System Debugging Tools Overview 9

2013.11.04

QII53027 Subscribe Send Feedback

About Altera System Debugging Tools

The Altera® system debugging tools help you verify your FPGA designs. As your product requirements continue to increase in complexity, the time you spend on design verification continues to rise. This manual provides a quick overview of the tools available in the system debugging suite and discusses the criteria for selecting the best tool for your design.

System Debugging Tools Portfolio

The Quartus® II software provides a portfolio of system design debugging tools for real-time verification of your design. Each tool in the system debugging portfolio uses a combination of available memory, logic, and routing resources to assist in the debugging process. The tools provide visibility by routing (or "tapping") signals in your design to debugging logic. The debugging logic is then compiled with your design and downloaded into the FPGA or CPLD for analysis. Because different designs can have different constraints and requirements, such as the number of spare pins available or the amount of logic or memory resources remaining in the physical device, you can choose a tool from the available debugging tools that matches the specific requirements for your design.

System Debugging Tools Comparison

Table 9-1: Debugging Tools Portfolio

Tool	Description	Typical Usage
System Console	Uses a Tcl interpreter to communicate with hardware modules instantiated in your design. You can use it with the Transceiver Toolkit to monitor or debug your design. System Console provides real-time in-system debugging capabilities. Using System Console, you can read from and write to Memory Mapped components in our system without the help of a processor or additional software. System Console uses Tcl as the fundamental infrastructure which means you can source scripts, set variables, write procedures, and take advantage of all the features of the Tcl scripting language.	You need to perform system-level debugging. For example, if you have an Avalon-MM slave or Avalon-ST interfaces, you can debug your design at a transaction level. The tool supports JTAG connectivity, but also supports connectivity to a simulation model, as well as TCP/IP connectivity to the target FPGA you wish to debug.
Transceiver Toolkit	The Transceiver Toolkit allows you to test and tune transceiver link signal quality. You can use a combination of bit error rate (BER), bathtub curve, and eye contour graphs as quality metrics. Auto Sweeping of physical medium attachment (PMA) settings allows you to quickly find an optimal solution.	You need to debug or optimize signal integrity of your board layout even before the actual design to be run on the FPGA is ready.
SignalTap ® II Logic Analyzer	This logic analyzer uses FPGA resources to sample test nodes and outputs the information to the Quartus II software for display and analysis.	You have spare on-chip memory and you want functional verification of your design running in hardware.
SignalProbe	This tool incrementally routes internal signals to I/O pins while preserving results from your last place-and-routed design.	You have spare I/O pins and you would like to check the operation of a small set of control pins using either an external logic analyzer or an oscilloscope.

Tool	Description	Typical Usage
Logic Analyzer Interface (LAI)	This tool multiplexes a larger set of signals to a smaller number of spare I/O pins. LAI allows you to select which signals are switched onto the I/O pins over a JTAG connection.	You have limited on-chip memory, and have a large set of internal data buses that you would like to verify using an external logic analyzer. Logic analyzer vendors, such as Tektronics and Agilent, provide integration with the tool to improve the usability of the tool.
In-System Sources and Probes	This tool provides an easy way to drive and sample logic values to and from internal nodes using the JTAG interface.	You want to prototype a front panel with virtual buttons for your FPGA design.
In-System Memory Content Editor	This tool displays and allows you to edit on-chip memory.	You would like to view and edit the contents of on-chip memory that is not connected to a Nios II processor. You can also use the tool when you do not want to have a Nios II debug core in your system.
Virtual JTAG Interface	This megafunction allows you to communicate with the JTAG interface so that you can develop your own custom applications.	You have custom signals in your design that you want to be able to communicate with.

Altera JTAG Interface (AJI)

With the exception of SignalProbe, each of the on-chip debugging tools uses the JTAG port to control and read back data from debugging logic and signals under test. System Console uses JTAG and other interfaces as well. The JTAG resource is shared among all of the on-chip debugging tools.

