Paper Review

Yuting Cao
University of South Florida
Email: cao2@mail.usf.edu

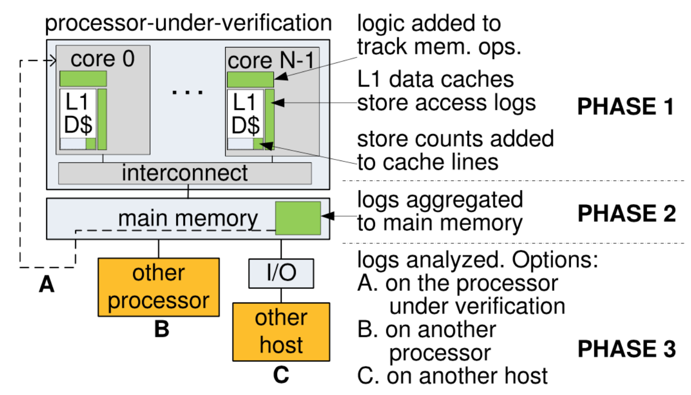

processor-under-verification
core 0
core N-1
L1 D$
L1 D$
interconnect
main memory
other processor
I/O
other host
A
B
C
logic added to track mem. ops.
L1 data caches store access logs
store counts added to cache lines
PHASE 1
logs aggregated to main memory
PHASE 2
logs analyzed. Options:
A. on the processor under verification
B. on another processor
C. on another host
PHASE 3

Fig. 1. System Overview. Each core is modified to track memory accesses. A portion of the L1 caches are temporarily reserved for logging memory accesses. A store counter is attached to each cache line to track the order of writes to the line. Finally, the analysis of the logged data is performed in software, which can be carried out on a processor/host available for the task.

***Abstract*—Briefly summarize all the papers I read about post silicon validation.**

I. Post-silicon validation of mulrtiprocessor memory consisitency [?]

monitor the system memory acccess information by changing a small portion of the l1 cache and an extra samll hardware logic. The logged information will then be transfered to central memory to conduct analysis process. The method used to detect memory error is by constructing a graph representation of memory interactions, an cycle indicates an error.

System overview is shown in Figure. **??**

II. Cant See the Forest for the Trees: State Restoration?s Limitations in Post-silicon Trace Signal Selection [1]

This paper proposed the the signal selection standard SSR(State Restoration Ratio) is not suitable for evaluating trace signal quality. It should be replaced by metric that allows good high-level behavioral coverage.

SRR measures the number of design states reconstructed from the signals observed. SRR= number of signals observed and restored / number of observed signals. The reason why SSR is not optimal is because

- SRR treat all signals equally. but some signals are more important than others
- SRR favors big arrays, which may not be useful for debugging at all

This paper proposed a new metric that is assertion coverage, as how many assertion are satisfied by the observed signals. The drawback it will introduce is the subjectivity and incompleteness. $WHY??$

This paper also introduced a new signal selection algorithm inspired from Google's PaperRank system. This algorithm is based on the connectivity of each instance, and by that find the most important signals. One thing is that it will avoid inclusion of entire array but select only relevant signals in stead. With experiment done with different test bench, it's proven that Paperbank algorithm allows much higher assertion coverage than SSR algorithm's selected signals.

One drawback of the Paperband algorithm is that it can't cover system-level assertions, because they don't trace interface signals. The phrase here is very odd, n̈on of the algorithms are able to cover system level assertions, since they do not trace interface signals. This illustrates that optimizing for a functional coverage metric like assertion coverage will lead to interface signals being emphasized over internal signals. V̈ERY CONFUSING.!!!!

CONCLUSION: This paper compared algorithm of Paperbank on netlist, PaperBank on RTL, SigSeT(SRR maximizing algorithm) with different metric including SRR, behavioral coverage, assertion coverage. The result shows that signal set with good SRR usually have less behavioral coverage and assertion coverage. The PaperBank on RTL model turns out to be the best algorithm for better debugging.

I don't think this algorithm is fit for my research since there are not much attention on communication signals.

III. SYSTEM-LEVEL TRACE SIGNAL SELECTION FOR POST-SILICON DEBUG USING LINEAR PROGRAMMING

This paper propose to automate trace signal selection instead of manual selection due to the increasing complexity of the system. It use the functional coverage as a metric and choose the low-level signals that produce a good coverage. And gradually refine the solution to allow the best behavioral coverage.

