Identify® Actel Edition
Tutorial

September 2010

http://solvnet.synopsys.com

Service Marks (^{SM})

Restricted Rights Legend

Printed in the U.S.A
September 2010

Contents

Chapter 1: Getting Started

The Debugging System 8
    Identify Instrumentor 9
    Identify Debugger 9

Design Flow 10

Tutorial Requirements 11
    Hardware/Software Environments 11

Chapter 2: The Tutorial Design

Design Schematic 14

Verilog Tutorial Design 15

VHDL Tutorial Design 18

Chapter 3: Instrumenting Your Design

Launching the Identify Instrumentor 22

Setting up the IICE 26
    Setting the Common IICE Parameters 26
    Setting the Individual IICE Parameters 27

Selecting the Instrumentation 30
    VHDL Design Instrumentation 30
    Verilog Design Instrumentation 33

Writing the Instrumented Design 36

Chapter 4: Implementing the Design

Synthesis 40

Place and Route 41

Program the Device 41

Chapter 5: Debugging Your Design

Starting the Identify Debugger 43

Specifying the JTAG Cable 45

Setting the JTAG Chain 46

Setting Up Triggers and Capturing Data 48
    Triggering on a Breakpoint 48
    Deactivating a Breakpoint 51
    Triggering on a Watchpoint 51
    Using the Complex Counter 54

Generating Waveforms 55

CHAPTER 1

Getting Started

The Identify® Actel Edition tool set is an innovative set of programmable hardware tools that lets you debug your HDL design:

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

The Identify Actel Edition tool set enables the debugging of FPGA designs, FPGA-based prototypes, and system-on-a-chip designs. For the first time, you can debug live hardware using intuitive, HDL-based debugging techniques that provide visibility into the internal operation of your system.

The Identify Actel Edition tool set easily integrates into your existing design flow so that minimal effort is required to begin the debugging of your HDL designs. To better understand how the tool set works and how it works with your HDL design flow, this guide provides comprehensive information about navigating through the Identify tools and integrating with your other design flow tools.

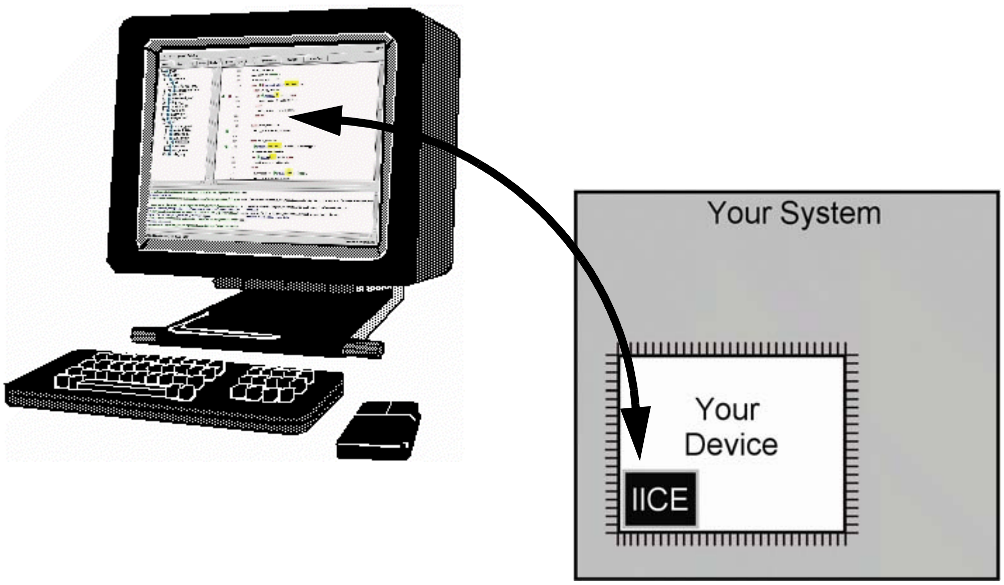
This remainder of this chapter describes:

- The Debugging System
- Design Flow
- Tutorial Requirements

The Debugging System

The Identify Actel Edition tool set is based on the principle of in-system debugging. Using thes Identify tool set allows you to debug your device in the target system, at target speed while still debugging at the HDL level.

The Identify instrumentor captures your device's internal states by inserting probe hardware (called an IICE™ – Intelligent In-Circuit-Emulator) into your design. The IICE captures internal design states based on user-specified trigger conditions. Data captured at the target device is transferred back to the host computer where it is transformed and displayed by the Identify debugger.



The Identify tool set is a dual-component software system consisting of:

- The Identify instrumentor which inserts and configures the IICE
- The Identify debugger which controls the IICE and displays data from the IICE.

The following sections briefly describe these two software components.

Identify Instrumentor

The Identify instrumentor reads and analyzes the HDL description of a design and provides you with detailed information about the signals that can be sampled and the locations in the source code where breakpoints can be set. This information is collectively referred to as the *instrumentation*.

The Identify instrumentor uses the HDL design files and your selected instrumentation information to create a custom IICE block. The Identify instrumentor then connects one or more IICE blocks to the appropriate locations in the design.

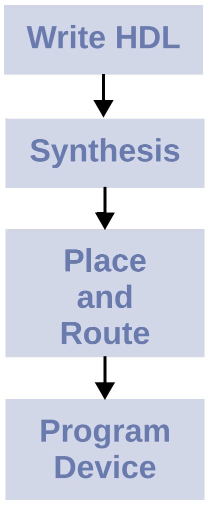
Finally, the Identify instrumentor re-writes your HDL design with the modifications necessary to implement and connect the IICE blocks. The modified HDL design is written to a different location so that it does not overwrite your original design.

Identify Debugger

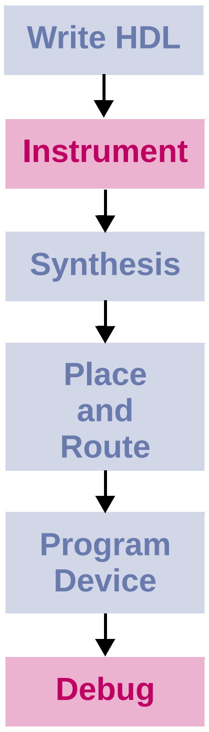
The Identify debugger lets you interact with your real hardware at the HDL level. In the Identify debugger, you set trigger conditions to determine when to capture data, and then view the captured data as either annotated source code or as waveforms.

Design Flow

Design flows for HDL design and debugging vary according to the type of hardware and device you use. Displayed below is the typical HDL design flow without the Identify tool set on the left and a typical HDL design flow with the Identify tool set on the right.



Design Flow without Identify Tool Set



Design Flow with Identify Tool Set

In the design flow without the Identify tool set, the first step is to create the HDL source files for the design. Next, the design is synthesized to the target device. Once synthesized, the design is placed and routed before it is finally implemented in the target device.

The design flow with the Identify tool set adds two steps to the standard flow – one step at the beginning and one step at the end. After the HDL source is created, the Identify instrumentor is used to create a debuggable design. This design is then run through the rest of the standard design flow. After the instrumented design has been implemented, the Identify debugger is then used to debug the design in the target system.

Tutorial Requirements

This tutorial guides you through the process of debugging a small HDL design in a real hardware environment. The tutorial teaches you how to generate an instrumented design for debugging and then how to debug that instrumented design.

The tutorial explains how the Identify tools are used in concert with your synthesis and place-and-route flow. However, the tutorial does not provide details and procedures for synthesizing and using place-and-route tools with your instrumented hardware design. For the purposes of this tutorial, it is assumed that you have a working implementation flow. To find out more about these processes, consult your synthesis and place-and-route tool vendor documentation for more information.

Hardware/Software Environments

This tutorial is intended to be performed with Actel hardware. The tutorial was developed using the following software:

- FPGA synthesis software: Synplify Pro E-2010.09A-1 or newer
- Actel Libero 9.0 or newer
- Identify software: Identify E-2010.09A or newer

CHAPTER 2

The Tutorial Design

The Identify Actel Edition tool set can debug a variety of HDL designs. To better understand the debugging process, this guide provides a small HDL design example. The example design is a simple 4-bit counter. It only requires a clock and a reset signal connection to run in the hardware. Two versions of the counter are provided: one in VHDL and one in Verilog.

- For the Verilog tutorial, see Verilog Tutorial Design
- For the VHDL tutorial, see VHDL Tutorial Design

Design Schematic

The following figure is a schematic representation of the simple state machine that is used for the tutorial design. The state machine is configured as a 4-bit counter; the state machine representation is shown to the left of the schematic.

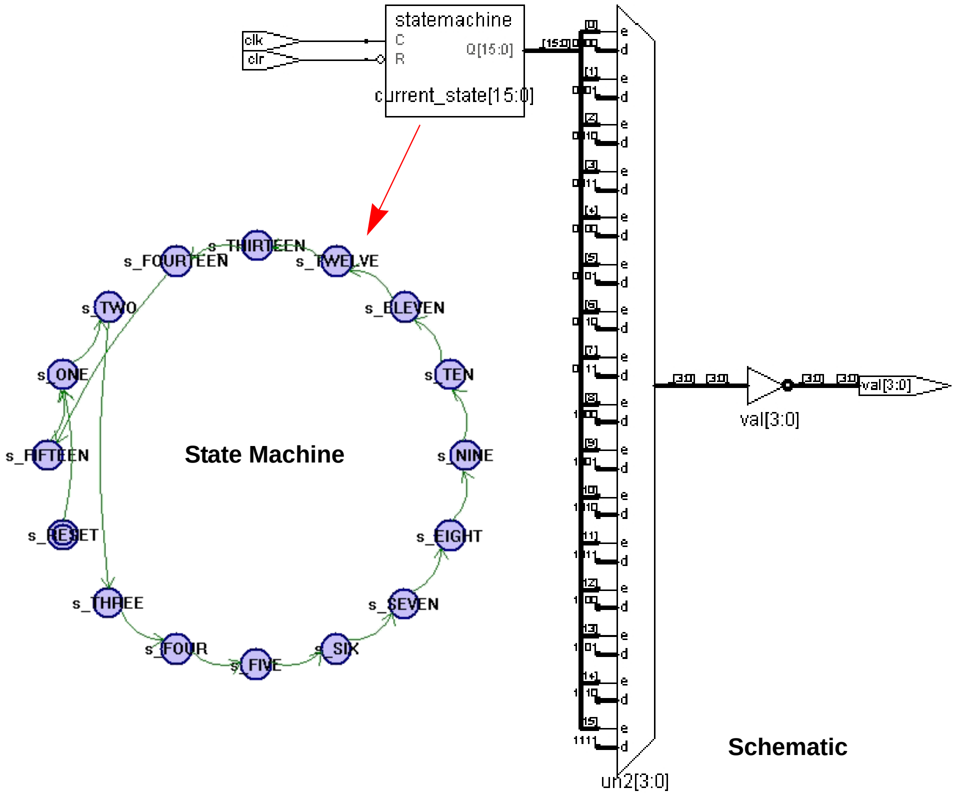


Figure 2-1: Tutorial design schematic

Verilog Tutorial Design

The tutorial design is implemented in Verilog as a single module with two always block statements. The first always block implements a state machine that controls the state of the system; the second always block computes the output values based on the current state. Parameter definitions make the module easier to read.

The tutorial design has two external inputs and one external output. The inputs are:

- `CLK` – the system clock for the design
- `CLR` – resets the state machine to a known state

The single output is `val`.

The board-mounted programmable device interconnects with the Identify software through a communications cable. For this tutorial:

- physically connect the `CLK` input to a clock generator on your target Actel FPGA system.
- connect the `CLR` input to a switch on your board that connects the signal to a zero value when pressed.

Direct the Actel place-and-route tool to make these connections.

When `CLR` is zero, the state machine enters state `s_RESET`. When `CLR` is no longer zero, the state machine transitions to state `s_ONE`, then `s_TWO`, and so on, on each clock cycle.

Based on the current state value, the val output is set to the values listed in the second always block. These output values represent a binary encoding of the output state. However, this binary encoding is inverted so that it can be used to drive active-low LED displays on your board (if they exist). For example, if the state machine is in state `s_ONE`, the val output is set to `1110`, and when the state machine is state `s_ELEVEN`, the output val is set to `0100`.

