ChipScope Pro Software and Cores User Guide

(ChipScope Pro Software v6.3i)

UG029 (v6.3.1) October 4, 2004

XILINX®

ChipScope Pro Software and Cores User Guide
UG029 (v6.3.1) October 4, 2004

The following table shows the revision history for this document.

	Version	Revision
04/09/02	1.0	Initial Xilinx release.

10/29/02	5.1	Added new Chapter 3 "Using the ChipScope Pro Core Inserter";
		Old Chapter 3 is new Chapter 4 "Using the ChipScope Pro Analyzer";
		Updated all chapters to be compatible with 5.1i tools;
		Revised version number to be in sync with version of tools.

03/06/03	5.2	Updated all chapters to be compatible with 5.2i tools; Updated version number to reflect version number of tools.

05/15/03	5.2.2	Chapter 1: Added the "Choice of Match Unit Counter" section to Table 1-3;
		Chapter 2: Added the "Selecting Match Unit Counter Width" section; updated several "trigger" screen shots;
		Chapter 3: Added "Selecting Match Unit Counter Width" section;
		Chapter 4: Updated screen shots in the "Configuring the Target Device(s)" section;
		Added "Displaying Configuration Status Information" section;
		Updated "Counter" section;
		Added notes to the "Depth" and "Samples per Trigger" sections;
		Updated "The VIO Console Window" section and most of its screen shots.

8/29/03	6.1	Updated all chapters to be compatible with 6.1i tools; Updated version number to reflect version number of tools; Added Chapter 5 "Tcl/JTAG Interface"
02/13/04	6.2	Updated all chapters to be compatible with 6.2i tools; Updated version number to reflect version number of tools; Chapter 2: Added the "Generating the ATC2 Core" section; Updated all chapters to reflect ATC2 compatibility; Miscellaneous edits for clarity or continuity.
06/30/04	6.3	Updated all chapters to be compatible with 6.3i tools; Updated version number to reflect version number of tools; Miscellaneous edits for clarity or continuity; Added MultiPRO cable information.
10/04/04	6.3.1	Minor text corrections.

Table of Contents

Chapter 2: Using the ChipScope Pro Core Generator

Schedule of Figures

Schedule of Tables

About This User Guide

This document provides users with information for using the ChipScope™ Pro tools:

- The ChipScope Pro Core Generator
- The ChipScope Pro Core Inserter
- The ChipScope Pro Analyzer

These tools integrate key logic analyzer hardware components with the target design inside Xilinx Virtex™, Virtex-E, Virtex-II, Virtex-II Pro™, Virtex-4™, Spartan™-II, Spartan-IIE and Spartan-3 devices (including the QPro™ variants of these families). The ChipScope Pro tools communicate with these components and provide the designer with a complete logic analyzer.

User Guide Contents

This user guide contains the following chapters:

- Chapter 1, "Introduction," describes the ChipScope Pro tools. These tools integrate key logic analyzer hardware components with the target design inside Xilinx Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE and Spartan-3 devices (including the QPro variants of these families). The ChipScope Pro tools communicate with these components and provide the designer with a complete logic analyzer.
- Chapter 2, "Using the ChipScope Pro Core Generator," explains how to use this graphical interface to generate the ChipScope Pro cores:
  - Integrated Controller core (ICON)
  - Integrated Logic Analyzer core (ILA)
  - Integrated Logic Analyzer with Agilent Trace Core (ILA/ATC)
  - Integrated Bus Analyzer for CoreConnect On-Chip Peripheral Bus core (IBA/OPB)
  - Integrated Bus Analyzer for CoreConnect Processor Local Bus core (IBA/PLB)
  - Virtual Input/Output core (VIO)
  - Agilent Trace Core 2 (ATC2)

  As a group, these cores are called the ChipScope Pro cores. After generating the ChipScope Pro cores, you can use the instantiation templates (that are provided) to quickly and easily insert the cores into VHDL or Verilog designs. After completing the instantiation and running synthesis, you can implement the design using the Xilinx ISE 6.3i implementation tools.

- Chapter 3, "Using the ChipScope Pro Core Inserter," explains how to use this post-synthesis tool to generate a netlist that includes the user design as well as ICON, ILA ILA/ATC, and ATC2 cores as needed, parameterized accordingly. The Core Inserter gives you the flexibility to quickly and easily use the ChipScope Pro debug functionality to analyze an already synthesized design, and without any HDL instantiation.
- Chapter 4, "Using the ChipScope Pro Analyzer," explains how to use this tool which interfaces directly to the ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB VIO, and ATC2 cores (collectively called the ChipScope Pro cores). You can configure your device, choose triggers, setup the console, and view the results of the capture on the fly. The data views and triggers can be manipulated in many ways, providing an easy and intuitive interface to determine the functionality of the design.
- Chapter 5, "Tcl/JTAG Interface," explains how to use this JTAG scripting interface which provides Tcl scripting access to the ChipScope Parallel cable JTAG communication library. The purpose of Tcl/JTAG is to provide a simple scripting system to access basic JTAG functions. In a few lines of Tcl script, you should be able to scan and manipulate the JTAG chain through standard Xilinx cables.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this website. You can also directly access these resources using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://support.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/apps/appsweb.htm
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://support.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://www.support.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *`design_name`*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[`*`option_name`*`]` *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr`** **`={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr`** **`={on|off}`**

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis . . .	Repetitive material that has been omitted	**`allow block`** *`block_name loc1 loc2 ... locn;`*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current file or in another file in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Handbook*.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Introduction

ChipScope Pro Tools Overview

As the density of FPGA devices increases, so does the impracticality of attaching test equipment probes to these devices under test. The ChipScope™ Pro tools integrate key logic analyzer hardware components with the target design inside Xilinx Virtex™, Virtex-E Virtex-II, Virtex-II Pro™, Virtex-4™, Spartan™-II, Spartan-IIE and Spartan-3 devices (including the QPro™ variants of these families). The ChipScope Pro tools communicate with these components and provide the designer with a complete logic analyzer.

ChipScope Pro Tools Description

Table 1-1: **ChipScope Pro Tools Description**

Tool	Description
ChipScope Pro Core Generator	Provides netlists and instantiation templates for the: • Integrated Controller Pro (ICON) core • Integrated Logic Analyzer Pro (ILA) cores • Agilent Trace Core (ILA/ATC) • Integrated Bus Analyzer for the IBM CoreConnect On-Chip Peripheral Bus (IBA/OPB) core • Integrated Bus Analyzer for CoreConnect Processor Local Bus (IBA/PLB) core • Virtual Input Output (VIO) core • Agilent Trace Core 2 (ATC2).
ChipScope Pro Core Inserter	Automatically inserts the ICON, ILA, ILA/ATC, and ATC2 cores into the user's synthesized design.
ChipScope Pro Analyzer	Provides device configuration, trigger setup, and trace display for the ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 cores. The various cores provide the trigger, control, and trace capture capability. The ICON core communicates to the dedicated Boundary Scan pins.
Tcl/JTAG Scripting	The Tcl/JTAG scriptable command interface makes it possible to interact with devices in a JTAG chain from a Tcl shell[a].

a. Tcl stands for Tool Command Language and a Tcl shell is a shell program that is used to run Tcl scripts. Tcl/JTAG requires the Tcl shell that is included in the Xilinx ISE 6.3i tool installation (`$XILINX/bin/nt/xtclsh.exe`).

The ChipScope Pro Analyzer tool supports the following download cables for communication between the PC and the devices in the JTAG Boundary Scan chain:

- Agilent E5904B Option 500, FPGA Trace Port Analyzer (Agilent E5904B TPA)
- MultiLINX™ (JTAG mode only)
- MultiPRO (JTAG mode only)
- Parallel Cable III
- Parallel Cable IV

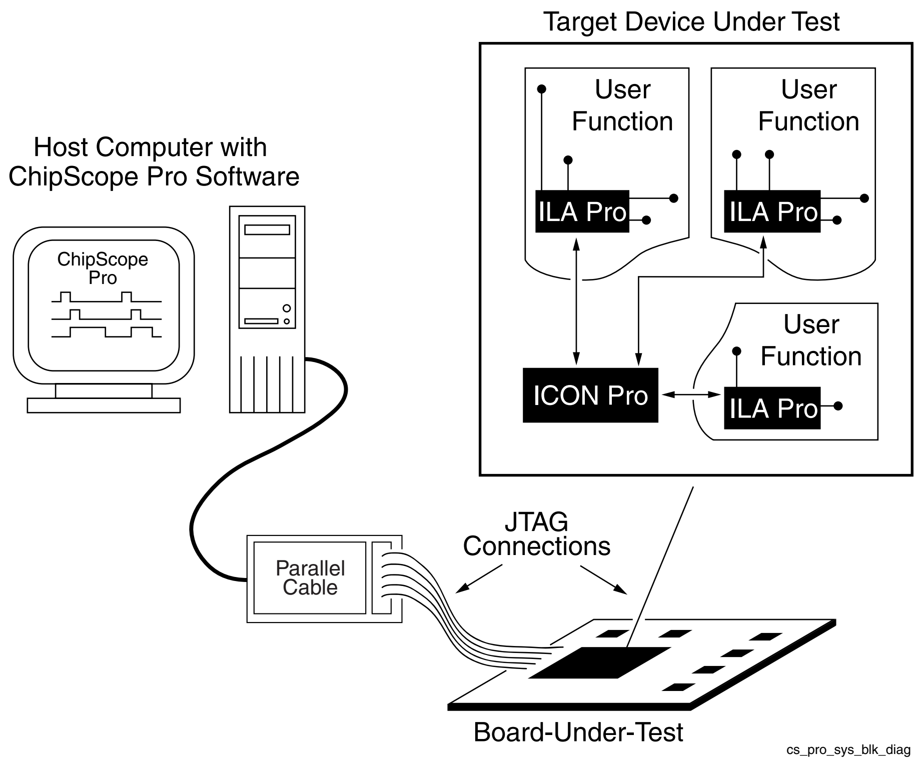


Figure 1-1: **ChipScope Pro System Block Diagram**

Figure 1-1 shows a block diagram of a ChipScope Pro system. Users can place the ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 cores (collectively called the ChipScope Pro cores) into their design by generating the cores with the ChipScope Pro Core Generator and instantiating them into the HDL source code. You can also insert the ICON, ILA, ILA/ATC, and ATC2 cores directly into the synthesized design netlist using the ChipScope Pro Core Inserter tool. The design is then placed and routed using the Xilinx ISE 6.3i implementation tools. Next, the user downloads the bitstream into the device under test and analyzes the design with the ChipScope Pro Analyzer software.

The ChipScope Pro Analyzer and ChipScope Pro cores contain many features that Xilinx FPGA designers need for thoroughly verifying their logic (Table 1-2). User-selectable data channels range from 1 to 256 and the sample buffer sizes range from 256 to 2 million(1) samples. Users can change the triggers in real time without affecting their logic. The

1. Available when using the ILA with Agilent Trace Core (ILA/ATC) in conjunction with the Agilent E5904B Option 500 trace port analyzer.

ChipScope Pro Analyzer leads designers through the process of modifying triggers and analyzing the captured data.

Table 1-2: **ChipScope Pro Features and Benefits**

Feature	Benefit
1 to 256 user-selectable data channels	Accurately captures wide data bus functionality.
User-selectable sample buffers ranging in size from 256 to 2 million samples	Large sample size increases accuracy and probability of capturing infrequent events.
Up to 16 separate trigger ports, each with a user-selectable width of 1 to 256 channels (for a total of up to 4096 trigger channels)	Multiple separate trigger ports increase the flexibility of event detection and reduce the need for sample storage.
Up to 16 separate match units per trigger port (up to 16 total match units) for a total of 16 different comparisons per trigger condition	Multiple match units per trigger ports increase the flexibility of event detection while conserving valuable resources.
All data and trigger operations are synchronous to the user clock at rates over 200 MHz	Capable of high-speed trigger event detection and data capture.
Trigger conditions implement either a boolean equation or a trigger sequence of up to 16 match functions	Can combine up to 16 trigger port match functions using a boolean equation or a 16-level trigger sequencer.
Data storage qualification condition implements a boolean equation of up to 16 match functions	Can combine up to 16 trigger port match functions using a boolean equation to determine which data samples will be captured and stored in on-chip memory.
Trigger and storage qualification conditions are in-system changeable without affecting the user logic	No need to single step or stop a design for logic analysis.
Easy-to-use graphical interface	Guides users through selecting the correct options.
Up to 15 independent ILA, ILA/ATC, IBA/OPB, IBA/PLB or VIO cores per device	Can segment logic and test smaller sections of a large design for greater accuracy.
Multiple trigger settings	Records duration and number of events along with matches and ranges for greater accuracy and flexibility.
Downloadable from the Xilinx Web site	Tools are easily accessible from the ChipScope Suite.

Design Flow

The ChipScope Pro Tools design flow (Figure 1-2) merges easily with any standard FPGA design flow that uses a standard HDL synthesis tool and the Xilinx ISE 6.3i implementation tools.

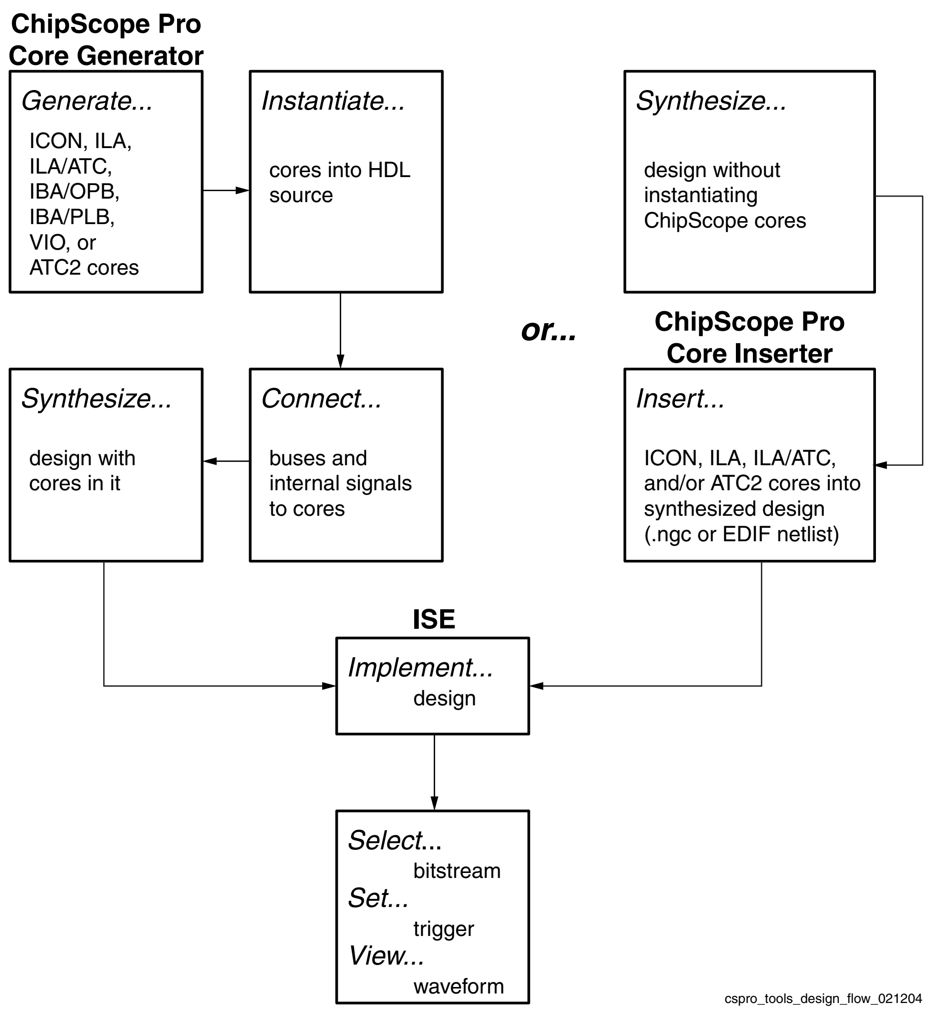

ChipScope Pro Core Generator
Generate...
ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 cores
Instantiate...
cores into HDL source
Synthesize...
design without instantiating ChipScope cores
or...
ChipScope Pro Core Inserter
Synthesize...
design with cores in it
Connect...
buses and internal signals to cores
Insert...
ICON, ILA, ILA/ATC, and/or ATC2 cores into synthesized design (.ngc or EDIF netlist)
ISE
Implement...
design
Select...
bitstream
Set...
trigger
View...
waveform
cspro_tools_design_flow_021204

Figure 1-2: **ChipScope Pro Tools Design Flow**

ChipScope Pro Cores Description

ICON Core

All of the ChipScope Pro cores use the JTAG Boundary Scan port to communicate to the host computer via a JTAG download cable. The ICON core provides a communications path between the JTAG Boundary Scan port of the target FPGA and up to 15 ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and/or ATC2 cores (as shown in Figure 1-1, page 1-2). The ICON core uses either the USER1 or USER2 JTAG Boundary Scan instructions for communication via the BSCAN_VIRTEX primitive. The unused USER1 or USER2 scan chain of the BSCAN_VIRTEX primitive can also be exported for use in your application, if needed.

ILA Core

The ILA core is a customizable logic analyzer core that can be used to monitor any internal signal of your design. Since the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components inside the ILA core. The ILA core consists of three major components:

- *Trigger input and output logic:*
  - Trigger *input* logic detects elaborate trigger events
  - Trigger *output* logic triggers external test equipment and other logic
- *Data capture logic:*
  - ILA cores capture and store trace data information using on-chip block RAM resources
  - ILA/ATC cores capture on-chip data and store trace data information in the Agilent E5904B Trace Port Analyzer
- *Control and status logic*:
  - Manages the operation of the ILA core

ILA Trigger Input Logic

The triggering capabilities of the ILA core and the ILA/ATC core include many features that are necessary for detecting elaborate trigger events. These features are described in Table 1-3, which spans multiple pages.

Table 1-3: **Trigger Features of the ILA and ILA/ATC Cores**

Feature	Description
Wide Trigger Ports	Each trigger port can be 1 to 256 bits wide.
Multiple Trigger Ports	Each ILA core and ILA/ATC core can have up to 16 trigger ports. The ability to support multiple trigger ports is necessary in complex systems where different types of signals or buses need to be monitored using separate match units.
Multiple Match Units per Trigger Port	Each trigger port can be connected to up to 16 match units. This feature enables multiple comparisons to be performed on the trigger port signals.
Boolean equation trigger condition	The trigger condition can consist of a Boolean AND or OR equation of up to 16 match unit functions.
Multi-level trigger sequencer	The trigger condition can consist of a multi-level trigger sequencer of up to 16 match unit functions.
Boolean equation storage qualification condition	The storage qualification condition can consist of a Boolean AND or OR equation of up to 16 match unit functions. ***Note:*** The storage qualification condition is not available on the ILA/ATC core.

Table 1-3: **Trigger Features of the ILA and ILA/ATC Cores *(Continued)***

Feature	Description
Choice of Match Unit Types	The match unit connected to each trigger port can be one of the following types: • **Basic comparator**: ♦ Performs '=' and '<>' comparisons. ♦ Compares up to 8 bits per slice. • **Basic comparator w/edges**: ♦ Performs '=' and '<>' comparisons. ♦ Detects high-to-low and low-to-high bit-wise transitions. ♦ Compares up to 4 bits per slice. • **Extended comparator**: ♦ Performs '=', '<>', '>', '>=', '<', and '<=' comparisons. ♦ Compares up to 2 bits per slice. • **Extended comparator w/edges**: ♦ Performs '=', '<>', '>', '>=', '<', and '<=' comparisons. ♦ Detects high-to-low and low-to-high bit-wise transitions. ♦ Compares up to 2 bits per slice. • **Range comparator**: ♦ Performs '=', '<>', '>', '>=', '<', '<=', 'in range', and 'not in range' comparisons. ♦ Compares 1 bit per slice. • **Range comparator w/edges**: ♦ Performs '=', '<>', '>', '>=', '<', '<=', 'in range', and 'not in range' comparisons. ♦ Detects high-to-low and low-to-high bit-wise transitions. ♦ Compares 1 bit per slice. All match units connected to a given trigger port are the same type.
Choice of Match Function Event Counter	All the match units of a trigger port can be configured with an event counter, with a selectable size of 1 to 32 bits. This counter can be configured at run time to count events in the following ways: • Exactly *n* occurrences ♦ Matches only when exactly n consecutive or non-consecutive events occur • At least *n* occurrences ♦ Matches and stays asserted once *n* consecutive or non-consecutive events occur • At least *n* consecutive occurrences ♦ Matches once *n* consecutive events occur, and stays asserted until the match function is not satisfied.

Table 1-3: **Trigger Features of the ILA and ILA/ATC Cores *(Continued)***

Feature	Description
Trigger Output Port	The internal trigger condition of the ILA core can be accessed using the optional trigger output port. This signal can be used as a trigger for external test equipment by attaching the signal to an output pin. However, it can also be used by internal logic as an interrupt, a trigger, or to cascade multiple ILA cores together. The trigger output port will have a determined amount of latency depending on the core type: • ILA core = 10 clock cycles • ILA/ATC core = 10 clock cycles • IBA/OPB core = 15 clock cycles • IBA/PLB core = 10 clock cycles The shape (level or pulse) and sense (active high or low) of the trigger output can be controlled at run-time.

Using Multiple Trigger Ports

The ability to monitor different kinds of signals and buses in the design requires the use of multiple trigger ports. For example, if you are instrumenting an internal system bus in your design that is made up of control, address, and data signals, then you could assign a separate trigger port to monitor each signal group (as shown in Figure 1-3).

If you connected all of these different signals and buses to a single trigger port, you would not be able to monitor for individual bit transitions on the **CE**, **WE**, and **OE** signals while looking for the **Address** bus to be in a specified range. The flexibility of being able to choose from different types of match units allows you to customize the ILA cores to your triggering needs while keeping resource usage to a minimum.

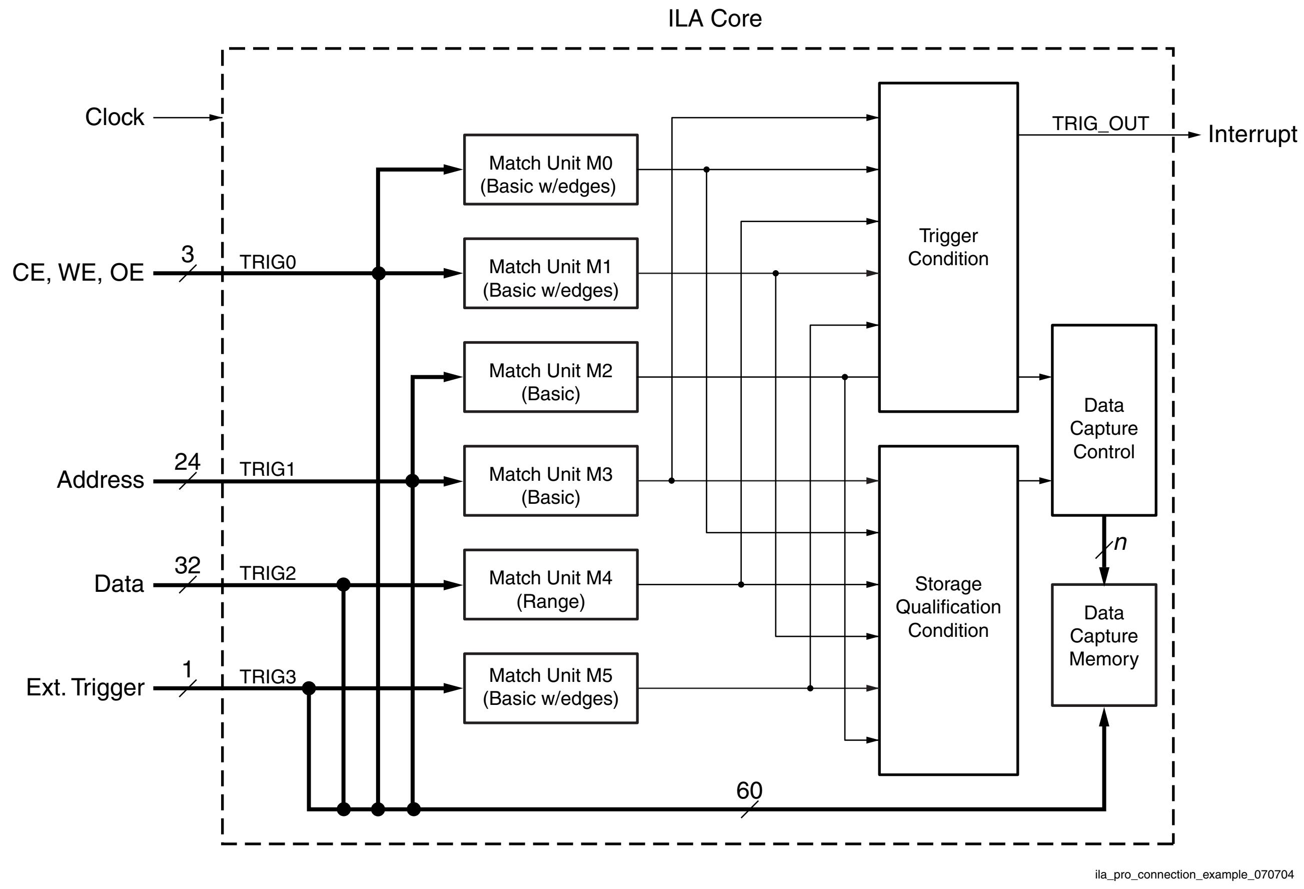


Figure 1-3: **LA Core Connection Example**

Using Trigger and Storage Qualification Conditions

The ILA, IBA/OPB and IBA/PLB cores implement both trigger and storage qualification condition logic. The ILA/ATC core only implements the trigger condition logic. The trigger condition is a Boolean or sequential combination of events that is detected by match unit comparators that are attached to the trigger ports of the core. The trigger condition is used to mark a distinct point of origin in the data capture window and can be located at the beginning, the end, or anywhere within the data capture window.

Similarly, the storage qualification condition is also a Boolean combination of events that is detected by match unit comparators that are subsequently attached to the trigger ports of the core. However, the storage qualification condition differs from the trigger condition in that it evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start the capture process and what data is captured.

In the ILA core example shown in Figure 1-3, suppose you want to do the following:

- Trigger on the first memory write cycle (**CE** = rising edge, **WE** = 1, **OE** = 0) to **Address** = 0xFF0000;
- Capture only memory read cycles (**CE** = rising edge, **WE** = 0, **OE** = 1) from **Address** = 0x23AACC where the **Data** values are between 0x00000000 and 0x1000FFFF;

To implement these conditions successfully, you would need to make sure that both the **TRIG0** and **TRIG1** trigger ports each have two match units attached to them: one for the trigger condition and one for the storage qualification condition. Here is how you would set up the trigger and storage qualification equations and each individual match unit to satisfy the conditions above:

- Trigger Condition = M0 && M2, where:
  - M0[2:0] = **CE**, **WE**, **OE** = "R10" (where 'R' means "rising edge")
  - M2[23:0] = **Address** = "FF0000"
- Storage Qualification Condition = M1 && M3 && M4, where:
  - M1[2:0] = **CE**, **WE**, **OE** = "R10" (where 'R' means "rising edge")
  - M3[23:0] = **Address** = "23AACC"
  - M4[31:0] = **Data** = in the range of 0x00000000 through 0x1000FFFF

The triggering and storage qualification capabilities of the ILA, IBA/OPB and IBA/PLB cores allow you to locate and capture exactly the information that you want without wasting valuable on-chip memory resources.

ILA Trigger Output Logic

The ILA core implements a trigger output port called TRIG_OUT. The TRIG_OUT port is the output of the trigger condition that is set up at run-time using the ChipScope Pro Analyzer. The shape (level or pulse) and sense (active high or low) of the trigger output can also be controlled at run-time. The latency of the TRIG_OUT port relative to the input trigger ports is 10 clock cycles.

The TRIG_OUT port is very flexible and has many uses. You can connect the TRIG_OUT port to a device pin in order to trigger external test equipment such as oscilloscopes and logic analyzers. Connecting the TRIG_OUT port to an interrupt line of an embedded PowerPC™ 405 or MicroBlaze™ processor can be used to cause a software event to occur. You can also connect the TRIG_OUT port of one core to a trigger input port of another core in order to expand the trigger and data capture capabilities of your on-chip debug solution.

ILA Data Capture Logic

Each ILA core can capture data using on-chip block RAM resources independently from all other cores in the design. Each ILA core can also capture data using one of two capture modes: *Window* and *N sample*s.

Window Capture Mode

In window capture mode, the sample buffer can be divided into one or more equal-sized sample windows. The window capture mode uses a single trigger condition event (i.e., a Boolean combination of the individual trigger match unit events) to collect enough data to fill a sample window.

In the case where the depth of the sample windows is a power of 2 up to 16384 samples, the trigger position can be set to the beginning of the sample window (trigger first, then collect), the end of the sample window (collect until the trigger event), or anywhere in between.

In the other case where the window depth is *not* a power of 2, the trigger position can only be set to the beginning of the sample window.

Once a sample window has been filled, the trigger condition of the ILA core is automatically re-armed and continues to monitor for trigger condition events. This process is repeated until all sample windows of the sample buffer are filled or the user halts the ILA core.

N Samples Capture Mode

The N Samples capture mode is similar to the Window capture mode except for two major differences:

- The number of samples per window can be any integer N from 1 to the sample buffer size minus 1
- The trigger position must always be at position 0 in the window

The N sample capture mode is useful for capturing the exact number of samples needed per trigger without wasting valuable capture storage resources.

Trigger Marks

The data sample in the sample window that coincides with a trigger event is tagged with a trigger mark. This trigger mark tells the ChipScope Pro Analyzer the position of the trigger within the window. This trigger mark consumes one extra bit per sample in the sample buffer.

Data Port

The ILA core provides the capability to capture data on a port that is separate from the trigger ports that are used to perform trigger functions. This feature is useful for limiting the amount of data to be captured to a relatively small amount since it is not always useful to capture and view the same information that is used to trigger the core.

However, in many cases it is useful to capture and view the same data that is used to trigger the core. In this case, you can choose for the data to consist of one or more of the trigger ports. This feature allows you to conserve resources while providing the flexibility to choose what trigger information is interesting enough to capture.

ILA Control and Status Logic

The ILA contains a modest amount of control and status logic that is used to maintain the normal operation of the core. All logic necessary to properly identify and communicate with the ILA core is implemented by this control and status logic.

ILA/ATC Core

The ILA/ATC core is a customizable logic analyzer core that is very similar to the ILA core, with the exception that it does not use on-chip block RAM resources to store captured trace data. Instead, the ILA/ATC core stores captured trace data in the two million sample deep trace buffer in the Agilent E5904B TPA. The ILA/ATC core consists of three major components :

- *Trigger input and output logic*:
  - Trigger *input* logic detects elaborate trigger events
  - Trigger *output* logic triggers external test equipment and other logic
- *Data capture logic:*
  - Captures and stores trace data information in the Agilent E5904B Trace Port Analyzer via a 38-pin MICTOR connector
- *Control and status logic:*
  - Manages the operation of the ILA/ATC core

ILA/ATC Trigger Input Logic

The trigger input capabilities of the ILA/ATC core are very similar to those of the ILA core. These features are described in ILA Trigger Input Logic, page 1-5.

Note: The ILA/ATC core does not have the storage qualification condition capability that the ILA, IBA/OPB, and IBA/PLB cores have.

ILA/ATC Trigger Output Logic

The trigger output capabilities of the ILA/ATC core are identical to those of the ILA core. These features are described in ILA Trigger Output Logic, page 1-10.

ILA/ATC Data Capture Logic

Each ILA/ATC core can capture data independently from all other cores in the design as long as each ILA/ATC core uses its own special data connector and the Agilent E5904B TPA. Currently, the ILA/ATC core only supports the single window capture mode.

Single Window Capture Mode

In single window capture mode, the entire sample buffer is viewed as a single sample window. A single trigger condition event (i.e., a Boolean combination of the individual trigger match unit events) is used to collect enough data to fill the entire sample buffer. The depth of the sample windows can be up to two million samples.

The trigger position can be set to the beginning of the sample window (trigger first, then collect), or at the end of the sample window (collect until the trigger event), or anywhere in between.

Trigger Marks

The data sample in the sample window that coincides with a trigger event is tagged with a trigger mark. This trigger mark tells the ChipScope Pro Analyzer the position of the trigger within the window. This trigger mark consumes one extra bit per sample in the sample buffer.

Timestamps

The Agilent TPA JTAG cable can record timestamps along with each of the samples captured by the ILA/ATC core. These timestamp values are measured from the first sample of the buffer and are displayed in nanoseconds in the various data views within the ChipScope Pro Analyzer. When timestamps are enabled, the maximum number of data samples is limited to the values shown in Table 1-5, page 1-13.

Data Port

The data port of the ILA/ATC core is used to capture data and transmit it to the Agilent TPA using external pins. The ILA/ATC data port are always separate from the trigger input ports. The ILA/ATC core uses 4, 8, 12, 16, or 20 external pins to transmit the data. A transmit clock pin is also used to synchronize the Agilent TPA with the ILA/ATC core.

The 2x and 4x transmit rates also use internal global clock buffer and clock management resources in the FPGA device. Table 1-4 shows how the transmit rate affects the usage of these internal device resources.

Table 1-4: **ILA/ATC Clock Resource Utilization**

Transmit Rate	Virtex, Virtex-E, Spartan-II and Spartan-IIE		Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3	
	Number of BUFGs	Number of CLKDLLs	Number of BUFGs	Number of DCMs
1x	0	0	0	0
2x	1	1	1	1
4x	1	2	2	1

The data is transmitted at 1, 2, or 4 times the rate of the CLK port of the ILA/ATC core, where the maximum CLK port rate is 200, 100, and 50 MHz, respectively. Table 1-5 shows the possible data rate and pin combinations and how they affect the width and depth of the DATA port.

Table 1-5: **ILA/ATC Core Capabilities**

Number of Data Pins	Transmit Rate	Max Width of DATA Port	Max Number of Samples (with timestamps)	Max CLK Port Frequency[a]
4	1x	3	2,097,120 (1,048,560)	200 MHz
4	2x	5	1,048,560 (1048560)	100 MHz
4	4x	11	524,280 (262,136)	50 MHz
8	1x	7	2,097,120 (1,048,560)	200 MHz
8	2x	13	1,048,560 (524,280)	100 MHz
8	4x	27	524,280 (262,136)	50 MHz
12	1x	11	2,097,120 (1,048,560)	200 MHz
12	2x	21	1,048,560 (524,280)	100 MHz
12	4x	43	524,280 (262,136)	50 MHz
16	1x	15	2,097,120 (1,048,560)	200 MHz
16	2x	29	1,048,560 (524,280)	100 MHz
16	4x	59	524,280 (262,136)	50 MHz
20	1x	19	2,097,120 (1,048,560)	200 MHz
20	2x	37	1,048,560 (524,280)	100 MHz
20	4x	75	524,280 (262,136)	50 MHz

a. The maximum clock pin frequency is always 200 MHz, but may differ from the CLK port frequency depending on the transmit rate.

The external data and clock pins of the ILA/ATC core are connected to the Agilent TPA using a special 38-pin MICTOR connector. Both the Agilent TPA and the special connector are described in the companion Agilent document called *Deep Storage with Xilinx ChipScope Pro and Agilent Technologies FPGA Trace Port Analyzer* which is available for download at http://www.xilinx.com/ise/verification/cspro_agilent_brochure.pdf.

ILA/ATC Control and Status Logic

The ILA contains a modest amount of control and status logic that is used to maintain the normal operation of the core. All logic necessary to properly identify and communicate with the ILA core is implemented by this control and status logic.

IBA/OPB Core

The IBA/OPB core is a specialized logic analyzer core specifically designed to debug embedded systems that contain the IBM CoreConnect On-Chip Peripheral Bus (OPB). The IBA/OPB core consists of four major components:

- A *protocol violation monitor:*
    - Detects and reports up to 32 violations of the IBM CoreConnect OPB bus protocol
- *Trigger input and output logic:*
    - *Trigger input logic* detects OPB bus and other user-defined events
    - *Trigger output logic* triggers external test equipment and other logic
- *Data capture logic:*
    - Captures and stores trace data information using on-chip block RAM resources
- *Control and status logic:*
    - Manages the operation of the IBA/OPB core

IBA/OPB Protocol Violation Monitor Logic

The IBA/OPB core includes a protocol violation monitor that can detect up to 32 different IBM CoreConnect OPB protocol violation errors. The protocol violations that can be detected by the IBA/OPB core are shown in Table 1-6, which spans multiple pages.

Table 1-6: **CoreConnect OPB Protocol Violation Error Description**[a]

Priority	Bit Encoding	Error	Description
1	011010	1.19.2	OPB_DBus changed state during a write operation before receipt of OPB_xferAck.
2	011001	1.19.1	OPB_ABus changed state during an operation before receipt of OPB_xferAck.
3	001100	1.6.1	OPB_ABus: No Mx_Select signal active and non zero OPB_ABus.
4	001101	1.7.1	OPB_DBus: No Mx_Select signal active and non zero OPB_DBus.
5	010101	1.13.1	OPB_xferAck: OPB_xferAck active with no Mx_select.
6	010110	1.13.2	OPB_xferAck: OPB_xferAck did not activate within 16 cycles of OPB_select.
7	010111	1.15.1	OPB_errAck: OPB_errAck active with no Mx_select.
8	000100	1.4.0	OPB_retry: OPB_retry and OPB_xferAck active in the same cycle.
9	000111	1.4.3	OPB_retry: OPB_retry active for more than a single cycle.
10	000000	1.2.1	OPB_MxGrant: More than 1 OPB_MxGrant signals active in same cycle.

Table 1-6: **CoreConnect OPB Protocol Violation Error Description[a]** ***(Continued)***

Priority	Bit Encoding	Error	Description
11	000001	1.2.2	OPB_MxGrant: An OPB_MxGrant signal is active for a non-owning master.
12	000010	1.3.1	OPB_BusLock: OPB_BusLock asserted without a grant in the previous cycle and without OPB_select.
13	000011	1.3.2	OPB_BusLock: Bus is locked and a master other than bus owner has been granted the bus.
14	001000	1.4.4	OPB_retry: OPB_select remained active after a retry cycle.
15	001001	1.4.5	OPB_retry: OPB_retry active with no Mx_select.
16	001110	1.8.1	OPB_Select: Mx_Select signal active without having control of the bus via OPB_MxGrant or OPB_busLock.
17	001111	1.8.2	OPB_Select: More than 1 Mx_Select signals active in the same cycle.
18	010000	1.9.1	OPB_RNW: OPB_RNW high with no Mx_select.
19	011011	1.19.3	OPB_RNW changed state during an operation before receipt of OPB_xferAck.
20	011100	1.19.4	OPB_select changed state during an operation before receipt of OPB_xferAck.
21	011101	1.19.5	OPB_BEBus changed state during a write or read operation before receipt of OPB_xferAck.
22	011110	1.20.3	Byte enable transfer not aligned with address offset.
23	011111	1.20.4	Byte enable transfer initiated with non contiguous byte enables.
24	000110	1.4.2	OPB_retry: Mx_Request from retried master remained active after a retry cycle.
25	000101	1.4.1	OPB_retry: OPB_BusLock remained active after a retry cycle.
26	010001	1.11.1	OPB_seqAddr: OPB_seqAddr active with no OPB_BusLock.
27	010010	1.11.2	OPB_seqAddr: OPB_seqAddr active with no Mx_select.
28	010011	1.11.3	OPB_seqAddr: OPB_ABUS did not increment properly during OPB_seqAddr.
29	010100	1.11.4	OPB_seqAddr: OPB_seqAddr was asserted without a transaction boundary.

Table 1-6: **CoreConnect OPB Protocol Violation Error Description[a] *(Continued)***

Priority	Bit Encoding	Error	Description
30	011000	1.16.1	OPB_ToutSup: OPB_ToutSup active with no Mx_select.
31	001010	1.5.1	OPB_Timeout: Arbiter failed to signal OPB_Timeout after 16 non-responding cycles.
32	001011	1.5.2	OPB_Timeout: OPB_Timeout active with no Mx_select.
33	111111	N/A	No errors.

a. Refer to Chapter 8 of the *OPB Bus Functional Model Toolkit User's Manual* document from IBM for more information on these CoreConnect OPB errors.

The protocol violation monitor detects and reports any errors that occur on the OPB bus. The error is reported as a 6-bit priority-encoded value that can be used as both trigger and data to the IBA/OPB core. Priority 1 is the highest priority error and masks any other lower priority errors, etc.

IBA/OPB Trigger Input Logic

The IBA core for the IBM CoreConnect On-Chip Peripheral Bus (IBA/OPB) is used to monitor the CoreConnect OPB bus of embedded MicroBlaze soft processor or Virtex-II Pro and Virtex-4 FX PowerPC 405 hard processor systems. Up to 16 different trigger groups can be monitored by the IBA/OPB core at any given time. The OPB signal groups that can be monitored are described in Table 1-7, page 1-17, which spans multiple pages.

The IBA/OPB core can also implement the same trigger and storage qualification condition equations as the ILA core. These features are described in the section called ILA Trigger Input Logic, page 1-5.

IBA/OPB Trigger Output Logic

The IBA/OPB core implements a trigger output port called TRIG_OUT. The TRIG_OUT port is the output of the trigger condition that is set up at run-time using the ChipScope Pro Analyzer. The latency of the TRIG_OUT port relative to the input trigger ports is 15 clock cycles.

The TRIG_OUT port is very flexible and has many uses. For example, you can:

- Connect the TRIG_OUT port to a device pin in order to trigger external test equipment such as oscilloscopes and logic analyzers
- Connect the TRIG_OUT port to an interrupt line of an embedded PowerPC 405 or MicroBlaze processor to cause a software event to occur
- Connect the TRIG_OUT port of one core to a trigger input port of another core in order to expand the trigger and data capture capabilities of your on-chip debug solution

Table 1-7: **OPB Signal Groups**

Trigger Group Name	Width	Description
OPB_CTRL	17	OPB combined control signals, including: • SYS_Rst • Debug_SYS_Rst • WDT_Rst • OPB_Rst • OPB_BE[3] • OPB_BE[2] • OPB_BE[1] • OPB_BE[0] • OPB_select • OPB_xferAck • OPB_RNW • OPB_errAck • OPB_timeout • OPB_toutSup • OPB_retry • OPB_seqAddr • OPB_busLock
OPB_ABUS	32	OPB address bus
OPB_DBUS	32	OPB combined data bus (logical OR of read and write data buses)
OPB_RDDBUS	32	OPB read data bus (from slaves)
OPB_WRDBUS	32	OPB write data bus (to slaves)
OPB_M*n*_CTRL	11	OPB control signals for master *n*, including: • M*n*_request • OPB_M*n*Grant • OPB_pendReq*n* • M*n*_busLock • M*n*_BE[3] • M*n*_BE[2] • M*n*_BE[1] • M*n*_BE[0] • M*n*_select • M*n*_RNW • M*n*_seqAddr where *n* is the master number (0 to 15)

Table 1-7: **OPB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
OPB_SL*m*_CTRL	4	OPB control signals slave *m*, including: • Sl*m*_xferAck • Sl*m*_errAck • Sl*m*_toutSup • Sl*m*_retry where *m* is the slave number (0 to 63)
OPB_PV	6	OPB protocol violation signals
TRIG_IN	User-defined	Generic trigger input

The IBA/OPB core can monitor not only CoreConnect OPB bus signals, but can also monitor generic design signals (using the TRIG_IN trigger group). This capability allows the user to correlate events that are occurring on the CoreConnect OPB bus with events elsewhere in the design. The IBA/OPB core can also be connected to other ChipScope Pro capture cores using the TRIG_IN and TRIG_OUT port signals to perform cross-triggering operations while monitoring different parts of the design.

IBA/OPB Data Capture Logic

The data capture logic capabilities of the IBA/OPB core are identical to those of the ILA core. These features are described in ILA Data Capture Logic, page 1-10.

IBA/OPB Control and Status Logic

The IBA/OPB contains a modest amount of control and status logic that is used to maintain the normal operation of the core. All logic necessary to properly identify and communicate with the IBA/OPB core is implemented by this control and status logic.

IBA/PLB Core

The IBA/PLB core is a specialized logic analyzer core specifically designed to debug embedded systems that contain the IBM CoreConnect Processor Local Bus (PLB). The IBA/PLB core consists of three major parts components:

- *Trigger input and output logic:*
    - *Trigger input logic* detects PLB bus and other user-defined events
    - *Trigger output logic* triggers external test equipment and other logic
- *Data capture logic:*
    - Captures and stores trace data information using on-chip block RAM resources
- *Control and status logic:*
    - Manages the operation of the IBA/PLB core

IBA/PLB Trigger Input Logic

The IBA core for the IBM CoreConnect Processor Local Bus (IBA/PLB) is used to monitor the PLB bus of embedded Virtex-II Pro and Virtex-4 FX PowerPC 405 processor systems. Up to 16 different trigger groups can be monitored by the IBA/PLB core at any given time.

The types of PLB signal groups that can be monitored are described in Table 1-8, page 1-19, which spans multiple pages.

The IBA/PLB core can also monitor other generic design signals (using the TRIG_IN trigger group) in addition to the PLB bus signals. This capability allows the user to correlate events that are occurring on the PLB bus with events elsewhere in the design. The IBA/PLB core can also be connected to other ChipScope Pro capture cores using the TRIG_IN and TRIG_OUT port signals to perform cross-triggering operations while monitoring different parts of the design.

The IBA/PLB core is also able to implement the same trigger and storage qualification condition equations as the ILA core. These features are described in the section called ILA Trigger Input Logic, page 1-5.

Table 1-8: **PLB Signal Groups**

Trigger Group Name	Width	Description
PLB_CTRL	26	PLB bus control signals, including: • SYS_plbReset • PLB_abort • PLB_BE(0) • PLB_BE(1) • PLB_BE(2) • PLB_BE(3) • PLB_BE(4) • PLB_BE(5) • PLB_BE(6) • PLB_BE(7) • PLB_busLock • PLB_masterID(0) • PLB_masterID(1) • PLB_masterID(2) • PLB_masterID(3) • PLB_Msize(0) • PLB_Msize(1) • PLB_PAValid • PLB_SAValid • PLB_rdPrim • PLB_RNW • PLB_size(0) • PLB_size(1) • PLB_size(2) • PLB_size(3) • PLB_wrPrim
PLB_ABUS	32	PLB address bus
PLB_RDDBUS	64	PLB read data bus (from slaves)
PLB_WRDBUS	64	PLB write data bus (to slaves)

Table 1-8: **PLB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
PLB_M*n*_CTRL	32	PLB control signals for master *n*, including: • PLB_M*n*AddrAck • PLB_M*n*_Busy • PLB_M*n*_Err • PLB_M*n*RdDAck • PLB_M*n*RdWdAddr(0) • PLB_M*n*RdWdAddr(1) • PLB_M*n*RdWdAddr(2) • PLB_M*n*RdWdAddr(3) • PLB_M*n*Rearbitrate • PLB_M*n*SSize(0) • PLB_M*n*SSize(1) • PLB_M*n*_WrDAck • M*n*_abort • M*n*_BE(0) • M*n*_BE(1) • M*n*_BE(2) • M*n*_BE(3) • M*n*_BE(4) • M*n*_BE(5) • M*n*_BE(6) • M*n*_BE(7) • M*n*_busLock • M*n*_MSize(0) • M*n*_MSize(1) • M*n*_priority(0) • M*n*_priority(1) • M*n*_request • M*n*_RNW • M*n*_size(0) • M*n*_size(1) • M*n*_size(2) • M*n*_size(3) where *n* is the master number (0 to 15)

Table 1-8: **PLB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
PLB_SL*m*_CTRL	12	PLB control signals slave *m*, including: • Sl*m*_addrAck • Sl*m*_rdDAck • Sl*m*_rdWdAddr(0) • Sl*m*_rdWdAddr(1) • Sl*m*_rdWdAddr(2) • Sl*m*_rdWdAddr(3) • Sl*m*_rearbitrate • Sl*m*_SSize(0) • Sl*m*_SSize(1) • Sl*m*_wait • Sl*m*_wrComp • Sl*m*_wrDAck where *m* is the slave number (0 to 15)
TRIG_IN	User-defined	Generic trigger input

IBA/PLB Trigger Output Logic

The IBA/OPB core implements a trigger output port called TRIG_OUT. The TRIG_OUT port is the output of the trigger condition that is set up at run-time using the ChipScope Pro Analyzer. The latency of the TRIG_OUT port relative to the input trigger ports is 10 clock cycles.

The TRIG_OUT port is very flexible and has many uses. You can connect the TRIG_OUT port to a device pin in order to trigger external test equipment such as oscilloscopes and logic analyzers. Connecting the TRIG_OUT to an interrupt line of an embedded PowerPC 405 or MicroBlaze processor can be used to cause a software event to occur. You can also connect the TRIG_OUT port of one core to a trigger input port of another core in order to expand the trigger and data capture capabilities of your on-chip debug solution.

IBA/PLB Data Capture Logic

The data capture capabilities of the IBA/PLB core are identical to those of the ILA core. These features are described in ILA Data Capture Logic, page 1-10.

IBA/PLB Control and Status Logic

The IBA/PLB core contains a modest amount of control and status logic that is used to maintain the normal operation of the core. All logic necessary to properly identify and communicate with the IBA/PLB core is implemented by this control and status logic.

VIO Core

The Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. Unlike the ILA and IBA cores, no on- or off-chip RAM is required. Four kinds of signals are available in a the VIO core:

- *Asynchronous inputs:*
  - These are sampled using the JTAG clock signal that is driven from the JTAG cable.
  - The input values are read back periodically and displayed in the ChipScope Pro Analyzer.
- *Synchronous inputs:*
  - These are sampled using the design clock.
  - The input values are read back periodically and displayed in the ChipScope Pro Analyzer.
- *Asynchronous outputs*
  - These are defined by the user in the ChipScope Pro Analyzer and driven out of the core to the surrounding design.
  - A logical 1 or 0 value may be defined for individual asynchronous outputs.
- *Synchronous outputs*
  - These are *defined* by the user in the ChipScope Pro Analyzer, *synchronized* to the design clock and *driven out* of the core to the surrounding design.
  - A logical 1 or 0 may be defined for individual synchronous outputs. Pulse trains of 16 clock cycles worth of 1's and/or 0's may also be defined for synchronous outputs.

Activity Detectors

Every VIO core input has additional cells to capture the presence of transitions on the input. Since the design clock will most likely be much faster than the sample period of the Analyzer, it's possible for the signal being monitored to transition many times between successive samples. The activity detectors capture this behavior and the results are displayed along with the value in the ChipScope Pro Analyzer.

In the case of a synchronous input, activity cells capable of monitoring for asynchronous and synchronous events are used. This feature can be used to detect glitches as well as synchronous transitions on the synchronous input signal.

Pulse Trains

Every VIO synchronous output has the ability to output a static 1, a static 0, or a pulse train of successive values. A pulse train is a 16 clock cycle sequence of 1's and 0's that drive out of the core on successive design clock cycles. The pulse train sequence is defined in the ChipScope Pro Analyzer and is executed only one time after it is loaded into the core.

