PAPER

Design for Hierarchical Two-Pattern Testability of Data Paths

Md. Altaf-Ul-AMIN[†], ***Nonmember***, **Satoshi OHTAKE**[†], ***Regular Member***, ***and*** **Hideo FUJIWARA**[†], ***Fellow***

SUMMARY This paper introduces the concept of hierarchical testability of data paths for delay faults. A definition of hierarchically two-pattern testable (HTPT) data path is developed. Also, a design for testability (DFT) method is presented to augment a data path to become an HTPT one. The DFT method incorporates a graph-based analysis of an HTPT data path and makes use of some graph algorithms. The proposed method can provide similar advantages to the enhanced scan approach at a much lower hardware overhead cost.

key words: *design for testability, delay testing, hierarchical testability, two-pattern testability*

1. Introduction

Two-pattern tests are necessary to detect delay defects in a circuit. The importance of detecting delay defects has soared in recent years to keep pace with the rapid increase in the speed of integrated circuits. A straightforward solution to two-pattern testability is the enhanced scan [1],[2]. But this incorporates very high area overhead and test application time. In the present work, we introduce hierarchical two-pattern testability of data paths. Hierarchical testability targeting stuck-at faults has been explored in a number of research works [3],[4]. These works address testability at register transfer level (RTL) and thus exploit the advantages of higher-level design hierarchy where the number of primitive elements in the design is greatly reduced. In these works it is shown that hierarchical testability is better than the gate-level based full-scan method in the context of test generation time, test application time and sometimes even area overhead. Our approach is hierarchical testability for delay faults. The design hierarchy we consider is RTL. At RTL a circuit can be divided into two parts: a controller and a data path. In this paper, we consider the data path only. We assume that all control inputs and status outputs of a data path are directly controllable and directly observable, respectively.

There are a number of delay fault models. Among these, the path delay fault model is more general and can overcome the limitations of other models. We develop HTPT data path targeting all detectable path delay faults. However, in the present work, we do not consider paths involving control inputs. The advantages of an HTPT data path are (i) the data path can be tested using any delay fault model, (ii) combinational ATPG can be used and (iii) the same fault coverage can be obtained as with the enhanced scan approach.

Analyzing the functionality of a circuit, some paths in the data path might be proven to be multiple clock tolerant paths. Delay in a multiple clock tolerant path is most likely to be caught by test for transition faults and need not be targeted for delay testing [5]. In this paper we developed our approach assuming that all paths of unity sequential depth are single clock tolerant. However, if some multiple clock tolerant paths exist and are discarded from the target fault list, our algorithm is still applicable and may result in less hardware overhead. The rest of our paper is organized as follows. Section 2 discusses the RTL path and its properties. Section 3 presents the definition and a graph based analysis of the HTPT data path. Section 4 explains our DFT method. Section 5 shows the experimental results. Section 6 concludes this paper.

2. The Concept of RTL Paths

A data path consists of hardware elements and buses. We assume that the buses in a data path are of the same bit width. However, we will relax this assumption afterwards. The RTL paths are the paths in an RTL circuit of a data path that, (i) start at a primary input (PI) and end at a register or (ii) start at a register and end at another/same (in case of feedback) register or (iii) start at a register and end at a primary output (PO) or (iv) start at a PI and end at a PO. It is obvious that the sequential depth of an RTL path is one. "PI1-R1" and "R1-ADD-MUX6-R4" are two examples of RTL paths in the arbitrary and simple data path of Fig. 1. We exclude the paths that start at a constant register. The reader may verify that the data path of Fig. 1 has total eighteen RTL paths.

In the lower hierarchy, each RTL path consists of a number of 1-bit wide paths. These individual 1-bit wide paths may be classified as robust, non-robust, functional sensitizable and functional unsensitizable paths. To guarantee the timing performance, it is necessary to test the robust, non-robust and functional sensitizable

Manuscript received October 2, 2001.
Manuscript revised January 21, 2002.

[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Ikoma-shi, 630–0101 Japan.

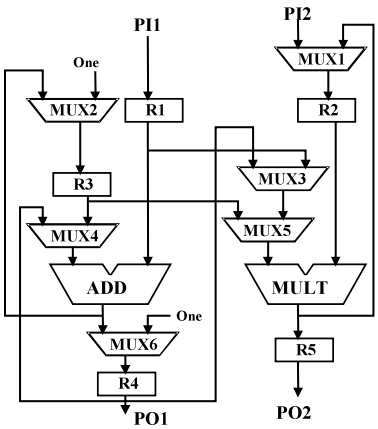


Fig. 1 An arbitrary data path.

(FS) paths [6]. To test a robust path, it is possible to find two vectors (a vector pair) that differ from each other by only at single bit [7]. An FS path needs to be tested in a group simultaneously with one or more other FS paths. Hence the testing of a group of FS paths requires two vectors, which may differ from each other at multiple bits [6]. From now on the testing of an RTL path will mean the testing of the robust, non-robust and functional sensitizable paths in it.

Path delay fault testing requires launching a transition at the start of a path by applying a pair of vectors, propagating the transition along the path and allowing fault effect observation from the end of the path [6]. However, many of the RTL paths in the data path neither start at a PI nor end at a PO. Therefore, some paths are necessary to ensure the flow of test data (test vectors and test responses) from PIs to appropriate registers and from appropriate registers to POs. Paths used for the flow of test vectors are referred to as *control paths* and paths used for test responses are referred to as *observation paths*. Any logic value can be propagated along the control paths and the observation paths. An RTL path may cross one or more multiplexers (MUXs) and operational modules. If an input of a MUX or an operational module is on an RTL path then this input is an *on-input*. Other input/inputs, which are not on the path, are called *off-inputs*.

