A Non-scan DFT Method at RTL Based on Fixed-control Testability to Achieve 100% Fault Efficiency

Satoshi Ohtake Shintaro Nagai Hiroki Wada Hideo Fujiwara

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0101, Japan

E-mail: {*ohtake, shinta-n, hiroki-w, fujiwara*}@*is.aist-nara.ac.jp*

Abstract *This paper proposes a non-scan design-for-testability method for register-transfer level circuits where a circuit consists of a controller and a data path. It achieves complete fault efficiency with low hardware overhead and at-speed testing.*

1. Introduction

With the advance in semiconductor technology, the complexity of VLSI designs is growing and the cost of testing is increasing. Therefore, it is necessary to reduce the cost of testing and to enhance the quality of testing. The cost of testing is estimated by test generation time and test application time. The quality of testing is estimated by fault efficiency[1]. Therefore, we have to reduce test generation time and test application time and to enhance fault efficiency. To ease the complexity of test generation, design-for-testability (DFT) techniques have been proposed. The most commonly used DFT techniques for sequential circuits are scan-based approaches[1]. These techniques modify sequential circuits so that automatic test pattern generation (ATPG) tools can achieve high fault efficiency in a reasonable time. However, these techniques sacrifice the possibility of at-speed testing[2] for fault efficiency enhancement. To avoid this disadvantage of scan techniques, several non-scan approaches have been investigated. On the other hand, since techniques of test generation and DFT at gate level face the problems arising out of huge number of elements and high complexities of the circuits at gate level, several techniques of test generation and DFT at register-transfer level (RTL) have been proposed recently.

In RTL design, a VLSI circuit is generally consists of two separate parts, a controller part and a data path part. The former is represented by a state transition graph (STG) and the latter is represented by hardware elements (e.g. registers, multiplexers and operational modules) and signal lines connecting them. A controller and a data path are interconnected by internal signals: *control signals* and *status signals*. Most of the DFT methods for RTL circuits were concerned with only either data paths or controllers, on the assumption that the control signals and the status signals are directly controllable and observable from the outside of the VLSI.

For controllers, Chakradhar et al.[3] proposed a non-scan DFT method at RTL. This method can achieve high fault efficiency but cannot always guarantee complete (100%) fault efficiency. Furthermore, it is applicable only to PLA-based sequential circuits. Our previous work [4, 5] proposed several non-scan DFT methods which achieve 100% fault efficiency. In these methods, given an STG, we first synthesize a sequential circuit from the STG. Then we generate test patterns, by a combinational ATPG tool, for the combinational part of the synthesized sequential circuit. Most of the generated test patterns can be applied to the combinational part of the sequential circuit using state transitions of the STG. However, there may exist some test patterns which cannot be applied using state transitions of the STG. In this case, we append an extra logic which provides extra transitions required for testing. Those test patterns are applied using the extra logic.

For data paths, several non-scan DFT methods at RTL have been reported, e.g. orthogonal scan[6, 7] and H-SCAN[8], which use normal data path flow as scan path instead of traditional scan path flow. These methods can reduce hardware overhead and test application time compared to the full-scan design. However, test generation time cannot be reduced because the test generation approach is the same as the full-scan design. To reduce test generation time, a *hierarchical test generation* approach was proposed by Murray and Hayes[9]. The hierarchical test generation of a data path consists of the following two steps: for individual combinational hardware elements, generate test patterns at gate level and generate *test plans* at RTL, where a test plan is a control sequence to propagate test patterns for a combinational hardware element from the primary inputs to the inputs of the hardware element and to propagate responses from the output of the hardware element to the primary outputs. Genesis [10]–[13] is an approach based on such a hierarchical test generation for data paths. Our previous work [14] presented a DFT method based on such hierarchical test generation and *strong testability*. Strong testability is a property of data paths which guarantees to generate test plans for all combinational hardware elements of the data path.

In our previous work [15], we proposed a DFT method for

[1] Fault efficiency is the ratio of the number of faults detected and proved redundant to the total number of faults.

an RTL circuit which consists of a controller part and a data path part. Given an RTL controller/data path circuit, we apply the DFT method of [4] to the controller part and apply that of [14] to the data path part. Our previous experimental results using benchmark circuits show that the hardware overheads of [4] and [14] were 3.5% and 4.0% on average for benchmark circuits, respectively. The test application times in [4] and [14] were reduced on average to 25.4% and 17.6% of the full-scan design, respectively. Furthermore, both of these DFT methods can achieve 100% fault efficiency and allow at-speed testing.

In the above-mentioned DFT methods, we assumed that both control signals and status signals between a controller and a data path are directly controllable and observable from the outside of circuits. However, if we consider a DFT method for the whole circuit consisting of both a controller and a data path, we have to remove this assumption by adding some extra logic to provide both controllability and observability of those control and status signals. In our previous work [15], we resolved this problem by (1) adding multiplexers on those control and status signals to connect directly from primary inputs and to primary outputs and (2) embedding an extra circuit in the controller side, called a *test plan generator*, which can generate test plans for the data path of an RTL circuit.

The proposed DFT method for controller/data path circuits has the following advantages:

- 100% fault efficiency can be achieved.
- At-speed testing can be performed.

Furthermore, from our experimental results,

- Test application time can be reduced significantly compared to the full-scan design.
- Test generation time can be reduced significantly compared to the full-scan design.

However, the proposed method has the following disadvantage:

- The hardware overhead is larger than that of the full-scan design.

The hardware overhead of the proposed method is dominated by extra logic corresponding to test plan generators for strongly testable data paths.

