FPGA-Adaptive Software Debug and Performance Analysis

Javier Orensanz, ARM
Stefano Zammattio, Altera

The availability of devices incorporating hardened ARM applications processors connected to on-chip FPGA fabric opens a world of possibilities to electronic designers, but introduces design, debug and optimization challenges outside the area of expertise of most. New development methodologies are required to address software and hardware integration issues and system-level performance optimizations at a price affordable by small and medium companies. This article outlines the latest innovations in on-chip debug logic, FPGA tools and software debug and analysis tools aimed to address these challenges.

Background: embedded software development tools and FPGA analysis tools
Before starting an in-depth analysis of the needs posed by mixed SoC FPGA devices it is useful to outline the traditional design methodologies available for software development on embedded processors and hardware development on FPGA.

Devices based on ARM® processors normally have a JTAG or Serial-Wire Debug port for software debug. Embedded software debuggers can attach to this port using a hardware debug probe, stop the processor, and read and modify the status of the processor and memory and peripherals connected to its AMBA® buses. This debug connection is essential for early stages of development, including board bring-up, boot code, functional validation of peripherals and development of kernel space drivers.

Many ARM processor-based devices also incorporate an Embedded Trace Macrocell™ (ETM) or a Program Trace Macrocell™ (PTM). These on-chip blocks monitor non-intrusively the instructions executed by the processor, compress the information and send it to an on-chip memory buffer or an external trace port. When this information is imported into the debugger, developers can view a history of all the instructions executed by the processor up to a certain point in time, which facilitates the debug of complex bugs: the kind of bugs that only appear from time to time and disappear the moment you instrument the code in an attempt to debug them.

FPGAs also offer trace capabilities. They also include JTAG ports, through which FPGA tools can configure on-chip logic analyzers to capture RTL signals on on-chip memory before, after or around a certain trigger. This trigger is normally configured to be a hardware error condition calculated as a combinatorial function of several signals in the RTL – for example, when a hardware state machine enters a certain stage, or when an input pin goes high or low.

Current mass market tools deal well with software problems and FPGA problems, but do not offer much help for problems arising from the integration of software and hardware, which are bound to happen when mixing processors and FPGAs on a die. These integration problems have been addressed by EDA tools in RTL
simulation and emulation environments, but EDA solutions are often too complex and expensive for companies other than silicon vendors.

Altera® and ARM have collaborated around on-chip debug logic, FPGA tools and software debug tools in order to create new methodologies that assist software development on the new Altera SoC FPGA devices. In this article we will use examples based on the ARM Development Studio 5™ (DS-5™) software toolchain and Altera SignalTap® tools in order to illustrate solutions, although the technologies implemented are generic in nature.

### Debugging across worlds

Finding out whether a complex problem is caused by a hardware bug or a software bug is normally the first step towards a fix. The main methodologies for debugging across worlds consist of:

1. Setting up an error condition in the software, and analyzing the state of the hardware around that point in time
2. Setting up an error condition in the hardware, and exploring what the software was doing around that point in time
3. Visualizing a history of software instructions and hardware RTL waveforms, and aligning them on events of interest in order to explore the relationship between the two

All these methodologies require dedicated debug hardware on the boundary between the hardened processor sub-system and the FPGA fabric. Altera SoC FPGA devices include a comprehensive implementation of ARM CoreSight™ on-chip debug and trace logic, including a cross-trigger matrix connecting input and output triggers from all the processors and trace macrocells in the processor sub-system with hardware triggers from the FPGA fabric. The cross-trigger matrix can be easily programmed from DS-5 Debugger to configure which hardware blocks generate triggers, and which components are affected by them.

![Cross-trigger matrix on Altera SoC FPGAs](image)

Software developers normally know that something is wrong because an assertion failure occurs, or because the software reaches an error handling function. Alternatively, they could step through the code until they locate the instruction after which the target crashes. Once that critical point in the software is located, they would only need to use DS-5 to configure the trace macrocell to generate a trigger on that instruction, and use SignalTap to configure the FPGA to capture RTL signals around the incoming trigger.
The next time the software is executed, when the processor reaches that instruction the FPGA will capture waveforms of the selected RTL signals around that point in time. Developers often decide to capture first waveforms of the system bus from the processor, in order to decode what memory-mapped component was being accessed at the time. However, once the faulty component has been identified, the same methodology can be used to analyze its internal hardware implementation.

Similarly, SignalTap can be used to configure the FPGA to generate a trigger when a particular combination of RTL signals occurs, while DS-5 is used to configure the cross-trigger matrix so that the FPGA trigger either stops the processor execution or starts the capture of software instruction trace.

Figure 2: Cross-triggering from the software world to the hardware world
Exploring the relationship between software and hardware

Despite its powerful features, cross triggering does not satisfy all the needs of developers working on software and hardware integration. Its main limitation is that it cannot be used as an exploratory tool: since it requires that the developer defines an error condition, it is only useful for fixing problems, not for finding them.

Altera SoCs implement a CoreSight System Trace Macrocell™ (STM), which provides a more appropriate tool for initial exploration of the relationship of software and hardware. The STM enables both software and hardware instrumentation. You can instrument the software by “printing” character strings to the STM when there are events of interest. Similarly, you can instrument the hardware by getting the STM to monitor RTL signals over periods of time. Every time one of those hardware signals changes or every time the software hits an interesting event, trace packets get generated by the STM, and sent for decoding to DS-5 Debugger.

This technology addresses a number of integration questions like the following:

- Are my FPGA peripherals being turned off when they are not needed by the software?
• Does the driver for a peripheral on the FPGA handle correctly shared access from two applications?
• How long does the software take to react to a peripheral interrupt or a change on an input pin of the FPGA?
• Are the data generated by FPGA peripherals always received and understood correctly by the software? Are any data being lost?

