Select the Correct Debug Methodology for Your Altera FPGA Design

Introduction

FPGA technology is advancing at an extremely fast rate. Density and performance mean devices can truly claim to fit in to the heart of the system. The largest Stratix III devices from Altera now have up to 340,000 logic elements, 16 Mbits of memory and 760 18x18 multipliers. These are complemented with phase lock loops and high-speed I/O capable of supporting the latest memory interfaces or newest serial protocols.

Tektronix®

Traditionally, device debug was relatively simple as it was possible to use a combination of simulation and board measurements to quickly identify a potential issue. Today however, devices are the fundamental building block of the system; they are much more complex with up to 6 million equivalent ASIC gates, and can take as much as a day to compile when predominantly full or when they have particularly arduous timing constraints.

Debug can on average take up to 50% of the development time, particularly when looking for a difficult issue. We therefore need to use innovative ways to speed up the flow. Tektronix, First Silicon Solutions (FS2), and Altera have teamed up to provide a robust solution to enable quick identification and troubleshooting of even the most difficult bugs.

Verification Methods

Simulation and verification are important stages of an FPGA design and form a large portion of both the development and debug stage of any design. They perform an important task at the initial development, to ensure the behavioral functionality of the design and to make certain the design is fit for manufacture, as well as providing an excellent way of performing timing verification.

In higher end applications, simulation also provides a good solution in complete system verification. This enables you to perform cycle accurate simulation of System-on-Chip solutions, allowing the engineering team to debug both hardware and software simultaneously. However, even the simplest of software instructions can take days to model and will require high-performance computing with hardware emulators to complete. This type of requirement is necessary for an ASIC development where at least initially there is no target hardware, so simulation is necessary.

FPGA design is a little different. Hardware is available at an earlier stage of the project, which means verification and debug can be achieved on the target hardware rather then on PC simulation. On-Chip debug provides many advantages besides speed. It allows for cycle accurate operation on the end unit and often shows bugs that are not always seen as part of the simulation process.

The real difficulty lies in actually identifying and analyzing the bug. We'll now explore a few of the methods for doing this.

Analyzing Nodes Using Traditional Methods

Traditionally, it has been possible to analyze a signal on a pin simply by placing a probe on a pin. Of course things have advanced significantly and FPGAs now have in excess of 1000 pins. Packaging technology has also moved forward, and most high-end solutions employ a combination of Ball Grid Array packaging, narrow pin pitches, and surface mount technology. This means you need to plan ahead if you wish to probe particular signals and include breakout connectors on a board.

A further consideration for FPGA debug is the fact that most of the desired signals or nodes are usually buried within the device making debug difficult. The obvious solution is to move an internal node onto a spare signal or pin for analysis. This was once extremely time-consuming, because moving a node to a signal would require the device to be completely recompiled.

Today, Altera offers a SignalProbe solution, which allows the user to reserve signals or pins and then allocate the required node to the signal when it needs to be examined. The real advantage of this solution is the ability to use an incremental compilation technique, which simply connects the node to the pin without recompiling the rest of the design, saving hours in compilation times.

SignalProbe is a valuable solution when analyzing single nodes or narrow bus interfaces. It is quick, simple to setup and configure, plus it does not require significant FPGA resources. However, one of the disadvantages is that it is more challenging to find a complex problem or system bug using this technique.

Embedded (On-Chip) Logic Analyzer Method

Many tools are now provided by both the FPGA vendors and third party EDA partners to provide on-chip logic analyzers within the target hardware. Altera for example provides the SignalTap II logic analyzer as part of their tool suite. The logic analyzers are built as part of the FPGA project and use FPGA logic and memory to create the solution. Logic is used to create the logic analyzer state machines, interface and trigger mechanism. The memory is used to store the captured data. The exact quantity of logic and memory is dependant on the application.

The embedded logic analyzer uses the JTAG interface within the FPGA to pass the captured data to an external PC where it is displayed.

These on-chip analyzers are extremely powerful and can allow for multiple capture levels or even multiple logic analyzers to be developed to support different clock domains. The use of embedded logic analyzers provide the ability to dramatically speed up the debug process and can reduce the time it takes to find a particularly demanding bug from days to hours.

The embedded logic analyzer does suffer from a number of drawbacks; because the logic analyzer function has to be embedded into the FPGA. This can have some impact on the system operation, for example:

- You may need a larger FPGA device than desired to accommodate the LA functionality.
- Logic analyzer function can have an impact on both FPGA fitting time and performance
- Your end product may need to be recompiled to remove the logic analyzer once the design is complete

Traditionally, there has been a further issue using the embedded logic analyzer within an FPGA, in that it was necessary to recompile the whole device every time a major change was required to the analyzer. Modifications such as advanced triggering, change of signal analysis, or trigger depth resulted in a long recompile time.

Recently, incremental compilation techniques have significantly sped up this aspect of the design, allowing you to quickly compile the operation of the logic analyzer or even the design fix itself, without the need to recompile the whole design. This technique can reduce compile times by up to 70%.