2.1 Paths through a MUX

In a data path, MUXs are very common elements and are used as interconnecting units. Let us consider a 2 to 1 MUX as shown in Fig. 2. Both A and B are n-bit wide. C is the control input. If C selects A then, (i) propagation of signals from A $(A_1, \ldots, A_n)$ to O $(O_1, \ldots, O_n)$ is robust (off-inputs remain stable at non-controlling value) and independent of the signals at B and (ii) there is no merging gate among the paths (A_1 to O_1), (A_2 to O_2), $\ldots$, (A_n to O_n) i.e. there are only n mutually independent (1-bit wide) paths from

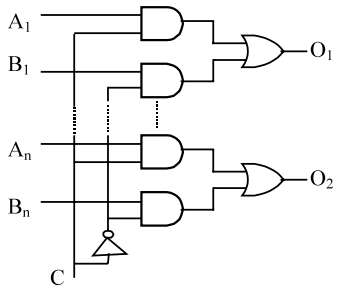


Fig. 2 n-bit wide 2 to 1 MUX.

A to O. The case for the paths from B to O is similar. Therefore, while testing any RTL path crossing one or more MUXs the select input/inputs should select the on-input/inputs of the MUX/MUXs and the off-input/inputs of the MUX/MUXs can be don't care. For example, to test the path "PI2-MUX1-R2" (Fig. 1), two-pattern vectors should be applied at $PI2$ and test responses should be captured at $R2$. The off-input of $MUX1$ may be don't care.

2.2 Paths through an Operational Module

Many RTL paths cross not only MUXs but also operational modules. In the following example, we discuss such a path.

Example 1: The path "R1-MUX3-MUX5-MULT-R5" in Fig. 1 crosses the operational module *MULT*. The segment "R1-MUX3-MUX5-" of this path is like a wire in a sense that the signal values at $R1$ appear unchanged at the output of the $MUX5$ if the control inputs of the MUXs select the on-inputs. Again the segment "-R5" on the output side of *MULT* is obviously like a wire. The core segment of this path is the part of the path inside the *MULT*. In other words the test vector set required to test the part of the path inside the *MULT* is the same as to test the whole path "R1-MUX3-MUX5-MULT-R5." These test vectors can be generated by separately considering the gate level circuit structure of the *MULT*. Obviously the bit width of these test vectors spans both inputs of the *MULT*. Suppose bits of the test vectors to be applied to the off-input of the *MULT* are not all don't cares. Hence to test the path "R1-MUX3-MUX5-MULT-R5," test vectors should be applied not only at $R1$ but also at the off-input of the *MULT*. Test vectors can be applied at the off-input of the *MULT* from register $R2$. Therefore, we say that the RTL path "R1-MUX3-MUX5-MULT-R5" is an HTPT path if, (i) there exist two control paths from PI/PIs to $R1$ and $R2$ that support the application of two-pattern vectors and (ii) there exists an observation path to propagate the test responses from $R5$ to a PO.

In Fig. 1, we can see that disjoint control paths "PI1-R1" and "PI2-MUX1-R2" are sufficient to apply

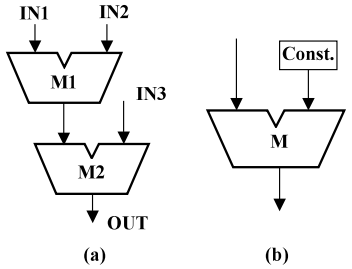


Fig. 3 (a) Chaining of modules, (b) A module connected to a constant register.

two-pattern vectors and the test response can be observed using the observation path "R5-PO2." It is noticeable that these control and observation paths can also test the RTL path "R2-MULT-R5." □

Though most of the operational modules commonly used in data paths have two inputs, there might be cases of chaining as shown in Fig. 3 (a). This type of module can be regarded as a 3-input operational module. Based on the similar reasons explained in Example 1 it can be realized that three control paths and only one observation path would be sufficient to test any RTL path that crosses such a module. Some modules in a data path may have their inputs connected to constant registers. If x inputs of an m-input operational module ($x < m$) are directly connected to constant registers then we regard such a module as an $(m - x)$-input operational module. The module M of Fig. 3 (b) is considered as a 1-input module.

3. The HTPT Data Path

In this section we define the HTPT data path and some other related terms. In the following subsections we present an analysis on HTPT data paths.

Definition 1: The *degree of an RTL path* is n, if it crosses an n-input operational module.

For example, the path "R1-MUX3-MUX5-MULT-R5" of Example 1 is an RTL path of degree 2. There might be paths in a data path that crosse no operational module. In Fig. 1, the path "PI2-MUX1-R2" crosses only a MUX and the path "PI1-R1" is simply a wire. These paths are considered as RTL paths of degree 1. The degree of an RTL path implies the number of control paths that are sufficient to ensure its hierarchical two-pattern testability.

Definition 2: An RTL path of degree n, which crosses an n-input operational module say M is an *HTPT path* if there exist n control paths C_1, $C_2, \ldots, C_n$ and an observation path P_1 such that, (i) C_1 is from a PI to the starting register of the path (in the case that it starts at a PI, C_1 is an empty path) and (ii) $C_2 \ldots C_n$ is from PI/PIs to a register/registers which are either directly or through MUX/MUXs connected to the $n - 1$ off-input/inputs of M (in the case that an off-input of M is connected to a PI, the corresponding control path is an empty path) and (iii) C_1, $C_2, \ldots, C_n$ support the application of two-pattern test and (iv) P_1 is from the ending register of the path to a PO (in the case that the path ends at a PO, P_1 is an empty path).

Definition 3: The set of control paths and an observation path that are sufficient to ensure the hierarchical two-pattern testability of an RTL path is referred to as the *test plan* of the path.

Definition 4: A data path is an *HTPT data path* if each of its RTL paths has a test plan.