The Verilog source code file for the tutorial design is displayed below:

```
module counter_self(clr, val, clk);
   output [3:0] val;
   input        clk;
   input        clr;
   reg [3:0] val;
   reg [3:0] current_state;
   parameter [3:0]
      s_RESET    = 0,
      s_ONE      = 1,
      s_TWO      = 2,
      s_THREE    = 3,
      s_FOUR     = 4,
      s_FIVE     = 5,
      s_SIX      = 6,
      s_SEVEN    = 7,
      s_EIGHT    = 8,
      s_NINE     = 9,
      s_TEN      = 10,
      s_ELEVEN   = 11,
      s_TWELVE   = 12,
      s_THIRTEEN = 13,
      s_FOURTEEN = 14,
      s_FIFTEEN  = 15;

   always @(posedge clk or negedge clr)
      begin
      if (clr == 1'b0)
         current_state = s_RESET; /*  4'b0000 */
      else begin
         case (current_state)
            s_RESET:    current_state = s_ONE;
            s_ONE:      current_state = s_TWO;
            s_TWO:      current_state = s_THREE;
            s_THREE:    current_state = s_FOUR;
            s_FOUR:     current_state = s_FIVE;
            s_FIVE:     current_state = s_SIX;
            s_SIX:      current_state = s_SEVEN;
            s_SEVEN:    current_state = s_EIGHT;
            s_EIGHT:    current_state = s_NINE;
            s_NINE:     current_state = s_TEN;
            s_TEN:      current_state = s_ELEVEN;
            s_ELEVEN:   current_state = s_TWELVE;
            s_TWELVE:   current_state = s_THIRTEEN;
            s_THIRTEEN: current_state = s_FOURTEEN;
            s_FOURTEEN: current_state = s_FIFTEEN;
            s_FIFTEEN:  current_state = s_ONE;
            default:    current_state = s_RESET;

         endcase /*  case(current_state) */
      end /*  else: !if(clr == 1'b0) */
   end /*  always @ (posedge clk or negedge clr) */
   always @(current_state)
      begin
         case (current_state)
            s_RESET:    val = 4'b1111;
            s_ONE:      val = 4'b1110;
            s_TWO:      val = 4'b1101;
            s_THREE:    val = 4'b1100;
            s_FOUR:     val = 4'b1011;
            s_FIVE:     val = 4'b1010;
            s_SIX:      val = 4'b1001;
            s_SEVEN:    val = 4'b1000;
            s_EIGHT:    val = 4'b0111;
            s_NINE:     val = 4'b0110;
            s_TEN:      val = 4'b0101;
            s_ELEVEN:   val = 4'b0100;
            s_TWELVE:   val = 4'b0011;
            s_THIRTEEN: val = 4'b0010;
            s_FOURTEEN: val = 4'b0001;
            s_FIFTEEN:  val = 4'b0000;
            default:    val = 4'b0000;

         endcase /*  case(current_state) */
      end /*  always @ (current_state) */
endmodule /*  counter_self */
/* EOF */
```

VHDL Tutorial Design

The tutorial design is implemented in VHDL as a single entity with two processes. The first process implements a state machine that controls the state of the system, and the second process computes the output values based on the state. This design has a user-defined type named state, which is used to make the design more readable.

The design has two inputs and one output. The inputs are:

- CLK – the system clock for the design
- CLR – resets the state machine to a known state

The single output is val.

The board-mounted programmable device interconnects with the Identify software through a communications cable. For this tutorial:

- physically connect the CLK input to a clock generator on your target Actel FPGA system.
- connect the CLR input to a switch on your board that connects the signal to a zero value when pressed.

Direct the Actel place-and-route tool to make these connections.

When CLR is zero, the state machine enters state s_RESET. When CLR is no longer zero, the state machine transitions to state s_ONE, then s_TWO, and so on, on each clock cycle.

Based on the current state value, the val output is set to the values listed in the second process. These output values represent a binary encoding of the output state. However, this binary encoding is inverted so that it can be used to drive active-low LED displays on your board (if they exist). For example, when the state machine is in state s_ONE, the val output is set to 1110, and when the state machine is state s_ELEVEN, the output val is set to 0100.

The VHDL source code file for tutorial design is displayed below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity counter_self is
   port(
      val : out unsigned ( 3 downto 0 );
      clr : in std_logic;
      clk : in std_logic
      );
end counter_self;
architecture rtl of counter_self is
   type state is (
      s_RESET,
      s_ONE,
      s_TWO,
      s_THREE,
      s_FOUR,
      s_FIVE,
      s_SIX,
      s_SEVEN,
      s_EIGHT,
      s_NINE,
      s_TEN,
      s_ELEVEN,
      s_TWELVE,
      s_THIRTEEN,
      s_FOURTEEN,
      s_FIFTEEN );
   signal current_state: state;
begin
   process(clk, clr)
   begin
      if clr = '0' then
         current_state <= s_RESET;
      elsif clk'event and clk = '1' then
         case current_state is
            when s_RESET    => current_state <= s_ONE;
            when s_ONE      => current_state <= s_TWO;
            when s_TWO      => current_state <= s_THREE;
            when s_THREE    => current_state <= s_FOUR;
            when s_FOUR     => current_state <= s_FIVE;
            when s_FIVE     => current_state <= s_SIX;
            when s_SIX      => current_state <= s_SEVEN;
            when s_SEVEN    => current_state <= s_EIGHT;
            when s_EIGHT    => current_state <= s_NINE;
            when s_NINE     => current_state <= s_TEN;
            when s_TEN      => current_state <= s_ELEVEN;
            when s_ELEVEN   => current_state <= s_TWELVE;
            when s_TWELVE   => current_state <= s_THIRTEEN;
            when s_THIRTEEN => current_state <= s_FOURTEEN;
            when s_FOURTEEN => current_state <= s_FIFTEEN;
            when s_FIFTEEN  => current_state <= s_ONE;
         end case;
      end if;
   end process;

   process( current_state )
   begin
      case current_state is
         when s_RESET    => val <= "1111";
         when s_ONE      => val <= "1110";
         when s_TWO      => val <= "1101";
         when s_THREE    => val <= "1100";
         when s_FOUR     => val <= "1011";
         when s_FIVE     => val <= "1010";
         when s_SIX      => val <= "1001";
         when s_SEVEN    => val <= "1000";
         when s_EIGHT    => val <= "0111";
         when s_NINE     => val <= "0110";
         when s_TEN      => val <= "0101";
         when s_ELEVEN   => val <= "0100";
         when s_TWELVE   => val <= "0011";
         when s_THIRTEEN => val <= "0010";
         when s_FOURTEEN => val <= "0001";
         when s_FIFTEEN  => val <= "0000";
      end case;
   end process;
end rtl;
-- EOF
```

CHAPTER 3

Instrumenting Your Design

The Identify instrumentor selects the design visibility (breakpoints and watchpoints) and special hardware configurations including complex event counters, and sampling and triggering modes for your design. The goal of the instrumentation process is to define an IICE and insert it into your HDL design. The instrumentation flow is:

- Launching the Identify Instrumentor
- Setting up the IICE
- Selecting the Instrumentation
- Writing the Instrumented Design

The HDL design and project files are included in a "tutorial" subdirectory under the Identify installation directory. This subdirectory includes the following files:

- `counter_self.v` (Verilog design file)
- `counter_verilog_actel.prj` (Verilog project file)
- `counter_self.vhd` (VHDL design file)
- `counter_vhdl_actel.prj` (VHDL project file)

Before you begin the tutorial, copy the files to a local directory and make sure that you have read and write permission for both the directory and files.

Note: While performing the tutorial, the active project (`.prj`) file will be updated; copying the files to a local directory preserves the original files installed in the tutorial directory.

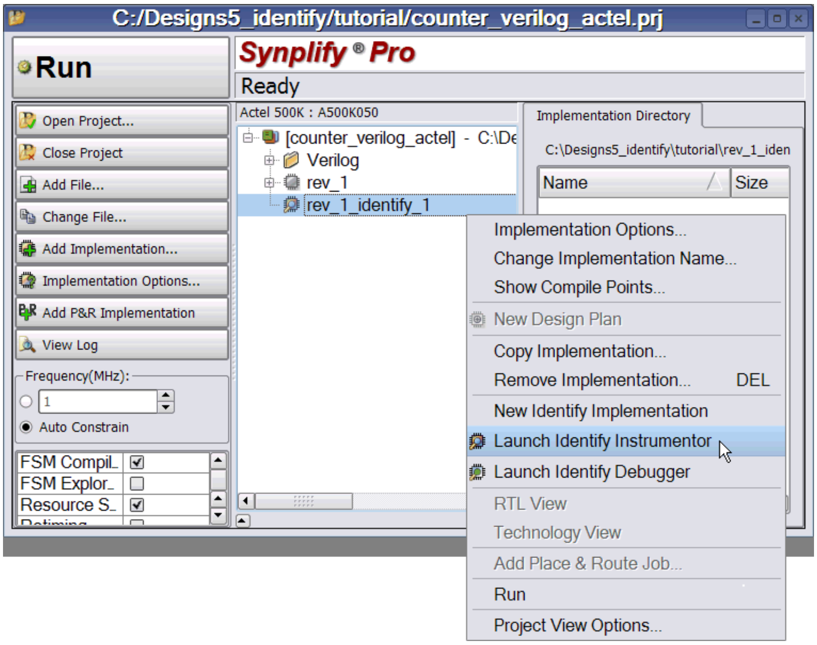
Launching the Identify Instrumentor

The Identify instrumentor is launched from the Synopsys Synplify Pro synthesis tool and is run prior to synthesis. Before starting the tutorial, copy the files from the tutorial directory to a local directory.

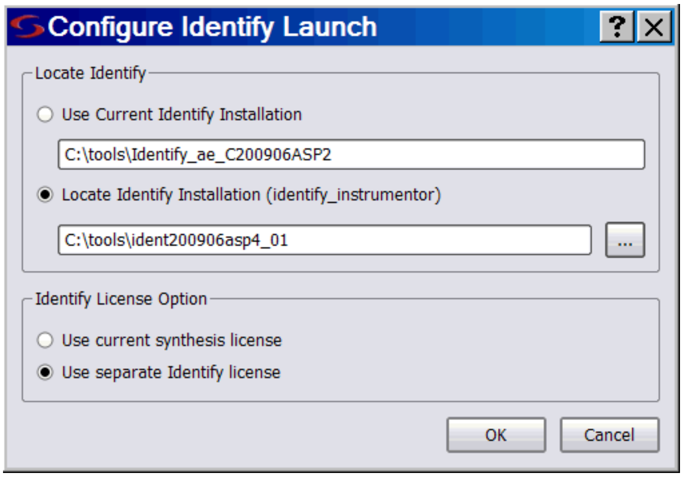
To launch the Identify instrumentor:

1. Start the Synopsys Synplify Pro synthesis tool.
2. In the project view, click the Open Project button to display the Open Project dialog box and click the Existing Project button.
3. Navigate to the tutorial directory where the Identify tool set is installed. This directory includes the HDL design files and a set of Actel-specific project files for both Verilog and VHDL implementations.
4. Select (open) the desired, Actel-specific project file.

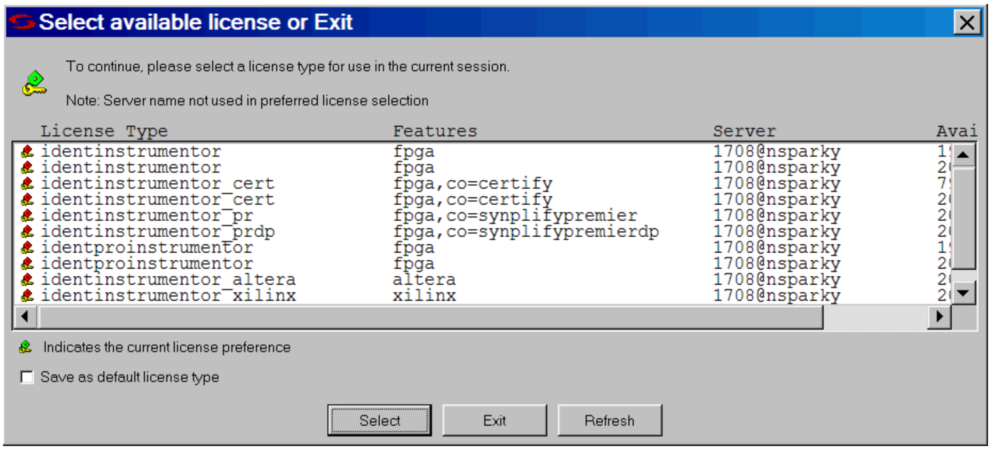
5. Right click on the Identify implementation and select Launch Identify Instrumentor from the popup menu.



6. If prompted, enter the location of the Identify installation in the Configure Identify Launch dialog box, click the Locate Identify Installation radio button, and click OK to launch the Identify instrumentor..


Configure Identify Launch
Locate Identify
Use Current Identify Installation
C:\tools\Identify_ae_C200906ASP2
Locate Identify Installation (identify_instrumentor)
C:\tools\ident200906asp4_01
...
Identify License Option
Use current synthesis license
Use separate Identify license
OK
Cancel

7. If prompted for a license, select a license from the list of available licenses displayed and click Select.



Select available license or Exit

To continue, please select a license type for use in the current session.

Note: Server name not used in preferred license selection

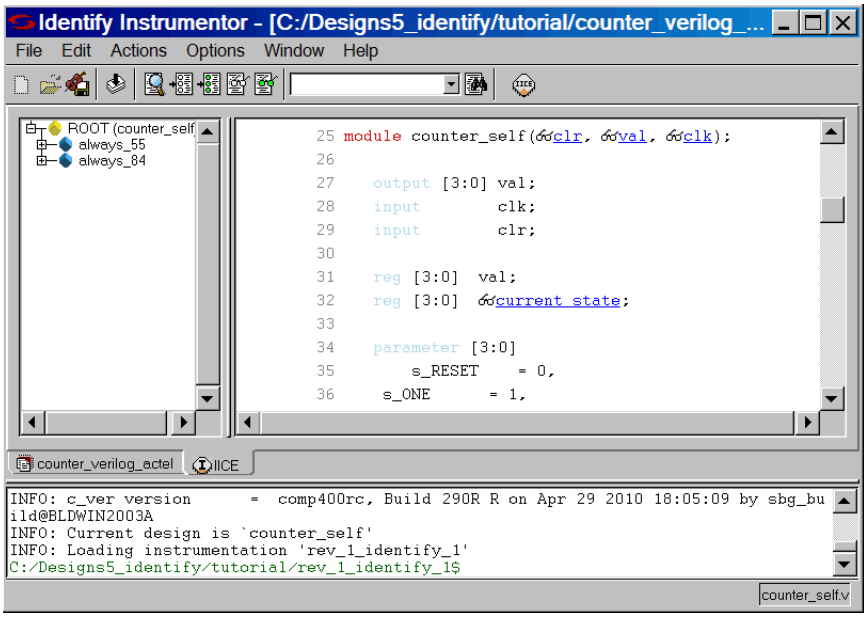
License Type	Features	Server	Avai
identinstrumentor	fpga	1708@nsparky	1
identinstrumentor	fpga	1708@nsparky	2
identinstrumentor_cert	fpga,co=certify	1708@nsparky	7
identinstrumentor_cert	fpga,co=certify	1708@nsparky	2
identinstrumentor_pr	fpga,co=synplifypremier	1708@nsparky	2
identinstrumentor_prdp	fpga,co=synplifypremierdp	1708@nsparky	2
identproinstrumentor	fpga	1708@nsparky	1
identproinstrumentor	fpga	1708@nsparky	2
identinstrumentor_altera	altera	1708@nsparky	2
identinstrumentor_xilinx	xilinx	1708@nsparky	2

Indicates the current license preference

☐ Save as default license type

Note: To avoid being prompted for a license each time you startup the Identify instrumentor, check the Save as default license type box before selecting your license.

The figure on the following shows the initial Identify instrumentor window as launched from the Synplify Pro tool on the Verilog version of the tutorial. The window shows the design hierarchy on the left and the HDL file content with all the potential instrumentation marked and available for selection on the right.


Identify Instrumentor - [C:/Designs5_identify/tutorial/counter_verilog_...
File Edit Actions Options Window Help
ROOT (counter_self
always_55
always_84
counter_verilog_actel
IICE
counter_self.v