In this paper, we present a new property of circuit structure of data paths called *fixed-control testability*. If a data path is fixed-control testable, hierarchical test generation can be applied and each test plan of combinational hardware elements can be composed of at most three control vectors. Therefore, a design of a test plan generator which generates test plans of the fixed-control testable data path is simpler than that of strongly testable one.

In this paper, we also propose a DFT method for data paths which makes a data path fixed-control testable and a test architecture for a whole circuit consisting of both a controller modified by [4] and a data path modified by the DFT method based on fixed-control testability proposed in this paper.

This paper is organized as follows: Section 2 gives the definition of controller/data path circuits and Section 3 shows overview of our DFT method. In Section 4, we introduce the DFT method of controllers reported in [4]. Section 5 presents a DFT method of data paths based on fixed-control testability. Section 6 presents a DFT method for whole circuits consisting of both controllers and data paths. Section 7 shows experimental results of the proposed method using some benchmark circuits and shows that our method proposed in this paper can reduce hardware overhead compared with that in [15].

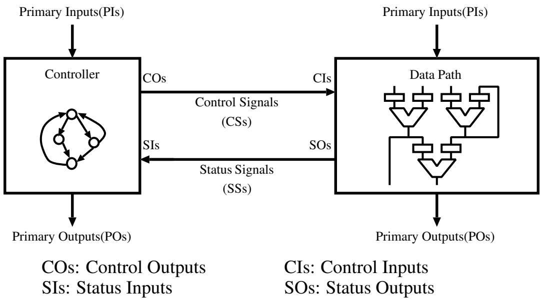


Figure 1: An RTL controller/data path circuit.

2. Preliminaries

In RTL description, a VLSI circuit generally consists of a controller and a data path as shown in Figure 1. The former is represented by an STG and the latter is represented by hardware elements (e.g. registers, multiplexers and operational modules) and signal lines connecting them. Each of the controller and the data path has *primary inputs* from the outside of the VLSI and *primary outputs* to the outside of the VLSI. The controller also has *status inputs* from the data path and *control outputs* to the data path. Similarly, the data path also has *control inputs* from the controller and *status outputs* to the controller. The signals from the controller to the data path are called *control signals*, and the signals from the data path to the controller are called *status signals*.

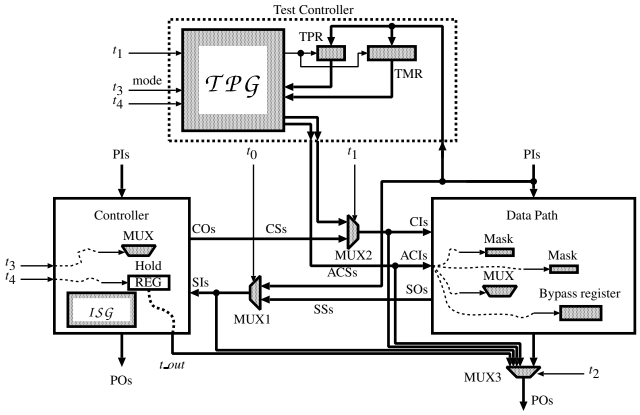
Data path

A data path consists of *hardware elements* and *signal lines*. Hardware elements are primary inputs, primary outputs, control inputs, status outputs, registers, multiplexors, and operational modules. A signal line connects two hardware elements with some bit width. Inputs of a hardware element in the data path can be classified into *data inputs* and *control inputs*. Examples of the control inputs are load enable signals of registers, selection signals of multiplexers and function selection signals of operational modules. Similarly, outputs of a hardware element of a data path can be classified into *data outputs* and *status outputs*. Comparators are examples of hardware elements having status outputs.

The following restrictions are introduced into our data path architecture in order to simplify the discussion, though they can be relaxed.

A1: All signal lines in the data path have the same bit width.

A2: An operational module has only one or two data inputs and only one data output.



ACSs: Additional Control Signals
ACIs: Additional Control Inputs
$\mathcal{TPG}$: Test Plan Generator
TPR: Test Pattern Register
TMR: Target Module Register
t_i: Test Pin

Figure 2: Test architecture of a controller/data path circuit.

A3: For any data input, there exists a path from a primary input. And for any data output, there exists a path to a primary output.

A4: Control inputs of a hardware element are connected directly to control inputs of the data path. And status outputs of a hardware element are connected directly to status outputs of the data path.

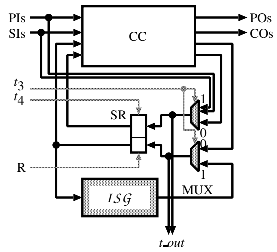
3. Overview

In our DFT method, given a controller/data path circuit described at RTL, we first apply the DFT method of [4] to the controller and apply the DFT method proposed in Section 5 to the data path of the circuit. In these DFT methods, we assume that the control signals and the status signals are directly controllable and observable from the outside of the circuit. However, these are internal signals between the controller and the data path in the controller/data path circuit. Thus, for testing of the controller, we have to enhance controllability of the status inputs and observability of the control outputs. Similarly, for testing of the data path, we have to enhance controllability of the control inputs and observability of the status outputs. In the DFT method proposed in this paper, we embed mechanisms to enhance controllability and observability of the control signals and the status signals in the same way as the method of [15] so that the testing methods of controllers and data paths can be applied. The test architecture of the controller/data path circuit of our method is shown in Figure 2. The circuit is configured by controlling test pins as shown in Table 1. Before explaining the details of the test architecture, we briefly introduce the DFT method for controllers in the next section and propose the DFT method of data paths that is the basis of the test architecture in Section 5.

Table 1: Configurations of test architecture.