3.1 Conditions for Control Paths to Support Two-Pattern Test

Here, we first briefly discuss the *thru* function [3]. The *thru* function allows the propagation of a logic value from an input to the output of an operational module without any change. For common two-input operational modules (e.g. adder, multiplier, etc.), a *thru* function from an input to the output can be realized by providing a constant at the other input. The necessary constant can be provided either by means of a support path or by adding a mask. For other modules *thru* functions can be realized by a MUX. Initially, we assume that a *thru* function exists from any input to the output of any operational module. Under this assumption, a control path can be represented by lines and the registers it crosses. However, such an assumption is not necessary for an HTPT data path and will be relaxed later on.

The general condition: Let $C_1, C_2, \ldots, C_n$ are n control paths starting at PI/PIs and ending at n different points EP1, EP2, ... , EPn (these ending points may be either registers or PIs). Let, a vector pair (v_1, v_2) span over n control paths. Hence each of v_1 and v_2 can be divided into n partial vectors. Each of the partial vectors is associated with a control path. Based on this, we represent v_1 and v_2 as $v_1 = v_{11}\&v_{12}\&\ldots v_{1n}$ and $v_2 = v_{21}\&v_{22}\&\ldots v_{2n}$ (the symbol '&' merely links the partial vectors). Bit-width of each of v_{11}, $v_{12}, \ldots v_{1n}$, v_{21}, $v_{22}, \ldots v_{2n}$ is equal to the bit-width of the data path. Let v_{11}, $v_{12}, \ldots v_{1n}$, v_{21}, $v_{22}, \ldots v_{2n}$ can be fed from PI/PIs according to a schedule such that v_{11}, $v_{12}, \ldots v_{1n}$ simultaneously appears at EP1, EP2,... EPn and in the next clock cycle v_{21}, $v_{22}, \ldots v_{2n}$ simultaneously appear at EP1, EP2,... EPn then we say that C_1, $C_2 \ldots C_n$ support the application of two-pattern test. □

The sequential depth of a control path is the number of registers that appear on the path. To denote the sequential depth of any control path say, C_1 we use the notation SD(C_1). In the following theorem we mention necessary and sufficient conditions for two control paths to support two-pattern test.

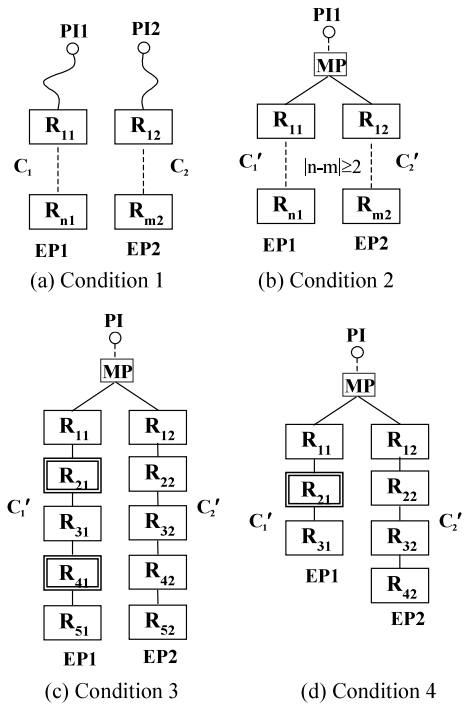


Fig. 4 Control paths depicting the conditions of Theorem 1.

Theorem 1: Two control paths C_1 and C_2 from PI/PIs to two different points EP1 and EP2 support the application of two-pattern test if and only if one of the following conditions is satisfied:

Condition 1: $C_1 \cap C_2 = \phi$ i.e. C_1 and C_2 are disjoint.

Condition 2: $|SD(C_1') - SD(C_2')| \geq 2$, where C_1' and C_2' are disjoint parts of C_1 and C_2 from EP1 and EP2 respectively to the nearest merging point of C_1 and C_2.

Condition 3: Either C_1' or C_2' crosses at least two hold registers.

Condition 4: When $|SD(C_1') - SD(C_2')| = 1$, the disjoint part (i.e. one of C_1' or C_2') with lower sequential depth crosses a hold register.

Proof: An arbitrary vector pair (v_1, v_2) involving two control paths can be represented as $v_1 = v_{11}\&v_{12}$ and $v_2 = v_{21}\&v_{22}$. In the following the sufficiency and the necessity of the above four conditions are discussed.

Sufficiency:

Condition 1: In Fig. 4 (a), m and n are two arbitrary numbers. Hence it represents a general topology that matches with Condition 1. Here, v_{11} and v_{21} can be fed from PI1 in consecutive clocks and similarly v_{12} and v_{22} can be fed from PI2. The general condition can be met by differing the feeding from PI1 and PI2 by $|m - n|$ clocks.

Condition 2: In Fig. 4 (b), m and n are two arbitrary numbers such that $|n - m| \geq 2$. Hence it represents a general topology that matches with Condition 2. All partial vectors should enter C_1' and C_2' clock by clock through the merging point MP. Hence any other merging between C_1 and C_2 before MP has no effect. Let $n > m$ and $n - m = r$. Therefore according to Condition 2, $r \geq 2$. The general condition can be met by first allowing v_{11} and v_{21} to enter C_1' in consecutive clocks and then after $r - 2$ clocks by allowing v_{12} and v_{22} to enter C_2' in consecutive clocks.

Table 1 Schedule of partial vectors corresponding to Fig. 4 (c).

