A Design-for-Debug (DfD) for NoC-based SoC Debugging via NoC

Hyunbean Yi[1], Sungju Park[2], and Sandip Kundu[1]
[1]Department of Electrical & Computer Engineering, University of Massachusetts, USA
bean@engin.umass.edu, kundu@ecs.umass.edu
[2]Department of Computer Science and Engineering, Hanyang University, Korea
parksj@mslab.hanyang.ac.kr

Abstract

This paper presents design-for-debug (DfD) methods for the reuse of network-on-chip (NoC) as a debug data path in an NoC-based system-on-chip (SoC). We propose on-chip core debug supporting logics which can support transaction-based debug. A debug interface unit is also presented to enable debug data transfer through an NoC between an external debugger and a core-under-debug (CUD). The proposed approach supports debug of designs with multiple clock domains. It also supports collection of trace signatures to facilitate debug of long pattern sequences. Experimental results show that single and multiple stepping through transactions are feasible with moderately low area overhead. We also present simulation result to verify proper operation of the debug components.

1. Introduction

As the complexity of an SoC grows, on-chip bus becomes a bottleneck for overall throughput, scalability, and communication among different clock domains. To overcome these limitations, network-on-chip (NoC) has emerged as a new communication architecture [1], [2]. In order to test such complex SoCs, modular testing is one of efficient methods to reduce test time [3], [4]. In modular testing, a test access mechanism (TAM) enables the exchange of test data between external pins of a chip and its embedded cores, and test wrappers provide an interface between a TAM and the embedded cores. In general, SoC debugging methods implement on- and off-chip debug support logic while optimizing the test infrastructure. Accordingly, the area overhead due to on-chip test and debug infrastructure is an issue in cost sensitive SoC designs.

Marinissen et al. [5] presented a test wrapper for embedded core test and introduced an efficient wrapper cells-to-scan chains configuration named as TAM chain. Amory et al. [6] presented a test wrapper for the reuse of NoC as a TAM. They also utilized the TAM chain. For the case where the core test bandwidth is higher than the Automated Test Equipment (ATE) test bandwidth, they reduced test length by enabling some wrapper cells to perform parallel-to-serial/serial-to-parallel conversion. Hussin et al. [7] also used the Amory's method [6] and added the bandwidth matching registers to the test wrapper to increase the test bandwidth utilization.

This paper focuses on the reuse of NoC as a debug data path for NoC-based SoC debugging. By adopting the proposed techniques, time-to-market as well as area overhead can be reduced. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 gives assumptions and problem statement and we present the proposed debug support components in detail in Section 4. Experimental results are given in Section 5 and we conclude the paper in Section 6.

2. Related work

B. Vermeulen et al. [8]-[10] introduced the serial debug infrastructure based on IEEE 1149.1 test access port (TAP) [11] and scan chain. They proposed debug control logics such as a debug breakpoint detector and a clock control logic to stop and run one or more CUDs. For on-chip bus-based SoC debug, ARM provides Embedded Trace Macrocell (ETM) which can trace an ARM processor core by monitoring AMBA bus [12], and First Silicon presented a tracer called On-Chip Instrumentation (OCI) to concurrently trace multiple cores [13].

In an NoC-based SoC, multiple transactions are performed at the same time and cores operate with different clocks. Accordingly, different strategies to monitor, stop, and run for debug are required. C. Ciordas et al. [14] introduced a generic NoC monitoring service concept based on event-based bit-level monitoring. However, a higher abstraction level of monitoring needs to be considered because the size of monitored data during the bit-level monitoring is relatively big and IP cores in an NoC-based SoC perform packet based communications [15] [16]. C. Ciordas et al. [15] proposed a transaction monitor for the Æthereal NoC-based SoC, and K. Goossens et al. [17] presented a general NoC-based SoC debug architecture conceptually and implemented message- and transaction-based debug.