```
module counter_self(clr, val, clk);

  output [3:0] val;
  input        clk;
  input        clr;

  reg [3:0]  val;
  reg [3:0]  current_state;

  parameter [3:0]
      s_RESET     = 0,
   s_ONE       = 1,
```

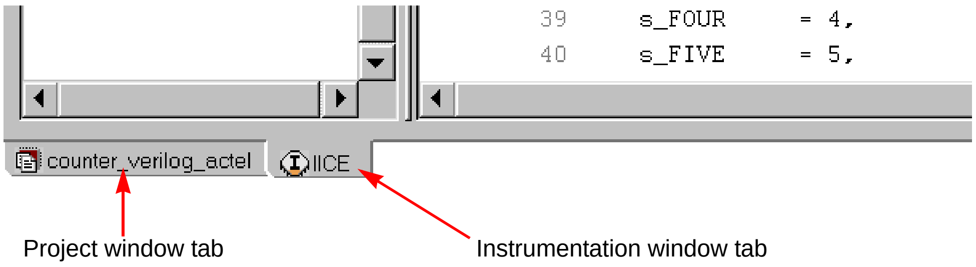
```
INFO: c_ver version      =  comp400rc, Build 290R R on Apr 29 2010 18:05:09 by sbg_bu
ild@BLDWIN2003A
INFO: Current design is `counter_self'
INFO: Loading instrumentation 'rev_1_identify_1'
C:/Designs5_identify/tutorial/rev_1_identify_1$
```

Setting up the IICE

After you have launched the Identify instrumentor for the tutorial project, you need to configure your IICE. The IICE settings common to all IICE units for an instrumentation are set in the project window, and the individual IICE settings unique to each IICE in a multi-IICE configuration are set on the IICE Configuration dialog box. Although the tutorial uses a single IICE, you must set both the common IICE parameters in the project window and the individual IICE parameters in the IICE Configuration dialog box.

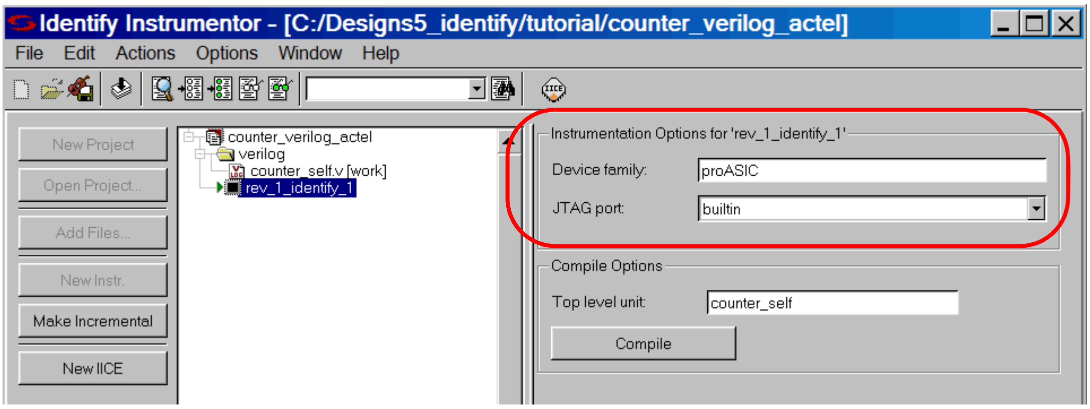
Setting the Common IICE Parameters

The common IICE parameters are set in the project window. To redisplay the project window, click the project window tab along the bottom of the instrumentation window.



After a project is loaded, the common IICE parameters appear in the Implementation Options block on the right side of the project window above the compile options as shown in the following figure. These parameters:

- show the device family
- select the JTAG port for the communication between the hardware and the Identify debugger



Instrumentation Options for 'rev_1_identify_1'
Device family: proASIC
JTAG port: builtin

Compile Options
Top level unit: counter_self

For the tutorial design:

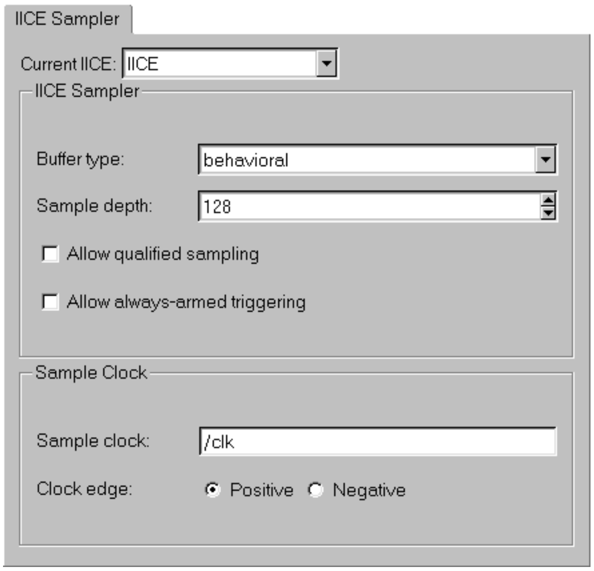
- The device family (proASIC) is specified in the selected project file and reported in the first field.
- Select which connection to use for the communication between the Identify debugger and your hardware from the drop-down list in the JTAG port field. For the tutorial, you can select either the builtin or soft connection.

Setting the Individual IICE Parameters

The individual IICE parameters are set by tabs on the IICE Configuration dialog box. Click on the Edit IICE settings icon in the toolbar or select Actions->Configure IICE from the menu to bring up the IICE Sampler tab shown in the following figure.

IICE Sampler Tab

The IICE Sampler tab defines the buffer type and sample depth of the data sampling hardware, controls the two optional sampling modes, and defines the sample clock and clock edge.



Current IICE: IICE

Buffer type: behavioral

Sample depth: 128

☐ Allow qualified sampling

☐ Allow always-armed triggering

Sample clock: /clk

Clock edge: (● Positive) ○ Negative

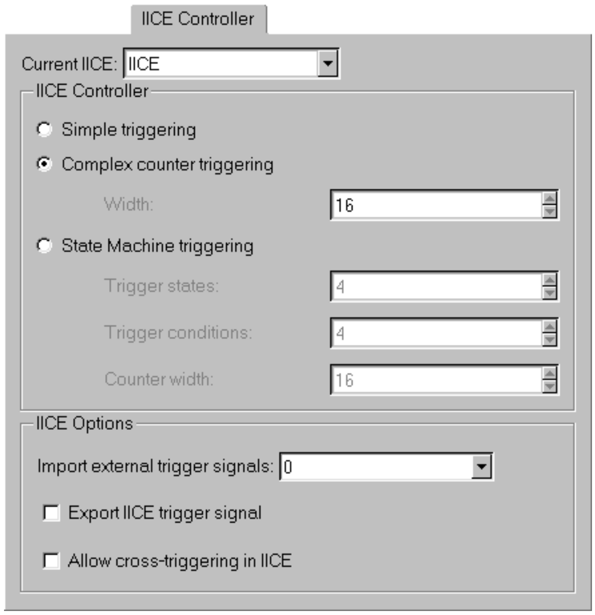
For the tutorial design:

- Leave Buffer type set to behavioral (only supported type)
- Select 128 for the sample buffer depth.
- Leave the Allow qualified sampling check box unchecked
- Leave the Allow always-armed sampling check box unchecked
- Enter `/clk` for the sample clock and select the positive polarity for the clock edge.

After you have set and/or verified the above IICE Sampler tab settings, click the IICE Controller tab.

IICE Controller Tab

The IICE Controller tab selects the type of triggering.



IICE Controller

Current IICE: IICE

IICE Controller

( ) Simple triggering
(●) Complex counter triggering
Width: 16
( ) State Machine triggering
Trigger states: 4
Trigger conditions: 4
Counter width: 16

IICE Options

Import external trigger signals: 0
☐ Export IICE trigger signal
☐ Allow cross-triggering in IICE

For the tutorial design:

- Make sure that the Complex counter triggering radio button is selected and that the Width is set to 16.
- Leave the Import external trigger signals value at 0.
- Leave the Export IICE trigger signal check box unchecked.
- Leave the Allow cross-triggering in IICE check box unchecked.

When all of the IICE configuration settings have been made or verified, click the OK button at the bottom of the dialog box.

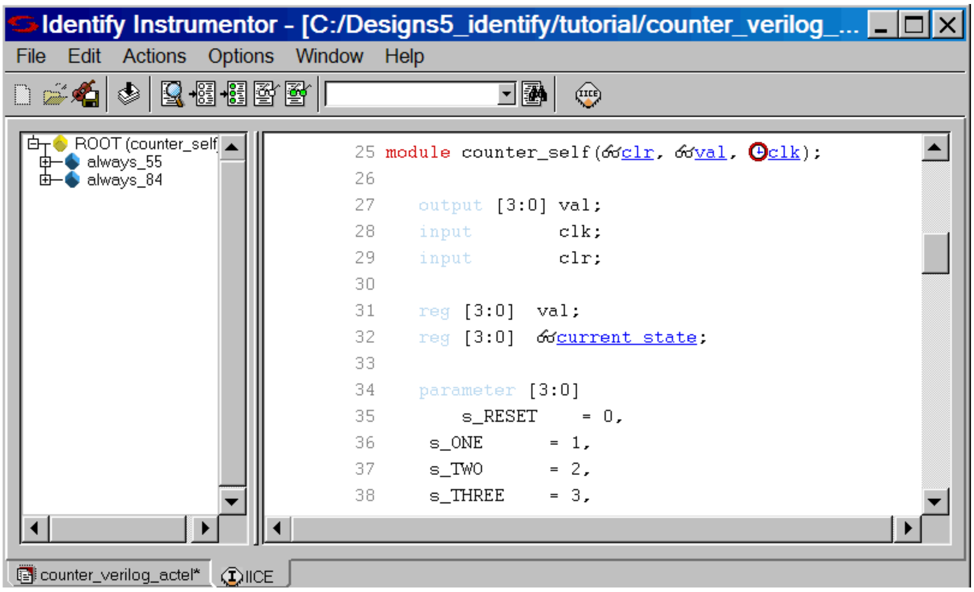
Selecting the Instrumentation

The steps for instrumenting a design file differ slightly from VHDL to Verilog.

- For VHDL designs, see VHDL Design Instrumentation, below
- For Verilog designs, see Verilog Design Instrumentation

VHDL Design Instrumentation

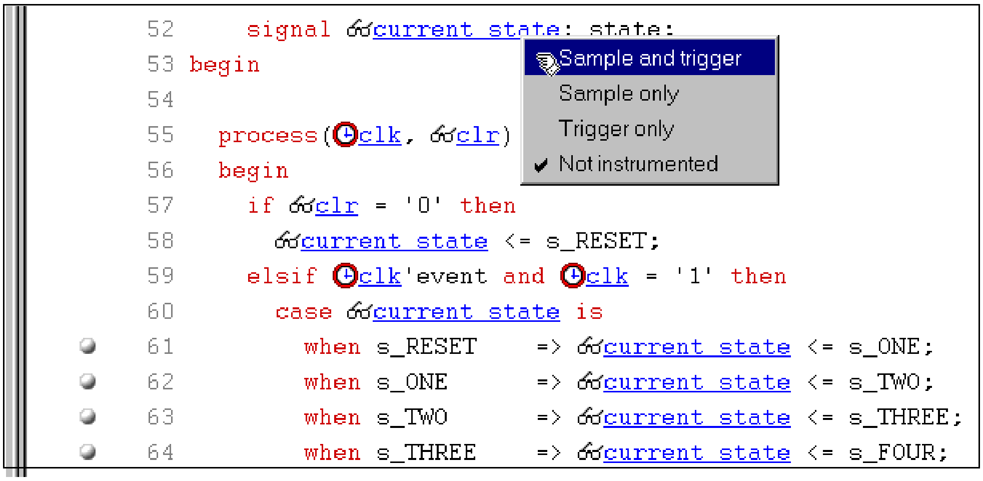
When your VHDL design compiles successfully, the instrumentation window displays the top-level entity VHDL code on the right and the hierarchy browser on the left (if the instrumentation window is not displayed, click on the "IICE" tab at the bottom of the project window). Use the hierarchy browser to navigate through your design. Clicking on a hierarchical node displays the corresponding VHDL source code in the source code display on the right.



