Post-silicon Timing Diagnosis Made Simple using Formal Technology

Daher Kaiss and Jonathan Kalechstain
Core CAD Technologies, Intel Corporation
Email: {dkaiss, jkalechs}@iil.intel.com

***Abstract*— With the increasing demand for microprocessor core operating frequencies, debugging post silicon synchronization (or speed) failures is a critical time consuming post silicon debug activity. Inability to complete the isolation of all possible speed failures on time, forces companies to go to market with products that run at a lower frequency than their upper frequency limits. This might cause revenue losses or lead to loss of market segment shares. Laser-Assisted Device Alternation (LADA) machines are the main vehicle for debugging post silicon speed failures at Intel. Operating such expensive machines consumes a substantial portion of the overall post silicon debug effort. Moreover, with the increasing complexity of manufacturing processes, these machines need to be renewed from one process generation to the next, which increases the product cost. This paper describes a novel method, based on formal technology, which brings a productivity breakthrough in isolating post-silicon speed failures. We demonstrate that in many cases optical probing using LADA can be fully replaced by our approach.**

I. Introduction

Due to the increasing design size and complexity of modern VLSI design and the decreasing time-to-market, design bugs are more likely to escape the pre-silicon verification and are only found after a chip has been manufactured. Therefore the efficiency of post-silicon debugging is becoming more critical to improve the productivity. With the rising demand for microprocessor core operating frequencies, challenges with on-die synchronization increase accordingly. Such synchronization challenges limit the upper frequency bound of a complex integrated circuit, and thus isolating and fixing performance-limiting circuits continues to consume a significant portion of the post-silicon validation bandwidth [1]. A *speedpath* is a frequency-limiting critical path which affects the performance of a chip [2], [3]. A speedpath that violates timing constraints at the post-silicon stage is called *failing speedpath*[4].

While pre-silicon static timing analysis [5], [6], [7] plays a vital role in facilitating fast and reasonably accurate measurement of circuit timing, post-silicon speed failures appear due to the use of simplified delay models, and due to the limited ability of such static timing analysis tools to consider the effects of logical interactions between signals. Such limitations are the reasons for the *mis-correlation* between the pre-silicon timing models and the post-silicon real behavior.

The process of debugging a speed failure on a multi-billion transistor microprocessor is a challenging, yet well structured process. It starts by applying test vectors to the microprocessor or by running a test program, such as end-user applications or functional tests, on the microprocessor until an error is detected. Such a process is applied on dedicated machines called *testers*. Post-silicon speed failures are normally observed when similar microprocessors produce different results on a tester at different frequencies. Post-silicon debugging is carried out to localize and rectify the root cause of the erroneous behavior. The fix of the failure is normally done by modifying the circuit either by replacing a cell/gate with a faster/slower one, or by performing a simple design retiming operation.

To assist the debugging process, design-for-test (DFT) features such as *scan* [8], [9] are added to the microprocessor. Such features increase the observability of the functional behavior of internal gates in the microprocessor. If the test fails, the values of the DFT scan gates are saved for debug purposes. For historical reasons, and due to cost reduction considerations, Intel didn't adopt the full-scan methodology. Instead, another technique which was developed to increase the debuggability of speed failures is based on on-die clock shrinking [10], [11] which helps narrow the list of failing source candidates. Such a technique is based on driving the microprocessor into clock regions (or domains) where global, regional and local clock distributions are introduced. In this way, post-silicon timing debugging can be improved by *controlling* the clock behavior of each of the clock domains in order to localize the source of the speed failure. The *tester* can be configured to operate the microprocessor at different clock frequencies for each of the above timing domains, and thus bind the source of the timing failure into smaller regions.

However, due to the large number of gates dominated by each clock domain (can reach *thousands* of sequential/storage signals per clock domain), there is still a need to narrow the list of failing source candidates into a smaller group of logic gates in an efficient and reliable manner. Such a technique, which is widely used at Intel today for debugging post-silicon speed failures, is a laser-based analytical technique, referred to as Laser Assisted Device Alteration - LADA [12]. LADA provides the ability to rapidly isolate failing speedpaths and their limiting components, down to the individual logic gate level with high confidence.

The LADA technique uses a laser incident from the device backside, to generate localized photocurrent within the active regions, temporarily altering transistor characteristics. Due to the different effect on PMOS versus NMOS devices, LADA can be used to speed up or slow down devices, so that when applied to devices in critical timing paths, performance

ISBN: 978-0-9835678-4-4.

limiting circuits can be rapidly isolated.

Despite the successes of LADA in isolating post-silicon speed failures, such a process is still time consuming and labor intensive. In addition to the high cost of the LADA machines (>$1M for each machine), they require operators, sometimes with special expertise, to configure the machine to optically probe the regions of interest. Such a debugging process might take hours up to weeks per speed failure, depending on the complexity of the failure. When dealing with hundreds of speed failures to debug per microprocessor project, the debug process can take months, putting the whole project schedule at risk, or alternatively, forcing project management to make compromises on the marketed microprocessor frequency. A previous attempt to address this issue was presented in [13], but it was based on logic simulation and suffered from accuracy limitations and thus resulted in a large number of false paths. Another SAT-based attempt to address debugging post-silicon failing speedpaths was presented at [14], however the authors use a model which is based on using copies of a gate to represent the value of a gate at different points in time. In the worst case, the size of the model may be exponentially larger than the original circuit.