1081-7735/08 $25.00 © 2008 IEEE
DOI 10.1109/ATS.2008.15

By attaching monitors and stop modules to the routers as well as the IP cores, they defined various scopes of debug such as SoC, IP, and NoC. In order to reduce routing overhead due to the debug control and data signals, S. Tang and Q. Xu [16] have tried reusing NoC as the trace data path. However, more routers have to be added in NoC because each core tracer has to have its own network interfaces (NI) and the NoC can accommodate the debug traffic.

3. Assumptions and problem statement

We assume that each core is wrapped with IEEE 1500 wrapper [18] and the wrappers can be configured and operate as presented in [5], [6] or [7]. Fig. 1 shows a test wrapper architecture including the Test/Debug Control & Protocol Interface (TDCPI). For interface protocol, AXI [19], DTL [20], and OCP [21] are widely used. In test or debug mode, the TDCPI is enabled and scan chains and test wrapper cells are configured by setting Config MUXs. To perform scan shift and scan capture operations, some test control logics such as shift counters, clock gating logics, and scan enable generation logic are added in the TDCPI.

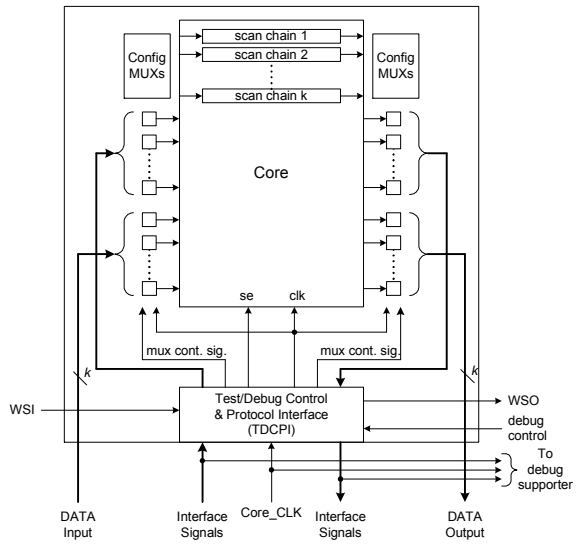


Figure 1. Test wrapper architecture with protocol interface

During a debug phase, when a certain event is generated by a monitor, a debug engineer or an on-chip debug component can stop the entire or part of the SoC and the engineer reads out the contents of scan chains. As the need arises, the engineer can apply specific functional patterns or state values and compare the results with a golden reference. To resume operation, the contents read out have to be restored into the scan chains. If special flip-flops for saving states such as the hold-scan cells [22], [23] and the swap registers [24] are implemented in cores, the operation resume time can be reduced. For more efficient NoC-based SoC debugging, three problems to be solved in this paper are:

Problem 1. No matter how quickly a debug controller tries stopping the SoC right after an event occurs, it is not possible to stop the SoC instantaneously because there is a signal propagation delay from the time the debug controller detects the event and generates a clock gating signal. In addition, even though the delay is predictable, the SoC stops after several cycles due to latency in clock distribution.

Problem 2. Most of the existing debug infrastructures use the IEEE 1149.1 serial path for scan dump. Therefore, it takes excessively long time to perform scan dump for large SoCs. The parallel data paths dedicated to debug can be added to reduce debugging time, but the routing area overhead due to the additional wires become too great.

Problem 3. When an SoC fails after billions of clock cycles, simulation based comparison of expected vs. actual responses is not feasible. There must be an alternative process to monitors internal nodes.