```
module counter_self(clr, val, clk);

  output [3:0] val;
  input        clk;
  input        clr;

  reg [3:0]  val;
  reg [3:0]  current_state;

  parameter [3:0]
        s_RESET    = 0,
     s_ONE      = 1,
     s_TWO      = 2,
     s_THREE    = 3,
```

Selecting Watch Points

In the source code display, scroll down and select the signal `current_state` on line 52 for instrumentation by clicking on the watch-point (glasses) icon displayed next to its name. When you click on the icon (or on the signal name), a popup menu is displayed to allow you to select how the watch-point signal is to be instrumented.



```
52     signal current_state: state;
53 begin
54
55   process(clk, clr)
56   begin
57     if clr = '0' then
58       current_state <= s_RESET;
59     elsif clk'event and clk = '1' then
60       case current_state is
61         when s_RESET     => current_state <= s_ONE;
62         when s_ONE       => current_state <= s_TWO;
63         when s_TWO       => current_state <= s_THREE;
64         when s_THREE     => current_state <= s_FOUR;
```

When you select a watch-point instrumentation type from the popup menu, the icon changes color according to the type selected as shown in the following table.

Icon Color	Watch-Point Selection
Green	Sample and trigger
Blue	Sample only
Pink	Trigger only
Clear (unfilled)	Not instrumented

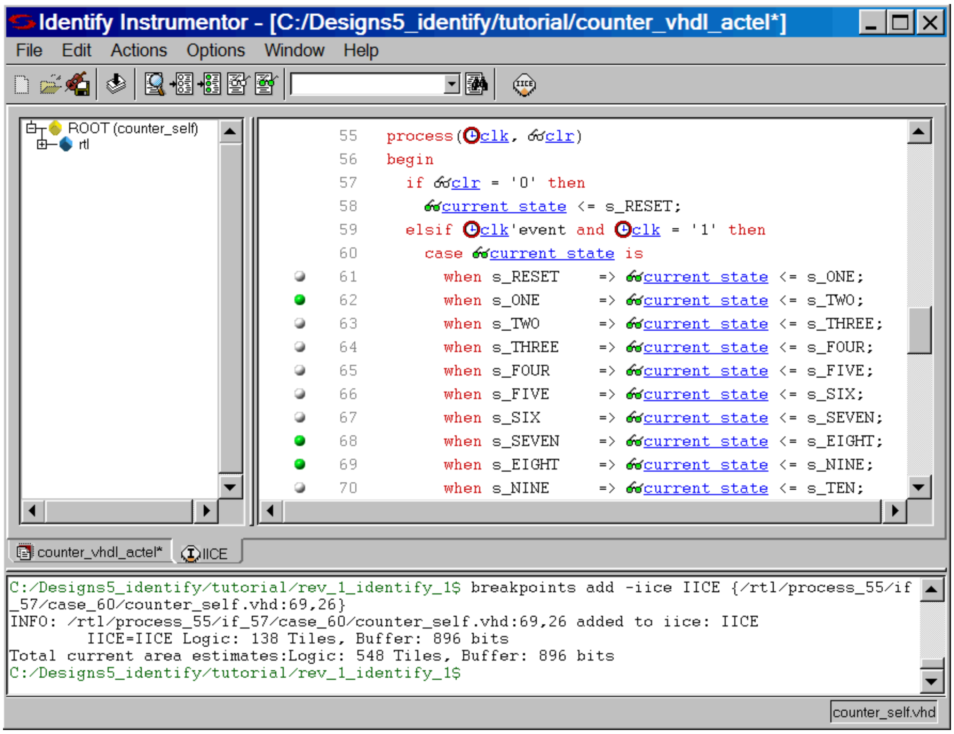
For the tutorial design, select Sample and trigger. The icons preceding each occurrence of the `current_state` signal in the VHDL code will be green.

The command that you would enter manually or in a shell script to define the watch-point signal (signals add -sample -trigger -iice IICE /current_state) and an estimate of the hardware overhead required to implement the selected instrumentation are displayed in the console window. The hardware estimate is based on the selected IICE technology.

Selecting Breakpoints

The icons to the left of the line numbers beginning on line 61 select the corresponding breakpoint for instrumentation. When selected, the color of the icon changes to green. Click on the icons on lines 62, 68, and 69 to select their corresponding breakpoints.

Similar to the watch-point selection, the Identify instrumentor displays the commands that you would enter manually or in a shell script to define the breakpoints and the hardware overhead required to implement the selected instrumentations in the console window.

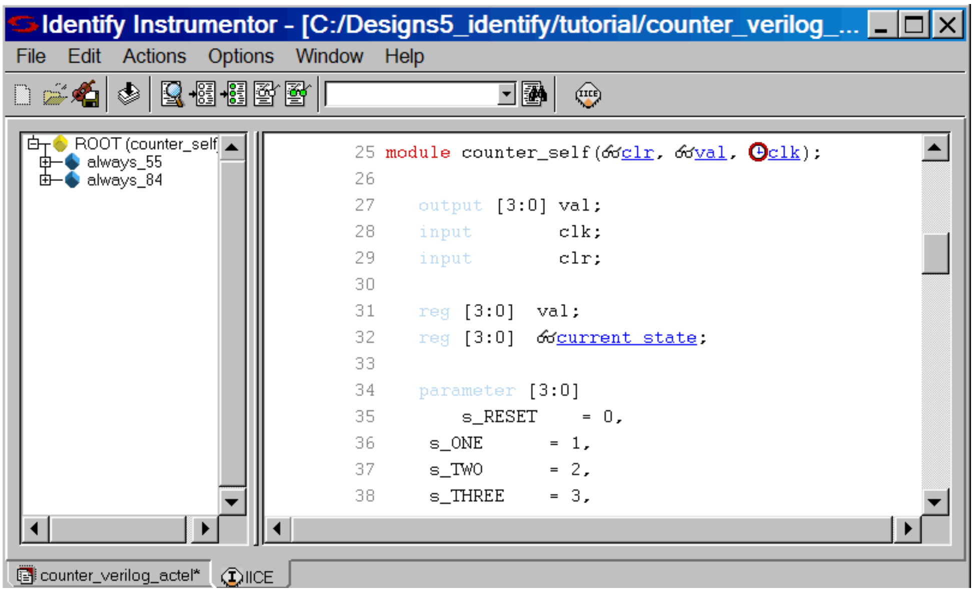


```
55      process(clk, clr)
56      begin
57        if clr = '0' then
58          current_state <= s_RESET;
59        elsif clk'event and clk = '1' then
60          case current_state is
○ 61          when s_RESET    => current_state <= s_ONE;
● 62          when s_ONE      => current_state <= s_TWO;
○ 63          when s_TWO      => current_state <= s_THREE;
○ 64          when s_THREE    => current_state <= s_FOUR;
○ 65          when s_FOUR     => current_state <= s_FIVE;
○ 66          when s_FIVE     => current_state <= s_SIX;
○ 67          when s_SIX      => current_state <= s_SEVEN;
● 68          when s_SEVEN    => current_state <= s_EIGHT;
● 69          when s_EIGHT    => current_state <= s_NINE;
○ 70          when s_NINE     => current_state <= s_TEN;
```

```
C:/Designs5_identify/tutorial/rev_1_identify_1$ breakpoints add -iice IICE {/rtl/process_55/if
_57/case_60/counter_self.vhd:69,26}
INFO: /rtl/process_55/if_57/case_60/counter_self.vhd:69,26 added to iice: IICE
        IICE=IICE Logic: 138 Tiles, Buffer: 896 bits
Total current area estimates:Logic: 548 Tiles, Buffer: 896 bits
C:/Designs5_identify/tutorial/rev_1_identify_1$
```

Verilog Design Instrumentation

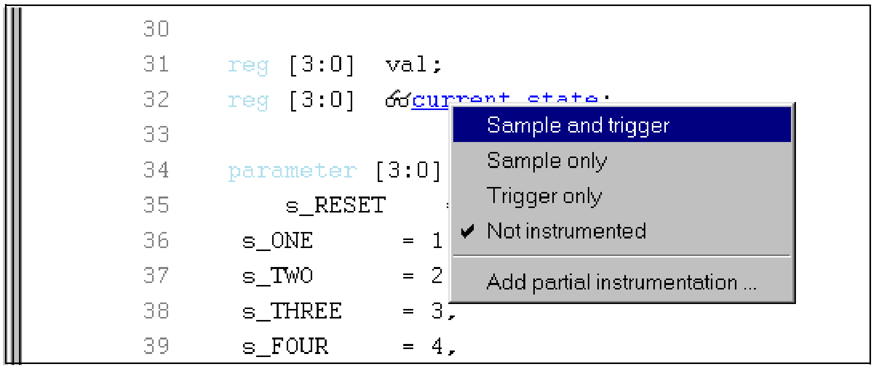
The instrumentation window displays the Verilog source code for the top-level module on the right and the design hierarchy in the hierarchy browser on the left. Use the hierarchy browser to navigate through your design. Clicking on a hierarchical node displays the corresponding Verilog source code in the source code display on the right.


Identify Instrumentor - [C:/Designs5_identify/tutorial/counter_verilog_...
File Edit Actions Options Window Help
ROOT (counter_self
always_55
always_84
counter_verilog_actel*
IICE

```
module counter_self(clr, val, clk);

  output [3:0] val;
  input        clk;
  input        clr;

  reg [3:0]  val;
  reg [3:0]  current_state;

  parameter [3:0]
        s_RESET    = 0,
   s_ONE       = 1,
   s_TWO       = 2,
   s_THREE     = 3,
```

Selecting Watch Points

In the source code display, select the registered signal `current_state` on line 32 for instrumentation by clicking on the watch-point (glasses) icon displayed next to its name. When you click on the icon (or on the signal name), a popup menu is displayed to allow you to select how the watch-point signal is to be instrumented.



