# Over-testing Reduction for Delay Faults through False Path Exclusion Using RTL Information

Yuki Yoshikawa[1], Satoshi Ohtake[2] and Hideo Fujiwara[2]
[1]Graduate School of Information Science, Hiroshima City University
3-4-1 Ozuka-higashi, Asaminami, Hiroshima 731-3194, Japan
[2]Graduate School of Information Science, Nara Institute of Science and Technology
Kansai Science City 630-0192, Japan
E-mail:[1]*yosikawa@hiroshima-cu.ac.jp*, [2]{*ohtake, fujiwara*}@*is.naist.jp*

## Abstract

*While design-for-testability (DFT) techniques are generally used in order to reduce test generation complexity, they induce over-testing problems. In general, DFT techniques make a large number of untestable paths testable. However delay on the path that becomes testable does not affect circuit performance because the path was originally untestable. Therefore we consider testing such path to be over-testing. In this work, we reduce the over-testing by identifying false paths using register transfer-level information. Our method identifies a subset of false paths within a reasonable time. Experimental results for some RTL benchmark circuits show the effectiveness of our false path identification method.*

## 1. Introduction

In today's competitive LSI market, test cost and product quality of LSIs are important factors that benefit production and contribute to customer satisfaction. Alleviating overkill of products is an important issue in order to keep manufacturing costs lower, that is, reducing the number of parts which are failed by its tests under evaluation but that never cause any failure in the application[1]. One of the solutions to this problem is reduction in over-testing caused by design-for-testability techniques, which is our objective. To achieve that, in this paper, we propose a method of identifying false paths using register-transfer level (RTL) information (most of the previous works have been done at gate level) and carefully exclude them from the target of testing.

For high speed circuits, delay testing is emphasized to guarantee the timing correctness of circuits in addition to testing logical correctness. The path delay fault (PDF) model can deal with accumulation of small delays along a path. Testing a PDF on a path detects a defect that induces a delay beyond a specified clock period[2]. The main disadvantage of the PDF model is that the number of faults in a circuit is extremely large because the number of paths is an exponential function of gates in the circuit. Furthermore, it is said that a large number of faults are functionally untestable, which includes both combinationally untestable and sequentially untestable ones.

To reduce test costs, design-for-testability (DFT) techniques such as enhanced scan [3] or standard scan with *skewed-load testing* [4] or *broadside testing* [5] are used. These techniques make all or most of the sequentially untestable paths testable. However, delay on the path that becomes testable by some DFT never affect the circuit performance. This is because no transition launched at the starting point of the untestable path is ever propagated to the ending point of the path during normal operation. Therefore we consider testing such path to be over-testing. The over-testing causes yield loss or overkill of products because good circuits in normal operations may be regarded as faulty ones under test mode. Moreover test generation time and test application time become shorter if the over-testing is reduced.

To reduce the over-testing, first, untestable paths in an original circuit are identified. Then the information about untestable paths should be propagated to the actual test environment, that is, paths that become testable by a DFT have to be excluded from testing. In the last decade years, several path identification methods have been proposed. For combinational circuits, the techniques presented in [6],[7],[8] are approaches for identifying combinationally untestable paths at gate-level. The works for sequential circuits [9, 10] are equally important. However, these gate-level approaches may be hard to handle all the exponential number of paths in large scale circuits. In [11], false paths are identified at RTL. However, their interest is to prevent an inaccurate decision about circuit performance caused by identifying false paths as critical paths. Hence they do not list identified false paths and do not consider shorter false paths than critical paths.

As an approach from upper level, we propose a method of identifying false paths[1] using RTL information such as

[1]In this paper, we mainly focus on identification of non-robust untestable paths, but it can be easily extended to identification of false paths.

load-enable signals of registers and select signals of multiplexers (MUXs). We deal with paths at RTL, called RTL paths. An RTL path is a path which starts at a primary input or a register and ends at a register or a primary output, which passes through only combinational modules. The total number of RTL paths in a circuit is much smaller than that of gate-level paths, therefore our path identification method can be performed in a reasonable amount of time. Experimental results for some RTL benchmark circuits show that our method can identify many RTL paths as false in a few seconds. In addition, we show that there are a lot of gate-level paths corresponding to the RTL paths that are identified as false.

## 2. Preliminaries

### 2.1. RTL circuit

Our path identification method approaches to structural RTL designs as shown in Figure 1. A structural RTL design consists of a controller represented by a finite state machine and a datapath composed of RTL modules, which are MUXs, combinational operation modules and registers, and RTL signal lines between them. They are connected to each other by control signal lines and status signal lines. The controller controls control inputs of the RTL modules. We assume that all the control inputs of the RTL modules in the datapath are directly controlled by the controller. On the other hand, status signals from the datapath are fed into the controller. We assume that state transitions are completely specified for all pairs of a state and an input vector.

