

Synopsys

Identify[®] Microsemi Edition Instrumentor User Guide

January 2018

SYNOPSYS[®]

Copyright Notice and Proprietary Information

© 2018 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.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Third-Party Links

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

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

January 2018

Contents

Chapter 1: Design Setup

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

Chapter 2: ICE Configuration

Multiple ICE Units	10
Common ICE Parameters	11
Individual ICE Parameters	16
ICE Sampler Tab	17
ICE Clock Tab	18
ICE Controller Tab	20
ICE Options Tab	21
RTD Tab	22

Chapter 3: Using the Instrumentor

Instrumentor Windows and Views	23
Control Panel	24
Search Panel	24
Hierarchy Browser	25
RTL Tab	27
Instrumentation Tab	27
Commands and Procedures	29
Opening Designs	30
Selecting Signals for Data Sampling	30
Instrumenting Buses	32
Partial Instrumentation	35
Multiplexed Groups	36
Sampling Signals in a Folded Hierarchy	37
Instrumenting the Verdi Signal Database	39
Instrumenting Signals Directly in the idc File	40
Selecting Breakpoints	42
Selecting Breakpoints Residing in Folded Hierarchy	42

Configuring the IICE	44
Real-time Debugging	44
Writing the Instrumented Design	48
Synthesizing Instrumented Designs	50
Listing Signals	50
Searching for Design Objects	51
Capturing Commands from the Tcl Script Window	52

Chapter 4: HAPS Deep Trace Debug

External Memory Instrumentation and Configuration	54
DTD Tab	55
Running Deep Trace Debug with SRAM Memory	57
SRAM Clocks	58
Sample Depth Calculation	58
Sample Clock Calculation	59

Chapter 5: Support for Instrumenting HDL

VHDL Instrumentation Limitations	62
Verilog Instrumentation Limitations	64
SystemVerilog Instrumentation Limitations	67

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 *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, which is then 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 3, Using the Instrumentor](#), the design is instrumented and saved. To save your instrumented design, select File->Save All from the main menu.

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 or “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 Preferences dialog box 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 the Edit IICE dialog box tabs. Tabs are available to support real-time debugging with a Mictor card, and the cross-triggering feature.

- [IICE Sampler Tab](#) – includes sample depth selection and defines the external memory configuration.
- [IICE Clock Tab](#) – defines the sample clock and clock edge.
- [IICE Controller Tab](#) – includes complex counter trigger width specification and selection of state machine triggering.
- [IICE Options Tab](#)– controls trigger-signal export and cross triggering.

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.

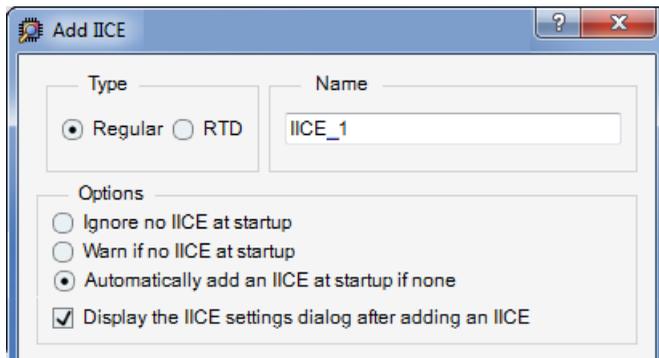
Multiple IICE Units

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



The instrumentor graphical window includes an Add IICE icon in the instrumentor menu bar to define an additional IICE unit for the current design. You can also select IICE->Add IICE from the Instrumentor drop-down menu in the main menu bar. Either action opens the Add IICE dialog box to allow you to define the type and name of the new IICE unit.

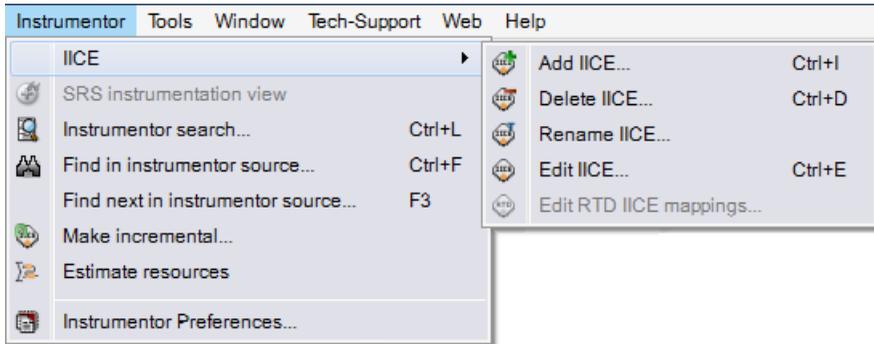


When you click OK, the HDL source code in the RTL window is redisplayed without any signals instrumented, the Instrumentation window is cleared, and the IICE selection reported in the status panel on the left is updated with the name of the IICE unit. When creating a new IICE unit:

- Select Regular (the default) to add a normal IICE unit or select RTD to add a HAPS-based real-time debug IICE unit; see [Real-time Debugging, on page 44](#).
- Optionally enter a name for the IICE unit in the Name field. By default, the IICE name is formed by adding an *_n* suffix to IICE (for example, IICE_0, IICE_1, etc.).

Deleting an IICE Unit

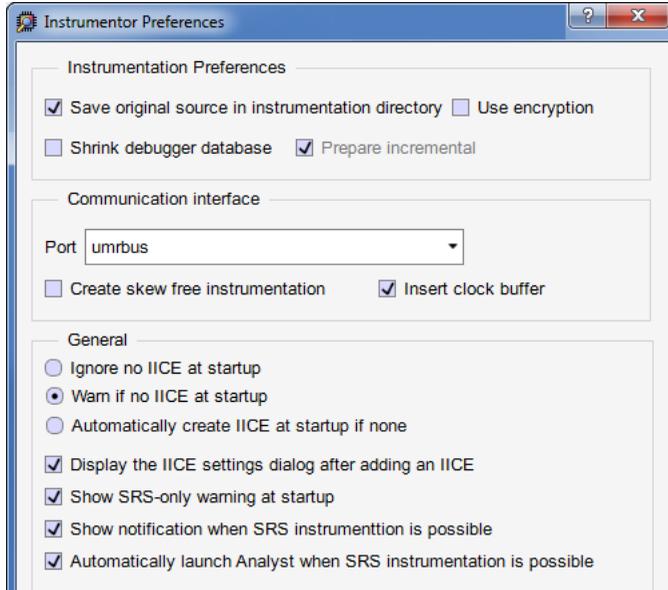
To delete an IICE unit from the design, either select Instrumentor->IICE from the top menu bar and select Delete IICE or click on the Delete IICE icon in the instrumentor graphics window.



When more than one IICE unit is defined, select the IICE unit in the Control Panel panel before you select Delete IICE (see [Individual IICE Parameters](#), on page 16).

Common IICE Parameters

The IICE parameters common to all IICE units in a multi-IICE configuration include the communication port setting and if the optional skew-free hardware is to be used. The parameters are set on the Instrumentor Preferences dialog box (select Instrumentor->Instrumentor Preferences from the top menu bar).



Instrumentation Preferences

The Instrumentation Preferences section includes the following check boxes:

- Save original source in instrumentation directory – Includes the original HDL source with the exported design files when checked.
- Use Encryption – Encrypts the original source.
- Shrink debugger database – when checked, the design database includes only instrumented hierarchies; when left unchecked, the full design database is loaded into the debugger.

Communication interface

The Communication interface section includes the following parameters and check boxes:

- Port – specifies the type of connection to be used to communicate with the on-chip IICE. The connection types available from the drop-down menu are:
 - builtin – indicates that the IICE is connected to the JTAG Tap controller available on the target device.

UJTAG Wrapper Support for Microsemi FPGA Devices

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

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

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

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

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

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

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

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

To resolve the error:

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

- `soft` – indicates that the Synopsys Tap controller is to be used. The Synopsys FPGA Tap controller is more costly in terms of resources because it is implemented in user logic and requires four user I/O pins to connect to the communication cable.
- `umrbus` – indicates that the IICE is connected to the target device through the UMRBus.
- Create Skew Free Instrumentation

The Create skew free instrumentation check box, when checked, incorporates skew-free 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-free 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.

