Automated Post-Silicon Debugging of Failing Speedpaths

Mehdi Dehbashi*
*Institute of Computer Science, University of Bremen
28359 Bremen, Germany
Email: dehbashi@informatik.uni-bremen.de

Görschwin Fey*†
†Institute of Space Systems, German Aerospace Center
28359 Bremen, Germany
Email: goerschwin.fey@dlr.de

***Abstract*—Debugging of speed-limiting paths (*speedpaths*) is a key challenge in development of *Very-Large-Scale Integrated* (VLSI) circuits as timing variations induced by process and environmental effects are increasing. This paper presents an approach to diagnose speedpaths under timing variations. First timing behavior of a circuit and corresponding variation models are converted into a functional domain. Then, our automated debugging based on *Boolean Satisfiability* (SAT) diagnoses speedpaths. The experimental results show the effectiveness of our approach on ISCAS'85 and ISCAS'89 benchmarks suites. In average, the diagnosis accuracy of 98.51% is achieved by our approach.**

***Keywords*—automated debugging, failing speedpath, timing variation**

I. Introduction

One of the major challenges in designing high-performance VLSI circuits is diagnosis and analysis of speedpaths. A *speedpath* is a frequency-limiting critical path which affects the performance of a chip [1] [2]. A speedpath that violates timing constraints at the post-silicon stage is called *failing speedpath* [3]. Speedpaths fail due to, e.g., timing variations induced by process, design and environmental effects [1].

Post-silicon validation involves applying test vectors to the chip in order to verify its correct behavior. When a speed failure is detected due to frequency constraints [4], the debug team identifies failing speedpaths. But this is a time-consuming process which requires a large effort. Thus, automated debugging approaches to identify failing speedpaths are necessary to speed up the process.

The loss of predictability in VLSI chips has triggered research on *Statistical Static Timing Analysis* (SSTA). The recent developments in SSTA are reviewed in [5]. SSTA methods analyze a circuit considering timing variations. The work in [3] proposes a formal procedure based on an *Integer Linear Programming* (ILP) formulation to diagnose segments of failing speedpaths due to process variations. The approach identifies segments of failing speedpaths that have a post-silicon delay larger than their estimated delay at the pre-silicon stage. *Parameterized Static Timing Analysis* (PSTA) is used in [6] to obtain a variational model for every candidate speedpath from a given set of potential candidates. These variational models are then combined to create a cost function. This PSTA-based cost function is utilized by a branch and bound approach to determine the most probable failing speedpaths. The approach needs a set of user-supplied paths as an input, and relies on post-silicon delay measurements prior to identifying failing speedpaths.

The approach in [4] employs clock shrinking on a tester combined with a CAD methodology to isolate failing speedpaths. A scan-based debug technique to failing speedpaths is presented in [7]. The technique is based on at-speed scan test patterns. In [8], a trace-based approach is presented to debug failing speedpaths. The approach uses trace buffers to provide real-time visibility to the speedpaths during the normal operation. The work in [9] uses on-chip delay sensors to improve timing prediction and to utilize them in order to isolate failing speedpaths. Each sensor is a sequence of logic gates with an approximate location on the layout. In [10], the diagnosis resolution is enhanced by processing failure logs at various slower-than-nominal clock frequencies. The work in [1] uses a statistical learning-based approach to predict failing speedpaths by measuring delays of a small set of representative speedpaths. A formal approach to find a small set of representative speedpaths in order to predict the timings of a large pool of target paths is proposed in [11]. A technique to automate debugging is presented in [12] by using failing functional tests and then algorithmically isolating failing speedpaths. The technique uses functional implications without incorporating timing information. Using only functional implications limits the diagnosis accuracy. An approach based on Boolean satisfiability has been presented in [13] to automate debugging. However, timing behavior of a circuit and timing variations are not considered in the model. Timing behavior of a circuit and timing variations are modeled in [14] in a functional domain.

In this paper we propose an approach based on *Boolean Satisfiability* (SAT) to automate debugging of failing speedpaths. Given a circuit and an erroneous behavior observed on circuit outputs due to timing variations, our approach extracts potential failing speedpaths. To automate debugging, first timing behavior of a circuit and corresponding timing variation models are converted into a functional domain [14]. Then a debugging instance is formulated in *Conjunctive Normal Form* (CNF). Afterwards our algorithm extracts failing speedpaths usig a SAT solver as an underlying engine. The diagnosis accuracy and the performance of the approach are experimentally shown on the ISCAS'85 and ISCAS'89 benchmark suites.