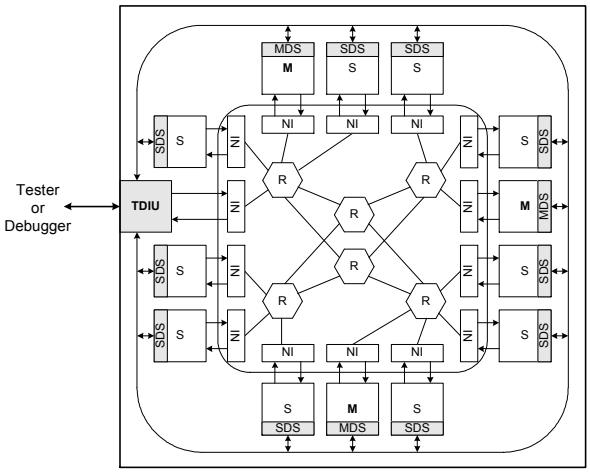
To solve the abovementioned problems, we also deploy a transaction based debug strategy. A transaction is initiated by a master (an initiator), executed by a slave (a target), and completed by the master [17]. Therefore, the initiations and completions of transactions can be analyzed by tracing the signals or packets of masters. Once an event occurs, core debug components prevent all masters from initiating new transactions and do not stop cores until all outstanding transactions are completed. Instead of stopping cores as soon as an event occurs, some information such as transaction counts, timer values, and the core from which the event is generated are recorded in an SoC. After all cores stop, a debug engineer reads out the information and uses them for debugging. By doing this, we can address the first problem and solve the second problem by reusing the test infrastructure and the NoC for debugging. To solve the third problem, we collect signature of select set of internal states. This is explained later in section 4.

4. Proposed Debug Components

4.1. Debug Architecture Overview

Fig. 2 shows the proposed debug architecture for an NoC-based SoC. For core debugging, the core debug supporters (CDS) are attached to the cores. There are two types of CDS. One is the master debug supporter (MDS) for master cores and the other is the slave debug

supporter (SDS) for slave cores. Each CDS operates with the functional clock of its core. A CDS monitors the signals between an NI and its core, distributes an event signal, and generates some debug control signals.



M: Master, S: Slave, R: Switch, NI: Network Interface,
MDS: Master Debug Supporter, SDA: Slave Debug Supporter,
TDIU: Test & Debug Interface Unit.

Figure 2. Debug architecture for NoC-based SoC

Once an event occurs, the following steps are performed to make all routers empty and stop all cores.

Step 1: The event is distributed to the TDIU and all CDS's. Each CDS stores its current transaction count and timer value in a debug information register as soon as the event is detected. The CDS which generates the event sets a special flag in its debug information register to '1'.
Step 2: Each MDS prevents its master core from initiating a new transaction and waits for all outstanding transactions issued by its master core to be completed.
Step 3: When all its outstanding transactions are completed, the MDS stops its master core, records its current transaction count and timer value, and informs the TDIU that its master core is stopped.
Step 4: When all master cores are stopped, the TDIU sends a control signal to enable each SDS to stop its slave core.
Step 5: On receiving the control signal from the TDIU, each SDS stops its slave core and records its current transaction count and timer value.
Step 6: The TDIU informs the external debugger that all cores are stopped.

During the next steps, all packets are consumed and all cores are stopped. Then, a debug engineer reads out the debug information and selects cores to be debugged using the TAP controller, and dumps scan contents or applies and observes debug data via NoC. For all cores or some selected cores including one or more associated master cores, single-transaction-step debugging can be performed by stopping and running the cores on a transaction basis. How to perform the single-transaction-step debugging is explained in more detail in the subsection 3 in this Section.

4.2. CDS and TDIU

Fig. 3 shows the proposed CDS. Each CDS operates with each core clock and the TDIU operates with a tester clock or a debugger clock. In other words, they operate totally asynchronously. Then, the debug components can communicate with each other by handshake. However, to minimize the number of wires, we do not use handshake but use a control signal interface strategy that a signal which is asserted once is not de-asserted until a debug process is over.