If a target RTL circuit is described as a functional RTL, it may be difficult to directly extract its structural RTL from the description. In such a case, the information about structure can be obtained during synthesis process. For example, the commercial high-level synthesis tool *Explorations tool* (Y Explorations, Inc.)[12] has a function to transform a functional RTL to its structural RTL.

### 2.2. RTL path

A path at gate-level is an ordered set of gates $\{g_0, g_1, .., g_n\}$, where $g_0$ is a primary input or a flip-flop and $g_n$ is a primary output or a flip-flop when we consider a sequential circuit. Also $g_i (1 \leq i \leq n-1)$ is a gate. The number of gate level paths in a circuit is extremely large. When we consider the circuit at RTL, gate level paths between two registers are dealt with as a bundle of paths. In this paper, we use the concept of RTL paths. An RTL path is a path which starts at a primary input or a register and ends at a register or a primary output, which passes through only combinational modules and has a bit width. The number of RTL paths in a circuit described at RTL is much smaller than that of gate-level paths in the circuit described at gate-level.

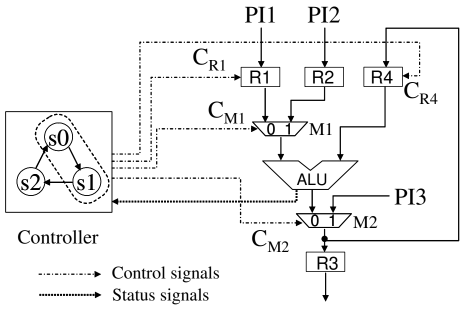


**Figure 1. An RTL circuit.**

### 2.3. Logic synthesis

Logic synthesis is the manipulation of logic specifications to create logic models as an interconnection of logic primitives. Thus logic synthesis determines the gate-level structure of a circuit. The possible configurations of a circuit are many. Optimization plays a major role, in connection with synthesis, in determining the gate-level netlist [13]. Our path identification method is applied to RTL paths in an RTL circuit and the information about the paths at RTL is propagated to gate-level paths in a gate-level circuit transformed by a logic synthesis. Then it is necessary to clarify the correspondence of RTL paths to gate-level paths. As one solution to achieve the clarification, we consider a restricted synthesis called module interface preserving-logic synthesis.

**Definition 1:(Module interface preserving-logic synthesis)**
Given an RTL circuit, if logic synthesis transforms each RTL module and each RTL signal line into an individual gate-level netlist and individual single-bit signal lines, respectively, the logic synthesis is referred to as *module interface preserving-logic synthesis* (MIP-LS). □

During an MIP-LS for an RTL circuit, each RTL module is transformed to an individual gate-level netlist. Then optimization is performed within each module. Each RTL signal line connecting RTL modules is split to single-bit signal lines. Therefore, for the RTL circuit, the connectivity of all the RTL modules is guaranteed to be propagated to a synthesized gate-level circuit through any MIP-LS.

## 3. False path identification

### 3.1. RTL false path

We define an RTL false path as follows.

**Definition 2: (RTL false path)**
An RTL path $p$ in an RTL circuit is *RTL false* if any gate-level path corresponding to $p$ in its gate-level circuit is gate-level false for any logic synthesis. □

Our objective is to identify bundles of gate-level paths as false by finding RTL false paths. However, it may be hard under any logic synthesis to map an RTL path to all the gate-level paths corresponding to the RTL path completely. To achieve the mapping, we restrict logic synthesis to MIP-LS.

**Definition 3: (RTL false path with respect to MIP-LS)** An RTL path $p$ in an RTL circuit is *RTL false with respect to MIP-LS* if any gate-level path corresponding to $p$ in its gate-level circuit is gate-level false for any MIP-LS. □

In this paper, we focus on identification of RTL false paths w.r.t. MIP-LS. RTL false paths w.r.t. MIP-LS are just referred to as RTL false paths in the rest of this paper.

Let $C$ and $C'$ be an RTL circuit and its gate-level circuit synthesized by an MIP-LS, respectively. Let $F = \{Fj|1 \leq j \leq m\}$ be a set of flip-flops in $C'$ corresponding to an $m$ bit register $R$ in $C$. $\tau(R)$ denotes a mapping from $R$ to $F$. Let $Rs$ and $Re$ be registers that are the starting register and the ending register of $p$, respectively. Let $M_1, M_2, ..., M_l$ be RTL modules on $p$. Suppose that $p$ passes through input ports $M_{1in}, M_{2in}, ..., M_{lin}$ and output ports $M_{1out}, M_{2out}, ..., M_{lout}$. Let $Q$ be a set of all gate-level paths between $\tau(Rs)$ and $\tau(Re)$ passing through $M_{1in}, M_{1out}, M_{2in}, M_{2out}, ..., M_{lin}, M_{lout}$ in order. $\delta(p)$ denotes a mapping from $p$ to $Q$.