- Insert clock buffer – when using the JTAG interface, enabling the check box automatically inserts two global clock buffers to the debug ip.

See [Chapter 3, *Connecting to the Target System*](#) in the *Debugger User Guide*, for a description of the communication interface.

General

The General section includes the following parameters and check boxes:

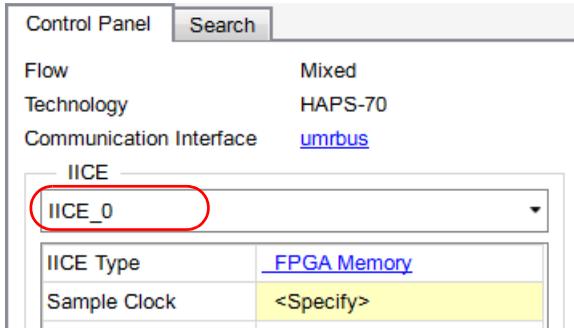
- Startup Controls – a set of start-up radio buttons:
 - Ignore no IICE at startup – take no action when opening the instrumentor and there is no IICE defined
 - Warn if no IICE at startup – issue a warning when opening the instrumentor and there is no IICE defined
 - Automatically create IICE at startup if none – create an IICE when opening the instrumentor and there is no IICE defined
- Display the IICE settings dialog after adding an IICE – after creating an IICE (either in the Add IICE dialog box or automatically at startup), display the IICE Settings dialog box.
- Show SRS-only warning at startup – when opening the instrumentor, issue a warning if only SRS instrumentation is possible.
- Show notification when SRS instrumentation is possible
- Automatically launch Analyst in SRS-only mode – When opening the instrumentor, automatically launch the HDL Analyst when SRS instrumentation is possible.

Individual IICE Parameters

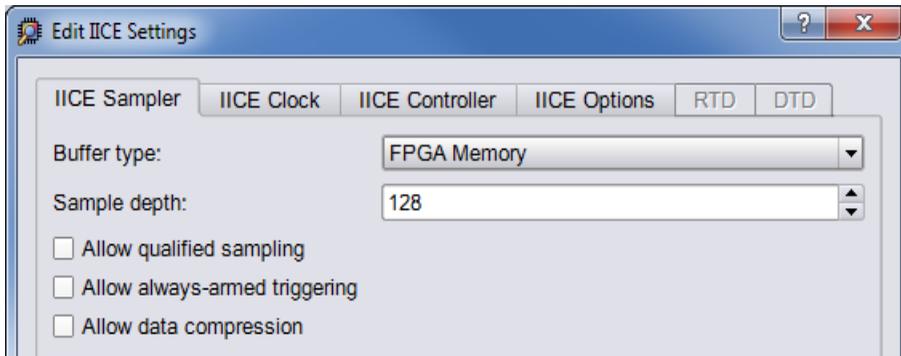


The individual parameters for each IICE are defined on a series of tabs of the Edit IICE Settings dialog box. Before you display this dialog box, make sure that the name of the target IICE unit appears in the Control Panel tab.

With the target IICE selected, either click the Edit IICE icon in the top menu



bar or click the entry for the IICE Type field in the Control Panel to display the Edit IICE Settings dialog box.



IICE Sampler Tab

The IICE Sampler tab shown above is the default tab and defines:

- Buffer type
- Sample depth
- Sampling/triggering options

Buffer Type

The Buffer type parameter specifies the type of RAM to be used to capture the on-chip signal data. The default value is FPGA Memory; the hapsram setting configures the IICE to use extended daughter board SRAM memory. For more information, see [Chapter 4, 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. The sample depth increases significantly with the deep-trace debug feature.

Allow Qualified Sampling

The Allow qualified sampling check box, when checked, causes the instrumentor to build an IICE block that is capable of performing qualified sampling. When qualified sampling is enabled, one data value is sampled each time the trigger condition is true. With qualified sampling, you can follow the operation of the design over a longer period of time (for example, you can observe the addresses in a number of bus cycles by sampling only one value for each bus cycle instead of a full trace). Using qualified sampling includes a minimal area and clock-speed penalty. For more information on qualified sampling, see [-qualified_sampling 0/1, on page 59](#).

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

IICE Clock Tab

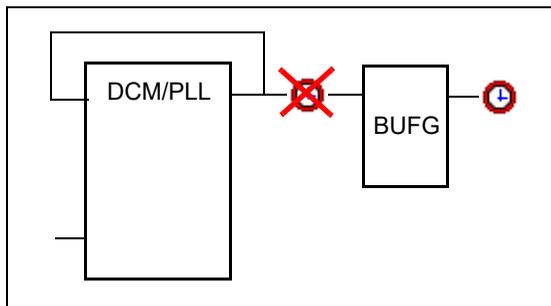
The IICE Clock tab defines the sample clock.