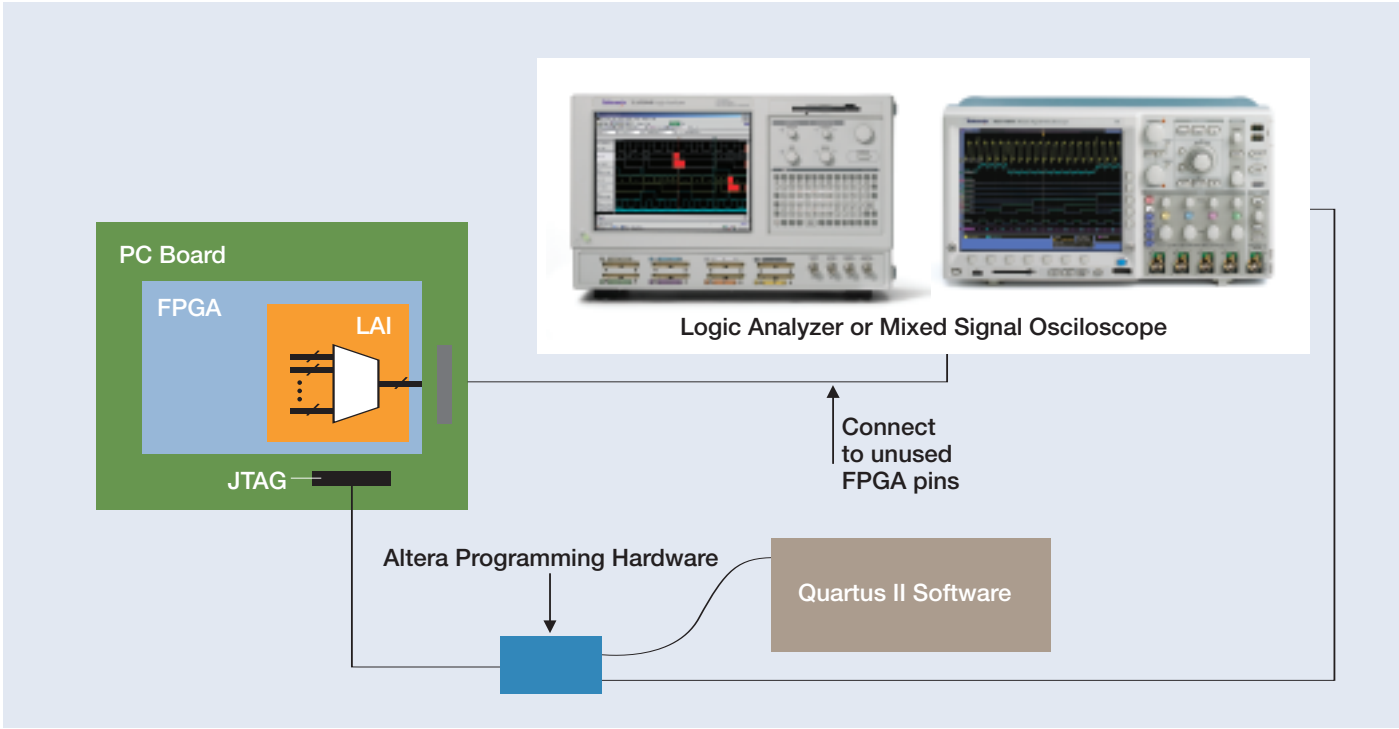

Figure 1. Logic Analyzer Interface and Hardware Setup.

Logic Analyzer Interface (LAI) Method

One of the more recent additions to FPGA debug techniques has been the introduction of the Logic Analyzer Interface (LAI) used to connect to external test equipment for logic analysis. The LAI uses a combination of the JTAG interface, as used in the SignalTap solution, and external signals, which connect directly to the Tektronix TLA Series logic analyzer or MSO Series mixed signal oscilloscope probe. The JTAG interface is used for control of the system, while the external test equipment probe is responsible for data capture. Figure 1 provides a high-level diagram of operation, showing how multiple LAIs can be developed for a design to support different system blocks or clock domains.

During the FPGA development, the user selects the number of pins they wish to set aside for the logic analysis. They then use the FPGA development tools to define the actual signals which they wish to look at as part of the system debug. The signals are then multiplexed onto the reserved FPGA pins.

During analysis, the external Logic Analyzer or Mixed Signal Oscilloscope uses FS2's FPGAView software to control the FPGA logic analyzer operation and to capture data. The external test equipment is responsible for triggering, data capture and signal de-muxing, thus reducing the overall logic, memory and routing resources required in the FPGA. While the LAI does require some FPGA resource; it is significantly less than the embedded logic analyzer solution.