```
30
31    reg [3:0]  val;
32    reg [3:0]  current_state;
33
34    parameter [3:0]
35          s_RESET
36     s_ONE        = 1
37     s_TWO        = 2
38     s_THREE      = 3,
39     s_FOUR       = 4,
```

Once selected, the icon changes color according to the type of watch-point instrumentation selected as shown in the following table.

Icon Color	Watch-Point Selection
Green	Sample and trigger
Blue	Sample only
Pink	Trigger only
Clear (unfilled)	Not instrumented

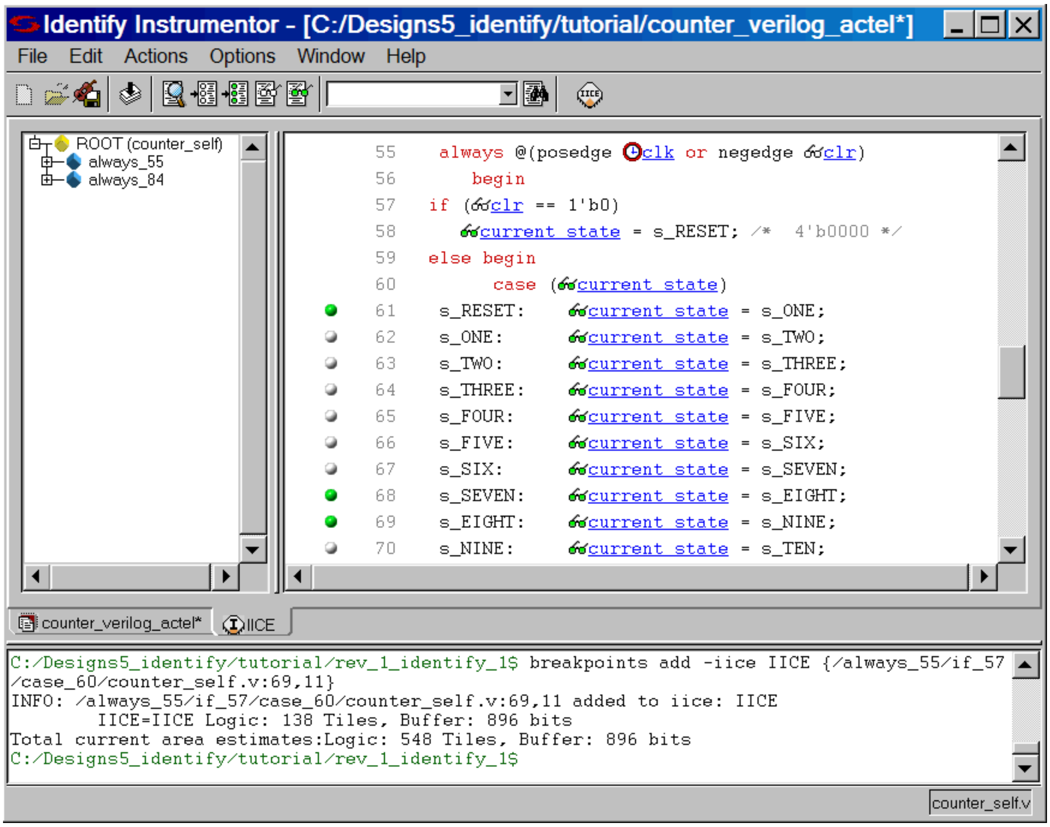
For the tutorial, select Sample and trigger. The icons preceding each occurrence of the `current_state` signal in the Verilog code will be green.

The command that you would enter manually or in a shell script to define the watch-point signal (signals add -sample -trigger -iice IICE /current_state) and an estimate of the hardware overhead required to implement the selected instrumentation are displayed in the console window. The hardware estimate is based on the IICE technology settings.

Selecting Breakpoints

Scroll down in the source code display until the body of the `always` block on line 55 is at the top of the instrumentation window. The icons to the left of the line numbers beginning on line 61 select the corresponding breakpoint for instrumentation. When selected, the color of the icon changes to green. Click on the icons on lines 62, 68, and 69 to select their corresponding break-points.

Similar to the watch-point selection, the Identify instrumentor displays the commands that you would enter manually or in a shell script to define the breakpoints and the hardware overhead required to implement the selected instrumentations in the console window.



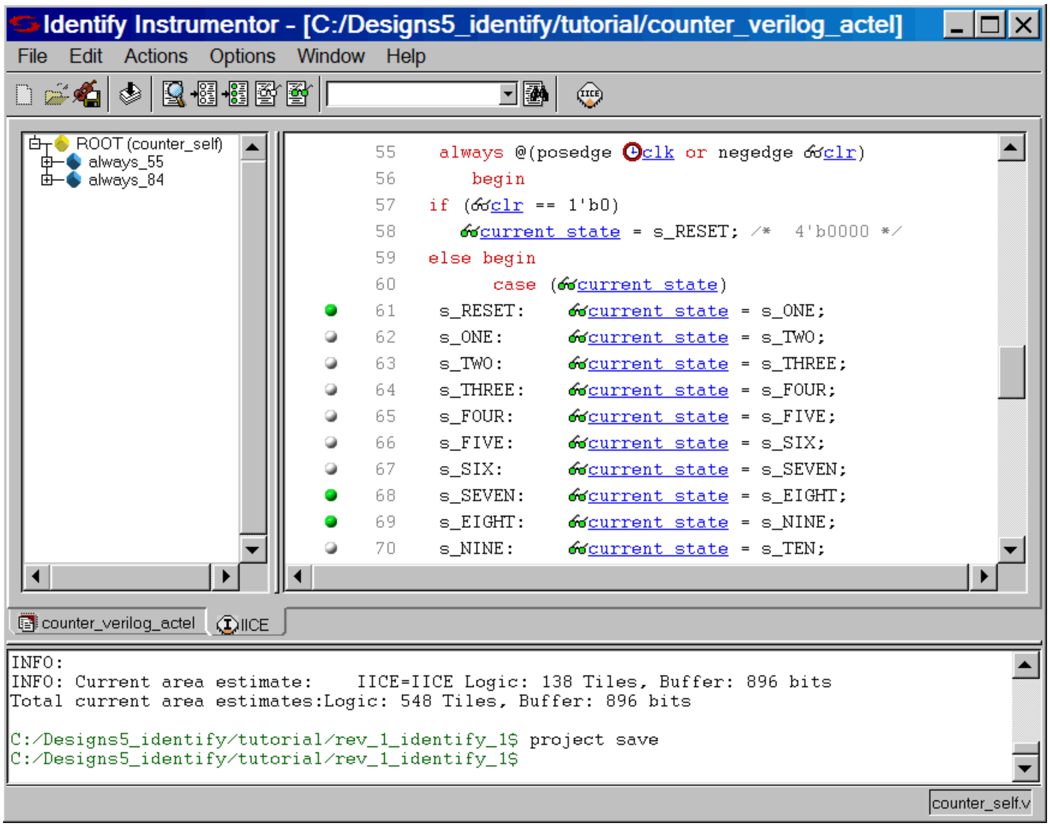
Identify Instrumentor - [C:/Designs5_identify/tutorial/counter_verilog_actel*]
File Edit Actions Options Window Help
ROOT (counter_self)
always_55
always_84
counter_verilog_actel*
IICE
counter_self.v

```
55     always @(posedge clk or negedge clr)
56        begin
57   if (clr == 1'b0)
58      current_state = s_RESET; /*  4'b0000 */
59   else begin
60         case (current_state)
61    s_RESET:    current_state = s_ONE;
62    s_ONE:      current_state = s_TWO;
63    s_TWO:      current_state = s_THREE;
64    s_THREE:    current_state = s_FOUR;
65    s_FOUR:     current_state = s_FIVE;
66    s_FIVE:     current_state = s_SIX;
67    s_SIX:      current_state = s_SEVEN;
68    s_SEVEN:    current_state = s_EIGHT;
69    s_EIGHT:    current_state = s_NINE;
70    s_NINE:     current_state = s_TEN;
```

```
C:/Designs5_identify/tutorial/rev_1_identify_1$ breakpoints add -iice IICE {/always_55/if_57
/case_60/counter_self.v:69,11}
INFO: /always_55/if_57/case_60/counter_self.v:69,11 added to iice: IICE
        IICE=IICE Logic: 138 Tiles, Buffer: 896 bits
Total current area estimates:Logic: 548 Tiles, Buffer: 896 bits
C:/Designs5_identify/tutorial/rev_1_identify_1$
```

Writing the Instrumented Design

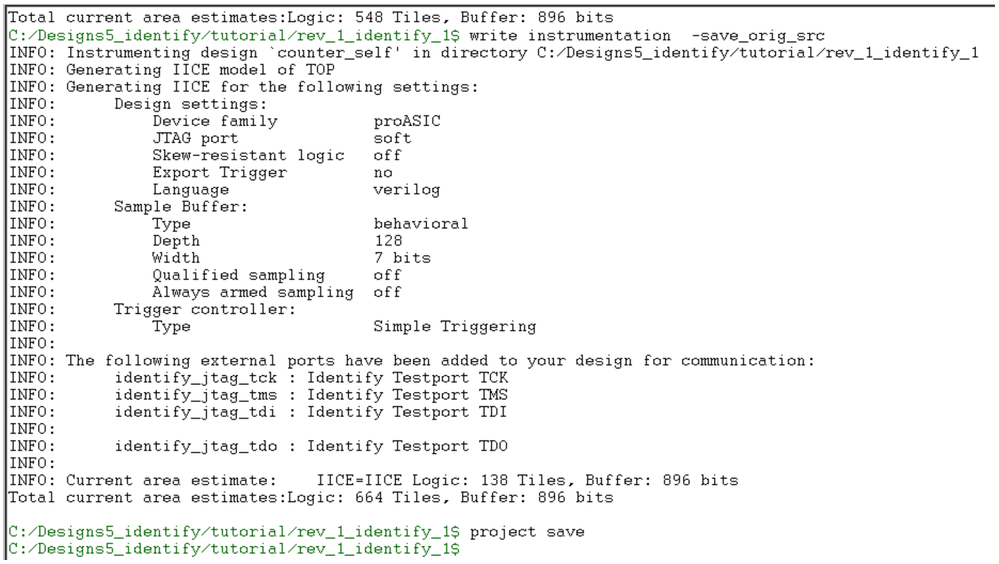
To write the instrumented design, select File->Save project from the menu or click on the Save and Instrument current project icon on the toolbar. Saving the project automatically creates a subdirectory named `rev_1_identify_1` which contains the instrumented HDL code for the tutorial design and also creates the Identify project file tutorial.bsp which is read by the Identify debugger. The new subdirectory and the project file are written to the tutorial directory.



```
always @(posedge clk or negedge clr)
  begin
if (clr == 1'b0)
   current state = s_RESET; /*  4'b0000 */
else begin
       case (current state)
 s_RESET:    current state = s_ONE;
 s_ONE:      current state = s_TWO;
 s_TWO:      current state = s_THREE;
 s_THREE:    current state = s_FOUR;
 s_FOUR:     current state = s_FIVE;
 s_FIVE:     current state = s_SIX;
 s_SIX:      current state = s_SEVEN;
 s_SEVEN:    current state = s_EIGHT;
 s_EIGHT:    current state = s_NINE;
 s_NINE:     current state = s_TEN;
```

```
INFO:
INFO: Current area estimate:    IICE=IICE Logic: 138 Tiles, Buffer: 896 bits
Total current area estimates:Logic: 548 Tiles, Buffer: 896 bits

C:/Designs5_identify/tutorial/rev_1_identify_1$ project save
C:/Designs5_identify/tutorial/rev_1_identify_1$
```

As shown in the following figure, the console window displays feedback as to which type of IICE was created.



```
Total current area estimates:Logic: 548 Tiles, Buffer: 896 bits
C:/Designs5_identify/tutorial/rev_1_identify_1$ write instrumentation  -save_orig_src
INFO: Instrumenting design `counter_self' in directory C:/Designs5_identify/tutorial/rev_1_identify_1
INFO: Generating IICE model of TOP
INFO: Generating IICE for the following settings:
INFO:      Design settings:
INFO:          Device family          proASIC
INFO:          JTAG port              soft
INFO:          Skew-resistant logic   off
INFO:          Export Trigger         no
INFO:          Language               verilog
INFO:      Sample Buffer:
INFO:          Type                   behavioral
INFO:          Depth                  128
INFO:          Width                  7 bits
INFO:          Qualified sampling     off
INFO:          Always armed sampling  off
INFO:      Trigger controller:
INFO:          Type                   Simple Triggering
INFO:
INFO: The following external ports have been added to your design for communication:
INFO:      identify_jtag_tck : Identify Testport TCK
INFO:      identify_jtag_tms : Identify Testport TMS
INFO:      identify_jtag_tdi : Identify Testport TDI
INFO:
INFO:      identify_jtag_tdo : Identify Testport TDO
INFO:
INFO: Current area estimate:    IICE=IICE Logic: 138 Tiles, Buffer: 896 bits
Total current area estimates:Logic: 664 Tiles, Buffer: 896 bits