Sample Clock

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

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



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

```

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

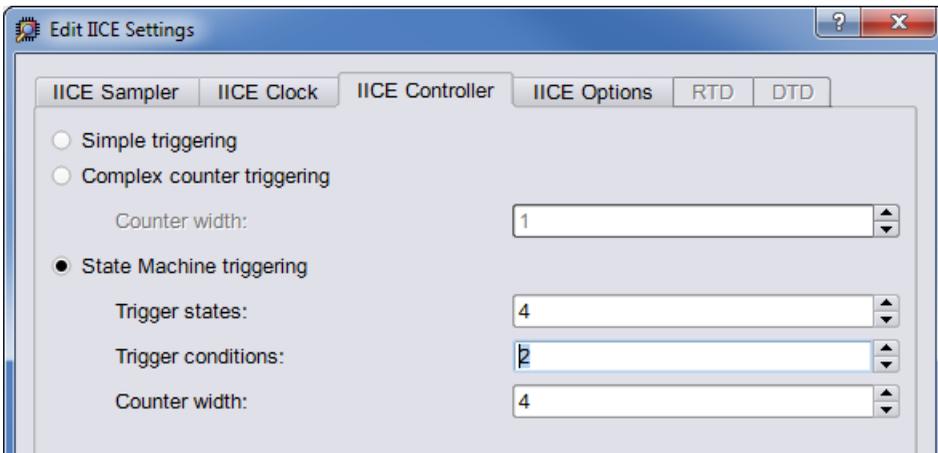
Note: The sample clock edge can only be set from the IICE Clock tab.

Clock Edge

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

IICE Controller Tab

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



Simple Triggering

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

Complex-Counter Triggering

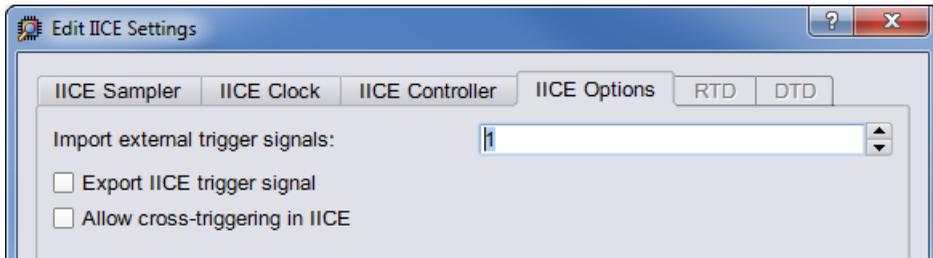
Complex-counter triggering augments the simple triggering by instrumenting a variable-width counter that can be used to create a more complex trigger function. Use the width setting to control the desired width of the counter.

State-Machine Triggering

State-machine triggering allows you to pre-instrument a variable-sized state machine that can be used to specify an ultimately flexible trigger condition. Use Trigger states to customize how many states are available in the state machine. Use Trigger condition to control how many independent trigger conditions can be defined in the state machine. For more information on state-machine triggering, see [State Machine Triggering, on page 63](#) in the *Debugger User Guide*.

IICE Options Tab

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



Import External Trigger Signals

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

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

Export IICE Trigger Signal

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

Allow cross-triggering in IICE

The Allow cross-triggering in IICE check box, when checked, allows this IICE unit to accept a cross-trigger from another IICE unit. For more information on cross-triggering, see [Cross Triggering, on page 35](#) in the *Debugger User Guide*.

RTD Tab

The RTD tab is active when the selected IICE unit is a type real-time debug. 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. For real-time debug configuration and setup information, see [Real-time Debugging, on page 44](#).

CHAPTER 3

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

The remainder of this chapter describes:

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

Instrumentor Windows and Views

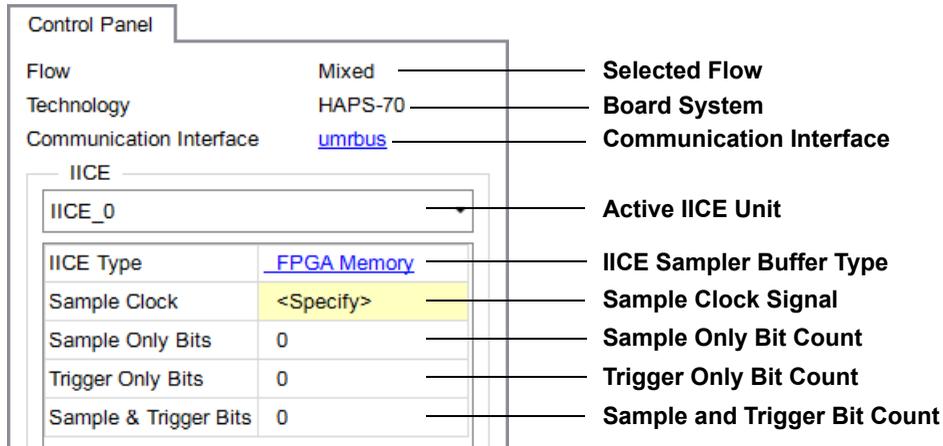
The Graphical User Interface (GUI) for the instrumentor is divided into five major areas:

- [Control Panel](#)
- [Search Panel](#)
- [Hierarchy Browser](#)
- [RTL Tab](#)
- [Instrumentation Tab](#)

In this section, each of these areas and their uses is described.

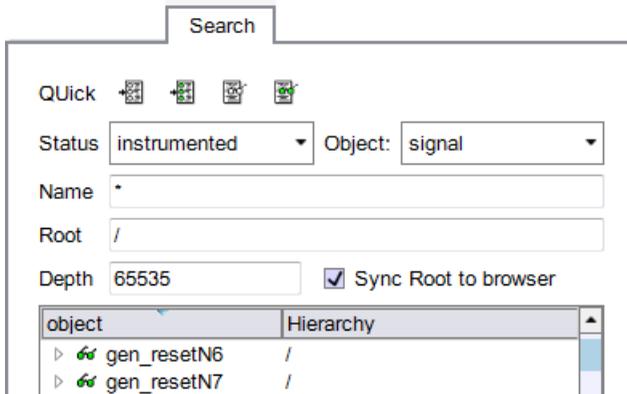
Control Panel

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



Search Panel

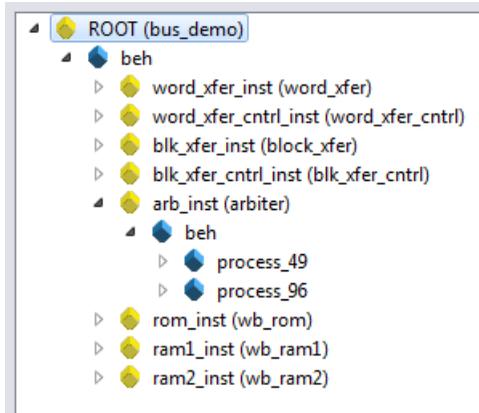
The Search panel is a general utility to search for signals, breakpoints, and/or instances. The panel includes an area for specifying the objects to find and an area for displaying the results of the search. For detailed information on using the Search panel, see [Searching for Design Objects, on page 51](#).



Hierarchy Browser

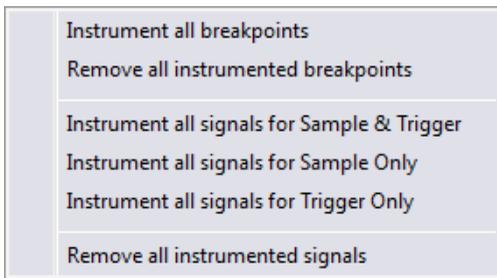
The hierarchy browser shows a graphical representation of the design hierarchy. At the top of the browser is the ROOT node. The ROOT node represents the top-level entity or module of your design. For VHDL designs, the first level below the ROOT is the architecture of the top-level entity. The level below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

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



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

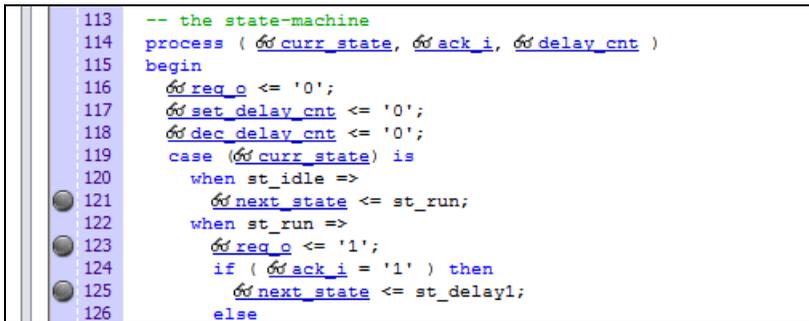


The selected operation is applied to all breakpoints or signal watchpoints at the selected level of hierarchy. You cannot instrument signals when a sample clock is included in the defined group.

Black-box modules are represented by a black icon, and their contents can not be instrumented. Also, certain modules cannot be instrumented (see [Chapter 5, Support for Instrumenting HDL](#), for a specific description).

RTL Tab

The RTL tab displays the HDL source code and is the default view when you launch the instrumentor. The code 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.

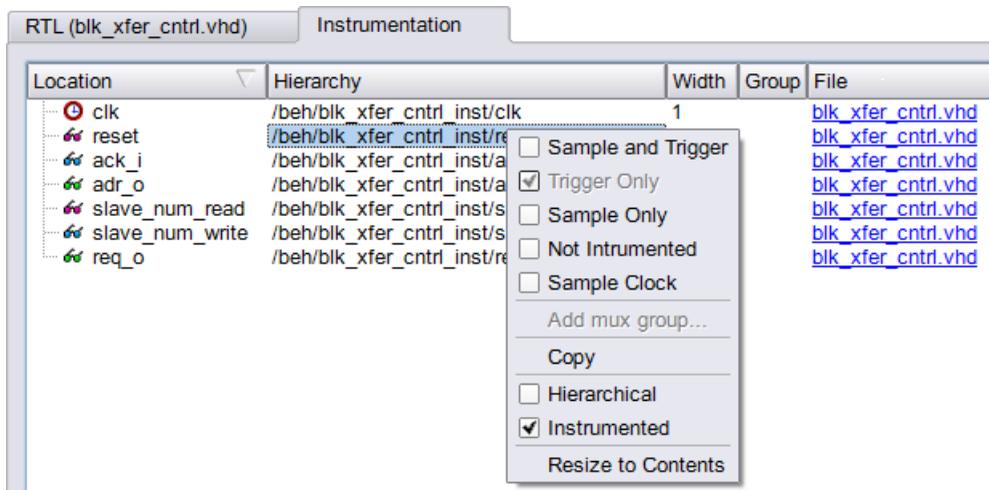
A screenshot of the RTL tab in the instrumentor. It shows a Verilog code snippet for a state machine. The code is displayed in a window with a light blue background. The text is as follows:

