# ChipScope Pro Software and Cores

## *User Guide*

**UG029 (v14.1) April, 24 2012**

XILINX®

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 03/01/11 | 13.1 | • Added 7 Series support for logic debug<br>• Removed IBA/PLB (not IBA/PLB46)<br>• Removed IBA/OPB<br>• Removed IBERT V4 GT11<br>• Added Startup trigger mode<br>• Added ChipScope Pro Analyzer IBERT sweep test plot<br>• Added standalone IBERT plot viewer<br>• Added 1/2, 1/4, 1/8 line rate support for GTH<br>• Added ICON, ILA, VIO, and ATC2<br>• Updated the CSE/Tcl section with new commands and changes |
| 07/06/11 | 13.2 | • Added information on support for cores and transceivers in Kintex™-7 FPGA devices.<br>• Added Digilent JTAG-SMT1 and JTAG-HS1 USB-to-JTAG download cables to section on Communications Requirements (Chapter 1).<br>• Minor corrections and content enhancements throughout. |
| 10/19/2011 | 13.3 | • Chapter 2: removed some sections on generating cores and included links to where the content now resides in datasheets. References throughout this document were updated accordingly.<br>• Chapter 3: In the section "Choosing ILA Core Capture Parameters," some significant clarifications were added and content re-organized.<br>• Other, minor, corrections and clarifications throughout. |
| 1/18/2012 | 13.4 | • Added support for Virtex-7 throughout the guide.<br>• Added new section, "RX Margin Analysis Panel" in Chapter 4". |
| 4/24/2012 | 14.1 | Updated for 14.1 release. Updates for KC705 board. |

Chapter 1

# 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 and other test and measurement hardware components with the target design inside the supported Xilinx® FPGA devices listed in the ISE® Design Suite Product Table [See Reference 16, p. 211]. The tools communicate with these components and provide the designer with a robust logic analyzer solution.

The ChipScope Pro Serial I/O Toolkit provides features and capabilities specific to the exploration and debug of designs that use the high-speed serial transceiver I/O capability of FPGAs. The IBERT (internal bit error ratio tester) core and related software provides access to the high-speed serial transceivers and perform bit error ratio analysis on channels composed of these transceivers. In this document, the transceivers are called MGTs (multi-gigabit transceivers). The IBERT core supports the high-speed serial transceivers found in the Xilinx Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6, and Virtex-5 FPGA devices listed in the ISE Design Suite Product Table [See Reference 16, p. 211].

## ChipScope Pro Tools Description

The following table gives a brief description of the various ChipScope Pro tools and cores.

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

| Tool | Description |
|---|---|
| Xilinx CORE Generator™ tool | Provides core generation capability for the ICON (integrated controller), ILA (integrated logic analyzer), VIO, (virtual input/output), and ATC2 (Agilent trace core) cores targeting all supported FPGA device families. Also provides core generation capability for the IBERT v2.0 core targeting the Virtex-7, Kintex-7, Virtex-6, Spartan-6, and Virtex-5 FPGA families. The Xilinx CORE Generator tool is part of the Xilinx ISE Design Suite tool installation. |
| IBERT Core Generator | Provides full design generation capability for the IBERT v1.0 core targeting the Virtex-5 devices. You select the MGTs and parameters governing the design, and the CORE Generator tool uses the ISE design suite to produce a configuration file. |
| Core Inserter | Automatically inserts the ICON, ILA, and ATC2 cores into your synthesized design. |

*Table 1-1:* **ChipScope Pro Tools Description *(Cont'd)***

| Tool | Description |
|---|---|
| PlanAhead™ Design Analysis Tool | Automatically inserts the ICON and ILA cores into the design netlist. For more information on this feature, go to PlanAhead Design Analysis Tool [See Reference 17, p. 211]. |
| ChipScope Pro Analyzer Tool | Provides in-system device configuration, as well as trigger setup, trace display, control, and status for the ICON, ILA,VIO, and IBERT cores. |
| ChipScope Engine Tcl (CSE/Tcl) Scripting Interface | The scriptable CSE/Tcl command interface makes it possible to interact with devices in a JTAG (Joint Test Action Group, IEEE standard) chain from a Tcl shell(1). |

**Notes:**

1. *Tcl* stands for *Tool Command Language*. The CSE/Tcl interface requires the Tcl shell program (called `xtclsh`) that is included in the ChipScope Pro and ISE tool installations or in the ActiveTcl 8.4 shell available from ActiveState [See Reference 24, p. 212].

This figure shows a block diagram of a system containing debug cores added using the ChipScope Pro tools. You can place the ICON, ILA, VIO, and ATC2 cores (collectively called the ChipScope Pro cores) into your design by generating the cores with the CORE Generator tool and instantiating them into the HDL source code. You can also insert the ICON, ILA, and ATC2 cores directly into the synthesized design netlist using the ChipScope Pro Core Inserter or PlanAhead tools. You then place and route your design using the ISE implementation tools. Next, download the bitstream into the device under test and analyze the design with the ChipScope Pro Analyzer tool.

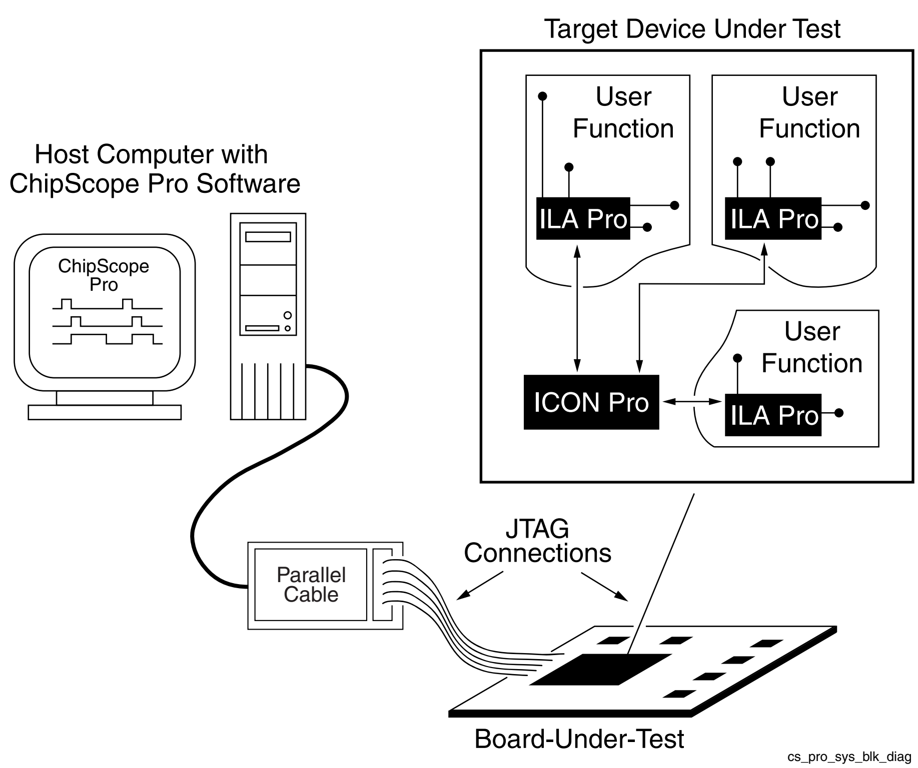


*Figure 1-1:* **ChipScope Pro System Block Diagram**

ChipScope Pro Analyzer tool supports the following download cables for communication between your computer and the devices in the JTAG boundary scan chain:

- Platform Cable USB
- Parallel Cable IV
- Digilent USB-to-JTAG cables
- ByteTools Catapult EJ-1 Ethernet-to-JTAG cable

The ChipScope Pro Analyzer tool and cores contain many features that you can use to verify your logic (Table 1-2). User-selectable data channels range from 1 to 4,096 and the sample buffer sizes range from 256 to 131,072 samples. You can change the triggers in real time without affecting your logic. ChipScope Pro Analyzer tool leads you through the process of modifying triggers and analyzing the captured data.

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

| Feature | Benefit |
|---|---|
| 1 to 4,096 user-selectable data channels | Accurately captures wide data bus functionality. |
| User-selectable sample buffers ranging in size from 256 to 131,072 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 your clock at rates up to 500 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 are 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 you through selecting the correct options. |

*Table 1-2:* **ChipScope Pro Logic Debug Features and Benefits *(Cont'd)***

| Feature | Benefit |
|---|---|
| Up to 15 independent ILA, VIO, or ATC2 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 [See Reference 18, p. 211]. |

## Design Flow

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

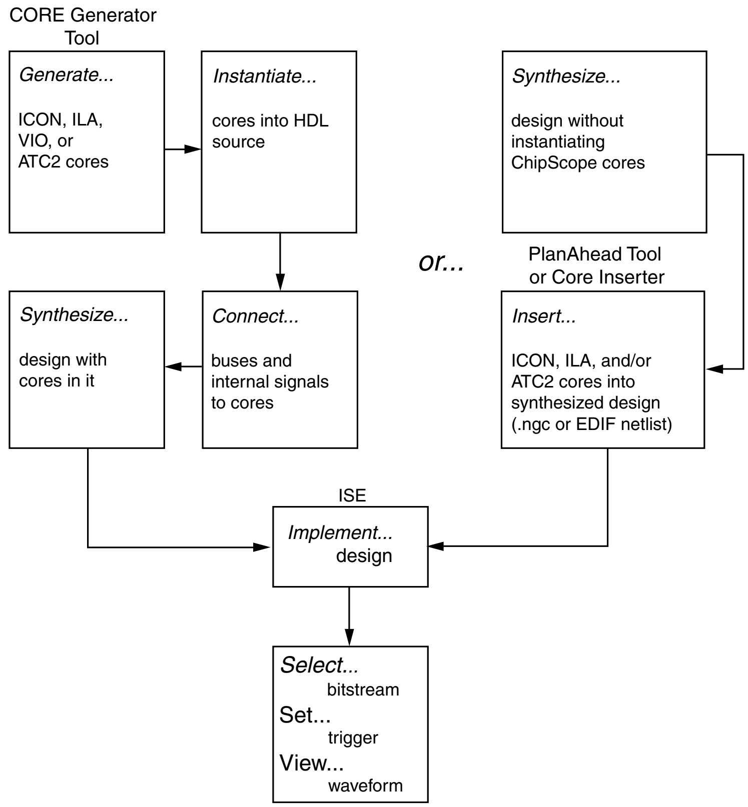


*Figure 1-2:* **ChipScope Pro Tools Design Flow**

# Using ChipScope Pro Cores in the PlanAhead Tool

You can add the ChipScope Pro cores to your design using the PlanAhead tool using one of two methods:

- HDL instantiation
- Netlist insertion

The HDL instantiation method is a two-step process that involves:

1. Using the IP Catalog in the PlanAhead tool to select, customize, and generate the desired ChipScope Pro debug core.
2. Using the HDL Editor in the PlanAhead tool to manually instantiate the IP component instance into your HDL source.

The HDL instantiation method is a good approach for users who want full control of all IP core parameters and connections to signals in their HDL design. The downside of HDL instantiation is that it requires the user to modify their source code. It can also be difficult to debug a design that is made up of multiple levels of hierarchy because signals of interest need to be brought to the debug core instance.

The Netlist insertion method is also a two step process that involves:

1. Selecting signals or nets in your design that are interesting for debugging purposes.
2. Specifying how you want these signals to be attached to debug IP cores.

The PlanAhead tool takes care of generating the debug IP core, inserting it into the design netlist, and connecting it to the nets of interest. A potential drawback to the netlist insertion method is that the HDL signals you want to debug could be optimized away or obfuscated during synthesis process. Most signals that are interesting for debug (such as outputs of registers, block RAM, etc.) are not adversely affected by the synthesis process. However, one way to ensure signals are preserved for debugging later in the flow is to attach the MARK_DEBUG attribute or property to signals in your design source (either your HDL or constraints files). For details about the MARK_DEBUG attribute and other constraints, refer to the ISE *Constraints Guide* [See Reference 14, p. 211].

The MARK_DEBUG property has the following benefits:

- Allows targeted debugging of signals in your design source without the overhead of HDL instantiation method
- Ensures signals to be debugged are preserved in the synthesized netlist
- Is compatible with Xilinx Synthesis Technology (XST) and third party FPGA synthesis tools such as Synopsys, Synplify Pro, and Mentor Graphics Precision.

For more details on the ChipScope debugging capabilities of the PlanAhead tool, refer to the *PlanAhead User Guide* [See Reference 17, p. 211].

## Using ChipScope Pro Cores in Embedded Processor and DSP Tool Flows

The cores (ICON, ILA, IBA, VIO, and ATC2) can also be used in the EDK and System Generator for DSP tool flows for embedded processor and DSP designs, respectively. For information on how to use the ChipScope Pro cores, see the EDK Platform Studio [See Reference 15, p. 211] and System Generator for DSP [See Reference 19, p. 211] documentation.

# ChipScope Pro Cores Description

## ICON Core

All the 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, VIO, and/or ATC2 cores (as shown in Figure 1-1, page 8).

For devices of the Spartan-3, Spartan-3E, Spartan-3A, and Spartan-3A DSP families, the ICON core uses either the USER1 or USER2 JTAG boundary scan instructions for communication via the BSCAN primitive. The unused USER1 or USER2 scan chain of the BSCAN primitive can also be exported for use in your application, if needed.

For all other supported devices, the ICON core uses any one of the USER1, USER2, USER3 or USER4 scan chains available via the BSCAN primitives. It is not necessary to export unused USER scan chains because each BSCAN primitive implements a single scan chain.

## ILA Core

The ILA core is a customizable logic analyzer core that can be used to monitor any internal signal of your design. Because 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.
- Control and status logic:
  - Manages the operation of the ILA core.

### ILA Trigger Input Logic

The triggering capabilities of the ILA core include many features that are necessary for detecting elaborate trigger events. These features are described in Table 1-3.

*Table 1-3:* **Trigger Features of the ILA Core**

| Feature | Description |
|---|---|
| Wide Trigger Ports | Each trigger port can be 1 to 256 bits wide. |
| Multiple Trigger Ports | Each ILA 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 must 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. |

*Table 1-3:* **Trigger Features of the ILA Core *(Cont'd)***

| Feature | Description |
|---|---|
| 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. |
| Choice of Match Unit Types | The match unit connected to each trigger port can be one of the following types:<br>• Basic comparator:<br>– Performs '=' and '<>' comparisons.<br>– Compares up to 8 bits per slice in LUT4-based[a] devices.<br>– Compares up to 19 bits per slice in Virtex-5 and Spartan-6 devices.<br>– Compares up to 20 bits per slice in all other LUT6[b]-based devices.<br>• Basic comparator w/edges:<br>– Performs '=' and '<>' comparisons.<br>– Detects high-to-low and low-to-high bit-wise transitions.<br>– Compares up to 4 bits per slice in LUT4-based devices.<br>– Compares up to 8 bits per slice in LUT6-based devices.<br>• Extended comparator:<br>– Performs '=', '<>', '>', '>=', '<', and '<=' comparisons.<br>– Compares up to 2 bits per slice in LUT4-based devices.<br>– Compares up to 8 bits per slice in LUT6-based devices.<br>• Extended comparator w/edges:<br>– Performs '=', '<>', '>', '>=', '<', and '<=' comparisons.<br>– Detects high-to-low and low-to-high bit-wise transitions.<br>– Compares up to 2 bits per slice in LUT4-based devices.<br>– Compares up to 8 bits per slice in LUT6-based devices.<br>• Range comparator:<br>– Performs '=', '<>', '>', '>=', '<', '<=', 'in range', and 'not in range' comparisons.<br>– Compares up to 1 bit per slice in LUT4-based devices.<br>– Compares up to 4 bits per slice in LUT6-based devices.<br>• Range comparator w/edges:<br>– Performs '=', '<>', '>', '>=', '<', '<=', 'in range', and 'not in range' comparisons.<br>– Detects high-to-low and low-to-high bit-wise transitions.<br>– Compares up to 1bit per slice in LUT4-based devices.<br>– Compares up to 4 bits per slice in LUT6-based devices.<br>All match units connected to a given trigger port are the same type. |

*Table 1-3:* **Trigger Features of the ILA Core *(Cont'd)***

| Feature | Description |
|---|---|
| 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 runtime to count events in the following ways:<br>• Exactly *n* occurrences<br>– Matches only when exactly *n* consecutive or non-consecutive events occur<br>• At least *n* occurrences<br>– Matches and stays asserted after *n* consecutive or non-consecutive events occur<br>• At least *n* consecutive occurrences<br>– Matches after *n* consecutive events occur, and stays asserted until the match function is not satisfied. |
| 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.<br>However, it can also be used by internal logic as an interrupt, a trigger, or to cascade multiple ILA cores together.<br>The trigger output port of the ILA core has a latency of 10 clock cycles.<br>The shape (level or pulse) and sense (active-High or active-Low) of the trigger output can be controlled at run-time. |

a. LUT4-based device families are Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 FPGAs (and the variants of these families).

b. LUT6-based device families include Zynq™ -7000 EPPs and, Virtex-5, Virtex-6, Spartan-6, Artix™-7, Kintex-7, and Virtex-7 FPGAs (and the variants of these families).

### 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 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.

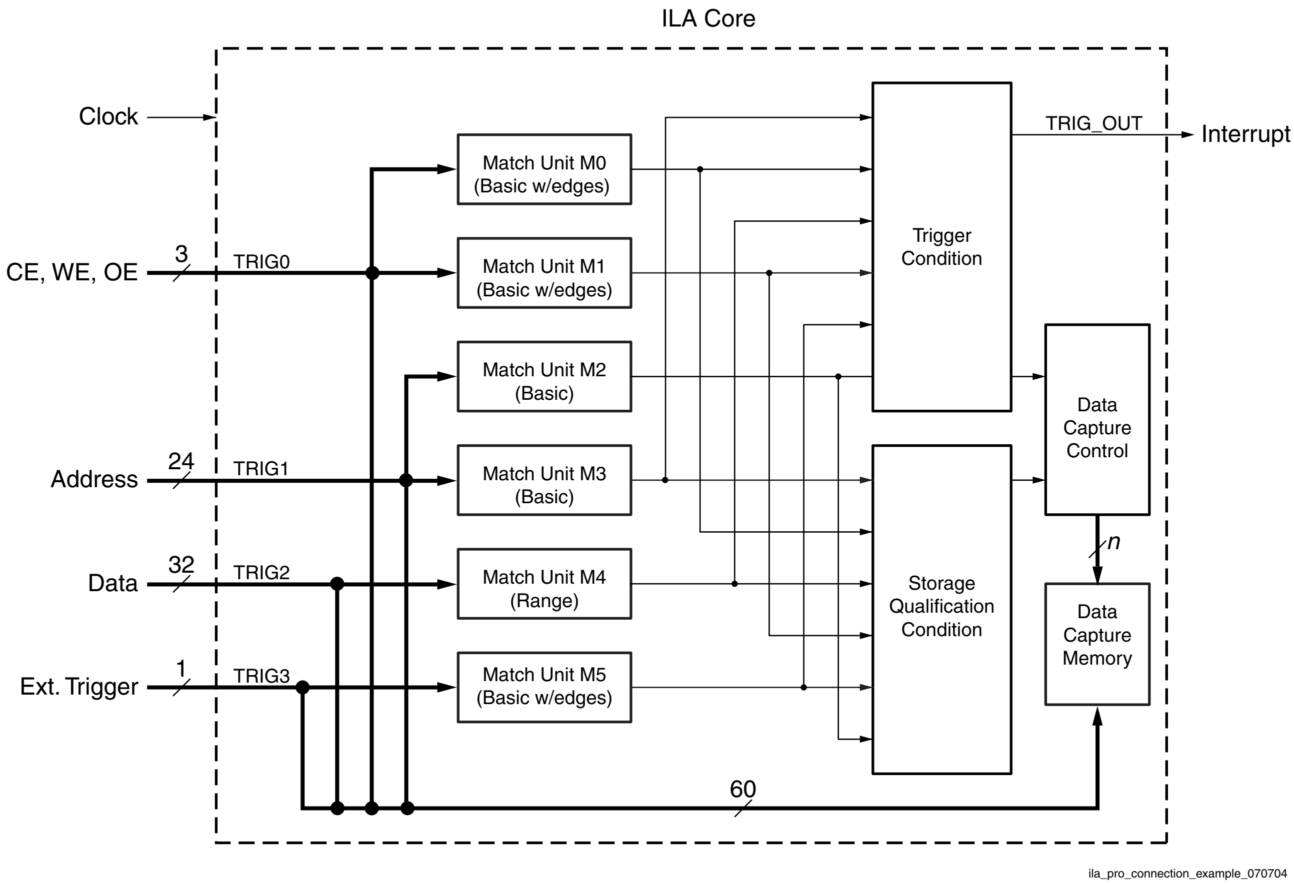


*Figure 1-3:* **ILA Core Connection Example**

### Using Trigger and Storage Qualification Conditions

The ILA core implements both trigger and storage qualification 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, page 15, 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 = “R01” (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 core 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 runtime using the ChipScope Pro Analyzer tool. The shape (level or pulse) and sense (active-High or active-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® 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 131,072 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.

After 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 until you halt 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 tool 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 because 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.

## 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 core, 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 tool.
- Synchronous inputs:
  - These are sampled using the design clock.
  - The input values are read back periodically and displayed in the ChipScope Pro Analyzer tool.
- Asynchronous outputs:
  - You define these in the ChipScope Pro Analyzer tool and drive them out of the core to the surrounding design.
  - A logical 1 or 0 value can be defined for individual asynchronous outputs.
- Synchronous outputs:
  - You *define* these in the ChipScope Pro Analyzer tool. They are *synchronized* to the design clock and *driven out* of the core to the surrounding design.
  - A logical 1 or 0 can be defined for individual synchronous outputs. Pulse trains of 16 clock cycles worth of 1’s and/or 0’s can also be defined for synchronous outputs.

### Activity Detectors

Every VIO core input has additional cells to capture the presence of transitions on the input. Because the design clock will most likely be much faster than the sample period of the ChipScope Pro Analyzer tool, it is 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 tool.

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 tool 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).

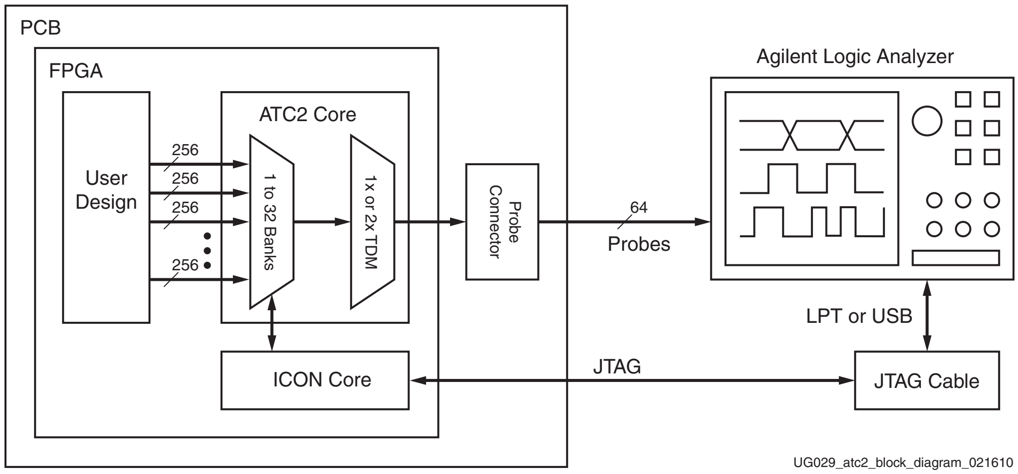


*Figure 1-4:* **ATC2 Core and System Block Diagram**

### ATC2 Core Data Path Description

The data path of the ATC2 core consists of:

- Up to 64 run-time selectable input signal banks that connect to your FPGA design
- Up to 64 output data pins that connect to the probe connectors of the Agilent logic analyzer
- Optional 2x time-division multiplexing (TDM) available on each output data pin that can be used to double the width of each individual signal bank from 64 to 128 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 [See Reference 25, p. 212]

The maximum number of data probe points available at runtime is calculated as:
(64 data ports) * (64 bits per data port) * (2x TDM) = 8,192 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).

## IBERT Core

The IBERT core has all the logic to control, monitor, and change transceiver parameters and perform bit error ratio tests. The IBERT core has three major components:

- BERT Logic
  - The BERT logic instantiates the actual transceiver component, and contains the pattern generators and checkers. A variety of patterns are available, including simple clock-type patterns, full PRBS (pseudo random bit sequence) patterns, and framed counter patterns utilizing commas and comma detection.
- Dynamic Reconfiguration Port (DRP) Logic
  - Each transceiver has a Dynamic Reconfiguration Port (DRP) on it, so that transceiver attributes can be changed in system. All attributes and DRP addresses are readable and writable in the IBERT core. The DRP for each transceiver can be accessed individually.
- Control and status logic
  - Manages the operation of the IBERT core.

### IBERT Design Flow

Because the IBERT is a self-contained design, the design flow is very simple. When using the ChipScope IBERT Core Generator to generate IBERT core designs for Virtex-5 devices, the design directory and bit file name are specified, options are chosen, and the Generator runs the entire implementation flow, including bitstream creation, in one step.

The design flow for generating IBERT core designs for Virtex-7, Kintex-7, Virtex-6, and Spartan-6 devices are very similar except the Xilinx CORE Generator tool is used. The main difference is that the design directory and device information is specified in the Xilinx CORE Generator project. In both cases, you are not required to explicitly run any other Xilinx tool to generate an IBERT core design bit file.

### IBERT Feature Descriptions

The features of the IBERT core vary according to the targeted FPGA device architecture. The MGT features that are supported are as follows:

- IBERT v2.0 for Virtex-5 FPGA GTX Transceivers (Table 1-4, page 21)
  - Full PMA control, including differential swing, emphasis, RX Equalization, and DFE.
  - Ability to change line rate at runtime.
  - Limited PCS support, including loopback (8b/10b encoding, clock correction, and channel bonding are not supported).
  - 40-bit fabric data width (4-byte mode).
- IBERT v2.0 core for Virtex-6 FPGA GTX transceivers (Table 1-5, page 23)
  - Full PMA control, including differential swing, emphasis, RX equalization, and DFE.
  - Ability to change line rates at runtime.
  - Ability to set reference clock sources at generate time.
  - Limited PCS support, including loopback. Pattern encoding, clock correction and channel bonding are not supported.

- IBERT v2.0 core for Virtex-6 FPGA GTH transceivers (Table 1-6, page 24)
  - Full PMA control, including differential swing, emphasis, RX equalization, and DFE.
  - Ability to set reference clock sources at generate time.
  - Limited PCS support, including loopback. Pattern encoding, clock correction, and channel bonding are not supported.
  - TX Diff Swing.
  - TX Pre-Emphasis and Post-Emphasis.
- IBERT v2.0 core for Spartan-6 FPGA GTP transceivers (Table 1-7, page 25)
  - Full PMA control, including differential swing, emphasis, RX equalization, and DFE.
  - Ability to change line rates at runtime.
  - Ability to set reference clock sources at generate time.
  - Limited PCS support, including loopback. Pattern encoding, clock correction, and channel bonding are not supported.
  - TX Diff Swing.
  - TX Pre-Emphasis.
- IBERT v2.02a core for 7 Series FPGA GTX transceivers (Table 1-8, page 26)
  - PMA control, including differential swing, and emphasis.
  - Ability to change line rates at generate time.
  - Ability to set reference clock sources at generate time.
  - Limited PCS support, including loopback. Pattern encoding, clock correction, and channel bonding are not supported.
- IBERT v2.00a core for 7 Series FPGA GTH Transceivers (Table 1-9, page 27.)
  - PMA control, including differential swing, and emphasis.
  - Ability to change line rates at generate time.
  - Ability to set reference clock sources at generate time.
  - Limited PCS support, including loopback. Pattern encoding, clock correction, and channel bonding are not supported.

*Table 1-4:* **IBERT v2.0 Core for the Virtex-5 FPGA GTX Transceivers**

| Feature | Description |
|---|---|
| Multiple GTX Transceivers | Up to eight transceivers can be selected per design. |
| Pattern Generator | One pattern generator per selected GTX transceiver is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTX transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTX transceiver is used. The same pattern set is available as the pattern generator. The pattern can be chosen for each GTX transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTX_DUAL tile can be either 32- or 40-bits wide and selectable at generate time. |

*Table 1-4:* **IBERT v2.0 Core for the Virtex-5 FPGA GTX Transceivers *(Cont'd)***

| Feature | Description |
|---|---|
| BERT Parameters | Number of bits received in error and total number of words received are gathered dynamically and read out by the ChipScope Pro Analyzer tool. |
| Polarity | The polarity of the TX or RX side of each GTX transceiver can be changed at runtime. |
| Reset | Each GTX transceiver and its BER counters can be reset independently. A reset is also available to reset the entire MGT, including PLLs. |
| Link and Lock Status | Link, DCM, and PLL lock status are gathered for each GTX transceiver in the core. |
| DRP Read | The contents of the DRP space for each GTX transceiver can be read independently of all others. |
| DRP Write | The contents of the DRP space for each GTX transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTX transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTX transceiver's ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

*Table 1-5:* **IBERT v2.0 Core for the Virtex-6 FPGA GTX Transceivers**

| Feature | Description |
|---|---|
| Multiple GTX Transceivers | Up to eight transceivers can be selected per design. |
| Pattern Generator | One pattern generator per selected GTX transceiver is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTX transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTX transceiver is used. The same pattern set is available as the pattern generator. The pattern can be chosen for each GTX transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTX transceiver can be either 16- or 20-bits wide and selectable at generate time. |
| BERT Parameters | Number of bits received in error and total number of words received are gathered dynamically and read out by the ChipScope Pro Analyzer tool. |
| Polarity | The polarity of the TX or RX side of each GTX transceiver can be changed at runtime. |
| Reset | Each GTX transceiver and its BER counters can be reset independently. A reset is also available to reset the entire MGT, including PLLs. |
| Link and Lock Status | Link, DCM, and PLL lock status are gathered for each GTX transceiver in the core. |
| DRP Read | The contents of the DRP space for each GTX transceiver can be read independently of all others. |
| DRP Write | The contents of the DRP for each GTX transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTX transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTX transceiver's ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

*Table 1-6:* **IBERT v2.0 Core for the Virtex-6 FPGA GTH Transceivers**

| Feature | Description |
|---|---|
| Multiple GTH Transceivers | Up to sixteen transceivers can be selected per design |
| Pattern Generator | One pattern generator per selected GTH transceiver (four per QUAD) is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTH transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTH transceiver is used (four per QUAD). The same pattern set is available as the pattern generator. The pattern can be chosen for each GTH transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTH QUAD can be either 16- or 20-bits wide and selectable at generate time. |
| BERT Parameters | Number of bits received in error and total number of words received are gathered dynamically and read out by the ChipScope Pro Analyzer tool. |
| Polarity | The polarity of the TX or RX side of each GTH transceiver can be changed at runtime. |
| Reset | The BER counters for each GTH transceiver can be reset independently. A reset is also available for the entire GTH QUAD, including PLLs. |
| Link and Lock Status | Link, DCM, and PLL lock status are gathered for each GTH transceiver in the core. |
| DRP Read | The DRP space for each GTH transceiver can be read independently of all others. |
| DRP Write | The contents of the DRP for each GTH transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTH transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTH transceiver ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

*Table 1-7:* **IBERT v2.0 Core for the Spartan-6 FPGA GTP Transceivers**

| Feature | Description |
|---|---|
| Multiple GTP Transceivers | Up to eight transceivers can be selected per design. |
| Pattern Generator | One pattern generator per selected GTP transceiver (two per DUAL) is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTP transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTP transceiver is used (two per DUAL). The same pattern set is available as the pattern generator. The pattern can be chosen for each GTP transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTP transceiver is 20 bits. |
| BERT Parameters | Number of bits received in error and total number of words received are gathered dynamically and read out by the ChipScope Pro Analyzer tool. |
| Polarity | The polarity of the TX or RX side of each GTP transceiver can be changed at runtime. |
| Reset | Each GTP transceiver and its BER counters can be reset independently. A reset is also available to reset the entire GTP transceiver, including PLLs. |
| Link and Lock Status | Link, DCM, and PLL lock status are gathered for each GTP transceiver in the core. |
| DRP Read | The contents of the DRP space for each GTP transceiver can be read independently of all others. |
| DRP Write | The contents of DRP for each GTP transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTP transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTP transceiver ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

*Table 1-8:* **IBERT v2.02a Core for the 7 Series FPGA GTX Transceivers**

| Feature | Description |
|---|---|
| Multiple GTX Transceivers | Up to fifteen transceivers can be selected per design. |
| Pattern Generator | One pattern generator per selected GTX transceiver is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTX transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTX transceiver is used. The same pattern set is available as the pattern generator. The pattern can be chosen for each GTX transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTX transceiver can be either 32- or 40-bits wide and selectable at generate time. |
| Polarity | The polarity of the TX side of each GTX transceiver can be changed at runtime. |
| Reset | Each GTX transceiver can be reset independently. A reset is also available to reset the entire MGT, including PLLs and CPLLs. |
| Link and Lock Status | Link, and CPLL/QPLL lock status are gathered for each GTX transceiver in the core. |
| DRP Read | The contents of the DRP space for each GTX transceiver can be read independently of all others. |
| DRP Write | The contents of the DRP for each GTX transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTX transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTX transceiver's ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

*Table 1-9:* **IBERT v2.00a core for 7 Series FPGA GTH Transceivers**

| Feature | Description |
|---|---|
| Multiple GTX Transceivers | Up to fifteen transceivers can be selected per design. |
| Pattern Generator | One pattern generator per selected GTX transceiver is used. PRBS 7-bit, PRBS 15-bit PRBS 23-bit, PRBS 31-bit, Clk 2x, and Clk 10x patterns are available. The desired pattern from that set can be selected individually for each GTX transceiver at runtime. |
| Pattern Checker | One pattern checker per selected GTX transceiver is used. The same pattern set is available as the pattern generator. The pattern can be chosen for each GTX transceiver at runtime. |
| Fabric Width | The FPGA fabric interface to the GTX transceiver can be either 32 or 40 bits wide and selectable at generate time. |
| Polarity | The polarity of the TX side of each GTX transceiver can be changed at runtime. |
| Reset | Each GTX transceiver can be reset independently. A reset is also available to reset the entire MGT, including PLLs and CPLLs. |
| Link and Lock Status | Link, and CPLL/QPLL lock status are gathered for each GTX transceiver in the core. |
| DRP Read | The contents of the DRP space for each GTX transceiver can be read independently of all others. |
| DRP Write | The contents of the DRP for each GTX transceiver can be changed at runtime, with single-bit granularity. |
| Ports Read | The contents of the registers that monitor the GTX transceiver ports can be read independently of others. |
| Ports Write | The contents of the registers that control the GTX transceiver's ports can be changed at runtime. |
| Status | The dynamic status information for the entire core can be read out of the core at runtime. |

You can modify many options in the ILA, 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 1-10 shows which design changes require resynthesizing.

*Table 1-10:* **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[1] |
| Changing the data signal source | No[1] |
| Changing the ILA clock signal | Yes |
| Changing the sample buffer depth | Yes |

**Notes:**

1. The ability to change existing trigger and/or data signal sources is supported by the ISE FPGA Editor tool, regardless of how the ChipScope cores were added to the design.

# System Requirements

## Operating System Requirements

The ChipScope Pro operating system requirements are described in the *ISE Design Suite Installation and Licensing Guide* [See Reference 14, p. 211].

## Software Tools Requirements

The Xilinx CORE Generator, ChipScope Pro Core Inserter, IBERT Core Generator, and CSE/Tcl tools require that ISE implementation tools 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.) CSE/Tcl requires the Tcl shell (called `xtclsh)` that is included in the ChipScope Pro and ISE tool installations.

