On Reusing Test Access Mechanisms for Debug Data Transfer in SoC Post-Silicon Validation

Xiao Liu and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {xliu,qxu}@cse.cuhk.edu.hk

Abstract

One of the main difficulties in post-silicon validation is the limited debug access bandwidth to internal signals. At the same time, SoC devices often contain dedicated bus-based test access mechanisms (TAMs) that are used to transfer test data between external testers and embedded cores. In this paper, we propose to reuse these precious TAM resources for real-time debug data transfer in post-silicon validation. This strategy significantly increases debug bandwidth with negligible routing overhead. To support different TAM architectures and debug scenarios, design for debug (DfD) structures are introduced at both core test wrapper level and system level. Simulation results demonstrate the effectiveness of the proposed approach at low DfD cost.

1 Introduction

Because of the high design complexity and the inaccurate abstracted models used in pre-silicon design and verification phases, today's complex system-on-a-chip (SoC) designs usually need to go through one or more re-spins to become bug-free [7], even though half of the system development effort is allocated to verification tasks [12]. With the ever-shrinking market window for integrated circuit (IC) product, a post-silicon validation strategy that helps identifying bugs effectively is of crucial importance.

Debugging silicon is an extremely complex process, wherein the main difficulty lies in the limited visibility of the circuit's internal signals. A widely-adopted technique utilized by the industry to mitigate this problem is to reuse the IEEE Std. 1149.1 (JTAG) test access port to run, halt and step embedded cores to observe whether the values in scan chains are expected values [17]. This technique is able to effectively identify those easy-to-find bugs that leave "evidences" when the SoC halts, but fails to find trickier bugs that manifest themselves only after a long time. Therefore, an emerging trend for today's complex SoC designs is to embed more design-for-debug (DfD) structures on-chip and to monitor and trace internal signals during normal operation [2, 4]. A large amount of trace data, however, require sizeable bandwidth to transfer and most today's SoC debug architectures introduce dedicated debug buses for this duty(e.g., [4, 13]). The routing of these debug buses inevitably causes significant DfD overhead to the design.

At the same time, SoC designs often contain dedicated bus-based test access mechanisms (TAMs) to deliver test stimuli and responses between automatic test equipment (ATE) and embedded cores [19]. For example, 140 TAM wires are fabricated on-chip for a complex video-processing SoC device [6], making all embedded cores visible to the external ATE. These precious design-for-test (DfT) structures, however, are usually left unused after manufacturing test. This is unfortunate because these TAM resources are able to provide a large communication bandwidth for internal signals.

Based on the above observation, in this paper, we propose to reuse the existing TAMs for silicon debug data transfer. This concept of reusing DfT structures for silicon debug is not new. Rather, it is similar to the strategy to reuse scan chains to "dump" data in post-silicon validation. The main difference lies in the fact that we are using these TAMs to transfer trace data at real-time. Because of this, we need to modify the design of core test wrappers so that those to-be-observed signals in a core can be traced out at functional mode through the wrapper. In addition, while different embedded cores connected to the same TAM bus are accessed sequentially during testing, in post-silicon validation, we may want to concurrently observe signals inside these cores. Specific DfD structures are introduced to address this issue as well in this work.

The remainder of this paper is organized as follows. Section 2 reviews related work in SoC test and debug infrastructure. In Sections 3, we present an overview of the proposed debug data transfer framework. Next, the newly-introduced DfD structures are introduced in Section 4. Section 5 then demonstrates how to extend the proposed method to support multi-core debug environments. Simulation results are presented in Section 6. Finally, Section 7 concludes this paper and describes some future work.

2 Related Work

Manufacturing test and post-silicon validation are challenging problems for today's complex SoC designs. A vast body of research has been endeavored to address the above issues. We briefly survey the related work in this section.