Test Pins					Operation
t_0	t_1	t_2	t_3	t_4	
0	0	0	0	0	Normal operation
1	0	1	$*$	$*$	Testing controller
0	1	$*$	$*$	$*$	Testing data path

$*$: depend on test patterns or test plans



CC: combinational logic block
SR: state register
$\mathcal{ISG}$: invalid test state genarator
t_3 : mode switching signal
t_4 : load/hold signal
t_out : state output signals

Figure 3: A controller augmented with an extra logic $\mathcal{ISG}$.

4. DFT Method for Controllers [4]

In this section, we give an overview of the DFT method [4] for a controller synthesized from an STG. The method achieves 100% fault efficiency with short test generation time and allows at-speed testing. In order to generate a test sequence that achieves 100% fault efficiency with short test generation time, we generate test patterns for the combinational part of the sequential circuit using a combinational ATPG tool. Each of test patterns consists of the values corresponding to primary inputs (PIs), status inputs (SIs) and a state register (SR) of the sequential circuit. In order to apply a test pattern to the combinational part, we have to set the corresponding value to the SR. If the value corresponds to a state reachable from the reset state of the STG, the value can be set to the SR using the original state transitions of the STG. Otherwise, the value cannot be set to the SR using the original state transitions of the STG. In order to set such a value to the SR, we append an extra logic called an *invalid test state generator*($\mathcal{ISG}$) to the controller as shown in Figure 3. The $\mathcal{ISG}$ generates all the values (called *invalid test states*) that appear in the test patterns but cannot be set to the SR using state transitions of the STG. In Figure 3, t_3 is used to select inputs of the SR: the outputs of the original combinational logic block or those of the $\mathcal{ISG}$ and t_4 is a load/hold signal and is utilized to reduce test application time.

The complete fault efficiency is preserved because the combinational logic block remains unchanged. Our experimental results using benchmark circuits show that the average hardware overhead of the $\mathcal{ISG}$ is only 3.5% and the average test application time of the method is 25.4% of that of the full-scan design[4].

5. DFT Method for Data Paths

In this section, we propose a new DFT method for RTL data paths.

5.1 Fixed-Control Testability

The DFT method is based on *fixed-control testability* which is a subclass of *strong testability*. Strong testability is proposed as a characteristic of data paths that guarantees applicability of *hierarchical test generation*[9]. Hierarchical test generation is a promising way for testing very large sequential circuits. In the hierarchical test generation, testing for each hardware element M proceeds as follows.

Step 1: Test patterns are generated for M (a combinational circuit) using a combinational ATPG tool.

Step 2: The test patterns are applied to M: the values are fed through primary inputs at appropriate times, so that the desired test patterns can be applied to M.

Step 3: The responses of M to the test patterns are propagated to primary outputs for observation.

A *test plan* specifies the control signals so that the test patterns and the responses can be propagated. Strong testability of a data path is defined as follows.

Definition 1: *Strong Testability [14]*
A data path is *strongly testable* if there exists a test plan for each combinational hardware element M that makes it possible to apply any pattern to M and to observe any response of M. □

A strongly testable data path has the following advantages.

- *Fast test pattern generation:*
  Test pattern generation time is short since a combinational ATPG tool can be used for each combinational hardware element separately.
- *Fast test plan generation:*
  Test plan generation time is short since test plans are generated at RTL (not at gate level).
- *100% fault efficiency:*
  100% fault efficiency can be achieved for the whole data path, since each hardware element M under consideration is a combinational circuit of small size and strong testability guarantees complete controllability and observability of M in the data path.

Consider a test plan TP of a combinational hardware element M in a strongly testable data path. The test plan TP can propagates any pattern of M along several paths CP from primary inputs to M. Similarly, TP propagates any response of M along several paths OP from M to primary outputs. The test plan TP generally consists of three phases: a *control phase*, a *test phase* and a *observation phase*. Here, let $R(CP)$ be a set of registers which are the nearest registers from M on paths in CP and let $R(OP)$ be a set of registers which are the nearest registers from M on paths in OP.

Control phase: The sequence of control vectors corresponding to the control phase in TP propagates any pattern from primary inputs of the data path to every register in $R(CP)$ if $R(CP) \neq \phi$. Otherwise, the control phase is not necessary.

Test phase: The control vector corresponding to the test phase in TP propagates any pattern from primary inputs and/or every register in $R(CP)$ and any response from data outputs of M to every register in $R(OP)$ and/or primary outputs.

Observation phase: The sequence of control vectors corresponding to the observation phase in TP propagates responses from every register in $R(OP)$ to primary outputs if $R(OP) \neq \phi$. Otherwise, the observation phase is not necessary.

For a controller/data path circuit, test plans of the data path must be applied to control signals. In the test architecture of [15], we generate the test plans from inside of the circuit by appending extra logic called a *test plan generator* ($\mathcal{TPG}$) which generate test plans. For a strongly testable data path, a test plan for a hardware element of the data path can be generally composed of not fixed control vectors such that control vectors of the test plan varies moment by moment. Therefore an $\mathcal{TPG}$ must be designed as a sequential circuit. If control vectors of a test plan do not vary, the $\mathcal{TPG}$ becomes combinational circuit and hence area overhead can be reduced. We introduce such a testability defined as follows.

Definition 2: *Fixed-Control Testability*
A data path is *fixed-control testable* if the following conditions hold.

C1: The data path is strongly testable.

C2: For each combinational hardware element M in the data path, a control sequence of each phase in a test plan of M is composed of only one control vector. □

In addition to advantages of strongly testable data paths, a fixed-control testable data path has the following advantages.

- *Simple test plans:*
  A test plan of a combinational hardware element is composed of at most three control vectors.