***Note:*** The version (including update revision) of the ChipScope Pro tools must match the version (including update revision) of the ISE tools that is used to implement the design that contains the ChipScope Pro cores.

## Communications Requirements

The ChipScope Pro Analyzer tool supports the following download cables (see Table 1-11) for communication between the PC and the devices in the JTAG boundary scan chain:

- Platform Cable USB-II
- Platform Cable USB
- Parallel Cable IV
- Digilent JTAG-SMT1 and JTAG-HS1 USB-to-JTAG download cables
- ByteTools Catapult EJ-1 Ethernet-to-JTAG cable [See Reference 27, p. 212]

***Note:*** Certain competing operations on the cable or device (such as configuring the device using the iMPACT tool while communicating with the ILA core in the device using the ChipScope Pro Analyzer tool) may render the design-under-test unusable. When in doubt about the results of competing cable/device operations, close the cable connection from the ChipScope Pro Analyzer tool until the competing operation has completed.

*Table 1-11:* **ChipScope Pro Download Cable Support**

| Download Cable | Features |
|---|---|
| Platform Cable USB II and Platform Cable USB[1] | • Uses the USB port (USB 2.0 or USB 1.1) to communicate with the boundary scan chain of the board-under-test<br>• Downloads at speeds up to 12 Mb/s throughput<br>• Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 5V down to 1.5V<br>• Windows and Red Hat Linux OS support |
| Parallel Cable IV[1] | • Uses the parallel port (i.e., printer port) to communicate with the boundary scan chain of the board-under-test<br>• Downloads at speeds up to 5 Mb/s throughput<br>• Contains an adjustable voltage interface that enables it to communicate with systems and device I/Os operating at 5 V down to 1.5 V<br>• Windows and Red Hat Linux OS support |
| Digilent JTAG-SMT1 and JTAG-HS1 USB-to-JTAG download cables | • Uses the USB port (USB 2.0 or USB 1.1) to communicate with the boundary scan chain of the board-under-test<br>• Downloads at speeds up to 30 Mb/s throughput<br>• Adjustable voltage interface supports device I/Os operating at 5 V down to 1.5 V<br>• Windows and Linux OS support<br>For more information, see the Digilent web page: http://www.digilentinc.com |
| ByteTools Catapult EJ-1 Ethernet-to-JTAG Cable | • Uses the Ethernet port to communicate with the boundary scan chain of the board-under-test<br>• For more information, see ByteTools Web page [See Reference 27, p. 212] |

**Notes:**

1. The Parallel Cable IV and Platform Cable USB cables are available for purchase from the Xilinx Online Store [See Reference 20, p. 211] (choose **Online Store > Programming Cables**).

### Board Requirements

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

- One or more supported 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 IV, Platform Cable USB, Digilent JTAG-HS1, or ByteTools download cable, then VREF (1.5-5.0V) and GND headers must be available for connecting to the Parallel Cable IV cable

## Software Installation and Licensing

The ChipScope Pro Analyzer tool can be installed both as a standalone ISE Lab Tools product (for example, in a lab environment where only the ChipScope Pro Analyzer tool is needed) or along with the rest of the ISE Design Suite tools. For software installation and licensing instructions, refer to the ISE Design Suite *Installation and Licensing Guide*, available in the ISE Documentation [See Reference 14, p. 211].

Chapter 2

# Using the Core Generator Tools

## Overview

This chapter provides instructions to use the Xilinx® CORE Generator™ tool to generate ChipScope™ Pro cores. As a group, these cores are called the ChipScope Pro logic debug cores.

After generating the cores, you can use the instantiation templates that are generated by the CORE Generator tool to quickly and easily insert the cores into your VHDL or Verilog design. After completing the instantiation and running synthesis, you can implement the design using the ISE® implementation tools.

## Using the Xilinx CORE Generator Tool with ChipScope Pro Cores

Before you can select the ChipScope Pro cores for generation, you must first set up a project in the CORE Generator tool. After setting up your project with the appropriate settings, you can find the ChipScope Pro cores in the CORE Generator tool by first clicking on the **View by Function** tab in the upper left panel, then by expanding the **Debug & Verification** and **ChipScope Pro** core sections of the browser. You can also find the ChipScope Pro cores by using the **View by Name** tab.

***Note:*** Core instantiation templates for the ChipScope Pro cores are found in the .vho file (for VHDL language flows) and .veo file (for Verilog language flows) that are created as part of the core generation process. Refer to the CORE Generator Help for more details.

***Note:*** Due to the nature of the ChipScope Pro cores, core simulation files created by the Xilinx CORE Generator tool for the ChipScope Pro cores are not valid for simulation. Empty black box entity architectures (for VHDL) or modules (Verilog) must be used for the ChipScope Pro cores when simulating designs that contain these cores.

### Core Generation

#### Generating Cores for ICON, ILA, VIO and ATC2 Cores

Core generation instructions for ICON, ILA, VIO, and ATC2 cores are included in the datasheets listed below. To locate the specific datasheets, navigate to http://www.xilinx.com, click **Documentation** at the top of the page, click the IP tab, and search under **Embedded Processing > Debug and Trace**.

- DS646, "LogiCORE IP ChipScope Pro Integrated Controller (ICON) (v1.06a)."
- DS299, "LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v1.05a)."
- DS284, "LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) (1.05a)."
- DS650, "Agilent Trace Core 2 (ATC2) (v1.05a)."

## Generating IBERT v2.0 Cores

Instructions for IBERT v2.0 core generation are included in the datasheets listed below. To locate the datasheets, navigate to http://www.xilinx.com/support/documentation, click the IP tab, and search under **Embedded Processing > Debug and Trace**.

- DS774 "LogiCORE IP ChipScope Pro IBERT for Virtex-5 GTX FPGA Transceivers."
- DS732 "LogiCORE IP ChipScope Pro IBERT for Virtex-6 GTX FPGA Transceivers."
- DS775 "LogiCORE IP ChipScope Pro IBERT for Virtex-6 GTH FPGA Transceivers."
- DS782 "LogiCORE IP ChipScope Pro IBERT for Spartan-6 GTP FPGA Transceivers."
- DS873 "LogiCORE IP ChipScope IBERT for 7 Series GTH Transceivers."
- DS872 "LogiCORE IP ChipScope IBERT for 7 Series GTX Transceivers."

Chapter 3

# Using the ChipScope Pro Core Inserter

## ChipScope Pro Core Inserter Overview

The ChipScope™ Pro Core Inserter is a post-synthesis tool used to generate a netlist that includes your design as well as parameterized ICON (integrated controller), ILA (integrated logic analyzer), and ATC2 (Agilent trace core) cores as needed. The ChipScope Pro Core Inserter gives you the flexibility to quickly and easily use the debug functionality to analyze an already synthesized design and to do so without any HDL instantiation.

***Note:*** The VIO (virtual input/output) and IBERT (internal bit error ratio) cores are not supported in the ChipScope Pro Core Inserter tool.

## Using the ChipScope Pro Core Inserter with the PlanAhead Tool

The ChipScope Pro Core Inserter tool is not intended for use with the PlanAhead™ tool. Instead, debug core insertion functionality has been incorporated into the PlanAhead tool environment. To make it easier to migrate from the ChipScope Pro Core Inserter tool to the ISE® PlanAhead tool environment, a CDC import command is provided in the PlanAhead tool. This allows the importing of the ChipScope Pro Core Inserter CDC project file into a PlanAhead tool project. For more details on the ChipScope core insertion capabilities of the PlanAhead tool, please refer to the *PlanAhead User Guide* [See Reference 17, p. 211].

## Using the ChipScope Pro Core Inserter with ISE Project Navigator

This section is provided for users of the Windows or Linux versions of ChipScope Pro and ISE tools.

The ChipScope Pro 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 recognizes and invokes the ChipScope Pro Core Inserter tool during the appropriate steps in the implementation flow. For more information on how the Project Navigator and the ChipScope Pro Core Inserter are integrated, refer to the Project Navigator section of the *ISE Software Documentation* [See Reference 14, p. 211].

## ChipScope Definition and Connection Source File

To use the ChipScope Pro Core Inserter tool to insert cores into a design processed by Project Navigator:

1. Add the 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. Click **Next** to advance to the Summary panel, then click **Finish** to create the file.

      ***Note:*** The ChipScope Definition and Connection File source type is only listed if Project Navigator detects a ChipScope Pro installation (software versions must match).
   b. To add an existing .cdc file, select **Project > Add Source** or **Project > Add Copy of Source**, then browse for the existing .cdc file.

      After selecting the .cdc file in the file browser, click **Open** and then click **OK** on the Adding Source Files dialog box. The .cdc file now displays in the Sources in Project window underneath the associated design module.
2. To create the cores and complete the signal connections, double-click the .cdc file in the Design panel. 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 ChipScope Pro Core Inserter tool as necessary (as shown in "ChipScope Pro Core Inserter Features," page 38), then close the ChipScope Pro Core Inserter tool.
4. When the associated top-level design is implemented in Project Navigator, the 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 Project Navigator settings help you implement a design with cores:

1. If you use the XST synthesis tool, set the **Keep Hierarchy** option to **Yes** or **Soft** 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 core insertion stage of the flow. If you do not use the **Keep Hierarchy** option, some of your nets and/or components can be combined with other logic into new components or otherwise optimized away. To keep the design hierarchy:
   a. Right-click the **Synthesize - XST** process and select **Process Properties**.
   b. In the Synthesis Options category, make sure the **Keep Hierarchy** property is set to **Soft** or **Yes** and click **OK**.
2. If you plan to use the ChipScope Pro Analyzer tool to configure older devices, you must make sure the bitstream generation options are set properly, as described in the steps below. The devices to which these steps apply include:
   - Virtex®-4 devices (or earlier)
   - Spartan®-3, Spartan-3E, Spartan-3A, Spartan-3AN (or earlier) devices

   a. In Project Navigator, right-click the **Generate Programming File** process and select **Process Properties**.

b. Select the **Startup Options** category.

c. Set the FPGA Start-Up Clock dropdown to **JTAG Clock**.

***Note:*** This option is not necessary for later FPGA device families or if you are planning to configure the FPGA device from a PROM or Flash device.

# Using the ChipScope Pro Core Inserter with Command Line Implementation

## Command Line Flow Overview

The ChipScope Pro Core Inserter supports a basic command line for batch core insertion. As shown in Figure 3-1, the ChipScope Pro Core Inserter command line flow consists of three steps:

1. Creating a CDC Project
2. Editing the CDC Project
3. Inserting Cores

The ChipScope Pro Core Inserter is invoked prior to calling ngdbuild to instantiate debug cores in the design. Nets are selected for debug using the Edit CDC Project mode to display the GUI and save net selections to the project. After successfully implementing the design and configuring the Xilinx® device with the resulting bitstream, the Analyzer is used for in-circuit design debug and verification. To change debug net selections or core settings after configuration, iterate back to the Edit CDC Project step and proceed through the flow to create a new bitstream for further debug and verification.

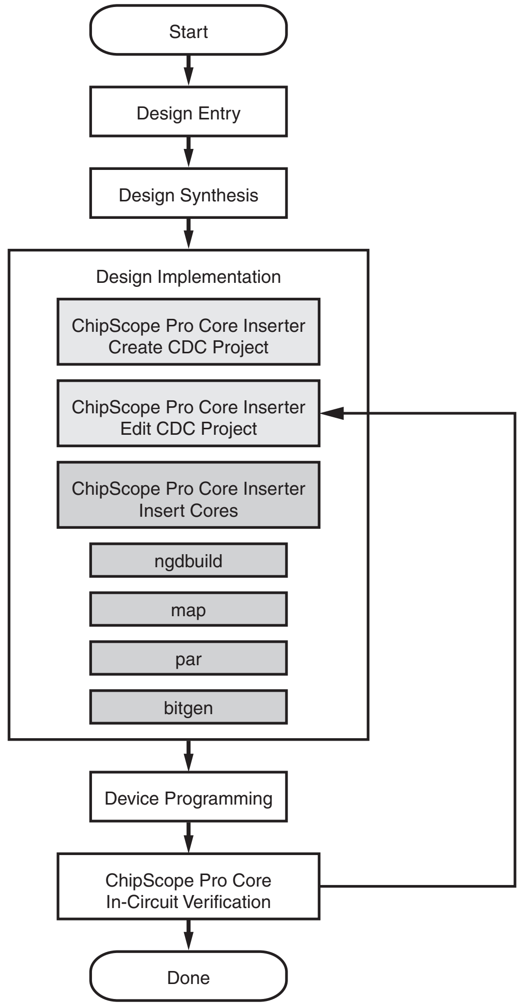


*Figure 3-1:* **Command Line Core Inserter Flow**

## Create CDC Project Step

The Create CDC Project step of the command line ChipScope Pro Core Inserter flow is used to create an empty skeleton CDC project file, as shown in Figure 3-2. The command line signature for this step is:

```
inserter -create <project.cdc>
```

***Note:*** This step does not bring up the ChipScope Pro Core Inserter graphical user interface (GUI).

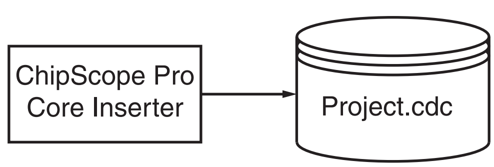


*Figure 3-2:* **Create CDC Project Step**

## Edit CDC Project Step

The Edit CDC Project step of the command line ChipScope Pro Core Inserter flow is used to bring up the ChipScope Pro Core Inserter GUI to edit an existing CDC project (see Figure 3-3). The ngcbuild tool is called during this step with the specified arguments following the -ngcbuild argument. The ngcbuild tool combines all netlists associated with the design into a single complete NGC netlist file. This allows the ChipScope Pro Core Inserter tool to provide full debug access to all levels and nodes in the design.

The command line signature for this step is:

```
inserter -edit <project.cdc> -ngcbuild [-p <partname>] [{-sd
<source_dir>}] [-dd <output_dir>] [-i] <inputdesign.{edn|ngc}>
<outputdesign.ngc>
```

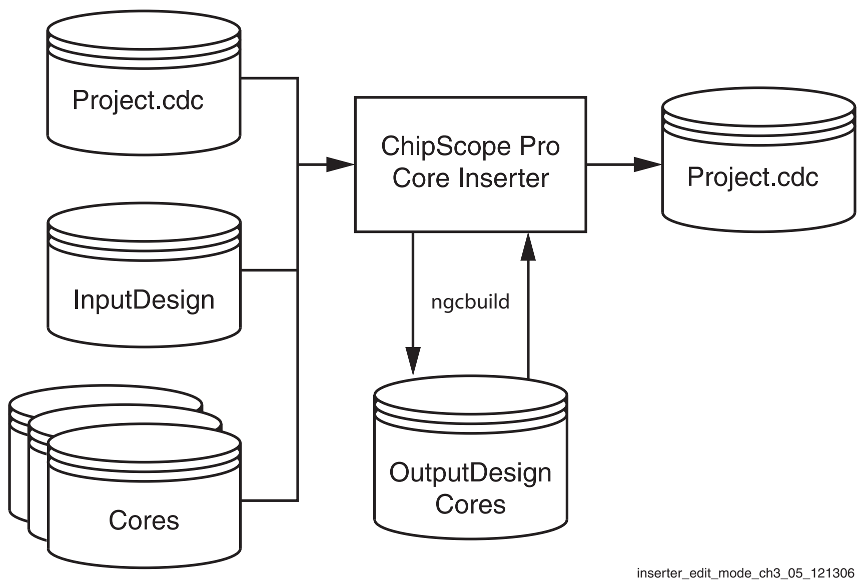


*Figure 3-3:* **Edit CDC Project Step**

## Insert Cores Step

The Insert Cores step of the command line ChipScope Pro Core Inserter flow is used to insert cores into a design based on an existing CDC project (see Figure 3-4). The ngcbuild tool is called during this step with the specified arguments following the -ngcbuild argument. The ngcbuild tool combines all netlists associated with the design into a single complete NGC netlist file. The cores are inserted into this single complete netlist.

The command line signature for this step is:

```
inserter -insert <project.cdc> -ngcbuild [-p <partname>] [{-sd
<source_dir>}] [-dd <output_dir>] [-i] <inputdesign.{edn|ngc}>
<outputdesign.ngc>
```

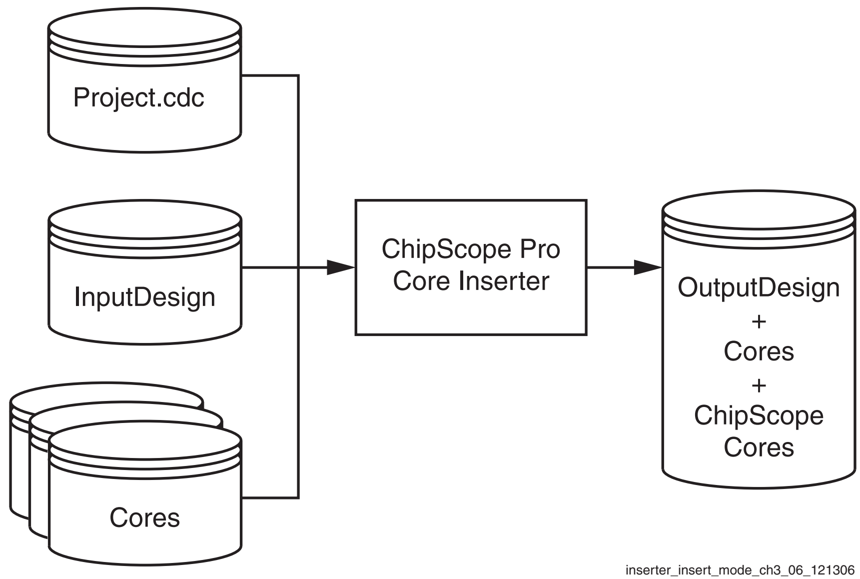


*Figure 3-4:* **Insert Cores Step**

# ChipScope Pro Core Inserter Features

## Working with Projects

Projects saved in the ChipScope Pro 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 Analyzer to import signal names.

When the ChipScope Pro Core Inserter is first opened, all the relevant fields are completely blank. Using the command **File > New** also results in this condition.

### 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 it or click **Open**.

### Saving Projects

If a project has changed during the course of a session, you are prompted to save the project upon exiting the ChipScope Pro 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 ChipScope Pro 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 ChipScope Pro 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** 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 Pro 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 49 for more information about setting these preferences.

### Inserting the Cores

ICON, ILA, 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 ChipScope Pro Core Inserter

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

## Specifying Input and Output Files

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

1. Specify the Input Design Netlist.
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.)

***Note:*** When the ChipScope Pro 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. In this case, these fields can only be changed by the Project Navigator tool and cannot be modified directly in the ChipScope Pro Core Inserter.

## Project Level Parameters

Three project level parameters (device family, SRL usage, and RPM usage) must be specified for each project.

### 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-6.

### Using SRLs

The **Use SRLs** checkbox determines whether or not the cores are generated using SRL16 and/or SRL32 components. 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 SRLs** checkbox is enabled by default to generate cores that use the optimized SRL16 technology.

### Using RPMs

The **Use RPMs** checkbox is used to select whether or not the individual cores are *relationally placed macros* (RPMs). This option places restraints on the place-and-route tool to optimize placement of all the logic for the core in one area. If your design uses most of the resources in the device, these placement constraints might 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**.

## Core Utilization

The Core Utilization panel on the left side of the ChipScope Pro Core Inserter tool main window displays an estimated count of the look-up table (LUT), flip-flop (FF), and block RAM (BRAM) resources that are consumed by the ChipScope cores that are being inserted into the design netlist. The core resource utilization counts are updated based on the selection of various core parameters that affect the makeup of the cores being inserted into the design netlist.

***Note:*** The LUT Count and FF Count core utilization features are only available for the Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 device families (and the variants of these families). The BRAM Count core utilization feature is available for all supported device families.

## Choosing ICON Options

The first options that must be specified are for the ICON core. The ICON core is the controller core that connects all ILA and ATC2 cores to the JTAG (Joint Test Action Group, IEEE standard) boundary scan chain.

### Selecting the Boundary Scan Chain

The ChipScope Pro Analyzer tool can communicate with the cores using either the USER1, USER2, USER3, or USER4 boundary scan chains. You can select the desired scan chain from the **Boundary Scan Chain** pull-down list. This option is not available when targeting the Spartan-3, Spartan-3E, Spartan-3A, or Spartan-3A DSP device families (or the variants of these families).

## Choosing ILA Trigger Options and Parameters

A new ILA unit is created in the device hierarchy on the left when the **New ILA Unit** button is clicked. The next step is to set up the ILA unit. The first ILA core tab panel sets up the trigger options for the ILA core.

### 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 select the number of trigger ports from the **Number of Input 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 data capture. 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 core (Table 3-1).

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

| Type | Bit Values[1] | Match Function | Bits Per Slice[2] | Description |
|---|---|---|---|---|
| Basic | 0, 1, X | '=', '<>' | LUT4-based: 8<br>Virtex-5, Spartan-6: 19<br>Other LUT6-based: 20 | 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, N | '=', '<>' | LUT4-based: 4<br>LUT6-based: 8 | 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 | '=', '<>', '>', '>=', '<', '<=' | LUT4-based: 2<br>LUT6-based: 16 | Can be used for comparing address or data signals where magnitude is important. |
| Extended w/edges | 0, 1, X, R, F, B, N | '=', '<>', '>', '>=', '<', '<=' | LUT4-based: 2<br>LUT6-based: 8 | Can be used for comparing address or data signals where a magnitude and transition detection are important. |

*Table 3-1:* **ILA Trigger Match Unit Types *(Cont'd)***

| Type | Bit Values[1] | Match Function | Bits Per Slice[2] | Description |
|---|---|---|---|---|
| Range | 0, 1, X | '=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range' | LUT4-based: 1<br>LUT6-based: 8 | 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, N | '=', '<>', '>', '>=', '<', '<=', 'in range', 'not in range' | LUT4-based: 1<br>LUT6-based: 4 | Can be used for comparing address or data signals where a range of values and transition detection are important. |

**Notes:**

1. Bit values: '0' = "logical 0"; '1' ="logical 1"; 'X' ="don't care"; 'R'= "0-to-1 transition"; 'F'= "1-to-0 transition"; 'B'= "any transition"; 'N' = "no transition".
2. 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. LUT4-based device families are Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 FPGAs (and the variants of these families). LUT6-based device families are Zynq™- 7000 EPPs. and Virtex-5, Virtex-6, Spartan-6, Artix™-7, Kintex™-7, and Virtex-7 FPGAs.

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 runtime 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 is not 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* is a standard Boolean equation trigger condition that 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 3-5.

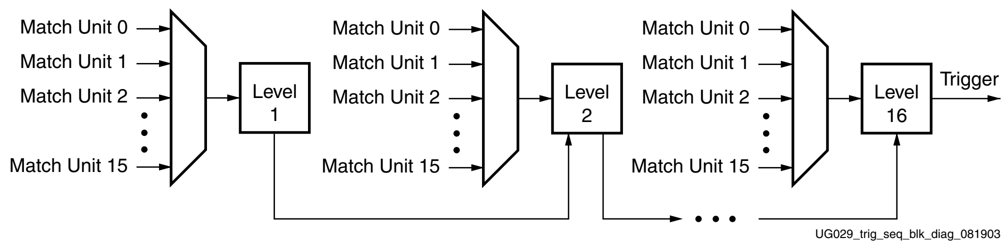


*Figure 3-5:* **Trigger Sequencer Block Diagram with 16 Levels and 16 Match Units**

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 runtime on a per level basis to transition from one level to the next. The trigger sequencer can be configured at runtime 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 trigger output port of the ILA core trigger condition module cannot be enabled in the ChipScope Pro Core Inserter tool. It can only be enabled by using the CORE Generator™ tool.

## Choosing ILA Core Capture Parameters

The second tab in the ChipScope Pro Core Inserter is used to set up the capture parameters of the ILA core.

### 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. The CORE Generator and ChipScope Pro Core Inserter have a resource utilization estimator feature that indicates exactly how many block RAM resources are used for a given combination of data width and data depth parameters.

### 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 4,096 bits (or 256 bits for Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 devices).

### Selecting the Data Type

The data captured by the ILA trigger port is controlled by the Data Same as Trigger checkbox and can come from two source types:

- Data Same as Trigger (checked ON)
  - You can use the signals connected to the trigger ports for triggering and for data capture. This mode is very common in most logic analyzers because 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 is not 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 4,096 bits (or 256 bits for Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 devices).
- Data Separate from Trigger (checked OFF)
  - 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
  - In the case of data not same as trigger, the Data Port Width parameter must be specified.

### 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. Use these checkboxes to select the individual trigger ports to be included in the aggregate data port. A maximum data width of 4,096 bits (or 256 bits for Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, and Virtex-4 devices) applies to the aggregate selection of trigger ports.

## Choosing ATC2 Data Capture Settings

If you are inserting an ATC2 core, the following sections describe the ATC2 data capture settings.

### Capture Mode

The Capture Mode setting 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.

### Max Frequency Range

The Max Frequency Range parameter is used to specify the maximum frequency range in which you expect to operate the ATC2 core. The implementation of the ATC2 core is optimized for the maximum frequency range selection. The valid maximum frequency ranges are 0-100 MHz, 101-200 MHz, 201-300 MHz, and 301-500 MHz. The maximum frequency range selection only has an affect on core implementation when the Capture Mode is set to STATE mode.

### Enable Auto Setup

The Enable Auto Setup option is used to enable a feature that allows the Agilent Logic Analyzer to automatically set up the appropriate ATC2 pin to Logic Analyzer pod connections. This feature also allows the Agilent Logic Analyzer to automatically determine the optimal phase and voltage sampling offsets for each ATC2 pin. This feature is enabled by default.

### Enable "Always On" Mode

The Enable "Always On" Mode option is used to force an ATC2 core to always enable its internal logic and output buffers. The "Always On" mode ensures that signal bank 0 is driven out to the ATD pins upon FPGA device configuration. This mode makes it possible to capture events that immediately follow device configuration without having to first set up the ATC2 core manually. This feature is disabled by default and is only available when the capture mode is set to TIMING mode.

### 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 must 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 64.

### 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, 32, or 64.

## TDM Rate

The *time division multiplexing* (TDM) rate is used to increase the amount of data transmitted over each data pin by as much as 200 percent. 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) because the ATCK pin is used as an extra data pin.

## Pin Parameters

The settings in the Individual Pin Settings table control the location, I/O standard, output drive and slew rate of each individual ATCK and ATD pin. 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 bring the ATCK and ATD pins through every level of hierarchy to the top-level of your design manually; 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.

### 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 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* in the *Xilinx Software Documentation* [See Reference 14, p. 211].

### 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 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 Signals

The **Net Connections** tab allows you to choose the signals to connect to the ILA core. If trigger is *separate* from data, then the clock, trigger, and data ports must be specified. When trigger *equals* data, only the clock and trigger/data ports must be specified. Double-click the **CLOCK PORT** label or click the plus sign (+) next to it to expand. No connection has been made, so the connection appears in red.

The ATC2 **Net Connections** tab allows you to choose the signals to connect to the ATC2 core. The clock and data ports must be specified. Expand the **Clock Net** label. No connection has been made, so the connection appears in red.

To change any core connection, select **Modify Connections**. The Select Net dialog box now appears. This dialog box provides an easy interface to choose nets to connect to the ILA 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 nets for the selected structure hierarchy level of the design 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 might 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 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 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 core appear in the pane at the upper right of the **Select Net** dialog box. If you are selecting nets for an ATC2 core, only the **Clock and Data** input port categories appear at the upper right of the **Select Net** dialog box. If multiple trigger or data ports exist, there are 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 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 Signals, Trigger Signals* (trigger port tab if applicable), *Data Signals* (or *Trigger/Data Signals*, 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 Nets Up** and **Move Nets Down** buttons to reorder the position of any selected connection. After the desired net connections have been made, click **OK** to return to the main ChipScope Pro Core Inserter window.

All the trigger and data nets must be chosen in this fashion. After you have chosen all the nets for a given bus, the ILA or ATC2 bus name changes from red to black.

After specifying the clock, trigger, and data nets, click **Insert**.

If you are using the ChipScope Pro 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 ChipScope Pro 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 ChipScope Pro 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 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 and clicking the **New ILA 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 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

Select the **Insert > Insert Core** menu option to invoke the core insertion step.

***Note:*** If you are using the ChipScope Pro Core Inserter flow in the ISE Project Navigator tool, click **Return** to Project Navigator instead of selecting the **Insert > Insert Core** option. The insertion of the cores happens automatically as part of the Translate process in the Project Navigator tool. Refer to "Using the ChipScope Pro Core Inserter with ISE Project Navigator," page 33 for details.

## Managing Project Preferences

The preference settings are organized into three categories: Tools, ISE Integration, and Miscellaneous.

The Tools section contains settings for the command line arguments used by the ChipScope Pro Core Inserter to launch the EDIF2NGD tool.

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

The Miscellaneous preferences section contains other settings that affect how the ChipScope Pro Core Inserter operates. For example, you can set up the ChipScope Pro Core Inserter to display the ports in the Select Net dialog box. This might 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 ChipScope Pro 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.

Also, you can set up the ChipScope Pro Core Inserter 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 ChipScope Pro Core Inserter project preferences to the installation defaults by clicking the **Reset** button.

# Using the ChipScope Pro Analyzer

## ChipScope Pro Analyzer Overview

The ChipScope™ Pro Analyzer tool interfaces directly to the ICON (integrated controller), ILA (integrated logic analyzer), VIO (virtual input/output), and IBERT (internal bit error ratio) cores, collectively called the ChipScope Pro logic analyzer cores.

***Note:*** Even though the ChipScope Pro Analyzer tool detects the presence of an ATC2 (Agilent trace core), an Agilent Logic Analyzer attached to a JTAG (Joint Test Action Group, IEEE standard) cable is required to control and communicate with the ATC2 core.

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.

The ChipScope Pro Analyzer tool is made up of two distinct applications: the *server* and the *client*. The ChipScope Pro Analyzer server is a command line application that connects to the JTAG chain of the target system using any of the supported JTAG download cables shown in Table 1-11, page 29. The ChipScope Pro Analyzer client is a graphical user interface (GUI) application that allows you to interact with the devices in the JTAG chain and the cores that are found in those devices.

The ChipScope Pro Analyzer server and client can be running on the same machine (local host mode) or on different machines (remote mode). Remote mode is useful when you must:

- Debug a system that is in a different location
- Share a single system resource with other team members
- Demonstrate a problem or feature to someone who is not at your location

## ChipScope Pro Analyzer Server Interface

If you desire to debug a target system that is connected directly to your local machine via a JTAG download cable, then you do not need to start the server manually. You must only start the server application manually when you desire to interact with the server from a remote client.

***Note:*** The ChipScope Pro Analyzer server application can handle only one client connection at a time.

The server can be started as follows:

- The ChipScope Pro Analyzer server is started on 32-bit Windows machines by executing

  ```
  <XILINX_ISE_INSTALL>\bin\nt\cse_server.exe <command line options>
  ```

- The ChipScope Pro Analyzer server is started on 64-bit Windows machines by executing

  `<XILINX_ISE_INSTALL>\bin\nt64\cse_server.exe <command line options>`
- The ChipScope Pro Analyzer server is started on 32-bit Linux machines by executing

  `<XILINX_ISE_INSTALL>/bin/lin/cse_server <command line options>`
- The ChipScope Pro Analyzer server is started on 64-bit Linux machines by executing

  `<XILINX_ISE_INSTALL>/bin/lin64/cse_server <command line options>`

Note that *<XILINX_ISE_INSTALL>* is the location at which the Xilinx® ISE® Design Suite tools are installed. The ChipScope Pro Analyzer server application has several *<command line options>* that are described in Table 4-1. You can customize the server scripts as needed.

*Table 4-1:* **ChipScope Pro Analyzer Server Command Line Options**

| Command Line Option | Description |
|---|---|
| -port <*portnumber*> | Used to specify the TCP/IP port number that is used by the client and server to establish a connection. The default port number is 50001. |
| -password <*password*> | Used to protect the server from unauthorized access. No password is set by default. |
| -l <*logfile*> | Used to specify the location of the log file. The default log file location is: `$HOME/.chipscope/cs_analyzer_<portnumber>.log` where `$HOME` is your home directory and *<portnumber>* is the TCP/IP port number used by the server. |

See "Setting up a Server Host Connection," page 60 for more information on how to connect to the server application from the ChipScope Pro Analyzer client application.

# ChipScope Pro Analyzer Client Interface

The ChipScope Pro Analyzer client 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 cores in the devices in the chain. Although all devices in the chain are displayed in the tree, only valid target devices and cores can be operated upon. Leaf nodes in the tree appear when further operations are available. For instance, a leaf node for each unit appears when that device is configured with a core-enabled bitstream. Context-sensitive menus are available for each level of hierarchy in the tree. To access the context-sensitive menu, right-click 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 the node and select **Rename**. To end the editing, press **Enter** or the up or down arrow key, or click another node in the tree.

## Signal Browser

The signal browser displays all the signals for the ILA or VIO 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 it, or right-click and select **Rename**. To end the editing, press **Enter** or the up or down arrow key, or click another node in the tree.

### Adding/Removing Signals from Views

To remove all the signals from either the waveform or listing view, right-click 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 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 the first signal, hold down the **Shift** key, and click the last signal in the group. To select a non-contiguous group of signals and buses, click 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 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 is the LSB after the bus is created. If the **Ctrl** key is used, the order of signals in the bus matches the order in which the signals are clicked, the first signal being the LSB.

After you have selected the signals, right-click any selected signal and select **Move to Bus > New Bus**. A new bus is created at the top of the **Data Signals and Buses** sub-tree (in the case of ILA 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 **Move to Bus**, and then the bus name in the following submenu. Added signals always go on the MSB-end of the bus. The **Move to Bus** operation removes the signal(s) from the list of individual signals when they are combined into the bus. The **Copy to Bus** operation is the same as **Move to Bus** except the signals remain as individual signals in the list.

## Reverse Bus Ordering

To reverse the order of the bits in a bus (that is, to make the LSB the MSB), right-click the bus and select **Reverse Bus Order**. The signal browser and all data views that contain that bus are 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 changes the bus radix in every data view in which it appears.

### Signed and Unsigned Decimal

It is possible to replace the value of a bus with either a signed or unsigned decimal value that is modified using the following equations:

*Bus Value = (<scale factor> * Data) + <offset>*

*Precision = <precision>*

Whether the selected bus radix is Signed Decimal or Unsigned Decimal, a dialog box appears, allowing you to specify three parameters: *<scale factor>*, *<offset>*, and *<precision>*.

- Scale Factor

  The first text field that precedes the **Data* text in the dialog box is a constant scale factor used to multiply the *Data* value. The default *<scale factor>* value is 10.

- Offset

  The second text field following the "+" text in the dialog box is the constant offset that is added to the scaled *Data* value.

- Precision

  The third text field in the dialog box is the *<precision>*, which specifies the number of decimal places after the decimal point. The default *<precision>* value is 0.

For example, if you wanted to display down to 10 decimal places a 16-bit bus that represents a sine wave which ranges from -0.5 to +1.5, you would set the three parameters as follows:

*<scale factor> = 3.0517578125E-5 (which is the same as 1/215)*

*<offset> = 0.5*

*<precision> = 10*

### 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. 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.

A default token can be used to set a default token value when no other VALUE matches are found. The @DEFAULT_TOKEN key can be used to set the default token name. The token name "HEX" is used if no @DEFAULT_TOKEN line is used. The comment character in a token file is "#". The first non-comment line of the token file must be "@FILE_VERSION=1.0.0".