**Theorem 1:** An RTL path $p$ in an RTL circuit $C$ is RTL false if at least one of the following three conditions is satisfied for any input sequence.

*Condition 1:* No transition is ever launched at the register of the starting point of $p$.

*Condition 2:* No transition at the starting point of $p$ is ever propagated to the ending point along $p$.

*Condition 3:* No value captured into the register at the ending point of $p$ is ever propagated to any primary output. □

**Proof:** We show that $\forall g \in \delta(p)$ in a gate-level circuit synthesized by any MIP-LS is gate-level false if at least one of the three conditions of Theorem 1 is satisfied for any input sequence.

A combinational RTL module $M_a$ in $C$ and its synthesized logic block $M'_a$ in $C'$ are guaranteed to have completely the same functionality if the outputs of $M_a$ are completely specified for the input domain $2^n$, where $n$ is the number of inputs of $M'_a$. For such a completely specified $M_a$, we can know the ability of propagating transitions through $M'_a$ at RTL completely by analyzing $M_a$. However, $M_a$ and $M'_a$ are not guaranteed to have the same functionality (the functionality of $M_a$ is a proper subset of that of $M'_a$) if the outputs of $M_a$ are incompletely specified. For some input sequence, $p$ may not satisfy some of the three conditions of Theorem 1 if there exists an incompletely specified $M_a$ that influences $p$. The influence means that unknown/unspecified values generated from $M_a$ affect to the values captured into $Rs$ or propagating values on $p$ or propagated values from $Re$. In such a case, $p$ cannot be RTL false. In the following discussion, without losing generality, suppose that no incompletely specified RTL module that influences $p$ exists.

Let $I_1$, $I_2$ and $I_3$ be sets of the input sequences that satisfy Conditions 1, 2 and 3 of Theorem 1, respectively. The set of input sequences $I_1 \cup I_2 \cup I_3$ contains all the input sequences.

*Condition 1:* Any input sequence $u \in I_1$ does not launch any transition at the starting register $Rs$ on $p$. Since there is no incompletely specified combinational module that affects $Rs$, $u$ does not launch any transition at $\forall f \in \tau(Rs)$, which is the starting point of $\forall g \in \delta(p)$.

*Condition 2:* For $\forall v \in I_2$, propagation of any transition launched at the starting point of $p$ is prevented at some module $M_c$ on $p$. Since there is no incompletely specified combinational module in the sub-circuit that affects off-inputs of $p$, propagation of any transition on $\forall g \in \delta(p)$ is also prevented at the logic block corresponding to $M_c$.

*Condition 3:* For $\forall w \in I_3$, propagation of any value captured into the ending register $Re$ on $p$ is prevented at some module $M_d$ or some register $R_d$ on every sequential path from $Re$ to every primary output. Since there is no incompletely specified combinational module that affects propagation of the captured value on $Re$ to the primary outputs, propagation of the value of $\forall h \in \tau(R_e)$ is also prevented at the logic block corresponding to $M_d$ or at $\forall y \in \tau(R_d)$.

For any input sequence, no transition is launched at the starting point of $\forall g \in \delta(p)$ or no transition is propagated to the ending point along $\forall g \in \delta(p)$ or no value captured into $\forall h \in \tau(R_e)$ is propagated to any primary output. Thus $\forall g \in \delta(p)$ is gate-level false. Therefore, $p$ is RTL false if at least one of the three conditions of Theorem 1 is satisfied for any input sequence. □

### 3.2. Control-dependent false path

A path to transfer data from one register to another is determined by the select signal of each MUX on the path. The timing of data transfer between registers is determined by the load-enable signal of each register. In this section, from the information on control signals, we show a sufficient condition for identifying RTL false paths. An RTL path identified from the conditions is referred to as a control-dependent false path (CFP).

The state register (SR) in a controller represents states of the controller. Control signals for each state and a next state are determined by the value of the SR and status signals from a datapath and/or the PI. We distinguish an RTL path starting at the SR in a controller from the other RTL paths. By considering state assignment, we can know the timing when transitions are launched for each bit of the SR and the directions of the transitions. Therefore we take the information about the state assignment and the directions of the transitions into account for identifying RTL false paths starting at the SR. A register in a datapath is referred to as a datapath register (DR).