This paper suggests a novel SAT-based method to dramatically reduce the effort of the LADA based speed debug, saving the cost of the machines, reducing resources for operating them, and reducing the time-to-market (TTM) for launching new products. We will show that our tool can potentially replace the LADA based debug process. The superiority of our approach is in its ability to model speed failures without the need to have a timing model as we use a zero delay model. In addition, we use efficient modeling which avoids the possibility of exponential model size. As it will be shown, such efficient modeling is translated into better performance with the ability to deal with large instances taken from the latest Intel microprocessor designs.

The rest of this paper is organized as follows. In the next section, we present the notion of *functional failing speedpath* and present useful characteristics of it. Section III presents a framework for a precise, yet flexible, representation of the circuit network. Section IV describes the way we isolate failing speedpaths. Section V shows how our algorithm deals with reconverging paths, while section VI describes multiple approaches to dealing with complexity. Experimental results are reported in Section VII. Future work is discussed in section VIII. We conclude in section IX.

II. Characteristics of a failing speedpath

Splitting the design into multiple clock domains enables a flexible way to control the *phase* of the clock dominating a set of sequentials. The basic idea of being able to change the relative phases is to give some paths more (or less) time to complete. By doing this, we can trigger or remove timing problems. One way to reduce the region of interest that contains the source of speed failure is to perform a "trial and error" analysis which changes the relative phase of a given clock. This process is performed in a semi-automatic manner

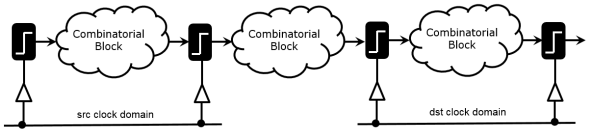

Fig. 1. Clock distribution

by iterating over the clock domains of the microprocessor. If after changing the phase of the clock of a given domain, the speed failure is still reproduced, we can then conclude that the signals responsible for the failure are not controlled by such a clock domain. We keep this process until we find the clock domain which eliminates the failure(s). This process is completed by finding two domains: the *source* (denoted by *src*) and the *destination* (denoted by *dst*). These are sets of sequential signals which bind combinational logic that contains the signal(s) responsible for the speed failure. See Fig. 1.

Each topological path starting from a sequential signal in the *src* domain, and ending at a sequential signal in the *dst* domain is called a *speed path*. A speedpath containing the errornous signal responsible for the speed failure is called a *failing speedpath*. A signal in a circuit M is said to be *toggling* at phase t in a trace π if the value of the signal at phase t is different from its value at phase $t-1$ in π. The failing speedpath originates normally from a *toggling* sequential in the *src* domain, at some phase, and the new value does not propagate properly. We refer to the first sequential in the path as the *root* signal. Notice that as part of the process of identifying the source and destination domains, the phase of the toggling sequential is detected as well. Another important characteristic of the failing speedpath is that it normally originates from *one* toggling root.

As mentioned earlier, the traditional method to isolate the failing speedpath is based on LADA machines. First, all the possible topological paths between signals in the *src* and the *dst* domains are computed. Then, using LADA machines, laser is used to temporarily alter the operating characteristics of transistors on the devices participating such paths. The device being tested is electrically stimulated and the device output is monitored. This technique is applied to the back side of the semiconductor device, thereby allowing direct access of the laser to the device active diffusion regions. The effect of the laser on the active transistor region is to generate a localized photocurrent. This photocurrent is a temporary effect and only occurs during the time that the laser is stimulating the target region. The creation of this photocurrent alters the transistor operating parameters, which may be observed as a change in the function of the device. The effect of this change in parameters may be to speed up or slow down the operation of the device. This makes LADA a suitable technique for determining critical timing paths within a semiconductor circuit [15].

From the perspective of logic behavior, one can consider the failing speedpath failure as a wrong propagation of a toggling

ISBN: 978-0-9835678-4-4.

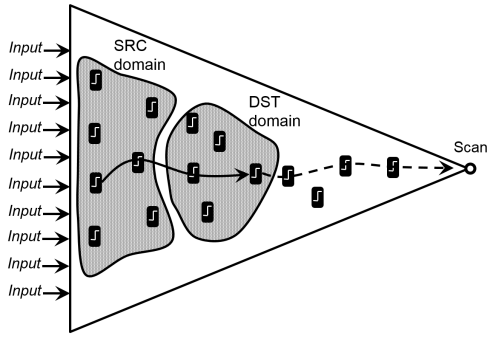

Fig. 2. A failing speedpath from SRC to DST, and its propagation

root sequential in the *src* domain. Instead of propagating a value v, the inverse value $\bar{v}$ propagates through its fanout logic and causes some *observable* (or scan signal) to get an inverted value which causes the test to fail. See the illustration in Fig. 2. The line from a sequential in the *src* domain (root) to the *dst* domain is the reported failing speedpath, while the dashed line is the propagation of its impact till the failing scan.