***Note:*** The "=" sign is a reserved character and cannot be part of the TOKEN string.

Below is an example token file:

```
#File version
@FILE_VERSION=1.0.0

# Default token value
@DEFAULT_TOKEN=ERROR

# Explicit token values
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
```

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 token values (for example, the bus is 8 bits wide while the specified tokens are only 4 bits wide), the tokens are padded to equal the width of the bus, using zeroes in the most-significant bit positions.

## Deleting Buses

To delete a bus, right-click it and select **Delete Bus**. The bus is immediately deleted in every data view it is resident.

## Type and Activity Persistence (VIO only)

VIO signals have two additional properties: *Type* and *Activity Persistence*. See "VIO Bus/Signal Activity Persistence," page 77 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.

## ChipScope Pro Analyzer 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 ChipScope Pro 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 ChipScope Pro 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 ChipScope Pro Analyzer tool 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 ChipScope Pro Analyzer tool. To save the new project under a different name, select **File > Save Project**. The project file has a .cpj `extension`.

#### Saving Projects

To rename the current project, or to save a copy to another filename, select **File > Save Project As**, type the new name in the File name dialog box, and click **Save**.

### Printing Waveforms

One of the features of ChipScope Pro is the ability to print a captured data waveform by using the **File > Print** menu option. Selecting the **File > Print** menu option starts the Print Wizard.

The Print Wizard consists of three consecutive windows:

1. (1 of 3) is the Print options and settings window
2. (2 of 3) is the Print waveform printout preview navigator window
3. (3 of 3) is the Print confirmation window

## Print Wizard (1 of 3) Window

The first Print Wizard window is used to set up various waveform printing options. The following sections describe these waveform printing options in more detail.

### 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 are present in the waveform printout using one of three methods:

- *Current View*: Print waveform data for all the signals and buses in the current view of the waveform window
- *All*: Print waveform data for all 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:
  - The starting window number
  - The sample number within that starting window
  - The ending window number
  - The sample number within the ending window

The default prints waveform data using the Current View method.

### Print Signal Names

You can choose to print the signal names on each page or only on the first page. When **Show Cursor Values** is enabled, this value also affects the display of the X/O cursor values.

#### X/O Cursor Values

You can 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, they appear on each page or only on the first page, depending on the Print Signal Names setting.

#### Footer

You can enable or disable the inclusion of a footer at the bottom of each page by selecting the Show Footer checkbox. The footer contains useful information including the ChipScope Pro Analyzer project name, waveform settings, print settings, and page numbers.

#### Navigation Buttons

The buttons at the bottom of the Print Wizard (1 of 3) window are defined as follows:

- *Page Setup*: Opens the page setup window
- *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.

### Print Wizard (2 of 3) Window

The second Print Wizard window shows a preview of the waveform printout.

#### 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 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 such that it pushes other signals/buses to another page, the total print preview page count at the top changes accordingly.

### Print Wizard (3 of 3) Window

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. 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.

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. 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.

### Page Setup

The Page Setup window can be invoked either from the Print Wizard (1 of 3) window or by using the **File > Page Setup** menu option.

***Note:*** In the ChipScope Pro 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 ChipScope Pro Analyzer program, change your default system printer, and restart the ChipScope Pro Analyzer program.

## Importing Signal Names

At the start of a project, all the signals in every core have generic names. You can rename the signals individually as described in "Renaming Signals, Buses, and Triggers Ports," page 53 or import a file that contains all the names of all the signals in one or more cores. The CORE Generator™, ChipScope Pro Core Inserter, EDK Platform Studio, System Generator for DSP, the PlanAhead™ design suite, and the FPGA Editor tools can create such files. To import signal names from a file, select **File > Import**. A **Signal Import** dialog box appears.

To select the signal import file, choose **Select New File**. A file dialog box appears, allowing you to navigate and specify the signal import file. After you choose the file, the **Unit/Device** combo box is 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 displays device numbers for those devices containing only ChipScope Pro capture cores.

If the signal import file contains signal names for only one core, the combo box is populated with names of the individual cores matching the type specified in the signal import file.

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 is displayed. If you choose to proceed, the signal names are applied to the cores as applicable.

## Exporting Data

Captured data from an ILA core can be exported to a file, for future viewing or processing. To export data, select **File > Export**. The Export Signals dialog box appears.

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:

- Signals and buses for that particular core
- Signals and buses present in the waveform viewer for the core

- Signals and buses in the listing viewer for the core
- Signals and buses in the bus plot viewer for the core

To export the signals, click **Export**. A dialog box appears from which you can specify the target directory and filename.

## Closing and Exiting the ChipScope Pro Analyzer

To exit the ChipScope Pro Analyzer tool, select **File > Exit**. The current active project is automatically saved upon exit.

## Viewing Options

You can hide or display the split pane on the left of the ChipScope Pro Analyzer window and the Message pane at the bottom of the window. By default, both are displayed the first time the ChipScope Pro Analyzer tool 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**.

## Setting up a Server Host Connection

The ChipScope Pro Analyzer client GUI application requires a connection to the ChipScope Pro Analyzer server application that is running on either the local or a remote system. Select **JTAG Chain > Server Host Setting**. This pops up the server settings dialog.

For local mode operation, the **Server** setting must be set to **localhost:<port>** (where **<port>** can be set to any unused TCP/IP port number). The default **<port>** number is 50001. In local mode, the **Password** setting is not necessary.

***Note:*** In local mode, the server starts automatically.

For remote mode operation, the **Server** setting must be set to an IP address (or appropriate system name) and port number in the format of 192.168.0.1:50001 (or servername:50001). The **Password** setting must be set to the same password you used when the server was started on the remote system. In remote mode, the connection is not actually established until you open a connection to a JTAG download cable, as described in the subsequent sections of this document.

***Note:*** In remote mode, the server needs to be started manually, as described in "ChipScope Pro Analyzer Server Interface," page 51.

## Opening a Parallel Cable Connection

To open a connection to the parallel cable, make sure the cable is connected to one of the parallel ports on your computer. Select **JTAG Chain > Xilinx Parallel Cable**. This pops up the Parallel Cable Selection configuration dialog box. You can choose the Parallel Cable III, Parallel Cable IV, or have the ChipScope Pro Analyzer tool autodetect the 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 (default), 2.5 MHz, 1.25 MHz, or 625 kHz. Choose the speed that makes the most sense for the board under test. Type the printer port name in the Port selection box (usually the default LPT1 is correct) and click **OK**. If successful, the ChipScope Pro Analyzer tool queries the boundary scan chain to determine its composition (see "Setting Up the Boundary Scan (JTAG) Chain," page 63).

If the ChipScope Pro Analyzer tool 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 tool installation program to install the required device driver software.

## Opening a Platform Cable USB Connection

To open a connection to the USB cable, make sure the cable is connected to one of the USB ports on your computer. Selecting the **JTAG Chain > Xilinx Platform USB Cable** menu option pops up a dialog window.

### Selecting the USB Port

ChipScope Pro Analyzer tool automatically populates the Port combobox with all available Xilinx® Platform Cable USB JTAG cables that are attached to the server machine to which you are connected. You can choose the USB port from a selection of port enumerations in the range of USB2<n>, where <n> is an integer value 1 through 127. The default port setting is USB21. The USB port enumeration number is based on the order in which the Platform Cable USB download cables are plugged into USB ports of the system. For instance, the first Platform Cable USB download cable plugged into the system is assigned the port enumeration of USB21, the second cable is assigned USB22, and so on. For newer Platform Cable USB download cables, the unique electronic serial number (ESN) that is read from the cable is displayed.

***Note:*** The enumerations are not necessarily preserved when the system is power cycled. Also, there is currently no way to identify a particular Platform Cable USB other than by physically plugging the cables into the system in a particular order.

### Blinking the LED on the Cable

To make the LED on the selected cable blink, click the **Blink LED** button in the dialog box.

### Selecting the Cable Speed

You can choose the speed of the cable from any of the settings: 12 MHz, 6 MHz, 3 MHz (default), 1.5 MHz, or 750 kHz. Choose the speed that makes the most sense for the board under test.

## Opening a Digilent USB JTAG Cable

When opening a connection to a Digilent USB JTAG cable (for instance, the JTAG-SMT1 or JTAG-HS1 cables), first make sure the cable is connected to one of the USB ports on your computer. Then select the **JTAG Chain > Digilent USB JTAG Cable** menu option.

### Selecting the Cable Device

The ChipScope Pro Analyzer tool automatically populates the Device combobox with all available Digilent USB JTAG cables that are attached to the server machine to which you are connected. You can select the desired Digilent USB JTAG cable from this list.

### Selecting the Cable Speed

After you make a cable device selection, the Speed combobox is populated with the set of TCK speeds that are supported by the selected cable. You can choose a cable speed from this list.

***Note:*** The maximum TCK speed for the cable is often limited by the speed at which the board under test is able to run. This might be far less than the maximum speed of the cable itself.

## Using Multiple Platform Cable USB Connections

To use the ChipScope Pro Analyzer tool with multiple cables, you need three things:

- Multiple Xilinx JTAG cables connected to one machine
- Multiple instances of the cs_server application running on one machine, each one listening to a different port
- Multiple instances of ChipScope Pro Analyzer tool running on that same machine, or a different machine (via the remote server feature)

### Connecting Multiple Xilinx JTAG Cables to One Machine

To interact with multiple JTAG cables connected to the same machine, you must first be able to connect multiple Platform Cable USB, Parallel Cable III, or Parallel Cable IV cables to the machine. For Platform Cable USB cables, you might need to use one or more USB hubs depending on how many cables you need. For PC3/PC4, you may need one or more parallel port extender cards.

***Note:*** Currently, enumerations are not associated with a particular physical Platform Cable USB cable. This means that rebooting your machine might result in different associations between enumerations and physical cables. One work-around is to unplug all cables and re-plug them in the order you wish for them to be enumerated.

### Setting Up ChipScope Pro Analyzer tool to Use Multiple Instances of cs_server

Set up the ChipScope Pro Analyzer tool to use multiple cables first by starting multiple instances of the `cs_server.exe` Windows application or `cs_server.sh` Linux application on the same machine using different ports. For example, to start up two servers on different ports on Linux, use:

```
# cs_server.sh -port 50001
# cs_server.sh -port 50002
```

### Starting and Configuring Multiple Instances of ChipScope Pro Analyzer

Start and configure multiple ChipScope Pro Analyzer client instances (see Table 4-2). Each instance of the ChipScope Pro Analyzer tool connects to a different cs_server and cable enumeration.

*Table 4-2:* **Configuration of Multiple Client Instances**

| Analyzer Instance # | Server Host Setting | Platform Cable USB Port # |
|---|---|---|
| 1 | <IP Address>:50001 | USB21 |
| 2 | <IP Address>:50002 | USB22 |

## Opening a JTAG Chain Plug-in Connection

The ChipScope Pro Analyzer tool supports select JTAG chain plug-in connections that can be opened using the **JTAG Chain > Open Plugin** menu option. Each JTAG chain plug-in has specific parameters that can be entered into the Plug-in Parameters field of the Open

Plug-in dialog box (consult with the plug-in provider for details). After the plug-in parameters are entered, click **OK** to open a connection to the JTAG plug-in.

## Polling the Auto Core Status

When the cores are armed, the interface cable queries 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 tool, it can often 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 ChipScope Pro Analyzer tool does not query the cores automatically to determine the status.

Note that this does not completely disable communication with the cable; it only disables the periodic polling when cores are armed. If one or more cores trigger after the polling has been turned off, the capture buffer is not 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 tool with one or more valid target devices. The first step is to set up all the devices in the boundary scan chain.

### Setting Up the Boundary Scan (JTAG) Chain

After the ChipScope Pro Analyzer tool has successfully communicated with a download cable, it automatically queries the boundary scan (JTAG) chain to find its composition. All Xilinx FPGA, CPLD, PROM, 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 cores. This information can be found in the BSDL file for the device. USERCODEs can be read out of the ChipScope Pro target FPGA devices by selecting **Read USERCODEs**.

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 tool is used in conjunction with other JTAG controllers (such as the System ACE 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 tool. In this case, the ChipScope Pro Analyzer tool must 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. Use the **Start transactions in Test-Logic/Reset, End in Run-Test/Idle** selection if the JTAG chain is shared with other JTAG controllers.

### Device Configuration

The ChipScope Pro Analyzer tool can configure target FPGA devices using the following download cables in JTAG mode only: Platform Cable USB, Parallel Cable III, Parallel Cable IV.

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. Alternatively, you can right-click the device in the project tree to get the same menu as **Device**.

After selecting the configuration mode, the JTAG Configuration dialog box opens. This dialog box has two sections: the first for selection of the JTAG device configuration file and the second for selection of a design-level CDC file. To select the configuration file to download into the device, click **Select New File** in the JTAG Configuration section. The Open Configuration File dialog box opens. Using the browser, select the device configuration file you want to use to configure the target device. After you locate and select the proper BIT file, click **Open** to return to the JTAG Configuration dialog box. You can also clear your existing project settings before configuring the device by selecting the **Clean previous project setting** check box.

***Note:*** Selecting the Clean previous project setting check box removes all signal names, buses, and other settings from your project. This operation cannot be undone.

To select a design-level CDC file that was generated either by the ChipScope Pro Core Inserter or PlanAhead tools, click the **Import Design-level CDC File** check box, then click **Select New File** in the Design-level CDC File section. The Open CDC File dialog box opens. Using the browser, select the CDC file you want to use with the target device. After you locate and select the proper CDC file, click **Open** to return to the JTAG Configuration dialog box. You can also automatically create buses from the signal names found in the CDC file by selecting the **Auto-create buses** check box. The ChipScope Pro Analyzer tool uses an algorithm to recognize bus elements where the element number is found at the end of the signal name and is surrounded by (), [], or {} or is separated from the base name by an underscore.

***Note:*** The CDC file is automatically selected if a design-level CDC file is found in the same directory as the configuration BIT file.

After the BIT and optional CDC files have been selected, click **OK** to configure the device.

### Observing the Configuration Progress

While the device is being configured, the status of the configuration is displayed at the bottom of the ChipScope Pro 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 various leaf nodes appear under each ChipScope Pro core unit.

### 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. 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 Messages pane. The IDCODE and USERCODE are also displayed in the JTAG Chain Setup dialog box (**JTAG Chain > JTAG Chain Setup)**.

### 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 pane.

***Note:*** All target devices contain two internal registers that contain status information: 1) the Configuration Status register (32 bits) and 2) 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, the implementation of that particular device determines whether any status information is present. Refer to the configuration documentation for the particular FPGA device for information on the definition of each specific configuration status bit.

For some devices, the JTAG Instruction register also contains status information. Use **Device** > **Show JTAG Instruction Register** to display this information in the messages pane for any device in the JTAG chain.

## Trigger Setup Window

To set up the trigger for an ILA core, select **Window > New Unit Windows** and the core desired. A dialog box is displayed for that core, and you can select the **Trigger Setup**, **Waveform**, **Listing**, **Bus Plot** and/or **Console** window(s) in any combination. Windows cannot be closed from this dialog box.

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 ILA core has its own Trigger Setup window which provides a graphical interface for you to set up triggers. The trigger mechanism inside each core can be modified at run-time without having to re-compile the design. The following sections describe how to modify the three components of the trigger mechanism:

- *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 the desired tab/button at the bottom of the window. To collapse, click the tab/button to the left of the expanded section you wish to collapse.

### Capture Settings

The capture settings section of the Trigger Setup window defines the number of windows, and where the trigger event occurs in each window. 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.

#### 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 has as many windows as possible with the defined samples per trigger. The trigger is always 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 core.

### 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 after the trigger condition occurs. Valid values are any positive integer from 1 to the depth of the capture buffer. The trigger mark always appears as sample 0 in the window. As many sample windows as possible are 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 core.

### 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 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 whether or not that match unit is individually negated (Boolean NOT) in the trigger condition.

The storage qualification condition can be configured to capture all data, or it can be set up to capture data that satisfies a Boolean AND or OR combination of all the enabled match units. 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.

## 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. One or more match functions can be defined in a Boolean equation or sequence in the Trigger Conditions section to specify the overall trigger condition of the core.

### Match Unit

The Match Unit field indicates which match unit the function applies to. You can expand the match field so it displays as individual trigger port bits in a 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 a specific match unit. It is displayed according to the **Radix** field. Double-clicking on the field makes it editable. Place the cursor before the value you want to change, and type a valid trigger character to overwrite that character. Or, select the field by single-clicking, and 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. The "?" character indicates that the nibble consists of a mixture of 1s, 0s, Xs, Rs, Fs, and Bs (where appropriate)
- *Octal:* X, ?, 0-7
- Binary: X (don't care), 0, 1, R (rising), F (falling), B (either transition), and N (no transitions). R, F, B, and N 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)

When **Bin** is chosen as the radix, positioning the mouse pointer over a specific character displays 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

If the match counter is present for a particular match unit, the text in the Counter column is black. If the counter is not present in the core, the text in that column is grayed out. To change the value of the match counter, click the counter cell to bring up the match unit counter dialog box.

The Counter field selects the number of match function events that must occur for the function to be satisfied.

- If occurring in exactly $n$ clock cycles is selected, then $n$ contiguous or $n$ noncontiguous events satisfies the match function counter condition.

- If occurring in at least *n* clock cycles is selected, then *n* contiguous or *n* noncontiguous events satisfies the match function counter condition, and it remains satisfied until the overall trigger condition is met.
- If occurring for at least *n* consecutive cycles is selected, then *n* contiguous events satisfies the match function counter condition, it remains 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 core.

## Trigger Conditions

A *trigger condition* is a Boolean equation or sequence of one or more match functions. The core captures 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.

### 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 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 box has two tabs: Boolean and Sequencer. When the Boolean tab is active, the trigger condition is 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 whether or not that match unit is 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 above the match unit table. The overall equation can also be negated, selectable using the Negate Whole Equation checkbox above the table. The resulting equation appears in the Trigger Condition Equation pane at the bottom of the window.

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.

A trigger sequence defined as *M0 => M1 => M3* 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 would like 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 you to select which type of signal is driven by the `trig_out` port of the ILA 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 trigger event.
- *Pulse (Low):* The output is a single clock cycle pulse of logic 0, 10 cycles after the trigger event.
- *Level (High):* The output transitions from a 0 to a 1, 10 cycles after the trigger event.
- *Level (Low):* The output transitions from a 1 to a 0, 10 cycles after the 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 S**ave Trigger Setup As File** dialog box opens, 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 opens, and you can navigate to the folder where the trigger settings file (with `a.ctj` extension) exists. After the trigger setting file is chosen, select **Open**, and those settings are 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. After 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 unit to ignore the trigger and storage qualification conditions and trigger immediately using a single sample window with the trigger position set to sample 0. 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 **Trigger Setup > Stop Acquisition**. Subsequent selections of **Trigger Setup > Run** cause the trigger to re-arm.

***Note:*** Stopping the data acquisition disarms the active trigger and any data captured up to that point in time is lost.

## Trigger Run Modes

The ChipScope Pro Analyzer tool supports three trigger run modes for the ILA core:

- Single
- Repetitive
- Startup

The trigger run mode is selected by using either the **Trigger Setup > Trigger Run Mode** menu option or the **Trigger Run Mode** toolbar button (enabled only when the Trigger Setup window is active).

### Single Trigger Run Mode

The single trigger run mode is used to arm the ILA core trigger and capture data as specified in the Trigger Setup window, upload and display the captured data, and stop. After stopping, only your interaction causes the trigger to become re-armed to repeat the process.

### Repetitive Trigger Run Mode

The repetitive trigger run mode is similar to the single trigger run mode, with one following exception:

In repetitive trigger run mode, instead of stopping after triggering and uploading/displaying captured data, the ChipScope Pro Analyzer tool automatically re-arms the ILA core trigger. The captured data from the previous trigger event continues to be displayed in the data viewers (waveform, listing, bus plot) as long as the subsequent trigger has not occurred. After the subsequent trigger occurs and the data capture buffer becomes full, the data viewers update to show this new data. This process repeats until the you manually stop the trigger.

#### Repetitive Trigger Logging

The captured data from each repetitive trigger run can be logged to a file. To do this, select **Trigger Setup > Setup Repetitive Trigger Logging** to open the setup dialog window. Use the **Browse** button to select the location for the log files. Each file created by the ChipScope Pro Analyzer tool includes a sequence number that corresponds to the repetitive trigger run iteration. Use the **Overwrite any existing files** check box to overwrite previous data log files.

The format for the data log files can be set to one of three available formats: value change dump (VCD) format, tab-delimited text format (ASCII), or the Agilent Technologies Fast Binary Data Format (FBDF). To select a format, click its radio button.

Use the **Signals to Export** combo box to select the set of signals and/or buses to export. The signals that can be exported include:

- Signals and buses for that particular core
- Signals and buses present in the waveform viewer for the core
- Signals and buses in the listing viewer for the core
- Signals and buses in the bus plot viewer for the core

Logging or exporting of signals during Single Trigger Run Mode is described in the Exporting Data section of this user guide.

### Startup Trigger Run Mode

The startup trigger run mode allows you to set up the ILA core to trigger on events that occur after FPGA device startup without having to use the ChipScope Pro Analyzer tool to arm the ILA core. Using the Startup Trigger Run Mode requires you to follow three steps:

1. Specify the trigger settings and save the startup trigger setup (CTJ) and design constraint (UCF) files.
2. Re-implement your design with the startup trigger setup design constraint file.
3. Use the ChipScope Pro Analyzer in Startup Trigger Run Mode to configure the device then upload and display the data that was captured after the trigger occurred.

To specify the trigger settings for use with trigger run mode, you must first set the Trigger Run Mode to **Single**. You then set up the match functions, trigger condition, and capture setup to your desired settings. After you have your trigger settings the way you want them, you use the **Trigger Setup > Save Trigger Startup Files** menu option to specify the required files:

- The NGD design file that corresponds to the design that is currently running in the FPGA device under test.

If using Project Navigator, the NGD design file is typically found in the same directory as the BIT file you used to configure the FPGA device under test.

If using the PlanAhead tool, the location of the NGD design file is typically found under your project directory location in the sub-directory `<project>.runs/<implementation run name>(where <project>` is called "project_1" by default, and `<implementation run name>` is called "impl_1" by default).

- CTJ file that contains your trigger settings.
- A UCF file that contains ILA core initialization settings that correspond to your desired trigger settings.

***Note:*** If you use the PlanAhead tool to implement your design, make sure you do not save your CTJ or UCF in the implementation run directory because the PlanAhead tool deletes all files in this directory before re-implementing your design.

After you save your trigger startup files, the ChipScope Pro Analyzer tool automatically converts the CTJ trigger settings into UCF design constraints that must be applied to your design. This requires you to add this new UCF file to your project and re-implement your design using either Project Navigator or the PlanAhead tool.

***Note:*** The startup trigger UCF file must be added to your project in addition to your original UCF design constraint files. Only one startup trigger UCF file per ILA core should be added to your design.

After you re-implement your design and create a new BIT file, configure the device with this BIT file using the ChipScope Pro Analyzer tool. To see if your startup trigger condition has occurred, first change the Trigger Run Mode to **Startup**. The software prompts you to specify the startup trigger files that were used during the implementation process to create the BIT file. In **Startup Trigger Run Mode**, the **Apply Settings and Arm Trigger** and **Stop Acquisition** toolbar buttons are highlighted in yellow, and the **Trigger Immediate** button is disabled. Also, the **Trigger Setup** window is disabled to prevent inadvertent changes to the trigger settings that do not match those saved in the BIT file.

Click the **Run** toolbar button to check the status of the ILA core. If the ILA core has triggered, and the data capture buffer is full, the ChipScope Pro Analyzer tool uploads and displays the captured data. If the ILA core has not triggered or if it has not completely filled the data capture buffer, the ILA core status is displayed at the bottom of the trigger setup window.

***Note:*** The startup trigger mode is armed immediately only after the FPGA device emerges from the "startup" state. To re-arm the trigger, you would need to re-configure the device. To change the startup trigger settings, you must change the Trigger Run Mode to **Single** and repeat the steps described in this section.

## Trigger and Capture Status

The status of the trigger logic for each ILA core is shown in the status bar at the bottom of the Trigger Setup window corresponding to the core. The following information is shown:

- Trigger state such as “Waiting for trigger”, “Capture started”, or “Slow or stopped clock” is shown in the lower-left status bar.
- Capture buffer state indicating how many samples have been captured is also shown in the lower-left status bar.
- Trigger modes, such as **SINGLE RUN**, **REPETITIVE RUN**, and **IDLE** are shown in the lower-right status box.

An icon that changes to indicate the state of the trigger and capture logic for the ILA core is also provided.

- A spinning, partially filled pie icon indicates that the trigger and/or capture logic is active
- A solidly filled pie icon indicates the capture buffer is full and the trigger logic is idle.

The trigger status icon also appears in the title bar of the minimized trigger setup window. This is useful for monitoring the trigger status of multiple ILA cores.

Detailed trigger and capture status information also appears in the Messages pane (or console window).

## Waveform Window

To view the waveform for a particular ILA core, select **Window > New Unit Windows**, and the core desired. A dialog box appears for that ChipScope Pro core unit, and you 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, and renaming. To perform a signal operation, right-click 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 red line shows the potential drop location.

***Note:*** Signals can be moved within a bus by first selecting one or more signals within the bus, then by using the Alt-UpArrow and Alt-DownArrow keystroke combinations.

### 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 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. After the drag is complete, a popup appears. Select **Zoom Area** to perform the zoom.

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**.

### Centering the Waveform

To center the waveform display around a specific point, select **Waveform > Go To**, and center the waveform display around the *X* and *O* markers as well as the previous or next trigger position.

### 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 appears, indicating the position of the cursor. Additionally, the status of all the signals and buses at that point is 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 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. Select **Ruler > Negative Time/Samples** in the right-click menu to toggle between a negative and/or positive range of samples.

### 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**.

### Data Capture Time Stamp

A time stamp indicating the date and time that the captured data was uploaded and displayed is shown in the lower-left corner of the Waveform window. If repetitive trigger run mode is enabled, a sequence number is also shown indicating how many times the ILA core has triggered and uploaded data successfully.

## Listing Window

To view the Listing window for a particular ILA core, select **Window > New Unit Windows**, and the core desired. A dialog box displays for that ChipScope Pro Unit, in which you can select any combination of **Trigger Setup**, **Waveform**, **Listing**, and/or **Bus Plot** windows. 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. 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 a signal or bus in the column heading.

### Bus and Signal Reordering

Buses and signals can be reordered in the Listing window. Click 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 column and selecting **Remove**. If you select **Remove All** , all signals and buses are 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 color of the line in the table is the same as the color of the cursor. To move the cursor to a different position in the table, right-click in the new location and repeat operation as before, or right-click the cursor handle in the first column, and drag it to the new location.

### Go to 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 window for a particular set of ILA buses, select **Window > New Unit Windows** and the core desired. A dialog box displays for that ChipScope Pro Unit, on which you can select any combination of 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. The Bus Plot window displays buses as a graph of the values for a bus over time, or the values of one bus as opposed to another's.

### 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, any number of buses can be displayed at once. When **data vs. data** is chosen, two buses must be selected and:

- Each point in the *x* coordinate of the plot equals the value of one of the buses at a particular time.
- The *y* coordinate equals the value of the other bus at the same time.

Each bus is displayed in a unique color and according to its radix. (Hexadecimal, binary, octal, token and ASCII radices are displayed as unsigned decimal values with scale factor = 1.0, precision = 0.)

### 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.

### 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 is displayed for that ChipScope Pro Unit, and you can select the **Console** window. (Windows cannot be closed from this dialog box.)

The Console window is for VIO cores only. To open the Console for a particular VIO core, double-click the **VIO Console** leaf node in the project tree. Here, you can see the status and activity of the VIO core input signals and modify the status of the VIO core output signals.

In the VIO Console window, you can perform all signal browser operations, such as bus creation, radix selection, and renaming, by right-clicking on a signal or bus in the column heading. The VIO Console window has a table with two columns: Bus/Signal and Value.

### 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 type determines how a 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 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 display type: Text
- VIO output signals have the following control types:

  - Text: ASCII text field
  - Push button (either active-High or active-Low)
  - Toggle button
  - Pulse train (synchronous outputs only)
  - Single pulse (synchronous outputs only)
- VIO output buses have two valid control 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 77 for a description of signal activity).

If the persistence is:

- *Infinite:* The activity is displayed in the column forever.
- *Long:* The activity is displayed in the column for 80 times the sample period
- *Short:* The activity is displayed in the column for 8 times the sample period

When the time limit on the persistence expires, a new activity is displayed. If no activity occurred in the last sample cycle, no activity is displayed in the Value column.

### Bus and Signal Reordering

Buses and signals can be reordered in the Waveform window. Click 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. Either right-click a signal or bus and select the operation desired, or use the standard Windows key combinations (**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 signal in the console. 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 according to the selection in the JTAG Scan Rate toolbar options. Each of the VIO core inputs captures, along with the current value of the signal, activity information about the signal because 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 appears 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 is 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.

#### 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 is 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 that you define. To edit the pulse train, click **Edit**. This brings up the Pulse Train dialog box. 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.

#### 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 is executed exactly once.

### VIO Core Menu and Toolbar Controls

When the VIO console is in focus, the VIO core-specific menu and toolbar controls can be used to change the behavior of the VIO core inputs or outputs, as applicable. The toolbar controls are described from left-to-right in the following sections.

#### JTAG Scan Rate

The **JTAG Scan Rate** at which the VIO core inputs are read is selectable via a combo box. The default scan rate is 250 ms. You can also set the sample period to 500 ms, 1 s, 2 s, 5 s or Manual Scan. When Manual Scan is chosen, the **Sample Once (S!)** button becomes enabled. At that point, the VIO core inputs are only read by pressing the **Sample Once** toolbar button or by selecting the **VIO > Sample Once** menu option.

#### Update Static Outputs

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 Outputs (U!)** toolbar button or select the **VIO > Update Static Outputs** menu option.

#### Reset All Outputs

To reset all outputs to their default state (0 for text fields and toggle buttons, all 0 pulse train for pulse trains) click the **Reset All Outputs** toolbar button or select the **VIO > Reset All Outputs** menu option.

#### Clear All Activity

At some point, it might be desirable to reset the activity display for all VIO core inputs. To do so, press the **Clear All Activity** toolbar button or select the **VIO > Clear All Activity** menu option. All input activity is reset, regardless of the selected persistence.

## System Monitor

Virtex®-5, Virtex-6, Virtex-7, and Kintex™-7 FPGA and Zynq™-7000 EPP devices include a System Monitor feature, also called Xilinx Analog-to-Digital Converter (XADC). When combined with a number of on-chip sensors, the ADC can measure FPGA physical operating parameters, including on-chip power supply voltages and die temperature. For more information refer to the following documents:

- *Virtex-5 FPGA System Monitor User Guide* [See Reference 22, p. 211]
- *Virtex-6 FPGA System Monitor User Guide* [See Reference 23, p. 211]
- *7 Series FPGAs XADC Dual 12-Bit 1MSPS Analog-to-Digital Converter User Guide* [See Reference 13, p. 211]

The ChipScope Pro Analyzer tool provides real-time JTAG access to the on-chip voltage and temperature sensors of the System Monitor primitive. All the on-chip sensors are available before and after the FPGA device has been configured with a valid bitstream. The System Monitor functionality does not require that you instantiate a System Monitor primitive block into your design. The only requirement is that the System Monitor-specific pins on the FPGA device are properly connected on the system board.

In the ChipScope Pro Analyzer project tree, each System Monitor-capable FPGA device in the JTAG chain has a System Monitor Console node. Right-clicking on the System Monitor node in the project tree shows an option for opening the System Monitor viewer. Left-clicking on the System Monitor node in the project tree shows the various sensors in the signal browser. In the signal (or sensor) browser, you can rename or change the display units of the various sensors.

### System Monitor Console

Each System Monitor sensor value can be displayed in a System Monitor Console history window or written to a log file. You can enable the following display values for each sensor:

- Current value that is read directly from the System Monitor sensor
- Device maximum and minimum values that are read directly from the System Monitor sensor peak detectors

- Sampled maximum and minimum values that are derived from all sensor values that have been collected by the ChipScope Pro Analyzer tool since opening a JTAG cable connection (or the last System Monitor reset)
- Windowed average, maximum, and minimum values that are calculated over a sliding window of sensor values that have been collected by the ChipScope Pro Analyzer tool since opening a JTAG cable connection (or the last System Monitor reset)

The *sampled* and *windowed* values that are calculated by the System Monitor viewer can be reset by clicking on the **Reset** button on the toolbar.

***Note:*** If the System Monitor is not reporting valid sensor data, the System Monitor Console displays an `Invalid Data` banner message across the window.

## System Monitor Console Toolbar

The System Monitor Console toolbar and right-click menu options provide a means to customize and interact with the System Monitor Console.

### JTAG Scan Rate

The **JTAG Scan Rate** at which the System Monitor sensor data is read is selectable via a combo box. The default scan rate is 1 s. You can also set the sample period to 1 s, 2 s, 5 s, 10 s, 30 s, 1 min, or Manual Scan. When Manual Scan is chosen, the **Sample Once (S!)** button becomes enabled. At that point, the System Monitor data is only read by pressing the **Sample Once** toolbar button or by selecting the **System Monitor > Sample Once** menu option.

### Window Depth

The depth of the window used in the sliding window calculations in the System Monitor viewer can be set using the **Window Depth** combo box on the toolbar or in the **System Monitor > Window Depth** menu. The depth of the sampling window can be set to 2, 4, 8, 16, 32, 64, or 128 samples.

### External Input

The System Monitor component monitors voltage levels on external sensors. You can view any external sensor input one at a time by using the **External Input** option. Valid External Input selections include:

- Any of the 16 user-defined VAUXP/VAUXN external sensors
- The V_P/V_N dedicated external sensors
- The V_REFP reference voltage input
- The V_REFN reference voltage input

Select **No Input** to disable the viewing of the external input (default).

***Note:*** The XADC block of 7 Series FPGAs contains an additional on-chip Block Memory voltage sensor (VCCBRAM) that is selectable in the External Input selection.

### Reset

The **Reset** button resets the System Monitor Console display.

#### Enable Logging

The **Enable Logging** toolbar button and **System Monitor > Enable Logging** menu option enables the file logging feature that saves the System Monitor sensor data in a text file for use in offline analysis.

### System Monitor Data Logging

The **System Monitor > Setup Logging** menu option opens the dialog window. The settings in this window are used to customize the *logging* feature.

#### Log File

The **Browse** button is used to select the location of the System Monitor log file. The default location is **<*CHIPSCOPE_INSTALL*>/bin/<*PLATFORM*>/system_monitor.log**, where **<*CHIPSCOPE_INSTALL*>** is the installation directory and **<*PLATFORM*>** is the operating system platform (nt, nt64, lin, lin64, or sol).

#### Log File Format

The System Monitor log file is a text file that can be formatted in two different ways: comma-separated value (CSV) file for machine processing or as a human-readable formatted file.

#### Log File Limit

The System Monitor logging system can generate a lot of data that consumes a large amount of disk space. To alleviate the problem, the log data can be split across multiple separate files based on a log file limit. The log file limit can be based on a specific number of samples or by file size (in kilobytes).

## IBERT v2.0 Console Window for Virtex-5 FPGA GTX Transceivers

To open the console for a ChipScope Pro IBERT v2.0 core for Virtex-5 FPGA GTX transceivers, select **Window> New Unit Windows** and the core desired. A dialog box displays for that core, and you can select the **IBERT V5 GTX Console**. Windows cannot be closed from this dialog box.

The same operation can by achieved by double-clicking on the **IBERT V5 GTX Console** leaf node in the project tree, or by right-clicking on the **IBERT V5 GTX Console** leaf node and selecting **Open IBERT V5 GTX Console**.

