WHITE PAPER

Intel® FPGA

FPGA Adaptive Software Debug and Performance Analysis

Authors

Javier Orensanz
Director of Product Management,
System Design Division
ARM

Stefano Zammattio
Product Manager
Intel® Corporation

Abstract

The availability of devices incorporating hardened ARM* applications processors closely coupled to an on-chip FPGA fabric opens a world of possibilities to electronic system designers. However, these devices also introduce novel design, debug, and optimization challenges. New development methodologies are required to address software and hardware integration issues and system-level performance optimizations efficiently at a price affordable by small- and medium-sized companies. This white paper outlines Intel's and ARM's latest innovations in on-chip debug logic, FPGAs, and software debug and analysis tools aimed to address these challenges.

Introduction

Current tools deal well with software problems and FPGA problems, but do not offer much help for problems arising in systems that tightly integrate software and custom hardware—as will inevitably occur in devices that contain processors and FPGAs on a single die. These integration debug challenges can be addressed using EDA tools in register transfer level (RTL) simulation and emulation environments, but these solutions are often too complex, slow, and expensive for companies who are not developing their own silicon devices.

Intel and ARM have collaborated on the development of FPGAs and software debug tools in order to create new methodologies that leverage the on-chip debug logic and enhance software development capabilities for the new Intel® SoCs. This white paper uses examples based on the ARM Development Studio 5* (DS-5*) Intel SoC FPGA Edition software tool chain and Intel Signal Tap tools to illustrate debug solutions, although the technologies implemented are generic in nature.

Intel SoCs

Cyclone® V and Arria® V SoC families consolidate two discrete devices into one to reduce system power, cost, and board size while increasing performance. Every SoC contains an FPGA fabric that has been tightly integrated with a hard processor system (HPS). The HPS consists of a dual-core ARM Cortex*-A9 processor, peripherals, and memory controller (see Figure 1). Many modern systems incorporate discrete processors and FPGAs but the communication between them is limited to the existing processor external interfaces. Although there are usually several potential interfaces to choose from, none of them are ideal. Bandwidth is generally low, latencies are often high, and there is usually significant effort in writing driver software and/or FPGA interface logic to enable efficient communication. In addition to these difficulties, the interfaces can be proprietary or vary in specification from processor device to device, making it difficult to reuse the design even if the general system requirement is similar.