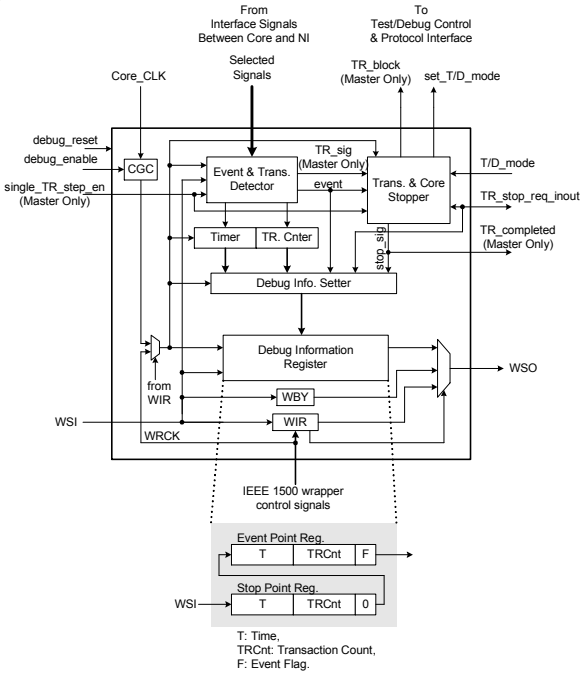


Figure 3. Core Debug Supporter (CDS)

The detailed descriptions of main components and their functions are follows:

• ***Clock gating cell (CGC) and Clock Multiplexer***: The CGC [25], [26] and the Clock Multiplexer [27] are used for glitch-free clock gating and switching.
• ***Event & Trans. Detector***: Generates events based on breakpoints programmed by the TAP controller and IEEE 1500. The Event & Trans. Detector in each MDS monitors the initiations and completions of transactions and sets TR_sig into '1' when all its outstanding transactions issued before an event (from itself or other CDS's) is generated are completed. The Event &

Transaction Detectors in each SDS simply counts the number of transactions and does not generate TR_sig because an SDS cannot find out that there are outstanding transaction request packets left in NoC.

• ***Transaction Counter***: Simply adds '1' whenever a new transaction is started. The counter value in each MDS is increased by one every time a new transaction is initiated by its master core, and the counter value in each SDS is increased by one every time a new transaction request packet is received in its slave core.

• ***Timer***: Counts the number of the core clocks and stops when the core stops.

• ***Transaction & Core Stopper***: Requests all MDS's and the TDIU to enter the process to stop cores through TR_stop_req_inout when an event occurs. When TR_stop_req_inout goes high, the Transaction & Core Stopper in each MDS drives TR_block high so that no new transactions can be initiated by its master core and waits for assertion of TR_sig. When TR_sig goes high, since TR_sig='1' means all outstanding transactions initiated by the master core are completed, the Transaction & Core Stopper sets set_T/D_mode to '1' to have the TDCPI (see Fig. 2) stop the core and configure the TAM chain, and sets stop_sig (stop_sig → TR_completed) to '1' to inform the TDIU of the completion of all its transactions and store debug information. When all masters complete all their outstanding transactions and all TR_completed signals are asserted, the TDIU, by setting debug_rdy to high, enables each SDS to stop its slave core and informs a debugger that all the cores are stopped. When T/D_mode (debug_rdy → T/D_mode) goes high, each SDS sets set_T/D_mode, and stop_sig to '1', respectively.

• ***Debug Information Register & Setter***: The Debug Information Register consists of the Event Point Register and the Stop Point Register. The current timer value (Time) and the current transaction counter value (TRCnt) are stored in the Event Point Register immediately when an event occurs. If the event is generated from itself (event='1'), then the event flag (F) is set to '1', but if the event is generated from other CDS (event='0' & TR_stop_req_inout='1'), then the event flag (F) is set to '0'. When the core stops (stop_sig='1'), the current timer value and the transaction counter value for the latest issued transaction are stored in the Stop Point Register. The event flag (F) in the Stop Point Register is always set to '0'.

The TDIU consists of the Master Under Debug (MUD) Selector, the Tester and Debugger (T/D) Interface, the Master Protocol Interface, and the TAP Controller. When an event occurs, the MUD Selector checks the transaction completions status of selected some or all MDS's. When all the transactions are completed, it stops all slave cores through T/D_mode. In test or debug mode, the T/D Interface block and the Master Protocol Interface block are required for data transmission between an external tester or debugger and on-chip NoC. If sufficient pins for testing and debugging are not provided, some serial-to-parallel/parallel-to-serial conversions are required for programming the MUD Selector and sending/receiving signals and data. The TAP controller is used to read the register values in the CDS's and selecting cores to be debugged. In order to control IEEE 1500, some simple TAP-to-IEEE 1500 control signal mapping logics are needed [6], [26]. All the debug control signals other than TR_completed are transferred through buses. Each bidirectional TR_stop_req_inout signal, among others, shares a bus called transaction stop request bus (TR_stop_req_bus). These buses simplify the interface among debug components running asynchronously and reduce routing overhead.

