NAIST-IS-DD0561036

Doctoral Dissertation

Studies on Design for Delay Testability and Over-testing Reduction for Delay Faults

Yuki Yoshikawa

March 23, 2007

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Yuki Yoshikawa

Studies on Design for Delay Testability and Over-testing Reduction for Delay Faults[*]

Yuki Yoshikawa

Abstract

LSIs are embedded in various digital systems. Testing of LSIs has an important role in order to realize dependable digital systems. Especially, delay testing is emphasized to guarantee the timing correctness of circuits in addition to testing to guarantee logical correctness. In semiconductor industry, scan techniques are generally used as a design-for-testability (DFT) to reduce some test costs. As a previous work, Amin et al. have proposed a non-scan DFT method that makes test application time shorter than that of scan techniques. However, area overhead caused by the DFT method is still large. In this dissertation, we propose a non-scan DFT method that can reduce area overhead while keeping the test quality. Experimental results show that area overhead of our proposed DFT method becomes about half of that required for the previous DFT method.

In a circuit, there are a lot of untestable faults in normal operation, which never affect the performance of the circuit even if there exist on. DFT techniques augment the circuit into easily testable one, however they also make a number of originally untestable faults testable. Testing such faults is called over-testing and it induces yield loss. We address reduction in the over-testing by identifying a subset of untestable faults using register transfer information, and also propose a method to avoid testing the faults identified as untestable. Experimental results show that our path identification method can identify many faults as untestable in reasonable time.

Keywords:

LSI testing, Delay fault, Design-for-testability, Over-testing, False path

[*] Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0561036, March 23, 2007.

List of Publications

Journal Paper

1. Yuki Yoshikawa, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, “Non-scan Design for Single-Port-Change Delay Fault Testability,” Information Processing Society of Japan, Vol. 47, No. 6, pp. 1619-1628, June 2006.

International Conferences

1. Yuki Yoshikawa, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, “Design for Testability Based on Single-Port-Change Delay Testing for Data Paths,” IEEE the 14th Asian Test Symposium (ATS), pp. 254-259, Dec. 2005.

2. Yuki Yoshikawa, Satoshi Ohtake and Hideo Fujiwara, “An Approach to Reduce Over-Testing of Path Delay Faults in Data Paths Using RT-level Information,” Digest of Papers, IEEE the 11th European Test Symposium (ETS), pp. 146-151, May 2006.

Technical Reports

1. Yuki Yoshikawa, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, “Design for Testability Based on Single-Port-Change Delay Testing for Data Paths,” Technical Report of IEICE (DC2004-58), Vol. 104, No. 478, pp.73-78, Fukuoka, Dec. 2004. (In Japanese)

2. Yuki Yoshikawa, Satoshi Ohtake, Michiko Inoue and Hideo Fujiwara, “Design for Testability Based on Single-Port-Change Delay Testing for Data Paths,” 52th FTC Workshop, Jan. 2005. (In Japanese)

3. Yuki Yoshikawa, Satoshi Ohtake and Hideo Fujiwara, “Reduction in Over-Testing of Delay Faults through False Paths Identification Using RTL Information,” Technical Report of IEICE (DC2006-87), Vol. 106, No. 528, pp.43-48, Feb. 2007.

Award

1. IEEE Kansai Section Student Paper Award, Feb. 2007.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, LSIs (Large scale Integration, LSI) are embedded in various digital systems and IT equipment represented by a mobile phone and a personal computer is widely spreading in our daily life. Although low cost, small size and low power consumption are important factors of LSI circuits, to guarantee the correct behavior of the circuit is more important, thus testing of LSIs is very important technique. LSI testing is to check whether there exist faults in a circuit or not, and it consists of two processes. One is test generation and the other is test application. Test generation is that an input sequence to detect a fault is generated. Test application is that the generated input sequence is applied to inputs of the circuit and the responses are compared with expected values.

With the progress of LSI design and manufacturing technologies, the speed of circuits has increased in recent years. For high speed circuits, delay testing is emphasized to guarantee the timing correctness of circuits in addition to testing to guarantee logical correctness. In order to model a delay defect in a circuit, there are some delay fault models[1]. A transition fault model and a path delay fault model are often used. Under the transition fault model, the extra delay caused by the fault is assumed to be large enough to prevent a signal transition (0 to 1, or 1 to 0) from reaching any observation point, which is a flip-flop or a primary output, at the time of observation. The main advantage of the transition fault model is that the number of faults in a circuit is linear in terms of the number of gates. On the other hand the path delay fault model, which is our target model, considers propagation delay of a transition along a path and can deal with accumulation of small delay on the path. In a sequential circuit, a path corresponds to an ordered set of gates between two flip-flops. Testing a path delay fault on a path detects a defect that induces a propagation delay beyond a specified clock period. Note that the number of paths in a circuit can be very large (possibly exponential in the number of gates). Hence, the path delay fault model is often used for a set of paths which are selected by some strategy, such as selecting paths with expected delays greater than a specified threshold. To test a path delay fault, consecutive two vectors are applied to FFs that are the starting points of the target path and other related paths, and the response has to be observed. However, test generation for sequential circuits is generally hard task. For such sequential circuits, design for testability (DFT) is an important approach to reduction in the test generation complexity. In this dissertation, the first work is to propose a DFT method with low area overhead and high quality.

Scan techniques are generally used as a DFT method in semiconductor industry because they are simple and can easily apply the techniques to circuits of complex structure. A fully enhanced scan technique has been proposed as a straightforward DFT method in order to apply any two vectors to each FF[5]. In this design, every flip-flop (FF) is replaced by an enhanced scan FF. Each enhanced scan FF can store arbitrary two bits. For a sequential circuit designed by this technique, we can apply any consecutive two vectors to its combinational logic part. However, area overhead caused by enhanced scan FFs is very high. The functional justification[4] and the scan shift[3] techniques with standard scan also have been proposed. Area overhead of these techniques is smaller than that of enhanced scan technique because every FF is replaced by standard scan FF that can store arbitrary one bit. However, any consecutive two vectors cannot be applied to each FF. Both enhanced scan and standard scan approaches have the following disadvantages, (1) long test application time because of scan-shift operation, (2) test application at the rated speed of a given circuit, called at-speed testing, cannot be performed.

To improve the disadvantages (1) and (2), non-scan DFT approaches at register transfer level (RTL) have been proposed[8]. This DFT method bases on hierarchical test generation[7]. An RT-level circuit generally consists of a controller represented by a finite state machine, and a data path represented by hardware elements (e.g., registers, multiplexers and operational modules), and lines to connect them. Hierarchical test generation consists of two processes, (i) generating test patterns at gate-level for each combinational block such as an operational module, (ii) generating control paths at RTL to justify the generated patterns from primary inputs (PIs) to registers which are inputs of each combinational block, and also generating observation paths at RTL to propagate the responses to primary outputs (POs). Amin et al, defined hierarchically two-pattern testable (HTPT) data path, in which any two-pattern tests can be applied to every combinational block from PIs and the responses can be observed at POs. Also they proposed a DFT method to augment a given data path to an HTPT data path[8]. The DFT method requires lower area overhead and shorter test application time than enhanced scan approach does.

In this dissertation, we present a new testability called single-port-change (SPC) two-pattern testability and propose a non-scan DFT method that makes every RTL path SPC two-pattern testable. An RTL path is a path passing through only combinational logic, which starts at a PI or a register and ends at a register or a PO. For a target RTL path of testing in a combinational block, an SPC two-pattern test launches transitions at the starting points of paths corresponding to the target RTL path while keeping the other related ports stable. During justification of the SPC two-pattern tests from PIs to the combinational block, the original hold function of a register can be used for stable inputs if the hold function exists on its control path. Hence, our proposed DFT method can reduce area overhead than that of HTPT supporting arbitrary two-pattern tests. According to the quality of two-pattern tests, testable path delay faults are generally classified into three classes: robust testable, non-robust testable and functional sensitizable (FS)[1]. SPC two-pattern tests can guarantee robust (resp. non-robust) test for a path if the path is robust (resp. non-robust) testable and can also test a subset of FS path delay faults.

DFT techniques are mentioned above. DFT techniques are surely useful in order to reduce test generation complexity, but on the other hand they induce over-testing. Our second work is to reduce the over-testing caused by the DFT techniques. It is generally known that a large number of paths in a circuit are false paths. A false path is a path that transition is never launched at the starting point of the path or launched transition is never propagated to the ending point along the path. For an original circuit before some DFT is done, it is impossible to test a path delay fault on a false path because any vector pair cannot activate the path delay fault under normal functions of the circuit. A DFT technique makes such untestable path delay faults testable by adding a mechanism to apply test vectors from outside. However the path delay faults that became testable do not affect the circuit performance even if they exist because they were originally inactive faults in normal operations. Therefore we consider that testing such path delay faults is over-testing. The over-testing induces yield loss because there is a possibility that a good circuit is judged as a faulty one. Moreover it makes test generation time and test application time long.

To reduce the over-testing, it is necessary to identify originally untestable path delay faults in a circuit and exclude them from the target of testing. In recent ten years, several path identification methods have been proposed. For combinational circuits, works presented in [22, 23, 24] are efficient approaches to identifying combinationally untestable path delay faults at gate-level. For sequential circuits, Krstic et al, proposed a method to find sequentially untestable path delay faults using time expansion model at gate-level [18]. For small scale circuits the method succeeded to identify sequentially untestable path delay faults within practical time. However it is conceivable that the method takes long CPU time for large scale circuits. In [13, 25], identification methods of multi-cycle paths at gate-level are proposed. The work presented in [26] identified false paths at RTL. However their interest is to prevent inaccurate decision of circuit performance caused by identifying false paths as critical paths, hence they do not consider correspondence between a path at RTL and a path at gate level.

In this dissertation, we propose a method of identifying a large number of sequentially untestable paths and multi-cycle paths using register-transfer level (RTL) information such as load-enable signals of registers and select signals of multiplexers (MUXs). The target path of this method is an RTL path which is a bundle of gate-level paths between two registers. The total number of RTL paths in a circuit is much smaller than the total number of gate-level paths, therefore our method can be performed in reasonable time.

The rest of this dissertation is organized as follows. Chapter 2 gives the basics of delay testing and path delay fault. In Chapter 3, we proposed a non-scan DFT method at RTL based on single-port-change two-pattern test. Experimental results show that our proposed method can reduce area overhead to about half compared to that of the non-scan DFT method for HTPT datapaths. In Chapter 4, we propose a path identification method using RTL information and also address a method to guarantee that the identified paths are never tested. In Chapter 5, this work is concluded and discusses directions for future work.

Chapter 2

Preliminaries

1. Delay test

To observe delay defects it is necessary to launch and propagate transition in a circuit. Launching transition requires application of a vector pair, $V = (v_1, v_2)$. The first vector v_1 initializes a given circuit while the second vector v_2 causes the desired transitions. Figure 2.1 illustrates the test application scheme for combinational circuits. In normal operation, only one clock (system clock) is used to control the input and output latches and its period is T_C. During test mode, the input and output latches are controlled by two different clocks: the input and output clocks, respectively. It is assumed that the period of these clocks, T_S, is larger than T_C. The input and output clocks are skewed by an amount equal to T_C. At time t_0 and t_1, v_1 and v_2 are applied to the primary inputs, respectively. Time $T_S = t_1 - t_0$ is assumed to be sufficient for all signals in the circuit to stabilize under the first vector v_1. After applying the second vector v_2, the circuit is allowed to settle down only until time t_2, where $t_2 - t_1 = T_C$. At time t_2, the primary output values are observed and compared to an expected value of a fault-free circuit to determine if there is a delay fault.