Table of Contents

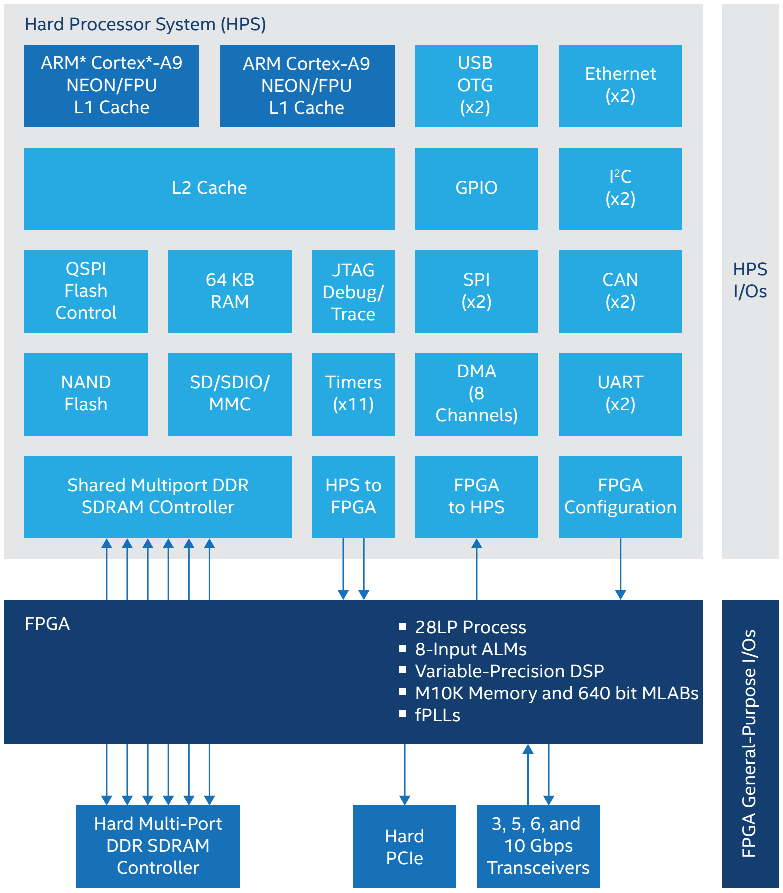


Figure 1. Intel SoC Block Diagram

The single-die integration of processor and FPGA fabric in Intel's SoCs enables implementation of high data bandwidth AMBA* connections between processor system, FPGA intellectual property (IP), and the memory system. No special software or interface logic is required—the registers of the FPGA IP are simply mapped directly into the processor memory space, making communication with software simple and direct. Having the two devices on the same chip also makes it possible to connect other interfaces and signals between the FPGA and processor, thereby delivering a much higher level of integration than is possible with discrete devices. For example, between the processor and FPGA there are 64 interrupt signals, a custom general- purpose I/O (GPIO) interface, signals enabling processor sleep, wake, and reset from the FPGA, and FPGA access to the ARM CoreSight* debug infrastructure implemented in the processor system. These features enable sophisticated processor or FPGA designs that previously were simply not possible. In addition, access to the CoreSight infrastructure delivers the ability to debug the system in ways that are not possible with discrete devices or other integrated processors or FPGAs.

Tools for Embedded Software Development and FPGA Analysis

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, memory, and peripherals that are 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 code and drivers.

Many ARM processor-based devices incorporate a CoreSight Embedded Trace Macrocell* (ETM) and/or a Program Trace Macrocell (PTM) to deliver trace capability. These on-chip blocks non-intrusively monitor the instructions executed by the processor, compress the information, and send it to an on-chip memory buffer and/or an external trace port. When this trace information is loaded into the debugger, developers can view a history of all the instructions executed by the processor up to a certain point in time. This history enables debug of complex problems—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 have JTAG ports and can offer similar trace debug capabilities. Intel FPGAs use a system known as Signal Tap, which automatically adds and configures IP that monitors RTL signals inside the FPGA design. Once triggered, RTL signal trace data from before, after, or around the trigger event is stored in on-chip memory. This trigger event normally reflects a hardware error condition, and may be calculated as a combinatorial function of several signals in the RTL—such as when a hardware state machine enters a certain stage and the critical RTL signals are set to an illegal set of values, or when an input pin goes high or low.

Debugging Across Worlds

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

- Triggering on an error condition in the software, and analyzing the state of the hardware around that point in time
- Triggering on an error condition in the hardware, and exploring what the software was doing around that point in time
- 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

Each of these methodologies requires dedicated debug hardware on the boundary between the hardened processor system and the FPGA fabric. Intel SoCs include a comprehensive implementation of ARM CoreSight on-chip debug and trace logic, including a cross-trigger matrix that connects input and output triggers from all the processors and trace macrocells in the processor sub-system and hardware triggers to and from the FPGA fabric (see Figure 2). The cross-trigger matrix can be easily programmed from the DS-5 Debugger to configure which hardware blocks generate triggers and which components are affected by them, as shown in Figure 3. Any debug trigger event can be used to simultaneously trigger the software and FPGA debug systems regardless of where the trigger was generated.

