Design-for-debug for post-silicon validation: Can high-level descriptions help?

Nicola Nicolici and Ho Fai Ko
Department of Electrical and Computer Engineering
McMaster University, Hamilton, ON L8S 4K1, Canada
Email: nicola@ece.mcmaster.ca, henryko@grads.ece.mcmaster.ca

Abstract

Post-silicon validation is an essential step in the design flow, which is needed to demonstrate that the implemented circuit meets its intended behavior. Due to lack of in-system controllability and observability, design-for-debug hardware is employed to aid post-silicon validation. A number of solutions have been proposed to implement the design-for-debug hardware, as well as to analyze the debug data that is acquired. Although the design entry is done at the register-transfer level, the existing approaches to aid post-silicon validation rely primarily on the information extracted from the gate level circuit descriptions. We anticipate that, as the design complexity continues to grow, extracting and processing circuit information at this level will become increasingly difficult. In this paper, we briefly summarize the known art and discuss some possible directions of investigation that can utilize high-level circuit descriptions to augment the existing solutions.

1. Introduction

The limited accuracy in circuit modeling and the exponential size of state spaces are the main reasons why pre-silicon verification is insufficient in eliminating all the design errors before tape-out. This problem is further aggravated by the increasing number of on-chip logic blocks and the complex transactions between them, as well as the continuous growth of embedded software as the product differentiating component in system-on-a-chip (SOC) designs. In order to reduce the cost of re-spins (both mask costs and the implementation time), it is essential to validate the implemented design and to identify the escaped bugs as soon as the first silicon is available. Post-silicon validation (PSV) is the task of searching for erroneous behavior in-silicon (or experimentally showing that the implemented design meets its intended behavior). If the erroneous behaviour is found, then a direct consequence is an added debug step for the identification and localization of bugs in silicon. This explains why this task is often referred to as post-silicon validation and debugging, or just silicon debug.

The inability to access internal signals in-system, which results in limited controllability and observability of the circuit, is a major challenge to PSV. For one to be able to gather as much data as possible from the circuit-under-debug (CUD) in order to understand the nature of the error, design-for-debug (DFD) hardware is commonly inserted into the design. Scan chains and trace buffers, which are detailed in the following section, are the two most commonly used DFD techniques. Both techniques have been successfully deployed in practice. However, as the hardware and software interactions between different blocks in an SOC are difficult to be verified pre-silicon, it is expected that more DFD hardware will be deployed in the future. In order to reduce the implementation cycle, while at the same time it is important to avoid an excessive overhead introduced by the DFD hardware, it is becoming a significant challenge to reach the suitable decisions for DFD insertion.

Since the debug hardware is fine-tuned to the CUD's specific needs based on the design description, the state-of-the-art tools rely on processing the netlist available at the gate level of design abstraction. Because the design entry for the debug controllers is done at the register-transfer level (RTL), a different perspective would be to leverage the descriptions available at the higher levels of design abstraction to help with exploring a larger implementation space for DFD insertion. For example, the order of scan cells in a scan chain constrains the set of two-pattern pairs that can be applied for speed-path debugging. It is possible that deciding to create the scan chain order at the RTL can benefit post-silicon validation and debugging. Similarly, we anticipate that deciding on which signals to trace (thus impacting the real-time observability) can benefit from the high-level design information. In this paper we outline a few challenges that arise when raising the level of design abstraction for implementing the known DFD techniques.

The organization of this paper is as follows. Section 2 overviews the scan and trace techniques, while Section 3 discusses the advantages and challenges when working at higher levels of design abstraction than the gate level. Section 4 provides the concluding remarks.

978-1-4244-4823-4/09/$25.00 ©2009 IEEE

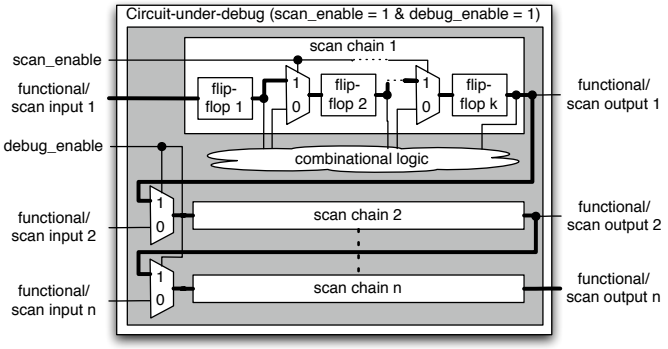


Figure 1. Scan-based debug

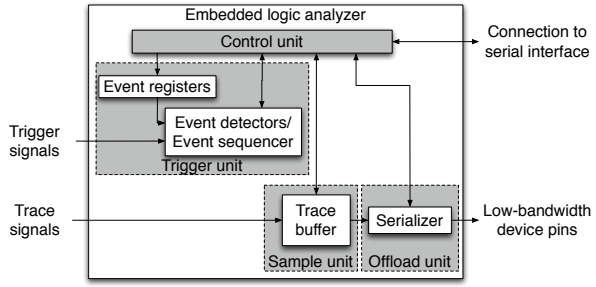
2. Design-for-Debug Techniques

In this section we overview the two main DFD techniques that rely on scan chains and trace buffers.