The remainder of this paper is organized as follows. Section II introduces preliminary information on timing parameters and speedpath debugging. Timing variation fault models are presented in Section III. Then, our methodology is presented in Section IV. Section V explains how timing behavior of a circuit and timing variations are converted into the functional domain. Then, the debugging method is demonstrated. Section VI presents experimental results on benchmark circuits. The last section concludes the work.

II. Preliminaries

A. Timing Parameters

The amount of time that a signal needs to propagate from the component inputs to its outputs is called *Delay*. *Timing*

This work has been funded in part by the German Research Foundation (DFG, grant no. FE 797/6-1).

variation is a change of the component's delay. An increase of the component delay due to timing variation is called $slowdown$. A decrease of the component delay due to timing variation is called $speedup$.

A *time unit* is considered to be an arbitrarily fine-grained but discrete unit of delay. The delays of gates and interconnects are assumed to be an integer multiple of one time unit. In a circuit where the shortest path delay is D_s time units, and the longest path delay is D_l time units, the current output O_t depends on the inputs of $I_{t-D_s}, I_{t-D_s-1}, \ldots, I_{t-D_l}$. Indices denote the times of input with a step of one time unit. Each index is also called *time step*.

A *clock period* is defined as T time units. The input to the combinational logic changes only once per *clock cycle* in synchronous circuits. The times of inputs with a step of one clock period are denoted by clock cycles. If the circuit has a clock period of T, the output at time step t depends on the inputs of the following clock cycles:

$$\forall i, a \leq i \leq b : [I_{t-iT-1}, \ldots, I_{t-(i+1)T}] \tag{1}$$

$$a = \lceil D_s/T \rceil - 1 \,, \quad b = \lceil D_l/T \rceil - 1$$

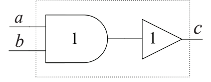
The times of input are partitioned according to the clock period T by this formula. In each clock cycle, the inputs are assumed to be fixed. For example, when $D_s = 1$, $D_l = 5$, $T = 5$, O_t depends on input values from time steps that fall within the previous clock cycle $[I_{t-1}, I_{t-2}, I_{t-3}, I_{t-4}, I_{t-5}]$, and in this clock cycle, the inputs do not change: $I_{t-1} = I_{t-2} = I_{t-3} = I_{t-4} = I_{t-5}$. When $T = 4$, O_t depends on input values from time steps that fall within the following clock cycles: $[I_{t-1}, I_{t-2}, I_{t-3}, I_{t-4}]$, $[I_{t-5}, I_{t-6}, I_{t-7}, I_{t-8}]$. This case is also called overclocking in which $T < D_l$. In this case, the current output depends on the inputs of multiple previous clock cycles. We note that, for our purpose, delay variation has the same effect as overclocking, since the delays of the gates will be scaled up, while the clock period remains the same.

When a slowdown occurs or the clock is overscaled, the longer paths fail because the input does not have enough time to propagate to the output. In this case, the current output result depends not only on the input of one previous clock cycle but also on the inputs of multiple previous clock cycles. The "older" inputs (the inputs of the clock cycles more distant from current time t) influence the output through longer paths and the "newer" inputs (the inputs of the clock cycles closer to the current time t) affect the output through shorter paths.

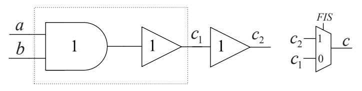
An *untimed gate* is a gate with a delay of one time unit. An original gate is converted to untimed gates by inserting buffers at the output of the corresponding gate [14] [15]. A circuit in which all components have a delay of one time unit is called *untimed circuit*.