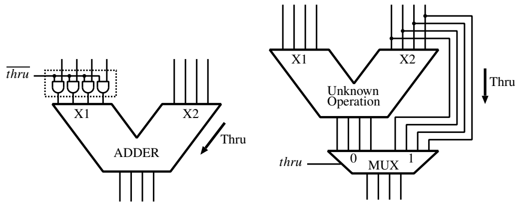
5.2 DFT Method for Data Paths

Overview

In our approach, for a data path, as many test patterns and responses of each hardware element as possible are propagated along existing data path flows in the data path. If test patterns and responses cannot be propagated along existing data path flows, the DFT method appends DFT elements (e.g. *masks*, *multiplexers* and *bypass registers*) to the data path to guarantee that the test patterns and the responses can be propagated along existing data path flows. We first provide a brief explanation of the DFT method.

Consider testing of a combinational hardware element M with two data inputs, x and y, in the data path. To test M, a value specified by a test pattern should be fed into x. We propagate the value along a path p from a primary input to x. If an operational module M' appears on p, the output value of M' will depend on the function and the input value(s) of M'.

In order to guarantee that the output of M' is completely controllable by the input port on p, *thru* function between the input port and the output is added to M'. Most of the popular operational modules (e.g. adder) can realize the thru function by using a *mask* element. The mask element generates a constant



(a) using a mask element (b) using a multiplexer

Figure 4: Examples of realizing thru function.

which is required to realize the thru function. Figure 4(a) illustrates an example of such mask element. If we cannot realize the thru function using the mask element, we realize the thru function using a multiplexer as shown in Figure 4(b). In the rest of this paper, we assume that, for every operational module, thru function can be realized by mask element in order to simplify the discussion.

However, we cannot achieve the fixed-control testability by adding only the thru functions. The thru functions guarantee controllability of a single path. In case of a hardware element which has two data inputs, a test pattern must be applied to both the inputs simultaneously. Presence of re-convergent paths in a data path can prevent such application of a test pattern to a hardware element which has two data inputs. In particular, this can happen if the propagation paths to the two data inputs of a hardware element start from the same primary input and have the same *sequential depth*[2]. Such re-convergent paths will cause a *timing conflict*, i.e. two different values are required on a primary input at the same time. In the concept of strong testability [14], such conflicts are resolved by using hold functions of registers where a register originally has a hold function or is augmented with a hold function. However, since use of hold functions of registers spoils fixed-control testability, we cannot use the hold functions. To resolve such conflicts, in this DFT method, some registers are bypassed by multiplexers and bypass registers are added to some signal lines.

In this DFT method, to control the thru functions, the multiplexers and the bypass registers, additional control inputs are appended as shown in Figure 2.

The goal of the DFT method is to make a given data path fixed-control testable with the minimum hardware overhead. Practical implementation of an algorithm for the DFT method also dictates that the computation time for the DFT insertion and test plan generation algorithms be manageable. We next describe a heuristic algorithm for the DFT method.

Procedure of DFT method

The heuristic algorithm for the DFT proceeds in the following three steps.

(Step 1) Construct a *control forest*: To determine paths to propagate any pattern, we construct a *control forest*. The control forest is a spanning out-forest[3] of the data path where primary inputs are roots, and its paths are used to propagate test patterns.

(Step 2): Construct an *observation forest*: To determine paths to propagate any response, we construct an *observation forest*. The observation forest is a spanning in-forest[4] of the data path where primary outputs are roots, and its paths are used to propagate responses.

(Step 3) Add DFT elements: To make the data path fixed-control testable, we add DFT elements (extra logic for thru function, multiplexer and bypass register) to the data path at strategic locations.

Details of the algorithm are the following.

Step 1: *Construct a control forest*

A path used for propagating test patterns, from a primary input to a data input of a hardware element, is called a *control path* to the data input. To reduce the hardware of DFT elements which will be added at the third step, the control path should be carefully chosen among all the paths from primary inputs to the data input. In our method, to keep the number of added mask elements where a mask element realizes thru function as small as

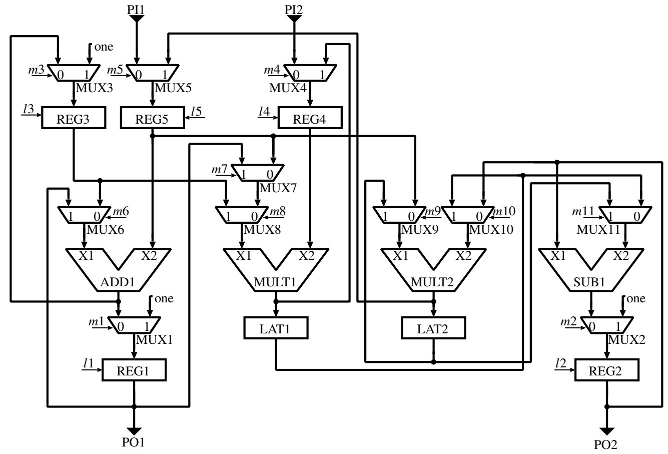


Figure 5: An example of a data path.

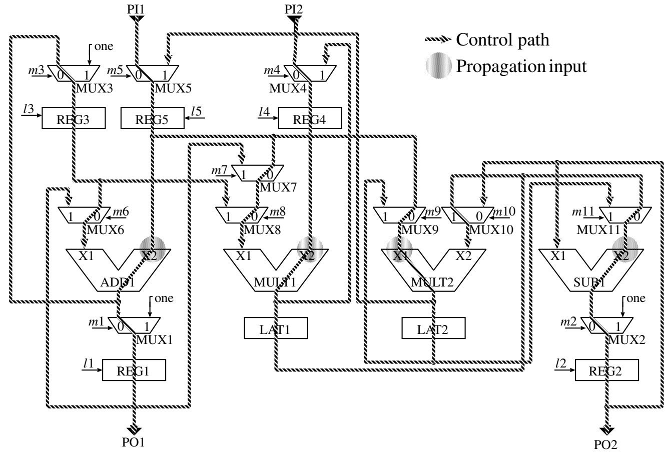


Figure 6: An example of a control forest.

