You Can Catch More Bugs With Transaction Level Honey

Miron Abramovici
DAFCA,
Miron.abramovici@dafca.com

Kees Goossens,
Bart Vermeulen
NXP,
kees.goossens@nxp.com

Jack Greenbaum
Greenhills Software
jackg@ghs.com

Neal Stollon
HDL Dynamics,
neals@hdldynamics.com

Adam Donlin
Xilinx Research,
Adam.donlin@xilinx.com

ABSTRACT
In this special session we explore holistic approaches to hardware/software debug that use or integrate transaction level models (TLMs). We present several TLM-based approaches to system-level diagnostics, ranging from use of most popular transaction level modeling languages through to hybrid technologies that combine TLMs with other well known diagnostic tools like in-silicon trace logic.

Categories and Subject Descriptors
C.5 [COMPUTER SYSTEM IMPLEMENTATION]

General Terms
Measurement, Design, Standardization, Verification.

Keywords
Transaction-level models, system diagnostics.

1. INTRODUCTION
System-level performance analysis, early software development and pre-silicion system verification are three popular use cases for transaction level models. In this special session we review how TLMs can enhance system-level diagnostics – verification that occurs post-implementation.

This paper is a collection of four extended abstracts that present different perspectives on the use of TLM for system-level diagnostics. Section 2 considers how TLM can improve diagnostics for systems with complex interconnect. Section 3 reviews TLM standardization efforts that extend into the field of system-level diagnostics. In section 4, the use of TLM to support high quality silicon instrumentation is discussed and section 5 presents the software architect's view of TLM and its use in software test and diagnostics.

2. Communication-centric debug by controlling transactions in-flight

2.1 Transaction-based multi-core debug
Systems on chip (SOC) contain many tens of IP cores, including hardware accelerators, programmable processors, and embedded memories. Hierarchical busses or networks on chip (NOC) implement deeply-pipelined concurrent communication between the cores. SOCs are hard to debug, because of the large number of concurrent processes. Second, the interactions are complex, due to deeply pipelined multi-threaded transactions, distributed shared memories, memory latencies with large variations between local and external memories, consistency and coherency for distributed cache protocols, etc. Finally, dynamic frequency scaling and multiple clock domains cause non-deterministic behaviour.

To debug a SOC it is infeasible to only use a software debugger for each processor, or use clock-cycle accurate system simulations. Therefore we propose to focus on the new communication complexity, i.e. the *interactions between the cores*. We also abstract multiple variable clock cycles to *transactions* of communication protocols such as AXI to provide a more intuitive common hardware/software debug interface that is more deterministic.

2.2 Hardware and Software Architecture

Our transaction-level communication-centric debug design flow[1] instruments a SOC with *monitors* to observe cores and/or their communication. Monitors (◇ in Figure 1) observe NOC transactions (flits) on the router links. They generate *events* on programmed happenings of interest, which the event-distribution interconnect (EDI) sends to transaction run/stop control blocks (RSCB). The ⊗ symbol indicates transaction-level RSCBs that control the flow of requests and responses between a master and all its slaves, or a slave and all its masters. The Ø symbols deal with single master-slave communications, for even finer control. An IEEE 1149.1 Test Access Port (TAP) and scan chains, already present on our SOCs, are used to program this infrastructure through test-point registers (TPR), and to read and modify functional data.

Our debug software offers an API to read/write TPRs and functional state, but also at the level of logical connections between masters and slaves. This uses the XML description from which the RTL of the NOC is generated. For example, the functional state of some or all connections between one or more masters and slaves can be retrieved from the RSCBs and network interfaces (NI). It is then displayed at the transaction level (active transactions or not, which data element, state of the request/response FIFOs, etc.), rather than a series of scanned bits. For maximum flexibility in run/stop control, all RSCBs can be enabled, started, or stopped independently based on any set of active monitors or TAP control. Based on this and using patterns, a set of "connections under debug" is easily specified at run time, which can then be single and multi-stepped independently.

2.3 Conclusions

Transaction-based debug will play a large role in debugging future SOCs where the complexity is in communication between IPs. Much research, both conceptual and practical, remains to bring this vision to life. This is joint research with B. Vermeulen, A. Nejad, and A. Hansson.

3. ESL Driven Instrumentation for System Diagnostics

Large amounts of effort and innovation in the last decade has been focused on verification and diagnostics at diverse levels of abstraction (ESL, RTL, gate level, emulation, in system/ on-chip instrumentation, and others). Less focus has been applied to the related area of integration and reuse of the verification and diagnostics information and supporting tool integration as part of a capability for the overall system validation and debug process that evolves through a product design and life cycle. As a result, system validation and diagnostic activities at different stages in a design are often performed in a state of isolation from any prior related analysis activities. While there are some arguable merits in creating independent tests and diagnostics at each design stage, the value of being able to access validation and diagnostic data from prior stages in the design flow has many advantages, from reduction in time to create new tests and diagnostic scenarios, to improving consistency between design stages, to improved abilities in automatically and formally correlating and cross-validating analysis from multiple design abstractions and views.