Required Arbitration Logic

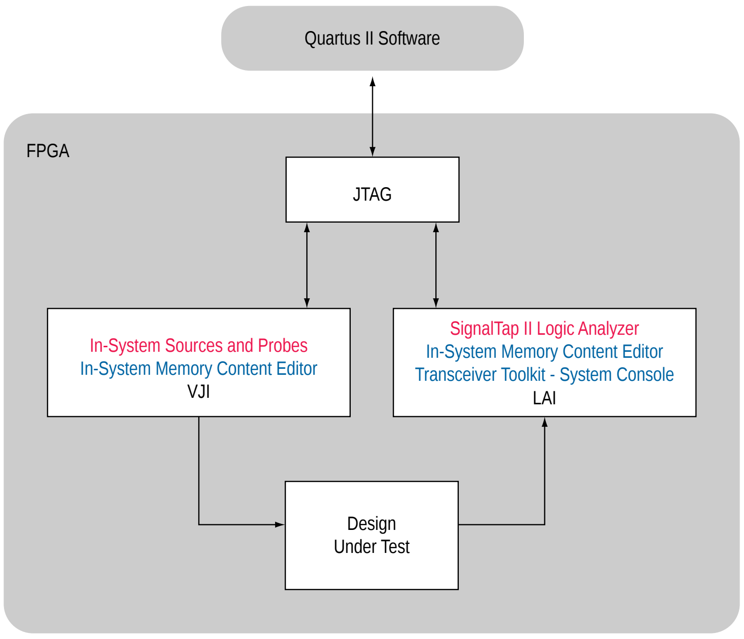
For all system debugging tools except System Console, the Quartus II software compiles logic into your design automatically to distinguish between data and control information and each of the debugging logic blocks, when the JTAG resource is required. This arbitration logic, also known as the System-Level Debugging (SLD) infrastructure, is shown in the design hierarchy of your compiled project as **sld_hub:sld_hub_inst**. The SLD logic allows you to instantiate multiple debugging blocks into your design and run them simultaneously. For System Console, you must explicitly insert debug IP cores into your design to enable debugging.

Debugging Ecosystem

To maximize debugging closure, the Quartus II software allows you to use a combination of the debugging tools in tandem to fully exercise and analyze the logic under test. All of the tools have basic analysis features built in; that is, all of the tools enable you to read back information collected from the design nodes that are connected to the debugging logic. Out of the set of debugging tools, the SignalTap II Logic Analyzer, the LAI, and the SignalProbe feature are general purpose debugging tools optimized for probing signals in your register transfer level (RTL) netlist. In-System Sources and Probes, the Virtual JTAG Interface, System Console, Transceiver Toolkit, and In-System Memory Content Editor, allow you to read back data from the debugging breakpoints, and to input values into your design during runtime.

Taken together, the set of on-chip debugging tools form a debugging ecosystem. The set of tools can generate a stimulus to and solicit a response from the logic under test, providing a complete debugging solution.

Figure 9-1: Debugging Ecosystem


Quartus II Software
FPGA
JTAG
In-System Sources and Probes
In-System Memory Content Editor
VJI
SignalTap II Logic Analyzer
In-System Memory Content Editor
Transceiver Toolkit - System Console
LAI
Design
Under Test

About Analysis Tools for RTL Nodes

The SignalTap II Logic Analyzer, SignalProbe, and LAI are designed specifically for probing and debugging RTL signals at system speed. They are general-purpose analysis tools that enable you to tap and analyze any routable node from the FPGA or CPLD. If you have spare logic and memory resources, the SignalTap II Logic Analyzer is useful for providing fast functional verification of your design running on actual hardware.

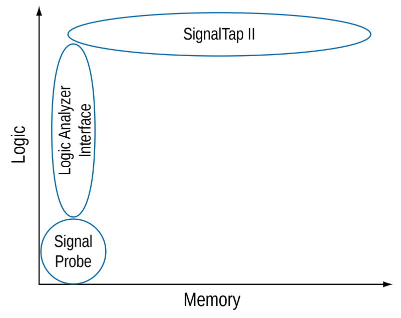
Conversely, if logic and memory resources are tight and you require the large sample depths associated with external logic analyzers, both the LAI and the SignalProbe make it easy to view internal design signals using external equipment.

Note: The SignalTap II Logic Analyzer is not supported on CPLDs, because there are no memory resources available on these devices.

Resource Usage

The most important selection criteria for these three tools are the available resources remaining on your device after implementing your design and the number of spare pins available. You should evaluate your preferred debugging option early on in the design planning process to ensure that your board, your Quartus II project, and your design are all set up to support the appropriate options. Planning early can reduce time spent during debugging and eliminate the necessary late changes to accommodate your preferred debugging methodologies.

