Visibility Enhancement for Silicon Debug

Yu-Chin Hsu Furshing Tsai Wells Jong Ying-Tsai Chang
Novas Software
2025 Gateway Place, no. 400
San Jose, CA 95110
+1-408-467-7888
{ychsu, ktsai, wtjong, ytchang}@novas.com

ABSTRACT
Several emerging Design-for-Debug (DFD) methodologies are addressing silicon debug by making internal signal values and other data observable. Most of these methodologies require the instrumentation of on-chip logic for extracting the internal register data from in situ silicon. Unfortunately, lack of visibility of the combinational network values impedes the ability to functionally debug the silicon part. Visibility enhancement techniques enable the virtual observation of combinational nodes with minimal computational overhead. These techniques also cover the register selection analysis for DFD and multi-level design abstraction correlation for viewing values at the register transfer level (RTL). Experimental results show that visibility enhancement techniques can leverage a small amount of extracted data to provide a high amount of computed combinational signal data. Visibility enhancement provides the needed connection between data obtained from the DFD logic and HDL simulation-related debug systems.

Categories and Subject Descriptors
B.7 [Integrated circuits]; J.6[Computer-aided engineering]

General Terms
IC Design, Verification

Keywords
Functional verification, Silicon Debug, Silicon validation

1. INTRODUCTION
The process for productizing and delivering an idea to market can typically be broken into two major stages: concept-to-prototype and prototype-to-volume. For IC designs, the first stage encompasses the time from creating the specification to the arrival of first silicon prototypes. The second stage involves testing and debug of these silicon prototypes using automated test equipment (ATE), performing in-situ system validation of the prototypes, and ensuring the manufacturing process is sufficient for volume production. Recent trends indicate that while the amount of time is stable to slightly decreasing for the first stage, the time required to move from prototype to volume production is increasing**[1]**,**[2]**. We propose new technologies that, working with on-chip debug infrastructure, address the portion of these trends associated with silicon debug during in-situ system validation.

When errors are detected, the silicon prototypes must be debugged for functional errors or diagnosed for physical defects. Increasing device complexity means efficient silicon debug is an essential step of the product development process**[3]**. Figure 1 shows the current average amount of time required for these two stages.

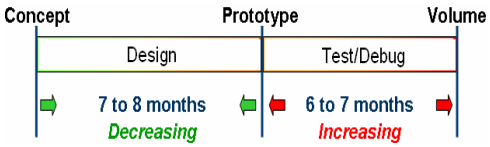

Figure 1

Silicon debug is difficult primarily due to the lack of internal signal value visibility. Key factors which make silicon debug difficult include:

- Limited silicon signal data visibility prevents understanding of internal behavior
- Silicon errors may either be functional bugs or physical defects – creating unrepeatable errors
- Silicon debug data, such as signal values, is usually associated with a gate-level netlist and not the RTL with which the designer is familiar
- Current environments for silicon debug are different from the design and verification environment (file formats, tools, etc.)

2. NEED FOR SILICON TEST AND DEBUG
Development of ICs begins with creation of the specification, spans a series of implementation and verification procedures, and then culminates with the manufacture of the first silicon prototypes. Throughout this process, engineering teams utilize a multitude of verification tools to verify the design for functionality, electrical and physical layout requirements. Because tools are bounded by capacity and performance, engineering teams must often balance accuracy and thoroughness against the availability of time and resource in pursuit of verification completion and tape-out. As a result prototypes must be checked for errors that escaped the verification process[4]. Common sources of errors are listed below:

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

- *Specification*: The design is not adequately specified or incorrect. Increases with rise in design complexity.
- *Coding*: The RTL code is incorrectly written or does not correctly specify functionality. Increases when verification methods and verification tool performance does not keep up with the growth in the amount of RTL-per-design.
- *Synthesis*: The output of synthesis does not match the intended functionality described by the RTL. Related to the quality of synthesis products.
- *Layout*: The layout does not meet all of the physical specification requirements. Increases with the lack of sufficient modeling and DRC tool quality.
- *Electrical:* Circuit values altered due to dynamic power consumption, crosstalk effects, and RF effects. Increases with smaller geometries and model inaccuracies.
- *Manufacturing*: Physical defects introduced during the manufacture process. Increases with smaller geometries and compounded by resulting difficulty in defect model accuracy.