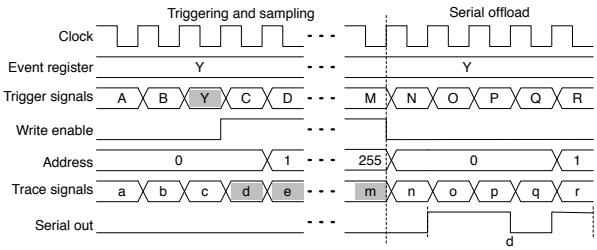
2.1. Scan-based Technique

The primary goal of the scan-based technique is to reuse the internal scan chains, which are placed in the CUD to increase the controllability and observability during manufacturing test [8], to acquire debug data during PSV. During manufacturing test, the functional pins are used as scan pins for loading multiple scan chains concurrently to reduce test time. However, during debug, these scan chains are concatenated, as shown in Figure 1. In this case the scan chain is loaded/unloaded through a serial interface, which is accessible in-system. By capturing data in the internal state elements and offloading them through the scan chains (called scan dump), failure analysis can be performed offline to identify bugs in a design. The design of the glue logic to reuse the scan chains for debug is well-documented in the literature, e.g., [11, 12, 20–22].

If the circuit's behavior is deterministic, i.e., all the inputs are controllable by the user during debug, a scan-based debug experiment can be started, stopped and resumed from any state of interest. Hence, once the failure is controllable and the debug experiment can be repeated by the user, scan chains are essential for debugging. They can be used from finding the root cause of a design error to speed-path debugging. However, before the debug experiment is reproducible, which is not the case for most failures in-field, there is little knowledge about what caused the failure at the observable outputs. Therefore, stopping the debug experiment and re-running it will not guarantee that the failure will occur again. One can let the circuit execution to continue by offloading the scan chains through shadow latches, however this may incur a larger area penalty; besides, until a scan dump is completed it is not possible to capture data in consecutive clock cycles. More importantly, the captured data is always done in reaction to an event of interest, and hence it is difficult to record the states that lead to that particular event, which may be crucial during debugging.



(a) Example of an embedded logic analyzer



(b) Timing diagram for utilizing ELA during debug

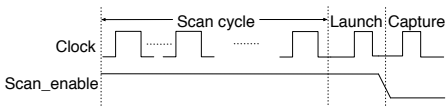
Figure 2. Trace-based debug

2.2. Trace-based Technique

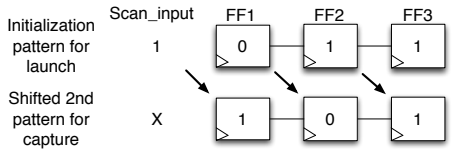
The previously-outlined limitations of scan chains are addressed through the use of embedded logic analyzers (ELAs) [2, 18, 19], which can trace a restricted group of signals in real-time before or after the event of interest. An example of an ELA is shown in Figure 2(a), which is divided into four components: control unit, trigger unit, sample unit, offload unit. The control unit contains one or more finite state machines (FSMs) with programmable registers. The programmable registers can be configured using a serial interface like JTAG [10] for receiving control instructions. This allows the FSMs to control the other units in the ELA to gather different sets of data in multiple experiments.

As shown in Figure 2(b), the trigger unit monitors a set of trigger signals for the occurrence of trigger events described using one or more programmable event registers and event detectors. When such events are detected, the control unit will start the data acquisition by activating the sample unit. The sample unit, which contains a trace buffer (i.e., embedded memory), acquires data on the selected set of trace signals in real-time. When the trace buffer is full, its content can be offloaded serially through low speed device pins. Alternatively, the trace buffer can be viewed as a circular buffer and the acquisition stops after the trigger event.

It is obvious that the amount of data that can be acquired is limited by the trace buffer *depth*, which limits the number of samples to be stored, and its *width*, which limits the number of trace signals sampled in each clock cycle. To reduce the amount of debug data to be stored on-chip, a finer control over when to acquire data has been explored [2, 6, 16] and compression techniques have been researched [3, 4, 7].