Clk	MP	C_1'					C_2'				
		R_{11}	R_{21} (H)	R_{31}	R_{41} (H)	R_{51}/ EP1	R_{12}	R_{22}	R_{32}	R_{42}	R_{52}/ EP2
0	v_{11}										
1	v_{21}	v_{11}									
2	v_{12}	v_{21}	v_{11}								
3	v_{22}		v_{21}	v_{11}			v_{12}				
4			v_{21}		v_{11}		v_{22}	v_{12}			
5			v_{21}		v_{11}			v_{22}	v_{12}		
6				v_{21}	v_{11}				v_{22}	v_{12}	
7					v_{21}	v_{11}				v_{22}	v_{12}
8						v_{21}					v_{22}

Condition 3: Any of C_1' and C_2', say C_1' crosses two hold registers. First allowing v_{11} and v_{21} to enter C_1' and then allowing v_{12} and v_{22} to enter C_2' the general condition can be met. Here, v_{11} and v_{21} might be required to hold for one or more clock cycles in the hold registers of C_1' depending on the position of the hold registers and the number of registers in C_1' and C_2'. An example that matches with this condition is shown in Fig. 4 (c) and the corresponding scheduling of the application of the partial vectors is shown in Table 1. In Fig. 4, registers with double line border are hold registers. Referring to Table 1, v_{11} and v_{21} are fed to C_1' in the 1st and 2nd clock and v_{12} and v_{22} are fed to C_2' in the 3rd and 4th clock. Notice that v_{11} and v_{12} simultaneously appear at EP1 and EP2 in the 7th clock and v_{21} and v_{22} do the same in the following clock. The contents of the registers for certain clocks that are not important to this scheduling are not shown in the table.

Condition 4: Any of C_1' and C_2' say C_1' crosses a hold register and so according to Condition 4, $SD(C_2') - SD(C_1') = 1$. Now, first allowing v_{11} to enter C_1' and then immediately allowing v_{12} and v_{22} to C_2' and then again allowing v_{21} to C_1' the general condition can be met. An example that matches with this condition is shown in Fig. 4 (d) and the corresponding scheduling of the application of the partial vectors is shown in Table 2. The explanation of Table 2 is similar to that of the Table 1.

Necessity:

Two control paths C_1 and C_2 that do not fulfill any of the above four conditions should satisfy all the following properties.

Table 2 Schedule of partial vectors corresponding to Fig. 4 (d).

Clk	MP	C_1'			C_2'			
		R_{11}	R_{21} (H)	R_{31}/ EP1	R_{12}	R_{22}	R_{32}	R_{42}/ EP2
0	v_{11}							
1	v_{12}	v_{11}						
2	v_{22}		v_{11}		v_{12}			
3	v_{21}		v_{11}		v_{22}	v_{12}		
4		v_{21}	v_{11}			v_{22}	v_{12}	
5			v_{21}	v_{11}			v_{22}	v_{12}
6				v_{21}				v_{22}

1. C_1 and C_2 are not disjoint.
2. $|SD(C_1') - SD(C_2')| = 0$ or 1.
3. None of C_1' or C_2' crosses two hold registers.
4. When $|SD(C_1') - SD(C_2')| = 1$, the disjoint part (i.e. one of C_1' or C_2') with lower sequential depth does not cross a hold register.

All possible cases fulfilling the above properties are as follows.

1. C_1 and C_2 are not disjoint and $|SD(C_1') - SD(C_2')| = 1$ or 0 but none of C_1' and C_2' crosses any hold register.
2. C_1 and C_2 are not disjoint and $|SD(C_1') - SD(C_2')| = 0$ and any one or both of C_1' and C_2' crosses a hold register.
3. C_1 and C_2 are not disjoint and $|SD(C_1') - SD(C_2')| = 1$ and the disjoint part with higher sequential depth crosses a hold register.

However, no scheduling can obviously be found for these cases to fulfill the general condition. Hence Condition 1, Condition 2, Condition 3 and Condition 4 are the only conditions for two control paths to support the two-pattern test. □

In general case, any number, say n control paths can merge in numerous fashions. Each of these fashions may be of different nature depending on the number and position of hold registers in each control path. Hence to find out necessary and sufficient conditions for n control paths to support two-pattern test is a complex task. However in the following theorem some sufficient conditions are mentioned

Theorem 2: n control paths support the application of two-pattern test if one of the following conditions is satisfied:

1. All n paths are disjoint
2. Mutually disjoint parts of $n-1$ paths from their end points cross two hold registers.
3. The merging fashion of n paths after MP is a tree with MP as the root and the difference of sequential depths of any two partial paths from MP to their respective end points is two or more, where MP is the nearest merging point of all n paths from their end points.

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

Here, one thing is noteworthy. Let an RTL path P start at a register R and crosses an operational module M. For P to be an HTPT path, a control path from a PI to R is necessary. Now, if R is a feedback register (FR) to M, the control path must not incorporate a *thru* function to M. In Fig. 1, *R*3 is an FR to *ADD*. Only one control path from a PI to *R*3 is "PI1-R1-ADD-MUX2-R3." This control path cannot be used to test the path "R3-MUX4-ADD-MUX6-R4." It cannot be used because the first vector of a vector pair is loaded at *R*3 to settle the signal lines throughout the path including the part of the path in *ADD*. In the next clock, the second vector is loaded at *R*3 to launch the desired transition. However, the first vector cannot do its job if the second vector propagates using a *thru* function to *ADD*. Therefore, we should always carefully exclude such control paths.

3.2 Graph Based Analysis of HTPT Data Paths

All RTL paths that cross an operational module having two or more inputs can be represented as a graph. We call this graph the *structural connectivity graph* (SCG) of the module. The nodes of the SCG consist of the module and the registers, PIs and POs that are at the starting and at the ending of the RTL paths through the module. The edges of the SCG represent the connections of the module with the other nodes. Let the set of nodes connected to each of the inputs of an n-input module be R_{i1}, R_{i2},... R_{in} and the set of nodes connected to the output of the module is R_o. Let R_{i1} = {r_{11}, r_{21}, r_{31} ... }, R_{i2} = {r_{12}, r_{22}, r_{32} ... },... R_{in} = {r_{1n}, r_{2n}, r_{3n} ... } and R_o= {r_1, r_2, r_3 ... }. From the SCG of an n-input module an $(n+1)$-partite graph can be generated. We call this the *register compatibility graph* (RCG) of the module. R_{i1}, R_{i2},... R_{in} and R_o are the $n+1$ set of nodes. The edges of the RCG are determined by the following rules.

