Efficient Automated Speedpath Debugging

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*—Speedpath diagnosis is one of the major challenges in designing high-performance *Very-Large-Scale Integrated* (VLSI) circuits due to timing variations caused by process variations and environmental effects. In this paper, an efficient approach to automate speedpath debugging is presented. The approach relies on converting the timing behavior of a circuit and its corresponding timing variations into functional domain. Afterwards, a SAT-based debug engine is utilized to extract potential failing speedpaths. The experimental results on ISCAS'85 and ISCAS'89 benchmark suites show that our approach achieves a 63% decrease in the size of model resulting in 54% decrease in the debugging time compared to previous work while having a high diagnosis accuracy.**

I. Introduction

Debugging of speedpaths is a major concern in developing high performance VLSI circuits. At the post-silicon stage, a speedpath may violate timing constraints due to timing variations induced by process variations and other environmental effects. After detecting a speed failure due to frequency constraints [1], the debugging starts to identify failing speedpaths. But this process requires a large effort which consumes a significant portion of the IC development cycle. In this case, automated approaches to debug speedpaths can reduce the development time of IC products.

The work in [2] utilizes integer linear programming to diagnose segments of failing speedpaths caused by process variations. On-chip delay sensors are used in [3] to improve timing prediction in order to isolate failing speedpaths. Failure logs are processed in [4] at various slower-than-nominal clock frequencies to enhance the diagnosis accuracy. Delays of a small set of representative speedpaths are measured in [5] to predict failing speedpaths using a statistical learning-based approach. The work in [6] uses at-speed scan test patterns to debug failing speedpaths. The real-time visibility of speedpaths is provided by using trace buffers in [7] to diagnose speedpaths. The work in [8] uses fault dictionary as well as forward propagation and backward implication techniques to diagnose stuck-at, stuck-open and delay faults. Diagnosis based on fault dictionaries is computationally efficient. But this requires a tight link between test generation and observation of erroneous behavior. Moreover, typically no model for variation is underlying dictionary based diagnosis.

The work in [9] investigates the relationship between the quality of the test vectors for post-silicon validation and the accuracy of yield-performance predictions. Functional tests and implications are utilized in [10] to automate speedpath debugging. Using only functional implications without incorporating timing information limits the diagnosis accuracy. The work in [11] presents a model based on Boolean satisfiability to automate debugging. But no timing information is included in the model. A timing analysis tool is proposed in [12] that integrates a pattern-dependent delay model into its analysis. An approach to automate the debugging of failing speedpaths has been presented in [13] using Boolean satisfiability. The approach constructs a time accurate model of a circuit based on a fine-grained but discrete time unit. Copies of a gate are used to represent the value of a gate at different points in time. Using multiple copies of a gate increases the size of the model and consecutively increases the debugging time.

This paper presents an efficient approach to automate speedpath debugging which integrates static timing analysis and functional analysis in order to efficiently construct a compact timing model of a circuit. As a result, the debugging time decreases significantly. Given an erroneous behavior observed on circuit outputs due to timing variations, a debugging instance based on Boolean satisfiability is created to automatically extract potential failing speedpaths. The approach is also utilized to debug an overclocked circuit which may fail to produce the expected outputs under timing variations. In comparison to previous work in [13], the experimental results on ISCAS'85 and ISCAS'89 benchmark suites show a 63% decrease in the size of the model resulting in 54% decrease in the debugging time. At the same time, our new approach achieves a high diagnosis accuracy.

The remainder of this paper is organized as follows. Section II introduces preliminary information. Our methodology is presented in Section III. Section IV explains effects of timing variations and overclocking on the function of a circuit. Then in this section, the debugging model and algorithm are demonstrated. Section V presents experimental results on benchmark circuits. The last section concludes the work.

II. Preliminaries

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

The *Arrival Time* (AT) of a signal is the time elapsed for a signal to arrive at a certain point. The maximum arrival time AT_{max} (minimum arrival time AT_{min}) at a certain point of a circuit is the maximum (minimum) amount of the time which a signal requires to propagate from primary inputs to that point. The *Propagation Time* (PT) of a certain point of a circuit along a particular path is the time required for a signal at that point to propagate to the output.

Each combinational circuit is represented by a directed acyclic graph $G = (K, E)$, referred to as the *circuit graph*, where K is the set of circuit nodes and E, the set of edges, corresponds to the gate input-output connections in the circuit. The *successors* of a node $g \in K$ are given by a set of nodes $A = \{g_j | \forall j, (g, g_j) \in E\}$, $|A|$ is the number of successors of g. The *predecessors* of a node $g \in K$ are given by a set of nodes $B = \{g_j | \forall j, (g_j, g) \in E\}$, $|B|$ is the number of predecessors of g. A *path* P from node g_1 to node g_r is a sequence of nodes $(g_1, g_2, \ldots, g_r)$ with $(g_i, g_{i+1}) \in E$.