```
113 -- the state-machine
114 process ( curr_state, ack_i, delay_cnt )
115 begin
116     req_o <= '0';
117     set_delay_cnt <= '0';
118     dec_delay_cnt <= '0';
119     case (curr_state) is
120     when st_idle =>
121         next_state <= st_run;
122     when st_run =>
123         req_o <= '1';
124         if ( ack_i = '1' ) then
125             next_state <= st_delay1;
126         else
```

The code is annotated with small circular icons in the left margin next to line numbers 121, 123, and 125. The signals curr_state, ack_i, delay_cnt, req_o, set_delay_cnt, dec_delay_cnt, and next_state are underlined and colored blue.

Instrumentation Tab

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



Commands and Procedures

The following sections describe basic instrumentor commands and procedures.

- [Opening Designs](#), on page 30
- [Selecting Signals for Data Sampling](#), on page 30
- [Instrumenting Buses](#), on page 32
- [Partial Instrumentation](#), on page 35
- [Multiplexed Groups](#), on page 36
- [Sampling Signals in a Folded Hierarchy](#), on page 37
- [Instrumenting the Verdi Signal Database](#), on page 39
- [Instrumenting Signals Directly in the idc File](#), on page 40
- [Selecting Breakpoints](#), on page 42
- [Selecting Breakpoints Residing in Folded Hierarchy](#), on page 42
- [Configuring the IICE](#), on page 44
- [Real-time Debugging](#), on page 44
- [Writing the Instrumented Design](#), on page 48
- [Synthesizing Instrumented Designs](#), on page 50
- [Listing Signals](#), on page 50
- [Searching for Design Objects](#), on page 51
- [Capturing Commands from the Tcl Script Window](#), on page 52

Opening Designs

To open the instrumentor from the synthesis tool:

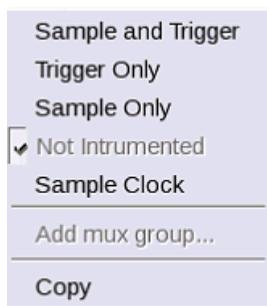
1. Create an Identify implementation by right clicking on an existing synthesis implementation and selecting New Identify Implementation from the popup menu.
2. Set/verify any technology, device mapping, or other pertinent options (see the implementation option descriptions in the *Synopsys FPGA Synthesis*) and click OK. A new, Identify implementation is added to the synthesis tool project view.

Launching the instrumentor displays the design hierarchy and the RTL file content with all the potential instrumentation marked and available for selection.

Selecting Signals for Data Sampling



To select a signal to be sampled, simply click on the watchpoint icon next to the signal name on the RTL tab; 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 19](#)) and bus segments can be individually specified (see [Instrumenting Buses, on page 32](#)). In addition, signals specified as Sample and trigger or Sample only can be included in multiplexed groups as described in [Multiplexed Groups, on page 36](#).

When the watchpoint icon is clear (unfilled), the signal has not been instrumented. The colors of the filled icons are described in the following table:

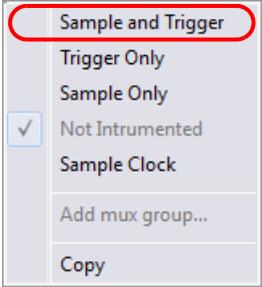
	Red	Signal is enabled for triggering only
	Green	Signal is enabled for both sampling and triggering
	Blue	Signal is enabled for sampling only

The example below shows signal grant1 being enabled for sample and trigger.

```

28 library IEEE;
29 use IEEE.std_logic_1164.all;
30 use IEEE.std_logic_arith.all;
31
32 entity arbiter is
33   port (
34     clk      : in std_logic;
35     clk_sys  : in std_logic;
36     reset    : in std_logic;
37     r1      : in std_logic;
38     r2      : in std_logic;
39     grant1   : out std_logic;
40     grant2   : out std_logic
41   );
42 end arbiter;
43
                
```

Signal "grant1" selected



The TCL Script window at the bottom displays the Tcl command that implements the selection (signals add -iice {IICE} -sample -trigger {/beh/arb_inst/grant1}) and the results of executing the command.

```

$
signals add -iice {IICE} -sample -trigger {/beh/arb_inst/grant1}
added for sampling and triggering to iice: IICE
Total instrumentation in bits: Sample Only 27, Trigger Only 5, Sample and trigger 13
Group wise instrumentation in bits:
Groupdefault:Sample Only 27, Sample and trigger 13
                
```

To disable a signal for sampling or triggering, select the signal on the RTL tab 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 51](#)).

Restrictions

Signals include a watchpoint icon adjacent to the signal name in the instrumentation window to indicate that the signal can be instrumented. The following input and output buffer signals, however, should never be instrumented as they cause an error in the synthesis tool during subsequent mapping:

- Input of IBUF (drives user logic)
- Input of IBUG (drives user logic)
- Output of OBUF (driven by user logic)
- Output of OBUFT (driven by user logic)

Instrumenting Buses

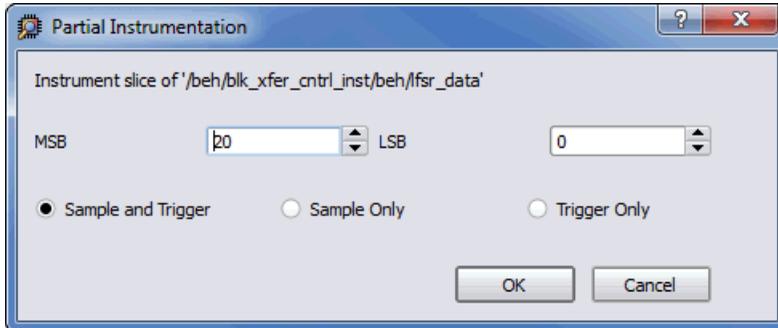
Entire buses, individual bits, or groups of bits of a bus can be individually instrumented.

- [Instrumenting a Partial Bus](#), on page 32
- [Instrumenting Single Bits of a Bus](#), on page 34
- [Instrumenting Non-Contiguous Bits or Bit Ranges](#), on page 34
- [Changing the Instrumentation Type](#), on page 34

Instrumenting a Partial Bus

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

1. Place the cursor over a bus that is not fully instrumented and select Add Partial Instrumentation to display the following dialog box.



- 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 34.

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.

- Select the type of instrumentation for the specified bit range from the radio buttons 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).

```

41 port (
42     Ⓞ clk           : in
43     Ⓞ reset        : in
44     Ⓟ adr_o        : out
45     Ⓞ block_size   : out

```

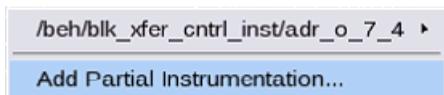
Instrumenting Single Bits of a Bus

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.



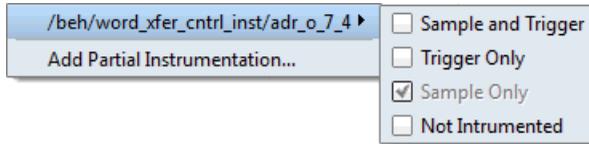
3. Specify the bit or bit range to be instrumented as previously described, select the type of instrumentation from the drop-down menu, and click OK. If the type of instrumentation is different from the existing instrumentation, the letter “P” will be yellow to indicate a mixture of instrumentation types.

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 on the RTL tab or through the Instrumentor Search 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 – all fields of the record are instrumented for sample and trigger
- Blue – all fields of the record are instrumented for sample only
- Red – all fields of the record are instrumented for trigger only
- Yellow – 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 <= r2;
35   P tt.c2 <= c2;
36   P tt.ele.r1 <= r1;
37   P tt.ele.c1 <= c1;
38

```

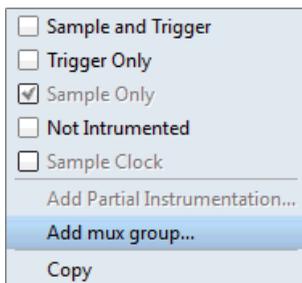
The Search Panel also uses the partial instrumentation icon to show the state of instrumentation on fields of partially instrumented records (see [Searching for Design Objects, on page 51](#)).