The IBERT v2.0 Console Window for Virtex-5 FPGA GTX transceivers is composed of:

- MGT/BERT Settings Panel
- DRP Settings Panel
- Port Settings Panel
- Sweep Test Settings Panel
- IBERT v2.0 Virtex-5 FPGA GTX Transceiver Toolbar and Menu Options

### MGT/BERT Settings Panel

The **MGT/BERT Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific control or status setting.

### MGT Settings

The **MGT Alias** setting is initially set to the MGT number of the GTX transceiver, but can be changed by selecting the field and typing in a new value.

The **Tile Location** setting denotes the X/Y coordinate of the GTX transceiver in the device.

The **MGT Link Status** indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTX transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NOT LOCKED (red).

The **PLL Status** indicator shows the lock status of the PLL that is connected to the GTX transceiver. The valid states of this status indicator are LOCKED (green) or NOT LOCKED (red).

The **Loopback Mode** setting is used to control the loopback mode of a particular GTX transceiver channel. The valid choices for loopback mode are:

- None: No feedback path is used.
- Near-End PCS: The circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTX channel.
- Near-End PMA: The circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, through the PMA, back through the PCS, and returns to the RX fabric interface.
- Far-End PMA: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the part of the GTX transceiver channel. For this GTX transceiver loopback mode, the signal comes into the RX pins, passes through the PMA circuitry, and returns immediately to the TX pins.
- Far-End PCS: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through part of the GTX transceiver channel. For this GTX loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.

The **DUAL Reset** button resets both GTX transceivers in the DUAL by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The **Channel Reset** button resets the GTX transceiver channel by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The **TX Polarity Invert** setting controls the polarity of the data sent out of the TX pins of the GTX transceiver channel. To change the polarity of the TX side of the GTX transceiver, check the TX Polarity Invert box.

The **TX Bit Error Inject** button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The **TX Diff Output Swing** combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The **TX Pre-Emphasis** combo box controls the amount of pre-emphasis on the transmitted signal. Change the value in the combo box to change the emphasis.

The **RX Polarity Invert** setting controls the polarity of the data received by the RX pins of the GTX channel. To change the polarity of the RX side of the GTX transceiver, check the RX Polarity Invert box.

The **RX AC Coupling Enabled** setting controls whether the built-in AC coupling capacitors are enabled or not.

The **RX Termination Voltage** setting controls which supply is used in the RX termination network.

The **RX Equalization** setting controls the internal RX equalization circuit.

### BERT Settings

The **TX Data Pattern** and **RX Data Pattern** settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The **RX Bit Error Ratio** field contains the currently calculated bit error ratio for the GTX transceiver channel. It is expressed as an exponent. For instance, 1.000E-12 means that one bit error happens (on average) for every trillion bits received.

The **RX Received Bit Count** field contains a running tally of the number of bits received. This count resets when the BERT Reset button is clicked.

The **RX Bit Error Count** field contains a running tally of the number of bit errors detected. This count resets when the BERT Reset button is clicked.

The **BERT Reset** button resets the bit error and received bit counters. It is appropriate to reset the BERT counters after the GTX channel is linked and stable.

### Clocking Settings

The **TX DCM Reset** button resets the DCM that uses the TXOUTCLK output to generate the TXUSRCLK and TXUSRCLK2 clocks.

The **RX DCM Reset** button resets the DCM that controls uses the RXRECCLK output to generate the RXUSRCLK and RXUSRCLK2 clocks.

The **TXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The T**XUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK2 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK2 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

## DRP Settings Panel

The **DRP Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name,** all the DRP attributes are displayed in alphabetical order. The **Radix** combo lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

When the radio button at the bottom of the panel is set to **View By Address**, the raw address are displayed in numerical order with their contents. The **Radix** combo lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

## Port Settings Panel

The **Port Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific MGT port. Not all ports are displayed because some are used in the IBERT design to send and receive data.

The **Radix** combo lets you choose to display the value in either Hex (hexadecimal) or Bin (binary). Some ports are read-only, and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text-fields, and placing the cursor in those fields, typing a new value, and pressing **Enter** write the new value to the MGT immediately.

## Sweep Test Settings Panel

The Sweep Test Settings panel is used to set up a channel test that sweeps through various transceiver settings. The TX and RX settings are for the same GTX transceiver. Sweeping through both TX and RX settings only work if the transceiver is set to one of the near end or external loopback modes. Sweeping through RX parameters only can be performed when the corresponding TX endpoint for the link resides in a different device or a different transceiver in the same device.

The Sweep Test Settings tabbed panel consists of four sections: the Parameter Settings. section, the Sampling Point Region Section, the Test Controls section, and the Test Results Section.

### Parameter Settings

Setting up the sweep test first involves setting up the sweep parameters.

The GTX transceiver parameters that are available for sweep default to the following:

- TX Diff Swing
- TX Pre-Emphasis
- RX EQ
- DFETAP1
- DFETAP2

The sweep parameters can be initialized in one of two ways:

- Click the **Clear All Parameters** button to clear all parameters to the Select option.
- Click the **Set Parameters to Current Values** button to set the parameters to their current values on the MGT/BERT Settings panel.

The order of the parameters in the Sweep Parameter table dictates how the parameters are swept. The values of the parameters near the top of the table are swept less frequently than the parameters near the bottom of the table. In other words, the parameters near the top of the table are in the outer loops of the sweep algorithm while the parameters near the bottom of the table are in the inner loops of the sweep algorithm.

Each parameter must be set up with a start and end value. The order of the parameter values cannot be changed. After you select the start value, the end values available for selection change automatically to include only valid selections. If you do not want to sweep through a parameter, set the start and end values for that parameter to the same value. The Sweep Value Count column indicates how many values are swept through for a particular parameter. After all sweep parameters have valid start and end values, the total number of sweep iterations are shown in the Total Iterations field.

To add or remove items in the table, click the **Add/Remove Parameters** button. This opens a dialog box with all available ports/attributes on the left, and the ones to sweep on the right. To add new parameters to sweep, click it in the left hand list, and click the > arrow button. To remove a parameter to sweep, click it in the right-hand list, and click the < arrow button. To specify the sweep order of the parameters, click it in the right hand list, and then the Up or Down buttons. To revert the sweep attributes and their order to the default setting, click the Reset to Default button. Click OK to apply the settings, or Cancel to exit without saving.

### Sampling Point Region

The sampling point is the horizontal point within the eye to sample. Visualize on transition region at the far left, and the next at the far right. The RX Sampling Point is one of 128 discrete sampling positions. In the Sampling Point Region section, choose the left and right edges of the sampling region.

### Test Controls

After the sweep test has been set up, the test can be started by clicking the **Start** button. After the **Start** button is clicked, the sweep parameter table are disabled, and the test starts running.

As the sweep test runs, the current sweep result file, current iteration, elapsed time, and estimated time remaining status indicators are displayed. The sweep results are shown in the text area near the bottom of the screen. The sweep test can be paused by clicking on the **Pause** button or stopped completely by clicking on the **Reset** button.

The results from a sweep test are displayed in the Test Results panel and are also sent to a sweep test result file. Clicking on the **Log File Settings** button brings up the dialog window.

You can set the location of the file and the number of sweep iterations stored in each file. If the total number of sweep iterations exceeds the file limit, multiple files with the starting iteration number appended to the base file name are created in the same directory as the initial result file.

### Test Results

The Test Results panel shows which is the current run, the elapsed time, and the estimated time remaining. Below this status information is a running log of the sweep test results. These results are also saved to the Log File.

### IBERT v2.0 Virtex-5 FPGA GTX Transceiver Toolbar and Menu Options

#### IBERT Console Options

The IBERT Console Options dialog allows you to select which columns and rows to display in the IBERT Console. The left-hand panel selects the MGTs by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the MGTs.

The right-hand panel selects which rows are displayed in the MGT/BERT settings. Use the **Check All** and **Uncheck All** buttons to select all or none of the rows. The **Default** button sets up the console to display the basic set of rows needed to determine the health of the channels.

#### Import/Export Dialog

In the Import/Export dialog box, you can save and recall settings from a specific MGT, or apply the setting of one MGT to others in the design. To import or export settings, use **IBERT_V5GTX > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the MGT settings- either MGT or File. If MGT is selected, choose among the available MGTs in the combo box. For File, click Browse and navigate to the settings file. Click **Next** to go to the next screen.

The second screen is the destination screen. Any combination of the MGTs in the IBERT design and a file are available. If File is enabled, click Browse to specify the file destination.

The third screen is the confirmation screen, summarizing the source and destination(s) for the settings. Click Apply to execute the import or export. This operation cannot be undone.

#### Reset All

To reset all the channels in the IBERT core, use **IBERT_V5GTX >Reset All** or click the **Reset All** button in the toolbar.

#### JTAG Scan Rate and Scan Now

The JTAG Scan Rate toolbar and **IBERT_V5GTX >JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_V5GTX >Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

## IBERT Console Window for Virtex-6 FPGA GTX Transceivers

To open the console for a ChipScope Pro IBERT core for Virtex-6 LXT/SXT/CXT families, select **Window > New Unit Windows** and the core desired. A dialog box displays for that core, and you can select the **IBERT V6 GTX Console**. Windows cannot be closed from this dialog box.

The same operation can by achieved by double-clicking on the **IBERT V6 GTX Console** leaf node in the project tree, or by right-clicking on the **IBERT V6 GTX Console** leaf node and selecting **Open IBERT V6 GTX Console**.

The IBERT Console for Virtex-6 LXT/SXT/CXT families GTX transceivers is composed of: "MGT/BERT Settings Panel", "DRP Settings Panel," page 89, "Port Settings Panel," page 89, and "IBERT Virtex-6 FPGA GTX Transceiver Toolbar and Menu Options," page 92.

## MGT/BERT Settings Panel

The **MGT/BERT Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific control or status setting.

### MGT Settings

The **MGT Alias** setting is initially set to the MGT number of the GTX transceiver, but can be changed by selecting the field and typing in a new value.

The **Tile Location** setting denotes the X/Y coordinate of the GTX transceiver in the device.

The **MGT Link Status** indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTX transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NOT LOCKED (red).

The **TX PLL Status** indicator shows the lock status of the TX PLL that is connected to the GTX transceiver. The valid states of this status indicator are LOCKED (green) or NOT LOCKED (red).

The **RX PLL Status** indicator shows the lock status of the RX PLL that is connected to the GTX transceiver. The valid states of this status indicator are LOCKED (green) or NOT LOCKED (red).

The **Loopback Mode** setting is used to control the loopback mode of a particular GTX transceiver channel. The valid choices for loopback mode are:

- None: No feedback path is used.
- Near-End PCS: The circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTX channel.
- Near-End PMA: The circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, through the PMA, back through the PCS, and returns to the RX fabric interface.
- Far-End PMA: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the part of the GTX transceiver channel. For this GTX transceiver loopback mode, the signal comes into the RX pins, passes through the PMA circuitry, and returns immediately to the TX pins.
- Far-End PCS: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through part of the GTX transceiver channel. For this GTX loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.
- Far-End Fabric: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the entire GTX transceiver channel and related fabric logic. For this GTX transceiver loopback mode, the signal comes into the RX pins, passes through the PMA and PCS, through a shallow fabric-based FIFO, back through the PCS and PMA, and finally returns to the TX pins.

The **Channel Reset** button resets the GTX transceiver channel by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The **TX Polarity Invert** setting controls the polarity of the data sent out of the TX pins of the GTX transceiver channel. To change the polarity of the TX side of the GTX transceiver, check the **TX Polarity Invert** box.

The **TX Bit Error Inject** button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The **TX Diff Output Swing** combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The **TX Pre-Emphasis** combo box controls the amount of pre-emphasis on the transmitted signal. Change the value in the combo to change the emphasis.

The **RX Polarity Invert** setting controls the polarity of the data received by the RX pins of the GTX channel. To change the polarity of the RX side of the GTX transceiver, check the **RX Polarity Invert** box.

The **RX AC Coupling Enabled** setting controls whether the built-in AC coupling capacitors are enabled or not.

The **RX Termination Voltage** setting controls which supply is used in the RX termination network.

The **RX Equalization setting** controls the internal RX equalization circuit.

### BERT Settings

The **TX/RX Data Pattern** settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The **RX Bit Error Ratio** field contains the currently calculated bit error ratio for the GTX transceiver channel. It is expressed as an exponent. For instance, 1.000E-12 means that one bit error happens (on average) for every trillion bits received.

The **RX Received Bit Count** field contains a running tally of the number of bits received. This count resets when the **BERT Reset** button is clicked.

The **RX Bit Error Count** field contains a running tally of the number of bit errors detected. This count resets when the **BERT Reset** button is clicked.

The **BERT Reset** button resets the bit error and received bit counters. It is appropriate to reset the BERT counters after the GTX channel is linked and stable.

### Clocking Settings

The **TXOUTCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXOUTCLK0 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **TXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **TXUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK2 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK2 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

## DRP Settings Panel

The **DRP Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name**, all the DRP attributes are displayed in alphabetical order. The **Radix** combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

When the radio button at the bottom of the panel is set to **View By Address**, the raw address are displayed in numerical order with their contents. The **Radix** combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

## Port Settings Panel

The **Port Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific MGT port. Not all ports are displayed because some are used in the IBERT design to send and receive data.

The **Radix** combo box lets you choose to display the value in either Hex (hexadecimal) or Bin (binary). Some ports are read-only, and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text-fields, and placing the cursor in those fields, typing a new value, and pressing **Enter** writes the new value to the MGT immediately.

## Sweep Test Settings Panel

The Sweep Test Settings panel is used to set up a channel test that sweeps through various transceiver settings. The TX and RX settings are for the same GTX transceiver. Sweeping through both TX and RX settings only works if the transceiver is set to one of the near-end or external loopback modes. Sweeping through RX parameters only can be performed when the corresponding TX endpoint for the link resides in a different device or a different transceiver in the same device.

The Sweep Test Settings tabbed panel consists of four sections: the Parameter Settings. section, the Sampling Point Region Section, the Test Controls section, and the Test Results Section.

### Parameter Settings

Setting up the sweep test first involves setting up the sweep parameters.

The GTX transceiver parameters that are available for sweep default to the following:

- TX Diff Swing
- TX Pre-Emphasis
- TX Post-Emphasis

- RX EQ

The sweep parameters can be initialized in one of two ways:

- Click the **Clear All Parameters** button to clear all parameters to the Select… option.
- Click the **Set Parameters to Current Values** button to set the parameters to their current values on the MGT/BERT Settings panel.

The order of the parameters in the Sweep Parameter table dictates how the parameters are swept. The values of the parameters near the top of the table are swept less frequently than the parameters near the bottom of the table. In other words, the parameters near the top of the table are in the outer loops of the sweep algorithm while the parameters near the bottom of the table are in the inner loops of the sweep algorithm.

Each parameter must be set up with a start and end value. The order of the parameter values cannot be changed. After you select the start value, the end values available for selection change automatically to include only valid selections. If you do not want to sweep through a parameter, set the start and end values for that parameter to the same value. The Sweep Value Count column indicates how many values are swept through for a particular parameter. After all sweep parameters have valid start and end values, the total number of sweep iterations are shown in the Total Iterations field.

To add or remove items in the table, click the **Add/Remove Parameters** button. This opens a dialog box with all available ports/attributes on the left, and the ones to sweep on the right. To add new parameters to sweep, click it in the left hand list, and click the **>** arrow button. To remove a parameter to sweep, click it in the right-hand list, and click the **<** arrow button. To specify the sweep order of the parameters, click it in the right hand list, and then the **Up** or **Down** buttons. To revert the sweep attributes and their order to the default setting, click the **Reset to Default** button. Click **OK** to apply the settings, or **Cancel** to exit without saving.

### Sampling Point Region

The sampling point is the horizontal point within the eye to sample. Visualize on transition region at the far left, and the next at the far right. The RX Sampling Point is one of 128 discrete sampling positions. In the Sampling Point Region section, choose the left and right edges of the sampling region.

### Test Controls

After the sweep test has been set up, the test can be started by clicking the **Start** button. After the **Start** button is clicked, the sweep parameter table are disabled, and the test starts running.

As the sweep test runs, the current sweep result file, current iteration, elapsed time, and estimated time remaining status indicators are displayed. The sweep results are shown in the text area near the bottom of the screen. The sweep test can be paused by clicking on the **Pause** button or stopped completely by clicking on the **Reset** button.

The results from a sweep test are displayed in the Test Results panel and are also sent to a sweep test result file. Clicking on the **Log File Settings** button brings up the dialog window.

You can set both the location of the file and the number of sweep iterations stored in each file. If the total number of sweep iterations exceeds the file limit, multiple files with starting iteration number appended to the base file name are created in the same directory as the initial result file.

### Test Results

The Test Results panel shows which is the current run, the elapsed time, and the estimated time remaining. Below this status information are three tabbed panels showing the results of the sweep test:

- Sweep Test Log
- Sweep Test Plots
- Sweep Test Info.

## Sweep Test Log

The Sweep Test Log tabbed panel is always enabled and contains the running log of the sweep test results. The information on this panel is shown using a text-only display. The results of the sweep test are also stored in a CSV log file.

### Sweep Test Plots

The Sweep Test Plots tabbed panel is enabled only after the sweep test has stopped running. The graphical data shown in the sweep test plot window corresponds to the data stored in the CSV log file. Each plot corresponds to a sweep across the unit interval (UI) of the received signal. The left and right edges of the data plot correspond to the settings in the Sample Point Region panel.

The main measurement that is observed in the sweep test plot is the width of the UI "opening" of the active plot at the lowest measured bit error ratio (BER). The horizontal marker can be moved up and down and the vertical markers can be moved left and right by clicking and dragging them. By right-clicking in the plot list area on the right side of the Sweep Test Plots panel, you can hide or show each plot, rename and re-color each plot, and set a plot as active.

### Sweep Test Info

The **Sweep Test Info** tabbed panel is also enabled only after the sweep test has stopped running. The tabular data shown in the **Sweep Test Info** panel corresponds to the data shown in the graphical **Sweep Test Plots** panel. Each row in the table correspond to a single sweep test plot. The table columns correspond to the following:

- Enable Plot: shows or hides the plot in the Sweep Test Plots panel. Check to show, uncheck to hide.
- Line Color: color of the line used in the Sweep Test Plots panel. Click to select new color.
- Plot Name: name of the plot. Click the text field to change plot name.
- Opening at the Lowest BER Level: width of the longest run of zero errors at the lowest BER level.
- The remaining columns correspond to the parameter value settings that were used to create the corresponding sweep test plot.

Click the column header to sort the rows in the sweep test panel.

***Note:*** One useful sort strategy is to sort the plots from the widest to the lowest UI opening by clicking the **Opening at the Lowest BER Level** column header.

### IBERT Virtex-6 FPGA GTX Transceiver Toolbar and Menu Options

#### IBERT Console Options

The IBERT Console Options dialog allows you to select which columns and rows to display in the IBERT Console. The left-hand panel selects the MGTs by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the MGTs.

The right-hand panel selects which rows are displayed in the MGT/BERT settings. Use the **Check All** and **Uncheck All** buttons to select all or none of the rows. The **Default** button sets up the Console to display the basic set of rows needed to determine the health of the channels. The Default rows are Tile Location, TX PLL Status, RX PLL Status, Loopback Mode, Channel Reset, TX Error Inject, Rx Sampling Point, TX Data Pattern, RX Data pattern, Rx Bit Error Ratio, Rx Received Bit Count, Rx Bit Error Count, BERT Reset, TXUSRCLK2 Freq, and RXUSRCLK2 Freq.

#### Import/Export Dialog

In the Import/Export dialog box, you can save and recall settings from a specific MGT, or apply the setting of one MGT to others in the design. To import or export settings, use **IBERT_V6GTX > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the MGT settings - either **MGT** or **File**. If **MGT** is selected, choose among the available MGTs in the combo box. For **File**, click **Browse** and navigate to the settings file. Click **Next** to go to the next screen.

The second screen is the destination screen. Any combination of the MGTs in the IBERT design and a file are available. If **File** is enabled, click **Browse** to specify the file destination.

The third screen is the confirmation screen, summarizing the source and destination(s) for the settings. Click **Apply** to execute the import or export. This operation cannot be undone.

#### Reset All

To reset all the channels in the IBERT core, use **IBERT_V6GTX > Reset All** or click the **Reset All** button in the toolbar.

#### JTAG Scan Rate and Scan Now

The **JTAG Scan Rate** toolbar and **IBERT_V6GTX > JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_V6GTX > Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

## IBERT Console Window for Virtex-6 FPGA GTH Transceivers

To open the console for a ChipScope Pro IBERT core for Virtex-6 HXT families, select **Window > New Unit Windows** and the desired core. A dialog box displays for that core, and you can select the **IBERT V6 GTH Console**. You cannot close windows from this dialog box.

You can achieve the same operation by double-clicking on the **IBERT V6 GTH Console** leaf node in the project tree, or by right-clicking on the **IBERT V6 GTH Console** leaf node and selecting **Open IBERT V6 GTH Console**.

The IBERT Console for Virtex-6 HXT families GTH transceivers is composed of: "MGT/BERT Settings Panel", "DRP Settings Panel," page 89, "Port Settings Panel,"

page 89, and "IBERT Virtex-6 FPGA GTX Transceiver Toolbar and Menu Options," page 92.

## MGT/BERT Settings Panel

The **MGT/BERT Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTH transceiver. Each row represents a specific control or status setting.

### MGT Settings

The **MGT Alias** setting is initially set to the MGT number of the GTH transceiver, but you can change it by selecting the field and typing in a new value.

The **Tile Location** setting denotes the X/Y coordinate of the GTH transceiver in the device.

The **MGT Link Status** indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTH transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NOT LOCKED (red).

The **PLL Status** indicator shows the lock status of the PLL that is a part of the GTH QUAD. The valid states of this status indicator are LOCKED (green) or NOT LOCKED (red).

The **Loopback Mode** setting is used to control the loopback mode of a particular GTH transceiver channel. The valid choices for loopback mode are:

- **None**: No feedback path is used.
- **Near-End PCS**: The circuit is wholly contained within the near-end GTH transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTH channel.
- **Far-End PCS**: The circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through part of the GTH transceiver channel. For this GTH loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.

The **QUAD Reset** button resets the GTH QUAD (four transceivers) by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The **TX Bit Error Inject** button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The **TX Diff Output Swing** combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The **TX Pre-Emphasis** combo box controls the amount of pre-emphasis on the transmitted signal. Change the value in the combo box to change the emphasis.

The **TX Post-Emphasis** combo box controls the amount of post-emphasis on the transmitted signal. Change the value in the combo box to change the emphasis.

The **RX Equalization** setting controls the internal RX equalization circuit.

### BERT Settings

The **TX/RX Data Pattern** settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The **RX Bit Error Ratio** field contains the currently calculated bit error ratio for the GTH transceiver channel. It is expressed as an exponent. For instance, 1.000E-12 means that one bit error happens (on average) for every trillion bits received.

The **RX Received Bit Count** field contains a running tally of the number of bits received. This count resets when the **BERT Reset** button is clicked.

The **RX Bit Error Count** field contains a running tally of the number of bit errors detected. This count resets when the **BERT Reset** button is clicked.

The **BERT Reset** button resets the bit error and received bit counters. It is appropriate to reset the BERT counters after the GTH channel is linked and stable.

### Clocking Settings

The **TXUSERCLKOUT Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSERCLKOUT port of the GTH transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSERCLKOUT Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSERCLKOUT port of the GTH transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

## DRP Settings Panel

The **DRP Settings** panel contains a table displaying the DRP attributes or addresses. Each column represents a specific active GTH QUAD, because there is one set of DRP attributes for each. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name**, all the DRP attributes are displayed in alphabetical order. The **Radix** combo lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press **Enter**. The new value is immediately set in the GTH transceiver.

When the radio button at the bottom of the panel is set to **View By Address**, the raw address are displayed in numerical order with their contents. The **Radix** combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press **Enter**. The new value is immediately set in the GTH transceiver.

## Port Settings Panel

The **Port Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTH QUAD. Each row represents a specific transceiver port. Not all ports are displayed, because some are used in the IBERT design to send and receive data.

The **Radix** combo box lets you choose to display the value in either Hex (hexadecimal) or Bin (binary). Some ports are read-only, and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text-fields. To change a value, place the cursor in those fields, type a new value, and press **Enter**. The new value is immediately written to the GTH transceiver.

## IBERT Virtex-6 FPGA GTH Transceiver Sweep Test Panel

### Sweep Test Settings Panel

The **Sweep Test Settings** panel is used to set up a channel test that sweeps through various transceiver settings. The TX and RX settings are for the same GTH transceiver. Sweeping through both TX and RX settings only works if the transceiver is set to one of the near-end or external loopback modes. Sweeping through RX parameters only can be performed when the corresponding TX endpoint for the link resides in a different device or a different transceiver in the same device.

The **Sweep Test Settings** panel consists of four sections: the **Parameter Settings** section, the **Sampling Point Region** section, the **Test Controls section**, and the **Test Results** section.

### Parameter Settings

Setting up the sweep test first involves setting up the sweep parameters. The GTH transceiver parameters that are available for sweep have the following defaults:

- TX Diff Swing
- TX Pre-Emphasis
- TX Post-Emphasis
- RX EQ

The sweep parameters can be initialized in one of two ways:

- Click the **Clear All Parameters** button to clear all parameters of the **Select…** option.
- Click the **Set Parameters to Current Values** button to set the parameters to their current values on the **MGT/BERT Settings** panel.

The order of the parameters in the **Sweep Parameter** table dictates how the parameters are swept. The values of the parameters near the top of the table are swept less frequently than the parameters near the bottom of the table. In other words, the parameters near the top of the table are in the outer loops of the sweep algorithm while the parameters near the bottom of the table are in the inner loops of the sweep algorithm.

Each parameter must be set up with a start and end value. The order of the parameter values cannot be changed. After you select the start value, the end values available for selection change automatically to include only valid selections. If you do not want to sweep through a parameter, set the start and end values for that parameter to the same value. The **Sweep Value Count** column indicates how many values are swept through for a particular parameter. After all sweep parameters have valid start and end values, the total number of sweep iterations appear in the **Total Iterations** field.

To add or remove items in the table, click the **Add/Remove Parameters** button. This opens a dialog box with all available ports/attributes on the left, and the ones to sweep on the right. To add new parameters to sweep, click it in the left hand list, and click the **>** button. To remove a parameter to sweep, click it in the right-hand list, and click the **<** button. To specify the sweep order of the parameters, click it in the right hand list, and then the **Up** or **Down** buttons. To revert the sweep attributes and their order to the default setting, click the **Reset to Default** button. Click **OK** to apply the settings, or **Cancel** to exit without saving.

### Sampling Point Region

The sampling point is the horizontal point within the eye to sample. Visualize on transition region at the far left, and the next at the far right. The RX Sampling Point is one of 128 discrete sampling positions. In the **Sampling Point Region** section, choose the left and right edges of the sampling region.

### Test Controls

After setting up the sweep test, start the test by clicking the **Start** button. The sweep parameter table is then disabled, and the test starts running.

As the sweep test runs, the current sweep result file, current iteration, elapsed time, and estimated time remaining status indicators are displayed. The sweep results are shown in the text area near the bottom of the screen. The sweep test can be paused by clicking the **Pause** button or stopped completely by clicking the **Reset** button.

The results from a sweep test are displayed in the **Test Results** panel and are also sent to a sweep test result file. Clicking on the **Log File Settings** button brings up the dialog window.

You can set both the location of the file and the number of sweep iterations stored in each file. If the total number of sweep iterations exceeds the file limit, multiple files with starting iteration number appended to the base file name are created in the same directory as the initial result file.

### Test Results

The **Test Results** panel identifies the current run, the elapsed time, and the estimated time remaining. Below this status information are three tabbed panels showing the results of the sweep test:

- Sweep Test Log
- Sweep Test Plots
- Sweep Test Info

### Sweep Test Log

The **Sweep Test Log** tab is always enabled and contains the running log of the sweep test results. The information there is shown using a text-only display. The results of the sweep test are also stored in a CSV log file.

### Sweep Test Plots

The **Sweep Test Plots** tab is enabled only after the sweep test has stopped running. The graphical data shown under this tab corresponds to the data stored in the CSV log file. Each plot corresponds to a sweep across the unit interval (UI) of the received signal. The left and right edges of the data plot correspond to the settings in the **Sample Point Region** panel.

The main measurement that is observed in the sweep test plot is the width of the UI "opening" of the active plot at the lowest measured bit error ratio (BER). The horizontal marker can be moved up and down and the vertical markers can be moved left and right by clicking and dragging them. By right-clicking in the plot list area on the right side of the **Sweep Test Plots** panel, you can hide or show each plot, rename and re-color each plot, and set a plot as active.

### Sweep Test Info

The **Sweep Test Info** tab enabled only after the sweep test has stopped running. The tabular data shown under this tab corresponds to the data shown under the graphical **Sweep Test Plots** tab. Each row in the table correspond to a single sweep test plot. The table columns correspond to the following:

- Enable Plot: shows or hides the plot in the **Sweep Test Plots** tab. Check to show, uncheck to hide.
- Line Color: color of the line used in the **Sweep Test Plots** tab. Click to select new color.
- Plot Name: name of the plot. Click the text field to change plot name.
- Opening at the Lowest BER Level: width of the longest run of zero errors at the lowest BER level.
- The remaining columns correspond to the parameter value settings that were used to create the corresponding sweep test plot.

Click the column header to sort the rows in the sweep test panel.

One useful sort strategy is to sort the plots from the widest to the lowest UI opening by clicking the **Opening at the Lowest BER Level** column header.

## IBERT Virtex-6 FPGA GTH Transceiver Toolbar and Menu Options

### IBERT Console Options

The IBERT Console Options dialog box allows you to select which columns and rows to display in the IBERT Console. The left-hand panel selects the transceivers by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the transceivers.

The right-hand panel selects which rows are displayed in the MGT/BERT settings. Use the **Check All** and **Uncheck All** buttons to select all or none of the rows. The **Default** button sets up the Console to display the basic set of rows needed to determine the health of the channels. The Default rows are Tile Location, TX PLL Status, RX PLL Status, Loopback Mode, Channel Reset, TX Error Inject, Rx Sampling Point, TX Data Pattern, RX Data pattern, Rx Bit Error Ratio, Rx Received Bit Count, Rx Bit Error Count, BERT Reset, TXUSRCLK2 Freq, and RXUSRCLK2 Freq.

### Import/Export Dialog Box

In the Import/Export dialog box, you can save and recall settings from a specific GTH transceiver, or apply the setting of one transceiver to others in the design. To import or export settings, ese **IBERT_V6GTH > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the transceiver settings, either **MGT** or **File**. If **MGT** is selected, choose among the available transceivers in the combo box. For **File**, click **Browse** and navigate to the settings file. Click **Next** to go to the next screen.

The second screen is the destination screen. Any combination of the GTH transceivers in the IBERT design and a file are available. If **File** is enabled, click **Browse** to specify the file destination.

The third screen is the **confirmation** screen, summarizing the source and destination(s) for the settings. Click **Apply** to execute the import or export. This operation cannot be undone.

### Reset All

To reset all the channels in the IBERT core, use **IBERT_V6GTH > Reset All** or click the **Reset All** button in the toolbar.

### JTAG Scan Rate and Scan Now

The **JTAG Scan Rate** toolbar and **IBERT_V6GTH > JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_V6GTH > Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

# IBERT Console Window for Spartan-6 FPGA GTP Transceivers

To open the console for a ChipScope Pro IBERT core for Spartan®-6 LXT devices, select **Window > New Unit Windows** and the core. A dialog box displays for that core, and you can select the **IBERT S6 GTP Console**. Windows cannot be closed from this dialog box.

The same operation can by achieved by double-clicking on the **IBERT S6 GTP Console** leaf node in the project tree, or by right-clicking on the **IBERT S6 GTP Console** leaf node and selecting **Open IBERT S6 GTP Console**.

The IBERT Console for Spartan-6 LXT platforms is composed of the MGT/BERT Settings Panel, the DRP Settings Panel, and the Port Settings Panel.

## MGT/BERT Settings Panel

The **MGT/BERT Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTP Transceiver. Each row represents a specific control or status setting.

### MGT Settings

The **MGT Alias** setting is initially set to the MGT number of the GTP transceiver, but can be changed by selecting the field and typing in a new value.

The **Tile Location** setting denotes the X/Y coordinate of the GTP transceiver in the device.

The **MGT Link Status** indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTP transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NOT LOCKED (red).

The **PLL Status** indicator shows the lock status of the PLL that is connected to the GTP. The valid states of this status indicator are LOCKED (green) or NOT LOCKED (red).

The **Loopback Mode** setting is used to control the loopback mode of a particular GTP transceiver channel. The valid choices for loopback mode are:

- None: No feedback path is used.
- Near-End PCS: The circuit is wholly contained within the near-end GTP transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTP transceiver channel GTP transceiver channel.
- Near-End PMA: The circuit is wholly contained within the near-end GTP transceiver channel. It starts at the TX fabric interface, passes through the PCS, through the PMA, back through the PCS, and returns to the RX fabric interface.

- Far-End PMA: The circuit originates and ends at some external channel endpoint (for example, a piece of test equipment or another device) but passes through the part of the GTP transceiver channel. For this GTP loopback mode, the signal comes into the RX pins, passes through the PMA circuitry, and returns immediately to the TX pins.
- Far-End PCS: The circuit originates and ends at some external channel endpoint (for example, a piece of test equipment or another device) but passes through part of the GTP transceiver channelGTP transceiver channel. For this GTP loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.
- Far-End Fabric: The circuit originates and ends at some external channel endpoint (for example, a piece of test equipment or another device) but passes through the entire GTP transceiver channelGTP transceiver channel and related fabric logic. For this GTP loopback mode, the signal comes into the RX pins, passes through the PMA and PCS, through a shallow fabric-based FIFO, back through the PCS and PMA, and finally returns to the TX pins.

The **Channel Reset** button resets the GTP transceiver channelGTP transceiver channel by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The **TX Polarity Invert** setting controls the polarity of the data sent out of the TX pins of the GTP transceiver channel. To flip the polarity of the TX side of the GTP transceiver, check the **TX Polarity Invert** box.

The **TX Bit Error Inject** button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The **TX Diff Output Swing** combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The **TX Pre-Emphasis** combo box controls the amount of pre-emphasis on the transmitted signal. Change the value in the combo to change the emphasis.

The **RX Polarity Invert** setting controls the polarity of the data received by the RX pins of the GTP transceiver channel. To flip the polarity of the RX side of the GTP transceiver, check the **RX Polarity Invert** box.

The **RX AC Coupling Enabled** setting controls whether the built-in AC coupling capacitors are enabled or not.

The **RX Termination Voltage** setting controls which supply is used in the RX termination network.

The **RX Equalization setting** controls the internal RX equalization circuit.

### BERT Settings

The **TX/RX Data Pattern** settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The **RX Bit Error Ratio** field contains the currently calculated bit error ratio for the GTP transceiver channel. It is expressed as an exponent. For instance, 1.000E-12 means that one bit error happens (on average) for every trillion bits received.

The **RX Received Bit Count** field contains a running tally of the number of bits received. This count resets when the **BERT Reset** button is pushed.

The **RX Bit Error Count** field contains a running tally of the number of bit errors detected. This count resets when the **BERT Reset** button is pushed.

The **BERT Reset** button resets the bit error and received bit counters. It is appropriate to reset the BERT counters after the GTP transceiver channel is linked and stable.

#### Clocking Settings

The **TXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK port of the GTP transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **TXUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK2 port of the GTP transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK port of the GTP transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

The **RXUSRCLK2 Freq (MHz)** indicator shows the approximate clocking frequency (in MHz) of the RXUSRCLK2 port of the GTP transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile-time.