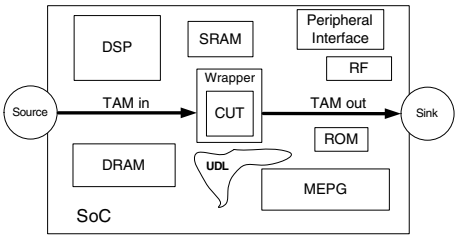
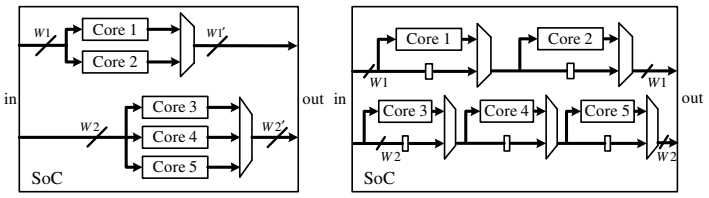


Figure 1. Conceptual SoC Test Architecture [20]



(a) Test Bus Architecture (b) TestRail Architecture

Figure 2. Test Bus and TestRail Architectures

2.1 SoC Test Architectures

Zorian et al. [20] presented a conceptual architecture for testing SoC devices, as illustrated in Fig. 1. The basic elements of this SoC test infrastructure include: (i). test source and sink that provide test stimuli and compare test responses (e.g., external ATE); (ii). test access mechanisms that transport test data from the source to the core under test (CUT) and from the CUT to the sink; (iii). core test wrapper that connects the core terminals to the rest of the chip and to the TAM, isolating the CUT from its environment during test.

The modular test architecture using bus-based TAMs, being flexible and scalable, are widely used in industry designs. The two most popular SoC test architectures, i.e., the Test Bus architecture [16] and the TestRail architecture [10] are depicted in Fig. 2. The number of TAM wires implemented on-chip varies with different SoC designs. Generally speaking, wider TAM width results in shorter testing time at a larger DfT area and routing cost. Also, large SoC designs usually implement more TAM wires on-chip to keep test cost under control. For example, 140 TAM wires are introduced in the SoC design in [6], which are able to provide a large communication bandwidth to embedded cores.

2.2 SoC Post-Silicon Validation Architectures

While we expect embedded cores to work in a "plug-and-play" fashion, a few "surprises" are often inevitably discovered in first silicon and requires silicon debug to identify the root causes [1]. Similar to the requirements for an efficient SoC manufacturing test strategy, an effective SoC post-silicon validation solution demands good controllability and observability of the design's internal nodes. In fact, as diagnosing an error is always much harder than detecting an error, silicon debug requires to increase these capabilities to a much higher level.

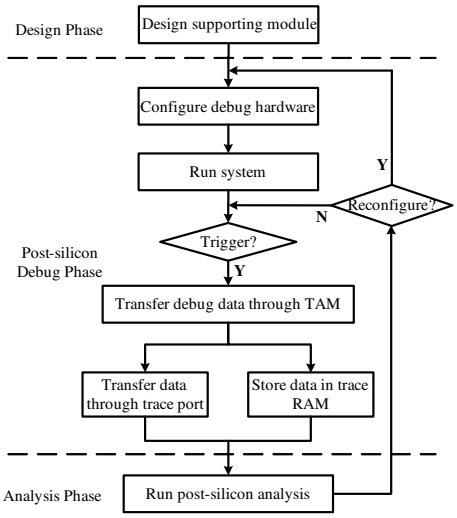


Figure 3. Post-Silicon Debug Flow.

Basic postmortem debuggability can be provided by capturing snapshots of the circuit's internal sequential elements through JTAG run-control interface and scan chains [9]. This low-cost technique however is not enough for tracking tricky bugs that manifest themselves after a long period of operation. To tackle this problem, dedicated DfD structures that facilitate to trace the circuit's operations at real-time in normal functional mode (e.g., the instruction flow of an embedded processor) are implemented in most modern SoC devices. In such systems with tracing (e.g., [4, 11]), typically hardware triggers are implemented to start and stop a trace process and filter the information to be traced out.

Today's SoC devices contain an increasing number of embedded cores, and it is essential to debug the complex interactions between multiple embedded processor cores and their active peripherals. Several multi-core debug solutions have been presented in the literature and adopted by the industry. For example, in the ARM *CoreSight* debug architecture [4], each ARM core is equipped with an embedded trace macrocell (ETM) that captures the processor's states [5]. The captured information can be stored to a trace buffer or exported immediately through a external trace port. Both require dedicated trace bus to deliver debug data from every to-be-traced core. With the ever-increasing SoC design complexity, the volume of trace data is expected to increase to identify bugs effectively, despite the use of cross-trigger and various trace qualification techniques (e.g., [3, 8, 15]). Delivering such large volume of debug data to the trace buffer or trace port requires a great bandwidth. Consequently, the routing cost for dedicated trace buses is quite high.