III. Methodology

An overview of our methodology is shown in Figure 1. Post-silicon validation of correct timing behavior involves applying test vectors to a chip and clock shrinking [14] [15]. In Figure

This work was supported in part by the European Union (Project DIAMOND, FP7-2009-IST-4-248613) and in part by the German Research Foundation (DFG, grant no. FE 797/6-1).

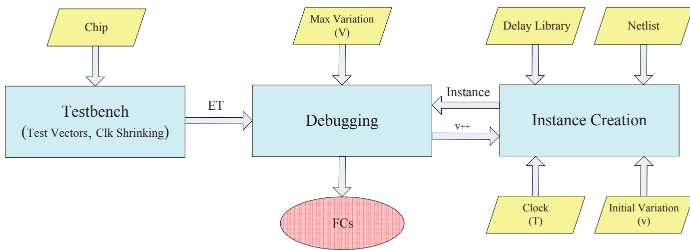

Fig. 1. Overview of proposed methodology

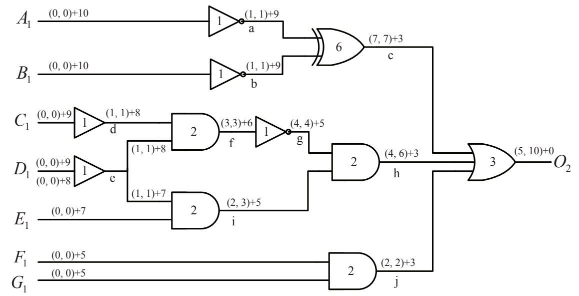

Fig. 2. Original circuit, T = 10

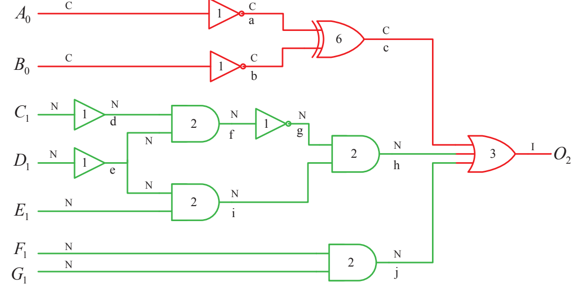

Fig. 3. Overclocked circuit, T = 9

1, this step is done in a *testbench*. When an error is observed on outputs or flipflops, this error is returned as an *Erroneous Trace* (ET). An erroneous trace includes the activating test vectors at the specified frequency and the observed error. An erroneous trace includes at least the input vectors of two clock cycles and is denoted as $ET(IN_0, IN_1, O_2, T)$. Parameters IN_0 and IN_1 are the test vectors of two consecutive clock cycles causing an error. The observed error and the clock period are shown by parameters O_2 and T.

A slowdown increases the delay of some paths. These paths may violate the frequency constraint and may create an erroneous trace. In this case, our goal is to find speedpaths which have failed and have created the erroneous output of the corresponding ET. To automated debugging, we also convert the timing behavior of the circuit in the presence of variations into the functional domain. In Figure 1, the *Instance Creation* engine constructs a functional model. The inputs of the instance creation engine are a netlist, a delay library, the clock period T and the timing variation v. We discuss the instance creation engine in Sections IV-B and IV-C in detail.

Having the debugging instance and an ET, debugging starts. First the inputs and output of the debugging instance are constrained according to the values of the corresponding ET. Then, debugging investigates whether a timing variation on a gate is observable as the erroneous output value of the corresponding ET. If debugging finds solutions, they are returned as fault candidates FCs and the algorithm terminates. Otherwise, if no solution is found, the timing variation v increases. Then, a new instance according to the new timing variation is created. This procedure repeats until the timing variation reaches maximum timing variation V.

In the following, first we investigate the effects of timing variations and overclocking on the function of a circuit (Section IV-A). Then, our debugging model and algorithm are presented which are able to diagnose failing speedpaths under timing variations and overclocking.

IV. Speedpath Debugging

A. Overclocking versus Timing Variation

In this section, first we explain the effect of overclocking on the function of a circuit [16]. Then, the effect of timing variations on the function of a circuit is investigated and compared to overclocking. Finally, a model is constructed which is able to convert the timing behavior of not only a normal circuit but also an overclocked circuit into the functional domain under timing variations.