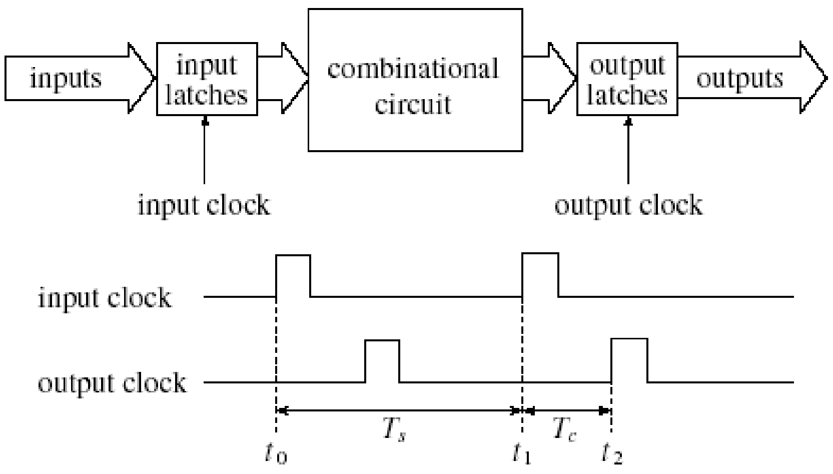

Figure 2.1. Delay test for combinational circuits.

2. Path delay fault

In this section, we show a path delay fault model and the classification. A path delay fault is classified according to their testability: robust testable, non-robust testable, functionally sensitizable and functionally unsensitizable.

Definition 1: (path, path delay fault)

A path p is an ordered set of gates $p = \{g_0, g_1, ..., g_n\}$, where g_0 is a primary input or a flip-flop and g_n is a primary output or a flip-flop. Also $g_i (1 \leq i \leq n-1)$ is a gate.

A fault on p is said to be a path delay fault if a transition launched at g_0 does not propagate to g_n within a specified limit. A path delay fault on p has two types, where a rising transition is launched at g_0 and a falling transition is launched at g_0. □

Definition 2: (Controlling value，Non-controlling value)

An input to a gate is said to have a controlling value (cv) if it determines the value of the gate output regardless of the values on the other inputs to the gate.

An input to a gate is said to have non-controlling value (*ncv*) if the values on the other inputs to the gate determines the value of the gate output. □

Definition 3: (On-input, Off-input)

An input is an on-input of path p if it is on p. An input is an off-input of path p if it is an input to a gate on p but it is not an on-input. □

2.1 Classification of path delay fault

Definition 4: (Robust off-input)

Let f denote the on-input to gate g_i in the target path. Let h denote an off-input to gate g_i The off-input h is called robust off-input with respect to an input vector pair (v_1, v_2) if:

1. there is a $cv \rightarrow ncv$ transition or, stable ncv on h when f has a $cv \rightarrow ncv$ transition, and
2. there is a stable ncv on h when f has a $ncv \rightarrow cv$ transition □

Definition 5: (Robust testable path delay fault)

A path delay fault where there exists an input vector pair (v_1, v_2) such that it activates the required transitions on the path and all off-inputs in the path are robust off-input is called a robust testable path delay fault. The vector pair (v_1, v_2) can test the robust testable fault even if there exists a propagation delay between the starting point of other path and a off-input of p □

Definition 6: (Non-robust off-input)

Let f denote the on-input to gate g_i in the target path. Let h denote an off-input to gate g_i The off-input h is called non-robust off-input with respect to an input vector pair (v_1, v_2) if

- there is a $ncv \rightarrow cv$ transition on f and a $cv \rightarrow ncv$ transition on h □

Definition 7: (Non-robust testable path delay fault)

A robust untestable path delay fault where there exists an input vector pair (v_1, v_2) such that it activates the required transitions on the path and at least one off-inputs in the path are non-robust off-input while the rest of the off-inputs are robust is called a non-robust testable path delay fault. The vector pair (v_1, v_2) can test the non-robust testable fault if there does not exist a propagation delay between the starting point of other path and a off-input of p $\square$

Definition 8: Functionally sensitizable off-input

Let f denote the on-input to gate g_i in the target path. Let h denote an off-input to gate g_i The off-input h is called functionally sensitizable off-input with respect to an input vector pair (v_1, v_2) if

- there is a $ncv \rightarrow cv$ transition on both f and h $\square$

Definition 9: Functionally sensitizable path delay fault

A non-robust untestable path delay fault where there exists an input vector pair (v_1, v_2) such that it activates the required transitions on the path and at least one off-inputs in the path is functionally sensitizable off-input while the rest of the off-inputs are either robust or non-robust is called a functionally sensitizable path delay fault. The vector pair (v_1, v_2) can test the functionally sensitizable fault only if there exist a propagation delay between the starting point of other path and a off-input of p $\square$

Chapter 3

Non-scan Design for Single-Port-Change Delay Fault Testability

1. Introduction

With the progress of VLSI design and manufacturing technologies, the speed of VLSI circuits has increased in recent years. High speed circuits need delay testing to verify that a given logic operates correctly at the desired clock speed. There are several delay fault models[1]. A path delay fault can model cumulative propagation delay along a path. A path for a sequential circuit corresponds to an ordered set of gates between two flip-flops. Testing a path delay fault on the path detects a defect that induces a propagation delay beyond a specified clock period. To test a path delay fault, a vector pair (two-pattern test) is required for FFs that are the starting points of the target path and other related paths. However, it is impossible to apply any two-pattern tests to the starting points. To enhance two-pattern testability for FFs, there are the functional justification[4] and the scan shift[3] techniques with standard scan. However, these techniques cannot still guarantee the application of any two-pattern. The enhanced scan [5](ES) approach that can apply any two-pattern incurs high area overhead. Moreover, scan approaches cause long test application time because of scan-shift operation.

Non-scan design-for-testability (DFT) approaches at register-transfer level

(RTL) based on hierarchical test generation have been proposed[7, 8]. The approaches utilize the data flow at RTL to test a circuit. The advantages are that the number of primitive elements at RTL is much smaller than that at gate level, and a number of gate-level paths between two registers are regarded as a bundled path, which is called RTL path[8]. An RTL path is a path passing through only combinational logic, which starts at a primary input (PI) or a register and ends at a register or a primary output (PO). For example, $R1 - m1 - m2 - Add1 - R5$ in Figure 3.7 is one of the RTL paths in LWF circuit. Hierarchical test generation consists of two processes: (i) generating test patterns for combinational blocks at gate-level, (ii) generating control paths to justify the generated patterns from PIs to registers that are inputs of every combinational block, and generating observation paths to propagate the responses to POs. Amin et al, defined hierarchically two-pattern testable (HTPT) data path[8], in which any two-pattern tests can be applied to every combinational block from PIs and the responses can be observed at POs. We presented a DFT method to augment a given data path to an HTPT data path, which requires lower area overhead and shorter test application time than enhanced scan approach does.

In this dissertation, we introduce a testability called single-port-change (SPC) two-pattern testability. A port means an input or an output of a primitive element at RTL, and it has a bit width. We propose a DFT method that guarantees to make every RTL path SPC two-pattern testable. For a target RTL path passing through a combinational block, an SPC two-pattern test launches transitions at the starting points of paths corresponding to the target RTL path while keeping the other related ports stable. The method of generating SPC two-pattern tests for a combinational block is explained in Section 3.1, and how to generate control and observation paths is shown in Section 5. During test application for each combinational block, the original hold function of a register can be used for stable inputs if the hold function exists on its control path. Hence, our proposed DFT method can reduce area overhead than that of HTPT using arbitrary two-pattern tests. According to the quality of two-pattern tests, testable path delay faults are generally classified into three classes: robust testable, non-robust testable and functional sensitizable (FS)[1].SPC two-pattern tests can guarantee robust (resp. non-robust) test for a path if the path is robust (resp. non-robust) testable and can also detect a subset of FS path delay faults (shown in Section 3.2).

Our experimental results show that the proposed method can reduce area overhead and test application time compared to those for HTPT.

2. Target circuit and fault

An RTL design generally consists of a controller and a data path, and they are connected each other by control signal lines and status signal lines. Our target part is the data path separated from the controller part. All the control signals and the status signals of the data path are assumed to be directly controllable and directly observable, respectively. In order to realize controllability and observability of control signals and status signals, respectively, we need some mechanism to generate control signals and to observe status signals in test mode. In this dissertation, the implementation of such mechanism is not considered.

A data path consists of hardware elements (e.g. PIs, POs, registers, multiplexers, operational modules, and observation modules) and lines to connect output ports of hardware elements with input ports of others. There are two types of input ports of a hardware element: data input ports and control input ports. Each data input port is reachable directly or indirectly from at least one PI. Each control input port is connected with control signal line. Similarly, there are two types of output ports of a hardware element: data output ports and status output ports. Each data output port is reachable directly or indirectly to at least one PO. Each status output port is connected with status signal line. An operational module has one or two data input ports, one data output port and at most one status output port, and an observation module has one or two data input ports, one status output port, at most one control input port. We assume that (i) all lines have same bit width. (ii) There is no chaining of operational modules. Note that chaining modules can be regarded as n input and one output operational module. We relax the second assumption by extending the consideration of two input modules. We target all the path delay faults except for faults on paths that start at control inputs or end at status outputs.

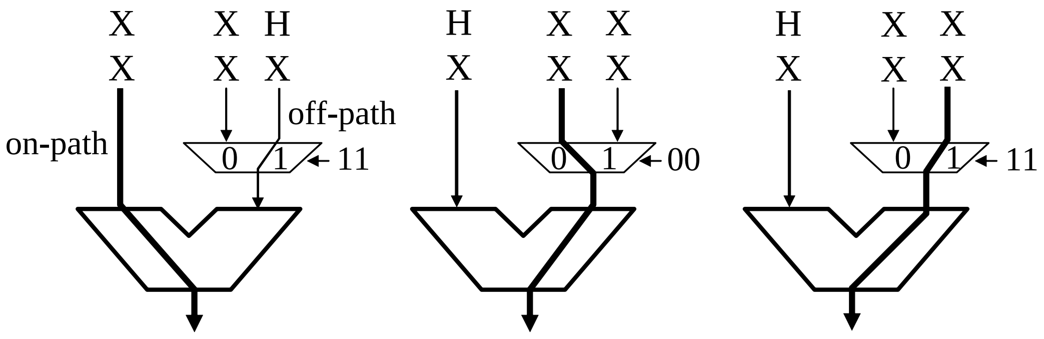

Figure 3.1. Constraints of ATPG to generate SPC two-pattern tests.