3 Overview of the Proposed Debug Data Transfer Framework

Before introducing the technical details of our debug data transfer framework with bus-based TAMs, let us examine the post-silicon debug flow used in our solution first, as depicted in Fig. 3. During the design phase, various DfD structures

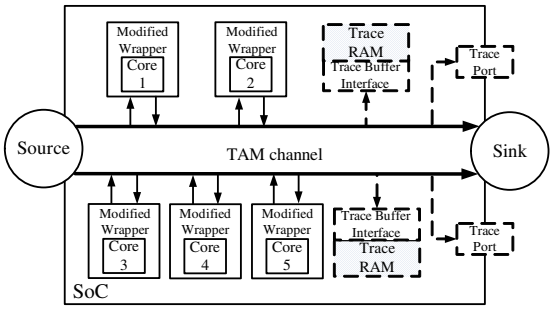


Figure 4. Proposed Debug Data Transfer Framework with Bus-Based TAMs.

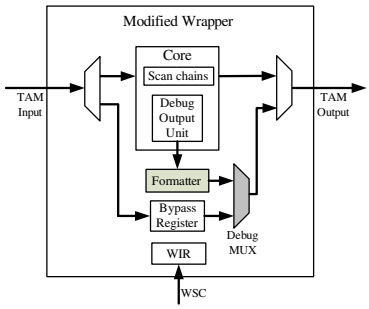


Figure 5. Modified Wrapper Design.

that support hardware triggers and core internal signals' observation are implemented. The trigger mechanisms can be simple triggers implemented with comparators and/or counters (e.g., [18]), complex cross-trigger network (e.g., [4, 14, 15]) or even reconfigurable trigger fabrics (e.g., [1]). The to-be-traced signals embedded deeply inside a core can be brought to core boundaries by either tapping through simple multiplexer-based network (e.g., [1, 17]) or packaging in a small first-in first-out (FIFO) queue such as the ARM embedded trace macrocell [4]. During the post-silicon debug process, we first enable the aforementioned DfD structures and then put the system into normal functional mode. Once a trigger condition is hit, the traced signals are transferred along bus-based TAMs to either an internal trace buffer or an external trace port. We often need to change trigger conditions and/or trace different signals during the silicon debug process. The above operations are conducted through reconfigurations.

The SoC test infrastructures, however, are not designed to support real-time debug data tracing at functional mode. We need to insert some DfD structures to make this possible, as shown in Fig. 4. First of all, the original test wrappers need to be modified so that the traced signals can flow into TAMs in normal functional mode. Secondly, trace buffers (if any) are attached to every TAM bus to store debug data from cores under debug. As there might be multiple embedded cores on a TAM bus and they might send debug data at the same time, certain mechanisms need to be designed to avoid data corruption.

It should be also pointed out that data transfer on TAMs are controlled by a global test clock signal and its speed might be slower than that of the functional cores. Similar to transferring debug data on ARM trace buses [4], designers need to take this into account when transporting trace data on TAMs.

4 Proposed DfD Structures

The main objective of the proposed solution is to facilitate real-time trace data transfer through TAM channel in functional mode. To support this, the core test wrapper needs to be modified so that the *raw* debug data can be written into internal trace memory or external trace port in appropriate format in mission mode. If trace buffer is utilized, we also need to design a proper trace buffer interface to deal with TAM with various widths, as shown in this section.

4.1 Modified Wrapper Design

From Fig. 5, it can be easily observed that we have introduced a *formatter* and a *debug MUX* into the modified wrapper to conduct its duty. The debug multiplexer is utilized to select the data source of the TAM in mission mode. A new wrapper instruction "WR_DEBUG" is introduced to enable real-time debug data transfer for core under debug (CUD) at functional mode, by controlling the debug MUX. That is, when this instruction is applied, the debug data coming out of *debug output unit* will be sent onto TAM through the formatter. Otherwise, the data in the bypass register will be delivered onto TAM.

The formatter is utilized to convert the raw debug data from the CUD into a format suitable for transfer. As discussed earlier, these debug data out of the CUD's debug output unit (see Fig. 5) can be signals directly tapped using a multiplexer-based network (e.g., [1, 17]) or data stored in a FIFO (e.g., [4]). In our debug data transfer framework, logic '0' is put onto TAM when no CUDs are sending data to the TAM (see Figure 6). Once a CUD plans to send out debug data (e.g., hardware trigger is hit), it will first send an identification tag (ID) which starts with a logic '1' so that the receiving side (e.g., trace buffer) is able to identify the start of a debug data transfer. The length of this trace process and the timestamp that the trace starts can be optionally sent onto the TAM before the actual debug data is sent out. In addition to the above, a complex formatter with a FIFO itself can temporarily store the trace data and compress them before sending them out, in order to increase the utilization rate of the TAM for debug data transfer and avoid debug data data loss.