4.3. Single Transaction Step Debugging

Our single-transaction-step debug (STSD) is performed by running and stopping master cores on a transaction basis. The single_TR_step and TR_stop_req_inout signals of the TDIU, which are directly connected to a debugger, are used for the STSD. Once a debugger sets single_TR_step to high, the Event & Trans. Detectors and the Transaction & Core Stoppers in MDS's block transactions (TR_block='1') and enter the STSD mode. Then, the STSD is controlled by the TR_stop_req signal. When TR_stop_req goes high, TR_block in each MDS goes low to enable a new transaction to be initiated. Then, each master is stopped only its one transaction later. After all masters are stopped, slaves are stopped and the masters' transactions are unblocked so that debug operations such as scan (or I/O) dump, debug data application and observation, and resuming normal operation can be performed. One STSD, finally, comes to an end by finishing the debug operations. By de-asserting TR_stop_req after a completion of an STSD, a debugger enables the cores to be ready to start another STSD. In the case that two or more masters and one or more slaves are selected for the STSD, each master executes one transaction but a slave can execute two or more transactions. Therefore, to perform the STSD of a slave, an appropriate master-slave pair has to be selected.

4.4. Interrupt for debugging

Long pattern sequences cannot be simulated. Consequently, golden reference values are not known. As an alternative debug method, the intermediate values

can be considered. A debugger need not wait for an event to be generated to stop and trace through internal states. At periodic intervals all cores are stopped and internal states are gathered. This is based on Periodic System Management Interrupt (PSMI) described originally by I. Silas et al. [28], who uncovered 100% of logic issues by using PSMI failures reproduction. These periodic stops allow state dump and synchronization between the chip's behavior and RTL simulation, and restores state to resume normal operation. With our debug components, an interrupt by a debugger and a debug cycle can easily be done by asserting debug_enable and TR_stop_req signals as shown in Fig. 4. By periodically repeating this debug cycle, a PSMI failure reproduction is performed.

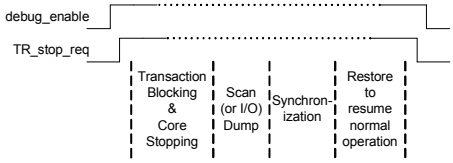


Figure 4. Interrupt by TR_stop_req signal and one debug cycle

5. Experimental Results

5.1. Area Overhead

In order to evaluate the area overhead, we used the open AMBA-based IP cores with 32-bit separate read and write data buses [29]. The cores are designed to interface with AMBA 2.0 bus. Hence, we assume that each core has AXI and designed a simple AXI logic as a protocol interface for scan test in each test wrapper and the TDIU. For the eight cores which are Leon3 processor, SDRAM controller, Ethernet MAC, VGA, GPIO, PS/2, Timer, and UART, we designed the IEEE 1500 wrappers and the TDCPIs with AXI. We also assume that each original core is already wrapped with IEEE 1500 and do not explicitly consider the area of the IEEE 1500 wrappers. The gate counts (# of 2-input NANDs) of the TDCPIs are arranged from 611 to 1519 according to the number of primary inputs and outputs of the core and the average gate count of the TDCPIs is 889. The size of registers in a CDS may vary depending on the complexity of its core. In this experiment, we fixed the numbers of the event detection registers and the transaction detection registers. An MDS has 16 32-bit event detection registers, 16 32-bit transaction detection registers, and a 48-bit timer. An SDS does not have transaction registers because it does not need to detect transactions. As a result, an MDS and an SDS are implemented with 10211 and 6115 in 2-input NAND equivalent gates, respectively.