Figure 9-2: Resource Usage per Debugging Tool



Overhead Logic

Any debugging tool that requires the use of a JTAG connection requires the SLD infrastructure logic, for communication with the JTAG interface and arbitration between any instantiated debugging modules. This overhead logic uses around 200 logic elements (LEs), a small fraction of the resources available in any of the supported devices. The overhead logic is shared between all available debugging modules in your design. Both the SignalTap II Logic Analyzer and the LAI use a JTAG connection.

For SignalProbe

SignalProbe requires very few on-chip resources. Because it requires no JTAG connection, SignalProbe uses no logic or memory resources. SignalProbe uses only routing resources to route an internal signal to a debugging test point.

For Logic Analyzer Interface

The LAI requires a small amount of logic to implement the multiplexing function between the signals under test, in addition to the SLD infrastructure logic. Because no data samples are stored on the chip, the LAI uses no memory resources.

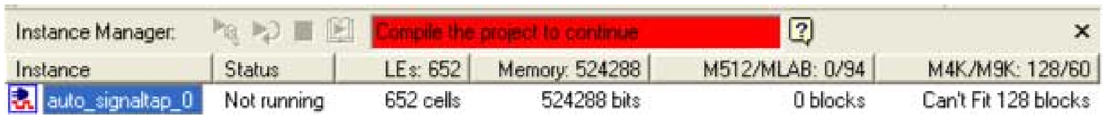
For SignalTap II

The SignalTap II Logic Analyzer requires both logic and memory resources. The number of logic resources used depends on the number of signals tapped and the complexity of the trigger logic. However, the amount of logic resources that the SignalTap II Logic Analyzer uses is typically a small percentage of most designs. A baseline configuration consisting of the SLD arbitration logic and a single node with basic triggering logic contains approximately 300 to 400 Logic Elements (LEs). Each additional node you add to the baseline configuration adds about 11 LEs. Compared with logic resources, memory resources are a more important factor to consider for your design. Memory usage can be significant and depends on how you configure your SignalTap II Logic Analyzer instance to capture data and the sample depth that your design requires for debugging. For the SignalTap II Logic Analyzer, there is the added benefit of requiring no external equipment, as all of the triggering logic and storage is on the chip.

Resource Estimation

The resource estimation feature for the SignalTap II Logic Analyzer and the LAI allows you to quickly judge if enough on-chip resources are available before compiling the tool with your design.

Figure 9-3: Resource Estimator



Instance Manager: Compile the project to continue

Instance	Status	LEs: 652	Memory: 524288	M512/MLAB: 0/94	M4K/M9K: 128/60
auto_signaltap_0	Not running	652 cells	524288 bits	0 blocks	Can't Fit 128 blocks

Pin Usage

For SignalProbe

The ratio of the number of pins used to the number of signals tapped for the SignalProbe feature is one-to-one. Because this feature can consume free pins quickly, a typical application for this feature is routing control signals to spare pins for debugging.

For Logic Analyzer Interface

The ratio of the number of pins used to the number of signals tapped for the LAI is many-to-one. It can map up to 256 signals to each debugging pin, depending on available routing resources. The control of the active signals that are mapped to the spare I/O pins is performed via the JTAG port. The LAI is ideal for routing data buses to a set of test pins for analysis.

For SignalTap II

Other than the JTAG test pins, the SignalTap II Logic Analyzer uses no additional pins. All data is buffered using on-chip memory and communicated to the SignalTap II Logic Analyzer GUI via the JTAG test port.

Usability Enhancements

The SignalTap II Logic Analyzer, SignalProbe, and LAI tools can be added to your existing design with minimal effects. With the node finder, you can find signals to route to a debugging module without making any changes to your HDL files. SignalProbe inserts signals directly from your post-fit database. The SignalTap II Logic Analyzer and LAI support inserting signals from both pre-synthesis and post-fit netlists.

Incremental Compilation

All three tools allow you to find and configure your debugging setup quickly. In addition, the Quartus II incremental compilation feature and the Quartus II incremental routing feature allow for a fast turnaround time for your programming file, increasing productivity and enabling fast debugging closure.

Both the LAI and SignalTap II Logic Analyzer support incremental compilation. With incremental compilation, you can add a SignalTap II Logic Analyzer instance or an LAI instance incrementally into your placed-and-routed design. This has the benefit of both preserving your timing and area optimizations from your existing design, and decreasing the overall compilation time when any changes are necessary during the debugging process. With incremental compilation, you can save up to 70% compile time of a full compilation.