B. Speedpath Debugging

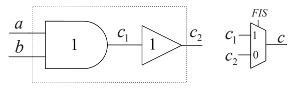
At the post-silicon stage, test vectors are applied to the chip and the clock period is reduced until an error is observed on outputs, registers or latches [6] [16]. This step is called clock shrinking. The error is detected by comparing the output values of the chip with the nominal output values obtained from simulation at the specified clock period. The activating test vectors at the specified frequency and the observed error



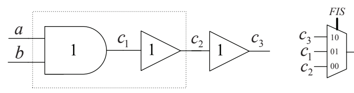
(a) An untimed AND gate with delay = 2



(b) One-time-unit slowdown fault injection



(c) One-time-unit speedup fault injection



(d) One-time-unit slowdown/speedup fault injection

Fig. 1. Fault models for timing variations

constitute an *Erroneous Trace* (ET). Having an erroneous trace, debugging starts to find failing speedpaths or some segments of failing speedpaths as fault candidates. In this paper, a gate is considered as the smallest segment on a failing speedpath. Here, a fault candidate includes both spatial and temporal information about the sensitized gate.

We assume that all registers are observable. In this case, an erroneous trace includes at least input vectors of two clock cycles as explained in Section II-A. An erroneous trace is denoted by ET and has the following parameters: $ET(I_{C0}, I_{C1}, O_{C2}, T)$. A test vector applied to the circuit inputs is denoted by I_C. Parameters I_{C0} and I_{C1} are the test vectors of two consecutive clock cycles causing an error. Parameter O_{C2} shows the observed error (erroneous output value) and parameter T is the clock period in which the error was observed.

III. Fault Model

In this section, we present slowdown and speedup fault models. These models will be used in our experiments in order to inject faults and to evaluate a circuit against timing variations.

A. Slowdown Fault Model

Figure 1(a) shows an untimed AND gate with a delay of two time units. To activate a slowdown of one time unit on a signal, we need the value of the signal one time step ago. Therefore, instead of selecting the value of a signal in the current time, its value at the previous time step is selected. To do this, a buffer with a delay of one time unit is inserted at the output of the untimed gate (Figure 1(b)). In a simulation tool, the value of signal c_1 at one time step ago can be found on the successor signal c_2. A multiplexer is added to select c_1 or c_2. The multiplexer delay is zero. The select line of the multiplexer is controlled by a *Fault Injection Signal* (FIS). When $FIS = 0$, signal c_1 is selected and the circuit has its normal behavior. If FIS is activated for one time step, signal c_2 is selected which has the value of signal c_1 at one time step ago. The slowdown fault model can be used to evaluate the silicon effects like mis-modeled logic cells, capacitive-coupling, and voltage droop. These silicon effects are investigated in [6].

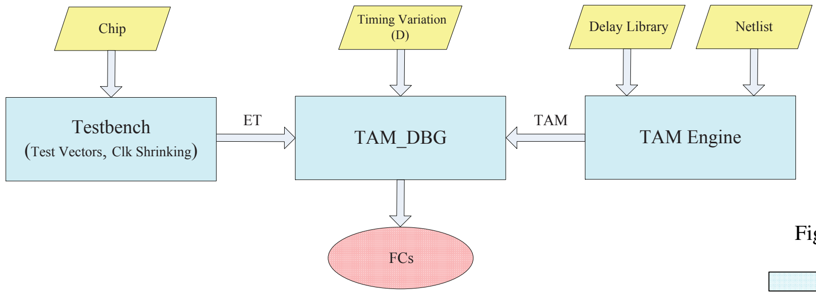


Fig. 2. Overview of proposed methodology

B. Speedup Fault Model

Figure 1(c) shows a model to activate speedup. It is assumed that a gate has a delay of more than one time unit. Therefore, there is at least one buffer at the output of an untimed gate. In this case, to activate a speedup of one time unit, no additional buffer is required. In Figure 1(c), signal c_2 holds the current value of the original gate output. The value of signal c_2 at the next time step can be found on signal c_1. Again when $FIS = 0$, the gate has a normal behavior. When $FIS = 1$ for one time step, a speedup of one time unit is activated by selecting signal c_1.

C. Slowdown and Speedup Fault Model

To have the ability to activate a speedup or a slowdown of one time unit, a multiplexer with three data inputs and one additional buffer is required. This case is shown in Figure 1(d). When $FIS = 00$, normal behavior is selected (signal c_2). When $FIS = 01$, a speedup of one time unit is activated by selecting signal c_1. When $FIS = 10$, a slowdown of one time unit is activated by selecting signal c_3.

When signal FIS is activated for one time step, a transient fault is injected. Signal FIS can be activated permanently or with a special time distribution to inject a permanent fault or a distributed fault.