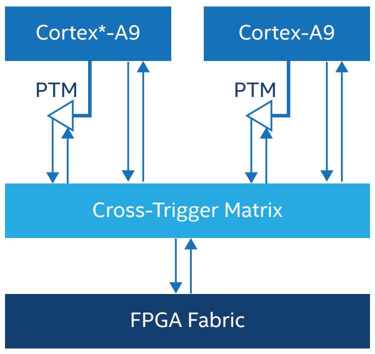


Figure 2. Cross-Trigger Matrix on an Intel SoC

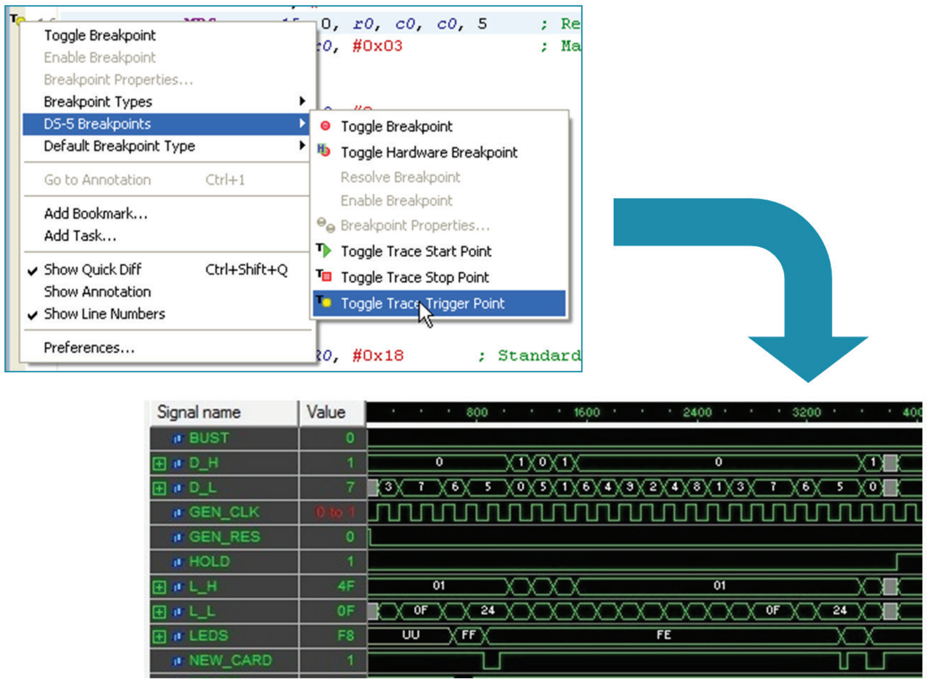


Figure 3. Cross-Triggering from the Software World to the Hardware World

Software developers normally know that something is wrong because an assertion failure occurs, or because the software enters an error handling function. If the processor just crashes, they might restart the system and step through the code until the crash occurs. Once that critical point or instruction in the software is located, they use DS-5 to configure the trace macrocell to generate a trigger on or before that instruction, and use Signal Tap to configure the FPGA to capture RTL signals around the same incoming trigger. The next time the software is executed and the processor reaches that critical instruction, the FPGA will capture waveforms of the selected RTL signals around that point in time.

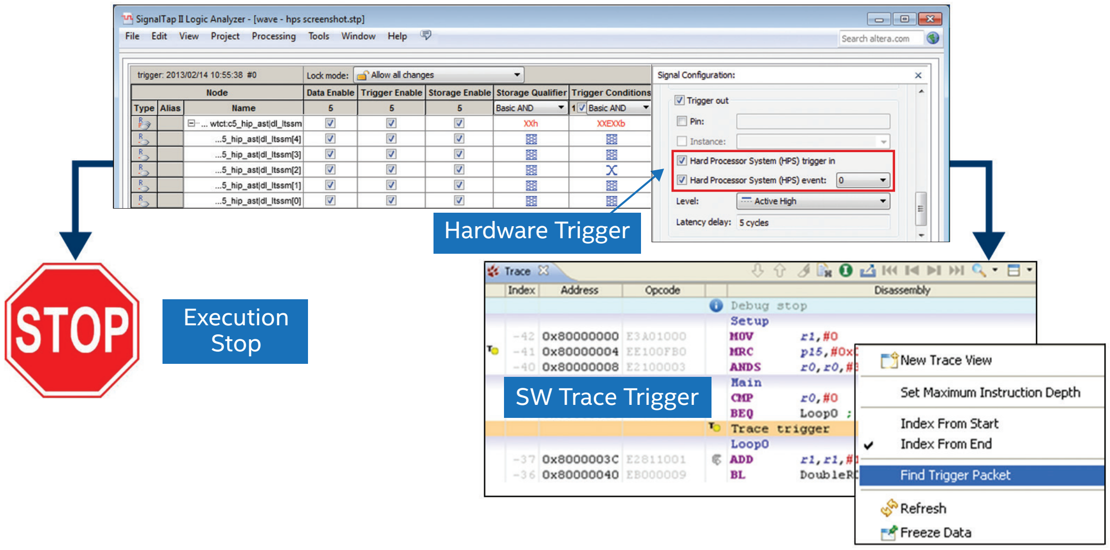
Developers often decide to initially capture waveforms of the system bus from the processor to decode what memory-mapped component was being accessed at the critical time. Once the faulty component has been identified, the same methodology can be used to analyze its internal hardware implementation.

Similarly, SignalT ap can be used to configure the FPGA to generate a trigger when a particular combination of RTL signals occurs, as shown in Figure 4. The DS-5 is then used to configure the cross-trigger matrix so that the FPGA trigger either stops the processor execution or starts the capture of the software instruction trace.