ATC2 Core

The Agilent Trace Core 2 (ATC2) is a customizable debug capture core that is specially designed to work with the latest generation Agilent logic analyzers. The ATC2 core provides external Agilent logic analyzers access to internal FPGA design nets (as shown in Figure 1-4).

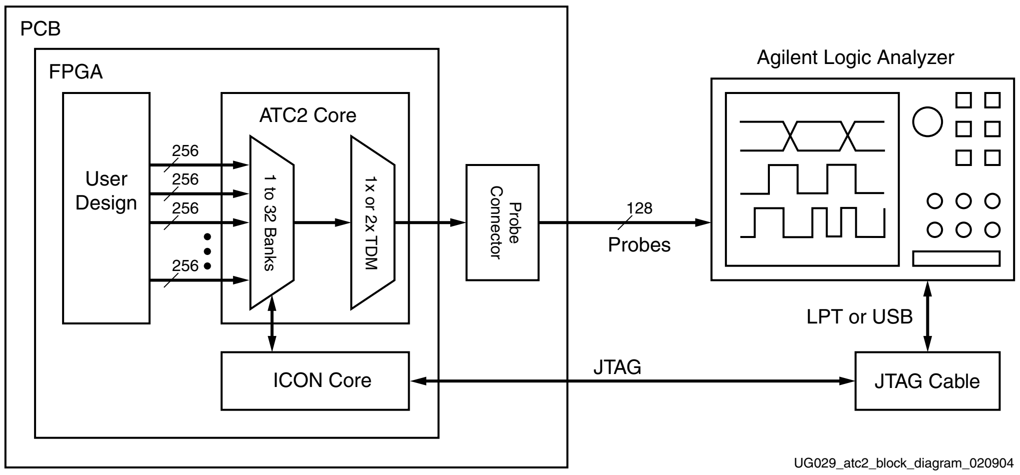


Figure 1-4: **ATC2 Core and System Block Diagram**

ATC2 Data Path Description

The data path of the ATC2 core consists of:

- Up to 32 run-time selectable input data ports that connect to the user's FPGA design
- Up to 128 output data pins that connect to an Agilent logic analyzer's probe connectors
- Optional 2x time-division multiplexing (TDM) available on each output data pin that can be used to double the width of each individual data port from 128 to 256 bits
- Supports both asynchronous timing and synchronous state capture modes
- Supports any valid I/O standard, drive strength, and output slew rate on each output data pin on an individual pin-by-pin basis
- Supports any Agilent probe connection technology (for more information, please refer to http://www.agilent.com/find/softtouch)

The maximum number of data probe points available at run time is calculated as: (32 data ports) * (128 bits per data port) * (2x TDM) = 8192 probe points.

ATC2 Core Data Capture and Run-Time Control

The external Agilent logic analyzer is used to trigger on and capture the data that passes through the ATC2 core. This allows you to take full advantage of the complex triggering, deep trace memory, and system-level data correlation features of the Agilent logic analyzer as well as the increased visibility of internal design nodes provided by the ATC2 core. The Agilent logic analyzer is also used to control the run-time selection of the active data port by communicating with the ATC2 core via a JTAG port connection (as shown in Figure 1-4).

ICON and ILA Core Resource Usage

Table 1-9 and Table 1-10 show the ICON core and ILA core resource utilization.

Table 1-9: **ICON and ILA Core CLB Usage in Virtex-II Pro Devices**

Trigger / Data Width	LUTs	Flip-Flops
8	177	171
16	186	187
32	216	221
64	252	287
128	336	425
256	501	675

This example uses a single ILA core with a single trigger port, a single basic match unit, data same as trigger, and 512 data samples.

Table 1-10: **ICON and ILA Core Block RAM Usage in Virtex-II Pro Devices**

Trigger / Data Width	Data Samples					
	512	1024	2048	4096	8192	16384
8	1	1	2	3	5	9
16	1	2	3	5	9	17
32	2	3	5	9	17	33
64	3	5	9	17	33	65
128	5	9	17	33	65	129
256	9	17	33	65	129	257

This example uses a single ILA core with a single trigger port, a single basic match unit, data same as trigger, and 512 data samples. Also note that one extra bit per sample is required for the trigger mark (e.g., a trigger/data width of 8 bits requires a sample width of 9 bits, etc.)

Synthesis Requirements

Users can modify many options in the ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 cores without resynthesizing. However, after changing selectable parameters (such as width of the data port or the depth of the sample buffer), the design must be resynthesized with new cores. Table 1-11 shows which design changes require resynthesizing.

Table 1-11: **Design Parameter Changes Requiring Resynthesis**

Design Parameter Change	Resynthesis Required
Change trigger pattern	No
Running and stopping the trigger	No
Enabling the external triggers	No
Changing the trigger signal source	No[a]
Changing the data signal source	No[a]
Changing the ILA clock signal	Yes
Changing the sample buffer depth	Yes

a. The ability to change existing trigger and/or data signal source is supported by the Xilinx ISE 6.3i FPGA Editor.

System Requirements

Software Tools Requirements

The ChipScope Pro Core Inserter requires that Xilinx ISE 6.3i implementation tools and a Tcl shell be installed on your system. (Tcl stands for Tool Command Language and a Tcl shell is a shell program that is used to run Tcl scripts.) Tcl/JTAG requires the Tcl shell that is included in the Xilinx ISE 6.3i tool installation (`$XILINX/bin/nt/xtclsh.exe`).

Communications Requirements

The ChipScope Pro Analyzer supports the following download cables (see Table 1-12, page 1-26) for communication between the PC and the devices in the JTAG Boundary Scan chain:

- Agilent E5904B TPA
- MultiLINX (JTAG mode only)
- MultiPRO (JTAG mode only)
- Parallel Cable III
- Parallel Cable IV

Table 1-12: **Download Cable Description**

Download Cable	Features
Agilent E5904B TPA	• Connects to host using 10/100 base-T ethernet • Supports remote configuration and debug • Downloads at speeds up to 30 Mb/s • Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 3.3V down to 1.5V
MultiLINX Cable	• Uses either the RS-232 (i.e., serial port) or USB 1.0 ports to communicate with the Boundary Scan chain of the board-under-test • Downloads at speeds up to 57.6 kb/s throughput when using RS-232 connection • Downloads at speeds up to 8 Mb/s throughput when using USB connection • Contains an adjustable voltage interface that enables it to communicate with systems and I/O's operating at 5V, 3.3V, or 2.5V
MultiPRO Cable	• Uses the parallel port (i.e., printer port) to communicate with the Boundary Scan chain of the board-under-test • Downloads at speeds up to 5 Mb/s throughput • Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 5V down to 1.5V
Parallel Cable III	• Uses the parallel port (i.e., printer port) to communicate with the Boundary Scan chain of the board-under-test • Downloads at speeds up to 500 kb/s throughput • Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 5V down to 2.5V
Parallel Cable IV	• Uses the parallel port (i.e., printer port) to communicate with the Boundary Scan chain of the board-under-test • Downloads at speeds up to 5 Mb/s throughput • Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 5V down to 1.5V

Note: The Parallel Cable IV cable is available for purchase from the Xilinx Online Store (from www.xilinx.com choose **Online Store → Programming Solutions → Programming Cables**).

Board Requirements

For the ChipScope Pro Analyzer and download cable to work properly with the board-under-test, the following board-level requirements must be met:

- One or more Virtex-II Series target devices must be connected to a JTAG header that contains the TDI, TMS, TCK, and TDO pins
- If another device would normally drive the TDI, TMS, or TDI pins of the JTAG chain containing the target device(s), then jumpers on these signals are required to disable these sources, preventing contention with the download cable
- If using the Parallel Cable III download cable, then V_{CC} (2.5V-5.0V) and GND headers must be available for powering the Parallel Cable III cable
- If using the Parallel Cable IV download cable, then VREF (1.5-5.0V) and GND headers must be available for connecting to the Parallel Cable IV cable
- If using the Agilent TPA download cable, a specially defined connector is required as described in the companion Agilent document called *Deep Storage with Xilinx ChipScope Pro and Agilent Technologies FPGA Trace Port Analyzer* which is available for download at http://www.xilinx.com/ise/verification/cspro_agilent_brochure.pdf.

Host System Requirements for Windows 2000/XP

The ChipScope Pro Analyzer, ChipScope Pro Core Generator, and ChipScope Pro Core Inserter tools run on PC systems running the Microsoft® Windows® operating system and meet the requirements outlined in Table 1-13.

Table 1-13: **PC System Requirements for ChipScope Pro 6.3i Tools**

OS Version	Memory	Java Environment
Windows 2000 SP2 or later	512 MB	Java Run-time Environment version 1.4.1_06 (automatically included in ChipScope Pro 6.3i software installation)
Windows XP Professional	512 MB	

Host System Requirements for Solaris 2.8 and 2.9

The ChipScope Pro Core Generator and ChipScope Pro Core Inserter tools run on workstation systems running Sun Microsystems Solaris operating system and meet the requirements outlined in Table 1-14.

Table 1-14: **Solaris Requirements for ChipScope Pro 6.3i Tools**

OS Version	Memory	Java Environment
Solaris 2.8 (32-bit)	512 MB	Java Run-time Environment version 1.4.1_06 (automatically included in ChipScope Pro 6.3i software installation)
Solaris 2.9 (32-bit)	512 MB	

Note: The ChipScope Pro Analyzer tool only runs on PC systems running the Microsoft Windows operating systems outlined in Table 1-13.

Host System Requirements for Linux

The ChipScope Pro Core Generator and ChipScope Pro Core Inserter tools run on workstation systems running the Linux operating system and meet the requirements outlined in Table 1-15.

Note: The Linux version of the ChipScope Pro 6.3i Core Generator and Core Inserter tools require that the Xilinx ISE 6.3i tools are installed on the target system and that the $XILINX environment variable is set up correctly.

Table 1-15: **Linux Requirements for ChipScope Pro 6.3i Tools**

OS Version	Memory	Java Environment
Red Hat Enterprise Linux WS 3 (32-bit)	512 MB	Java Run-time Environment version 1.4.1_06 (automatically included in ChipScope Pro 6.3i software installation)

Note: The ChipScope Pro Analyzer tool only runs on PC systems running the Microsoft Windows operating systems outlined in Table 1-13.

ChipScope Pro Software Installation

Installing ChipScope Pro Software for Windows 2000/XP

After downloading the ChipScope Pro Tools in the form of a self-extracting executable file (i.e., `ChipScope_Pro_6_3i_pc.exe`):

1. Make sure that you have administrator privileges on the target installation system in order to install the cable drivers correctly.
2. Choose **Start** → **Run**.
3. Browse for `ChipScope_Pro_6_3i_pc.exe`.
4. Choose **Run**.
5. Follow the install wizard instructions.
6. Use your 16-digit registration ID when prompted. You must register your ChipScope Pro 6.3i product at http://www.xilinx.com/chipscope in order to obtain your valid registration ID.

Note: The Parallel Cable driver is automatically installed if it is not already installed on the system.

Installing ChipScope Pro Software for Solaris 2.8 and 2.9

After downloading the ChipScope Pro Tools in the form of a compressed tape archive file (i.e., `ChipScope_Pro_6_3i_sol.tar.gz`):

1. Make sure that your Solaris operating system has all required patches (for more details, refer to the `README.sparc` file found in the installation archive)
2. Make sure that the **`gzip`** and **`tar`** programs are in your executable path.
3. Copy the `ChipScope_Pro_6_3i_sol.tar.gz` file to the desired installation directory.
4. Change directory to the desired installation directory.
5. Uncompress and extract the `ChipScope_Pro_6_3i_sol.tar.gz` file using the following command:
   ```
   gzip -cd ChipScope_Pro_6_3i_sol.tar.gz | tar xvf -
   ```
   This will create a directory called “chipscope” under the current working directory.
6. Set up the CHIPSCOPE environment variable to point to the chipscope installation:
   For csh:
   ```
   setenv CHIPSCOPE /path_to_chipscope_parent/chipscope
   ```
   For sh:
   ```
   set CHIPSCOPE=/path_to_chipscope_parent/chipscope
   export CHIPSCOPE
   ```
7. Run the ChipScope Pro Core Inserter and ChipScope Pro Core Generator tools:
   For the ChipScope Pro Core Inserter:
   ```
   $CHIPSCOPE/bin/sol/inserter.sh
   ```
   For the ChipScope Pro Core Generator:
   ```
   $CHIPSCOPE/bin/sol/gengui.sh
   ```

Installing ChipScope Pro Software for Linux

After downloading the ChipScope Pro Tools in the form of a compressed tape archive file (i.e., `ChipScope_Pro_6_3i_lin.tar.gz`):

1. Make sure that your Linux operating system has all required patches (for more details, refer to the `README.LINUX` file found in the installation archive)
2. Make sure that the Xilinx ISE 6.3i tools are installed on your system and that the $XILINX environment variable is set up correctly.
3. Make sure that the **`gzip`** and **`tar`** programs are in your executable path.
4. Copy the `ChipScope_Pro_6_3i_lin.tar.gz` file to the desired installation directory.
5. Change directory to the desired installation directory.
6. Uncompress and extract the `ChipScope_Pro_6_3i_lin.tar.gz` file using the following command:

   ```
   gzip -cd ChipScope_Pro_6_3i_lin.tar.gz | tar xvf -
   ```

   This will create a directory called "chipscope" under the current working directory.
7. Set up the CHIPSCOPE environment variable to point to the chipscope installation:

   For csh:

   ```
   setenv CHIPSCOPE /path_to_chipscope_parent/chipscope
   ```

   For sh:

   ```
   set CHIPSCOPE=/path_to_chipscope_parent/chipscope
   export CHIPSCOPE
   ```
8. Run the ChipScope Pro Core Inserter and ChipScope Pro Core Generator tools:

   For the ChipScope Pro Core Inserter:

   ```
   $CHIPSCOPE/bin/lin/inserter.sh
   ```

   For the ChipScope Pro Core Generator:

   ```
   $CHIPSCOPE/bin/lin/gengui.sh
   ```

Installing the Java Run-time Environment

The Java Run-time Environment (JRE) version 1.4.1_06 used by the ChipScope Pro 6.3i tools is automatically included under the ChipScope Pro 6.3i installation directory.

Chapter 2

Using the ChipScope Pro Core Generator

Core Generator Overview

The ChipScope Pro Core Generator tool is a graphical user interface used to generate the following cores:

- Integrated Controller core (ICON)
- Integrated Logic Analyzer core (ILA)
- Integrated Logic Analyzer with Agilent Trace Core (ILA/ATC)
- Integrated Bus Analyzer for CoreConnect On-Chip Peripheral Bus core (IBA/OPB)
- Integrated Bus Analyzer for CoreConnect Processor Local Bus core (IBA/PLB)
- Virtual Input/Output core (VIO)
- Agilent Trace Core 2 (ATC2)

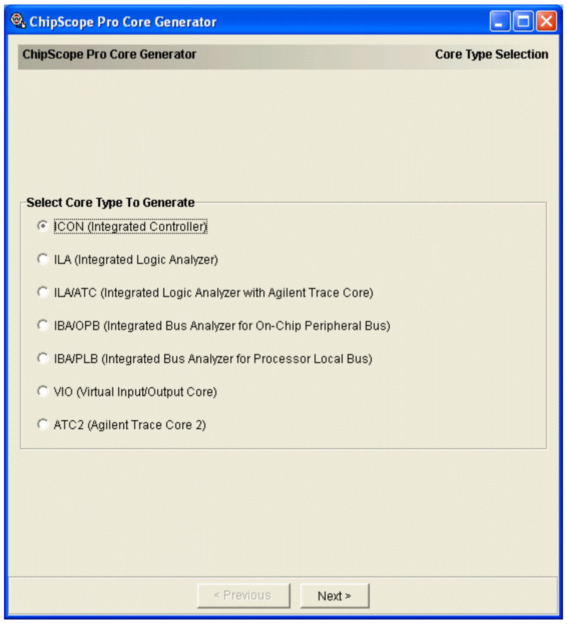
As a group, these cores are called the ChipScope Pro cores. After generating the ChipScope Pro cores, you can use the instantiation templates (that are provided) to quickly and easily insert the cores into their VHDL or Verilog design. After completing the instantiation and running synthesis, you can implement the design using the Xilinx ISE 6.3i implementation tools.

Generating an ICON Core

The Core Generator gives you the ability to define and generate a customized ICON core to use with one or more ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 capture cores in VHDL and Verilog designs. You can customize control ports (that is, the number of ChipScope Pro cores to be connected to the ICON core) and customize the use of the Boundary Scan primitive component (for example, BSCAN_VIRTEX2) that is used for JTAG communication.

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`) and example code in VHDL and Verilog specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select **ICON (Integrated Controller)** core (Figure 2-1, page 2-2), and click **Next**.

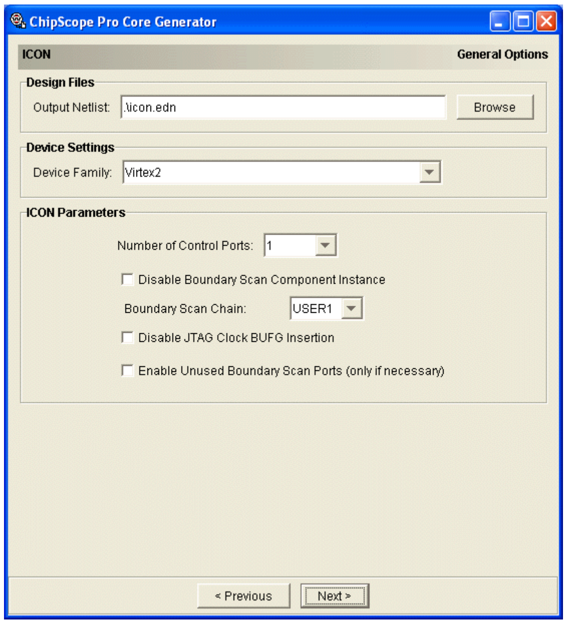

ChipScope Pro Core Generator
ChipScope Pro Core Generator — Core Type Selection

Select Core Type To Generate
- [✓ ICON (Integrated Controller)]
- ILA (Integrated Logic Analyzer)
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

Figure 2-1: **Selecting the ICON Core**

General ICON Core Options

The second screen in the Core Generator is used to set up the of the general ICON core options (Figure 2-2).


ChipScope Pro Core Generator
ICON — General Options

Design Files
Output Netlist: .\icon.edn

Device Settings
Device Family: Virtex2

ICON Parameters
Number of Control Ports: 1
☐ Disable Boundary Scan Component Instance
Boundary Scan Chain: USER1
☐ Disable JTAG Clock BUFG Insertion
☐ Enable Unused Boundary Scan Ports (only if necessary)

Figure 2-2: **ICON Core General Options**

Choosing the File Destination

The destination for the ICON EDIF file (`icon.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, the user can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. Use the pull-down selection to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Entering the Number of Control Ports

The ICON core can communicate with up to 15 ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 capture core units at any given time. However, individual capture core units cannot share their control ports with any other unit. Therefore, the ICON core needs up to 15 distinct control ports to handle this requirement. You can select the number of control ports from the Number of Control Ports pull-down list.

Disabling the Boundary Scan Component Instance

The Boundary Scan primitive component (for example, BSCAN_VIRTEX2) is used to communicate with the JTAG Boundary Scan logic of the target FPGA device. The Boundary Scan component extends the JTAG test access port (`TAP`) interface of the FPGA device so that up to two internal scan chains can be created. The ChipScope Pro Analyzer communicates with the ChipScope Pro cores by using one of the two internal scan chains (`USER1` or `USER2`) provided by the Boundary Scan component.

Since ChipScope Pro cores do not use both internal scan chains of the Boundary Scan component, it is possible to share the Boundary Scan component with other elements in the user's design. The Boundary Scan component can be shared with other parts of the design by using one of two methods:

- Instantiate the Boundary Scan component inside the ICON core and include the unused Boundary Scan scan chain signals as port signals on the ICON core interface.
- Instantiate the Boundary Scan component somewhere else in the design and attach either the `USER1` or `USER2` scan chain signals to corresponding port signals the ICON core interface.

The Boundary Scan component is instantiated inside the ICON core by default. Use the **Disable Boundary Scan Component Instance** checkbox to disable the instantiation of the Boundary Scan component.

Selecting the Boundary Scan Chain

The ChipScope Pro Analyzer can communicate with the ChipScope Pro cores using either the USER1 or USER2 boundary scan chains. If the Boundary Scan component is instantiated inside the ICON core, then you can select the desired scan chain from the Boundary Scan Chain pull-down list.

Disabling JTAG Clock BUFG Insertion

If the Boundary Scan component is instantiated inside the ICON core, then it is possible to disable the insertion of a `BUFG` component on the JTAG clock signal. Disabling the JTAG clock `BUFG` insertion causes the implementation tools to route the JTAG clock using normal routing resources instead of global clock routing resources. By default, this clock is placed on a global clock resource (`BUFG`). To disable this `BUFG` insertion, check select the **Disable JTAG Clock BUFG Insertion** checkbox. This should only be done if global resources are very scarce; placing the JTAG clock on regular routing, even high-speed backbone routing, introduces skew. Make sure the design is adequately constrained to minimize this skew.

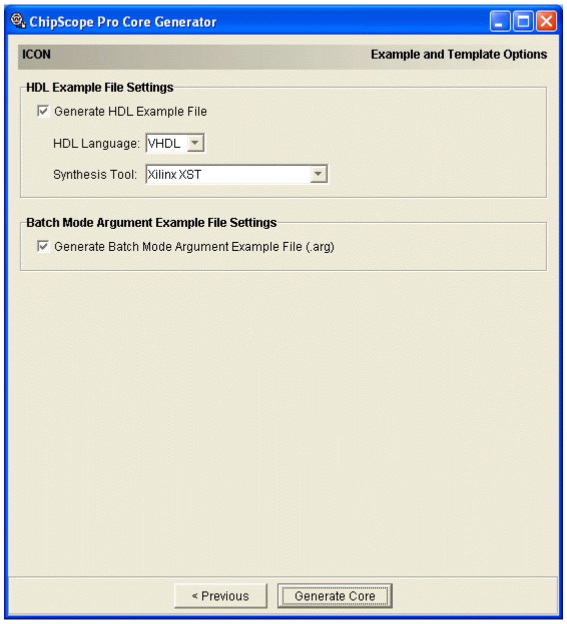
Including Boundary Scan Ports

The Boundary Scan primitive has two sets of ports: `USER1` and `USER2`. These ports provide an interface to the Boundary Scan TAP controller of the FPGA device. Since the ICON core uses only one of the `USER1` or `USER2` scan chain ports for communication purposes, the unused `USER2` or `USER1` port signals are available for use by other design elements, respectively. If the Boundary Scan component is instantiated inside the ICON core, then selecting the **Include Boundary Scan Ports** checkbox provides access to the unused `USER1` or `USER2` scan chain interfaces of the Boundary Scan component.

Note: The Boundary Scan ports should be included *only* if the design needs them. If they are included and not used, some synthesis tools do not connect the ICON core properly, causing errors during the synthesis and implementation stages of development.

Creating Example Templates

After selecting the parameters for the ICON core, click **Next** to view the Example and Template Options (Figure 2-3).


ChipScope Pro Core Generator

ICON — Example and Template Options

HDL Example File Settings
- ☑ Generate HDL Example File
- HDL Language: VHDL
- Synthesis Tool: Xilinx XST

Batch Mode Argument Example File Settings
- ☑ Generate Batch Mode Argument Example File (.arg)

Figure 2-3: **ICON Core Example Template Options**

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the ICON core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

