

Welcome to the “Debugging FPGA-Based PowerPC 440 Designs: *Using the Agilent Logic Analyzer with ChipScope Pro and Eclipse Software Development Kit*” session. In this session we’ll introduce a method for using an Agilent logic analyzer along with the Xilinx hardware and software debug tools to debug designs using the PowerPC 440 that’s built into the new Virtex-5 FXT FPGAs.

Agenda

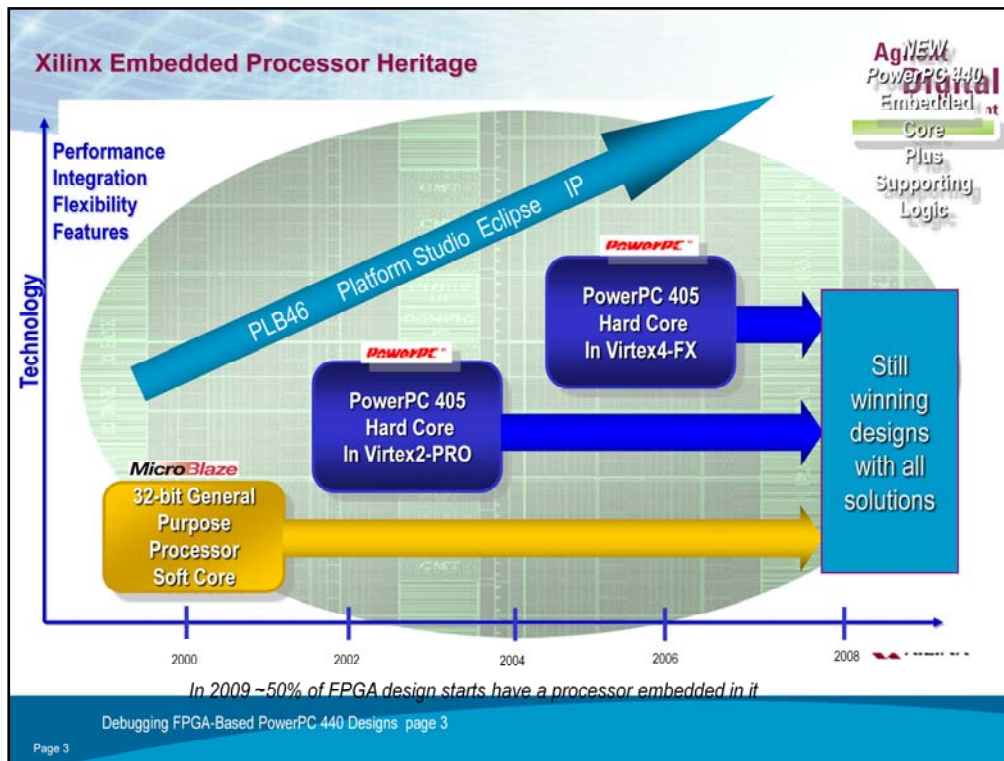
Agilent
Digital
Measurement
Forum

Xilinx overview featuring Virtex-5 FXT

Logic analysis with Xilinx

Example system





First let's have a review of where Xilinx stands with regards to embedded processing.

Xilinx has been working in processors since the year 2000 when we began work on the Microblaze soft processor. In 2001, we released the Virtex2PRO family of devices that featured (among other industry firsts) an embedded, industry standard PowerPC405. We've been supporting both soft and hardened processors since then.

The first market is an embedded market that requires a low power soft, highly configurable solution, which is of course, Microblaze.

The second is an embedded market that requires an industry standard solution that can run industry standard compilers and OS's, move huge amounts of data fast, and achieve a higher level of performance than a soft processor can. Our solution to that market is of course the PowerPC.


We're onto our third generation of a PowerPC offering and this generation is both evolutionary and revolutionary. It's evolutionary in that the PowerPC in Virtex-5FXT is the PowerPC440. In many respects it's compatible with the PowerPC405. However the new, integrated block of interconnect logic built around the PowerPC440 is revolutionary, providing an unprecedented level of integration and performance, never before seen in an FPGA. This new block will enable an entire new class of FPGA hosted microprocessor designs.

It's worth pointing out that regardless of whether you're doing MicroBlaze or PowerPC, you use the embedded development kit which has the same PLB-based peripherals. So designs can migrate easily between the PowerPC and the Microblaze. The software we used is called Platform Studio. Be sure to point this out as it's referred to several times later in the presentation.

Introducing Virtex®-5 FXT Platform FPGAs

Agilent
Digital
Measurement
Forum

LXT SXT FXT



Built with ultra-fast ExpressFabric™ technology
Fully integrated PCI Express® Endpoint Blocks
Fully integrated Tri-Mode (10/100/1000 Mbps) Ethernet MAC blocks
High-Performance 550 MHz Clock Management Tile (CMT) with PLLs
Flexible SelectIO with ChipSync technology

XILINX®

Debugging FPGA-Based PowerPC 440 Designs page 4
Page 4


We'd like to first introduce you to the new Virtex-5 FXT devices. Virtex-5 FXT was formally announced back in April. Since then it's had one of the fastest ramps of any new Xilinx Virtex family.

The Virtex-5 FXT devices share some common features with the other "T" family members of the Virtex-5 family which, by the way, are called LXT and SXT. So the LXT, SXT, and FXT members of the Virtex-5 family share all of the features shown on the slide we're looking at right now.

First, all of the "T" family members are built with Virtex-5's Express Fabric technology. The new 6-input LUT-based logic cell combined with the new more efficient routing structures that give the Virtex-5 fabric the highest performance possible.

Introducing Virtex®-5 FXT Platform FPGAs


Agilent Digital Measurement Forum



High-Speed Gigabit Transceivers GTX

- 8 ~ 24 built-in GTX Transceivers delivering up to 6.5 Gbps

Higher Performance



FXT

CrossFabric™ technology
 Express® Endpoint Blocks
 10/100/1000 Mbps Ethernet MAC blocks
 High-Performance 550 MHz Clock Management Tile (CMT) with PLLs
 Flexible SelectIO with ChipSync technology

XILINX®

Debugging FPGA-Based PowerPC 440 Designs page 5


Page 5

The FXT Family has faster gigabit transceivers, which we call the GTX's. While the GTP transceivers on the LXT and SXT operate at up to 3.75 Gb/sec, the new GTX transceivers on the FXT's operated at up to 6.5 Gb/sec. This will allow the FXT devices to capture additional markets, for instance PCI Express Gen-2, which requires 5.0 Gb/sec.

FXTs have up to 24 GTX's.

Agilent
Digital
Measurement
System


Introducing Virtex®-5 FXT Platform FPGAs



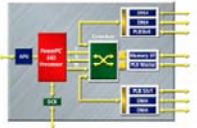
High-Speed Gigabit Transceivers GTX

- 8 ~ 24 built-in GTX Transceivers delivering up to 6.5 Gbps

Higher Performance



FXT




Integrated PowerPC 440 Processor Blocks

- 1~2 IBM PowerPC 440 processor cores delivering 1,100 DMIPS @ 550MHz
- Advanced processor block architecture with 128-bit connectivity crossbar switch

Higher Performance

ProcessFabric™ technology
 Express® Endpoint Blocks
 10/100/1000 Mbps Ethernet MAC
 High-Performance 550 MHz Clock Management Tile (CMT) with PLLs
 Flexible SelectIO with ChipSync technology



Debugging FPGA-Based PowerPC 440 Designs page 6

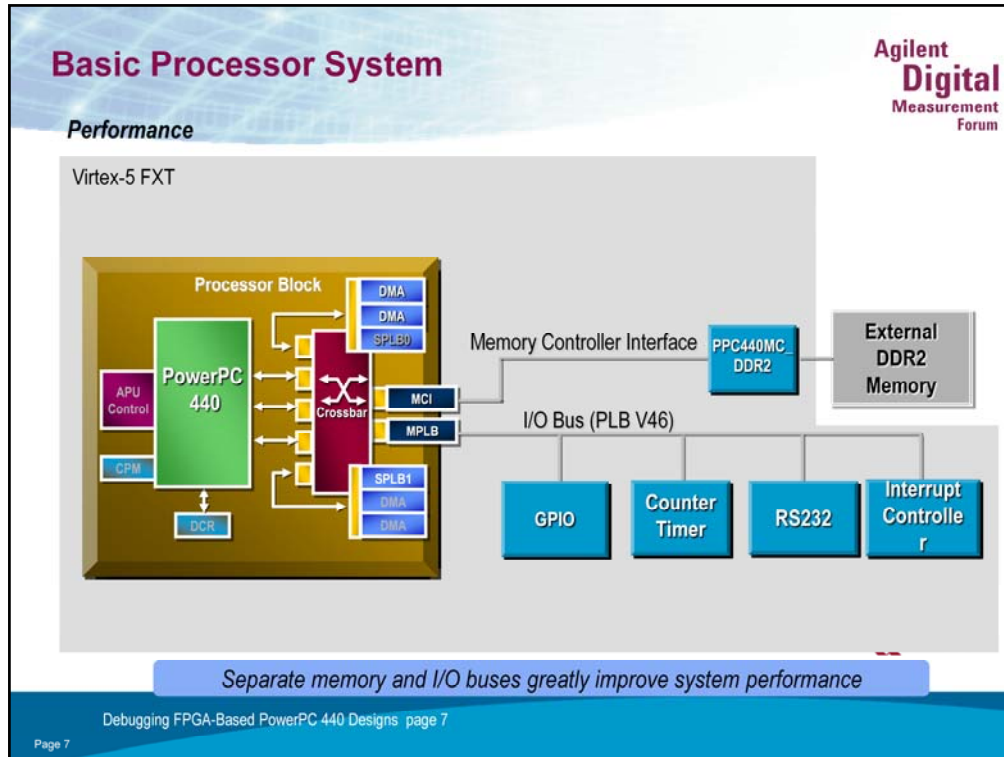
Page 6

And of course, the Virtex-5 FXT also has the new embedded PowerPC 440 block.