Rule 1: If there exist n control paths C_1, C_2... C_n from PI/PIs to any node $r_{j1} \in R_{i1}$, $r_{k2} \in R_{i2}$,... $r_{ln} \in R_{in}$, that support the application of two-pattern test, there exist edges between any two nodes of r_{j1}, r_{k2},... and r_{ln}.

Rule 2: If there exists an observation path from any node $r_m \in R_o$ to a primary output, there exist edges between r_m to any node in R_{i1} and R_{i2}, and ... R_{in}.

Example 2: The SCG of *MULT* of Fig. 1 is shown in Fig. 5 (a). Figure 5 (b) shows the RCG of *MULT* generated under the assumption that a *thru* function exists from any input to the output of any operational module. Table 3 shows that which edge/edges in Fig. 5 (b) are related to which control or observation path/paths. □

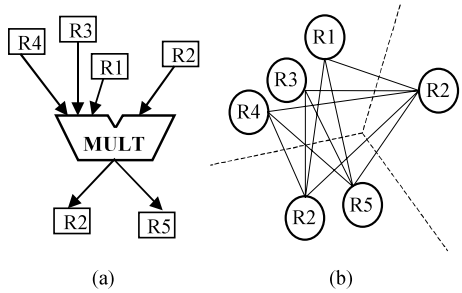


Fig. 5 (a) SCG and (b) RCG of *MULT* of Fig. 1.

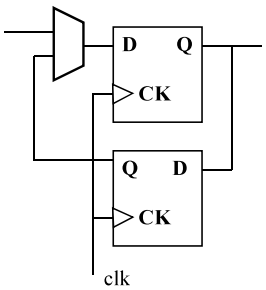


Fig. 6 Rotating enhanced flip-flop.

Table 3 List of edges of the graph in Fig. 5 (b) and corresponding paths.

R1-R2	PI1-R1 And PI2-MUX1-R2
R3-R2	PI1-R1-ADD-MUX2-R3 And PI2-MUX1-R2
R4-R2	PI1-R1-ADD-MUX6-R4 And PI2-MUX1-R2
R1-R5, R2-R5, R3-R5, R4-R5	R5-PO2
R1-R2, R2-R2, R3-R2, R4-R2	R2-MULT-R5-PO2

Definition 5: An edge in the RCG of a module from any node in R_o to any node in R_{i1}or $R_{i2,}\ldots$ orR_{in}, actually represents an RTL path through the module and hence we refer to such an edge as a *path edge*.

Definition 6: An edge in the RCG between any two nodes of R_{i1},$R_{i2,}\ldots$ andR_{in} is a *control edge*.

Definition 7: If a sub-graph of the RCG of a module with only one node from each of R_{i1}, $R_{i2,}\ldots R_{in}$ and R_o is a clique, then this is referred to as a *test clique*.

A test clique implies the existence of n control paths and one observation path, which are sufficient to test n RTL paths passing through n inputs of the module, i.e. a test clique in the RCG of an n-input module incorporates test plan for n paths.

Theorem 3: All RTL paths through a module are two-pattern testable if the path edges corresponding to all RTL paths through the module exist in its RCG and each path edge is part of some test clique.

Proof: If a path edge is part of a test clique, then this test clique is sufficient to guarantee the two-pattern testability of the corresponding path. Now if each path edge is part of some test clique, then each RTL path passing through the module is two-pattern testable. □

4. Details of the DFT Method

This section discusses the DFT elements we utilize in our approach and also the algorithm for efficient addition of the DFT elements to augment a data path to an HTPT one.

4.1 The DFT Elements

We consider three types of DFT elements in our approach. They are MUXs, *thru* functions and rotating enhanced flip-flops (REFFs). The operation of a MUX is well known. We briefly explained about the *thru* function in Sect. 3.1. An REFF consists of two flip-flops and a MUX as shown in Fig. 6. The control input of the MUX is used as the mode selector. In normal mode the REFF behaves like a normal flip-flop. Using normal mode two bits can be loaded to the REFF. Just after loading two bits, the mode can be changed to test mode. In test mode the bits exchange their position at every clock but remain stored in the REFF. Hence an REFF can be regarded as a 2-bit hold register. Of a two-pattern vector, the bit of which vector should be loaded first to the REFF depends on the time of loading and the time of applying the vectors.

4.2 Algorithm for Adding DFT Elements

Figure 7 illustrates the flowchart of our algorithm. In the following subsections we describe in detail the techniques and heuristics we use for different steps of the flowchart. For simplicity we describe our algorithm assuming that operational modules in the data path has a maximum two inputs and there is no chaining of modules. This means there is no RTL path of degree 3 or more in the data path. Most of the benchmarks satisfy this condition. Also, our approach can be extended for data paths with modules having more than two inputs.

4.2.1 Selecting Potential Control and Observation Paths

Every register (other than the constant registers) of a data path is at the starting of some RTL path and also at the ending of some RTL path. Control paths are therefore necessary from PIs to all registers and so are the observation paths from all registers to POs. However the number of paths in the data path can be very large and we use some heuristics to select some potential control and observation paths. We make a set CP of potential control paths and a set OP of potential