IV. Methodology

Figure 2 shows the overall view of our methodology. At the post-silicon stage, the correct timing behavior of a circuit is validated by applying test vectors to the chip while clock shrinking is performed. In Figure 2, this step is performed in a testbench environment. When an erroneous trace is observed due to violating frequency constraints, debugging starts to find failing speedpaths. An *Erroneous Trace* (ET) includes the test vectors activating a timing fault at a specified frequency and the corresponding erroneous output values.

Test vectors should be applied at-speed to the chip as the test vectors in consecutive clock cycles may activate a timing fault. A tester can be used to apply test vectors unintrusively [4]. In microprocessor-based systems, *Software-Based Self-Testing* (SBST) methods [17] can be effectively used to apply test vectors at-speed in order to validate the timing behavior of an internal module.

Having an erroneous trace, our goal is to automatically find potential speedpaths which have failed and have created the erroneous output values of the corresponding erroneous trace. To automate debugging, first we convert the timing behavior of a circuit into the functional domain. When the timing behavior of a circuit is available in the functional domain, formal verification methods can comprehensively analyze the timing effects of the circuit. In Figure 2, the TAM engine models the timing behavior of a circuit in the functional domain with a discrete unit of delay. The inputs of the TAM engine are a netlist and a delay library. The output of the TAM engine is called *Time Accurate Model* (TAM).

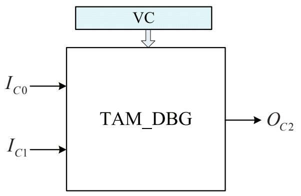


Fig. 3. TAM_DBG instance for combinational circuits

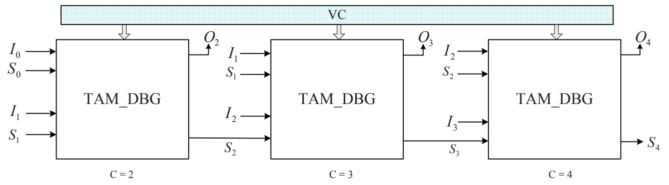


Fig. 4. TAM_DBG instance for sequential circuits

Having the TAM and an ET, the debugging process starts. This step is denoted by TAM_DBG in Figure 2. In the TAM_DBG engine, timing variation models are added according to the user-defined maximum timing variation D. The timing variation model can vary the value of a signal along the time axis. The behavior of timing variations is controlled by a constraint called *Variation Control* (VC). In this case, debugging investigates whether a timing variation on a signal is observable as the erroneous output value of the corresponding erroneous trace. This investigation is performed by constraining the output and inputs of the created model to the values of the erroneous trace. The TAM_DBG engine identifies fault candidates whose timing variation may cause the erroneous behavior of the ET.

For combinational circuits, we assume that they are a part of synchronous circuits. Thus, the inputs to the combinational logic changes only once every cycle. For combinational circuits, the instance of Figure 3 is created by our framework.

In sequential circuits, if all registers are observable, then they are treated like the combinational circuit of Figure 3. In this case, I_C includes both inputs and state bits. But when not all registers are observable, the error may be detected several clock cycles after fault activation. In this case, the sequential circuit is unrolled as many times as the number of clock cycles constituting the erroneous trace. In each unrolled clock cycle, a TAM_DBG model is created and the VC controls the timing variations on the whole created instance. Figure 4 shows a sequential circuit. The erroneous trace has three clock cycles. In each clock cycle, a TAM_DBG model is used where its output (O_{i+1}, S_{i+1}) depends on the inputs and state bits of two clock cycles $(I_{i-1}, S_{i-1}, I_i, S_i)$. The overall model is formulated as follows:

$$\Phi_C = \prod_{i=1}^{n} TAM\_DBG\,(\,I_{i-1}, S_{i-1}, I_i, S_i, \quad O_{i+1}, S_{i+1}\,) \quad (2)$$

Parameter n is the length of erroneous trace which indicates the number of clock cycles needed to observe the error. The diagnosis accuracy depends on the granularity of the time unit and the accuracy of the variation models. A time unit should be selected such that timing variations are an integer multiple of one time unit.