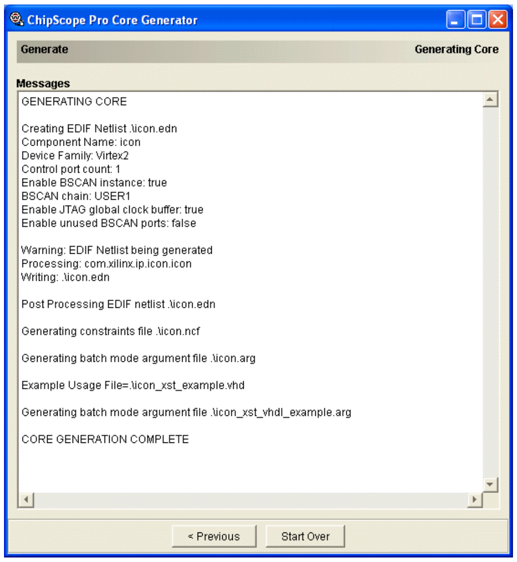
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `icon.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `icon.arg` file is used with the command line program called "**`generate.exe`**". The `icon.arg` file contains all of the arguments necessary for generating the ICON core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An ICON core can be generated by running "**generate.exe icon_pro -f=icon.arg**" at the command prompt on Windows systems or by running "**generate.sh icon_pro -f=icon.arg**" at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the ICON core parameters, click **Generate Core** to create the EDIF netlist, NCF constraint file, and applicable code examples. A message window opens (Figure 2-4), the progress information appears, and the CORE GENERATION COMPLETE message signals the end of the process. The user can then either go back and respecify different options or click **Start Over** to generate new cores.



```
GENERATING CORE

Creating EDIF Netlist .\icon.edn
Component Name: icon
Device Family: Virtex2
Control port count: 1
Enable BSCAN instance: true
BSCAN chain: USER1
Enable JTAG global clock buffer: true
Enable unused BSCAN ports: false

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.icon.icon
Writing: .\icon.edn

Post Processing EDIF netlist .\icon.edn

Generating constraints file .\icon.ncf

Generating batch mode argument file .\icon.arg

Example Usage File=.\icon_xst_example.vhd

Generating batch mode argument file .\icon_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-4: **ICON Core Generation Complete**

Using the ICON Core

To instantiate the example ICON core HDL files into your design, use the following guidelines to connect the ICON core port signals to various signals in your design:

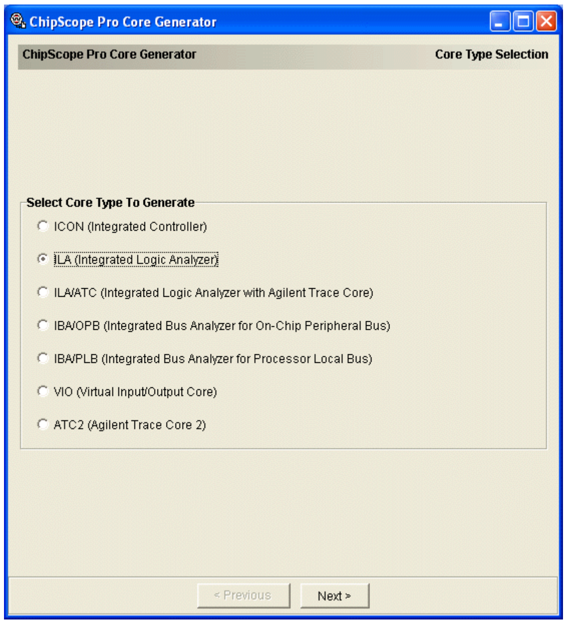
- Connect one of the ICON core's unused `CONTROL*` port signals to a control port of only one ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core instance in the design
- Do not leave any unused `CONTROL*` ports of the ICON core unconnected as this will cause the implementation tools to report an error. Instead, use an ICON core with the same number of `CONTROL*` ports as you have ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO or ATC2 cores.

Generating an ILA Core

The ChipScope Pro Core Generator allows you to define and generate a customized ILA capture core to use with HDL designs. You can customize the number, width, and capabilities of the trigger ports. You can also customize the maximum number of data samples stored by the ILA core, and the width of the data samples (if different from the trigger ports).

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), a signal import file (`*.cdc`) and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select **ILA (Integrated Logic Analyzer)**, and click **Next** (Figure 2-5).

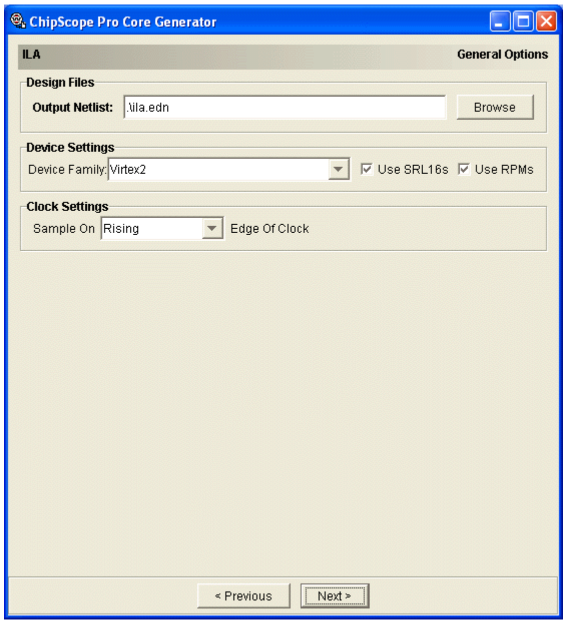


Select Core Type To Generate
- ICON (Integrated Controller)
- [✓ ILA (Integrated Logic Analyzer)]
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

Figure 2-5: **Selecting the ILA Core**

General ILA Core Options

The second screen in the Core Generator is used to set up the of the general ILA core options (Figure 2-6).


ChipScope Pro Core Generator

ILA General Options

Design Files

Output Netlist: .\ila.edn

Device Settings

Device Family: Virtex2 ☑ Use SRL16s ☑ Use RPMs

Clock Settings

Sample On Rising Edge Of Clock

Figure 2-6: **ILA Core General Options**

Choosing the File Destination

The destination for the ILA EDIF netlist (`ila.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the ILA core is optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Selecting the Clock Edge

The ILA unit can use either the rising or falling edges of the CLK signal to trigger and capture data. The Clock Settings pull-down list is used to select either the rising or falling edge of the `CLK` signal as the clock source for the ILA core.

Using SRL16s

The ILA core normally uses the SRL16 feature of the FPGA device to increase performance and decrease the area used by the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of SRL16s by the ILA core can be disabled by deselecting the **Use SRL16s** checkbox. It is recommended that the **Use SRL16s** checkbox remain enabled.

Note: SRL16s must be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

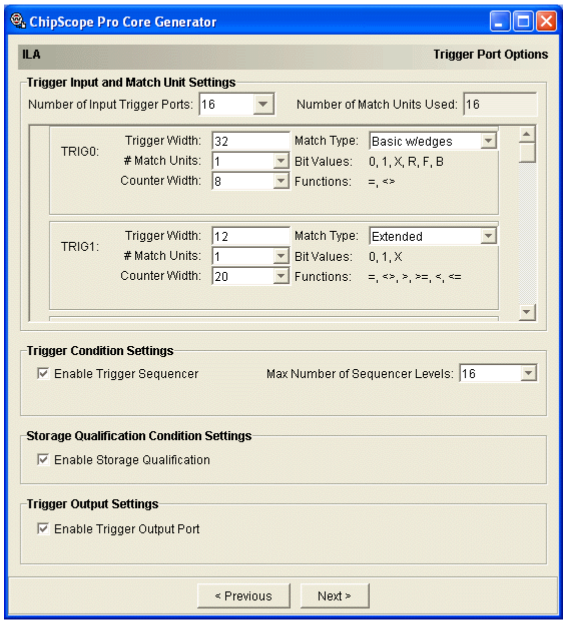
Using RPMs

The ILA core normally uses relationally placed macros (RPMs) to increase the performance of the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of RPMs by the ILA core can be disabled by deselecting the **Use RPMs** checkbox. It is recommended that the **Use RPMs** checkbox remain enabled for these device families.

Note: RPMs cannot be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

ILA Core Trigger Port Options

After you have set up the general ILA core options, click **Next**. This takes you to the third screen in the Core Generator that is used to set up the of the ILA core trigger port options (Figure 2-7).



Trigger Input and Match Unit Settings

Number of Input Trigger Ports: 16 | Number of Match Units Used: 16

TRIG0:
Trigger Width: 32 | Match Type: Basic w/edges
Match Units: 1 | Bit Values: 0, 1, X, R, F, B
Counter Width: 8 | Functions: =, <>

TRIG1:
Trigger Width: 12 | Match Type: Extended
Match Units: 1 | Bit Values: 0, 1, X
Counter Width: 20 | Functions: =, <>, >, >=, <, <=

Trigger Condition Settings

[✓] Enable Trigger Sequencer | Max Number of Sequencer Levels: 16

Storage Qualification Condition Settings

[✓] Enable Storage Qualification

Trigger Output Settings

[✓] Enable Trigger Output Port

Figure 2-7: **ILA Core Trigger Port Options**

Selecting the Number of Trigger Ports

Each ILA core can have up to 16 separate trigger ports that can be set up independently. After you choose a number from the Number of Trigger Ports pull-down list, a group of options appears for each trigger port. The group of options associated with each trigger port is labeled with `TRIG`*n*, where *n* is the trigger port number 0 to 15. The trigger port options include trigger width, number of match units connected to the trigger port, and the type of these match units.

Entering the Width of the Trigger Ports

The individual trigger ports are buses that are made up of individual signals or bits. The number of bits used to compose a trigger port is called the *trigger width*. The width of each trigger port can be set independently using the TRIG*n* Trigger Width field. The range of values that can be used for trigger port widths is 1 to 256.

Selecting the Number of Trigger Match Units

A match unit is a comparator that is connected to a trigger port and is used to detect events on that trigger port. The results of one or more match units are combined together to form what is called the overall trigger condition event that is used to control the capturing of data. Each trigger port `TRIG`*n* can be connected to 1 to 16 match units by using the # Match Units pull-down list.

Selecting one match unit conserves resources while still allowing some flexibility in detecting trigger events. Selecting two or more trigger match units allows a more flexible trigger condition equation to be a combination of multiple match units. However, increasing the number of match units per trigger port also increases the usage of logic resources accordingly.

Note: The aggregate number of match units used in a single ILA core cannot exceed 16, regardless of the number of trigger ports used.

Selecting the Match Unit Type

The different comparisons or match functions that can be performed by the trigger port match units depend on the type of the match unit. Six types of match units are supported by the ILA cores (Table 2-1, which spans multiple pages).

Table 2-1: **ILA Trigger Match Unit Types**

Type	Bit Values[a]	Match Function	Bits Per Slice[b]	Description
Basic	0, 1, X	'=', '<>'	8	Can be used for comparing data signals where transition detection is not important. This is the most bit-wise economical type of match unit.
Basic w/edges	0, 1, X, R, F, B	'=', '<>'	4	Can be used for comparing control signals where transition detection (e.g., low-to-high, high-to-low, etc.) is important.
Extended	0, 1, X	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where magnitude is important.
Extended w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where a magnitude and transition detection are important.
Range	0, 1, X	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values is important.
Range w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values and transition detection are important.

a. Bit values: '0' means "logical 0", '1' means "logical 1", 'X' means "don't care", 'R' means "0-to-1 transition", 'F' means "1-to-0 transition", and 'B' means "any transition".

b. The Bits Per Slice value is only an approximation that is used to illustrate the relative resource utilization of the different match unit types. It should not be used as a hard estimate of resource utilization.

Use the TRIG*n* Match Type pull-down list to select the type of match unit that applies to all match units connected to the trigger port. However, as the functionality of the match unit increases, so does the amount of resources necessary to implement that functionality. This flexibility allows you to customize the functionality of the trigger module while keeping resource usage in check.

Selecting Match Unit Counter Width

The match unit counter is a configurable counter on the output of the each match unit in a trigger port. This counter can be configured at run time to count a specific number of match unit events. To include a match counter on each match unit in the trigger port, select a counter width from 1 to 32. The match counter will not be included on each match unit if the Counter Width combo box is set to Disabled. The default Counter Width setting is Disabled.

Enabling the Trigger Condition Sequencer

The trigger condition sequencer can be either a Boolean equation, or an optional trigger sequencer that is enabled by checking the **Enable Trigger Sequencer** checkbox. A block diagram of the trigger sequencer is shown in Figure 2-8.

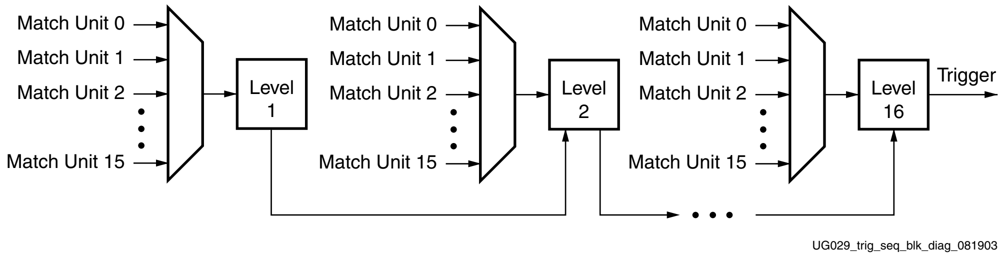


Figure 2-8: **Trigger Sequencer Block Diagram (with 16 levels and 16 match units)**

The trigger sequencer is implemented as a simple cyclical state machine and can transition through up to 16 states or levels before the trigger condition is satisfied. The transition from one level to the next is caused by an event on one of the match units that is connected to the trigger sequencer. Any match unit can be selected at run time on a per level basis to transition from one level to the next. The trigger sequencer can be configured at run time to transition from one level to the next on either contiguous or non-contiguous sequences of match function events.

Enabling the Storage Qualification Condition

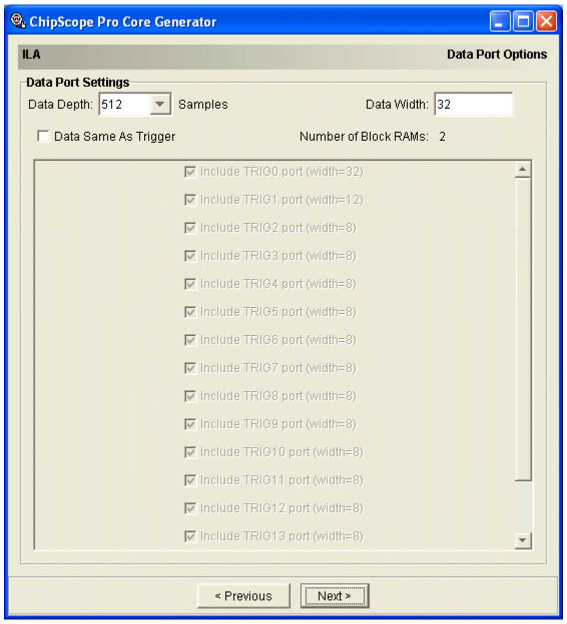
In addition to the trigger condition, the ILA core can also implement a storage qualification condition. The storage qualification condition is a Boolean combination of match function events. These match function events are detected by the match unit comparators that are subsequently attached to the trigger ports of the core. The storage qualification condition differs from the trigger condition in that it evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start the capture process and what data to capture. The storage qualification condition can be enabled by checking the **Enable Storage Qualification** checkbox.

Enabling the Trigger Output Port

The output of the ILA trigger condition module can be brought out to a port signal by checking the **Enable Trigger Output Port** checkbox. The trigger output port is used to trigger external test equipment by attaching the port signal to a device pin in the HDL design. The trigger output port can also be attached to other logic or ChipScope Pro cores in the design to be used as a trigger, an interrupt, or another control signal. The shape (level or pulse) and sense (active high or low) of the trigger output can also be controlled at run-time. The clock latency of the ILA trigger output port is 10 clock (`CLK`) cycles with respect to the trigger input ports.

ILA Core Data Port Options

After you have set up the ILA core trigger port options, click **Next**. This takes you to the fourth screen in the Core Generator that is used to set up the of the ILA core data port options (Figure 2-9).



Data Depth: 512 Samples

Data Width: 32

☐ Data Same As Trigger

Number of Block RAMs: 2

☑ Include TRIG0 port (width=32)
☑ Include TRIG1 port (width=12)
☑ Include TRIG2 port (width=8)
☑ Include TRIG3 port (width=8)
☑ Include TRIG4 port (width=8)
☑ Include TRIG5 port (width=8)
☑ Include TRIG6 port (width=8)
☑ Include TRIG7 port (width=8)
☑ Include TRIG8 port (width=8)
☑ Include TRIG9 port (width=8)
☑ Include TRIG10 port (width=8)
☑ Include TRIG11 port (width=8)
☑ Include TRIG12 port (width=8)
☑ Include TRIG13 port (width=8)

Figure 2-9: **ILA Core Data Port Options**

Selecting the Data Depth

The maximum number of data sample words that the ILA core can store in the sample buffer is called the *data depth*. The data depth determines the number of data width bits contributed by each block RAM unit used by the ILA unit.

For the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families), you can set the data depth to one of six values (Table 2-2).

Table 2-2: **Maximum Data Widths for Virtex-II/-II Pro/-4, Spartan-3**

	Depth 512	Depth 1024	Depth 2048	Depth 4096	Depth 8192	Depth 16384
1 block RAM	31	15	7	3	1	--
2 block RAMs	63	31	15	7	3	1
4 block RAMs	127	63	31	15	7	3
8 block RAMs	255	127	63	31	15	7
16 block RAMs	--	255	127	63	31	15
32 block RAMs	--	--	255	127	63	31
64 block RAMs	--	--	--	255	127	63
128 block RAMs	--	--	--	--	255	127
256 block RAMs	--	--	--	--	--	255

Note: One extra bit per sample is required for the trigger mark (that is, a trigger/data width of 7 bits requires a full sample width of 8 bits, etc.)

For the Virtex, Virtex-E, Spartan-II and Spartan-IIE device families (including the QPro variants of these families), you can set the data depth to one of five values (Table 2-3).

Table 2-3: **Maximum Data Widths for Virtex/-E, Spartan-II/-IIE**

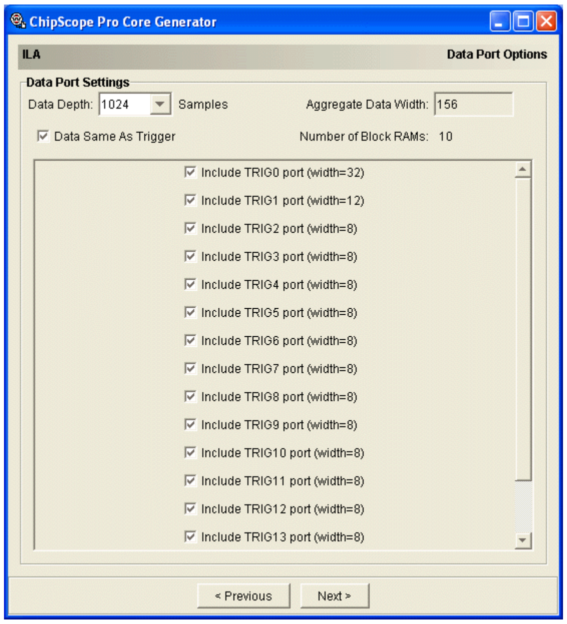
	Depth 256	Depth 512	Depth 1024	Depth 2048	Depth 4096
1 block RAM	15	7	3	1	--
2 block RAMs	31	15	7	3	1
4 block RAMs	63	31	15	7	3
8 block RAMs	127	63	31	15	7
16 block RAMs	255	127	63	31	15
32 block RAMs	--	255	127	63	31
64 block RAMs	--	--	255	127	63
128 block RAMs	--	--	--	255	127
256 block RAMs	--	--	--	--	255

Note: One extra bit per sample is required for the trigger mark (e.g., a trigger/data width of 7 bits requires a full sample width of 8 bits, etc.)

Selecting the Data Type

The data captured by the ILA trigger port can come from two different source types:

- **Data Same as Trigger (Figure 2-10, page 2-14)**
    - The data and trigger ports are identical. This mode is very common in most logic analyzers, since you can capture and collect any data used to trigger the core.
    - Individual trigger ports can be selected to be included in the data port. If this selection is made, then the `DATA` input port will not be included in the port map of the ILA core.
    - This mode conserves CLB and routing resources in the ILA core, but is limited to a maximum aggregate data sample word width of 256 bits.
- **Data Separate from Trigger (Figure 2-9, page 2-12)**
    - The data port is completely independent of the trigger ports.
    - This mode is useful when you want to limit the amount of data being captured.


ChipScope Pro Core Generator

ILA — Data Port Options

Data Port Settings

Data Depth: 1024 Samples — Aggregate Data Width: 156

☑ Data Same As Trigger — Number of Block RAMs: 10

☑ Include TRIG0 port (width=32)
☑ Include TRIG1 port (width=12)
☑ Include TRIG2 port (width=8)
☑ Include TRIG3 port (width=8)
☑ Include TRIG4 port (width=8)
☑ Include TRIG5 port (width=8)
☑ Include TRIG6 port (width=8)
☑ Include TRIG7 port (width=8)
☑ Include TRIG8 port (width=8)
☑ Include TRIG9 port (width=8)
☑ Include TRIG10 port (width=8)
☑ Include TRIG11 port (width=8)
☑ Include TRIG12 port (width=8)
☑ Include TRIG13 port (width=8)

Figure 2-10: **ILA Core Data Same As Trigger Options**

Entering the Data Width

The width of each data sample word stored by the ILA core is called the *data width*. If the data and trigger words are independent from each other, then the maximum allowable data width depends on the target device type and data depth. However, regardless of these factors, the maximum allowable data width is 256 bits.

Selecting the Data-Same-As-Trigger Ports

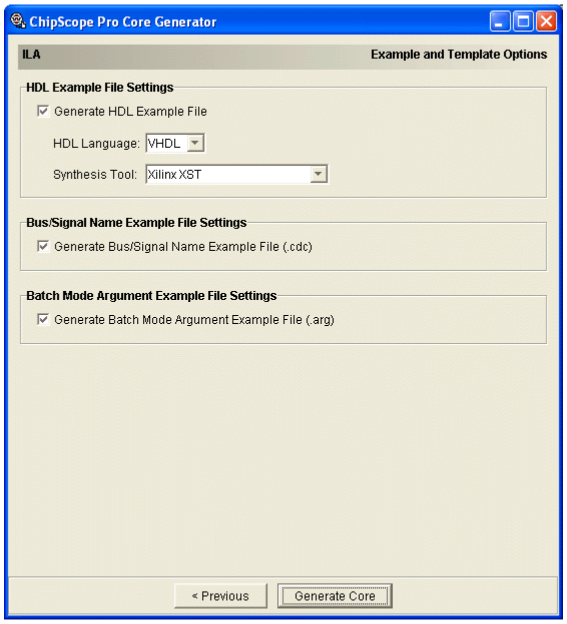
If the **Data Same As Trigger** checkbox is selected, then a checkbox for each `TRIG`*`n`* port appears in the data port options screen. These checkboxes should be used to select the individual trigger ports that will be included in the aggregate data port. Note that selecting the individual trigger ports automatically updates the Aggregate Data Width field accordingly. A maximum data width of 256 bits applies to the aggregate selection of trigger ports.

Number of Block RAMs

As the data depth and data width selections are changed, the Number of Block RAMs field notifies you of how many block RAMs will be used by the ILA core. The trigger mark is automatically taken into account when calculating this value.

Creating Example Templates

After selecting the parameters for the ILA core, click **Next** to view the Example and Template Options (Figure 2-11).


ChipScope Pro Core Generator
ILA
Example and Template Options
HDL Example File Settings
Bus/Signal Name Example File Settings
Batch Mode Argument Example File Settings
< Previous
Generate Core

☑ Generate HDL Example File

HDL Language: VHDL

Synthesis Tool: Xilinx XST

☑ Generate Bus/Signal Name Example File (.cdc)

☑ Generate Batch Mode Argument Example File (.arg)

Figure 2-11: **ILA Core Example and Template Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the ILA core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

Bus/Signal Name Example Files (.CDC)

The bus/signal name example file for the ILA core (for example, `ila.cdc`) contains generic information about the trigger and data ports of the ILA core. The `ila.cdc` file will be created if you select the **Generate Bus/Signal Name Example File (.cdc)** checkbox. You can use the `ila.cdc` file as a template to change trigger and/or data port signal names, create buses, and so on. The modified `ila.cdc` file can then be imported into the ChipScope Pro Analyzer tool and applied to the appropriate ILA core by using the **File → Import** option.

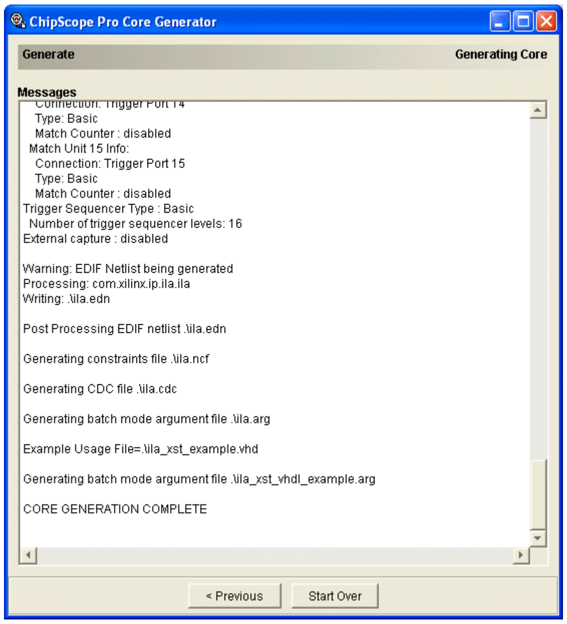
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `ila.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `ila.arg` file is used with the command line program called "**generate.exe**". The `ila.arg` file contains all of the arguments necessary for generating the ILA core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An ILA core can be generated by running "**generate.exe ila_pro -f=ila.arg**" at the command prompt on Windows systems or by running "**generate.sh ila_pro -f=ila.arg**" at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the ILA core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the `CORE GENERATION COMPLETE` message signals the end of the process (Figure 2-12). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
    Connection: Trigger Port 14
    Type: Basic
    Match Counter : disabled
  Match Unit 15 Info:
    Connection: Trigger Port 15
    Type: Basic
    Match Counter : disabled
Trigger Sequencer Type : Basic
  Number of trigger sequencer levels: 16
External capture : disabled

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.ila.ila
Writing: .\ila.edn

Post Processing EDIF netlist .\ila.edn

Generating constraints file .\ila.ncf

Generating CDC file .\ila.cdc

Generating batch mode argument file .\ila.arg

Example Usage File=.\ila_xst_example.vhd

Generating batch mode argument file .\ila_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-12: **ILA Core Generation Complete**

Using the ILA Core

To instantiate the example ILA core HDL files into your design, use the following guidelines to connect the ILA core port signals to various signals in your design:

- Connect the ILA core's CONTROL port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the ILA core's data and trigger port signals to "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- Make sure the data and trigger source signals are synchronous to the ILA clock signal (`CLK`)

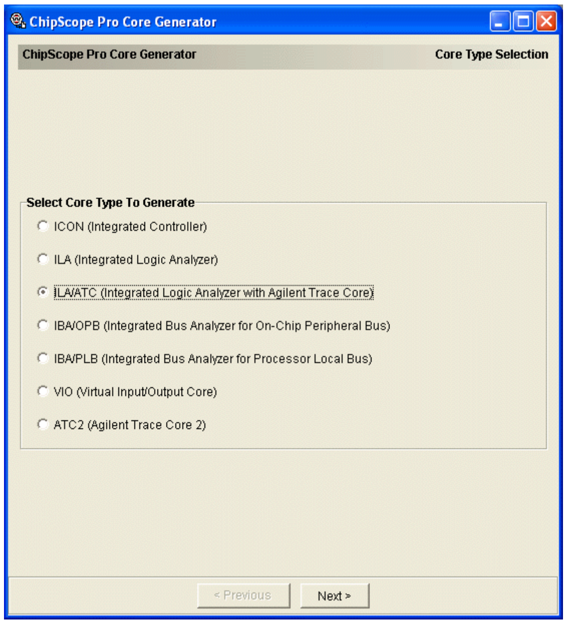
Generating an ILA/ATC Core

The ILA/ATC core is similar to an ILA core except that the data is captured off-chip by the Agilent E5904B Option 500 Trace Port Analyzer (Agilent TPA). The ILA/ATC core is connected to the Agilent TPA cable using a special 4 to 20 data pin trace port connection.

The ChipScope Pro Core Generator allows you to define and generate a customized ILA/ATC capture core to use with HDL designs. You can customize the number, width, and capabilities of the trigger ports. You can also customize the maximum number of data pins to be used as samples stored by the ILA/ATC core, the location of these pins and the width of the data samples.

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select **ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)**, and click **Next** (Figure 2-13).


ChipScope Pro Core Generator

ChipScope Pro Core Generator — Core Type Selection

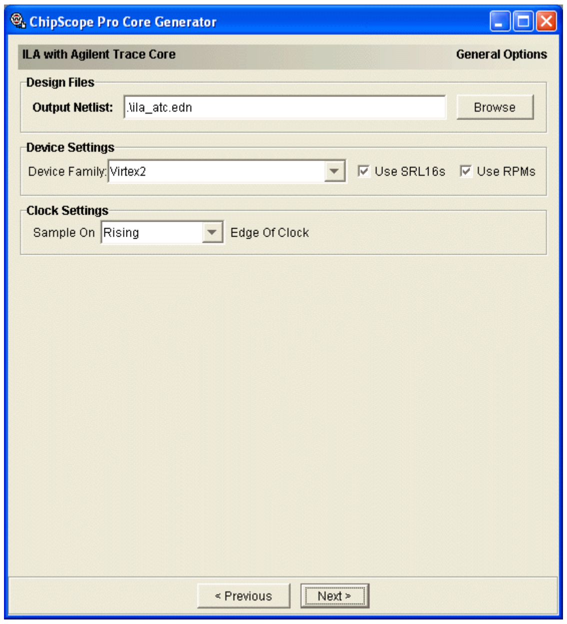
Select Core Type To Generate

- ICON (Integrated Controller)
- ILA (Integrated Logic Analyzer)
- [✓ ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)]
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

Figure 2-13: **Selecting the ILA/ATC Core**

General ILA/ATC Core Options

The second screen in the Core Generator is used to set up the of the general ILA/ATC core options (Figure 2-14).



ChipScope Pro Core Generator

ILA with Agilent Trace Core — General Options

Design Files
Output Netlist: .\ila_atc.edn

Device Settings
Device Family: Virtex2 [✓ Use SRL16s] [✓ Use RPMs]

Clock Settings
Sample On: Rising Edge Of Clock

Figure 2-14: **ILA/ATC Core General Options**

Choosing the File Destination

The destination for the ILA/ATC EDIF netlist (`ila_atc.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the ILA/ATC core is optimized for the selected device family. Use the pull-down list to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Selecting the Clock Edge

The ILA/ATC unit can use either the rising or falling edges of the `CLK` signal to trigger and capture data. The Clock Settings pull-down list is used to select either the rising or falling edge of the `CLK` signal as the clock source for the ILA/ATC core.

Using SRL16s

The ILA/ATC core normally uses the SRL16 feature of the FPGA device to increase performance and decrease the area used by the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of SRL16s by the ILA/ATC core can be disabled by deselecting the **Use SRL16s** checkbox. It is recommended that the **Use SRL16s** checkbox remain enabled.

Note: SRL16s must be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

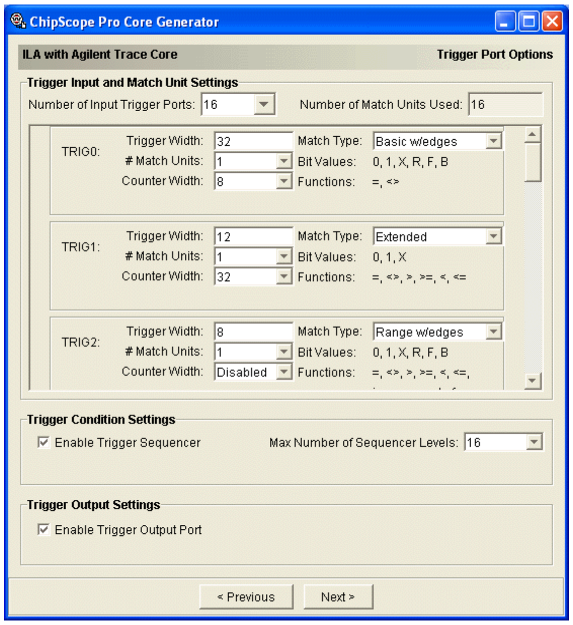
Using RPMs

The ILA/ATC core normally uses relationally placed macros (RPMs) to increase the performance of the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of RPMs by the ILA/ATC core can be disabled by deselecting the **Use RPMs** checkbox. It is recommended that the **Use RPMs** checkbox remain enabled for these device families.

Note: RPMs cannot be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

ILA/ATC Core Trigger Port Options

After you have set up the general ILA/ATC core options, click **Next**. This takes you to the third screen in the Core Generator that is used to set up the of the ILA/ATC core trigger port options (Figure 2-15).



Trigger Input and Match Unit Settings

Number of Input Trigger Ports: 16
Number of Match Units Used: 16

TRIG0:
Trigger Width: 32
Match Units: 1
Counter Width: 8
Match Type: Basic w/edges
Bit Values: 0, 1, X, R, F, B
Functions: =, <>

TRIG1:
Trigger Width: 12
Match Units: 1
Counter Width: 32
Match Type: Extended
Bit Values: 0, 1, X
Functions: =, <>, >, >=, <, <=

TRIG2:
Trigger Width: 8
Match Units: 1
Counter Width: Disabled
Match Type: Range w/edges
Bit Values: 0, 1, X, R, F, B
Functions: =, <>, >, >=, <, <=,

Trigger Condition Settings

☑ Enable Trigger Sequencer
Max Number of Sequencer Levels: 16

Trigger Output Settings

☑ Enable Trigger Output Port

Figure 2-15: **ILA/ATC Core Trigger Port Options**

Selecting the Number of Trigger Ports

Each ILA/ATC core can have up to 16 separate trigger ports that can be set up independently. After you choose a number from the Number of Trigger Ports pull-down list, a group of options appears for each trigger port. The group of options associated with each trigger port is labeled with `TRIG`*n*, where *n* is the trigger port number 0 to 15. The trigger port options include trigger width, number of match units connected to the trigger port, and the type of these match units.

Entering the Width of the Trigger Ports

The individual trigger ports are buses that are made up of individual signals or bits. The number of bits used to compose a trigger port is called the *trigger width*. The width of each trigger port can be set independently using the TRIG*n* Trigger Width field. The range of values that can be used for trigger port widths is 1 to 256.

Selecting the Number of Trigger Match Units

A match unit is a comparator that is connected to a trigger port and is used to detect events on that trigger port. The results of one or more match units are combined together to form what is called the overall trigger condition event that is used to control the capturing of data. Each trigger port `TRIG`*n* can be connected to 1 to 16 match units by using the # Match Units pull-down list.

Selecting one match unit conserves resources while still allowing some flexibility in detecting trigger events. Selecting two or more trigger match units allows a more flexible trigger condition equation to be a combination of multiple match units. However, increasing the number of match units per trigger port increases the usage of logic resources accordingly.

Selecting the Match Unit Type

The different comparisons or match functions that can be performed by the trigger port match units depend on the type of the match unit. Six types of match units are supported by the ILA/ATC cores (Table 2-4).

Table 2-4: **ILA/ATC Trigger Match Unit Types**

Type	Bit Values[a]	Match Function	Bits Per Slice[b]	Description
Basic	0, 1, X	'=', '<>'	8	Can be used for comparing data signals where transition detection is not important. This is the most bit-wise economical type of match unit.
Basic w/edges	0, 1, X, R, F, B	'=', '<>'	4	Can be used for comparing control signals where transition detection (e.g., low-to-high, high-to-low, etc.) is important.
Extended	0, 1, X	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where magnitude is important.
Extended w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where a magnitude and transition detection are important.
Range	0, 1, X	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values is important.
Range w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values and transition detection are important.

a. Bit values: '0' means "logical 0", '1' means "logical 1", 'X' means "don't care", 'R' means "0-to-1 transition", 'F' means "1-to-0 transition" and 'B' means "any transition".

b. The Bits Per Slice value is only an approximation that is used to illustrate the relative resource utilization of the different match unit types. It should not be used as a hard estimate of resource utilization.

Use the TRIG*n* Match Type pull-down list to select the type of match unit that will apply to all match units connected to the trigger port. However, as the functionality of the match unit increases, so does the amount of resources necessary to implement that functionality. This flexibility allows you to customize the functionality of the trigger module while keeping resource usage in check.

Selecting Match Unit Counter Width

The match unit counter is a configurable counter on the output of the each match unit in a trigger port. This counter can be configured at run time to count a specific number of match unit events. To include a match counter on each match unit in the trigger port, select a counter width from 1 to 32. The match counter will not be included on each match unit if the Counter Width combo box is set to Disabled. The default Counter Width setting is Disabled.

Enabling the Trigger Condition Sequencer

The trigger condition can be either a Boolean equation, or an optional trigger sequencer that is enabled by checking the **Enable Trigger Sequencer** checkbox. A block diagram of the trigger sequencer is shown in Figure 2-8, page 2-11.

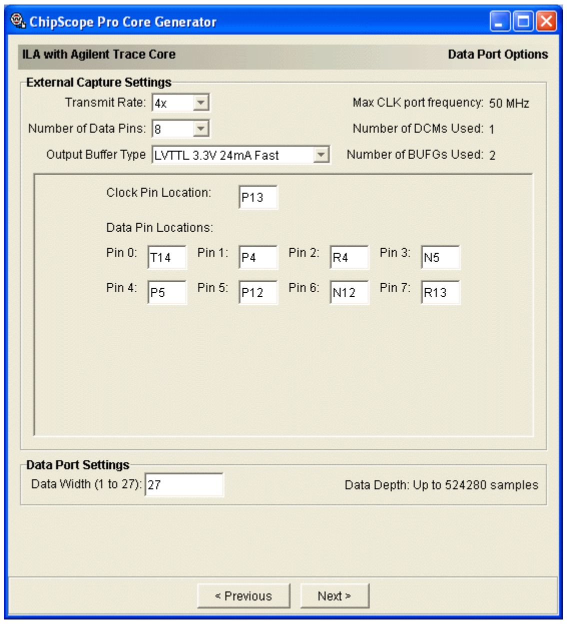
The trigger sequencer is implemented as a simple cyclical state machine and can transition through up to 16 states or levels before the trigger condition is satisfied. The transition from one level to the next is caused by an event on one of the match units that are connected to the trigger sequencer. Any match unit can be selected at run time on a per level basis to transition from one level to the next.

Enabling the Trigger Output Port

The output of the ILA/ATC trigger condition module can be brought out to a port signal by checking the **Enable Trigger Output Port** checkbox. The trigger output port is can be used to trigger external test equipment by attaching the port signal to a device pin in the HDL design. The trigger output port can also be attached to other logic or ChipScope Pro cores in the design to be used as a trigger, an interrupt, or another control signal. The shape (level or pulse) and sense (active high or low) of the trigger output can also be controlled at run-time. The clock latency of the ILA/ATC trigger output port is 10 clock (`CLK`) cycles with respect to the trigger input ports.

ILA/ATC Core Data Port Options

After you have set up the ILA/ATC core trigger port options, click **Next**. This takes you to the fourth screen in the Core Generator that is used to set up the of the ILA core data port options (Figure 2-16).


ChipScope Pro Core Generator
ILA with Agilent Trace Core
Data Port Options
External Capture Settings
Data Port Settings
< Previous
Next >

Transmit Rate: 4x
Number of Data Pins: 8
Output Buffer Type LVTTL 3.3V 24mA Fast
Max CLK port frequency: 50 MHz
Number of DCMs Used: 1
Number of BUFGs Used: 2

Clock Pin Location: P13

Data Pin Locations:

Pin 0: T14 Pin 1: P4 Pin 2: R4 Pin 3: N5
Pin 4: P5 Pin 5: P12 Pin 6: N12 Pin 7: R13

Data Width (1 to 27): 27
Data Depth: Up to 524280 samples

Figure 2-16: **ILA/ATC Core Data Port Options**

Transmit Rate

The ILA/ATC core does not use on-chip memory resources to store the captured trace data. Instead, it transmits the data to be captured to an Agilent TPA that is attached to a special connector via FPGA device pins. The data can be transmitted out the device pins at the same rate as the incoming DATA port (transmit rate = 1x), twice the rate as the DATA port (transmit rate = 2x) or four times the DATA port rate (transmit rate = 4x).

Maximum CLK Port Frequency

The maximum output clock frequency is 200 MHz. The incoming `CLK` port rate is limited by the maximum output clock frequency as well as the transmit rate. If the transmit rate is 1x, then the maximum CLK port frequency is 200 MHz. If the transmit rate is 2x, then the maximum CLK port frequency is 100 MHz. Finally, if the transmit rate is 4x, then the maximum CLK port frequency is 50 MHz.

Clock Resource Utilization

The ILA/ATC core will use dedicated clock resources if the transmit rate is 2x or 4x. The number of clock resources required by the ILA/ATC core is shown in Table 2-5.

Table 2-5: **ILA/ATC Clock Resource Utilization**

Transmit Rate	Virtex, Virtex-E, Spartan-II and Spartan-IIE		Virtex-II, Virtex-II Pro, Virtex-4 and Spartan-3	
	Number of BUFGs	Number of CLKDLLs	Number of BUFGs	Number of DCMs
1x	0	0	0	0
2x	1	1	1	1
4x	1	2	2	1

Number of Data Pins

The ILA/ATC core can use 4, 8, 12, 16, or 20 output data pins for external capture.

Output Buffer Type

You can select the type of output buffer used for the output clock and data pins. The types of output data buffers that are supported by the ILA/ATC cores depend on the device family (Table 2-6).

Table 2-6: **ILA/ATC Output Buffer Types by Device Family**

Buffer Type	Virtex-4, Virtex-II Pro, Virtex-II, Spartan-3	Virtex-E, Spartan-IIE	Virtex, Spartan-II
LVTTL 3.3V 24mA Fast	Yes	Yes	Yes
LVTTL 3.3V 12mA Fast	Yes	Yes	Yes
LVCMOS 3.3V 24mA Fast	Yes	No	No
LVCMOS 3.3V 12mA Fast	Yes	No	No
LVCMOS 2.5V 24mA Fast	Yes	No	No
LVCMOS 2.5V 12mA Fast	Yes	Yes	Yes
LVCMOS 1.8V 16mA Fast	Yes	No	No
LVCMOS 1.8V 12mA Fast	Yes	Yes	No
LVDCI 3.3V	Yes	No	No
LVDCI 2.5V	Yes	No	No
LVDCI 1.8V	Yes	No	No

Output Clock and Data Pin Locations

The clock and data pins are instantiated inside the ILA/ATC core for your convenience. This means that although you do not have to manually bring the clock and data pins through every level of hierarchy to the top-level of your design, you do need to specify the location of these pins in the Core Generator. The pin locations are then added to the `*.ncf` file of the ILA/ATC core.

Data Width and Depth

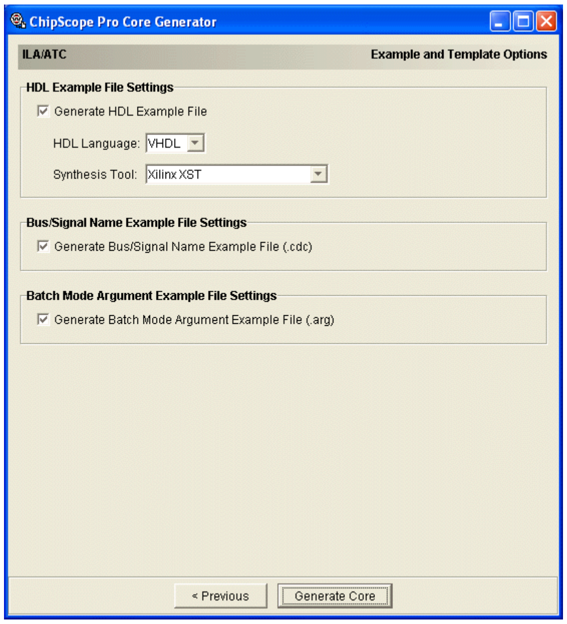
The data width and depth of the ILA/ATC core depend on the transmit rate and the number of data pins (Table 2-7).

Table 2-7: **ILA/ATC Core Capabilities**

Number of Data Pins	Transmit Rate	Max Width of DATA Port	Max DATA Depth (with timestamps)
4	1x	3	2,097,120 (1,048,560)
4	2x	5	1,048,560 (1048560)
4	4x	11	524,280 (262,136)
8	1x	7	2,097,120 (1,048,560)
8	2x	13	1,048,560 (524,280)
8	4x	27	524,280 (262,136)
12	1x	11	2,097,120 (1,048,560)
12	2x	21	1,048,560 (524,280)
12	4x	43	524,280 (262,136)
16	1x	15	2,097,120 (1,048,560)
16	2x	29	1,048,560 (524,280)
16	4x	59	524,280 (262,136)
20	1x	19	2,097,120 (1,048,560)
20	2x	37	1,048,560 (524,280)
20	4x	75	524,280 (262,136)

Creating Example Templates

After selecting the parameters for the ILA/ATC core, click **Next** to view the Example and Template Generation Options (Figure 2-17).



HDL Example File Settings

☑ Generate HDL Example File

HDL Language: VHDL

Synthesis Tool: Xilinx XST

Bus/Signal Name Example File Settings

☑ Generate Bus/Signal Name Example File (.cdc)

Batch Mode Argument Example File Settings

☑ Generate Batch Mode Argument Example File (.arg)

Figure 2-17: **ILA/ATC Core Example and Template Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the ILA/ATC core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

Bus/Signal Name Example Files (.CDC)

The bus/signal name example file for the ILA/ATC core (for example, `ila_atc.cdc`) contains generic information about the trigger and data ports of the ILA/ATC core. The `ila_atc.cdc` file will be created if you select the **Generate Bus/Signal Name Example File (.cdc)** checkbox. You can use the `ila_atc.cdc` file as a template to change trigger and/or data port signal names, create buses, and so on. The modified `ila_atc.cdc` file can then be imported into the ChipScope Pro Analyzer tool and applied to the appropriate ILA/ATC core by using the **File** → **Import** option.

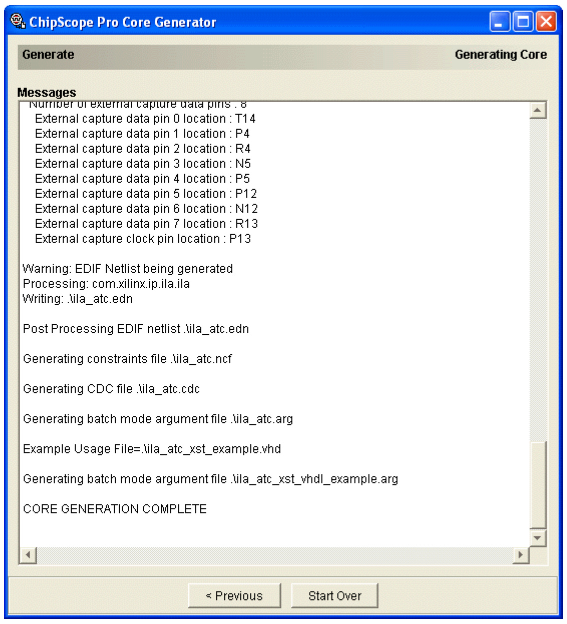
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `ila_atc.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `ila_atc.arg` file is used with the command line program called "**generate.exe**". The `ila_atc.arg` file contains all of the arguments necessary for generating the ILA/ATC core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An ILA/ATC core can be generated by running "**generate.exe ila_pro_atc -f=ila_atc.arg**" at the command prompt on Windows systems or by running "**generate.sh ila_pro_atc -f=ila_atc.arg**" at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the ILA/ATC core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the `CORE GENERATION COMPLETE` message signals the end of the process (Figure 2-18). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
  Number of external capture data pins : 8
   External capture data pin 0 location : T14
   External capture data pin 1 location : P4
   External capture data pin 2 location : R4
   External capture data pin 3 location : N5
   External capture data pin 4 location : P5
   External capture data pin 5 location : P12
   External capture data pin 6 location : N12
   External capture data pin 7 location : R13
   External capture clock pin location : P13

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.ila.ila
Writing: .\ila_atc.edn

Post Processing EDIF netlist .\ila_atc.edn

Generating constraints file .\ila_atc.ncf

Generating CDC file .\ila_atc.cdc

Generating batch mode argument file .\ila_atc.arg

Example Usage File=.\ila_atc_xst_example.vhd

Generating batch mode argument file .\ila_atc_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-18: **ILA/ATC Core Generation Complete**

Using the ILA/ATC Core

To instantiate the example ILA/ATC core HDL files into your design, use the following guidelines to connect the ILA/ATC core port signals to various signals in your design:

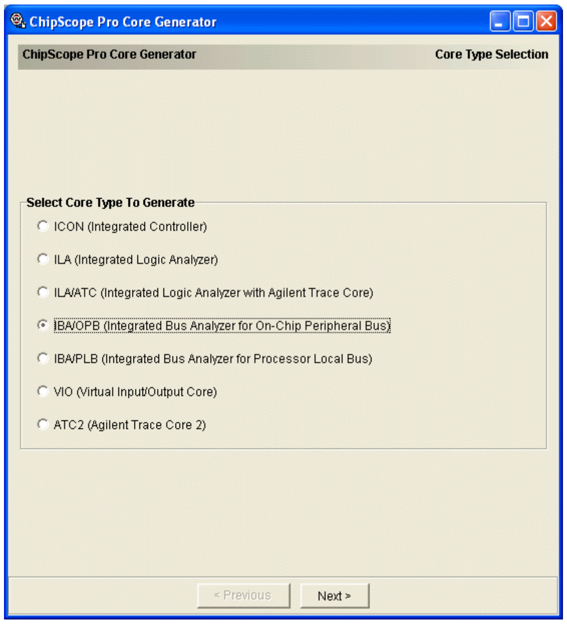
- Connect the ILA/ATC core's `CONTROL` port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the ILA/ATC core's data and trigger port signals to "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- Make sure the data and trigger source signals are synchronous to the ILA/ATC clock signal (`CLK`)

Generating the IBA/OPB Core

The ChipScope Pro Core Generator allows you to define and generate a customized IBA core for monitoring and debugging CoreConnect OPB buses in your HDL designs. You can customize the number of OPB masters and slaves as well as the types of OPB signals that you want to use as triggers to your IBA/OPB core. You can also customize the maximum number of data samples stored by the IBA/OPB core and the width of the data samples (if different from the trigger ports).

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), a signal import file (`*.cdc`), a protocol violation bus token file (`*.tok`) and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select **IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)**, and click **Next** (Figure 2-19).

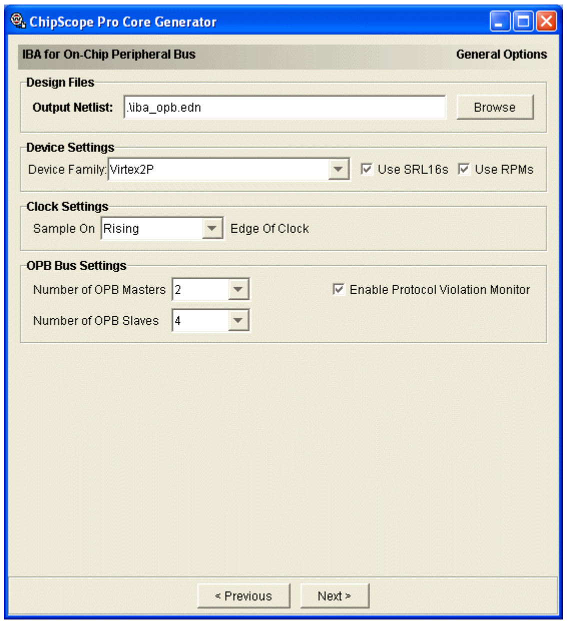


Select Core Type To Generate
- ICON (Integrated Controller)
- ILA (Integrated Logic Analyzer)
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- [✓ IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)]
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

Figure 2-19: **Selecting the IBA/OPB Core**

General IBA/OPB Core Options

The second screen in the Core Generator is used to set up the of the general IBA/OPB core options (Figure 2-20).



Design Files
Output Netlist: .\iba_opb.edn

Device Settings
Device Family: Virtex2P
☑ Use SRL16s ☑ Use RPMs

Clock Settings
Sample On Rising Edge Of Clock

OPB Bus Settings
Number of OPB Masters 2
Number of OPB Slaves 4
☑ Enable Protocol Violation Monitor

Figure 2-20: **IBA/OPB Core General Options**

Choosing the File Destination

The destination for the IBA/OPB EDIF netlist (`iba_opb.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the IBA/OPB core is optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II Pro is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Selecting the Clock Edge

The IBA/OPB unit can use either the rising or falling edges of the CLK signal to trigger and capture data. The Clock Settings pull-down list is used to select either the rising or falling edge of the CLK signal as the clock source for the IBA/OPB core.

OPB Bus Settings

The IBA/OPB core is designed to passively attach to the OPB bus/arbiter component in your embedded PowerPC or MicroBlaze processor design. Like the OPB bus and arbiter components, the IBA/OPB core is capable of handling up to 16 OPB masters and 64 OPB slaves.

Note: It is very important to specify the same number of OPB masters and slaves that you have on the OPB bus of your embedded processor design, otherwise the IBA/OPB core may not function properly.

Another key feature of the IBA/OPB core is its ability to monitor the OPB bus for up to 32 different bus protocol violations (Table 2-8, which spans multiple pages). To include this functionality in the IBA/OPB core, select the **Enable Protocol Violation Monitor** checkbox.

Table 2-8: **CoreConnect OPB Protocol Violation Error Description**[a]

Priority	Bit Encoding	Error	Description
1	011010	1.19.2	`OPB_DBus` changed state during a write operation before receipt of `OPB_xferAck`.
2	011001	1.19.1	`OPB_ABus` changed state during an operation before receipt of `OPB_xferAck`.
3	001100	1.6.1	`OPB_ABus`: No `Mx_Select` signal active and non zero `OPB_ABus`.
4	001101	1.7.1	`OPB_DBus`: No `Mx_Select` signal active and non zero `OPB_DBus`.
5	010101	1.13.1	`OPB_xferAck`: `OPB_xferAck` active with no `Mx_select`.
6	010110	1.13.2	`OPB_xferAck`: `OPB_xferAck` did not activate within 16 cycles of `OPB_select`.
7	010111	1.15.1	`OPB_errAck`: `OPB_errAck` active with no `Mx_select`.
8	000100	1.4.0	`OPB_retry`: `OPB_retry` and `OPB_xferAck` active in the same cycle.
9	000111	1.4.3	`OPB_retry`: `OPB_retry` active for more than a single cycle.
10	000000	1.2.1	`OPB_MxGrant`: More than 1 `OPB_MxGrant` signals active in same cycle.
11	000001	1.2.2	`OPB_MxGrant`: An `OPB_MxGrant` signal is active for a non-owning master.
12	000010	1.3.1	`OPB_BusLock`: `OPB_BusLock` asserted without a grant in the previous cycle and without `OPB_select`.
13	000011	1.3.2	`OPB_BusLock`: Bus is locked and a master other than bus owner has been granted the bus.

Table 2-8: **CoreConnect OPB Protocol Violation Error Description[a] *(Continued)***

Priority	Bit Encoding	Error	Description
14	001000	1.4.4	`OPB_retry`: `OPB_select` remained active after a retry cycle.
15	001001	1.4.5	`OPB_retry`: `OPB_retry` active with no `Mx_select`.
16	001110	1.8.1	`OPB_Select`: `Mx_Select` signal active without having control of the bus via `OPB_MxGrant` or `OPB_busLock`.
17	001111	1.8.2	`OPB_Select`: More than one `Mx_Select` signals active in the same cycle.
18	010000	1.9.1	`OPB_RNW`: `OPB_RNW` high with no `Mx_select`.
19	011011	1.19.3	`OPB_RNW` changed state during an operation before receipt of `OPB_xferAck`.
20	011100	1.19.4	`OPB_select` changed state during an operation before receipt of `OPB_xferAck`.
21	011101	1.19.5	`OPB_BEBus` changed state during a write or read operation before receipt of `OPB_xferAck`.
22	011110	1.20.3	Byte enable transfer not aligned with address offset.
23	011111	1.20.4	Byte enable transfer initiated with non contiguous byte enables.
24	000110	1.4.2	`OPB_retry`: `Mx_Request` from retried master remained active after a retry cycle.
25	000101	1.4.1	`OPB_retry`: `OPB_BusLock` remained active after a retry cycle.
26	010001	1.11.1	`OPB_seqAddr`: `OPB_seqAddr` active with no `OPB_BusLock`.
27	010010	1.11.2	`OPB_seqAddr`: `OPB_seqAddr` active with no Mx_select.
28	010011	1.11.3	`OPB_seqAddr`: `OPB_ABUS` did not increment properly during `OPB_seqAddr`.
29	010100	1.11.4	`OPB_seqAddr`: `OPB_seqAddr` was asserted without a transaction boundary.
30	011000	1.16.1	`OPB_ToutSup`: `OPB_ToutSup` active with no Mx_select.

Table 2-8: **CoreConnect OPB Protocol Violation Error Description[a] *(Continued)***

Priority	Bit Encoding	Error	Description
31	001010	1.5.1	`OPB_Timeout`: Arbiter failed to signal `OPB_Timeout` after 16 non-responding cycles.
32	001011	1.5.2	`OPB_Timeout`:`OPB_Timeout` active with no `Mx_select`.
33	111111	N/A	No errors

a. Refer to Chapter 8 of the *OPB Bus Functional Model Toolkit User's Manual* from IBM for more information on these CoreConnect OPB errors. (This manual is included in the IBM CoreConnect installation.)

Using SRL16s

The IBA/OPB core normally uses the SRL16 feature of the FPGA device to increase performance and decrease the area used by the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of SRL16s by the IBA/OPB core can be disabled by deselecting the **Use SRL16s** checkbox. It is recommended that the **Use SRL16s** checkbox remain enabled.

Note: SRL16s must be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

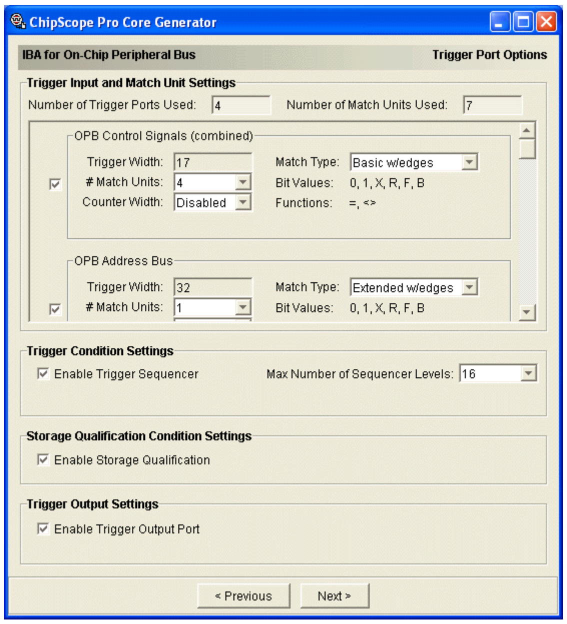
Using RPMs

The IBA/OPB core normally uses relationally placed macros (RPMs) to increase the performance of the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of RPMs by the IBA/OPB core can be disabled by deselecting the **Use RPMs** checkbox. It is recommended that the **Use RPMs** checkbox remain enabled for these device families.

Note: RPMs cannot be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

IBA/OPB Core Trigger Port Options

After you have set up the general IBA/OPB core options, click **Next**. This takes you to the third screen in the Core Generator that is used to set up the of the IBA/OPB core trigger port options (Figure 2-21).



Trigger Input and Match Unit Settings

Number of Trigger Ports Used: 4 Number of Match Units Used: 7

OPB Control Signals (combined) ☑
Trigger Width: 17 Match Type: Basic w/edges
Match Units: 4 Bit Values: 0, 1, X, R, F, B
Counter Width: Disabled Functions: =, <>

OPB Address Bus ☑
Trigger Width: 32 Match Type: Extended w/edges
Match Units: 1 Bit Values: 0, 1, X, R, F, B

Trigger Condition Settings
☑ Enable Trigger Sequencer Max Number of Sequencer Levels: 16

Storage Qualification Condition Settings
☑ Enable Storage Qualification

Trigger Output Settings
☑ Enable Trigger Output Port

Figure 2-21: **IBA/OPB Core Trigger Port Options**

Selecting the OPB Signal Groups as Trigger Ports

The OPB bus is divided into logical signal groups as described in Table 2-9, page 2-35, which spans multiple pages. You can select any of these OPB signal groups as trigger ports by selecting the checkbox to the left of each group. Note that the IBA/OPB core is limited to either 16 trigger ports or 16 total match units used, whichever limit is reached first. The protocol violation monitor is automatically checked and used as a trigger port if it is enabled on the IBA/OPB General Options panel (Figure 2-20, page 2-30).

Table 2-9: **OPB Signal Groups**

Trigger Group Name	Width	Description
OPB_CTRL	17	OPB combined control signals, including: • SYS_Rst • Debug_SYS_Rst • WDT_Rst • OPB_Rst • OPB_BE[3] • OPB_BE[2] • OPB_BE[1] • OPB_BE[0] • OPB_select • OPB_xferAck • OPB_RNW • OPB_errAck • OPB_timeout • OPB_toutSup • OPB_retry • OPB_seqAddr • OPB_busLock
OPB_ABUS	32	OPB address bus
OPB_DBUS	32	OPB combined data bus (logical OR of read and write data buses)
OPB_RDDBUS	32	OPB read data bus (from slaves)
OPB_WRDBUS	32	OPB write data bus (to slaves)

Table 2-9: **OPB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
OPB_M*n*_CTRL	11	OPB control signals for master *n*, including: • M*n*_request • OPB_MnGrant • OPB_pendReq*n* • M*n*_busLock • M*n*_BE[3] • M*n*_BE[2] • M*n*_BE[1] • M*n*_BE[0] • M*n*_select • M*n*_RNW • M*n*_seqAddr where *n* is the master number (0 to 15)
OPB_SL*m*_CTRL	4	OPB control signals slave *m*, including: • Sl*m*_xferAck • Sl*m*_errAck • Sl*m*_toutSup • Sl*m*_retry where *m* is the slave number (0 to 63)
OPB_PV	6	OPB protocol violation monitor signals
TRIG_IN	User-defined	Generic trigger input

Entering the Width of the Trigger Ports

The individual trigger ports are buses that are made up of individual signals or bits. The number of bits used to compose a trigger port is called the *trigger width*. Most of the trigger port signal groups for the IBA/OPB have predefined widths and are not user-editable. However, the width of generic trigger port can be set independently using the Trigger Width field for that group. The range of values that can be used for the generic trigger port width is 1 to 256.

Selecting the Number of Trigger Match Units

A match unit is a comparator that is connected to a trigger port and is used to detect events on that trigger port. The results of one or more match units are combined together to form the overall trigger condition event that is used to control the capturing of data. Each trigger port can be connected to 1 to 16 match units by using the # Match Units pull-down list.

Selecting one match unit conserves resources while still allowing some flexibility in detecting trigger events. Selecting two or more trigger match units allows a more flexible trigger condition equation to be a combination of multiple match units. However, increasing the number of match units per trigger port also increases the usage of logic resources accordingly.

Selecting the Match Unit Type

The different comparisons or match functions that can be performed by the trigger port match units depend on the type of the match unit. Six different types of match units are supported by the IBA/OPB cores (Table 2-10).

Table 2-10: **IBA/OPB Trigger Match Unit Types**

Type	Bit Values[a]	Match Function	Bits Per Slice[b]	Description