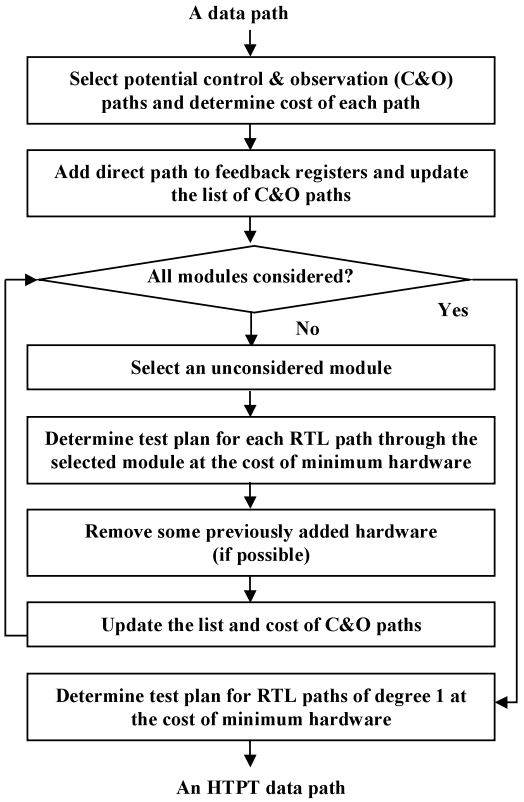


Fig. 7 Flow-chart of the algorithm.

observation paths for each register R. The first members of CP are the shortest depth paths from each PI to R. We favor the shortest depth paths because they help to reduce the test application time. To determine the shortest depth control paths, we represent the data path as a port digraph [8]. The digraph of the data path of Fig. 1 is shown in Fig. 8 (a). A node represents a port in the data path and any edge say, (u, v) implies that either a metal line connects u to v or u and v are input port and output port, respectively of the same element. The shortest depth control paths can be selected by performing a modified *breadth first search* (BFS) on the digraph. The BFS is performed n times where n is the number of PIs of the data path. Each of n PIs are once considered as the source node. Figure 8 (b) shows the breadth first tree (thick lines) of the digraph of Fig. 8 (a) considering PI1 as the source node. This tree contains the shortest depth paths from PI1 to each register reachable from PI1. Similarly the members of OP of any register R are the shortest depth paths from R to each PO. The shortest depth observation paths can be determined by reversing the edges of the digraph and performing similar BFS m times where m is the number of POs. For each member of CP and OP of each register we maintain a variable, which we refer to as the "cost" of the path. The cost of the path is nothing but the number of *thru* functions associated with the path, i.e. the cost of a *thru* function is 1.

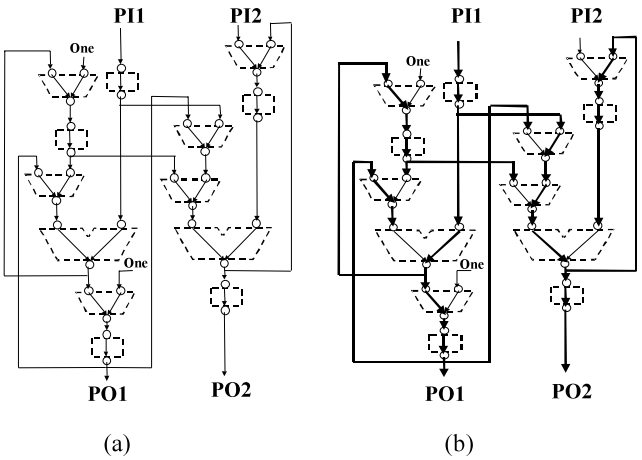


Fig. 8 (a) Port digraph of the data path of Fig.1, (b) Breadth first tree of the digraph of fig.8(a).

4.2.2 Adding Direct Paths to Feedback Registers

We first consider the FRs of all modules. Suppose a register R_f appears on both input and output sides of the SCG of a module M. This implies that R_f is an FR to M. Now, if a control path from a PI to R_f incorporates a *thru* function to M, then this control path cannot be used to test any RTL path that crosses M (explained in Sect. 3.1). So we first find in the CP of R_f for a control path without a *thru* function to M. If we succeed, we switch to another FR of M or FR of other module. If no such control path is found in the CP of R_f we search the entire data path. For this search, we remove the edges from inputs to output of M in the digraph and then find shortest path from each PI to R_f until a path is found. If a path is found, this path is added to the CP of R_f. However the failure of such a search implies that there is no control path from any PI to R_f without a *thru* function to M. Under such a situation a direct path is added to R_f using a test MUX from a PI. This PI is chosen based on the control paths in CPs of the registers connected to the other input of M (input to which R_f is not connected). If the majority of the nodes connected to the other input of M have control path from a PI, we choose other PI for R_f. This direct path is included to the CP of R_f. In case the data path has single input, it might be difficult to find a suitable PI. In such a situation we augment each flip-flop of R_f to REFF. This information is added to the paths in the CP of R_f. We then switch to another FR of M or FR of another module until all FRs in the data path are considered. We start to consider the FRs of the modules nearer to PIs first. Sometimes a single MUX can be used for more than one FR of a module. In Fig. 9, the test MUX is providing control paths from $PI2$ to both $R3$ and $R4$. In the case of a single input Data path, if it becomes necessary to augment flip-flops of more than one register to REFF, we use a global

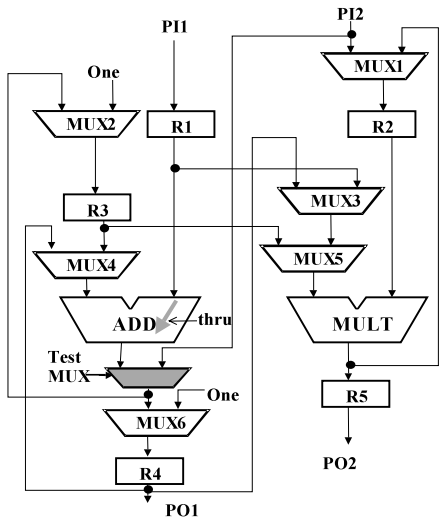


Fig. 9 The HTPT equivalent of the data path of Fig. 1.