[2]The number of registers on a path is called sequential depth of the path.

[3]A directed forest in which each data input is reachable from some primary input.

[4]A directed forest in which there is path from each data output to some primary output.

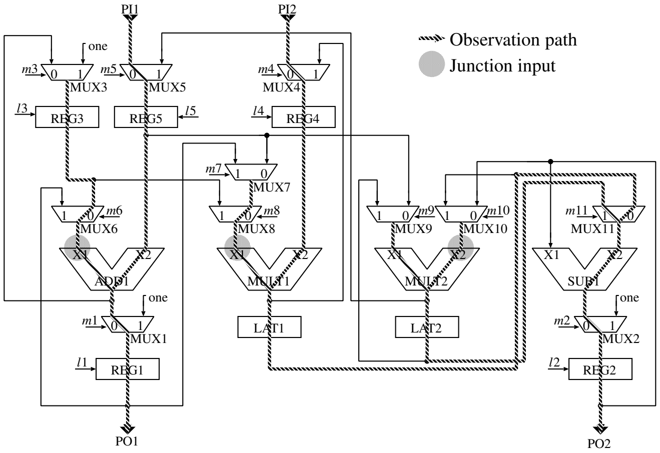


Figure 7: An example of a observation forest.

possible, we choose the control paths for each data input such that collection of all the chosen paths form an out-forest of the data path. This choice guarantees that a mask element is added to *only one* of the data input of each two-input operational module. We call the resulting out-forest a *control forest*. Since the smaller sequential depth intuitively contributes to reduce the test application time, we also pay attention to the the sequential depth of the path while choosing a path. Thus, we construct the control forest as a shortest[5] spanning out-forest of the data path. Here, for each two-input operational module M, we call one of the data input which is not a leaf of the control forest a *propagation input* of M and the other *not-propagation input* of M.

Let us consider the data path of Paulin[12] shown in Figure 5. We show an example of control forest of the data path in Figure 6. Propagation inputs of the example are X2s of the adder ADD1, the multiplier MULT1 and the subtracter SUB1, and X1 of the multiplier MULT2.

Step 2: *Construct an observation forest*

A path used for propagating any response of a hardware element from its data output to a primary output is called an *observation path* of the data output. In the observation path, we utilize control paths as much as possible because control paths have the capabilities of propagating any pattern. However, in most case, there exist no control path from the data output port of a hardware element to a primary output. In such a case, we construct an observation path by connecting the minimum number of control paths. If there exist several observation paths each consisting of a minimum number of control paths, we choose an observation path such that the collection of all chosen observation paths for all data outputs in the data path form an in-forest. We call the in-forest an *observation forest*. Here, for each two-input operational module M, if its not-propagation input is on the observation forest, we call the non-propagation input a *junction input*. If M has a junction input and no thru function between the junction input and its data output, thru function may be added to M to propagate any response from the junction input to the data output. Therefore, we construct the observation forest as a shortest[6] spanning in-forest of the data path concerned with the number of the not-propagation inputs.

We show an example of observation forest of the data path of Paulin in Figure 7. Junction inputs of the example is X1s of ADD1 and MULT1 and X2 of MULT2.

Step 3: *Add DFT elements*

DFT elements are added by the following three steps so that test patterns and responses can be propagated along the control paths and observation paths, respectively.

Step 3.1: *Add thru function to some functional modules on control paths*

For each operational module M, if there exist no thru function between its propagation input and its data output, we augment M with thru function between them by adding mask element on its not-propagation input.

Step 3.2: *Add multiplexers and bypass registers on control paths*

In the step 3.1, it guarantees that any pattern can be propagated from a primary input to a data input via a single control path. However, for each two-input operational module M, two patterns must be propagated from primary input(s) to both two data inputs of M by way of two control paths simultaneously. If these control paths are from the same primary input PI and have the same sequential depth, such re-convergent paths will cause a timing conflict. An example of such re-convergent paths are PI1→ MUX5→ REG5→ ADD1→ MUX3→ REG3→ MUX6 and PI1→ MUX5→ REG5→ ADD1→ MUX1→ REG1→MUX6 of Figure 6. To resolve such conflicts, we add a multiplexer or bypass register to one of these paths in the following two cases. Here, let M' be the fanout element in the re-convergent paths and let p be the path from the data output of M' to propagation input of M on the control path from PI to the propagation input. Let d be the sequential depth of p.

- $d \neq 0$

  If a register on p can be bypassed by adding a multiplexer, the sequential depth of p becomes $d-1$ and thus timing conflict can be resolved. Since such bypassing a register may cause a combinational cycle, the connection of the multiplexer should be carefully determined. Let R be a register which is the nearest to data output of M' on p. We insert the multiplexer to the signal line just after the data output of R. If there exists registers on the control paths between PI and M', the data input, which is not on p, of the inserted multiplexer is connected to the

[5] Any path from a primary input to a data input in the forest is shortest in terms of the sequential depth among all the paths from primary inputs to the port.