C:/Designs5_identify/tutorial/rev_1_identify_1$ project save
C:/Designs5_identify/tutorial/rev_1_identify_1$
```

Please note that depending on your JTAG port selection in the project window, you may see a notice that external ports have been added to the instrumented version of your design. This type of information varies with your device family, JTAG port settings, and IICE options. For example, if you select JTAG port soft, four extra ports (three input and one output) are added to your design. In this case, you must have access to these ports on your board in order to connect the JTAG communication cable to these ports. Refer to the User Guide for more information.

CHAPTER 4

Implementing the Design

At this time, you should have successfully completed the instrumentation of the tutorial design and saved the instrumentation project.

Implementing the design involves the following steps:

- Synthesizing the VHDL or Verilog
- Placing and routing to produce a bit file
- Programming the device with the bit file

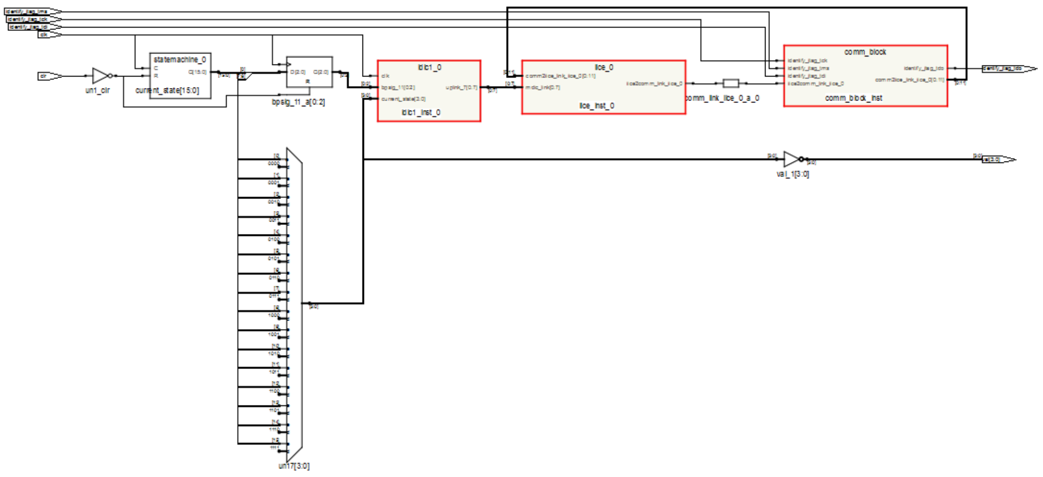
Synthesis translates the instrumented VHDL or Verilog code to a mapped netlist. Place-and-route tools further process this EDIF netlist to create the implementation of the instrumented design.

Synthesis

The Identify Actel Edition tool set is expressly designed to work with the Synplify Pro synthesis tool. In the tutorial directory where you saved the project at the end of instrumentation phase, you will have two new entries:

- `counter_verilog_actel.bsp` or `counter_vhdl_actel.bsp` – the project file that contains all of your project settings. This file is read when you synthesize your instrumented project in the synthesis tool. This file also can be used to reread your project into the Identify instrumentor or to read your instrumented project into the Identify debugger in standalone mode.
- `rev_1_identify_1/instr_sources` – a subdirectory containing the instrumented HDL source code for the tutorial design (`counter_self.v` or `counter_self.vhd`) plus an additional HDL file named `syn_dics.v` (Verilog source) or `syn_dics.vhd` (VHDL source). The additional syn_discs file contains the logic that implements the IICE.

To synthesize your project in Synplify Pro, highlight the Identify implementation and click the Run button. Following synthesis, you can view the additional IICE sampling logic in the RTL view (shown below in red) or Technology view.



Place and Route

After synthesis, run the file generated by the Synplify Pro synthesis tool through your Actel place and route tool. Running the file of the instrumented design through place and route is identical to running place and route on the original design. Refer to your place and route tool documentation for further information.

If you have selected the soft JTAG port setting, four external ports have been added to the tutorial design. These ports are:

- identify_jtag_tdi (input serial data IN signal.)
- identify_jtag_tck (input asynchronous clock signal)
- identify_jtag_tms (input control signal)
- identify_jtag_tdo (output serial data OUT signal.)

You must provide proper pin locations for these ports to your place and route tools. These pins must be connected to your JTAG cable to enable communication with the Identify debugger. Please refer to the user guide for more information about this process.

Program the Device

Programming the target device with the bit file of the instrumented design is identical to programming the target device with the original design's bit file. Please refer to your programming tool documentation for further information.

CHAPTER 5

Debugging Your Design

This chapter shows you how to debug the tutorial design you instrumented earlier. To proceed with debugging, your design must be synthesized, run through place and route, and programmed into the Actel FPGA device. Also, the device must be connected to the host machine using the proper cable and pin connections. The debugging process consists of setting up the debugging environment, enabling triggers in the IICE, and examining the data downloaded from your hardware. The debug flow is:

- Starting the Identify Debugger
- Specifying the JTAG Cable
- Setting the JTAG Chain
- Setting Up Triggers and Capturing Data
- Generating Waveforms

Starting the Identify Debugger

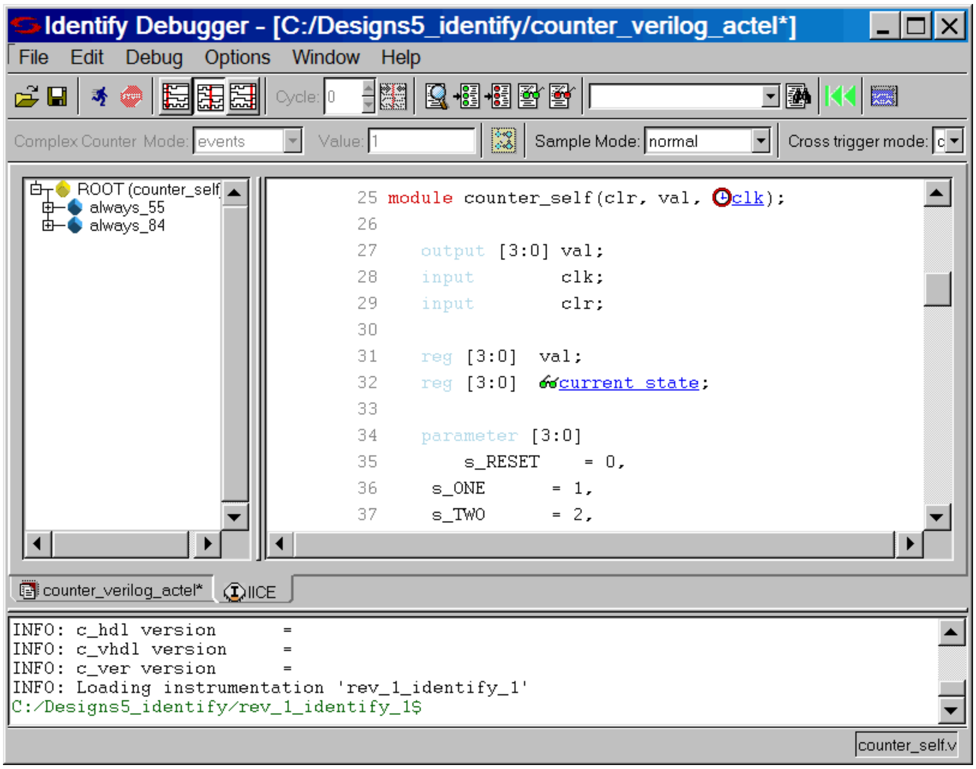
To launch the Identify debugger from the Synplify Pro synthesis tool:

1. Highlight the Identify implementation.
2. With the right mouse button, select Launch Identify Debugger from the popup menu or click the Launch Identify Debugger icon in the top menu bar.

3. If you are prompted for a license, select a license from the list of available licenses and click Select.

Note: To avoid being prompted for a license each time you startup the Identify debugger, check the Save as default license type box before clicking Select.

The Identify debugger automatically loads the project file for the tutorial design and opens the instrumentation window with the hierarchy browser displayed on the left and the HDL source code for the design displayed on the right. Note that the only instrumentations that are visible in the source code display are the breakpoints and watchpoints that you selected during the instrumentation phase.



```
module counter_self(clr, val, clk);

  output [3:0] val;
  input        clk;
  input        clr;

  reg [3:0]  val;
  reg [3:0]  current_state;

  parameter [3:0]
      s_RESET    = 0,
   s_ONE      = 1,
   s_TWO      = 2,
```

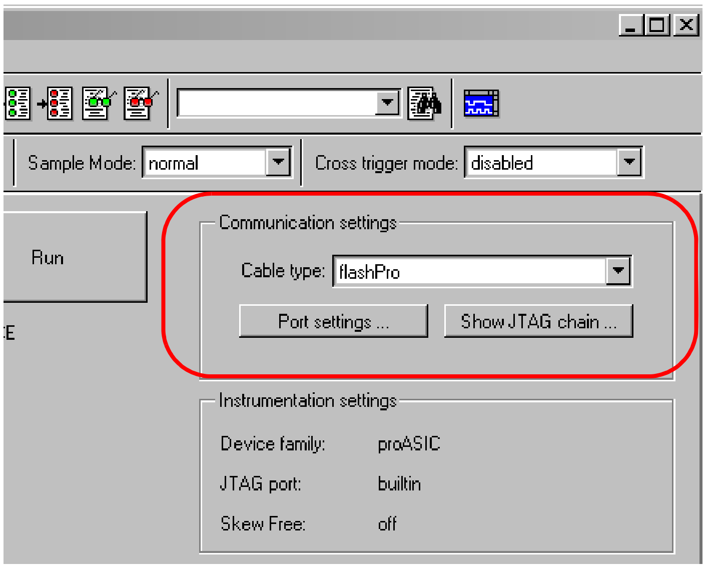
```
INFO: c_hdl version       =
INFO: c_vhdl version      =
INFO: c_ver version       =
INFO: Loading instrumentation 'rev_1_identify_1'
C:/Designs5_identify/rev_1_identify_1$
```

Specifying the JTAG Cable

The type of cable used to communicate with the FPGA must be specified so that the correct protocol is used to communicate with the physical device. The JTAG cable and cable port are set in the Communication settings section of the project window after the project is read in. To restore this window, click its tab at the bottom of the window.

Note: The instrumentation settings that you specified during the instrumentation phase are also displayed in the project window. You cannot edit these settings – they are for information purposes only.

The hardware used to develop this tutorial used an Actel flashPro cable – your cable type may differ. Set the cable type and also set the Port Settings value (lpt1, lpt2, lpt3, or lpt4) to match the host-machine port connected to your JTAG cable. The default setting is lpt1.



Sample Mode: normal
Cross trigger mode: disabled

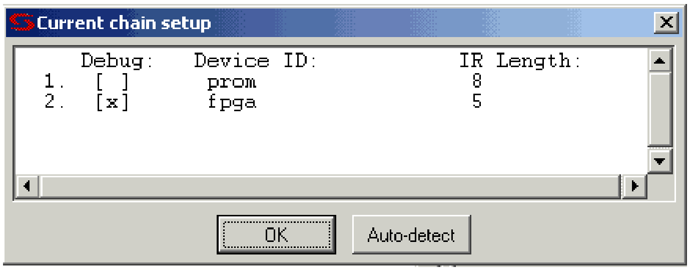
Communication settings
Cable type: flashPro

Instrumentation settings
Device family: proASIC
JTAG port: builtin
Skew Free: off

Setting the JTAG Chain

JTAG connections on an FPGA board usually chain devices together to form a serial chain of devices. This chain includes PROMs and other FPGA devices present on the board.

The Identify debugger automatically detects the JTAG chain at the beginning of the debug session. You can review the JTAG chain settings by clicking the Show JTAG chain button in the Communications settings section of the project window.



```
     Debug:    Device ID:            IR Length:
 1.   [ ]       prom                   8
 2.   [x]       fpga                   5
```

To enable the Identify debugger to properly communicate with the target device, the device chain must be configured correctly. If, for some reason, the JTAG chain cannot be successfully configured, you must manually specify the chain through a series of chain instructions entered in the console window.

Configuring a device chain is very similar to the steps required to program the device with a JTAG programmer.

For the Identify debugger, the devices in the chain must be known and specified. The following information is required to configure the device chain:

- the number of devices in the JTAG chain
- the length of the JTAG instruction register for each device

Instruction register length information is usually available in the `.bsd` file for the particular device. Specifically, it is the Instruction_length attribute listed in the `.bsd` file.

For the Actel board used in developing this tutorial, the following sequence of commands was used to specify a chain consisting of a PROM followed by the FPGA. The instruction length of the PROM is 8 while the instruction length of the FPGA is 5. Note that the chain select command identifies the instrumented device to the system. Identifying the instrumented device is essential when a board includes multiple FPGAs.

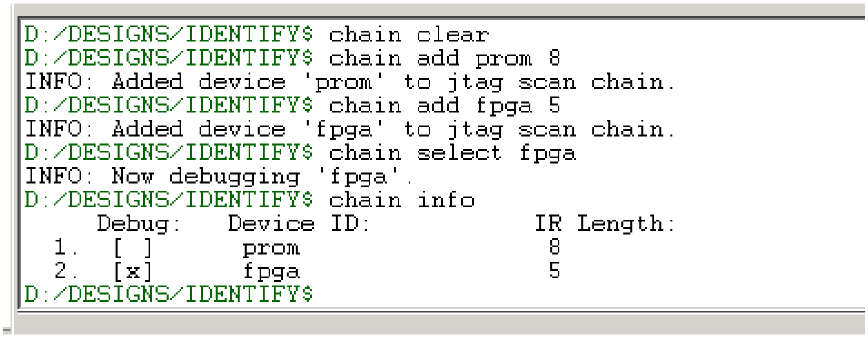
Note: The names PROM and FPGA have no meaning to the Identify debugger – they simply are used for convenience. The two devices could be named device1 and device2, and the debugger would function exactly the same.

Again, the sequence of chain commands is specific to the JTAG chain on your board, so while these are the chain commands for the board used to develop this tutorial, your board will most likely be different.

Type the following sequence in the console window of the Identify debugger:

```
chain clear
chain add prom 8
chain add fpga 5
chain select fpga
chain info
```

The following figure shows the results of the above command sequence.



