Paper Review

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

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

I. 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, ïon 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.

II. 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. **??**

III. 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.

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

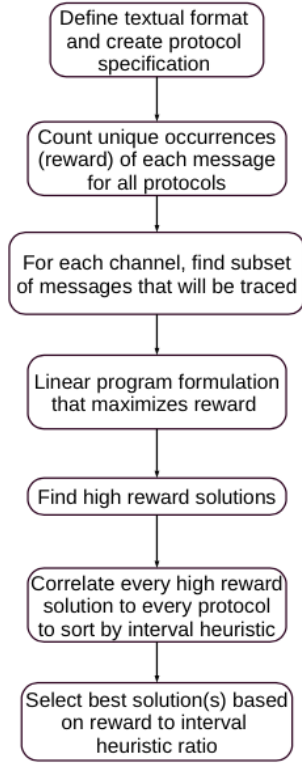Fig. 1. Step-by-step flow of global view and block view selection method

Fig. 3. Spacing example for a nine message sequence with four messages covered. Red circles indicate message is observable
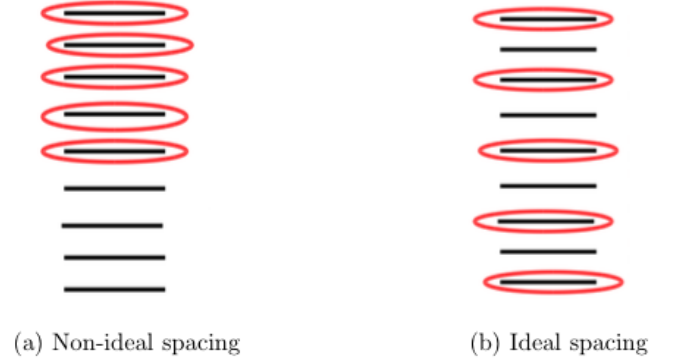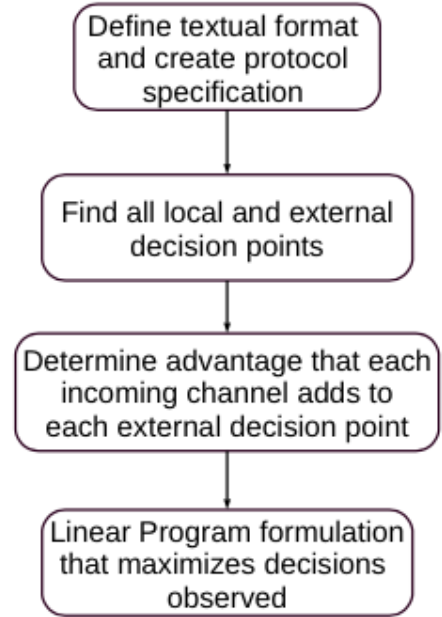
(a) Non-ideal spacing   (b) Ideal spacing

Fig. 4. Step-by-step flow of control view selection method

Fig. 2. Linear Program Formulation

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

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

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

$$0 \leq y_i \leq 1$$

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

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.

IV. VALIDATION OF SoC FIRMWARE HARDWARE FLOWS: CHANLLENGES AND SOLUTION DIRECTIONS [3]

- Challenges of SoC firmware hardware flows
- 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)

V. 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