### DRP Settings Panel

The **DRP Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTP transceiver. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name**, all the DRP attributes are displayed in alphabetical order. The **Radix** combo box lets you choose between Hex and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

When the radio button at the bottom of the panel is set to **View By Address**, the raw address are displayed in numerical order with their contents. The **Radix** combo box lets you choose between Hex and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press Enter. The new value is immediately set in the MGT.

### Port Settings Panel

The **Port Settings** panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTP transceiver. Each row represents a specific MGT port. Not all ports are displayed because some are used in the IBERT design to send and receive data.

The **Radix** combo lets you choose to display the value in either Hex or Bin (binary). Some ports are read-only, and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text-fields, and placing the cursor in those fields, typing a new value, and pressing **Enter** writes the new value to the MGT immediately.

## IBERT Spartan-6 FPGA GTP Sweep Test Panel

### Sweep Test Settings Panel

The Sweep Test Settings panel is used to set up a channel test that sweeps through various transceiver settings. The TX and RX settings are for the same GTP transceiver. Sweeping through both TX and RX settings only works if the transceiver is set to one of the near-end or external loopback modes. Sweeping through RX parameters only can be performed when the corresponding TX endpoint for the link resides in a different device or a different transceiver in the same device.

The **Sweep Test Settings** panel consists of four sections: the **Parameter Settings** section, the **Sampling Point Region** section, the **Test Controls** section, and the **Test Results** section.

### Parameter Settings

Setting up the sweep test first involves setting up the sweep parameters.

The GTP transceiver parameters that are available for sweep default to the following:

- TX Diff Swing
- TX Pre-Emphasis
- RX EQ

The sweep parameters can be initialized in one of two ways:

- Click the **Clear All Parameters** button to clear all parameters under the **Select…** option.
- Click the **Set Parameters to Current Values** button to set the parameters to their current values on the **MGT/BERT Settings** panel.

The order of the parameters in the Sweep Parameter table dictates how the parameters are swept. The values of the parameters near the top of the table are swept less frequently than the parameters near the bottom of the table. In other words, the parameters near the top of the table are in the outer loops of the sweep algorithm while the parameters near the bottom of the table are in the inner loops of the sweep algorithm.

Each parameter must be set up with a start and end value. The order of the parameter values cannot be changed. After you select the start value, the end values available for selection change automatically to include only valid selections. If you do not want to sweep through a parameter, set the start and end values for that parameter to the same value. The **Sweep Value Count** column indicates how many values are swept through for a particular parameter. After all sweep parameters have valid start and end values, the total number of sweep iterations are shown in the **Total Iterations** field.

To add or remove items in the table, click the **Add/Remove Parameters** button. This opens a dialog box with all available ports/attributes on the left, and the ones to sweep on the right. To add new parameters to sweep, click it in the left-hand list, and click the **>** button. To remove a parameter to sweep, click it in the right-hand list, and click the **<** button. To specify the sweep order of the parameters, click it in the right-hand list, and then the **Up** or **Down** buttons. To revert the sweep attributes and their order to the default setting, click the **Reset to Default** button. Click **OK** to apply the settings, or **Cancel** to exit without saving.

#### Sampling Point Region

The sampling point is the horizontal point within the eye to sample. Visualize on transition region at the far left, and the next at the far right. The RX Sampling Point is one of 128 discrete sampling positions. In the **Sampling Point Region** section, choose the left and right edges of the sampling region.

#### Test Controls

After setting up the sweep test, start the test by clicking the **Start** button. The sweep parameter table is then disabled, and the test starts running.

As the sweep test runs, the current sweep result file, current iteration, elapsed time, and estimated time remaining status indicators are displayed. The sweep results are shown in the text area near the bottom of the screen. The sweep test can be paused by clicking on the **Pause** button or stopped completely by clicking on the **Reset** button.

The results from a sweep test are displayed in the **Test Results** panel and are also sent to a sweep test result file. Clicking on the **Log File Settings** button brings up the dialog box.

You can set both the location of the file and the number of sweep iterations stored in each file. If the total number of sweep iterations exceeds the file limit, multiple files with starting iteration number appended to the base file name are created in the same directory as the initial result file.

#### Test Results

The **Test Results** panel shows which is the current run, the elapsed time, and the estimated time remaining. Below this status information are three tabbed panels showing the results of the sweep test:

- Sweep Test Log
- Sweep Test Plots
- Sweep Test Info

#### Sweep Test Log

The Sweep Test Log tabbed panel is always enabled and contains the running log of the sweep test results. The information on this panel is shown using a text-only display. The results of the sweep test are also stored in a CSV log file.

### IBERT Spartan-6 FPGA GTP Toolbar and Menu Options

#### IBERT Console Options

The IBERT Console Options dialog allows you to select which columns and rows to display in the IBERT Console. The left-hand panel selects the MGTs by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the MGTs.

The right-hand panel selects which rows are displayed in the MGT/BERT settings. Use the **Check All** and **Uncheck All** buttons to select all or none of the rows. The **Default** button sets up the Console to display the basic set of rows needed to determine the health of the channels. The Default rows are Tile Location, TX PLL Status, RX PLL Status, Loopback Mode, Channel Reset, TX Error Inject, Rx Sampling Point, TX Data Pattern, RX Data pattern, Rx Bit Error Ratio, Rx Received Bit Count, Rx Bit Error Count, BERT Reset, TXUSRCLK2 Freq, and RXUSRCLK2 Freq.

### Import/Export Dialog

The Import/Export Dialog allows you to save and recall settings from a specific MGT, or apply the setting of one MGT to others in the design. To import or export settings, use **IBERT_S6GTP > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the MGT settings- either **MGT** or **File**. If **MGT** is selected, choose among the available MGTs in the combo box. For **File**, click **Browse** and navigate to the settings file. Click **Next** to go to the next screen.

The second screen is the destination screen. Any combination of the MGTs in the IBERT design and a file are available. If **File** is enabled, click Browse to specify the file destination.

The third screen is the confirmation screen, summarizing the source and destination(s) for the settings. Click **Apply** to execute the import or export. This operation cannot be undone.

### Reset All

To reset all the channels in the IBERT core, use **IBERT_S6GTP > Reset All** or click the **Reset All** button in the toolbar.

### JTAG Scan Rate and Scan Now

The **JTAG Scan Rate** toolbar and **IBERT_S6GTP > JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_S6GTP > Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

# IBERT Console Window for 7 Series FPGA GTX Transceivers

To open the console for a ChipScope Pro IBERT core for Virtex-7/Kintex-7 FPGA GTX transceivers, select **Window > New Unit Windows** and the core desired. A dialog box displays for that core, and you can select the **IBERT K7 GTX Console**.

The same operation can be achieved by double-clicking the **IBERT K7 GTX Console** leaf node in the project tree, or by right-clicking the **IBERT K7 GTX Console** leaf node and selecting **Open IBERT K7 GTX Console**.

The IBERT Console for Kintex-7 GTX transceivers is composed of the following:

- MGT/BERT Settings Panel
- DRP Settings Panel
- Port Settings Panel
- IBERT Kintex-7 FPGA GTX Transceiver Toolbar and Menu Options

***Note:*** The IBERT Console for Virtex-7 FPGA GTX transceivers is identical to the Kintex-7 version that is referred to in the rest of this section.

## MGT/BERT Settings Panel

The MGT/BERT Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific control or status setting.

### MGT Settings

The MGT Alias setting is initially set to the MGT number of the GTX transceiver, but can be changed by selecting the field and typing in a new value.

The Tile Location setting denotes the X/Y coordinate of the GTX transceiver in the device. The MGT Link Status indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTX transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NO Link (red).

The CPLL/QPLL Status indicator shows the lock status of the CPLL/QPLL that is connected to the GTX transceiver. The valid states of this status indicator are CPLL/QPLL LOCKED (green) or CPLL/QPLL NOT LOCKED (red).

The Loopback Mode setting controls the loopback mode of a particular GTX transceiver channel. The valid choices for loopback mode are:

- None: no feedback path is used.
- Near-End PCS: the circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTX channel.
- Near-End PMA: the circuit is wholly contained within the near-end GTX transceiver channel. It starts at the TX fabric interface, passes through the PCS, through the PMA, back through the PCS, and returns to the RX fabric interface.
- Far-End PMA: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the part of the GTX transceiver channel. For this GTX transceiver loopback mode, the signal comes into the RX pins, passes through the PMA circuitry, and returns immediately to the TX pins.
- Far-End PCS: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through part of the GTX transceiver channel. For this GTX loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.
- Far-End Fabric: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the entire GTX transceiver channel and related fabric logic. For this GTX transceiver loopback mode, the signal comes into the RX pins, passes through the PMA and PCS, through a shallow fabric-based FIFO, back through the PCS and PMA, and finally returns to the TX pins.

The Channel Reset button resets the GTX transceiver channel by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The TX Polarity Invert setting controls the polarity of the data sent out of the TX pins of the GTX transceiver channel. To change the polarity of the TX side of the GTX transceiver, check the TX Polarity Invert box.

The TX Bit Error Inject button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The TX Diff Output Swing combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The TX Pre-Cursor and TX Post-Cursor combo boxes control the amount of pre-emphasis on the transmitted signal. Change the value in the combo boxes to change the pre-emphasis settings.

The RX Polarity Invert setting controls the polarity of the data received from the RX pins of the GTX transceiver channel. To change the polarity of the RX side of the GTX transceiver, check the RX Polarity Invert box.

The RX Termination Mode and RX Termination Voltage settings are used to control the receiver termination characteristics of the GTX channel. Use the RX Termination Mode combo box to control the mode for the termination voltage. Use the RX Termination Voltage combo box to control the common mode when RX Termination Mode is set to Programmable.

#### BERT Settings

The TX/RX Data Pattern settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The BERT Settings also include RX Received Bit Counter, RX Bit Error Count, and Bit Error Ratio (BER) indicator. The BERT Reset button is used to reset these counters.

#### Clocking Settings

The TXUSRCLK Freq (MHz) indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile time.

The TXUSRCLK2 Freq (MHz) indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK2 port of the GTX transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile time.

### DRP Settings Panel

The DRP Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name**, all the DRP attributes display in alphabetical order. The Radix combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, click in the text field for the value, type in a new value, and press **Enter**. The new value is set immediately in the MGT.

When the radio button at the bottom of the panel is set to **View By Address**, the raw addresses are displayed in numerical order with their contents. The Radix combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press **Enter**. The new value is immediately set in the MGT.

### Port Settings Panel

The Port Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTX transceiver. Each row represents a specific MGT port. Not all ports are displayed because some are used in the IBERT design to send and receive data.

The Radix combo lets you choose to display the value in either Hex (hexadecimal) or Bin (binary). Some ports are read-only and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text fields, and placing the cursor in those fields, typing a new value, and pressing **Enter** writes the new value to the MGT immediately.

## RX Margin Analysis Panel

Use the RX Margin Analysis panel to set up a channel test that sweeps through various transceiver settings to help find the optimal channel settings. The TX and RX settings shown in this panel are for the same GTX transceiver. Sweeping through both TX and RX settings only works if the transceiver is set to one of the near-end or external loopback modes. Sweeping through RX parameters only can be performed when the corresponding TX endpoint for the link resides in a different device or a different transceiver in the same device.

The tabbed Sweep Test Settings panel consists of four sections: MGT/Parameter Settings, Scan Settings, Test Controls, and Test Results.

### MGT/Parameter Settings

Use the MGT combo box to select the GTX transceiver that will participate in the sweep test.

***Note:*** Only one GTX transceiver can be swept at a time.

Before setting up the sweep test, set up the sweep parameters. While you can use any GTX parameters during the parameter sweep operation, the most useful GTX transceiver parameters available for sweep are the defaults:

- TX Diff Swing
- TX Pre-Cursor
- TX Post-Cursor

To add or remove items in the table, click the Add/Remove Parameters button. This opens a dialog box with all available ports/attributes on the left, and the ones to sweep on the right. To add a new sweep parameter, first select the parameter in the left-hand list then click the right arrow button to add it to the list on the right. To remove a parameter to sweep, first select it in the right-hand list, and click the left arrow button. To specify the sweep order of the parameters, click it in the right hand list, and then the Up or Down buttons. To revert the sweep attributes and their order to the default setting, click the Reset to Default button. Click OK to apply the settings, or Cancel to exit without saving.

The sweep parameters can be initialized in one of the following ways:

- Click the Clear All Parameters button to clear all parameters to the Select… option.
- Click the Set Parameters to Current Values button to set the parameters to their current values on the MGT/BERT Settings panel.

The order of the parameters in the Sweep Parameter table dictates how the parameters are swept. The values of the parameters near the top of the table are swept less frequently than the parameters near the bottom of the table. In other words, the parameters near the top of the table are in the outer loops of the sweep algorithm while the parameters near the bottom of the table are in the inner loops of the sweep algorithm.

Each parameter must be set up with a start and end value. The order of the parameter values cannot be changed. After you select the start value, the end values available for selection change automatically to include only valid selections. If you do not want to sweep through a parameter, set the start and end values for that parameter to the same value. The Sweep Value Count column indicates how many values are swept through for a particular parameter. After all sweep parameters have valid start and end values, the total number of sweep iterations are shown in the Total Iterations field.

### Scan Settings

The Virtex-7 and Kintex-7 GTX transceivers include a circuit called RX Eye Scan that performs a comparison between the received data at two different locations in the sample region:

- At the optimal sampling location, called "Data Sample"
- At sampling location offset in the horizontal and/or vertical direction, called "Offset Sample"

If the bit value sampled at the "Data Sample" location and the "Offset Sample" location match, then this not considered an error. However, if the two bit values do not match, then this is considered an error bit and causes the bit error ratio (BER) to increase.

The GTX RX Margin Analysis feature supports two different scan algorithms:

- 2D Full Scan: Scans all horizontal and vertical offset sampling points within the "eye". This mode is useful for creating a statistical eye map used to analyze the entire two-dimensional received eye margin.
- 1D Bathtub: Scans all horizontal sampling points through the 0 vertical row offset. This mode is useful for performing quick analysis of received phase margin.

The horizontal and vertical offsets can be controlled independently. Each offset can be controlled to have an interval and range. The Range combo box controls the maximum and minimum offset values of the scan. The Interval combo box controls how many rows or columns are skipped between each row or column that is scanned.

### Test Controls

After the sweep test has been set up, the test can be started by clicking the Start button. After the Start button is clicked, the sweep parameter table is disabled and the test starts running.

As the sweep test runs, the current sweep result file, current iteration, elapsed time, and estimated time remaining status indicators are displayed. The sweep results are shown in the Sweep Test Log text area near the bottom of the screen. The sweep test can be paused by clicking on the Pause button or stopped completely by clicking on the Stop & Reset button.

The results from a sweep test are displayed in the Test Results panel and are also sent to a sweep test result file. Clicking on the Log File Settings button brings up the dialog window. You can set both the location of the file and the number of sweep iterations stored in each file. If the total number of sweep iterations exceeds the file limit, multiple files with starting iteration number appended to the base file name are created in the same directory as the initial result file.

The Dwell controls allow you to specify how long a measurement is taken at a particular scan offset location. The BER option is used to specify the lowest bit error ratio that should be achieved before moving to the next scan offset. The Time option is used to specify how much time should be spent accumulating data before moving to the next scan offset.

### Test Results

The Test Results panel shows the current iteration, elapsed time, and estimated time remaining to complete the test. Below this status information are three tabbed panels showing the results of the sweep test:

- Sweep Test Log
- Sweep Test Plots
- Sweep Test Info

**Sweep Test Log**: The Sweep Test Log tabbed panel is always enabled and contains the running log of the sweep test results. The information on this panel is shown using a text-only display. The results of the sweep test are also stored in a CSV log file.

**Sweep Test Plots**: The Sweep Test Plots tabbed panel is enabled only after the sweep test has stopped running. The graphical data shown in the sweep test plot window corresponds to the data stored in the CSV log file.

If the 2D Full Scan algorithm is used, each plot corresponds to a sweep across the horizontal and the vertical ranges specified in the Scan Settings panel. The color of the plot region corresponds to the BER accumulation level at each offset sample point. Use the selection buttons in the Plot List area on the right side of the Sweep Test Plots panel to show one plot at a time. Use the right-click menu options to rename the plot.

Below the plot area is the Plot Type combo box. Contour (filled) is chosen by default which displays contour lines at each integer value (e-1, e-2, etc.) as well as halfway in between (5e-1, 5e-2, etc). Choosing Contour(lines) will display the same plot, but with the lines colored, and no fill color between them. Choosing Discrete values will not perform an interpolation between the values, and display each scanned point as one block of color according to the BER value.

If the 1D Bathtub scan algorithm is used, each plot corresponds to a sweep across the unit interval (UI) of the received signal. The left and right edges of the data plot correspond to the Horizontal settings in the Scan Settings panel. In 1D Bathtub mode, the main measurement that is observed in the sweep test plot is the width of the UI "opening" of the active plot at the lowest measured bit error ratio (BER). Use the selection buttons in the plot list area on the right side of the Sweep Test Plots panel hide or show each plot. Use the right-click menu options to rename or re-color each plot and set a plot as “active”. The horizontal and vertical markers help you take measurements on the “active” plot. The horizontal marker can be moved up and down and the vertical markers can be moved left and right by clicking and dragging them.

**Sweep Test Info**: The Sweep Test Info panel is also enabled only after the sweep test has stopped running. The tabular data shown in the Sweep Test Info panel corresponds to the data shown in the graphical Sweep Test Plots panel. Each row in the table corresponds to a single sweep test plot.

In 2D Full Scan algorithm mode, the table columns correspond to the following:

- Enable Plot: shows or hides the plot in the Sweep Test Plots panel. Check to show, uncheck to hide.
- Plot Name: name of the plot. Click the text field to change plot name.

In 1D Bathtub scan algorithm mode, the table columns correspond to the following:

- Enable Plot: shows or hides the plot in the Sweep Test Plots panel. Check to show, uncheck to hide.

- Line Color: color of the line used in the Sweep Test Plots panel. Click to select new color.
- Plot Name: name of the plot. Click the text field to change plot name.

The remaining columns correspond to the parameter value settings that were used to create the corresponding sweep test plot. Click the column header to sort the rows in the sweep test panel.

***Note:*** One useful sort strategy is to sort the plots from the widest to the lowest UI opening by clicking the **Opening at the Lowest BER Level** column header.

## IBERT 7 Series FPGA GTX Transceiver Toolbar and Menu Options

### IBERT Console Options

The IBERT Console Options dialog box allows you to select which columns and rows to display in the IBERT Console. The left panel selects the MGTs by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the MGTs.

The right panel selects which rows are displayed in the MGT/BERT settings. The Default button sets up the Console to display the basic set of rows needed to determine the health of the channels. The Default rows are Tile Location, PLL Status, Loopback Mode, Channel Reset, TX Polarity Invert, TX Error Inject, TX Diff Output Swing, TX Pre-Cursor, TX Post-Cursor, TX Data Pattern, TXUSRCLK Freq, and TXUSRCLK2 Freq.

### Import/Export Dialog

In the Import/Export dialog box, you can save and recall settings from a specific MGT or apply the setting of one MGT to others in the design. To import or export settings, select **IBERT_K7GTX > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the MGT settings, either **MGT** or **File**. If MGT is selected, choose among the available MGTs in the combo box. For File, click **Browse** and navigate to the settings file.

The second wizard screen is the destination screen. Any combination of the MGTs in the IBERT design and a file are available. If File is enabled, click **Browse** to specify the file destination.

The third screen is the confirmation screen, summarizing the source and destination(s) for the settings. Click **Apply** to execute the import or export. This operation cannot be undone.

### Reset All

To reset all the channels in the IBERT core, select **IBERT_K7GTX > Reset All** or click the **Reset All** button in the toolbar.

### JTAG Scan Rate and Scan Now

The **JTAG Scan Rate** toolbar and **IBERT_K7GTX > JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_K7GTX > Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

## IBERT Console Window for 7 Series FPGA GTH Transceivers

To open the console for a ChipScope Pro IBERT core for Virtex-7 FPGA GTH transceivers, select **Window > New Unit Windows** and the core desired. A dialog box displays for that core, and you can select the **IBERT V7 GTH Console**.

The same operation can be achieved by double-clicking the **IBERT V7 GTH Console** leaf node in the project tree, or by right-clicking the **IBERT V7 GTH Console** leaf node and selecting **Open IBERT V7 GTH Console**.

The IBERT Console for Virtex-7 GTH transceivers is composed of the following:

- MGT/BERT Settings Panel
- DRP Settings Panel
- Port Settings Panel
- IBERT Virtex-7 FPGA GTH Transceiver Toolbar and Menu Options

### MGT/BERT Settings Panel

The MGT/BERT Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTH transceiver. Each row represents a specific control or status setting.

#### MGT Settings

The MGT Alias setting is initially set to the MGT number of the GTH transceiver, but can be changed by selecting the field and typing in a new value.

The Tile Location setting denotes the X/Y coordinate of the GTH transceiver in the device. The MGT Link Status indicator is displays the status of the link detection logic that is connected to the receiver of a particular GTH transceiver channel. If the channel is linked (green), the measured line rate is displayed. If the channel is not linked, it displays NO Link (red).

The CPLL/QPLL Status indicator shows the lock status of the CPLL/QPLL that is connected to the GTH transceiver. The valid states of this status indicator are CPLL/QPLL LOCKED (green) or CPLL/QPLL NOT LOCKED (red).

The Loopback Mode setting controls the loopback mode of a particular GTH transceiver channel. The valid choices for loopback mode are:

- None: no feedback path is used.
- Near-End PCS: the circuit is wholly contained within the near-end GTH transceiver channel. It starts at the TX fabric interface, passes through the PCS, and returns immediately to the RX fabric interface without ever passing through the PMA side of the GTH channel.
- Near-End PMA: the circuit is wholly contained within the near-end GTH transceiver channel. It starts at the TX fabric interface, passes through the PCS, through the PMA, back through the PCS, and returns to the RX fabric interface.
- Far-End PMA: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the part of the GTH transceiver channel. For this GTH transceiver loopback mode, the signal comes into the RX pins, passes through the PMA circuitry, and returns immediately to the TX pins.
- Far-End PCS: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through part of the

GTH transceiver channel. For this GTH loopback mode, the signal comes into the RX pins, passes through the PMA, through the PCS, back through the PMA, and returns to the TX pins.

- Far-End Fabric: the circuit originates and ends at some external channel endpoint (for example, external test equipment or another device) but passes through the entire GTH transceiver channel and related fabric logic. For this GTH transceiver loopback mode, the signal comes into the RX pins, passes through the PMA and PCS, through a shallow fabric-based FIFO, back through the PCS and PMA, and finally returns to the TX pins.

The Channel Reset button resets the GTH transceiver channel by clearing and resetting all internal PMA and PCS circuitry as well as the related fabric interfaces.

The TX Polarity Invert setting controls the polarity of the data sent out of the TX pins of the GTH transceiver channel. To change the polarity of the TX side of the GTH transceiver, check the TX Polarity Invert box.

The TX Bit Error Inject button inverts the polarity of a single bit in a single transmitted word. The receiver endpoint of the channel that is connected to this transmitter should detect a single bit error.

The TX Diff Output Swing combo box controls the differential swing of the transmitter. Change the value in the combo box to change the swing.

The TX Pre-Cursor and TX Post-Cursor combo boxes control the amount of pre-emphasis on the transmitted signal. Change the value in the combo boxes to change the pre-emphasis settings.

The RX Polarity Invert setting controls the polarity of the data received from the RX pins of the GTH transceiver channel. To change the polarity of the RX side of the GTH transceiver, check the RX Polarity Invert box.

The RX Termination Mode and RX Termination Voltage settings are used to control the receiver termination characteristics of the GTH channel. Use the RX Termination Mode combo box to control the mode for the termination voltage. Use the RX Termination Voltage combo box to control the common mode when RX Termination Mode is set to Programmable.

### BERT Settings

The TX/RX Data Pattern settings are used to select the data pattern that is used by the transmit pattern generator and receive pattern checker, respectively. These patterns include PRBS 7, 15, 23, and 31, and Clk 2x and 10x.

The BERT Settings also include RX Received Bit Counter, RX Bit Error Count, and Bit Error Ratio (BER) indicator. The BERT Reset button is used to reset these counters.

### Clocking Settings

The TXUSRCLK Freq (MHz) indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK port of the GTH transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile time.

The TXUSRCLK2 Freq (MHz) indicator shows the approximate clocking frequency (in MHz) of the TXUSRCLK2 port of the GTH transceiver. The accuracy of this status indicator depends on the frequency of the system clock that was specified at compile time.

## DRP Settings Panel

The DRP Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTH transceiver. Each row represents a specific DRP attribute or address.

When the radio button at the bottom of the panel is set to **View By Attribute Name**, all the DRP attributes display in alphabetical order. The Radix combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, click in the text field for the value, type in a new value, and press **Enter**. The new value is set immediately in the MGT.

When the radio button at the bottom of the panel is set to **View By Address**, the raw addresses are displayed in numerical order with their contents. The Radix combo box lets you choose between Hex (hexadecimal) and Bin (binary). To change a value, just click in the text field where the value is, type in a new value, and press **Enter**. The new value is immediately set in the MGT.

## Port Settings Panel

The Port Settings panel contains a table that is made up of one or more vertical columns and horizontal rows. Each column represents a specific active GTH transceiver. Each row represents a specific MGT port. Not all ports are displayed because some are used in the IBERT design to send and receive data.

The Radix combo lets you choose to display the value in either Hex (hexadecimal) or Bin (binary). Some ports are read-only and not editable. Those cells in the table look like labels. The ports in the table that are editable look like text fields, and placing the cursor in those fields, typing a new value, and pressing **Enter** writes the new value to the MGT immediately.

## IBERT 7 Series FPGA GTH Transceiver Toolbar and Menu Options

### IBERT Console Options

The IBERT Console Options dialog box allows you to select which columns and rows to display in the IBERT Console. The left panel selects the MGTs by location. Use the **Check All** and **Uncheck All** buttons to select all or none of the MGTs.

The right panel selects which rows are displayed in the MGT/BERT settings. The Default button sets up the Console to display the basic set of rows needed to determine the health of the channels. The Default rows are Tile Location, PLL Status, Loopback Mode, Channel Reset, TX Polarity Invert, TX Error Inject, TX Diff Output Swing, TX Pre-Cursor, TX Post-Cursor, TX Data Pattern, TXUSRCLK Freq, and TXUSRCLK2 Freq.

### Import/Export Dialog

In the Import/Export dialog box, you can save and recall settings from a specific MGT or apply the setting of one MGT to others in the design. To import or export settings, select **IBERT_V7GTH > Import/Export Wizard** or click the **Import/Export Wizard** button in the toolbar.

The first screen of the wizard chooses the source of the MGT settings, either **MGT** or **File**. If MGT is selected, choose among the available MGTs in the combo box. For File, click **Browse** and navigate to the settings file.

The second wizard screen is the destination screen. Any combination of the MGTs in the IBERT design and a file are available. If File is enabled, click **Browse** to specify the file destination.

The third screen is the confirmation screen, summarizing the source and destination(s) for the settings. Click **Apply** to execute the import or export. This operation cannot be undone.

### Reset All

To reset all the channels in the IBERT core, select **IBERT_V7GTH > Reset All** or click the **Reset All** button in the toolbar.

### JTAG Scan Rate and Scan Now

The **JTAG Scan Rate** toolbar and **IBERT_V7GTH > JTAG Scan Rate** menu options are used to select how frequently the ChipScope Pro Analyzer tool queries the IBERT core for status information. The default is 1s between queries, but it can be set to 250 ms, 500 ms, 1s, 2s, 5s, or Manual Scan. When Manual Scan is selected, use **IBERT_V7GTH > Scan Now** or the **Scan Now** (or **S!**) toolbar button to query the IBERT core.

## Standalone IBERT Sweep Test Plot Viewer

The plot viewer in the Sweep Test panel of the IBERT Console can only be used to sweep test results while you are connected to the live device under test. If you want to view the sweep test results offline, you can use the standalone IBERT sweep test plot viewer:

- On Windows (32-bit) platforms: `<install dir>\bin\nt\ibertplotter.exe`
- On Windows (64-bit) platforms: `<install dir>\bin\nt64\ibertplotter.exe`
- On Linux (32-bit) platforms: `<install dir>/bin/lin/ibertplotter`
- On Linux (64-bit) platforms: `<install dir>/bin/lin64/ibertplotter`

You can use the standalone IBERT sweep test plot viewer to view CSV sweep test result files that were created by running IBERT sweep tests on the following transceivers:

- Spartan-6 FPGA GTP transceivers
- Virtex-5 FPGA GTX transceivers
- Virtex-6 FPGA GTX transceivers
- Virtex-6 FPGA GTH transceivers
- 7 Series FPGA GTX transceivers

You can also use the IBERT sweep test plot viewer to view multiple sweep test result CSV files at the same time. This is useful for comparing the results of different sweep test runs.

## Help Pages

The ChipScope Pro Analyzer tool help pages contain information for only the currently opened versions of the software and each of the core units. Selecting **Help > About: ChipScope Software** displays the version of the 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**.

## ChipScope Pro ILA Waveform Toolbar Features

In addition to the menu options, other ChipScope Pro Analyzer ILA waveform commands are available on a toolbar residing directly below the ChipScope Pro Analyzer menu. 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.

The toolbar buttons 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 **JTAG Chain > Auto Core Status Poll**
- **Run**: Same as **Trigger Setup > Run** (**F5**)
- **Stop**: Same as **Trigger Setup > Stop Acquisition** (**F9**)
- **Trigger Immediate**: Same as **Trigger Setup > Trigger Immediate** (**Ctrl+F5**)
- **Go To X Marker:** Same as **Waveform > Go To > Go To X Marker**
- **Go To O Marker:** Same as **Waveform > Go To > Go To O Marker**
- **Go To Previous Trigger:** Same as **Waveform > Go To > Trigger > Previous**
- **Go To Next Trigger:** Same as **Waveform > Go To > Trigger > Next**
- **Zoom In:** Same as **Waveform > Zoom > Zoom In**
- **Zoom Out:** Same as **Waveform > Zoom > Zoom Out**
- **Fit Window:** Same as **Waveform > Zoom > Zoom Fit**

## ChipScope Pro Analyzer Command Line Options

On Windows systems, the ChipScope Pro Analyzer tool can be started either from the command line or from the **Start** menu.

- On 32-bit Windows systems, you can invoke the ChipScope Pro Analyzer tool from the command line by running:

  `<XILINX_ISE_INSTALL>\bin\nt\analyzer.exe`
- On 64-bit Windows systems, you can invoke the ChipScope Pro Analyzer tool from the command line by running:

  `<XILINX_ISE_INSTALL>\bin\nt64\analyzer.exe`
- On 32-bit Linux systems, you can invoke the ChipScope Pro Analyzer tool from the command line by running:

  `<XILINX_ISE_INSTALL>/bin/lin/analyzer`
- On 64-bit Linux systems, you can invoke the ChipScope Pro Analyzer tool from the command line by running:

  `<XILINX_ISE_INSTALL>/bin/lin64/analyzer`

  where `<XILINX_ISE_INSTALL>` represents the location where the Xilinx ISE Design Suite tools are installed.

### Optional Arguments

The following command line options are available, if run from the command line:

`-geometry <width>x<height>+<left edge x coord>+<top edge y coord>`

Set location, width and height of the ChipScope Pro Analyzer tool 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 ChipScope Pro Analyzer tool exits. The default is:
`%userprofile%\.chipscope\cs_analyzer.ini`

`-log <path and filename>`

`-log stdout`

Write log messages to the specified file. Specifying `stdout` writes to standard output. The default is: `$HOME/.chipscope/cs_analyzer.log`

## Windows Command Line Example

```
C:\Xilinx\12.3\ISE_DS\ISE\bin\nt\analyzer.exe -log c:\proj\t\t.log -
init C:\proj\t\t.ini -project c:\proj\t\t.cpj -geometry 1000x300+30+600
```

Chapter 5

# ChipScope Engine Tcl Interface

## Overview

This interface provides Tcl scripting access to JTAG (Joint Test Action Group, IEEE standard) download cables using the communication library in the ChipScope™ logic analyzer engine. The purpose of the CSE/Tcl interface is to provide a simple scripting system to access basic JTAG, FPGA, and VIO (virtual input/output) core functions. In a few lines of Tcl script, you can scan and manipulate devices in the JTAG chain (and VIO cores in those devices) through standard Xilinx® JTAG cables.

For further information on JTAG, see *Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan* [See Reference 12, p. 211]. For information about Tcl, see Tcl Developer Xchange [See Reference 26, p. 212].

### Requirements

- A computer system running one of the supported operating systems described in the *ISE Design Suite Release Notes Guide* [See Reference 14, p. 211].
- A supported JTAG cable such as Platform Cable USB, Parallel Cable IV, or Parallel Cable III.
- A Tcl shell (`xtclsh` is provided in the ChipScope Pro and ISE® tool installations) or the ActiveTcl 8.4 shell [See Reference 24, p. 212].
- The required environment variables are set up by using `xtclsh.exe` (on Windows) or `xtclsh` (on Linux).

### Limitations

The ChipScope Engine Tcl interface package favors simplicity over performance. Some commands such as **`::chipscope::csejtag_tap_shift_chain_ir`** and **`::chipscope::csejtag_tap_shift_chain_dr`** transfer bits as hexadecimal strings (for example, "`30010A7`") 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 CSE/Tcl an easy-to-use means to interact with devices and cores in the JTAG chain.

***Note:*** The CSE/Tcl interface is only compatible with software that uses the CSE/Tcl interface to the JTAG cable communication device (such as the ChipScope Pro Analyzer tool and the Embedded Development Kit (EDK) XMD software debugger tool).

# CSE/Tcl Command Summary

The CSE/Tcl interface commands belong to a namespace called **::chipscope::**. The CSE/Tcl interface is comprised of four categories of commands (see Table 5-1).

*Table 5-1:* **CSE/Tcl Command Categories**

| Category | Description |
|---|---|
| CseJtag | JTAG interface status and control commands (see "CseJtag Tcl Commands"). |
| CseFpga | FPGA status and configuration commands (see "CseFpga Tcl Commands," page 120). |
| CseCore | ChipScope Pro core status commands (see "CseCore Tcl Commands," page 122). |
| CseVIO | ChipScope Pro VIO core status and control commands (see "CseVIO Tcl Commands," page 122). |

## CseJtag Tcl Commands

The CseJtag Tcl command category consists of four commands (see Table 5-2), each having one or more subcommands.

*Table 5-2:* **CseJtag Tcl Commands**

| Command | Description |
|---|---|
| `::chipscope::csejtag_session` | Manages CseJtag sessions. A session is used to maintain all data and messaging associated with a JTAG target. See Table 5-3 for a summary of all subcommands for this command. |
| `::chipscope::csejtag_db` | Interacts with the CseJtag JTAG database. The CseJtag JTAG database contains all data associated with known JTAG devices. See Table 5-4 for a summary of all subcommands for this command. |
| `::chipscope::csejtag_target` | Manages connections to CseJtag targets, such as JTAG download cables, JTAG emulators, and other JTAG devices. See Table 5-5, page 119 for a summary of all subcommands for this command. |
| `::chipscope::csejtag_tap` | Interacts with the JTAG Test Access Port (TAP) of CseJtag targets. Operations include navigating the TAP state machine and shifting data into and out of the TAP. See Table 5-6, page 120 for a summary of all subcommands for this command. |

A summary of the CseJtag Tcl subcommands is shown in Table 5-3. See "CseJtag Tcl Commands," page 123 for additional information about these commands.