3. SPC two pattern testability

3.1 SPC two-pattern test

In this section, a combinational block that consists of combinational hardware elements on an input cone to a register is considered at RTL. We refer to an RTL path that is a target of testing as *on-path.* As opposed to on-path, we refer to an RTL path that supports the propagation of a transition launched at the starting point of an on-path along the on-path as *off-path.* For the input port of an operational module on an on-path, one of the RTL paths passing through the other port can be an off-path (See the left picture of Figure 3.1). In this dissertation, we assume that an operational module has one or two input ports and there is no chaining module, hence the number of off-paths is at most one for each on-path. An SPC two-pattern test is a pair of two consecutive vectors that launches transitions at the port corresponding to the starting point of the on-path and sets stable two consecutive vectors for the other ports of the combinational block. When SPC two-pattern tests are applied to a combinational block, the select signal of each MUX is fixed with an on-path or an off-path being selected. Amin[8] showed that while the select signal of a MUX is fixed, propagation of the signals from the selected input to the output is independent of the signals at the other input. Therefore the on-path is testable if SPC two-pattern tests can be applied to the starting points of the on-path and the off-path.

SPC two-pattern tests for combinational blocks can be generated by using a combinational test generation algorithm with constraints. To describe the constraints, we use the notation X and H. X denotes that it is possible to generate arbitrary vector and H means that the vector just before is held. In Figure 3.1, we show an example of constraints for ATPG. XX for an on-path (a bold line in the figure) denotes that it is possible to generate arbitrary two vectors consecutively. XH denotes that the first vector is an arbitrary vector and the second vector is the same as the first one. This is the input constraint for off-path. As we mentioned above, for the inputs other than those on on-path, off-path and the select signal line of each MUX, we do not care generated vectors, hence we denote them as merely XX.

3.2 Quality of SPC two-pattern test

Smith[11] showed that a path delay fault is testable by a robust test if and only if there exits a robust single-input change (SIC) test for this fault, and Gharaybeh[12] showed that the same applies to non-robust tests. Their theorems show that there exist SIC robust tests for robust testable path delay faults and SIC non-robust tests for non-robust testable path delay faults. At gate level consideration, an SIC two-pattern test launches a transition for 1 bit of inputs of a combinational block, while an SPC two-pattern test can launch transitions for any bits of inputs of the corresponding port. Hence SPC two-pattern test can completely cover an SIC two-pattern test. In other words, there exists an SPC robust (resp. non-robust) test for a robust (resp. non-robust) testable path delay fault without loss of test quality.

The remaining testable path delay faults are FS path delay faults. To test these faults, transitions are needed at multiple inputs. A FS path delay fault that needs transitions for some inputs of only on-path can be tested using an SPC two-pattern test. However, faults that need transitions for some inputs of both an on-path and an off-path cannot be tested under the concept. We will experimentally examine how many FS faults become untestable.

3.3 SPC two-pattern testability

We define SPC two-pattern testability for RTL paths. To test an RTL path that does not pass through an operational module with two input ports, one control path and one observation path are sufficient to test the RTL path. Control paths are the paths to justify test patterns from PIs to each register and observation paths are the paths to propagate the responses to POs. If we consider only one control path, we need not care about timing conflict to justify test patterns. Timing conflict means that more than or equal to two values are required to the same PI at the same time. Hence it is certainly possible to generate a control path by using a *thru* function[14], A thru function is added to an operational module in order to propagate a value along a control path or an observation path without changing the value . The realization of a thru function is shown in Section 5. To test an RTL path $p \in P$ that passes through an operational module having two input ports, it is necessary to justify test patterns from a PI or PIs to appropriate registers by a pair of control paths C_1, C_2 and propagate test responses from an appropriate register to a PO by an observation path O_p, where C_1 is the path from a PI to the starting register of an on-path p, and C_2 is the path from a PI to the starting register of an off-path.

Definition 10: An RTL path p is SPC two-pattern testable if there exists a pair of control paths C_1 and C_2 that can apply SPC two-pattern tests to the combinational block and O_p that can observe the test responses.

Conditions for control paths

Here, to simplify the following discussion, we assume that there exists a thru function for each input port of every operational module in a data path. In the next section, we will propose an efficient DFT algorithm to add thru function to data paths. In order to support the application of SPC two-pattern tests with a pair of control paths C_1 and C_2, the difference between the sequential depths of C_1 and that of C_2 and/or the number of *hold* registers on C_1 and that on C_2 should be considered. The sequential depth of a control path C_i is the number of registers that appear on C_i and is denoted as $SD(C_i)$. Let EP_1 and EP_2 be the ending point of C_1 and that of C_2, respectively. If C_1 and C_2 are not disjoint, let C'_1 and C'_2 be the paths from the diverging point of C_1 and C_2 to EP_1 and

EP_2, respectively. In the following theorem, we show necessary and sufficient conditions for a pair of control paths C_1 and C_2 to support SPC two-pattern tests.

Theorem 1: A pair of control paths C_1 and C_2 can justify SPC two-pattern tests to their ending points EP_1 and EP_2 if and only if C_1 and C_2 satisfy one of the following five conditions.

1. C_1 and C_2 are disjoint.
2. $|SD(C'_1) - SD(C'_2)| \geq 2$
3. There exist at least two hold registers on C'_1.
4. There exists at least one hold register on C'_2.
5. There exists at least one hold register on C'_1 and $SD(C'_2) - SD(C'_1) = 1$ □

Examples of these conditions are shown in Figure 3.2.

Proof: An arbitrary SPC two-pattern test (V_1, V_2) is represented as $V_1 = v_{11}\&v_{21}$ and $V_2 = v_{12}\&v_{22}$. v_{11} and v_{12} are applied to an on-path. v_{21} and v_{22} are applied to an off-path and they are the same value.

Sufficiency: Since we assume that there exists a thru function between each input and the output of every operational module, we have only to consider timing conflicts. If C_1 and C_2 satisfy Condition 1, it is obviously possible to justify any SPC two-pattern test from PIs to EP_1 and EP_2 (see Condition 1 of Figure 3.2). With regard to Conditions 2,3,4 and 5, although C_1 and C_2 are not disjoint, it is also possible to justify any SPC two-pattern test without a timing conflict. In Condition 2, we first apply the first and the second partial vectors consecutively to the PI for the control path with higher sequential depth. Then we apply consecutively the remaining two vectors to the same PI. In Condition 3, we first load v_{11} and v_{12} into two hold registers on C'_1 and hold the values, secondly we apply consecutively v_{21} and v_{22} to the PI. In Condition 4, we first load v_{21} into hold register on C'_2 and hold the value. Then we apply v_{11} and v_{12} consecutively. In Condition 5, we first load v_{11} into hold register on C'_1 and hold v_{11}, then we apply v_{21}, v_{22} and v_{12} consecutively.

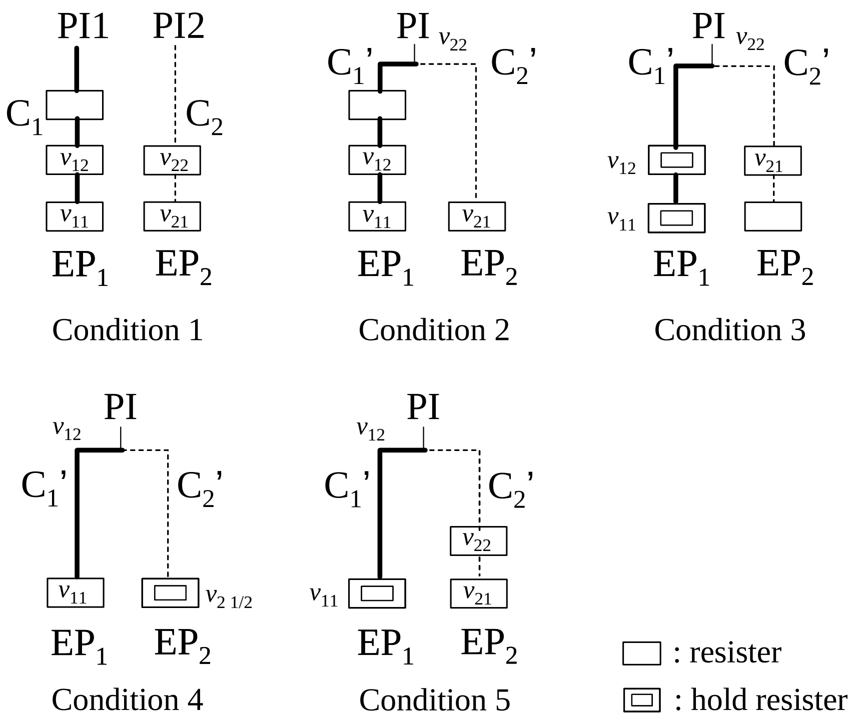

Figure 3.2. Conditions for C_1 and C_2.

Necessity: We assume that two control paths C_1 and C_2 do not satisfy any of the above five conditions. Such control paths satisfy all the following properties.

1. C_1 and C_2 are not disjoint.
2. $|SD(C'_1) - SD(C'_2)| < 2$
3. The number of hold registers on C'_1 is at most one.
4. There is no hold register on C'_2.
5. There is no hold register on C'_1 if $SD(C'_2) - SD(C'_1) = 1$.

All the possible pairs of control paths C_1 and C_2 that satisfy all the above properties are as follows.

- C_1 and C_2 are not disjoint and $|SD(C'_1) - SD(C'_2)| = 1$ and there is no hold register on both C'_1 and C'_2.
- C_1 and C_2 are not disjoint and $|SD(C'_1) - SD(C'_2)| = 0$ and there is no hold register on both C'_1 and C'_2.
- C_1 and C_2 are not disjoint and $SD(C'_1) - SD(C'_2) = 1$ and there is only one hold register on C'_1.
- C_1 and C_2 are not disjoint and $|SD(C'_1) - SD(C'_2)| = 0$ and there is only one hold register on C'_1.

Any pair of control paths C_1 and C_2 described above can not guarantee SPC two-pattern test. Therefore five conditions are the only conditions for a pair of control paths C_1 and C_2 to justify SPC two-pattern tests from a PI or PIs to EP_1 and EP_2. □

Here we consider relaxation of the assumption of the number of input ports of an operational module. The following theorem shows the sufficient conditions for an operational module with n input ports.

Theorem 2: n control paths support the application of SPC two-pattern tests for an RTL path p if either of the following conditions is satisfied. □

1. Any pair of n control paths are disjoint.
2. With regard to each pair of control paths for off-paths that are not disjoint, the mutually disjoint parts from the diverging point to both ending points cross at least one hold register.

The proof of this theorem is similar to that of Theorem 1.

As we mentioned in this subsection, to guarantee SPC two-pattern test, a register with hold function is needed even if the difference between sequential depths of C_1 and that of C_2 is zero. However to guarantee arbitrary two-pattern test in such case, we need more complex hardware element for DFT.

Conditions for observation paths

To observe a test response, the value captured at the ending register of an RTL path has to be propagated to a PO without changing its value. Fortunately, we need not care about timing conflict because only one observation path is sufficient to propagate the value. Hence to guarantee the propagation, it is sufficient to add a thru function to each operational module on the observation path.

4. DFT method for RTL data path

In this section, we propose a DFT method that makes RTL paths in a data path SPC two-pattern testable.