As a result, the average area overhead by adding a TDCPI and a CDS in a core is about 33%. This area overhead is relatively large for the example cores. However, for large NoC-based SoC circuits, the total area overhead of all the debug components can be acceptable because the gate count of the TDIU including the TAP controller is only 1015 and it interfaces with all CDS's with only 5 buses.

5.2. Simulation

Fig. 5 shows a simulation result of a core-stopping process by debug components. Master 1 and Master 2 run at 500 MHz, and Slave 1 and Slave 2 run at 250 MHz and 125 MHz, respectively. As soon as an event is generated in Slave 2, its SDS informs other CDS's and

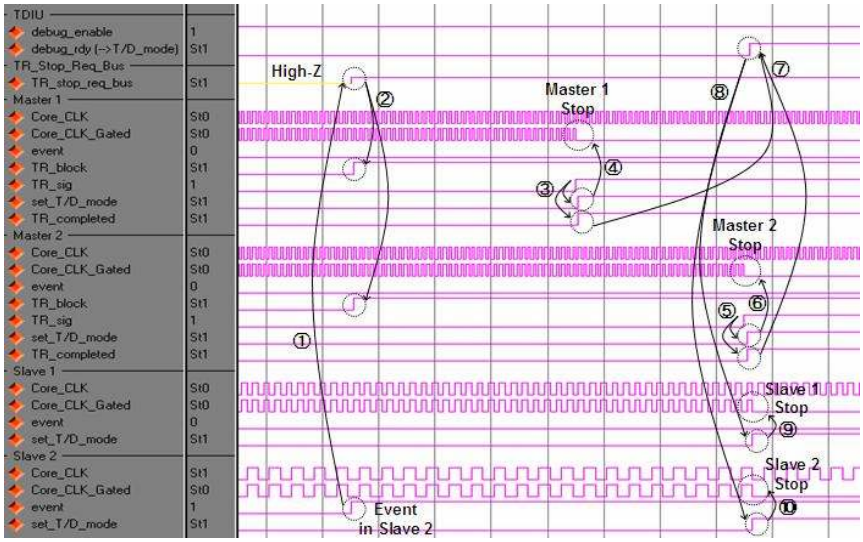


Figure 5. Simulation result of core-stopping process

TDIU of the event through TR_stop_req_bus and each MDS blocks a new transaction (① and ②). In this simulation, all outstanding transactions of Master 1 are completed (③) and then Master 1 stops first (④). When all outstanding transactions of Master 2 are completed (⑤), the TDIU stops all slaves using T/D_mode because all the outstanding transactions are completed. Even though each core is stopped by a gated core clock (Core_CLK_Gated), its CDS and TDPI keep running at Core_CLK.

6. Conclusions

This paper describes DfD methods to use an NoC for debugging of an NoC-based SoC. We designed a core test wrapper and added a test interface unit (TDIU). To reuse this test infrastructure including an NoC for debug, all outstanding packets in an NoC have to be consumed before the cores stop. Accordingly, we use a transaction based debugging strategy and presented the core debug supporting logic. Each debug supporting logic operates with its core functional clock, detects the initiations and completions of transactions, and records the times and transaction counts at which an event occurred and its core stopped. This allows debug support for multiple clock domains. The DfD scheme proposed supports a single transaction step debugging and can facilitate PSMI trace. By using our DfD techniques, testing and debugging of a large NoC-based SoC can be efficiently performed without adding new parallel paths or using the slow IEEE 11491.1 as a debug data path.

References

[1] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, pp. 70-80, 2002.

[2] P. P. Pande et al., "Design, Synthesis, and Test of Networks on Chips," *IEEE Design & Test of Computers*, pp. 404-412, 2005

[3] E. J. Marinissen and T. Waayers et al., "Infrastructure for modular SOC testing," *Proceedings IEEE Custom Integrated Circuits Conf.,* pp. 671-678, 2004.

[4] T. Waayers et al., "Definition of a robust Modular SOC Test Architecture; Resurrection of the single TAM daisy-chain," *Proceedings IEEE Int. Test Conf.*, Paper 25.3, pp. 1-10, 2005.