(a) Clocking for at-speed debugging using scan



(b) Pattern correlation for at-speed debug using scan

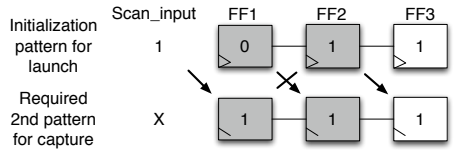
Figure 3. Scan for at-speed debug

3. How Can High-level Descriptions Help?

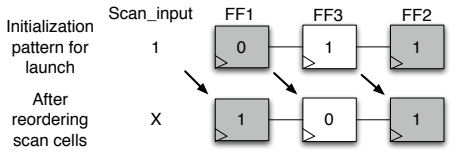
Before discussing several directions where high-level descriptions can help, it is essential to point out that design capture for many components of the the DFD infrastructure is already done at the RTL. Examples include: scan controllers [20], embedded logic analyzers [1,13], assertion checkers [5] and trigger units [16], compression logic [3,4].

Nonetheless, some key components of the DFD infrastructure, such as scan chains, are currently inserted at the gate level of design abstraction. When dealing with design errors that affect only the static behavior of the circuit, the order of scan cells in the scan chain does not matter. This is because a single pattern is applied to the circuit using a low-speed clock, which can be the same as the load/unload clock. However, when debugging at-speed, a pair of patterns are required to sensitize the speed-paths. The first pattern, labeled as the initialization pattern, is shifted into the scan chains and by keeping the scan chains in the shift mode during the launch cycle, the second pattern is obtained by shifting the initialization pattern for one clock cycle as shown in Figure 3. Subsequently the response is captured after an at-speed clock pulse. This type of two-pattern application through scan chains is called launch-off-shift or skewed-load [8] and it is employed for delay-fault testing.

There is, however, a correlation between the initialization pattern and the excitation pattern which may prohibit some pattern pairs to be applied to the CUD. Using the pattern pair given for FF1 and FF2 in Figure 4(a) and the current scan cell order as an example, one can see that the required pattern pair for FF1 and FF2 cannot be obtained. This problem can be addressed by reordering the scan cells in the scan chain, as shown in Figure 4(b). As it can be seen from Figure 4, if the neighboring scan cells drive different cones of logic then speed-paths in each cone can be debugged independently. Capturing this type of constraints in the scan chain design process can be done easier at the RTL, due to reduced design complexity. With some preliminary work presented in [14], more research can be done to embed other types of constraints that affect the speed-paths, such as simultaneous switching noise.



(a) Source of pattern correlation in scan chains



(b) Pattern correlation is removed after reordering

Figure 4. Reordering scan cells for debug

With respect to the trace-based technique, it is important to note that instead of improving observability of a design by acquiring more data into the trace buffer, solutions that focus on better utilizing the acquired data have been proposed recently. For example, [9] introduced the idea to restore data in the combinational part of the circuit using the information of the state elements obtained from scan dumps. On the other hand, [15] restores data in state elements across multiple time frames by using the acquired data on the subset of trace signals and the circuit information in the gate level netlist . These methods give the debug engineers more data for analysis during PSV. However, the amount of data that can be restored is dependent on the set of signals that are traced by the ELA. As a result, [15] also proposed an automated solution, which is followed up in [17], to analyze the circuit netlist to recommend which signals to trace. The existing approaches for trace signal selection can be captured using the flow from Figure 5 and they are motivated by the fact that data restoration can be done using simulation-based techniques at the gate level.

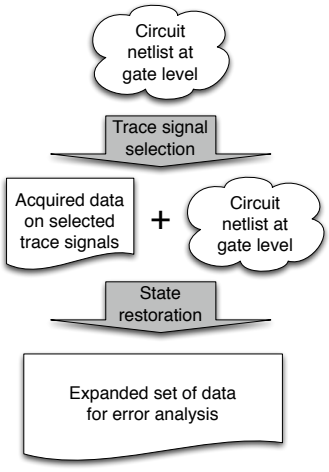


Figure 5. Trace signal selection and state restoration using gate level netlist

When using only the gate level information from the CUD, it is often difficult or very time consuming to identify essential signals for tracing, which designers can select easily by exploiting the high-level design knowledge. For example, it is common for debug engineers to acquire data on the control signals (e.g. pipe control signals), such that additional information (e.g. values in the pipe data registers) can be reconstructed offline. On the one hand, when analyzing only the gate level netlist, it is not obvious how to categorize which circuit elements belong to the control path or the data path. On the other hand, this information can be easily extracted from circuit descriptions at the RTL.

It takes, however, tremendous effort to correlate the information from high-level circuit description with what actually is happening in silicon during PSV. For example, one may not be able to find signals identified at the RTL at the gate level description. This is due to the fact that it is designed to pertain different information in circuit descriptions at different levels of design abstraction. As a result, we envision that future researches for aiding the debug process during PSV should place focus on extracting circuit information from multiple levels of design abstraction.