4.1 DFT element

Additional hardware elements of DFT are multiplexer (MUX), hold function and thru function. We use a MUX to make a new RTL path from a PI to a register. A hold function is added to a register for the purpose of holding the value according to need, and it is realized by adding a MUX just before the register to feedback a value from the output to the input(Figure. 3.3). A thru function is explained briefly in Section 4. For a common module, such as adder or multiplier, it is realized by providing a constant value to the other input(Figure. 3.4). It can be provided by adding a mask element. A mask element generates a constant depending on its control signal. For a more complex module or a module with one input port, we cannot realize the thru function by only providing a constant, then we deal with the thru function by bypassing the module using a MUX(Figure. 3.5).

4.2 Algorithm for adding DFT elements

The flow of the proposed DFT algorithm is shown in Figure 3.6.

step 1: There are some RTL paths that start at a register and go back to the same register. There are many cases where SPC two-pattern tests cannot be applied to such an RTL path because it is structurally difficult to satisfy the

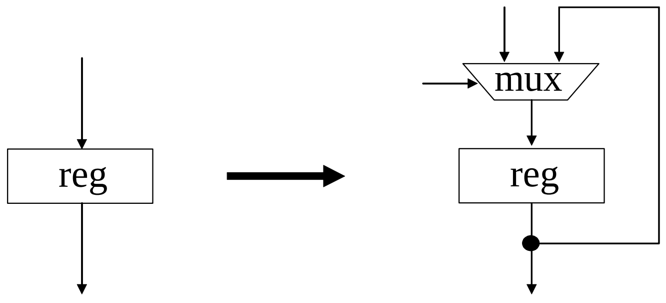

Figure 3.3. Adding hold function by a MUX

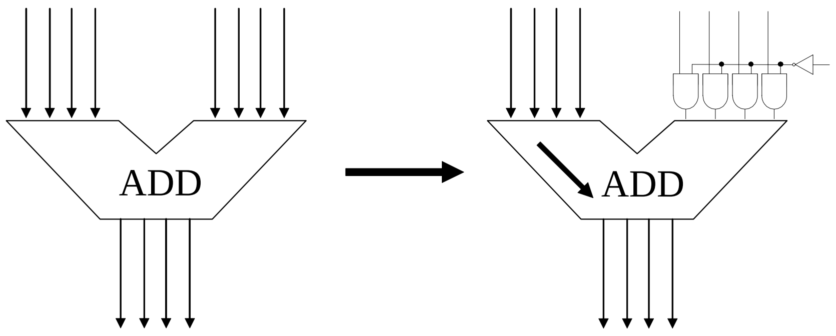

Figure 3.4. Adding thru function by a mask element

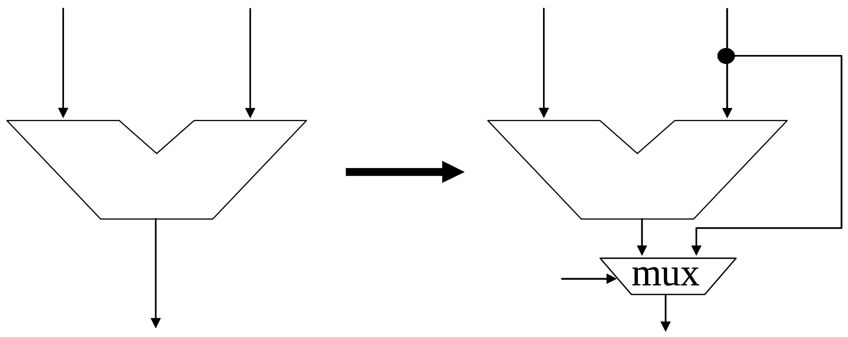

Figure 3.5. Adding thru function by a MUX

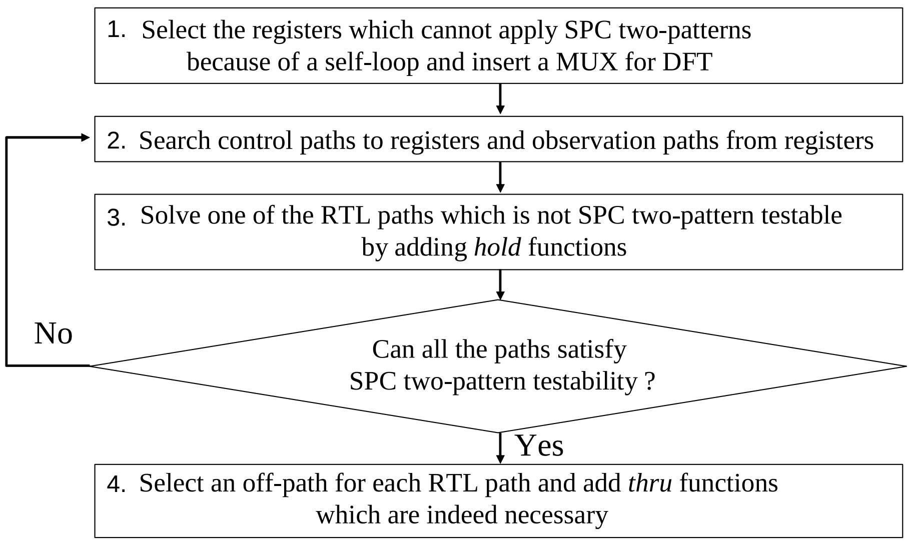
1. Select the registers which cannot apply SPC two-patterns because of a self-loop and insert a MUX for DFT
2. Search control paths to registers and observation paths from registers
3. Solve one of the RTL paths which is not SPC two-pattern testable by adding *hold* functions
Can all the paths satisfy SPC two-pattern testability ?
No
Yes
4. Select an off-path for each RTL path and add *thru* functions which are indeed necessary

Figure 3.6. Flow of our DFT algorithm.

conditions of Theorem 1. Since it is only possible to make such an RTL path SPC two-pattern testable by adding MUX (hold function cannot solve this problem) and making a new control path from a PI, we first find such structures. To find RTL paths forming a loop we consider a circuit as a circuit graph consisting of four types of nodes, R, Op, Fo and M, and directed edges. The nodes of type R, Op, Fo and M correspond to a register, an operational module, a fanout and a MUX, respectively, and they are connected by directed edges corresponding to the signal lines of the circuit. We refer to the loop that starts at R-type node and go back to the same node without passing through any other R-type node as a self-loop.

We consider a self-loop R_i-M_j-Op_k-R_i. It is impossible to apply SPC two-pattern tests to the RTL path corresponding to the self-loop if there is no M-type node, which can be reached from a PI without passing through the self-loop, between the Op_k and the R_i. Such an RTL path can be solved by inserting a MUX between Op_j and R_i, and adding a new path from a PI to the MUX. Here we consider the self-loop, $R1$-$m1$-$m2$-$Add1$-$m5$-$R1$ in figure 3.7 as an example, and corresponding nodes in its circuit graph are named R_1, M_1, M_2, Op_1 and M_5,

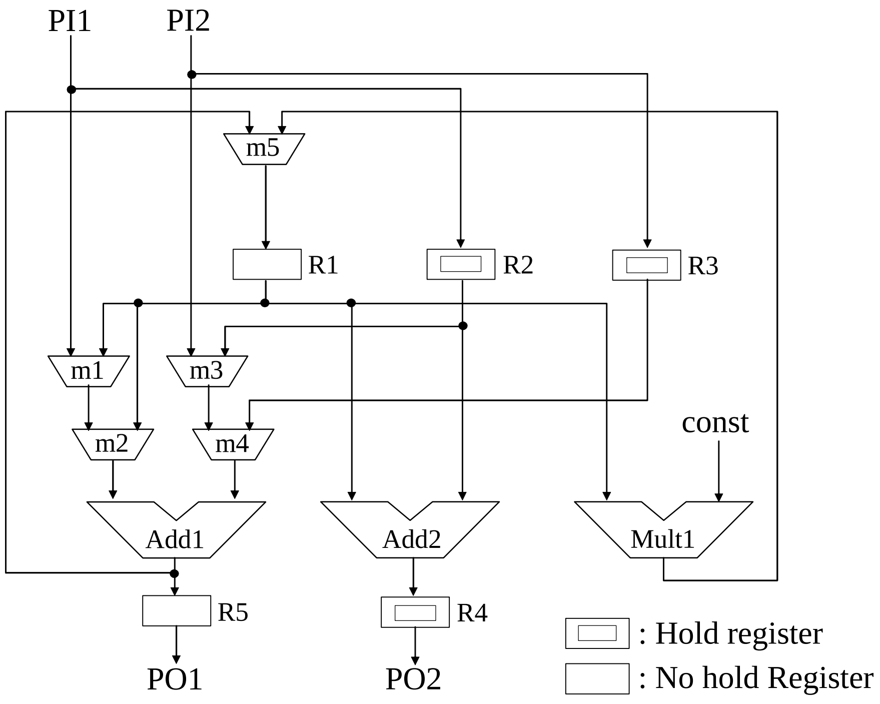

Figure 3.7. LWF benchmark circuit

respectively. There is no M-type node between Op_1 and R_1 which can be reached from PI without passing through the self-loop, hence a MUX is added to the place between Op_1 and R_1, and then a new RTL path $PI1$-MUX-$R1$ is made. When there are some PIs in a circuit, we select the PI such that the pair of control paths is disjoint to satisfy the first condition of Theorem 1. However if there is only one PI in the circuit, we make a new RTL path from the PI. In this case, if the pair of control paths may not satisfy any conditions from second to fifth, hold function is added in step 3.

step 2: In this step, candidates for control and observation paths to each register are selected using heuristics. The decision of control and observation paths will be made in step 3.

In order to reduce area overhead and test application time, control paths is selected as they form trees whose source nodes are PIs, accordingly each register is reachable from a PI via a control path with the minimum sequential depth.

To search such control paths, we represent the data path as a port graph $G = (V, E)$[14]. V is the set of all input ports and output ports of modules, and E is the set of all directed edges corresponding to the signal lines in the data path and relation between an input and an output of each module (we call the latter edge inside edge). We apply *breadth first search* (BFS) with respect to the number of registers to the port graph. From the result of the search, we obtain trees that contain the information of control paths with the minimum sequential depth from PIs to registers. The search ends when all the registers become reachable. In [8][14], to search control paths they also make use of BFS. In this paper, we add a new condition for search which takes advantage of the feature of SPC two-pattern testability. Considering the conditions of Theorem 1, it is desirable that there exists a hold register on a control path. Therefore we choose a path starting at a hold register if there are some paths that can be chosen at the same sequential depth. Figure 3.8 shows the port graph for LWF and candidates of control paths for each register in LWF.