Basic	0, 1, X	'=', '<>'	8	Can be used for comparing data signals where transition detection is not important. This is the most bit-wise economical type of match unit.
Basic w/edges	0, 1, X, R, F, B	'=', '<>'	4	Can be used for comparing control signals where transition detection (e.g., low-to-high, high-to-low, etc.) is important.
Extended	0, 1, X	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where magnitude is important.
Extended w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where a magnitude and transition detection are important.
Range	0, 1, X	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values is important.
Range w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values and transition detection are important.

a. Bit values: '0' means "logical 0", '1' means "logical 1", 'X' means "don't care", 'R' means "0-to-1 transition", 'F' means "1-to-0 transition" and 'B' means "any transition".

b. The Bits Per Slice value is only an approximation that is used to illustrate the relative resource utilization of the different match unit types. It should not be used as a hard estimate of resource utilization.

Use the TRIG*n* Match Type pull-down list to select the type of match unit that will apply to all match units connected to the trigger port. However, as the functionality of the match unit increases, so does the amount of resources necessary to implement that functionality. This flexibility allows you to customize the functionality of the trigger module while keeping resource usage in check.

Selecting Match Unit Counter Width

The match unit counter is a configurable counter on the output of the each match unit in a trigger port. This counter can be configured at run time to count a specific number of match unit events. To include a match counter on each match unit in the trigger port, select a counter width from 1 to 32. The match counter will not be included on each match unit if the Counter Width combo box is set to Disabled. The default Counter Width setting is Disabled.

Enabling the Trigger Condition Sequencer

The trigger condition can be either a Boolean equation, or an optional trigger sequencer that is enabled by checking the **Enable Trigger Sequencer** checkbox. A block diagram of the trigger sequencer is shown in Figure 2-8, page 2-11.

The trigger sequencer is implemented as a simple cyclical state machine and can transition through up to 16 states or levels before the trigger condition is satisfied. The transition from one level to the next is caused by an event on one of the match units that is connected to the trigger sequencer. Any match unit can be selected at run time on a per level basis to transition from one level to the next. The trigger sequencer can also be configured at run time to transition from one level to the next on either contiguous or non-contiguous sequences of match function events.

Enabling the Storage Qualification Condition

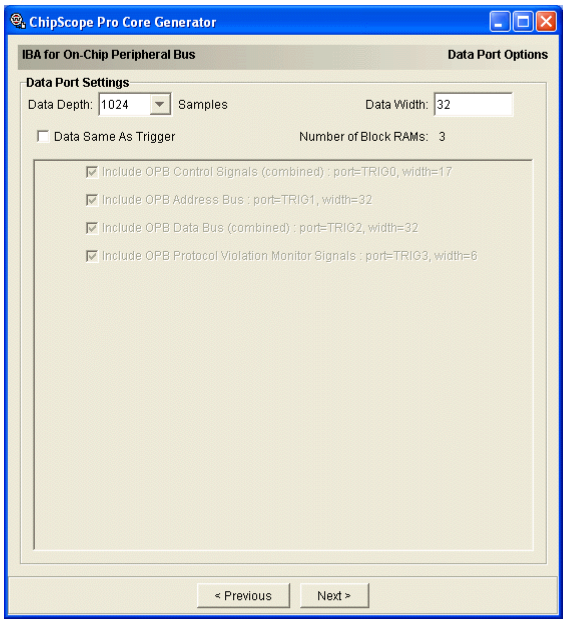
In addition to the trigger condition, the IBA/OPB core can also implement a storage qualification condition. The storage qualification condition is a Boolean combination of match function events. These match function events are detected by the match unit comparators that are subsequently attached to the trigger ports of the core. The storage qualification condition differs from the trigger condition in that it evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start the capture process and what data to capture. The storage qualification condition can be enabled by checking the **Enable Storage Qualification** checkbox.

Enabling the Trigger Output Port

The output of the IBA/OPB trigger condition module can be brought out to a port signal by checking the **Enable Trigger Output Port** checkbox. The trigger output port is can be used to trigger external test equipment by attaching the port signal to a device pin in the HDL design. The trigger output port can also be attached to other logic or ChipScope Pro cores in the design to be used as a trigger, an interrupt, or another control signal. The clock latency of the IBA/OPB trigger output port is 15 clock (`OPB_CLK`) cycles with respect to the trigger input ports.

IBA/OPB Core Data Port Options

After you have set up the IBA/OPB core trigger port options, click **Next**. This takes you to the fourth screen in the Core Generator that is used to set up the of the IBA/OPB core data port options (Figure 2-22).


ChipScope Pro Core Generator

IBA for On-Chip Peripheral Bus — Data Port Options

Data Port Settings

Data Depth: 1024 Samples — Data Width: 32

☐ Data Same As Trigger — Number of Block RAMs: 3

☑ Include OPB Control Signals (combined) : port=TRIG0, width=17

☑ Include OPB Address Bus : port=TRIG1, width=32

☑ Include OPB Data Bus (combined) : port=TRIG2, width=32

☑ Include OPB Protocol Violation Monitor Signals : port=TRIG3, width=6

Figure 2-22: **IBA/OPB Core Data Port Options**

Selecting the Data Depth

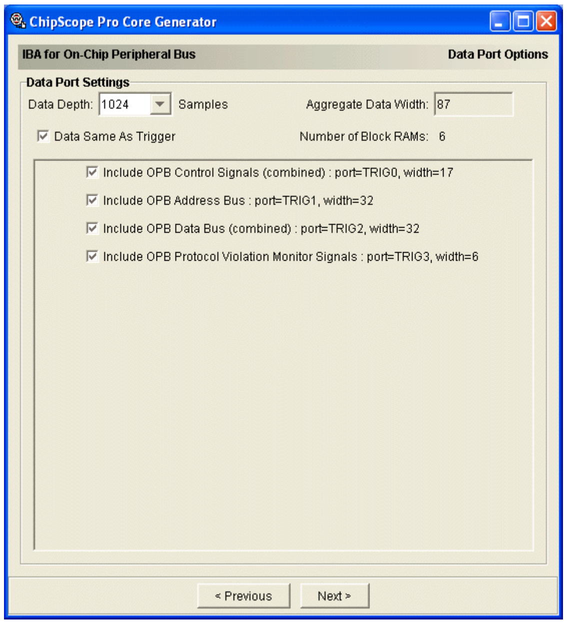
The maximum number of data sample words that the IBA/OPB core can store in the sample buffer is called the *data depth*. The data depth determines the number of data width bits contributed by each block RAM unit used by the IBA/OPB unit.

For the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families), you can set the data depth to one of six values (Table 2-2, page 2-13). For the Virtex, Virtex-E, Spartan-II, and Spartan-IIE device families (including the QPro variants of these families), you can set the data depth to one of five values (Table 2-3, page 2-13).

Selecting the Data Type

The data captured by the IBA/OPB trigger port can come from two different source types:

- **Data Same as Trigger (Figure 2-23)**
  - The data and trigger ports are identical. This mode is very common in most logic analyzers, since you can capture and collect any data that is used to trigger the core.
  - Individual trigger ports can be selected to be included in the data port. If this selection is made, then the DATA input port will not be included in the port map of the IBA/OPB core.
  - This mode conserves CLB and routing resources in the IBA/OPB core, but is limited to a maximum aggregate data sample word width of 256 bits.
- **Data Separate from Trigger (Figure 2-22, page 2-39)**
  - The data port is completely independent of the trigger ports.
  - This mode is useful when you want to limit the amount of data being captured.



Data Port Settings

Data Depth: 1024 Samples

Aggregate Data Width: 87

☑ Data Same As Trigger

Number of Block RAMs: 6

☑ Include OPB Control Signals (combined) : port=TRIG0, width=17

☑ Include OPB Address Bus : port=TRIG1, width=32

☑ Include OPB Data Bus (combined) : port=TRIG2, width=32

☑ Include OPB Protocol Violation Monitor Signals : port=TRIG3, width=6

Figure 2-23: **IBA/OPB Core Data Same As Trigger Options**

Entering the Data Width

The width of each data sample word stored by the IBA/OPB core is called the *data width*. If the data and trigger words are independent from each other, then the maximum allowable data width depends on the target device type and data depth. However, regardless of these factors, the maximum allowable data width is 256 bits.

Selecting the Data-Same-As-Trigger Ports

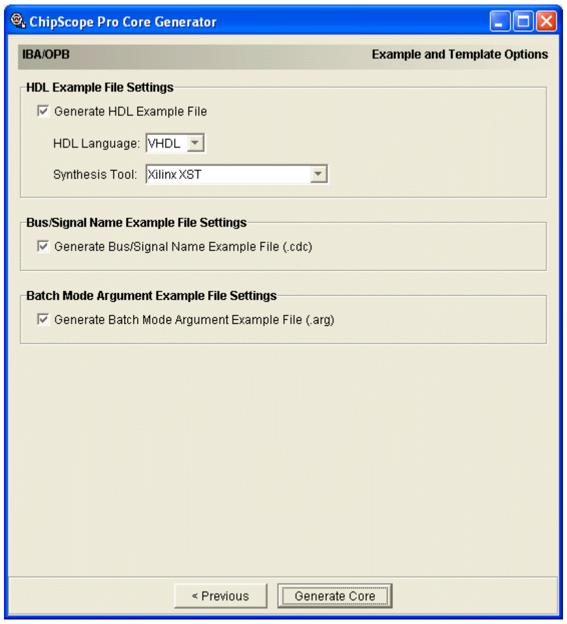
If the **Data Same As Trigger** checkbox is selected, then a checkbox for each `TRIGn` port appears in the data port options screen. These checkboxes should be used to select the individual trigger ports that will be included in the aggregate data port. Note that selecting the individual trigger ports automatically updates the Aggregate Data Width field accordingly. A maximum data width of 256 bits applies to the aggregate selection of trigger ports.

Number of Block RAMs

As the data depth and data width selections are changed, the Number of Block RAMs field notifies you of how many block RAMs will be used by the IBA/OPB core. The trigger mark is automatically taken into account when calculating this value.

Creating Example Templates

After selecting the parameters for the IBA/OPB core, click **Next** to view the Example and Template Options (Figure 2-24).


ChipScope Pro Core Generator

IBA/OPB — Example and Template Options

HDL Example File Settings

[✓] Generate HDL Example File

HDL Language: VHDL

Synthesis Tool: Xilinx XST

Bus/Signal Name Example File Settings

[✓] Generate Bus/Signal Name Example File (.cdc)

Batch Mode Argument Example File Settings

[✓] Generate Batch Mode Argument Example File (.arg)

Figure 2-24: **IBA/OPB Core Example and Template Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the IBA/OPB core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

Bus/Signal Name Example Files (.CDC)

The bus/signal name example file for the IBA/OPB core (for example, `iba_opb.cdc`) contains generic information about the trigger and data ports of the IBA/OPB core. The `iba_opb.cdc` file will be created if you select the **Generate Bus/Signal Name Example File (.cdc)** checkbox. You can use the `iba_opb.cdc` file as a template to change trigger and/or data port signal names, create buses, and so on. The modified `iba_opb.cdc` file can then be imported into the ChipScope Pro Analyzer tool and applied to the appropriate IBA/OPB core by using the **File** → **Import** option.

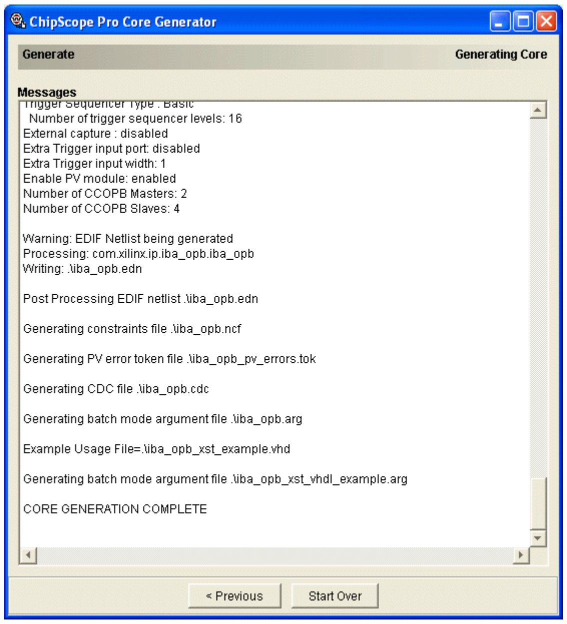
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `iba_opb.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `iba_opb.arg` file is used with the command line program called "**generate.exe**". The `iba_opb.arg` file contains all of the arguments necessary for generating the IBA/OPB core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An IBA/OPB core can be generated by running "**generate.exe iba_opb -f=iba_opb.arg**" at the command prompt on Windows systems or by running "**generate.sh iba_opb -f=iba_opb.arg**" at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the IBA/OPB core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the CORE GENERATION COMPLETE message signals the end of the process (Figure 2-25). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
Trigger Sequencer Type : Basic
 Number of trigger sequencer levels: 16
External capture : disabled
Extra Trigger input port: disabled
Extra Trigger input width: 1
Enable PV module: enabled
Number of CCOPB Masters: 2
Number of CCOPB Slaves: 4

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.iba_opb.iba_opb
Writing: .\iba_opb.edn

Post Processing EDIF netlist .\iba_opb.edn

Generating constraints file .\iba_opb.ncf

Generating PV error token file .\iba_opb_pv_errors.tok

Generating CDC file .\iba_opb.cdc

Generating batch mode argument file .\iba_opb.arg

Example Usage File=.\iba_opb_xst_example.vhd

Generating batch mode argument file .\iba_opb_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-25: **IBA/OPB Core Generation Complete**

Using the IBA/OPB Core

To instantiate the example IBA/OPB core HDL files into your design, use the following guidelines to connect the IBA/OPB core port signals to various signals in your design:

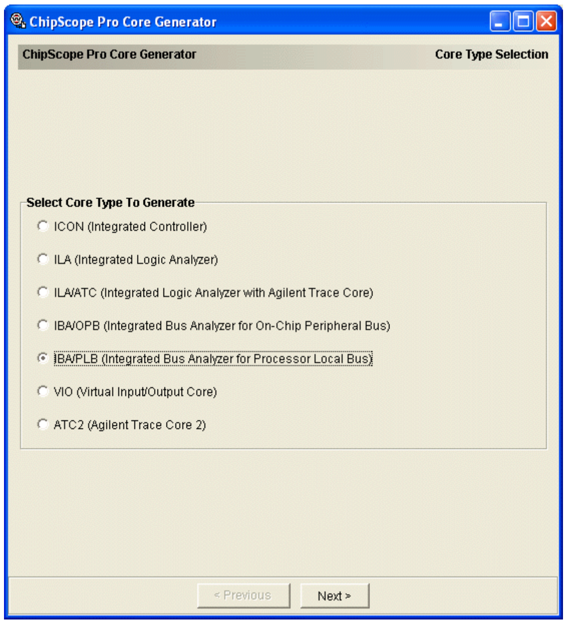
- Connect the IBA/OPB core's CONTROL port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the IBA/OPB core's data, trigger, and OPB port signals to "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- Make sure the data and trigger source signals are synchronous to the IBA/OPB clock signal (`OPB_CLK`)

Generating the IBA/PLB Core

The ChipScope Pro Core Generator allows you to define and generate a customized IBA core for monitoring and debugging CoreConnect PLB buses in your HDL designs. You can customize the number of PLB masters and slaves as well as the types of PLB signals that you want to use as triggers to your IBA/PLB core. You can also customize the maximum number of data samples stored by the IBA/PLB core and the width of the data samples (if different from the trigger ports).

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), a signal import file (`*.cdc`) and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select **IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)**, and click **Next** (Figure 2-26).

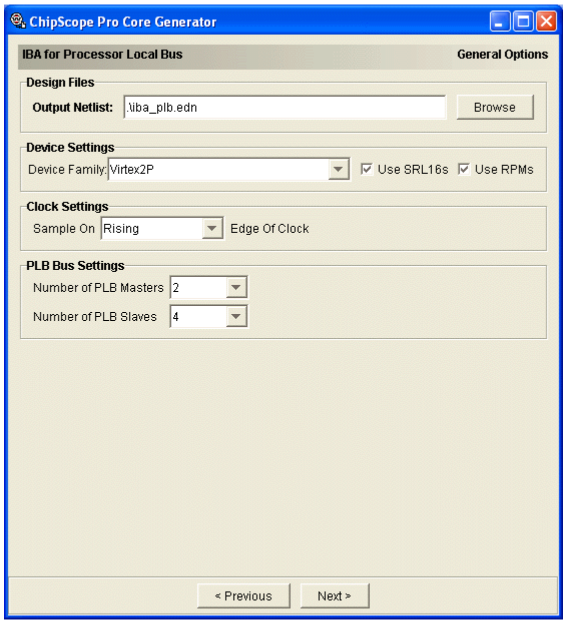


Select Core Type To Generate
- ICON (Integrated Controller)
- ILA (Integrated Logic Analyzer)
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- [✓ IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)]
- VIO (Virtual Input/Output Core)
- ATC2 (Agilent Trace Core 2)

Figure 2-26: **Selecting the IBA/PLB Core**

General IBA/PLB Core Options

The second screen in the Core Generator is used to set up the of the general IBA/PLB core options (Figure 2-27).



Design Files
Output Netlist: .\iba_plb.edn
Device Settings
Device Family: Virtex2P
[✓] Use SRL16s [✓] Use RPMs
Clock Settings
Sample On Rising Edge Of Clock
PLB Bus Settings
Number of PLB Masters 2
Number of PLB Slaves 4

Figure 2-27: **IBA/PLB Core General Options**

Choosing the File Destination

The destination for the IBA/PLB EDIF netlist (`iba_plb.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the IBA/PLB core is optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II Pro is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Selecting the Clock Edge

The IBA/PLB unit can use either the rising or falling edges of the CLK signal to trigger and capture data. The Clock Settings pull-down list is used to select either the rising or falling edge of the `CLK` signal as the clock source for the IBA/PLB core.

PLB Bus Settings

The IBA/PLB core is designed to passively attach to the PLB bus/arbiter component in your embedded PowerPC or MicroBlaze processor design. Like the PLB bus and arbiter components, the IBA/PLB core is capable of handling up to 16 PLB masters and 16 PLB slaves.

Note: It is very important to specify the same number of PLB masters and slaves that you have on the PLB bus of your embedded processor design, otherwise the IBA/PLB core may not function properly.

Using SRL16s

The IBA/PLB core normally uses the SRL16 feature of the FPGA device to increase performance and decrease the area used by the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of SRL16s by the IBA/PLB core can be disabled by deselecting the **Use SRL16s** checkbox. It is recommended that the **Use SRL16s** checkbox remain enabled.

Note: SRL16s must be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

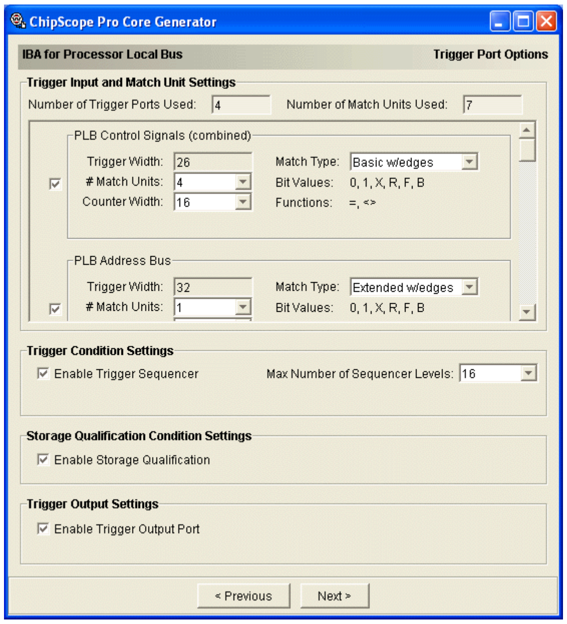
Using RPMs

The IBA/PLB core normally uses relationally placed macros (RPMs) to increase the performance of the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of RPMs by the IBA/PLB core can be disabled by deselecting the **Use RPMs** checkbox. It is recommended that the **Use RPMs** checkbox remain enabled for these device families.

Note: RPMs cannot be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

IBA/PLB Core Trigger Port Options

After you have set up the general IBA/PLB core options, click **Next**. This takes you to the third screen in the Core Generator that is used to set up the of the IBA/PLB core trigger port options (Figure 2-28).



Trigger Input and Match Unit Settings

Number of Trigger Ports Used: 4
Number of Match Units Used: 7

PLB Control Signals (combined) [✓]
Trigger Width: 26
Match Units: 4
Counter Width: 16
Match Type: Basic w/edges
Bit Values: 0, 1, X, R, F, B
Functions: =, <>

PLB Address Bus [✓]
Trigger Width: 32
Match Units: 1
Match Type: Extended w/edges
Bit Values: 0, 1, X, R, F, B

Trigger Condition Settings

[✓] Enable Trigger Sequencer
Max Number of Sequencer Levels: 16

Storage Qualification Condition Settings

[✓] Enable Storage Qualification

Trigger Output Settings

[✓] Enable Trigger Output Port

Figure 2-28: **IBA/PLB Core Trigger Port Options**

Selecting the PLB Signal Groups as Trigger Ports

The PLB bus is divided into logical signal groups as described in Table 2-11, which spans multiple pages. You can select any of these PLB signal groups as trigger ports by selecting the checkbox to the left of each group. Note that the IBA/PLB core is limited to either 16 trigger ports or 16 total match units used, whichever limit is reached first.

Table 2-11: **PLB Signal Groups**

Trigger Group Name	Width	Description
`PLB_CTRL`	26	PLB combined control signals, including: • SYS_plbReset • PLB_abort • PLB_BE(0) • PLB_BE(1) • PLB_BE(2) • PLB_BE(3) • PLB_BE(4) • PLB_BE(5) • PLB_BE(6) • PLB_BE(7) • PLB_busLock • PLB_masterID(0) • PLB_masterID(1) • PLB_masterID(2) • PLB_masterID(3) • PLB_Msize(0) • PLB_Msize(1) • PLB_PAValid • PLB_SAValid • PLB_rdPrim • PLB_RNW • PLB_size(0) • PLB_size(1) • PLB_size(2) • PLB_size(3) • PLB_wrPrim
`PLB_ABUS`	32	PLB address bus
`PLB_RDDBUS`	64	PLB read data bus (from slaves)
`PLB_WRDBUS`	64	PLB write data bus (to slaves)

Table 2-11: **PLB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
`PLB_Mn_CTRL`	32	PLB control signals for master *n*, including: • PLB_M*n*AddrAck • PLB_M*n*_Busy • PLB_M*n*_Err • PLB_M*n*RdDAck • PLB_M*n*RdWdAddr(0) • PLB_M*n*RdWdAddr(1) • PLB_M*n*RdWdAddr(2) • PLB_M*n*RdWdAddr(3) • PLB_M*n*Rearbitrate • PLB_M*n*SSize(0) • PLB_M*n*SSize(1) • PLB_M*n*_WrDAck • M*n*_abort • M*n*_BE(0) • M*n*_BE(1) • M*n*_BE(2) • M*n*_BE(3) • M*n*_BE(4) • M*n*_BE(5) • M*n*_BE(6) • M*n*_BE(7) • M*n*_busLock • M*n*_MSize(0) • M*n*_MSize(1) • M*n*_priority(0) • M*n*_priority(1) • M*n*_request • M*n*_RNW • M*n*_size(0) • M*n*_size(1) • M*n*_size(2) • M*n*_size(3) where *n* is the master number (0 to 15)

Table 2-11: **PLB Signal Groups *(Continued)***

Trigger Group Name	Width	Description
`PLB_SLm_CTRL`	12	PLB control signals slave *m*, including: • Sl*m*_addrAck • Sl*m*_rdDAck • Sl*m*_rdWdAddr(0) • Sl*m*_rdWdAddr(1) • Sl*m*_rdWdAddr(2) • Sl*m*_rdWdAddr(3) • Sl*m*_rearbitrate • Sl*m*_SSize(0) • Sl*m*_SSize(1) • Sl*m*_wait • Sl*m*_wrComp • Sl*m*_wrDAck where *m* is the slave number (0 to 15)
`TRIG_IN`	User-defined	Generic trigger input

Entering the Width of the Trigger Ports

The individual trigger ports are buses that are made up of individual signals or bits. The number of bits used to compose a trigger port is called the *trigger width*. Most of the trigger port signal groups for the IBA/PLB have predefined widths and are not user-editable. However, the width of generic trigger port can be set independently using the Trigger Width field for that group. The range of values that can be used for the generic trigger port width is 1 to 256.

Selecting the Number of Trigger Match Units

A match unit is a comparator that is connected to a trigger port and is used to detect events on that trigger port. The results of one or more match units are combined together to form what is called the overall trigger condition event that is used to control the capturing of data. Each trigger port can be connected to 1 to 16 match units by using the # Match Units pull-down list.

Selecting one match unit conserves resources while still allowing some flexibility in detecting trigger events. Selecting two or more trigger match units allows a more flexible trigger condition equation to be a combination of multiple match units. However, increasing the number of match units per trigger port also increases the usage of logic resources accordingly.

Selecting the Match Unit Type

The different comparisons or match functions that can be performed by the trigger port match units depend on the type of the match unit. Six different types of match units are supported by the IBA/PLB cores (Table 2-12).

Table 2-12: **IBA/PLB Trigger Match Unit Types**

Type	Bit Values[a]	Match Function	Bits Per Slice[b]	Description
Basic	0, 1, X	'=', '<>'	8	Can be used for comparing data signals where transition detection is not important. This is the most bit-wise economical type of match unit.
Basic w/edges	0, 1, X, R, F, B	'=', '<>'	4	Can be used for comparing control signals where transition detection (e.g., low-to-high, high-to-low, etc.) is important.
Extended	0, 1, X	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where magnitude is important.
Extended w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where a magnitude and transition detection are important.
Range	0, 1, X	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values is important.
Range w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values and transition detection are important.

a. Bit values: '0' means "logical 0", '1' means "logical 1", 'X' means "don't care", 'R' means "0-to-1 transition", 'F' means "1-to-0 transition" and 'B' means "any transition".

b. The Bits Per Slice value is only an approximation that is used to illustrate the relative resource utilization of the different match unit types. It should not be used as a hard estimate of resource utilization.

Use the TRIG*n* Match Type pull-down list to select the type of match unit that will apply to all match units connected to the trigger port. However, as the functionality of the match unit increases, so does the amount of resources necessary to implement that functionality. This flexibility allows you to customize the functionality of the trigger module while keeping resource usage in check.

Selecting Match Unit Counter Width

The match unit counter is a configurable counter on the output of the each match unit in a trigger port. This counter can be configured at run time to count a specific number of match unit events. To include a match counter on each match unit in the trigger port, select a counter width from 1 to 32. The match counter will not be included on each match unit if the Counter Width combo box is set to Disabled. The default Counter Width setting is Disabled.

Enabling the Trigger Condition Sequencer

The trigger condition can be either a Boolean equation, or an optional trigger sequencer that is enabled by checking the **Enable Trigger Sequencer** checkbox. A block diagram of the trigger sequencer is shown in Figure 2-8, page 2-11.

The trigger sequencer is implemented as a simple cyclical state machine and can transition through up to 16 states or levels before the trigger condition is satisfied. The transition from one level to the next is caused by an event on one of the match units that is connected to the trigger sequencer. Any match unit can be selected at run time on a per level basis to transition from one level to the next. The trigger sequencer can also be configured at run time to transition from one level to the next on either contiguous or non-contiguous sequences of match function events.

Enabling the Storage Qualification Condition

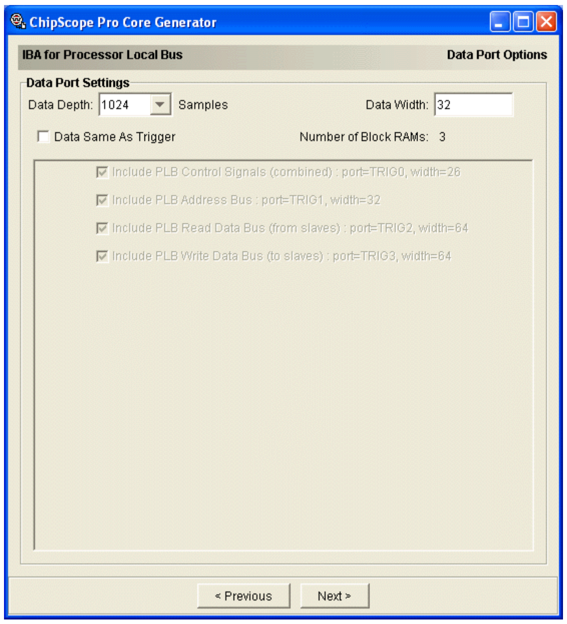
In addition to the trigger condition, the IBA/PLB core can also implement a storage qualification condition. The storage qualification condition is a Boolean combination of events that are detected by the match unit comparators that are subsequently attached to the trigger ports of the core. The storage qualification condition differs from the trigger condition in that it evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start the capture process and what data to capture. The storage qualification condition can be enabled by checking the **Enable Storage Qualification** checkbox.

Enabling the Trigger Output Port

The output of the IBA/PLB trigger condition module can be brought out to a port signal by checking the **Enable Trigger Output Port** checkbox. The trigger output port is can be used to trigger external test equipment by attaching the port signal to a device pin in the HDL design. The trigger output port can also be attached to other logic or ChipScope Pro cores in the design to be used as a trigger, an interrupt, or another control signal. The clock latency of the IBA/PLB trigger output port is 15 clock (`PLB_CLK`) cycles with respect to the trigger input ports.

IBA/PLB Core Data Port Options

After you have set up the IBA/PLB core trigger port options, click **Next**. This takes you to the fourth screen in the Core Generator that is used to set up the of the IBA/PLB core data port options (Figure 2-22).



Data Depth: 1024 Samples
Data Width: 32
Data Same As Trigger
Number of Block RAMs: 3
Include PLB Control Signals (combined) : port=TRIG0, width=26
Include PLB Address Bus : port=TRIG1, width=32
Include PLB Read Data Bus (from slaves) : port=TRIG2, width=64
Include PLB Write Data Bus (to slaves) : port=TRIG3, width=64

Figure 2-29: **IBA/PLB Core Data Port Options**

Selecting the Data Depth

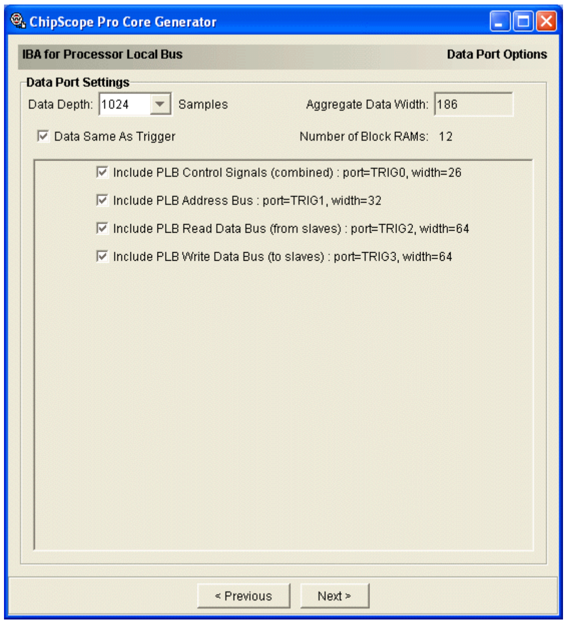
The maximum number of data sample words that the IBA/PLB core can store in the sample buffer is called the *data depth*. The data depth determines the number of data width bits contributed by each block RAM unit used by the IBA/PLB unit.

For the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families), you can set the data depth to one of six values (Table 2-2, page 2-13). For the Virtex, Virtex-E, Spartan-II and Spartan-IIE device families (including the QPro variants of these families), you can set the data depth to one of five values (Table 2-3, page 2-13).

Selecting the Data Type

The data captured by the IBA/PLB trigger port can come from two different source types:

- **Data Same as Trigger (Figure 2-30, page 2-54)**
  - The data and trigger ports are identical. This mode is very common in most logic analyzers, since you can capture and collect any data that is used to trigger the core.
  - Individual trigger ports can be selected to be included in the data port. If this selection is made, then the DATA input port will not be included in the port map of the IBA/PLB core.
  - This mode conserves CLB and routing resources in the IBA/PLB core, but is limited to a maximum aggregate data sample word width of 256 bits.
- **Data Separate from Trigger (Figure 2-29, page 2-53)**
  - The data port is completely independent of the trigger ports.
  - This mode is useful when you want to limit the amount of data being captured.



Data Port Settings

Data Depth: 1024 Samples

Aggregate Data Width: 186

☑ Data Same As Trigger

Number of Block RAMs: 12

☑ Include PLB Control Signals (combined) : port=TRIG0, width=26

☑ Include PLB Address Bus : port=TRIG1, width=32

☑ Include PLB Read Data Bus (from slaves) : port=TRIG2, width=64

☑ Include PLB Write Data Bus (to slaves) : port=TRIG3, width=64

Figure 2-30: **IBA/PLB Core Data Same As Trigger Options**

Entering the Data Width

The width of each data sample word stored by the IBA/PLB core is called the *data width*. If the data and trigger words are independent from each other, then the maximum allowable data width depends on the target device type and data depth. However, regardless of these factors, the maximum allowable data width is 256 bits.

Selecting the Data-Same-As-Trigger Ports

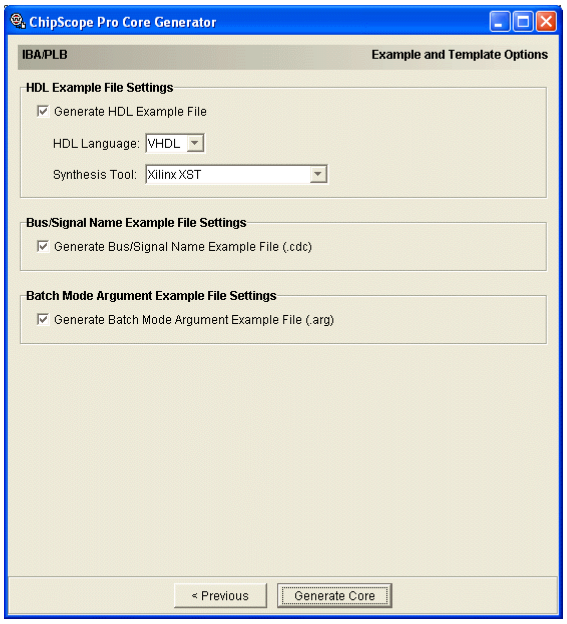
If the **Data Same As Trigger** checkbox is selected, then a checkbox for each `TRIGn` port appears in the data port options screen. These checkboxes should be used to select the individual trigger ports that will be included in the aggregate data port. Note that selecting the individual trigger ports automatically updates the Aggregate Data Width field accordingly. A maximum data width of 256 bits applies to the aggregate selection of trigger ports.

Number of Block RAMs

As the data depth and data width selections are changed, the Number of Block RAMs field notifies you of how many block RAMs will be used by the IBA/PLB core. The trigger mark is automatically taken into account when calculating this value.

Creating Example Templates

After selecting the parameters for the IBA/PLB core, click **Next** to view the Example and Template Options (Figure 2-31).



HDL Example File Settings

☑ Generate HDL Example File

HDL Language: VHDL

Synthesis Tool: Xilinx XST

Bus/Signal Name Example File Settings

☑ Generate Bus/Signal Name Example File (.cdc)

Batch Mode Argument Example File Settings

☑ Generate Batch Mode Argument Example File (.arg)

Figure 2-31: **IBA/PLB Core Example and Template Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the IBA/PLB core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

Bus/Signal Name Example Files (.CDC)

The bus/signal name example file for the IBA/PLB core (for example, `iba_plb.cdc`) contains generic information about the trigger and data ports of the IBA/PLB core. The `iba_plb.cdc` file will be created if you select the **Generate Bus/Signal Name Example File (.cdc)** checkbox. You can use the `iba_plb.cdc` file as a template to change trigger and/or data port signal names, create buses, and so on. The modified `iba_plb.cdc` file can then be imported into the ChipScope Pro Analyzer tool and applied to the appropriate IBA/PLB core by using the **File** → **Import** option.

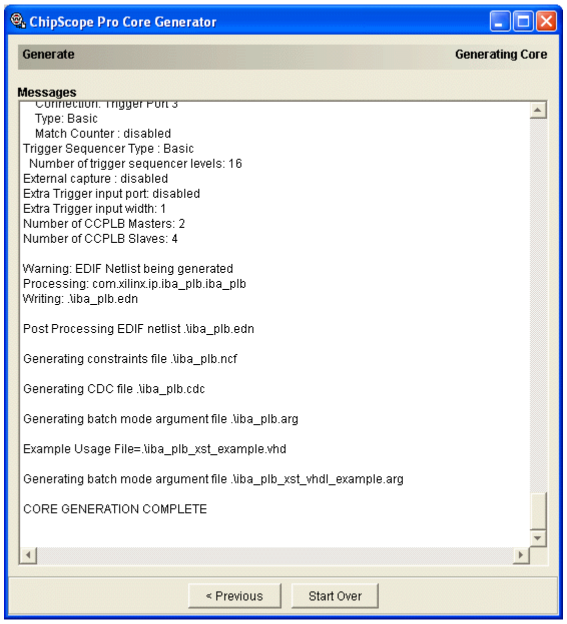
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `iba_plb.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `iba_plb.arg` file is used with the command line program called “**generate.exe**”. The `iba_plb.arg` file contains all of the arguments necessary for generating the IBA/PLB core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An IBA/PLB core can be generated by running “**generate.exe iba_plb -f=iba_plb.arg**” at the command prompt on Windows systems or by running “**generate.sh iba_plb -f=iba_plb.arg**” at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the IBA/PLB core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the CORE GENERATION COMPLETE message signals the end of the process (Figure 2-32). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
   Connection: Trigger Port 3
   Type: Basic
   Match Counter : disabled
Trigger Sequencer Type : Basic
 Number of trigger sequencer levels: 16
External capture : disabled
Extra Trigger input port: disabled
Extra Trigger input width: 1
Number of CCPLB Masters: 2
Number of CCPLB Slaves: 4

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.iba_plb.iba_plb
Writing: .\iba_plb.edn

Post Processing EDIF netlist .\iba_plb.edn

Generating constraints file .\iba_plb.ncf

Generating CDC file .\iba_plb.cdc

Generating batch mode argument file .\iba_plb.arg

Example Usage File=.\iba_plb_xst_example.vhd

Generating batch mode argument file .\iba_plb_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-32: **IBA/PLB Core Generation Complete**

Using the IBA/PLB Core

To instantiate the example IBA/PLB core HDL files into your design, use the following guidelines to connect the IBA/PLB core port signals to various signals in your design:

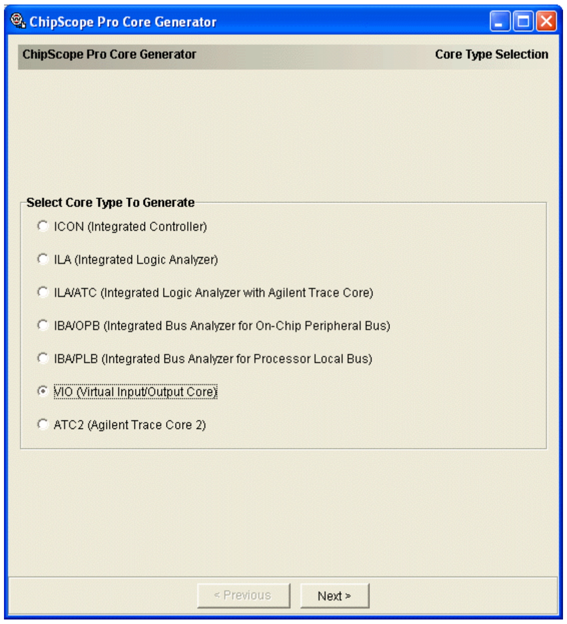
- Connect the IBA/PLB core's CONTROL port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the IBA/PLB core's data, trigger, and PLB port signals to "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- Make sure the data and trigger source signals are synchronous to the IBA/PLB clock signal (PLB_CLK)

Generating the VIO Core

The ChipScope Pro Core Generator allows you to define and generate a customized VIO core for adding virtual inputs and outputs to your HDL designs. You can customize the virtual inputs and outputs to be synchronous to a particular clock in your design or to be completely asynchronous with respect to any clock domain in your design. You can also customize the number of input and output signals used by the VIO core.

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), a signal import file (`*.cdc`) and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select, **VIO (Virtual Input/Output Core)** and click **Next** (Figure 2-33).



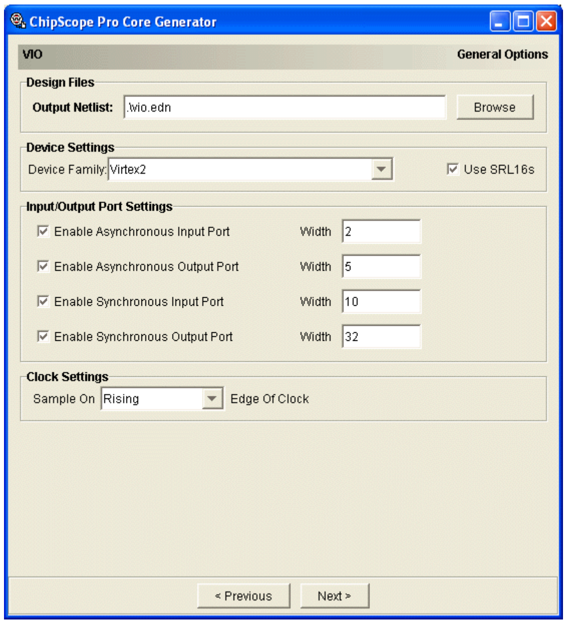
Select Core Type To Generate

- ICON (Integrated Controller)
- ILA (Integrated Logic Analyzer)
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- [✓ VIO (Virtual Input/Output Core)]
- ATC2 (Agilent Trace Core 2)

Figure 2-33: **Selecting the VIO Core**

General VIO Core Options

The second screen in the Core Generator is used to set up the of the general VIO core options (Figure 2-34).



Design Files
Output Netlist: .\vio.edn

Device Settings
Device Family: Virtex2
☑ Use SRL16s

Input/Output Port Settings
☑ Enable Asynchronous Input Port Width 2
☑ Enable Asynchronous Output Port Width 5
☑ Enable Synchronous Input Port Width 10
☑ Enable Synchronous Output Port Width 32

Clock Settings
Sample On Rising Edge Of Clock

Figure 2-34: **VIO Core General Options**

Choosing the File Destination

The destination for the VIO EDIF netlist (`vio.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the VIO core is optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture. The ChipScope Pro Core Generator supports the Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, and Spartan-3 device families (including the QPro variants of these families). Virtex-II is the default target device family.

Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Using SRL16s

The VIO core normally uses the SRL16 feature of the FPGA device to increase performance and decrease the area used by the core. If the device family is Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 (including the QPro variants of these families), the usage of SRL16s by the VIO core can be disabled by deselecting the **Use SRL16s** checkbox. It is recommended that the **Use SRL16s** checkbox remain enabled.

Note: SRL16s must be used with the Virtex, Virtex-E, Spartan-II, or Spartan-IIE device families (including the QPro variants of these families).

Using Asynchronous Input Signals

The VIO core will include asynchronous inputs when the **Enable Asynchronous Input Signals** checkbox is enabled. When enabled, you can specify that up to 256 asynchronous input signals should be used by entering a value in the Width text field. Asynchronous input signals are inputs to the VIO core and can be used as outputs from your design, regardless of the clock domain.

Using Asynchronous Output Signals

The VIO core will include asynchronous outputs when the **Enable Asynchronous Output Signals** checkbox is enabled. When enabled, you can specify that up to 256 asynchronous output signals should be used by entering a value in the Width text field. Asynchronous output signals are outputs from the VIO core and can be used as inputs to your design, regardless of the clock domain.

Using Synchronous Inputs

The VIO core will include synchronous inputs when the **Enable Synchronous Input Signals** checkbox is enabled. When enabled, you can specify that up to 256 synchronous input signals should be used by entering a value in the Width text field. Synchronous input signals are inputs to the VIO core and can be used as outputs from your design, as long as those design signals are synchronous to the CLK signal of the VIO core.

Using Synchronous Outputs

The VIO core will include synchronous outputs when the **Enable Synchronous Output Signals** checkbox is enabled. When enabled, you can specify that up to 256 synchronous output signals should be used by entering a value in the Width text field. Synchronous output signals are outputs from the VIO core and can be used as inputs to your design, as long as those design signals are synchronous to the CLK signal of the VIO core.

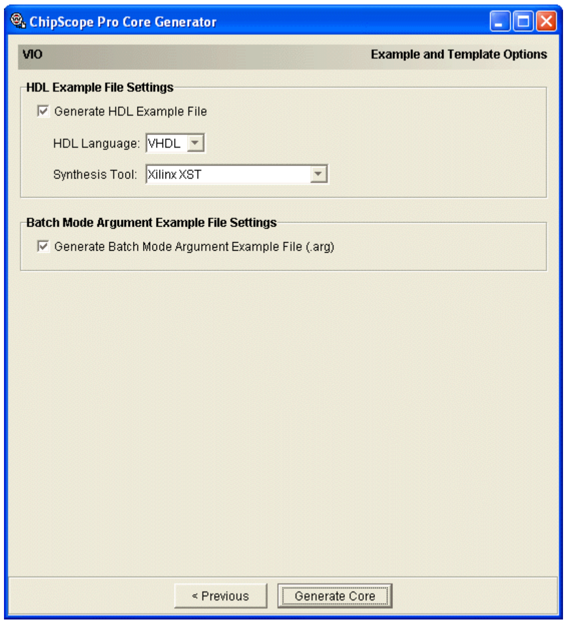
Selecting the Clock Edge

The VIO unit can use either the rising or falling edges of the CLK signal to capture and generate data on the synchronous input and output signals, respectively. The Clock Settings pull-down list is used to select either the rising or falling edge of the CLK signal as the clock source for the VIO core.

Note: The clock edge can only be selected if synchronous inputs and/or outputs are used.

Creating Example Templates

After selecting the parameters for the VIO core, click **Next** to view the Example and Template Generation Options (Figure 2-35).



HDL Example File Settings

☑ Generate HDL Example File

HDL Language: VHDL

Synthesis Tool: Xilinx XST

Batch Mode Argument Example File Settings

☑ Generate Batch Mode Argument Example File (.arg)

Figure 2-35: **VIO Core Example Template Generation Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the VIO core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

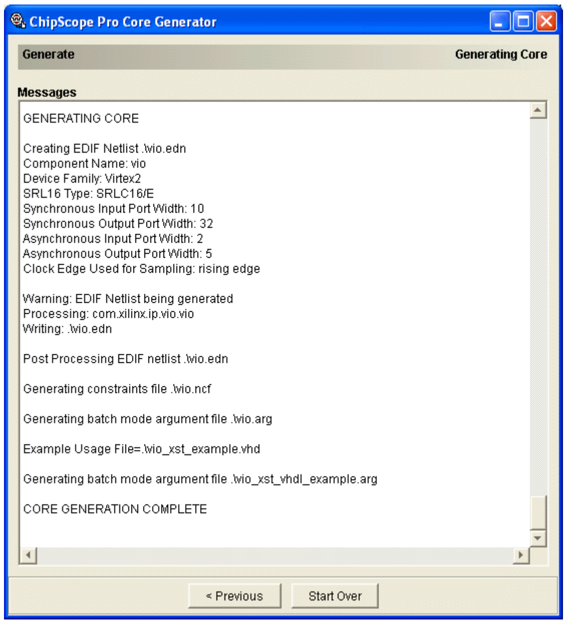
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `vio.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The vio.arg file is used with the command line program called **"generate.exe"**. The `vio.arg` file contains all of the arguments necessary for generating the VIO core without having to use the ChipScope Pro Core Generator GUI tool.

Note: A VIO core can be generated by running **"generate.exe vio-f=vio.arg"** at the command prompt on Windows systems or by running **"generate.sh vio -f=vio.arg"** at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the VIO core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the `CORE GENERATION COMPLETE` message signals the end of the process (Figure 2-36). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
GENERATING CORE

Creating EDIF Netlist .\vio.edn
Component Name: vio
Device Family: Virtex2
SRL16 Type: SRLC16/E
Synchronous Input Port Width: 10
Synchronous Output Port Width: 32
Asynchronous Input Port Width: 2
Asynchronous Output Port Width: 5
Clock Edge Used for Sampling: rising edge

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.vio.vio
Writing: .\vio.edn

Post Processing EDIF netlist .\vio.edn

Generating constraints file .\vio.ncf

Generating batch mode argument file .\vio.arg

Example Usage File=.\vio_xst_example.vhd

Generating batch mode argument file .\vio_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-36: **VIO Core Generation Complete**

Using the VIO Core

To instantiate the example VIO core HDL files into your design, use the following guidelines to connect the VIO core port signals to various signals in your design:

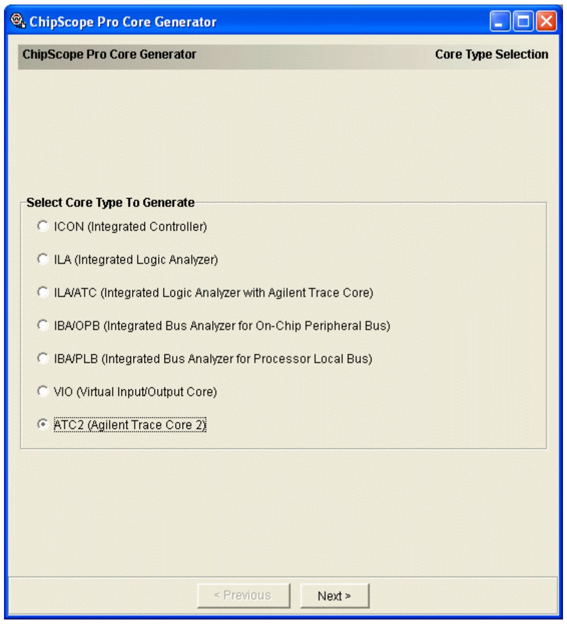
- Connect the VIO core's CONTROL port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the VIO core's asynchronous and synchronous input signals to a "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- For best results, make sure the synchronous input source signals are synchronous to the VIO clock signal (CLK); also make sure the synchronous output sink signals are synchronous to the VIO clock signal (CLK)

Generating the ATC2 Core

The ChipScope Pro Core Generator allows you to define and generate a customized ATC2 core for adding external Agilent logic analyzer capture capabilities to your HDL designs. You can customize the number of pins (and their characteristics) to be used for external capture as well as how many input data ports you need. You can also customize the type of capture mode (state or timing) to be used as well as the TDM compression mode (1x or 2x).

After the Core Generator validates the user-defined parameters, it generates an EDIF netlist (`*.edn`), a netlist constraint file (`*.ncf`), a signal import file (`*.cdc`) and example code specific to the synthesis tool used. You can easily generate the netlist and code examples for use in normal FPGA design flows.

The first screen in the Core Generator offers the choice to generate either an ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, or ATC2 core. Select, **ATC2 (Agilent Trace Core 2)** and click **Next** (Figure 2-37).



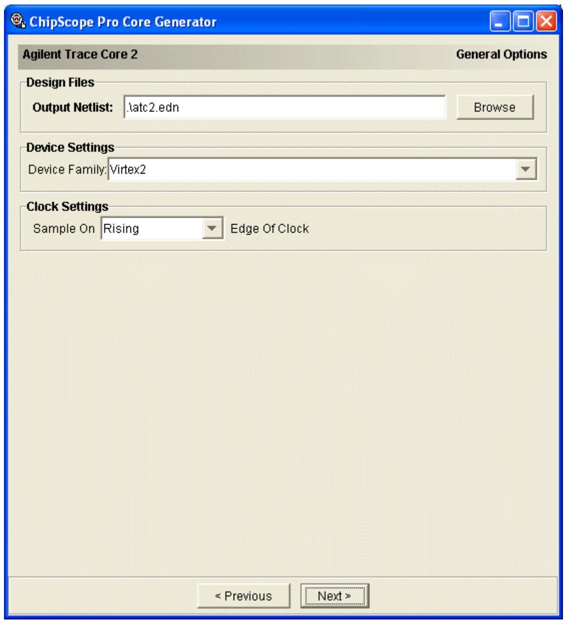
Select Core Type To Generate

- ICON (Integrated Controller)
- ILA (Integrated Logic Analyzer)
- ILA/ATC (Integrated Logic Analyzer with Agilent Trace Core)
- IBA/OPB (Integrated Bus Analyzer for On-Chip Peripheral Bus)
- IBA/PLB (Integrated Bus Analyzer for Processor Local Bus)
- VIO (Virtual Input/Output Core)
- [✓ ATC2 (Agilent Trace Core 2)]

Figure 2-37: **Selecting the ATC2 Core**

General ATC2 Core Options

The second screen in the Core Generator is used to set up the of the general ATC2 core options (Figure 2-38).



Design Files
Output Netlist: .\atc2.edn

Device Settings
Device Family: Virtex2

Clock Settings
Sample On Rising Edge Of Clock

Figure 2-38: **ATC2 Core General Options**

Choosing the File Destination

The destination for the ATC2 EDIF netlist (`atc2.edn`) is displayed in the Output Netlist field. The default directory is the Core Generator install path. To change it, you can either type a new path in the field, or choose **Browse** to navigate to a new destination.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the ATC2 core is optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture. For the ATC2 core, the ChipScope Pro Core Generator only supports the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families). Virtex-II is the default target device family.

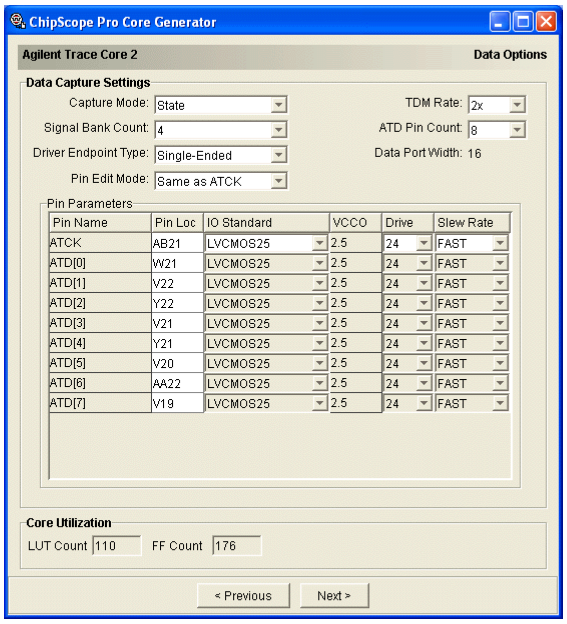
Note: Cores generated for Virtex-II, Virtex-II Pro, Virtex-4, or Spartan-3 devices do not work for Virtex, Virtex-E, Spartan-II, or Spartan-IIE devices.

Selecting the Clock Edge

The ATC2 core can use either the rising or falling edges of the CLK signal to capture data on the input data port signals. The CLK port is also used as a clock source for internal logic that performs data pin calibration. The Clock Settings pull-down list is used to select either the rising or falling edge of the CLK signal as the clock source for the ATC2 core.

ATC2 Core Data Options

After you have set up the general ATC2 core options, click **Next**. This takes you to the third screen in the Core Generator that is used to set up the of the ATC2 core data capture options (Figure 2-39).



Data Capture Settings

Capture Mode: State
Signal Bank Count: 4
Driver Endpoint Type: Single-Ended
Pin Edit Mode: Same as ATCK
TDM Rate: 2x
ATD Pin Count: 8
Data Port Width: 16

Pin Parameters

Pin Name	Pin Loc	IO Standard	VCCO	Drive	Slew Rate
ATCK	AB21	LVCMOS25	2.5	24	FAST
ATD[0]	W21	LVCMOS25	2.5	24	FAST
ATD[1]	V22	LVCMOS25	2.5	24	FAST
ATD[2]	Y22	LVCMOS25	2.5	24	FAST
ATD[3]	V21	LVCMOS25	2.5	24	FAST
ATD[4]	Y21	LVCMOS25	2.5	24	FAST
ATD[5]	V20	LVCMOS25	2.5	24	FAST
ATD[6]	AA22	LVCMOS25	2.5	24	FAST
ATD[7]	V19	LVCMOS25	2.5	24	FAST

Core Utilization

LUT Count 110
FF Count 176

Figure 2-39: **ATC2 Core Data Capture Options**

Capture Mode

The capture mode of the ATC2 core can be set to either "State" mode for synchronous data capture to the CLK input signal or to "Timing" mode for asynchronous data capture. In "State" mode, the data path through the ATC2 core uses pipeline flip-flops that are clocked on the CLK input port signal. In "Timing" mode, the data path through the ATC2 core is composed purely of combinational logic all the way to the output pins. Also, in "Timing" mode, the ATCK pin is used as an extra data pin.

Signal Bank Count

The ATC2 core contains an internal, run-time selectable data signal bank multiplexer. The Signal Bank Count setting is used to denote the number of data input ports or signal banks the multiplexer will implement. The valid Signal Bank Count values are 1, 2, 4, 8, 16, or 32.

Driver Endpoint Type

The Driver Endpoint Type setting is used to control whether single-ended or differential output drivers are used on the ATCK and ATD output pins. All ATCK and ATD pins must use the same driver endpoint type.

TDM Rate

The ATC2 core does not use on-chip memory resources to store the captured trace data. Instead, it transmits the data to be captured by an Agilent logic analyzer that is attached to the FPGA pins using a special probe connector. The data can be transmitted out the device pins at the same rate as the incoming DATA port (TDM rate = 1x) or twice the rate as the DATA port (TDM rate = 2x). The TDM rate can be set to "2x" only when the capture mode is set to "State".

ATD Pin Count

The ATC2 core can implement any number of ATD output pins in the range of 4 through 128.

Data Port Width

The width of each input data port of the ATC2 core depends on the capture mode and the TDM rate. In "State" mode, the width of each data port is equal to (ATD pin count) * (TDM rate). In "Timing" mode, the width of each data port is equal to (ATD pin count + 1) * (TDM rate) since the ATCK pin is used as an extra data pin.

Pin Edit Mode

The Pin Edit Mode parameter is a time saving feature that allows you to change the IO Standard, Drive and Slew Rate pin parameters on individual pins or together as a group of pins. Setting the Pin Edit Mode to "Individual" allows you to edit the parameters of each pin independently from one another. Setting the mode to "Same as ATCK" allows you to change the ATCK pin parameters and forces all ATD pins to the same settings. You need to set unique pin locations for each individual pin regardless of the Pin Edit Mode.

Pin Parameters

The output clock (ATCK) and data (ATD) pins are instantiated inside the ATC2 core for your convenience. This means that although you do not have to manually bring the ATCK and ATD pins through every level of hierarchy to the top-level of your design, you do need to specify the location and other characteristics of these pins in the Core Generator. These pin attributes are then added to the `*.ncf` file of the ATC2 core. Using the settings in the Pin Parameters table, you can control the location, I/O standard, output drive and slew rate of each individual ATCK and ATD pin.

Pin Name

The ATC2 core has two types of output pins: ATCK and ATD. The ATCK pin is used as a clock pin when the capture mode is set to "State" and is used as a data pin when the capture mode is set to "Timing". The ATD pins are always used as data pins. The names of the pins cannot be changed.

Pin Loc

The Pin Loc column is used to set the location of the ATCK or ATD pin.

IO Standard

The IO Standard column is used to set the I/O standard of each individual ATCK or ATD pin. The I/O standards that are available for selection depend on the device family and driver endpoint type. The names of the I/O standards are the same as those that are found in the IOSTANDARD section of the Constraints Guide in the Xilinx Software Manual (http://toolbox.xilinx.com/docsan/xilinx6/books/docs/cgd/cgd.pdf).

VCCO

The VCCO column setting denotes the output voltage of the pin driver and depends on the IO Standard selection.

Drive

The Drive column setting denotes the maximum output drive current of the pin driver and ranges from 2 to 24 mA, depending on the IO Standard selection.

Slew Rate

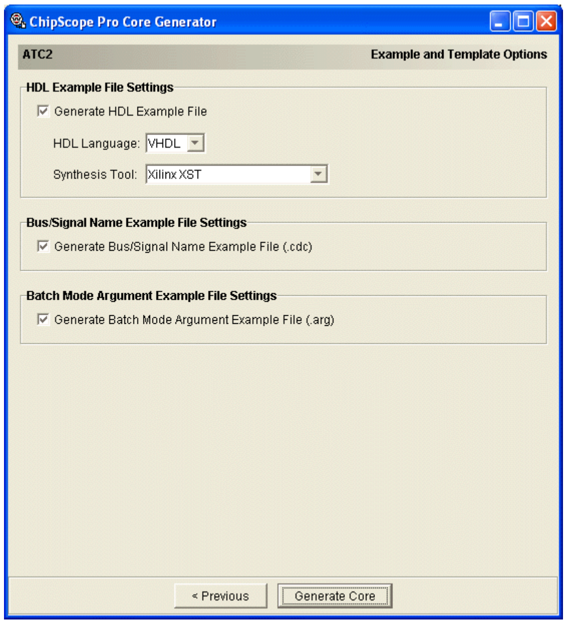
The Slew Rate column can be set to either FAST or SLOW for each individual ATCK or ATD pin.

Core Utilization

The ATC2 core generator has a core resource utilization monitor that estimates the number of look-up tables (LUTs) and flip-flops (FF) used by the ATC2 core, depending on the parameters used. The ATC2 core never uses block RAM or additional clock resources (for example, BUFG or DCM components).

Creating ATC2 Example Templates

After selecting the data capture parameters for the ATC2 core, click **Next** to view the Example and Template Generation Options (Figure 2-40).



HDL Example File Settings
- [✓] Generate HDL Example File
- HDL Language: VHDL
- Synthesis Tool: Xilinx XST

Bus/Signal Name Example File Settings
- [✓] Generate Bus/Signal Name Example File (.cdc)

Batch Mode Argument Example File Settings
- [✓] Generate Batch Mode Argument Example File (.arg)

Figure 2-40: **ATC2 Core Example Template Generation Options**

HDL Example Files

You can choose to construct an example HDL instantiation template by selecting the **Generate HDL Example File** and then selecting which synthesis tool and language to use. The synthesis tools supported are:

- Exemplar Leonardo Spectrum
- Synopsys FPGA Compiler
- Synopsys FPGA Compiler II
- Synopsys FPGA Express
- Synplicity Synplify
- XST (Xilinx Synthesis Technology)

Specifically tailored attributes and options are embedded in the HDL instantiation template for the various synthesis tools. To generate the ATC2 core without any HDL example files, deselect the **Generate HDL Example File** checkbox.

Bus/Signal Name Example Files (.CDC)

The bus/signal name example file for the ATC2 core (for example, `atc2.cdc`) contains generic information about the data ports of the ATC2 core. The `atc2.cdc` file will be created if you select the **Generate Bus/Signal Name Example File (.cdc)** checkbox. You can use the `atc2.cdc` file as a template to change data port signal names, create buses, and so on. The modified `atc2.cdc` file can then be imported into the Agilent logic analyzer.

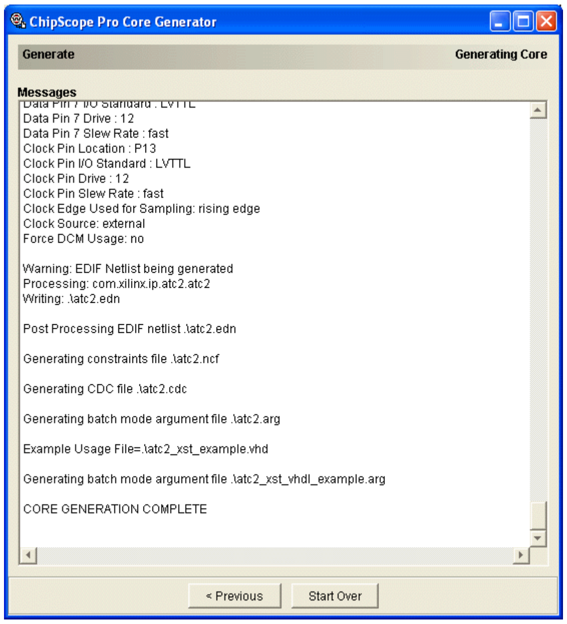
Batch Mode Generation Argument Example Files

You can also create a batch mode argument example file (for example, `atc2.arg`) by selecting the **Generate Batch Mode Argument Example File (.arg)** checkbox. The `atc2.arg` file is used with the command line program called "**generate.exe**". The `atc2.arg` file contains all of the arguments necessary for generating the ATC2 core without having to use the ChipScope Pro Core Generator GUI tool.

Note: An ATC2 core can be generated by running "**generate.exe atc2 -f=atc2.arg**" at the command prompt on Windows systems or by running "**generate.sh atc2 -f=atc2.arg**" at the UNIX shell prompt on Solaris systems.

Generating the Core

After entering the ATC2 core parameters, click **Generate Core** to create the netlist and applicable code examples. A message window opens, the progress information appears, and the `CORE GENERATION COMPLETE` message signals the end of the process (Figure 2-41). You can select to either go back and specify different options or click **Start Over** to generate new cores.