III. Logic presentation of the microprocessor

A circuit design is modeled at the gate level in terms of combinational signals and sequential signals. Sequential signals can be of two types: (1) a latch, which is a device that transports its input to its output when the clock signal is high (`T`), and holds the output value when the clock signal is low (`F`), and (2) a flip flop, which is a device that transports the previous value of the input when the clock signal rises, and holds the output otherwise.

We consider *ternary* modeling of circuit node values. A value could be one of the *binary* values, `T` or `F`, or an *undefined* value, $\perp$ (elsewhere also denoted by `X`). Given a ternary input vector sequence π, and an initial ternary state s, n_t will denote the value of node n in a circuit M at time t after 3-valued simulation of M with π starting in s.

A circuit M can be represented by a collection of *next-state functions* (NSFs) of the sequentials as well as of the outputs, where a NSF is a function of current and next-state values of inputs and sequentials. For example, consider the circuit M which is illustrated in Fig. 3. It consists of five inputs $a, b, c, clk1$ and $clk2$, one latch l, one flop f, and an output (o) which is the output of the circuit. We denote the current state value of a variable "v" using v and the next state value of the same variable using v'. This way, the next state function of the output o is $l' \vee f'$, while the NSF of the active-high latch l is $(clk1' \wedge a' \wedge b') \vee (\neg clk1' \wedge l)$. The NSF of the rising-edge flop f is $(\neg clk2 \wedge clk2' \wedge c) \vee (\neg(\neg clk2 \wedge clk2') \wedge f)$. Available convenient representations for next state functions can be BDDs [16] or boolean expressions (simple graph data structures for representing propositional logic, where nodes of the graph represent binary operation $\wedge, \vee$, with an annotation whether a variable is negated or not, and variables appear as leafs). We adopted boolean expressions in our work since uniqueness of BDDs is not needed. Modeling of sequential logic is done using a *compact* representation of infinite variable sequences. For a signal v, an infinite sequence of *propositional* variables $\{v_0, v_1, v_2, \cdots\}$ represents symbolically its sequential behavior. This allows one to reduce sequential verification problems to propositional satisfiability. The sequence representations can be unrolled to a desired depth k, producing k propositional variables $\{v_0, v_1, \cdots v_{k-1}\}$, which represent all the possible first k values of the signal v. This representation is suitable not only for modeling sequential behavior of inputs, but also for internal combinational signals, sequential signals, and outputs. For example, for a given output o in Figure 3, the sequential behavior is represented by a disjunction of the sequences representing l and f. Similarly, we can define the behavior of any sequential signal by using NSFs.

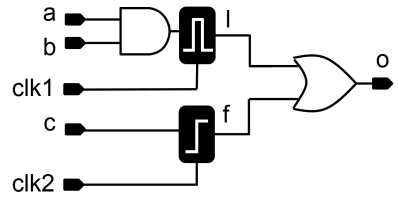

Fig. 3. Example of latch, flop and output functions

IV. Detecting failing speedpaths using formal technology

Our goal is to detect the failing speedpath within the hundreds (sometimes thousands) of speedpaths between the sequential signals in the *src* and *dst* domains, and thus bypass the manual and expensive optical probing stage. We provide the following inputs to the tool:

- Logic representation of the circuit in gate level Register Transfer Level (RTL) format (e.g. Verilog)
- A stimuli file containing the microprocessor simulation trace. This file results from simulating the test program on the RTL presenting the circuit starting from a given initial state. Since such trace might consist of thousands of simulation cycles, our tool extracts only a short window of it (see below)
- The name of the *src* and *dst* clock domains
- The phase when a *src* candidate toggled. It will be denoted by t_{src}.
- The scan signal where the failure was observed
- The phase in which the failure at the scan signal was observed. It will be denoted by t_{scan}.

Definition 4.1: Given a circuit M at a given state s, which is a result of 3-valued simulation of M with a ternary input vector sequence π at time t, a signal l is called to be *sensitive* at time t if flipping the value of l at time t in s causes the value of the failing *scan* signals to flip at a given phase t_{scan}.

A failing speedpath is thus a topological path starting from from some *sensitive* sequential in the *src* domain at time t_{src}. Detecting failing speedpaths starts after detecting the *src* and

dst clock domains. The idea behind our solution is to model the *symptom* of the speedpath failure using logic representation. Our method is based on modeling two machines: The *good* and the *bad* machine. The good machine models the functional behavior of the test as it is reproduced on RTL simulation when it is assumed to exhibit the correct functional behavior. The bad machine models the failure at the tester where the speed failure happens. Finding the failing speedpath is performed by running BMC [17], [18] comparing the sequential behavior of the two machines. The bound k for BMC is defined by the user (default value is set as 50) and it is an upper bound to the maximum length of the sequential depth between the phase of the failing scan and any sequential in the *src* domain.