Next we search observation paths with the minimum depth. The search from each register to a PO makes use of observation trees. Observation trees are made by performing the BFS from each PO on the port graph that is generated by reversing the direction of edges, then the BFS prioritize the path on a control tree to share thru function between control paths and observation paths if there is a branch.

step 3: For one of the RTL paths that are not SPC two-pattern testable, we modify it into SPC two-pattern testable path by adding a hold function to its starting register. In this step, RTL paths whose pairs of control paths have not yet been determined are dealt with. We first judge RTL paths one by one whether it satisfies one of the conditions of Theorem 1 or not. If the RTL path is SPC two-pattern testable, the pair of control paths generated in step 2 is determined. However, if the RTL path has no pair of control paths satisfying any one of the five conditions at all, it is sufficient to add a hold function to one of the registers that can be the starting points of off-paths in order to satisfy condition 4 of Theorem 1. Among the registers, a hold function is added to the register with the smallest sequential depth. Consequently, more control paths share the hold function because a set of control paths forms trees. Here we consider testing

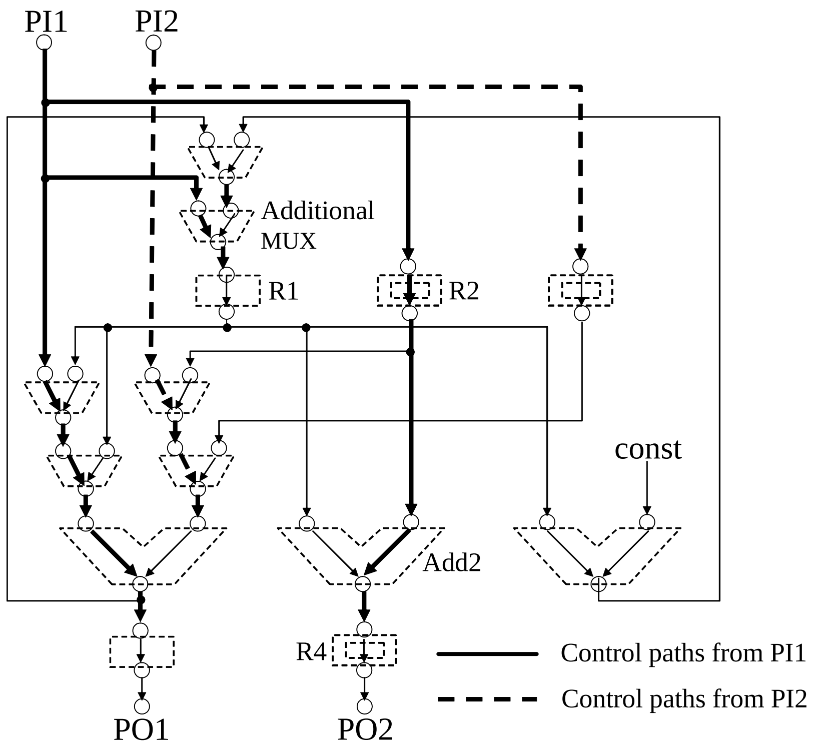

Figure 3.8. The port graph and candidates of control paths for LWF

of RTL path $R2$-$Add2$-$R4$ in figure 3.8. The control paths for $R2$ and $R1$ are $PI1$-$R2$ and $PI1$-Additional MUX-$R1$. The additional MUX was already added between $m5$ and $R1$ in step 1. Since the pair of control paths cannot satisfy any conditions of Theorem 1, a hold function is added to $R1$. If a hold function is added, go back to step 2 and make the control trees again for the modified circuit. Then only unsolved RTL paths will be target of step 3 again.

step 4: We consider how to realize shorter test application time when there are some choices of off-paths for testing an on-path. We first try to select an off-path having a control path with the minimum sequential depth among them and disjointed from the control path for on-path. If there does not exist such an off-path, that of the minimum depth is selected. We assumed that thru functions are available for all the input ports of all operational modules, however some of them may not be necessary. It is indeed necessary to add a thru function between

Table 3.1. Circuit characteristics.

Circuit	BW	#PIs	# POs	# REGs	# RTL path	Area
Paulin	16	2	2	7	29	10,550
LWF	16	2	2	5	19	3,322
RISC	32	1	3	40	10,108	94,302
MPEG	8	7	16	241	651	77,554

an input port and an output port, corresponding to inside edges on control or observation paths, of an operational module. To realize a thru function, we first search a support path[14] considering timing conflict. A support path is a path from a PI to an input of an operational module, which can justify a constant. If there does not exist such a path, we add a mask element or a MUX for bypass to realize it.

5. Experimental Results

In this section, we evaluate the effectiveness of the proposed DFT method compared to the previous DFT method for HTPT[8] with regard to area overhead and test application time. The DFT method that guarantees HTPT has similar advantages to enhanced-scan approach and can reduce the area overhead and the test application time. The circuit characteristics of RTL benchmarks used in the experiments are shown in Table 3.1. Paulin, LWF are widely used circuits. RISC and MPEG[1] are more practical and larger circuits designed by industry. In this experiment, we used the logic synthesis tool Design Compiler (Synopsys). To generate SPC two-pattern tests, we used the combinational test generation algorithm that supports constraints[15].

With regard to robust and non-robust path delay faults, the fault coverage of our method is equal to that for HTPT. The CPU time required for our proposed DFT algorithm is as follows. For LWF, Paulin and MPEG, their CPU times are

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

about 0.1 seconds. For RISC, the CPU time is 3.94 seconds. Table 3.2 shows the results of area overhead, test generation time and test application time. For all benchmark circuits, area overhead of the proposed DFT method is lower than that of the DFT method for HTPT. The difference between area overhead of the proposed method and that of the previous one become large if there are many registers that are reached from the same PI and at the same sequential depth. For 8 bit Paulin, the test generation time of our proposed method is shorter than that of HTPT. The reason is that Paulin has two multipliers, and for path delay faults on paths passing through a multiplier, the test generation time with the input constraint for single-port-change tends to become shorter than that with no input constraint. For 8bit LWF and 16 bit LWF, the test generation times of our method become longer than that of HTPT. The reason is that LWF has an adder and for an adder, the test generation time with the input constraint for single-port-change tends to become longer than that with no input constraint. For 16 bit Paulin, RISC and MPEG, we cannot evaluate the test generation time because the number of faults is extremely large.

For 8bit Paulin, 8bit LWF and 16 bit LWF, the test application times are 1,594,259 cycles, 49,916 cycles and 2,096,465 cycles, respectively. Those results are smaller than that for HTPT data path. The reason is that the previous method adds extra registers to these circuits. In such case, extra one cycle is necessary for loading data into such a register. For RISC and MPEG, it is not practical to test all paths in the data path because the number of paths is extremely large. Therefore we consider the critical parts that affect the difference between test application time of the proposed method and that of previous one. For RISC, an ALU is critical part and its number of tests is denoted as T_{ALU} in the table. The proposed method can reduce 25% compared to the previous one. For MPEG, a sub circuit composed of 64 identical structures of modules is critical. The number of tests is denoted as T_M in the table. For both methods, the test application times are almost the same.

In subsection 3.2, we showed that SPC two-pattern tests can test a subset of FS path delay faults. Here, we show the number of FS path delay faults in three simple operational modules that can be tested by SPC two-pattern tests. For an adder and a subtracter, there is no FS path delay fault. All the faults in an adder or a subtracter can be robust or non-robust path delay faults. For an 8 bit multiplier, 947 of the total 49328 FS path delay faults are tested. From these results, SPC two-pattern tests do not always test all the FS path delay faults of an operational module. On the other hand, if any two-pattern test can be applied, all the FS path delay faults are tested. For every RTL path in an HTPT data path, we can apply any two-pattern test. If it is necessary to test FS path delay faults of such an operational module that is SPC two-pattern test resistant, we can guarantee application of arbitrary two-pattern tests by applying the DFT method of HTPT only for the module.

In Chapter 4, we propose a method of identifying false paths at RTL. The objective is to reduce over-testing caused by DFT techniques. The path identification method identifies RTL path as control-dependent false paths (CFPs). Path delay faults on gate-level paths corresponding to CFPs are excluded from the target of testing, as a result, not only over-testing but also test generation time and test application time are reduced. The detail of the path identification method is shown in Chapter 4, however excluding path delay faults on paths identified as false can reduce test generation time and test application time of this experiments. For LWF and Paulin, we can identify 3 of the total 19 RTL paths and 11 of the total 29 RTL paths as CFPs, respectively. For 8 bit Paulin, 8bit LWF and 16 bit LWF, test generation times become 1,956 seconds, 33 seconds and 551 seconds, respectively. Test application time for 8 bit Paulin, 8 bit LWF and 16 bit LWF become 785,136 cycles, 38,913 cycles and 1,638,660 cycles, respectively. Test application times become about half of those for HTPT if our path identification method is applied.

Table 3.2. Results of DFT and test generation.

		Area overhead[%]		TG time [sec]		Test application time[cyc]	
Circuit	BW	Proposed	HTPT	Proposed	HTPT	Proposed	HTPT
Paulin	8	5.13	11.56	3,906	5,828	1,594,259	1,645,335
	16	3.30	7.43	-	-	-	-
LWF	8	7.43	15.25	42	35	49,916	74,792
	16	6.38	13.99	700	805	2,096,465	3,162,124
RISC	32	0.64	1.99	-	-	$3T_{ALU}$+2	$4T_{ALU}+2$
MPEG	8	4.64	9.35	-	-	$186T_M$+2079	$186T_M$+2016

6. Summary

We have presented a concept of single-port-change (SPC) two-pattern testability and proposed an efficient non-scan DFT method for data path. The proposed method can reduce area overhead and test application time compared to the previous DFT method for hierarchically two-pattern testability without losing the quality of test.

Chapter 4

Over-Tsting Reduction for Path Delay Faults in Controller-Datapath Circuits Using RTL Information

1. Introduction

In today's competitive LSI market, test cost and product quality of LSIs are important factors which contribute to benefit of productions and customer satisfaction. Alleviating overkill of products is an important issue in order to keep the manufacturing cost lower, that is, reducing the number of parts which are failed by its tests under evaluation but which never cause any failure in the application[27]. One of the solutions of this issue is reduction in over-testing caused by design-for-testability techniques, which is our objective. To realize 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.

With the progress of VLSI design and manufacturing technologies, the speed of VLSI circuits has increased in recent years. For high speed circuits, delay testing is emphasized to guarantee the timing correctness of circuits in addition to testing logical correctness. A path delay fault model can deal with accumulation of small delays along a path, and testing a path delay fault on a path detects a defect which induces a delay beyond a specified clock period[1]. The main disadvantage of the path delay fault model is the number of faults in a circuit is extremely large because the number of paths is exponential to the circuit size. Besides, it is said that a large number of path delay faults are functionally untestable, which include both combinationally untestable and sequentially untestable.

To reduce test costs, design-for-testability (DFT) techniques such as enhanced scan [5] or standard scan with *skewed-load testing* [3] or *broadside testing* [4] are used. These techniques make most or all the sequentially untestable path delay faults testable. However, the path delay faults which became testable by the DFT never affect the circuit performance even if they exist on paths. This is because a transition launched at the starting point of the untestable path is never propagated to the ending point of the path during normal operation. Therefore we consider that testing such path delay faults is 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. Also test generation time and test application time become shorter if the over-testing is reduced.

A basic flow of reduction in the over-testing is as follows. First we identify sequentially untestable path delay faults. Next the identified path delay faults are removed from the fault list targeted by test generation. And then test generation is performed for each path delay fault in the fault list, and generated patterns are applied to a circuit. However, during test application, patterns generated by the test generation may still test the excluded faults accidentally because the generated pattern to test some path delay fault in the fault list may propagate transitions along several paths and may activate other path delay faults simultaneously. Therefore, the over-testing have still unresolved. We have two tasks to reduce the over-testing. One is to identify as many untestable faults as possible. The other is to generate two-pattern tests that do not test the path delay faults removed from the fault list.