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

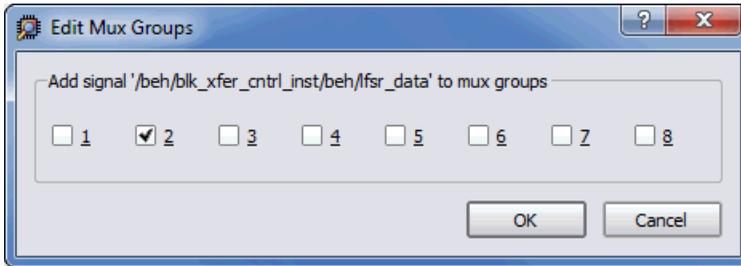
Multiplexed Groups

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 pop-up 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 75 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 21 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 within the hierarchy.



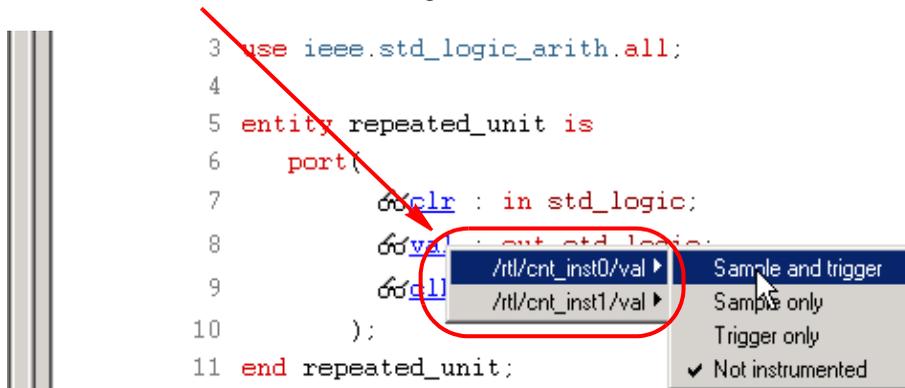
The choices are displayed in terms of an absolute signal path name originating at the top-level entity or module. The list of choices for a particular signal is accessed by clicking the watchpoint icon or corresponding signal.

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 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 22](#) and [Displaying Data from Folded Signals, on page 31](#) 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.

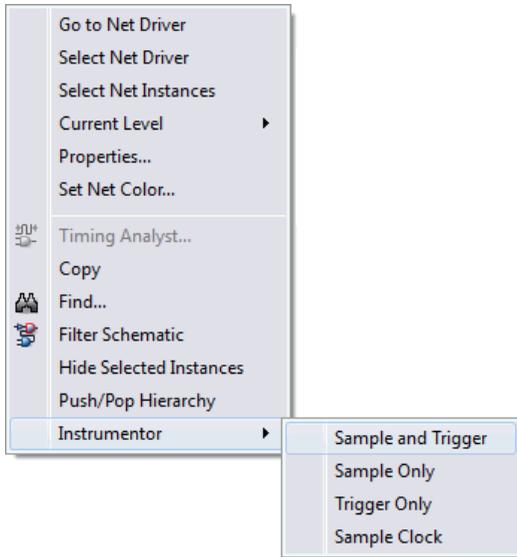
4. Instrument the sample clock (a sample clock is required by the instrumentor).
5. Configure the IICE and instrument the design.

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 50](#) 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 HDL Analyst (SRS file) 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 using this technique:

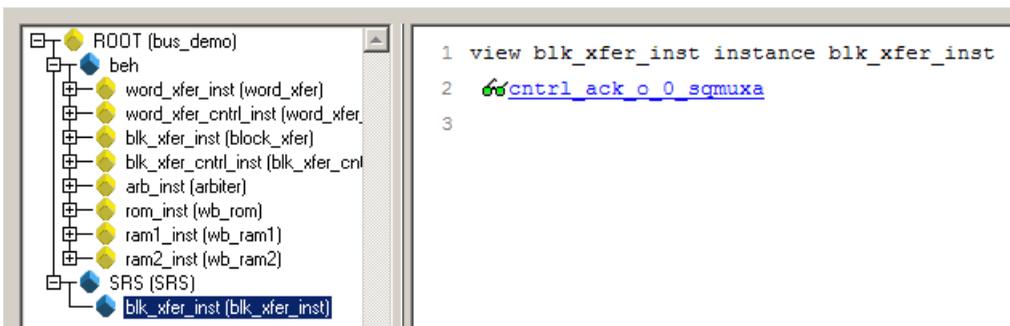
1. Save the instrumented design in the synthesis tool to create the IDC file.
2. Open the RTL view (SRS file) by clicking the SRS instrumentation view icon.
3. In the schematic view, highlight the net of the signal to be instrumented.
4. With the net highlighted, click the right mouse button, select Instrumentor from the popup menu, and select the type of instrumentation to be applied.



When selected, the instrumentation is automatically applied to the open instrumentor view.

5. Save the updated instrumentation and rerun compilation.

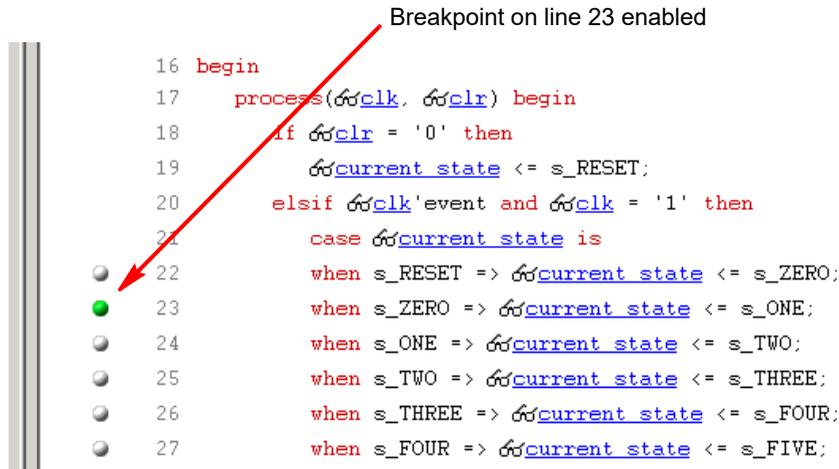
When you open the debugger, an SRS entry is included in the hierarchy browser; selecting this entry displays the additional signal or signals added to the instrumentation. Selecting a signal in the instrumentation window brings up the Watchpoint Setup dialog box to allow a trigger expression to be assigned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



Note that trigger expressions on signals added to the instrumentation 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.



```

16 begin
17   process(s_clk, s_clr) begin
18     if s_clr = '0' then
19       s_current_state <= s_RESET;
20     elsif s_clk'event and s_clk = '1' then
21       case s_current_state is
22         when s_RESET => s_current_state <= s_ZERO;
23         when s_ZERO => s_current_state <= s_ONE;
24         when s_ONE => s_current_state <= s_TWO;
25         when s_TWO => s_current_state <= s_THREE;
26         when s_THREE => s_current_state <= s_FOUR;
27         when s_FOUR => s_current_state <= s_FIVE;

```

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

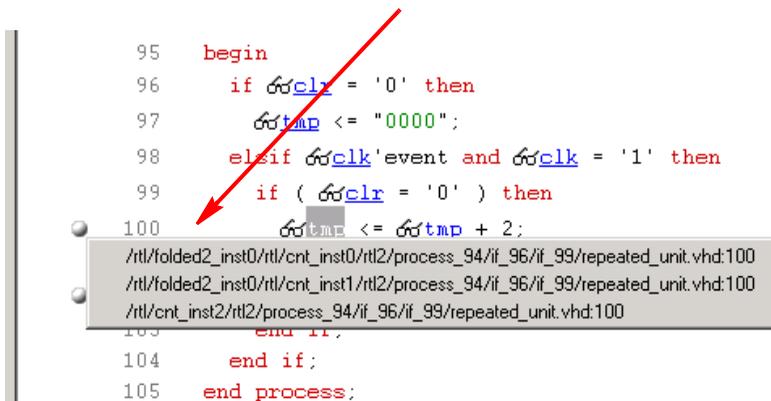
If a design contains entities or modules that are instantiated more than once, the design is termed to have *folded* hierarchy. By definition, there will be more than one instance of every breakpoint in a folded entity or module. To allow you to instrument a particular instance of a folded breakpoint, 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 the corresponding line entry in the list displayed.