The good machine is built as follows: the machine has one output which is the failing observed/scan signal. For the rest of this paper, we will refer to the failing observed signal as the failing scan signal. The sequential logic that drives the scan signal is modeled based on the real modeling in the circuit. The inputs, the output and the internal signals (both combinational and sequential signals) are constrained by the concrete values taken from each phase of the given trace window (of length 50). If for some reason, a signal does not have a trace in the given simulation traces, it is modeled as `X`. All signals have dual rail modeling [19], [20], [21], where each signal is modeled by a pair of variables.

Since we are dealing with complete microprocessor simulation, building the logic model of the complete full-chip was normally beyond the capacity limits of our internal logic modeling tools. Normally, the Verilog that models the schematics is very low in its abstraction level, compared to normal RTL which represents the abstract model of the design. In order to overcome such capacity issues, we black-box the irrelevant blocks and keep the ones between the block containing the failing scan, and the block containing the signals in the *src* domain.

The bad machine is built similarly to the good machine, but with the following differences:

- Assuming that the scan signal is failing at phase t_{scan}, and assuming that the trace of the scan signal in the good machine is $[v_0\ v_1\ \cdots\ v_{t_{scan}}\ \cdots\ v_{k-1}]$ (where every v_i is a ternary value), we constrain the behavior of the failing scan signal in the bad machine with $[v_0\ v_1\ \cdots\ \neg v_{t_{scan}}\ \cdots\ v_{k-1}]$. In other words, if the variable of the failing scan at time t_{scan} is annotated by v and the concrete simulation value of the failing scan at the same time is c, then we add a constraint that v is equivalent to $\neg c$. Notice that it is necessary to have a binary trace value for the failing scan signal at the failing phase t_{scan}. Otherwise, the algorithm aborts.
- The set of the *src* candidate sequentials will be denoted by *S*. For each sequential $s \in S$, we add a new XOR gate with two entries: the first is fed by the sequential s and the second is fed by a new free inputs $s\_control$. The logic that was fed originally by the sequential s will be fed now by the new XOR node. Each XOR signal enables modeling the flipping of the value of the sequential s in the sense that if the control signal $s\_control$ has a value of `T`, then the output of the XOR is simply the inverse of the value of s. See Fig. 4 for illustration.

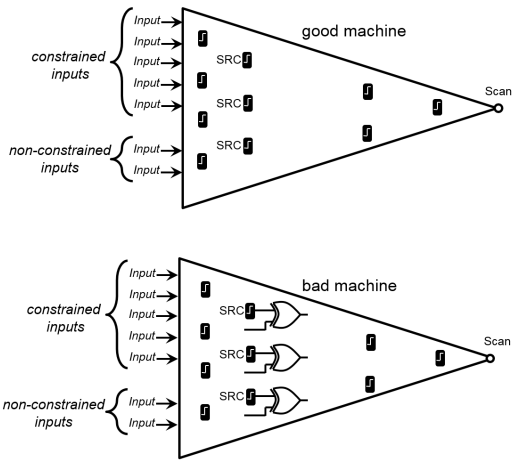

Fig. 4. Good and bad machine modeling

- Assuming that the failing speedpath originates from a sequential which toggles at a given phase t_{src}, we assume that the value of each control input $s\_control$ is a free variable at phase t_{src}, and is constrained to `F` for the rest of the phases:

$$\forall_{s \in S}.\forall_{0 \le t \le k \wedge t \neq t_{src}}.(s\_control_t = F) \qquad (1)$$

- Since we assume that the failing speedpath originates from only *one* source, we add an extra constraint that only one control variable can be `T` (at phase t_{src}).

A SAT-based bounded model checker (BMC) is called to find a satisfying assignment to the above constraints. If a satisfying assignment is found, then we extract the root sequential out of the counter examples by finding the control variable which got a value of `T`. Extracting a complete path is done by backward traversal from the scan signal, by comparing the values of each signal in the good and bad machine.

Definition 4.2: A *failing functional speedpath* is a sequence of pairs of the form $\{(sig_0, ph_0), (sig_1, ph_1), \ldots, (sig_n, ph_n)\}$, where $ph_0 \le ph_1 \le \ldots \le ph_n$, sig_i is a signal name, and ph_i is the *first* phase where the signal sig_i got different values between the good and the bad machines.