***Note:*** Refer to the file csejtagglobals.tcl in the ChipScope Pro tool installation for all CseJtag Tcl global variable declarations.

*Table 5-3:* **Summary of ::chipscope::csejtag_session Subcommands**

| Subcommand | Description |
|---|---|
| `create` | Creates and initializes a session. |
| `destroy` | Destroys and frees up memory resources used by an existing session. |
| `get_api_version` | Gets the CseJtag API library version information. |
| `send_message` | Sends a message using the session message router function. |

*Table 5-4:* **Summary of ::chipscope::csejtag_db Subcommands**

| Subcommand | Description |
|---|---|
| `add_device_data` | Adds device records to the JTAG database. |
| `lookup_device` | Looks up device information in the JTAG database. |
| `get_device_name_for_idcode` | Gets the name of a device from the JTAG database by using an `IDCODE`. |
| `parse_bsdl` | Extracts device data for a JTAG device by parsing a Boundary Scan Description Language (BSDL) buffer. |
| `parse_bsdl_file` | Extracts device data for a JTAG device by parsing a Boundary Scan Description Language (BSDL) file. |

*Table 5-5:* **Summary of ::chipscope::csejtag_target Subcommands**

| Subcommand | Description |
|---|---|
| `open` | Opens a connection to a JTAG target and associate it with a session. |
| `close` | Closes the connection to an open JTAG target and remove it from the session. |
| `is_connected` | Tests and returns the connection status of the target. |
| `lock` | Attempts to obtain an exclusive lock on a JTAG target. |
| `unlock` | Releases an exclusive lock on a JTAG target. |
| `get_lock_status` | Gets the lock status of a JTAG target. |
| `clean_locks` | Releases all cable locks and cleans up lock-related resources. |
| `flush` | Flushes the data buffer of a JTAG target. |
| `set_pin` | Sets the value of a JTAG target TAP pin. |
| `get_pin` | Gets the value of a JTAG target TAP pin. |
| `pulse_pin` | Pulses a JTAG target TAP pin. |
| `wait_time` | Waits for a specified amount of time. |
| `get_info` | Gets information associated with a JTAG target. |

Table 5-6: **Summary of ::chipscope::csejtag_tap Subcommands**

| Subcommand | Description |
|---|---|
| `autodetect_chain` | Attempts to automatically detect all information pertaining to the JTAG chain currently connected to the target. |
| `interrogate_chain` | Scans the JTAG chain to determine the length of the chain and the `IDCODE` information of each device in the chain. |
| `get_device_count` | Gets the number of devices in the JTAG chain. |
| `set_device_count` | Sets the number of devices in the JTAG chain. |
| `get_irlength` | Gets the instruction register (IR) length of a device. |
| `set_irlength` | Sets the instruction register (IR) length of a device. |
| `get_device_idcode` | Gets the `IDCODE` of a device. |
| `set_device_idcode` | Sets the `IDCODE` of a device. |
| `navigate` | Navigates to a JTAG `TAP` state. |
| `shift_chain_ir` | Shifts a stream of bits into and out of the instruction register of the JTAG chain. |
| `shift_chain_dr` | Shifts a stream of bits into and out of the data register of the JTAG chain. |
| `shift_device_ir` | Shifts a stream of bits into and out of the instruction register of a particular device in the JTAG chain. |
| `shift_device_dr` | Shifts a stream of bits into and out of the data register of a particular device in the JTAG chain. |

## CseFpga Tcl Commands

The CseFpga Tcl command category consists of several commands (see Table 5-7).

***Note:*** Refer to the file csefpgaglobals.tcl in the ChipScope Pro tool installation for all CseFpga Tcl global variable declarations.

Table 5-7: **CseFpga Tcl Commands**

| Command | Description |
|---|---|
| `::chipscope::` `csefpga_configure_device` | Configures an FPGA device with the contents of a byte array containing the contents of a `.bit`, `.rbt`, or `.mcs` file. |
| `::chipscope::` `csefpga_configure_device_with_file` | Configures an FPGA device with the contents of a `.bit`, `.rbt`, or `.mcs` file. |
| `::chipscope::csefpga_get_config_reg` | Reads the configuration register bits of the target FPGA device. |
| `::chipscope::` `csefpga_get_instruction_reg` | Reads the instruction register of the target FPGA device and format the configuration-specific status bits. |

*Table 5-7:* **CseFpga Tcl Commands**

| Command | Description |
|---|---|
| ::chipscope::csefpga_get_usercode | Reads the USERCODE register of the target FPGA device. |
| ::chipscope:: csefpga_get_user_chain_count | Determines the number of USER scan chain registers in the target FPGA device. |
| ::chipscope:: csefpga_is_config_supported | Tests if configuration is supported for the target FPGA device. |
| ::chipscope::csefpga_is_configured | Returns the configuration status of an FPGA device. |
| ::chipscope:: csefpga_is_sys_mon_supported | Tests if System Monitor commands are supported for the target FPGA device. |
| ::chipscope:: csefpga_run_sys_mon_command_sequence | Executes a sequence of reads and writes from/to System Monitor registers. |
| ::chipscope::csefpga_get_sys_mon_reg | Reads from a System Monitor register. |
| ::chipscope::csefpga_set_sys_mon_reg | Writes to a System Monitor register. |
| ::chipscope::csefpga_assign_config_data_to_device | Assign configuration mask data from a buffer to a specific device. |
| ::chipscope::csefpga_assign_config_data_file_to_device | Assign configuration mask data from a file to a specific device. |

## CseCore Tcl Commands

The CseCore Tcl command category consists of several commands (see Table 5-8).

***Note:*** Refer to the file csecoreglobals.tcl in the ChipScope Pro tool installation for all CseCore Tcl global variable declarations.

*Table 5-8:* **CseCore Tcl Commands**

| Command | Description |
|---|---|
| `::chipscope::csecore_get_core_count` | Gets the number of cores attached to an ICON (integrated controller) core that targets an FPGA device and attaches to a particular `USER` scan chain register. |
| `::chipscope::csecore_get_core_status` | Retrieves the static status word from the target ChipScope Pro core. |
| `::chipscope::csecore_is_cores_supported` | Tests if a the target FPGA device supports ChipScope Pro cores. |

## CseVIO Tcl Commands

The CseVIO Tcl command category consists of several commands (see Table 5-9).

***Note:*** Refer to the file csevioglobals.tcl in the ChipScope Pro tool installation for all CseVIO Tcl global variable declarations.

*Table 5-9:* **CseVIO Tcl Commands**

| Command | Description |
|---|---|
| `::chipscope::csevio_get_core_info` | Reads the status word from the target VIO core. |
| `::chipscope::csevio_is_vio_core` | Determines whether the target core is a VIO core. |
| `::chipscope::csevio_init_core` | Initializes global variables associated with the target VIO core. |
| `::chipscope::csevio_terminate_core` | Removes global variables and releases memory associated with the target VIO core. |
| `::chipscope::csevio_define_signal` | Defines a name for a specified VIO signal bit. |
| `::chipscope::csevio_define_bus` | Defines a name for a grouping of VIO signal bits (called a `bus`). |
| `::chipscope::csevio_undefine_name` | Removes a VIO signal/bus name and all associated information. |
| `::chipscope::csevio_write_values` | Writes values to the specified signal/bus of the target VIO core. |
| `::chipscope::csevio_read_values` | Reads values from the specified signal/bus of the target VIO core. |

# CseJtag Tcl Commands

The following CseJtag Tcl Commands are described in detail in this section:

- ::chipscope::csejtag_session create
- ::chipscope::csejtag_session destroy
- ::chipscope::csejtag_session get_api_version
- ::chipscope::csejtag_session send_message
- ::chipscope::csejtag_target open
- ::chipscope::csejtag_target close
- ::chipscope::csejtag_target is_connected
- ::chipscope::csejtag_target lock
- ::chipscope::csejtag_target unlock
- ::chipscope::csejtag_target get_lock_status
- ::chipscope::csejtag_target clean_locks
- ::chipscope::csejtag_target flush
- ::chipscope::csejtag_target set_pin
- ::chipscope::csejtag_target get_pin
- ::chipscope::csejtag_target pulse_pin
- ::chipscope::csejtag_target wait_time
- ::chipscope::csejtag_target get_info
- ::chipscope::csejtag_tap autodetect_chain
- ::chipscope::csejtag_tap interrogate_chain
- ::chipscope::csejtag_tap get_device_count
- ::chipscope::csejtag_tap set_device_count
- ::chipscope::csejtag_tap get_irlength
- ::chipscope::csejtag_tap set_irlength
- ::chipscope::csejtag_tap get_device_idcode
- ::chipscope::csejtag_tap set_device_idcode
- ::chipscope::csejtag_tap navigate
- ::chipscope::csejtag_tap shift_chain_ir
- ::chipscope::csejtag_tap shift_device_ir
- ::chipscope::csejtag_tap shift_chain_dr
- ::chipscope::csejtag_tap shift_device_dr
- ::chipscope::csejtag_db add_device_data
- ::chipscope::csejtag_db lookup_device
- ::chipscope::csejtag_db get_device_name_for_idcode
- ::chipscope::csejtag_db get_irlength_for_idcode
- ::chipscope::csejtag_db parse_bsdl
- ::chipscope::csejtag_db parse_bsdl_file

## ::chipscope::csejtag_session create

This is typically the first subcommand call made to the ChipScope Engine. The session handle that is returned by this command allows you to open and control JTAG targets. This command also initializes the session with data obtained from various data files located in the default directory called `<LIBCSEJTAG_DLL_PATH>/../../data/cse`, where `<LIBCSEJTAG_DLL_PATH>` denotes the absolute location of the `libCseJtag.dll` file.

### Syntax

```
::chipscope::csejtag_session create messageRouterFn [opt_args...]
```

***Note:*** `[opt_args...]` is an optional list of arguments in string or list of string form.

### Arguments

*Table 5-10:* **Arguments for Subcommand ::chipscope::csejtag_session create**

| Argument | Type | Description |
|---|---|---|
| `messageRouterFn` | Required | Message router function name. Use a value of `0` to route all messages to `stdout`. Sample function declaration:<br>`proc messageRouterFn {handle msgFlags msg} { ... }`<br>msgFlags return one of the following:<br>`$CSE_MSG_ERROR`<br>`$CSE_MSG_WARNING`<br>`$CSE_MSG_STATUS`<br>`$CSE_MSG_INFO`<br>`$CSE_MSG_NOISE`<br>`$CSE_MSG_DEBUG` |
| `-server <host>` | Optional | Creates a session associated with the ChipScope server host name denoted by `<cs_server_host_name>`. |
| `-port <portnum>` | Optional | Creates a session associated with the ChipScope server port number denoted by `<cs_server_port_number>`. |

### Returns

A session handle. An exception is thrown if the command fails.

### Example

1. Create a new session with no optional arguments.

   ```
   %set handle [::chipscope::csejtag_session create messageRouterFn]
   ```

2. Create a new session using the client/server libraries to a server called `lab_machine` at port "50001".

   ```
   %set handle [::chipscope::csejtag_session create messageRouterFn
   -server "lab_machine" -port "50001"]
   ```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_session destroy

This command destroys an existing session and free all resources previously used by that session.

### Syntax

```
::chipscope::csejtag_session destroy handle
```

### Arguments

*Table 5-11:* **Arguments for Subcommand ::chipscope::csejtag_session create**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

An exception is thrown if the command fails.

### Example

Destroy the specified session

```
%::chipscope::csejtag_session destroy $handle
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_session get_api_version

This command retrieves the version of the CseJtag API library.

### Syntax

```
::chipscope::csejtag_session get_api_version
```

### Arguments

There are no arguments for this command.

### Returns

A Tcl list containing API version information. List elements are in the format:

```
{apiVersion versionString}
```

The `apiVersion` is the API version number and `versionString` is the build version number. An exception is thrown if command fails.

### Example

Obtain a list containing the API version number and the build number version string

```
%set api_info [::chipscope::csejtag_session get_api_version]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_session send_message

This subcommand sends a message to the message router function of the CseJtag library.

### Syntax

```
::chipscope::csejtag_session send_message handle msgType msg
```

### Arguments

*Table 5-12:* **Arguments for Subcommand ::chipscope::csejtag_session send_message**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `msgType` | Required | The type of message that must be set to one of the following:<br>• `$CSE_MSG_ERROR`<br>• `$CSE_MSG_WARNING`<br>• `$CSE_MSG_STATUS`<br>• `$CSE_MSG_INFO`<br>• `$CSE_MSG_NOISE`<br>• `$CSE_MSG_DEBUG` |
| `msg` | Required | The message string. |

### Returns

An exception is thrown if the command fails.

### Example

Send the message `"Hello World!"` to the message router function.

```
%::chipscope::csejtag_session send_message $handle $CSE_MSG_INFO
"Hello World!"
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target open

This subcommand opens a JTAG target device and associates it with a session.

***Note:*** Currently, only one JTAG target can be opened per session.

### Syntax

```
::chipscope::csejtag_target open handle targetName
progressCallbackFunc [opt_args...]
```

***Note:*** `[opt_args...]` is an optional list of arguments in string or list of string form.

### Arguments

*Table 5-13:* **Arguments for Subcommand ::chipscope::csejtag_target open**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| targetName | Required | Name of the JTAG target to open. See Table 5-14 for available `targetName` and `[optional args...]` combinations. If `targetName` is set to `$CSEJTAG_TARGET_AUTO`, then the first available JTAG cable target is opened. |
| progressCallbackFunc | Required | Progress callback function that can be used to monitor progress of JTAG target operations. The format of the progress callback function is: `proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}` The progress callback function must return either `$CSE_STOP or $CSE_CONTINUE`. If no progress callback function is necessary, a `0` should be passed into this argument position. |

Table 5-14 shows valid combinations of targetName argument values and their optional arguments.

*Table 5-14:* **Argument targetName and [optional args...] combinations**

| targetName | [optional args...] |
|---|---|
| `$CSEJTAG_TARGET_AUTO` | N/A |
| `$CSEJTAG_TARGET_PARALLEL` | `"port={LPT1 \| LPT2 \| LPT3}"` `"frequency={5000000 \| 2500000 \| 200000}"` |
| `$CSEJTAG_TARGET_PLATFORMUSB` | `"port=USB2 (aliased to USB21) \| USB21 \| USB22 \| USB23 \| ..."` `"ESN=<electronic serial number string>"` `"frequency={12000000 \| 6000000 \| 3000000 \| 1500000 \| 750000}"` |

### Returns

A list in the format:

```
{target_name plugin_name fw_ver driver_ver plugin_ver vendor frequency
port full_name target_uid rawinfo target_flags}
```

Where:

| Value | Description |
|---|---|
| `target_name` | The same as the `targetName` string |
| `plugin_name` | The plugin library name string |
| `fw_ver` | The firmware version string |
| `driver_ver` | The driver version string |
| `plugin_ver` | The plugin version string |
| `vendor` | The vendor string |
| `frequency` | The frequency string |
| `port` | The port string |
| `full_name` | The full target name string |
| `target_uid` | The target-unique ID string. For the Xilinx Platform Cable USB, this is the Electronic Serial Number (ESN). |
| `rawinfo` | The raw target info string |
| `target_flags` | The integer containing target-specific flags |

***Note:*** An exception is thrown if the subcommand fails.

### Example

1. Try to autodetect and open the target cable. Returns information on the opened target.

```
%set targetInfo [::chipscope::csejtag_target open $handle
$CSEJTAG_TARGET_AUTO progressFunc]
```

2. Try to open a Parallel cable in the port LPT1 with a frequency of 200000. Returns information on the opened target.

```
%set targetInfo [::chipscope::csejtag_target open $handle
$CSEJTAG_TARGET_PARALLEL progressFunc "port=LPT1" "frequency=200000"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target close

This subcommand closes a previously opened JTAG target device.

### Syntax

```
::chipscope::csejtag_target close handle
```

### Arguments

*Table 5-15:* **Arguments for Subcommand ::chipscope::csejtag_target close**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create.` |

### Returns

An exception is thrown if the subcommand fails.

### Example

Close the current target in the specified session.

```
%::chipscope::csejtag_target close $handle
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target is_connected

This subcommand tests the connection status of a JTAG target device.

### Syntax

```
::chipscope::csejtag_target is_connected handle
```

### Arguments

*Table 5-16:* **Arguments for Subcommand ::chipscope::csejtag_target is_connected**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

Connection status:

- 1 indicates connection to target is open and active
- 0 indicates closed.

An exception is thrown if the subcommand fails.

### Example

Return current target in the specified session.

```
%set isConnected (::chipscope::csejtag_target is_connected $handle)
```

Back to list of all CseJtag Tcl Commands

# ::chipscope::csejtag_target lock

This subcommand attempts to obtain an exclusive lock on a previously opened JTAG target device.

## Syntax

```
::chipscope::csejtag_target lock handle msWait
```

## Arguments

*Table 5-17:* **Arguments for Subcommand ::chipscope::csejtag_target lock**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create.` |
| `msWait` | | Wait time in milliseconds before giving up (-1 means wait until lock is gained). |

## Returns

The lock status in the form of one of the following:

- `$CSEJTAG_LOCKED_ME`
- `$CSEJTAG_LOCKED_OTHER`
- `$CSEJTAG_UNKNOWN`

An exception is thrown if the subcommand fails.

## Example

Attempt to obtain an exclusive target lock and wait at least 1000 milliseconds. Obtains the status of the lock.

```
%set lockStatus [::chipscope::csejtag_target lock $handle 1000]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target unlock

This subcommand releases an exclusive lock on a previously opened and locked JTAG target device.

### Syntax

```
::chipscope::csejtag_target unlock handle
```

### Arguments

*Table 5-18:* **Arguments for Subcommand ::chipscope::csejtag_target unlock**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Unlock the target in the specified session.

```
%::chipscope::csejtag_target unlock $handle
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target get_lock_status

This subcommand retrieves the lock status for the target device.

### Syntax

```
::chipscope::csejtag_target get_lock_status handle
```

### Arguments

*Table 5-19:* **Arguments for Subcommand ::chipscope::csejtag_target get_lock_status**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

Status of the lock in the form of one of the following:

- `$CSEJTAG_LOCKED_ME`
- `$CSEJTAG_LOCKED_OTHER`
- `$CSEJTAG_UNKNOWN`

An exception is thrown if the subcommand fails.

### Example

Obtain the current lock status.

```
%set lockStatus [::chipscope::csejtag_target get_lock_status $handle]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target clean_locks

This subcommand cleans up all JTAG target locks.

***Note:*** This subcommand must be used only as a last resort. The subcommand kills all sharing semaphores, including those used by other processes and applications. It currently only cleans up locks for JTAG cable targets.

### Syntax

```
::chipscope::csejtag_target clean_locks handle
```

### Arguments

*Table 5-20:* **Arguments for Subcommand ::chipscope::csejtag_target clean_locks**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Clean locks as a last resort because the application closed unexpectedly and `::chipscope::csejtag_target open` does not open the target successfully.

```
%::chipscope::csejtag_target clean_locks $handle
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target flush

This subcommand flushes the buffer associated with a previously opened and locked JTAG target device.

***Note:*** The JTAG target must be locked by using the **`::chipscope::csejtag_target lock`** subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_target flush handle
```

### Arguments

*Table 5-21:* **Arguments for Subcommand ::chipscope::csejtag_target flush**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Attempt to flush an opened and locked buffer of a JTAG target to make data writes occur immediately.

```
%::chipscope::csejtag_target flush $handle
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target set_pin

This subcommand sets the value of a JTAG TAP pin for a previously opened and locked JTAG target device.

***Note:*** The JTAG target must be locked by using the **`::chipscope::csejtag_target lock`** subcommand before calling this subcommand.

If using this function to change the JTAG TAP state, please be aware that the CseJtag Tcl library does not keep track of the JTAG TAP state. Before using any of the `::chipscope::csejtag_tap` subcommands, use the `::chipscope::csejtag_tap navigate` subcommand to set the JTAG TAP state machine to the `$CSEJTAG_TEST_LOGIC_RESET` state.

### Syntax

```
::chipscope::csejtag_target set_pin handle pin value
```

### Arguments

*Table 5-22:* **Arguments for Subcommand ::chipscope::csejtag_target set_pin**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `pin` | | JTAG TAP pin identifier `{$CSEJTAG_TMS | $CSEJTAG_TDI}`. To change the `$CSEJTAG_TCK` pin, use the subcommand `::chipscope::csejtag_target pulse_pin`. |
| `value` | | JTAG TAP pin value {1=set, 0=clear} |

### Returns

An exception is thrown if the subcommand fails.

### Example

Set the TMS pin to `1`.

```
%::chipscope::csejtag_target set_pin $handle $CSEJTAG_TMS 1
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target get_pin

This subcommand retrieves the value of a JTAG TAP pin for a previously opened and locked JTAG target device.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_target get_pin handle pin
```

### Arguments

*Table 5-23:* **Arguments for Subcommand ::chipscope::csejtag_target get_pin**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `pin` | Required | JTAG TAP pin identifier `{$CSEJTAG_TMS | $CSEJTAG_TCK | $CSEJTAG_TDI | $CSEJTAG_TDO}`. |

### Returns

JTAG TAP pin value {1=set, 0=clear}

An exception is thrown if the subcommand fails.

### Example

Get the current value of the TDO pin.

```
%set value [::chipscope::csejtag_target set_pin $handle $CSEJTAG_TDO]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target pulse_pin

This subcommand pulses the value of a JTAG TAP pin for a previously opened and locked JTAG target device.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

If using this function to change the JTAG TAP state, please be aware that the CseJtag Tcl library does not keep track of the JTAG TAP state. Before using any of the `::chipscope::csejtag_tap` subcommands, use the `::chipscope::csejtag_tap navigate` subcommand to set the JTAG TAP state machine to the `$CSEJTAG_TEST_LOGIC_RESET` state.

### Syntax

```
::chipscope::csejtag_target pulse_pin handle pin count
```

### Arguments

*Table 5-24:* **Arguments for Subcommand ::chipscope::csejtag_target pulse_pin**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| pin | Required | JTAG TAP pin identifier {`$CSEJTAG_TMS | $CSEJTAG_TCK | $CSEJTAG_TDI`}. |
| count | Required | Number of times to pulse the JTAG TAP pin (pulse means driving a `0`, then a `1`, then a `0` on the pin). |

### Returns

An exception is thrown if the subcommand fails.

### Example

Pulse the TCK pin five times.

```
%::chipscope::csejtag_target pulse_pin $handle $CSEJTAG_TCK 5
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target wait_time

This subcommand waits for a specified amount of time (in microseconds).

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_target wait_time handle usecs
```

### Arguments

*Table 5-25:* **Arguments for Subcommand ::chipscope::csejtag_target wait_time**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `usecs` | | Number of microseconds to wait. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Instruct the JTAG target to wait 1000 microseconds before performing another operation.

```
%::chipscope::csejtag_target wait_time $handle 1000
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_target get_info

This subcommand retrieves information from a previously opened JTAG target.

***Note:*** A JTAG target lock does not need to be obtained prior to calling this function.

### Syntax

```
::chipscope::csejtag_target get_info handle
```

### Arguments

*Table 5-26:* **Arguments for Subcommand ::chipscope::csejtag_target get_info**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

A list in the format:

```
{target_name plugin_name fw_ver driver_ver plugin_ver vendor frequency
port full_name target_uid rawinfo target_flags}
```

Where:

| Value | Description |
|---|---|
| `target_name` | The name of the JTAG target |
| `plugin_name` | The plugin library name string |
| `fw_ver` | The firmware version string |
| `driver_ver` | The driver version string |
| `plugin_ver` | The plugin version string |
| `vendor` | The vendor string |
| `frequency` | The frequency string |
| `port` | The port string |
| `full_name` | The full target name string |
| `target_uid` | The target-unique ID string |
| `rawinfo` | The raw target info string |
| `target_flags` | The integer containing target-specific flags |

***Note:*** An exception is thrown if the subcommand fails.

### Example

Obtain information about the current JTAG target.

```
%set targetInfo [::chipscope::csejtag_target get_info $handle]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap autodetect_chain

This subcommand attempts to automatically detect the composition of the JTAG chain. The subcommand first obtains the number of devices and IDCODE values for devices in the JTAG chain. The IR lengths are then determined for the devices in the JTAG chain that have an IDCODE. The IR lengths for devices that do not have corresponding IDCODEs must be assigned manually. Upon success, all pertinent device information is determined and set in the session. Some IEEE 1149.1 non-compliant devices might not be compatible with this subcommand and might cause the entire chain to be detected incorrectly or not at all.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap autodetect_chain handle algorithm
```

### Arguments

*Table 5-27:* **Arguments for Subcommand ::chipscope::csejtag_tap autodetect_chain**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| algorithm | | Algorithm used to determine the composition of the JTAG chain. Can be set to one of `{$CSEJTAG_SCAN_DEFAULT \| $CSEJTAG_SCAN_TLRSHIFT \| $CSEJTAG_SCAN_WALKING_ONES}`<br>The `CSEJTAG_SCAN_WALKING_ONES` algorithm is:<br>• Set each device into `BYPASS` by shifting long stream of 1's into `IR`.<br>• Shift `DR` pattern into `TDI` and wait for pattern on `TDO`. The number of shifts determines the number of devices in the JTAG chain.<br>• Perform the `CSEJTAG_SCAN_TLRSHIFT` algorithm to get `IDCODE`s for each device.<br>The `CSEJTAG_SCAN_TLRSHIFT` algorithm is:<br>• Navigate to `TLR`<br>Shift out bits until all `IDCODE`s (or `BYPASS` bits) are read. |

### Returns

An exception is thrown if the subcommand fails to detect the chain completely. In the case of such an error, the devices in the JTAG chain must be detected and assigned manually.

### Example

Attempt to automatically detect the chain using the default algorithm.

```
%::chipscope::csejtag_tap autodetect_chain $handle
$CSEJTAG_SCAN_DEFAULT
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap interrogate_chain

This subcommand scans the JTAG chain to obtain the IDCODE and number of devices in the chain. Some IEEE 1149.1 non-compliant devices might not be compatible with this subcommand and can cause the entire chain to be detected incorrectly or not at all. This command does not update the instruction register (IR) lengths of each device.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap interrogate_chain handle algorithm
```

### Arguments

*Table 5-28:* **Arguments for Subcommand ::chipscope::csejtag_tap interrogate_chain**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `algorithm` | | Algorithm used to determine the composition of the JTAG chain. Can be set to one of `{$CSEJTAG_SCAN_DEFAULT \| $CSEJTAG_SCAN_TLRSHIFT \| $CSEJTAG_SCAN_WALKING_ONES}`<br>The `CSEJTAG_SCAN_WALKING_ONES` algorithm is:<br>• Set each device into `BYPASS` by shifting long stream of 1's into `IR`<br>• Shift `DR` pattern into `TDI` and wait for pattern on `TDO`. The number of shifts determines the number of devices in the JTAG chain.<br>• Perform the `CSEJTAG_SCAN_TLRSHIFT` algorithm to get `IDCODE`s for each device.<br>The `CSEJTAG_SCAN_TLRSHIFT` algorithm is:<br>• Navigate to `TLR`<br>• Shift out bits until all `IDCODE`s (or `BYPASS` bits) are read |

### Returns

An exception is thrown if the subcommand fails.

### Example

Attempt to interrogate the chain using the default algorithm.

```
%::chipscope::csejtag_tap interrogate_chain $handle
$CSEJTAG_SCAN_DEFAULT
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap get_device_count

This subcommand is used to get the number of devices in the current JTAG chain.

***Note:*** The JTAG target must be locked by using the **`::chipscope::csejtag_target lock`** subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap get_device_count handle
```

### Arguments

*Table 5-29:* **Arguments for Subcommand ::chipscope::csejtag_tap get_device_count**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |

### Returns

The number of devices in the chain.

An exception is thrown if the subcommand fails.

### Example

Obtain the number of devices in the JTAG chain.

```
%set deviceCount [::chipscope::csejtag_tap get_device_count $handle]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap set_device_count

This subcommand is used to set the number of devices in the current JTAG chain.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap set_device_count handle count
```

### Arguments

*Table 5-30:* **Arguments for Subcommand ::chipscope::csejtag_tap set_device_count**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `count` | | Number of devices in the JTAG chain. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Set the number of devices in the JTAG chain to four.

```
%::chipscope::csejtag_tap set_device_count $handle 4
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap get_irlength

This subcommand retrieves the instruction register (IR) length of a device in the current JTAG chain. The IR length is used to determine the amount of padding required to shift an instruction into a device register. TAP shift and navigate operations do not work until all devices have the IR lengths set up correctly. The `::chipscope::csejtag_tap autodetect_chain` subcommand automatically sets up IR lengths for all devices in the chain that support the `IDCODE` command.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. Also, the device count must be set prior to calling this subcommand using the `::chipscope::csejtag_tap set_device_count`.

### Syntax

```
::chipscope::csejtag_tap get_irlength handle deviceIndex
```

### Arguments

*Table 5-31:* **Arguments for Subcommand ::chipscope::csejtag_tap get_irlength**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |

### Returns

The length of the IR for the device.

An exception is thrown if the subcommand fails.

### Example

Get the IR length of the device at index 0.

```
%set irLength [::chipscope::csejtag_tap get_irlength $handle 0]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap set_irlength

This subcommand sets the instruction register (IR) length of a single device in the current JTAG chain. The `IR` length is used to determine the amount of padding required to shift an instruction into a device register. TAP shift and navigate operations do not work until all devices have the IR lengths set up correctly. The `::chipscope::csejtag_tap autodetect_chain` subcommand automatically sets up IR lengths for all devices in the chain that support the `IDCODE` command.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. Also, the device count must be set prior to calling this subcommand using the `::chipscope::csejtag_tap set_device_count`.

### Syntax

```
::chipscope::csejtag_tap set_irlength handle deviceIndex irLength
```

### Arguments

*Table 5-32:* **Arguments for Subcommand ::chipscope::csejtag_tap set_irlength**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `irLength` | | Length of the IR (in bits) |

### Returns

An exception is thrown if the subcommand fails.

### Example

Set the IR length of the device at index 0 to 11 bits.

```
%::chipscope::csejtag_tap set_irlength $handle 0 11
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap get_device_idcode

This subcommand returns the 32-bit IDCODE for a given device in the current JTAG chain. If the device does not support the IDCODE instruction, a null string is returned.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. Also, the device count must be set prior to calling this subcommand using the `::chipscope::csejtag_tap set_device_count`.

### Syntax

```
::chipscope::csejtag_tap get_device_idcode handle deviceIndex
```

### Arguments

*Table 5-33:* **Arguments for Subcommand ::chipscope::csejtag_tap get_device_idcode**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |

### Returns

A 32-character string of ones and zeros representing the 32-bit IDCODE of the device.

An exception is thrown if the subcommand fails.

### Example

Get the IDCODE of the device at index 0

```
%set idcode [::chipscope::csejtag_tap get_device_idcode $handle 0]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap set_device_idcode

This subcommand sets the `IDCODE` for a given device in the current JTAG chain. Passing a null string indicates the device does not support the `IDCODE` instruction.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. Also, the device count must be set prior to calling this subcommand using the `::chipscope::csejtag_tap set_device_count`.

### Syntax

```
::chipscope::csejtag_tap set_device_idcode handle deviceIndex idcode
```

### Arguments

*Table 5-34:* **Arguments for Subcommand ::chipscope::csejtag_tap set_device_idcode**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `idcode` | | A 32-character string of ones and zeros representing the 32-bit `IDCODE` of the device. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Set the IDCODE of the device at index 0 to `01010101010101010101010101010101`.

```
%::chipscope::csejtag_tap set_device_idcode $handle 0
"01010101010101010101010101010101"
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap navigate

This subcommand is used to change the state of the TAP of a device in the JTAG chain.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap navigate handle newState clockRepeat
microseconds
```

### Arguments

*Table 5-35:* **Arguments for Subcommand ::chipscope::csejtag_tap navigate**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `newState` | | New state to navigate into. |
| `clockRepeat` | | Number of additional times to pulse the TCK pin after entering the new state. |
| `microseconds` | | Number of microseconds to sleep after navigating to the new state. |

### Returns

An exception is thrown if the subcommand fails.

### Example

Navigate the TAP state to Test Logic Reset and keep it in this state for five additional clock cycles.

```
%::chipscope::csejtag_tap navigate $handle $CSEJTAG_TEST_LOGIC_RESET 5
0
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap shift_chain_ir

This subcommand is used to shift a stream of bits into and out of the instruction register of the JTAG chain. No device padding is performed by this subcommand. For device-indexed `IR` shifting, see "::chipscope::csejtag_tap shift_device_ir," page 152.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap shift_chain_ir handle shiftMode exitState
progressCallbackFunc bitCount hextdibuf [-hextdimask hextdimaskval] [-
hextdomask hextdomaskval]
```

### Arguments

*Table 5-36:* **Arguments for Subcommand ::chipscope::csejtag_tap shift_chain_ir**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `shiftMode` | | `{CSJTAG_SHIFT_READ \| CSJTAG_SHIFT_WRITE \| CSJTAG_SHIFT_READWRITE)` |
| `exitState` | | State to end in after shift is complete (`CSEJTAG_SHIFT_IR` if no state change is desired). |
| `progressCallbackFunc` | | Progress callback function that can be used to monitor progress of JTAG target operations. The format of the progress callback function is: `proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}` The progress callback function must return either `$CSE_STOP or $CSE_CONTINUE`. If no progress callback function is necessary, a 0 should be passed into this argument position. |
| `bitCount` | | Number of bits to shift. |
| `hextdibuf` | | Data buffer that holds the data bits to be written into TDI. The least-significant bit is shifted into TDI first. |
| `-hextdimask hextdimaskval` | Optional | Specifies that a mask word hextdimaskval should be applied to the data buffer bits before the data is shifted into the TDI pin of the JTAG TAP. |
| `-hextdomask hextdomaskval` | | Specifies that a mask word hextdomaskval should be applied to the data buffer bits after the data is shifted out of the TDO pin of the JTAG TAP. |

### Returns

A buffer that is full of the data that is shifted out of the TDO pin of the JTAG TAP.

An exception is thrown if the subcommand fails.

### Example

This function shifts in 64 ones into the instruction register, captures the 64 bits of received data, and navigates to the Run Test Idle state when finished.

```
%set hextdobuf [::chipscope::csejtag_tap shift_chain_ir $handle
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 64
“FFFFFFFFFFFFFFFF”]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap shift_device_ir

This subcommand is used to shift a stream of bits into and out of the instruction register of a particular device the JTAG chain. Device padding is performed by this subcommand by putting all other devices into BYPASS mode. This subcommand must be called before `::chipscope::csejtag_tap shift_device_dr` to ensure all non-target devices as in BYPASS mode, otherwise unexpected and unintended results can occur. For raw data shifting into the chain IR, see "::chipscope::csejtag_tap shift_chain_ir," page 150.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. Also, the number of bits shifted into the device IR must be exactly equal to the IR length of the device, or the subcommand will fail.