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. Cross-triggering requires the developer to define an error condition that is only useful for capturing relevant debug data rather than actually identifying the cause of a system-level problem.

Intel's SoCs implement a CoreSight System Trace Macrocell (STM) (Figure 5), which provides a more appropriate tool for initial exploration of the relationship between software and hardware. The STM enables both software and hardware instrumentation. The software is instrumented by “printing” character strings to the STM when there are events of interest. Similarly, the hardware is instrumented by routing signals of interest to the STM. Every time one of the STM-routed hardware signals changes or the software hits an interesting event, the STM generates and sends trace packets to the DS-5 Debugger.



trigger: 2013/02/14 10:55:38 #0 — Lock mode: Allow all changes

Type	Alias	Name	Data Enable 5	Trigger Enable 5	Storage Enable 5	Storage Qualifier Basic AND	Trigger Conditions 1 ✓ Basic AND
		... wtct:c5_hip_ast|dl_ltssm	✓	✓	✓	XXh	XXEXXb
		...5_hip_ast|dl_ltssm[4]	✓	✓	✓	▨	▨
		...5_hip_ast|dl_ltssm[3]	✓	✓	✓	▨	▨
		...5_hip_ast|dl_ltssm[2]	✓	✓	✓	▨	X
		...5_hip_ast|dl_ltssm[1]	✓	✓	✓	▨	▨
		...5_hip_ast|dl_ltssm[0]	✓	✓	✓	▨	▨

Signal Configuration:
- ✓ Trigger out
- ☐ Pin:
- ☐ Instance:
- ✓ Hard Processor System (HPS) trigger in
- ✓ Hard Processor System (HPS) event: 0
- Level: Active High
- Latency delay: 5 cycles

```
Index  Address     Opcode      Disassembly
                               Debug stop
                               Setup
 -42   0x80000000  E3A01000    MOV     r1,#0
 -41   0x80000004  EE100FB0    MRC     p15,#0x
 -40   0x80000008  E2100003    ANDS    r0,r0,#
                               Main
                               CMP     r0,#0
                               BEQ     Loop0 ;
                               Trace trigger
                               Loop0
 -37   0x8000003C  E2811001    ADD     r1,r1,#
 -36   0x80000040  EB000009    BL      DoubleR
```

Figure 4. Cross-Triggering from the Hardware World to the Software World

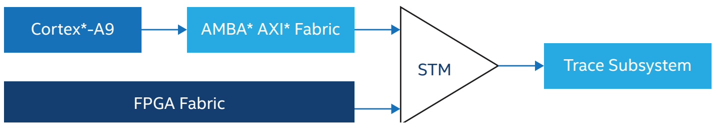


Figure 5. Software and Hardware Connection to a STM

This technology addresses a number of integration questions, including:

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

The use of a STM to monitor hardware signals delivers the added advantage of being able to display all data sources in the single user interface provided by the DS-5 Debugger. This interface is more efficient than having to continuously switch between software and FPGA tools, especially when the developer is not an FPGA expert.

Finally, Intel SoCs implement on-chip CoreSight global timestamps, which provide a common time base distributed across 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 are more suitable for exploratory stages of development.

Reducing Software Development Costs

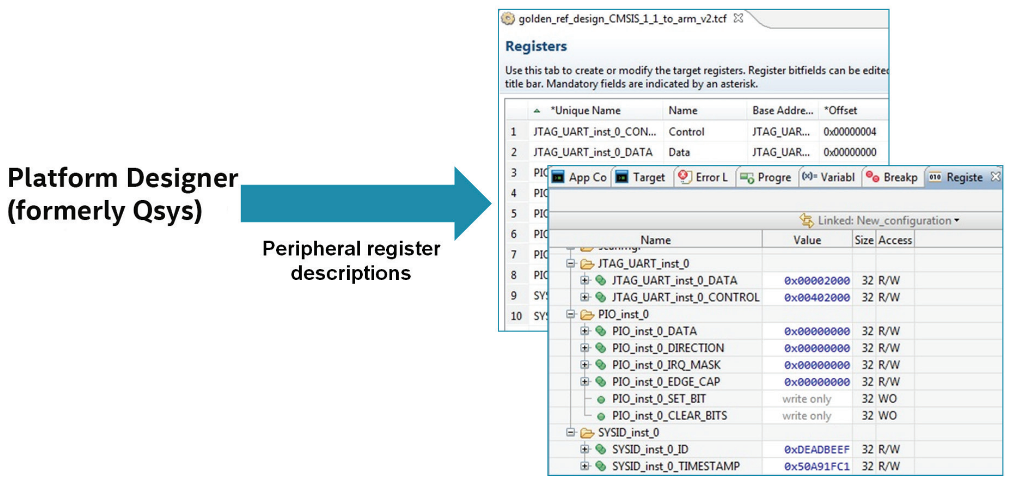
The goal of good development tools is not only to facilitate the debug of complex problems, but also to make development more efficient and, in general, reduce time to market. However, sometimes these tools have more to do with the convenience and usefulness of standard product features rather than the availability of specific powerful features.