3. SILICON DEBUG AND DIAGNOSIS ACTIVITY

Upon arrival of the first silicon prototypes and before the device is manufactured in volume, three primary activities occur: verification of the manufacturing process through the application of test patterns, validation of the prototype and its parent system, and analysis of several die to review yield suitability for manufacturing in high volume (Figure 2). Ideally, these activities occur sequentially to minimize debug efforts, although many engineering teams perform these activities concurrently. The amount of time required for each of these activities varies according to several factors including device complexity, device size, system complexity, process geometry, amount of resources, and the actual number of errors in the prototype. An additional factor commonly overlooked is the debug methodology and the amount of corresponding on-chip logic useable for silicon debug.

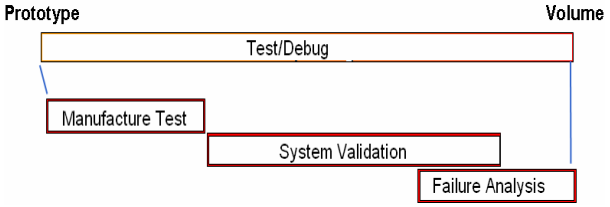

Figure 2

As designs move toward smaller geometries, the chance of errors increases and makes first-time silicon success less likely. Errors must be exposed and debugged during the prototype stage in devices and systems that become more complex with each generation.

3.1 Manufacturing Test

Manufacturing test based on scan-based design-for-test (DFT) and automatic-test-pattern-generation (ATPG) is a common methodology utilized in approximately 82% of designs[5]. With scan, ATEs can efficiently apply ATPG-based test programs to the silicon to check if the manufacturing process introduced physical defects. However, this check is only as thorough as the test program itself.

Since most test programs are generated by ATPG tools and target the simple single-stuck-at fault (SSF) model, the quality of the test program is related to amount of correlation between the SSF models and the actual defects. This model assumes that defects will create behavior similar to a single pin tied to a logical one or zero throughout the application of test pattern stimulus and regardless of its order. Recent research at Stanford University shows that only five percent of the defects in devices based on 70 nanometer geometries behavior as an SSF[6]. This defect behavior trend implies that many defective silicon prototypes will pass the initial manufacturing tests and be placed in situ for system validation.

3.2 System Validation

Whereas manufacturing testing relies on widely adopted DFT methodologies, system validation has no equivalently adopted methodology. The steps to achieve system validation are understood: run system-level applications while the device is in situ to ensure it operates according to specification. However, when unexpected silicon behavior arises, many engineering teams resort to ad-hoc debug flows. Ideally, the internal signal values extracted from the silicon device is at some interesting point in time and is examined to understand the device state. This examination is only possible if sufficient design-for-debug (DFD) logic has been implemented onto silicon.

3.2.1 Design-for-Debug Logic

Currently, around five percent of all designs contain some type of DFD[8]. The simplest type of DFD reuses existing DFT such as scan chains. In order to use scan chains to debug the system or validation board, the device must be able to change from mission mode to debug mode and then shift values out of the scan chains primary outputs. With the values of the scan chains available, it is possible to reconstruct the internal states. More sophisticated DFD methodologies have been implemented that add internal monitoring and breakpoints [4][16]. Other techniques instrument additional circuit into a design to extract values to be observed through pins or memory buffers during run real-time observation [17].

3.2.2 Data to Debug Flow

For debugging device and system errors that manifest in situ, DFD is a key enabler. However, DFD focuses on silicon data acquisition and not on the interpretation or analysis of the data. Usually, the data must be converted and imported into existing verification and debug tools. A typical setup for a system validation test setup with DFD is shown in Figure 3.