The LAI approach allows for more flexibility with triggering and capture. The solution uses memory external to the FPGA to save the trigger data, which can either be within the test equipment itself, or in an external module, such as one provided by FS2. The trigger capability is also enhanced and offers both more options and levels than those provided with the embedded logic analyzer.

The LAI is capable of supporting high speed systems and acquiring data at speeds of over 200 MHz. In addition it has the capability of securing timing data, which means it is possible to look at the data on a bus and understand the latency between channels.

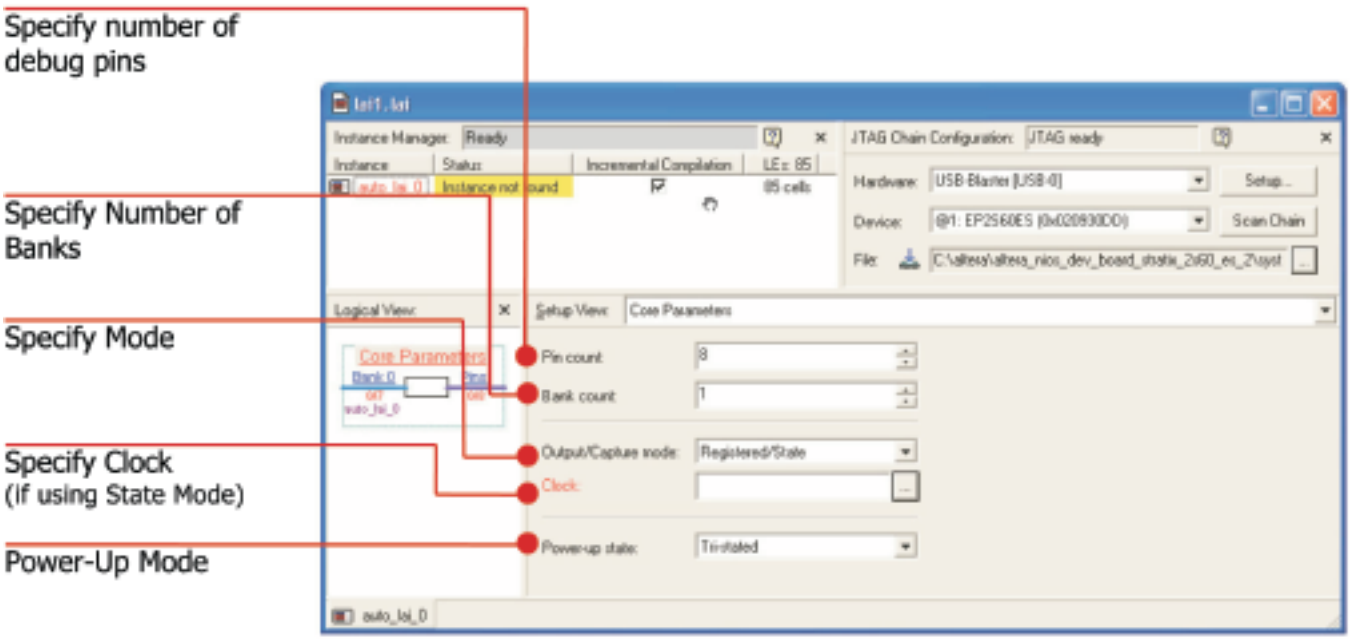

Figure 2. Example of Altera's Logic Analyzer Interface Editor used to define and insert test core.

The LAI can also be supported by incremental compilation, allowing you to add the interface later in the design and modify it with minimal compile times. However, you will want to plan ahead to ensure external pins are made available for the external test equipment probes.

Using FPGAView

Using FPGAView consists of these easy steps:

Step 1. Configure and insert the appropriate test core into your FPGA design

Step 2. Configure FPGAView to match your debug environment

Step 3. Establish the mapping of FPGA pins to MSO or TLA logic analyzer channels

Step 4. Make your measurement

Each of these steps is described in more detail in the following sections.

Step 1. Insert Core

The first step is to configure the test core and insert it into your design. In this example, you will want to use Altera's LAI editor to create a test core that best suits your need (See Figure 2). With most test cores you can specify the following parameters:

Pin Count: Signifies the number of pins you want dedicated to your external test equipment interface.

Bank Count: Signifies the number of internal signals that you want to map to each pin.

Output/Capture Mode: Selects the type of acquisition you want to perform. You can select Combination/Timing or Registered/State.

Clock: If you selected a capture mode of Registered/State, this allows you to select the sample clock for the test core.

After selecting the appropriate parameters for your debug requirements, you need to select which pins will be used by the test core for output. You will also need to select which signals are to be probed and groups those signals into banks.