This paper focus on the communications between IPs and try to maximize the coverage of each protocol messages.

The algorithm proposed in this paper is mainly protocol based. As showed in 1, this algorithm

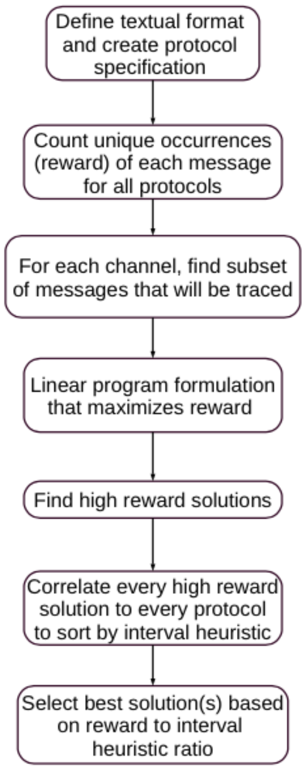
- will first define a family of protocol in an understandable format. then decompose all messages in protocol into a messages contain only one data filed.
- For each Channel, form a set of messages that it covers

- Using a linear program to find a set of signals to trace that allows the maximum frequency coverage. see Figure. 2
- For high reward solutions, calculate heuristic intervals. See Figure. 3
- Select best solution with small I/1 $+ FC$ where I is interval score and FC is frequency coverage.

When find bugs and we want to refine the select signals to enable better root cause finding. There are several ways we can use.

- Block-Specific Views. For possible root cause blocks, assign messages in protocol family into each block and apply coverage-interval algorithm to them
- Control View
  As referred in Fig. 4, we define all protocols and then find local and external decision points. Local decision points refers the messages with conditions. External decision point is messages that have same sender, data fields but different receiver. Calculating the advantage of the external decision point is NOT VERY CLEAR FOR ME YET.

Fig. 2. Step-by-step flow of global view and block view selection method


Define textual format and create protocol specification
Count unique occurrences (reward) of each message for all protocols
For each channel, find subset of messages that will be traced
Linear program formulation that maximizes reward
Find high reward solutions
Correlate every high reward solution to every protocol to sort by interval heuristic
Select best solution(s) based on reward to interval heuristic ratio

IV. On the cusp of a validation wall [2]

This paper goes through the definition of silicon validation and reason why it's important, together with the current techniques used for pre and post silicon validation.

Fig. 3. Linear Program Formulation

$$\text{maximize}: \sum_{\forall m_i \in M} r_i y_i \text{ (maximize frequency coverage)}$$

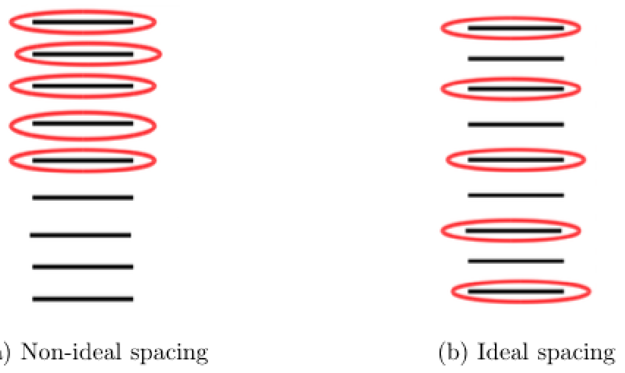
$$\text{subject to}: \sum_{\forall q_i \in Q} c(q_i) x_i \leq C \text{ (cost constraint)}$$

$$\sum_{\forall q_i \in Q} x_i \leq y_j \quad \forall m_j \in M$$

$$0 \leq y_i \leq 1$$

$$x_i \in \{0, 1\}$$

Fig. 4. Spacing example for a nine message sequence with four messages covered. Red circles indicate message is observable



Validation is the activity of ensuring a product satisfies its reference specifications, runs with relevant software and hardware, and meets user expectations.

Numbers of processor bug are growing for every new generation and the bugs are becoming more diverse and complex. which makes silicon testing even harder.

