Identify®
Quick Start Guide

March 2013

http://solvnet.synopsys.com

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

Destination Control Statement

Disclaimer

Registered Trademarks (®)

Trademarks (™)

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 2013

Contents

Quick Start Guide

Quick Start Guide

Before You Start

Before you can start to use the Identify® instrumentor (and the Identify® debugger), the Identify software must be installed and you must have a license to run the software. The Identify software uses a floating license based of the FLEXnet licensing technology and the Synopsys Common Licensing software. If you don't have a license, you will be prompted to supply one when you attempt to start the Identify instrumentor. Please see the *Installation Guide* on the download page for licensing/configuration information.

Start the Tool

The Identify instrumentor (and the Identify debugger) can be started in the graphical mode, shell mode, or, when run in conjunction with the Synplify, Synplify Pro, or Synplify Premier synthesis tool or the Certify tool, can be launched in the graphical mode directly from the synthesis tool's user interface as described in Synopsys FPGA Synthesis Tool Launch, on page 12.

Graphical Mode

To start the Identify instrumentor or Identify debugger in the graphical mode, do any of the following depending on your platform/operating system:

- In Windows, select Programs->Synopsys->Identify Instrumentor or Programs->Synopsys->Identify Debugger from the start menu or enter the appropriate command at the command prompt:

    installDirectory/bin/identify_instrumentor

    installDirectory/bin/identify_debugger

- In Linux, enter the appropriate command at the command prompt:

    installDirectory/bin/identify_instrumentor

    installDirectory/bin/identify_debugger

The Identify instrumentor or Identify debugger can also be run from a Tcl startup file. To start the tool with a Tcl script, enter the appropriate command:

    installDirectory/bin/identify_instrumentor -f *fileName*.tcl

    installDirectory/bin/identify_debugger -f *fileName*.tcl

Shell Mode

Both the Identify instrumentor and Identify debugger can be started in a shell mode and controlled by Tcl commands. To start either tool in the shell mode, enter the appropriate command at the command prompt:

    installDirectory/bin/identify_instrumentor_shell

    installDirectory/bin/identify_debugger_shell

Vendor Licenses

When you explicitly start the Identify instrumentor or Identify debugger from the operating system, you are prompted to select the license type from the licenses available for your configuration. The following figure shows a Select available license dialog box with a full complement of license types.
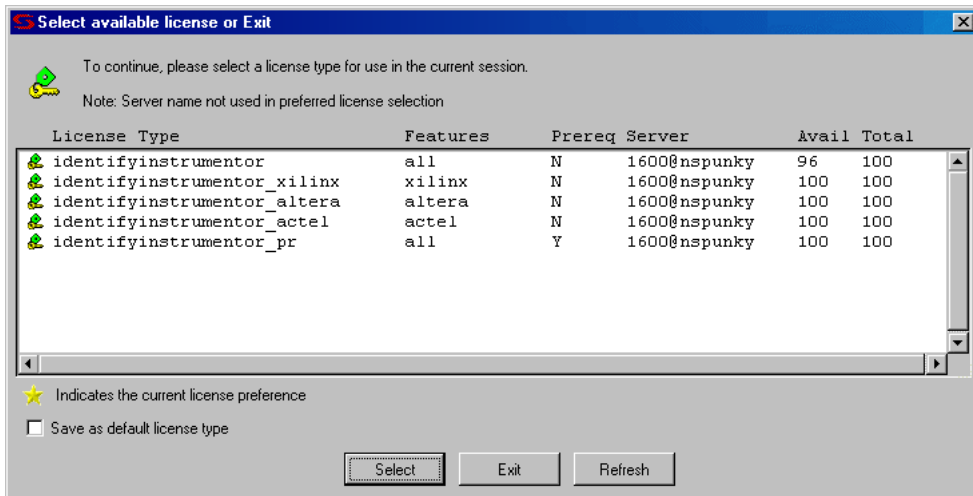
To select a license type, highlight (click on) the entry and then click the Select button. To avoid being prompted for the license type each time you start the Identify instrumentor or Identify debugger, check the Save as default license type box before clicking the Select button.

Note: When you select a vendor-only license, you cannot use another vendor's device in your design.

After opening the Identify instrumentor or Identify debugger, you can change the license type from within the tool by selecting Help->Preferred License Selection from the menu to display the Preferred license selection dialog box and then selecting a new license type as described in the previous paragraph.