[6] A path from a data output to a primary output in the forest is shortest if the number of not-propagation inputs on the path is smallest among all paths from the data output to primary outputs.

data output of the register which is the nearest to M' among these registers. Otherwise, the data input is connected to *PI*. This way of connection makes no combinational cycle. Consider testing of the multiplexer. For justification of test patterns, although control paths of the multiplexer form re-convergent paths, it is guaranteed that sequential depths of these paths are different from each other. For propagation of responses, since a multiplexer is added to path p whose end is a propagation input, a observation path for the multiplexer can be constructed without DFT elements. An example of insertion of a multiplexer *TM* is shown in Figure 8. The multiplexer is inserted to the signal line just after the data output of REG3 and the right data input is connected to the data output of REG5. To test the multiplexer MUX6, we use *TM* so that paths PI1→ MUX5→ REG5→ ADD1→ MUX1→REG1→ MUX6 and PI1→ MUX5→ REG5→ TM→ MUX6 are used as control paths of MUX6.

- $d = 0$

  We insert a bypass register to the signal line just after the data output of M'. Testing of the bypass register is also possible in the way as in step 3.1.

Here, we consider the possibility of reusing the DFT elements. It is generally conceivable that DFT elements which are nearer to primary input are more reusable than those which are nearer to primary outputs. Therefore, we first process two-input operational module which are nearer to primary input and whose control paths start from the same primary input and have the same sequential depth.

Step 3.3: *Add thru function to some functional modules on observation path*

If there exist junction inputs on the observation path of M, timing conflicts may occur while propagating any response from the data output z of M to the primary output using the observation path as follows. Any response can be easily propagated along the control path parts using the thru functions. However, it is more complicated to propagate any response through a junction input j of an operational module M_j on the observation path. If M_j has a thru function between j and its data output l, any response can be propagated through j using the thru function. However, if M_j has no thru function between j and l, any response can not be propagated through j. In this case, we consider use of a constant c which is applied to the propagation input k of M_j by way of the control path of k and realizes thru function between j and l. We call the control path of k the *support path* of k. This may cause a timing conflict between the support path of k and a control path used for propagating either any pattern to M or a similar constant to a preceding operational module on the observation path. The timing conflict is checked and mask elements are added by the following way. Notice that operational modules which have junction inputs are considered in the order of the data path flow. Let M'_j be the nearest operational module to j on the observation path part between z and j such that M'_j has junction input on the path and thru function between the junction input and its data output can be realized by using the support path of its propagation input. Notice that if such module does not exist, we assume that M'_j is M. Let I be a set of primary inputs which are used for propagating test patterns from primary inputs in I to M and responses from M to j. We assume that control timing of every primary input of I are determined by the preceding processing. In this situation, no timing conflict occur on the following three conditions.

C1: The primary input which is the start of the support path of k is different from every primary input of I.

C2: Although C1 does not hold, control timing of the primary input for the support path of k is different from the other required control timing for the primary input.

C3: Although C1 and C2 do not hold, control timing of the primary input for the support path of k can be changed to control timing which is different from the other required control timing for the primary input by using multiplexers or bypass registers added at the above steps on the support path or the other required control timing for the primary input can be changed by using multiplexers or bypass registers added at the above steps on the observation path part from the data output of M'_j to j.

If none of these conditions hold, M_j is augmented with thru function between j and l by adding mask element on k.

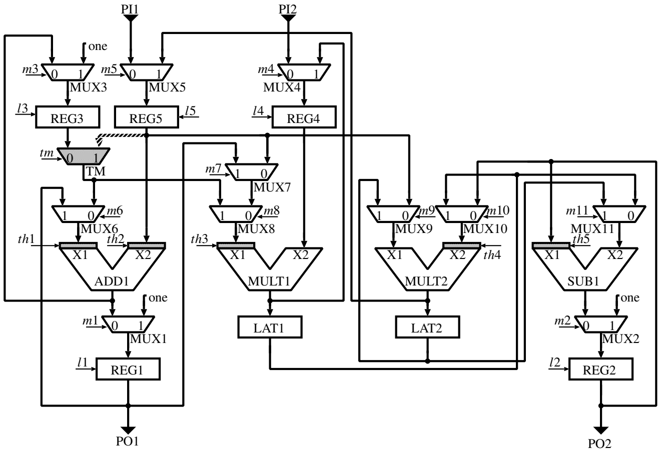


Figure 8: An example of a fixed-control testable data path.

An example of application of the DFT method is shown in Figure 8. In this example, a mask element is added to data inputs X1 and X2 of ADD1, X1 of MULT1, X2 of MULT2 and X1 of SUB1. A multiplexer is added to the signal line just after the register REG3 and the right input of the multiplexer is connected to the data output of the register REG5. Additional control inputs arose to control these DFT elements are *th*1 to *th*5 and *tm*.

5.3 Test Plan Generation for Data Paths

Primary and preliminary tests

To test a hardware element M, test vectors for M are propagated to the input ports along the control paths of the data inputs of M. If M has two data inputs and is included in a cycle in the data path, then the control paths of its data input x may go thru M itself. For example, in Figure 6, the control path of X1 of ADD1 goes thru ADD1 itself. In such a case, a fault of M affect the value propagated thru M and prevent the desired values from reaching x. Thus, we first observe the value propagated through M using the observation path of the data output of M. If the observed value is not the expected one, a fault is detected. We call this test a *preliminary test*. Notice that we execute the preliminary test for each test vector of M, since affect of a fault depends on the vector value. If no fault is detected in the preliminary test, the test vector is applied to the two data inputs of M and the response of M is observed. We call this test a *primary test*. However we have to generate test plans for preliminary tests as well as for primary tests, we consider test plan generation for only primary tests. Test plans for preliminary tests can be generated similarly.