Clearly, $t_{src} \le ph_i \le t_{scan}$, and the path starts with a pair (s, t_{src}) for some sequential s in the *src* domain, and ends with a pair $(scan, t_{scan})$. Other signals in the speedpath can be either sequential or combinational ones. For the rest for the paper, the failing functional speedpath will be referred to as the failing speedpath. Generating multiple paths is performed via an iterative process where new constraints are added at each iteration to direct the BMC engine not to reproduce the found path for the next time. We annotate the value of a

signal s in the good/bad machine at time t with s_t^{good} and s_t^{bad} respectively. The constraint which is added to prevent a failing functional speedpath P from being generated again is:

$$\neg \bigwedge_{(s,t)\in \boldsymbol{P}} (s_t^{good} \neq s_t^{bad}) \quad (2)$$

Notice that paths which do not go through any sequential in the *dst* domain are excluded from the tool report. The last iteration happens when BMC does not find any new speedpath.

V. Handling reconverging paths

In this section, we describe the main challenge in achieving this kind of symbiosis between timing analysis and formal technology. Consider the simple AND gate illustrated in Fig. 5. From a logic analysis perspective, regardless whether it is

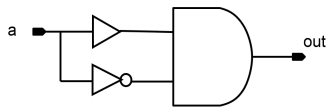

Fig. 5. Differences between timing and logic analysis

logic simulation or formal analysis, the output of the circuit at output *out* is `F`, even if the value of the input a transitions from `F` to `T`. Though, from a timing perspective, a transition from `F` to `T` at the input a might be propagated at different speeds through the buffer and inverter, resulting in two `T`'s at the entries of the AND gate and causing the output *out* to get a value of `T` for a short period of time. We call this phenomenon a *glitch* and it can be one of the reasons for speed failures as the output *out* can be captured with the wrong value. Clearly, the logic representation of the circuit does not capture the speed behavior described in the simple AND illustration, and it definitely limits the tool from being able to isolate real failing speedpaths. Looking into the problem in a more generic view, the problem results when the cone of influence of the scan signal contains internal signals which form a *reconvergence* point of different paths starting from the same root signal. The propagation of the toggling value on the root signals is *masked* by a value of two or more signals feeding the same reconvergence point, which masks further propagation of the value till the failing scan signal.

In this section, we describe a novel technique for dealing with masking values at reconvergence points. The idea is based on performing a naive *topological* analysis on the cone of influence to detect the reconvergence points. For each convergence signal s, with n inputs denoted by I_i where $0 \leq i < n$, we perform the following modeling modifications to the bad machine:

- For each input I_i, we introduce a new MUX gate with two entries. The first entry will capture the signl I_i from the good machine, while the second entry will capture the signals I_i from bad machine. The selector of the MUX gate will be a new free variable $sensitivity\_selector\_I_i$.

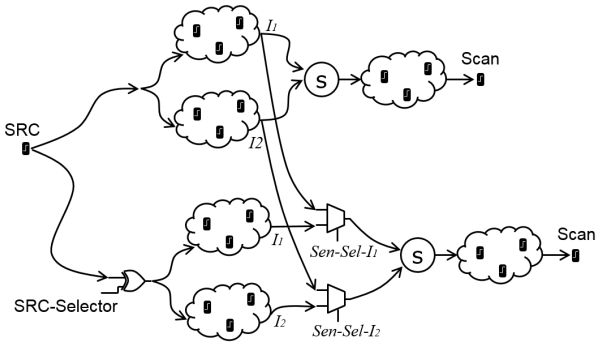

Fig. 6. Handling reconvergence points

- Each signal driven by the signal s will be now driven by the new MUX gate.
- We add a constraint that only one of the sensitivity selector variables $sensitivity\_selector\_I_i$ $(0 \leq i < n)$ can be `T`. To clarify, this constraint is added for each reconvergence signal separately.

The basic assumption behind the above is the fact that a valid failing speedpath contains signals with only *one* driver that has a value in the bad machine which is inverse to the value in the good machine. By adding the above MUX gates, adding an assumption that only one selector can be `T` ensures that only one of the immediate drivers has an inverse value between the good and the bad machines. Fig. 6 illustrates the approach. The upper part of the illustration models the good machine while the lower part models the bad one. The XOR gate models flipping the value of SRC if the selector is `T`. The two MUX gates driving s are responsible for handling the reconverging signals I_1 and I_2, while their selectors guarantee that only one value out of I_1 and I_2 propagates to s in the bad machine. Notice that extracting the failing speedpath is done with a simple modification: for each reconvergence signal, we go backwards at the immediate driver with the active sensitivity selector.

VI. Dealing with complexity

Another challenge that we faced was run time of the algorithm for instances with a large sequential depth between the failing scan signal and the *src* candidates. The cone of influence was computed using a naive breadth-first search (BFS) on the sequential signals up to *src* candidates. In some cases, this computation resulted in cones with thousands of sequential signals which caused the core BMC engine to choke. We have developed an iterative process for computing the cone of influence based on functional detection of sensitivity of sequential signals.

Sensitivity of a sequential signal s at phase t is detected using our algorithm by assuming that s is the *src* candidate and assuming that $t_{src} = t$. The motivation behind the above iterative expansion process is that if a sequential signal is not sensitive during the window $[t_{src}, t_{scan}]$, then there is no need

to expand the cone over it, and thus it can be abstracted and considered as an input (constrained by the trace).