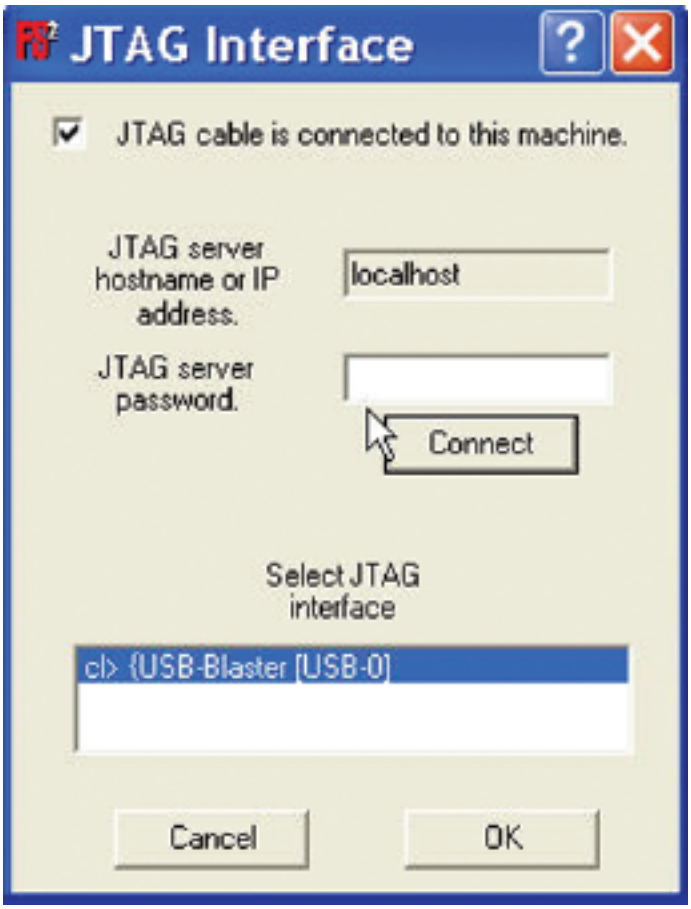

Figure 3. Configuring the connection to the JTAG programming cable.

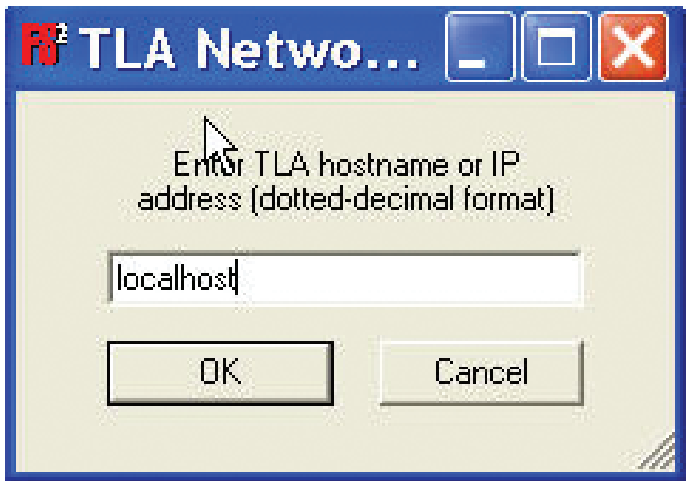

Figure 4a. Configuring the connection to the TLA.

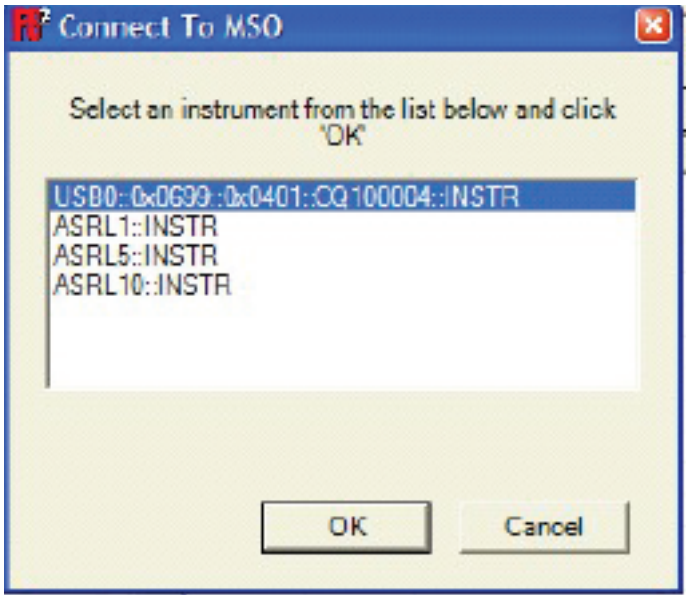

Figure 4b. Configuring the connection to the MSO.

Step 2. Configure FPGAView to match your debug environment

From the FPGAView window, you establish the connection to the JTAG programming cable (See Figure 3) as well as connecting to the external test equipment. Figures 4a and 4b show the connection to the TLA Series logic analyzer, MSO4000 Series mixed signal oscilloscope, or PC workstation. These configurations provide you with the flexibility needed to match your debug challenges.