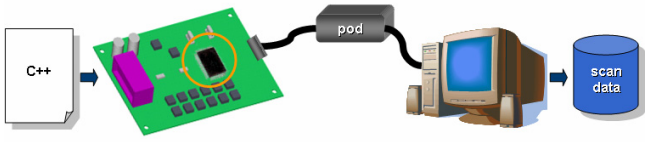

Figure 3

The flow begins with data extraction or capturing internally in the device. The captured data, under the instructions from a software control program, moves out of the device and across the validation board. The data then crosses a physical connector and across a pod to match electrical characteristics of the PC port, such as a USB. The control program then must convert the data into a format acceptable for either textually viewing or another application such as HDL simulation-oriented debug tool. Note that since most of the DFD reuses DFT which is implemented at the structural or gate-level, the signal data coming out is based on gate-level instances.

4. VISIBILITY ENHANCEMENT FOR SILICON DEBUG

As described, current DFD flow provide very limited signal access and visibility into a device. The approaches using trace buffering or debug bus provide flexibility in what to observe in a device, but the number of signals that can be observed is usually very limited. DFD based on scan DFT registers allows observation of states in scan chain, but it is still limited by only the register signals. In most of the cases, the data provided is from the registers, not combinational logic nodes. As a result of restricted test control and response observation, finding and analyzing the problems in silicon is dreadfully tedious and time consuming.

Furthermore, as a design is first mapped to gate level before silicon implementation, the data extracted from device is usually mapped to gate level netlist. While designers normally do not design at the gate level, it becomes very hard to debug a problem when only very limited data is available. To effectively debug such a complicated gate level design, one possible solution is to map the data back to RTL design.

We propose a new technology to resolve the above issues. The methodology includes three main steps: essential signal analysis, on-the-fly data expansion, and gate to RTL correlation.

1. *Essential Signal Analysis*: determine an optimal set of control registers to be planted into the device for best visibility.
2. *Data expansion*: Map the captured silicon data to corresponding signals in HDL design, and expand the non-visible signals from the captured data based on knowledge of design function.
3. *Gate-to-RTL Correlation*: Map the gate level signals back to RTL and allow one to debug the problem at higher level of abstraction.

4.1 Essential Signal Analysis

Essential Signal Analysis technology analyzes assertion, RTL, or netlist HDL code to decide the minimum "essential" signals needed for debug. "Essential" signals are a subset of the whole signal set, from which the value of the whole signal set can be derived by data expansion. ESA starts by inferring the HDL description to derive the logic equations for the design. Then, it recursively traverses inferred logic netlist, analyzes the logic equations for each functional block, and extracts the essential signals into a file to be used by data expansion.

Given a limited resource overhead for DFD instrumentation, ESA determines the set of registers to be instrumented with DFD logic to obtain the maximal visibility of a device. The quality of essential signal analysis is closely related to the capability of data expansion. If the data expansion can only compute the value of combinational logic from the storage elements, then all the registers need to be captured to achieve one hundred percent visibility. If a data expansion can calculate across multiple clock cycles from the captured data, the signals that needed to be captured will be only be a subset of the state registers.

The ESA technology can also be applied in other areas such as assertion, and regression simulation. For debugging assertion failures, ESA analyzes the design and the selected assertions to extract the minimal set of design signals needed for debugging each assertion. By dumping just these signals and evaluating the assertions after simulation, overhead during simulation is reduced, the process is parallelized, and overall turnaround time is improved.

For regression simulation, ESA analyzes the RTL or netlist code to find the storage elements, memory elements, and the primary inputs. Then, instead of dumping every signal in the design, only the essential signals are dumped, and full visibility can be achieved using the data expansion technology. Since the number of essential signals in a typical design is significantly smaller than the total number of signals, the simulation run time overhead to dump only the essential signals can be relatively small. The methodology can reduce the total simulation runs for debug from two to one.