```
Data Pin 7 I/O Standard : LVTTL
Data Pin 7 Drive : 12
Data Pin 7 Slew Rate : fast
Clock Pin Location : P13
Clock Pin I/O Standard : LVTTL
Clock Pin Drive : 12
Clock Pin Slew Rate : fast
Clock Edge Used for Sampling: rising edge
Clock Source: external
Force DCM Usage: no

Warning: EDIF Netlist being generated
Processing: com.xilinx.ip.atc2.atc2
Writing: .\atc2.edn

Post Processing EDIF netlist .\atc2.edn

Generating constraints file .\atc2.ncf

Generating CDC file .\atc2.cdc

Generating batch mode argument file .\atc2.arg

Example Usage File=.\atc2_xst_example.vhd

Generating batch mode argument file .\atc2_xst_vhdl_example.arg

CORE GENERATION COMPLETE
```

Figure 2-41: **ATC2 Core Generation Complete**

Using the ATC2 Core

To instantiate the example ATC2 core HDL files into your design, use the following guidelines to connect the ATC2 core port signals to various signals in your design:

- Connect the ATC2 core's CONTROL port signal to an *unused* control port of the ICON core instance in the design
- Connect all unused bits of the ATC2 core's asynchronous and synchronous input signals to a "0". This prevents the mapper from removing the unused trigger and/or data signals and also avoids any DRC errors during the implementation process
- For best results, make sure the "State" mode input data port signals are synchronous to the ATC2 clock signal (CLK); this is not important for "Timing" mode input data port signals

Using the ChipScope Pro Core Inserter

Core Inserter Overview

The ChipScope Pro Core Inserter is a post-synthesis tool used to generate a netlist that includes the user design as well as parameterized ICON, ILA, ILA/ATC, and ATC2 cores as needed. The Core Inserter gives you the flexibility to quickly and easily use the ChipScope Pro debug functionality to analyze an already synthesized design, and without any HDL instantiation.

Note: The IBA/OPB, IBA/PLB and VIO cores are currently not supported in the Core Inserter.

Using the Core Inserter with ISE Project Navigator on Windows

This section is provided for users of the Windows version of ChipScope Pro 6.3i and the Windows version of Xilinx ISE 6.3i. If you are using the Solaris or Linux versions of ChipScope Pro 6.3i with ISE 6.3i, refer to Using the Core Inserter with ISE Project Navigator on Solaris or Linux, page 3-4.

The Core Inserter `.cdc` file can be added as a new source file to the Project Navigator source file list. In addition to this, the Project Navigator tool will also recognize and invoke the Core Inserter tool during the appropriate steps in the implementation flow. For more information on how the Project Navigator and the Core Inserter are integrated, refer to the Project Navigator section of the Xilinx ISE Software Manuals (http://support.xilinx.com/support/software_manuals.htm).

ChipScope Definition and Connection Source File

To insert ChipScope cores via the ChipScope Pro Core Inserter into a design processed by the Xilinx ISE 6.3i Project Navigator tool, follow these steps:

1. Add ChipScope Definition and Connection File (`.cdc`) to the project and associate it with the appropriate design module.
   a. To create a new `.cdc` file, select **Project** → **New Source**, then select **ChipScope Definition and Connection File** and give the file a name (Figure 3-1, page 3-2). Click through the remaining dialog boxes using the default settings, as needed.

Note: The ChipScope Definition and Connection File source type is only listed if Project Navigator 6.3i detects a ChipScope Pro 6.3i installation (the respective versions must match).

.

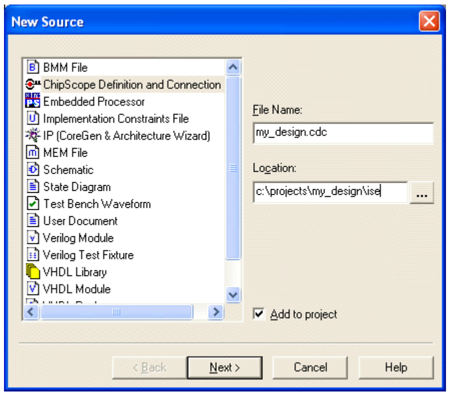


Figure 3-1: **Creating a New .cdc Source**

b. To add an existing `.cdc` file, select **Project** → **Add Source** or **Project** → **Add Copy of Source**, then browse for the existing `.cdc` file.

When prompted, associate the `.cdc` file with the appropriate top-level design module. The `.cdc` file should now be displayed in the Sources in Project window underneath the associated design module (Figure 3-2).

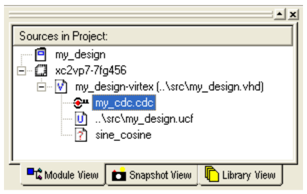


Figure 3-2: **The .cdc Source File**

2. To create the ChipScope Pro Cores and complete the signal connections, double-click the `.cdc` file in the Sources in Project window. This runs the Synthesis (if applicable) and Translate processes, as necessary, and then opens the `.cdc` file in the ChipScope Pro Core Inserter tool.
3. Modify the cores and connections in the Core Inserter tool as necessary (as shown in the section called ChipScope Pro Core Inserter Menu Features, page 3-6), then close the Core Inserter tool.
4. When the associated top-level design is implemented in Project Navigator, the ChipScope Pro cores are automatically inserted into the design netlist as part of the Translate phase of the flow. There is no need to set any properties to enable this to happen. The `.cdc` is in the project and associated with the design module being implemented and causes the cores to be inserted automatically.

Useful Project Navigator Settings

The following are useful Project Navigator settings to help you implement a design with ChipScope Pro cores:

1. If you use the XST synthesis tool, enable the **Keep Hierarchy** option to preserve the design hierarchy and prevent the XST tool from optimizing across all levels of hierarchy in your design. Using the **Keep Hierarchy** option preserves the names of nets and other recognizable components during the ChipScope Pro core insertion stage of the flow. If you do not use the **Keep Hierarchy** option, some of your nets and/or components may be combined with other logic into new components or otherwise optimized away. To keep the design hierarchy:
   a. Select **Edit** → **Preferences** to bring up the Preferences dialog box.
   b. Select the **Processes** tab.
   c. Set the Property Display Level combobox dropdown to **Advanced** and click **OK**.
   d. Right-click on the **Synthesize** process and select the **Properties...** option.
   e. Make sure the **Keep Hierarchy** option is checked and click OK.
2. Prior to using the ChipScope Pro Analyzer to download your bitstream into your device, make sure the bitstream generation options are set properly:
   a. In the Project Navigator, right-click on the **Generate Programming File** process and select the **Properties...** option.
   b. Select the **Startup options** tab.
   c. Set the FPGA Start-Up Clock dropdown to **JTAG Clock**.

Using the Core Inserter with ISE Project Navigator on Solaris or Linux

This section is provided for users of the Solaris or Linux versions of ChipScope Pro 6.3i and the Solaris or Linux versions of Xilinx ISE 6.3i, respectively. If you are using the Windows version of ChipScope Pro 6.3i with ISE 6.3i. Refer to Using the Core Inserter with ISE Project Navigator on Windows, page 3-1.

Using the Solaris or Linux ChipScope Pro 6.3i Core Inserter with Solaris or Linux Xilinx ISE 6.3i implementation tools, respectively, requires some flow manipulation to ensure the correct files are implemented.

Note: You must follow all of these steps after making any modifications to the HDL source or after deleting any of the implementation data. Simply running through the standard ISE flow will not insert the ChipScope Pro cores into the design. If you are using the XST flow, the Core Inserter will write over the output of XST, replacing it with a netlist (`.ngc`) file that will contain the design as well as the ChipScope Pro cores. To change core properties or connectivity, or to run implementation on the design without ChipScope Pro cores present, the Synthesis step must be run again.

To insert ChipScope Pro cores via the ChipScope Pro Core Inserter into a design processed by Xilinx ISE 6.3i, follow these steps:

1. **Synthesize** and **Translate** the design.

   After creating a project and adding the HDL or EDIF/XST source files, **Synthesize** (if required for the selected flow) and **Translate** the design by double-clicking **Translate** under **Implement Design**. After the design has been translated, the ISE process view should look like Figure 3-3. The example shown consists of a single HDL file that is synthesized using XST.

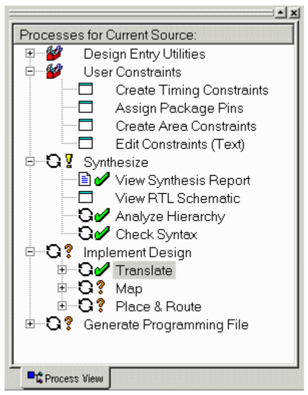


Figure 3-3: **Xilinx ISE Process View Before Core Insertion**

2. **Run** the ChipScope Pro Core Inserter.

   The Solaris ChipScope Core Inserter is run by executing `$CHIPSCOPE/bin/sol/inserter.sh`.

   The Linux ChipScope Core Inserter is run by executing `$CHIPSCOPE/bin/lin/inserter.sh`.

3. **Browse** for the top level design to fill in the **Input Design Netlist**. Verify the following parameters are set under the **Device** tab:

   - For Exemplar, Synopsys, and Synplicity designs:

     ```
     Input Design Netlist = ProjectDir\design.{edf|edn}
     Output Design Netlist = ProjectDir\_ngo\design.ngo
     Output Directory = ProjectDir\_ngo
     ```

   - For XST designs:

     ```
     Input Design Netlist = ProjectDir\design.ngc
     Output Design Netlist = ProjectDir\design.ngc
     Output Directory = ProjectDir\_ngo
     ```

     Browse for the input *design*.edf, *design*.edn, or *design*.ngc file. The Core Inserter will detect the presence of the ISE project when it finds the _ngo directory, and the two output fields will be filled in automatically. This can be done only after the **Translate** step has been run at least once.

4. Complete ICON and ILA core insertion in the ChipScope Pro Core Inserter.

   Define ChipScope Pro Core parameters and connect all signals, then insert the cores. The Core Inserter places new NGO files for the top level, as well as for the ICON and ILA cores, in the _ngo directory. Save the project before exiting the Core Inserter.

5. Rerun the **Translate** step.

   When you return to Xilinx ISE 6.3i from the Core Inserter, you should see that the green check mark by the **Translate** step has disappeared (Figure 3-4). Double-click **Translate**, or right-click **Translate** and select **Run**. The RUN command will start with the new NGO files.

   Note: Do *not* select **Rerun All**, as this will rerun Synthesis and the first portion of **Translate**, which will undo the Core Insertion that was just performed.

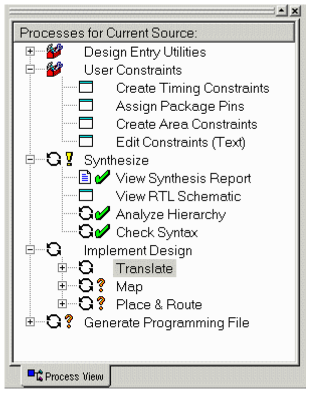

Processes for Current Source:
- Design Entry Utilities
- User Constraints
  - Create Timing Constraints
  - Assign Package Pins
  - Create Area Constraints
  - Edit Constraints (Text)
- Synthesize
  - View Synthesis Report
  - View RTL Schematic
  - Analyze Hierarchy
  - Check Syntax
- Implement Design
  - Translate
  - Map
  - Place & Route
- Generate Programming File

Figure 3-4: **Xilinx ISE Process View After Core Insertion**

6. Continue with **Implementation**.

Using the Core Inserter with Command Line Implementation

Since the Core Inserter sometimes generates a file of a different format than synthesis (an NGO file instead of an EDIF netlist), the implementation flow is slightly different. If you use the command line, use the NGO or NGC file produced by the Core Inserter instead of the EDIF or XST netlist.

For JTAG configuration:

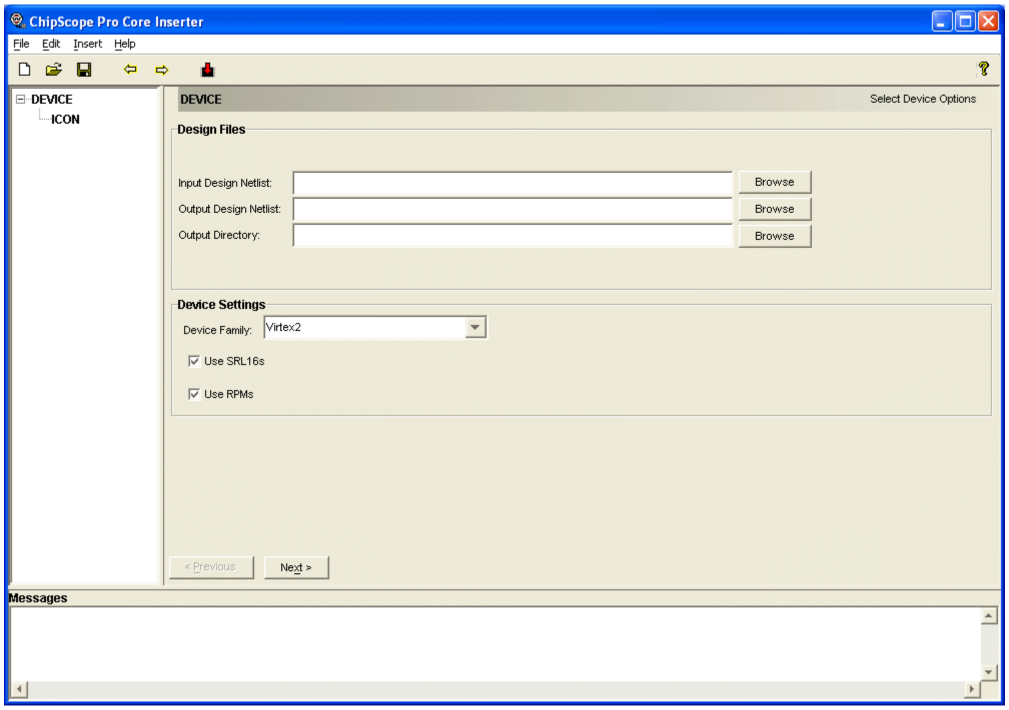
```
ngdbuild -sd <netlist path> -p <part type> base_ila.ngo
map base_ila
par -ol 5 -w base_ila base_ila
bitgen -w -g UserID:<user ID> -g StartupClk:JTAGClk base_ila base_ila
```

ChipScope Pro Core Inserter Menu Features

Working with Projects

Projects saved in the Core Inserter hold all relevant information about source files, destination files, core parameters, and core settings. This allows you to store and retrieve information about core insertion between sessions. The project file (`.cdc` extension) can also be used as an input to the ChipScope Pro Analyzer to import signal names.

When the ChipScope Core Inserter is first opened, all the relevant fields are completely blank. Using the command **File** → **New** also results in this condition (Figure 3-5).



Input Design Netlist:
Output Design Netlist:
Output Directory:
Device Family: Virtex2
☑ Use SRL16s
☑ Use RPMs

Figure 3-5: **Blank Core Inserter Project**

Opening an Existing Project

To open an existing project, select it from the list of recently opened projects, or select **File** → **Open Project**, and **Browse** to the project location. After you locate the project, you can either double click on it or click **Open**.

Saving Projects

If a project has changed during the course of a session, you will be prompted to save the project upon exiting the Core Inserter. You can also save a project by selecting **File** → **Save**. To rename the current project or save it to another filename, select **File** → **Save As**, type in the new name, and click **Save.**

Refreshing the Netlist

The Core Inserter automatically reloads the design netlist if it detects that the netlist has changed since the last time it was loaded. However, you can force the Core Inserter to refresh the netlist by selecting **File** → **Refresh Netlist**.

Inserting and Removing Units

You can insert new units into the project by selecting **Edit** → **New ILA Unit**, **Edit** → **New ILA/ATC Unit**, or **Edit** → **New ATC2 Unit**. You can remove a unit by selecting **Edit** → **Remove Unit** after choosing which unit to delete.

Setting Preferences

You can set the ChipScope Core Inserter project preferences by selecting **Edit** → **Preferences**. They are organized into three categories: Tools, ISE Integration, and Miscellaneous. Refer to Managing Project Preferences, page 3-28 for more information about setting these preferences.

Inserting the Cores

ICON, ILA, ILA/ATC, and ATC2 cores are inserted when the flow is completed, or by selecting **Insert** → **Insert Core**. If all channels of all the capture cores are not connected to valid signals, an error message results.

Exiting the Core Inserter

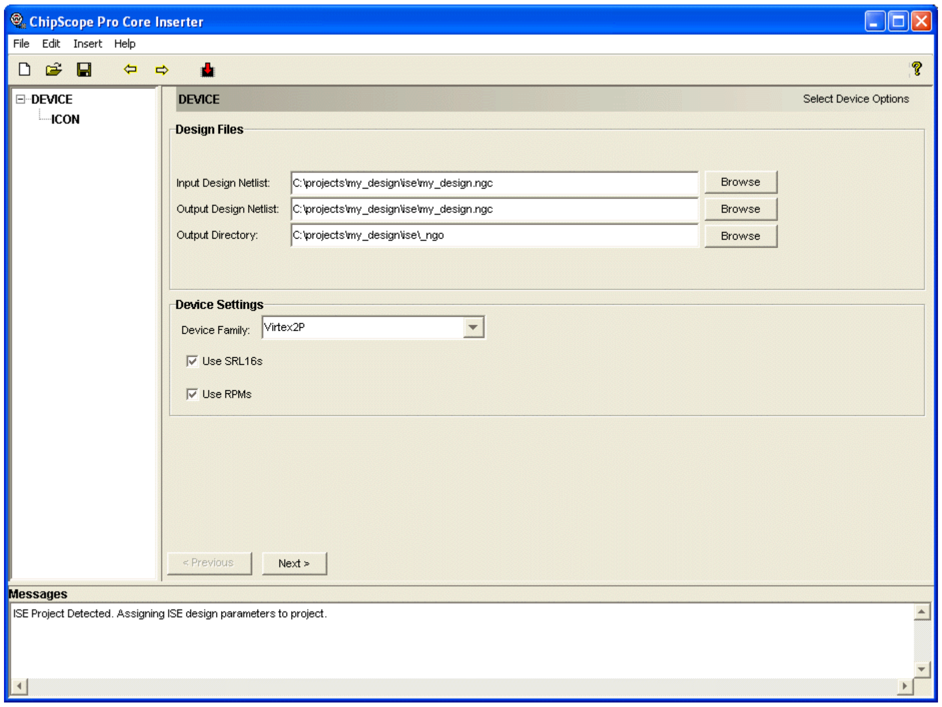
To exit the ChipScope Core Inserter, select **File** → **Exit**.

Specifying Input and Output Files

The ChipScope Core Inserter works in a step-by-step process.

1. Specify the Input Design Netlist (Figure 3-5, page 3-6).
2. Click **Browse** to navigate to the directory where the netlist resides.
3. Modify the Output Design Netlist and Output Directory fields as needed. (These fields are automatically filled in initially.)

Figure 3-6 shows a project with input and output files specified.



Input Design Netlist: C:\projects\my_design\ise\my_design.ngc

Output Design Netlist: C:\projects\my_design\ise\my_design.ngc

Output Directory: C:\projects\my_design\ise_ngo

Device Family: Virtex2P

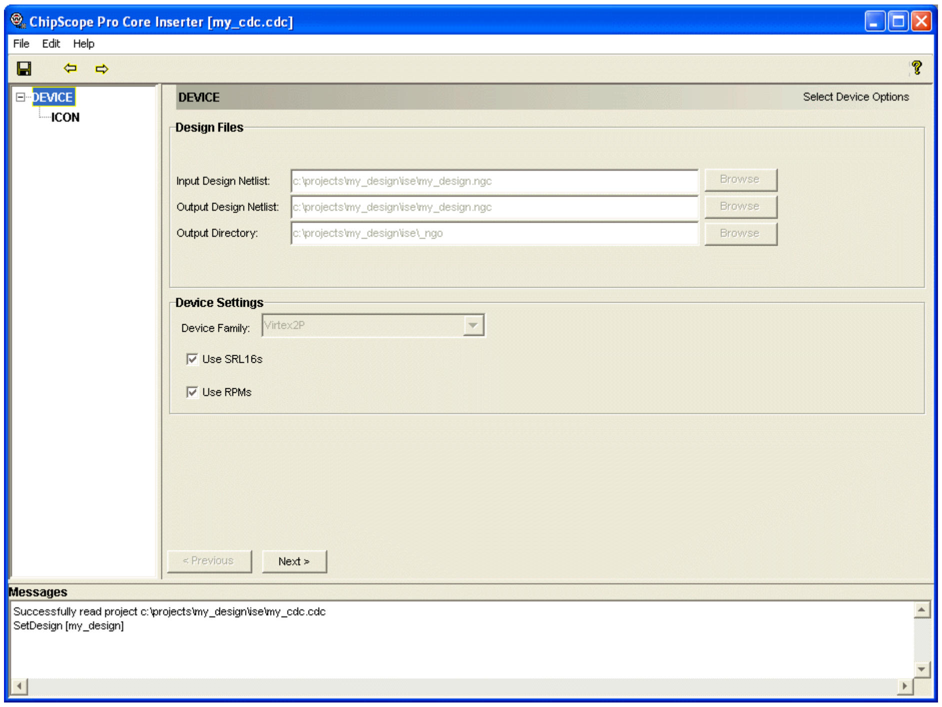
☑ Use SRL16s

☑ Use RPMs

ISE Project Detected. Assigning ISE design parameters to project.

Figure 3-6: **Core Inserter Project with Files Specified**

Note: When the Core Inserter is invoked from the Project Navigator tool, the Input Design Netlist, Output Design Netlist, Output Directory and Device Family fields are automatically filled in (Figure 3-7, page 3-9). In this case, these fields can only be changed by the Project Navigator tool and cannot be modified directly in the Core Inserter.


ChipScope Pro Core Inserter [my_cdc.cdc]

File Edit Help

DEVICE
ICON

DEVICE — Select Device Options

Design Files

Input Design Netlist: c:\projects\my_design\ise\my_design.ngc

Output Design Netlist: c:\projects\my_design\ise\my_design.ngc

Output Directory: c:\projects\my_design\ise_ngo

Device Settings

Device Family: Virtex2P

☑ Use SRL16s

☑ Use RPMs

< Previous Next >

Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-7: **Core Inserter as Launched from Project Navigator**

Project Level Parameters

Three project level parameters (device family, SRL16 usage, and RPM usage) must be specified for every project. Due to the increased depth of Virtex-II device block RAM, different cores can be generated to take advantage of these deeper RAMs.

Selecting the Target Device Family

The target FPGA device family is displayed in the Device Family field. The structure of the ICON and capture cores are optimized for the selected device family. Use the pull-down selection to change the device family to the desired architecture.

The default target device family is Virtex-II.

Using SRL16s

The **Use SRL16s** checkbox is used to select whether or not the cores will be generated using SRL16 and SRL16E components. This option is only available for the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families). If the checkbox is *not* selected, the SRL16 components are replaced with flip-flops and multiplexers, which affects the size and performance of the generated cores.

The **Use SRL16s** checkbox is checked by default to generate cores that use the optimized SRL16 technology.

Using RPMs

The **Use RPMs** checkbox is used to select whether the individual cores should be Relationally Placed Macros (RPMs). This option places restraints on the place-and-route tool to optimize placement of all the logic for the ChipScope Pro core in one area. If your design uses most of the resources in the device, these placement constraints may not be met. The **Use RPMs** checkbox is checked by default in order to generate cores that are optimized for placement. When this step is completed, click **Next**.

Choosing ICON Options

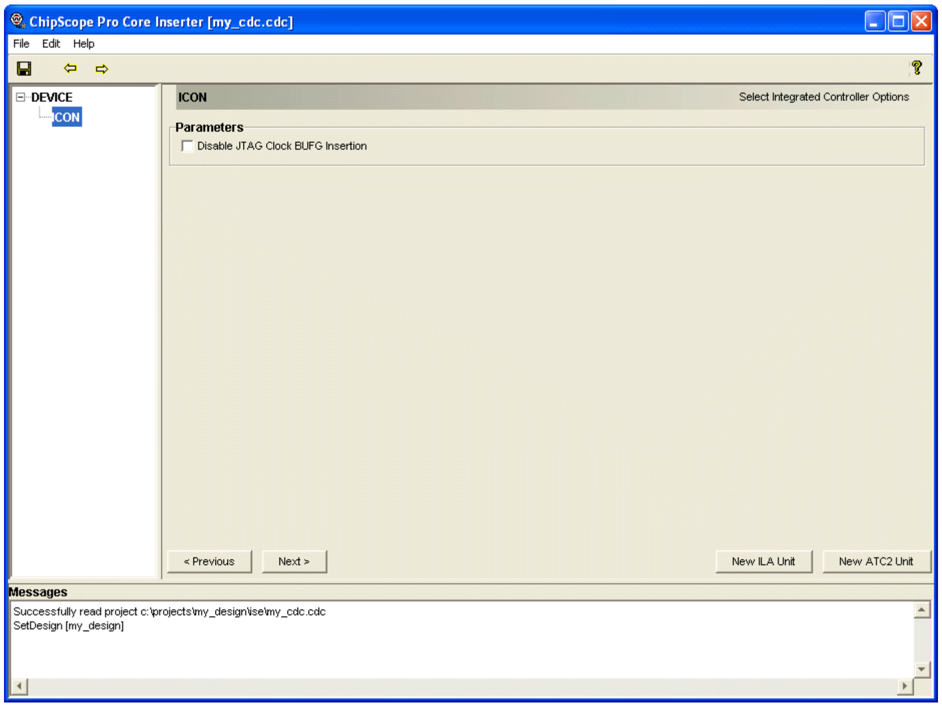
The first options that need to be specified are for the ICON core. The ICON core is the controller core that connects all ILA, ILA/ATC, and ATC2 cores to the JTAG boundary scan chain. The ICON core has the parameters shown in Figure 3-8.

Disabling JTAG Clock BUFG Insertion

Disabling the JTAG clock BUFG insertion causes the implementation tools to route the JTAG clock using normal routing resources instead of global clock routing resources. By default, this clock is placed on a global clock resource (`BUFG`). To disable this `BUFG` insertion, check the **Disable JTAG Clock BUFG Insertion** checkbox.

Note: This should only be done if global resources are very scarce; placing the JTAG clock on regular routing, even high-speed backbone routing, introduces skew. Make sure the design is adequately constrained to minimize this skew.

After selecting the desired ICON parameters (Figure 3-8) click **Next** or **New ILA Unit** to automatically create an ILA core and display the Select Logic Analyzer Options window. To add a new ATC2 core instead of an ILA core, click either **New ATC2 Unit**. To add a new ILA/ATC core, select the **Edit** → **New ILA/ATC Unit** menu option.

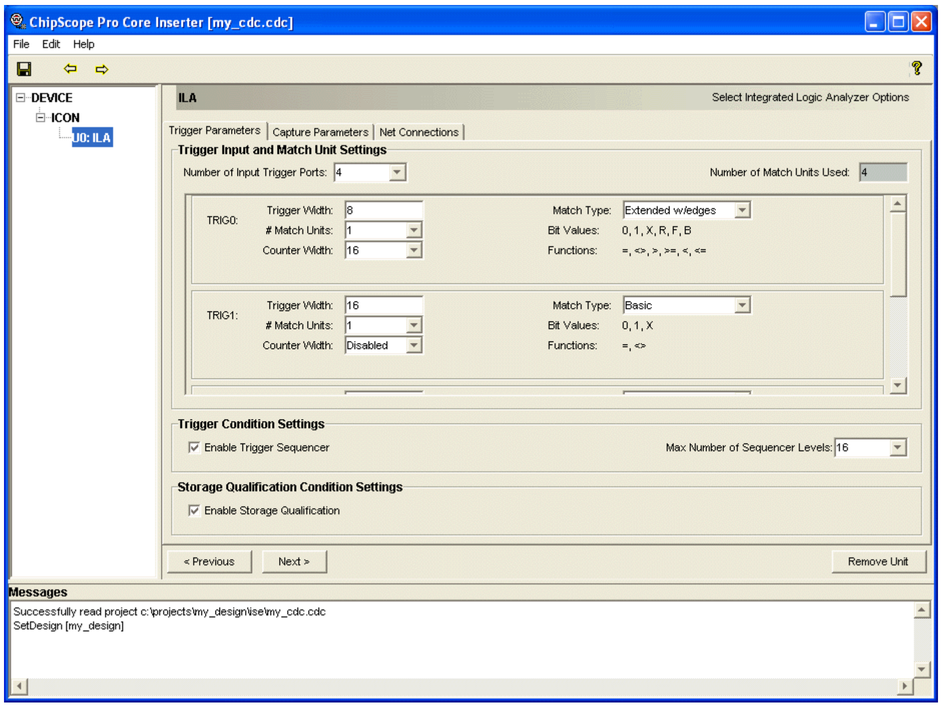

ChipScope Pro Core Inserter [my_cdc.cdc]
File
Edit
Help
DEVICE
ICON
ICON
Select Integrated Controller Options
Parameters
Disable JTAG Clock BUFG Insertion
< Previous
Next >
New ILA Unit
New ATC2 Unit
Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-8: **ICON Options**

Choosing ILA or ILA/ATC Trigger Options and Parameters

Notice in Figure 3-9 that a new ILA unit has been created in the device hierarchy on the left. The next step is to set up the ILA unit. Figure 3-9 shows a sample of the first tab in the ILA options and parameters sequence. The first tab sets up the trigger options for the ILA or ILA/ATC core.



Trigger Input and Match Unit Settings

Number of Input Trigger Ports: 4
Number of Match Units Used: 4

TRIG0:
Trigger Width: 8
# Match Units: 1
Counter Width: 16
Match Type: Extended w/edges
Bit Values: 0, 1, X, R, F, B
Functions: =, <>, >, >=, <, <=

TRIG1:
Trigger Width: 16
# Match Units: 1
Counter Width: Disabled
Match Type: Basic
Bit Values: 0, 1, X
Functions: =, <>

Trigger Condition Settings

[✓] Enable Trigger Sequencer
Max Number of Sequencer Levels: 16

Storage Qualification Condition Settings

[✓] Enable Storage Qualification

Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-9: **ILA Trigger Parameters**

Selecting the Number of Trigger Ports

Each ILA or ILA/ATC core can have up to 16 separate trigger ports that can be set up independently. After you select the number of trigger ports from the Number of Trigger Ports pull-down list, a group of options appears for each of these ports. The group of options associated with each trigger port is labeled with **TRIG*n***, where *n* is the trigger port number 0 to 15. The trigger port options include trigger width, number of match units connected to the trigger port, and the type of these match units.

Entering the Width of the Trigger Ports

The individual trigger ports are buses that are made up of individual signals or bits. The number of bits used to compose a trigger port is called the *trigger width*. The width of each trigger port can be set independently using the TRIG*n* Trigger Width field. The range of values that can be used for trigger port widths is 1 to 256.

Selecting the Number of Trigger Match Units

A match unit is a comparator that is connected to a trigger port and is used to detect events on that trigger port. The results of one or more match units are combined together to form the overall trigger condition event that is used to control the capturing of data. Each trigger port **TRIG*n*** can be connected to 1 to 16 match units by using the # Match Units pull-down list.

Note: The aggregate number of match units used in a single ILA core cannot exceed 16, regardless of the number of trigger ports used.

Selecting one match unit conserves resources while still allowing some flexibility in detecting trigger events. Selecting two or more trigger match units allows a more flexible trigger condition equation to be a combination of multiple match units. However, increasing the number of match units per trigger port also increases the usage of logic resources accordingly.

Selecting the Match Unit Type

The different comparisons or match functions that can be performed by the trigger port match units depend on the type of the match unit. Six types of match units are supported by the ILA and ILA/ATC cores (Table 3-1).

Table 3-1: **ILA Trigger Match Unit Types**

Type	Bit Values[a]	Match Function	Bits Per Slice[b]	Description
Basic	0, 1, X	'=', '<>'	8	Can be used for comparing data signals where transition detection is not important. This is the most bit-wise economical type of match unit.
Basic w/edges	0, 1, X, R, F, B	'=', '<>'	4	Can be used for comparing control signals where transition detection (e.g., low-to-high, high-to-low, etc.) is important.
Extended	0, 1, X	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where magnitude is important.
Extended w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<='	2	Can be used for comparing address or data signals where a magnitude and transition detection are important.
Range	0, 1, X	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values is important.
Range w/edges	0, 1, X, R, F, B	'=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range'	1	Can be used for comparing address or data signals where a range of values and transition detection are important.

a. Bit values: '0' means "logical 0", '1' means "logical 1", 'X' means "don't care", 'R' means "0-to-1 transition", 'F' means "1-to-0 transition" and 'B' means "any transition".

b. The Bits Per Slice value is only an approximation that is used to illustrate the relative resource utilization of the different match unit types. It should not be used as a hard estimate of resource utilization.

Use the TRIGn Match Type pull-down list to select the type of match unit that will apply to all match units connected to the trigger port. However, as the functionality of the match unit increases, so does the amount of resources necessary to implement that functionality. This flexibility allows you to customize the functionality of the trigger module while keeping resource usage in check.

Selecting Match Unit Counter Width

The match unit counter is a configurable counter on the output of the each match unit in a trigger port. This counter can be configured at run time to count a specific number of match unit events. To include a match counter on each match unit in the trigger port, select a counter width from 1 to 32. The match counter will not be included on each match unit if the Counter Width combo box is set to Disabled. The default Counter Width setting is Disabled.

Enabling the Trigger Condition Sequencer

The standard Boolean equation trigger condition can be augmented with an optional trigger sequencer by checking the **Enable Trigger Sequencer** checkbox. A block diagram of the trigger sequencer is shown in Figure 3-10.

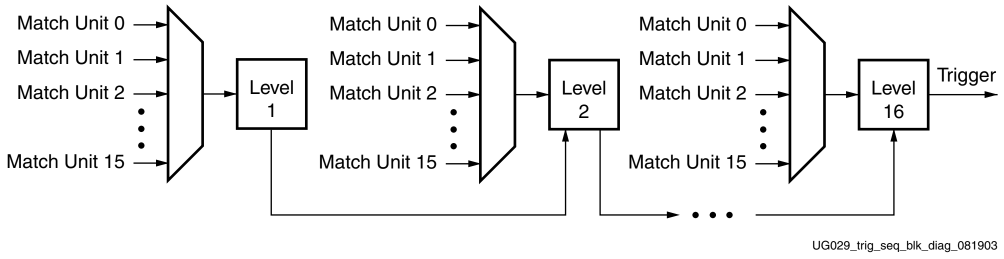


Figure 3-10: **Trigger Sequencer Block Diagram with 16 Levels and 16 Match Units**

The trigger sequencer is implemented as a simple cyclical state machine and can transition through up to 16 states or levels before the trigger condition is satisfied. The transition from one level to the next is caused by an event on one of the match units that is connected to the trigger sequencer. Any match unit can be selected at run time on a per level basis to transition from one level to the next. The trigger sequencer can be configured at run time to transition from one level to the next on either contiguous or non-contiguous sequences of match function events.

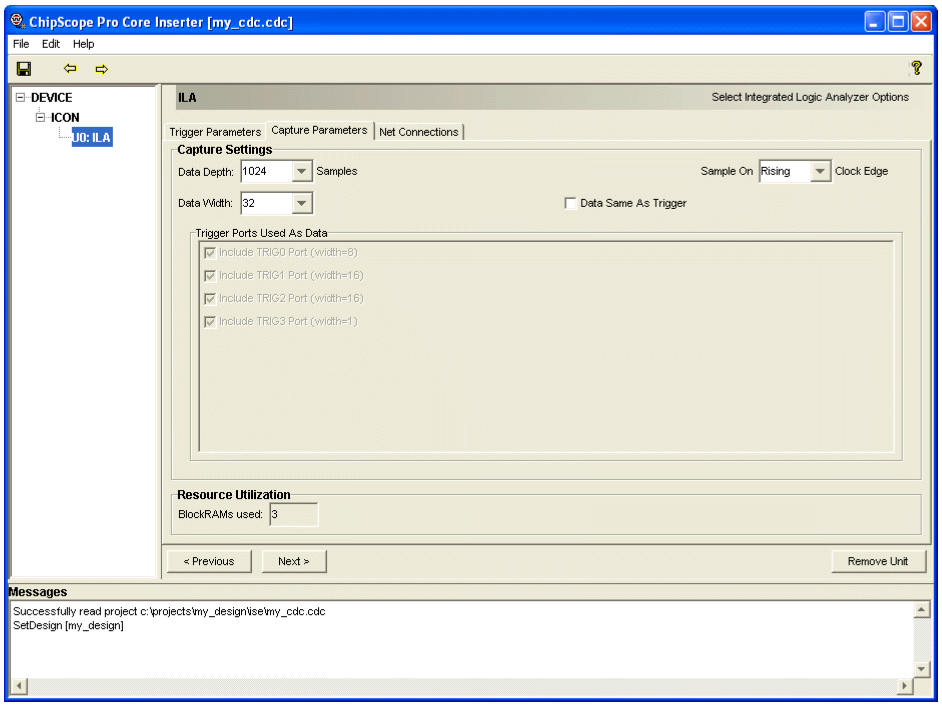
Enabling the Storage Qualification Condition

In addition to the trigger condition, the ILA core can also implement a storage qualification condition. The storage qualification condition is a Boolean combination of match function events. These match function events are detected by the match unit comparators that are subsequently attached to the trigger ports of the core. The storage qualification condition differs from the trigger condition in that it evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start the capture process and what data to capture. The storage qualification condition can be enabled by checking the **Enable Storage Qualification** checkbox.

Note: The storage qualification condition is not available in the ILA/ATC core.

Choosing ILA Core Capture Parameters

The second tab in the Core Inserter that is used to set up the of the capture parameters of the ILA core (Figure 3-11). The capture parameters for ILA are different from ILA/ATC (see Figure 3-13, page 3-17).



Capture Settings

Data Depth: 1024 Samples

Sample On Rising Clock Edge

Data Width: 32

Data Same As Trigger

Trigger Ports Used As Data

- Include TRIG0 Port (width=8)
- Include TRIG1 Port (width=16)
- Include TRIG2 Port (width=16)
- Include TRIG3 Port (width=1)

Resource Utilization

BlockRAMs used: 3

Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-11: **ILA Core Capture Parameters**

Selecting the Data Depth

The maximum number of data sample words that the ILA core can store in the sample buffer is called the *data depth*. The data depth determines the number of data width bits contributed by each block RAM unit used by the ILA unit.

For the Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3 device families (including the QPro variants of these families), you can set the data depth to one of six values (Table 3-2).

Table 3-2: **Maximum Data Widths for Virtex-II/-II Pro/-4, Spartan-3**

	Depth 512	Depth 1024	Depth 2048	Depth 4096	Depth 8192	Depth 16384
1 block RAM	31	15	7	3	1	--
2 block RAMs	63	31	15	7	3	1
4 block RAMs	127	63	31	15	7	3
8 block RAMs	255	127	63	31	15	7
16 block RAMs	--	255	127	63	31	15
32 block RAMs	--	--	255	127	63	31
64 block RAMs	--	--	--	255	127	63
128 block RAMs	--	--	--	--	255	127
256 block RAMs	--	--	--	--	--	255

Note: One extra bit per sample is required for the trigger mark (e.g., a trigger/data width of 7 bits requires a full sample width of 8 bits, etc.)

For the Virtex, Virtex-E, Spartan-II and Spartan-IIE device families (including the QPro variants of these families), you can set the data depth to one of five values (Table 3-3).

Table 3-3: **Maximum Data Widths for Virtex/-E, Spartan-II /-IIE**

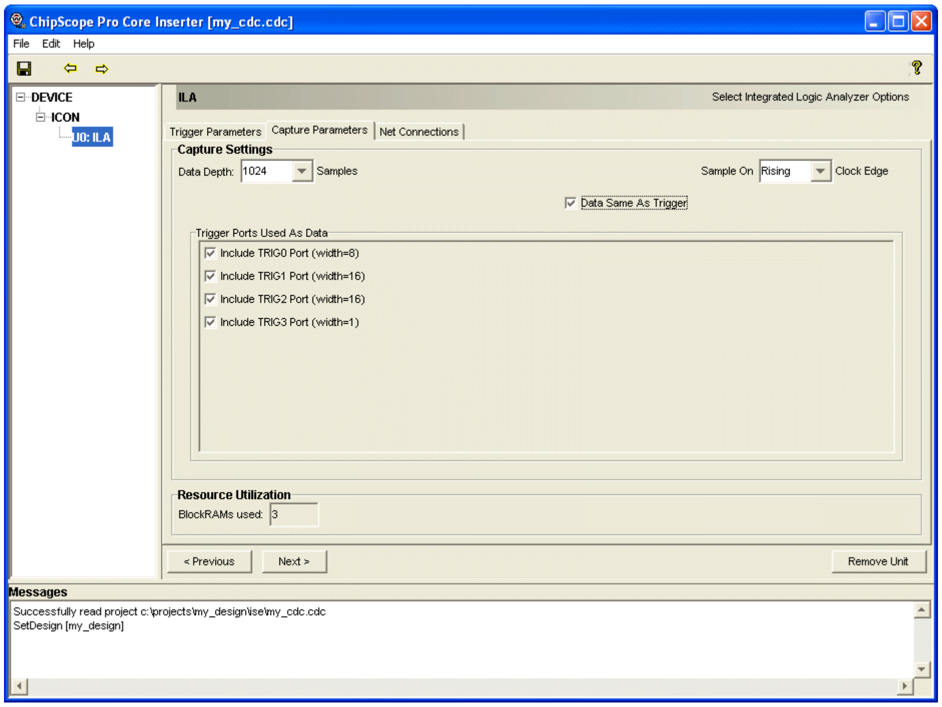
	Depth 256	Depth 512	Depth 1024	Depth 2048	Depth 4096
1 block RAM	15	7	3	1	--
2 block RAMs	31	15	7	3	1
4 block RAMs	63	31	15	7	3
8 block RAMs	127	63	31	15	7
16 block RAMs	255	127	63	31	15
32 block RAMs	--	255	127	63	31
64 block RAMs	--	--	255	127	63
128 block RAMs	--	--	--	255	127
256 block RAMs	--	--	--	--	255

Note: One extra bit per sample is required for the trigger mark (e.g., a trigger/data width of 7 bits requires a full sample width of 8 bits, etc.)

Selecting the Data Type

The data captured by the ILA trigger port can come from two source types:

- **Data Separate from Trigger (Figure 3-11)**
  - The data port is completely independent of the trigger ports
  - This mode is useful when you want to limit the amount of data being captured
- **Data Same as Trigger (Figure 3-12)**
  - The data and trigger ports are identical. This mode is very common in most logic analyzers, since you can capture and collect any data that is used to trigger the core.
  - Individual trigger ports can be selected to be included in the data port. If this selection is made, then the DATA input port will not be included in the port map of the ILA core.
  - This mode conserves CLB and routing resources in the ILA core, but is limited to a maximum aggregate data sample word width of 256 bits.



Capture Settings
Data Depth: 1024 Samples
Sample On Rising Clock Edge
[✓] Data Same As Trigger

Trigger Ports Used As Data
[✓] Include TRIG0 Port (width=8)
[✓] Include TRIG1 Port (width=16)
[✓] Include TRIG2 Port (width=16)
[✓] Include TRIG3 Port (width=1)

Resource Utilization
BlockRAMs used: 3

Messages
```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-12: **ILA Core Data Same As Trigger Parameters**

Selecting the Data-Same-As-Trigger Ports

If the **Data Same As Trigger** checkbox is selected, then a checkbox for each **TRIG*n*** port appears in the data options screen. These checkboxes should be used to select the individual trigger ports that will be included in the aggregate data port. Note that selecting the individual trigger ports automatically updates the Aggregate Data Width field accordingly. A maximum data width of 256 bits applies to the aggregate selection of trigger ports.

Entering the Data Width

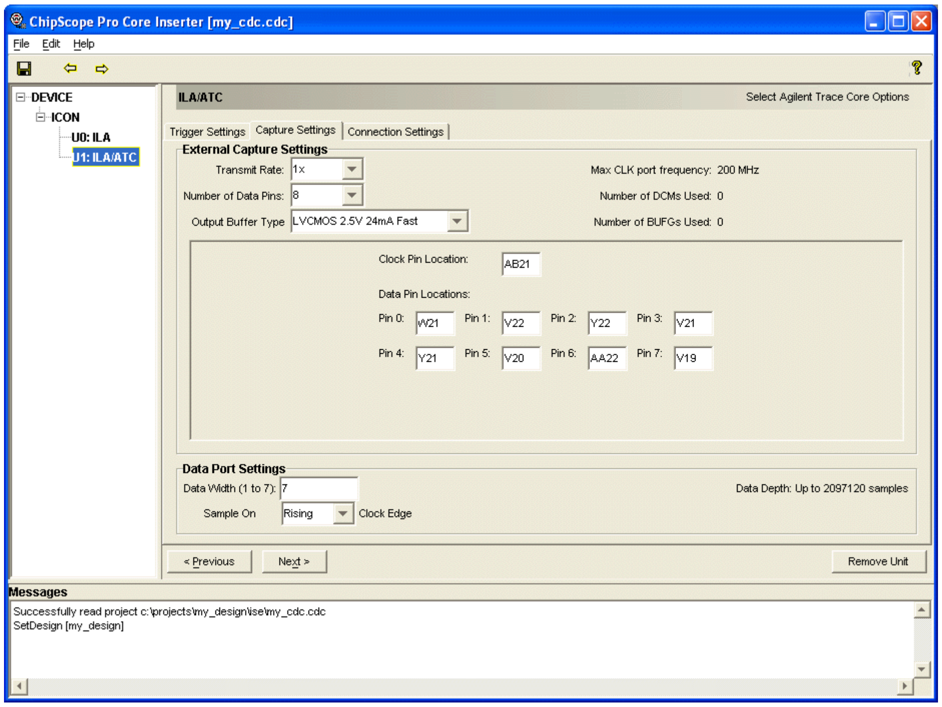
The width of each data sample word stored by the ILA core is called the *data width*. If the data and trigger words are independent from each other, then the maximum allowable data width depends on the target device type and data depth. However, regardless of these factors, the maximum allowable data width is 256 bits.

Number of Block RAMs

As the data depth and data width selections are changed, the Number of Block RAMs field notifies you of how many block RAMs will be used by the ILA core. The trigger mark is automatically taken into account when calculating this value.

Choosing ILA/ATC Capture Settings

If you are inserting an ILA/ATC core, the Capture Settings look like those in Figure 3-13.



External Capture Settings

Transmit Rate: 1x

Number of Data Pins: 8

Output Buffer Type: LVCMOS 2.5V 24mA Fast

Max CLK port frequency: 200 MHz

Number of DCMs Used: 0

Number of BUFGs Used: 0

Clock Pin Location: AB21

Data Pin Locations:

Pin 0: W21 Pin 1: V22 Pin 2: Y22 Pin 3: V21

Pin 4: Y21 Pin 5: V20 Pin 6: AA22 Pin 7: V19

Data Port Settings

Data Width (1 to 7): 7

Data Depth: Up to 2097120 samples

Sample On: Rising Clock Edge

Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-13: **ILA/ATC Core Data Settings**

Transmit Rate

The ILA/ATC core does not use on-chip memory resources to store the captured trace data. Instead, it transmits the data to be captured to an Agilent E5904B TPA that is attached to a special connector via FPGA device pins. The data can be transmitted out the device pins at the same rate as the incoming DATA port (transmit rate = 1x), twice the rate as the DATA port (transmit rate = 2x) or four times the DATA port rate (transmit rate = 4x).

Maximum CLK Port Frequency

The maximum output clock frequency is 200 MHz. The incoming CLK port rate is limited by the maximum output clock frequency as well as the transmit rate. If the transmit rate is 1x, then the maximum CLK port frequency is 200 MHz. If the transmit rate is 2x, then the maximum CLK port frequency is 100 MHz. Finally, if the transmit rate is 4x, then the maximum CLK port frequency is 50 MHz.

Clock Resource Usage

The ILA/ATC core will use dedicated clock resources if the transmit rate is 2x or 4x. The number of clock resources required by the ILA/ATC core are shown in Table 3-4.

Table 3-4: **ILA/ATC Clock Resource Utilization**

Transmit Rate	Virtex, Virtex-E, Spartan-II and Spartan-IIE		Virtex-II, Virtex-II Pro, Virtex-4, and Spartan-3	
	Number of BUFGs	Number of CLKDLLs	Number of BUFGs	Number of DCMs
1x	0	0	0	0
2x	1	1	1	1
4x	1	2	2	1

Output Buffer Type

You can select the type of output buffer used for the output clock and data pins. The types of output data buffers that are supported by the ILA/ATC cores depend on the device family (Table 3-5).

Table 3-5: **ILA/ATC Output Buffer Types by Device Family**

Buffer Type	Virtex-II, Virtex-II Pro, Virtex-4, Spartan-3	Virtex-E, Spartan-IIE	Virtex, Spartan-II
LVTTL 3.3V 24mA Fast	Yes	Yes	Yes
LVTTL 3.3V 12mA Fast	Yes	Yes	Yes
LVCMOS 3.3V 24mA Fast	Yes	No	No
LVCMOS 3.3V 12mA Fast	Yes	No	No
LVCMOS 2.5V 24mA Fast	Yes	No	No
LVCMOS 2.5V 12mA Fast	Yes	Yes	Yes
LVCMOS 1.8V 16mA Fast	Yes	No	No
LVCMOS 1.8V 12mA Fast	Yes	Yes	No
LVDCI 3.3V	Yes	No	No
LVDCI 2.5V	Yes	No	No
LVDCI 1.8V	Yes	No	No

Number of Data Pins

The ILA/ATC core can use 4, 8, 12, 16, or 20 output data pins for external capture.

Output Clock and Data Pin Locations

The clock and data pins are instantiated inside the ILA/ATC core for your convenience. This means that although you do not have to manually bring the clock and data pins through every level of hierarchy to the top-level of your design, you do need to specify the location of these pins in the Core Generator. The pin locations are then added to the `*.ncf` file of the ILA/ATC core.

Data Width and Depth

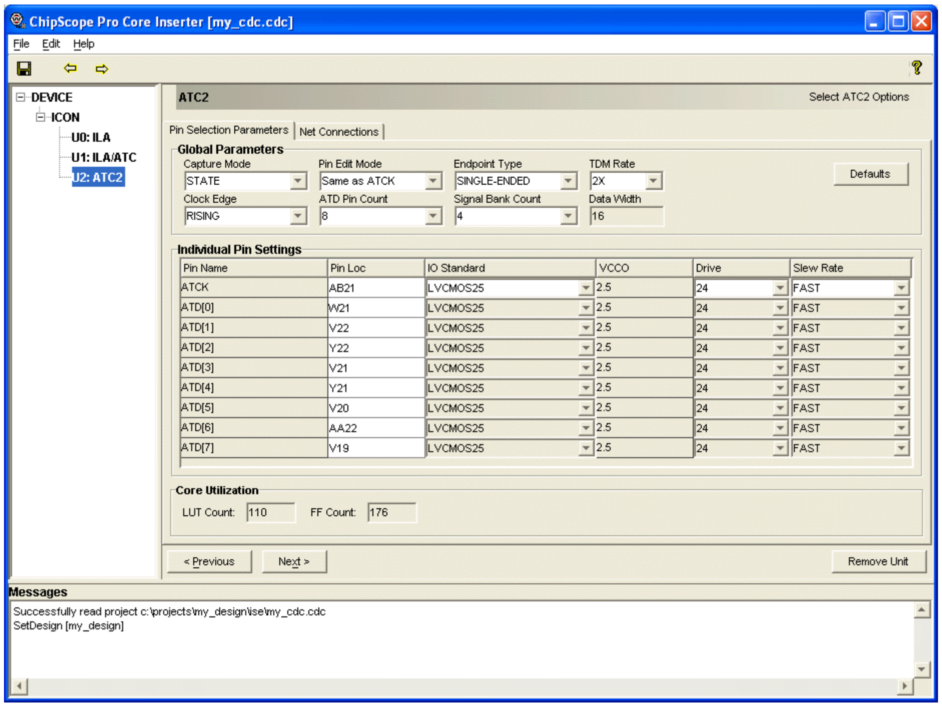
The data width and depth of the ILA/ATC core depend on the transmit rate and the number of data pins (Table 3-6).