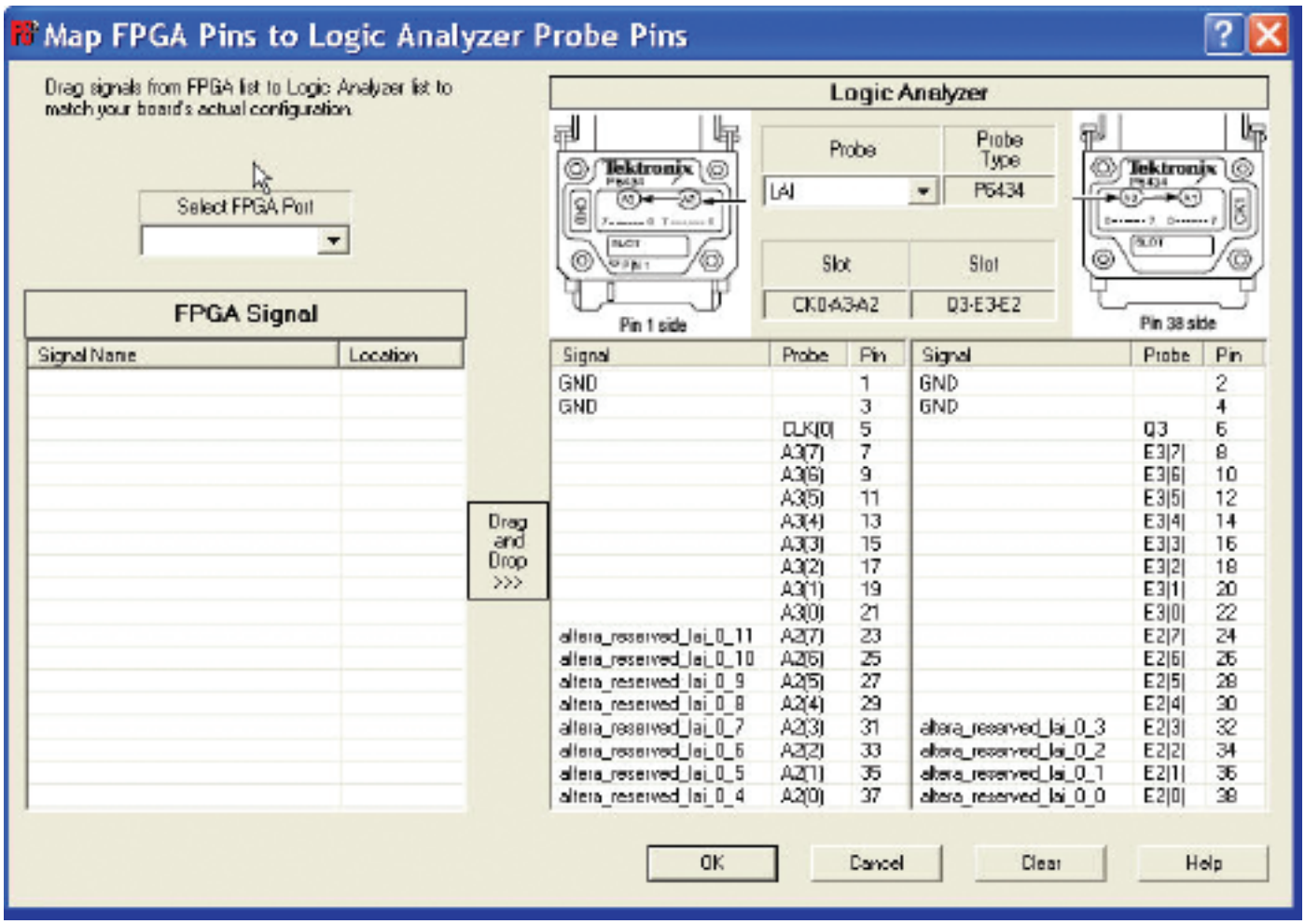

Figure 5. FPGAView maps pins quickly and easily.

Step 3. Map FPGA Pins to Logic Analyzer or Mixed Signal Oscilloscope

The next step is to map the physical connection between the FPGA pins and the TLA Series logic analyzer or MSO Series mixed signal oscilloscope. This will allow FPGAView to automatically update the signal names displayed on the TLA or MSO to match those of the signals in the FPGA design currently being monitored by the test core.

To do this, simply click on the Probes button to bring up a drag-and-drop window for connecting the test core output signal names with the correct channels on the test equipment (See Figure 5). This assignment process is only necessary once for a given target connection.