4.2 Data Expansion

The DFD logic provides captured signal data from the registers of the silicon device. In some cases, only a limited set of registers are provided. The raw data is simply a stream of 1s and 0s. This data must be properly formatted and processed for use in a debug tool.

4.2.1 Translate Data for the HDL Domain

In order to be useful, the raw signal data captured by DFD logic must be translated to correlate with the name and time in simulation world for HDL-based debugging tools. There are a number of steps to this process:

- Temporal transformation – correlate time or cycle to the captured data from silicon world to virtual world
- Signal mapping – the data must be associated with the signal names in the HDL design.
- Format transformation – the data is made available in standard VCD format or Novas FSDB format

4.2.2 Expand Captured Data

When only a subset of data is available, debugging of a silicon error is very difficult. Data expansion is a technology that computes the missing data, using the knowledge of the design function. For example, with only flop and latch values are available, the combinational logic between the registers can be calculated (Figure 4) through Boolean calculations.

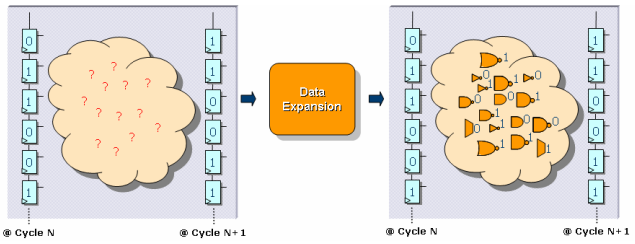

Figure 4

There are two primary differences with this technique compared to simulation. First, since the data is ultimately passed to debug system which is an interactive activity, where only a specific portion of the design is under inspection at any specific time, "on-the-fly" data expansion where only the needed value will be computed is a must for users. Second, because temporal information is already embedded in the data, time-processing is not necessary for functional debug. These two differences result in much higher performance compared to commercial HDL simulators.

4.3 Correlate Gate-Level Signals to RTL

After the data has been captured, translated, and expanded, users can apply standard debug capabilities to isolate the root cause of problems, all while operating on the gate-level design description. Synthesized gate-level designs are typically difficult to understand because synthesis tools perform transformations that do not necessarily provide a one-to-one mapping to RTL code. To understand the meaning of gate-level logic, it is important to have the ability to correlate back to RTL design.

In general, not every net of a gate design will have a corresponding net in the RTL circuit due to optimization. However, it would be useful to localize a region related to one net. It is also reasonable to assume that registers do have one-to-one correspondence in typical applications. It is hence natural to try mapping the registers first, and using this information to localize net correspondence of the combinational signals.

In the following, a framework to solve this problem in the context of structural dependency graph and approximate graph matching is proposed. In this context, one could solve the latch matching problem quite easily. Internal net matching via localization could also be done under this context after the registers and essential signals are matched, albeit with more uncertainty due to the optimization introduced in synthesis. The idea of mapping using structural dependency graph imitates the process of human debugging process where one often locate the corresponding areas by looking at registers in the fan-in and fan-out cones first, and project the correlating statement from these boundary registers.

4.3.1 Structural dependency graph

The structural dependency graph is a directed graph representation of a circuit. It should always represent circuits at the same fine-grained level, i.e. gate level, for easy comparison. The nodes are registers and essential signals that are to be matched. The directed edges represent signal dependency. The structural dependency graph could hence be constructed at internal control/data flow graph level. One could also attach weights on the edges to represent more structural information, e.g. logic level. The concept of weight could be quite essential for RTL circuit description, in which asymmetric function units are often rendered symmetrically with edges only. These assumptions greatly simplify the circuit. The construction of structural dependency graph for RTL circuit has some degree of freedom that should be tuned according to their performance.