Table 3-6: **ILA/ATC Core Capabilities**

Number of Data Pins	Transmit Rate	Max Width of DATA Port	Max DATA Depth (with timestamps)
4	1x	3	2,097,120 (1,048,560)
4	2x	5	1,048,560 (1048560)
4	4x	11	524,280 (262,136)
8	1x	7	2,097,120 (1,048,560)
8	2x	13	1,048,560 (524,280)
8	4x	27	524,280 (262,136)
12	1x	11	2,097,120 (1,048,560)
12	2x	21	1,048,560 (524,280)
12	4x	43	524,280 (262,136)
16	1x	15	2,097,120 (1,048,560)
16	2x	29	1,048,560 (524,280)
16	4x	59	524,280 (262,136)
20	1x	19	2,097,120 (1,048,560)
20	2x	37	1,048,560 (524,280)
20	4x	75	524,280 (262,136)

Choosing ATC2 Data Capture Settings

If you are inserting an ILA/ATC core, the Capture Settings look like those in Figure 3-14.



Global Parameters

Capture Mode	Pin Edit Mode	Endpoint Type	TDM Rate
STATE	Same as ATCK	SINGLE-ENDED	2X
Clock Edge	**ATD Pin Count**	**Signal Bank Count**	**Data Width**
RISING	8	4	16

Individual Pin Settings

Pin Name	Pin Loc	IO Standard	VCCO	Drive	Slew Rate
ATCK	AB21	LVCMOS25	2.5	24	FAST
ATD[0]	W21	LVCMOS25	2.5	24	FAST
ATD[1]	V22	LVCMOS25	2.5	24	FAST
ATD[2]	Y22	LVCMOS25	2.5	24	FAST
ATD[3]	V21	LVCMOS25	2.5	24	FAST
ATD[4]	Y21	LVCMOS25	2.5	24	FAST
ATD[5]	V20	LVCMOS25	2.5	24	FAST
ATD[6]	AA22	LVCMOS25	2.5	24	FAST
ATD[7]	V19	LVCMOS25	2.5	24	FAST

Core Utilization

LUT Count: 110 FF Count: 176

Messages

```
Successfully read project c:\projects\my_design\ise\my_cdc.cdc
SetDesign [my_design]
```

Figure 3-14: **ATC2 Core Data Capture Settings**

Capture Mode

The capture mode of the ATC2 core can be set to either "State" mode for synchronous data capture to the CLK input signal or to "Timing" mode for asynchronous data capture. In "State" mode, the data path through the ATC2 core uses pipeline flip-flops that are clocked on the CLK input port signal. In "Timing" mode, the data path through the ATC2 core is composed purely of combinational logic all the way to the output pins. Also, in "Timing" mode, the ATCK pin is used as an extra data pin.

Clock Edge

The ATC2 unit can use either the rising or falling edges of the CLK signal to capture data and run the internal calibration logic. The Clock Edge pull-down list is used to select either the rising or falling edge of the `CLK` signal as the clock source for the ATC2 core.

Pin Edit Mode

The Pin Edit Mode parameter is a time saving feature that allows you to change the IO Standard, Drive, and Slew Rate pin parameters on individual pins or together as a group of pins. Setting the Pin Edit Mode to "Individual" allows you to edit the parameters of each pin independently from one another. Setting the mode to "Same as ATCK" will allow you to change the ATCK pin parameters and will force all ATD pins to the same settings. You need to set unique pin locations for each individual pin regardless of the Pin Edit Mode.

ATD Pin Count

The ATC2 core can implement any number of ATD output pins in the range of 4 through 128.

Endpoint Type

The Endpoint Type setting is used to control whether single-ended or differential output drivers are used on the ATCK and ATD output pins. All ATCK and ATD pins must use the same driver endpoint type.

Signal Bank Count

The ATC2 core contains an internal, run-time selectable data signal bank multiplexer. The Signal Bank Count setting is used to denote the number of data input ports or signal banks the multiplexer will implement. The valid Signal Bank Count values are 1, 2, 4, 8, 16, or 32.

TDM Rate

The ATC2 core does not use on-chip memory resources to store the captured trace data. Instead, it transmits the data to be captured by an Agilent logic analyzer that is attached to the FPGA pins using a special probe connector. The data can be transmitted out the device pins at the same rate as the incoming DATA port (TDM rate = 1x) or twice the rate as the DATA port (TDM rate = 2x). The TDM rate can be set to “2x” only when the capture mode is set to “State”.

Data Width

The width of each input data port of the ATC2 core depends on the capture mode and the TDM rate. In “State” mode, the width of each data port is equal to (ATD pin count) * (TDM rate). In “Timing” mode, the width of each data port is equal to (ATD pin count + 1) * (TDM rate) since the ATCK pin is used as an extra data pin.

Pin Parameters

The output clock (ATCK) and data (ATD) pins are instantiated inside the ATC2 core for your convenience. This means that although you do not have to manually bring the ATCK and ATD pins through every level of hierarchy to the top-level of your design, you do need to specify the location and other characteristics of these pins in the Core Generator. These pin attributes are then added to the `*.ncf` file of the ATC2 core. Using the settings in the Pin Parameters table, you can control the location, I/O standard, output drive and slew rate of each individual ATCK and ATD pin.

Pin Name

The ATC2 core has two types of output pins: ATCK and ATD. The ATCK pin is used as a clock pin when the capture mode is set to “State” and is used as a data pin when the capture mode is set to “Timing”. The ATD pins are always used as data pins. The names of the pins cannot be changed.

Pin Loc

The Pin Loc column is used to set the location of the ATCK or ATD pin.

IO Standard

The IO Standard column is used to set the I/O standard of each individual ATCK or ATD pin. The I/O standards that are available for selection depend on the device family and driver endpoint type. The names of the I/O standards are the same as those in the IOSTANDARD section of the *Constraints Guide* (http://toolbox.xilinx.com/docsan/xilinx6/books/docs/cgd/cgd.pdf) in the *Xilinx ISE 6 Software Manuals and Help - PDF Collection.*

VCCO

The VCCO column setting denotes the output voltage of the pin driver and depends on the IO Standard selection.

Drive

The Drive column setting denotes the maximum output current drive of the pin driver and ranges from 2 to 24 mA, depending on the IO Standard selection.

Slew Rate

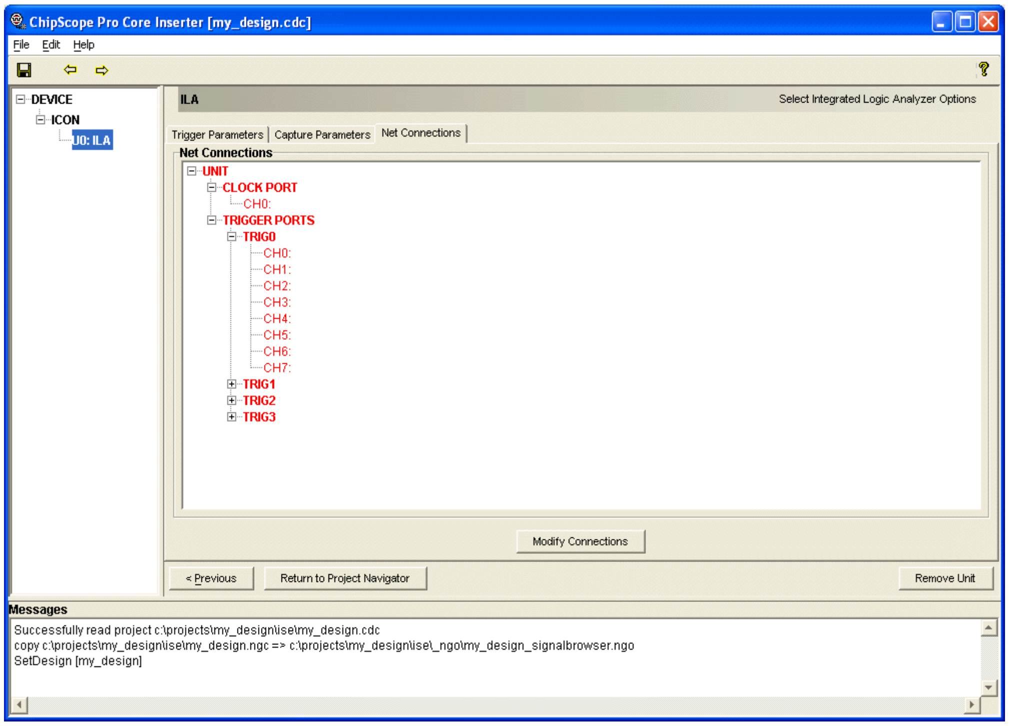
The Slew Rate column can be set to either FAST or SLOW for each individual ATCK or ATD pin.

Core Utilization

The ATC2 core generator has a core resource utilization monitor that estimates the number of look-up tables (LUTs) and flip-flops (FF) used by the ATC2 core, depending on the parameters used. The ATC2 core never uses block RAM or additional clock resources (for example, BUFG or DCM components).

Choosing Net Connections for ILA or ILA/ATC Signals

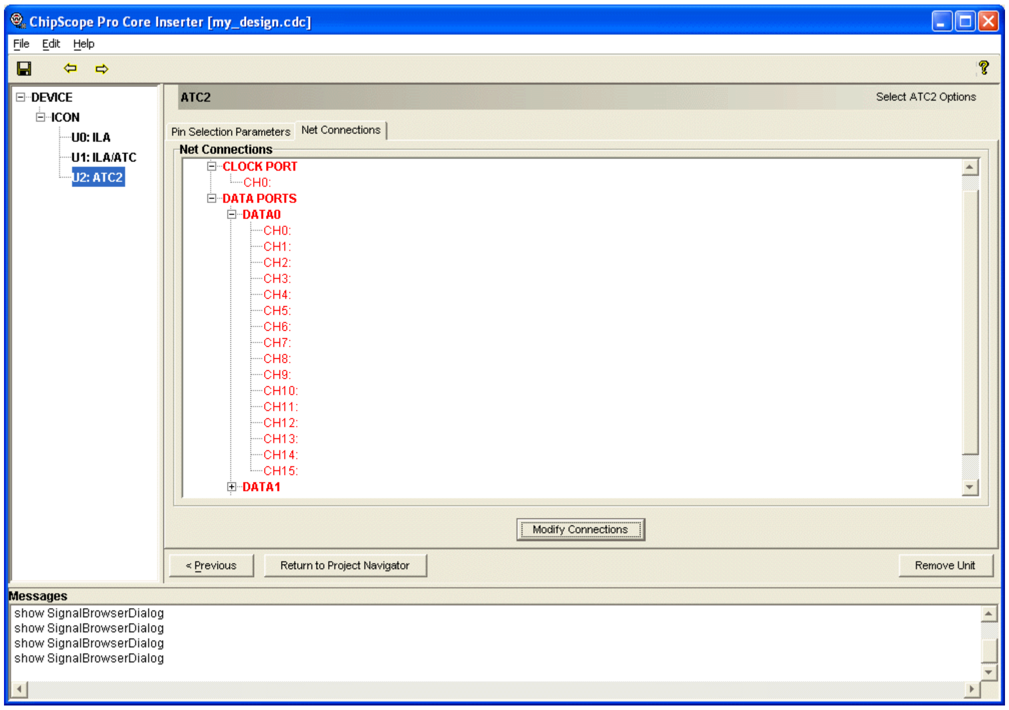
The **Net Connections** tab (Figure 3-15) allows you to choose the signals to connect to the ILA or ILA/ATC core. If trigger is separate from data, then Clock, Trigger, and Data must be specified. When trigger equals data, only Clock and Trigger/Data must be specified. Double-click on the **Clock Net** label or click on the plus sign (**+**) next to it to expand as shown in Figure 3-15. No connection has been made, so the connection appears in red.



```
Successfully read project c:\projects\my_design\ise\my_design.cdc
copy c:\projects\my_design\ise\my_design.ngc => c:\projects\my_design\ise\_ngo\my_design_signalbrowser.ngo
SetDesign [my_design]
```

Figure 3-15: **ILA and ILA/ATC Net Connections**

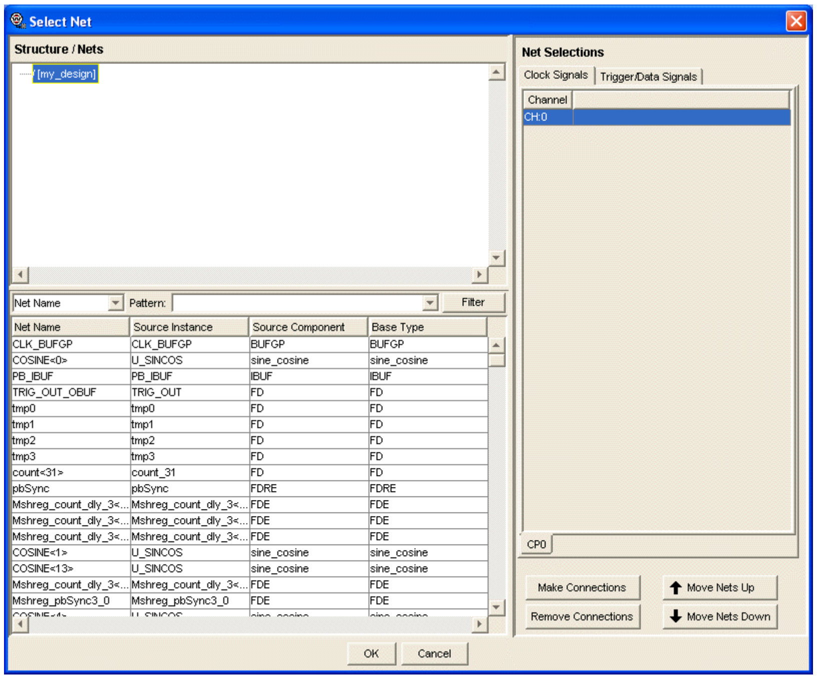
The ATC2 **Net Connections** tab (Figure 3-16) allows you to choose the signals to connect to the ATC2 core. The Clock and Data ports must be specified. Double-click on the **Clock Net** label or click on the plus sign (**+**) next to it to expand as shown in Figure 3-16. No connection has been made, so the connection appears in red.



```
show SignalBrowserDialog
show SignalBrowserDialog
show SignalBrowserDialog
show SignalBrowserDialog
```

Figure 3-16: **ATC2 Net Connections**

To change any core connection, select **Modify Connections**. The Select Net dialog box now appears (Figure 3-17).



Net Name	Source Instance	Source Component	Base Type
CLK_BUFGP	CLK_BUFGP	BUFGP	BUFGP
COSINE<0>	U_SINCOS	sine_cosine	sine_cosine
PB_IBUF	PB_IBUF	IBUF	IBUF
TRIG_OUT_OBUF	TRIG_OUT	FD	FD
tmp0	tmp0	FD	FD
tmp1	tmp1	FD	FD
tmp2	tmp2	FD	FD
tmp3	tmp3	FD	FD
count<31>	count_31	FD	FD
pbSync	pbSync	FDRE	FDRE
Mshreg_count_dly_3<...	Mshreg_count_dly_3<...	FDE	FDE
Mshreg_count_dly_3<...	Mshreg_count_dly_3<...	FDE	FDE
Mshreg_count_dly_3<...	Mshreg_count_dly_3<...	FDE	FDE
COSINE<1>	U_SINCOS	sine_cosine	sine_cosine
COSINE<13>	U_SINCOS	sine_cosine	sine_cosine
Mshreg_count_dly_3<...	Mshreg_count_dly_3<...	FDE	FDE
Mshreg_pbSync3_0	Mshreg_pbSync3_0	FDE	FDE

Figure 3-17: **Select Net Dialog Box**

This dialog box provides an easy interface to choose nets to connect to the ILA, ILA/ATC or ATC2 cores. The hierarchical structure of the design can be traversed using the Structure/Nets pane on the upper left of the Select Net dialog box. All the design's nets of the selected structure hierarchy level appear in the table on the lower left pane of the Select Net dialog box. The following net information is displayed in this table:

- *Net Name:* The name of the net as it appears in the EDIF netlist. The net name may be different than the corresponding signal name in the HDL source due to renaming and other optimizations during synthesis.
- *Source Instance:* The instance name of the lower-level hierarchical component from which the net at the current level of hierarchy is driven. The source instance does not necessarily describe the originating driver of the net.
- *Source Component:* The type of the component described by the Source Instance.
- *Base Type:* The type of the lowest level driving component of the net. The base type is either a *primitive* or *black box* component.

All of the net identifiers described above can be filtered for key phrases using the **Pattern** text box and **Filter** button. Also, nets can be sorted in ascending and descending order based on the various net identifiers by selecting the appropriate net identifier button in the column headers of the net selection table.

Note: The net names are sorted in alpha-numeric or "bus element" order whenever possible. Common delimiters such as "[", "(", etc., are used to identify possible bus element nets.

The tabs for **Clock**, **Data**, and **Trigger** inputs of the ILA or ILA/ATC core appear in the pane at the upper right of the Select Net dialog box. If you are selecting nets for an ATC2 core, then only the Clock and Data input port categories will appear at the upper right of the Select Net dialog box. If multiple trigger or data ports exist, there will be multiple sub-tabs on the bottom of the Net Selections pane, respectively. Nets that are selected at a given level of hierarchy can be connected to inputs of the ILA, ILA/ATC, or ATC2 capture cores by following these steps:

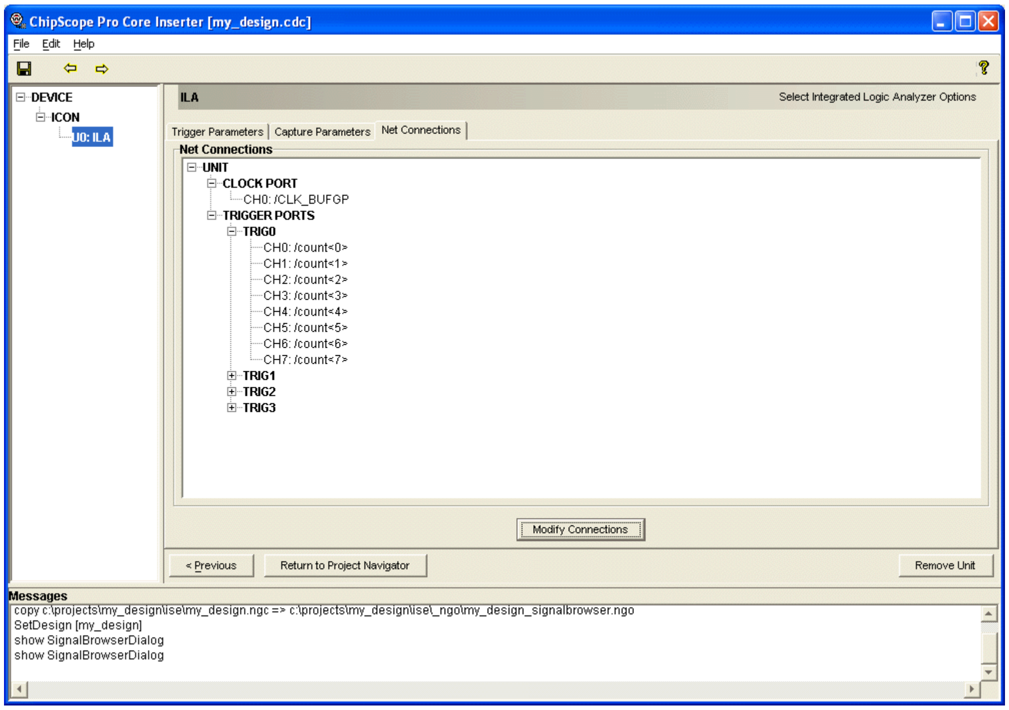
1. In the lower-left table of the Select Net dialog box, select the net(s) that you want to connect to the capture core.

   Note: You can select multiple nets to connect to an equivalent number of capture core input connections. Hold down the **Shift** key and use the left mouse button to select contiguous nets. Use a combination of the **Ctrl** key and left mouse button to select non-contiguous nets. You can also connect a single net to multiple capture core input signals by selecting a single net and multiple capture core port signals.
2. In the upper-right tabbed panel of the Select Net dialog box, select the desired capture core input category: Clock, Trigger (trigger port tab if applicable), Data (or Trigger/Data, if trigger is same as data).
3. In the right-hand table of capture core inputs, select the channel(s) that you want to connect to the selected net(s).

   Note: You can select multiple capture core inputs to connect to an equivalent number of nets. Hold down the **Shift** key and use the left mouse button to select contiguous ILA core inputs. Use a combination of the **Ctrl** key and left mouse button to select non-contiguous ILA core inputs. You can also connect a single net to multiple capture core input signals by selecting a single net and multiple capture core port signals.
4. In the lower-right part of the Select Net dialog box, click the **Make Connections** button to make a connection between the selected nets and capture core inputs.

Use the **Remove Connections** button to remove any existing connections. Use the **Move Up** and **Move Down** buttons to reorder the position of any selected connection. Once the desired net connections have been made, click **OK** to return to the main Core Inserter window.

All the nets for Trigger and Data must be chosen in this fashion. After you have chosen all the nets for a given bus, the ILA, ILA/ATC, or ATC2 bus name changes from red to black (see Figure 3-18).



```
Net Connections
UNIT
  CLOCK PORT
    CH0: /CLK_BUFGP
  TRIGGER PORTS
    TRIG0
      CH0: /count<0>
      CH1: /count<1>
      CH2: /count<2>
      CH3: /count<3>
      CH4: /count<4>
      CH5: /count<5>
      CH6: /count<6>
      CH7: /count<7>
    TRIG1
    TRIG2
    TRIG3
```

Messages

```
copy c:\projects\my_design\ise\my_design.ngc => c:\projects\my_design\ise\_ngo\my_design_signalbrowser.ngo
SetDesign [my_design]
show SignalBrowserDialog
show SignalBrowserDialog
```

Figure 3-18: **Specifying Data Connections**

After specifying the Clock, Trigger, and Data nets, click **Next**.

If you are using the Core Inserter in stand-along mode, a dialog box appears asking if you want to proceed with Core Insertion . If **Yes**, the cores are generated, inserted into the netlist, and an NGO file is created with the EDIF2NGD tool. Details of this process can be viewed in the Messages pane at the bottom of the window. A Core Generation Complete message in the Messages pane indicates successful insertion of ChipScope cores.

If you are using the Core Inserter as part of the Project Navigator mode, a dialog box appears asking if you want to return to Project Navigator. If **Yes**, the Core Inserter settings are saved and you are returned to the Project Navigator tool. The actual core generation and insertion processes take place in the proper sequence as deemed necessary by the Project Navigator tool.

Adding Units

Each device can support up to 15 ILA, ILA/ATC, or ATC2 units, depending on block RAM availability and unit parameters. To add another ILA unit to the project, select **Edit** → **New ILA Unit**, or go to the ICON Options window by clicking on ICON in the tree on the left pane (Figure 3-8, page 3-10) and clicking the **New ILA Unit** button.

To add another ILA/ATC unit to the project, select **Edit** → **New ILA/ATC Unit**, or go to the ICON Options window by clicking on **ICON** in the tree on the left pane (Figure 3-8, page 3-10) and clicking the **New ILA/ATC Unit** button.

To add another ATC2 unit to the project, select **Edit** → **New ATC2 Unit**, or go to the ICON Options window by clicking on **ICON** in the tree on the left pane (Figure 3-8, page 3-10) and clicking the **New ATC2 Unit** button.

You can set up the parameters for the additional units by using the same procedure as described above.

Inserting Cores into Netlist

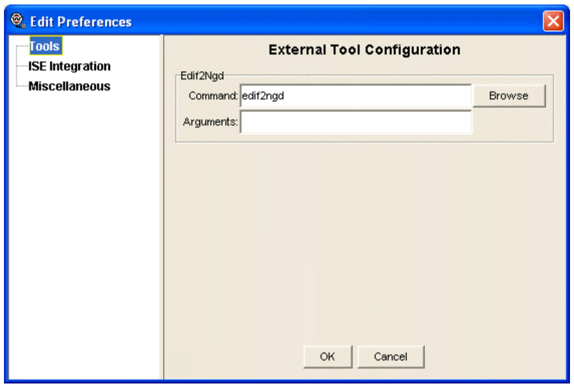
The core insertion step can be invoked by selecting the **Insert** → **Insert Core** menu option or by clicking **Insert Core** on the toolbar.

Note: If you are using the ChipScope Pro Core Inserter flow in the Xilinx ISE 6.3i Project Navigator tool, click **Return to Project Navigator** instead of selecting the **Insert** → **Insert Core** option. The insertion of the cores will happen automatically as part of the **Translate** process in the Project Navigator tool. Refer to Using the Core Inserter with ISE Project Navigator on Windows, page 3-1 for more details.

Managing Project Preferences

The preference settings are organized into three categories: *Tools, ISE Integration, and Miscellaneous*. These preference settings are shown in Figure 3-19, Figure 3-20, page 3-29, and Figure 3-21, page 3-29, respectively.

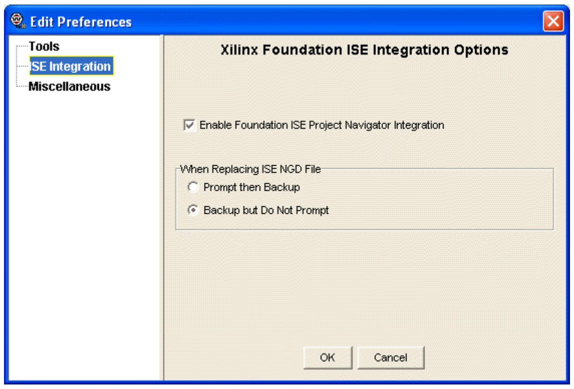
The Tools section (Figure 3-19) contains settings for the command line arguments used by the Core Inserter to launch the EDIF2NGD tool.



Command: edif2ngd
Arguments:

Figure 3-19: **Core Inserter Tools Preference Settings**

The ISE Integration section (Figure 3-20) contains settings that affect how the Core Inserter integrates with the Xilinx ISE Project Navigator tool. When ISE integration is enabled (the default), the Core Inserter automatically searches the current working directory for ISE temporary netlist directory called _ngo. If a valid ISE _ngo directory is found, the Core Inserter project will be set up automatically to overwrite the intermediate NGD files of the ISE project with those produced by the Core Inserter. The ISE Integration preferences can be set by the user to prompt the user before overwriting any intermediate NGD files.



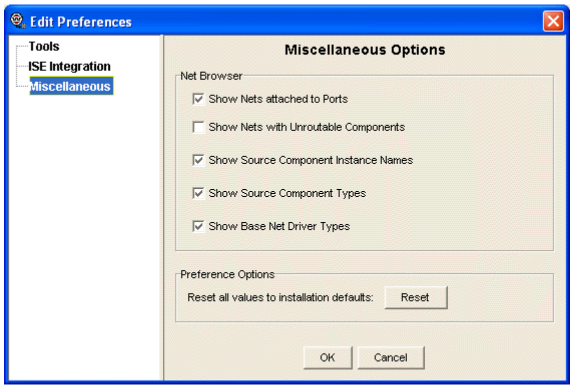
Xilinx Foundation ISE Integration Options

[✓] Enable Foundation ISE Project Navigator Integration

When Replacing ISE NGD File
( ) Prompt then Backup
(●) Backup but Do Not Prompt

Figure 3-20: **Core Inserter ISE Integration Preference Settings**

The Miscellaneous preferences section (Figure 3-21) contains other settings that affect how the Core Inserter operates. For instance, the Core Inserter can be set up by the user to display the ports in the Select Net dialog box. This may be desired if the cores are being inserted into a lower-level EDIF netlist, instead of the top level. These port nets are shown in gray in the Select Net dialog box. The Core Inserter can also be set up to display nets that are illegal for connection in the Select Net dialog box. When this preference option is enabled, any illegal nets are shown in red in the Select Net dialog box.



Miscellaneous Options

Net Browser
[✓] Show Nets attached to Ports
[ ] Show Nets with Unroutable Components
[✓] Show Source Component Instance Names
[✓] Show Source Component Types
[✓] Show Base Net Driver Types

Preference Options
Reset all values to installation defaults:

Figure 3-21: **Core Inserter Miscellaneous Preference Settings**

Also, the Core Inserter can be set up by the user to disable the display of source component instance names, source component types, and base net driver types in the Select Net dialog box. You can reset the Core Inserter project preferences to the installation defaults by clicking on the **Reset** button.

Chapter 4

Using the ChipScope Pro Analyzer

Analyzer Overview

The ChipScope Pro Analyzer tool interfaces directly to the ICON, ILA, ILA/ATC, IBA/OPB, IBA/PLB, VIO, and ATC2 cores (collectively called the ChipScope Pro cores). You can configure your device, choose triggers, setup the console, and view the results of the capture on the fly. The data views and triggers can be manipulated in many ways, providing an easy and intuitive interface to determine the functionality of the design.

Note: Even though the ChipScope Pro Analyzer tool will detect the presence of an ATC2 core, an Agilent Logic Analyzer attached to a Xilinx JTAG cable is required to control and communicate with the ATC2 core.

Analyzer Interface

The ChipScope Pro Analyzer interface consists of four parts:

- *Project tree* in the upper part of the split pane on the left side of the window
- *Signal browser* in the lower part of the split pane on the left side of the window
- *Message pane* at the bottom of the window
- *Main window* area

Both the project tree/signal browser split pane and the Message pane can be hidden by deselecting those options in the View menu. Additionally, the size of each pane can be adjusted by dragging the bar located between the panes to a new location. Each pane can be maximized or minimized by clicking on the arrow buttons on the pane separator bars.

Project Tree

The project tree is a graphical representation of the JTAG chain and the ChipScope Pro cores in the devices in the chain. Although all devices in the chain are displayed in the tree, only valid target devices (Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan-II, Spartan-IIE, Spartan-3 and their QPro variants) can contain ChipScope Pro cores and be operated upon. Leaf nodes in the tree appear when further operations are available.

For instance, a leaf node for each ChipScope Pro unit appears when that device is configured with a ChipScope Pro core-enabled bitstream. Context-sensitive menus are available for each level of hierarchy in the tree. To access the context-sensitive menu, right-click on the node in the tree. Device and unit renaming, child window opening, device configuration, and project operations can all be done through these menus.

To rename a device or core unit node in the project tree, right-click on the node and select **Rename**. To end the editing, press **Enter** or the up or down arrow key, or click on another node in the tree.

Signal Browser

The signal browser displays all the signals for the core selected in the project tree. Signals can be renamed, grouped into buses, and added to the various data views using context-sensitive menus in the signal browser.

Renaming Signals Buses, and Triggers Ports

To rename a signal, bus, or trigger port name in the signal browser, double click on it, or right-click and select **Rename**. To end the editing, press **Enter** or the up or down arrow key, or click on another node in the tree.

Adding/Removing Signals from Views

To remove all the signals from either the waveform or listing view, right-click on any data signal or bus in the signal browser and select **Clear All** → **Waveform** or **Clear All** → **Listing**. In the case of a VIO core, to remove all the signals from the VIO console, right-click on any signal or bus, and select **Clear All** → **Console.** Similarly, all signals and buses can be added to the views through the **Add All to View** menu options. Selected signals and buses can be added through the **Add to View** menu options.

To select a contiguous group of signals and buses, click on the first signal, hold down the **Shift** key, and click on the last signal in the group. To select a non-contiguous group of signals and buses, click on each of the signals/buses in turn while holding down the **Ctrl** key. When you use this method, the order of the signals in the bus are in the order in which you select them.

Combining and Adding Signals Into Buses

For ILA and IBA cores, only data signals can be combined into buses. For VIO cores, signals of a particular type can be grouped together to form buses. To combine signals into buses, select the signals using the **Shift** or **Ctrl** keys as described above. When the **Shift** key is used, the uppermost signal in the tree will be the LSB once the bus is created. If the **Ctrl** key is used, the signals will be in ordered in the bus the same order that they are clicked, the first signal being the LSB.

After you have selected the signals, right click on any selected signal and select **Add to Bus** → **New Bus**. A new bus will be created at the top of the **Data Signals and Buses** sub-tree (in the case of ILA and IBA cores), or at the top of that particular sub-tree (in the case of VIO). To add a signal or signals into an existing bus, select the signals and select **Add to Bus**, and then the bus name in the following submenu. Added signals always go on the MSB-end of the bus.

Reverse Bus Ordering

To reverse the order of the bits in a bus (i.e. make the LSB the MSB), right click on the bus and select **Reverse Bus Order**. The signal browser and all data views that contain that bus will be immediately updated, and the bus values recalculated.

Bus Radices

Each bus can be displayed in the data views in any one of the following radices:

- ASCII
- Binary
- Hexadecimal
- Octal
- Signed decimal
- Token
- Unsigned decimal

ASCII is only available if the number of bits in the bus is evenly divisible by 8. Changing the radix will change the bus radix in every data view it is resident.

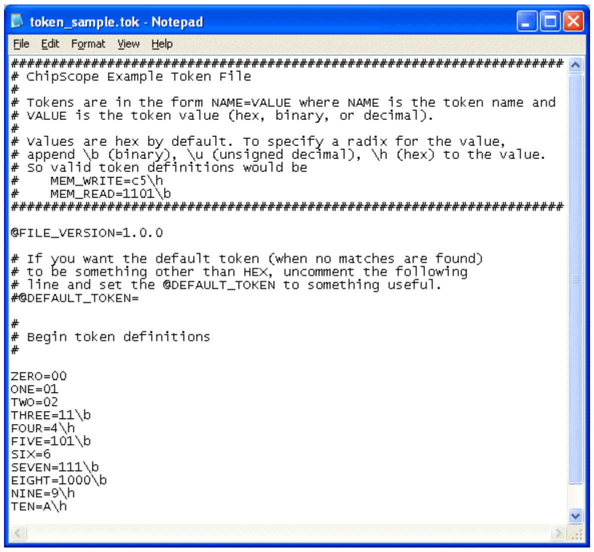
Signed and Unsigned

When either *signed* or *unsigned* is chosen as a bus radix, a small dialog box appears for you to enter three additional parameters: *scale factor*, *offset* and *precision*.

- *Scale factor* is a multiplier value to use when calculating bus values. For instance, if the LSB of a 4-bit bus is 0010, and the scale factor is set to 2.0, the actual displayed bus value will be 4 (given a precision of 0). If the scale factor is set to 0.1, then the actual displayed bus value will be 0.2 (given a precision of 1). The default scale factor is 1.0.
- *Offset* is a constant value that will be added to the scaled bus value. The default offset is 0.
- *Precision* is the number of decimal places to display after the decimal point. The default precision is 0.

Token

Tokens are string labels that are defined in a separate ASCII file and can be assigned to a particular bus value. These labels can be useful in applications such as address decoding and state machines. The token file (`.tok` extension) has a very simple format, and can be created or edited in any text editor. An example token file is provided in the token directory in the ChipScope Pro install path (Figure 4-1).



```
#######################################################################
# ChipScope Example Token File
#
# Tokens are in the form NAME=VALUE where NAME is the token name and
# VALUE is the token value (hex, binary, or decimal).
#
# Values are hex by default. To specify a radix for the value,
# append \b (binary), \u (unsigned decimal), \h (hex) to the value.
# So valid token definitions would be
#    MEM_WRITE=c5\h
#    MEM_READ=1101\b
#######################################################################

@FILE_VERSION=1.0.0

# If you want the default token (when no matches are found)
# to be something other than HEX, uncomment the following
# line and set the @DEFAULT_TOKEN to something useful.
#@DEFAULT_TOKEN=

#
# Begin token definitions
#

ZERO=00
ONE=01
TWO=02
THREE=11\b
FOUR=4\h
FIVE=101\b
SIX=6
SEVEN=111\b
EIGHT=1000\b
NINE=9\h
TEN=A\h
```

Figure 4-1: **Example Token File**

Tokens are chosen by selecting a bus, then choosing **Bus Radix** → **Token** from the right-click menu. A dialog box opens and you can choose the token file. If the bus is wider than the tokens specify (such as choosing 4-bit tokens for an 8-bit bus) the upper bits are assumed 0 for the tokens to apply. Figure 4-2 shows such a waveform, with the example token file in Figure 4-1 applied to an 8-bit bus.

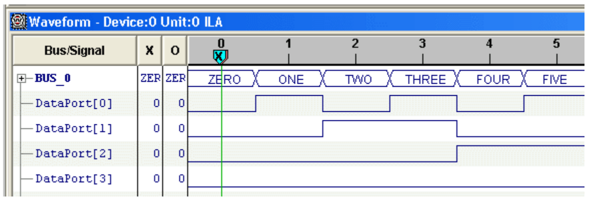


Figure 4-2: **Example Waveform with Tokens**

Deleting Buses

To delete a bus, right click on it and select **Delete Bus**. The bus is immediately deleted in every data view it is resident.

Type and Persistence (VIO only)

VIO signals have two additional properties: *Type* and *Persistence*. See VIO Bus/Signal Activity Persistence, page 4-36 for explanations of these properties.

Message Pane

The Message pane displays a scroll list of status messages. Error messages appear in red. The Message pane can be resized by dragging the split bar above it to a new location. This also changes the height of the project tree/signal browser split pane.

Main Window Area

The main window area can display multiple child windows (such as Trigger, Waveform, Listing, Plot windows) at the same time. Each window can be resized, minimized, maximized, and moved as needed.

Analyzer Menu Features

Working with Projects

Projects hold important information about the ChipScope Pro Analyzer program state, such as signal naming, signal ordering, bus configurations, and trigger conditions. They allow you to conveniently store and retrieve this information between Analyzer sessions

When you first run the ChipScope Pro Analyzer tool, a new project is automatically created and is titled **new project**. To open an existing project, select **File** → **Open Project**, or select one of the recently used projects in the **File** menu. The title bar of the Analyzer and the project tree displays the project name. If the new project is not saved during the course of the session, a dialog box appears when the Analyzer is about to exit, asking you if you wish to save the project.

Creating and Saving A New Project

To create a new project, select **File** → **New Project.** A new project called **new project** is created and made active in the Analyzer. To save the new project under a different name, select **File** → **Save Project**. The project file will have a `.cpj` extension.

Saving Projects

To rename the current project, or to save a copy to another filename, select **File** → **Save Project As** (Figure 4-3), type the new name in the File name dialog box, and click **Save**.

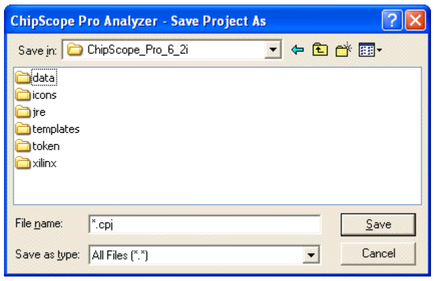


Figure 4-3: **Saving a Project**

Printing Waveforms

One of the features of ChipScope Pro 6.3i is the ability to print a captured data waveform (as shown in Figure 4-4) by using the **File** → **Print** menu option (as shown in Figure 4-5). Selecting the **File** → **Print** menu option starts the Print Wizard.

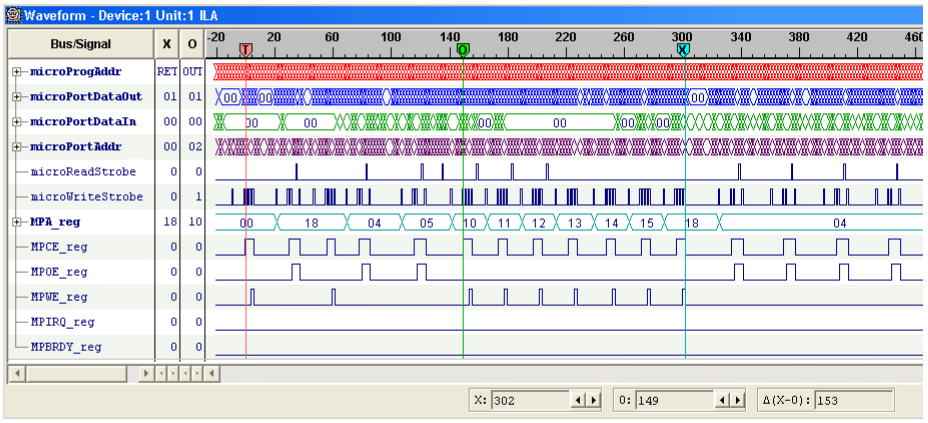


Figure 4-4: **Example Waveform**

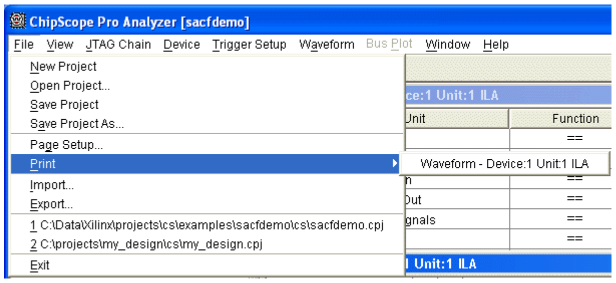

ChipScope Pro Analyzer [sacfdemo]
File View JTAG Chain Device Trigger Setup Waveform Bus Plot Window Help
New Project
Open Project...
Save Project
Save Project As...
Page Setup...
Print
Waveform - Device:1 Unit:1 ILA
Import...
Export...
1 C:\Data\Xilinx\projects\cs\examples\sacfdemo\cs\sacfdemo.cpj
2 C:\projects\my_design\cs\my_design.cpj
Exit

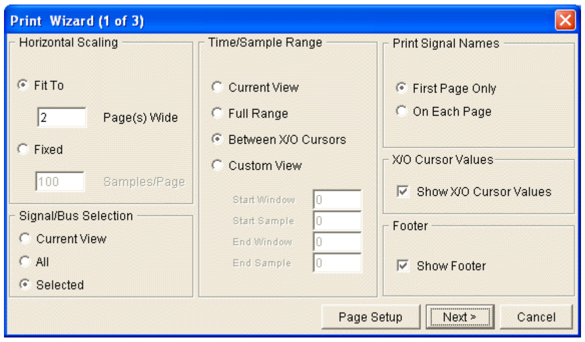
Figure 4-5: **Selecting the File → Print Option**

The Print Wizard consists of three consecutive windows:

1. (1 of 3) is the Print options and settings window (Figure 4-6)
2. (2 of 3) is the Print waveform printout preview navigator window (Figure 4-8, page 4-9)
3. (3 of 3) is the Print confirmation window (Figure 4-11, page 4-11)

Print Wizard (1 of 3) Window

The first Print Wizard window (shown in Figure 4-6) is used to set up various waveform printing options. The following sections describe these waveform printing options in more detail.



Print Wizard (1 of 3)

Horizontal Scaling
[✓ Fit To] 2 Page(s) Wide
Fixed 100 Samples/Page

Signal/Bus Selection
Current View
All
[✓ Selected]

Time/Sample Range
Current View
Full Range
[✓ Between X/O Cursors]
Custom View
Start Window 0
Start Sample 0
End Window 0
End Sample 0

Print Signal Names
[✓ First Page Only]
On Each Page

X/O Cursor Values
[✓ Show X/O Cursor Values]

Footer
[✓ Show Footer]

Figure 4-6: **Print Wizard (1 of 3)**

Horizontal Scaling

You can control the amount of waveform data that prints to each column of pages using one of two methods:

- *Fit To*: Fit the waveform to one or more columns of pages
- *Fixed*: Fit a specific number of waveform samples on each column of pages

The default fits the entire waveform printout to a single column of pages wide.

Signal/Bus Selection

You can control which signals and buses will be present in the waveform printout using one of three methods:

- *Current View*: Print waveform data for all of the signals and buses in the current view of the waveform window
- *All*: Print waveform data for all of the signals and buses available in the entire core unit
- *Selected*: Print waveform data for only those signals and buses that are currently selected in the waveform window

The default prints waveform data using the Current View method.

Time/Sample Range

You can control the range of time units or number of samples printed using one of four methods:

- *Current View*: Print waveform data using the same range of samples that is present in the current waveform view.
- *Full Range*: Print waveform data using a range of samples consisting of all samples in the entire sample buffer.
- *Between X/O Cursors*: Print waveform data using a range of samples starting with the X cursor and ending with the O cursor (or vice versa)
- *Custom View*: Print waveform data using a range of samples defined by a starting window and sample number and an ending window and sample number.

The default prints waveform data using the Current View method.

Print Signal Names

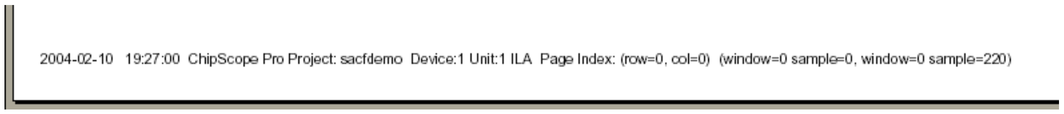
You can choose to print the signal names (and X/O cursor values) on each page or only on the first page. Printing the X/O cursor values on the first page only is useful when you assemble multiple printed pages together to form a larger multi-dimensional plot.

X/O Cursor Values

You can also choose whether or not to include the X/O cursor values in the waveform printout. If you choose to display the X/O cursor values in the waveform printout, then they will either appear on each page or only on the first page, depending on the Print Signal Names setting (see previous section "Print Signal Names").

Footer

You can enable or disable the inclusion of a footer at the bottom of each page by selecting the Show Footer checkbox. An example of the information that appears in the footer is shown in Figure 4-7.



2004-02-10 19:27:00 ChipScope Pro Project: sacfdemo Device:1 Unit:1 ILA Page Index: (row=0, col=0) (window=0 sample=0, window=0 sample=220)

Figure 4-7: **Waveform Printout Footer Example**

Navigation Buttons

The buttons at the bottom of the Print Wizard (1 of 3) window (Figure 4-6, page 4-7) are defined as follows:

- *Page Setup*: Opens the page setup window (refer to Figure 4-12, page 4-11)
- *Next*: Opens the Print Wizard (2 of 3) window
- *Cancel*: Closes the Print Wizard window without printing.

Clicking on the **Next** button takes you to the Print Wizard (2 of 3) window, described in Print Wizard (2 of 3) Window, page 4-9.

Print Wizard (2 of 3) Window

The second Print Wizard window (shown in Figure 4-8) shows a preview of the waveform printout.

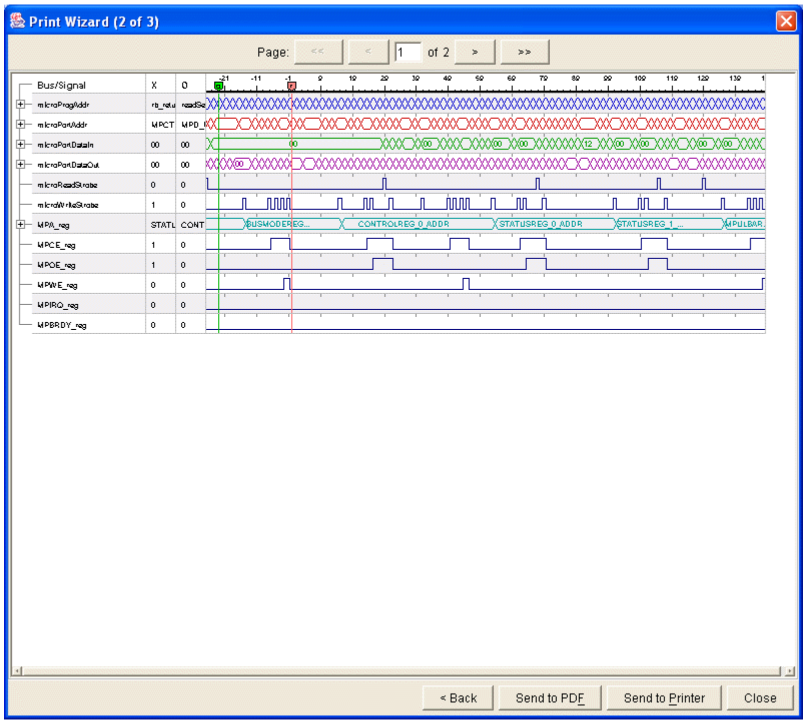


Figure 4-8: **Print Wizard (2 of 3)**

Page Preview Buttons

The buttons at the top of the page control which page of the waveform printout is being previewed as follow:

- The << and >> buttons go to the first and last preview pages, respectively
- The < and > buttons go to the previous and next preview pages, respectively
- The text box in the middle can be used to go to a specific preview page

Navigation Buttons

The buttons at the bottom of the Print Wizard (2 of 3) window (Figure 4-8) are defined as follows:

- *Back*: Returns to the Print Wizard (1 of 3) window
- *Send to PDF*: Opens the Print Wizard (3 of 3) window for writing directly to a PDF File
- *Send to Printer*: Opens the Print Wizard (3 of 3) window for sending to a printer
- *Close*: Closes the Print Wizard window without printing

Bus Expansion and Contraction

You can manipulate the waveform by expanding and contracting the buses in the print preview window. For example, if you expand a bus in Figure 4-8, page 4-9 such that it pushes other signals/buses to another page, the total print preview page count at the top will change accordingly, as shown in Figure 4-9.

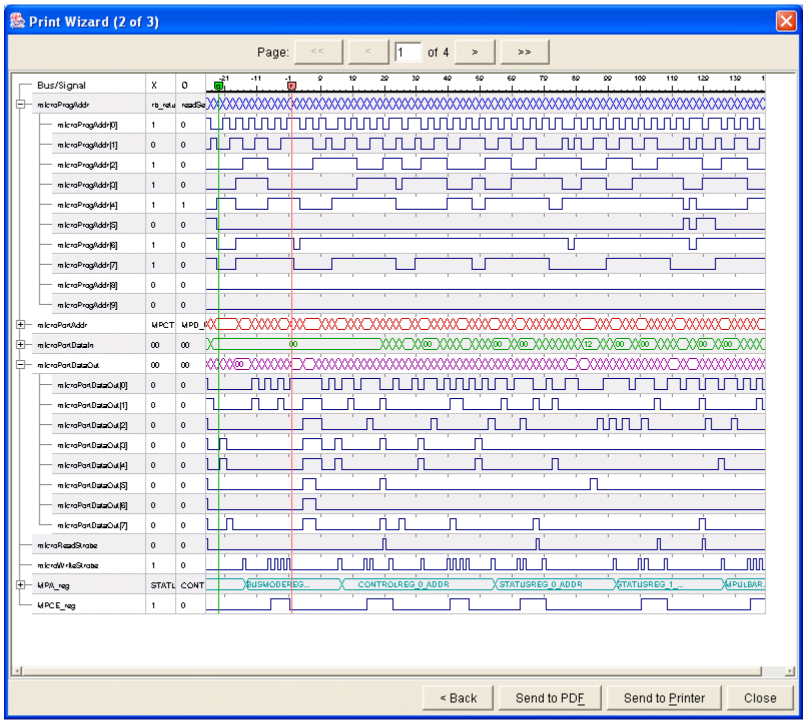
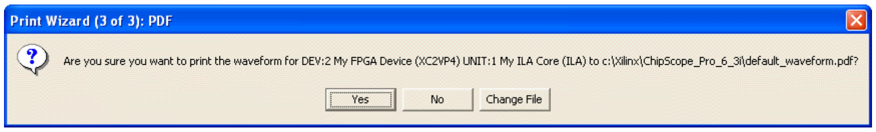


Figure 4-9: **Expanding Buses in Print Wizard (2 of 3)**

Print Wizard (3 of 3)

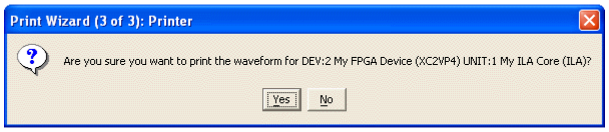
In the Print Wizard (2 of 3) window, clicking on the **Send to PDF** button goes to the Print Wizard (3 of 3) PDF confirmation window (see Figure 4-10). Clicking on the **Yes** button causes the waveform printout to be written to the specified PDF file while clicking on the **No** button returns you to the Print Wizard (2 of 3) window. Clicking on **Change File** opens a file browser window that allows you to select or create a new PDF file.