In recent ten years, several path identification methods have been proposed. For combinational circuits, techniques presented in [22, 23, 24] are efficient approaches to identifying combinationally untestable path delay faults at gate-level. For sequential circuits, Krstic et al, proposed a method to find sequentially untestable path delay faults using time expansion model at gate-level [18]. For small scale circuits the method succeeded to identify sequentially untestable path delay faults within practical time. However it is conceivable that the method takes long CPU time for large scale circuits. In [13, 25], identification methods of multi-cycle paths at gate-level are proposed. In the work [26], false paths are identified at RTL. However their interest is to prevent inaccurate decision of 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.

We propose a method of identifying a large number of sequentially untestable paths and multi-cycle paths using register-transfer level (RTL) information such as load-enable signals of registers and select signals of multiplexers (MUXs). The target path of this method is an RTL path which is a bundle of gate-level paths between two registers. The total number of RTL paths in a circuit is much smaller than the total number of gate-level paths, therefore our method can be performed in reasonable time. Experimental results for some RTL benchmark circuits show that our path identification method can identify many RTL paths as false in a few seconds. We also show that there are a lot of gate-level paths, which are false paths, corresponding to the identified RTL paths.

We also address a method to guarantee that the identified untestable path delay faults are never tested by using the concept of single-port-change (SPC) two-pattern testability proposed in Chapter 3. The concept of SPC was originally proposed to reduce area overhead required for DFT techniques. In this Chapter, we utilize the concept to reduce the over-testing. An SPC two-pattern test changes the second vector at only one port and sets stable for the other ports. In other words, The SPC two-pattern test can test path delay faults on the paths starting from the port whose second vectors are changed. Therefore the SPC two-pattern tests never test path delay faults on paths starting from other ports.

In our experiments, reduction in the over-testing for SPC two-pattern tests compared to that for unconstrained (normal) two-pattern tests is shown and our proposed method never tests path delay faults which are identified as false. Moreover, we show that if unconstrained two-pattern tests for the same fault list are generated, on average 10-60 % of path delay faults identified as false are accidentally tested during test application.

2. Preliminaries

2.1 RTL circuit

Our path identification method uses RTL information. In this paper, we consider structural RTL designs as shown in Figure 4.1. A structural RTL design consists of a controller represented by a finite state machine, and a datapath represented by RTL modules such as MUXs, operational 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 hardware elements (e.g., load-enable signals of registers and select signals of MUXs) in the datapath. 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, we may not obtain such RTL information from the description. In such a case, we can obtain the information about structure during synthesis process.

2.2 RTL path

A path at gate-level is an ordered set of gates $\{g_0, g_1, .., g_a\}$, where g_0 is a primary input or a flip-flop and g_a is a primary output or a flip-flop when we consider a sequential circuit. Also $g_h(1 \leq h \leq a-1)$ 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 an RTL path. 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 is passing 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.

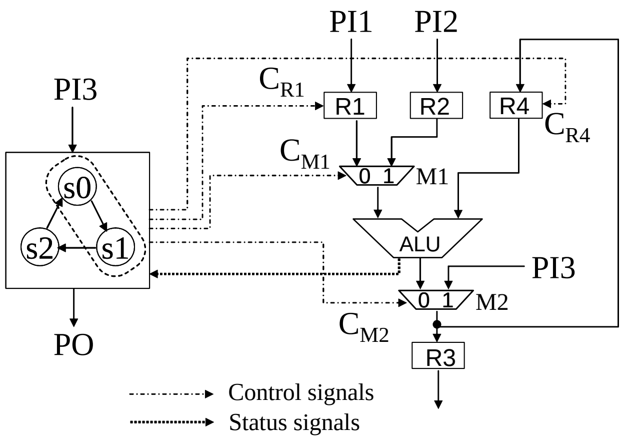

Figure 4.1. An RTL circuit.

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 [2]. In this paper, we apply a method of false path identification to RTL paths in an RTL circuit and propagate the information about the paths at RTL 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 module interface preserving-logic synthesis.

Definition 11:(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 one 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 one 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 12: (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. $\square$ Our approach is to identify bundles of gate-level paths corresponding to RTL false paths as false. However, it may be hard under any logic synthesis to match an RTL path to all the gate-level paths corresponding to the RTL path completely. To achieve the matching, we restrict logic synthesis to MIP-LS.

Definition 13: (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. $\square$

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 D and D' be an RTL circuit and its gate-level circuit synthesized by an MIP-SL, respectively. Let $F = \{Fj | 1 \leq j \leq m\}$ be a set of flip-flops in D' corresponding to an m bit register R in D. $\tau(R)$ 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 3: An RTL path p in an RTL circuit D is RTL false if 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. $\square$
Proof: We show that $\forall g \in \delta(p)$ in a gate-level circuit synthesized by any MIP-LS is gate-level false.

A combinational RTL module M_a in D and its synthesized logic block M'_a in D' 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 an RTL path p none of the three conditions of Theorem 3 can be satisfied 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 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 a set of the input sequences that satisfy Conditions 1, 2 and 3 of Theorem 3, 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 drives the ending point of

p 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 on $\forall h \in \tau(R_e)$ is also prevented at the logic block corresponding to M_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 one of the three conditions of Theorem 3 is satisfied for any input sequence. $\square$

3.2 Control-dependent false path

From an RTL description of a circuit, for each pair of a state and an input vector, we can obtain information about state transitions of the controller and the control signals such as load-enable signals of registers and select signals of MUXs. A path to transfer a data from a register to another register is determined by the select signal of each MUX on the path. Timing of data transfer between registers is determined by the load-enable signal of each register. In this section, from the information of control signals, we define sufficient conditions for identifying RTL false paths. The RTL path identified from the conditions is referred to as a control-dependent false path (CFP).

Figure 4.2 shows a synthesized gate-level controller. The state register (SR) 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. This is because we can know the timing where transitions are launched for each bit of the SR by considering state assignment. A register in a datapath is referred to as a datapath register (DR).

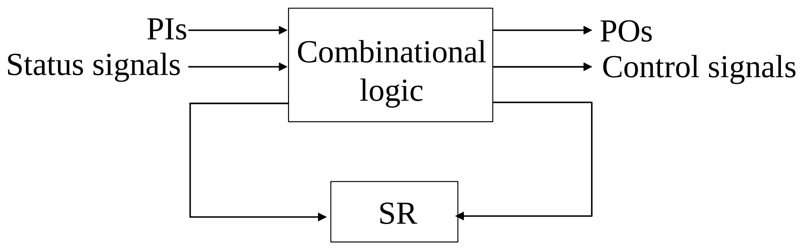

Figure 4.2. An implementation of a controller.

RTL paths from DR/PI to DR/PO/SR

Let P be a set of RTL paths which start 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, holds its value. Note that if the register does not have *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 4.1. When M_1 and M_2 select p, $p_{M_1} = 0$ and $p_{M_2} = 0$. Let S_y and S_z be states of a controller. When S_y transits to S_z, let k be the time at S_y and let $k + 1$ be the time at S_z.

We first consider uncontrollability and unobservability of registers. Then we define conditions of control signals to identify false paths.

Definition 14: (Uncontrollability of register R at time t)

Let q be an RTL path ending at R. For each state at time $t - 1$, if at least one of the following conditions is satisfied, R is uncontrollable at time t.

- For each q, control signals 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 t in Figure 4.4. Reg3 is not a source register because the data is prevented at the MUX at time t.

Definition 15: (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 one of the following three conditions is satisfied for any state transition from time k to $k+1$.
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 3. 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 3 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 3 is also satisfied. If Condition 3(1) of Lemma 1 is satisfied for the state

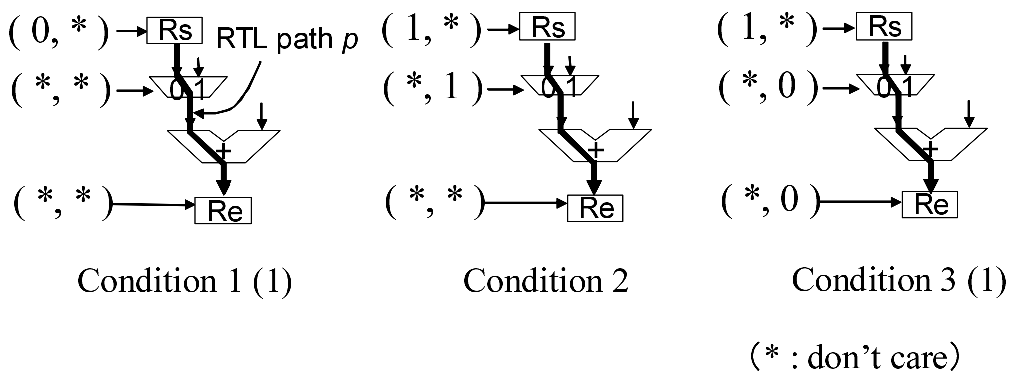

Figure 4.3. Examples of Condition 1(1), 2, and 3(1) of Lemma 1.

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 3 is also satisfied.

One of the three conditions of Theorem 3 is satisfied for any input sequence if one of the three conditions of Lemma 1 is satisfied for any state transition. Therefore, Lemma 1 are properly included in Theorem 3. □

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 4.3. 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 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 not 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 k; that is, there is no transition at *Rs*. The unobservable condition of Condition 3(2) is that the captured value of *Re*

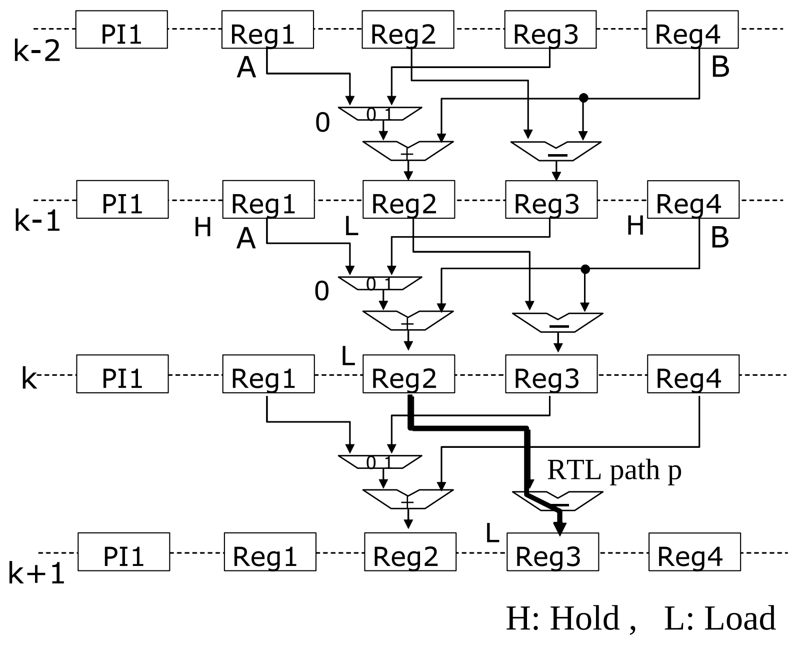

Figure 4.4. An example of Condition 1(2) of Lemma 1.

