapsTrak II

$$f_{sampling} \leq \left\lceil \frac{f_{SRAM}}{\frac{N_{signal} + 6}{90 N_{slot}}} \right\rceil + 2$$

HapsTrak 3

In the above expressions:

- Number of HapsTrak slots used: N_{slot}
- Number of signals to be sampled (instrumented): N_{signal}
- SRAM bus frequency: f_{SRAM}

For example, if $f_{SRAM} = 100MHz$, $N_{slot} = 1$, and $N_{signal} = 1900$, the maximum sampling frequency is 3.44MHz.

Hardware Configuration Verification

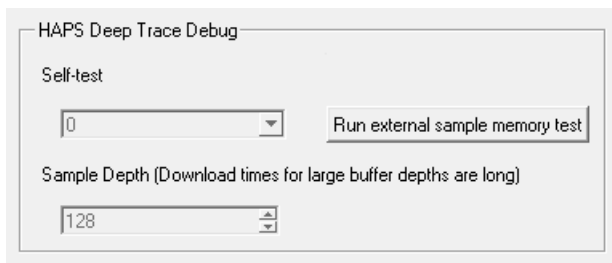
A self-test is available for verifying the deep trace debug hardware configuration. The self-test writes data patterns to the memory and reads back the data pattern written to detect configuration errors, connectivity problems, and frequency mismatches.

The self test is normally executed:

- following the initial setup to verify the hardware configuration against the instrumentation
- during routine operations whenever a hardware problem is suspect
- whenever the physical configuration is modified (changing any of the IICE Sampler dialog box configuration settings such as relocating the SRAM daughter card to another connector)

To run the self-test from the debugger GUI:

1. Open the project view.
2. Click the IICE icon.
3. Select one of the two patterns (pattern 0 or pattern 1) from the Self-test drop-down menu.
4. Click the Run SRAM tests button.



Selecting 0 uses one test pattern, and selecting 1 uses another pattern. To ensure adequate testing, repeat the command using alternate pattern settings.

The self-test can also be run from the command line using the following syntax:

iice sampler -runselftest 1|0

CHAPTER 4

Support for Instrumenting HDL

The debug environment fully supports the synthesizable subset of both Verilog and VHDL design languages. Designs that contain a mixture of VHDL and Verilog can be debugged – the software reads in your design files in either language.

There are some limitations on which parts of a design can be instrumented by the instrumentor. However, in all cases you can always instrument all other parts of your design.

The instrumentation limitations are usually related to language features. These limitations are described in this chapter.

- [VHDL Instrumentation Limitations](#), on page 32
- [Verilog Instrumentation Limitations](#), on page 34
- [SystemVerilog Instrumentation Limitations](#), on page 37

VHDL Instrumentation Limitations

The synthesizable subsets of VHDL IEEE 1076-1993 and IEEE 1076-1987 are supported in the current release of the debugger.

Design Hierarchy

Entities that are instantiated more than once are supported for instrumentation with the exception that signals that have type characteristics specified by unique generic parameters cannot be instrumented.

Subprograms

Subprograms such as VHDL procedures and functions cannot be instrumented. Signals and breakpoints within these specific subprograms cannot be selected for instrumentation.

Loops

Breakpoints within loops cannot be instrumented.

Generics

VHDL generic parameters are fully supported as long as the generic parameter values for the entire design are identical during both instrumentation and synthesis.

Transient Variables

Transient variables defined locally in VHDL processes cannot be instrumented.

Breakpoints and Flip-flop Inferencing

Breakpoints inside flip-flop inferring processes can only be instrumented if they follow the coding styles outlined below:

For flip-flops with asynchronous reset:

```
process (clk, reset, ...) begin
    if reset = '0' then
        reset_statements;
    elsif clk'event and clk = '1' then
        synchronous_assignments;
    end if;
end process;
```

For flip-flops with synchronous reset or without reset:

```
process (clk, ...) begin
    if clk'event and clk = '1' then
        synchronous_assignments;
    end if;
end process;
```

Or:

```
process begin
    wait until clk'event and clk = '1'
        synchronous_assignments;
end process;
```

The reset polarity and clock-edge specifications above are only exemplary. The debug software has no restrictions with respect to the polarity of reset and clock. A coding style that uses wait statements must have only one wait statement and it must be at the top of the process.

Using any other coding style for flip-flop inferring processes will have the effect that no breakpoints can be instrumented inside the corresponding process. During design compilation, the instrumentor issues a warning when the code cannot be instrumented.

Verilog Instrumentation Limitations

The synthesizable subsets of Verilog HDL IEEE 1364-1995 and 1364-2001 are supported.

Subprograms

Subprograms such as Verilog functions and tasks cannot be instrumented. Signals and breakpoints within these specific subprograms cannot be selected for instrumentation.

Loops

Breakpoints within loops cannot be instrumented.

Parameters

Verilog HDL parameters are fully supported. However, the values of all the parameters throughout the entire design must be identical during instrumentation and synthesis.

Locally Declared Registers

Registers declared locally inside a named `begin` block cannot be instrumented and will not be offered for instrumentation. Only registers declared in the module scope and wires can be instrumented.

Verilog Include Files

There are no limitations on the instrumentation of `'include` files that are referenced only once. When an `'include` file is referenced multiple times as shown in the following example, the following limitations apply:

- If the keyword `module` or `endmodule`, or if the closing `)` of the module port list is located inside a multiply-included file, no constructs inside the corresponding module or its submodules can be instrumented.
- If significant portions of the body of an `always` block are located inside a multiply-included file, no breakpoints inside the corresponding `always` block can be instrumented.

If either situation is detected during design compilation, the instrumentor issues an appropriate warning message.

As an example, consider the following three files:

adder.v File

```
module adder (cout, sum, a, b, cin);
  parameter size = 1;
  output cout;
  output [size-1:0] sum;
  input cin;
  input [size-1:0] a, b;
  assign {cout, sum} = a + b + cin;
endmodule
```

adder8.v File

```
`include "adder.v"
module adder8 (cout, sum, a, b, cin);
  output cout;
  parameter my_size = 8;
  output [my_size - 1: 0] sum;
  input [my_size - 1: 0] a, b;
  input cin;
  adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

adder16.v File

```
`include "adder.v"
module adder16 (cout, sum, a, b, cin);
  output cout;
  parameter my_size = 16;
  output [my_size - 1: 0] sum;
  input [my_size - 1: 0] a, b;
  input cin;
  adder #(my_size) my_adder (cout, sum, a, b, cin);
endmodule
```

There is a workaround for this problem. Make a copy of the include file and change one particular include statement to refer to the copy. Signals and breakpoints that originate from the copied include file can now be instrumented.

Macro Definitions

The code inside macro definitions cannot be instrumented. If a macro definition contains important parts of some instrumentable code, that code also cannot be instrumented. For example, if a macro definition includes the `case` keyword and the controlling expression of a `case` statement, the `case` statement cannot be instrumented.

Always Blocks

Breakpoints inside a synchronous flip-flop inferring an `always` block can only be instrumented if the `always` block follows the coding styles outlined below:

For flip-flops with asynchronous reset:

```
always @(posedge clk or negedge reset) begin
    if (!reset) begin
        reset_statements;
    end
    else begin
        synchronous_assignments;
    end;
end;
```

For flip-flops with synchronous reset or without reset:

```
always @(posedge clk) begin
    synchronous_assignments;
end process;
```

The reset polarity and clock-edge specifications and the use of `begin` blocks above are only exemplary. The instrumentor has no restrictions with respect to these other than required by the language.

For other coding styles, the instrumentor issues a warning that the code is not instrumentable.

SystemVerilog Instrumentation Limitations

The synthesizable subsets of Verilog HDL IEEE 1364-2005 (SystemVerilog) are supported with the following exceptions.

Typedefs

You can create your own names for type definitions that you use frequently in your code. SystemVerilog adds the ability to define new net and variable user-defined names for existing types using the typedef keyword. Only typedefs of supported types are supported.

Struct Construct

A structure data type represents collections of data types. These data types can be either standard data types (such as int, logic, or bit) or, they can be user-defined types (using SystemVerilog typedef). Signals of type structure can only be sampled and cannot be used for triggering; individual elements of a structure cannot be instrumented, and it is only possible to instrument (sample only) an entire structure. The following code segment illustrates these limitations:

```
module lddt_P_Struc_top (
  input sigsig_clk, sigsig_rst,
  .
  .
  .
  output struct packed {
    logic_nibble up_nibble;
    logic_nibble lo_nibble;
  } sig_oport_P_Struc_data
);
```

In the above code segment, port signal `sig_oport_P_Struc_data` is a packed structure consisting of two elements (`up_nibble` and `lo_nibble`) which are of a user-defined datatype. As elements of a structure, these elements cannot be instrumented. The signal `sig_oport_P_Struc_data` can be instrumented for sampling, but cannot be used for triggering (setting a watch point on the signal is not allowed). If this signal is instrumented for sample and trigger, the instrumentor allows only sampling and ignores triggering.

Union Construct

A union is a collection of different data types similar to a structure with the exception that members of the union share the same memory location. Trigger-expression settings for unions are either in the form of serialized bit vectors or hex/integers with the trigger bit width representing the maximum available bit width among all the union members. Trigger expressions using enum are not possible.

The example below shows an acceptable sample code segment for a packed union; the trigger expression for union d1 can be defined as:

```
typedef union packed {
    shortint u1;
    logic signed [2:1] [1:2] [4:1] u2;
    struct packed {
        bit signed [1:2] [1:2] [2:1] st1;
        struct packed {
            byte unsigned st2;
        } u3_int;
    } u3;
    logic [1:2] [0:7] u4;
    bit [1:16] u5;
} union_dt;

module top (
    input logic clk,
    input logic rst,
    input union_dt d1,
    output union_dt q1,
    ...

```

Arrays

Partial instrumentation of multi-dimensional arrays and multi-dimensional arrays of struct and unions are not permitted.

Interface

Interface and interface items are not supported for instrumentation and cannot be used for sampling or triggering. The following code segment illustrates this limitation:

```

interface ff_if (input logic clk, input logic rst,
    input logic din, output logic dout);
modport write (input clk, input rst, input din, output dout);
endinterface: ff_if

module top (input logic clk, input logic rst,
    input logic din, output logic dout) ;

    ff_if ff_if_top(.clk(clk), .rst(rst), .*);
    sff UUT (.ff_if_0(ff_if_top.write));
endmodule

```

In the above code segment, the interface instantiation of interface `ff_if` is `ff_if_top` which cannot be instrumented. Similarly, interface item `modport write` cannot be instrumented.

Port Connections for Interfaces and Variables

Instrumentation of named port connections on instantiations to implicitly instantiate ports is not supported.

Packages

Packages permit the sharing of language-defined data types, typedef user-defined types, parameters, constants, function definitions, and task definitions among one or more compilation units, modules, or interfaces. Instrumentation within a package is not supported.

Concatenation Syntax

The concatenation syntax on an array watchpoint signal is not accepted by the debugger. To illustrate, consider a signal declared as:

```
bit [3:0] sig_bit_type;
```

To set a watchpoint on this signal, the accepted syntax in the debugger is:

```
watch enable -iice IICE {/sig_bit_type} {4'b1001}
```

The 4-bit vector cannot be divided into smaller vectors and concatenated (as accepted in SystemVerilog). For example, the below syntax is not accepted:

```
watch enable -iice IICE {/sig_bit_type} {{2'b10,2'b01}}
```


CHAPTER 5

Using the Instrumentor

The instrumentor performs the following functions:

- defines the instrumentation for the user's HDL design
- creates the instrumented HDL design
- creates the associated IICE
- creates the design database

The remainder of this chapter describes:

- [Instrumentor Windows](#)
- [Commands and Procedures](#)

Instrumentor Windows

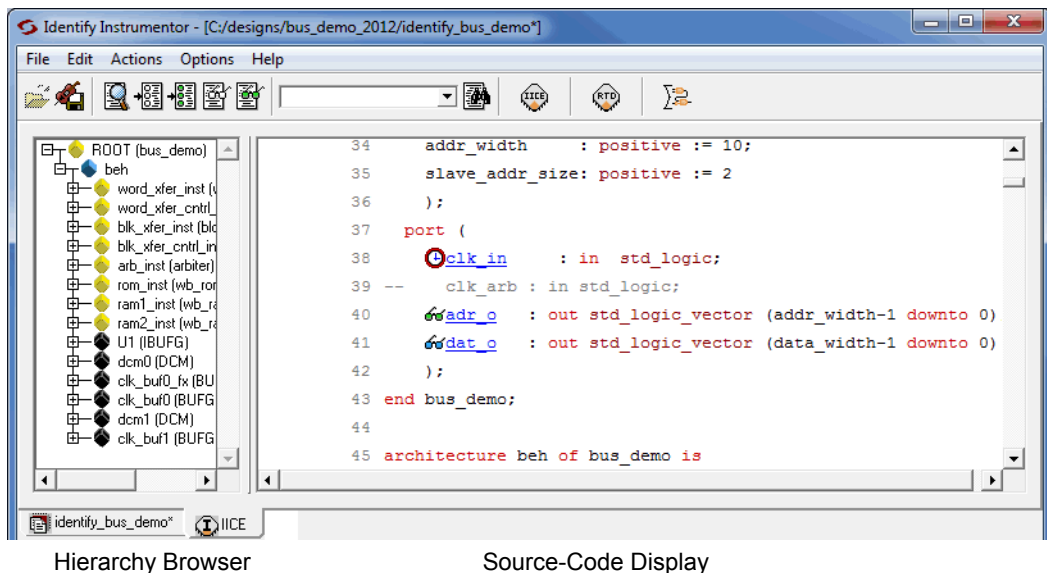
The Graphical User Interface (GUI) for the instrumentor is divided into three major areas:

- [Instrumentation Window](#)
- [Project Window](#)
- [Console Window](#)

In this section, each of these areas and their uses is described. The following discussions assume that an HDL design has been loaded into the instrumentor.

Instrumentation Window

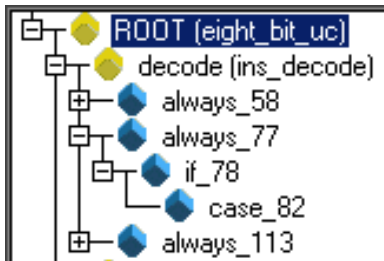
The instrumentation window includes a hierarchy browser on the left and the source-code display on the right. The window is displayed when you open a synthesis project (`prj`) file in the instrumentor by either launching the instrumentor from a synthesis project or explicitly loading a synthesis project into the instrumentor.



Hierarchy Browser

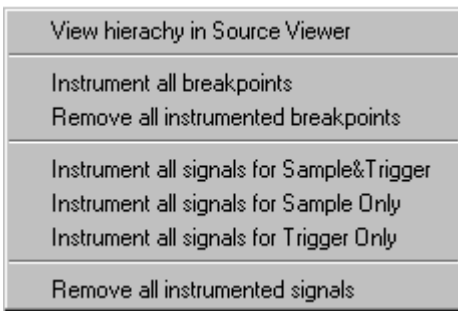
The hierarchy browser on the left shows a graphical representation of the design's hierarchy. At the top of the browser is the ROOT node. The ROOT node represents the top-level entity or module of your design. For VHDL designs, the first level below the ROOT is the architecture of the top-level entity. 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 visible in the adjacent source code display.

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.

Note: You cannot instrument signals when a sample clock is included in the defined group.

The popup menu functions can be duplicated in the console window using the instrumentor hierarchy command in combination with either the instrumentor breakpoints or signals command. A typical instrumentor command sequence to instrument the signal set within a design would be:

signals add [hierarchy find -type signal /]










In the above sequence, the slash (/) indicates that all of the signals in the root hierarchy (entire design) are to be instrumented. If you specify a lower level of hierarchy following the slash, the command only applies to that hierarchical level. For more information on the instrumentor breakpoints, signals, and hierarchy commands, see the *Reference Manual*.

Black-box modules are represented by a black icon, and their contents can not be instrumented. Also, certain modules cannot be instrumented (see [Chapter 4, Support for Instrumenting HDL](#), for a specific description). Modules that cannot be instrumented are displayed in a disabled state in a gray font.

Source Code Display

The HDL source code in the source code display is annotated with signals that can be probed and breakpoints that can be selected. Signals that can be selected for probing by the IICE are underlined, colored in blue, and have small watchpoint icons next to them. Source lines that can be selected as breakpoints have small circular icons in the left margin adjacent to the line number.