Considering D and T as the delay of the circuit and the clock period, respectively, the output at clock cycle $i{+}1$ (O_{i+1}) depends on the inputs of the following clock cycles [16]:

$IN_i,\ IN_{i-1},\ \ldots,\ IN_{i-b} \qquad b = \lceil D/T \rceil - 1$

The indices indicate the clock cycles. For example, when $D = 10$ and $T = 9$, the output at clock cycle 2 (O_2) depends on the inputs of clock cycles 1 and 0 ($IN_1,\ IN_0$). In the remainder of the paper, to sake of simplicity, we discuss the case in which output O_2 depends on the inputs of two previous clock cycles IN_1 and IN_0. In this case, the signals on longer paths fail to arrive at outputs as their delay is longer than the clock period. Therefore, the output at clock cycle 2 (O_2) is affected by the inputs of clock cycle 1 (IN_1) through shorter paths and the inputs of clock cycle 0 (IN_0) through longer paths. The function of the circuit changes such that the output depends on a special combination of IN_1 and IN_0. In the following, we explain how to construct the new function with an example.

In the original circuit of Figure 2, first AT_{max} and AT_{min} are calculated by traversing the circuit from inputs to outputs. In the figure, the arrival times are shown as a pair (AT_{min}, AT_{max}). Afterwards, by traversing the circuit from output to inputs through a path, the propagation time of each point of the path is calculated. In Figure 2, the propagation time PT of each point is written after arrival times. The number on each point p of the circuit denotes $(AT_{min}, AT_{max}) + PT_p$. During the backward traversal along a path, three cases N, C and I may occur:

$$Case(p) = \begin{cases} N & \text{if } AT_{max} + PT_p \leq T, \\ C & \text{if } T < AT_{min} + PT_p, \\ I & \text{if } AT_{min} + PT_p \leq T < AT_{max} + PT_p. \end{cases} \quad (1)$$

Cases N, C and I are also called *normal cases*. Having the case of each point in the circuit, the compact timing model of a circuit in the functional domain can be constructed. In case N (*Non-critical case*), the signal at point p of the circuit has enough time to be propagated to the output along the corresponding path. Because the sum of the maximum arrival time from inputs to that point of the circuit (AT_{max}) and the required time to propagate a signal from that point to the output is smaller than clock period T. This point of the circuit is called non-critical point and is denoted by N. In this case, the function of this point of the circuit depends on only some inputs of IN_1 which fall in the input cone of the corresponding point.

Case C (*Critical case*) implies a critical point and is denoted by C. As in this point, no signal has enough time to propagate to the output, the function of this point depends on only some inputs of IN_0 which fall in the input cone of the corresponding point. In case I (*Indefinite case*), the function of point p is affected by a combination of inputs IN_1 and IN_0. The backward traversal starting at an I point always leads to independent C and N points.

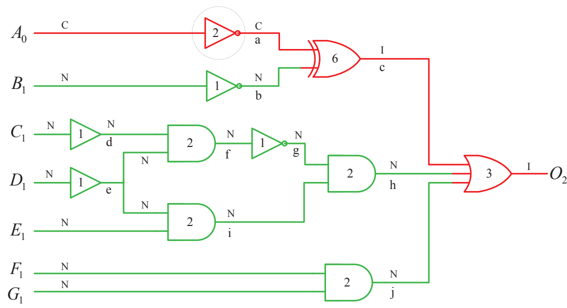

Fig. 4. Faulty circuit with a slowdown at gate a, T = 10

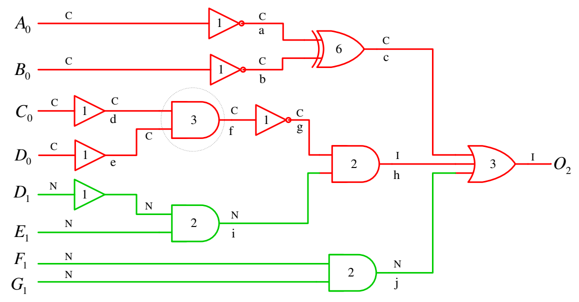

Fig. 5. Overclocked circuit with a slowdown at gate f, T = 9

Considering the example of Figure 3, when $T = 9$, the clock is overscaled. Because the clock period ($T = 9$) is less than the delay of the longest path ($D = 10$). Therefore, output O_2 depends on a combination of IN_1 and IN_0. In this example, point O_2 satisfies the condition of case I and is marked as I. In Figure 3, a point case is written in capital on top of the wire. A wire name is written by a lower case letter under the wire. By backward traversal of point O_2, this case is decomposed to case C and case N. Inputs A and B are critical and are considered at clock cycle 0. Other inputs are non-critical and considered at clock cycle 1. In the figure, non-critical parts are shown by green color. Other parts have red color.