Incremental Routing

SignalProbe uses the incremental routing feature. The incremental routing feature runs only the Fitter stage of the compilation. This leaves your compiled design untouched, except for the newly routed node or nodes. With SignalProbe, you can save as much as 90% compile time of a full compilation.

Automation Via Scripting

As another productivity enhancement, all tools in the on-chip debugging tool set support scripting via the `quartus_stp` Tcl package. For the SignalTap II Logic Analyzer and the LAI, scripting enables user-defined automation for data collection while debugging in the lab. The System Console includes a full Tcl interpreter for scripting.

Remote Debugging

You can perform remote debugging of your system with the Quartus II software via the System Console. This feature allows you to debug equipment deployed in the field through an existing TCP/IP connection.

There are two Application Notes available to assist you.

- Application Note 624 describes how to set up your NIOS II system to use the System Console to perform remote debugging.
- Application Note 693 describes how to set up your Altera SoC to use the SLD tools to perform remote debugging.

Related Information

- **Application Note 624: Debugging with System Console over TCP/IP**
- **Application Note 693: Remote Debugging over TCP/IP for Altera SoC**

Suggested On-Chip Debugging Tools for Common Debugging Features

Table 9-2: Tools for Common Debugging Features *(1)*

Feature	SignalProbe	Logic Analyzer Interface (LAI)	SignalTap II Logic Analyzer	Description
Large Sample Depth	N/A	X	—	An external logic analyzer used with the LAI has a bigger buffer to store more captured data than the SignalTap II Logic Analyzer. No data is captured or stored with SignalProbe.

Feature	SignalProbe	Logic Analyzer Interface (LAI)	SignalTap II Logic Analyzer	Description
Ease in Debugging Timing Issue	X	X	—	External equipment, such as oscilloscopes and mixed signal oscilloscopes (MSOs), can be used with either LAI or SignalProbe. When used with the LAI, external equipment provides you with access to timing mode, which allows you to debug combined streams of data.
Minimal Effect on Logic Design	X	X *(2)*	X *(2)*	The LAI adds minimal logic to a design, requiring fewer device resources. The SignalTap II Logic Analyzer has little effect on the design, because it is set as a separate design partition. SignalProbe incrementally routes nodes to pins, not affecting the design at all.

Feature	SignalProbe	Logic Analyzer Interface (LAI)	SignalTap II Logic Analyzer	Description
Short Compile and Recompile Time	X	X (2)	X (2)	SignalProbe attaches incrementally routed signals to previously reserved pins, requiring very little recompilation time to make changes to source signal selections. The SignalTap II Logic Analyzer and the LAI can take advantage of incremental compilation to refit their own design partitions to decrease recompilation time.
Triggering Capability	N/A	N/A	X	The SignalTap II Logic Analyzer offers triggering capabilities that are comparable to commercial logic analyzers.
I/O Usage	—	—	X	No additional output pins are required with the SignalTap II Logic Analyzer. Both the LAI and SignalProbe require I/O pin assignments.

Feature	SignalProbe	Logic Analyzer Interface (LAI)	SignalTap II Logic Analyzer	Description
Acquisition Speed	N/A	—	X	The SignalTap II Logic Analyzer can acquire data at speeds of over 200 MHz. The same acquisition speeds are obtainable with an external logic analyzer used with the LAI, but might be limited by signal integrity issues.
No JTAG Connection Required	X	—	X	A FPGA design with the LAI requires an active JTAG connection to a host running the Quartus II software. SignalProbe and SignalTap II do not require a host for debugging purposes.

Feature	SignalProbe	Logic Analyzer Interface (LAI)	SignalTap II Logic Analyzer	Description
No External Equipment Required	—	—	X	The SignalTap II Logic Analyzer logic is completely internal to the programmed FPGA device. No extra equipment is required other than a JTAG connection from a host running the Quartus II software or the stand-alone SignalTap II Logic Analyzer software. SignalProbe and the LAI require the use of external debugging equipment, such as multimeters, oscilloscopes, or logic analyzers.

Notes to Table:

1. • X indicates the recommended tools for the feature.
   • — indicates that while the tool is available for that feature, that tool might not give the best results.
   • N/A indicates that the feature is not applicable for the selected tool.
2. When used with incremental compilation.

About Stimulus-Capable Tools

The In-System Memory Content Editor, In-System Sources and Probes, and Virtual JTAG interface enable you to use the JTAG interface as a general-purpose communication port. Though all three tools can be used to achieve the same results, there are some considerations that make one tool easier to use in certain applications than others. In-System Sources and Probes is ideal for toggling control signals. The In-System Memory Content Editor is useful for inputting large sets of test data. Finally, the Virtual JTAG interface is well suited for more advanced users who want to develop their own customized JTAG solution.