```

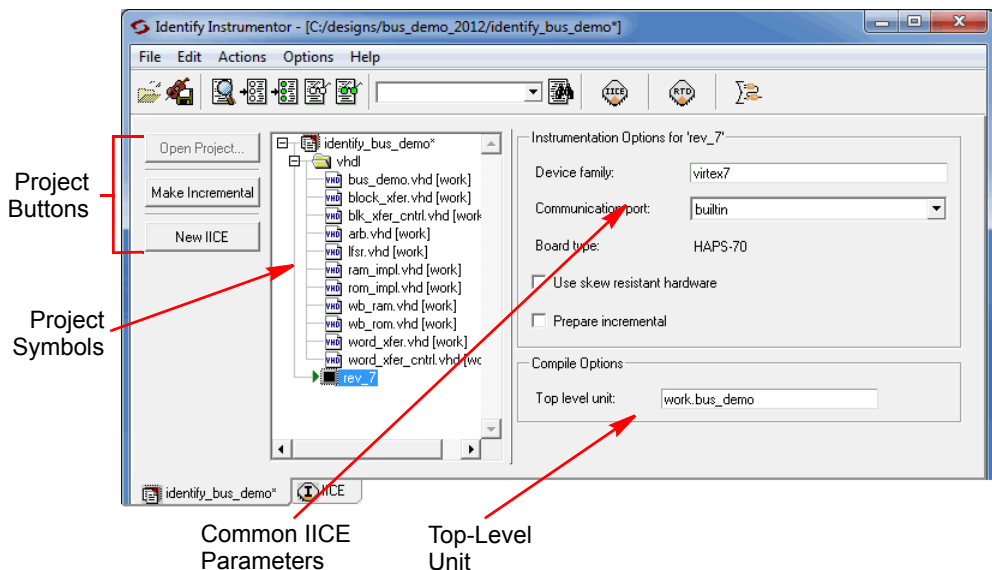
56  begin
57      if  clr = '0' then
58           current state <= s_RESET;
59      elsif  clk'event and  clk = '1' then
60          case  current state is
 61          when s_RESET      =>  current state <= s_ONE;
 62          when s_ONE        =>  current state <= s_TWO;

```

Project Window

The project window is displayed when you first start up the instrumentor (to invoke the instrumentor, see [Opening Designs, on page 47](#)).

The project window includes a set of project management buttons and a symbolic view of the project on the left and an area for setting the common IICE parameters on the right (the area below the common IICE parameters shows the top-level unit when the design was compiled).



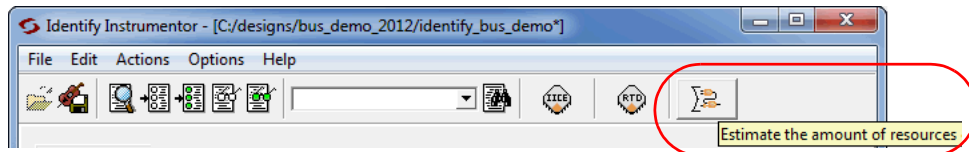
The project window can be displayed at any time by clicking the project name tab that remains visible when an instrumentation window is displayed.

Console Window

The console window appears below the project or instrumentation window and displays a history of instrumentor commands that have been executed, including those executed by menu selections and button clicks. The instrumentor console window allows you to enter commands as well as to view the results of those commands. Command history recording is available through a transcript command (see the *Reference Manual*).

```
INFO: D:/Designs/8-bit-verilog/data_mux.v(1): compiling module data_mux
INFO: D:/Designs/8-bit-verilog/alu.v(1): compiling module alu
WARNING: D:/Designs/8-bit-verilog/alu.v(83): full_case directive is effective
: might cause synthesis - simulation differences
INFO: D:/Designs/8-bit-verilog/ins_rom.v(1): compiling module ins_rom
INFO: D:/Designs/8-bit-verilog/spcl_regs.v(1): compiling module spcl_regs
INFO: D:/Designs/8-bit-verilog/io.v(1): compiling module io
INFO: Current design is 'eight_bit_uc'
D:/tools/ident200_021R/bin$
```

When you save your instrumentation (click Save project's active implementation), the information at the bottom of the display reports the total number of instrumented signals to implement in each IICE. To display the resource estimates for the FPGA device, click the Estimate the amount of resources consumed by the IICE(s) icon.



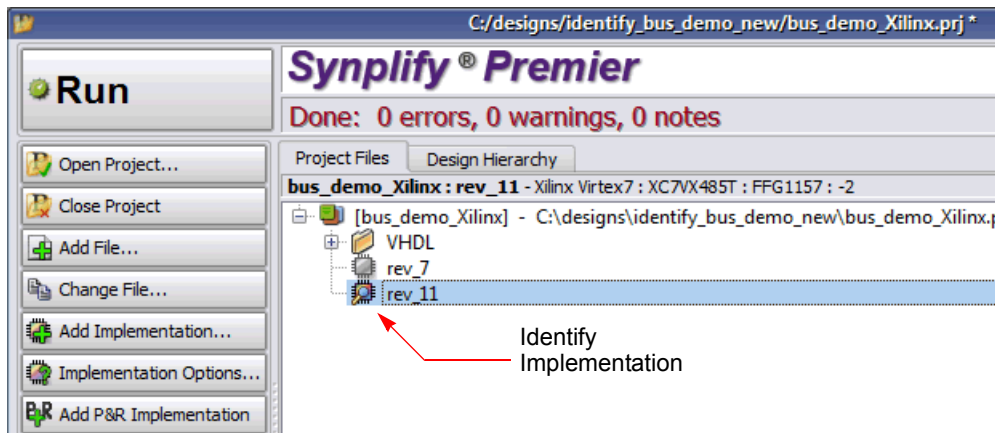
Commands and Procedures

The following sections describe basic instrumentor commands and procedures.

Opening Designs

To launch the instrumentor from the Synopsys synthesis tool (Synplify, Synplify Pro, Synplify Premier, or Certify) graphical interface:

1. Right click on the implementation and select New Identify Implementation from the popup menu.
2. Set/verify any technology, device mapping, or other pertinent options (see the implementation option descriptions in the *Synopsys FPGA Synthesis* or *Certify User Guide*) and click OK. A new, implementation is added to the synthesis tool project view.



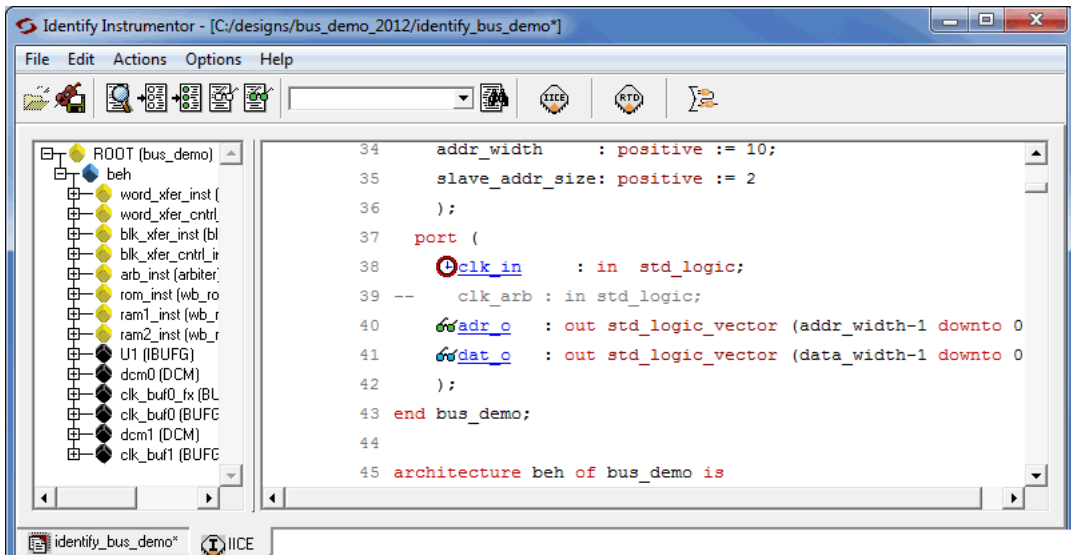
3. Right click on the Identify implementation and select Launch Identify Instrumentor from the popup menu to launch the instrumentor.

Launching the instrumentor from a Synopsys synthesis tool:

- Brings up the instrumentor graphical window.
- Extracts the list of design files, their work directories, and the device family from the project file.

- Automatically compiles the design files using the synthesis tool compiler and displays the design hierarchy and the HDL file content with all the potential instrumentation marked and available for selection.

The following figure shows the initial instrumentor graphical window with the hierarchy browser and RTL tab.



When importing a synthesis tool project into the instrumentor, the working directory is automatically set from the corresponding project file. See [Chapter 1, Design Setup](#), for details on setting up and managing projects

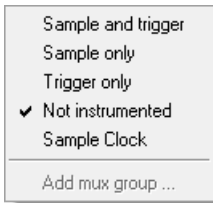
Executing Script Files

A script file is a convenient way to capture a Tcl command sequence you would like to repeat. To execute a script file, select the File->Execute Script menu selection in the instrumentor user interface and navigate to the location of your script file or use the source command (see Chapter 2, *Alphabetical Command Reference*, in the *Reference Manual*).

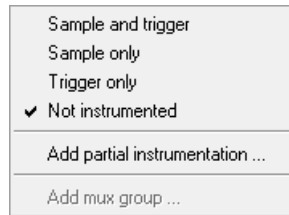
Selecting Signals for Data Sampling



To select a signal to be sampled, simply click on the watchpoint icon next to the signal name in the instrumentation window; a popup menu appears that allows the signal to be selected for sampling, triggering, or both.



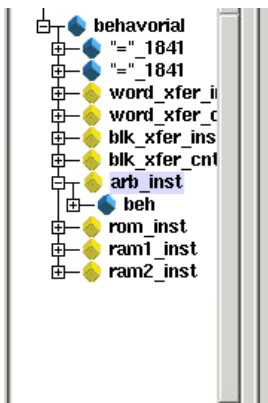
Scalar Signal Popup



Bus Signal Popup

To control the overhead for the trigger logic, always instrument signals that are not needed for triggering with the Sample only selection (the watchpoint icon is blue for sample-only signals).

Qualified clock signals can be specified as the Sample Clock (see [Sample Clock, on page 17](#)) and bus segments can be individually specified (see [Instrumenting Buses, on page 51](#)). In addition, signals specified as Sample and trigger or Sample only can be included in mux groups as described in [Multiplexed Groups, on page 55](#).