4.3.2 Approximate graph matching

After building the structural dependency graphs of the two circuits under debugging, one could match these two graphs in the context of approximate graph matching [12] which is in general an NP problem. However, we believe it is more manageable in the current context due to a number of reasons:

1. Mapping on inputs and outputs are known. They provide good starting points for successive mapping.
2. This algorithm could be facilitated with user specified mapping. However, one should also provide a mechanism in which the user could easily change an incorrectly specified mapping correspondence and see the corresponding updated mapping. This means the mapping algorithm should be robust enough for online update.
3. High performance designs have shallow logic, and the number of registers within the fan-in cone of a register is in general small. The resulting graph mapping problem is less ambiguous and easier.
4. A suitable translation of the RTL circuit will produce similar structural dependency graphs for mapping. Structural dependencies at the RTL-level description disappear in the gate-level description only if it is optimized away due to logical independence. On the other hand, additional gate-level structural dependencies are due to resource sharing. We believe these transformations do not change the global structural dependency graph significantly.
5. Heuristics exist that could solve the approximate graph mapping in a reasonable time [12]. However, we believe more refined heuristics are needed for our purpose.

5. CONNECTION TO DEBUG TOOLS

After the data is process with data expansion and correlation, full visibility of signal values at the RTL is achieved. At this point, the data can be passed to a debug tool. Debug systems such as Novas Verdi provide viewing of the signal waveforms, values annotated on RTL source code, and values annotated on schematic views. The environment that normally provides debug of the virtual design as it operates in software simulation can now provide debug the actual silicon device in the same manner.

6. EXPERIMENTAL RESULTS

We conducted some experiments on the effectiveness of the gate to RTL correlation technology. We took an RTL design, and synthesized it into gates both with and without preserving hierarchy. Table 1 shows the results for the one with preserving hierarchy. In this design, there are 990 signals in the gate level design. Among these 990 signals, there are 207 signals with corresponding signals in the RTL design, albeit with minor naming changes. The majority of these 207 signals are registers. For the rest of signals, there is no direct correspondence to the

RTL design. We applied our algorithm to decide where the signals might be generated. We found 684 signals in the gate-level design that map to a single statement in RTL, 54 signals come from two statements, and 21 signals from three statements. There are 24 signals map to more than three statements.

Table 1

	Count	Percentage
Total	990	
Can be mapped	207	21%
1 statement	684	69%
2-3 statements	54	5.4%
More than 3 statements	21	2.1%
Other (floating, scan chain …)	24	2.5%

In the second experiment, we performed the gate-to-RTL correlation algorithm on a flatten gate level design. Because the search scope is larger in a flatten design, the algorithm generally maps to more statements for the internal nets as shown in Table 2.

Table 2

	Count	Percentage
Total	952	
Can be mapped	180	19%
1 statement	307	32%
2-3 statements	194	20%
More than 3 statements	213	22%
Other (floating, scan chain …)	58	6%

In the third experiment, we ran some scenarios to examine the effectiveness of essential signal analysis technology. We selected several real designs and analyze the essential signals of the designs. Then, we applied data expansion technology to calculate the missing value of the design. The results are shown in Table 3. For the first small design, we found about 9.5% of the signals are essential. The second and third experiments show the percentage of essential signals could span between 5 to 20% of the total signals. For all these cases, full visibility can be achieved by data expansion.

Table 3

Working scope	# of signals	# of essential signals	% of signals
Case 1	138842	13190	9.5%
Case 2	6808801	992248	14.6%
Case 3	4568074	173544	3.8%

In the fourth experiment, we show the result of the ESA technology used in regression simulation. In both cases, it shows with very little overhead on simulation time, we are able to obtain a simulation dump file for full visibility debug. Furthermore, it shows the size of the dump file is significantly smaller than that by full dump simulation.