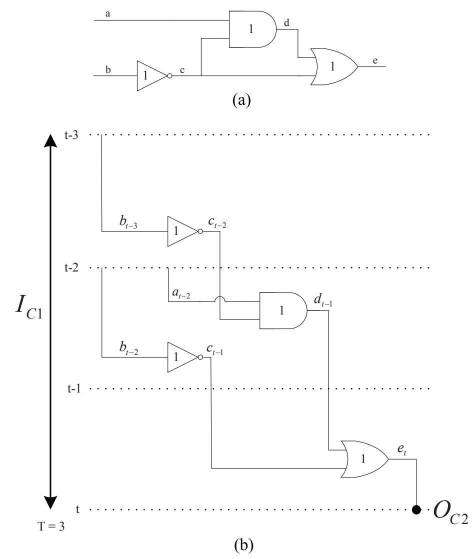


Fig. 5. (a) Untimed circuit (b) TAM circuit

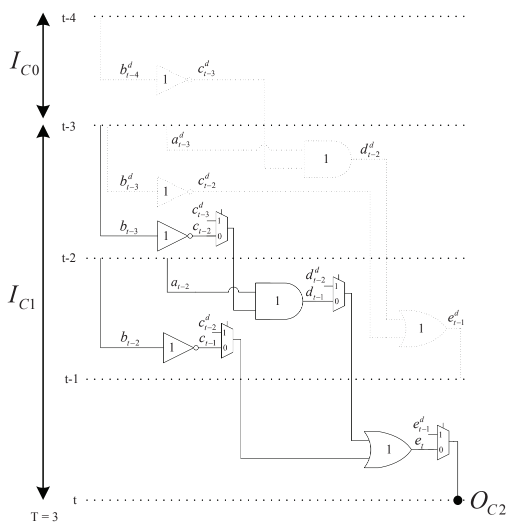


Fig. 6. TAM_DBG instance

V. TAM-Based Debugging

In this section, first we shortly demonstrate the TAM model by an example from [14]. Then, we utilize it to debug failing speedpaths.

A. Model

To construct the TAM, first all gates are converted to *untimed gates*. Each untimed gate has a delay of one time unit. This process is performed by inserting buffers at the output of each gate. After this step, all components in the circuit have a delay of one time unit. In the example of Figure 5(a), to sake of simplicity, it is assumed that all gates already have a delay of one time unit. Having an untimed circuit, the TAM circuit is created. The underlying idea to construct the TAM is to use copies of a gate to represent the value of a gate at different points in time. Therefore, if an untimed gate is exercised several times at different time steps (e.g. due to reconvergent fanout), one copy of the untimed gate in each related time step is created. In Figure 5(b), gate NOT is duplicated as it has been exercised at two different times through different paths. In the TAM circuit of Figure 5(b), each signal s_t represents an original signal s at time step t. In the worst case, the size of the TAM may be exponentially larger than the original circuit. However, in our experiments the increase in size was manageable.

Furthermore, a constraint to model a clock period is applied as mentioned in Section II-A. In Figure 5(b), the clock period is $T = 3$. Therefore, the signals b_{t-2} and b_{t-3} should have the same value. To model maximum timing variation D, D additional copies of the TAM are created. Figure 6 shows this case when $D = 1$. Multiplexers on the outputs of TAM gates can activate a timing variation by selecting the value of a signal from different time steps. The select lines of multiplexers are controlled by *Variation Control* (VC) modeling timing variations. When there is a maximum timing variation $D = 1$ at $T = 3$, therefore the output O_{C2} may depend on the inputs of two clock cycles I_{C0} and I_{C1}. In this case, the older inputs (I_{C0}) affect the output through longer paths. The newer inputs (I_{C1}) affect the output through shorter paths.

After creating the instance of Figure 6, debugging starts. The inputs I_{C0} and I_{C1} and the output O_{C2} are constrained to the values of the erroneous trace obtained from a testbench. Then, debugging answers the following question: Which fault candidate at which time step can be activated to cause the erroneous behavior of the corresponding erroneous trace?

B. Algorithm

Figure 7 shows the debugging algorithm as pseudocode. The inputs of the algorithm are a TAM circuit, an erroneous trace ET, and maximum variation delay D. The output of the debugging algorithm is a set of fault candidates $\mathcal{F}$. Each fault candidate $F_i \in \mathcal{F}$, $i = 1, 2, \dots, |\mathcal{F}|$, includes the location and the time step of a gate.