```

27  port (
28      clk      : in std_logic;
29      reset    : in std_logic;
30      req1     : in std_logic;
31      req2     : in std_logic;
32      grant1   : out std_logic;
33      grant2   : out std_logic;
34  );
35  end arbiter;
36
37  architecture beh of arbiter is
38      type states is ( st_idle1, st_idle2, st_grant1, st_grant2
39      signal curr_state, next_state : states;

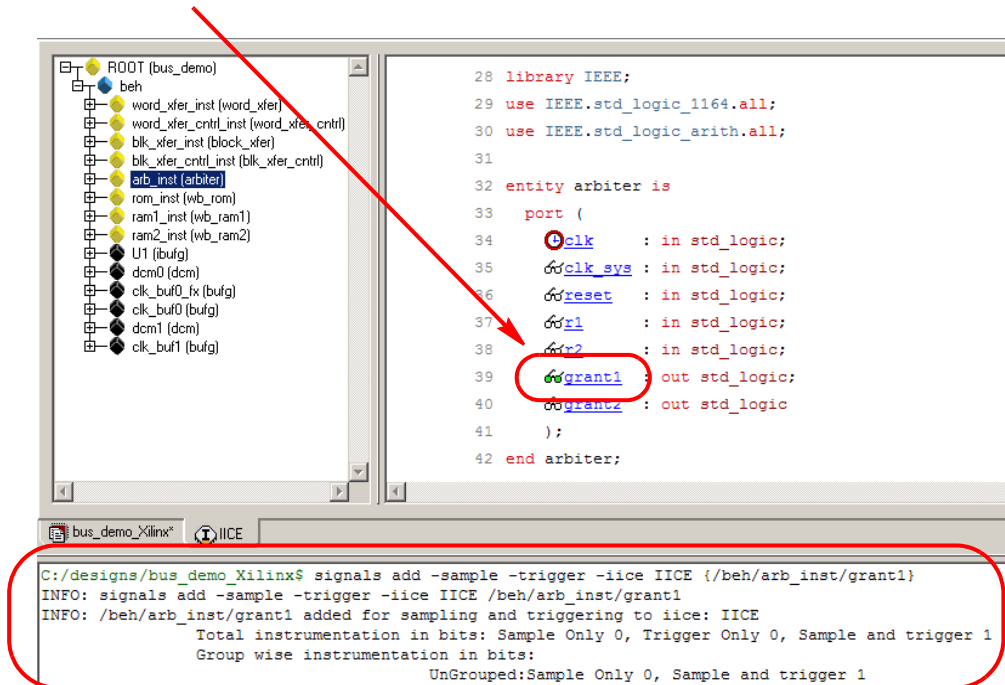
```

Signals that can be selected

If the icon is clear (unfilled), the signal is disabled for sampling (not instrumented), and if the icon is red, the signal is enabled for triggering only. If the icon is blue, the signal is enabled for sampling only, and if the icon is green,

the signal is enabled for both sampling and triggering. In the example below, notice that when signal grant1 is enabled, the console window displays the text of the command that implements the selection and the results of executing the command.

Signal “grant1” selected



To disable a signal for sampling or triggering, select the signal in the instrumentation window and then select Not instrumented from the popup menu; the watchpoint icon will again be clear (unfilled).

Note: You can use Find to recursively search for signals and then instrument selected signals directly from the Find dialog box (see [Searching for Design Objects, on page 70](#)).

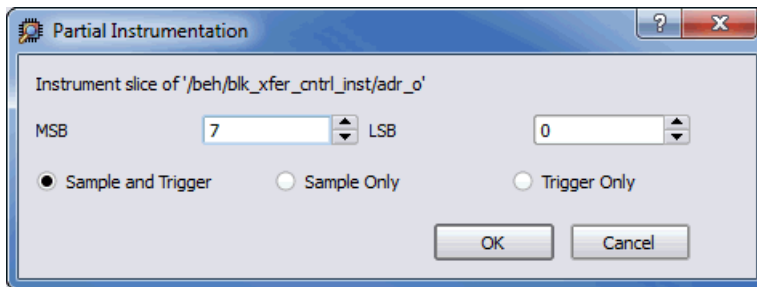
Instrumenting Buses

Entire buses or individual or groups of bits of a bus can be independently instrumented.

Instrumenting a Partial Bus

To instrument a sequence (range) of bits of a bus:

1. Place the cursor over a bus that has not been fully instrumented and select Add partial instrumentation to display the following dialog box.







2. In the dialog box, enter the most- and least-significant bits in the MSB and LSB fields. Note that the bit range specified is contiguous; to instrument non-contiguous bit ranges, see the section, [Instrumenting Non-Contiguous Bits or Bit Ranges](#), on page 52.

Note: When specifying the MSB and LSB values, the index order of the bus must be followed. For example, when defining a partial bus range for bus [63:0] (or “63 downto 0”), the MSB value must be greater than the LSB value. Similarly, for bus [0:63] (or “0 upto 63”), the MSB value must be less than the LSB value.

3. Select the type of instrumentation for the specified bit range from the drop-down menu and click OK.

When you click OK, a large letter “P” is displayed to the left of the bus name in place of the watchpoint icon. The color of this letter indicates if the partial bus is enabled for triggering only (red), for sampling only (blue), or for both sampling and triggering (green).

```
277 input  MEM RD,  MEM WR;  
278 input [63:0]  DATA IN;  
279 output [63:0]  DATA OUT;
```

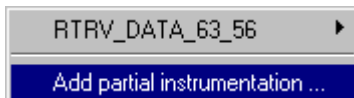
Instrumenting Single Bits of a Bus

To instrument a single bit of a bus, enter the bit value in the MSB field of the Add partial bus dialog box, leave the LSB field blank, and select the instrumentation type from the drop-down menu as previously described.

Instrumenting Non-Contiguous Bits or Bit Ranges

To instrument non-contiguous bits or bit ranges:

1. Instrument the first bit range or bit as described in one of the two previous sections.
2. Re-position the cursor over the bus, click the right mouse button, and again select Add partial instrumentation to redisplay the Add partial bus dialog box. Note that the previously instrumented bit or bit range is now displayed.



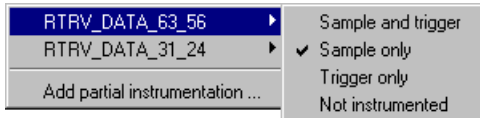
3. Specify the bit or bit range to be instrumented as previously described, select the type of instrumentation from the drop-down menu, and click OK. If the type of instrumentation is different from the existing instrumentation, the letter “P” will be yellow to indicate a mixture of instrumentation types.

Note: Bits cannot overlap groups (a bit cannot be instrumented more than once).

Changing the Instrumentation Type

To change the instrumentation type of a partial bus:

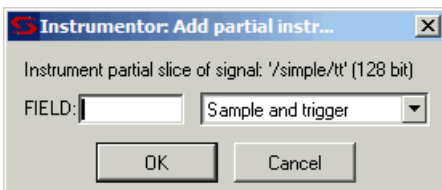
1. Position the cursor over the bus and click the right mouse button.
2. Highlight the bit range or bit to be changed and select the new instrumentation type from the adjacent menu.



Note: The above procedure is also used to remove the instrumentation from a bit or bit range by selecting Not instrumented from the menu.

Partial Instrumentation

Partial instrumentation allows fields within a record or a structure to be individually instrumented. Selecting a compatible signal for instrumentation, either in the RTL window or through the Find dialog box, enables the partial instrumentation feature and displays a dialog box where the field name and its type of instrumentation can be entered.



When instrumented, the signal is displayed with a P icon in place of the watchpoint (glasses) icon to indicate that portions of the record are instrumented. The P icon is the same icon that is used to show partial instrumentation of a bus and uses a similar color coding:

- Green indicates that all fields of the record are instrumented for sample and trigger
- Blue indicates that all fields of the record are instrumented for sample only

- Red indicates that all fields of the record are instrumented for trigger only
- Yellow indicates that not all fields of the record are instrumented the same way

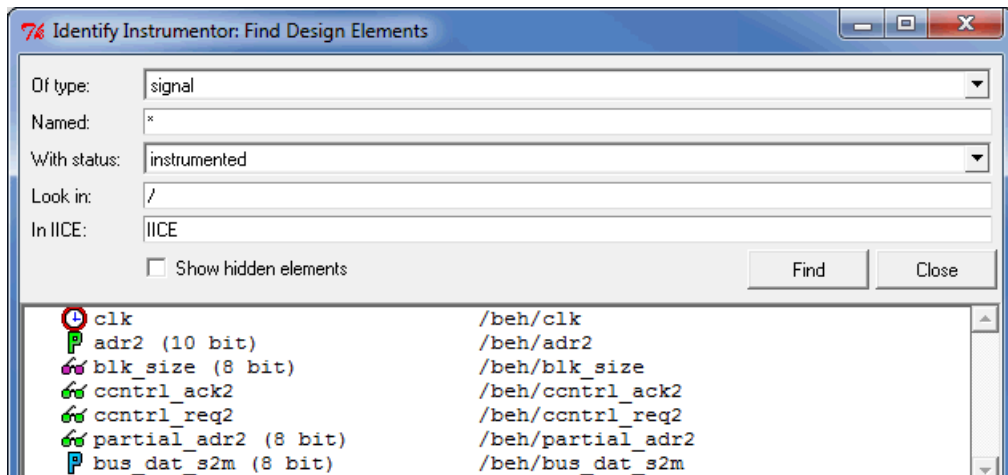
The figure below shows the partial instrumentation icon on signal `tt`. The yellow color indicates that the individual fields (`tt.r2` and `tt.c2`) are assigned different types of instrumentation.