Design Flow with the Identify Tool Set

Because the Identify tool set produces an instrumented version of your design, using these tools should be considered a pre-processing step in your current design flow. This Quick Start guide assumes that you currently have a working FPGA design flow for a Xilinx, Altera, or Microsemi device. If not, use a simple design (for example, the tutorial design counter_self) to establish a working flow that effectively moves an HDL design through synthesis, place and route, and programming of the chip. Only after this flow is established should you attempt to debug a design with the Identify tool set.

Before you begin, it is also helpful to note the amount of unused resources available in your original design. The Identify instrumentor provides an estimate of the additional resources necessary for instrumentation in the target device which can help maximize the debugging visibility.

The following figure shows a typical design flow using the Identify tool set.

Working HDL

Create project
in synthesis tool

Launch Inst or
Import Project

Select
Instrumentation

Configure IICE

Create
Instrumented
Design

**Identify
Instrumentor**

Synthesize

Place and Route

Open
Identify Project
in Debugger

Set Trigger
Condition

Run Debug
Hardware

View
Design Data
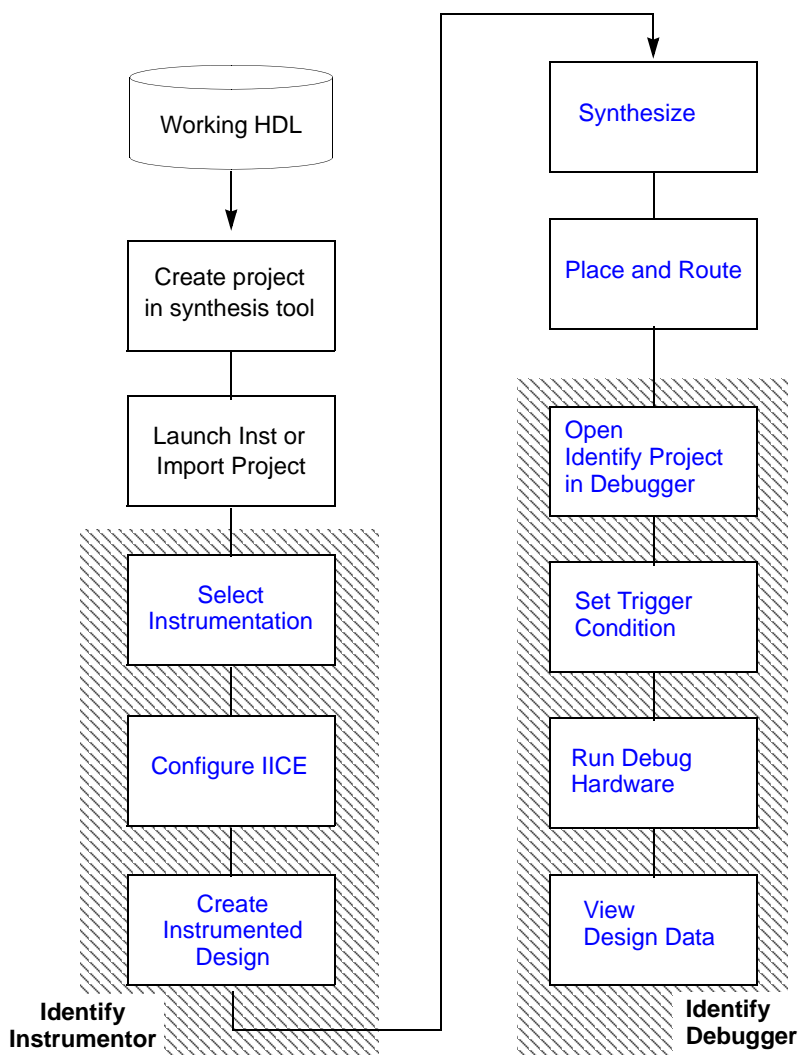
**Identify
Debugger**

Create a New Project

You can create an Identify project by doing any of the following:

- Open a project in the Synplify, Synplify Pro, Synplify Premier, or Certify synthesis tool and then launch the Identify instrumentor directly from the synthesis tool.

- Explicitly open the Identify instrumentor and import an existing Synopsys FPGA synthesis project.

Synopsys FPGA Synthesis Tool Launch

The Identify instrumentor is normally launched directly from the Synplify, Synplify Pro, Synplify Premier, or Certify graphical user interface by selecting Run->Launch Identify Instrumentor.