In line 2, the TAM circuit is duplicated D times. Then, multiplexers are inserted to model timing variations (line 3). Set SEL is a set of select lines of multiplexers. The variable $sel_i \in SEL$, $i = 1, 2, \dots, m$, holds the integer value of the select lines related to multiplexer i. Line 5 constrains the CNF by parameters of the erroneous trace ET. Variable d is a variable for timing variation and is initialized to 1 (line 7). The constraint of line 10 guides debugging to find minimum timing variations causing the erroneous behavior. If the CNF is satisfiable, all solutions are extracted and debugging finishes (lines 14-15). Otherwise, the previous constraint is removed (line 19) and d increases (line 20) in order to find a solution with an increased timing variation in the next step. This procedure repeats until d reaches the maximum timing variation D (line 22).

When there are multiple erroneous traces, one TAM_DBG instance for each erroneous trace is created. For transient faults, select lines of a single gate in different instances and in different time steps should be independent, because in each erroneous trace, some independent transient faults may be activated. For permanent faults, select lines of a single gate in different instances and in different time steps are connected to each other. A transient fault and a permanent fault can be distinguished by reapplying test vectors to detect whether the erroneous behavior is observed again.

VI. Experimental Results

In this section, we use TAM-based debugging experimentally to debug logic circuits under timing variations. The experiments are carried out on a Quad-Core AMD Phenom(tm)

```
function $TAM\_DBG\,(In:\ TAM, ET, D,\ Out: \mathcal{F})$
$Duplicate\,(\,TAM, D\,)$
$SEL = Insert\_Multiplexers()$
$sel_i \in SEL,\ i = 1, 2, \ldots, m$
$Add\_Constraint\,(\,ET:\ I_{C0}, I_{C1}, O_{C2}, T\,)$
$\mathcal{F} = \emptyset$
$d = 1$
do
{
  $Add\_Constraint\,(\,(\sum_{i=1}^{m} sel_i) = d\,)$

  if $Solve() == SAT$ then
  {
    $\mathcal{F} = Extract\_All\_Solutions()$
    break
  }
  else
  {
    $Remove\_Constraint\,(\,(\sum_{i=1}^{m} sel_i) = d\,)$
    $d = d + 1$
  }
} while $d \leq D$
end function
```

Fig. 7. TAM-based debugging

II X4 965 Processor (3.4 GHz, 8 GB main memory) running Linux. We use the combinational and sequential circuits of ISCAS'85 and ISCAS'89 benchmark suites to evaluate our approach. We synthesize the circuits using Synopsys Design Compiler with Nangate 45nm Open Cell Library [18]. The TAM-based debugging described in this paper is implemented using C++ in the WoLFram environment [19]. For the experiments, one time unit is $0.01ns$. MiniSAT is used as underlying SAT solver [20].

We utilize a simulation testbench to obtain the effect of timing variations on the outputs. The simulation testbench is implemented using Verilog in the ModelSim environment. In the simulation testbench, there are two instances of a circuit: golden instance and faulty instance. The outputs of these two instances are compared to detect an error and constitute an erroneous trace. A single slowdown fault of one time unit is injected in the circuit to create a faulty instance. Several points in the circuit are chosen as fault locations. At a clock period T, random test vectors are generated and applied to the golden instance and the faulty instance of the circuit. If no error is observed for the activated faults, the clock period is decreased (clock shrinking). Having a new clock period, the procedure repeats until an erroneous behavior is observed. Test vectors activating the fault at the specified frequency and the corresponding erroneous output values constitute an erroneous trace. The erroneous trace is given to TAM-based debugging. Having the initial erroneous trace, TAM-based debugging starts to find potential fault candidates.

Table I presents the experimental results. The table shows the circuit name (first column), the total number of gates (#Gates), the required run time (Time) measured in CPU seconds (s), and the final number of fault candidates (#FC). Each fault candidate is a gate indicating if a slowdown of one time unit at the appropriate time step on the output of the corresponding gate occurs, the erroneous behavior of the erroneous trace is created. The number of fault candidates (#FC) indicates the diagnosis accuracy of TAM-based debugging. A smaller number of #FC indicates a higher accuracy. The diagnosis accuracy may increase by having higher quality erroneous traces [21]. As the diagnosis accuracy is opposite to the number of fault candidates, we define the diagnosis accuracy as follows:

$$Diagnosis\_Accuracy = 1 - \frac{\#FC}{\#Orig} \tag{3}$$

The ninth column shows the diagnosis accuracy in percent. In this column, a larger number indicates a higher diagnosis accuracy.