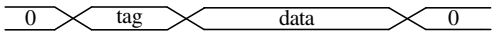


Figure 6. Debug Data Transfer on TAM.

There are some differences when we configure the wrapper for Test Bus architecture and TestRail architecture, considering the case when only one core is permitted to occupy a TAM (see Section 5 for the case when multiple cores send debug data onto a TAM). In Test Bus architecture (see Fig. 2(a)), a system-level multiplexer is used to select the core that connects to TAM. Therefore, all cores can trace their debug data simultaneously and it is this multiplexer controlled by debug controller that determines which CUD can send debug data out at a particular time. For TestRail architecture (see Fig. 2(b)),

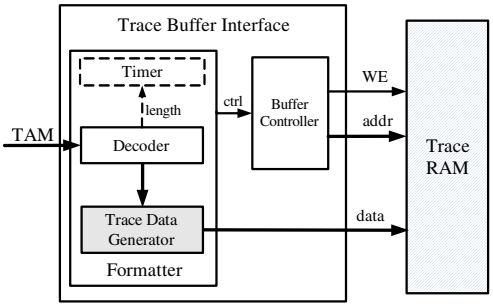


Figure 7. Trace Buffer Interface Design.

however, as the TAM connects all cores in a daisy chain manner, only one core can be put in debug mode and send debug data while the other cores should be put in normal functional mode, in order to avoid data corruption because of contention.

4.2 Trace Buffer Interface Design

Fig. 7 illustrates the structure of the trace buffer interface for the case when debug data is transferred to internal trace memories. Because the TAM width can be an arbitrary value different from the trace memory bit-width, we also need a formatter to conduct the matching. In addition, the formatter needs to detect the start of debug data transfer by identifying the non-'0' ID with decoder module. The data length is stored into timer and is used to count down the trace data to be written into the memory.

In addition, we can also conduct debug data processing inside the trace buffer interface before writing them into the trace memory. For example, temporal information can be introduced and data compression can be conducted in the module of trace data generator to facilitate debug efficiency.

5 Sharing TAM for Multi-Core Debug Data Transfer

Since embedded cores communicate with each other during normal operation, it is often necessary to debug the complex interactions between multiple cores, especially for SoCs with many embedded processors. In the proposed debug data transfer framework, it is likely that some interacting cores connect to the same TAM and send out debug data concurrently. This is not a problem for Test Bus architecture because only one core is allowed to send data onto TAM. However, for TestRail architecture, multiple cores are daisy chained in one TAM and the debug data might get corrupted because of contention, if care is not taken. We therefore introduce a "core masking" strategy to tackle this problem. In order to support real multi-core debug that sends data onto the TAM concurrently, we propose to split the TAM into several sub-channels and these CUDs send data to different sub-channel to achieve the above objective without data corruption.

5.1 Core Masking for TestRail Architecture

Core masking strategy resolves the aforementioned debug data corruption problem in TestRail architecture by allowing

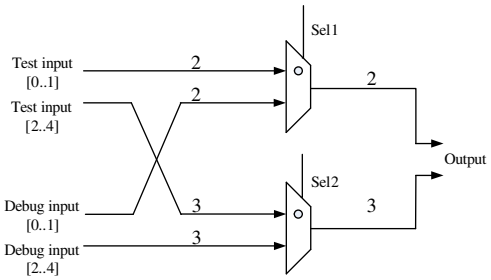


Figure 8. Sharing TAM for Debug Data Transfer with Channel Split.

Mode	Sel1	Sel2	Output[0..1]	Output[2..4]
Other	1	1	Test[0..1]	Test[2..4]
Share (low-half)	0	1	Debug[0..1]	Test[2..4]
Share (high-half)	1	0	Test[0..1]	Debug[2..4]
Debug	0	0	Debug[0..1]	Debug[2..4]

Table 1. Mux Control for the Channel Split Strategy.

only one core to send debug data onto the TAM at a specific time. Different from [4] that utilizes dedicated control signals to control every core, we introduce a one-bit *mask* signal generated from a *monitor* module to connect to every CUD on the same TAM. Because of the unidirectional characteristic of TestRail architecture, this monitor connects to the far end of this TAM.