Folded breakpoint on line 100 selected



```

95     begin
96         if  $\text{clk}$  = '0' then
97              $\text{tmp}$  <= "0000";
98         elsif  $\text{clk}$ 'event and  $\text{clk}$  = '1' then
99             if (  $\text{clr}$  = '0' ) then
100                 $\text{tmp}$  <=  $\text{tmp}$  + 2;
                /rtl/folded2_inst0/rtl/cnt_inst0/rtl2/process_94/if_96/if_99/repeated_unit.vhd:100
                /rtl/folded2_inst0/rtl/cnt_inst1/rtl2/process_94/if_96/if_99/repeated_unit.vhd:100
                /rtl/cnt_inst2/rtl2/process_94/if_96/if_99/repeated_unit.vhd:100
103            end if;
104        end if;
105    end process;

```

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.

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

```

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

```

See the *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 Edit IICE icon to change them. [Chapter 2, IICE Configuration](#), discusses how to set these parameters for both single- and multi-IICE configurations, and [Chapter 4, HAPS Deep Trace Debug](#) describes setting the parameters for the HAPS deep trace debug feature.

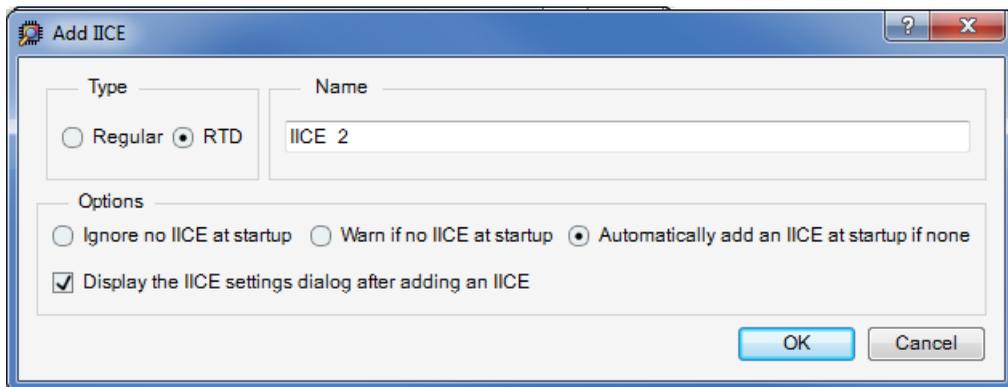
Real-time Debugging

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 special IICE is defined in either the user interface or by command entry in the TCL Script shell.

To specify the IICE from the user interface, click the Add IICE icon to display the following dialog box. Select the RTD radio button and click OK.



To define the IICE from the TCL Script shell, enter the `iice new` command:

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

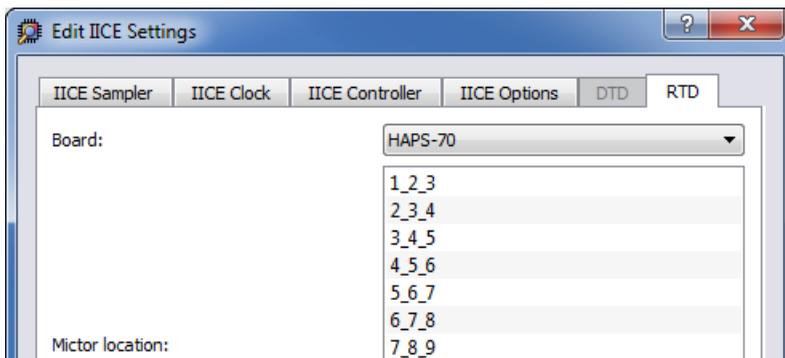
In the command syntax, *iiceID* is the name of the new IICE and, if omitted, defaults to an incremental number (for example, IICE_0).

Either of the above methods creates a new, real-time IICE for the design with all of the signals “not instrumented.”

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

Edit IICE Settings Tab

The Edit IICE Settings tab for the real-time debugging feature includes a drop-down menu for selecting the Mictor daughter card locations. A HapsTrak 3-to-HapsTrak II adapter is required.

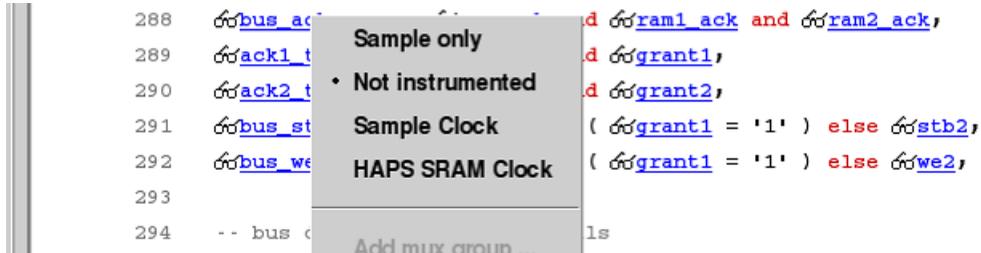


On the Edit IICE Settings tab:

1. Specify the HapsTrak® 3 connector locations for the Mictor board by clicking on the connector set.
2. When finished with the above entries, click the OK button at the bottom of the tab.

Instrumenting the Real-Time Debug Signals

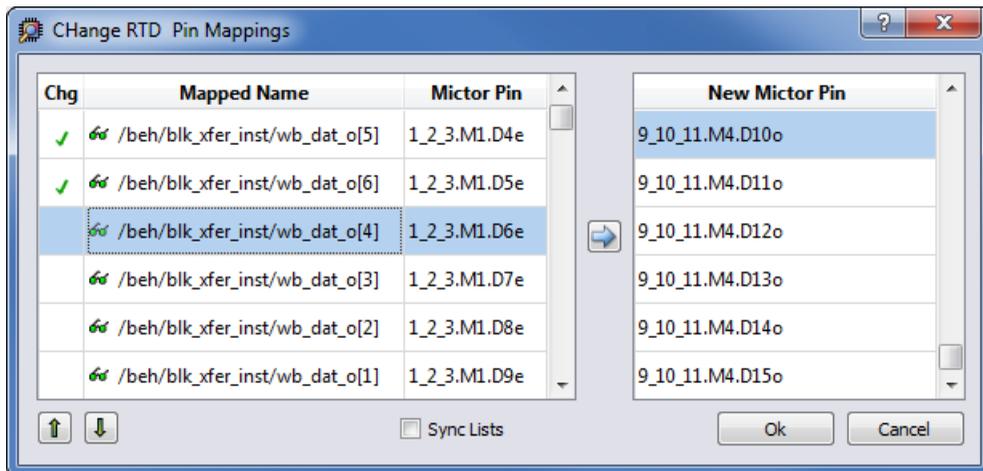
Instrumenting signals for real-time debugging is similar to normal instrumentation in that signal watchpoints and breakpoints are identified and activated in the instrumentation window. The exception is that the only watchpoint selection available from the popup menu for real-time debugging signals is 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 RTD Pin Mappings dialog box. To display the dialog box, click the Edit RTD IICE mappings icon in the top menu bar.



Individual assignments can be changed by:

1. Highlighting the assignment in the left panel (use the Up or Down arrows if necessary).

2. Select the new Mictor pin in the right panel (use the vertical scroll bar if necessary).
3. Click the arrow located between the panels.