[5] E. J. Marinissen et al., "Wrapper design for embedded core test," *Proceedings IEEE Int. Test Conf.*, pp. 911 – 920, 2000.

[6] A. M. Amory et al., "Wrapper Design for the Reuse of a Bus, Network-on-Chip, or Other Functional Interconnect as Test Access Mechanism," *IET Comput. Digit. Tech.*, pp. 197-206, 2007.

[7] F. A. Hussin et al., "Optimization of NoC Wrapper Design Under Bandwidth and Test Time Constraints," *Proceedings IEEE Europ. Test Symp.*, pp. 35-42, 2007.

[8] B. Vermeulen and S. K. Goel, "Design for Debug: Catching Design Errors in Digital Chips," *IEEE Design & Test of Computers*, 19(3):35–43, 2002.

[9] B. Vermeulen et al., "IEEE 1149.1-Compliant Access Architecture for Multiple Core Debug on Digital System Chips," *Proceedings IEEE Int. Test Conf.*, pp. 55–63, 2002.

[10] B. Vermeulen et al., "Core-Based Scan Architecture for Silicon Debug," *Proceedings IEEE Int. Test Conf.*, pp. 638–647, 2002.

[11] IEEE Computer Society, IEEE Standard Test Access Port and Boundary-Scan Architecture-IEEE Std. 1149.1-2001, IEEE Press, 2001.

[12] ARM, ETM10 Technical Reference Manual, Nov. 2003.

[13] N. Stollon et al., "Multi-Core Embedded Debug for Structured ASIC Systems," *Proceedings DesignCon*, 2004.

[14] C. Ciordas et al., "An event-based monitoring service for networks on chip," *ACM TODAES*, 10(4):702–723, 2005.

[15] C. Ciordas et al., "Transaction Monitoring in Networks on Chip: The On-Chip Run-Time Perspective," *Proceedings IEEE IES*, pp. 1-10, 2006.

[16] S. Tang and Q. Xu, "A Multi-Core Debug Platform for NoC-Based Systems," *Proceedings of the Design, Automation and Test in Europe*, pp. 870-875, 2007.

[17] K. Goossens et al., "Transaction-Based Communication-Centric Debug," *Proceedings Int. Symp. on Networks-on-Chip*, pp. 95-106, 2007.

[18] IEEE Computer Society, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits, Aug. 2005.

[19] ARM, AMBA AXI Protocol Specification, V. 1.0, 2003.

[20] Philips Semiconductors, Device Transaction Level (DTL) Protocol Specification, V. 2.2, 2002.

[21] OCP International Partnership, Open core protocol specification 2.0, 2003.

[22] R. Kuppuswamy et al., "Full Hold-Scan Systems in Microprocessors: Cost/Benefit. Analysis," *Intel Technology Journal*, Vol. 18, No. 1, Feb. 2004.

[23] H. Yi and S. Kundu, "On Design of Hold Scan Cell for Hybrid Operation of a Circuit," *Proceedings IEEE Europ. Test Symp.*, 2008.

[24] H. Al-Asaad and P. Moore, "Non-Concurrent On-Line Testing Via Scan Chains," *IEEE Systems Readiness Technology Conference*, pp. 683-689, 2006.

[25] M. Beck et al., "Logic Design for On-Chip Test Clock Generation – Implementation Details and Impact on Delay Test Quality," *Proceedings of the Design, Automation and Test in Europe*, pp. 56-61, 2005.

[26] H. Yi et al., "Low Cost Scan Test for IEEE 1500-Based SoC," *IEEE Transactions on Instrumentation and Measurement*, pp. 1071-1078, May 2008.

[27] VLSI-WORLD, Glitch free safe clock switching, http://www.vlsi-world.com/content/view/64/47/1/0/.

[28] I. Silas et al., "System-Level Validation of the Intel® Pentium® M Processor," Intel Technology Journal, pp. 37-43, 2003.

[29] J. Gaisler et al., Gaisler Research IP Core's Manual, Ver. 1.0.16, 2007.