**RTL paths starting at DR/PI**

Let $P$ be a set of RTL paths that starts at a DR or a PI. Now we consider whether $p \in P$ is a CFP or not. Let $Rs$ and $Re$ be the starting register and the ending register of

$p$, respectively. Let $C_{Rs}$ and $C_{Re}$ be load-enable signals of $Rs$ and $Re$, respectively. If the load-enable signal of a register is equal to '1', the register loads a value; otherwise, it holds its value. Note that if the register does not have a hold function, we assume that the register has a load enable signal line and the value of that signal is always '1'. If the starting point of $p$ is a PI or the ending point of $p$ is a PO, the PI and the PO are treated as a register with no hold function. Let $M_i$ and $C_{M_i}$ $(1 \leq i \leq n)$ be a MUX on $p$ and its select signal, respectively, where $n$ is the number of MUXs on $p$. Let $C_M^k$ be the control value of $M$ at time $k$. When $M_i$ selects the input on $p$, the value of the select signal is denoted as $p_{M_i}$. For example, suppose that $p$ is the RTL path $R_1$-$M_1$-$ALU$-$M_2$-$R_3$ in Figure 1. When $M_1$ and $M_2$ select $p$, $p_{M_1} = 0$ and $p_{M_2} = 0$.

In the following discussion, we consider identification of non-robust untestable paths. Under single fault assumption (a single path is only affected by delay), the delay on a non-robust untestable path does not affect the circuit performance. In this paper, a gate-level path is said to be false if the path is non-robust untestable. If we relax the single fault assumption, the delay on a non-robust untestable path may affect the circuit performance depending on the delays on its off-paths. Our path identification method can also deal with functionally unsensitizable paths by simple extension of the sufficient condition for non-robust untestable.

We first consider uncontrollability and unobservability of registers. Then we show a sufficient condition of control signals to identify RTL false paths in Lemma 1.

**Definition 4: (Uncontrollability of register $R$ at time $t$)** Let $q$ be an RTL path whose ending register is $R$. For each state at time $t-1$, if both of the following conditions are satisfied, $R$ is uncontrollable at time $t$.

- For each $q$, control signal values of all the MUXs on $q$ at time $t$ and those at time $t-1$ are the same.
- Each source register $R_q$ of $R$ is uncontrollable at time $t-1$ or $C_{R_q}^{t-1} = 0$, where a source register of $R$ at time $t$ is a register whose value affects $R$ at time $t$. □

An example of source registers is the following: Reg1 and Reg4 are the source registers of Reg2 at time $k$ in Figure 3. Reg3 is not a source register because the data transfer is prevented at the MUX at time $k$.

**Definition 5: (Unobservability of register $R$ at time $t$)** Let $r$ and $Rr$ be an RTL path starting at $R$ and its ending register, respectively. For each state at time $t+1$, if every $r$ satisfies at least one of the following conditions, $R$ is unobservable at time $t$.

- $\exists i | 1 \leq i \leq n, C_{M_i}^{t+1} \neq r_{M_i}$, where $n$ is the number of MUXs on $r$.
- $C_{Rr}^{t+1} = 0$.
- $Rr$ is unobservable at time $t+1$. □

**Lemma 1:** An RTL path $p$ is RTL false if at least one of the following three conditions is satisfied for any state transition from time $k$ to $k+1$.

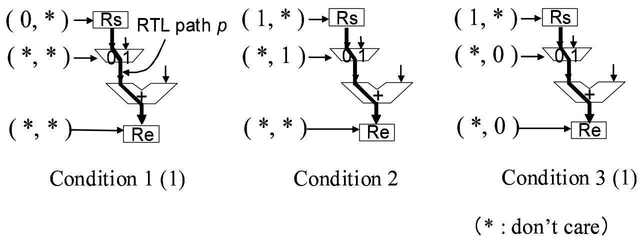


**Figure 2. Examples of Conditions 1(1), 2, and 3(1) of Lemma 1.**

*Condition 1:* (1) $C_{Rs}^k = 0$ or (2) $Rs$ is uncontrollable at time $k$.
*Condition 2:* $\exists i | 1 \leq i \leq n, C_{M_i}^{k+1} \neq p_{M_i}$, where $n$ is the number of MUXs on $p$.
*Condition 3:* (1) $C_{Re}^{k+1} = 0$ or (2) $Re$ is unobservable at time $k+1$. □

**Proof:** We show that Lemma 1 is properly included in Theorem 1. For any input sequence with any initial state, the input sequence causes at most all the state transitions considered in Lemma 1. If any state transition satisfies at least one of the conditions of Lemma 1, then the input sequence also satisfies at least one of the conditions of Lemma 1.