Data: scan, t_{src}, t_{scan}
Result: Compute cone of influence
$t_{current}$ = t_{scan};
ExpansionList = {} ;
while $\{t_{current} \neq t_{src}\}$ **do**
  C = ComputeConeOfInfluence(scan, ExpansionList);
  L = ExtractSetOfSequentialsAtBoundary (C);
  S = FilterSensitive (L, $t_{current}$);
  if *empty* S **then** $t_{current}$ = $t_{current}$ - 1;
  else ExpansionList = ExpansionList $\cup$ S ;
end
return ComputeConeOfInfleuce(ExpansionList) ;

A pseudocode explaining the steps of the algorithm for computing the cone of influence is presented herein: The function ComputeConeOfInfluece accepts as an argument a list of sequential signals and computes the cone of influence starting from the failing *scan* signal going backwards while stopping at the first sequential signals which do not belong to the list of sequential signals included in ExpansionList. The function ExtractSetOfSequentialsAtBoundary computes the sequential signals at the boundary of the cone of influence. The function FilterSensitive finds sensitive sequential signals belonging to the set L at phase $t_{current}$. The algorithm keeps expanding at sensitive sequentials at phase $t_{current}$ till no more sensitive sequentials are found, and only then it decreases the phase $t_{current}$. The algorithm stops at phase t_{src}. For illustration, during the first iteration, *scan* will be detected as sensitive at t_{scan} and thus it will be added to ExpansionList.

The algorithm computes a sub-cone of the cone computed by the naive BFS approach, and thus the detection of the failing speedpaths happens in a smaller cone which BMC can handle. The algorithm is sound in the sense that if a failing speedpath exists in the cone generated by the naive BFS approach, then it is guaranteed to be detected at the sub-cone generated by the iterative expansion algorithm. The proof for the soundness of the algorithm is based on showing a contradiction between the existence of such a path, and the fact that the algorithm didn't expand on a sequential in the boundary of the the sub-cone. Illustration of the proof is presented in Fig. 7.

Let M be a circuit with a failing scan signal $scan$ at phase t_{scan} and let C be the topological cone of influence of $scan$ computed using the naive BFS, where the stop points are primary inputs of M or sequential signals driving *src* candidates. Let us denote the set of internal sequential signals in C by L. Let L_{SRC} be a set of *src* candidates where $L_{SRC} \subseteq L$. Let C' be the topological cone of influence of $scan$ computed by expanding on a set of internal sequential signals L', where $L' \subseteq L$. We denote the phase when a sequential was expanded with t^e and the first phase when the sequential was sensitive by t^s.

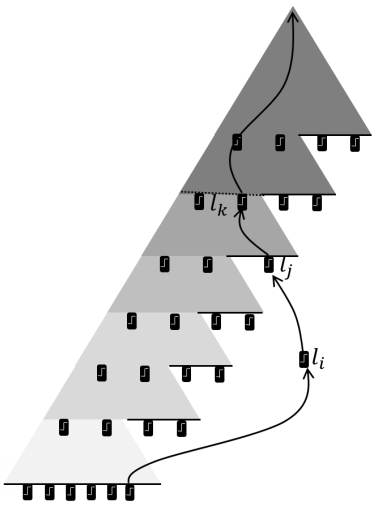

Fig. 7. Iterative expansion soundness

Lemma 6.1: If a sequential signal l_i is sensitive at phase t_i^s, and it was expanded in the iterative expansion process at phase t_i^e, then $t_i^s \leq t_i^e$.

Proof: If l_i is sensitive at phase t_i^s, then there is some failing speedpath P starting at (l_i, t_i^s) and containing the sequentials $\{(l_i, t_i^s), (l_{i+1}, t_{i+1}^s), \ldots, (l_n, t_n^s)\}$ (recall that $l_n = scan$) where every sequential l_i drives l_{i+1} through a combinational cloud, and $t_j^s \leq t_{j+1}^s$ for $i \leq j < n$. Recall also that if (l, t) belongs to a path, then l is sensitive at time t. Let us denote a list of corresponding phases $\{t_i^e, t_{i+1}^e, \ldots, t_n^e\}$ annotating for each sequential l_i the phase t_i^e when it was expanded in the iterative expansion. Recall that t_n^e is the phase when *scan* was expanded and t_n^s and is the phase when the it was sensitive. Both values should be equal to t_{scan}.

We will first show that for each j where $i \leq j < n$, that if $t_j^s \leq t_j^e$ then $t_{j+1}^e < t_j^s$. Let us assume on the contrary that $t_j^s \leq t_{j+1}^e$, since the algorithm detects sensitivity of l_j at the window $[t_{src}, \ldots, t_{j+1}^e]$, and the fact that l_j was found sensitive only at phase t_j^e, means that the algorithm didn't detect sensitivity of l_j at t_j^s which contradicts the fact that l_j is sensitive at t_j^s.