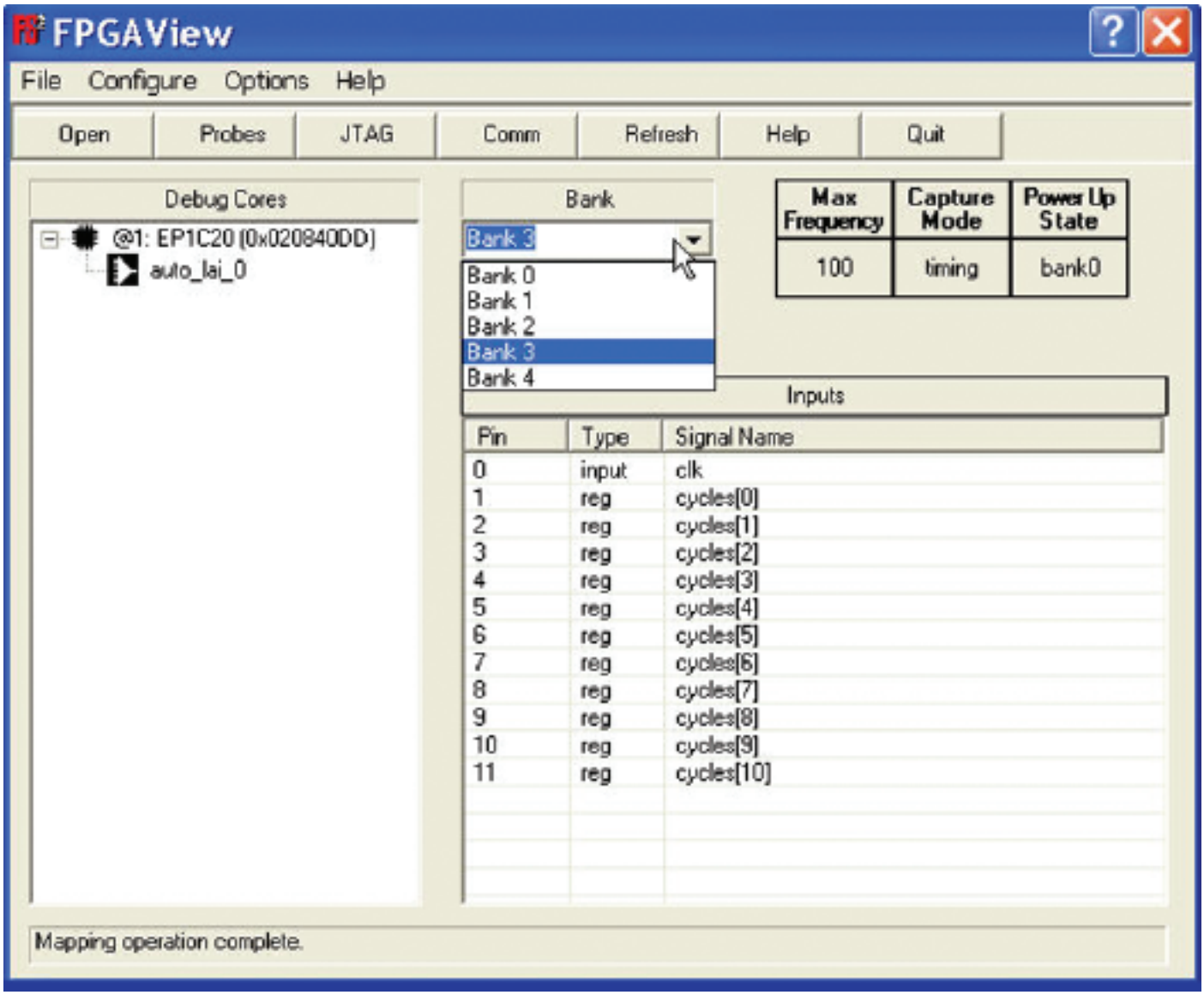

Figure 6. Select desired Bank of signals to measure.

Step 4. Make Your Measurement

The Bank list pull-down lets you select which Bank you want to measure. Once the Bank is selected, FPGAView communicates to your FPGA via the JTAG interface and configures the test core so that the desired Bank is selected.

FPGAView also programs the TLA Series logic analyzer or MSO Series mixed signal oscilloscope with these names into the assigned channels making it easy to interpret your measurement results. To measure a different set of internal signals, you simply choose a different bank of signals (See Figure 6). Correlating these FPGA signals with other signals in your system is done automatically by the full-featured TLA Series or MSO Series (See Figures 7a and 7b).