If Condition 1(1) is satisfied for a state transition, $Rs$ holds a value. If Condition 1(2) is satisfied for the state transition, a value is not changed at $Rs$ although $Rs$ loads a new value. For both the conditions, no transition is launched at $Rs$, which is the starting register of $p$. If Condition 1 of Lemma 1 is satisfied for the state transition, Condition 1 of Theorem 1 is also satisfied for the state transition. If Condition 2 of Lemma 1 is satisfied for the state transition, $M_i$ prevents the propagation of a transition. Thus, for the state transition, Condition 2 of Theorem 1 is also satisfied. If Condition 3(1) of Lemma 1 is satisfied for the state transition, no transition is captured to $Re$. If Condition 3(2) is satisfied for the state transition, no captured value at $Re$ is propagated to any primary output. Thus, for the state transition, Condition 3 of Theorem 1 is also satisfied.

At least one of the three conditions of Theorem 1 is satisfied for any input sequence if at least one of the three conditions of Lemma 1 is satisfied for any state transition. Therefore, Lemma 1 are properly included in Theorem 1.□

Any gate-level path $g \in \delta(p)$ is non-robust untestable if $p$ is RTL false. The second condition of Lemma 1 means that at least one MUX on $p$ does not select $p$ at time $k+1$. If the condition is extended such that at least one MUX on $p$ does not select $p$ at time $k$ and $k+1$, any gate-level path corresponding to $p$ is functionally unsensitizable.

Conditions 1(1), 2 and 3(1) of Lemma 1 show the conditions of control signals for two time frames $k$ and $k+1$. Examples of them are shown in Figure 2. If Condition 1(1) of Lemma 1 is satisfied for a state transition, no transition is launched at $Rs$ at time $k$. If Condition 2 is satisfied for

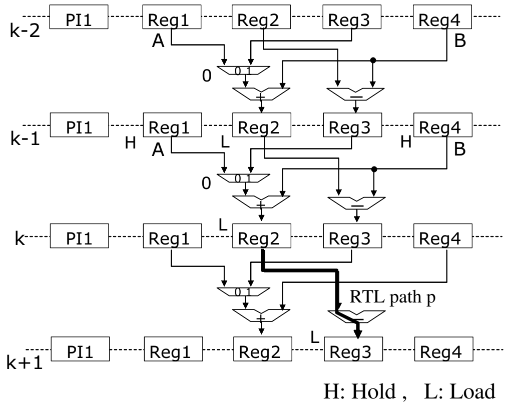


**Figure 3. An example of Condition 1(2) of Lemma 1.**

the state transition, no transition at the starting point of $p$ is propagated to the ending point along $p$ at time $k+1$. If Condition 3(1) is satisfied for the state transition, no value is captured to *Re* at time $k+1$. No transition is launched at *Rs* or no transition at *Rs* is ever captured into *Re* if one of the three conditions is satisfied for any state transition.

Conditions 1(2) and 3(2) are the conditions for the upstream and downstream of the path, respectively. The uncontrollable condition of Condition 1(2) is that the captured value of *Rs* at time $k$ is the same as the previous value of *Rs* even if the load-enable signal of *Rs* is 'load' at time $k$; that is, there is no transition at *Rs*. The unobservable condition of Condition 3(2) is that the captured value of *Re* at time $k+1$ is not propagated to any PO even if the load-enable signal of *Re* is 'load' at time $k+1$. We show examples of these conditions in Figures 3 and 4, which are time expansion models of an RTL circuit.

In Figure 3, the target RTL path $p$ is Reg2-Sub-Reg3. Reg2 loads a new value 'A+B' at time $k$. However, the value is the same as the previous value that Reg2 loaded at time $k-1$ because the select signal of the MUX is the same at time $k$ and $k-1$. Moreover, Reg1 and Reg4 hold their values at time $k-1$. This is one of the cases where Condition 1(2) is satisfied. If a control signal of a source register at time $k-1$ is 'load', we will recursively check whether the value captured into the source register is the same as the previous one.

In Figure 4, the target RTL path $p$ is Reg2-MUX-Add-Reg3. The transition launched at Reg2 is captured to Reg3 at $k+1$. However, the value cannot propagate to any PO because Reg1 does not capture the value at time $k+2$. If Reg1 captures the value, we will recursively check the next time frame at time $k+3$. For the other path from Reg3 to PO1, since the MUX on the RTL path does not select the path at $k+1$, propagation of the value is prevented by the MUX.