Thus $t_{j+1}^e < t_j^s$ and since $t_j^s < t_{j+1}^s$, we conclude that $t_{j+1}^e < t_{j+1}^s$. Based on that, $t_i^e < t_i^s$ implies $t_j^e < t_j^s$ for each $i \leq j \leq n$, implying that $t_n^e < t_n^s$. This is a contradiction since it means that the expansion phase for the *scan* signal is less than the sensitive phase of the same signal which contradicts the fact that they should be equal.

■

Theorem 6.2: If the algorithm detects a set of failing speedpaths SP for the cone C, then it will detect the same set of failing speedpaths for the sub-cone C'.

Proof: Let us assume on the contrary that there is a failing speedpath $P = \{(l_1, t_1), (l_2, t_2), \ldots, (l_n, t_n)\} \in SP$ which is not detected in C'. Then there exists a sequential

Test No.	# signals in cone	# of inputs on boundary	# of latches in cone	# of reconverg. signals	# of iterations	path length (in phases)	**# of paths**	Run Time (Sec.)
1	296	26	2	4	5	3	**1**	248
2	509	67	14	11	6	4	**1**	278
5	405	54	3	12	11	8	**1**	214
7	305	19	3	0	6	4	**1**	290
9	248	11	1	0	1	1	**1**	186
13	517	50	14	26	55	44	**1**	227
15	497	83	4	3	7	4	**1**	222
16	1528	212	59	86	14	8	**1**	745
18	27696	3009	635	8569	31	16	**1**	7168
6	3025	617	43	650	15	8	**2**	434
11	2403	345	22	209	12	7	**2**	318
12	1798	258	58	236	33	20	**2**	442
10	855	164	8	27	8	5	**3**	222
17	25895	7279	294	1070	30	16	**3**	6458
21	21864	4618	165	2266	33	18	**3**	3395
8	855	164	8	27	8	5	**4**	242
22	1545	303	46	5	12	6	**5**	5555
14	837	90	39	29	23	12	**6**	619
4	4665	704	106	1149	31	18	**7**	579
19	8789	994	125	2132	26	14	**7**	1713
20	26226	4035	168	2422	27	14	**15**	3285
3	4931	675	167	689	27	14	**40**	780

TABLE I
FAILING SPEEDPATHS FOUND ON NEXT GENERATION INTEL MICROPROCESSOR

signal $(l_i, t_i^s) \in P$ where l_i belongs to L but not L', and there is a combinational path from l_i to a boundary sequential l_j where $(l_j, t_j^s) \in P$ and $l_j \in L'$, and there is a combinational path from l_j to an internal sequential l_k where $(l_k, t_k^s) \in P$ and $l_k \in L'$. Recall that $t_j^s \leq t_k^s$. Let us assume that the l_k was expanded at phase t_k^e. Since l_k is part of the path, then based on lemma 6.1, $t_k^s \leq t_k^e$, and thus $t_j^s \leq t_k^e$. Recall that l_j is driving l_k, and l_k was expanded at phase t_k^e. The fact that the iterative expansion couldn't detect sensitivity for l_j during the window $[t_{src} \ldots t_k^e]$, and the fact that t_j^s belongs to that window, contradicts the fact that l_j is sensitive at t_j^s. ■

VII. RESULTS

We are currently at the early deployment stage of our application to the post-silicon speedpath debug lab responsible for the quality of the next generation Intel microprocessor. Most of the speedpaths shown in table I were detected using LADA first, and our tool was run afterwards to demonstrate its ability to detect the same failing speedpaths. In all the testcases shown in the table, we successfully found the same failing speedpath which was detected by LADA. For some cases, it took about two weeks trying detect the failing speedpath using LADA with no success, but after running the tool, the tool was able to isolate the failing speedpath easily. In other cases, our tool was run before LADA and was able to detect the failing speedpath and thus LADA was bypassed totally. Table I presents some information about each failing speedpath, when the cone of influence was produced using the iterative expansion algorithm. Column 2 shows that number of the signals in the cone computed by the iterative expansion algorithm. Column 3 shows the number of variables at the boundary of the cone, while column 4 shows the number of the internal sequential signals in the cone of influence. Column 5 shows the number of the internal reconvergence signals in the cone while column 6 shows the number of the expansion iterations to compute the cone of influence. Columns 7 shows the path length in phases from the path *root* to $scan$ while column 8 shows the number of the paths detected by the tool. Run time of the tool is shown in column 8. These results were produced on a 2.6 GHz Intel(R) Xeon(R) CPU processor. The run time demonstrates that isolation of post-silicon failing speedpaths can be completed in less than two hours using our tool compared to the costly, manual LADA based process, which took about a day to debug in average per failing speedpath.