Effective validation needs

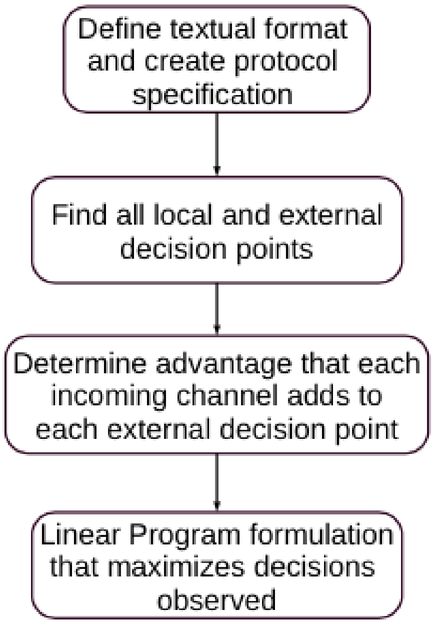
- modular validation (virtual platform) :
  virtual platform helps find problems before post silicon validation, forcing the combination of pre and post silicon validation. And also enables early software development on the virtual platform.
  Formal verification vs Dynamic verification
- good analog and simulation.
- test generator
  To ensure all cases covered by test generator in stead of single area.
- coverage measurement
- assisted insertion of test
- debugging features.

What I learned: Post silicon validation is very complex and effort intensive compared to pre-silicon validation. Not very automated, which is needed in scholar field.

V. Validation of SoC Firmware Hardware Flows: Chanllenges and Solution Directions [3]

- Challenges of SoC firmware hardware flows

Fig. 5. Step-by-step flow of control view selection method

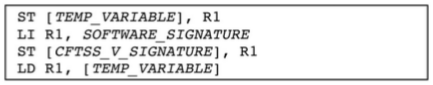

Define textual format and create protocol specification
Find all local and external decision points
Determine advantage that each incoming channel adds to each external decision point
Linear Program formulation that maximizes decisions observed

- specification error found late in product life-cycle. goal: analyze architecture specificaion methods: 1. executable specification written in programing language or formalism. (SystemC) question?paper said it?s not good because ?effort to develop executable specification is too high for adoption by the architecture team. " 2. formal-specification language. (Proemela)
- stand alone FW or HW validation is problematic goal: co-validation. methods: Do FW validation in the early stage by using a virtual platform of the HW as a development environment for FW. Start FW development simultaneously with HW development, to start the validation effort early.
- distribution of flows across many IPs and subsystem. Each flow has its own functionality. goal: need to separate concerns, allow better modularity in validation. methods: Design for verification (A little confused)

VI. Effective Post-Silicon Validation of System-on-Chips Using Quick Error Detection [4]

reading: ref 9 about priori and 46 for algorithm about software signature and flow checking This paper introduced a method called Quick Error Detection (QED) for systematically creating test cases that can quickly detect bugs inside components. The advantage of QED is it has a much shorter error detection latencies and increase the bug coverage. The Fig. 7 shows the bug activation criteria and bug effects tables. Bug activation criteria is a set of conditions that mush be satisfied to activate a bug scenario. Bug effects is the resulting wrong behavior. Variable X, Y can be defined as any numbers with bound predefined.

Fig. 6. Pseudo assembly code for the CFTSS-V operation inserted at the beginning of each ?block of instructions.?


```
ST [TEMP_VARIABLE], R1
LI R1, SOFTWARE_SIGNATURE
ST [CFTSS_V_SIGNATURE], R1
LD R1, [TEMP_VARIABLE]
```

QED systematically transforms original tests into new QED tests families. New tests will have bounded error detection latency.

Methods: transformations:

- Error Detection Using Duplicated Instruction for Validation (EDDI-V)
  Bound error detection latency for bugs inside processor cores. Duplicate instructions and registers, and with same instruction, check if the corresponding value are same. If not, error find.
- PLC (for uncore components)
  Insert special PLC operations at very fine granularities across memory (and I/O) spaces using targeted instructions. It does self checking on selected set of variables
  .
  Compatible with EDDI-V
- CFCSS-V and CFTSS-V (for core's control flow)
  - CFCSS-V
    Checks the control flow of a test program during runtime against the control flow graph constructed during the test program?s compile time. Instead of checking between basic blocks, check between "blocks of instructions" that may contain arbitrary number of instructions (determined by $Inst\_min$ and $Inst\_max$) to balance trade offs between error detection latency and intrusiveness. Checking flow between every instruction will result in short error detection latency but increase the intrusiveness. While multiple blocks will lead to longer error detection latency.
    Basic Idea: each block of instructions is assigned a unique integer as software signature. Detail need to be read from another paper.
  - CFTSS-V see fig. 5 confused
- Comparison of QED transformations in fig. 6