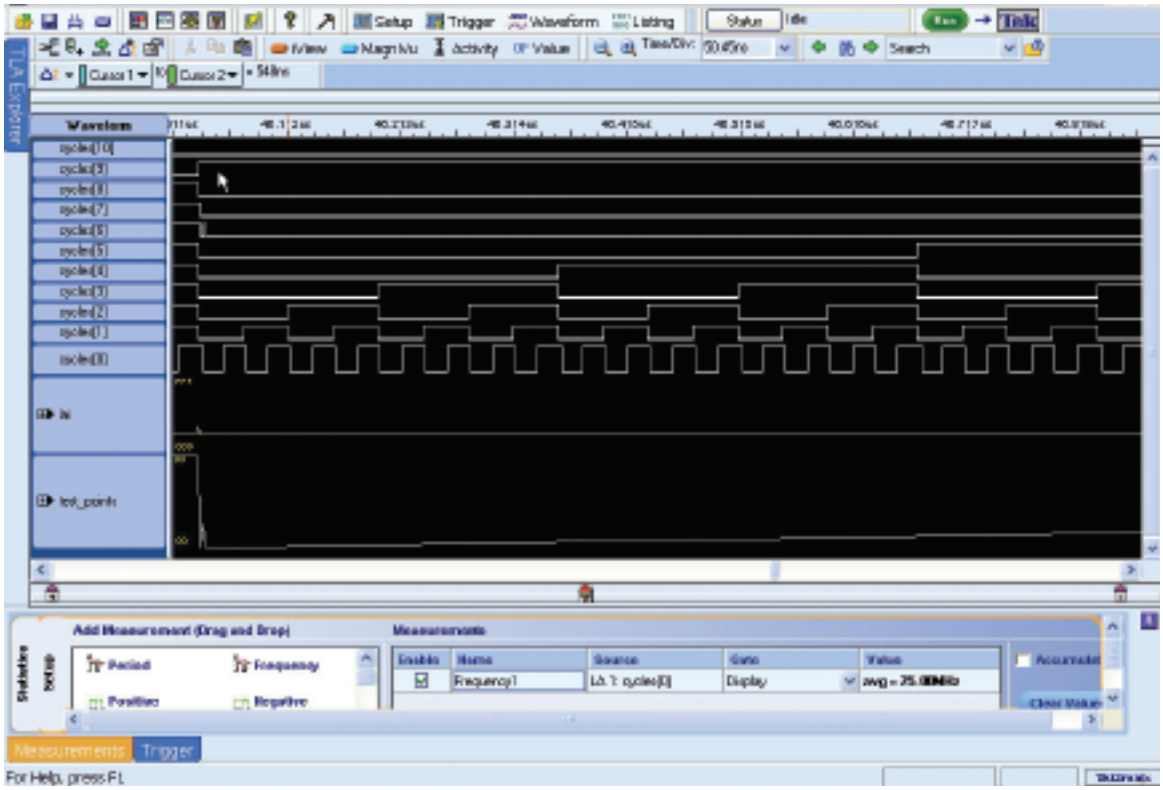

Figure 7a. TLA Series logic analyzer automates and simplifies many measurements.

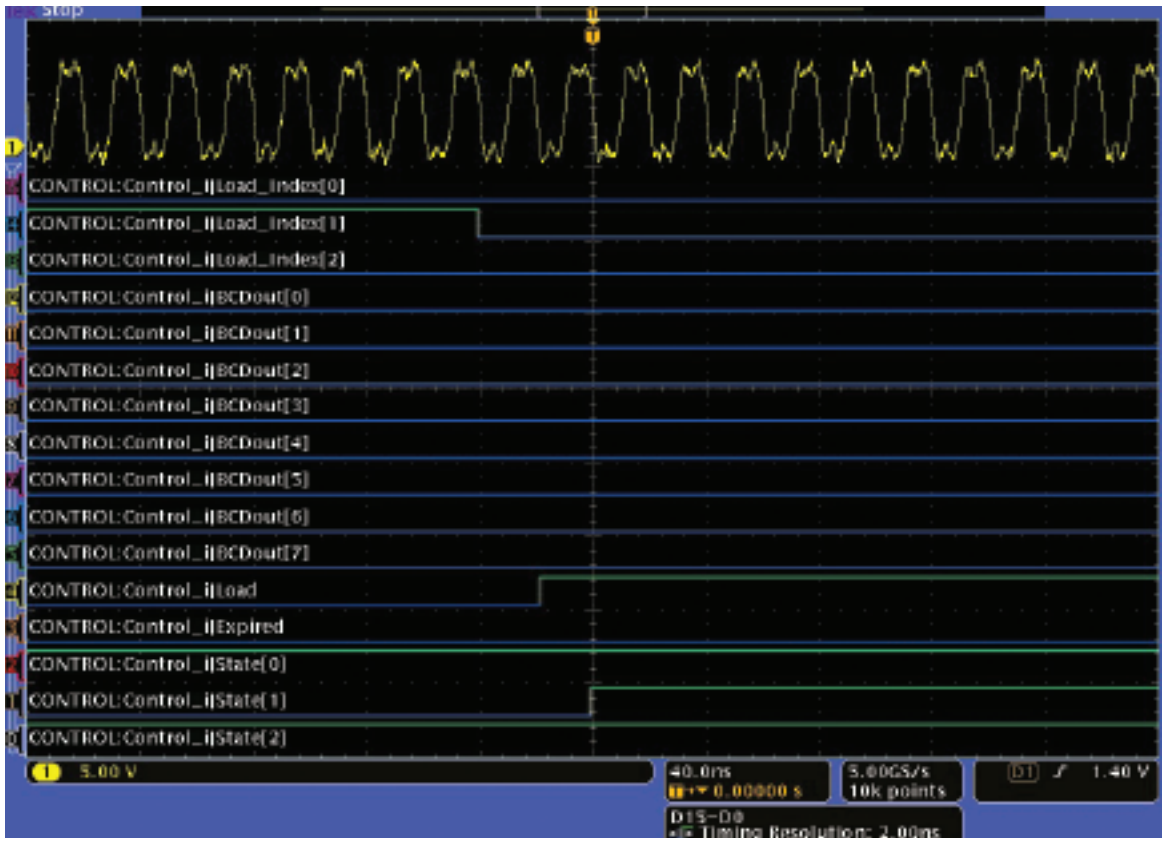

Figure 7b. MSO Series mixed signal oscilloscope and FPGAView simplify FPGA system debug.