Our ongoing research is currently investigating a two-tier framework for analyzing circuit information. The first tier starts by analyzing the circuit description at the RTL. This information is then used to prune the search space during circuit analysis in the second tier, which is done at the next level of design abstraction (i.e., gate level). Since the search space at the lower level of design abstraction is pruned, the time required for analyzing the larger circuit description can be controlled to an acceptable level for complex designs. Moreover, when information from both design abstraction levels are presented, a more comprehensive view of the design can be created. This, in turn, allows automated tools to have a better picture of the behavior of the concerned design and thus, gives a more accurate analysis and recommendation on how DFD hardware can be designed or utilized to improve circuit observability.

4. Conclusion

A number of solutions have been proposed in the literature to improve observability using DFD hardware during post-silicon validation and debugging. The existing solutions generally rely on information extracted from the circuit descriptions at a lower-level of design abstraction, such as the gate level. As design complexity continues to increase, the runtime of these solutions for analyzing the designs may become unacceptable. We have discussed some possible directions of investigation on how high-level descriptions can help with building DFD hardware that will ultimately aid observability. For the same complexity reason motivated by large designs, we expect that subsequent steps in the debug flow, such as data analysis, can also leverage the DFD hardware built using the high-level descriptions.

References

[1] M. Abramovici. In-System Silicon Validation and Debug. *IEEE Design and Test of Computers*, 25(3):216–223, 2008.

[2] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 7–12, 2006.

[3] E. Anis and N. Nicolici. Low Cost Debug Architecture using Lossy Compression for Silicon Debug. In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe*, pages 225–230, 2007.

[4] E. Anis and N. Nicolici. On Using Lossless Compression of Debug Data in Embedded Logic Analysis. In *Proceedings of the IEEE International Test Conference*, 2007. Paper 18.3.

[5] M. Boule, J.-S. Chenard, and Z. Zilic. Adding Debug Enhancements to Assertion Checkers for Hardware Emulation and Silicon Debug. In *Proceedings of the IEEE International Conference on Computer Design*, 2006.

[6] M. Boule, J.-S. Chenard, and Z. Zilic. Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 613–620, 2007.

[7] M. Burtscher, I. Ganusov, S. Jackson, J. Ke, P. Ratanaworabhan, and N. Sam. The VPC Trace-Compression Algorithms. *IEEE Transactions on Computers*, 54(11):1329–1344, Nov 2005.

[8] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000.

[9] Y.-C. Hsu, F. Tsai, W. Jong, and Y.-T. Chang. Visibility Enhancement for Silicon Debug. In *Proceedings of the IEEE/ACM Design Automation Conference*, pages 13–18, 2006.

[10] IEEE JTAG 1149.1-2001 Std. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. IEEE Computer Society, 2001.

[11] D. Josephson. The Manic Depression of Microprocessor Debug. In *Proceedings of the IEEE International Test Conference*, pages 657–663, Oct 2002.

[12] D. Josephson and B. Gottlieb. The Crazy Mixed up World of Silicon Debug. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 665–670, 2004.

[13] H. F. Ko, A. B. Kinsman, and N. Nicolici. Distributed Embedded Logic Analysis for Post-Silicon Validation of SOCs. In *Proceedings of the IEEE International Test Conference*, 2008. Paper 16.3.

[14] H. F. Ko and N. Nicolici. Functional Scan Chain Design at RTL for Skewed-load Delay Fault Testing. In *Proceedings of the 13th IEEE Asian Test Symposium*, pages 454–459, 2004.

[15] H. F. Ko and N. Nicolici. Algorithms for State Restoration and Trace Signals Selection for Data Acquisition in Silicon Debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(2):285–297, Feb 2009.

[16] H. F. Ko and N. Nicolici. Resource-Efficient Programmable Trigger Units for Post-Silicon Validation. In *Proceedings of the IEEE European Test Symposium*, pages 17–22, 2009.

[17] X. Liu and Q. Xu. Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation. In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe*, pages 1338–1343, 2009.

[18] K. Morris. On-Chip Debugging - Built-in Logic Analyzers on your FPGA. *Journal of FPGA and Structured ASIC*, 2(3), Jan 2004.

[19] M. Riley and M. Genden. Cell Broadband Engine Debugging for Unknown Events. *IEEE Design and Test of Computers*, 24(5):486–493, 2007.

[20] G. Van Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone are Not Enough. In *Proceedings of the IEEE International Test Conference*, pages 892–902, 1999.

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

[22] B. Vermeulen, T. Waayers, and S. Goel. Core-Based Scan Architecture for Silicon Debug. In *Proceedings of the IEEE International Test Conference*, pages 638–647, Oct 2002.