Table 4

	CASE 1		CASE 2	
	Simulation time	**FSDB file size**	**Simulation time**	**FSDB file size**
Full dump	506 sec.	290 MB	6 hr. 39 min.	2.1 GB
No dump	132 sec.	N/A	2 hr. 12 min.	N/A
ES dump	174 sec.	25 MB	2 hr. 29 min.	61.7 MB

7. CONCLUSIONS

Debugging silicon prototypes is very difficult, but important nonetheless. Emerging DFD methodologies address the design aspect for debug, but are significantly improved when combined with visibility enhancement technologies. These technologies provide the ability to analyze the control registers needed for DFD, expand the captured data to determine the combinational node values, and correlation to present gate-level data at the RTL. The result of combining DFD and visibility enhancement technologies is a significant amount of silicon data available for viewing debugging designs similarly to the methods utilized in the software simulation verification stage. As a result, understanding in-situ silicon prototype operation is greatly enhanced, resulting shorter times to locate and isolate problems that occur during silicon validation.

Our experiments show the proposed technology is effective and should greatly enhance the comprehension of silicon prototype operation in order to locate and isolate problems that occur during silicon test.

8. ACKNOWLEDGMENTS

Our thanks to the Novas Colleagues for their work towards the development of the system.

9. REFERENCES

[1] Smith, Gary. “ASIC Design Times Spiral Out of Control: User Wants and Needs”, Gartner, Inc., January 29, 2002. p.4-12

[2] Smith, Gary; Tan, Sharon. “Conservative Times, Conservative Designs in EDA: User Wants and Needs”, Gartner, Inc., February 10, 2004. p.6-17

[3] H. Hao and R. Avra, "Structured design-for-debug – the SuperSPARC™-II methodology and implementation", in *Proceedings International Test Conference*, 1995, pp. 175-183.

[4] Vermeulen, Bart; Oostdijk, Steven; Bouwman, Frank, "Test and Debug Strategy of the PNX8525 Nexperia™ Digital Video Platform System Chip", in *Proceedings International Test Conference*, 2001, pp. 121-130.

[5] Goering, Richard. “Scan design called portal for hackers”. EE Times. October 25, 2004

[6] McCluskey, E.J., et al. “ELF-Murphy Data on Defects and Test Sets”, in *VLSI Test Symposium, 2004.*

[7] “International Technology Roadmap for Semiconductors” 2003, p. 15.

[8] Wilson, Ron. “Silicon Debug Tools Lengthen Their Reach”. EE Times. July 17, 2003.

[9] Bertacco, Valeria, Maurizio Damiani, Stefano Quer: “Cycle-based Symbolic Simulation of Gate-level Synchronous Circuits”, *Proceeding of the 36th Design Automation Conference,* 1999, p. 392-396.

[10] Bryant, Randal E. “Graphical-Based Algorithms for Boolean Function Manipulation”, *IEEE Transactions On Computers*, 35(8):677-691, 1986.

[11] Ganai, Malay; Aziz, Adnan; Kuehlmann Andreas, “Enhancing Simulation with BDDs and ATPG”, in *Proceeding of the 36th Design Automation Conference*, 1999, p. 385-390.

[12] Hsu, Yu-Chin; Tabbara, Bassam; Chen, Yirng-An; Tsai Furshing, “Advanced Techniques for RTL Debugging”, in *Proceeding of the 40th Design Automation Conference*, 2003, p. 362-367.

[13] J. R. Burch, and V. Singhal, Robust Latch Mapping for Combinational Equivalence Checking, in *Proceeding of International Conference on Computer Aided Design*, 1998, pp 563-569.

[14] D. Anastasakis, R. Domiano, H-K Ma, T. Stanion, A Practical and Efficient Method for Compare-Point Matching, in *Proceeding of the 39th Design Automation Conference*, 2002, p. 305-310.

[15] J. Wang, K Zhang, and G. Chirn, Algorithms for approximate graph matching. Information Sciences, 82, 1995, p. 45-74.

[16] Bart Vermeulen, Zalfany Urfianto, and Sandeep Kumar Goel, Automatic Generation of Breakpoint Hardware for Silicon Debug, in *Proceeding of the 41st Design Automation Conference, 2004, p. 514-517.*

[17] M. Abramovici et al. A Reconfigurable Design-for-Debug Infrastructure for SoCs *Proceeding of the 43rd Design Automation Conference, July, 2006.*