The *core masking* strategy works as follows. The *monitor* unit keeps observing the activities on the TAM. Once it detects a non-'0' ID signal (i.e., a transfer request), it will assert the mask signal. For those cores that run normally and do not transfer debug data onto the TAM, the debug multiplexer in their wrappers (see Fig. 5) will select the data source from bypass register when detecting the assertion of the mask signal.

It should be noted that because a TAM goes through the bypass in every core on the TAM, it is possible that when one core's ID arrives at the monitor, some other CUDs on the TAM have also sent transfer requests. Debug data corruption will inevitably occur in such case. Designers can reconfigure the triggers in those CUDs to avoid this situation. A better solution however is to let those CUDs to be able to actually send data at the same time, as illustrated in the following "channel split" method.

5.2 Channel Split

The "channel split" strategy is to divide the TAM into several sub-channels so that every CUD that needs to send debug data concurrently can occupy a sub-channel by itself. Apparently, this capability is achieved under the condition that these CUDs' debug bandwidth requirements can be satisfied with the sub-channels, and the formatters in their corresponding wrappers need to be configured accordingly.

To apply this methodology in TestRail architecture, the debug multiplexer inside the core wrapper (see Fig. 5) is replaced by a multiplexor network. Fig. 8 presents an example in which a 5-bit TAM is split into a 2-bit sub-channel and a 3-bit sub-

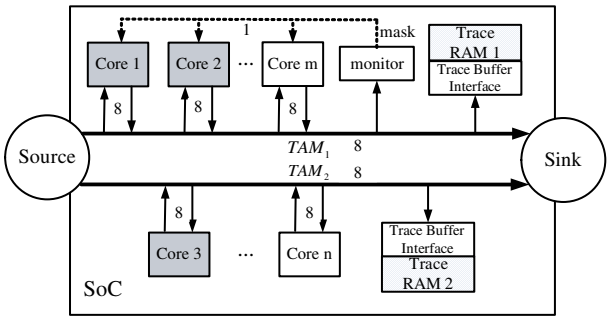


Figure 9. Experimental Setup.

channel for debug data transfer. Here 'Test input' and 'Debug input' are connected to bypass register and debug output unit in the wrapper, respectively. By doing so, only part of the TAM is occupied by this core since the other part goes through bypass register inside the wrapper. To make this happen, again, we need to introduce extra wrapper instruction. By decoding this instruction, the control signals for the multiplexor network are set to appropriate values and an example is illustrated in Table 1.

To apply this methodology in Test Bus architecture, in addition to the above modifications, the system-level multiplexer (see Fig. 2(a)) needs to be expanded to connect the combined sub-channels to the TAM while keeping its original exclusive characteristics, controlled by the system debug controller.

While the above channel split method effectively supports concurrent multi-core debug data transfer, it might not be enough if the total debug bandwidth requirements exceed what can be provided by the TAM. In such case, we might have to add dedicated trace bus to resolve this issue.

6 Experimental Results

To verify the proposed debug data transfer framework, we present simulation results for two debug scenarios for a hypothetical SoC. This SoC uses a TestRail architecture and contains two 8-bit wide TAMs, as shown in Fig. 9. Both TAMs connect to an internal trace memory with 16-bit data bit-width. We target on debugging Core 1 and Core 2 connected on TAM_1 and Core 3 on TAM_2. The trace data bit-widths out of the debug output unit for all these 3 cores are 16-bit.

In our first experiment, we present simulation results for the "core masking" strategy. As shown in Fig. 10, after trigger request (*trigger_core1*) is activated from Core 1, the corresponding debug data transfer operation starts, which can be observed from the rising edge of write enable signal (*we_trace_RAM1*) together with the increment of address signal (*addr_trace_RAM1*). Debug data (*dataout_trace_RAM1*) are thus written into the trace memory on TAM_1. It can be also seen that the mask signal (*mask*) is asserted after detecting this event. During the debug data transfer process for Core 1, Core 2 has also detected a trigger hit (*trigger_core2* gets asserted), but because of the asserted mask signal, Core 2 could not send debug data onto TAM_1 and we can see a constant value '0x0040' that is sent from core 1. Only after this transfer process is finished, *mask* signal is de-asserted. Core 2 then occupies TAM_1 and starts to transfer its debug data. It can be also observed in Fig. 10 that, since TAM_2 is a separate channel, the debug data transfer for Core 3 works independently.