If your synthesis project currently does not include an Identify implementation, you are prompted to create one. Click OK.
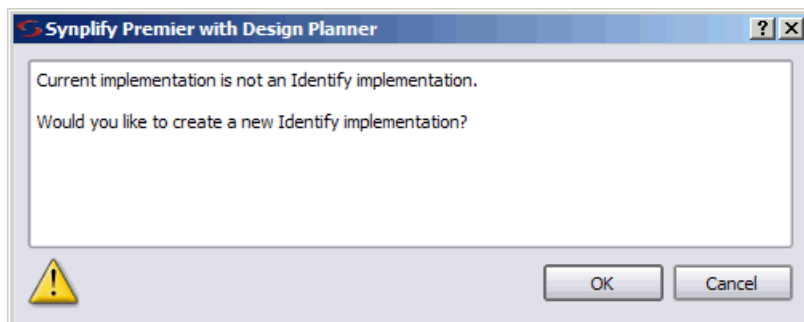
If you have not set up your synthesis tool, you are first prompted for the path to the Identify installation.
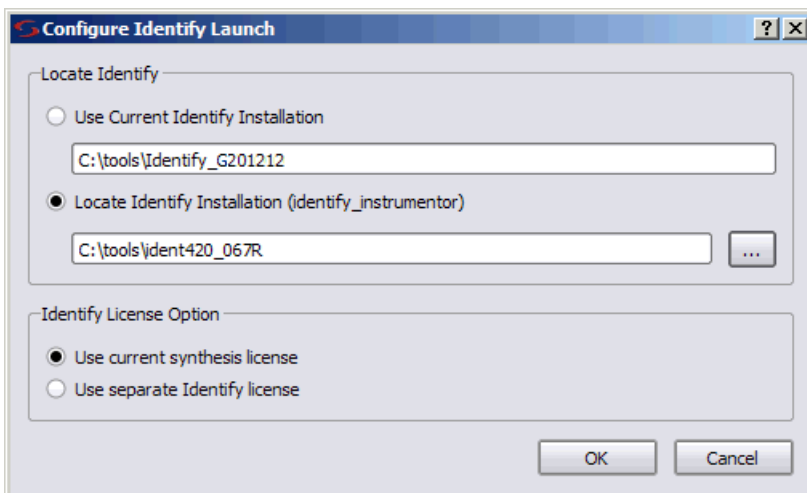
Enter/verify the path to the Identify installation and the license option. When you click OK, you are prompted first to save your project changes. Click OK.
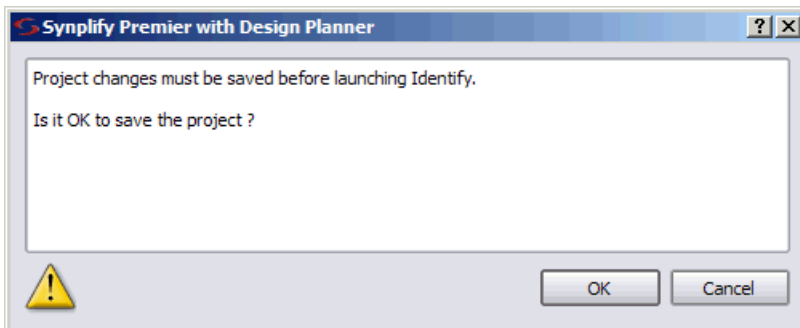
Clicking the OK button:

- Automatically imports the project (prj) file for the open project in the synthesis tool into the Identify instrumentor.

- Automatically compiles the project.

- Opens the project in the Identify instrumentor graphical interface showing all potential watchpoints and breakpoints.

Importing a Synthesis Project

You can also start the Identify instrumentor (or Identify debugger) in the graphical interface and import the synthesis project directly into the open Identify instrumentor or debugger. To explicitly import a synthesis project:

1. Open the Identify instrumentor in graphical mode (see Graphical Mode, on page 8).

2. In the project view, do any of the following:

   – select File->Open project from the menu

   – click on the Open existing project icon in the menu bar

   – click the Open Project button

3. In the Open Project file dialog box, navigate to the synthesis project, click on the project (prj) file, and click Open.

Opening the project:

- Displays the instrumentation window with the hierarchy browser on the left and the HDL source code on the right.

- Automatically compiles the project which allows the Identify instrumentor to determine all of the potential locations for instrumenting breakpoints and watchpoints.