There are 5 members of the Virtex-5 FXT family and we should point out that, in fact, the 3 largest members of the PowerPC 440 family have TWO embedded PowerPC 440 embedded processor blocks. And what we'll see in the coming slides is that, unlike our PowerPC 405 implementations, the PowerPC 440 embedded block has a great deal of supporting logic built right into the embedded block. This allows your design to run faster and makes it easier to implement your design.

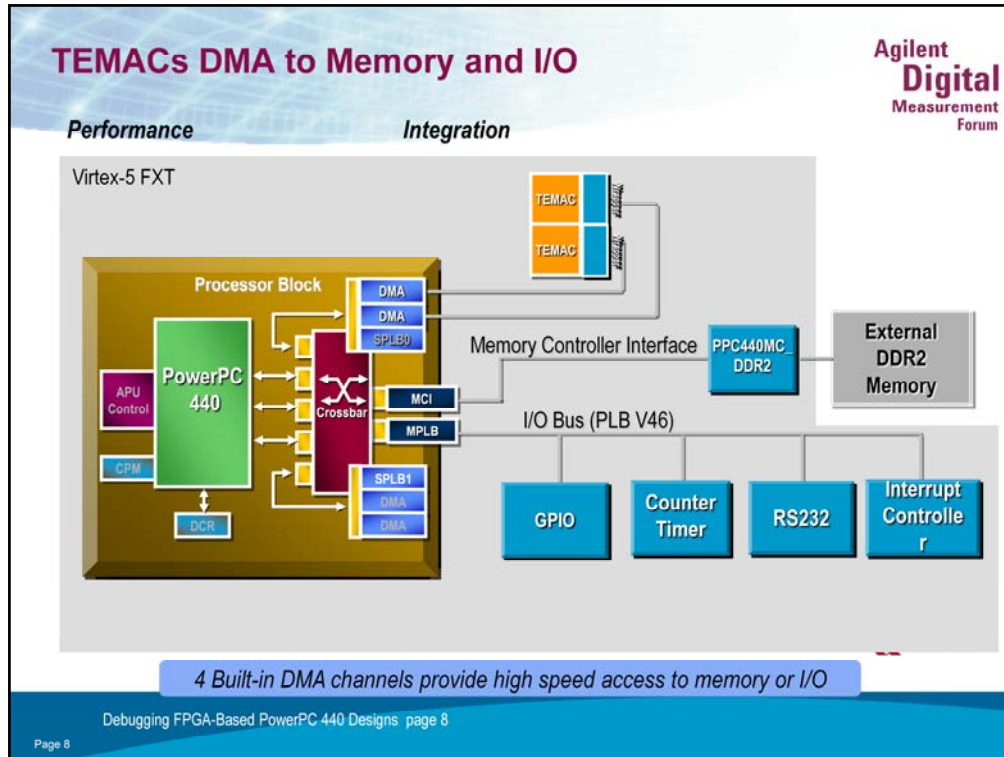
We'll look in greater detail at what this new logic consists of in the coming slides, and then we'll show how to use the Agilent and Xilinx tools to debug designs with the PowerPC 440 executing code.

This isn't a session on the inner workings of the PowerPC 440 microprocessor, but we should point out that the PowerPC 440 is a superscalar industry standard RISC processor that operates at up to 1100 DMIPs when operating with a 550 MHz clock. We currently have support for Linux and VxWorks built into our Embedded Development Tools.



This is a baseline embedded system. A processor, some DDR2 memory and some peripherals. Note that the peripherals are connected to the I/O or Master PLB interface, while the memory is connected to the memory controller interface bus. Nothing too fancy. And also note how the processor accesses the memory and peripherals through the crossbar.

This looks OK. Now what about the peripheral blocks connected to the other crossbar inputs? <<advance to next slide>>

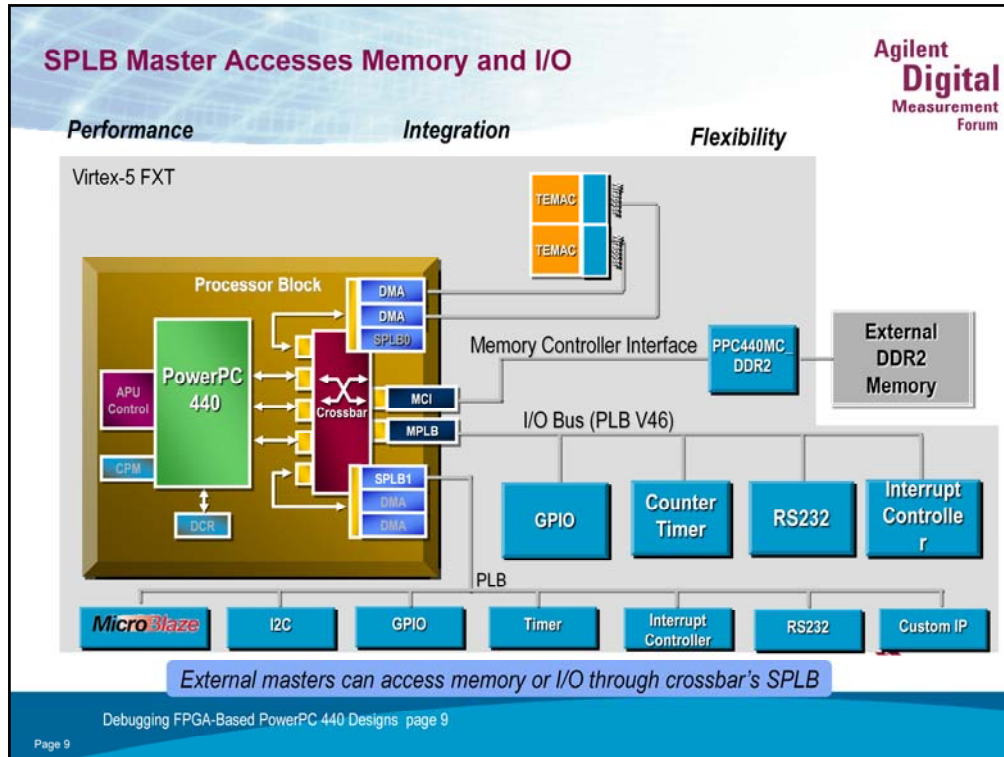


In many designs, FPGA and otherwise, it's often desirable for not only the processor, but other design subsystems to have access to the processor memory. To enable that, we have additional inputs to the embedded processor block. One of the inputs are DMA controllers. The DMA controllers move data between peripherals and the external memory very efficiently.

As an illustration, in our simple example we've now added two embedded TEMACS to the two DMA channels in the upper peripheral block. (There are similar DMA channels in the lower peripheral block) The TEMACS are connected to the processor block through a thin wrapper that is provided with our embedded development tools.

There's still an entirely unused peripheral block connected to the bottom input to the crossbar. What are some things that can be hooked to it? Well we illustrated the use of the DMA interfaces on the upper peripheral block. On the lower peripheral block, we'll illustrate the use of the slave PLB.

The slave PLB allows the PowerPC 440's peripheral block to appear as a slave on some other PLB. What might another PLB be in this context? <<advance>>



Well, like we said earlier, all of Xilinx's processor systems have standardized on the PLB. We also mentioned that you can have as many Microblazes in an FPGA as will fit, so here we show a complete MicroBlaze system with the PowerPC's memory and I/O appearing as a peripheral to the MicroBlaze system. The MicroBlaze has access to everything that the PowerPC processor block has access to.

This is a poor-mans multiprocessing system. We have more peripheral support for multiprocessing. This example is really just meant to illustrate the use of the Slave PLB interface, and that it allows "multi-ported" access to the memory and I/O of the PPC440 processor block.

Hopefully at this point you're starting to think about some things you might be able to use the PPC 440 Embedded Block for. Let's look at an example of what one of our customers used it for.

Actual Customer Configurable System-on-Chip Platform

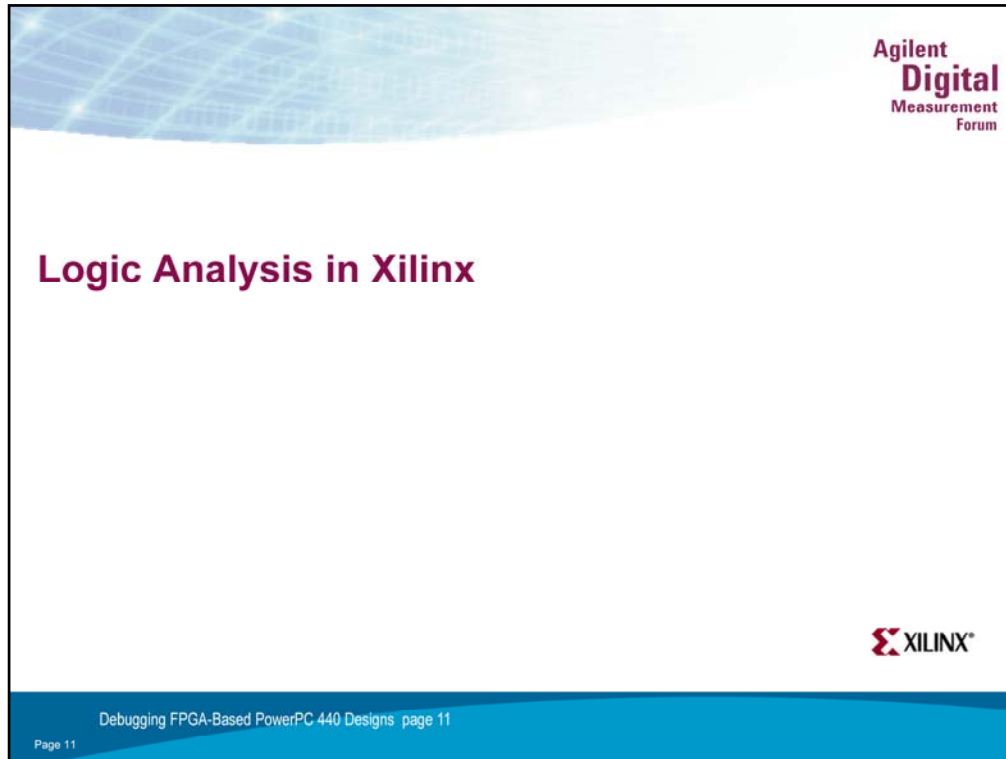
- Design ported from Virtex-4 FX60 to Virtex-5 FX70T in less than 3 weeks
- Virtex-5 FXT Integrated platform reduced logic utilization by 25%
- Over 90% FPGA resources available for user application
- PowerPC 440 System using integrated Gigabit Ethernet with Open Source Linux