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 4.4 and 4.5, which are time expansion models of an RTL circuit.

In Figure 4.4, 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 each value 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 4.5, 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, the MUX on the RTL path does

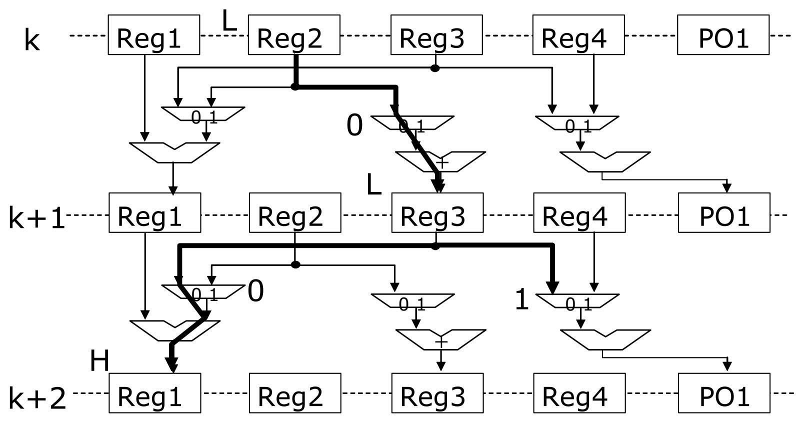

Figure 4.5. An example of Condition 3(2) of Lemma 1.

not select the path at $k+1$; that is, propagation of the value is prevented by the MUX.

A path delay fault (PDF) on any gate-level path $g \in \delta(p)$ is non-robust untestable if p is an RTL false.

RTL paths starting at SR-ff

For RTL paths from SR-ff, 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 to DRs do not satisfy Condition 1 (1) of Lemma 1. Here we consider transitions from each flip-flop in the SR (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 4.1. 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 one of the three conditions of Lemma 1 is satisfied for all the time t when a rising

	S_k	S_m	S_n
FF0	0	1	1
FF1	1	1	1
⋮		⋮	⋮
FFn	0	1	0

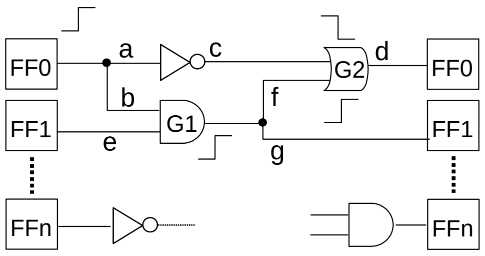

Figure 4.6. State assignment and a counter example.

(resp. falling) transition launches at the SR-ff of the starting point of the RTL path.

Singh et al, proposed a method to identify untestable paths between SR-ffs and SR-ffs by considering a state assignment and consecutive three states[15]. However we found that the method is not sufficient to identify the paths as untestable. Consider FF_i and FF_j in the SR and consecutive three states $S_k \rightarrow S_m \rightarrow S_n$ on a state assignment. Now we assume that a transition launches at FF_i when S_k transfers to S_m. In [15], they consider that the transition at FF_i is not propagated to FF_j if the value of FF_j does not change on the state assignment when S_m transfers to S_n. Therefore the paths from FF_i to FF_j are untestable if a transition at FF_i is not propagated to FF_j for any consecutive three states. Figure 4.6 is a counter example.

According to the path identification method in [15], they consider that the 0$\rightarrow$1 transition of $FF0$ at S_m is not propagated to $FF0$ at S_n because the value at $FF0$ does not change when S_m transfers to S_n. However the path $FF0$-b-G1-f-G2-d-$FF0$ is non-robust testable. Actually, $FF0$ captures the faulty value 0 if there exists a path delay fault on the path $FF0$-b-G1-f-G2-d-$FF0$. Hence the path $FF0$-b-G1-f-G2-d-$FF0$ is testable.

4. Reduction in over-testing

To reduce over-testing, we consider that path delay faults on paths identified as false are excluded from a fault list targeted by test generation, and then test generation is performed to the fault list. However, during test application, patterns

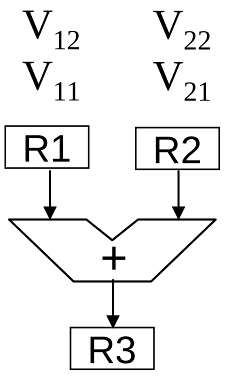

Figure 4.7. A simple RTL structure.

generated by the test generation may still detect the excluded faults accidentally. This is because a generated pattern to detect some path delay fault in the fault list may also activate other path delay faults simultaneously. Actually, our experimental results show that 10-60 % of the removed path delay faults is accidentally tested if we generate two-pattern tests with no input constraint. However, we can avoid the over-testing by generating single-port-change two-pattern tests that is introduced in Chapter 3.

4.1 Test generation and test application

We first explain a case where sequentially path delay faults removed from a fault list are tested by generating two-pattern tests. Figure 4.7 shows a simple RTL structure and there are two RTL paths, $R1$-*Adder*-$R3$ and $R2$-*Adder*-$R3$. We assume that $R2$-*Adder*-$R3$ is identified as a CFP. Then path delay faults on gate-level paths corresponding to $R2$-*Adder*-$R3$ are removed from a fault list. For the fault list, we generate a set of two-pattern tests without input constraints, i.e., any two-pattern tests can be generated. Let the first vectors for $R1$ and $R2$ be V_{11} and V_{21}, respectively. Let the second vectors be V_{12} and V_{22}, respectively. During test application, transitions may be launched at the starting point of $R2$-*Adder*-$R3$ because V_{21} and V_{22} can be different vectors. Therefore some path delay faults on gate-level paths corresponding to $R2$-*Adder*-$R3$ may be tested even if they are removed from the fault list.

Next, we show that the removed path delay faults are never tested by generating SPC two-pattern tests. Consider the same fault list as the above instance. For the fault list, we generate a set of SPC two-pattern tests such that transitions are launched at the only output port of $R1$. During test application, transitions are never launched at the starting point of $R2$-$Adder$-$R3$ because V_{21} and V_{22} are the same vectors. Therefore the removed path delay faults are never tested.

We assume enhanced scan design as a method to apply any two-pattern tests to each FF. Note that if other DFT techniques such as non-scan DFT techniques or broadside technique and skewed-load technique with standard scan are applied, we have to consider incidental activation during justification of test patterns. The incidental activation depends on scan-chain ordering for a standard scan or how to generate control paths. Analyzing alleviation of over-testing for other DFT techniques is our future work.

5. Experimental results

In this section, we evaluate the effectiveness of identifying a subset of RTL false paths as control-dependent false paths (CFPs) in Section 5.1 and reduction in over-testing in Section 5.2. The circuit characteristics of RTL benchmarks are shown in Table 4.1. LWF, Tseng, Paulin and JWF are widely used benchmark circuits. MPEG and RISC[1] are more practical and larger circuit provided by industry. These benchmarks are described at RTL and each circuit consists of a controller and a datapath. The last two columns show the number of RTL paths. The columns DR and SR-ff show that the number of RTL paths where an RTL path starts at a datapath register or a PI and that of RTL paths where an RTL path starts at an FF in the SR, respectively.

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

Table 4.1. Circuit characteristics of benchmarks.

							# RTL paths	
Circuit	Bit width	#PIs	# POs	# REGs	# States	Area	#DRs	#SR-ffs
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

5.1 Evaluation of the number of CFPs and the number of gate-level false paths

Table 4.2 shows the result for the number of RTL paths identified as CFPs. The second, third and fourth columns under DR show that the number of RTL paths identified as a CFP starting at DR, that of RTL paths corresponding to starting at DRs, and the ratio of the number of CFPs to that of RTL paths, respectively. For JWF circuit, 117 of 153 RTL paths (76.5 %) is identified as CFPs within 0.1 second. MPEG has many registers with no hold function. Therefore the starting registers may launch transitions and the ending registers may capture the propagated transitions every clock cycle. Hence the number of CFPs is small. However the CPU time is still less than 0.1 second. For even 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 are results for RTL paths starting at SR-ffs with rising transitions and that 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 that of RTL paths starting at DRs. For RISC, a large number of RTL paths are identified. RISC has a state which calculates some data using an ALU module and writes back the data to a register in a register file. RTL paths from registers in the register file to registers in the register file passing through the ALU are activated at only the state. If state assignments are done as transitions at each SR-ff become fewer at the states, the number of RTL paths identified as CFPs increase.

Table 4.2. The number of RTL paths identified as CFPs.

	DR			SR-ff: Rise			SR-ff: Fall		
Circuit	#CFPs	#RTL paths	%	#CFPs	#RTL paths	%	#CFPs	#RTL paths	%
LWF	3	19	16 %	4	26	15 %	4	26	15 %
Tseng	6	20	30 %	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 %

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 for 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 4.3 shows the result of gate-level false corresponding to RTL paths identified as CFPs. We extracted gate-level paths from the longest path, which have 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 extracted 300,000 and 200,000 gate-level paths from the longest path, respectively. The second column is the number of gate-level paths corresponding to CFPs, that is, these are gate level false paths which were identified by our method. The third column is the total number of gate level paths starting at DRs. The fourth column is the ratio of the second column to the third column. For Tseng, our method identified 2,337 of 5,192 (45 %) gate level paths as false paths. For LWF, Tseng and Paulin, the ratio of gate level false paths to the total is almost same as that of CFPs to the total RTL paths. If an RTL path which is passing a large scale operational

Table 4.3. The number of gate-level paths corresponding to CFPs.

	DR			SR-ff: Rise			SR-ff: Fall		
Circuit	#GL false	#GL path	%	#GL false	#GL path	%	#GL false	#GL path	%
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 %

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 extracted 300,000 paths are starting at DRs, and there is no false paths corresponding to CFPs. The reason is that gate level paths corresponding to the CFPs have short propagation delays, hence they did not included among the extracted 300,000 paths.

5.2 Reduction in over-testing

Here we report results of reduction in over-testing for each benchmark. Note that robust testable path delay faults and non-robust testable path delay faults are considered in this result (We evaluate this result using TetraMax by Synopsys but it cannot deal with functionally sensitizable path delay faults).

Figure 4.8 shows that the ratio of the number of path delay faults on paths identified as CFPs to that of tested path delay faults as an example. For all the path delay faults in an original sequential circuit, we cannot perform sequential test generation within practical time. Hence we do not know the exact number of untestable path delay faults in the circuit. However, for the circuit augmented by some DFT technique such as full scan, the ratio of untestable path delay faults decreases because sequentially untestable path delay faults become testable. path delay faults corresponding to CFPs are not only a subset of sequentially untestable path delay faults in the original circuit but also a part of combinationally untestable path delay faults. If we normally generate uncon-

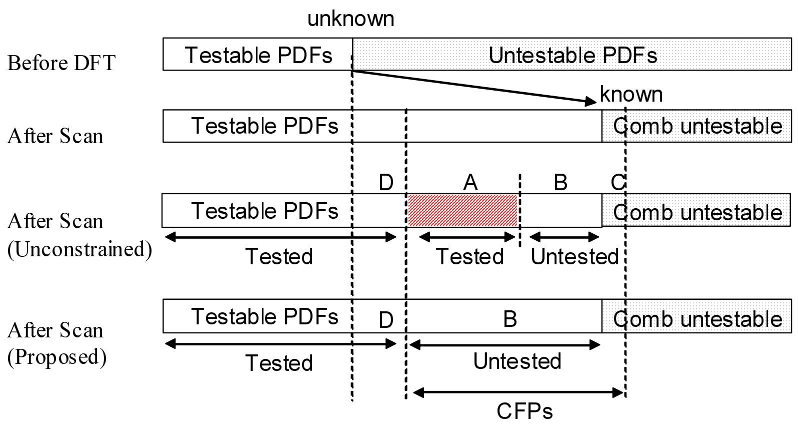

Figure 4.8. The ratio of path delay faults to tested path delay faults.

strained two-pattern tests, some path delay faults corresponding to CFPs are tested (See the third graph in Figure 4.8). On the other hand, our proposed method never test all the path delay faults corresponding to CFPs.