Summary

Thoughtful planning during the design phase, and selection of the best debug methodology that suits your needs, will save you time during the debug phase and simplify the process. The various methodologies each have their own strengths and weaknesses, but the introduction of FPGAView makes the Logic Analyzer Interface (LAI) approach more appealing than ever. The ability to correlate internal FPGA signal activity to board-level signals delivers total insight into your embedded design. The following table provides a comparison of debug techniques available with Altera FGPA's.

Comparison of Debug Techniques Available with Altera FPGA's

Feature	SignalProbe	SignalTap II Embedded Analyzer	Logic Analyzer Interface (LAI)	Description
Large Sample Depth	○	◒	●	An external logic analyzer used with the LAI has a bigger buffer to store more captured data than SignalTap II Logic Analyzer. No data is captured or stored with SignalProbe.
Ease in Debugging Timing Issue	○	◒	●	An external logic analyzer used with the LAI provides you with access to timing mode, enabling you to debug combined streams of data.
Minimal Effect on Logic Design	●	●*	●*	The LAI adds minimal logic to a design, requiring fewer device resources. The SignalTap II Logic Analyzer has little effect on the design when it is set as a separate design partition using incremental compilation. SignalProbe incrementally routes nodes to pins, not affecting the design at all.
Short Compile and Recompile Time	●	●*	●*	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	○	●	●	The SignalTap II Logic Analyzer offers triggering capabilities that are comparable to basic logic analyzers.
I/O Usage	◒	●	◒	No additional output pins are required with the SignalTap II Logic Analyzer. Both the LAI and SignalProbe require I/O pin assignments.
Acquisition Speed	○	●	◒	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 signal integrity issues may limit this.
No JTAG Connection Required	●	◒	◒	An FPGA design with the SignalTap II Logic Analyzer or the LAI requires an active JTAG connection to a host running the Quartus II software. SignalProbe does not require a host for debugging purposes.
External Equipment	◒	●	◒	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 software. SignalProbe and the LAI require the use of external debugging equipment.

* When used with incremental compilation.

● Suggested best tool for the feature.

◒ Tool is available for that feature, but may not give the best results.

○ Feature is not applicable for the selection tool.

Contact Tektronix:

ASEAN / Australasia (65) 6356 3900
Austria +41 52 675 3777
Balkans, Israel, South Africa and other ISE Countries +41 52 675 3777
Belgium 07 81 60166
Brazil & South America (11) 40669400
Canada 1 (800) 661-5625
Central East Europe, Ukraine and the Baltics +41 52 675 3777
Central Europe & Greece +41 52 675 3777
Denmark +45 80 88 1401
Finland +41 52 675 3777
France +33 (0) 1 69 86 81 81
Germany +49 (221) 94 77 400
Hong Kong (852) 2585-6688
India (91) 80-22275577
Italy +39 (02) 25086 1
Japan 81 (3) 6714-3010
Luxembourg +44 (0) 1344 392400
Mexico, Central America & Caribbean 52 (55) 5424700
Middle East, Asia and North Africa +41 52 675 3777
The Netherlands 090 02 021797
Norway 800 16098
People's Republic of China 86 (10) 6235 1230
Poland +41 52 675 3777
Portugal 80 08 12370
Republic of Korea 82 (2) 6917-5000
Russia & CIS +7 (495) 7484900
South Africa +27 11 206 8360
Spain (+34) 901 988 054
Sweden 020 08 80371
Switzerland +41 52 675 3777
Taiwan 886 (2) 2722-9622
United Kingdom & Eire +44 (0) 1344 392400
USA 1 (800) 426-2200
For other areas contact Tektronix, Inc. at: 1 (503) 627-7111
Updated 12 November 2007

For Further Information
Tektronix maintains a comprehensive, constantly expanding collection of application notes, technical briefs and other resources to help engineers working on the cutting edge of technology. Please visit **www.tektronix.com**

Copyright © 2008, Tektronix. All rights reserved. Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supersedes that in all previously published material. Specification and price change privileges reserved. TEKTRONIX and TEK are registered trademarks of Tektronix, Inc. All other trade names referenced are the service marks, trademarks or registered trademarks of their respective companies.

01/08 DM 54W-21382-0

Tektronix®
Enabling Innovation