Agilent
Digital
Measurement
Forum

Debugging FPGA-Based PowerPC 440 Designs page 10

Page 10

Here's what one customer did. They had a PowerPC405 design in a Virtex-4 FX60 and they wanted to migrate to the PowerPC 440 and the embedded capabilities that came with that. Since the embedded development tools are the same, and the IP is the same they were able to migrate their design the Virtex-5 FX70T in less than 3 weeks. The embedded logic built into the FXT left a lot more logic for their user designs. They also got the design running with Open Source Linux, which Xilinx is now beginning to do work with.

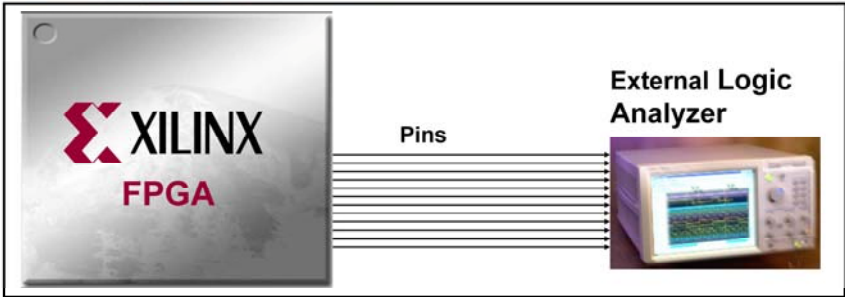


So now lets begin to dig deeper into the logic analysis capabilities of Xilinx FPGAs.

Old Logic Analysis Method

Dedicated pins connected to logic analyzer

Agilent Digital Measurement Forum



The diagram illustrates the 'Old Logic Analysis Method' for debugging a Xilinx FPGA. On the left is a square component labeled 'XILINX FPGA'. On the right is a photograph of an 'External Logic Analyzer' device. Multiple horizontal lines, labeled 'Pins', connect the FPGA to the logic analyzer, representing dedicated I/O pins used for debugging.

- Requires Extensive Dedicated I/O for Debug
 - Driving signals to external I/O introduces additional problems
- Inflexible solution
 - Difficult or impossible to add additional debug pins if needed
- Limited visibility to on-chip activity

XILINX®


Debugging FPGA-Based PowerPC 440 Designs page 12

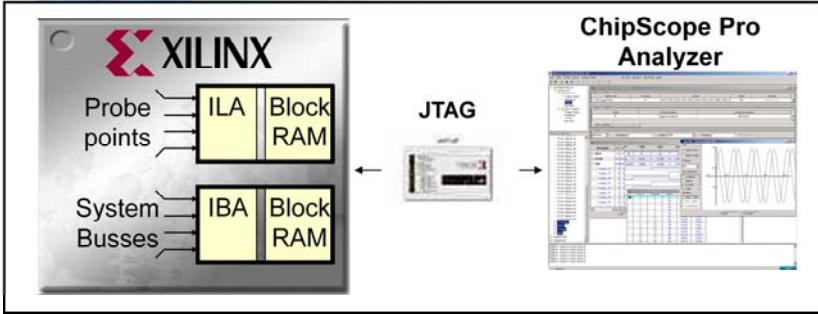
Page 12

FPGA developers routinely identify logic analyzers as one of their top 2 in-circuit debug tools (the other top tool is usually a scope). FPGA customers determine early in their development process how many pins they want to dedicate to logic analysis. These customers often build in a test MUX inside the FPGA that allows a customer to select a single group of signals to bring out to the pins at a single time. This technique minimizes the number of pins dedicated for debug. Typical pin count for debug is in the range of 8 to 32 pins while internal nodes feeding the MUX is typically in the order of 32 to a hundred channels. When the customer finishes debug, the test MUX circuitry and pins for debug remain in the design. Taking them out would change the design characteristics and require a new round of debug.


ChipScope Pro On-Chip Debug

Integrated Logic Analyzer Core





- No I/O pins required for debug
 - Access via the JTAG Port
- On-Chip access to every signal and node in the FPGA design
 - Driving signals to external I/O introduces additional problems
- Add and remove cores at any time in the design process



Debugging FPGA-Based PowerPC 440 Designs page 13

Page 13

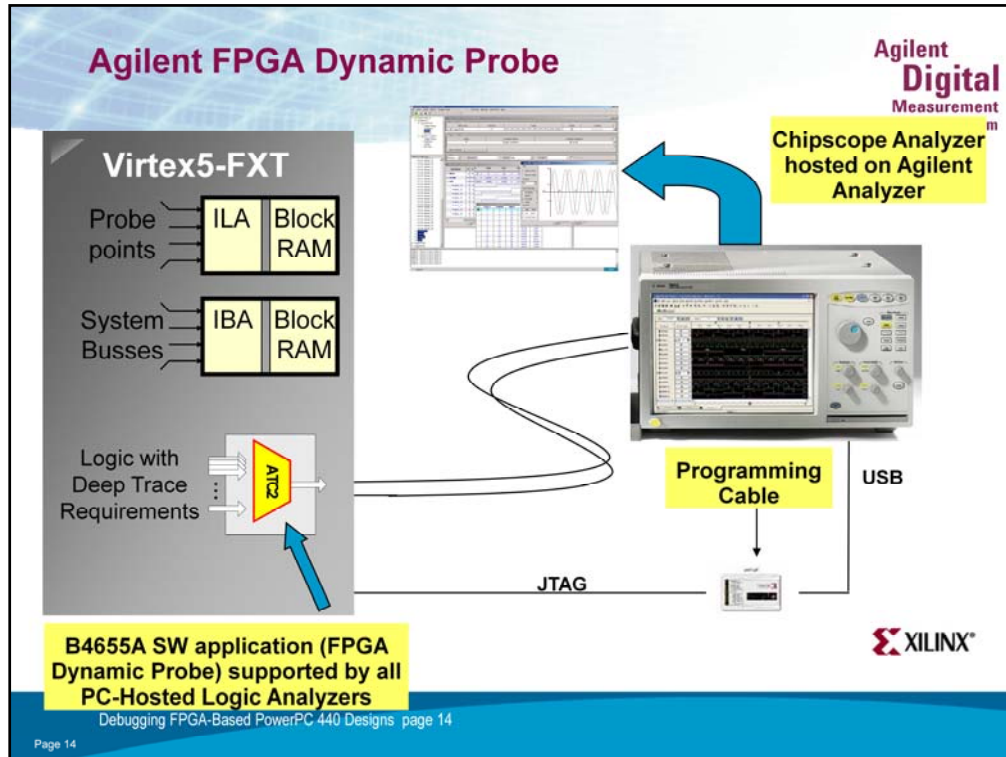
Xilinx recognized early on that we could make use of the resources inside the FPGA and provide users low-cost, rudimentary logic analysis capabilities. This spawned the ChipScope family of products.

ChipScope makes use of the internal blockrams that are in all Xilinx FPGAs and uses them for storage of the logic analysis trace data. (The rest of the logic analyzer circuitry is built from gates and flip-flops).

The ChipScope logic analyzer is controlled via the FPGA's JTAG interface. Once the blockrams are filled with trace data, it can then be transferred to the user's PC for formatting and display. The JTAG tool that communicates with the FPGA is known by customers as a "cable." Note that with ChipScope, there are two kinds of analyzer: ILA which is an internal Logic Analyzer. This is used for displaying signals within customer's logic. IBA is the internal Bus Analyzer, which is used for connecting to buses within customer's Embedded Processor designs.

We'll be using both capabilities in this session.

In some cases, ChipScope's limited memory depth and dedicating part of the FPGA memory to debug instead of the design, can limit a designer's ability to debug effectively. That's where Agilent stepped in!