A timing variation may occur at every point in the circuit. In the example circuit of Figure 4, we assume that there is a slowdown of one time unit at the output of gate a. Therefore, the delay of gate a increases from 1 to 2. When $T = 10$ and there is no timing variation, the circuit has normal behavior. But by having a timing variation at point a, some paths of the circuit become critical. In this example, path (A, a, c, O) becomes critical. Therefore, input A is taken at clock cycle 0. Although the overclocking has the same effect on all paths, i.e., decrease of the clock period for all paths of the circuit, timing variation may affect only some special paths of the circuit, i.e., increase of the delay for some paths of the circuit. This is seen by comparing Figure 3 and Figure 4.

Having an overclocked circuit, some paths are critical and some paths are non-critical. In this case, a slowdown fault may change the function of a circuit more or less severely. This case is depicted in Figure 5. In the example, T is 9 and there is a slowdown fault at the output of gate f. Therefore, more paths become critical. *If a section of the circuit has contributed in both a critical path and a non-critical path, that section of the circuit is duplicated.* In the example of Figure 5, gate e contributes in one critical path (D, e, f, g, h, O) and one non-critical path (D, e, i, h, O). Thus, it is duplicated such that in the critical path the input is taken from clock cycle 0 (D_0) and in the non-critical path the input is taken from clock cycle 1 (D_1).

We assume that the function of a gate itself does not change. However, there can be more precise models which can also change the function of a gate. Assume that there is a signal which has been propagated and arrived at the middle of a gate and does not have enough time to propagate to the output of the gate. Therefore, the function of a gate can change depending on internal structure of a gate.

B. Debugging Model

A debugging model is an instance including timing variation models. The model is able to activate a timing variation at every point of a circuit, i.e., it is able to change the function of each point of a circuit due to an activated timing variation.

In the previous section, we showed what happens if a slowdown fault at a special point of a circuit is activated. However, a slowdown fault may occur at every point of a circuit and may change the function of the corresponding points and circuit outputs. Therefore, we add timing variation as a parameter to Formula 1 in order to obtain the timing behavior of each point of the circuit. The new formula is written as follows:

$$Case(p,v) = \begin{cases} N' & \text{if } AT_{max} + PT_p + v \leq T, \\ C' & \text{if } T < AT_{min} + PT_p + v, \\ I' & \text{if } AT_{min} + PT_p + v \leq T < AT_{max} + PT_p + v. \end{cases} \tag{2}$$

In this formula, parameter v denotes the considered slowdown at point p. Cases N', C' and I' are also called *variation cases*. These cases indicate the case of a point of a circuit when a timing variation (slowdown) occurs. In case N', even when having slowdown v, point p is a non-critical point. Case C' denotes a critical point when considering slowdown v. In case I', the function of point p, with considering slowdown v, depends on a combination of the inputs of clock cycle 1 and clock cycle 0.

The normal case of point p in a circuit can be N, I or C. While the variation case of point p can be N', I' or C', which show the case of point p when a timing variation occurs. The possible transitions from a normal case to a variation case are shown in Figure 6 (a). In the following, a transition is denoted by a dot. For example a transition from case N to case N' is denoted by $N.N'$. When the normal case of a point is N, a slowdown increases the propagation time of the corresponding point. Therefore, case N may be converted to cases N', I' or C', i.e., transitions $N.N'$, $N.I'$ or $N.C'$ may occur. Also increasing the propagation time in case I may lead to cases I' or C', i.e., transitions $I.I'$ or $I.C'$. The transitions including an indefinite normal case (I) or an indefinite variation case (I') are called *indefinite transitions*. Indefinite transitions include $N.I'$, $I.I'$ and $I.C'$. Other transitions are called *definite transitions*: $N.N'$, $N.C'$ and $C.C'$.

As the arrival times are zero for primary inputs, there is no indefinite case for a primary input. Therefore, the primary inputs can have only non-critical and critical cases. The transition states for a primary input are shown in Figure 6 (b). *If a point of the circuit has the non-critical case ($Case(p) = N$), but it is converted to a critical point when considering timing variation v ($Case(p,\ v) = C'$), this variability is modeled by inserting a multiplexer at point p changing its behavior in order to debug the circuit.*

In the example of Figure 7, clock period T is 10 and there can be a slowdown of one time unit in every point of the circuit. In this example, $Case(A) = Case(a) = N$, while $Case(A, 1) = Case(a, 1) = C'$. Therefore, a multiplexer at point A and one at point a is inserted in order to change the timing behavior of the corresponding points. Points B and b also have the same conditions. Moreover, these sections of the circuit are duplicated enabling to model the effect of both clock cycle 1 and clock cycle 0. Therefore, gates a and b are duplicated. If there is a timing variation for example at point