```

31
32 signal P tt: matrix_element1;
33 begin
34   P tt.r2 <= cc_r2;
35   P tt.c2 <= cc_c2;
36   P tt.ele.r1 <= cc_r1;

```

The Find dialog also uses the partial instrumentation icon to show the state of instrumentation on fields of partially instrumented records.

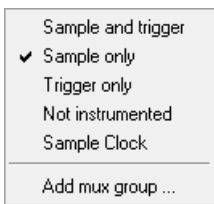


Note: Partial instrumentation can only be added to a field or record one slice-level down in the signal hierarchy.

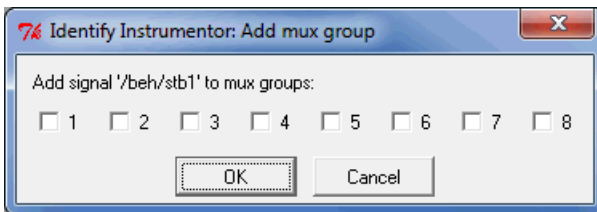
Multiplexed Groups

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.



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](#), on page 78 of 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 [Selecting Multiplexed Instrumentation Sets](#), on page 23 in the *Debugger User Guide*.

Sampling Signals in a Folded Hierarchy

When a design contains entities or modules that are instantiated more than once, it is termed to have a *folded* hierarchy (folded hierarchies also occur when multiple instances are created within a `generate` loop). By definition, there will be more than one instance of every signal in a folded entity or module. To allow you to instrument a particular instance of a folded signal, the instrumentor automatically recognizes folded hierarchies and presents a choice of all possible instances of each signal with the hierarchy.



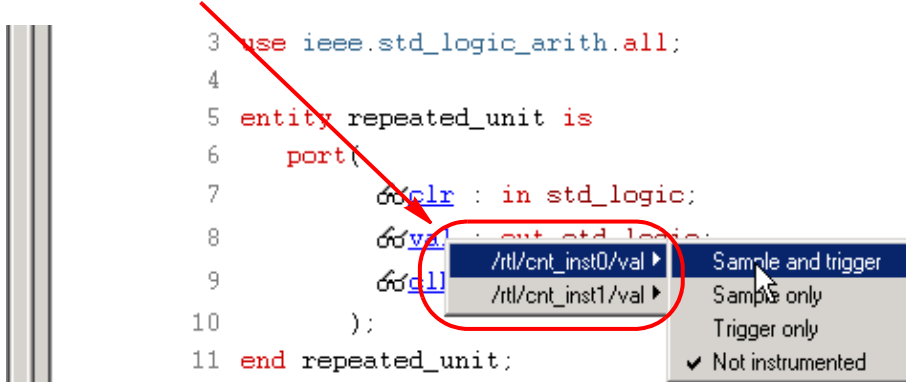
The choices are displayed in terms of an absolute signal path name originating at the top-level entity or module. The list of choices for a particular signal is accessed by clicking the watchpoint icon or corresponding signal.

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.

The list of instrumentable instances of signal **val**



The color of the watchpoint icon is determined as follows:

- If no instances of the signal are selected, the watchpoint icon is clear.
- If some, but not all, instances of the signal are defined for sampling, the watchpoint icon is yellow.
- If all instances are defined for sampling, the color of the watchpoint icon is determined by the type of sampling specified (all instances sample only: blue, all instances trigger only: red, all instances sample and trigger: green, all instances in any combination: yellow).

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 *Reference Manual* for more information.

To disable an instance of a signal that is currently defined for sampling, click on the watchpoint icon or signal, select the instance from the list displayed, and select Not instrumented.

For related information on folded hierarchies in the debugger, see [Activating/Deactivating Folded Instrumentation, on page 24](#) and [Displaying Data from Folded Signals, on page 32](#) in the *Debugger User Guide*.

Instrumenting the Verdi Signal Database

The instrumentor can import signals directly from the Verdi platform. After performing behavioral analysis and generating the essential signal database (ESDB), the essential signal list from the Verdi platform is brought directly into the instrumentor where the signals are instrumented. To bring in the essential signal list:

1. Load the project into the instrumentor.
2. Parse the essential signal list from the ESDB using the command:

verdi getsignals *ESDBpath*

In the above syntax, *ESDBpath* is the location where *es.esdb++* is installed. For example:

```
verdi getsignals path/es
```

3. Instrument the essential signal list using the command:

verdi instrument

The signals are automatically instrumented as sample and trigger.

4. Instrument the sample clock (a sample clock is required by the instrumentor).
5. Configure the IICE and instrument the design.

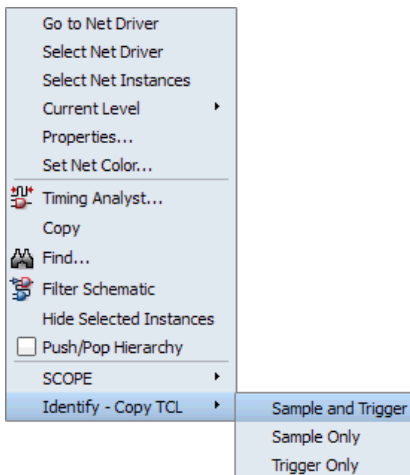
The instrumented design is then synthesized, placed and routed, and programmed into the FPGA. The debugger samples the data and generates the fast signal database (FSDB) which is then displayed in the Verdi nWave viewer. For more information on the fast signal database and using the Verdi nWave viewer, see [Generating the Fast Signal Database, on page 53](#) in the *Debugger User Guide*.

Instrumenting Signals Directly in the idc File

In addition to the methods described in the previous sections, signals can be instrumented directly within the *srs* file in the synthesis tool outside of the instrumentor. This methodology facilitates updates to a previous instrumentation and also allows signals within a parameterized module, which were

previously unavailable for instrumentation, to be successfully instrumented. This technique is referred to as “post-compile instrumentation.” To instrument a signal directly within the synthesis tool using this technique:

1. Compile the existing instrumented design.
2. Open the RTL view (srs file).
3. Highlight the net of the signal to be instrumented.
4. With the net highlighted, click the right mouse button, select Identify - Copy TCL from the popup menu, and select the type of instrumentation to be applied.



5. Create a new or open an existing identify.idc file in the *designName/rev_n* directory and paste the signal string into the file. The following figure shows the cntrl_ack_o_0_sqmuxa signal (from the block_xfer block) on line 10 being pasted into the file as a sample-only signal.