One "convenience feature" available in most professional debuggers is the modification of the user interface to display the features of the specific device being debugged. This modification involves the ability to display device specific memory-mapped peripheral or module registers in groups with names, bitfields, and descriptions equivalent to those found in the device's documentation and reference

software. For example, if you want to configure a UART with a certain baud rate, you would just go to the UART peripheral window, select CONTROL register, and select the correct baud rate from a drop-down menu. The software debugger would then automatically translate this request into a write access to the right memory address with the right data value. This ability is essential for effective hardware functional validation and driver development.

When developing on FPGAs, things are a bit more complicated. FPGA vendors like Intel normally provide a library of peripheral IP, such as encryption/decryption blocks, mathematical algorithm acceleration blocks, and peripheral interface controllers. It is up to the hardware developer to decide how many of these blocks are implemented in the FPGA and where they are located in the processor's memory map. This means that it is not possible for the software debugger vendor to provide a pre-built peripheral register view.

The solution to this problem requires communication between the FPGA design tools and the software debugger. In particular, Intel's Platform Designer (formerly Qsys) and design flow is capable of generating and exporting peripheral register description files for the FPGA design. The DS-5 Debugger can automatically import these files and adapt the debugger display to include the FPGA-based peripheral registers and bitfields. If the FPGA hardware changes, the DS-5 Debugger imports the new register description files and its system view adapts seamlessly to the new memory map (Figure 6).


Platform Designer (formerly Qsys)
Peripheral register descriptions
golden_ref_design_CMSIS_1_1_to_arm_v2.tcf
App Co
Target
Error L
Progre
Variabl
Breakp
Registe
Linked: New_configuration

Registers

Use this tab to create or modify the target registers. Register bitfields can be edited
title bar. Mandatory fields are indicated by an asterisk.

	*Unique Name	Name	Base Addre...	*Offset
1	JTAG_UART_inst_0_CON...	Control	JTAG_UAR...	0x00000004
2	JTAG_UART_inst_0_DATA	Data	JTAG_UAR...	0x00000000
3	PIO			
4	PIO			
5	PIO			
6	PIO			
7	PIO			
8	PIO			
9	SYS			
10	SYS			

Name	Value	Size	Access
JTAG_UART_inst_0			
JTAG_UART_inst_0_DATA	0x00002000	32	R/W
JTAG_UART_inst_0_CONTROL	0x00402000	32	R/W
PIO_inst_0			
PIO_inst_0_DATA	0x00000000	32	R/W
PIO_inst_0_DIRECTION	0x00000000	32	R/W
PIO_inst_0_IRQ_MASK	0x00000000	32	R/W
PIO_inst_0_EDGE_CAP	0x00000000	32	R/W
PIO_inst_0_SET_BIT	write only	32	WO
PIO_inst_0_CLEAR_BITS	write only	32	WO
SYSID_inst_0			
SYSID_inst_0_ID	0xDEADBEEF	32	R/W
SYSID_inst_0_TIMESTAMP	0x50A91FC1	32	R/W

Figure 6. Automatic Import and Display of Peripheral Registers on the DS-5 Debugger

While a large part of the FPGA design often consists of IP blocks provided by the FPGA vendor, developers frequently design their own IP blocks with proprietary register interfaces. Developers can manually add their register interface to the DS-5 view. However, if a peripheral description file for the IP blocks is added to the Platform Designer (formerly Qsys) component then the Intel development flow will integrate the custom IP register map to the description files in the same way as with standard Intel components. This ability enables DS-5 to deliver a complete view of the SoC system memory-map automatically even if custom FPGA IP blocks are used.

System-Level Performance Analysis

Over time, product developers are placing greater emphasis on debugging performance issues in an effort to squeeze more functionality out of the same hardware, or to reduce power consumption. Performance and power analysis tools have therefore become a major area of focus for debug tool vendors.