Select Desired Instrumentation

When your design compiles successfully, your design with its possible instrumentation is displayed in the instrumentation window. The hierarchy browser on the left shows the design hierarchy and is used to browse your design. Double clicking any hierarchy symbol takes you to its corresponding code location in the HDL source code display on the right.

In the source code display, each potential *watchpoint* is indicated by a glasses icon prefixing the signal name, and each potential *breakpoint* is indicated by a circle in the left margin of the source code.

14

```
                   56      begin
Watchpopint        57        if 👓clr = '0' then
                   58          👓current_state <= s_RESET;
                   59        elsif 👓clk'event and 👓clk = '1' then
                   60          case 👓current_state is
Breakpoint      ●  61            when s_RESET    => 👓current_state <= s_ONE;
                ●  62            when s_ONE      => 👓current_state <= s_TWO;
```
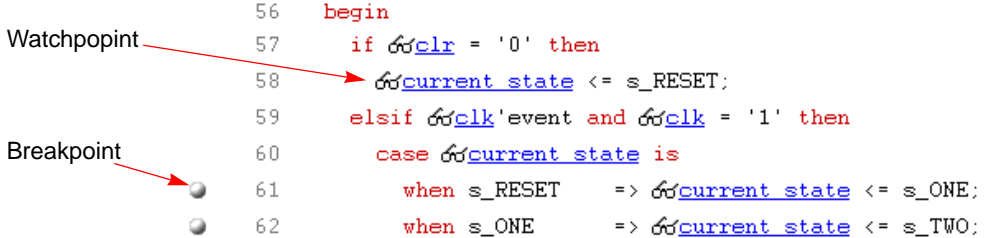
Watchpoints

A watchpoint is instrumented by clicking the watchpoint icon adjacent to the signal name and selecting Sample Only, Trigger Only, or Sample and Trigger from the pop-up menu.

- Sample-only signals are sampled by the debug logic. You can view the data from a sample-only signal, but you cannot use the signal to trigger the hardware during runtime. The lenses of the watchpoint icon are blue for sample-only signals.

- Trigger-only signals are not sampled by the debug logic and have no visibility. These signals are used to create trigger conditions to trigger the debug logic (IICE) during runtime. The lenses of the watchpoint icon are red for trigger-only signals.

- Sample and trigger signals connect to both the sample logic and trigger logic of the IICE. These signals are visible and can be used to trigger the debug logic. The lenses of the watchpoint icon are green for sample and trigger signals.

Note: Other selections are available for defining watchpoint types on partial buses and on individual fields within a record or a structure; see the *User Guide* for more information.

Breakpoints

Breakpoints are instrumented by clicking on the circle icon to the left of the breakpoint line. A breakpoint is essentially a control-flow trigger. It is used at runtime to trigger the debug logic based on the flow through case and if/then/else statements. During debug, a breakpoint triggers the IICE whenever the corresponding branch in the code becomes active.

Resource Estimates

The Identify instrumentor uses the synthesis tool to provide an estimate of the resources on the target device required to incorporate the additional instrumentation logic. The estimate depends on the number of signals and breakpoints selected for the instrumentation, as well as other IICE settings such as device family and sample depth. Please see the next section, *Configure the IICE,* for information about these settings.

Note: The Identify instrumentor is not aware of the size of the target device and it is possible to add more instrumentation to the design than will fit on the device. Exceeding the device capacity causes errors during synthesis or place and route.

Configure the IICE

Configuring an IICE to match your debugging requirements involves the setting of a series of IICE parameters. The common IICE parameters are set in the project window and apply to all IICE units defined for the currently-active implementation; 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.

Common IICE Parameters

The common IICE parameters for the currently-active instrumentation are set or reported in the project window after a design is successfully compiled. All IICE units in a multi-IICE configuration share these same parameter values. To redisplay the project window, click the project window tab at the bottom of the window.

- Device Family – the device family is specified in the synthesis tool and 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 Identify instrumentor.

- JTAG Port – the *JTAG port* selection box is normally set to builtin. This setting configures the IICE to use the built-in JTAG resources of the device. If access to the built-in JTAG port is not available, you must use the soft JTAG scheme which inserts a JTAG controller into the design and connects it to four user-defined pins (see the *User Guide* for more information about the soft JTAG).