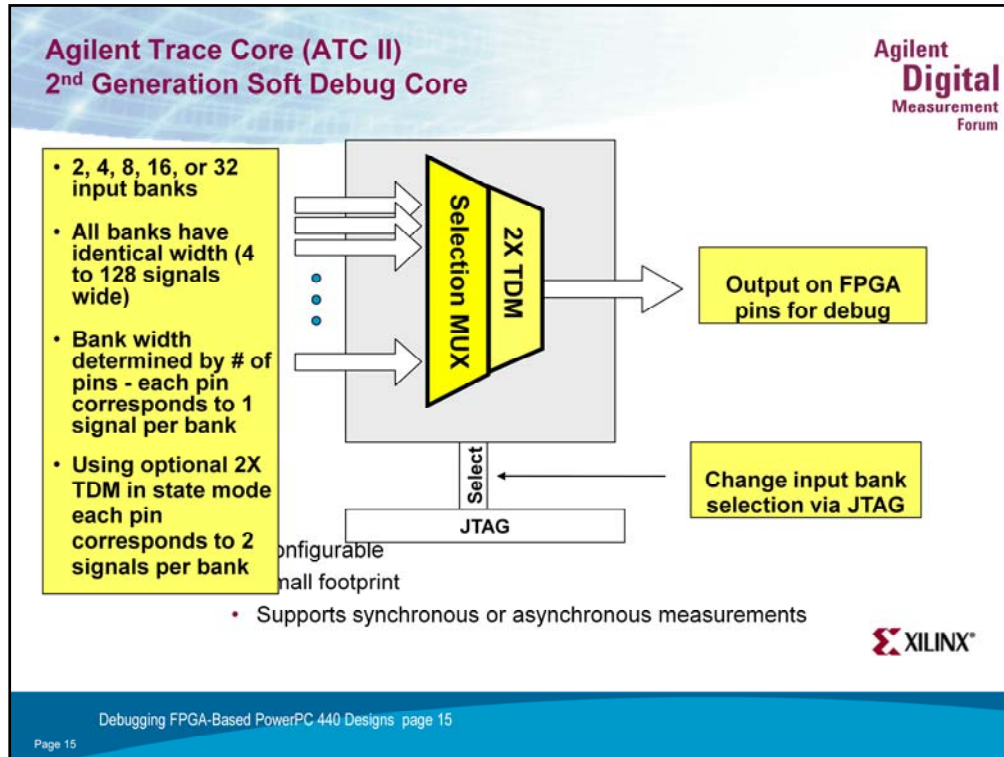


Agilent produced a product called FPGA Dynamic Probe addresses these limitations by providing a complete debug environment incorporating the logic analyzer. FPGA Dynamic Probe works hand-in-hand with ChipScope.

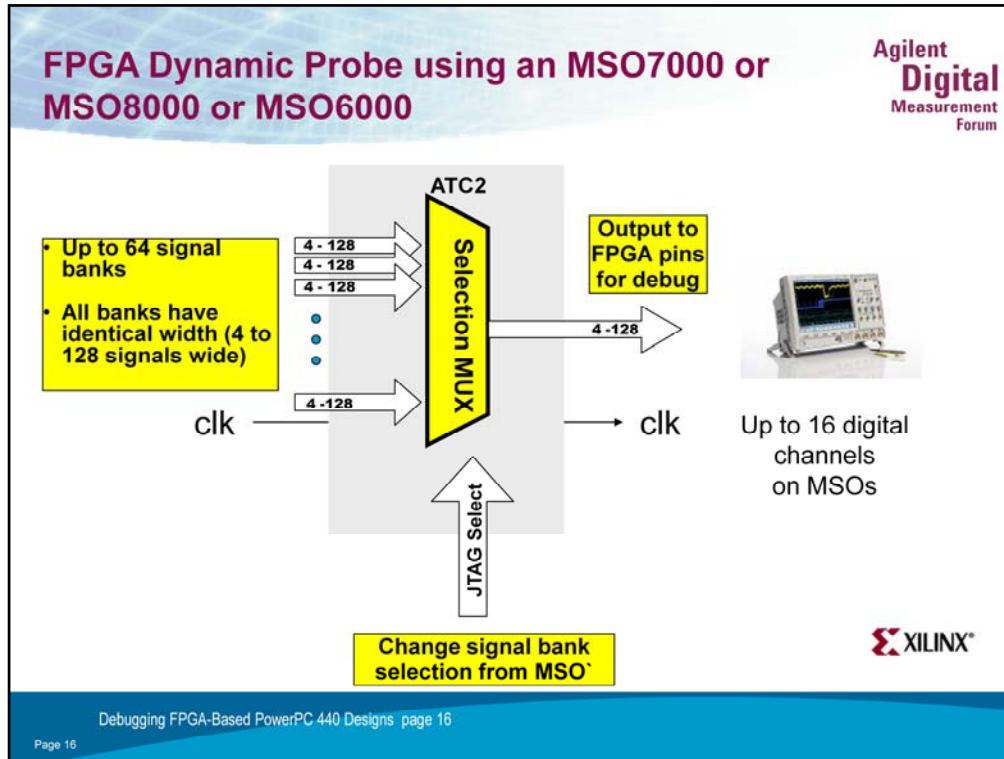
Let's take a look at the components of the solution to get a better understanding of how this works.

Making use of FPGA Dynamic Probe starts with an assessment of the internal nodes of the FPGA that may benefit from logic analysis access, and the type of measurement (state analysis or timing analysis) that needs to be made. The user then inserts a core called Agilent Trace Core II (ATC2) into the design. The inputs to the core are the potential internal nodes to be probed by the logic analyzer, and the outputs are routed to a set number of physical pins dedicated to debug. The debug pins are physically connected to the logic analyzer using any of Agilent's logic analyzer probes (note that Agilent's 17-channel Soft Touch probe makes an excellent choice with its small footprint and high fidelity).

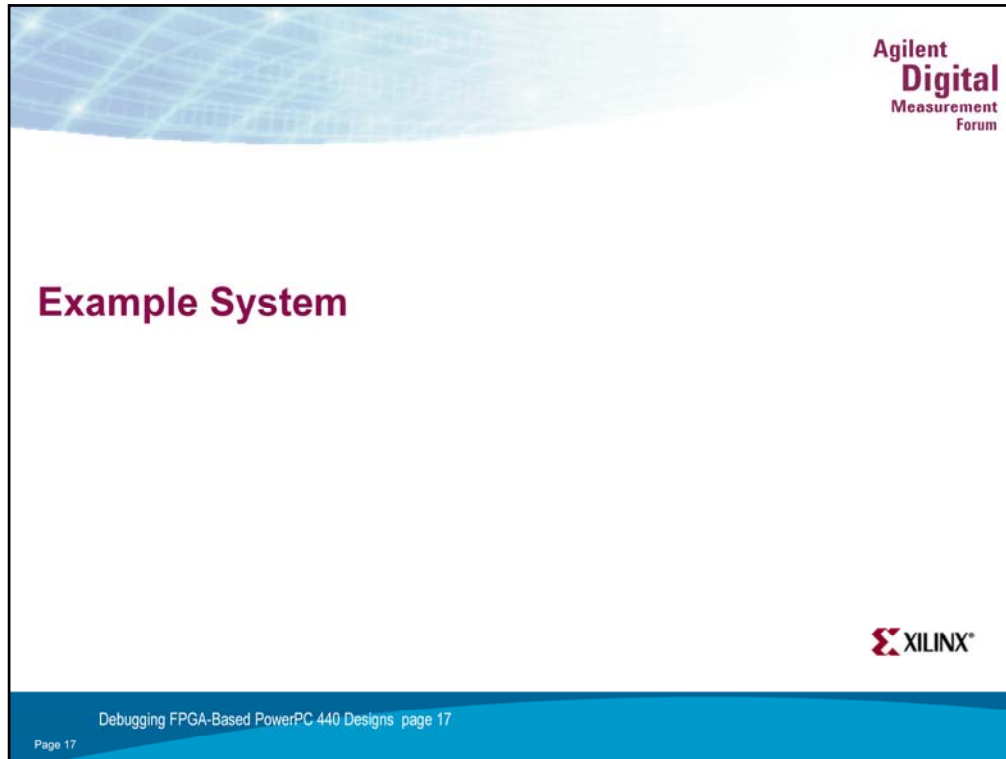
The cores can be thought of as MUXes that are controlled via the JTAG connection to the FPGA offered through the standard Xilinx JTAG cable used to download and program the FPGA. This whole system is controlled via software that runs on the logic analyzer. The new 16900 Series mainframes are supported as are the 1680 and 1690 Series from Agilent.



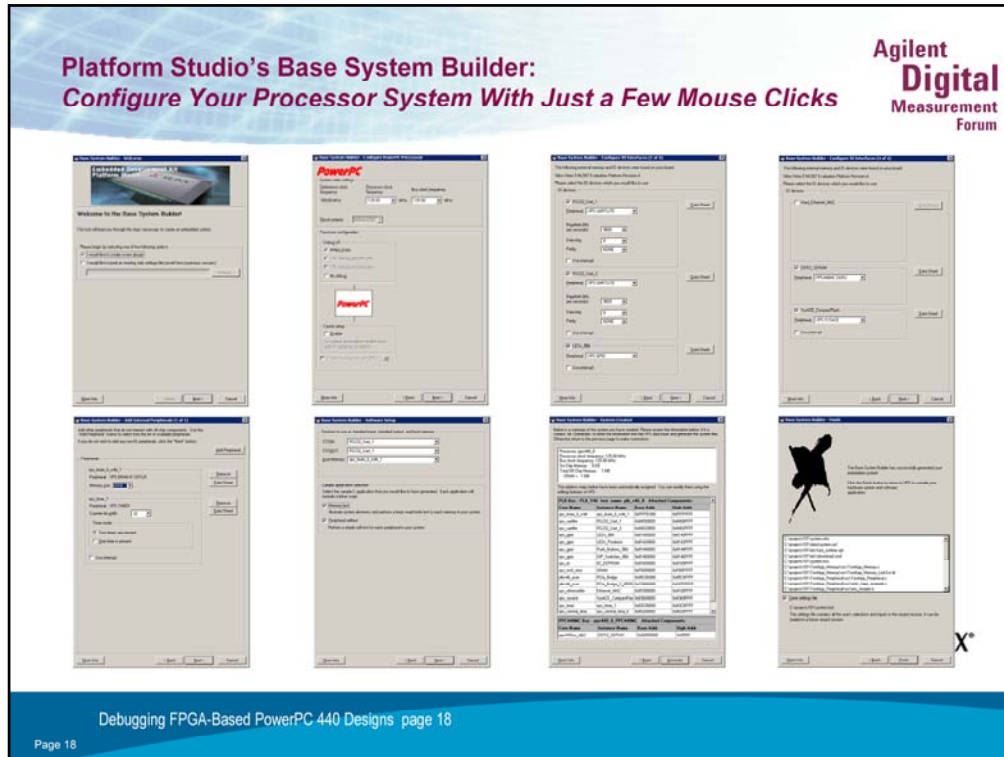
Here's a closer look at the Agilent Trace Core 2. The users signals are connected to the MUX inputs. The output of the MUX drives I/O pins on the FPGA. The Which Mux is used is controlled by JTAG.