### Syntax

```
::chipscope::csejtag_tap shift_device_ir handle deviceIndex shiftMode
exitState progressCallbackFunc bitCount hextdibuf [-hextdimask
hextdimaskval] [-hextdomask hextdomaskval]
```

### Arguments

*Table 5-37:* **Arguments for Subcommand ::chipscope::csejtag_tap shift_device_ir**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |
| `shiftMode` | | `{CSJTAG_SHIFT_READ \| CSJTAG_SHIFT_WRITE \| CSJTAG_SHIFT_READWRITE)` |
| `exitState` | | State to end in after shift is complete (`CSEJTAG_SHIFT_IR` if no state change is desired). |
| `progressCallbackFunc` | | Progress callback function that can be used to monitor progress of JTAG target operations. The format of the progress callback function is:<br>`proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}`<br>The progress callback function must return either `$CSE_STOP or $CSE_CONTINUE`. If no progress callback function is necessary, a `0` should be passed into this argument position. |
| `bitCount` | | Number of bits to shift. |
| `hextdibuf` | | Data buffer that holds the data bits to be written into TDI. The least-significant bit is shifted into TDI first. |
| `-hextdimask hextdimaskval` | Optional | Specifies that a mask word hextdimaskval should be applied to the data buffer bits before the data is shifted into the TDI pin of the JTAG TAP. |
| `-hextdomask hextdomaskval` | | Specifies that a mask word `hextdomaskval` should be applied to the data buffer bits after the data is shifted out of the TDO pin of the JTAG TAP. |

### Returns

A buffer that is full of the data that is shifted out of the TDO pin of the JTAG TAP.

An exception is thrown if the subcommand fails.

### Example

This function shifts in 11 ones into the instruction register of the device at index 1, captures the 11 bits of received data, and navigates to the Run Test Idle state when finished.

```
%set hextdobuf [::chipscope::csejtag_tap shift_device_ir $handle 1
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 11 "7FF"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap shift_chain_dr

This subcommand is used to shift a stream of bits into and out of the data register (DR) of the JTAG chain. No device padding is performed by this subcommand. For device-indexed DR shifting, see `::chipscope::csejtag_tap shift_device_dr`.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand.

### Syntax

```
::chipscope::csejtag_tap shift_chain_dr handle shiftMode exitState
progressCallbackFunc bitCount hextdibuf [-hextdimask hextdimaskval] [-
hextdomask hextdomaskval]
```

### Arguments

*Table 5-38:* **Arguments for Subcommand ::chipscope::csejtag_tap shift_chain_dr**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `shiftMode` | | `{CSJTAG_SHIFT_READ | CSJTAG_SHIFT_WRITE | CSJTAG_SHIFT_READWRITE}` |
| `exitState` | | State to end in after shift is complete (`CSEJTAG_SHIFT_DR` if no state change is desired). |
| `progressCallbackFunc` | | Progress callback function that can be used to monitor progress of JTAG target operations. The format of the progress callback function is:<br>`proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}`<br>The progress callback function must return either `$CSE_STOP or $CSE_CONTINUE`. If no progress callback function is necessary, a `0` should be passed into this argument position. |
| `bitCount` | | Number of bits to shift. |
| `hextdibuf` | | Data buffer that holds the data bits to be written into `TDI`. The least-significant bit is shifted into `TDI` first. |
| `-hextdimask hextdimaskval` | Optional | Specifies that a mask word `hextdimaskval` should be applied to the data buffer bits before the data is shifted into the TDI pin of the JTAG TAP. |
| `-hextdomask hextdomaskval` | | Specifies that a mask word `hextdomaskval` should be applied to the data buffer bits after the data is shifted out of the TDO pin of the JTAG TAP. |

### Returns

A buffer that is full of the data that is shifted out of the TDO pin of the JTAG TAP.

An exception is thrown if the subcommand fails.

### Example

This function shifts in 64 ones into the instruction register, captures the 64 bits of received data, and navigates to the `Run Test Idle` state when finished.

```
%set hextdobuf [::chipscope::csejtag_tap shift_chain_dr $handle
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 64
"FFFFFFFFFFFFFFFF"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_tap shift_device_dr

This subcommand is used to shift a stream of bits into and out of the data register of a particular device the JTAG chain. Device padding is performed by this subcommand by assuming all non-target devices are in BYPASS mode, then adding the necessary heading and trailing bits to accommodate for the position of the target device in the chain. For raw data shifting into the chain DR, see `::chipscope::csejtag_tap shift_chain_dr`.

***Note:*** The JTAG target must be locked by using the `::chipscope::csejtag_target lock` subcommand before calling this subcommand. This subcommand must be called before `::chipscope::csejtag_tap shift_device_dr` to ensure all non-target devices as in `BYPASS` mode, otherwise unexpected and unintended results can occur.

### Syntax

```
::chipscope::csejtag_tap shift_device_dr handle deviceIndex shiftMode
exitState progressCallbackFunc bitCount hextdibuf [-hextdimask
hextdimaskval] [-hextdomask hextdomaskval]
```

### Arguments

*Table 5-39:* **Arguments for Subcommand ::chipscope::csejtag_tap shift_device_dr**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |
| `shiftMode` | | `{CSJTAG_SHIFT_READ \| CSJTAG_SHIFT_WRITE \| CSJTAG_SHIFT_READWRITE}` |
| `exitState` | | State to end in after shift is complete (`CSEJTAG_SHIFT_DR` if no state change is desired). |
| `progressCallbackFunc` | | Progress callback function that can be used to monitor progress of JTAG target operations. The format of the progress callback function is:<br>`proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}`<br>The progress callback function must return either `$CSE_STOP` or `$CSE_CONTINUE`. If no progress callback function is necessary, a `0` should be passed into this argument position. |
| `bitCount` | | Number of bits to shift. |
| `hextdibuf` | | Data buffer that holds the data bits to be written into TDI. The least-significant bit is shifted into TDI first. |
| `-hextdimask hextdimaskval` | Optional | Specifies that a mask word `hextdimaskval` should be applied to the data buffer bits before the data is shifted into the TDI pin of the JTAG TAP. |
| `-hextdomask hextdomaskval` | | Specifies that a mask word `hextdomaskval` should be applied to the data buffer bits after the data is shifted out of the TDO pin of the JTAG TAP. |

### Returns

A buffer that is full of the data that is shifted out of the TDO pin of the JTAG TAP.

An exception is thrown if the subcommand fails.

### Example

This function shifts in 11 ones into the data register of the device at index 1, captures the 11 bits of received data, and navigates to the Run Test Idle state when finished.

```
%set hextdobuf [::chipscope::csejtag_tap shift_device_dr $handle 1
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 11 “7FF”]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db add_device_data

This subcommand is used to read device records from a file and add it to the memory-based lookup table inside the CseJtag library.

***Note:*** The file format and device record structure is the same as the idcode.lst file.

### Syntax

```
::chipscope::csejtag_db add_device_data handle filename buf bufLen
```

### Arguments

*Table 5-40:* **Arguments for Subcommand ::chipscope::csejtag_db add_device_data**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `filename` | | String containing filename from which device records are read |
| `buf` | | String containing device records in the same format and structure as the `idcode.lst` file. |
| `bufLen` | | Size of the buffer (in bytes or characters) |

### Returns

An exception is thrown if the subcommand fails.

### Example

Adding data from the file `my_idcode.lst` to the internal device database. Also, store the data record buffer and buffer size in local variables.

```
%::chipscope::csejtag_db add_device_data $handle "my_idcode.lst"
$my_idcode_buf $my_idcode_bufLen
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db lookup_device

This subcommand is used to look up a device in the database using the device IDCODE.

### Syntax

```
::chipscope::csejtag_db lookup_device handle idcode
```

### Arguments

*Table 5-41:* **Arguments for Subcommand ::chipscope::csejtag_db lookup_device**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by ::chipscope::csejtag_session create. |
| idcode | Required | IDCODE for the desired device. |

### Returns

A list in the format:

```
{deviceName irlen cmd_bypass}
```

where

`deviceName`

String containing the name of the device

`irlen`

Number of bits in the IR of the device

`cmd_bypass`

String containing the BYPASS instruction for the device (usually all ones)

An exception is thrown if the subcommand fails.

### Example

Look in the database for the device information belonging to IDCODE 01010101010101010101010101010101.

```
%set deviceInfo [::chipscope::csejtag_db lookup_device $handle
"01010101010101010101010101010101"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db get_device_name_for_idcode

This subcommand is used to get the name of a device in the database using the device `IDCODE`.

### Syntax

```
::chipscope::csejtag_db get_device_name_for_idcode handle idcode
```

### Arguments

*Table 5-42:* **Arguments for Subcommand ::chipscope::csejtag_db get_device_name_for_idcode**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `idcode` | | `IDCODE` for the desired device. |

### Returns

A string containing the device name.

An exception is thrown if the subcommand fails.

### Example

Look in the database for the name of the device belonging to `IDCODE` `01010101010101010101010101010101`.

```
%set deviceName [::chipscope::csejtag_db get_device_name_for_idcode
$handle "01010101010101010101010101010101"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db get_irlength_for_idcode

This subcommand is used to get the IR length of a device in the database using the device IDCODE.

### Syntax

```
::chipscope::csejtag_db get_irlength_for_idcode handle idcode
```

### Arguments

*Table 5-43:* **Arguments for Subcommand ::chipscope::csejtag_db get_irlength_for_idcode**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by ::chipscope::csejtag_session create. |
| idcode | | IDCODE for the desired device. |

### Returns

A string containing the size of the IR (in bits).

An exception is thrown if the subcommand fails.

### Example

Look in the database for the IR length of the device belonging to IDCODE 01010101010101010101010101010101.

```
%set irlen [::chipscope::csejtag_db get_irlength_for_idcode $handle
"01010101010101010101010101010101"]
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db parse_bsdl

This subcommand is used to extract device information from a Boundary Scan Description Language (BSDL) buffer.

### Syntax

```
::chipscope::csejtag_db parse_bsdl handle filename buf bufLen
```

### Arguments

*Table 5-44:* **Arguments for Subcommand ::chipscope::csejtag_db parse_bsdl**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `filename` | | Filename of local BSDL file (for debugging only) |
| `buf` | | Buffer containing the contents of the entire BSDL file |
| `bufLen` | | Size of the buffer `buf` (in bytes or characters) |

### Returns

A list in the format:

```
{deviceName irlen idcode cmd_bypass}
```

Where:

`deviceName`

String containing the name of the device

`irlen`

Number of bits in the `IR` of the device

`idcode`

`IDCODE` of the device

`cmd_bypass`

String containing the BYPASS instruction for the device (usually all ones)

An exception is thrown if the subcommand fails.

### Example

Extract device information from the file `device.bsd` that was placed in the buffer `bsdl_buf` of size `bsdl_bufLen`.

```
%::chipscope::csejtag_db parse_bsdl $handle "device.bsd" $bsdl_buf
$bsdl_bufLen
```

Back to list of all CseJtag Tcl Commands

## ::chipscope::csejtag_db parse_bsdl_file

This subcommand is used to extract device information from a Boundary Scan Description Language (BSDL) file.

### Syntax

```
::chipscope::csejtag_db parse_bsdl_file handle filename
```

### Arguments

*Table 5-45:* **Arguments for Subcommand ::chipscope::csejtag_db parse_bsdl_file**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `filename` | | Filename of local BSDL file. |

### Returns

A list in the format:

```
{deviceName irlen idcode cmd_bypass}
```

Where:

`deviceName`
: String containing the name of the device

`irlen`
: Number of bits in the `IR` of the device

`idcode`
: `IDCODE` of the device

`cmd_bypass`
: String containing the `BYPASS` instruction for the device (usually all ones)

An exception is thrown if the subcommand fails.

### Example

Extract device information from the file `device.bsd`.

```
%::chipscope::csejtag_db parse_bsdl_file $handle "device.bsd"
```

Back to list of all CseJtag Tcl Commands

# CseFpga Tcl Commands

The following CseFpga Commands are described in detail in this section:

- ::chipscope::csefpga_configure_device
- ::chipscope::csefpga_configure_device_with_file
- ::chipscope::csefpga_get_config_reg
- ::chipscope::csefpga_get_instruction_reg
- ::chipscope::csefpga_get_usercode
- ::chipscope::csefpga_get_user_chain_count
- ::chipscope::csefpga_is_config_supported
- ::chipscope::csefpga_is_configured
- ::chipscope::csefpga_is_sys_mon_supported
- ::chipscope::csefpga_run_sys_mon_command_sequence
- ::chipscope::csefpga_get_sys_mon_reg
- ::chipscope::csefpga_set_sys_mon_reg

## ::chipscope::csefpga_configure_device

Configures an FPGA device with the contents of a byte array containing the contents of a BIT, RBT, or MCS file.

### Syntax

```
::chipscope::csefpga_configure_device handle deviceIndex format
fileData fileDataByteLen progressFunc<optional args>
```

### Arguments

*Table 5-46:* **Arguments for Subcommand ::chipscope::csefpga_configure_device**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by ::chipscope::csejtag_session create |
| deviceIndex | Required | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| format | Required | Format of the configuration file. Valid choices are bit, rbt, and mcs. |
| fileData | Required | Contents of configuration file in an array of bytes. The bit file must be read in "binary mode". Other formats may be read in "binary mode" or "text mode". Do not remove Windows/unix end-of-line characters from fileData. |
| fileDataByteLen | Required | Length of fileData byte array (in bytes). |
| <optional args> | Optional | Enables additional device configuration options. A list of configuration options is provided in Table 5-47. |
| progressFunc | Required | Function to show progress while shifting in configuration data into the device. The format of the progress callback function is:<br>proc progressFunc (handle totalCount<br>CurrentCount progressStatus) {...}<br>The progress callback function must return either $CSE_STOP or $CSE_CONTINUE. If no progress callback function is necessary, a 0 should be passed into this argument position. |

*Table 5-47:* **Configuration Options**

| Option Name | Values | Description |
|---|---|---|
| reset_device | reset_device=true, reset_device=false | Controls whether or not device is reset during configuration. Default is "reset_device=true". |
| shutdown_sequence | shutdown_sequence= true, shutdown_sequence= false | Use the JTAG SHUTDOWN command to shut down device. For Spartan®-3 and Spartan-6 FPGA devices, this option has the same effect as "reset_device=true". Default is "shutdown_sequence=false". |

Table 5-47: **Configuration Options *(Cont'd)***

| Option Name | Values | Description |
|---|---|---|
| verify_internal_done | verify_internal_done =true, verify_internal_done =false | Read internal device DONE status after configuration from the device JTAG instruction register. Default is "verify_internal_done=true". |
| verify_external_done | verify_external_done =true, verify_external_done =false | Read external device DONE status after configuration from the configuration status register. If you tie DONE device package pins together, the DONE status is a logical AND of all device DONE pins which have their DONE device package pins tied together. Default is "verify_external_done=false". |
| verify_crc | verify_crc=true, verify_crc=false | Read CRC status after configuration from the device configuration status register. Default is "verify_crc=false". |
| use_assigned_config_data | use_assigned_config_data=true, use_assigned_config_data=false | Use the configuration and/or mask data provided by ::chipscope::csefpga_assign_config_data_to_device or ::chipscope::csefpga_assign_config_data_file_to_device. Default is "use_assigned_config_data=false". |

## Returns

A bitfield containing the resulting status of the configuration. The bitfield can be logically AND'ed with one or more of the following values to check the corresponding status information:

```
$CSE_INTERNAL_DONE_HIGH_STATUS
$CSE_EXTERNAL_DONE_HIGH_STATUS
$CSE_CRC_ERROR_STATUS
$CSE_BITSTREAM_READ_ENABLED
$CSE_BITSTREAM_WRITE_ENABLED
```

An exception is thrown if the command fails.

## Example

Configure the third device in the JTAG chain with the file mydesign.bit.

```
%set filename "mydesign.bit"
%set fp [open $filename r]
%fconfigure $fp -translation binary -blocking 1
%set fileData [read $fp]
%close $fp
%set configStatus [::chipscope::csefpga_configure_device $handle 2
"bit" $CSE_DEFAULT_OPTIONS $fileData [file size $filename]
"progressCallBack"]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_configure_device_with_file

Configures an FPGA device with the contents of a .bit, .rbt or .mcs file.

### Syntax

```
::chipscope::csefpga_configure_device_with_file handle deviceIndex
filename <optional args> progressFunc
```

### Arguments

*Table 5-48:* **Arguments for Subcommand ::chipscope::csefpga_configure_device_with_file**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | Required | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `filename` | Required | Filename of the `.bit`, `.rbt`, or `.mcs` configuration file. |
| `<optional args>` | Optional | Enables additional device configuration options. A list of configuration options is provided in Table 5-47. |
| `progressFunc` | Required | Function to show progress while shifting in configuration data into the device. The format of the progress callback function is: `proc progressFunc (handle totalCount` `CurrentCount progressStatus) {...}` The progress callback function must return either `$CSE_STOP or $CSE_CONTINUE`. If no progress callback function is necessary, a 0 should be passed into this argument position. |

### Returns

A bitfield containing the resulting status of the configuration. The bitfield can be logically AND'ed with one or more of the following values to check the corresponding status information:

```
$CSE_INTERNAL_DONE_HIGH_STATUS
$CSE_EXTERNAL_DONE_HIGH_STATUS
$CSE_CRC_ERROR_STATUS
$CSE_BITSTREAM_READ_ENABLED
$CSE_BITSTREAM_WRITE_ENABLED
```

An exception is thrown if the command fails.

### Example

Configure the third device in the JTAG chain with the file mydesign.bit.

```
%set fileName "mydesign.bit"
%set configStatus [::chipscope::csefpga_configure_device $handle 2
$fileName $CSE_DEFAULT_OPTIONS "progressCallBack"]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_get_config_reg

Reads the configuration register bits of the target FPGA device.

### Syntax

```
::chipscope::csefpga_get_config_reg handle deviceIndex bitCount
```

### Arguments

*Table 5-49:* **Arguments for Subcommand ::chipscope::csefpga_get_config_reg**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `bitCount` | | Length of the configuration register (in bits) |

### Returns

A list in the format:

`{hexReg bitNameBuf}`

where:

`hexReg`

String containing the register value (in hexadecimal).

`bitNameBuf`

A comma separated list of strings that represent configuration register bit names.

An exception is thrown if the command fails.

### Example

Read the contents of the configuration register of the third device in the JTAG chain.

```
%set ConfigReg [csefpga_get_config_reg $handle 2 $DeviceBitCount]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_get_instruction_reg

Reads the instruction register of the target FPGA device and format the configuration-specific status bits.

### Syntax

```
::chipscope::csefpga_get_instruction_reg handle deviceIndex bitCount
```

### Arguments

*Table 5-50:* **Arguments for Subcommand ::chipscope::csefpga_get_instruction_reg**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `bitCount` | | Length of the configuration register (in bits) |

### Returns

A list in the format:

`{hexReg bitNameBuf}`

where:

`hexReg`

String containing the register value (in hexadecimal).

`bitNameBuf`

A comma separated list of strings that represent configuration register bit names.

An exception is thrown if the command fails.

### Example

Read the contents of the configuration register of the third device in the JTAG chain.

```
%set InstReg [csefpga_get_instruction_reg $handle 2 $DeviceBitCount]
```

Back to list of all CseFpga Tcl Commands

# ::chipscope::csefpga_get_usercode

Reads the `USERCODE` register of the target FPGA device.

## Syntax

```
::chipscope::csefpga_get_usercode handle deviceIndex
```

## Arguments

*Table 5-51:* **Arguments for Subcommand ::chipscope::csefpga_get_usercode**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |

## Returns

The contents of the `USERCODE` register (in hexadecimal).

An exception is thrown if the command fails.

## Example

Read the contents of the `USERCODE` register of the third device in the JTAG chain.

```
%set usercode [csefpga_get_usercode $handle 2]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_get_user_chain_count

Determines the number of `USER` scan chain registers in the target FPGA device.

### Syntax

```
::chipscope::csefpga_get_user_chain_count handle idcode
```

### Arguments

*Table 5-52:* **Arguments for Subcommand ::chipscope::csefpga_get_user_chain_count**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `idcode` | | `IDCODE` for the desired device. |

### Returns

The number of `USER` scan chain registers in the device (0 if the device does not have any `USER` scan chain registers).

An exception is thrown if the command fails.

### Example

Get the number of USER scan chain registers supported by the device to which $idcode refers.

```
%set numUserRegs [csefpga_get_user_chain_count $handle $idcode]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_is_config_supported

Tests if configuration is supported for the target FPGA device.

### Syntax

```
::chipscope::csefpga_is_config_supported handle idcode
```

### Arguments

*Table 5-53:* **Arguments for Subcommand ::chipscope::csefpga_is_config_supported**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by ::chipscope::csejtag_session create. |
| idcode | | IDCODE for the desired device. |

### Returns

Returns 1 if configuration of the device referred to by idcode is supported by the csefpga_configure_device command, otherwise returns 0.

An exception is thrown if the command fails.

### Example

Determine if the device referred to by $idcode can be configured.

```
%set isConfigurable [csefpga_is_config_supported $handle $idcode]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_is_configured

Returns the configuration status of an FPGA device.

### Syntax

```
::chipscope::csefpga_is_configured handle deviceIndex
```

### Arguments

*Table 5-54:* **Arguments for Subcommand ::chipscope::csefpga_is_configured**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |

### Returns

Returns 1 if the device referred to by `deviceIndex` is configured, otherwise returns 0.

An exception is thrown if the command fails.

### Example

Get the configuration status of the third device in the JTAG chain.

```
%set isConfigured [csefpga_is_configured $handle 2]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_is_sys_mon_supported

Tests if a System Monitor commands are supported for the target FPGA device.

### Syntax

```
::chipscope::csefpga_is_sys_mon_supported handle idcode
```

### Arguments

*Table 5-55:* **Arguments for Subcommand ::chipscope::csefpga_is_sys_mon_supported**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `idcode` | | `IDCODE` for the desired device. |

### Returns

Returns 1 if the device referred to by `idcode` contains a System Monitor block, otherwise returns 0.

An exception is thrown if the command fails.

### Example

Determine if the device referred to by `$idcode` contains a System Monitor block

```
%set hasSysMon [csefpga_is_sys_mon_supported $handle $idcode]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_run_sys_mon_command_sequence

Executes a sequence of reads and writes from/to System Monitor registers.

### Syntax

```
::chipscope::csefpga_run_sys_mon_command_sequence handle deviceIndex
[list hexAddresses...] [list hexInData...] [list setModes...]
commandCount
```

### Arguments

*Table 5-56:* **Arguments for Subcommand ::chipscope::csefpga_run_sys_mon_command_sequence**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `[list hexAddresses...]` | | List of System Monitor DRP register addresses (in hexadecimal) to be accessed. This number of elements in this list is dictated by `commandCount`. |
| `[list hexInData...]` | | List of data to be written to System Monitor DRP registers specified by the list of `hexAddresses`. The `hexInData` element is only written if the corresponding `setModes` element is non-zero. This number of elements in this list is dictated by `commandCount`. |
| `[list setModes...]` | | List of setMode flags. Zero indicates read data from register, non-zero indicates write. This number of elements in this list is dictated by `commandCount`. |
| `commandCount` | | Number of commands (and number of elements in each list argument) |

### Returns

A list containing commandCount elements each of which represents a register read value. Note that the data element is valid if the corresponding `setModes` element is zero.

An exception is thrown if the command fails.

### Example

For the second device in the JTAG chain, write `0x55AA` to the system monitor DRP register address `0x10` and read from DRP register address `0x11`.

```
%set hexOutData [csefpga_run_sys_mon_command_sequence $handle 1 [list
10 11] [list 55AA 0000] [list 1 0] 2]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_get_sys_mon_reg

Reads from a System Monitor register.

### Syntax

```
::chipscope::csefpga_get_sys_mon_reg handle deviceIndex hexAddress
```

### Arguments

*Table 5-57:* **Arguments for Subcommand ::chipscope::csefpga_get_sys_mon_reg**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `hexAddress` | | System Monitor DRP register address (in hexadecimal) to be read from. |

### Returns

A data value (in hexadecimal) read from the System Monitor DRP register at address `hexAddress`.

An exception is thrown if the command fails.

### Example

For the second device in the JTAG chain, read the System Monitor register at address `0x07`.

```
%set hexOutData [csefpga_get_sys_mon_reg $handle 1 7]
```

Back to list of all CseFpga Tcl Commands

## ::chipscope::csefpga_set_sys_mon_reg

Writes to a System Monitor register.

### Syntax

```
::chipscope::csefpga_set_sys_mon_reg handle deviceIndex hexAddress
hexInData
```

### Arguments

*Table 5-58:* **Arguments for Subcommand ::chipscope::csefpga_set_sys_mon_reg**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to *n*-1) in the *n*-length JTAG chain. |
| `hexAddress` | | System Monitor DRP register address (in hexadecimal) to be written to. |
| `hexInData` | | Data value (in hexadecimal) to be written to the System Monitor DRP register. |

### Returns

An exception is thrown if the command fails.

### Example

For the second device in the JTAG chain, write `0xABCD` to the System Monitor register at address `0x09`.

```
%csefpga_set_sys_mon_reg $handle 1 9 abcd
```

Back to list of all CseFpga Tcl Commands

# CseCore Tcl Commands

The following CseFpga Commands are described in detail in this section:

- ::chipscope::csecore_get_core_count
- ::chipscope::csecore_get_core_status
- ::chipscope::csecore_is_cores_supported

## ::chipscope::csecore_get_core_count

Gets the number of cores attached to an ICON core that is in the target FPGA device and attached to a particular USER scan chain register.

### Syntax

```
::chipscope::csecore_get_core_count handle deviceIndex userRegNumber
```

### Arguments

*Table 5-59:* **Arguments for Subcommand ::chipscope::csecore_get_core_count**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `deviceIndex` | | Device index (0 to $n$-1) in the $n$-length JTAG chain. |
| `userRegNumber` | | BSCAN block `USER` register number (starting with 1) |

### Returns

The number of cores.

An exception is thrown if the command fails.

### Example

Get the number of cores attached to the ICON core in the `USER3` register of the third device in the JTAG chain.

```
%set coreCount [csecore_get_core_count $handle 2 3]
```

Back to list of all CseCore Tcl Commands

## ::chipscope::csecore_get_core_status

Retrieves the static status word from the target ChipScope Pro core.

### Syntax

```
::chipscope::csecore_get_core_status handle [list deviceIndex
userRegNumber coreIndex] bitCount
```

### Arguments

*Table 5-60:* **Arguments for Subcommand ::chipscope::csecore_get_core_status**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |
| `bitCount` | | Length of status word (in number of bits) |

### Returns

A nested list. The outer list contains two elements: an inner list and a string representing the core status (in hexadecimal). The inner list contains the following elements:

| Element | Description |
|---|---|
| `manufacturerId` | Manufacturer ID (integer) |
| `coreType` | Core type (integer) |
| `coreMajorVersion` | Core major version (integer) |
| `coreMinorVersion` | Core minor version (integer) |
| `coreRevision` | Core revision (integer) |

***Note:*** An exception is thrown if the command fails.

### Example

Get core status of first core connected to ICON core inside fourth device on the second `USER` register.

```
%set coreRef [list 3 2 0]
%set coreStatus [csecore_get_core_status $handle $coreRef]
```

Back to list of all CseCore Tcl Commands

## ::chipscope::csecore_is_cores_supported

Tests if a the target FPGA device supports ChipScope Pro cores.

### Syntax

```
::chipscope::csecore_is_cores_supported handle idcode
```

### Arguments

*Table 5-61:* **Arguments for Subcommand ::chipscope::csecore_is_cores_supported**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create`. |
| `idcode` | | `IDCODE` for the desired device. |

### Returns

Returns 1 if the device referred to by `idcode` supports ChipScope Pro cores, otherwise returns 0.

An exception is thrown if the command fails.

### Example

Determine if device referred to by `$idcode` supports ChipScope Pro cores

```
%set supportsCores [csecore_is_cores_supported $handle $idcode]
```

Back to list of all CseCore Tcl Commands

# CseVIO Tcl Commands

The following CseFpga Commands are described in detail in this section:

- ::chipscope::csevio_get_core_info
- ::chipscope::csevio_is_vio_core
- ::chipscope::csevio_init_core
- ::chipscope::csevio_terminate_core
- ::chipscope::csevio_define_signal
- ::chipscope::csevio_define_bus
- ::chipscope::csevio_undefine_name
- ::chipscope::csevio_write_values
- ::chipscope::csevio_read_values

## ::chipscope::csevio_get_core_info

Reads the status word from the target VIO core.

### Syntax

```
::chipscope::csevio_get_core_info handle [list deviceIndex
userRegNumber coreIndex] coreInfoTclArray
```

### Arguments

*Table 5-62:* **Arguments for Subcommand ::chipscope::csevio_get_core_info**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to `ICON` has index 0. |
| `coreInfoTclArray` | | Tcl array name. After the command is executed successfully, the array contains information described in the Returns section below. |

### Returns

Returns core information through the coreInfoTclArray argument including the following elements:

| Element | Description |
|---|---|
| `manufacturerId` | Manufacturer ID (integer) |
| `$CSEVIO_MANUFACTURER_ID` | Manufacturer's ID, 1 for Xilinx. |
| `$CSEVIO_CORE_TYPE` | Core Type field, different for each ChipScope Core, VIO should be 9. |
| `$CSEVIO_CORE_MAJOR_VERSION` | Major release version. |
| `$CSEVIO_CORE_MINOR_VERSION` | Minor release version. |
| `$CSEVIO_CORE_REVISION` | Revision. |
| `$CSEVIO_CG_MAJOR_VERSION` | CoreGen specific field (for cores generated using 10.1 and later). |
| `$CSEVIO_CG_MINOR_VERSION` | CoreGen specific field (for cores generated using 10.1 and later). |
| `$CSEVIO_CG_MINOR_VERSION_ALPHA` | CoreGen specific field (for cores generated using 10.1 and later). |
| `$CSEVIO_ASYNC_INPUT_COUNT` | Number of Asynchronous Inputs signals used. |
| `$CSEVIO_SYNC_INPUT_COUNT` | Number of Synchronous Inputs signals used. |
| `$CSEVIO_ASYNC_OUTPUT_COUNT` | Number of Asynchronous Outputs signals used. |
| `$CSEVIO_SYNC_OUTPUT_COUNT` | Number of Synchronous Outputs signals used. |

***Note:*** An exception is thrown if the command fails.

### Example

Get core info of the VIO core that is connected to the first control port of the ICON core inside fourth device on second USER register. Print out the number of asynchronous inputs that the VIO core has.

```
%set coreRef [list 3 2 0]
%csevio_get_core_info $handle $coreRef coreInfoTclArray
%puts stdout "$coreInfoTclArray($CSEVIO_ASYNC_INPUT_COUNT)"
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_is_vio_core

Determines whether the target core is a VIO core.

### Syntax

```
::chipscope::csevio_is_vio_core handle [list deviceIndex userRegNumber
coreIndex]
```

### Arguments

*Table 5-63:* **Arguments for Subcommand ::chipscope::csevio_is_vio_core**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to `ICON` has index 0. |

### Returns

Returns 1 if the core is a VIO core, otherwise returns 0.

An exception is thrown if the command fails.

### Example

Determine if the first core connected to ICON core inside fourth device on second USER register is a VIO core

```
%set coreRef [list 3 2 0]
%set isVIO [csevio_is_vio_core $handle $coreRef]
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_init_core

Initializes global variables associated with the target VIO core.

### Syntax

```
::chipscope::csevio_init_core handle [list deviceIndex userRegNumber
coreIndex]
```

### Arguments

*Table 5-64:* **Arguments for Subcommand ::chipscope::csevio_init_core**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | Required | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to `ICON` has index 0. |

### Returns

An exception is thrown if the command fails.

### Example

Initialize the VIO core connected to ICON core inside fourth device on second `USER` register is a VIO core

```
%set coreRef [list 3 2 0]
%csevio_init_core $handle $coreRef
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_terminate_core

Removes global variables and releases memory associated with the target VIO core.

### Syntax

```
::chipscope::csevio_terminate_core handle [list deviceIndex
userRegNumber coreIndex]
```

### Arguments

*Table 5-65:* **Arguments for Subcommand ::chipscope::csevio_terminate_core**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |

### Returns

An exception is thrown if the command fails.

### Example

Terminates the VIO core connected to ICON core inside fourth device on second `USER` register is a VIO core

```
%set coreRef [list 3 2 0]
%csevio_terminate_core $handle $coreRef
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_define_signal

Defines a name for a specified VIO signal bit. Requires that `csevio_init_core` be called first.

### Syntax

```
::chipscope::csevio_define_signal handle [list deviceIndex
userRegNumber coreIndex] name flags bitIndex
```

### Arguments

*Table 5-66:* **Arguments for Subcommand ::chipscope::csevio_define_signal**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |
| `name` | | Name to be given to the signal. All input signal names must be unique with respect to other input signals. All output signal names must be unique with respect to other output signals. |
| `flags` | | Flags used to determine the port type to which the signal belongs. Possible flag values include:<br>• `$CSEVIO_SYNC_OUTPUT`<br>• `$CSEVIO_SYNC_INPUT`<br>• `$CSEVIO_ASYNC_OUTPUT`<br>• `$CSEVIO_ASYNC_INPUT` |
| `bitIndex` | | Bit index into the port. Used to determine what port signal to assign to the name. |

### Returns

An exception is thrown if the command fails.

### Example

For the VIO core connected to ICON core inside fourth device on second `USER` register, define a signal called `status_bit` that is assigned to bit 0 of the `ASYNC_INPUT` port:

```
%set coreRef [list 3 2 0]
%set csevio_define_signal $handle $coreRef "status_bit"
$CSEVIO_ASYNC_INPUT 0
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_define_bus

Defines a name for a grouping of VIO signal bits (called a “bus”). Requires that csevio_init_core be called first.

### Syntax

```
::chipscope::csevio_define_bus handle [list deviceIndex userRegNumber
coreIndex] name flags bitIndexArray arrayLen
```

### Arguments

*Table 5-67:* **Arguments for Subcommand ::chipscope::csevio_define_bus**

| Argument | Type | Description |
|---|---|---|
| handle | Required | Handle to the session that is returned by ::chipscope::csejtag_session create |
| [list deviceIndex userRegNumber coreIndex] | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block USER register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |
| name | | Name to be given to the bus. All input bus names must be unique with respect to other input buses. All output bus names must be unique with respect to other output buses. |
| flags | | Flags used to determine the port type to which the bus belongs. Possible flag values include:<br>• $CSEVIO_SYNC_OUTPUT<br>• $CSEVIO_SYNC_INPUT<br>• $CSEVIO_ASYNC_OUTPUT<br>• $CSEVIO_ASYNC_INPUT |
| [list bitIndicies...] | | List containing the bit indices of the signals in the bus. Left-most element in the list is the LSB. |

### Returns

An exception is thrown if the command fails.

### Example

For the VIO core connected to ICON core inside fourth device on second USER register, define a bus called “control_bus” that is assigned to bits 3:0 of the SYNC_OUTPUT port:

```
%set coreRef [list 3 2 0]
%set csevio_define_bus $handle $coreRef "control_bus"
$CSEVIO_SYNC_OUTPUT [list 0 1 2 3]
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_undefine_name

Removes a VIO signal/bus name and all associated information.

### Syntax

```
::chipscope::csevio_undefine_name handle [list deviceIndex
userRegNumber coreIndex] name flags
```

### Arguments

*Table 5-68:* **Arguments for Subcommand ::chipscope::csevio_undefine_name**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |
| `name` | | Name of the signal or bus to be removed. |
| `flags` | | Flags used to determine the port type to which the signal or bus belongs. Possible flag values include:<br>• `$CSEVIO_SYNC_OUTPUT`<br>• `$CSEVIO_SYNC_INPUT`<br>• `$CSEVIO_ASYNC_OUTPUT`<br>• `$CSEVIO_ASYNC_INPUT` |

### Returns

An exception is thrown if the command fails.