- Use skew resistant hardware – the Use skew resistant hardware option is normally not checked; this option is used with designs that have no global clock buffer available for the JTAG clock. For more information, see the *User Guide*.
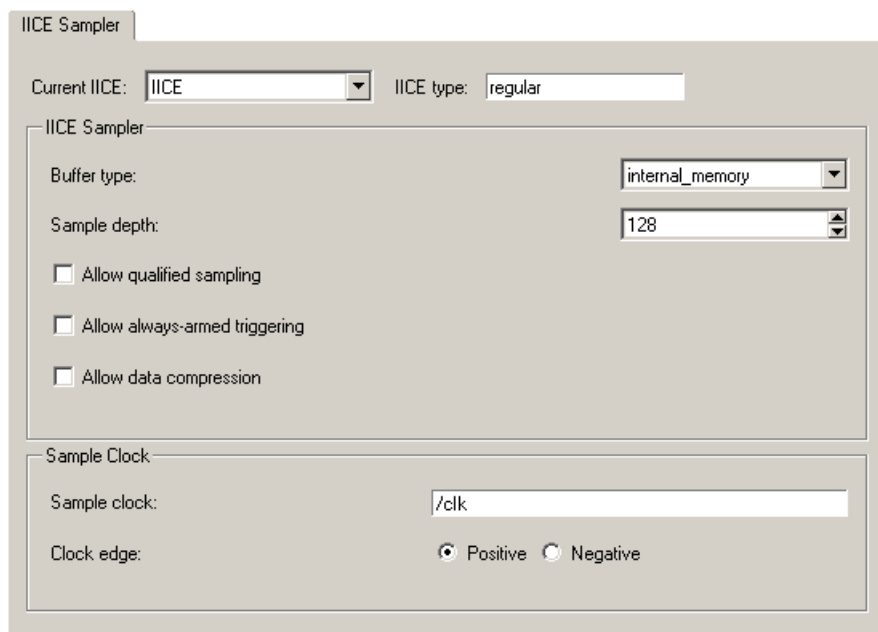
Individual IICE Parameters

The individual parameters for each IICE are defined on the two tabs of the Configure IICE dialog box. To display this dialog box, select Actions->Configure IICE from the menu or click on the Edit IICE settings icon. When setting parameters for an individual IICE in a multi-IICE configuration, use the Current IICE field to specify the target IICE.

IICE Sampler Tab

The IICE Sampler tab controls the size of the IICE sample buffer and defines the sample clock.

When setting these parameters:

- Current IICE – 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 – 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.

- Buffer type – 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 additionally use external HAPSRAM for the HAPS Deep Trace Debug feature.

- Sample depth – 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. For information on using the advanced sampling modes, see the *User Guide.*

- Allow qualified sampling – when checked, causes the Identify instrumentor to build an IICE block that is capable or 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 slight area and clock-speed penalty.

- Allow always-armed triggering – 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 slight area and clock-speed penalty.

- Allow data compression – when checked, adds compression logic to the IICE to support sample data compression in the Identify debugger (see the *User Guide* for more information). When unchecked (the default), compression logic is excluded from the IICE, and data compression in the Identify 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 – 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.

Note: You can also specify the sample clock signal by right-clicking on the watchpoint icon (or signal name) and selecting Sample Clock from the popup menu.