```
D:/DESIGNS/IDENTIFY$ chain clear
D:/DESIGNS/IDENTIFY$ chain add prom 8
INFO: Added device 'prom' to jtag scan chain.
D:/DESIGNS/IDENTIFY$ chain add fpga 5
INFO: Added device 'fpga' to jtag scan chain.
D:/DESIGNS/IDENTIFY$ chain select fpga
INFO: Now debugging 'fpga'.
D:/DESIGNS/IDENTIFY$ chain info
     Debug:   Device ID:          IR Length:
  1.  [ ]      prom                8
  2.  [x]      fpga                5
D:/DESIGNS/IDENTIFY$
```

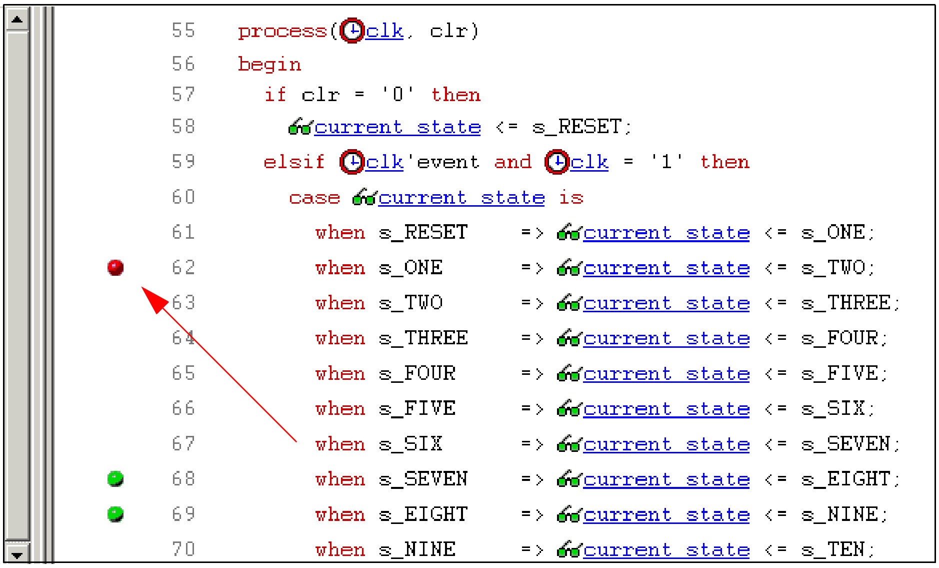
Setting Up Triggers and Capturing Data

Trigger operations include:

- Triggering on a Breakpoint
- Deactivating a Breakpoint
- Triggering on a Watchpoint
- Using the Complex Counter

Triggering on a Breakpoint

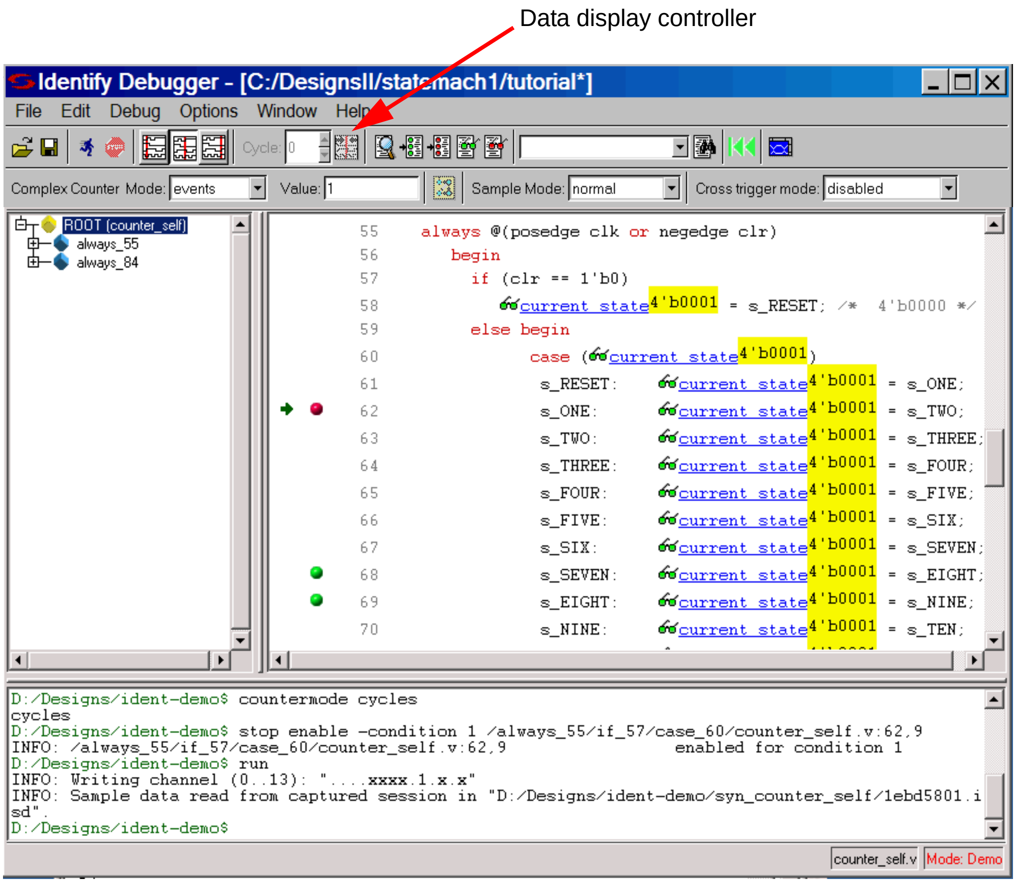
In the source code display, use the scroll bar to scroll down until the first breakpoint on line 62 is visible on the left of the source code and then click on the breakpoint to activate it.



```
55  process(clk, clr)
56  begin
57    if clr = '0' then
58      current_state <= s_RESET;
59    elsif clk'event and clk = '1' then
60      case current_state is
61        when s_RESET    => current_state <= s_ONE;
62        when s_ONE      => current_state <= s_TWO;
63        when s_TWO      => current_state <= s_THREE;
64        when s_THREE    => current_state <= s_FOUR;
65        when s_FOUR     => current_state <= s_FIVE;
66        when s_FIVE     => current_state <= s_SIX;
67        when s_SIX      => current_state <= s_SEVEN;
68        when s_SEVEN    => current_state <= s_EIGHT;
69        when s_EIGHT    => current_state <= s_NINE;
70        when s_NINE     => current_state <= s_TEN;
```

Notice that the breakpoint icon changes from green to red indicating that the breakpoint is active. This breakpoint triggers when the `current_state` signal has the value `s_ONE` on the positive edge of the sample clock.

Now that you have an active trigger condition, you can arm the IICE trigger circuits on the FPGA device. To arm the IICE trigger circuits, click the Run icon in the menu bar or redisplay the project window by clicking its tab at the bottom of the instrumentation window and then clicking the large Run button. Using the Run button in the project window allows you to individually select one or more IICE units in a multi-IICE configuration (the Run icon in the menu bar only arms the active IICE).



Cycle: 0

Complex Counter Mode: events

Value: 1

Sample Mode: normal

Cross trigger mode: disabled

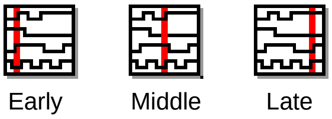
```
always @(posedge clk or negedge clr)
   begin
     if (clr == 1'b0)
        current_state4'b0001 = s_RESET; /*  4'b0000 */
     else begin
           case (current_state4'b0001)
             s_RESET:    current_state4'b0001 = s_ONE;
             s_ONE:      current_state4'b0001 = s_TWO;
             s_TWO:      current_state4'b0001 = s_THREE;
             s_THREE:    current_state4'b0001 = s_FOUR;
             s_FOUR:     current_state4'b0001 = s_FIVE;
             s_FIVE:     current_state4'b0001 = s_SIX;
             s_SIX:      current_state4'b0001 = s_SEVEN;
             s_SEVEN:    current_state4'b0001 = s_EIGHT;
             s_EIGHT:    current_state4'b0001 = s_NINE;
             s_NINE:     current_state4'b0001 = s_TEN;
```

```
D:/Designs/ident-demo$ countermode cycles
cycles
D:/Designs/ident-demo$ stop enable -condition 1 /always_55/if_57/case_60/counter_self.v:62,9
INFO: /always_55/if_57/case_60/counter_self.v:62,9                enabled for condition 1
D:/Designs/ident-demo$ run
INFO: Writing channel (0..13): "....xxxx.1.x.x"
INFO: Sample data read from captured session in "D:/Designs/ident-demo/syn_counter_self/1ebd5801.i
sd".
D:/Designs/ident-demo$
```

Clicking on the Run icon (or the Run button) downloads the trigger information to the IICE. The IICE now waits for the trigger to occur. When the trigger occurs, the sampled data is transferred back to the debugger where it is displayed in the source code display next to the sampled signals. Notice that the values in yellow adjacent to the sampled signals are the data sampled from the FPGA. Also, the small arrow to the left of the breakpoint icon indicates which breakpoint triggered (identifying which breakpoint triggered is important when multiple breakpoints are active).

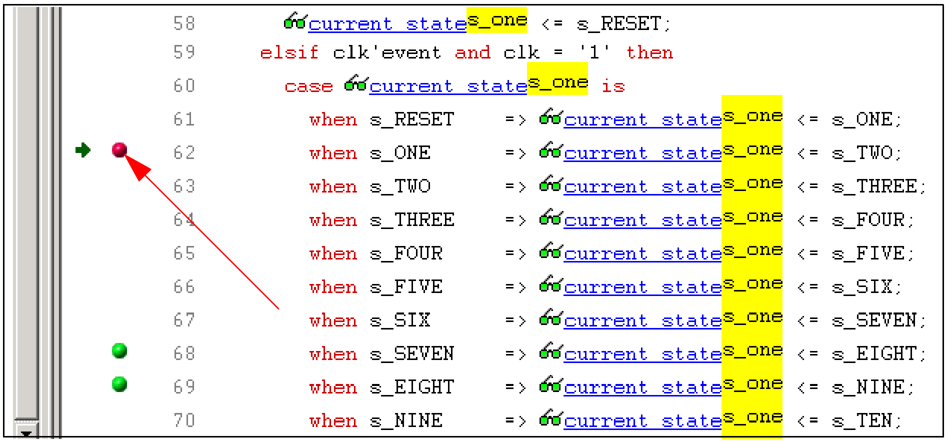
The Cycle display in the middle of the menu bar shows the value zero. This is the point in the sample data buffer where the trigger occurred. By clicking on the up-down arrows on the right, you can increase or decrease the cycle count to show values before or after the trigger point.

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



Deactivating a Breakpoint

The current breakpoint needs to be deactivated so that you can activate a single watchpoint in the next step. The breakpoint on line 62 must be deactivated because the trigger condition is cumulative; that is, all activated breakpoints and/or watchpoints always combine to create a complex trigger condition. Complex triggers are a powerful feature of the Identify debugger. However, in this particular case, only simple behavior is shown. To deactivate the current breakpoint on line 62, click on the button for the breakpoint once again so that the button changes from red back to green.



```
58        current_states_one <= s_RESET;
59      elsif clk'event and clk = '1' then
60        case current_states_one is
61          when s_RESET    => current_states_one <= s_ONE;
62          when s_ONE      => current_states_one <= s_TWO;
63          when s_TWO      => current_states_one <= s_THREE;
64          when s_THREE    => current_states_one <= s_FOUR;
65          when s_FOUR     => current_states_one <= s_FIVE;
66          when s_FIVE     => current_states_one <= s_SIX;
67          when s_SIX      => current_states_one <= s_SEVEN;
68          when s_SEVEN    => current_states_one <= s_EIGHT;
69          when s_EIGHT    => current_states_one <= s_NINE;
70          when s_NINE     => current_states_one <= s_TEN;
```

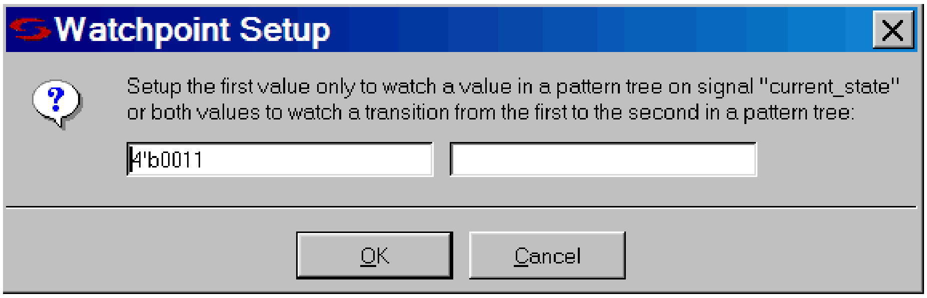
Triggering on a Watchpoint

The other method of specifying a trigger is the watchpoint. A watchpoint trigger can be specified on any sampled signal. The Watchpoint Setup dialog box contains a full HDL parser so that any legal VHDL or Verilog expression that evaluates to a constant can be used. When the signal with the watchpoint trigger changes to the value of the watch expression, the Identify debugger triggers.

To set a simple watchpoint:

1. Click on `current_state` signal
2. Select Set trigger expressions from the popup menu
3. In the First value field, enter `s_THREE` (VHDL) or `4'b0011` (Verilog)

4. Click OK

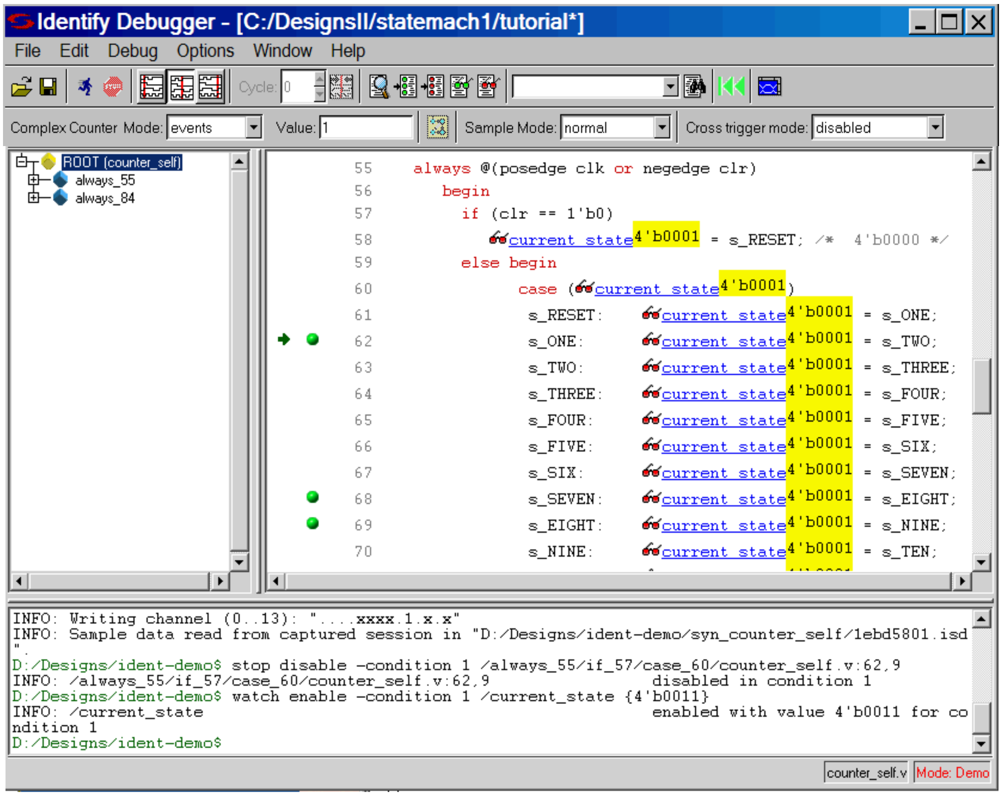


Watchpoint Setup

Setup the first value only to watch a value in a pattern tree on signal "current_state" or both values to watch a transition from the first to the second in a pattern tree:

4'b0011

This operation creates a watchpoint trigger on the `current_state` signal that triggers when `current_state` is `s_THREE` (VHDL) or `4'b0011` (Verilog). The setting of the watchpoint trigger is signified by the icons next to the `current_state` signal changing from green to red.



Identify Debugger - [C:/DesignsII/statemach1/tutorial*]

Cycle: 0

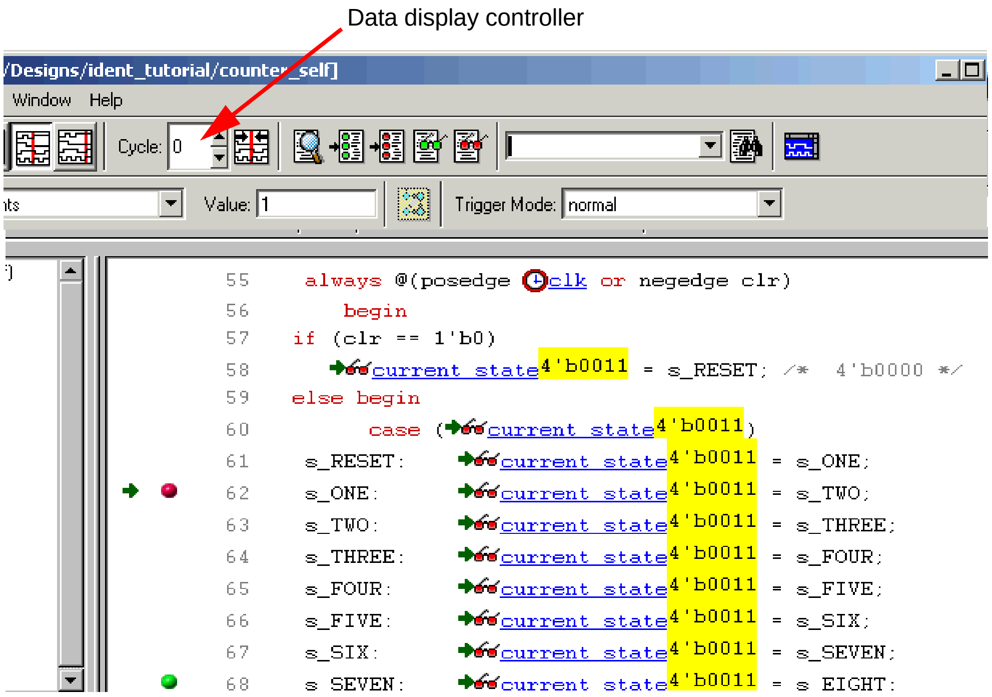
Complex Counter Mode: events Value: 1 Sample Mode: normal Cross trigger mode: disabled