REFF register connected to the only PI of the data path. Such an REFF register can be used as a PI for two-pattern testing. This is because an REFF can store two bits for as long as it is necessary. Direct paths then can be added from the global REFF register to FRs using test MUXs. We use global REFF because the hardware overhead incurred by registers is very high. Whenever we add any DFT element, we update the CPs of registers affected by the addition. Addition of test MUXs and REFF registers creates some more paths of degree 1 in the data path. We should also consider the two-pattern testability of these paths. Because of adding DFT elements in this step, the SCGs of modules in the data path remain unchanged.

4.2.3 Determining Test Plans

We discussed our algorithm for adding some DFT elements in the previous section. In this section, we add more DFT elements to ensure the test plans for all RTL paths. First we address the RTL paths of degree 2 and then of degree 1.

RTL paths of degree 2: We consider all RTL paths passing through a module at a time. The following four steps are performed for each 2-input operational module.

Generating RCG from SCG (step 1): Let M be a 2-input module and R_{i1}, R_{i2} and R_o are the set of nodes connected to the left input, right input and output of the module. Each member of R_{i1} and R_{i2} has a set of control paths CP and each member of R_o has a set of observation path OP. Let $r_{j1} \in R_{i1}$ and $r_{k2} \in R_{i2}$. Now we look for two control paths, one from CP of r_{j1} and other from CP of r_{k2}, such that they satisfy any one of the four conditions of Theorem 1. In case we find more than one pair of control paths to satisfy any condition of Theorem 1, we choose the pair having lowest cost. We add a control edge between r_{j1} and r_{k2} in the RCG and keep a record of the lowest cost control paths that support this edge. We should keep in mind that in the case that r_{j1} and r_{k2} is a FR to M, we cannot consider a control path for r_{j1} or r_{k2} incorporating a *thru* function to M unless r_{il} or r_{jr} is augmented to an REFF register. If we fail to find a control edge between r_{j1} and r_{k2} we switch to another pair of nodes. We search for control edges between every possible pair of nodes with one node from R_{i1} and one node from R_{i2}. For each member of R_o, we choose the lowest cost observation path from its OP and add path edges to the RCG as described in rule 2 in Sect. 3.2.

Adding DFT elements to Augment RCG (step 2): In the RCG, some node/nodes of R_{i1} or R_{i2} may not be connected to any control edge. For any such node we add DFT elements to create a control edge in the RCG incorporating this node. First we try by adding a direct path to such a node from some PI by means of a test MUX. In the worst case we augment such a node to an REFF register. However, if it becomes necessary to augment more than one register, we create a global REFF register as mentioned in Sect. 4.2.2. Direct paths can then be created from the global REFF register to the required nodes by means of test MUXs. The RCG of M is now sufficient for two-pattern testability of any RTL path through it. However, when a direct path to a node is created, we check whether this path can be utilized to remove some DFT elements added for the previously processed modules. A direct path to a node might be used instead of some other necessary control path/paths to the same node, which incorporates a *thru* function.

Minimizing control edges in RCG (step 3): The RCG may have some excess control edges. Of all the control edges we select the minimum number of them that fulfill the condition that each node in R_{i1} or R_{i2} is connected to at least one control edge. The minimum number of control edges will create the minimum number of test cliques in the RCG that are sufficient for test plans of RTL paths through M.

Adding DFT elements for thru Functions (step 4): At this step, we relax the assumption that a *thru* function exists from any input to the output of any operational module. We first check whether a *thru* function that we need really exists (can be implemented by a support path) or whether we should add a DFT element (mask or others) for it. To do this we consider one test clique at a time. A test clique in the RCG of a two-input operational module incorporates two control paths and one observation path. We search for support paths for the *thru* functions associated to these three paths using some manual calculation. A support path may have a timing conflict with control paths or other support paths. Timing conflict means it becomes necessary to provide different values at the same PI at the same time. We add mask elements to realize the

thru functions for which any support path cannot be found without timing conflict. The cost of the *thru* function realized by a mask element becomes zero and we update the cost of all control and observation paths, which includes this *thru* function. We consider all the test cliques in the RCG of M in a similar way.

The above four steps are performed for all 2-input modules considering the modules nearer to the PIs first. When we finish all the 2-input modules, the testability of all the RTL paths of degree 2 is ensured.

RTL paths of degree 1: We now consider the testability of RTL paths of degree 1. Let R_s and R_e be the start and end registers of an RTL path of degree 1. We choose the lowest cost path from the CP of R_s as the control path and the lowest cost path from the OP of R_e as the observation path. If any *thru* function of cost 1 is associated with these paths, we try to realize it using a support path. In case we cannot find any support path without timing conflict, we add a mask element or a multiplexer to realize them. We consider all RTL paths of degree 1 in a similar way. Figure 9 shows the HTPT equivalent of the data path of Fig. 1 augmented by following the algorithm described above.

4.3 Relaxing the Assumption of Bit Width

In practice, a data path may not be of consistent bit width. It is not a problem, if a control path becomes gradually narrower or an observation path becomes gradually wider in their respective downstream. However, in the case of a control path, it is a problem if it is narrower than its end point anywhere in the upstream. Such a problem can be tackled by means of a MUX or an REFF and a MUX. In the case of an observation path, it is a problem if it becomes narrower than its start point anywhere in the downstream. Such a problem can be tackled by using one or more MUXs.

5. Experimental Results

In this section we present experimental results to compare our method with the enhanced scan approach. For comparing area overhead we applied our method to data paths of three benchmark circuits and a RISC processor provided by industry. The characteristics of these data paths are shown in Table 4. In this table PIs and POs denote the number of primary inputs and primary outputs of the data path respectively. REGs, MUXs and OPs are the numbers of registers, multiplexers and operational modules in the data path. The last column of Table 4 shows the areas of the data paths generated by the logic synthesis tool Design Compiler (Synopsys).

Table 5 shows the results regarding hardware overhead. In both our method and the enhanced scan approach, the percentage of area overhead decreases with the increase in bit width of the data paths. However, the area overhead incurred by our method is always much lower. For our DFT method, the columns MUX, *thru* and REFF show the number of added MUXs, *thru* functions and REFF registers.