Logic Analyzer Interface

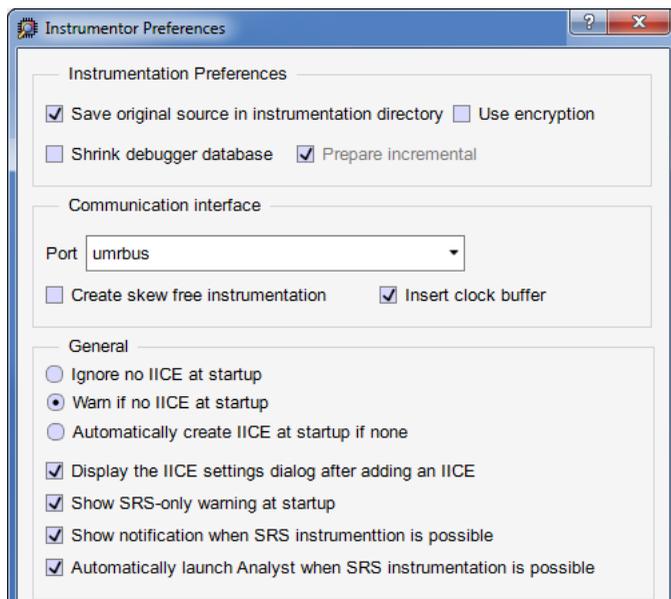
The logic analyzer interface at the Mictor connector is configured in the debugger (see [Logic Analyzer Interface Parameters, on page 51](#) in the *Debugger User Guide*).

Writing the Instrumented Design

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

1. Specify IICE parameters/HAPS settings
2. Select signals to sample
3. Select breakpoints to instrument
4. Optionally include the original HDL source

To include the original HDL source with the exported design files, open the Instrumentor Preferences dialog box and enable the Save original source in instrumentation directory check box. If the original source is to be encrypted, additionally check the Use encryption check box.



Finally, select File->Save All from the main synthesis tool menu 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.

To include the original HDL source, select the Save original source in instrumentation directory check box in the Instrumentor Preferences dialog box (select Instrumentor->Instrumentor Preferences from the main menu to display the dialog box) before you save and instrument your project.

When the Use encryption check box is additionally selected, the original sources are encrypted when they are written. 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 export the files to the unsecure machine, you are prompted to reenter the encryption password when you open the design in the debugger. The decrypted files are never written to the unsecure machine's hard disk.

For maximum security when selecting an encryption password:

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

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

Synthesizing Instrumented Designs

When you save your instrumentation, the synthesis tool creates the set of files and subdirectories required by the debugger. These files and subdirectories are then exported from the database to an external directory location. This location can be local to your system (when running the debugger on the same machine) or the exported directory can be copied to a remote system using tar or file transfer protocol (FTP).

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, select Instrumentor->Instrumentor Search from the top menu bar to display the Instrumentor Search dialog box and then click the Search for instrumented signals icon in the Quick Search area in the upper left corner. The result of listing the signals is displayed in the Instrument Search dialog box.

List Signals Available for Instrumentation



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

List Instrumented Breakpoints



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

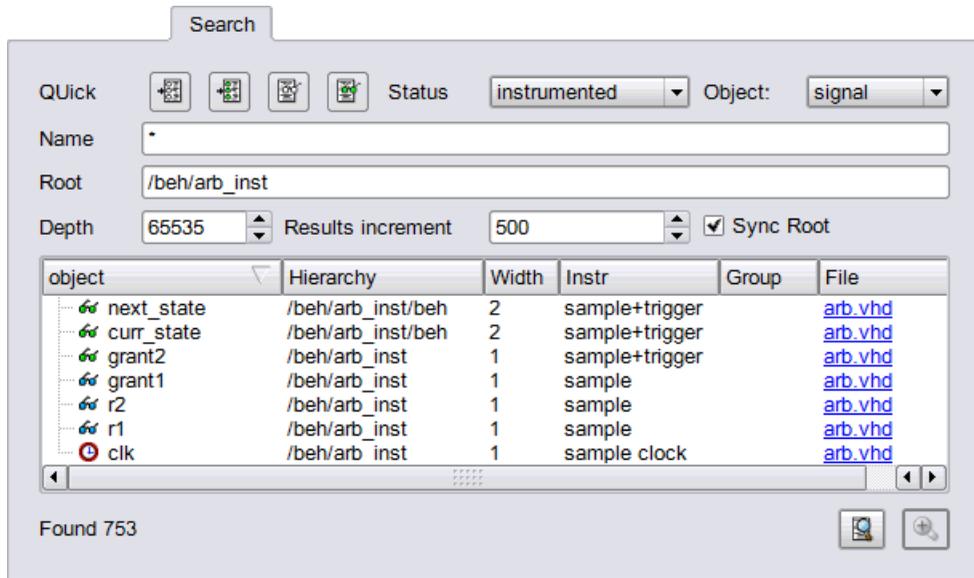
List Breakpoints Available for Instrumentation



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

Searching for Design Objects

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



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

- Quick icons – presets the conditions for instrumented or non-instrumented watchpoints or breakpoints (see [Listing Signals, on page 50](#)).
- Status: – specifies the status of the object to be found from the drop-down menu. The values can be instrumented, sample trigger, sample_only, trigger_only, not-instrumented, or “*” (any). The default is “*” (any).
- Object: – specifies the type of object to search for from the drop-down menu: breakpoint, signal, instance, or “*” (any). The default is “*” (any).
- Name: – specifies a name, or partial name to search for in the design. Wild cards are allowed in the name. The default is “*” (any).
- Root: – specifies the location in the design hierarchy to begin the recursive search. Root (“/”) is the default setting.
- Depth: – specifies the depth of the sample buffer to be searched.

- **Results increment:** – adds *results increment* items to the displayed list. Click the View more search results button to the right of the search button to list more results.
- **Sync Root:** – When checked, synchronizes the root of the search path to the currently selected root in the hierarchy browser on the right.

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

The results area at the bottom of the Search panel is interactive. 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.

Capturing Commands from the Tcl Script Window

To capture all text written to the console window, use the log console command (see the *Reference Manual*). To capture all commands executed in the console window use the transcript command (see the *Reference Manual*). To clear the text from the console window, use the clear command.

CHAPTER 4

HAPS Deep Trace Debug

The HAPS Deep Trace Debug feature is available to HAPS system users with Synopsys HAPS-60 series prototyping boards and using HAPS SRAM_1x1_HTII daughter boards.

Using external memory for the sample buffer provides a significantly deeper, signal-trace buffer.

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.

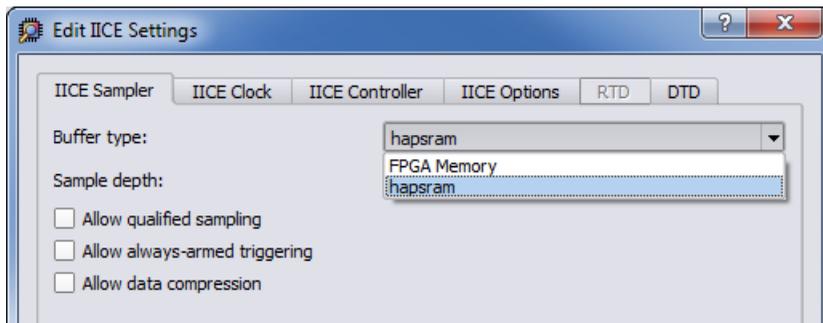
The HAPS deep-trace debug feature is described in the following sections:

- [External Memory Instrumentation and Configuration](#), on page 54
- [DTD Tab](#), on page 55
- [Running Deep Trace Debug with SRAM Memory](#), on page 57

External Memory Instrumentation and Configuration

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

The deep trace debug feature is enabled in the instrumentor from the IICE Sampler tab by setting the Buffer type to hapsram from the drop-down menu.