We should point out that the ATC2 core can be used with more than a Logic Analyzer. ATC2 can also be used with the Agilent MSO. The same capabilities described in the previous slides also exist in the MSO's solution.

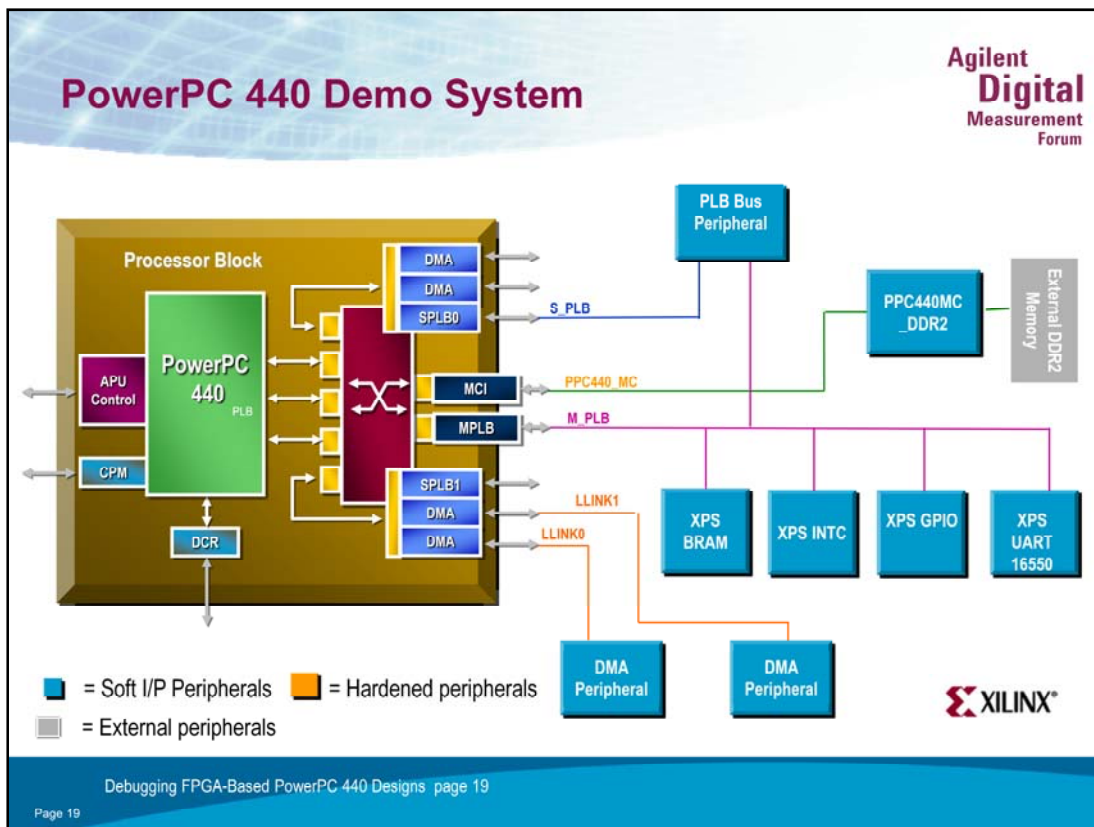


So now that we understand the Virtex-5FXT's processor block, and we understand the logic analysis capabilities of the Agilent and Xilinx tools, let's look at an example system.



Before we get started, the question always comes up regarding how to actually build a processor system in an FPGA. Well in the Xilinx Platform Studio, it's very easy. There's a capability built into Platform Studio called "Base System Builder". Base system builder allows you to build platform system for a large number of the development boards out there. Base System Builder knows about each development board's capabilities and steps you through a wizard that allows you to select the different features that you want your processor system to have. Referencing the slides above, the steps that you would go through would be:

- 1) What board do you have
- 2) What processor do you want to use (PPC405, PPC440, Microblaze)
- 3) What peripherals do you want to use
- 4) What memory in your system (BRAM, DDR, DDR2, Flash)
- 5) Do you want some sample software applications?
- 6) Do you want them in on chip memory or off chip (Base system builder will create the appropriate linker script)
- 7) It provides a summary of all your selections, along with an address map
- 8) And you now have a system designed and all of the appropriate files have been set up so you can begin using Platform Studio with your specific design



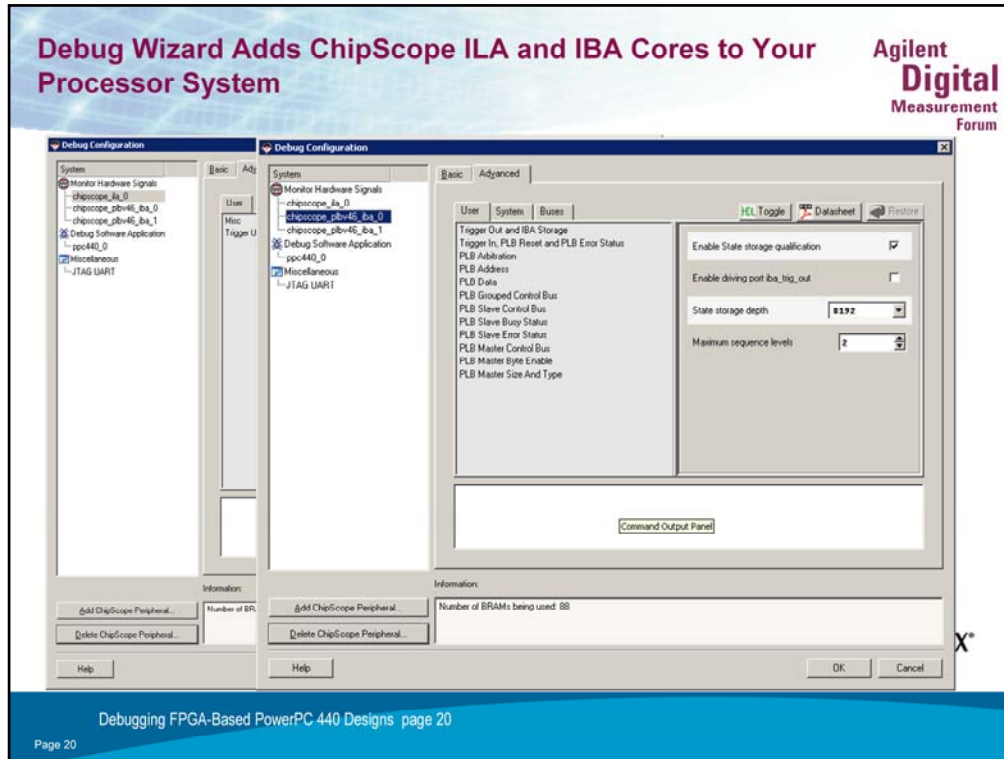
This is our system that we're going to be debugging. You should recognize the PowerPC 440 Embedded block from our discussion earlier. We'll have our external memory connected to the Memory I/F, also called the MCI or Memory Controller Interface.

We'll have a suite of generic peripherals connected to the I/O or MPLB interface. Those peripherals you can see are a BRAM, an interrupt controller, some general purpose I/O and a UART.

We'll have a generic PLB master connected to the PLB SLV0 interface and some Note that this same PLB peripheral is a slave on the MPLB interface. This is so commands can be written to that peripheral instructing it to issue commands on the slave PLB interface.

DMA exercisers connected to the lower DMA channels. We will set them up such that they will be doing wraparound DMA transfers from DDR2 memory through the DMA peripheral, and back to memory.

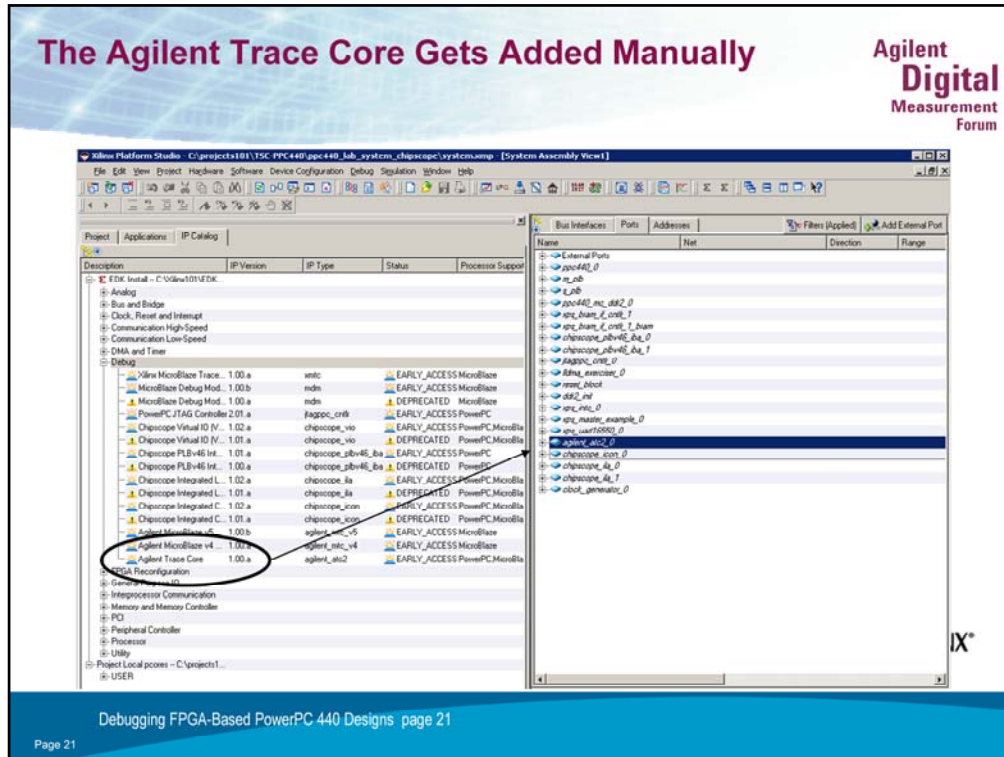
The PowerPC 440 embedded block is designed such that we can have traffic on both the MCI and the MPB at the same time. Further, using all of the debug tools available to us, we can display and debug all of these interfaces at the same time as well. Let's see how.