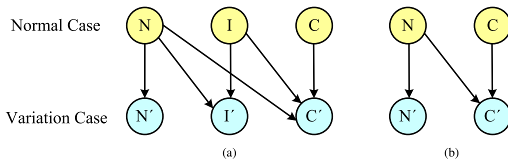

Fig. 6. Transitions from normal cases to variation cases for a point p: (a) p is an internal point, (b) p is a primary input

a, the corresponding multiplexer is activated. In this case, the input is taken from clock cycle 0. Otherwise, the input is taken from clock cycle 1. From an input with a multiplexer on it, along the successor nodes, the multiplexers are also inserted at the successor nodes whose normal cases or variation cases are indefinite, i.e., successor nodes with indefinite transitions.

As the debugging instance of Figure 7 shows, a multiplexer modeling timing variation can be activated in order to change the function of the circuit according to a point at which a slowdown occurs. The paths including multiplexers are called *potentially-critical paths*.

In the overclocked circuit of Figure 8, paths (A, a, c, O) and (B, b, c, O) are critical paths. In this case, a timing variation of one time unit renders some paths as potentially-critical paths, i.e., a timing variation on each point of this path can convert the non-critical path to a critical path. Paths (C, d, f, g, h, O) and (D, e, f, g, h, O) are potentially-critical paths. The gates on potentially-critical paths are duplicated in order to model the effect of timing variations in the functional domain.

For debugging, the inputs and the output of the debugging instance are constrained to the inputs and the output values of the corresponding erroneous trace (ET). Then, the debugging engine answers the following question by activating the multiplexers: If there is a timing variation at a point of the circuit, can the erroneous behavior of the corresponding erroneous trace be observed? This investigation is performed by activating the select lines of multiplexers and observing its effect on the output. If the debugging instance is satisfiable, the debugging returns a set of fault candidates. Otherwise, timing variation v increases and a new instance is constructed until reaching maximum timing variation V.

C. Instance Creation Algorithm

Figure 9 shows the algorithm to create the debugging instance in pseudocode. The inputs of the algorithm are an original circuit, a delay library, clock period T and timing variation v. In the algorithm the *original circuit* is traversed and a *new circuit* is constructed on the fly. A wire in the original circuit is called *original wire*. A wire in the new circuit is called *new wire*. For each wire of the original circuit, four flags are considered: $N.N'$, $C.C'$, $N.C'$, $Indefinite$ (line 3). For example, if the flag $N.N'$ is 1, it shows that the original wire has already been visited through a non-critical path. In this case, the corresponding new wire in the new circuit has already been saved in the variable $w_{N.N'}$. For case $N.C'$, two variables are considered: one variable to save the new wire for the clock cycle 1 (non-critical wire $w_{N.C'}$), one variable to save the new wire for the clock cycle 0 (critical wire $w'_{N.C'}$).

If an original wire is visited for the first time through a non-critical path, the flag $N.N'$ is set. If this original wire is again visited through another non-critical path, we do not traverse back the original wire again. Only its corresponding stored new wire is connected to the parent node. There is the same scenario for case $C.C'$ and case $N.C'$.

In line 6, AT_{min} and AT_{max} are calculated by *Static Timing Analysis* (STA). Then, the algorithm traverses the circuit back starting at each *Primary Output* (PO) (lines 7-11). In line 10, the newly created outputs in the new circuit are stored in