**RTL paths starting at SR-ff**

For RTL paths from flip-flops in the SR (SR-ff), Lemma 1

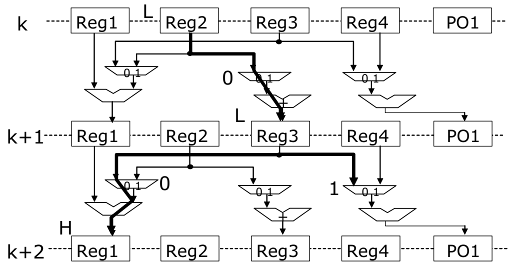


**Figure 4. An example of Condition 3(2) of Lemma 1.**

can also be applied. The SR in a controller uploads a new value every clock cycle. It means that $C_{Rs}$ always becomes 1 (load). Therefore RTL paths from the SR-ff to DRs do not satisfy Condition 1 (1) of Lemma 1. Here we consider transitions from each SR-ff. The relation between states and values of the flip-flops is determined by state assignments. We can obtain the information on state assignments during logic synthesis or designers can also determine state assignments before logic synthesis. From the information on state assignments and state transition, we can know the timing when a transition is launched at each flip-flop. For example, let us consider state assignments to the controller in Figure 1. We assume that the SR in the controller consists of two flip-flops $(FF0, FF1)$, and $(0,0)$, $(0,1)$ and $(1,1)$ are assigned to $S_0$, $S_1$ and $S_2$, respectively. When $S_0$ transfers to $S_1$, $FF1$ has a rising transition.

An RTL path is false with respect to a rising (resp. falling) transition if at least one of the three conditions of Lemma 1 is satisfied for all the time $t$ when a rising (resp. falling) transition launches at the SR-ff of the starting point of the RTL path.

### 3.3. Experimental results

In this section, we evaluate the effectiveness of identifying a subset of RTL false paths as control-dependent false paths (CFPs). The circuit characteristics of RTL benchmarks are shown in Table 1. LWF, Tseng, Paulin and JWF are widely used benchmark circuits. MPEG and RISC[2] are more practical and larger circuits provided by industry. Each circuit consists of a controller and a datapath. The first eight columns show circuit name, the bit width of the datapath, the numbers of the PIs, the POs, the registers and the states in the controller, and the total area of the circuit, respectively. Logic synthesis was performed by DesignCompiler (Synopsys). The last two columns show the number of RTL paths. Columns DR and SR-ff show the number of RTL paths where an RTL path starts at a datapath register or a PI and the number of RTL paths where an RTL path starts at an FF in the SR, respectively.

Table 2 shows the results for the number of RTL paths

[2]These circuits were provided for the Joint Research (1997-2001) with Semiconductor Technology Academic Research Center (STARC).

identified as CFPs. The second, third and fourth columns under DR show the number of RTL paths identified as CFPs starting at DR, the number of RTL paths starting at DRs, and the ratio of the number of CFPs to that of RTL paths, respectively. For LWF, Tseng, Paulin, JWF and MPEG, CPU times required for identifying CFPs were less than 1 second. For JWF circuit, many RTL paths (117 of 153) were identified as CFPs. MPEG has many registers with no hold function. Therefore, many starting registers launch transitions and many ending registers capture the propagated transitions every clock cycle. Hence the number of CFPs is small. For RISC, the CPU time required for identifying 1,235 CFPs of 10,181 RTL paths is about 10 seconds. The next three columns and the last three columns show results for RTL paths starting at SR-ffs with rising transitions and starting at SR-ffs with falling transitions, respectively. For all the circuits except for RISC, the ratios of RTL paths identified as CFPs are similar to those of RTL paths starting at DRs. For RISC, a large number of RTL paths are identified.

Moreover, we evaluated the number of RTL paths identified as CFPs when only Conditions 1(1), 2 and 3(1) of Lemma 1 are applied, where the conditions are for identifying CFPs within two time frames. For Tseng, the number of RTL paths identified as CFPs, which is starting at DR, was reduced from 8 to 6. However the other results of the number of RTL paths identified as CFPs was not changed. It means that almost all the CFPs that were identified by considering all the conditions of Lemma 1 were identified. For very large scale circuits, identification based on only two time frames may be efficient to reduce the time required for finding CFPs.

Table 3 shows the result of gate-level false paths corresponding to RTL paths identified as CFPs. We extracted gate-level paths from the longest path, which has larger propagation delay, by using the "report path" function of the timing analysis tool Prime Time (Synopsys). For LWF, Tseng, Paulin and JWF, we extract all the paths in each circuit. For MPEG and RISC, we extract 300,000 and 200,000 gate-level paths from the longest path, respectively. The second column shows the number of gate-level paths corresponding to CFPs. We can say that these gate-level false paths were identified by our method. The third column shows the total number of gate-level paths starting at DRs. The fourth column shows the ratio of the second column to the third column. For Paulin, our method identified 10,598 of 33,476 (32%) gate- level paths as false paths within 1 second. We perform sequential test generation (TetraMax, Synopsys) for only 100 paths in Paulin, then it took 498 second to identify 51 paths as untestable. If an RTL path that passes a large scale operational module such as a multiplier is identified as a CFP, a large number of gate-level paths are identified as false paths. For MPEG, most of the extracted 300,000 paths start at DRs. Gate-level paths corresponding to CFPs starting at DRs were not included among the extracted paths. One reason is that gate-level paths corresponding to the identified CFPs have short propagation delays, hence they were not included among the extracted paths. As another reason, MPEG may have a small number of false paths.