Similar to the base system builder wizard, our Platform Studio software has built into it the “Debug Wizard” which allows you to easily add debugging peripherals to your design. Since you’re doing a processor based design, at a minimum you will probably want to add an IBA core—IBA: Internal Bus Analyzer—to your designs’ PLB’s. Additionally, Debug Wizard enables you to add ILA –internal logic analyzer—cores to your design as well. You’d use that for debugging a peripheral of your own design, or of the DMA interfaces (which is what we’re going to use it for).

The Agilent Trace Core Gets Added Manually

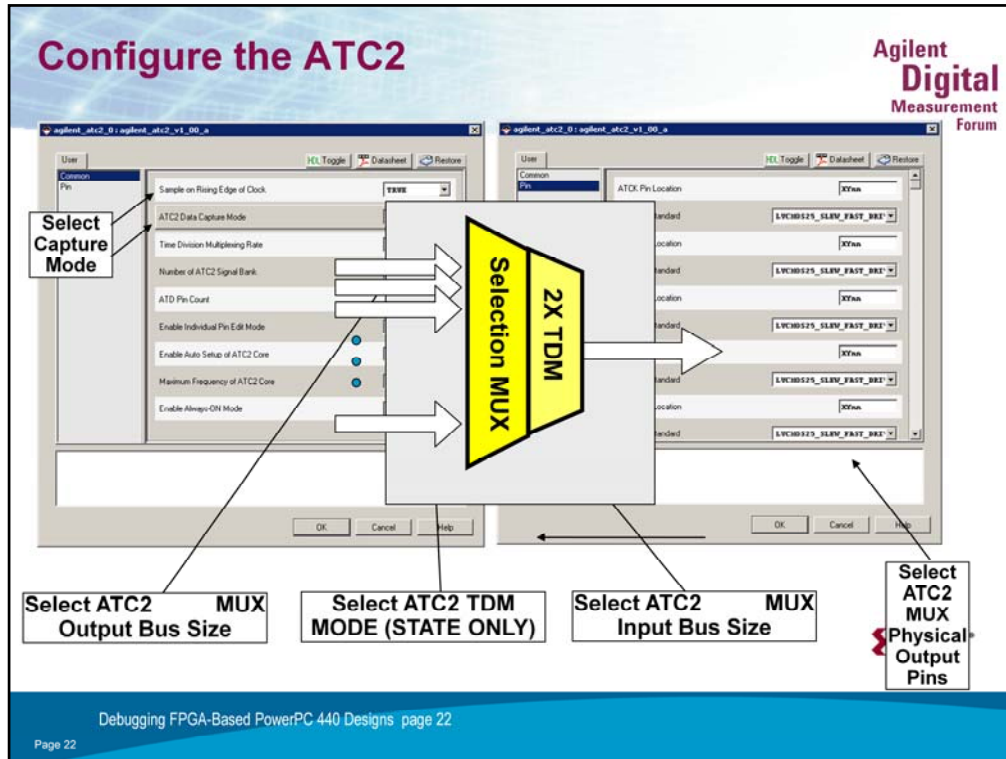
Agilent
Digital
Measurement
Forum



Debugging FPGA-Based PowerPC 440 Designs page 21

Page 21

Unfortunately the Agilent Trace Core, which is what we use with FPGA Dynamic Probe, is added in a separate step. It's simple to add, however. You simply go to the IP catalog (shown) and drag it over to the system architecture side.

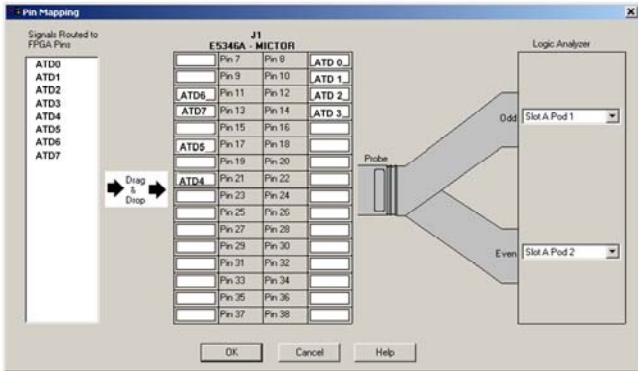



Once the ATC2 core is added, you double-click on it to get to the two configuration screens. All of the options shown should be related back to the block diagram on slide 16.

Agilent Logic Analyzer Dynamic Probe Pin Mapping Simplifies Setup (34 channel Mictor)

Agilent
Digital

Measurement
Forum






Graphically Define Physical Connections

FPGA to Connector
Connector to Logic Analyzer

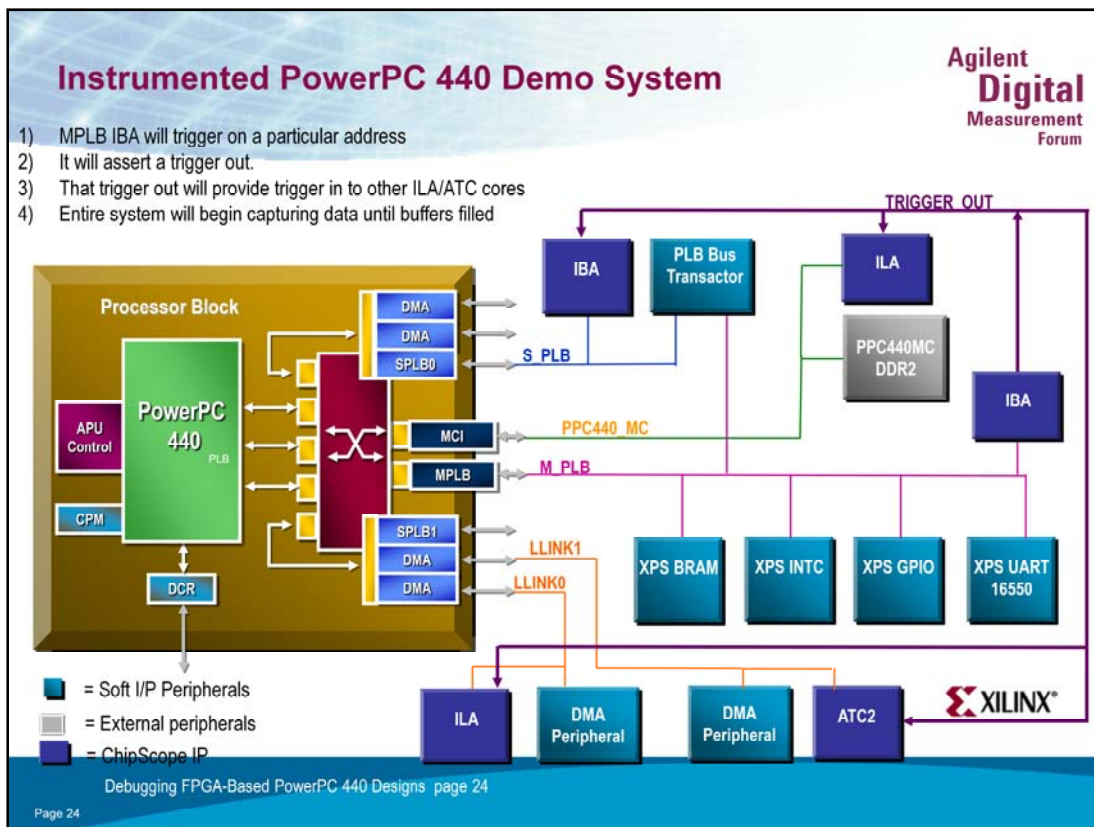
Spend Time Debugging and Verifying Designs
Instead of Documenting Signal Connections

Debugging FPGA-Based PowerPC 440 Designs page 23

Page 23



One of the necessary chores associated with using a Logic Analyzer is that of setting up the waveform display, setting up of labels and things like that. All of that needs to be done before you ever actually capture any data. (Actually you can do it whenever you want, but if you do it early, then you spend a lot of time trying to remember what each trace is connected to. Not fun). One of the things that Agilent did when they design FPGA Dynamic Probe was that they ensured that configuring it would be easy. And they've succeeded! FPGA Dynamic Probe has knowledge of what kind of connector you've connected to the FPGA, be it Mictor, softtouch, or flying leads. With that, it's then an easy task to pull in the signal names from the Xilinx tools. You drag and drop them onto the Pin Mapping screen on the logic analyzer and configuration takes minutes, not hours.



Now that we know how to add the different debugging options, let's see what it would look like in practice with our example design.

This slide shows where the ChipScope peripherals are connected and also illustrates the direction of the triggering. The slide is set up to do a build where each feature is added and can be talked about with each mouse click

Click 1: Shows an Internal Bus Analyzer connected to the Master PLB, which is the system's main I/O bus. We will program the IBA to trigger on a particular PLB address that corresponds to an access to the PLB Bus transactor. At the same time we will be setting a breakpoint in our code to also stop code execution when the line of code that accesses the PLB Bus Transactor is reached.