The second column in the table shows the total number of gates in the original circuit. By inserting buffers at the output of the original gates, it is converted to an untimed circuit. The total number of gates in the untimed circuits is shown in the third column. Afterwards, the TAM circuit is constructed. The fourth column shows the total number of gates in the TAM circuits. In our experiments, the circuit c6288 of ISCAS'85 is omitted. This circuit is a multiplier known to have a very large number of paths and requires special approaches like [22] to handle the paths.

For circuit c17, the number of original gates is 6, while the number of untimed gates is 26. It shows that 20 (26 - 6) additional buffers have been added to the original circuit to create the untimed circuit.

The number of untimed gates for c499 is larger than for c432. But the number of TAM gates for c432 is larger than for c499. This shows that in c432 there are more paths in comparison to c499 which consequently increase the size of the TAM. The number of fault candidates for c432 is 20 (20 gates), while the number of fault candidates for c499 is 2 (2 gates). In our experiments, all fault candidates together highlight failing speedpaths or some segments of failing speedpaths. Each fault candidate includes the location and the time step of fault activation.

Also the number of untimed gates for c5315 is larger than for c3540, while the number of TAM gates for c3540 is larger than for c5315, which indicates a larger number of paths in c3540. Therefore, the required time to create the TAM increases for c3540 which is shown in the table. In the table, usually debugging needs a longer time than the TAM creation process. But when a circuit has a large number of paths (for example c3540 and c7552), the time for TAM creation increases.

For sequential circuits, we assume that all registers are observable. In the table, #Gates for sequential circuits indicates the number of combinational components excluding the number of flip flops. As the table shows, the debugging time for the most of sequential circuits is longer than the TAM creation time. For circuits s15850, s35932, and s38584 which are the larger circuits and have more paths, the TAM construction time is longer than the debugging time.

The final row in the table shows the average values for each column. On average, the diagnosis accuracy of 98.51% is achieved by our approach.

VII. Conclusion

This paper introduced a methodology to automate debugging for logic circuits under timing variations. In the framework, first the timing behavior of a circuit is converted into the functional domain under a discrete model of time unit. The new circuit is called *Time Accurate Model* (TAM). Then,

TABLE I
TAM-BASED DEBUGGING

Circuit	#Gates when Time Unit = 0.01ns			Time (s)			#FC	Diag. Accuracy
Comb.	Original	Untimed	TAM	TAM	DBG	Total		%
c17	6	26	35	0.00	194.97	194.97	2	66.67
c432	115	511	15446	135.01	1473.17	1608.18	20	82.61
c499	179	840	4358	4.60	212.58	217.18	2	98.88
c880	172	814	6483	14.95	1258.92	1273.87	17	90.12
c1355	238	1112	14338	93.33	2024.13	2117.46	26	89.08
c1908	142	658	5171	6.68	260.50	267.18	3	97.89
c2670	280	1296	8817	35.28	1391.03	1426.31	19	93.21
c3540	391	1792	50664	2347.64	1010.63	3358.27	10	97.44
c5315	632	3042	18283	290.24	1158.21	1448.45	16	97.47
c7552	772	3657	58468	3240.66	2093.90	5334.56	21	97.28
Seq.								
s27	9	40	79	0.01	260.47	260.48	3	66.67
s298	59	277	893	0.20	468.80	469.00	6	89.83
s386	67	300	575	0.09	387.46	387.55	5	92.54
s444	83	370	1158	0.45	554.80	555.25	7	91.57
s526	97	442	1235	0.49	562.96	563.45	7	92.78
s713	96	421	3538	4.44	907.97	912.41	13	86.46
s838	130	642	2250	2.03	799.09	801.12	11	91.54
s953	216	967	3567	4.40	685.47	689.87	9	95.83
s1196	280	1279	9124	30.16	1509.14	1539.30	21	92.50
s1238	278	1278	10842	49.59	1100.84	1150.43	15	94.60
s1494	315	1443	4373	6.54	271.74	278.28	3	99.05
s5378	635	2884	8355	41.62	566.05	607.67	7	98.90
s9234	813	3836	17473	274.10	722.63	996.73	9	98.89
s15850	1537	7389	64402	4365.84	654.56	5020.40	5	99.67
s35932	3630	17728	18869	865.25	474.14	1339.39	4	99.89
s38584	6438	29651	89018	9394.33	701.84	10096.20	1	99.98
Average	677.31	3180.58	16069.77	815.69	834.85	1650.54	10.08	98.51