Module	Area	
	Experiment 1	Experiment 2
Formatter (in wrapper)	380	289
Trace Buffer Interface	651	706
MUX (in wrapper)	160	151
Monitor	139	

Table 2. DfD Area Cost.

Our second experiment is to validate the *channel split* method applied on TAM_1, in which Core 1 and Core 2 share the TAM wires by utilizing the lower half and the upper half 4-bits, respectively. It should be noted to accommodate the bandwidth decrease in such case, the trace data from each debug output unit and the the CUD formatter module is reduced to 8 bits and 4 bits, respectively. As depicted in Fig. 11, after trigger signal (*trigger_core1*) is asserted in Core 1, 16-bit data (*dataout_trace_RAM*) is written into trace memory, in which only half of it contains useful record information '0x41' from core 1. When the debug data transfer is also activated for Core 2 (*trigger_ core2* gets asserted), the 16-bit data now contain the debug data for both Core 1 and Core 2. As can be observed in Fig. 11, the 'ID' signals for Core 1 ('0x01') and Core 2 ('0x02') are first written into trace memory before the actual debug data.We can easily analyze the mixed debug data inside the trace memory according to the following rules:

$$debug\_data\_core1 = data\_RAM[7:0]$$

$$debug\_data\_core2 = data\_RAM[15:8]$$

Finally, in terms of DfD cost, Table 2 present the silicon area of each newly-introduced DfD unit in the above two debug strategies using a commercial synthesis tool. The DfD cost of each core wrapper, trace buffer interface and monitor are all in the hundreds of 2-input NAND gates range, which is quite small. Considering the proposed technique saves the routing cost for dedicated trace bus, the total DfD cost for SoC post-silicon validation is significantly reduced.

7 Conclusion and Future Work

In this paper, we present a new silicon debug data transfer framework by reusing existing on-chip TAM resources. We propose to modify the design of core test wrappers so that core internal signals can be traced out at functional mode. We also describe two techniques that enable multiple cores to send debug data to the same TAM. Simulation results show the correct behavior of the proposed solutions.

Currently, the SoC test architecture design and optimization focuses on test time reduction [19]. The resultant test architecture thus may not be efficient in terms of debug data transfer. In our future work, we plan to take the cores' debug requirements into consideration during test architecture design so that the TAM bandwidth utilization rate for debug can be enhanced.

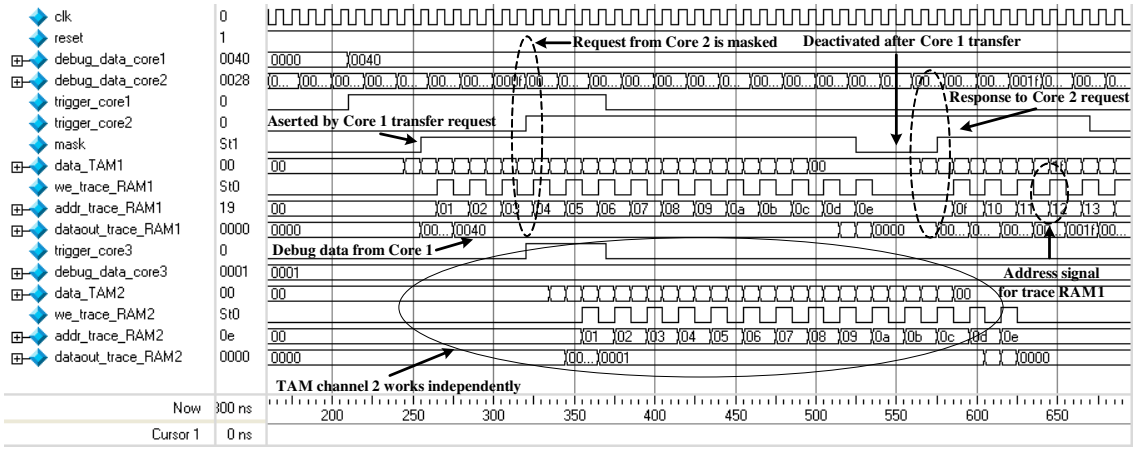


Figure 10. Simulation Results for Core Masking Strategy.