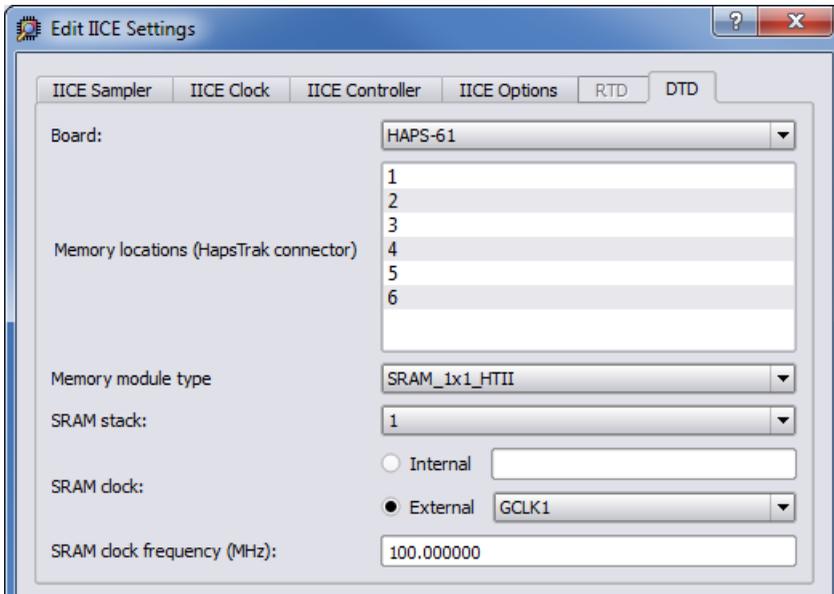


Selecting hapsram enables the DTD tab.

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

DTD Tab

The DTD tab is enabled when the Buffer type parameter on the IICE Sampler tab is set to hapsram and the target HAPS board system is a HAPS-60 (when the technology in the synthesis tool is set to Synopsys HAPS-60).



The individual parameters on the tab are defined in the following table. The parameters can also be set directly from the TCL Script window using the `iice sampler` command (see [Instrumentor iice sampler Options](#), on page 56 in the *Reference Manual* for complete syntax).

Parameter	Description
Board	The Board parameter is determined by the HAPS board configuration being used by the synthesis tool and cannot be changed within the instrumentor.
Memory locations	Specifies the HapsTrak connector location where the daughter board or boards are installed. Connector selection where the daughter boards are physically connected is done by selecting one or more HapsTrak locations 1 through 6 of the daughter boards for the FPGA under debug.
Memory module type	The HapsTrak daughter board type is selected using this drop-down option (the only selection is SRAM_1x1_HTII).
SRAM stack	Sets the stack depth for the external SRAM memory to allow card stacking. The depth of SRAM on each daughter card is 4M locations of 72-bit words for HTII SRAM boards. To increase the external SRAM memory depth beyond 4M x 72, the daughter boards can be stacked. For the HTII type SRAM, 1, 2, or 4 daughter cards can be stacked for the selected SRAM location. The stack number specified applies to all connector locations specified by SRAM locations.
SRAM clock	The clock to the SRAM daughter board can originate from the clock used within the design (Internal) or from an external clock source present on the HAPS board (see SRAM Clocks, on page 58).
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 board stacks using the FPGA internal PLL output as the SRAM clock are listed in SRAM Clocks, on page 58 .

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 Instrumentor from the drop-down menu.
3. In the instrumentor, complete the instrumentation of the design and click the Edit IICE 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 DTD tab and set the following parameters:
 - Memory locations – select the HapsTrak connector location where the daughter card is physically installed (locations 1 through 6).
 - SRAM stack – specify the number of SRAM daughter cards stacked at the connector location.
 - Memory module type – specify the board type from the drop-down menu (the only type available is SRAM_1x1_HTII).
 - SRAM clock – 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 58](#).
 - 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. 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 43](#) 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 from the minimum depth to the maximum configured depth.

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_1x1_HTI1 daughter card stacks using the internal PLL output as the SRAM clock are:

1 SRAM stack	96 to 155 MHz
2 SRAM card stack	92 to 116 MHz
3 SRAM card stack	Not Supported
4 SRAM card stack	80to 110 MHz

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: N_{SRAM}
- Number of 72-bit words per SRAM card: N_{word}
(4194304 for HapsTrak II)
- 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

For example, if $N_{slot} = 1$, $N_{SRAM} = 1$, $N_{word} = 4M$ (4194304) and $N_{signal} = 1900$, the maximum sampling depth for K samples for the SRAM board is 198K.

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.

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

HapsTrak II

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

CHAPTER 5

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 62
- [Verilog Instrumentation Limitations](#), on page 64
- [SystemVerilog Instrumentation Limitations](#), on page 67

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.

Scalar Signal Syntax

The values of scalar signals of type `std_logic` must be enclosed in single quotes in both the GUI and the shell as shown in the following command:

```
watch enable -iice IICE -condition 0 /my_signal {'0'}
```

Entering a scalar signal either without quotes or in double quotes results in an error. Conversely, a vector signal must be entered without quotes as shown in the following command:

```
watch enable -iice IICE -condition 0 /my_bus {1010}
```

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,
    ...

```

The maximum bit width of all elements is 16 which requires a serialized 16-bit vector to define the trigger. For example, to set st1 (2x2x2x1bit):

```
st1[1][1][2]=0
st1[1][1][1]=0
st1[1][2][2]=1
st1[1][2][1]=1
st1[2][1][2]=0
st1[2][1][1]=1
st1[2][2][2]=1
st1[2][2][1]=0
```

Similarly, to set st2:

```
(unsigned int) 200 = (bin) 11001000
```

The trigger expression is defined as:

```
16b'  00110110  11001000
      |   st1   |   st2   |
```

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}}
```

Index

A

always-armed triggering 18

B

black boxes 26

breakpoint icon
color coding 43

breakpoints
in folded hierarchy 42
instance selection 43
listing available 50
listing instrumented 50
selecting 42

buffers
instrumenting restrictions 32

buses
instrumenting partial 32

C

clocks
edge selection 20
sample 19

complex triggering 20

Configure IICE dialog box 10, 16
IICE Controller tab 20, 22
IICE Sampler tab 17

console window operations 52

D

designs
writing instrumented 48

dialog boxes
Configure IICE 10, 16

directories
instrumentation 50

E

encrypting source files 49
essential signal database 39

F

files
encrypting source 49
idc 40
IICE core 50
project 8
folded hierarchy 37

H

hardware
skew-free 14
HDL source
including in project 49
hierarchy
folded 37
hierarchy browser
popup menu 26
hierarchy browser window 25

I

idc file
editing 40
IICE
configuration 9
IICE Controller tab 20, 22
IICE parameters
buffer type 17
common 11
individual 10, 16
JTAG port 13
IICE Sampler tab 17
IICE settings

- sample clock 19
- sample depth 17
- instrumentation
 - partial records 35
- instrumentation directory 50
- instrumenting partial buses 32

J

- JTAG port
 - IICE parameter 13

L

- limitations
 - Verilog instrumentation 64, 67
 - VHDL instrumentation 62

M

- mixed language considerations 61
- multi-IICE
 - tabs 10, 16
- multiplexed groups
 - assigning 36

O

- original source
 - including 49

P

- parameterized modules
 - instrumenting 40
- parameters
 - IICE 9
 - IICE common 11
- partial buses
 - instrumenting 32
- passwords
 - encryption/decryption 49
- project files 8
- projects
 - instrumenting 7

Q

- qualified sampling 17

R

- RAM resources 17
- records
 - partially instrumented 35
- restrictions
 - instrumenting buffers 32

S

- sample clock 19
- sample clock calculation
 - SRAM 59
- sampling
 - in folded hierarchy 37
 - qualified 17
- sampling signals 30, 31, 38, 42, 44, 46, 48, 50
- settings
 - sample clock 19
 - sample depth 17
- signals
 - disabling sampling 32
 - exporting trigger 22
 - instance selection 38
 - listing available 50
 - listing instrumented 50
 - sampling
 - selection 30, 31, 38, 42, 44, 46, 48, 50
- simple triggering 20
- skew-free hardware 14
- source files
 - encrypting 49
- SRAM clocks 58
- state-machine triggering 21
- synthesizing designs 50

T

- trigger signal
 - exporting 22
- triggering

- always-armed [18](#)
- complex [20](#)
- simple [20](#)
- state machine [21](#)

V

- Verdi platform [39](#)
- Verilog
 - instrumentation limitations [64, 67](#)
- VHDL
 - instrumentation limitations [62](#)

W

- watch icon
 - color coding [38](#)
- windows
 - hierarchy browser [25](#)