One important reason to choose Intel SoCs is their ability to use FPGA hardware blocks to accelerate software functions. For example, fast Fourier transform (FFT) decoders, digital filters, and DES decryption algorithms implemented in the FPGA fabric can be used to free up the processor, which can then either perform another task in parallel or just go to sleep and save power consumption. For these reasons, it is essential that development 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, implement the right number and type of IP blocks on the FPGA, partition the software to make efficient use of them, and balance the load between software and 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 Linux* or Andriod operating systems, analysis tools that rely on OS instrumentation and statistical sampling like the ARM DS-5 Streamline* performance analyzer are preferred.

The Streamline performance analyzer makes use of a Linux driver that runs 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 software and hardware counters. The following types of events are supported:

- Operating system events—processor load, memory usage, and network load.
- Processor events—branch mispredictions, number of instruction interlocks, or level- 1 cache hits and misses.
- System events—level-2 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 events— "printf-style" annotations in the applications, used to report events of interest that the user would like to correlate with performance counters.

When this information is displayed on a timeline (Figure 7), the interactions between software and hardware are easy for the developer to visualize and understand, enabling optimization of the target as a complete system.

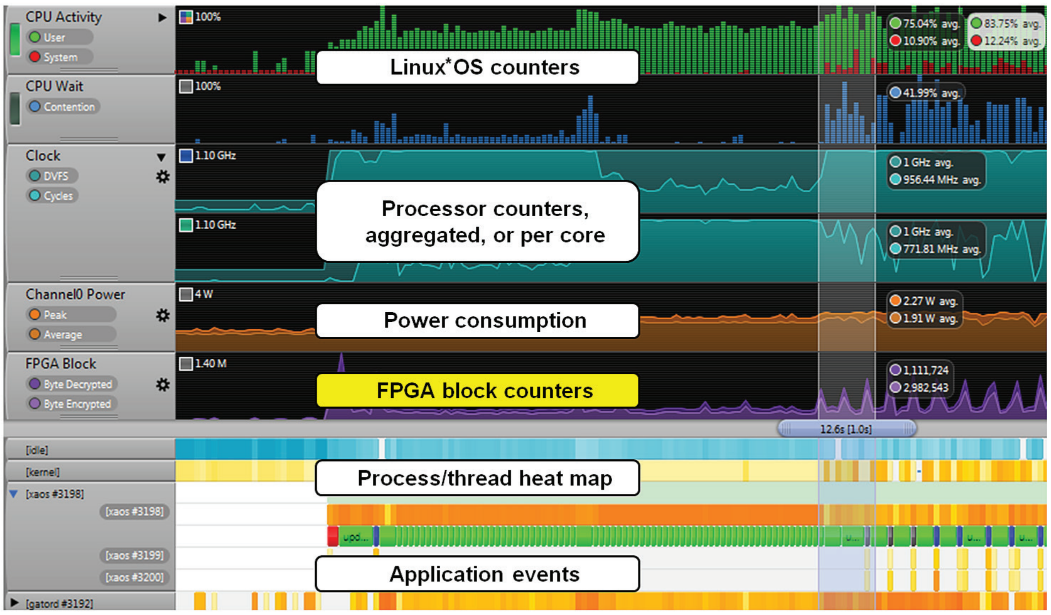


Figure 7. Timeline View in ARM DS-5 Streamline

On fixed hardware targets, the information provided by Streamline can only be used to optimize the software. However, on an Intel SoC, it can be used to simultaneously optimize both 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 FPGA 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 to monitor and visualize voltage and current consumption on a number of power rails on the target. On SoCs, these power rails would normally be the ones used to power the CPU subsystem, FPGA core, and FPGA I/Os, but you could also choose to monitor the main power supply of the board or even 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 class of SoCs integrating ARM applications processors and Intel FPGA fabric opens 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 FPGA tools, on-chip debug hardware, and software debug and analysis tools, so that development with these devices, and making the most of their features, is as easy and efficient as software development on fixed processor devices.

References

- SoC Overview: www.altera.com/socfpga
- Cyclone V SoC Hard Processor System: www.altera.com/products/fpga/features/cyv-soc-hps.html
- ARM Development Studio 5 (DS-5) Intel SoC FPGA Edition Toolkit:
  - www.altera.com/DS5AE
  - www.arm.com/ds5altera

WP-01198-1.1