Are you sure you want to print the waveform for DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA) to c:\Xilinx\ChipScope_Pro_6_3i\default_waveform.pdf?

Figure 4-10: **Print Wizard (3 of 3) for Sending to a PDF File**

In the Print Wizard (2 of 3) window, clicking on the Send to Printer button goes to the Print Wizard (3 of 3) Printer confirmation window (see Figure 4-11). Clicking on the **Yes** button causes the waveform printout to be sent to the printer while clicking on the **No** button returns you to the Print Wizard (2 of 3) window.


Print Wizard (3 of 3): Printer

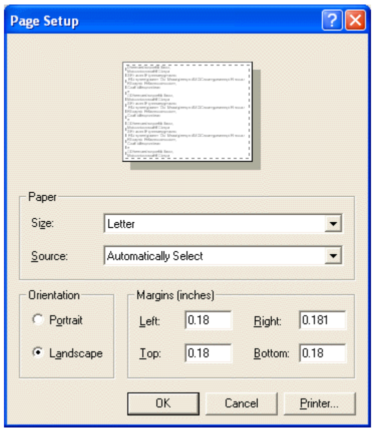
Are you sure you want to print the waveform for DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA)?

Figure 4-11: **Print Wizard (3 of 3) for Sending to a Printer**

Page Setup

The Page Setup window (shown in Figure 4-12) can be invoked either from the Print Wizard (1 of 3) window (Figure 4-6, page 4-7) or by using the **File** → **Page Setup** menu option.

Note: In the ChipScope Pro 6.3i Analyzer program, you can print only to the default system printer. Changing the target printer in the print setup window does not have any effect. To change printers, you must close the Analyzer program, change your default system printer, and restart the Analyzer program.


Page Setup

Paper

Size: Letter

Source: Automatically Select

Orientation

Portrait

Landscape [✓]

Margins (inches)

Left: 0.18 Right: 0.181

Top: 0.18 Bottom: 0.18

Figure 4-12: **Page Setup Window**

Importing Signal Names

At the start of a project, all of the signals in every ChipScope Pro core have generic names. You can rename the signals individually as described in Renaming Signals Buses, and Triggers Ports, page 4-2, or import a file that contains all the names of all the signals in one or more cores. The ChipScope Pro Core Generator, ChipScope Pro Core Inserter, Synplicity Certify, and the Xilinx FPGA Editor tools can create such files. To import signal names from a file, select **File** → **Import**. A Signal Import dialog box will appear (Figure 4-13).

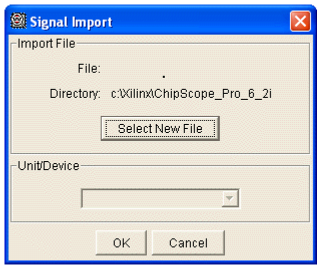


Figure 4-13: **Blank Signal Import Dialog Box**

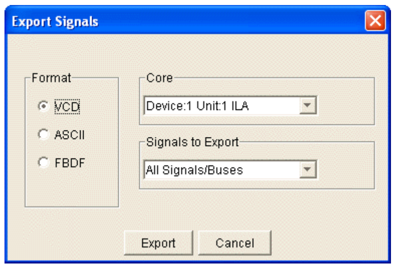
To select the signal import file, select **Select New File**. A file dialog box will appear for you to navigate and specify the signal import file. After you choose the file, the Unit/Device combo box will be populated, according to the core types specified in the signal import file. If the signal import file contains signal names for more than one core, the combo box will contain device numbers for all devices that contain only ChipScope Pro capture cores.

If the signal import file contains signal names for only one core, the combo box will be populated with names of the individual cores that match the type specified in the signal import file. If the import file is a file from Synplicity Certify, you will also have the option of choosing a device name from the Certify file as well as the device in the JTAG chain.

To import the signal names, click **OK**. If the parameters in the file do not match the parameters of the target core or cores, a warning message will be displayed. If you choose to proceed, the signal names will be applied to the cores as applicable.

Exporting Data

Captured data from an ILA or IBA core can be exported to a file, for future viewing or processing. To export data, select **File** → **Export**. The Export Signals dialog box appears (Figure 4-14).



Device:1 Unit:1 ILA

All Signals/Buses

Figure 4-14: **Export Signals Dialog Box**

Three formats are available: *value change dump* (**VCD**) format, *tab-delimited* **ASCII** format, or the Agilent Technologies Fast Binary Data Format (**FBDF**). To select a format, click its radio button. To select the target core to export, select it from the Core combo box.

Different sets of signals and buses are available for export. Use the Signals to Export combo box to select:

- All the signals and buses for that particular core, or
- All the signals and buses present in the core's waveform viewer, or
- All the signals and buses in the core's listing viewer, or
- All the signals and buses in the core's bus plot viewer

To export the signals, click **Export**. A file dialog box will appear from which you can specify the target directory and filename.

Closing and Exiting the Analyzer

To exit the ChipScope Pro Analyzer, select **File** → **Exit**. The current active project is automatically saved upon exit.

Viewing Options

The split pane on the left of the Analyzer window and the Message pane at the bottom of the window can both be hidden or displayed per the user's choice. Both are displayed the first time the Analyzer is launched. To hide the project tree/signal browser split pane, uncheck it under **View** → **Project Tree**. To hide the Message pane, uncheck it under **View** → **Messages**.

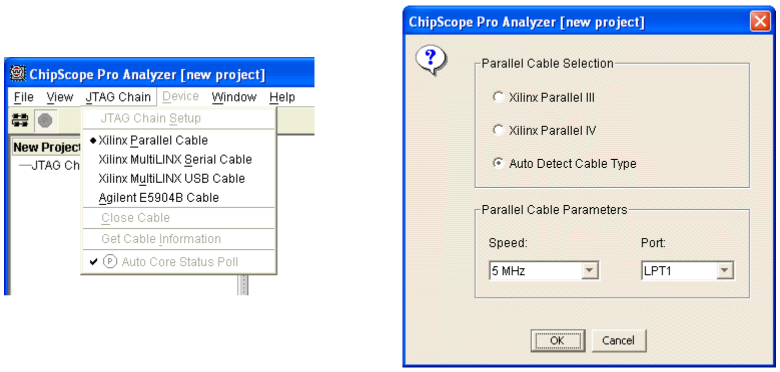
Opening a Parallel Cable Connection

The ChipScope Pro Analyzer supports the Parallel Cable III, Parallel Cable IV, MultiPRO (JTAG mode only), MultiLINX (JTAG mode only), and Agilent E5904B Trace Port Analyzer (Agilent E5904B TPA) cables. To open a connection to the Parallel Cable (including the MultiPRO cable), make sure the cable is connected to one of the computer's parallel ports. Select **JTAG Chain→ Xilinx Parallel Cable** (Figure 4-15). This pops up the Parallel Cable Selection configuration dialog box. You can choose the Parallel Cable III, Parallel Cable IV, or have the Analyzer autodetect the cable type.

Note: In order to open a connection to the MultiPRO cable, select either Parallel Cable IV or Auto Detect Cable Type.

If the Parallel Cable IV or Auto Detect Cable Type option is selected, you can choose the speed of the cable; the choices are 10 MHz, 5 MHz, 2.5 MHz (default), 1.25 MHz, or 625 kHz. Choose the speed that makes the most sense for the board in question. Type the printer port name in the Port selection box (usually the default LPT1 is correct) and click **OK**. If successful, the Analyzer queries the Boundary Scan chain to determine its composition (see Setting Up the Boundary Scan (JTAG) Chain, page 4-16).

If the Analyzer returns the error message "Failed to Open Communication Port", verify that the cable is connected to the correct LPT port. If you have not installed the Parallel Cable driver, follow the instructions in the ChipScope Pro Software installation program to install the required device driver software.


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Window Help
New Project
JTAG Ch
JTAG Chain Setup
Xilinx Parallel Cable
Xilinx MultiLINX Serial Cable
Xilinx MultiLINX USB Cable
Agilent E5904B Cable
Close Cable
Get Cable Information
Auto Core Status Poll
ChipScope Pro Analyzer [new project]
OK
Cancel

Parallel Cable Selection

- Xilinx Parallel III
- Xilinx Parallel IV
- [✓ Auto Detect Cable Type]

Parallel Cable Parameters

Speed: 5 MHz
Port: LPT1

Figure 4-15: **Opening a Parallel Cable Connection**

Opening an Agilent E5904B Cable Connection

To connect to the Agilent E5904B cable, make sure it has been properly configured beforehand with a valid IP address, sub-net mask, gateway, etc. To open a connection, select **JTAG Chain → Agilent E5904B Cable**. This will bring up the Agilent E5904B Cable Options dialog box (Figure 4-16).

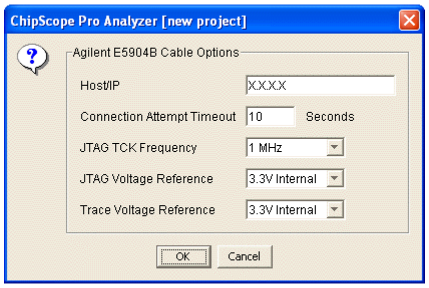
The Host/IP is the IP address of the cable. To properly configure the IP address of the Agilent cable, please consult the Agilent E5904B option, FPGA 500 trace port analyzer documentation.

The Connection Attempt Timeout text field indicates how long the ChipScope Analyzer should attempt to communicate to the cable unsuccessfully before timing out.

The JTAG TCK Frequency combo box indicates the speed of the JTAG interface from the Agilent cable to the JTAG chain. Choose a frequency that is appropriate for your system, given signal integrity, and other concerns.

The JTAG Voltage Reference combo box indicates which voltage supply to use for the communication to the JTAG chain: 3.3V internal, 2.5V internal, or an external supply.

The Trace Voltage Reference combo box indicates which voltage supply to use for the trace port data pins: 3.3V internal, 2.5V internal, or an external supply.


ChipScope Pro Analyzer [new project]
Agilent E5904B Cable Options
OK
Cancel

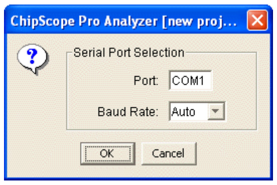
Host/IP: XXXX
Connection Attempt Timeout: 10 Seconds
JTAG TCK Frequency: 1 MHz
JTAG Voltage Reference: 3.3V Internal
Trace Voltage Reference: 3.3V Internal

Figure 4-16: **Agilent E5904B Cable Options Dialog Box**

Opening a MultiLINX connection

To connect to a MultiLINX cable, make sure the cable is properly connected to either the serial or USB port and select **Cable → Xilinx MultiLINX Serial Cable** or **Cable → Xilinx MultiLINX USB Cable**. If MultiLINX USB Cable is chosen, no other setup is necessary. The ChipScope Analyzer will automatically query the USB ports and connect to the cable. If MultiLINX Serial Cable is chosen, a configuration dialog box will open (Figure 4-17). Select the correct port and baud rate. The default values will work in most cases.

Note: The ChipScope Pro Analyzer supports only the JTAG configuration mode for the MultiLINX cable, regardless of port connection.



Port: COM1
Baud Rate: Auto

Figure 4-17: **MultiLINX Serial Port Options Dialog Box**

Polling the Auto Core Status

When ChipScope Pro cores are armed, the interface cable will query the cores on a regular basis to determine the status of the capture. If other programs are using the cable at the same time as the ChipScope Pro Analyzer, it may be beneficial to turn this polling off. This can be done in the **JTAG Chain** menu by un-checking **JTAG Chain** → **Auto Core Status Poll**. If this option is unchecked, when the Run or Trigger Immediate operation is performed, the Analyzer will not query the cores automatically to determine the status.

Note that this does not completely disable communication with the cable; it will only disable the periodic polling when cores are armed. If one or more cores trigger after the polling has been turned off, the capture buffer will not be downloaded from the device and displayed in any of the data viewer(s) until the **Auto Core Status Poll** option is turned on again.

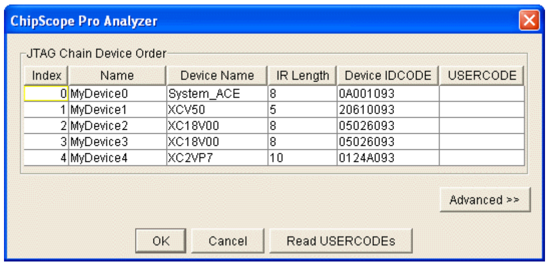
Configuring the Target Device(s)

You can use the ChipScope Pro Analyzer software with one or more valid target devices. The first step is to set up all of the devices in the Boundary Scan chain.

Setting Up the Boundary Scan (JTAG) Chain

After the Analyzer has successfully communicated with a download cable, it automatically queries the Boundary Scan (JTAG) chain to find its composition. All Xilinx Virtex/-E/-II/-II Pro/-4, Spartan-II/-IIE/-III,/XL, 9500/XL/XV, 4000XL/XLA, 18V00, Platform FLASH PROMs, CoolRunner™, CoolRunner-II, and System ACE™ devices are automatically detected. The entire IDCODE can be verified for valid target devices. To view the chain composition, select **JTAG Chain** → **JTAG Chain Setup**. A dialog box appears with all detected devices in order.

For devices that are not automatically detected, you must specify the IR (Instruction Register) length to insure proper communication to the ChipScope Pro cores. This information can be found in the device's BSDL file. The following example has one Virtex-II device (XC2V1000™), two serial PROMs (XC18V00™), and one Virtex-II Pro device (XC2VP7™) in a chain (Figure 4-18). USERCODEs can be read out of the ChipScope Pro target devices (only the XC2V1000 and XC2VP7 devices in this example) by selecting **Read USERCODEs**.



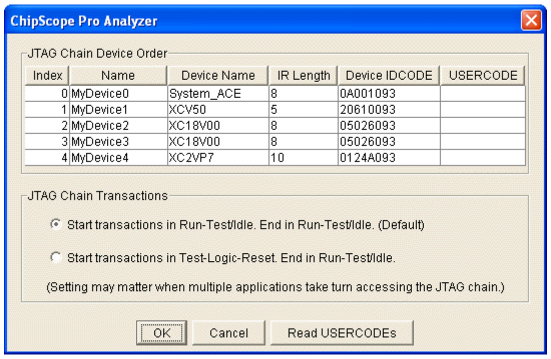
JTAG Chain Device Order

Index	Name	Device Name	IR Length	Device IDCODE	USERCODE
0	MyDevice0	System_ACE	8	0A001093	
1	MyDevice1	XCV50	5	20610093	
2	MyDevice2	XC18V00	8	05026093	
3	MyDevice3	XC18V00	8	05026093	
4	MyDevice4	XC2VP7	10	0124A093	

Figure 4-18: **Boundary Scan (JTAG) Setup Window**

The ChipScope Pro Analyzer tool automatically keeps track of the test access port (TAP) state of the devices in the JTAG chain, by default. If the ChipScope Pro Analyzer is used in conjunction with other JTAG controllers (such as the System ACE CompactFlash (CF) controller or processor debug tools), then the actual TAP state of the target devices can differ from the tracking copy of the ChipScope Pro Analyzer. In this case, the ChipScope

Pro Analyzer should always put the TAP controllers into a known state (for example, the Run-Test/Idle state) before starting any JTAG transaction sequences. Clicking on the **Advanced** button on the JTAG Chain Device Order dialog box reveals the parameters that control the start and end states of JTAG transactions (Figure 4-19). Use the second parameter if the JTAG chain is shared with other JTAG controllers.



Index	Name	Device Name	IR Length	Device IDCODE	USERCODE
0	MyDevice0	System_ACE	8	0A001093	
1	MyDevice1	XCV50	5	20610093	
2	MyDevice2	XC18V00	8	05026093	
3	MyDevice3	XC18V00	8	05026093	
4	MyDevice4	XC2VP7	10	0124A093	

JTAG Chain Transactions

[✓ Start transactions in Run-Test/Idle. End in Run-Test/Idle. (Default)]

Start transactions in Test-Logic-Reset. End in Run-Test/Idle.

(Setting may matter when multiple applications take turn accessing the JTAG chain.)

Figure 4-19: **Advanced JTAG Chain Parameters Setup Window**

Device Configuration

The ChipScope Pro Analyzer can configure target FPGA devices using the following download cables in JTAG mode only: Xilinx Parallel Cable III, Xilinx Parallel Cable IV, Xilinx MultiPRO, Xilinx MultiLINX, or Agilent E5904B TPA.

If the target device is to be programmed using a download cable by way of the JTAG port, select the **Device** menu, select the device you wish to configure, and select the **Configure** menu option. Only valid target devices can be configured and are, therefore, the only devices that have the **Configure** option available (Figure 4-20). Alternatively, you can right-click on the device in the project tree to get the same menu as **Device**.

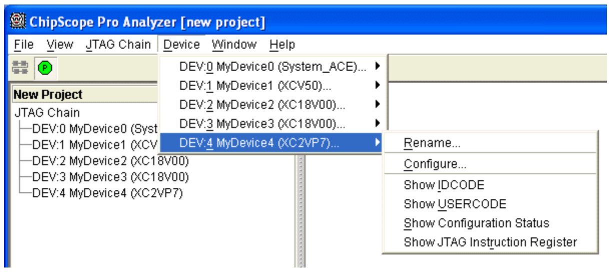

ChipScope Pro Analyzer [new project]
File
View
JTAG Chain
Device
Window
Help
New Project
JTAG Chain
DEV:0 MyDevice0 (Syst
DEV:1 MyDevice1 (XCV
DEV:2 MyDevice2 (XC18V00)
DEV:3 MyDevice3 (XC18V00)
DEV:4 MyDevice4 (XC2VP7)
DEV:0 MyDevice0 (System_ACE)...
DEV:1 MyDevice1 (XCV50)...
DEV:2 MyDevice2 (XC18V00)...
DEV:3 MyDevice3 (XC18V00)...
DEV:4 MyDevice4 (XC2VP7)...
Rename...
Configure...
Show IDCODE
Show USERCODE
Show Configuration Status
Show JTAG Instruction Register

Figure 4-20: **Device Menu Options**

After selecting the configuration mode, the JTAG Configuration dialog box opens (Figure 4-21). This dialog box reflects the configuration choice, and defaults to a blank entry for the configuration file.

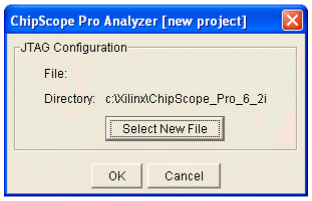

ChipScope Pro Analyzer [new project]
JTAG Configuration
File:
Directory: c:\Xilinx\ChipScope_Pro_6_2i
Select New File
OK
Cancel

Figure 4-21: **Selecting a Bitstream**

To select the BIT file to download, click on **Select New File**. The Open Configuration File dialog box (Figure 4-22) opens. Using the browser, select the device file you want to use to configure the target device. It is important to select a BIT file generated with the proper BitGen settings. Specifically, the **-g StartupClk:JtagClk** option must be used in BitGen in order for configuration to be successful.

Once you locate and select the proper device file, click **Open** to return to the JTAG Configuration dialog box (Figure 4-21).

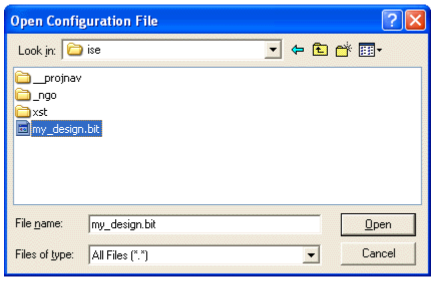


Figure 4-22: **Opening a Configuration File**

Once the BIT file has been chosen, click **OK** to configure the device.

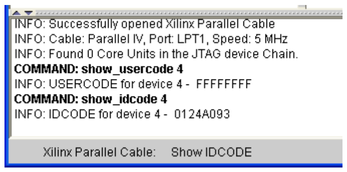
Observing Configuration Progress

While the device is being configured, the status of the configuration is displayed at the bottom of the Analyzer window. If the DONE status is not displayed, a dialog box opens, explaining the problem encountered during configuration. If the download is successful, the target device is automatically queried for ChipScope Pro cores, and the project tree is updated with the number of cores present. A folder is created for each core unit found and **Trigger Setup**, **Waveform**, and **Listing** leaf nodes appear under each ChipScope Pro unit. A **Bus Plot** leaf node will appear only if the core unit is determined to be an ILA core.

Displaying JTAG User and ID Codes

One method of verifying that the target device was configured correctly is to upload the device and user-defined ID codes from the target device. The user-defined ID code is the 8-digit hexadecimal code that can be set using the BitGen option **-g UserID**.

To upload and display the user-defined ID code for a particular device, select the **Show USERCODE** option from the **Device** menu for a particular device (Figure 4-20, page 4-17). Select the **Show IDCODE** option from the **Device** menu to display the fixed device ID code for a particular device. The results of these queries are displayed in the Message window (Figure 4-23). The IDCODE and USERCODE can also be displayed in the JTAG Chain Setup dialog box (**JTAG Chain** → **JTAG Chain Setup**, Figure 4-18, page 4-16).



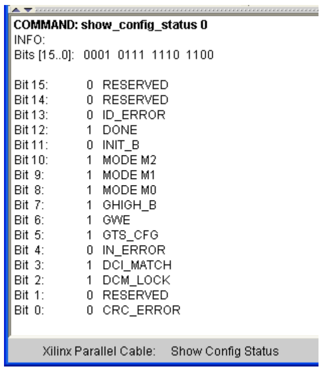
```
INFO: Successfully opened Xilinx Parallel Cable
INFO: Cable: Parallel IV, Port: LPT1, Speed: 5 MHz
INFO: Found 0 Core Units in the JTAG device Chain.
COMMAND: show_usercode 4
INFO: USERCODE for device 4 -  FFFFFFFF
COMMAND: show_idcode 4
INFO: IDCODE for device 4 -  0124A093
```

Figure 4-23: **Device USERCODE and IDCODE**

Displaying Configuration Status Information

The 32-bit configuration status register contains information such as status of the configuration pins and other internal signals. If configuration problems occur, select **Show Configuration Status** from the **Device** menu for a particular target device to display this information in the messages window (Figure 4-24).

Note: All target devices contain two internal registers that contain status information. These two registers are the Configuration Status Register (32 bits) and the JTAG Instruction Register (variable length, depending on the device). Only valid target devices have a Configuration Status Register. Although all devices have a JTAG Instruction Register that can be read, whether any status information is present depends on the implementation of that particular device. Refer to the data sheet for a particular device for more information.

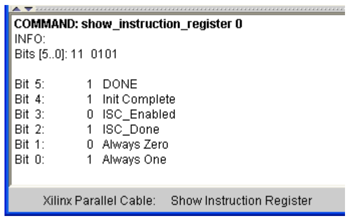


```
COMMAND: show_config_status 0
INFO:
Bits [15..0]:  0001  0111  1110  1100

Bit 15:        0  RESERVED
Bit 14:        0  RESERVED
Bit 13:        0  ID_ERROR
Bit 12:        1  DONE
Bit 11:        0  INIT_B
Bit 10:        1  MODE M2
Bit  9:        1  MODE M1
Bit  8:        1  MODE M0
Bit  7:        1  GHIGH_B
Bit  6:        1  GWE
Bit  5:        1  GTS_CFG
Bit  4:        0  IN_ERROR
Bit  3:        1  DCI_MATCH
Bit  2:        1  DCM_LOCK
Bit  1:        0  RESERVED
Bit  0:        0  CRC_ERROR
```

Figure 4-24: **Displaying Device Configuration Status**

For some devices, the JTAG instruction register contains status information as well. Use **Device** → **Show Instruction Register** to display this information in the messages window for any device in the JTAG chain (Figure 4-25).



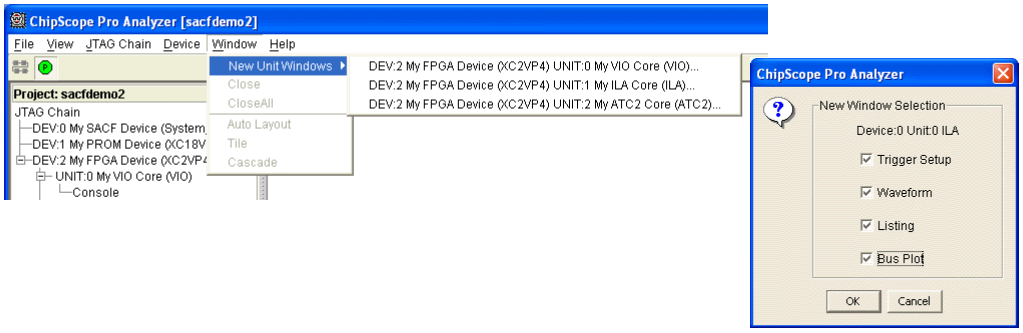
```
COMMAND: show_instruction_register 0
INFO:
Bits [5..0]: 11  0101

Bit 5:        1  DONE
Bit 4:        1  Init Complete
Bit 3:        0  ISC_Enabled
Bit 2:        1  ISC_Done
Bit 1:        0  Always Zero
Bit 0:        1  Always One
```

Figure 4-25: **Displaying Device Instruction Register Status**

Trigger Setup Window

To set up the trigger for a ChipScope Pro ILA or IBA core, select **Window** → **New Unit Windows** and the ChipScope Pro core desired(Figure 4-26). A dialog box will be displayed for that ChipScope Pro core, and you can select the **Trigger Setup**, **Waveform**, **Listing**, **Bus Plot** and/or **Console** window(s) in any combination (Figure 4-26). Windows cannot be closed from this dialog box.


ChipScope Pro Analyzer [sacfdemo2]
File View JTAG Chain Device Window Help
Project: sacfdemo2
JTAG Chain
DEV:0 My SACF Device (System
DEV:1 My PROM Device (XC18V
DEV:2 My FPGA Device (XC2VP4
UNIT:0 My VIO Core (VIO)
Console
New Unit Windows
Close
CloseAll
Auto Layout
Tile
Cascade
DEV:2 My FPGA Device (XC2VP4) UNIT:0 My VIO Core (VIO)...
DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA)...
DEV:2 My FPGA Device (XC2VP4) UNIT:2 My ATC2 Core (ATC2)...
ChipScope Pro Analyzer
New Window Selection
OK
Cancel

Device:0 Unit:0 ILA
- ✓ Trigger Setup
- ✓ Waveform
- ✓ Listing
- ✓ Bus Plot

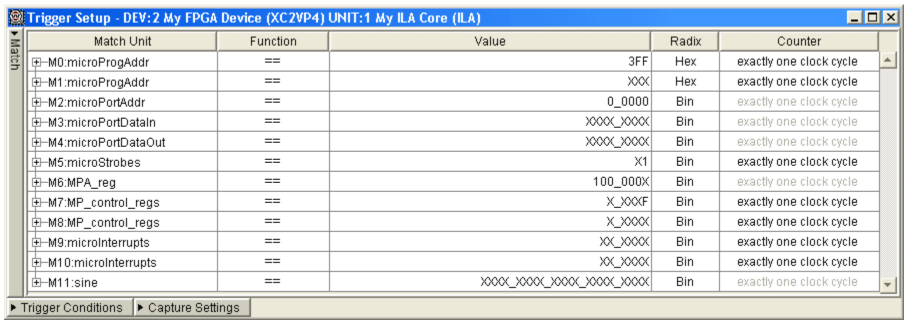
Figure 4-26: **Opening New Unit Windows**

The same operation can by achieved by double-clicking on the **Trigger Setup** leaf node in the project tree, or by right-clicking on the **Trigger Setup** leaf node and selecting **Open Trigger Setup**.

Each ChipScope Pro ILA, ILA/ATC, and IBA core has its own Trigger Setup window which provides a graphical interface for the user to set up triggers. The trigger mechanism inside each ChipScope Pro core can be modified at run-time without having to re-compile the design. The following sections describe how to modify the trigger mechanism's three components:

- *Match Functions:* Defines the match or comparison value for each match unit
- *Trigger Conditions:* Defines the overall trigger condition based on a binary equation or sequence of one or more match functions
- *Capture Settings:* Defines how many samples to capture, how many capture windows, and the position of the trigger in those windows

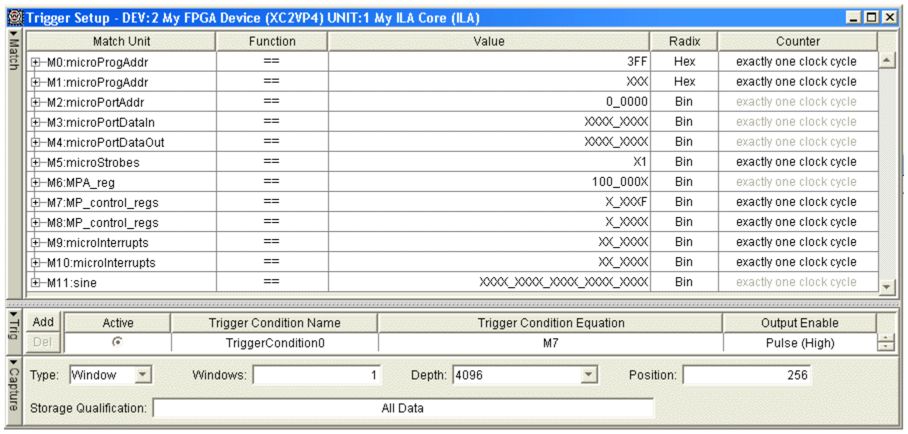
Each component is expandable and collapsible in the Trigger Setup window. To expand, click on the desired button at the bottom of the window (Figure 4-27).


Trigger Setup - DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA)
Match
Trigger Conditions
Capture Settings

Match Unit	Function	Value	Radix	Counter
M0:microProgAddr	==	3FF	Hex	exactly one clock cycle
M1:microProgAddr	==	XXX	Hex	exactly one clock cycle
M2:microPortAddr	==	0_0000	Bin	exactly one clock cycle
M3:microPortDataIn	==	XXXX_XXXX	Bin	exactly one clock cycle
M4:microPortDataOut	==	XXXX_XXXX	Bin	exactly one clock cycle
M5:microStrobes	==	X1	Bin	exactly one clock cycle
M6:MPA_reg	==	100_000X	Bin	exactly one clock cycle
M7:MP_control_regs	==	X_XXXF	Bin	exactly one clock cycle
M8:MP_control_regs	==	X_XXXX	Bin	exactly one clock cycle
M9:microInterrupts	==	XX_XXXX	Bin	exactly one clock cycle
M10:microInterrupts	==	XX_XXXX	Bin	exactly one clock cycle
M11:sine	==	XXXX_XXXX_XXXX_XXXX_XXXX	Bin	exactly one clock cycle

Figure 4-27: **Trigger Setup Window with Only Match Functions Expanded**

To collapse, click on the button to the left of the expanded section you wish to collapse (Figure 4-28).



Trigger Setup - DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA)

Match Unit	Function	Value	Radix	Counter
M0:microProgAddr	==	3FF	Hex	exactly one clock cycle
M1:microProgAddr	==	XXX	Hex	exactly one clock cycle
M2:microPortAddr	==	0_0000	Bin	exactly one clock cycle
M3:microPortDataIn	==	XXXX_XXXX	Bin	exactly one clock cycle
M4:microPortDataOut	==	XXXX_XXXX	Bin	exactly one clock cycle
M5:microStrobes	==	X1	Bin	exactly one clock cycle
M6:MPA_reg	==	100_000X	Bin	exactly one clock cycle
M7:MP_control_regs	==	X_XXXF	Bin	exactly one clock cycle
M8:MP_control_regs	==	X_XXXX	Bin	exactly one clock cycle
M9:microInterrupts	==	XX_XXXX	Bin	exactly one clock cycle
M10:microInterrupts	==	XX_XXXX	Bin	exactly one clock cycle
M11:sine	==	XXXX_XXXX_XXXX_XXXX_XXXX	Bin	exactly one clock cycle

Add	Active	Trigger Condition Name	Trigger Condition Equation	Output Enable
Del	●	TriggerCondition0	M7	Pulse (High)

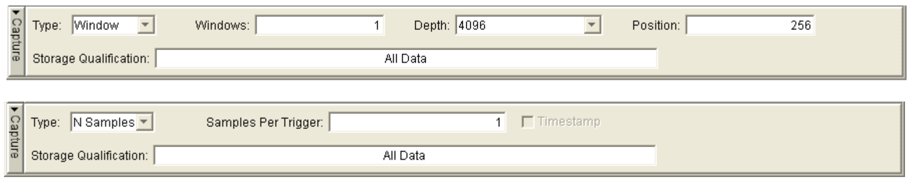
Type: Window Windows: 1 Depth: 4096 Position: 256

Storage Qualification: All Data

Figure 4-28: **Trigger Setup Window with All Sections Expanded**

Capture Settings

The capture settings section of the Trigger Setup window (Figure 4-29) defines the number of windows, and where the trigger event occur in each of those windows. A window is a contiguous sequence of samples containing one (and only one) trigger event. If an invalid number is entered for any parameter, the text field turns red, and an error is displayed in the Message pane.



Type: Window Windows: 1 Depth: 4096 Position: 256

Storage Qualification: All Data

Type: N Samples Samples Per Trigger: 1 Timestamp

Storage Qualification: All Data

Figure 4-29: **Capture Settings**

Type

The Type combo box in the capture settings defines the type of windows to use. If **Window** is selected, the number of samples in each window must be a power of two. However, the trigger can be in any position in the window. If **N Samples** is selected, the buffer will have as many windows as possible with the defined samples per trigger. The trigger will always be the first sample in the window if **N Samples** is selected.

Windows

The Windows text field is only available when **Window** is selected in the Type combo box. The number of windows is specified in this field and can be any positive integer from 1 to the depth of the capture buffer.

Depth

The Depth combo box is only available when **Window** is selected in the Type combo box. The Depth combo box defines the depth of each capture window. It is automatically populated with valid selections when values are typed into the Windows text field. Only powers of two are available.

Note: When the overall trigger condition consists of at least one match unit function that has a counter that is set to either **Occurring in at least *n* cycles** or **Lasting for at least *n* consecutive cycles**, the Window Depth or Samples Per Trigger setting cannot be less than eight samples. This is due to the pipelined nature of the trigger logic inside the ILA, ILA/ATC, IBA/OPB and IBA/PLB cores.

Position

The Position text field is only available when **Window** is selected in the Type combo box. The Position field defines the position of the trigger in each window. Valid values are integers from 0 to the depth of the capture buffer minus 1.

Samples Per Trigger

The Sample Per Trigger text field is only available when **N Samples** is selected in the Type combo box. Samples per trigger defines how many samples to capture once the trigger condition occurs. Valid values are any positive integer from 1 to the depth of the capture buffer. The trigger mark will always appear as sample 0 in the window. There will be as many sample windows as possible captured, given the overall sample depth.

Note: When occurring in at least *n* cycles or occurring for at least *n* consecutive cycles is selected for a match unit, and that match unit is a part of the overall trigger condition, the Window Depth or Samples Per Trigger cannot be less than 8. This is due to pipeline effects inside the ILA, ILA/ATC, or IBA core.

Timestamp

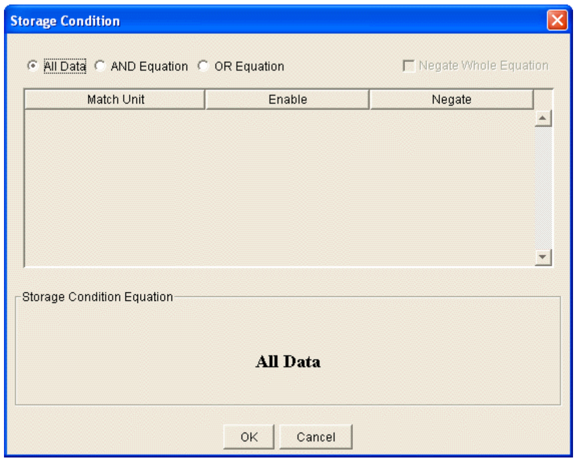
The **Timestamp** checkbox enables timestamps for capturing. This option is only enabled for ILA/ATC cores. The timestamp value will appear in the horizontal ruler of the Waveform and Listing windows.

Storage Qualification

The storage qualification condition is a Boolean combination of events that are detected by the match unit comparators that are subsequently attached to the trigger ports of the core. The storage qualification condition evaluates trigger port match unit events to decide whether or not to capture and store each individual data sample. The trigger and storage qualification conditions can be used together to define when to start (or finish) the capture process and what data is captured, respectively.

The Storage Condition dialog box has a table of all the match units. Each match unit occupies a row in the table. The Enable column indicates if that match unit is part of the trigger condition. The Negate column indicates if that match unit should be individually negated (Boolean NOT) in the trigger condition.

The storage qualification condition can be configured to capture all data (as shown in Figure 4-30, page 4-23), or it can be set up to capture data that satisfies a Boolean AND or OR combination of all the enabled match units (as shown in Figure 4-31, page 4-23). The overall Boolean equation can also be negated, selectable using the **Negate Whole Equation** checkbox above the table. The resulting equation appears in the Storage Condition Equation pane at the bottom of the window.



[✓ All Data] AND Equation OR Equation

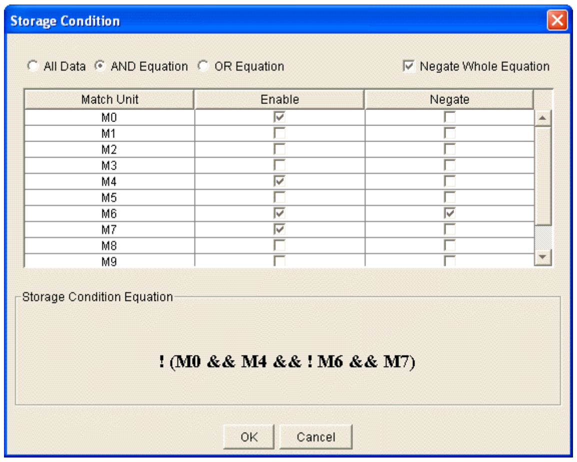
Negate Whole Equation

Match Unit	Enable	Negate

Storage Condition Equation

All Data

Figure 4-30: **Storage Qualification Condition Set to Capture All Data**



All Data [✓ AND Equation] OR Equation

[✓ Negate Whole Equation]

Match Unit	Enable	Negate
M0	✓	
M1		
M2		
M3		
M4	✓	
M5		
M6	✓	✓
M7	✓	
M8		
M9		

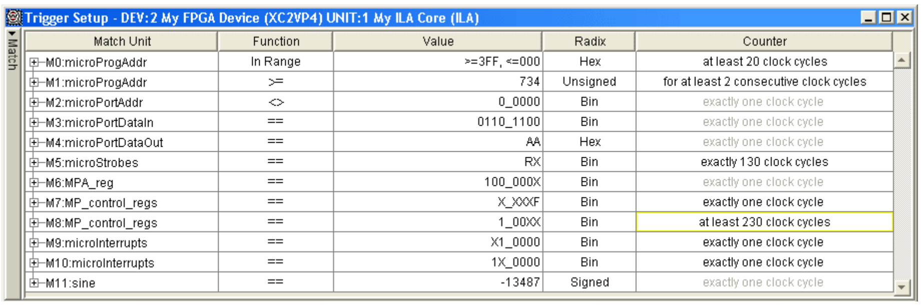
Storage Condition Equation

! (M0 && M4 && ! M6 && M7)

Figure 4-31: **Storage Qualification Condition Using Boolean Equation**

Match Functions

A match function is a definition of a trigger value for a single match unit. All the match functions are defined in the Match Functions section of the Trigger Setup window (Figure 4-32). One or more match functions will be defined in an equation or sequence in the Trigger Conditions section to specify the overall trigger condition of the ChipScope Pro core.


Trigger Setup - DEV:2 My FPGA Device (XC2VP4) UNIT:1 My ILA Core (ILA)
Match

Match Unit	Function	Value	Radix	Counter
M0:microProgAddr	In Range	>=3FF, <=000	Hex	at least 20 clock cycles
M1:microProgAddr	>=	734	Unsigned	for at least 2 consecutive clock cycles
M2:microPortAddr	<>	0_0000	Bin	exactly one clock cycle
M3:microPortDataIn	==	0110_1100	Bin	exactly one clock cycle
M4:microPortDataOut	==	AA	Hex	exactly one clock cycle
M5:microStrobes	==	RX	Bin	exactly 130 clock cycles
M6:MPA_reg	==	100_000X	Bin	exactly one clock cycle
M7:MP_control_regs	==	X_XXXF	Bin	exactly one clock cycle
M8:MP_control_regs	==	1_00XX	Bin	at least 230 clock cycles
M9:microInterrupts	==	X1_0000	Bin	exactly one clock cycle
M10:microInterrupts	==	1X_0000	Bin	exactly one clock cycle
M11:sine	==	-13487	Signed	exactly one clock cycle

Figure 4-32: **Setting the Match Functions**

Match Unit

The Match Unit field indicates which match unit the function applies to. Clicking on the **+** symbol next to the match unit number (or double clicking on the field) will expand that match unit so it is displayed as individual trigger port bits in at tree structure. Individual values for each bit can then be viewed and set.

Function

The Function combo box selects which type of comparison is done. Only those comparators that are allowed for that match unit are listed.

Value

The Value field selects exactly which trigger value to apply to that match unit. It is displayed according to the Radix field. Double-clicking on the field will make it editable. Place the cursor before the value you want to change, and typing a valid trigger character will overwrite that character. Or, select the field by single-clicking, then proceed by typing the trigger characters. Valid characters for the different radices are:

- *Hex:* X, 0-9, and A-F. X indicates that all four bits of that nibble are don't cares. A "?" indicates that the nibble consists of a mixture of 1's, 0's, X's, R's, F's, and B's (where appropriate)
- *Octal:* X, ?, 0-7
- Binary: X (don't care), 0, 1, R (rising), F (falling), and B (either transition). R, F, and B are only available if the match unit can detect transitions (Basic w/edges, Extended w/edges, Range w/edges)
- *Unsigned:* 0-9 (0 to 2^n-1 for an n-bit bus)
- *Signed:* 0-9 (-2^{n-1} to 2^{n-1} - 1 for an n-bit bus)

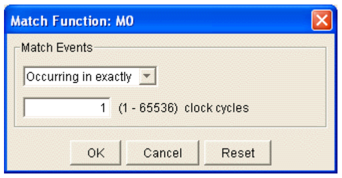
Also, when **Bin** is chosen as the radix, positioning the mouse pointer over a specific character will display a tool-tip, indicating the name and position of that bit.

Radix

The Radix combo box selects which radix to display in the Value field. Values are **Hex**, **Octal**, **Bin**, **Signed** (not allowed for **In Range** and **Out of Range** comparisons), and **Unsigned**.

Counter

The Counter field selects how many match function events must occur for the function to be satisfied. If the match counter is present for a particular match unit, the text in the Counter column will be in black text. If the counter is not present in the core, the text in that column will be grayed out. To change the value of the match counter, click on the counter cell, which will bring up the match unit counter dialog box (Figure 4-33).



Occurring in exactly
1 (1 - 65536) clock cycles

Figure 4-33: **Setting up the Match Counter**

The Counter field selects how many match function events must occur for the function to be satisfied.

- If occurring in exactly *n* clock cycles is selected, then *n* contiguous or *n* noncontiguous events will satisfy the match function counter condition.
- If occurring in at least *n* clock cycles is selected, then *n* contiguous or *n* noncontiguous events will satisfy the match function counter condition and will remain satisfied until the overall trigger condition is met.
- If occurring for at least *n* consecutive cycles is selected, then *n* contiguous events will satisfy the match function counter condition and will remain satisfied until the overall trigger condition is met or the match function value is no longer satisfied.

Note: When the overall trigger condition consists of at least one match unit function that has a counter set to either **Occurring in at least *n* cycles** or **Lasting for at least *n* consecutive cycles**, the Window Depth or Samples Per Trigger setting cannot be less than eight samples. This is due to the pipelined nature of the trigger logic inside the ILA, ILA/ATC, IBA/OPB or IBA/PLB cores.

Trigger Conditions

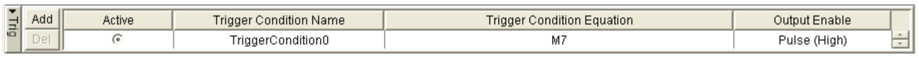
A trigger condition is a Boolean equation or sequence of one or more match functions. The ChipScope Pro core will capture data based on the trigger condition. More than one trigger condition can be defined. To add a new trigger condition, click the **Add** button. To delete a trigger condition, highlight any cell in the row and click **Del**. Although many trigger conditions can be defined for a single core, only one trigger condition can be chosen (active) at any one time.

Active

The Active field is a radio button that indicates which trigger condition is the currently active one.

Trigger Condition Name Field

The Trigger Condition Name field provides a mnemonic for a particular trigger condition. Trigger Condition *n* is used by default (Figure 4-34).



Add / Del	Active	Trigger Condition Name	Trigger Condition Equation	Output Enable
	●	TriggerCondition0	M7	Pulse (High)

Figure 4-34: **Viewing the Trigger Condition**

Trigger Condition Equation

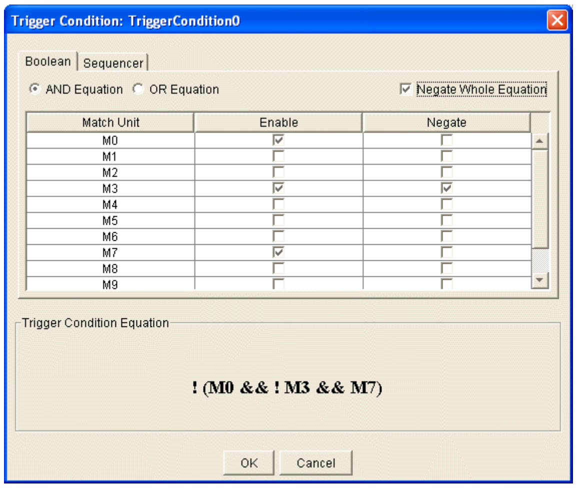
The Condition Equation field displays the current Boolean equation or state sequence of match functions that make up the overall trigger condition. By default, a logical AND of all the match functions present (one match function for each match unit) is the trigger condition. To change the trigger condition, click on the **Condition Equation** field, which brings up the Trigger Condition dialog box.

Trigger Condition Editor Dialog Box

If a trigger sequencer is present in the core, the Trigger Condition dialog will have two tabs: *Boolean* and *Sequencer*. When the Boolean tab is active, the trigger condition of a Boolean equation of the available match units. When the sequencer tab is active, the trigger condition is a state machine, where each state transition is triggered by a match function being satisfied.

The Boolean tab of the Trigger Condition dialog box has a table of all the match units. Each match unit occupies a row in the table. The Enable column indicates if that match unit is part of the trigger condition. The Negate column indicates if that match unit should be individually negated (Boolean NOT) in the trigger condition.

All the enabled match units can be combined in a Boolean AND or OR operation, selectable using the radio buttons below the match unit table. The overall equation can also be negated, selectable using the **Negate** checkbox below the table. The resulting equation appears in the Trigger Condition Equation pane at the bottom of the window (Figure 4-35).



Trigger Condition: TriggerCondition0

Boolean | Sequencer

[✓ AND Equation] OR Equation — Negate Whole Equation ✓

Match Unit	Enable	Negate
M0	✓	
M1		
M2		
M3	✓	✓
M4		
M5		
M6		
M7	✓	
M8		
M9		

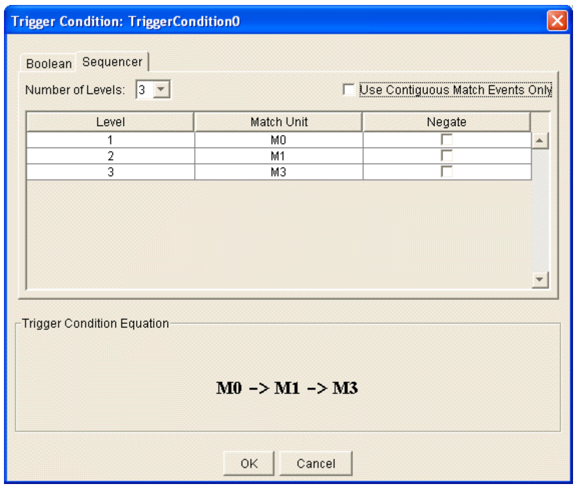
Trigger Condition Equation

! (M0 && ! M3 && M7)

OK Cancel

Figure 4-35: **Setting the Trigger Condition Boolean Equation**

The Sequencer tab of the Trigger Condition dialog box has a combo box from which you can select the number of levels in the trigger sequence and a table listing all the levels. The sequencer begins at Level 1 and proceeds to Level 2 when the match unit specified in Level 1 has been satisfied. The number of levels available is a parameter of the core, up to a maximum of 16 levels. Each level can look for a match unit being *satisfied* or *not satisfied*. To negate a level (for instance, to look for the absence of a particular match function) check the Negate cell for that level. A representation of the sequence appears in the Trigger Condition Equation pane at the bottom of the window (Figure 4-36).


Trigger Condition: TriggerCondition0
Boolean
Sequencer
Number of Levels: 3
Use Contiguous Match Events Only
Trigger Condition Equation
OK
Cancel

Level	Match Unit	Negate
1	M0	☐
2	M1	☐
3	M3	☐

M0 -> M1 -> M3

Figure 4-36: **Setting the Trigger Condition Sequencer**

The trigger sequence in Figure 4-36 can be satisfied by the eventual occurrence of match unit events M0 followed by M1 followed by M3 (with any occurrence or non-occurrence of events in between). Enable the **Use Contiguous Match Events Only** checkbox if you desire the trigger sequence to be satisfied only upon contiguous transitions from M0 to M1 to M3 (and not, for instance, the transitions of M0 followed by M1 followed by !M1 followed by M3).

Output Enable

If the trigger output is present in the core, a column named Output Enable becomes available. This cell is a combo box that allows the user to select which type of signal will be driven by the **trig_out** port of the ILA, ILA/ATC, or IBA core.

- *Disabled:* The output is a constant 0.
- *Pulse (High):* The output is a single clock cycle pulse of logic 1, 10 cycles after the actual trigger event.
- *Pulse (Low)* The output is a single clock cycle pulse of logic 0, 10 cycles after the actual trigger event.
- *Level (High)* The output transitions from a 0 to a 1 10 cycles after the actual trigger event.
- *Level (Low)* The output transitions from a 1 to 0 10 cycles after the actual trigger event.

Saving and Recalling Trigger Setups

All the information in the Trigger Setup window can be saved to a file for recall later with the current project or other projects. To save the current trigger settings, select **Trigger Setup** → **Save Trigger Setup**. A Save Trigger Setup As File dialog box will open, and the trigger settings can be saved in any location, with a `.ctj` extension. To load a trigger settings file into the current project, select **Trigger Setup** → **Read Trigger Setup**. A Read Trigger Setup file dialog box will open, and you can navigate to the folder where the trigger settings file (with a `.ctj` extension) exists. Once the trigger setting file is chosen, select **Open**, and those settings will be loaded into the Trigger Settings window.