Click 2: The 2nd Internal Bus Analyzer is connected to the other side of the PLB bus transactor. The idea is that the processor will write an instruction (over the MPLB) to the PLB Bus Transactor, instructing it to execute a command over the Slave PLB. This 2nd IBA will be able to capture that access.

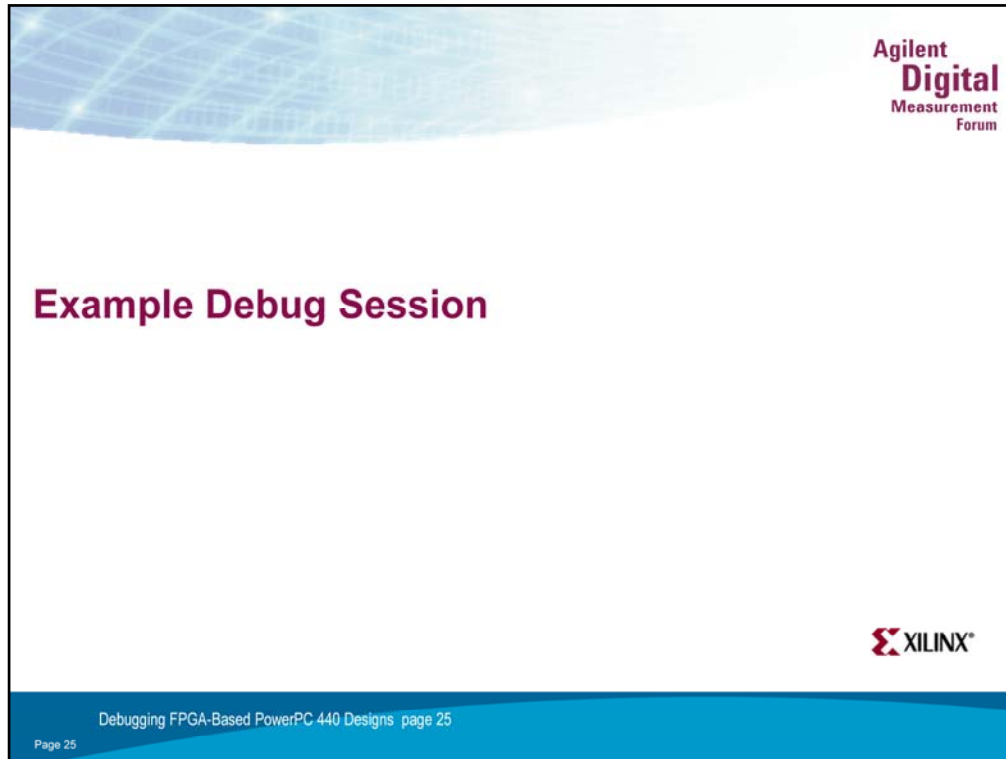
Click 3: This shows the Internal Logic Analyzer connected to the Memory Controller Interface bus which connects the PowerPC 440 Embedded Block. There isn't an Internal Bus Analyzer module for the MCI, so we have to use an ILA and connect it manually. This will allow us monitor must traffic between the soft memory controller and the PowerPC 440 Embedded Block.

Click 4: Here we connect another ILA one of the DMA channels. Like the MCI there isn't currently an IBA for the DMA controllers, so we connect the ILA to the DMA interface manually.

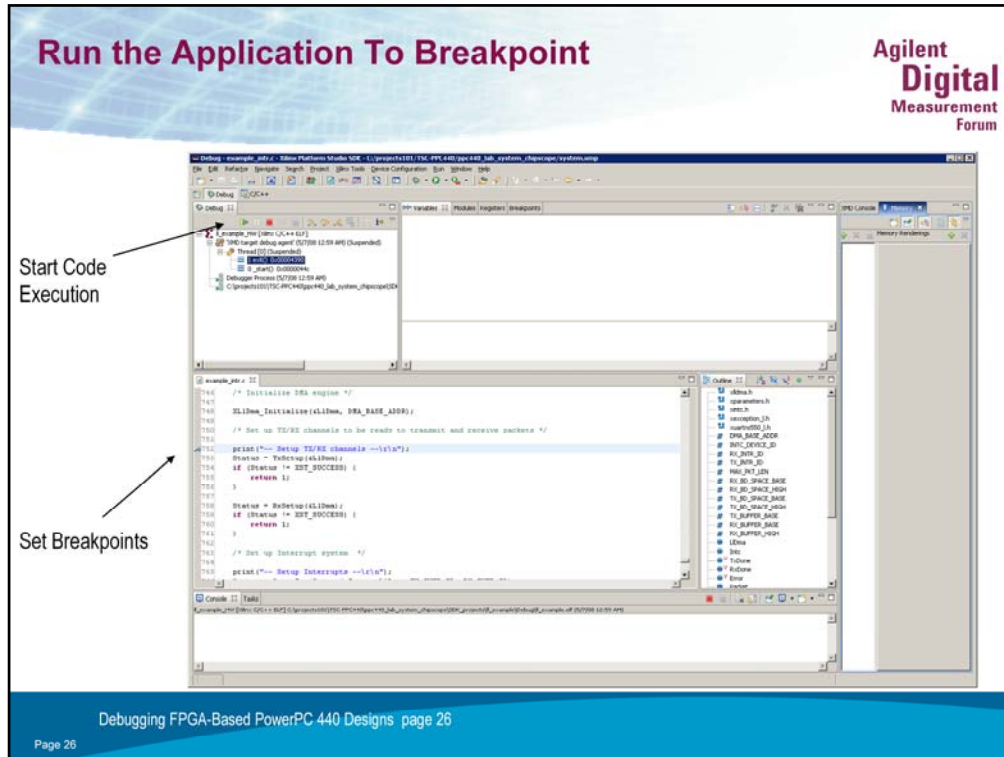
Click 5: Now suppose we have another DMA channel connected to a very high speed peripheral (like an ethernet MAC or a gigabit transceiver) that sends many megabytes of data, and you want to be able to view and store that. That's where the ATC2 core, which connects directly to the Agilent Logic Analyzer, comes in. ILA's and IBA's use the FPGA internal block RAM resources and they are a limited commodity. Being able to pipe the DMA data out to the Agilent Logic Analyzer provides the FPGA design a much larger amount of storage for their logic analyzer trace data.

Click 6: Now the great thing is that all of this can be set up to trigger on one event (if you want). Click again, and we'll step through it.

Click 7: MPLB has an address sent out. When it sees that address, the ChipScope IBA is set to trigger, and also asserts it's trigger out. The trigger out, then triggers all of the other ILA/IBA/ATC2 cores. When that happens all of the cores start capturing data until their buffers are full.



So let's see how we'd use this setup in an actual debug scenario.



Remember this is a processor application, so we're going to be executing some code. To do that we have the eclipse software development environment. Eclipse provides separate viewpoints, depending on whether your writing and compiling code or debugging code on hardware.

In this example we're going to be debugging, and we're going to set a breakpoint at the start of code execution, then one at the instruction that accesses the MPLB.

Once the two breakpoints are set (simply by double-clicking in the margin next to the line of code that you're interested in), we start the code. It will break at the entrance to "main".

Agilent
Digital
Measurement
Forum

Project: pcc_440_debug

Trigger Setup - DEV1-MylDevice4 (XC5VX70T1) UNIT3-MylA3 (LA)

Match Unit	Function	Value	Radius	Counter
M1_LLINO_LL_Tx_Data	==	XXXX_XXXX	Hex	disabled
M1_LLINO_LL_Rx_Rem	==	XXXX	Bin	disabled
M2_LLINO_LL_Tx_SOP_n	==	0	Bin	disabled
M2_LLINO_LL_Tx_EOP_n	==	X	Bin	disabled
M4_LLINO_LL_Tx_SOP_n	==	X	Bin	disabled
M5_LLINO_LL_Tx_EOP_n	==	X	Bin	disabled
M6_LLINO_LL_Tx_ScrRdy_n	==	X	Bin	disabled
M7_LLINO_LL_Tx_DstRdy_n	==	X	Bin	disabled
M8_LLINO_LL_Tx_Data	==	XXXX_XXXX	Hex	disabled
M9_LLINO_LL_Rx_Rem	==	X	Hex	disabled
M10_LLINO_LL_Rx_SOP_n	==	X	Bin	disabled
M11_LLINO_LL_Rx_EOP_n	==	X	Bin	disabled
M12_LLINO_LL_Rx_SOP_n	==	X	Bin	disabled
M13_LLINO_LL_Rx_EOP_n	==	X	Bin	disabled
M14_LLINO_LL_Rx_ScrRdy_n	==	X	Bin	disabled
M15_LLINO_LL_Rx_DstRdy_n	==	X	Bin	disabled

Synopsys: DEV1: 4 UNIT: 3

Data Port:

- LLINO_LL_Rx_Data
- LLINO_LL_Rx_EOP
- LLINO_LL_Rx_SOP
- LLINO_LL_Rx_Rem
- LLINO_LL_Rx_SOP_n

Trigger Setup:

Add Active Trigger Condition Name Trigger Condition Equation
 CH1 * TriggerCondition0 M2

Type Window Windows 1 Depth 8192 Position 128

Storage Qualification All Data

```
COMMAND: set_match_function 4 3 2 0 3 1 0
COMMAND: set_trigger_condition 4 3 3 1 0 0F 0000 0000 0000 5555
COMMAND: set_storage_qualification 4 3 FFFF FFFF FFFF FFFF
COMMAND: run 4 3
COMMAND: upload 4 3
```

Info- Device 4 Unit 2: Waiting for core to be armed

Upload Done

Now that the code is paused at main, we can set up ChipScope to trigger on the MPLB access, then arm ChipScope. It will wait for the code to be restarted and it will wait for, in our case, that particular MPLB address.