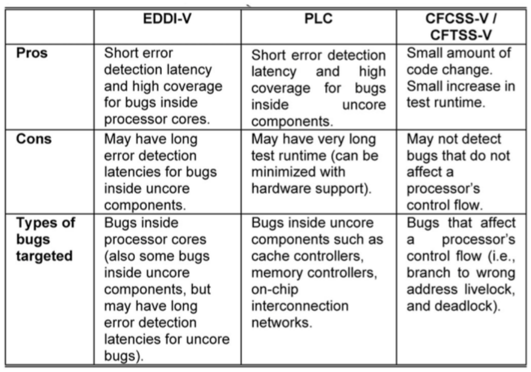
VII. A Common Platform for bridging Pre- and Post-Silicon Verification in Mixed-Signal Designed /citecommonplatform

Motivation: majority of current verification is for digital or analog designs only. Great demand for new test technology to verify both digital and analog together. Physical bus increase due to the rising complexity of the interactions between analog and digital modules and smaller feature sizes.

Advantagement: allows simulation test case reusing for post-silicon verification. Development of a bidirectional bridging approach. It's generic, can be applied to a lot areas.

Concepts:

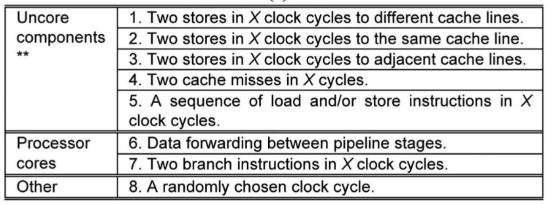
Fig. 7. COMPARISON OF QED TRANSFORMATIONS



	EDDI-V	PLC	CFCSS-V / CFTSS-V
Pros	Short error detection latency and high coverage for bugs inside processor cores.	Short error detection latency and high coverage for bugs inside uncore components.	Small amount of code change. Small increase in test runtime.
Cons	May have long error detection latencies for bugs inside uncore components.	May have very long test runtime (can be minimized with hardware support).	May not detect bugs that do not affect a processor's control flow.
Types of bugs targeted	Bugs inside processor cores (also some bugs inside uncore components, but may have long error detection latencies for uncore bugs).	Bugs inside uncore components such as cache controllers, memory controllers, on-chip interconnection networks.	Bugs that affect a processor's control flow (i.e., branch to wrong address livelock, and deadlock).

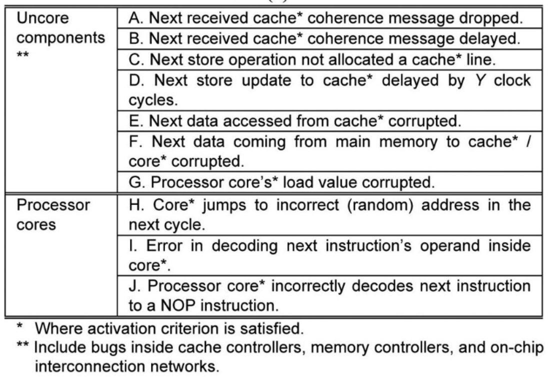
Fig. 8. bug activation criteria and bug effects
(a) BUG ACTIVATION CRITERIA. (b) BUG EFFECTS

(a)



Uncore components **	1. Two stores in *X* clock cycles to different cache lines.
	2. Two stores in *X* clock cycles to the same cache line.
	3. Two stores in *X* clock cycles to adjacent cache lines.
	4. Two cache misses in *X* cycles.
	5. A sequence of load and/or store instructions in *X* clock cycles.
Processor cores	6. Data forwarding between pipeline stages.
	7. Two branch instructions in *X* clock cycles.
Other	8. A randomly chosen clock cycle.

(b)



Uncore components **	A. Next received cache* coherence message dropped.
	B. Next received cache* coherence message delayed.
	C. Next store operation not allocated a cache* line.
	D. Next store update to cache* delayed by *Y* clock cycles.
	E. Next data accessed from cache* corrupted.
	F. Next data coming from main memory to cache* / core* corrupted.
	G. Processor core's* load value corrupted.
Processor cores	H. Core* jumps to incorrect (random) address in the next cycle.
	I. Error in decoding next instruction's operand inside core*.
	J. Processor core* incorrectly decodes next instruction to a NOP instruction.