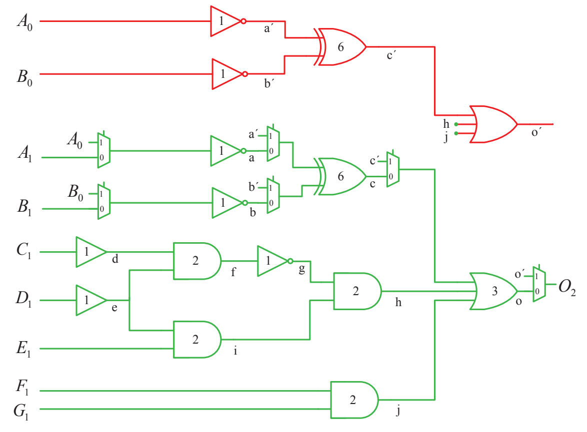

Fig. 7. Debugging instance, T = 10, slowdown = 1

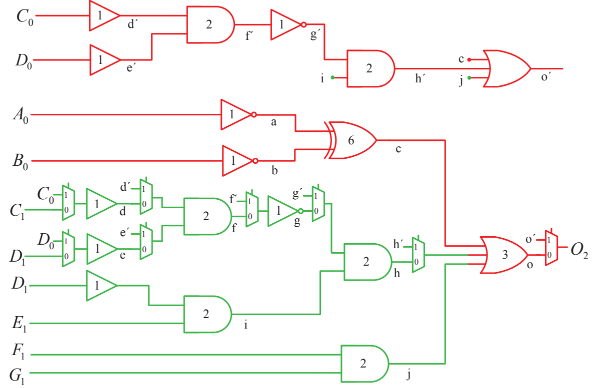

Fig. 8. Debugging instance, T = 9, slowdown = 1

set *New_Out*. The inputs of function *Traverse* are an original gate and its propagation time (line 13). Function *Traverse* constructs a new gate (lines 36-42) after all of its inputs in the original circuit have already been visited. The output of function *Traverse* is the output of the newly created gate in the new circuit. In line 15, each input in of original gate $gate$ is selected iteratively. The case of input in is calculated according to Formula 1 and Formula 2 (line 17). The cases are categorized into definite cases and indefinite cases. In the definite cases, the behavior of the point in the normal case and in the case of a timing variation is certain and fixed. Lines 19, 23, and 27 handle the definitive cases. In the indefinite cases (line 31), the behavior of the point in the normal case or in the case of a timing variation is not known in advance, i.e., the function of this point of the circuit depends on a special combination of the inputs of clock cycle 1 and clock cycle 0. This combination may change depending on the propagation time of the current point in a path.

If a definite point has already been visited (lines 20, 24, and 28), it is not traversed back again through a path with the same case. In this case, only the stored wire of the corresponding point is used to create the parent node. Otherwise, the backward traversal continues (lines 21, 25, and 29). In the algorithm, whenever an indefinite case is visited, the backward traversal continues (line 33) until reaching a definite case.

After traversing all of the inputs of an original gate, a new gate is created. If at least one of the gate inputs is a *Primary Input* (PI) with case $N.C'$ (potentially-critical point) (line 36), or one of the gate inputs is along a potentially-critical path (line 37), then two gates and one multiplexer are created in the new circuit (lines 38-40). This case can be seen in the example of Figure 7 where gate a, gate a', and a multiplexer are created. Otherwise, one new gate is created (line 42). After creating the new gate, the output of the new gate is saved in its corresponding variable on the original gate, and also its

```
1   function DBG_Instance (In : Circ, DelayLib, T, v)
2
3   Orig_Wire Flags: N.N', C.C', N.C', Indefinite
4   Orig_Wire New_Wires: w_N.N', w_C.C', w_N.C', w'_N.C'
5
6   STA()
7   foreach out ∈ PO do
8   {
9     gate = pre(out)
10    New_Out = New_Out ∪ Traverse(gate, 0)
11  }
12
13  W_out Traverse(gate, PT)
14  {
15   foreach in ∈ gate.inputs do
16   {
17    switch Case(in).Case(in, v)
18    {
19     case N.N':
20       if N.N' == 1 then W_in = W_in ∪ w_N.N'
21       else W_in = W_in ∪ Traverse(pre(in), PT + gate.D)
22
23     case C.C':
24       if C.C' == 1 then W_in = W_in ∪ w_C.C'
25       else W_in = W_in ∪ Traverse(pre(in), PT + gate.D)
26
27     case N.C':
28       if N.C' == 1 then W_in = W_in ∪ w_N.C' ∪ w'_N.C'
29       else W_in = W_in ∪ Traverse(pre(in), PT + gate.D)
30
31     case I.I' || I.C' || N.I':
32       // Indefinite
33       W_in = W_in ∪ Traverse(pre(in), PT + gate.D)
34    }
35   }
36   if (∃in : in ∈ PI && in_current_case == N.C') ||
37      (∃w : w ∈ W_in && w is mux_out) then
38      W_out = W_out ∪ Create_Gate(W_in)
39      W_out = W_out ∪ Create_Gate'(W_in)
40      W_out = W_out ∪ Create_Mux()
41   else
42      W_out = W_out ∪ Create_Gate(W_in)
43
44   Set_Flags_and_New_Wires(gate.output, W_out)
45   return W_out
46  }
47
48  end function
```

Fig. 9. Debugging Instance Creation

corresponding flag is set (line 44).

In sequential circuits, 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 debugging instance is created where its output depends on the inputs and state bits of two clock cycles.

V. Experimental Results

In this section, we use our proposed debugging approach to debug logic circuits under timing variations. The experiments are carried out on a Dual-Core AMD Opteron(tm) Processor 2220 SE (2.8 GHz, 32 GB main memory) running Linux. The combinational and sequential circuits of ISCAS'85 and ISCAS'89 benchmark suites are used to evaluate our approach. We synthesize the circuits using Synopsys Design Compiler with Nangate 45nm Open Cell Library [17]. The speedpath debugging approach described in this paper is implemented using C++ in the WoLFram environment [18]. For the experiments, one time unit is $0.01ns$. MiniSAT is used as underlying SAT solver [19].