Test plans

In the DFT procedure, for each hardware element M, we determine control paths and an observation path of M. In the steps 3.1 and 3.2 of the DFT, we guarantee controllability of the control paths of M by using DFT elements. Similarly, in the step 3.3 of the DFT, we also guarantee observability of the observation path of M and controllability of support paths corresponding to modules which have junction input on the observation paths with paying attention to timing conflicts between the support paths and the control paths by using DFT elements. Therefore, for testing of M, the collection of the control paths and the support paths can be activated by applying a control vector C to control inputs (including control inputs appended by the DFT) of the data path. Similarly, the collection of the observation path and the support paths can be activated by applying a control vector O to the control inputs.

First, we consider the control sequences of the control phase and the observation phase of the test plan of M. The control sequence of the control (resp. observation) phase in the test plan of M is composed of only C (resp. O). Consider the lengths of the control sequences of the control phase and the observation phase. Here, let d_c and d_o be the maximum sequential depth among the control paths and the sequential depth of the observation path, respectively. Let d_s be the maximum sequential depth among paths in the collection of the observation path and the support paths. The length of the control sequence of the control phase and that of the observation phase are $\max(d_c, d_s - d_o)$ and d_o, respectively.

Table 2: Coordinate intersection of C and O.

		C		
	$\cap$	0	1	$*$
	0	0	1	0
O	1	0	1	1
	$*$	0	1	$*$

$*$ = don't care

Table 3: An example of a test plan.

t	PI1	PI2	$m1$	$m2$-4	$m5$	$m6$	$m7$-11	$l1$	$l2$-4	$l5$	$th1$	$th2$	$th3$-5	tm	PO1	PO2	Phase
1	C	$*$	0	$*$	0	$*$	$*$	1	$*$	1	1	$*$	$*$	1	-	-	Cont.
2	C	$*$													-	-	
3	$*$	$*$	0	$*$	0	T	$*$	1	$*$	1	1	1	$*$	1	-	-	Test
4	$*$	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	$*$	O	-	Obs.

C : apply test pattern to PI
O : observe responce at PO
T : apply test pattern to control input
$*$: don't care
- : need not observe

Next, we consider the control vector T of test phase in the test plan. The control vector of test phase in the test plan is basically obtained by the intersection of C and O formed from the respective coordinate intersection in Table 2. If M has a control input, since a test pattern corresponding to the control input have to be applied to the control input, it is needed to modify the obtained control vector to a control vector whose coordinate corresponding to the control input is the test pattern. The way to apply the test pattern to the control input is presented in the next section.

The test plan is composed of at most three vectors C, T and O. The length of the test plan is $\max(d_c, d_s - d_o) + d_o + 1$.

For example, a test plan of MUX6 of Figure 8 is shown in Table 3.

6. A Method of DFT for RTL Controller/Data Path Circuits

In this section, we propose a DFT method for a whole circuit which consists of both a controller and a data path. In the DFT method, we adopt our DFT method [4] to the controller part and our DFT method proposed in Section 5 to the data path part. In these DFT methods for the controller part and the data path part, we assumed that internal signals, control signals and the status signals, are directly controllable and observable from outside of the circuit. Here, we remove this assumption. The goal of the DFT method proposed in this section is to provide complete controllability and observability of these internal signals (control and status signals) required for applying test patterns and observing responses.

6.1 Testing of Controllers

For testing of a controller part in a circuit, let us consider to provide complete controllability and observability of status signals and control signals, respectively. It is achieved by appending additional test I/O pins (primary inputs and outputs) and connecting them directly to the status inputs and the control outputs of a controller of the circuit. However, the pin overhead becomes large and it may be unacceptable for practical designs. Instead, the pin overhead is avoided as follows: the status inputs of the controller are directly controlled from primary inputs of the data path by adding a multiplexer "MUX1", and the control outputs of the controller are directly observed from primary outputs of the data path by adding a multiplexer "MUX3" as shown in Figure 2. This approach is acceptable because it is generally conceivable that the status inputs (resp. the control outputs) of the controller have smaller bit-width than the primary inputs (resp. the primary outputs) of the data path, and we need not use the data path part at testing of the controller. In testing of the controller, these multiplexers "MUX1" and "MUX3" are configured as shown in Table 1.

6.2 Testing of Data Paths

For a data path in a circuit, let us consider a hardware element M which has data inputs, control inputs, data outputs and status outputs. Each test pattern of M consists of patterns for the data inputs of M and that for the control inputs. We call the former a *data input pattern* and the latter a *control input pattern*. Notice that, for testing such a hardware element, we must apply test patterns to both the data inputs and the control inputs. A test plan of M propagates the data input pattern to the data inputs of M from primary inputs. The test plan also applies control input pattern to the control inputs. The test plan also propagates the responses appeared on the data outputs of M to the primary outputs. The response appeared on the status outputs of M is observable from status outputs of the data path.

Control vectors of the test plan of M and the control input patterns for M must be applied to the control inputs (including control inputs appended by the DFT) of the data path. Similarly, the responses of M must be observed from the status outputs of the data path. If we use additional test I/O pins to make these control inputs and status outputs directly controllable and observable, respectively, from the outside of the circuit, the pin overhead becomes large. The problem of the pin overhead can be avoided by the following way. Here, we first consider the observability of status outputs of the data path. In general, since the bit-width of primary outputs of a controller of the circuit is smaller than that of the status outputs, we cannot use the primary outputs for observing the status outputs. However, the bit-width of the primary outputs of the data path is larger than the status outputs. While testing the data path, it is sufficient to observe the response of a test vector either from primary output or from status outputs. The status outputs and the primary outputs need not to be observed simultaneously. Therefore, we can observe the status outputs using the primary outputs of the data path. In the test architecture of Figure 2, this is achieved by a multiplexer MUX3.