```
always @(posedge clk or negedge clr)
  begin
    if (clr == 1'b0)
        current_state4'b0001 = s_RESET; /*  4'b0000 */
    else begin
          case (current_state4'b0001)
            s_RESET:     current_state4'b0001 = s_ONE;
            s_ONE:       current_state4'b0001 = s_TWO;
            s_TWO:       current_state4'b0001 = s_THREE;
            s_THREE:     current_state4'b0001 = s_FOUR;
            s_FOUR:      current_state4'b0001 = s_FIVE;
            s_FIVE:      current_state4'b0001 = s_SIX;
            s_SIX:       current_state4'b0001 = s_SEVEN;
            s_SEVEN:     current_state4'b0001 = s_EIGHT;
            s_EIGHT:     current_state4'b0001 = s_NINE;
            s_NINE:      current_state4'b0001 = s_TEN;
```

```
INFO: Writing channel (0..13): "....xxxx.1.x.x"
INFO: Sample data read from captured session in "D:/Designs/ident-demo/syn_counter_self/1ebd5801.isd
".
D:/Designs/ident-demo$ stop disable -condition 1 /always_55/if_57/case_60/counter_self.v:62,9
INFO: /always_55/if_57/case_60/counter_self.v:62,9          disabled in condition 1
D:/Designs/ident-demo$ watch enable -condition 1 /current_state {4'b0011}
INFO: /current_state                                        enabled with value 4'b0011 for co
ndition 1
D:/Designs/ident-demo$
```

counter_self.v Mode: Demo

Click the Run icon or button to download the watchpoint trigger information to the IICE. When signal `current_state` reaches the value s_THREE (VHDL) or `4'b0011` (Verilog), the IICE triggers and sends the captured data buffer to the debugger. Using the data display controller, you can now browse back and forth through the debugger data buffer to view the design activity.



```
always @(posedge clk or negedge clr)
    begin
if (clr == 1'b0)
    current_state 4'b0011 = s_RESET; /* 4'b0000 */
else begin
        case (current_state 4'b0011)
 s_RESET:        current_state 4'b0011 = s_ONE;
 s_ONE:          current_state 4'b0011 = s_TWO;
 s_TWO:          current_state 4'b0011 = s_THREE;
 s_THREE:        current_state 4'b0011 = s_FOUR;
 s_FOUR:         current_state 4'b0011 = s_FIVE;
 s_FIVE:         current_state 4'b0011 = s_SIX;
 s_SIX:          current_state 4'b0011 = s_SEVEN;
 s_SEVEN:        current_state 4'b0011 = s_EIGHT:
```

Using the Complex Counter

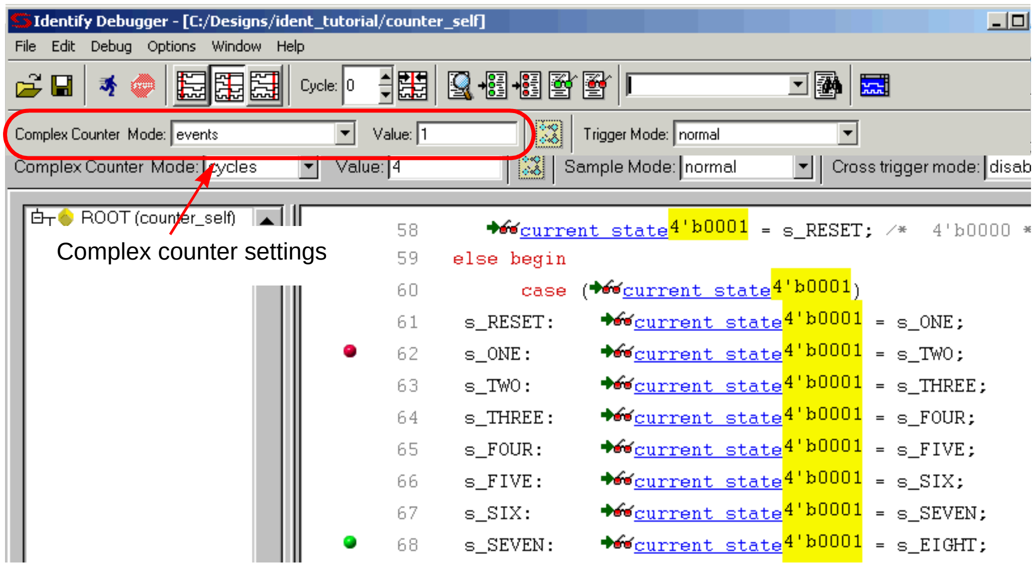
The tutorial design is instrumented using a 16-bit complex counter. Up to this point, the design has been debugged without using the counter. The counter is made to have no effect by setting the complex counter mode to events (the default) and setting the counter value to 1.

For this portion of the tutorial, the complex counter is set to the cycles mode. This mode waits for a breakpoint and/or watchpoint trigger event and then counts *N* cycles before triggering the sample buffer. Here, *N* is the counter value.

To display the operation of the cycle counter:

1. Change the watchpoint of signal `current_state` to `s_ONE` (VHDL) or `4'b0001` (Verilog).
2. Set the counter mode to cycles and the counter value to 4.
3. Click the Run icon or button and wait for the data to download.

The value at time zero should be `s_FIVE` (VHDL) or `4'b0101` (Verilog) as shown in the figure.


Identify Debugger - [C:/Designs/ident_tutorial/counter_self]
File Edit Debug Options Window Help
ROOT (counter_self)
Complex counter settings

Complex Counter Mode: events Value: 1 Trigger Mode: normal

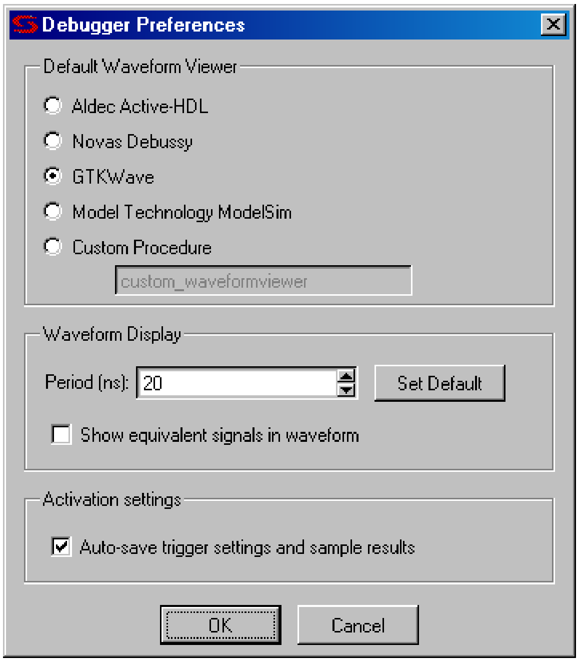
Complex Counter Mode: cycles Value: 4 Sample Mode: normal Cross trigger mode: disab

Cycle: 0

```
        current_state4'b0001 = s_RESET; /*   4'b0000 *
    else begin
            case (current_state4'b0001)
     s_RESET:       current_state4'b0001 = s_ONE;
     s_ONE:         current_state4'b0001 = s_TWO;
     s_TWO:         current_state4'b0001 = s_THREE;
     s_THREE:       current_state4'b0001 = s_FOUR;
     s_FOUR:        current_state4'b0001 = s_FIVE;
     s_FIVE:        current_state4'b0001 = s_SIX;
     s_SIX:         current_state4'b0001 = s_SEVEN;
     s_SEVEN:       current_state4'b0001 = s_EIGHT;
```

Generating Waveforms

The data captured from the design can be displayed as waveform data by using the waveform display capabilities of the GTKWave freeware waveform viewer. Viewer setup is controlled by the Waveform Viewer Preferences dialog box. Selecting Options->Waveform preference from the menu bar brings up the dialog box shown below.



Default Waveform Viewer

- Aldec Active-HDL
- Novas Debussy
- [✓ GTKWave]
- Model Technology ModelSim
- Custom Procedure: custom_waveformviewer

Waveform Display

Period (ns): 20

☐ Show equivalent signals in waveform

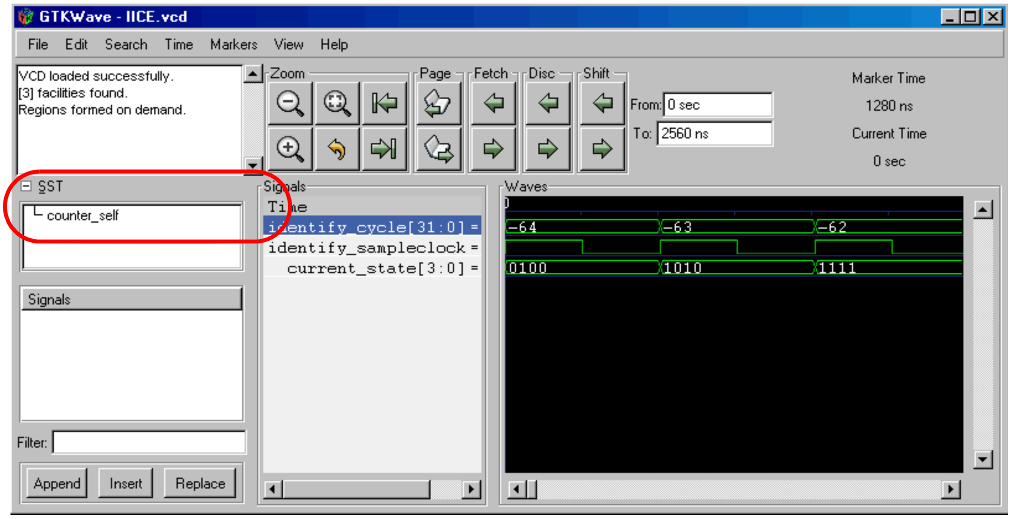
Activation settings

☑ Auto-save trigger settings and sample results

The GTKWave viewer is selected by default. The Display Period sets the period for the waveform display and is independent of the design speed.

After running the Identify debugger, the waveform viewer is displayed by selecting Window->Waveform from the menu or by clicking the Open Waveform Display icon in the menu bar.

All sampled signals in the design are included in the waveform display. Two additional signals are automatically added to the top of the display. The first signal, identify_cycle, reflects the trigger location in the sample buffer. The second signal, identify_sampleclock, is a reference that shows every clock edge. The following figure shows a typical waveform view with the identify_cycle and identify_sampleclock highlighted.


GTKWave - IICE.vcd
File Edit Search Time Markers View Help
VCD loaded successfully.
[3] facilities found.
Regions formed on demand.
Zoom Page Fetch Disc Shift
From: 0 sec
To: 2560 ns
Marker Time
1280 ns
Current Time
0 sec
SST
counter_self
Signals
Filter:
Append Insert Replace
Signals
Time
identify_cycle[31:0] =
identify_sampleclock =
current_state[3:0] =
Waves
-64 -63 -62
0100 1010 1111