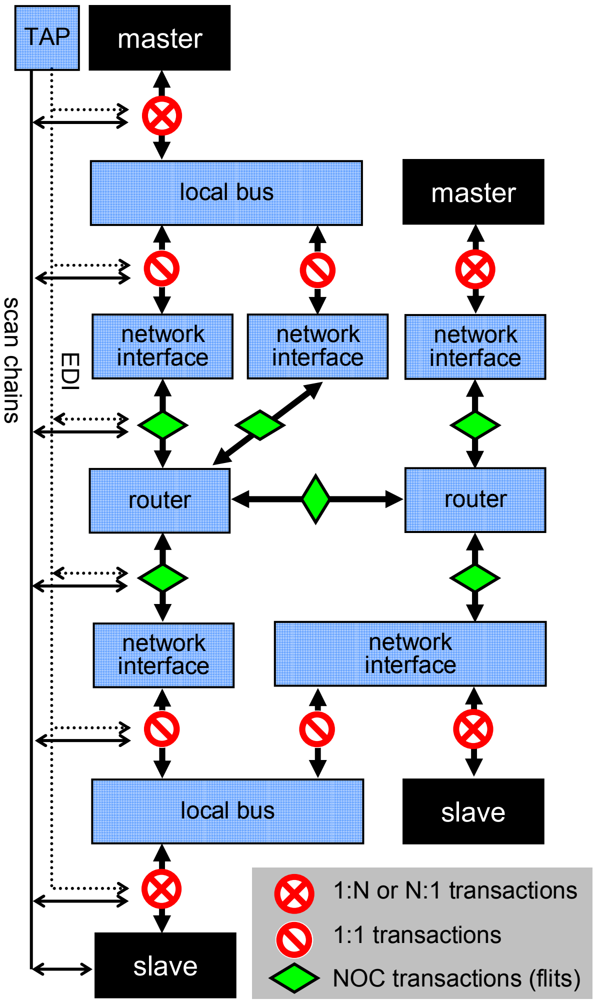

Figure 1: Masters, slaves, NoC and debug infrastructure

In the best of worlds, each stage of a design effort should be able to access, in a common format and description, analysis information and diagnostic results from prior stages in the design process. The reality of most design

[1] B. Vermeulen, et al. Debugging distributed-shared-memory communication at multiple granularities in Networks on Chip (NOCS), 2008; K. Goossens, et al. The Æthereal network on chip: Concepts, architectures, and implementations. *Design and Test of Comp.*, 22(5), 2005.

flows falls far short of this goal. This is, in part, due to the historical legacy of a variety of different design descriptions being used at different stages in the analysis (ie. ESL (SystemC), RTL (Verilog), gate (EDIF), etc.) and modeling limitations of being able to resolve issues like timing resolution and model abstraction between different domains. Nevertheless, it can be argued that this "tower of babel" syndrome in being able to reuse all of the prior information available for a design is a contributor to the verification and validation shortfall crisis that faces many complex designs.

As an example for brevity, consider two steps in the design flow, ESL and in On-Chip System Instrumentation. These provide an interesting comparison since they occur at the diametric ends of a system design - ESL focused on the initial concepts of the design, and On-Chip Instrumentation utilized at the final physical product stages where integration of (physical) hardware and software make up a system. They share common concerns (focus on the end to end operation and performance of a system) and scope (addressing both hardware and software integration and optimization) that are not typically addressed at other design stages (RTL as an example, while arguably the most comprehensive area of focus for design verification and validation activities, does not easily integrate software analysis and is often limited in analysis to partitioned sub-systems of a design, than analysis of the total system).

ESL modeling and system analysis, used at the conceptual front end of SoC design for initial functional and performance analysis, allows parallel hardware (although abstracted) and software analysis and verification and system parameter (bus, memory) optimization. ESL based design, as arguably the first stage where substantive analysis of a design is done, becomes a prime mover for improving access to validation and diagnostic information that can be reused in subsequent design stages. ESL provides additional potential to be extended to drive a range of subsequent analysis activities, as it is still in formative stages of development and standardization.

Instrumentation and hardware-based analysis is used to facilitate system verification and performance analysis of physical (FPGA and/or ASIC) implementations. Instrumentation capabilities include varieties of system trace, triggering, and other monitoring and run control. While instrumentation is applied to a range of scenarios ranging from logic analysis to performance monitoring, modern ASSP devices focus most instrumentation resources on software related analysis (ex. processor breakpoints, run control and trace) and access, control, and monitoring of on-chip system (bus and memory) resources.