VIII. FUTURE WORK

We have already started to see first cases where the vision of eliminating the need for optical probing for speed debug becomes a reality. Our next steps are to penetrate this technology to be used across the different microprocessor projects at Intel. Our next challenge is to eliminate the need to generate the RTL simulation trace, which today consumes long hours per test. Our alternative will be to get partial trace from the tester which produces the trace values for the scan signals only. Our algorithm will work the same, but will have to deal with more signals that do not have concrete value, but have `X` value instead. We hope that the scan signals will have enough coverage of the sequential signals so that they will be able to propagate concrete values coming from the scan

signals, forward and backward, and thus eliminating the X values. Current results are encouraging and we are optimistic that we will be able to reduce (and possibly eliminate) the LADA effort for all Intel microprocessor projects.

IX. Summary

We have introduced a new SAT-based algorithm that enables detecting failing speedpaths that are detected at the post-silicon debug stage. The value that this method brings is by reducing (and possibly eliminating) the cost of post-silicon speed debug using optical probing which is done today at Intel using LADA machines. Such a process consumes expensive machines, operators and costly TTM. Our method uses formal technology to model the incorrect behavior of the silicon from a functional perspective. We introduced a novel technique to model glitches by introducing new MUX gates in the reconvergence gates.

References

[1] S. Mitra, S. A. Seshia, N. Nicolici, "Post-silicon Validation Opportunities, Challenges and Recent Advances", in Design Automation Conference (DAC), 2010.

[2] P. Bastani, K. Killpack, L.-C. Wang, and E. Chiprout, "Speedpath prediction based on learning from a small set of examples", in *Design Automation Conf.*, 2008, pp. 217222.

[3] L. Lee, L.-C. Wang, P. Parvathala, and T. M. Mak, "On silicon-based speed path identification", in *VLSI Test Symp.*, 2005, pp. 3541.

[4] L. Xie, A. Davoodi, and K. K. Saluja, "Post-silicon diagnosis of segments of failing speedpaths due to manufacturing variations", in *Design Automation Conf.*, 2010, pp. 274279.

[5] T.M. McWilliams, "Verification of timing constraints on large digital systems", in *DAC*, 1980, pp. 139-147.

[6] G. Martin, J. Berrie, T. Little, D. Mackay, J. McVean, D. Tomsett, L. Weston. "An integrated LSI design aids system", in *Microelectronics Journal*, Vol. 12, Issue 4, 1981, Pages 1822.

[7] R. Hitchcock, G.L. Smith, and D.D. Cheng. "Timing analysis of computer hardware", in *IBM Journal of Research and Development (IBM)*, Vol. 26, Issue 1, 1982, pp. 100105.

[8] R. S. Venkataraman, "A Technique for Fault Diagnosis of Defects in Scan Chains", in *Int. Test Conference Proc.*, 2001, pp. 268-277.

[9] K. Cheng, "Partial Scan Designs Without Using a Separate Scan Clock", in *VLSI Test Symposium. Proc., 13th IEEE*, pp. 277-282.

[10] S. Rusu and S. Tam, "Clock Generation and Distribution for the First IA-64 Microprocessor", in *IEEE Solid State Circuits Conference*, 2000, pp. 176-177.

[11] S. Tam, S. Rusu, U. Nagarji Desai, R. Kim, Ji Zhang, I. Young, "Clock Generation and Distribution for the First IA-64 Microprocessor", in *IEEE Journal of Solid State Circuits*, Vol. 35, 2000, pp. 1545- 1552.

[12] R. Rowlette; T. Eiles, "Critical Timing Analysis in Microprocessors Using Near-IR Laser Assisted Device Alteration (LADA)", in *Proc. IEEE International Test Conf.*, 2003, pp. 264-273.

[13] R. McLaughlin; S. Venkataraman and C. Lim, "Automated Debug of speedpath Failures Using Functional Tests", in *VLSI Test Symposium*, 2009, pp. 91-96.

[14] M. Dehbashi; G. Fey, "Automated Post-Silicon Debugging of Failing Speedpaths", in *Asian Test Symposium*, 2012, pp. 13-18.

[15] C. H. Kong and E. P. Castro. "Application of LADA for Post-Silicon Test Content and Diagnostic Tool Validation", in *Proceedings of the 32nd International Symposium for Testing and Failure Analysis*, pp. 4317, 2006.

[16] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation" in *IEEE Transactions on Computers*, Vol. 35 Issue 8, 1986, pp. 677-691.

[17] A. Biere, A. Cimatti, E. Clarke. "Symbolic model checking without BDDs" in *Tools and Algorithms for the Construction and Analysis of Systems*, Vol. 1579, 1999, pp. 193-207.

[18] A. Biere, A. Cimatti, E. Clarke, M. Fujita. Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs" in *DAC*, 1999.

[19] R. E. Bryant, "Boolean Analysis of MOS Circuits", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 6, No. 4, 1987, pp. 634 - 649.

[20] C-J.H. Seger, R.E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories", in *Formal Methods in System Design*, Vol 6, No. 2, 1995, pp. 147-189.

[21] D. Kaiss, M. Skaba, Z. Hanna, Z. Khasidashvili, "Industrial Strength SAT-based Alignability Algorithm for Hardware Equivalence Verification", in *FMCAD*, 2007.