We implemented a simulation testbench using Verilog in the ModelSim environment. The simulation testbench is utilized to obtain the effect of timing variations on the outputs. There are two instances of a circuit in the simulation testbench: 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 random points in the circuit are chosen as fault locations. Random test vectors are generated and applied to the golden instance and the faulty instance of the circuit at a clock period T. If no error is observed for the activated faults, the clock period is decreased (clock shrinking). Using a shorter clock period, the procedure repeats until erroneous behavior is observed. Clock shrinking and random test vectors can activate the timing faults on short paths. Therefore, timing faults on short paths can also be diagnosed. 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 the debugging engine. Having the initial erroneous trace, debugging starts to find potential fault candidates.

The experimental results are presented in Table I. 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). In the table, a column with name TAM shows the results of the approach presented in [13]. A column with name NEW indicates the approach presented in this paper.

The total number of gates in an original circuit, TAM and our instance is shown in columns 2 through 4. Column 5 shows the amount of decrease in the size of the NEW model in comparison to the TAM model in percent. The times in the table are indicated as the required time to create a debugging instance (*Instance Creation Time*), the required time to debug (*Debugging Time*) and the total time. Column 12 shows the amount of decrease in the total time of the NEW approach in comparison to the TAM approach in percent.

In Section $\#FC$ of the table, column $Gate$ shows the total number of gates returned as fault candidates. In this column, 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 total number of possible paths constituted by the gates in column $Gate$ is shown in column $Path$. A path does not necessarily start at a primary input node reaching a primary output node. The path can be a segment in the middle of the circuit. The length of the shortest path from a real fault location to the fault candidates is shown in column $Dist$. Therefore, if the fault candidates include the real fault location, the distance is zero. A smaller number of paths in $Path$ with a smaller number of gates on them ($Gate$) with a shorter distance from a real fault location indicates a higher diagnosis accuracy. The diagnosis accuracy can be increased by having higher quality erroneous traces [20]. One advantage of our approach is automatically extracting a segment of a path as fault candidate.

As the table indicates, the number of gates in our model is always smaller than the number of gates used in the approach in [13] (comparison of column 3 and column 4). Therefore, the memory consumption of our approach is less than the TAM approach. On average, our approach needs 2580 gates while the TAM needs 7030 gates which implies the new approach has 63% decrease in the size of the model in comparison to the TAM.

The required time to create a debugging instance in the new approach is always shorter than the TAM approach. This decrease of the instance creation time is tangible especially for large and complex circuits. The debugging time depends on not only the size of the model but also on the number of