So given this common set of analysis goals and needs between ESL and Instrumentation, are there common tools and flows to encourage consistency and reuse of validation and related diagnostics. With very limited exception, even tools (ex. GDB) and data formats (ex. VCD) that may be commonly used, do not provide a useful infrastructure for end to end integrated system analysis. This is a promising area for research and commercial development that can make a significant impact of current design flows.

4. Using Transactions for In-system Silicon Validation and Debug

Since complete pre-silicon system-level verification is practically impossible, in-system silicon validation must tackle many aspects of the behavior of a new SoC, such as hardware-software integration, corner cases not reached in verification, operation under stress conditions, adaptive control of temperature, voltage, and power, and digital-analog interactions. In-system silicon validation is done with severely limited observability and controllability of the internal activities in the chip, and has to deal with non-deterministic system operation and lack of time-specific expected values. Because of these reasons, silicon validation and debug has become the most time-consuming and the most unpredictable phase of the development cycle of a new SoC.

First we review a new silicon validation approach. Pre-silicon, instrumentation tools guide the insertion of reconfigurable instruments into the RTL model of the SoC, and generate an instrumented RTL model that is processed by standard synthesis-based design flows. The instrumentation creates an infrastructure platform that is dynamically configured and operated post-silicon by post-silicon tools. Dynamic in-system configuration enables continuous reuse of the instrumentation for a variety of applications including logic analysis, assertions in silicon, on-chip functional block test, performance monitoring, programmable fault and error injection, and hardware-software co-debug.

Then we present the use of transactions for in-system silicon validation and debug. Transactions have been used to raise the level of abstraction in analyzing the operation of an SoC for pre-silicon verification. The new approach brings the advantages of transactions to the silicon domain. We show how reconfigurable on-chip instruments are dynamically configured as different transaction engines that detect, record, and analyze transactions. Following the activity of the SoC as sequences of transactions is much more effective for validation and debug than analyzing bit-level waveforms. Transaction engines can also generate on-chip user-specified transactions to increase the controllability of the SoC.

We also show the results obtained in validating and debugging five chips using the new silicon validation and debug technology. Subsections

5. TLM Diagnostics and Embedded Software Development

A key problem in embedded systems is access patterns on external bus interfaces. The issues to be analyzed relate to both software testing after the system is implemented, and to performance in the architecture/partitioning phase. From a performance point of view questions to answer include "is there enough bandwidth available in my system to move the data I need to move? Back when we built board level systems, visibility of the hardware/software interface was easy. One connected a logic analyzer to the busses, and all was revealed. But today's systems have hidden busses, and complex interactions between software and busses. Virtual platforms today run as fast as many of the embedded systems they simulate. TLM Diagnostics with these virtual platforms provide visibility to design and analyze today's systems.

Take for example a system that provides video over USB. Questions to answer include "Can a given processor be used to implement this system?" and "How much memory is required for this system?". A designer may be looking at several hardware solutions including a custom SoC or a COTS device either with an on-chip USB interface or a less expense COTS device and external USB interface. One might try to do back of the envelop calculations to determine if the processor has enough "MIPS" and if the peripheral bus interface has enough bandwidth to transfer the chosen video format from the video source to the USB interface. In theory the bus bandwidth questions are easy to answer by considering the amount of data to transfer and the speed of the connection between the USB interface and the processor. But USB isn't that simple. There is a great deal of control traffic between the USB Device interface driver above and beyond the data payload. How does one estimate that? This is where TLM Diagnostics come in. Using a virtual platform one can run the exact driver and application code and get real bandwidth and CPU "MIPS" numbers. Using this information a low-risk hardware BOM can be derived. Without this virtual platform, you must either take large risks, provide large and costly margins in performance, or build several physical systems and analyze them.

The same visibility that allows bandwidth analysis in the partitioning phase provides important visibility when testing software. Code coverage testing provides assurance that a given set of test cases actually tests, or covers, the functionality of the software under test. Many high reliability and high security systems require not just that every instruction is covered, so called "statement coverage", but also that paths through conditionals have been covered. In the extreme every path through multiple conditionals is verified, but more restricted metrics like Modified Condition/Decision Coverage (MCDC) are more common. This type of coverage analysis requires either instrumenting the code, or access to the program counter sequence actually executed during the execution of the test. Instrumenting the code is intrusive, so it is highly desirable to observe the system externally. While many processors to provide trace ports, when hardware trace isn't available, Virtual Platforms provide the necessary visibility.