Table 4.4 shows the results of the reduction in over-testing for each benchmark circuit. The second column is the total number of PDFs. The third column is the number of PDFs on CFPs, which corresponds to 'A+B+C' shown in Figure 4.8. The PDFs are generated by using the "write path" function, which is for generating a fault list, of Prime Time in order to perform test generation and fault simulation. The fourth column is the number of PDFs excluding PDFs on paths that are identified as false, which corresponds to the total PDFs-(A+B+C) in Figure 4.8. The fifth column shows the number of tested PDFs among PDFs shown in the third column, which corresponds to 'A' and its ratio to the third column. In terms of reduction in over-testing, when we generate SPC two-pattern tests for fault lists excluding PDFs corresponding to CFPs, the removed PDFs are never tested. However if we generate unconstrained two-pattern tests for the same fault list, 10-60% of the removed PDFs is accidentally tested (over-testing).

Table 4.4. Reduction in over-testing for data paths.

Circuit	#PDFs (Total)	#PDFs corresp. to CFPs (A+B+C)	#PDFs Total-(A+B+C)	#Over-tested PDFs(A)	
				Unconstrained	Proposed
LWF	3,598	619	2,979	392 (63%)	0
Tseng	5,868	2,928	22,940	1,185 (40%)	0
Paulin	37,171	19,477	17,694	2,195 (11%)	0
JWF	45,658	37,150	9,402	3,452 (9%)	0
MPEG	41,521	340	41,181	220 (65%)	0
RISC	25,496	6,974	18,522	509 (7.3%)	0

6. Summary

In this chapter, first we have proposed a method to identify a large number of sequentially untestable path delay faults in a controller-data path circuit using RTL information. However even if the untestable path delay faults are removed from a fault list targeted by test generation, the generated test patterns incidentally test the removed faults. Hence we have also proposed a method that never test path delay faults identified as control-dependent false paths by using single-port-change (SPC) two-pattern testability. In the experimental results, we showed that our proposed method never tests path delay faults that were identified as CFPs. On the other hand, if we normally generate unconstrained two-pattern tests, 10-60 % of path delay faults identified as CFPs were accidentally over-tested.

Chapter 5

Conclusions and Future Works

Delay testing is an important technique to guarantee the timing correctness of a given circuit. To reduce the test generation complexity design-for-testability (DFT) techniques are generally used. On the other hand, over-testing problem is induced by influence on the DFT techniques. We have proposed a non-scan DFT method for data paths as our first work, and also have proposed a false path identification method in order to reduce the over-testing.

In Chapter 3, we have presented a concept of single-port-change (SPC) two-pattern testability and proposed an efficient non-scan DFT method to make every RTL path that has to be tested SPC two-pattern testable. Our proposed method reduced area overhead and test application time compared to the previous DFT method for hierarchically two-pattern testability without losing the quality of test with respect to robust test and non-robust. For an RTL path that is not SPC two-pattern testable (in this situation, also arbitrary two-pattern cannot be applied), our DFT method adds a hold function to the register that is the starting point of the RTL path, but on the other hand the previous DFT method may have to add an extra register. Therefore, our DFT method can reduce area overhead in such situation. We showed the effectiveness of our DFT method experimentally.

In Chapter 4, we have proposed a method of identifying false path using RTL information. The method identifies a subset of RTL false paths as control-dependent false path (CFPs) analyzing control signals such as select signal of a mux and road enable signal of a register. In the experimental results, we show that a large number of untestable path delay faults are identified in reasonable time. Moreover, SPC two-pattern tests never activate the identified path delay faults, but on the other hand normal two-pattern tests accidentally activate 10-60 % of them.

Finally, we discuss directions for our future works. An RTL circuit consists of a data path and a controller, and the two components are connected each other. The data path occupies the majority of the RTL circuit, therefore in our work, we first proposed a DFT method for the data path separated from the controller. Our future work is to propose a DFT method taking the delay testing between a controller and a data path into account. In our second work, a method of false path identification for the controller-datapath circuit was considered. However, non-separated RTL circuits can be also considered, which is called functional RTL description. For a functional RTL circuit, control signals of a mux and a register are not clear. False path identification for such circuits is our future work.

Acknowledgements

Many people have supported me during my Ph.D. studies. I would like to acknowledge their kind support.

I would like to thank to my supervisor, Professor Hideo Fujiwara, for his gracious guidance, a lot of advices and directions during my study.

I would also like to thank Professor Minoru Ito, for his valuable comments on this thesis.

I would like to thank to Associate Professor Michiko Inoue for her important and helpful suggestions on this work.

I am thankful to Assistant Professor Satoshi Ohtake for his frequent discussions, useful suggestions and a lot of advices.

I am also thankful to Assistant Professor Tomokazu Yoneda for his friendly discussions and comments.

I would like to thank Assistant Pfofessor Tsuyoshi Iwagaki of Japan Advanced Institute of Science and Technology for his kindness support.

I am grateful to Professor Tomoo Inoue of Hiroshima City University for his friendly advice and useful comments.

I am deeply thankful to the member of STARC (Semiconductor Technology Academic Research Center), Mr. Shunsuke Miyamoto (STARC), Mr. Jun Matsushima (Renesas), Mr. Kazuhisa Okada (Sharp), Mr. Mamoru Mukuno (Sanyo) and Mr. Yoichi Onisi (STARC) for their valuable comments.

This work was supported in part by STARC under the Research Project and in part by 21st Century COE (Center of Excellence) Program (Ubiquitous Networked Media Computing).

My thanks also go to the present and former members of our laboratory.

Finally, I wish to thank my parents and my brother for their continuing support and encouragement.

References

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

[2] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuit*, McGraw-Hill,Inc., 1994.

[3] J. Savir and S. Patel, "Scan-based transition test," IEEE Trans. on CAD, Vol. 12, No.8, pp. 1232-1241, Aug. 1993.

[4] J. Savir and S. Patel, "Broad-side delay test," IEEE Trans. on CAD, Vol. 13, No.8, pp. 1057-1064, Aug. 1994.

[5] 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.

[6] Tapan J. Chakraborty, Vishwani D. Agrawal, Michael L. Bushnell：“Design for testability for path delay faults in sequential circuits," Proc. DAC 93, pp. 453-457, 1993.

[7] B. T. Murray and J. P. Hayes, "Hierarchical test generation using pre computed tests for modules," IEEE Trans. on Computer Aided Design, VOL. 9, NO. 6, pp594-603, Jun. 1990.

[8] Md. Altaf-Ul-Amin, S. Ohtake and H. Fujiwara, "Design for hierarchical two-pattern testability of data paths," IEICE Trans. on Information and Systems, Vol. E85-D, No. 6, pp. 975-984, Jun. 2002.

[9] Md.Altaf-Ul-Amin, S.Ohtake and H.Fujiwara,"Design for two-pattern testability of Controller-Data Path Circuits," IEICE Trans.on Information and Systems,Vol.E86-D,No.6,pp.1042-1050,Jun.2003.

[10] T. Iwagaki, S. Ohtake and H. Fujiwara,"A design scheme for delay testing of controllers using state transition information," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences (Special Section

on VLSI Design and CAD Algorithms), Vol. E87-A, No. 12, pp.3200-3207, Dec. 2004.

[11] G. L. Smith, “Model for Delay Faults Based Upon Paths,” Proceeding of International Test Conference, pp. 342-349, Nov. 1985.

[12] M. A. Gharaybeh, M. L. Bushnell and V. D. Agrawal, “Classification and Test Generation for Path-Delay Faults Using Single Stuck-at Fault Tests,” Journal of Electronic Testing：Theory and Applications, Vol. 11, No. 1, pp. 55-67, Aug. 1997.

[13] Wei-Cheng Lai, Angela Krstic, Kwang-Ting(Tim), “On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set,” Proc. 18th VLSI Test Symp., pp. 15-20, 2000.

[14] H. Wada, T. Masuzawa, K. K. Saluja and H. Fujiwara, “Design for strong testability of RTL data paths to provide complete fault efficiency,” Proc. Int. Conf. on VLSI Design, pp. 300-305, 2000.

[15] V. Singh, M. Inoue, K. K. Saluja and H. Fujiwara, “Instruction-based delay fault self-testing of processor cores,” Proc. International Conference on VLSI Design, pp. 933-938, Jan. 2004.

[16] Y. Yoshikawa, S. Ohtake, M. Inoue and H. Fujiwara, “Design for Testability Based on Single-Port-Change Delay Testing for Data paths,” Proc. Asian Test Symp., pp. 254-259, 2005.

[17] T. J. Chakraborty, V. D. Agrawal, M. L. Bushnell, “Design for Testability for Path Delay Faults in Sequential Circuits,” Proc. 30th ACM/IEEE Design Automation Conference. pp. 453-457, 1993.

[18] A. Krstic. S. T. Chakradhar, K. T. Cheng, “Testable Path Delay Fault Cover for Sequential Circuits,” Proc. European Design Automation Conference. pp. 220-226, Sep. 1996

[19] M. Reddy, I. Pomeranz, S. Kajihara, “On Selecting Testable Paths in Scan Designs,” Proc. European Test Workchop. 2002.

[20] Y. Yoshikawa, S. Ohtake and H. Fujiwara, “An Approach to Reduce Overtesting of Path Delay Faults in Data Paths Using RT-level Information,” Digest of Papers, 11th IEEE European Test Symp., pp. 146-151, May. 2006.

[21] S. Padmanaban and S. Tragoudas, “A critical Path Selection Method for Delay Testing,” Proc. International Test Conf, pp. 232-240, 2004.

[22] K. T. Cheng and H. C. Chen, “Classification and identification of nonrobust untestable path delay faults”, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 8, pp. 845-853, Aug. 1996.

[23] 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, Jan. 1997.

[24] S. M. Reddy, S. Kajihara and I. Pomeranz, “An Efficient Method to Identify Untestable Path Delay Faults”, Proc. Asian Test Symp., pp. 233-238, Nov. 2001.

[25] K. Yang and K. T. Cheng, “Efficient Identification of Multi-Cycle False Path,” ASP-DAC pp. 360-365, 2006.

[26] 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.

[27] P. Maxwell, “The Design, Implementation and Analysis of Test Experiments”, Proc. International Test Conference, pp. 1-9, Nov. 2006