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 global clock resource of the chip.
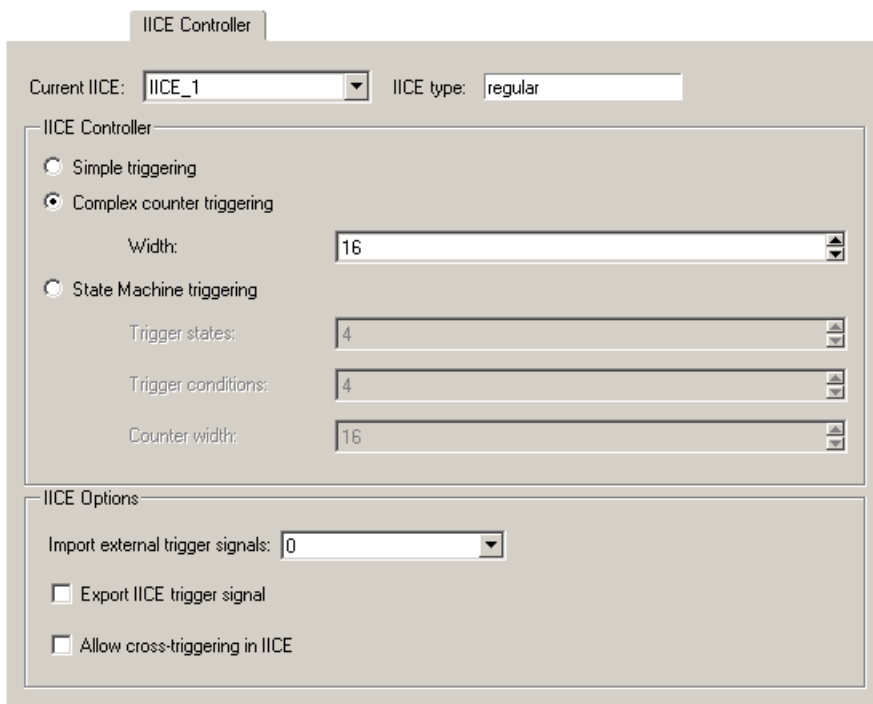
Note: If you need help determining the hierarchical path of your clock, try finding it in the HDL source viewer. You may then add and remove it for instrumentation. The full hierarchical path of the signal will be echoed to the TCL command line. Remember that a signal cannot be used as the sample clock if it is instrumented.

- Clock edge – determines 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 customizes the trigger logic available for triggering the sample buffer.

- **Current IICE** – 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** – 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.

- **Simple triggering** – uses a single trigger event to trigger the sample buffer.

- **Complex counter triggering** – uses advanced trigger operations such as counting the number of trigger events or delaying a trigger event by a set number of clock cycles. See the *User Guide* for more information on complex-counter triggering

- **State Machine triggering** – creates fully flexible trigger conditions including capturing samples based on sequences of trigger events and the occurrence of external triggers. See the *User Guide* for more information on state machine triggering.

- Import external trigger signals – allows triggers from one or more external sources 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 Identify instrumentation. Selecting this option automatically selects state-machine triggering.

- Export IICE trigger signal – brings out the IICE's global trigger signal to the top level of your design. This signal can then be used to trigger a logic analyzer or to trigger another IICE.

- Allow cross triggering in IICE – when enabled, allows the current IICE unit to accept a cross-trigger from another local IICE unit.

Create the Instrumented Design

When you are satisfied with the instrumentation, save your design by either selecting File->Save Project from the Identify instrumentor menu bar or by clicking on the Save project's activated implementation icon on the toolbar. Saving the project automatically adds a set of files to the Identify implementation directory which are then used by the synthesis tool to incorporate the instrumented logic into the design netlist.

Once you have instrumented your design, you will begin the process of implementing the design and then debugging it. Note that any changes to the Identify project or to the design files can cause the current project to become invalid. Take care not to change or otherwise overwrite the current project. The Identify debugger takes many precautions to ensure correct sample data, and does not allow debugging of a design that has been changed or the viewing of files that have been modified. These types of changes may require you to re-run both synthesis and place and route.

Signals can be instrumented directly in the synthesis tool outside of the Identify instrumentor to allow signals within a parameterized module, which were previously unavailable for instrumentation, to be successfully instrumented. For more information on this type of instrumentation, see "Instrumenting Signals Directly in the idc File" in the *User Guide*.

The Identify software allows you to create multiple instrumentations for a single design using different IICE configuration parameters. For information on multiple instrumentations, see the *User Guide*.

Synthesize and Place and Route the Design

Repeat Original Design Flow with Instrumented Design

After saving your instrumentation, run the design through the synthesis tool design flow.

Note: Always synthesize and place and route your instrumented design using all of the original constraints and settings.

Read the Project into the FPGA Synthesis Tool

When you launch the Identify instrumentor from the synthesis tool, the project file is automatically updated to describe the resulting Identify directory/file structure for the instrumented design. Synthesizing the design in the synthesis tool adds the IICE sampling logic to the design netlist.

Add JTAG Clock Constraints

During synthesis, it is important that the JTAG clocks added by Identify are properly constrained (the JTAG clock runs at a very slow speed and must not be optimized to the speed of the design clocks). These clock constraints are handled automatically by the Identify instrumentor by the syn_dics.sdc constraint file from the *implementation*/instr_sources directory.

Any signal with a name that includes identify_clk must be constrained to run at 25MHz (40ns) or less. For the place and route tools, the same precautions must be taken to constrain the JTAG clocks to run at 25MHz (40ns) or less.