By using a System Trace Macrocell to monitor hardware signals you get the added advantage of being able to display all data sources in a single user interface, which is provided by DS-5 Debugger. This is normally more efficient than having to continuously switch between software and FPGA tools, especially when the software developer is not an FPGA expert.

Finally, Altera SoC FPGAs implement on-chip CoreSight global timestamps, which provide a common time base distributed to all the trace macrocells in the hardware. These global timestamps enable the user to correlate instruction trace, software events and hardware events over long periods of time. Global timestamps provide an alternative correlation mechanism to triggers that is more suitable for exploratory stages.

Reducing software development costs

The goal of development tools is not only to facilitate the debug of complex problems, but also to make development more efficient and, in general, reduce time to market. Sometimes this has more to do with the convenience and usefulness of standard product features than the availability power features.

One “convenience feature” available in most professional debuggers is the ability to display memory-mapped SoC peripheral registers as groups of registers with names, bitfields and descriptions equivalent to those found in the peripheral’s documentation. For example, if you want to configure a UART with a certain baud rate, you would just go to the UART peripheral window, select its CONTROL register and pick up the correct baud rate from a list. The software debugger would then automatically translate this into a write access to the right memory address with the right data value. This is really handy for hardware functional validation and driver development.

When developing on FPGAs things become a bit more complicated. FPGA vendors like Altera normally provide a library of FPGA hardware such as encryption/decryption blocks, mathematical algorithm acceleration blocks and peripheral controllers. However, it is up to the hardware developer to decide how many of these blocks are synthesized on the FPGA and where they are located in the processor’s memory map, which means that it is not possible for the software debugger to provide peripheral register views for them out of the box.

The solution to this problem requires communication between the FPGA synthesis tools and the software debugger. In particular, the Altera QSys system configuration tool is capable of generating and exporting peripheral register description files for complete FPGA designs, and DS-5 Debugger can automatically import them, so that as the FPGA hardware changes, its system views adapt seamlessly.

Most times the largest part of the FPGA design consists of a number of IP blocks provided by the FPGA vendor. For custom FPGA blocks, the software developer can generate peripheral description files manually, but manual editing is time consuming and error prone, and therefore not the best approach for standard IP blocks.
System-level performance analysis

Over time, more and more emphasis is being put by product developers on debugging performance issues in an effort to squeeze more functionality out of the same hardware, or to reduce its power consumption. Performance and power analysis tools have become a major area of focus for tools vendors.

One important reason to choose mixed SoC and FPGA devices is the ability to use FPGA hardware blocks to accelerate software jobs. For example, FFT decoders or DES decryption algorithms in the FPGA fabric can be used to free up the processor, which can either perform another task in parallel or just go to sleep and save power consumption. For these devices it is essential that tools provide visibility of the relative levels of utilization of the processors and FPGA IP blocks over time. This information can then be used by the designer to optimize the whole system, implementing the right number and type of IP blocks on the FPGA, partitioning the software to make efficient use of them, and balancing the load between hardware resources.

Although instruction trace is often used for optimizing software codecs and other performance critical software, it is not the right kind of analysis tool for understanding system-level performance bottlenecks. For ARM applications processors running complex operating systems such as Linux and Android, analysis tools that rely on OS instrumentation and statistical sampling are normally preferred. The ARM DS-5 Streamline performance analyzer is an example of such type of tool.

The Streamline performance analyzer makes use of a Linux driver running on the target to sample information from the target at regular time intervals and every time that there is a task switch. The information captured is provided by counters for the following types of events:

- Operating system events such as processor load, memory usage and network load.
- Processor events such as branch mispredictions, level1 cache hits and misses, or number of instruction interlocks.
- System events such as level2 cache misses, GPU utilization or GPU cache misses. These counters are made available by the relevant system IP blocks as memory mapped registers, and enable the user to spot system-level bottlenecks.
- Software annotations, used by applications to report events of interest that the user would like to correlate with performance counters.
When this information is visualized together on a timeline, the interactions between software and hardware are made apparent to the developer, who has better information to optimize the target as a complete system.

On fixed hardware targets, the information provided by Streamline can only be used to change the software. However, on mixed SoC and FPGA devices it can be used to simultaneously optimize hardware and software, by choosing the right FPGA acceleration blocks and adapting the software to use those blocks efficiently. The only infrastructure required in the hardware is memory-mapped registers that count the level of utilization of each different IP block. Streamline can then be configured to access those new counters and display their value over time, correlated with CPU activity and other system-level counters.

Users interested in power consumption can extend Streamline with an ARM Energy Probe or National Instruments data acquisition hardware in order to monitor and visualize voltage and current consumption on a number of power rails on the target. On FPGA targets these power rails would normally be the ones used to power the CPU subsystem, FPGA core and FPGA I/O, but they could also monitor the main power supply of the whole product. Again, by visualizing the dependency of power consumption with software activity and system utilization, and being able to easily benchmark the energy consumption required to complete a task, developers can optimize the system for power consumption and battery life.

**Conclusion**

The new breed of devices containing ARM applications processors and FPGA fabric open a wealth of possibilities for faster, cheaper and more energy efficient electronic products. The innovation in the hardware has been matched by innovation in the on-chip debug hardware, FPGA tools and software debug and analysis tools, so that developing on these devices and making the most of their power features is as easy and efficient as software development on fixed SoCs.

Altera SoC devices are going into mass production in March 2013, and will be supported by DS-5 from day one. For more information, see [http://www.altera.com/soc](http://www.altera.com/soc) and [www.arm.com/ds5](http://www.arm.com/ds5).