System Console provides system-level debugging at a transaction level, such as with Avalon-MM slave or Avalon-ST interfaces. You can communicate to a chip through JTAG, virtual peripheral interface (VPI) for simulation models, and TCP/IP protocols. System Console uses a Tcl interpreter to communicate with hardware modules that you have instantiated into your design.

In-System Sources and Probes

In-System Sources and Probes is an easy way to access JTAG resources to both read and write to your design. You can start by instantiating a megafunction into your HDL code. The megafunction contains source ports and probe ports for driving values into and sampling values from the signals that are connected to the ports, respectively. Transaction details of the JTAG interface are abstracted away by the megafunction. During runtime, a GUI displays each source and probe port by instance and allows you to read from each probe port and drive to each source port. The GUI makes this tool ideal for toggling a set of control signals during the debugging process.

Push Button Functionality

A good application of In-System Sources and Probes is to use the GUI as a replacement for the push buttons and LEDs used during the development phase of a project. Furthermore, In-System Sources and Probes supports a set of scripting commands for reading and writing using `quartus_stp`. When used with the Tk toolkit, you can build your own graphical interfaces. This feature is ideal for building a virtual front panel during the prototyping phase of the design.

In-System Memory Content Editor

The In-System Memory Content Editor allows you to quickly view and modify memory content either through a GUI interface or through Tcl scripting commands. The In-System Memory Content Editor works by turning single-port RAM blocks into dual-port RAM blocks. One port is connected to your clock domain and data signals, and the other port is connected to the JTAG clock and data signals for editing or viewing.

Generate Test Vectors

Because you can modify a large set of data easily, a useful application for the In-System Memory Content Editor is to generate test vectors for your design. For example, you can instantiate a free memory block, connect the output ports to the logic under test (using the same clock as your logic under test on the system side), and create the glue logic for the address generation and control of the memory. At runtime, you can modify the contents of the memory using either a script or the In-System Memory Content Editor GUI and perform a burst transaction of the data contents in the modified RAM block synchronous to the logic being tested.

Virtual JTAG Interface Megafunction

The Virtual JTAG Interface megafunction provides the finest level of granularity for manipulating the JTAG resource. This megafunction allows you to build your own JTAG scan chain by exposing all of the JTAG control signals and configuring your JTAG Instruction Registers (IRs) and JTAG Data Registers (DRs). During runtime, you control the IR/DR chain through a Tcl API, or with System Console. This feature is meant for users who have a thorough understanding of the JTAG interface and want precise control over the number and type of resources used.

System Console

System Console is a framework that you can launch from the Quartus II software to start services for performing various debugging tasks. System Console provides you with Tcl scripts and a GUI to access the Qsys system integration tool to perform low-level hardware debugging of your design, as well as identify a module by its path, and open and close a connection to a Qsys module. You can access your design at a system level for purposes of loading, unloading, and transferring designs to multiple devices.

Test Signal Integrity

System Console also allows you to access commands that allow you to control how you generate test patterns, as well as verify the accuracy of data generated by test patterns. You can use JTAG debug commands in

System Console to verify the functionality and signal integrity of your JTAG chain. You can test clock and reset signals.

Board Bring-Up and Verification

You can use System Console to access programmable logic devices on your development board, perform board bring-up, and perform verification. You can also access software running on a Nios II or Altera SoC processor, as well as access modules that produce or consume a stream of bytes.

Test Link Signal Integrity with Transceiver Toolkit

Transceiver Toolkit runs from the System Console framework, and allows you to run automatic tests of your transceiver links for debugging and optimizing your transceiver designs. You can use the Transceiver Toolkit GUI to set up channel links in your transceiver devices, and then automatically run EyeQ and Auto Sweep testing to view a graphical representation of your test data.

Document Revision History

Table 9-3: Document Revision History

Date	Version	Changes
November 2013	13.1.0	Dita conversion. Added link to Remote Debugging over TCP/IP for Altera SoC Application Note.
June 2012	12.0.0	Maintenance release.
November 2011	10.0.2	Maintenance release. Changed to new document template.
December 2010	10.0.1	Maintenance release. Changed to new document template.
July 2010	10.0.0	Initial release

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.

Related Information

Quartus II Handbook Archive