```

1 iice new {IICE} -type regular
2 iice controller -iice {IICE} none
3 iice sampler -iice {IICE} -depth 128
4
5 signals add -iice {IICE} -silent -trigger {/SRS/blk_xfer_inst/cntrl_ack_o_0_sqmuxa}
6 signals add -iice {IICE} -silent -sample {/beh/arb_inst/reset}
7 signals add -iice {IICE} -silent -trigger -sample {/beh/arb_inst/beh/curr_state}\
8 {/beh/arb_inst/beh/next_state}
9 iice clock iice {IICE} -edge positive {/clk}
10 signals add -sample {/SRS/blk_xfer_inst/cntrl_ack_o_0_sqmuxa}

```

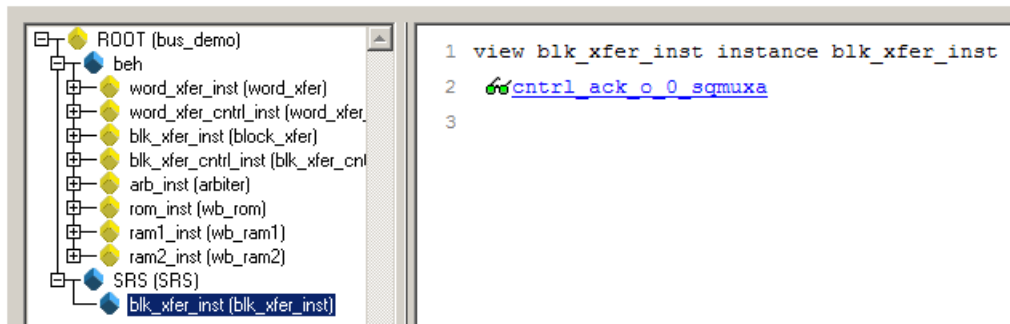
Note: If you are creating a new `identify.idc` file, you must add the IICE definition on lines 1, 2, and 3 to the beginning of the file as shown in the above figure.

6. Edit the entry and add an `-iice` option to the line as shown in the example below (the Copy TCL command does not automatically include the IICE unit in the entry):

```
signals add -iice {IICE} -sample
           {/SRS/blk_xfer_inst/cntrl_ack_o_0_sqmuxa}
```

7. Save the edited `identify.idc` file and rerun synthesis.

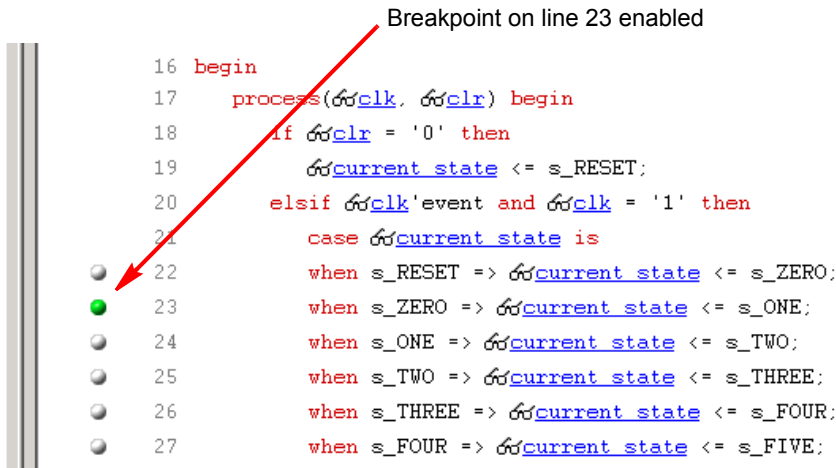
When you open the debugger, an SRS entry is included in the hierarchy browser; selecting this entry displays the additional signal or signals added to the `identify.idc` file. Selecting a signal in the instrumentation window brings up the Watchpoint Setup dialog box to allow a trigger expression to be assigned to the defined signal.



Note that trigger expressions on signals added to the `identify.idc` file must use the VHDL style format.

Selecting Breakpoints

Breakpoints are used to trigger data sampling. Only the breakpoints that are instrumented in the instrumentor can be enabled as triggers in the debugger. To instrument a breakpoint in the instrumentor, simply click on the circular icon to the left of the line number. The color of the icon changes to green when enabled.



Once a breakpoint is instrumented, the instrumentor creates trigger logic that becomes active when the code region in which the breakpoint resides is active.

In the above example, the code region of the instrumented breakpoint is active if the variable `current_state` is state zero (`s_ZERO`) and the signal `clr` is not '0' when the clock event occurs.

Selecting Breakpoints Residing in Folded Hierarchy

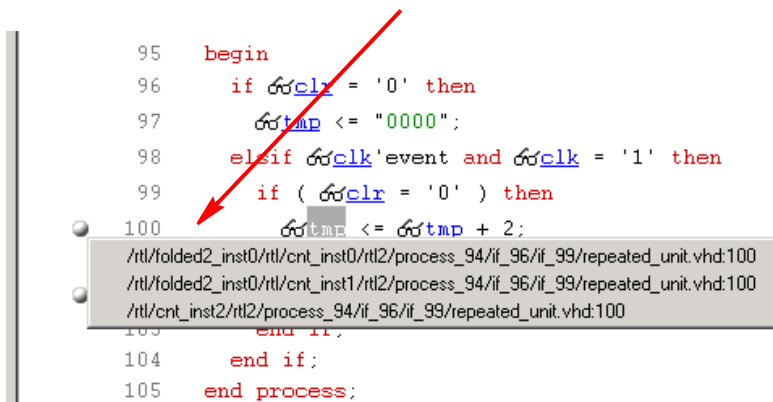
If a design contains entities or modules that are instantiated more than once, the design is termed to have *folded* hierarchy. By definition, there will be more than one instance of every breakpoint in a folded entity or module. To allow you to instrument a particular instance of a folded breakpoint, the instrumentor automatically detects folded hierarchy and presents a choice of all possible instances of each breakpoint.

The choices are displayed in terms of an absolute breakpoint path name originating at the top-level entity or module. The list of choices for a particular breakpoint is accessed by clicking on the breakpoint icon to the left of the line number.

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.

Folded breakpoint on line 100 selected



The color of the breakpoint icon is determined as follows:

- If no instances of the breakpoint are selected, the icon is clear in color.
- If some, but not all, instances of the breakpoint are selected, the icon is yellow.
- If all instances are selected, the icon is green.

Alternately, 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 *Reference Manual* 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.

Configuring the IICE



If the IICE configuration parameters for the active IICE need to be changed, use the Actions->Configure IICE menu selection or the Edit IICE settings icon to change them. [Chapter 2, IICE Configuration](#), discusses how to set these parameters for both single- and multi-IICE configurations and for the HAPS deep trace debug feature.

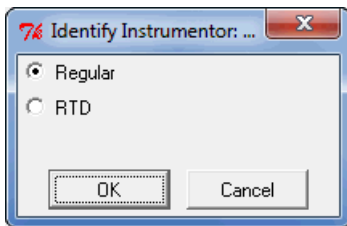
Real-time Debugging

Real-time debugging is a feature that provides scope or logic analyzer access to instrumented signals directly through a Mictor board interface connector installed on the HAPS board.

Enabling the Real-time Debugging Feature

To use the real-time debugging feature, a Synopsys HAPS device family is specified in the synthesis tool and a special IICE is defined in either the user interface or by command entry in the console window. The feature cannot be used with the incremental flow.

To specify the IICE from the user interface, open the Project view and click the New IICE button to display the following dialog box. Select the RTD radio button and click OK.



To define the IICE from the console window, enter the `iice new` command:

```
iice new [iiceID] -type rtd
```

In the command syntax, *iiceID* is the name of the new IICE and, if omitted, defaults to an incremental number (for example, IICE_0).

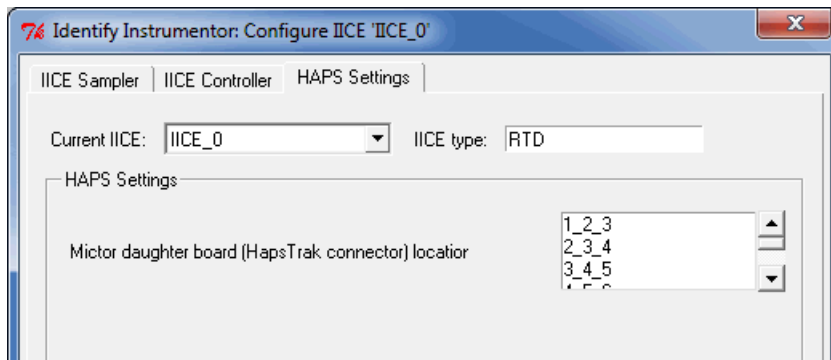
Either of the above methods creates a new, real-time IICE for the design with all of the signals “not instrumented.” This new IICE is identified by the “R” symbol in the IICE tab.



Before you can instrument any of the signals, you must configure the HAPS Settings tab as outlined in the next section.

HAPS Settings Tab

The HAPS Settings tab for the real-time debugging feature includes a drop-down menu for selecting the Mictor daughter card locations. The available locations are determined by the Synopsys HAPS device selection on the Device tab of the Implementation Options dialog box in the synthesis tool.

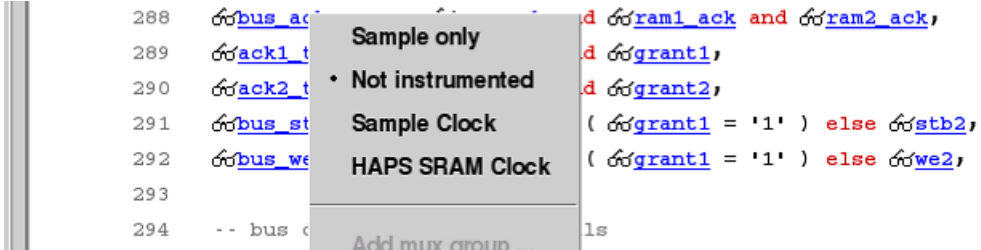


On the IICE Sampler tab:

1. Specify one or more Mictor board HapsTrak connector locations by clicking on the connector/connector set.
2. When finished, click the OK button at the bottom of the tab.

Instrumenting the Real-Time Debug Signals

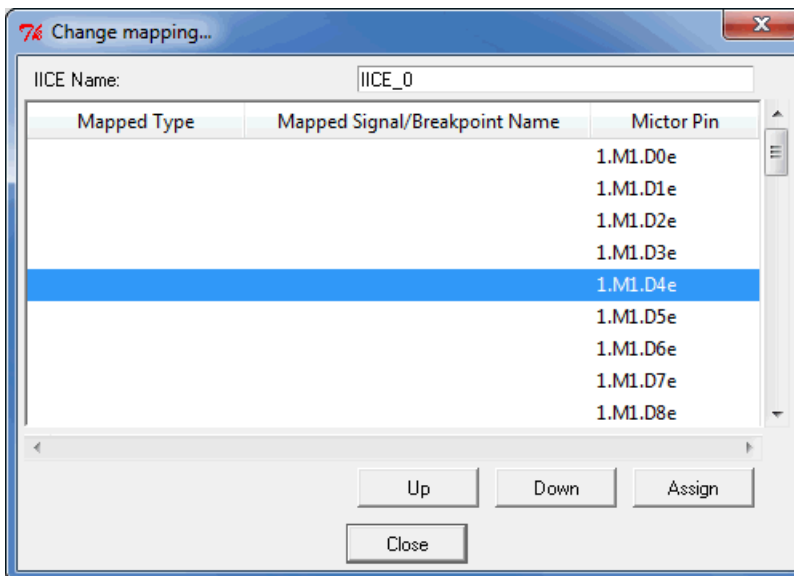
Instrumenting signals for real-time debugging is similar to normal instrumentation in that signal watchpoints and breakpoints are identified and activated in the instrumentation window. The exception is that the only watchpoint selection available from the popup menu for real-time debugging signals is Sample only.



Viewing the Signal Assignments



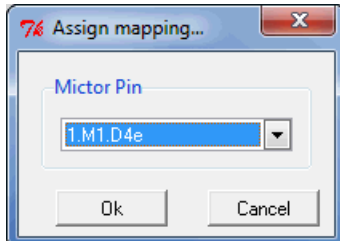
Watchpointed signals are automatically assigned to the specified Mictor daughter board pin locations. These assignments are listed in the Change mapping dialog box. To display the dialog box, click the 'RTD' icon (Change 'RTD' type IICE signals/breakpoints mapping) in the top menu bar.



Individual assignments can be changed by highlighting the assignment to be changed and then either:

- moving the assignment up or down using the Up or Down buttons.

- clicking the Assign button to display the Assign mapping dialog box and then selecting the new pin location from the drop-down menu and clicking OK.



Logic Analyzer Interface

The logic analyzer interface at the Mictor connector is configured in the debugger (see [Logic Analyzer Interface Parameters](#), on page 54 in the *Debugger User Guide*).

Writing the Instrumented Design

To create the instrumented design, you must first complete the following steps:

1. Define a synthesis project with all source files defined
2. Specify IICE parameters/HAPS settings
3. Select signals to sample
4. Select breakpoints to instrument
5. Optionally include the original HDL source

Note: To include the original HDL source with the project, select Options->Instrumentation preference from the menu to display the Instrumentation Preferences dialog box and select the Save original source in instrumentation directory check box. If the original source is to be encrypted, additionally select the Use encryption check box. Selecting Save original source in instrumentation directory saves the original HDL source to the orig_sources subdirectory in the instrumentation directory when you instrument your design.



Finally, click on the Save project's active instrumentation icon to capture your instrumentation. Saving a project's instrumentation generates an *instrumentation design constraints* (idc) file and adds compiler pragmas in the form of constraint files to the design RTL for the instrumented signals and break points. This information is then used by the synthesis tool to incorporate the instrumentation logic (IICE and COMM blocks) into the synthesized netlist. If you are including an encrypted HDL source (Use encryption box checked), you are first prompted to supply a password for the encryption.

Including Original HDL Source

Including the original HDL source with the instrumented project simplifies 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.

As explained in [Debugging on a Different Machine, on page 44](#) in the *Debugger User Guide*, you can simply copy the entire implementation directory to the debug system; the Identify project will be able to locate the original sources for display. To include the original HDL source, select the Save original source in instrumentation directory check box from the Instrumentor Preferences dialog box (Options->Instrumentation preferences) before you save and instrument your project. The clear text or encrypted source is included in the orig_sources subdirectory.

When the Use encryption check box is additionally selected, the original sources are encrypted when they are written into the orig_sources subdirectory. The encryption is based on a password that is requested when you write out the instrumented project. Encryption allows you to debug on a machine that you feel would not be sufficiently secure to store your sources. After you transfer the instrumentation directory to the unsecure machine, you are prompted to reenter the encryption password when you open the project in the debugger.

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

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

The decrypted files are never written to the unsecure machine's hard disk. Users are discouraged from transferring the `instr_sources` directory to the unsecure machine, as this directory is not required for debugging.

Synthesizing Instrumented Designs

When you save your instrumentation, the following files and subdirectories are updated or created in the synthesis project implementation directory (*projectName/rev_n*):

- an `identify.idc` file describing the instrumented watchpoints and break-points; this file can be manually edited to add additional watchpoints (see [Instrumenting Signals Directly in the `idc` File](#), on page 58).
- `instr_sources` subdirectory containing:
 - the IICE core file (`syn_dics.v`)
 - a Synopsys FPGA constraints file (`syn_dics.sdc`) – this file is used directly by the synthesis tool
 - a Synopsys FPGA compiler design constraints file (`syn_dics.cdc`)
- an optional `orig_sources` subdirectory containing the original HDL files in either encrypted or clear-text format
- an `identify.db` encrypted data file

After the files are updated, the compiler reads the compiler design constraints and adds the appropriate hyper-source attributes to the SRS. The instrumentation design constraints are then read and inserted into the instrumentation logic. The instrumented logic is now visible in the SRM view; the SRS view is not required as the RTL no longer includes the instrumented source code.

Note: When instrumenting a VHDL file that is not compiled into the work library, make sure that the `syn_dics.vhd` file is included in the synthesis project ahead of the user design files. Additionally, this file must be compiled into the work library.

Multi-FPGA Debug

A multi-FPGA debug flow is available when using the Certify partition-driven synthesis tool which allows a design to be partitioned among multiple FPGAs and subsequently debugged through a single Identify debugger session. For information on using this flow, see *Using the Multi-FPGA Debug Flow* in the *Identify Interface* section of the *Certify User Guide*.

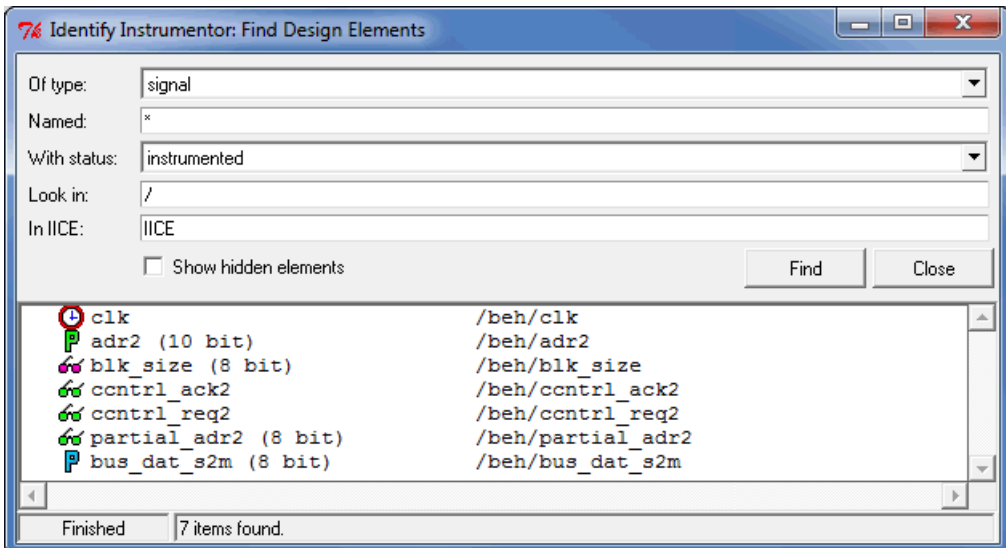
Listing Signals