TABLE I
SPEEDPATH DEBUGGING

Circuit	#Gates				Instance Creation Time (s)		Debugging Time (s)		Total Time (s)			#FC		
Comb.	Original	TAM	NEW	%Decrease	TAM	NEW	TAM	NEW	TAM	NEW	%Decrease	Gate	Path	Dist.
c17	6	16	**9**	**43.75**	0.00	0.00	260.63	267.22	**260.63**	267.22	-2.53	2	1	0
c432	115	7128	**3089**	**56.66**	241.11	**0.16**	2156.11	1999.59	2397.22	**1999.75**	**16.58**	20	1	0
c499	179	1824	**1429**	**21.66**	13.58	**0.10**	294.75	287.52	308.33	**287.62**	**6.72**	2	1	0
c880	172	2860	**534**	**81.33**	45.39	**0.09**	1796.25	1631.90	1841.64	**1631.99**	**11.38**	17	1	0
c1355	238	6044	**4215**	**30.26**	170.95	**0.19**	2889.47	2462.80	3060.42	**2462.99**	**19.52**	26	2	0
c1908	142	2240	**276**	**87.68**	18.19	**0.04**	391.61	372.61	409.80	**372.65**	**9.07**	3	1	0
c2670	280	3980	**776**	**80.50**	109.40	**0.32**	2022.07	2044.68	2131.47	**2045.00**	**4.06**	19	1	0
c3540	391	22934	**5314**	**76.83**	3063.39	**0.40**	1061.53	674.73	4124.92	**675.13**	**83.63**	6	2	0
c5315	632	7714	**1381**	**82.10**	504.08	**0.89**	827.67	768.74	1331.75	**769.63**	**42.21**	6	3	0
c7552	772	24738	**15533**	**37.21**	4008.44	**4.26**	3568.92	2332.99	7577.36	**2337.25**	**69.15**	22	1	0
Seq.														
s27	9	36	**29**	**19.44**	0.01	**0.00**	361.54	374.11	**361.55**	374.11	-3.47	3	1	0
s298	59	414	**125**	**69.81**	0.20	**0.01**	647.91	652.75	**648.11**	652.76	-0.72	6	1	0
s386	67	262	**146**	**44.27**	0.09	**0.01**	543.74	546.38	**543.83**	546.39	-0.47	5	2	0
s444	83	554	**184**	**66.79**	0.53	**0.03**	749.76	738.20	750.29	**738.23**	**1.61**	7	1	0
s526	97	568	**219**	**61.44**	0.60	**0.02**	745.57	754.66	**746.17**	754.68	-1.14	7	1	0
s713	96	1726	**592**	**65.70**	18.55	**0.06**	1439.69	1301.63	1458.24	**1301.69**	**10.74**	13	1	0
s838	130	956	**340**	**64.44**	6.77	**0.11**	1167.48	1123.31	1174.25	**1123.42**	**4.33**	11	1	0
s953	216	1670	**367**	**78.02**	16.85	**0.11**	990.54	960.95	1007.39	**961.06**	**4.60**	9	1	0
s1196	280	4244	**591**	**86.07**	111.89	**0.12**	2370.67	2078.69	2482.56	**2078.81**	**16.26**	21	3	0
s1238	278	5046	**742**	**85.30**	158.46	**0.12**	1729.87	1528.15	1888.33	**1528.27**	**19.07**	15	1	0
s1494	315	2078	**398**	**80.85**	22.38	**0.09**	415.31	374.21	437.69	**374.30**	**14.48**	3	1	0
s5378	635	3800	**1556**	**59.05**	143.61	**2.81**	844.62	752.11	988.23	**754.92**	**23.61**	7	1	0
s9234	813	7894	**3621**	**54.13**	428.18	**6.24**	1159.88	951.07	1588.06	**957.31**	**39.72**	9	2	0
s15850	1537	26258	**3230**	**87.70**	5478.62	**41.89**	1145.70	580.37	6624.32	**622.26**	**90.61**	5	1	0
s35932	3630	7606	**4178**	**45.07**	1374.00	**249.00**	745.58	567.67	2119.58	**816.67**	**61.47**	4	1	0
s38584	6438	40188	**18214**	**54.68**	11123.10	**423.06**	1198.25	264.74	12321.40	**687.80**	**94.42**	1	1	0
Average	677.31	7029.92	**2580.31**	**63.30**	1040.71	**28.08**	1212.50	1015.07	2253.21	**1043.15**	**53.70**	9.6	1.3	0.0

fault candidates. When the number of fault candidates is larger, the solver (debugging engine) needs a longer time to extract fault candidates. The total time is the sum of the instance creation time and the debugging time. As the table shows, for the large circuits, the new approach needs a shorter total time. On average, the new approach spends 1043 seconds while the TAM approach spends 2253 seconds which indicates a 54% decrease in the required total time.

For circuit c880, the number of gates as fault candidates is 17 which constitute one path. While for circuit c1355, there are 26 gates as fault candidates constituting 2 paths. Also the distance is zero which shows the set of fault candidates includes the real fault location. For all circuits, the diagnosis accuracy of the new approach is same as the TAM approach for single faults.

VI. CONCLUSION

We introduced a methodology to automate debugging for logic circuits under timing variations. The approach was based on converting the timing behavior of a circuit and its corresponding timing variations into the functional domain. Having the new circuit in the functional domain and an erroneous trace, our debugging approach finds potential failing speedpaths. The experimental results on ISCAS'85 and ISCAS'89 benchmarks suites show a 63% decrease in the size of model resulting in 54% decrease in the debugging time in comparison to previous work while our new approach achieves a high diagnosis accuracy.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] H. Cox and J. Rajski, "A method of fault analysis for test generation and fault diagnosis," *IEEE Trans. on CAD*, vol. 7, no. 7, pp. 813–833, 1988.

[9] M. Sauer, A. Czutro, B. Becker, and I. Polian, "On the quality of test vectors for post-silicon characterization," in *European Test Symposium*, 2012, pp. 1–6.

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

[11] 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.

[12] D. Tadesse, R. I. Bahar, and J. Grodstein, "Test vector generation for post-silicon delay testing using SAT-based decision problems," *J. Electronic Testing*, vol. 27, no. 2, pp. 123–136, 2011.

[13] M. Dehbashi and G. Fey, "Automated post-silicon debugging of failing speedpaths," in *Asian Test Symposium*, 2012, pp. 13–18.

[14] 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.

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

[16] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Int'l Conf. on CAD*, 2011, pp. 667–673.

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

[18] 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.

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

[20] 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.