variation models are inserted in the TAM. Afterwards, TAM-based debugging finds potential fault candidates including their spatial and temporal information. The experimental results showed that the approach achieves 98.51% diagnosis accuracy.

REFERENCES

[1] 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. 217–222.

[2] L. Lee, L.-C. Wang, P. Parvathala, and T. M. Mak, "On silicon-based speed path identification," in *VLSI Test Symp.*, 2005, pp. 35–41.

[3] 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. 274–279.

[4] K. Killpack, S. Natarajan, A. Krishnamachary, and P. Bastani, "Case study on speed failure causes in a microprocessor," *IEEE Design & Test of Computers*, vol. 25, no. 3, pp. 224–230, 2008.

[5] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: From basic principles to state of the art," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 589–607, 2008.

[6] S. Onaissi, K. R. Heloue, and F. N. Najm, "PSTA-based branch and bound approach to the silicon speedpath isolation problem," in *Int'l Conf. on CAD*, 2009, pp. 217–224.

[7] J. Zeng, R. Guo, W.-T. Cheng, M. Mateja, J. Wang, K.-H. Tsai, and K. Amstutz, "Scan based speed-path debug for a microprocessor," in *European Test Symposium*, 2010, pp. 207–212.

[8] X. Liu and Q. Xu, "On signal tracing for debugging speedpath-related electrical errors in post-silicon validation," in *Asian Test Symposium*, 2010, pp. 243–248.

[9] M. Li, A. Davoodi, and L. Xie, "Custom on-chip sensors for post-silicon failing path isolation in the presence of process variations," in *Design, Automation and Test in Europe*, 2012, pp. 1591–1596.

[10] V. J. Mehta, M. Marek-Sadowska, K.-H. Tsai, and J. Rajski, "Timing-aware multiple-delay-fault diagnosis," *IEEE Trans. on CAD*, vol. 28, no. 2, pp. 245–258, 2009.

[11] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Design Automation Conf.*, 2010, pp. 386–391.

[12] R. McLaughlin, S. Venkataraman, and C. Lim, "Automated debug of speed path failures using functional tests," in *VLSI Test Symp.*, 2009, pp. 91–96.

[13] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[14] M. Dehbashi, G. Fey, K. Roy, and A. Raghunathan, "On modeling and evaluation of logic circuits under timing variations," in *EUROMICRO Symp. on Digital System Design*, 2012.

[15] K. P. Ganeshpure and S. Kundu, "On ATPG for multiple aggressor crosstalk faults," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 5, pp. 774–787, 2010.

[16] K. Killpack, C. V. Kashyap, and E. Chiprout, "Silicon speedpath measurement and feedback into EDA flows," in *Design Automation Conf.*, 2007, pp. 390–395.

[17] M. Psarakis, D. Gizopoulos, E. Sánchez, and M. S. Reorda, "Microprocessor software-based self-testing," *IEEE Design & Test of Computers*, vol. 27, no. 3, pp. 4–19, 2010.

[18] *Nangate 45nm Open Cell Library*, http://www.nangate.com.

[19] A. Sülflow, U. Kühne, G. Fey, D. Große, and R. Drechsler, "WoLFram – a word level framework for formal verification," in *IEEE/IFIP Int'l Symposium on Rapid System Prototyping*, 2009, pp. 11–17.

[20] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[21] M. Dehbashi, A. Sülflow, and G. Fey, "Automated design debugging in a testbench-based verification environment," in *EUROMICRO Symp. on Digital System Design*, 2011, pp. 479–486.

[22] W. Qiu and D. M. H. Walker, "An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit," in *Int'l Test Conf.*, 2003, pp. 592–601.