The instrumentor includes a set of menu commands and, in most cases, icons for listing watchpoint and breakpoint conditions.

List Instrumented Signals



To view all of the signals currently instrumented in the entire design, use the Actions->Show Instrumented Signals menu selection or click the Show Instrumented Signals icon. The result of listing the signals is displayed in the Find dialog box.



List All Signals

To view all of the signals in the design, use the Actions->Show All Signals menu selection.

List Signals Available for Instrumentation



To see only the signals in the design available for instrumentation, use the Actions->Show Possible Signals menu selection or the Show Possible Signals icon.

List Instrumented Breakpoints



To list all of the breakpoints that have been instrumented, use the Actions->Show Instrumented Breakpoints menu selection or the Show Instrumented Breakpoints icon.

List All Breakpoints

To list all breakpoints, use the Actions->Show All Breakpoints menu selection.

List Breakpoints Available for Instrumentation



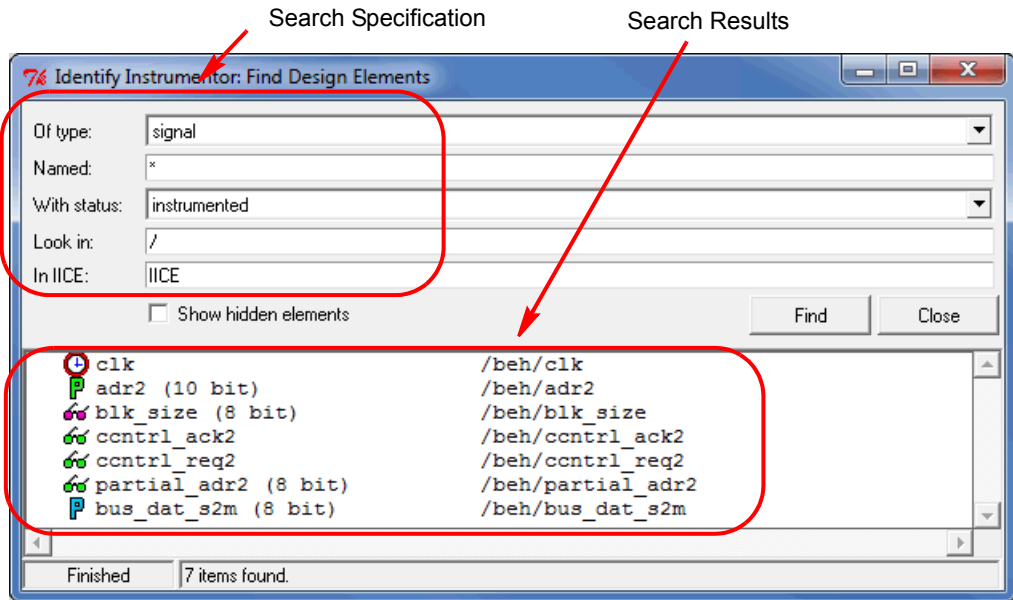
To list all of the breakpoints that are available for instrumentation, use the Actions->Show Possible Breakpoints menu selection or the Show Possible Breakpoints icon.

Searching for Design Objects



The Find dialog box is a general utility to search for signals, breakpoints, and/or instances. To invoke the Find dialog box, use the Edit->Find menu selection or the Display find dialog icon.

The Find dialog box includes an area for specifying the objects to find and an area for displaying the results of the search.



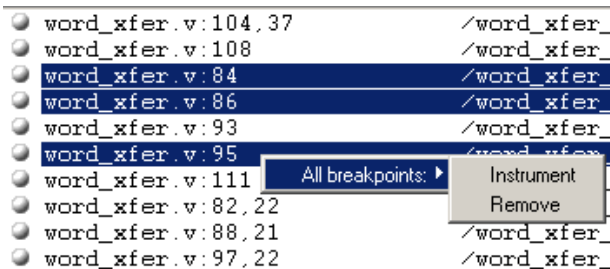
The search specification includes these options:

- **Of type:** – specify which type of object to search for: breakpoint, signal, instance, or “*” (any). The default is “*” (any).
- **Named:** – specify a name, or partial name to search for in the design. Wild cards are allowed in the name. The default is “*” (any).
- **With status:** – specify the status of the object to be found. The values can be instrumented, sample trigger, sample_only, trigger_only, not-instrumented, or “*” (any). The default is “*” (any).
- **Look in:** – specify the location in the design hierarchy to begin the recursive search. The root (“/”) is the default setting.
- **In IICE:** – a read-only field that indicates which IICE is to be searched when multiple IICEs are defined. To search another IICE, close the Find dialog box, click the desired IICE tab, and reopen the Find dialog box.

The search specification also includes a **Show hidden elements** check box. When this check box is enabled, the search also includes objects that would not normally be searched such as breakpoints in dead code.

The search results window shows each object found along with its hierarchical location. In addition, for breakpoints and signals, the results window includes the corresponding icon (watchpoint or breakpoint) that indicates the instrumentation status of the qualified signal or breakpoint.

To change the instrumentation status of a signal, click directly on the watchpoint icon and select the instrumentation type from the popup menu. You can use the Ctrl and Shift keys to select multiple signals and then apply the change to all of the selected signals. To toggle the instrumentation status of a breakpoint, click the breakpoint icon. You also can use the Ctrl and Shift keys to select multiple breakpoints and then apply the change to all selected breakpoints from the popup menu.



Console Text

To capture all text written to the console, use the log console command (see the *Reference Manual*). Alternately, you can click the right mouse button inside the console window and select Save Console Output from the menu.

To capture all commands executed in the console window use the transcript command (see the *Reference Manual*).

To clear the text from the console, use the clear command or click the right mouse button from within the console window and select Clear from the menu.

Index

A

always-armed triggering 16

B

black boxes 44

breakpoint icon
color coding 62

breakpoints
finding 70
in folded hierarchy 61
instance selection 62
listing all 70
listing available 70
listing instrumented 70
selecting 60

buses
instrumenting partial 51

C

clocks
edge selection 18
sample 17

complex triggering 20

Configure IICE dialog box 14
IICE Controller tab 19, 21
IICE Sampler tab 14

console window 46

console window operations 72

D

designs
writing instrumented 66

devices
supported families 11

dialog boxes
Configure IICE 14

directories
instrumentation 68

E

encrypting source files 67
essential signal database 58

F

files
encrypting source 67
idc 58
IICE core 68
project 8
script 48
folded hierarchy 56

H

hardware
skew-resistant 12
HDL source
including in project 67
hierarchy
folded 56
hierarchy browser
popup menu 43
hierarchy browser window 42

I

idc file
editing 58
IICE
configuration 9
technology settings 11
IICE Controller tab 19, 21
IICE core file
compiling 68

IICE parameters

- buffer type [15](#)
- common [11](#)
- individual [14](#)
- JTAG port [12](#)

IICE Sampler tab [14](#)

IICE selection

- multi-IICE [15](#), [19](#)

IICE settings

- sample clock [17](#)
- sample depth [16](#)

instances

- finding [70](#)

instrumentation

- partial records [53](#)

instrumentation directory [68](#)

instrumenting partial buses [51](#)

J

JTAG port

- IICE parameter [12](#)

L

limitations

- Verilog instrumentation [34](#), [37](#)
- VHDL instrumentation [32](#)

M

mixed language considerations [31](#)

multi-IICE

- tabs [14](#)

multiplexed groups

- assigning [55](#)

O

objects

- finding [70](#)

original source

- including [67](#)

P

parameterized modules

instrumenting [58](#)

parameters

- IICE [9](#)
- IICE common [11](#)

partial buses

- instrumenting [51](#)

passwords

- encryption/decryption [67](#)

project files [8](#)

projects

- instrumenting [7](#)

Q

qualified sampling [16](#)

R

RAM resources [15](#)

records

- partially instrumented [53](#)

resource estimation [46](#)

S

sample clock [17](#)

sample clock calculation

- SRAM [29](#)

sampling

- in folded hierarchy [56](#)
- qualified [16](#)

sampling

- signals [49](#), [56](#), [60](#), [63](#), [65](#), [66](#), [67](#), [69](#), [70](#)

script file [48](#)

searches

- objects [70](#)

settings

- IICE technology [11](#)
- sample clock [17](#)
- sample depth [16](#)

signals

- disabling sampling [50](#)
- exporting trigger [20](#)
- finding [70](#)
- instance selection [57](#)
- listing all [70](#)

- listing available [70](#)
- listing instrumented [69](#), [70](#)
- sampling
 - selection [49](#), [56](#), [60](#), [63](#), [65](#), [66](#), [67](#), [69](#), [70](#)
- simple triggering [20](#)
- skew-resistant hardware [12](#)
- source files
 - encrypting [67](#)
- SRAM clocks [28](#)
- state-machine triggering [20](#)
- synthesizing designs [68](#)

T

- technology settings
 - IICE [11](#)
- trigger signal
 - exporting [20](#)
- triggering
 - always-armed [16](#)
 - complex [20](#)
 - simple [20](#)
 - state machine [20](#)

V

- Verdi platform [58](#)
- Verilog
 - instrumentation limitations [34](#), [37](#)
- VHDL
 - instrumentation limitations [32](#)

W

- watch icon
 - color coding [57](#)
- windows
 - console [46](#)
 - hierarchy browser [42](#)