Altera Place and Route

For most place and route tools, the flow should be unchanged from the original design flow. In the case of Altera Quartus, it may be necessary to add the following two files to the Quartus project directory before you can create the bit file.

- *projectName*_**instr/instr_sources/{syn_sld_node.v|syn_sld_node.vhd}**
- *projectName*_**instr/instr_sources/syn_sld_node.esf**

Note: These files are only required when a design uses the Altera built-in JTAG controller. The files are not added to the Quartus project, but are simply copied to the Quartus project directory where they are accessed. Quartus reports an error whenever these files are required and they are not included in the project directory.

Program the Instrumented Bit File to the Target Device

When all of the required project files are in place, program the instrumented bit file into the targeted device to load the design logic and the required instrumented logic.

Open the Identify Project in the Debugger

Any project that has been instrumented by the Identify instrumentor and then synthesized can be opened in the Identify debugger. The Identify debugger appears similar to the Identify instrumentor except that only instrumented signals and breakpoints are displayed in the Identify debugger. If the Identify debugger is being run on a machine that is different from the host where the design was instrumented, see *Debugging on a Different Machine* in Chapter 8 of the *User Guide*.

Set Trigger Condition

Setting the trigger condition involves setting breakpoints and/or watchpoints in the source code window to trigger the IICE when the associated condition occurs.

Setting Breakpoints

Potential breakpoints are indicated by a green circle in the margin to the left of the source code. Clicking on a breakpoint activates the breakpoint and changes the color of the circle from green to red.

Setting Watchpoints

Watchpoint triggers can be specified on any sampled signal. The watchpoint condition, which is any legal VHDL or Verilog expression that evaluates to a constant, is set through the user interface.

To set a simple watchpoint:

1. Click on the signal

2. Select Set trigger expressions from the popup menu to display the Watchpoint Setup dialog box

3. In the First value field, enter a value for the watch expression.

4. Click OK

The setting of the watchpoint trigger is noted by the breakpoint icon next to the watched signal changing from green to red.

Multiple Breakpoints and Watchpoints

When an instrumented design has more than one activated breakpoint, the breakpoint events are ORed together which effectively allows the breakpoints to operate independently – only one activated breakpoint must trigger to cause the sampling buffer to acquire its sample.

When an instrumented design has more than one activated watchpoint, the watchpoint events are ANDed together which effectively causes the watchpoints to be dependent on each other – all activated watchpoint events must occur coincidently to cause the sampling buffer to acquire its sample.

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

Run Debug Hardware

This guide assumes that the Identify debugger communicates with the instrumentation through the same cable that was originally used to program the device. If another communications methodology is being used, see the *Connecting to the Target System* chapter in the *User Guide*. Before running the debug hardware, test your communications setup with the com check command.

The com check command:

- checks the cable connection
- auto-detects the devices on the JTAG chain
- auto-detects the device with instrumentation that matches the current project

If errors are reported, see Communication Errors, on page 28 for possible explanations.

After all of the desired breakpoints and/or watchpoints have been activated, the IICE trigger circuits on the FPGA device are then armed and wait for the watchpoint condition to occur. To arm the IICE trigger circuits on the active IICE, click the Arm current IICE for triggering icon. To arm more than one IICE in a multi-IICE configuration, open the project window in the Identify debugger, check the individual IICE units to be armed, and then click the Run button. Either of these actions downloads the trigger information to the IICE. The IICE now waits for the trigger condition (watchpoint or breakpoint) to occur.

When a watchpoint or breakpoint trigger occurs, sampling is stopped (the hardware continues to run), and the sampled data is transferred back to the debugger where it is displayed in yellow adjacent to the sampled signals in the source code. A small arrow is displayed to the left of the breakpoint or watchpoint icon to indicate the condition responsible for the trigger (identifying the trigger condition is important when multiple breakpoints or watchpoints are active).

View Design Data

The sample buffer display can be varied by time. The Cycle display in the middle of the menu bar shows the value zero. This is the point in the sample data buffer where the trigger occurred. By clicking on the up-down arrows on the right, you can increase or decrease the cycle count to show sample buffer values before or after the trigger point.

You can change where the trigger point is in the buffer by selecting one of the Early, Middle, or Late buttons and again clicking on the Run button. The trigger location changes the next time that the IICE triggers.