**Table 1. Circuit characteristics of benchmarks.**

| Circuit | Bit width | #PIs | # POs | # REGs | # States | Area | # RTL paths DR | # RTL paths SR-ff |
|---|---|---|---|---|---|---|---|---|
| LWF | 8 | 3 | 2 | 6 | 4 | 1,561 | 19 | 26 |
| Tseng | 8 | 4 | 3 | 7 | 5 | 2,975 | 20 | 42 |
| Paulin | 8 | 3 | 2 | 8 | 6 | 3,391 | 29 | 67 |
| JWF | 8 | 6 | 5 | 15 | 8 | 4,758 | 153 | 408 |
| MPEG | 8 | 7 | 16 | 241 | 163 | 77,554 | 651 | 2,152 |
| RISC | 32 | 1 | 3 | 39 | 10 | 97,739 | 10,181 | 38,122 |

## 4. Reduction in Over-testing

To reduce over-testing, we consider that PDFs on paths identified as false are excluded from a fault list targeted by test generation, and then test generation is performed for the fault list. However, during test application, patterns generated by the test generation may still detect the excluded faults accidentally. This is because a generated pattern to detect some PDF in the fault list may also activate other PDFs simultaneously. In this work, we evaluate such circumstance. In this experiment, we adopt an enhanced scan technique as a DFT method. Table 4 shows the results of over-tested PDFs for each benchmark circuit. Fault lists are generated by using the "write path" function of Prime Time in order to perform test generation and fault simulation. The second column shows the total number of PDFs. The third column shows the number of PDFs on CFPs. We generate test patterns for fault lists excluding PDFs on CFPs, then perform fault simulation for fault lists including the total PDFs. The fourth column shows the number of tested PDFs among PDFs shown in the third column. Our experimental results show that 10-60% of the removed PDFs is accidentally tested (over-tested) if we generate two-pattern tests with no input constraint.

To avoid the over-testing, we use constrained two-pattern tests called single-port-change (SPC) two-pattern tests[14]. The concept of SPC two-pattern tests was originally proposed by authors to reduce area overhead required for DFTs compared to that required for DFTs by our previous work[15]. We can utilize the concept for avoiding the over-testing completely. An SPC two-pattern test changes the second vector at only one port and sets stable for the other ports. Here a port means an output of a register, and it has bit width. An SPC two-pattern test can test PDFs on paths starting at the port whose second vectors are changed. With respect to test quality, SPC two-pattern tests guarantee robust (resp. non-robust) test for a PDF if the PDF is robust (resp. non-robust) testable[14]. In terms of reduction in over-testing, when SPC two-pattern tests are generated

Table 2. Number of RTL paths identified as CFPs.

| | DR | | | SR-ff: Rise | | | SR-ff: Fall | | |
|---|---|---|---|---|---|---|---|---|---|
| Circuit | #CFP | #RTL path | ratio | #CFP | #RTL path | ratio | #CFP | #RTL path | ratio |
| LWF | 3 | 19 | 16 % | 4 | 26 | 15 % | 4 | 26 | 15 % |
| Tseng | 8 | 20 | 40 % | 13 | 42 | 31 % | 11 | 42 | 26 % |
| Paulin | 12 | 29 | 41 % | 25 | 67 | 37 % | 30 | 67 | 45 % |
| JWF | 117 | 153 | 76 % | 285 | 408 | 70 % | 319 | 408 | 78 % |
| MPEG | 32 | 651 | 5 % | 64 | 2,152 | 3 % | 64 | 2,152 | 3 % |
| RISC | 1,235 | 10,181 | 12 % | 28,411 | 38,122 | 75 % | 18,968 | 38,122 | 50 % |

Table 3. Number of gate-level paths corresponding to CFPs.

| | DR | | | SR-ff: Rise | | | SR-ff: Fall | | |
|---|---|---|---|---|---|---|---|---|---|
| Circuit | #GL false | #GL path | ratio | #GL false | #GL path | ratio | #GL false | #GL path | ratio |
| LWF | 314 | 2,180 | 14 % | 330 | 1,461 | 23 % | 350 | 1,461 | 24 % |
| Tseng | 2,337 | 5,192 | 45 % | 142 | 414 | 34 % | 236 | 414 | 57 % |
| Paulin | 10,598 | 33,476 | 32 % | 10,387 | 26,898 | 39 % | 12,197 | 26,898 | 45 % |
| JWF | 9,760 | 32,522 | 30 % | 36,331 | 49,094 | 74 % | 41,018 | 49,094 | 84 % |
| MPEG | 0 | 257,552 | 0 % | 922 | 39,398 | 2 % | 192 | 3050 | 6 % |
| RISC | 1,76 | 174,063 | 1 % | 1,375 | 6,396 | 21 % | 9,882 | 19,541 | 51 % |