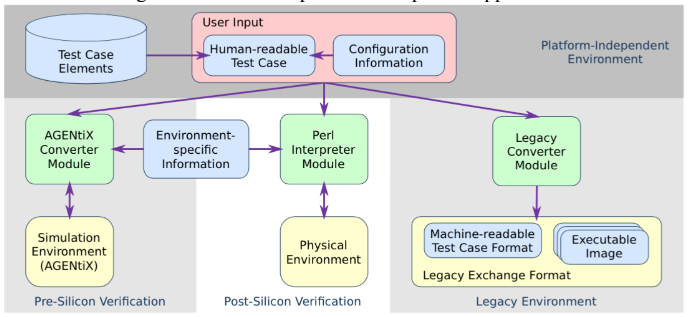
* Where activation criterion is satisfied.
** Include bugs inside cache controllers, memory controllers, and on-chip interconnection networks.

- Exerciser runs on the DUV to create a simulator-like environment for post-si verification. Exerciser can generate test cases for pre-defined scenarios, run on target and auto checking.
- directed tests: predefined, written by hand
- random tests: generated by transaction generators
- direct random tests: hybrid

Method detaisl:

- veication activities need to be planned ahead. (determine objectivies, how to achieve them. Done seperately for pre and post sili, may cause problems)
- as showed in fig. /refappr
- Test cases will be written in scripting language (pearl or python) to allow a high level abstractionand option to store and process data. Also allows for all 3 types of test cases generation
-

Fig. 9. Basic Concept of the Proposed Approach



VIII. Debugging Multi-Core System-on-Chip

- Introduction Design of an SOC: slowly decrease the abstraction level by level, adding details iteratively until it's ready for fabrication. Each level the system will be verified using verification technique.
- Why debug is difficult
  1) Limited internal observability: data volume too big, can't be all recorded
  2) Asynchronicity and Consistent Global State: Different clock frequency between IPs globally-asynchronous locally synchronous design style: valid-accept handshake global state sampled at greatest common divisor of frequencies
  3) Non-Determinism and Multiple Traces non determinism of shared slave create faulty traces
- Debugging an SoC 3 types of errors:
  1) Within a trace: permanent/transient error. (cause error for all following states)
  2) Between traces: constant when occurs in every run (deterministic) / intermittent(non-deterministic)
  3) Between systems: intrusive (changes the behavior of the system -probe effect)
  4) Debugging process: graph 5.9 b
- Debug Methods Properties:
  1) structural abstraction: which part of system to observe within one abstraction level, at what granularity
  2) temporal abstraction: what and how often we observe
  3) behavioral abstraction: what logical function is executed by a hw module
  4) data abstraction: how we interpret data scope is combination of structural and temporal abstraction

  Existing debug methods:
  1) physical and optical debug: non-intrusive, lowest level of abstraction can only access wires close to the surface due to metal layers
  2) logical debug: built in support (design for debug DfD) to increase internal observability and controllability tradeoff b/w real-time behavior ¡-¿ amount of state inspected examples that address problems

caused by asynchronicity, inconsistency of global states , non-determinism or multiple traces

    a) Latch Divergence Analysis: running a cpu many times, compare right traces and wrong traces easily automated. doesn't distinguish noise in substate due to intermittent errors
    b) Deterministic Replay record non deterministic order and force deterministic A lot of specific method that I don't understand now
    c) Use of Abstraction for Debug distinguish b/w inter-process communication and intra-process computation. Allows filtering only useful info

3) future research need debug method for parallel software and hw

References

[1] S. Ma, D. Pal, R. Jiang, S. Ray, and S. Vasudevan, "Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2840819.2840820

[2] P. Patra, "On the cusp of a validation wall," *IEEE Design Test of Computers*, vol. 24, no. 2, pp. 193–196, March 2007.

[3] Y. Abarbanel, E. Singerman, and M. Y. Vardi, "Validation of soc firmware-hardware flows: Challenges and solution directions," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 2:1–2:4. [Online]. Available: http://doi.acm.org/10.1145/2593069.2596692

[4] D. Lin, T. Hong, Y. Li, S. Eswaran, S. Kumar, F. Fallah, N. Hakim, D. S. Gardner, and S. Mitra, "Effective post-silicon validation of system-on-chips using quick error detection," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 10, pp. 1573–1590, 2014.