Running/Arming the Trigger

After setting up the trigger, select **Trigger Setup** → **Run** to arm it. The trigger stays armed until the trigger condition is satisfied or you disarm the trigger. Once the trigger condition is satisfied, the core captures data according to the capture settings. When the sample buffer is full, the core stops capturing data. The data is then uploaded from the core and is displayed in the Waveform and/or Listing windows.

To force the trigger, select **Trigger Setup** → **Trigger Immediate**. This causes the ChipScope Pro unit to ignore the trigger condition and trigger immediately. After the sample buffer fills with data, the trigger disarms and the captured data appears in the Waveform and/or Listing window(s).

Stopping/Disarming the Trigger

To disarm the trigger, select **TriggerSetup** → **Stop Acquisition**. If the trigger condition has been satisfied at least once before the acquisition is stopped, the ChipScope Pro Analyzer program disarms the trigger and downloads/displays the captured data. Subsequent selections of **TriggerSetup** → **Run** cause the trigger to re-arm.

Waveform Window

To view the waveform for a particular ChipScope Pro ILA or IBA core, select **Window** → **New Unit Windows**, and the ChipScope Pro core desired. A dialog box will be displayed for that ChipScope Pro Unit, and the user can select the **Trigger Setup**, **Waveform**, **Listing**, and/or **Bus Plot** window, or any combination. Windows cannot be closed from this dialog box. The same operation can be achieved by double-clicking on the **Waveform** leaf node in the project tree, or right-clicking on the **Waveform** leaf node and selecting **Open Waveform**.

The Waveform window displays the sample buffer as a waveform display, similar to many modern simulators and logic analyzers. All signal browser operations can also be performed in the waveform window, such as bus creation, radix selection, renaming, etc. To perform a signal operation, right-click on a signal or bus in the Bus/Signal column.

Bus and Signal Reordering

Buses and signals can be reordered in the Waveform window. Select one or more signals and buses, and drag it to its new location. A ghost image of the signal or signals appears with the cursor, and a red line shows the potential drop location..

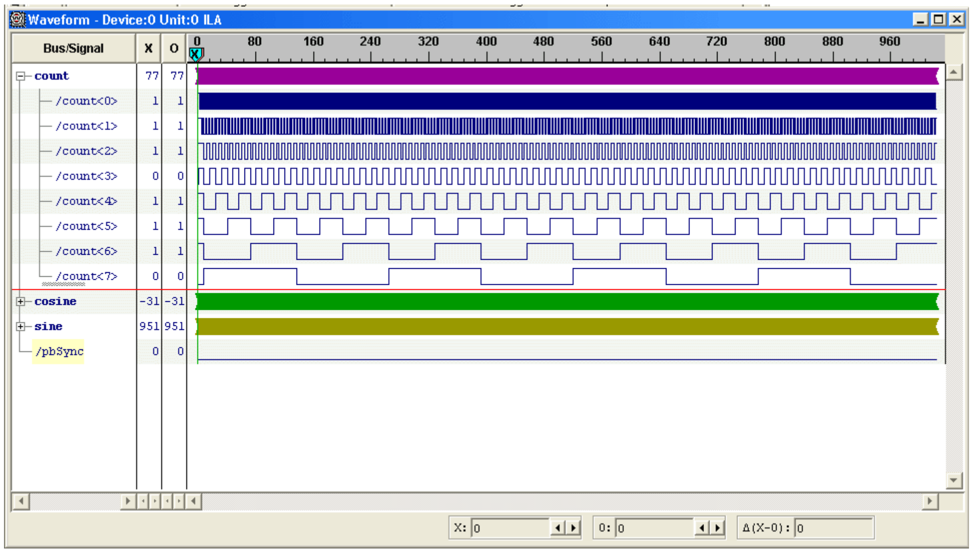


Figure 4-37: **Reordering Buses or Signals in the Waveform**

Cut/Copy/Paste/Delete Signals and Buses

Signals and buses can be cut, copied, pasted, or deleted using right-click menus. Select one or more signals and/or buses, right click on a selected signal or bus, and select the operation desired. Alternatively, the standard Windows key combinations are available (**Ctrl+X** for cut, **Ctrl+C** for copy, **Ctrl+V** for paste, **Del** for delete).

Zooming In and Out

Select **Waveform** → **Zoom** → **Zoom In** to zoom in to the center of the waveform display, or right-click in the waveform section and select **Zoom** → **Zoom In**. To zoom out from a waveform, use **Waveform** → **Zoom** → **Zoom Out**, or right-click in the waveform and select **Zoom** → **Zoom Out**.

To view the entire waveform display select **Waveform** → **Zoom** → **Zoom Fit**, or right click in the waveform and select **Zoom** → **Zoom Fit**.

To zoom into a specific area, just use the left mouse button to drag a rectangle in the waveform display. Once the drag is complete, a popup appears. Select **Zoom Area** to perform the zoom (Figure 4-38).

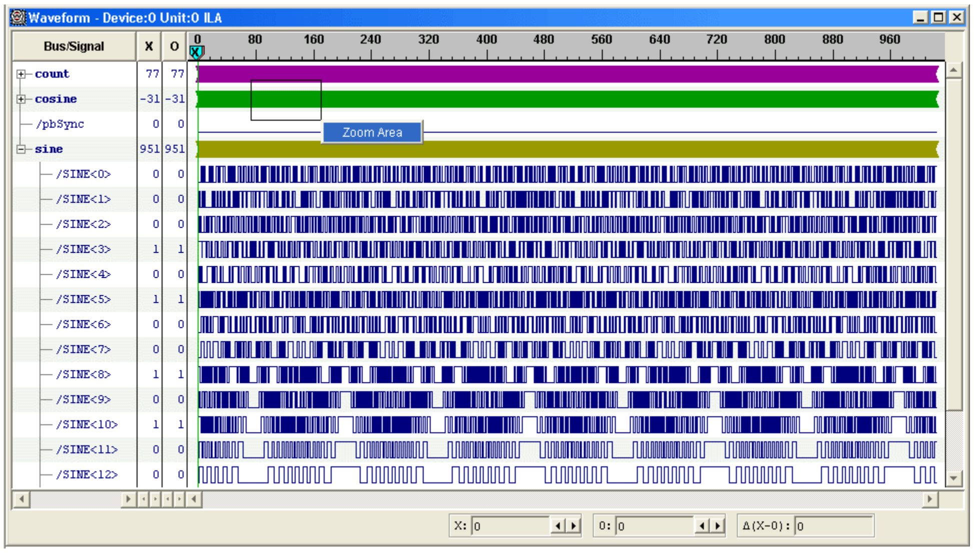


Figure 4-38: **Zoom Area Using the Automatic Popup Menu**

To zoom in to the space marked by the X and O cursors, select **Waveform** → **Zoom** → **Zoom X, O**, or right-click in the waveform and select **Zoom** → **Zoom X, O**. Other zoom features include zooming to the previous zoom factor by selecting **Zoom** → **Zoom Previous**, zooming to the next zoom factor by selecting **Zoom** → **Zoom Forward**, and zoom to a specific range of samples by selecting **Zoom** → **Zoom Sample** (Figure 4-39).

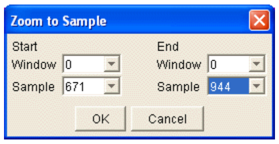


Figure 4-39: **Zoom to Sample Range**

Centering the Waveform

Center the waveform display around a specific point in the waveform by selecting **Waveform** → **Go To**, then centering the waveform display around the *X* and *O* markers, as well as the previous or next trigger position, or right-click in the waveform and select **Go To** (Figure 4-40).

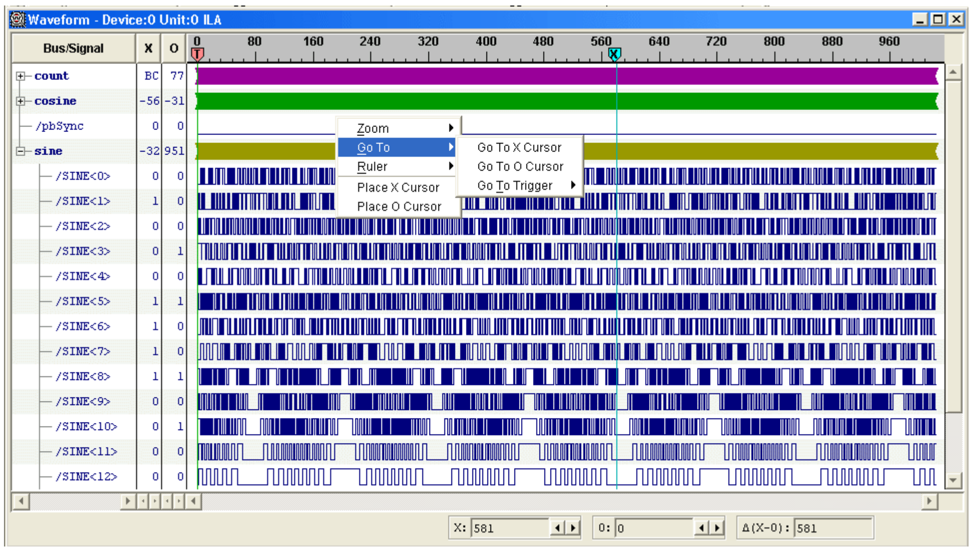


Figure 4-40: **Centering the Waveform on a Marker**

Cursors

Two cursors are available in the Waveform window: X and O. To place a cursor, right-click anywhere in the waveform section, and select **Place X Cursor** or **Place O Cursor.** A colored vertical line will appear indicating the cursor's position. Additionally, the status of all the signals and buses at that point will be displayed in the X or O column. The position of both cursors, and the difference in position of the cursors appears at the bottom of the Waveform window. Both cursors are initially placed at sample 0.

To move a cursor, either right-click in a new location in the waveform, or drag the cursor using the handles (X or O labels) in the waveform header, or drag the cursor-line itself in the waveform. Special drag icons will appear when the mouse pointer is over the cursor.

Sample Display Numbering

The horizontal axis of the waveform can be displayed as the sample number relative to the sample window (default) or by the overall sample number in the buffer. To display the sample number starting over at 0 for each window, select **Ruler** → **Sample # in Window** in the right-click menu. To display the sample number as an overall sample count in the buffer, select **Ruler** → **Sample # in Buffer** in the right-click menu. You can also select toggle the way that the samples that occur before the trigger marker are shown in the ruler (either negative or positive) by selecting **Ruler** → **Negative Time/Samples** in the right-click menu.

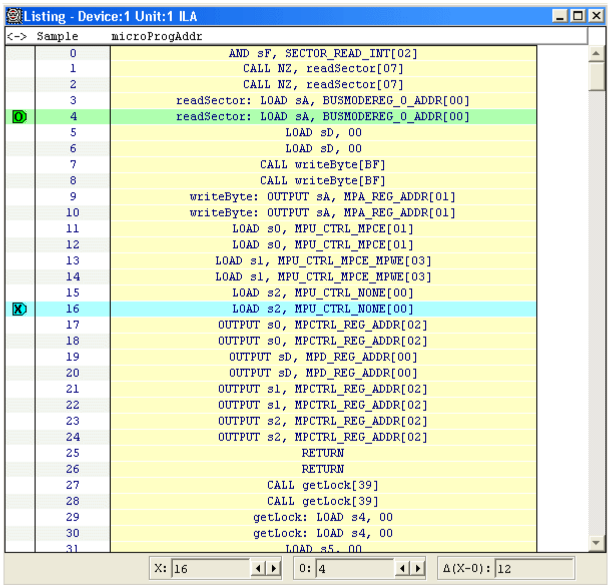
Displaying Markers

A static red vertical bar is displayed at each trigger position. A static black bar is displayed between two windows to indicate a period of time where no samples were captured. To not display either of these markers, un-check them on the right-click menu under **Markers** → **Window Markers** or **Markers** → **Trigger Markers**.

Listing Window

To view the Listing window for a particular ChipScope Pro ILA or IBA core, select **Window** → **New Unit Windows**, and the ChipScope Pro core desired. A dialog box will be displayed for that ChipScope Pro Unit, and the user can select the **Trigger Setup**, **Waveform**, **Listing**, and/or **Bus Plot** window, or any combination. Windows cannot be closed from this dialog box. The same operation can be achieved by double-clicking on the **Listing** leaf node in the project tree, or right-clicking on the **Listing** leaf node and selecting **Open Listing**.

The Listing window displays the sample buffer as a list of values in a table. Individual signals and buses are columns in the table (Figure 4-41). All signal browser operations can also be performed in the listing window, such as bus creation, radix selection, and renaming. To perform a signal operation, right-click on a signal or bus in the column heading.



<->	Sample	microProgAddr
	0	AND sF, SECTOR_READ_INT[02]
	1	CALL NZ, readSector[07]
	2	CALL NZ, readSector[07]
	3	readSector: LOAD sA, BUSMODEREG_0_ADDR[00]
O	4	readSector: LOAD sA, BUSMODEREG_0_ADDR[00]
	5	LOAD sD, 00
	6	LOAD sD, 00
	7	CALL writeByte[BF]
	8	CALL writeByte[BF]
	9	writeByte: OUTPUT sA, MPA_REG_ADDR[01]
	10	writeByte: OUTPUT sA, MPA_REG_ADDR[01]
	11	LOAD s0, MPU_CTRL_MPCE[01]
	12	LOAD s0, MPU_CTRL_MPCE[01]
	13	LOAD s1, MPU_CTRL_MPCE_MPWE[03]
	14	LOAD s1, MPU_CTRL_MPCE_MPWE[03]
	15	LOAD s2, MPU_CTRL_NONE[00]
X	16	LOAD s2, MPU_CTRL_NONE[00]
	17	OUTPUT s0, MPCTRL_REG_ADDR[02]
	18	OUTPUT s0, MPCTRL_REG_ADDR[02]
	19	OUTPUT sD, MPD_REG_ADDR[00]
	20	OUTPUT sD, MPD_REG_ADDR[00]
	21	OUTPUT s1, MPCTRL_REG_ADDR[02]
	22	OUTPUT s1, MPCTRL_REG_ADDR[02]
	23	OUTPUT s2, MPCTRL_REG_ADDR[02]
	24	OUTPUT s2, MPCTRL_REG_ADDR[02]
	25	RETURN
	26	RETURN
	27	CALL getLock[39]
	28	CALL getLock[39]
	29	getLock: LOAD s4, 00
	30	getLock: LOAD s4, 00
	31	LOAD s5, 00

X: 16 O: 4 Δ(X-O): 12

Figure 4-41: **The Listing View**

Bus and Signal Reordering

Buses and signals can be reordered in the Listing window. Simply click on a signal or bus heading in the table, and drag it to a new location.

Removing Signals/Buses

Individual signals and buses can be removed from the Listing window by right-clicking anywhere in the signal's column and selecting **Remove**. If **Remove All** is selected, all signals and buses will be removed.

Cursors

Cursors are available in the Listing window the same way as in the Waveform window. To place a cursor, right-click in the data section of the Listing window, and select either **Place X Cursor** or **Place O Cursor**. That line in the table will be colored the same as the cursor color. To move the cursor to a different position in the table, either right-click in the new location and do the same operation as before, or right-click on the cursor handle in the first column, and drag it to the new location.

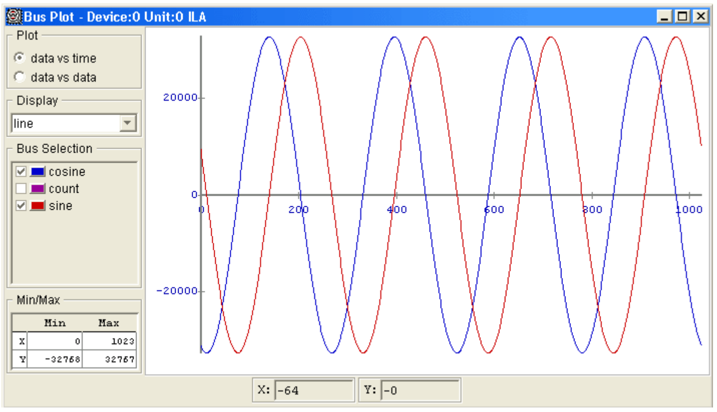
Goto Cursors

To automatically scroll the listing view to a cursor, right-click and select **Go To** → **Go To X Cursor** or **Go To** → **Go To O Cursor**.

Bus Plot Window

To view the Bus Plot[1] window for a particular set of ILA or IBA buses, select **Window** → **New Unit Windows** and the core desired. A dialog box will be displayed for that ChipScope Pro Unit, and the user can select the Trigger Setup, Waveform, Listing, and /or Bus Plot window, or any combination. Windows cannot be closed from this dialog box. The same operation can be achieved by double-clicking on the **Bus Plot** in the project tree, or right-clicking on **Bus Plot** and selecting **Open Bus Plot**.

Any buses for a particular core can be displayed in the Bus Plot window (Figure 4-42). The Bus Plot window displays buses as a graph of a bus's values over time, or one bus's values vs. another's.



Plot: [✓ data vs time] data vs data

Display: line

Bus Selection: ✓ cosine, count, ✓ sine

	Min	Max
X	0	1023
Y	-32768	32767

X: -64 Y: -0

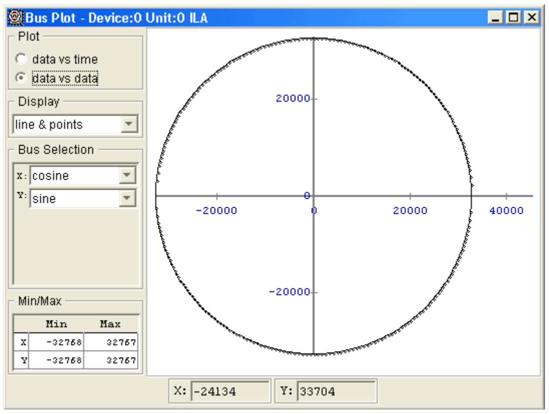
Figure 4-42: **The Bus Plot Window: Data vs. Time**

1. Bus Plot is not available for ILA/ATC cores.

Plot Type

Plot types are chosen in the upper left group of radio buttons. There are two plot types: *data vs. time* and *data vs. data*. When data vs. time is chosen (Figure 4-42), any number of buses may be displayed at one. When data vs. data is chosen (Figure 4-43), two buses need to be selected, and each point in the plot's *x* coordinate will be the value of one of the buses at a particular time, and the *y* coordinate will the value of the other bus at the same time.

Each bus will have its own color, and will be displayed according to its radix (hexadecimal, binary, octal, token and ASCII radices are displayed as unsigned decimal values with scale factor = 1.0, precision = 0).



Plot: data vs time [✓ data vs data]

Display: line & points

Bus Selection: X: cosine, Y: sine

	Min	Max
X	-32768	32767
Y	-32768	32767

X: -24134 Y: 33704

Figure 4-43: **The Bus Plot Window: Data vs. Data**

Display Type

The bus plot can be displayed using lines, points, or lines and points. The display type affects all bus values being displayed.

Bus Selection

The bus selection control allows you to select the individual buses to plot (in data vs. time mode) or the buses to plot against one another (in data vs. data mode). The color of each bus can be changed by clicking on the colored button next to the bus name (Figure 4-42, page 4-33).

Min/Max

The Min/Max display is used to show the maximum and minimum values of the axis in the current view of the bus plot.

Cursor Tracking

The X: and Y: displays at the bottom of the bus plot indicate the current X and Y coordinates of the mouse cursor when it is present in the bus plot view.

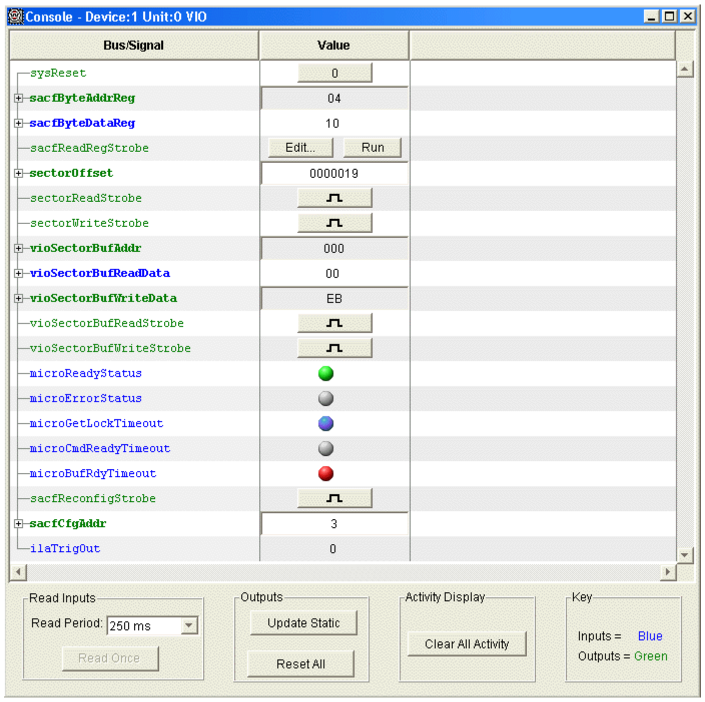
VIO Console Window

To open the Console window for a VIO core, select **Window** → **New Unit Windows**, and the core desired. A dialog box will be displayed for that ChipScope Pro Unit, and the user can select the **Console** window. (Windows cannot be closed from this dialog box.)

The Console window is for VIO cores only. The Console allows users to see the status and activity of the VIO core input signals and modify the status of the VIO core output signals. To open the console for a particular VIO core, double-click on the **Console** leaf node in the project tree.

All signal browser operations can also be performed in the Console window, such as bus creation, radix selection, renaming, etc. To perform a signal operation, right-click on a signal or bus in the column heading.

The Console window has a table with two columns: Bus/Signal and Value (Figure 4-44).


Console - Device:1 Unit:0 VIO

Bus/Signal	Value
sysReset	0
sacfByteAddrReg	04
sacfByteDataReg	10
sacfReadRegStrobe	Edit... Run
sectorOffset	0000019
sectorReadStrobe	⎍
sectorWriteStrobe	⎍
vioSectorBufAddr	000
vioSectorBufReadData	00
vioSectorBufWriteData	EB
vioSectorBufReadStrobe	⎍
vioSectorBufWriteStrobe	⎍
microReadyStatus	🟢
microErrorStatus	⚪
microGetLockTimeout	🔵
microCmdReadyTimeout	⚪
microBufRdyTimeout	🔴
sacfReconfigStrobe	⎍
sacfCfgAddr	3
ilaTrigOut	0

Read Inputs: Read Period: 250 ms

Key: Inputs = Blue, Outputs = Green

Figure 4-44: **The VIO Console Window**

Bus/Signal Column

The Bus/Signal column contains the name of the bus or signal in the VIO core. If it is a bus, it can be expanded or contracted to view or hide the constituent signals in the bus. In addition to all the operations available in the signal manager, two additional parameters can be set through the right-click menus- type and activity persistence.

VIO Bus/Signal Type

The signal's type determines how that signal is displayed in the Value column of the VIO Console. Different types are available depending on the type of VIO signal:

- VIO input signals have the following types (display types):
  - Text: ASCII characters
  - LEDs
    - Choose between Red, Blue, and Green LEDs
    - Either active high or active low
- VIO input buses have only one valid type - Text.
- VIO outputs have the following types (control types):
  - Text (ASCII text field)
  - Push Button (either active high or low)
  - Toggle Button
  - Pulse Train (synchronous outputs only)
  - Single Pulse (synchronous outputs only)
- VIO output buses have two valid types
  - Text
  - Pulse Train (synchronous output buses only)

VIO Bus/Signal Activity Persistence

The persistence of a signal indicates how long the activity is displayed in the Value Column (see Value Column, page 4-37 for a description of signal activity). If the persistence is *Infinite* the activity will be displayed in the column forever. If the persistence is *Long*, the activity will be displayed in the column for 80 times the sample period. If the persistence is *Short*, the activity will be displayed in the column for 8 times the sample period.

When the time limit on the persistence expires, a new activity will be displayed. If no activity occurred in the last sample cycle, no activity will be displayed in the Value column.

Bus and Signal Reordering

Buses and signals can be reordered in the Waveform window. Simply click on a signal or bus, and drag it to its new location. A red line then appears in the Bus/Signal column indicating the potential drop location

Cut/Copy/Paste/Delete Signals and Buses

Individual signals and buses can be cut, copied, pasted, or deleted using right-click menus. Right-click on a signal or bus, and select the operation desired. Alternatively, the standard Windows key combinations are available (**Ctrl+X** for cut, **Ctrl+C** for copy, **Ctrl+V** for paste, **Del** for delete).

Value Column

The Value column displays the current value of each of the signals in the console (see Figure 4-44, page 4-35). In the case of VIO core inputs, those cells are non-editable. Buses are displayed according to their selected radix. The VIO core inputs are updated periodically by default, according to a drop down combo box at the bottom of the console. Each of the VIO core inputs captures, along with the current value of the signal, activity information about the signal since the last time the input was queried. At high design speeds, it is possible for a signal to be sampled as a 0, then have the signal transition from a 0 to a 1, then back to a 0 again before the signal is sampled again.

In the case of synchronous inputs, the activity is also detected with respect to the design clock. This can be useful in detecting glitches. If a 0 to 1 transition is detected, an up arrow will appear alongside the value. If a 1 to 0 transition is detected, a down arrow appears. If both are detected, a two-headed arrow is displayed. The length of time the activity is displayed in the table is called the *persistence*. The persistence is also individually selectable via the right-click menu.

Note: The activity arrow will be displayed in black if the activity is synchronous and red if it is asynchronous.

You can choose the VIO signal/bus value type by right-clicking on the signal or bus and selecting the Type menu choice, as shown in Figure 4-45.

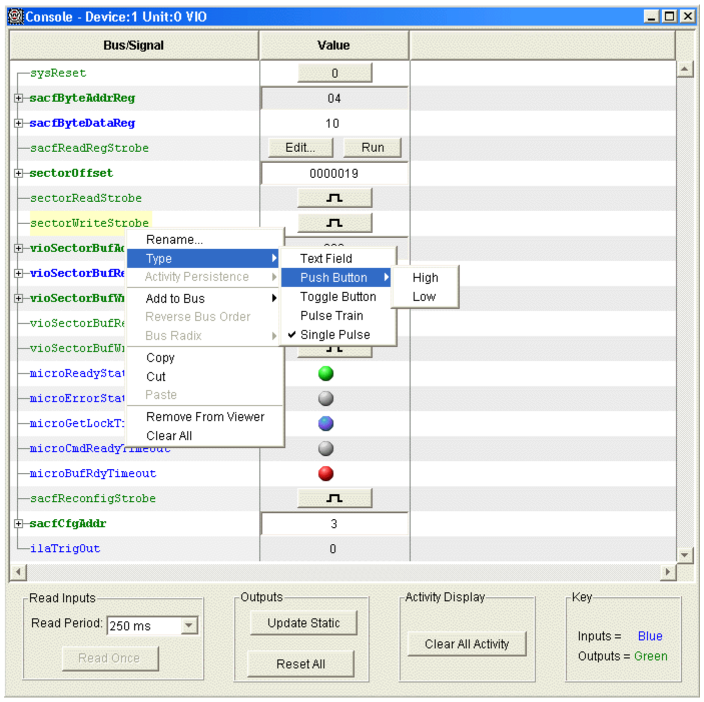


Figure 4-45: **The Type Selection Menu**

Text Field

When the **Text Field** type is selected, a text field is available for input using only the following valid characters:

- 0 and 1 for individual signals and binary buses
- 0-9, A-F for hex buses
- 0-7 for octal buses
- Valid signed and unsigned integers

Push Button

The **Push Button** type simulates an actual push button on a PCB. The inactive value is set when the button is not pressed in (0 for active high, 1 for active low). As long as the button is pressed in, the active value will be output from the VIO core.

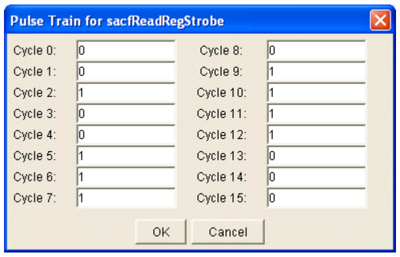
Toggle Button

The **Toggle Button** type switches between a 1 and a 0 with a single click.

Pulse Train (Synchronous outputs only)

The **Pulse Train** output type provides a control for synchronous outputs. A pulse train is a 16-cycle train of 1's and 0's, defined by the user. To edit the pulse train, click **Edit**. This brings up the Pulse Train dialog box (see Figure 4-46). One text field is available for each cycle in the pulse train. The text fields are populated by default according to the last value of the bus or signal. For buses, the fields are always displayed in binary to allow explicit control over each of the individual signals.

When **Run** is clicked, the pulse train is executed one time. This allows fine control over the output with respect to the design clock.


Pulse Train for sacfReadRegStrobe

Cycle 0:	0	Cycle 8:	0
Cycle 1:	0	Cycle 9:	1
Cycle 2:	1	Cycle 10:	1
Cycle 3:	0	Cycle 11:	1
Cycle 4:	0	Cycle 12:	1
Cycle 5:	1	Cycle 13:	0
Cycle 6:	1	Cycle 14:	0
Cycle 7:	1	Cycle 15:	0

Figure 4-46: **The Pulse Train Dialog**

Single Pulse (Synchronous outputs only)

The **Signal Pulse** control is a special kind of push button. When the button is pressed, instead of the core driving a constant active value for the duration of the button being pressed, a pulse train with a single high cycle will be executed exactly once.

Global Console Controls

Located at the bottom of the Console window are controls that affect all the inputs or outputs, as applicable (see Figure 4-47).

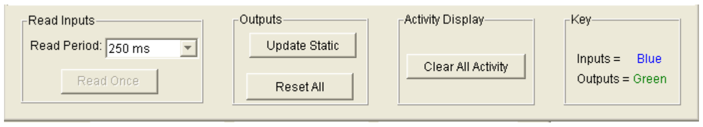


Figure 4-47: **Global Controls and Key**

Read Inputs

The **Read Period** at which the VIO core inputs are read is selectable via a combo box. The default sample period is 250 ms. You can also set the sample period to 500 ms, 1s, 2s, or Manual Scan.

When Manual Scan is chosen, the **Read Once** button becomes enabled. At that point, the VIO core inputs are only read when the **Read Once** button is pressed.

Update Static

By default, when one VIO core output is changed, information is immediately sent to the VIO core to set up that particular output. To update all non pulse train outputs at once, click **Update Static**.

Reset All

To reset all outputs to their default state (0 for text fields and toggle buttons, all 0 pulse train for pulse trains) click **Reset All**.

Activity Display

At some point, it may be desirable to reset the activity display for all VIO core inputs. To do so, press the **Clear All Activity** button. All input activity will be reset, regardless of the selected persistence.

Key

The key defines the colors in the console.

Help

Viewing the Help Pages

The ChipScope Pro Analyzer help pages contain information for only the currently opened versions of the ChipScope Pro software and each of the ChipScope Pro core units. Selecting **Help** → **About: ChipScope Software** displays the version of the ChipScope Pro Software. Selecting **Help** → **About: Cores** displays detailed core parameters for every detected core. Individual core parameters can be displayed by right-clicking on the unit in the project tree and selecting **Show Core Info**.

Also, you do not need to reinstall the ChipScope Pro tools to convert your evaluation version to a full version. You can also register an evaluation version of the ChipScope Pro Analyzer by selecting the **Help** → **Register ChipScope Pro** menu option and typing in the appropriate full-version registration ID. More information on how to obtain a full version of ChipScope Pro is available at http://www.xilinx.com/chipscope_pro.

ChipScope Pro Main Toolbar Features

In addition to the menu options, other ChipScope Pro Analyzer commands are available on a toolbar residing directly below the ChipScope Pro Analyzer menu (Figure 4-48). The second set of toolbar buttons is available only when the Trigger Setup window is open. The third and fourth sets of toolbar buttons are only available when the Waveform window is active.

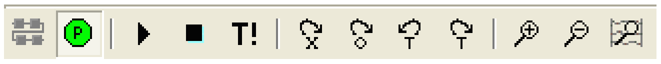


Figure 4-48: **Main ChipScope Pro Analyzer Toolbar Display**

The toolbar buttons (from left to right) correspond to the following equivalent menu options:

- **Open Cable/Search JTAG Chain:** Automatically detects the cable, and queries the JTAG chain to find its composition
- **Turn On/Off Auto Core Status Polling**: Green icon means polling is on, red icon means polling is off. Same as **Cable** → **Auto Core Status Poll**
- **Run**: Same as **TriggerSetup** → **Run** (**F5**)
- **Stop**: Same as **TriggerSetup** → **Stop** (**F9**)
- **Trigger Immediate**: Same as **TriggerSetup** → **Trigger Immediate** (**Ctrl+F5**)
- **Go To X Marker:** Same as **TriggerSetup** → **Go To** → **X Marker**
- **Go To O Marker:** Same as **TriggerSetup** → **Go To** → **O Marker**
- **Go To Previous Trigger:** Same as **Waveform** → **Go To** → **Previous Trigger**
- **Zoom In:** Same as **Waveform** → **Zoom In**
- **Zoom Out:** Same as **Waveform** → **Zoom Out**
- **Fit Window:** Same as **Waveform** → **Fit Window**

ChipScope Pro Analyzer Command Line Options

The ChipScope Pro Analyzer can be started either from the command line or from the **Start** menu. By default, the analyzer is installed to:
`C:\Xilinx\ChipScope_Pro_6_1i\analzyer.exe`

Optional Arguments

The following command line options are available, if run from the command line:

-geometry <*width*>x<*height*>+<*left edge x coordinate*>+<*top edge y coordinate*>

Set location, width and height of the Analyzer program window.

-project <*path and filename*>

Reads in specified project file at start. Default is not to read a project file at start up.

-init <*path and filename*>

Read specified `init` file at start up and write to the same file when the Analyzer exits. The default is: `%userprofile%\.chipscope\cs_analyzer.ini`

-log <*path and filename*> or -log stdout

Write log messages to the specified file. Specifying `stdout` will write to standard output. The default is: `%userprofile%\.chipscope\cs_analyzer.log`

Command Line Example

```
C:\Xilinx\ChipScope_Pro_6_1i\analyzer.exe -log c:\proj\t\t.log -init
C:\proj\t\t.ini -project c:\proj\t\t.cpj -geometry 1000x300+30+600
```

Tcl/JTAG Interface

Overview

This simple interface provides Tcl scripting access to Xilinx JTAG download cables via the ChipScope JTAG communication library. The purpose of Tcl/JTAG is to provide a simple scripting system to access basic JTAG functions. In a few lines of Tcl script, you should be able to scan and manipulate the JTAG chain through standard Xilinx cables.

Note: More information on JTAG can be found at http://www.xilinx.com/xapp/xapp139.pdf. More information on Tcl can be found at http://www.tcl.tk.

Requirements

- MS Windows 2000 Professional or Windows XP Professional
- Supported Xilinx JTAG cable such as Parallel Cable III/IV, MultiPRO, or MultiLINX
- A Xilinx Tcl shell (`xtclsh.exe` is provided in the Xilinx ISE 6.3i tool installation); other non-Xilinx Tcl shells are currently *not* supported and will *not* work with the Tcl/JTAG interface.
- Make sure the dynamically linked library files (`.dll`) provided in this Tcl/JTAG package are in your **`$PATH`** environment variable or in the current working directory.

Limitations

The Tcl/JTAG interface package favors simplicity over performance. Some commands such as **`jtag_shiftir`** and **`jtag_shiftdr`** transfer bits as strings (for example, "0001000") instead of as packed binary data structures. The extra overhead in converting particularly large data strings does result in some loss of performance; however, the simple design of the application programming interface (API) and the use of the Tcl scripting language makes Tcl/JTAG an easy-to-use means to interact with devices in the JTAG chain.

Note: Tcl/JTAG is only compatible with software that uses the JTAGComm interface to the JTAG cable communication device (such as the ChipScope Pro tool and the XMD tool that is part of the Xilinx Embedded Development Kit). Tools such as Xilinx iMPACT do not use the JTAGComm interface and therefore are *not* compatible for use with Tcl/JTAG scripts or programs.

Tcl/JTAG Command Summary

A brief summary of the Tcl/JTAG commands is shown in Table 5-1. See the Command Details, page 5-3 section for additional information about these commands.

Table 5-1: **Tcl/JTAG Command Summary**

Tcl/JTAG Command	Description
`jtag_autodetect`	Automatically detects devices in the JTAG chain.
`jtag_close`	Closes a JTAG cable connection.
`jtag_devicecount`	Gets or sets the number of devices in the JTAG chain.
`jtag_irlength`	Gets or sets length of the instruction register (IR).
`jtag_lock`	Lock a JTAG cable resource.
`jtag_navigate`	Get or set the JTAG test access port (TAP) state.
`jtag_open`	Opens a JTAG cable connection.
`jtag_scanchain`	Get a list of device IDCODEs by scanning the JTAG chain.
`jtag_shiftir`	Shift bits into the JTAG instruction register (IR).
`jtag_shiftdr`	Shift bits into the JTAG data register (DR).
`jtag_unlock`	Unlock a JTAG cable resource.
`jtag_version`	Get the current version of the Tcl/JTAG interface.

Command Details

jtag_autodetect

This command is used to autodetect the devices in the current JTAG chain. This function calls jtag_scanchain first to obtain IDCODEs for devices in the chain. The devices with known IDCODEs have IR lengths automatically assigned. Unrecognized devices must have their IR lengths assigned manually with jtag_irlength.

Syntax

```
jtag_autodetect handle
```

Required Arguments

`handle`

Handle to the cable connection that is returned by jtag_open.

Returns

A list of devices in the chain. List elements are in the format:

```
{IDCODE DEVICETYPE IRLENGTH}
```

An exception is thrown if the detect chain fails to return a valid JTAG chain.

Example

1. Autodetect the devices in the JTAG chain and assign the list of devices to the variable called *mychain*

```
%set mychain [jtag_autodetect $handle]
```

jtag_close

This command is used to close a JTAG cable connection. The handle is no longer valid after closing the connection.

Syntax

```
jtag_close handle
```

Required Arguments

`handle`

Handle to the cable connection that is returned by jtag_open.

Returns

`1`=success, `0`=failure

Example

1. Close the JTAG cable connection.

```
%jtag_close $handle
```

jtag_devicecount

This command is used to get or set the device count. If the count is specified, set the device count. If no count is specified, return the current number of devices in the JTAG chain. This function is called by jtag_autodetect to set the number of devices automatically.

Syntax

```
jtag_devicecount handle [count]
```

Required Arguments

`handle`

Handle to the cable connection that is returned by jtag_open.

Optional Arguments

`count`

New JTAG chain device count

Returns

1=success, 0=failure for set operation if **`count`** parameter is provided

Current device count for get operation if **`count`** parameter is not provided

Examples

1. To set the JTAG chain to three devices:

```
%jtag_devicecount $handle 3
```

2. To print the number of devices:

```
%puts "Number of devices in the chain: [jtag_devicecount $handle]"
```

jtag_irlength

This command is used to get or set the instruction register length for a device in the JTAG chain.

Syntax

```
jtag_irlength handle device [length]
```

Required Arguments

`handle`

Handle to the cable connection

`device`

Device number (0..chainlength-1)

Optional Arguments

`length`

Length to set

Returns

1=success, 0=failure if **`length`** is provided

Current length of the IR if **`length`** parameter is not provided

Examples

1. If the **`length`** parameter is provided, set the instruction register length of the given device.

```
%jtag_irlength $handle 0 5
```

2. If the **`length`** parameter is not provided, return the IR length of the given device

```
%puts "IR Length of device 0 is [jtag_irlength $handle 0]"
```

jtag_lock

This command is used to try to obtain an exclusive lock on the cable resource. If **`mswait`** is provided, block a maximum of **`mswait`** milliseconds while waiting for the resource. If **`mswait`** is not provided, this command will not block.

Note: A lock should always be obtained before performing any JTAG transactions. Failure to do so may result in unexpected behavior.

Syntax

```
jtag_lock handle [mswait]
```

Required Arguments

`handle`

Handle to the cable connection

Optional Arguments

`mswait`

Time to wait (in milliseconds)

Returns

1=success, 0=failure to get lock

Example

1. Try to obtain a JTAG connection lock and wait at least 1000 milliseconds: If the lock fails after 1000 ms, then a failure will be asserted by returning a value of 0.

```
%jtag_lock $handle 1000
```

jtag_navigate

This command is used to set or get the current JTAG TAP state. If state is provided, navigate the JTAG TAP state machine to the given stable state:

- Test Logic/Reset (TLR)
- Run Test/Idle (RTI)

The **`state`** parameter must be one of **`{TLR | RTI}`**. The **`cycles`** parameter represents the number of extra times to toggle the JTAG TAP clock signal (**`TCK`**) after arriving at the stable state.

Syntax

```
jtag_navigate handle [state [cycles]]
```

Required Arguments

`handle`

Handle to the cable connection

Optional Arguments

`state`

JTAG state **`{TLR | RTI}`**

`cycles`

Number of extra times to toggle TCK after entering the stable state (default is 0)

Returns

1=success, 0=failure when the state parameter is provided

Current state when the state parameter is not provided

Examples

1. To navigate to run test idle and clock 15 times,

```
%jtag_navigate $handle RTI 15
```

2. If no state is provided, return the current tap state.

```
%puts "Current state is [jtag_navigate $handle]"
```

jtag_open

This command is used to open a cable connection and returns a handle to the opened cable. This handle is used by all other Tcl/JTAG commands to access the JTAG cable functions. The **`jtag_open`** command may be called with optional parameters to set up specific JTAG cable options.

Syntax

```
jtag_open [type] [port] [frequency]
```

Optional Arguments

Set of parameters for the JTAG cable. These are passed directly to the cable driver.

Xilinx Parallel Cable III Arguments

`type`

Cable type for Xilinx Parallel Cable III is **`xilinx_parallel3`**

`port`

Parallel port identifier

Xilinx Parallel Cable IV (and MultiPRO) Arguments

`type`

Cable type for Xilinx Parallel Cable IV (and MultiPRO) is **`xilinx_parallel4`**

`port`

Parallel port identifier (such as **`lpt1`**, **`lpt2`**, and so on)

`frequency`

TCK clock frequency in Hz **`{200000 | 2500000 | 5000000} (default is 5000000)`**

Xilinx MultiLINX Arguments

`type`

Cable type for Xilinx MultiLINX is **`multilinx`**

`port`

USB or RS-232 serial port identifier **`{USB | COM1 | COM2 | COM3 | COM4}`**

`baud`

RS-232 serial baud rate **`{AUTO | 9600 | 19200 | 38400 | 57600}`**

Note: The **`baud`** argument should not be used when the **`port`** argument is set to **`USB`**

Agilent E5904B Option 500 FPGA Trace Port Analyzer Arguments

`type`

Cable type for the Agilent E5904B Option 500 FPGA Trace Port Analyzer (Agilent E5904B TPA) is **`agilent_gateway`**

`host`

Host name or IP address of the Agilent E5904B TPA

`port`

Port number (default is 6470)

`frequency`

TCK clock frequency in kHz **`{500 | 1000 | 2000 | 4000 | 5000 | 10000 | 20000 | 30000} (default is 10000)`**

`timeout`

Connection timeout in seconds (default is 10 seconds)

`mode`

Connection mode **`{Exclusive | Session}`** (default is **`Exclusive`**)

`vref`

JTAG pin reference voltage type **`{Internal | External}`** (default is **`Internal`**)

`tvref`

Trace pin reference voltage type **`{Internal | External}`** (default is **`Internal`**)

`voltage`

Internal JTAG pin reference voltage in mV **`{2500 | 3300}`** (default is **`3300`**)

Note: Note: this argument will be ignored if **`vref`** is set to **`External`**.

`tvoltage`

Internal trace pin output voltage in mV **`{2500 | 3300}`** (default is **`3300`**)

Note: This argument will be ignored if **`tvref`** is set to **`External`**.

Returns

Handle to the opened cable if successful; nothing if open function failed.

Examples

1. Autodetect the parallel cable type

```
%set handle [jtag_open]
```

2. Manually force a Parallel Cable III open:

```
%set handle [jtag_open port=lpt1 type=xilinx_parallel3]
```

3. Manually force a Parallel Cable IV or MultiPRO open:

```
%set handle [jtag_open port=lpt1 type=xilinx_parallel4
frequency=5000000]
```

4. Manually force a MultiLINX open:

```
%set handle [jtag_open port=COM1 type=multilinx baud=57600]
```

5. Manually force an Agilent E5904B TPA open:

```
%set handle [jtag_open type=agilent_gateway host=192.168.1.3
port=6470 frequency=2000 timeout=10 mode=Exclusive vref=Internal
voltage=3300 tvref=Internal tvoltage=2500]
```

jtag_shiftir

This command is used to shift bits into the specified device's instruction register.

Syntax

```
jtag_shiftir handle
    [[ [-constant {0 | 1}] [-bitcount count] ] | [-buffer buffer] ]
    [-endstate state] [-device devicenumber] [-discard]
```

Required Arguments

All arguments are passed directly to the cable driver.

handle

Handle to the cable connection

-constant {0 | 1}

Shift in a constant 0 or constant 1. Only required if **-buffer** is not specified.

-bitcount count

Number of bits to shift. Only required if **-buffer** is not specified. If **-buffer** is specified, the bitcount is automatically computed.

-buffer buffer

buffer is a binary string of bits. Buffer to shift in to TDI. Format is a string consisting of 0's and 1's like "1100101". Only required if **-constant** is not specified.

Note: Bits are shifted LSB > MSB in the order of the Tcl string index. This means that "100" will first shift 1 into TDI followed by 0 followed by 0. If you don't like this order, make a simple function to reverse your strings before passing to the buffer.

Optional Arguments

All arguments are passed directly to the cable driver.

-endstate state

State to navigate into after the shift operation. One of {**TLR** | **RTI** | **SIR**}. Default is **SIR**.

-device devicenumber

Shift bits into and out of **devicenumber** by padding extra bits. IR lengths must be set up correctly to use this argument.

If no **-device** is specified, no padding is done and shifts apply to the entire jtag chain.

-discard

Do not return output buffer from the shift. This is an optimization to allow ignoring the TDO pin when writing data to the chain.

Returns

Bits shifted out of the JTAG chain. If **-discard** option is specified, nothing is returned.

Examples

1. Shift the BYPASS instruction into device 2

```
%jtag_shiftir $handle -constant 1 -bitcount 6 -device 2 -discard
```

2. Shift USERCODE instruction into chain, end in TLR, and capture last instruction

```
%set oldir [jtag_shiftir $handle -buffer "000100" -endstate TLR]
```

jtag_shiftdr

This command is used to shift bits into the specified device's data register.

Syntax

```
jtag_shiftdr handle
   [[ [-constant {0 | 1}] [-bitcount count] ] | [-buffer buffer] ]
   [-endstate state] [-device devicenumber] [-discard]
```

Required Arguments

All arguments are passed directly to the cable driver.

`handle`

Handle to the cable connection

`-constant {0 | 1}`

Shift in a constant 0 or constant 1. Only required if **`-buffer`** is not specified.

`-bitcount count`

Number of bits to shift. Only required if **`-buffer`** is not specified. If **`-buffer`** is specified, the bitcount is automatically computed.

`-buffer buffer`

`buffer` is a binary string of bits. Buffer to shift in to TDI. Format is a string consisting of 0's and 1's like "1100101". Only required if **`-constant`** is not specified.

Note: Bits are shifted LSB > MSB in the order of the Tcl string index. This means that "100" will first shift 1 into TDI followed by 0 followed by 0. If you don't like this order, make a simple function to reverse your strings before passing to the buffer.

Optional Arguments

All arguments are passed directly to the cable driver.

`-endstate state`

State to navigate into after the shift operation. One of **`{TLR | RTI | SDR}`**. Default is **`SDR`**.

`-device devicenumber`

Shift bits into and out of **`devicenumber`** by padding extra bits. IR lengths must be set up correctly to use this argument.

If no **`-device`** is specified, no padding is done and shifts apply to the entire jtag chain.

`-discard`

Do not return output buffer from the shift. This is an optimization to allow ignoring the TDO pin when writing data to the chain.

Returns

Bits shifted out of the JTAG chain. If **`-discard`** option is specified, nothing is returned.

Examples

1. Shift 32 bits out of device 2

```
%set mydata [jtag_shiftdr $handle -constant 0 -bitcount 32 -device
2]
```

2. Shift bits into chain and end in TLR

```
%jtag_shiftdr $handle -buffer "11110000" -endstate TLR -discard
```

jtag_scanchain

This command is used to scan the JTAG chain for the current device count and an IDCODE for each device that supports it. Typical users do not need to call this function. Instead, call the jtag_autodetect function.

Side effects:

This navigates to test-logic-reset to obtain IDCODEs from devices that support the **`idcode`** command. The device count is set to the number of devices found in the chain.

Required Arguments

All arguments are passed directly to the cable driver.

`handle`

Handle to the cable connection

Returns

List of device IDCODEs (0 when not supported) for each device in the chain

Examples

1. Capture list of IDCODEs into variable **`idcode_list`**.

```
%set idcode_list [jtag_scanchain $handle]
```

jtag_unlock

This command is used to release an exclusive lock on a cable resource.

Required Arguments

`handle`

Handle to the cable connection

Returns

1=success, 0=failure

Example

1. Attempt to release lock on **`handle`**.

```
%jtag_unlock $handle
```

jtag_version

This command is used to get the current Tcl/JTAG API version.

Returns

Current version of the package

Example

1. Get current Tcl/JTAG version:

```
%set current_version [jtag_version]
```

Tcl/JTAG Example

The following example shows how you can have complete access to a JTAG chain over a Xilinx parallel cable by using just a few Tcl commands:

1. From the DOS prompt, navigate to the ChipScope Pro install directory and invoke your **`tcl`** shell. The executable is `tclsh.exe` for the standard one, or `xtclsh.exe` if you have the Xilinx tools installed. You receive a prompt like this: %
2. Load the **`tcljtag`** library and support functions. Assuming you are in the ChipScope installation directory, type the following at the **`tcl`** shell prompt:

   ```
   %source tcljtag.tcl
   ```

   You should see a message similar to "Attaching XYZ" where XYZ is an address.
3. Open a cable. Assuming you have a Xilinx Parallel Cable III, Parallel Cable IV, or MultiPRO cable attached on a standard **`lpt1`** port, type:

   ```
   %set handle [jtag_open]
   ```

   If the cable open operation was successful, **`handle`** will see something like `cable0x10f0068` (the numbers will vary).
4. Get an exclusive lock on the cable resource:

   ```
   %jtag_lock $handle
   ```
5. Autodetect the devices in the jtag chain:

   ```
   %jtag_autodetect $handle
   ```

   If the autodetect was successful, you see a list of devices found, for example:

   ```
   {80a30093 xcv600e 5} {21028093 xc2v1000 6} {05026093 xc18v00 8}
   ```
6. Navigate to **Test** → **Logic** → **Reset**. According to the JTAG specification, this will load the IDCODE or BYPASS instruction into all the instruction registers:

   ```
   %jtag_navigate $handle TLR
   ```
7. Shift some 1 bits into the data registers. Returned values represent IDCODEs of the devices in the JTAG chain:

   ```
   %jtag_shiftdr $handle -constant 1 -bitcount 255 -endstate TLR
   ```
8. Release the cable lock:

   ```
   %jtag_unlock $handle
   ```
9. Close the cable:

   ```
   %jtag_close $handle
   ```
10. Exit the Tcl shell:

    ```
    %exit
    ```