Table 4. Number of over-tested PDFs.

| Circuit | Total #PDFs | #PDFs on CFPs | #Over-tested PDFs |
|---|---|---|---|
| LWF | 3,598 | 619 | 392 (63%) |
| Tseng | 5,868 | 2,928 | 1,185 (40%) |
| Paulin | 37,171 | 19,477 | 2,195 (11%) |
| JWF | 45,658 | 37,150 | 3,452 (9%) |
| MPEG | 41,521 | 340 | 220 (65%) |
| RISC | 25,496 | 6,974 | 509 (7.3%) |

for fault lists excluding PDFs corresponding to CFPs, the removed PDFs are never tested.

## 5. Conclusion

In this paper, we have introduced a concept of register-transfer level (RTL) false paths and proposed a method of identifying them. Our experimental results showed that a large number of gate-level false paths were found by identifying RTL false paths. Time required for the identification was much shorter than that for the identification at gate-level. To reduce over-testing, path delay faults (PDFs) on the identified false paths are excluded from the fault list targeted by test generation. However even if the PDFs are removed from the fault list, the generated test patterns incidentally test the removed faults. Therefore, we have also proposed a method to avoid over-testing by generating single-port-change two-pattern tests. In the experimental results, we showed that 10-60% of removed PDFs is over-tested if we normally generate unconstrained two-pattern tests.

**Acknowledgment**

The authors would like to thank Profs. Michiko Inoue and Tomokazu Yoneda of Nara Institute of Science and Technology for their valuable discussion and their cooperation. This work was supported in part by Semiconductor Technology Academic Research Center (STARC) under the Research Project, in part by 21st Century Center of Excellence (COE) Program (Ubiquitous Networked Media Computing), and in part by Japan Society for the Promotion of Science (JSPS) for Young Scientists (B) (No.17700062).

## References

[1] P. Maxwell, "The design, implementation and analysis of test experiments," Proc. International Test Conference, pp.1–9, Nov. 2006.

[2] A. Krstic and K.T. Cheng, Delay Fault Testing for VLSI Circuits, Kluwer Academic Publishers, 1998.

[3] B.I. Devadas and G.E. Stong, "Design for testability using scanpath techniques for path delay test and measurement," Proceeding of International Test Conf., pp.365–374, 1991.

[4] J. Savir and S. Patel, "Scan-based transition test," IEEE Trans. on CAD, vol.12, no.8, pp.1232–1241, Aug. 1993.

[5] J. Savir and S. Patel, "Broad-side delay test," IEEE Trans. on CAD, vol.13, no.8, pp.1057–1064, Aug. 1994.

[6] K.T. Cheng and H.C. Chen, "Classification and identification of non-robust untestable path delay faults," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.15, no.8, pp.854–853, Aug. 1996.

[7] S. Kajihara, K. Kinoshita, I. Pomeranz, and S.M. Reddy, "A method for identifying robust dependent and functionally unsensitizable paths," Proc. Int. Conf. on VLSI Design, pp.82–87, 1997.

[8] S.M. Reddy, S. Kajihara, and I. Pomeranz, "An efficient method to identify untestable path delay faults," IEEE the 10th Asian test symposium, pp.233–238, Nov. 2001.

[9] A. Krstic, S.T. Chakradhar, and K.T. Cheng, "Testable path delay fault cover for sequential circuits," Proc. European Design Automation Conference, pp.220–226, Sep. 1996.

[10] R. Tekumalla and P.R. Menon, "Identifying redundant path delay faults in sequential circuits," Proc. 9th International Conf. VLSI Design, pp.406–411, Jan 1996.

[11] M. Nourani and A. Papachristou, "False path exclusion in delay analysis of RTL structures," IEEE Trans on VLSI Systems, vol.10, no.1, pp.30–43, Feb. 2002.

[12] Y Explorations, Inc., Explorations tool, http://www.yxi.com/index.html.

[13] G.D. Micheli, Synthesis and Optimization of Digital Circuit, McGraw-Hill, 1994.

[14] Y. Yoshikawa, S. Ohtake, M. Inoue, and H. Fujiwara, "Design for testability based on single-port-change delay testing for data paths," IEEE the 14th Asian Test Symposium, pp.254–259, Dec. 2005.

[15] M.A. Amin, S. Ohtake, and H. Fujiwara, "Design for hierarchical two-pattern testability of data paths," IEEE the 10th Asian test symposium, pp.11–16, Nov. 2001.