Arm FPGA Dynamic Probe

Agilent Digital Measurement Forum

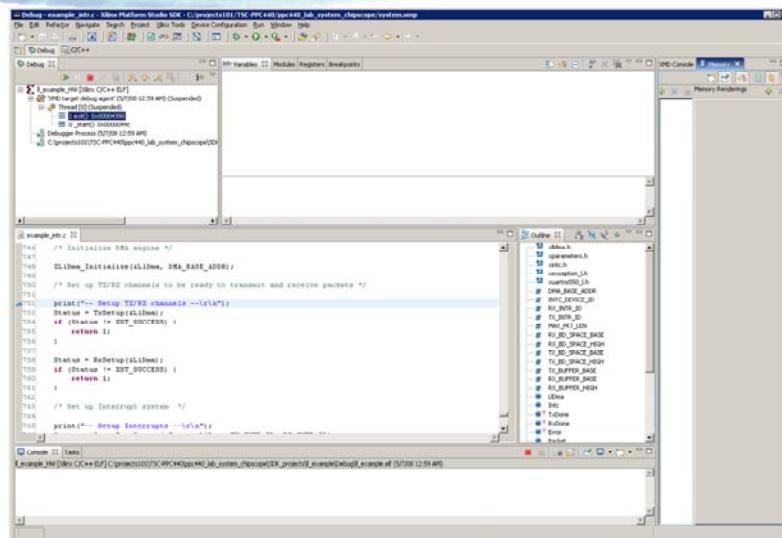
The screenshot displays the Agilent Logic Analyzer software interface. At the top, a menu bar includes File, Edit, View, Setup, Tools, Markers, Run/Stop, Overview, Window, and Help. Below the menu bar is a toolbar with various icons for file operations, analysis, and navigation. A status bar at the top shows a measurement: $t_{M1} \text{ to } t_{M2} = 24 \text{ ns}$. The main workspace is divided into three panes: Probes, Modules, and Windows. In the Probes pane, 'FPGA Dynamic Probe-1' is selected, and a context menu is open with options: Properties, Setup..., and Bank Selection... In the Modules pane, 'My 16910A-1' is selected under 'Slot B'. In the Windows pane, 'Listing-1' and 'Waveform-1' are visible, each with a 'Show' button. The Xilinx logo is in the bottom right corner.

Debugging FPGA-Based PowerPC 440 Designs page 28

Page 28

Before restarting code execution, we also need to arm FPGA dynamic probe. This is done on the Agilent Logic Analyzer.

Continue Execution



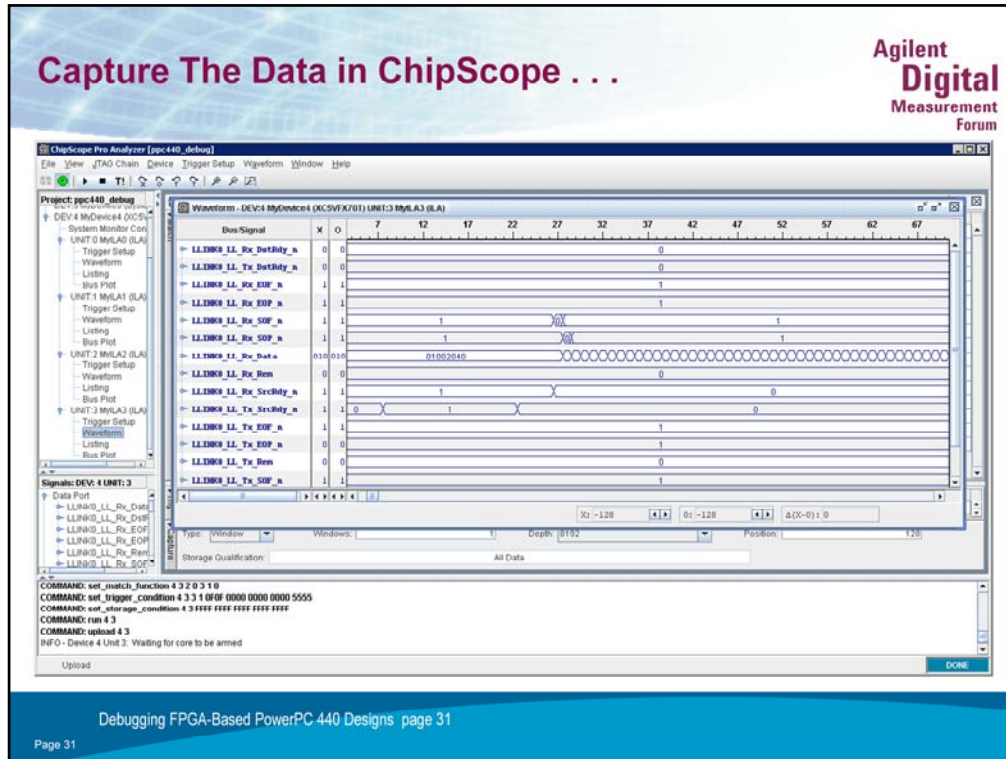
XILINX®

Now that all of our cores are armed, we can continue execution.

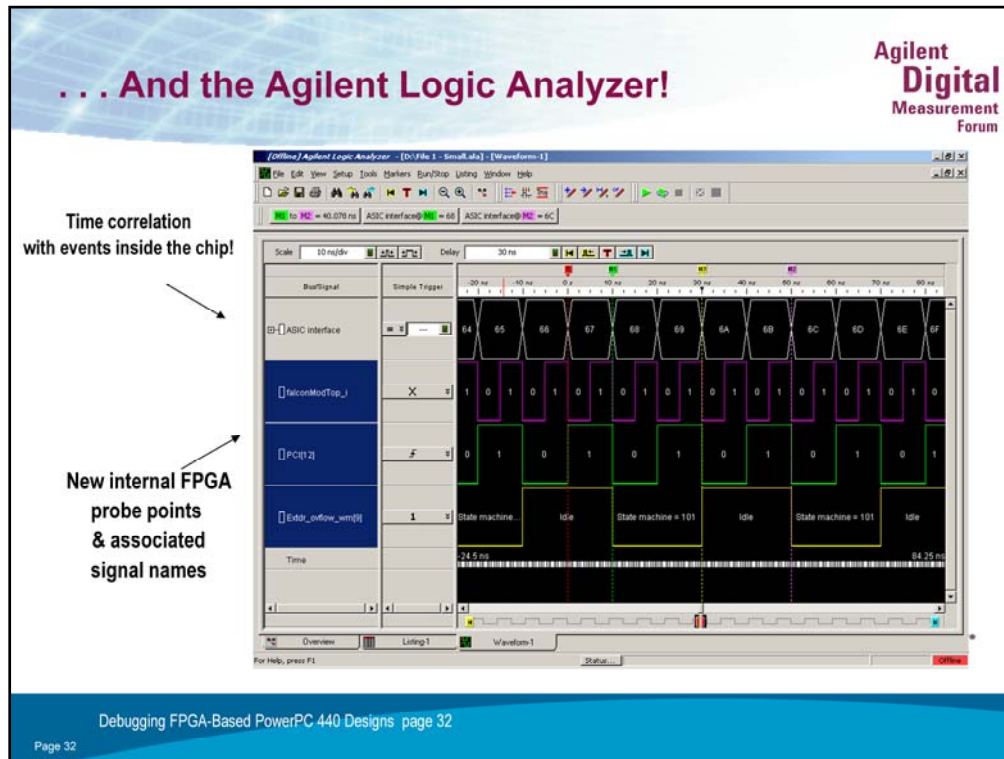
Now That Code is Executing

Let's recap what will happen (since this isn't a real demo!)

- The code is running. Presumably the code sets the DMA channels in motion, does a bunch of DDR2 memory accesses and finally . . .
- Executes the PLB command that goes out the MPLB and sets the PLB Bus Transactor in motion
- The MPLB's IBA triggers, which then triggers the rest of the cores, and lots of data gets captured.



This is the Xilinx ChipScope ILA/IBA display that shows all of the data captured. The IBAs show actual PLB transactions, the ILA's show whatever data you've connected the ILA's to. This waveform display is no Agilent Logic Analyzer waveform display, but it's still a very highly flexible, configurable display that allows you to view the data in a number of different ways.



And of course, the data that's been going out through the ATC 2 core to the Agilent Logic Analyzer is displayed there. The key thing is that all of these displays have triggered on the same event, which gives a nice time correlation between events. What that allows you to do is quickly get to the bottom of "is it hardware or software" types of questions. In addition, by using the FPGA Dynamic Probe, you can capture huge amounts of waveform data.

Conclusion

Agilent
Digital
Measurement
Forum

Using the Xilinx Embedded Development Kit tool-suite, along with Xilinx ChipScope Pro, and Agilent FPGA Dynamic Probe, an unprecedented level of FPGA-based Hardware/Software debug is possible.

This kind of thing can't be done in an ASIC without dedicated logic built in.

Leverage this capability to get your next project bug-free faster!



Xilinx Debug Solutions

Agilent
Digital
Measurement
Forum



ChipScope Pro on-chip debug tools

- Part of integrated 10.1 tool-suite
- 60-Day Evaluation version

- ChipScope Pro Serial IO Toolkit
 - Add-on feature to ChipScope Pro
 - 60-Day Evaluation version



- Agilent FPGA Dynamic Probe
 - Purchased separately from Agilent
 - Acquisition option for your new Agilent logic analyzer

INX®

Call to Action

Agilent
Digital
Measurement
Forum

Download the 60 days Free eval Xilinx 10.1 SW

http://www.xilinx.com/products/design_resources/design_tool/index.htm.

Comes with ChipScope Pro Eval

Next time you have a system level problem, use the combination of the Eclipse Software Debugger, ChipScope PRO and the FPGA Dynamic Probe to isolate the problem.

For further information contact craig.abramson@xilinx.com