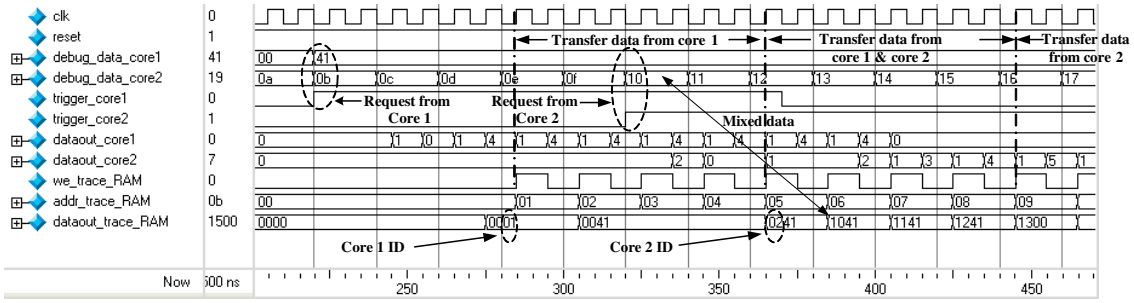


Figure 11. Simulation Results for Channel Split Strategy.

8 Acknowledgements

The authors would like to thank Dr. Shan Tang from T3G Technology Co., Ltd. for his insightful comments to this work.

References

[1] M. Abramovici, *et al.* A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 7–12, 2006.

[2] Altera Inc. Design Debugging Using the SignalTap II Embedded Logic Analyzer. http://www.altera.com.

[3] E. Anis and N. Nicolici. On Using Lossless Compresstion of Debug Data in Embedded Logic Analysis. In *Proceedings IEEE International Test Conference (ITC)*, paper 18.3, 2007.

[4] ARM Ltd. How CoreSight Technology Gets Higher Performance, More Reliable Product to Market Quicker. http://www.arm.com.

[5] ARM Ltd. Embedded Trace Macrocell Architecture Specification. http://www.arm.com/.

[6] S. K. Goel, *et al.* Test Infrastructure Design for the Nexperia^{TM} Home Platform PNX8550 System Chip. In *Proceedings Design, Automation, and Test in Europe (DATE) Designers Forum*, pages 108–113, 2004.

[7] A. B. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems On-chip: A Review. In *IEE Proceedings, Computers and Digital Techniques*, pages 197–207, 2006.

[8] A. B. T. Hopkins and K. D. McDonald-Maier. Trace Algorithms for Deeply Integrated Complex and Hybrid SoCs. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 641–646, 2007.

[9] G. Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pages 892–902, 1999.

[10] E. J.Marinissen, *et al.* A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, 1998.

[11] MIPS Technologies Inc. EJTAG Trace Control Block Specification. http://www.mips.com.

[12] Semiconductor Industry Association (SIA). *The International Technology Roadmap for Semiconductors (ITRS): 2003 Edition*. http://public.itrs.net/Files/2003ITRS/Home2003.htm, 2003.

[13] N. Stollon, R. Leatherman, B. Ableidinger, and E. Edgar. Multi-Core Embedded Debug for Structured ASIC Systems. http://www.fs2.com/.

[14] S. Tang and Q. Xu. A Multi-Core Debug Platform for NoC-Based Systems. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 870–875, 2007.

[15] S. Tang and Q. Xu. In-band Cross-trigger Event Transmission for Transaction-based Debug. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 414–419, 2008.

[16] P. Varma and S. Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–302, 1998.

[17] B. Vermeulen, T. Waayers, and S. Bakker. IEEE 1149.1-Compliant Access Architecture for Multiple Core Debug on Digital System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 55–63, 2002.

[18] B. Vermeulen, T. Waayers, and S. K. Goel. Core-Based Scan Architecture for Silicon Debug. In *Proceedings IEEE International Test Conference (ITC)*, pp. 638–647, 2002.

[19] Q. Xu and N. Nicolici. Resource-Constrained System-on-a-Chip Test: A Survey. *IEE Proceedings, Computers and Digital Techniques*, 152(1):67–81, Jan. 2005.

[20] Y. Zorian, E. J. Marinissen, and S. Dey. Testing Embedded-Core-Based System Chips. *IEEE Computer*, 32(6):52–60, June 1999.