### Example

For the VIO core connected to ICON core inside fourth device on second `USER` register, undefine a bus called `control_bus` that is assigned to bits of the `SYNC_OUTPUT` port:

```
%set coreRef [list 3 2 0]
%set csevio_undefine_name $handle $coreRef “control_bus”
$CSEVIO_SYNC_OUTPUT
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_write_values

Writes values to the specified signal/bus of the target VIO core.

### Syntax

```
::chipscope::csevio_write_values handle [list deviceIndex
userRegNumber coreIndex] outputTclArray
```

### Arguments

*Table 5-69:* **Arguments for Subcommand ::chipscope::csevio_write_values**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 `to` *n*-1) in the *n*-length JTAG chain<br>• BSCAN block USER register number (starting with 1)<br>• Index for core unit. First core unit connected to ICON has index 0. |
| `outputTclArray` | | Name of a Tcl array. The index into the array is the name of an output signal or bus defined by csevio_define_signal or csevio_define_bus, respectively. For signals that belong to the `SYNC_OUTPUT` port, the postfix "`.pulsetrain`" can be appended to the name and a string hexadecimal values can be specified (LSB is right-most character). In order to use the pulse train, 16 values must be passed in, with the first value sent in to the right. Each value must be byte aligned. This means that a single signal requires two hexadecimal characters for each value. If the value is greater than 8 bits, then two additional characters are required per value, and so on. Each element of the array must be unset manually after this command is called to reuse the array. |

### Returns

An exception is thrown if the command fails.

### Example

Assumptions for these examples:

- `coreRef` has already been set to the VIO core
- A signal called "`reset`" is defined as a bit in the `SYNC_OUTPUT` port
- A bus called "`instruction`" is defined as an 8-bit bus in the `ASYNC_OUTPUT` port

1. Set the `reset` signal to 0 and set the `instruction` bus to `FF`

```
%set outputTclArray(reset) 0
%set outputTclArray(instruction) FF
%csevio_write_values $handle $coreRef outputTclArray
```

2. Send a single clock cycle pulse of 1 followed by 0s to the `reset` signal

```
%set outputTclArray(reset.pulsetrain) 00000000000000000000000000000001
%csevio_write_values $handle $coreRef outputTclArray
```

Back to list of all CseVIO Tcl Commands

## ::chipscope::csevio_read_values

Reads values from the specified signal/bus of the target VIO core.

### Syntax

```
::chipscope::csevio_read_values handle [list deviceIndex userRegNumber
coreIndex] inputTclArray
```

### Arguments

*Table 5-70:* **Arguments for Subcommand ::chipscope::csevio_read_values**

| Argument | Type | Description |
|---|---|---|
| `handle` | Required | Handle to the session that is returned by `::chipscope::csejtag_session create` |
| `[list deviceIndex userRegNumber coreIndex]` | | A list containing three elements:<br>• Device index (0 to *n*-1) in the *n*-length JTAG chain<br>• BSCAN block `USER` register number (starting with 1)<br>• Index for core unit. First core unit connected to `ICON` has index 0. |
| `inputTclArray` | | Name of a Tcl array. The index into the array is the name of an input signal or bus defined by `csevio_define_signal` or `csevio_define_bus`, respectively. Special postfixes can be used to specify various states of the input signal or buses:<br>• `".value"` specifies the signal/bus value (same as no postfix)<br>• `".activity_up"` specifies asynchronous low-to-high activity<br>• `".activity_down"` specifies asynchronous high-to-low activity<br>• `".sync_activity_up"` specifies the synchronous low-to-high activity (only valid for `SYNC_INPUT` signals/buses)<br>• `".sync_activity_down"` specifies the synchronous high-to-low activity (only valid for `SYNC_INPUT` signals/buses) |

### Returns

An exception is thrown if the command fails.

### Example

Assumptions for this example:

- `coreRef` has already been set to the VIO core.
- A signal called "status" is defined as a bit in the `SYNC_INPUT` port.
- A bus called "data_bus" is defined as an 8-bit bus in the `ASYNC_INPUT` port.

1. Get the values of "`status`" and "`data_bus`" and print them to stdout

```
%csevio_read_values $handle $coreRef inputTclArray
```

```
% puts stdout “status = $inputTclArray(status.value)”
% puts stdout “data_bus = $inputTclArray(data_bus)”
```

2. Get the various activity states of the “status” signal and print them to stdout:

```
%csevio_read_values $handle $coreRef inputTclArray
% puts stdout “up = $inputTclArray(status.activity_up)”
% puts stdout “dn = $inputTclArray(status.activity_down)”
% puts stdout “sup = $inputTclArray(status.sync_activity_up)”
% puts stdout “sdn = $inputTclArray(status.sync_activity_down)”
```

Back to list of all CseVIO Tcl Commands

# CSE/Tcl Examples

The ChipScope Pro installation includes an example Tcl script that uses the CseJtag Tcl interface. This example opens a Xilinx Parallel cable or Xilinx Platform USB cable and scans the JTAG chain and returns information about the devices found in the chain. The example script is located in the following location:

```
<XILINX_ISE_INSTALL>\cse\tcl\csejtag_example1.tcl
```

The script can be run in the Tcl shell (xtclsh) that is included with Xilinx ISE Design Suite or in the ActiveTcl 8.4 Tcl shell (tclsh) from ActiveState Software Inc. ([See Reference 24, p. 212]). To run the Tcl example in a command line shell, change to the directory where `csejtag_example1.tcl` is located (see above). Next, follow these instructions for the particular operating system:

- On 32-bit Windows operating systems:
  - To use a Xilinx Parallel Cable, type:

    ```
    <XILINX_ISE_INSTALL>\bin\nt\xtclsh csejtag_example1.tcl -par
    ```
  - To use a Xilinx Platform Cable USB, type:

    ```
    <XILINX_ISE_INSTALL>\bin\nt\xtclsh csejtag_example1.tcl -usb
    ```
- On 64-bit Windows operating systems:
  - To use a Xilinx Parallel Cable, type:

    ```
    <XILINX_ISE_INSTALL>\bin\nt64\xtclsh csejtag_example1.tcl -par
    ```
  - To use a Xilinx Platform Cable USB, type:

    ```
    <XILINX_ISE_INSTALL>\bin\nt64\xtclsh csejtag_example1.tcl -usb
    ```
- On 32-bit Linux operating systems:
  - To use a Xilinx Parallel Cable, type:

    ```
    <XILINX_ISE_INSTALL>/bin/lin/xtclsh csejtag_example1.tcl -par
    ```
  - To use a Xilinx Platform Cable USB, type:

    ```
    <XILINX_ISE_INSTALL>/bin/lin/xtclsh csejtag_example1.tcl -usb
    ```
- On 64-bit Linux operating systems:
  - To use a Xilinx Parallel Cable, type:

    ```
    <XILINX_ISE_INSTALL>/bin/lin64/xtclsh csejtag_example1.tcl -par
    ```
  - To use a Xilinx Platform Cable USB, type:

    ```
    <XILINX_ISE_INSTALL>/bin/lin64/xtclsh csejtag_example1.tcl -usb
    ```

Other example Tcl scripts (such as the CSE VIO example Tcl script called csevio_example1.tcl) can be found in the same directory as csejtag_example1.tcl. These scripts offer examples of other CSE/Tcl function calls.

Appendix A

# ChipScope Pro Tools Troubleshooting Guide

## Overview

This appendix provides instructions to validate your ChipScope™ Pro tools install and ISE® software integration. The purpose of this appendix is to assist you with common errors and issues you might face when using the ChipScope Pro tools. In addition, this appendix provides a general practices for troubleshooting your ChipScope Pro and Xilinx® JTAG-based programming cable installation. Each problem description has one or more of the following sections:

- Issue(s): This section lists ways to verify whether the problem described is the same as the problem at hand generally in the form of an error message or warning.
- Solution(s) or work-around(s): This section lists ways to resolve the problem at hand.

# ChipScope Pro Tools Installation Troubleshooting

Common Error Messages/problems that point to issues with a ChipScope Pro tools install are shown in Table A-1.

*Table A-1:* **Troubleshooting ChipScope Pro Tools Installation Issues**

<table>
<tr><th>Issue(s)</th><th>Solution(s) or Work-Around(s)</th></tr>
<tr><td>On either Windows or Linux systems, double clicking on a <filename>.cdc file in the ISE tool does not launch the inserter and the following error appears in the console:<br><br>ERROR: Unable to find the Chipscope Exe at NotHere/inserterlauncher.exe</td>
<td rowspan="4">1. Check that your environment is set up correctly by ensuring that the following three parameters are set correctly:<br>– The CHIPSCOPE environment variable needs to be set to a valid Xilinx ChipScope Pro tools installation to operate correctly.<br>- On Windows systems, select Start > Settings > Control Panel > System > Advanced > System Variables. Set the environment variable CHIPSCOPE to point to your installation. Typically this would be C:\Xilinx\<version number>\ChipScope<br>- On Linux systems, set and environment variable CHIPSCOPE to point to your installation. For example: setenv CHIPSCOPE /tools/xilinx/<version number>/chipscope<br>– The XILINX environment variable needs to be set to a valid Xilinx ISE tool installation directory for ChipScope Pro tools to operate correctly.<br>- On Windows systems, select Start > Settings > Control Panel > System > Advanced > System Variables. Set the environment variable XILINX to point to your installation. Typically this would be C:\Xilinx\<version number>\ISE<br>- On Linux systems, at the completion of the installation process, the installation program creates an environment variables file for you. Go to your Xilinx ISE installation directory and source <settings file> (where <settings file> is settings32.sh, settings32.csh, settings64.sh, or settings64.csh, depending on your OS and shell type).<br>2. From inside the ISE tool, check and make sure that the Edit > Preferences > ISE General > Integrated Tools is set to <install>/bin/<platform><br>– <install> is the ChipScope Pro tools installation location<br>– <platform> is lin, lin64, nt, nt64 for 32-bit Linux OS, 64-bit Linux OS, 32-bit Windows OS, and 64-bit Windows OS, respectively. Note that <platform> for ChipScope Pro tools must match that of the ISE tools.</td></tr>
<tr><td>On Windows systems, when running the inserter from the Start menu you see a splash screen but the ChipScope Pro Core Inserter tool does not start up.</td></tr>
<tr><td>On Windows systems, when running inserter from the command line you see a pop-up dialog box with the following message:<br><br>inserter.exe - entry point not found</td></tr>
<tr><td>On Linux systems, after running the inserter.sh script to launch the core inserter you see the following errors:<br><br>ERROR: Unable to locate Xilinx ISE <release number> tools in path!<br><br>ERROR: You must set the CHIPSCOPE environment variable before running this tool</td></tr>
</table>

# Xilinx JTAG Programming Cable Troubleshooting

This section describes how to determine if your Xilinx JTAG programming cable is connected correctly and how to troubleshoot common Xilinx JTAG (Joint Test Action Group, IEEE standard) cable connection issues. Here is a list of issues that are covered in this section:

- For verifying correct cable connections, see Table A-3, page 197.
- For troubleshooting issues related to “INFO: Cable connection failed” messages, see Table A-4, page 197.
- For troubleshooting issues related to “ERROR:iMPACT:2246 - A reference voltage has not been detected...” messages, see Table A-5, page 199.
- For troubleshooting issues related to “ERROR: No devices detected while scanning the JTAG chain”, “ERROR: Failed detecting JTAG device chain”, or “ERROR: Opened Xilinx Platform USB Cable but failed to detect JTAG Chain” messages, see Table A-6, page 200.
- For troubleshooting issues related to “ERROR: Socket Open Failed. localhost/127.0.0.1:50001” messages, see Table A-7, page 202.

Table A-2: **Verifying Correct Platform Cable USB Connection**

<table>
<tr><th>Issue</th><th>Solution or Workaround</th></tr>
<tr><td>How can I tell if I am connecting to the Platform Cable USB correctly?</td><td>1. Start the ChipScope Pro Analyzer tool.<br>2. Select the JTAG Chain menu option.<br>3. Select the Platform Cable USB option.<br>4. Make sure the Speed and Port options are set correctly.<br>5. Click OK.<br>6. Look for messages similar to the following:<br><pre>COMMAND: open_platform_usb_cable FREQUENCY=3000000 PORT=USB21
INFO: Started ChipScope host (localhost:50001)
INFO: Opened socket connection: localhost 50001
localhost/127.0.0.1
INFO: Connecting to cable (Usb Port - USB21).
INFO: Checking cable driver.
INFO:  Driver file xusbdfwu.sys found.
INFO:  Driver version: src=1027, dest=1027.
INFO:  Driver windrvr6.sys version = 8.1.1.0.
INFO:  WinDriver v8.11 Jungo (c) 1997 - 2006 Build Date: Oct 16
2006 X86 32bit SYS 12:35:07, version = 811.
INFO:  Cable PID = 0008.
INFO:  Max current requested during enumeration is 300 mA.
INFO: Type = 0x0005.
INFO:  Cable Type = 3, Revision = 0.
INFO:  Setting cable speed to 3 MHz.
INFO: Cable connection established.
INFO: Firmware version = 2301.
INFO: File version of
C:/Xilinx/11.1/ChipScope/xilinx/data/xusb_xp2.hex = 2401.
INFO: Firmware hex file version = 2401.
INFO: Downloading
C:/Xilinx/11.1/ChipScope/xilinx/data/xusb_xp2.hex.
INFO: Downloaded firmware version = 2401.
INFO: PLD file version = 200Dh.
INFO:  PLD version = 200Dh.</pre></td></tr>
</table>

*Table A-3:* **Verifying Correct JTAG Parallel Cable IV Connection**

<table>
<tr><th>Issue(s)</th><th>Solution(s) or Work-Around(s)</th></tr>
<tr><td>How can I tell if I am connecting to the Parallel Cable IV correctly?</td><td>1. Start the ChipScope Pro Analyzer tool.<br>2. Select the JTAG Chain menu option.<br>3. Select the Platform Cable USB option.<br>4. Make sure the Speed and Port options are set correctly.<br>5. Click OK.<br>6. Look for messages similar to the following:<br><pre>COMMAND: open_parallel_cable FREQUENCY=5000000 PORT=LPT1
INFO: Started ChipScope host (localhost:50001)
INFO: Opened socket connection: localhost 50001 localhost/127.0.0.1
INFO: Connecting to cable (Parallel Port - LPT1).
INFO: Checking cable driver.
INFO:  Driver windrvr6.sys version = 8.1.1.0.
INFO:  WinDriver v8.11 Jungo (c) 1997 - 2006 Build Date: Oct 16 2006
X86 32bit SYS 12:35:07, version = 811.
INFO:  LPT base address = 0378h.
INFO:  ECP base address = 0778h.
INFO:  ECP hardware is detected.
INFO: Cable connection established.
INFO: Connecting to cable (Parallel Port - LPT1) in ECP mode.
INFO: Checking cable driver.
INFO:  Driver xpc4drvr.sys version = 1.0.4.0.
INFO:  LPT base address = 0378h.
INFO:  Cable Type = 1, Revision = 10.
INFO:  Setting cable speed to 5 MHz.
INFO: Cable connection established.</pre></td></tr>
</table>

*Table A-4:* **Troubleshooting Platform Cable USB Connection Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. On attempting to connect to the cable you see the following messages in the console.<br>INFO: Cable connection failed.<br>ERROR: Failed to open Xilinx Platform USB Cable. See message(s) above.<br>This issue occurs when your cable is installed but not connected. If the cable is connected and you still see this issue it might be damaged or require a firmware update. | You should first check to see if the cable is connected. Go to Issue #2. |
| 2. Is your cable is plugged into the appropriate USB port? | If NO: Connect the cable and attempt to connect in the ChipScope Pro Analyzer tool.<br>If YES: Go to Issue #3. |

*Table A-4:* **Troubleshooting Platform Cable USB Connection Issues *(Cont'd)***

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 3. Has the correct cable been specified at connection? | If NO or NOT SURE: In the ChipScope Pro Analyzer tool, check that the correct cable has been selected in the JTAG Chain menu. Select the correct cable and retry connection.<br>If YES: Go to Issue #4. |
| 4. Are the Server Host Settings set correctly? | If NO or NOT SURE: In the ChipScope Pro Analyzer tool, select JTAG Chain > Server Host Settings. In the dialog box, check to make sure that the correct server hostname (or IP address) and port are used. For connecting to cables on the local system, use localhost:50001.<br>If YES: Go to Issue #5. |
| 5. If you are trying to connect to a particular Platform Cable USB, is the correct Port setting selected? | If NO or NOT SURE: In the ChipScope Pro Analyzer tool, select JTAG Chain > Platform Cable USB. In the dialog box, check to make sure that the Port setting is set to the correct port enumeration for the desired cable. For instance, for the first enumerated cable, select port USB21.<br>If YES: Go to Issue #6. |
| 6. A firmware update might be required. | To update the firmware:<br>1. Open a DOS shell and set the environment variable by entering:<br>SET XIL_IMPACT_ENV_USB2_FORCE_CPLD_UPDATE=TRUE<br>2. Start iMPACT by entering impact in the DOS shell.<br>3. Select Xilinx USB Cable from the Cable Communication Setup dialog box and wait for the update to be completed.<br>4. Exit iMPACT.<br>5. Clear the environment variable in the DOS shell by entering:<br>SET XIL_IMPACT_ENV_USB2_FORCE_CPLD_UPDATE=<br>For additional details on setting environment variables and for setting environment variables in Linux-based operating systems, see (Xilinx Answer Record 11630).<br>If this does not work, then you should open a case with Xilinx Technical Support including the following information:<br>• Xinfo<br>• cs_analyzer.log |

*Table A-5:* **Troubleshooting Cable Reference Voltage Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. On attempting to connect to the cable you see that the LED on the cable is orange (and not green) and the following messages in the console:<br>ERROR: ERROR:iMPACT:2246 - A reference voltage has not been detected on the ribbon cable interface to the target system (pin 2). Check that power is applied to the target system and that the ribbon cable is properly seated at both ends. The status LED on Platform Cable USB is GREEN if target voltage is in the proper range and applied to the correct pin. | The issue here is usually an incorrect voltage on the VCC. Go to Issue #2. |
| 2. Is the target board powered on? | If NO: Check that the board is powered on correctly.<br>If YES: Go to Issue #3. |
| 3. Is the ribbon cable firmly plugged into the target board connector and the Platform Cable USB connector? | If NO: Re-seat the ribbon cable in both connectors.<br>If YES: Go to Issue #4. |
| 4. Is the VCC voltage on the cable connection at the appropriate voltage level? | If NO or NOT SURE: Use a test instrument to probe the voltage on the board to make sure it is in the appropriate range.<br>If YES: then open a case with Xilinx Technical Support including the following information:<br>• Xinfo<br>• cs_analyzer.log<br>• If possible, a screen shot of voltage level on the VCC of the cable at the cable connection on the target board. |

*Table A-6:* **Troubleshooting JTAG Device Detection Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. On attempting to connect to the cable you see one or more of the following messages in the console:<br>ERROR: No devices detected while scanning the JTAG chain<br>ERROR: Failed detecting JTAG device chain<br>ERROR: Opened Xilinx Platform USB Cable but failed to detect JTAG Chain.<br>This is often due to JTAG chain problem where TDI or TDO are disconnected or pulled high or low. This generally indicates board-related issues. | Go to Issue #2. |
| 2. Are the Cable TDI and TDO connected correctly at the cable header? | If NO or NOT SURE: If possible, try using a different board, cable, or cable connector to locate the fault. Modify connections so that you have a valid chain. A different ribbon cable or fly leads for connection might help. Alternatively, a different board might not have the issue.<br>If YES: Go to Issue #3. |
| 3. Is switching noise elsewhere on the board causing the failure to detect the JTAG chain? | If YES or NOT SURE: If possible, use a board-level reset to put other devices into their reset state and try to detect the JTAG chain again. Asserting this reset might reduce noise on the board during JTAG operations.<br>If NO: Go to Issue #4. |
| 4. Are non-Xilinx devices in the JTAG chain? | If YES: Check to make sure that any active-low TRST# pins of these non-Xilinx devices are pulled high, then try to detect the JTAG chain again.<br>If NO: Go to Issue #5. |
| 5. Is the active-low PROG# pin of any Virtex®-4, Virtex, Virtex-E or Spartan®-II/-E devices in the JTAG chain being held low? | If YES: Make sure the PROG# pins of these devices are pulled high. A low pulse on PROG# resets the JTAG TAP controller for these devices and prevent any operation on the chain.<br>If NO: Go to Issue #6. |

*Table A-6:* **Troubleshooting JTAG Device Detection Issues *(Cont'd)***

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 6. Do you have more than five devices in your JTAG chain and use un-buffered TCK and TMS nets? | If YES: You might need to buffer your TCK and TMS signals. A general rule of thumb is that for chains over five devices you should use buffers. The LS244 is an example of a buffer that has been used successfully with Xilinx devices. As specified by the IEEE 1149.1 standard, the TMS and TDI pins both have internal pull-up resistors. These internal pull-up resistors of 50-150k are active, regardless of the mode selected. Please refer to the appropriate configuration user guide for the FPGA device family to obtain the resistor value:<br>• *Spartan-3 FPGA Configuration User Guide* [See Reference 7, p. 211]<br>• *Spartan-6 FPGA Configuration User Guide* [See Reference 8, p. 211]<br>• *Virtex-4 FPGA Configuration User Guide* [See Reference 9, p. 211]<br>• *Virtex-5 FPGA Configuration User Guide* [See Reference 10, p. 211]<br>• *Virtex-6 FPGA Configuration User Guide* [See Reference 11, p. 211]<br>If NO: Go to Issue #7 |
| 7. Are you running TCK to fast? | If YES or NOT SURE: Reduce the speed of the cable using the JTAG Chain > Xilinx Platform USB Cable > Speed option to slow the frequency of the TCK pin to the lowest setting.<br>If NO: Open a case with Xilinx Technical Support including the following information:<br>• Xinfo<br>• cs_analyzer.log<br>• Screenshots of the JTAG lines (TDI, TDO, TCK and TMS) during a JTAG operation. Preferably the screen shot is close to the target FPGA and focussed in on one rising edge of TCK. |

*Table A-7:* **Troubleshooting Server Host Connection Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. On attempting to connect to the cable you see one or more of the following messages in the console:<br>ERROR: Socket Open Failed. localhost/127.0.0.1:50001<br>java.net.ConnectException: Connection refused<br>These messages can appear in cases where another application had control of the cable (for example, the iMPACT tool). Alternatively, it is possible that another device or application on your system is denying access to the port/socket. | Go to Issue #2.. |
| 2. Are you running any firewall applications that might prevent the ChipScope Pro Analyzer tool from connecting to a TCP/IP socket? | If YES: Disable any applications that might be denying the access to the TCP/IP socket and try to reconnect to the cable.<br>If NO: Go to Issue #3. |
| 3. Have you been using any other Xilinx application to access the cable? | If YES: It is possible that the other application has not released the cable lock. This situation can often be worked around by closing down the other application. If this does not help, try running the following commands in iMPACT batch mode to clean the stale cable locks:<br>> impact -batch<br># cleancablelock<br># exit<br>If NO: Open a case with Xilinx Technical Support including the following information:<br>• Xinfo<br>• cs_analyzer.log |

# ChipScope Pro Analyzer Core Troubleshooting

This section deals with issues where the ChipScope Pro Analyzer tool can access the cable and the JTAG chain but is either not detecting any debug cores in the Xilinx FPGAs or is having trouble triggering an ILA core or displaying captured ILA core data.

- For troubleshooting issues related to "INFO: Found 0 Core Units in the JTAG device Chain" messages, see Table A-8.
- For troubleshooting issues related to "Waiting for upload" messages, see Table A-9, page 206.
- For troubleshooting issues related to "ERROR: Fatal - Did not find trigger mark in buffer. Data buffer may be corrupt!" messages, see Table A-10, page 208.

*Table A-8:* **Troubleshooting Core Detection Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. On attempting to connect to the cable you see the following message in the console:<br>INFO: Found 0 Core Units in the JTAG device Chain<br>There are a number of reasons why the ChipScope Pro Analyzer tool might not be able to detect the core units. The ChipScope Pro Analyzer tool polls the JTAG chain for a status word that indicates the number and type(s) of core(s) in the device. The reading of the status word can result in corrupt date either by noise on the JTAG TAP signals or a timing issue in the design that affects the core. | Go to Issue #2.. |
| 2. Are the ChipScope Pro ICON (integrated controller), ILA (integrated logic analyzer), VIO (virtual input/output), and/or ATC2 (Agilent trace core) debug cores implemented correctly or even present in the design? | If NO or NOT SURE: Verify that the cores are in the design by performing the following steps:<br>1. Open the FPGA Editor to edit the placed-and-routed NCD file of your design.<br>2. Go to **Tools** > **ILA**. A window displays all the probed signals. If an error message appears specifying, "There is no ILA Core," your design does not contain an ILA debug core.<br>If no cores are detected, you must go back to your design and determine why the core was not implemented. The synthesis and translate reports can tell you if all the netlists are correctly implemented, including the ILA Core and the ICON Core. Also check that the NCF (netlist constraints file) associated with the core was applied. If the core netlist file (*.ngc/ngo) was moved during implementation the associated constraint file (*.ncf) might not have been moved accordingly.<br><br>If YES: Go to Issue #3. |

*Table A-8:* **Troubleshooting Core Detection Issues *(Cont'd)***

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 3. Has the JTAG logic been released by the programming tool? | If NO or NOT SURE: Programming the FPGA with the iMPACT tool instead of the ChipScope Pro Analyzer tool can solve this problem. Try performing the following steps:<br>1. Run iMPACT, to program the FPGA.<br>2. Exit iMPACT.<br>3. Run ChipScope Pro Analyzer tool.<br>If YES: Go to Issue #4. |
| 4. Is there a Xilinx System ACE™ MPM device in the JTAG Chain? | If YES: There is a known issue when the JTAG chain contains a System ACE MPM. The core unit might not be found. To work around the problem, put the following line in the ChipScope Pro Analyzer tool project file (.cpj) and read it in before opening the cable:<br>avoidUserRegDeviceX=1,2<br>Replace the "X" with the position index number of the V50E device. Use index 0 for the first device in the chain. Information messages about skipping scanning on the V50E device are placed in the cs_analyzer.log file.<br>To use the updated .cpj file, follow the steps listed above. Be sure to load the .cpj file immediately after ChipScope Pro Analyzer tool launches before any other action.<br>If NO: Go to Issue #5. |
| 5. Are there any non-Xilinx devices in the JTAG chain? | If YES: You must enter the instruction register length for any non-Xilinx devices in the chain in the ChipScope Pro Analyzer GUI. To find this, you can examine the BSDL file corresponding to the device. There will be an entry similar to below:<br>attribute INSTRUCTION_LENGTH of <entity name> : entity is XX;<br>If you do not enter correct instruction register lengths, the ChipScope Pro Analyzer tool cannot set the JTAG device offsets correctly and will not be able to identify the devices or debug cores.<br>If NO: Go to Issue #6. |
| 6. Has the device correctly exited from the device start-up sequence? | If NO: The ChipScope Pro Analyzer tool might not be able to find the core because the configuration options are not set correctly. If the BitGen option LCK_cycle (or Release DLL in Project Navigator) is not set to "Nowait," the ChipScope Pro Analyzer tool might not be able to detect the core because the ChipScope Pro core must be initialized with the release of GWE. Setting this option to "Nowait" (which is the default option) might solve this problem.<br>If YES: Go to Issue #7. |

*Table A-8:* **Troubleshooting Core Detection Issues *(Cont'd)***

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 7. Is the DONE pin of the FPGA device being held low on the board?<br>If there are multiple FPGAs on the board with the DONE pins tied together, this error might occur in a situation where DONE is held Low by unconfigured devices and configuration does not fully complete. | If YES: To work around this issue, ensure that all FPGAs on the board are configured or that the DriveDONE bitgen option is set for the target device.<br>If NO: Go to Issue #8. |
| 8. Have the core constraints being applied correctly? | If NO or NOT SURE: Check that a PERIOD constraint has been added to the clock used as the Clock for your ILA. If there is no constraint on this net in the ISE project, the ChipScope constraints are not applied correctly and might result in the cores not being recognized. Also check that the NCF file associated with the core was applied. If the core netlist file (*.ngc/ngo) was moved during implementation the associated constraint file (*.ncf) might not have been moved.<br>If YES: Open a case with Xilinx Technical Support including the following information:<br>• cs_analyzer.log<br>• Archived ISE project including inserter project (<project_name>.cdc) |

*Table A-9:* **Troubleshooting ILA Core Triggering Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. After arming the trigger of the ILA core, you see the following message in the status bar:<br>Waiting for upload<br>but nothing happens. Possible reasons for this problem:<br>• The trigger condition is never met<br>• The trigger clock (clock mapped to the ILA Core) is stopped<br>• BUFG is not being used on JTAG CLK (for the ICON). | Go to Issue #2. |
| 2. Is the trigger condition ever met? | If NO or NOT SURE: Check the message at the bottom of the ChipScope Pro Analyzer tool window. If it is similar to **Waiting for trigger, Sample buffer has 0 samples(0%)**:<br>Go to **Trigger Setup** and **Trigger Immediate**. If the ChipScope Pro Analyzer tool starts the acquisition and shows the samples (the waveform appears), your design is fine; the clock is running, but your trigger condition never occurs.<br>In the **Trigger Setup** windows, ensure that you have set the condition correctly if you are certain that this event (the trigger condition) happens in your design.<br>If YES: Go to Issue #3. |
| 3. Is the clock that is connected to the ILA core running? | If NO or NOT SURE: If the message at the bottom of the window is similar to "Waiting for Core to be armed, slow or stopped clock," the trigger condition is not the problem -- the ILA Core does not have a valid clock and is not able to start the acquisition.<br>To fix this, ensure that you have mapped a valid clock using either the ChipScope Pro Core Inserter tool or in your RTL code (if manually using generated cores). If you are not sure if the clock mapped to the ILA core is running, try to connect your system clock instead (or a clock that you are sure is running).<br>If YES: Go to Issue #4. |
| 4. Have you used a BUFG on the ICON JTAG clock? | If NO: If the JTAG clock does not use a BUFG, the **Waiting for upload** message can appear. You must re-implement your design, ensuring that this attribute is set for your ICON generation.<br>If YES: Go to Issue #5. |

*Table A-9:* **Troubleshooting ILA Core Triggering Issues *(Cont'd)***

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 5. Have the core constraints been applied correctly? | If NO or NOT SURE: Check that a PERIOD constraint has been added to the clock used as the CLK input to the ILA core. If there is no constraint on this net in the ISE project, the timing constraints are not applied correctly and might result in the cores not being recognized. Also check that the NCF file associated with the core was applied. If the core netlist file (*.ngc/ngo) was moved during implementation the associated constraint file (*.ncf) might not have been moved accordingly.<br>If YES: Go to Issue #6. |
| 6. Is the JTAG TCK clock running too fast? | If YES or NOT SURE: Reduce the speed of the cable using the JTAG Chain > Xilinx Platform USB Cable > Speed option to slow the frequency of the TCK pin to the lowest setting.<br>If NO: Open a case with Xilinx Technical Support including the following information:<br>• cs_analyzer.log<br>• Archived ISE project including inserter project (<filename>.cdc) |

*Table A-10:* **Troubleshooting Corrupt ILA Core Data Buffer Issues**

| Issue(s) | Solution(s) or Work-Around(s) |
|---|---|
| 1. After arming the trigger of the ILA core, you see the following message in the console:<br>ERROR: Fatal - Did not find trigger mark in buffer. Data buffer may be corrupt!<br>This is likely to a timing issue in the design or core that is affecting the data being sampled, thus corrupting the data in the buffer. | Go to Issue #2. |
| 2. Have you applied offset in/out and period constraints? | If YES: Be sure to apply these to all applicable signals to ensure timing issues are not causing this issue.<br>If NO: Go to Issue #3. |
| 3. Have you used a BUFG on the ICON JTAG clock? | If NO: If the JTAG clock does not use a BUFG, the "ERROR: Fatal - Did not find trigger mark in buffer. Data buffer may be corrupt!" message can appear. You must re-implement your design, ensuring that this attribute is set for your ICON generation.<br>If YES: Go to Issue #4. |
| 4. Have the core constraints been applied correctly? | If NO or NOT SURE: Check that a PERIOD constraint has been added to the clock used as the CLK input to the ILA core. If there is no constraint on this net in the ISE project, the timing constraints are not applied correctly and might result in the ILA core control logic not working correctly. Also check that the NCF file associated with the core was applied. If the core netlist file (*.ngc/ngo) was moved during implementation the associated constraint file (*.ncf) might not have been moved accordingly.<br>If YES: Go to Issue #5. |
| 5. Is the design and core meeting timing? | If NO: Simplify the ILA core by reducing the number of trigger ports, number of trigger bits per port, data width, and/or data depth in order to get this design to meet timing again.<br>If YES: Open a case with Xilinx Technical Support including the following information:<br>• cs_analyzer.log<br>• Archived ISE project including inserter project (<filename>.cdc) |

# Gathering Information for Xilinx Technical Support

## Obtaining Xinfo Information

The Xinfo application is used to collect system info used by Xilinx tools which gives installation logs and environment details that are useful for debug.

### On Windows

1. Run **Start > Run > Xinfo**
2. In the Xinfo application, select **File > Export > Save as Text file**.

### On Linux

1. Run xinfo from a shell prompt
2. In the Xinfo application, select **File > Export > Save as Text file**.

## Obtaining ChipScope Pro Analyzer Log File Information

The cs_analyzer.log file contains a log of console messages from your last ChipScope Pro Analyzer tool session.

### On Windows

The cs_analyzer.log file is stored in your %homepath%/.chipscope directory. This location is typically the same as C:/Documents and Settings/<username>/.chipscope.

### On Linux

The cs_analyzer.log file is stored in your $HOME/.chipscope directory.

## Obtaining ChipScope Pro Core Inserter Tool Log File Information

The cs_inserter.log file contains a log of console messages from your last ChipScope Pro Core Inserter tool session.

### On Windows

The cs_analyzer.log file is stored in your %homepath%/.chipscope directory. This location is typically the same as C:/Documents and Settings/<username>/.chipscope.

### On Linux

The cs_analyzer.log file is stored in your $HOME/.chipscope directory.

## Obtaining an Archived ISE Tool Project

1. In the ISE Project Navigator tool, select Project > Archive
2. This creates a `<project_name>.zip` file with all project files. If you are using the ChipScope Pro Core Inserter tool with your project, make sure that the `<filename>.cdc` is part of the project before archiving.

# Appendix B

# References

Documents specific to multi-gigabit serial transceivers:


1. UG076, *Virtex-4 FPGA RocketIO Multi-Gigabit Transceiver User Guide*
2. UG196, *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*
3. UG198, *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*
4. UG366, *Virtex-6 FPGA GTX Transceivers User Guide*
5. UG371, *Virtex-6 FPGA GTH Transceivers User Guide*
6. UG386, *Spartan-6 FPGA GTP Transceivers User Guide*


Xilinx® Virtex® FPGA and Spartan® FPGA references:


7. UG332, *Spartan-3 Generation Configuration User Guide*
8. UG380, *Spartan-6 FPGA Configuration User Guide*
9. UG071, *Virtex-4 FPGA Configuration User Guide*
10. UG191, *Virtex-5 FPGA Configuration User Guide*
11. UG360, *Virtex-6 FPGA Configuration User Guide*
12. XAPP139 *Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan*
13. UG480, *7 Series FPGAs XADC Dual 12-Bit MSPS Analog-to-Digital Converter User Guide*


Xilinx Tools and Solutions


14. ISE® Design Suite Documentation
15. Embedded Development Kit (EDK) Documentation
16. ISE Design Suite Software Matrix
17. PlanAhead™ Design Analysis Tool
18. Xilinx Support
19. System Generator for DSP
20. Xilinx Online Store
21. Silicon Stepping
22. UG192, *Virtex-5 System Monitor User Guide*
23. UG370, *Virtex-6 System Monitor User Guide*

Other references:

24. ActiveState
25. Agilent Technologies
26. Tcl Developer Xchange
27. Byte Tools
28. UG480, *7 Series FPGAs XADC Dual 12-Bit MSPS Analog-to-Digital Converter User Guide*