Early   Middle   Late

Waveform Display

In addition to displaying the sampled data for the selected signals, the Identify debugger can export the sample buffer contents for display in a variety of waveform viewers including Synopsys DVE and GTKWave.

Select GTKWave Preference

Select Options->Debugger preferences from the menu bar. Verify that GTKWave is the selected choice in the Waveform Viewer Preferences dialog box.

Note: GTKWave is the freeware waveform viewer that is distributed with the Identify tool set.

Click "Waveform" Button

Select Window->Waveform or click the icon labeled Open Waveform Display in the Identify debugger toolbar. A GTKWave waveform display is shown which displays all of the sampled data for each of the sampled signals. The Identify debugger adds two signals to this waveform:

- identify_cycle – an integer that shows the position in the sample buffer. A value of 0 indicates the cycle in which the trigger event occurred.

- identify_sampleclock – a single-bit signal that shows the edges of the reference clock.

Communication Errors

The following are common errors that you may encounter when setting up communications with the Identify debugger.

" ERROR: Communication is stuck at one/zero. Please check the cable connection.

The Identify debugger is unable to communicate with the instrumented device. This error is usually attributed to a cable connection problem. Make sure that the cable is correctly connected between the parallel/USB port and the JTAG port of the board and verify that the cable type is set correctly in the project editor (select File->Edit Project or use the com cabletype TCL command).

Note: IMPORTANT – This error is often caused by an incorrect parallel port setting. Please try all choices for the communications port setting using either the command line or the Configure Port Settings dialog box available by clicking the Port settings button in the Identify debugger project window.

" ERROR: Cannot find valid instrumented design.

This error indicates that the design on the programmable chip is NOT the instrumented version of the design. Verify that the bit file you are programming is actually created from the instrumented sources and that the debug logic (IICE) has not been removed during implementation. Verification can often be done by searching the intermediate place and route files (for example, a Xilinx NCD file) for the word "identify."

" ERROR: Instrumented design on FPGA differs from design loaded into Identify Debugger.

This error indicates that the Identify debugger cannot find a device in the JTAG chain that has been instrumented by the Identify instrumentor. In this case, the ID of the instrumentation does not match the ID of the currently loaded project. Please verify that the correct project is loaded for the corresponding bit file. The error occurs when the project is re-instrumented without regenerating a bit file. If you have changed the design or its instrumentation, you must create a new bit file before debugging the design.

" ERROR: No hardware devices were found. Please check the cable connection.

This error indicates that no devices are visible in the JTAG identification register chain. The error is usually caused by a bad cable connection or an incorrect cable type setting.

" ERROR: The hardware has not seen an active clock edge of the sample clock.

This error usually indicates that the Identify hardware is functioning correctly and that the Identify debugger is able to communicate with the device, but that the sample clock is not being toggled. This error can be caused if the wrong clock is chosen in the Identify instrumentor. Please verify that the correct sample clock is selected using the iice clock command. Also check that the clock signal is using the global clock resources of the chip.

> **Note:** This error often occurs when the clock signal is not assigned correctly to the clock pin on the board. Verify that, during placement, the clock signal (sample clock) is correctly assigned to the chip pin that is connected to the clock oscillator on the board.

" ERROR: Hardware driver failure.

This error usually indicates that the correct port driver is not installed on the debug system. Please see the release notes for help installing the port driver.

Clock Skew

When the data returned from the Identify debugger appears incorrect or does not properly relate to the given trigger condition, there may be an issue with clock skew on the JTAG clock. Make sure that the identify_clk signal is using the global clock resources on the chip. If there is no clock buffer available, you may have the Identify instrumentor build skew-resistant hardware. Please see the *User Guide* for more information on skew-free hardware.

Debug Mode

If you are experiencing a communication error that is not listed above or if you have not been able to resolve the error, go to the SolvNet website at www.solvnet.com. If may be helpful to run the Identify debugger in "debug" mode as outlined below:

1. If open, close the Identify debugger.

2. Right-click the Identify debugger shortcut on the desktop and select Properties from the popup menu to display the Properties dialog box.

3. Append the -debug flag to the path to the executable in the target field.

4. Restart the Identify debugger and reopen the project.

5. Enter the following two commands at the command prompt in the Console window:

    chain clear
    com check

6. When using the SolvNet website, include a copy of the log file and any other important details about your particular setup/flow as well as the Identify project (prj) file.