Table 4 Circuit characteristics.

Circuit	Bit width	PIs	POs	REGs	MUXs	OPs	Area
Paulin	16	2	2	7	11	4	10528
LWF	16	1	1	5	5	3	3512
Tseng	16	3	2	6	7	7	7802
RISC	32	1	3	40	84	19	94357

Table 5 Hardware overhead.

Circuit	Bit width	Enhanced scan	Our method			
		Hardware Overhead (%)	Hardware Overhead (%)	MUX	*thru*	REFF
Paulin	8	42.77	8.52	3	2	0
	16	27.66	5.53			
	32	16.23	3.36			
LWF	8	65.99	12.75	0	0	1
	16	59.23	11.28			
	32	56.55	10.88			
Tseng	8	42.39	4.76	1	2	0
	16	31.99	3.93			
	32	21.65	2.62			
RISC	32	35.27	2.04	2	2	1

Table 6 Test application time for 100% coverage of robust and non-robust testable paths (cycles).

Circuit	Bit width	Enhanced scan	Our method
Paulin	8	442888	5950
LWF	8	61580	7510
Tseng	8	106720	2936

The tool we used for the experiments to compare test application time are DelayPath and TestGen from Synopsys. DelayPath can generate path list for designs of up to 5000 gates. And TestGen can generate tests for only robust and nonrobust testable paths. Because of these limitations we conducted our experiments for the data path of Paulin, LWF and Tseng considering their bit width only 8. Also, test vectors that cover only robust and non-robust testable paths are considered. 100% coverage is obtained for robust and non-robust testable paths in both our method and the enhanced scan approach. Identifying and generating tests for functional sensitizable paths in a large multi-level circuit is a hard task [6]. However, if test can be generated for such paths, both our method and the enhanced scan approach allow their application.

Table 6 shows the results of our experiments regarding test application time. For all three data paths the test application time of our method is much shorter compared to that of the enhanced scan approach. This is because, in the enhanced scan approach, the test vectors and test responses are shifted in to and shifted out from the enhanced scan chain serially. Contrary to this our method supports the parallel propagation of test

vectors and test responses via data path lines.

6. Conclusions

The concept of hierarchical testability for delay faults is introduced in this paper. Precomputed vector pairs are propagated via control paths from primary inputs to appropriate locations and are applied in a two-pattern fashion to test paths of unity sequential depth. Test responses are propagated via observation paths from the end points of paths under test to primary outputs. A DFT method is also presented that can be applied to augment a data path to become a hierarchically two-pattern testable one. Priority has been given to use the existing paths in the data path as control paths and observation paths. Experimental results show that the area overhead and test application time of our DFT method is smaller compared to those of the enhanced scan approach for some benchmark data paths.

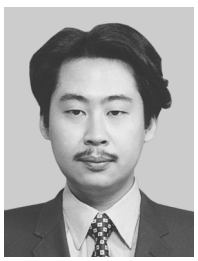
Acknowledgments

This work was sponsored in part by NEDO (New Energy and Industrial Technology Development Organization) through the contract with STARC (Semiconductor Technology Academic Research Center) and supported by Japan Society for the Promotion of Science (JSPS) under the Grant-in-Aid for Scientific Research and by Foundation of Nara Institute of Science and Technology under the grant for activity of education and research.

References

[1] B.I. Dervisoglu and G. E. Stong, "Design for testability: Using scan path techniques for path-delay test and measurement," Proc. Int. Test Conf., pp.365–374, 1991.
[2] S. Dasgupta, R.G. Walther, and T. W. Williams, "An enhancement to LSSD and some application of LSSD in reliability, availability and serviceability," Proc. Fault Tolerant Computing Symp., FTCS-11., pp.32–34, 1981.
[3] S. Ohtake, H. Wada, T. Masuzawa, and H. Fujiwara, "A non-scan DFT method at register-transfer level to achieve complete Fault efficiency," Proc. ASP-DAC, pp.599–604, 2000.
[4] I. Ghosh, A. Raghunathan, and N. K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst., vol.16, no.9, pp.1001–1014, 1997.
[5] W.-C. Lai, A. Krstic, and K.-T. Cheng, "Functionally testable path delay faults on a microprocessor," IEEE Design & Test of Computers, pp.6–14, Oct.-Dec. 2000.
[6] A. Krstic and K.-T. Cheng, Delay Fault Testing for VLSI Circuits, Kluwer Academic Publishers, 1998.
[7] W. Wang and S.K Gupta "Weighted random robust path delay testing of synthesized multilevel circuits," Proc. 1994 IEEE VLSI Test Symp., pp.291–297, 1994.
[8] 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.

Md. Altaf-Ul-Amin received his B.Sc. degree in electrical and electronic engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka and M.S. degree in electrical, electronic and systems engineering from Universiti Kebangsaan Malaysia (UKM). Currently he is pursuing his PhD degree in Nara Institute of Science and Technology (NAIST), Japan. His research interests are design and design for testability of digital, analog and mixed-mode VLSI circuits.



Satoshi Ohtake received the B.E. degree in computer science from the University of Electro-Communications, Tokyo, Japan, in 1995, and M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 1997 and 1999, respectively. He was a Research Fellow of the Japan Society for the Promotion of Science from 1998 to 1999. Presently he is an Assistant Professor of Graduate School of Information Science, Nara Institute of Science and Technology. His research interests are VLSI CAD, design for testability, delay test and test pattern generation. He is a member of IEEE Computer Society.

Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Award in 1991, 2000 and 2001, Okawa Prize for Publication in 1994, IEEE Computer Society Meritorious Service Award in 1996, and IEEE Computer Society Outstanding Contribution Award in 2001. He is an advisory member of IEICE Trans. on Information and Systems and an editor of IEEE Trans. on Computers, J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a member of the Information Processing Society of Japan.