We next consider the controllability of control inputs of the data path. In general, the number of primary inputs of the controller is smaller than that of control inputs of the data path. Therefore, we cannot use the primary inputs for applying test plans and control input patterns of hardware elements to the control inputs of the data path. Moreover, in testing of data path, since the primary inputs of the data path are used to apply data input patterns of hardware elements, we cannot use them to apply the test plans and the control input patterns to the control inputs of the data path simultaneously. Therefore, we append an extra circuit called a *test controller* to generate control input patterns as shown in Figure 2.

Table 4: Configuration of test controllers.

Mode		Function
t_3	t_4	
0	0	Setting TPR and TMR
0	1	Generating a vector of justification phase
1	0	Generating a vector of test phase
1	1	Generating a vector of observation phase

Test controller

Test plans are generated for all the combinational hardware elements in a data path of a circuit. In our test architecture shown in Figure 2, all the test plans of the data path are generated by a *test controller*. The test controller consists of a *test plan generator*($\mathcal{TPG}$), a *test pattern register*(TPR) and a *target module register*(TMR). Let us consider the testing of a combinational hardware element M, which has data inputs and control inputs, in the data path. The TMR is used to store the index of a test plan either for a preliminary test or a primary test of M. The bit width of the TMR is $\log_2 m$ where m is the number of test plans for all the combinational hardware elements in the data path. The $\mathcal{TPG}$ generates the test plan of M if the index of the test plan is stored in the TMR. The $\mathcal{TPG}$ is designed as a combinational circuit and controlled by $t3$ and $t4$ as shown in Table 4. The $\mathcal{TPG}$ generates a control vector for a phase of the test plan. The control input pattern of a test pattern for M is pre-stored in the TPR before entering the control phase and is applied to the control inputs by way of $\mathcal{TPG}$ in the test phase. The load enable signal for TPR and TMR is controlled from the $t3$ and $t4$ by way of TMR as shown in Table 4. That is, if $(t3,t4) = (0,0)$ is applied, test patterns for control inputs and an index of a test plan is loaded into TPR and TMR from some primary inputs of the data path, otherwise, they hold their values. The mode switching signal t_1 is used to disable DFT elements of the data path in the normal operation mode of the circuit.

We evaluate hardware overhead of $\mathcal{TPG}$s of controller/data path circuits in the next section. Although the hardware overhead depends on data path designs, for controller/data path circuits dominated data flow, it is conceivable that hardware overhead of the $\mathcal{TPG}$ is low.

We also consider testing of a $\mathcal{TPG}$. Since the $\mathcal{TPG}$ is not used at the normal operation, we test the $\mathcal{TPG}$ only to confirm that the test plans are generated correctly. It is performed by observing primary outputs of a data path (see Figure 2).

7. Experimental Results

In this section, we evaluate effectiveness of our proposed method by experiments. Circuit characteristics of RTL benchmark circuits used in the experiments is shown in Table 5. The

Table 5: Circuit characteristics.

Circuit	Area(*gate*)	Controller						Data path					
		#PI	#PO	#State	#Status	#Control	Area(*gate*)	#PI	#PO	|bit|	#Reg.	#Mod.	Area(*gate*)
GCD	1524.50	1	1	4	3	7	169.40	32	16	16	3	1	1350.90
JWF	6875.40	1	0	8	0	38	199.50	80	80	16	14	3	6671.70
LWF	1986.20	1	0	4	0	8	57.70	32	32	16	5	3	1924.30
PAULIN	24965.60	1	0	6	0	16	123.50	64	64	32	7	4	24833.70
RISC	62287.60	1	2	11	54	62	3986.90	32	96	32	40	4	58157.90

circuits GCD, JWF, LWF and PAULIN are popularly used examples and the circuit RISC is a practical and large design. In our experiments, we used a logic synthesis tool AutoLogicII (Mentor Graphics) with its sample libraries to synthesize these benchmark circuits. In this table, column "Area" denotes the total areas after synthesis. Here, areas are estimated using gate equivalent of the library cell area. Columns "Controller" and "Data path" denote the characteristics of controller parts and data path parts, respectively: columns "#PI", "#PO" and "Area" denote the numbers of primary inputs and primary outputs and circuit area of respective parts. Columns "#State", "#Status" and "#Control" in "Controller" denote the numbers of states, of status inputs and of control outputs. Columns "|bit|", "#Reg." and "#Mod." in "Data path" denote the bit width of data paths and the numbers of registers and of operational modules in data paths. In the row "RISC", the number of the status signals is larger than that of the primary inputs of the data path. In our DFT, twenty-two primary output pins are changed to primary input/output pins by appending tri-state buffers. However, the hardware overhead of this modification is negligible.

Test generation results are shown in Table 6. The sequential/combinational ATPG tool TestGen (Synopsis) is used in this experiments on Ultra60 model 2360 (Sun Microsystems). Columns "Test generation time", "Test application time" and "Fault efficiency" denote test generation time in second, test application time in clock cycles and fault efficiency. In each of these columns, columns "Original", "Full-scan", "Prev. work" and "This work" denote the results of the original circuits (without DFT), of the circuits modified by the full-scan design, of the circuits modified by the method of our previous work[15] and of the circuits modified by our proposed method in this work. Test generation time of the proposed method is almost the same as the method of [15]. Test generation time of the proposed method is shorter than that of the full-scan. Especially, for the circuit RISC, the proposed method can reduce to 1/700 of the full-scan design and can enhance fault efficiency compared with the full-scan design. For the circuit, fault efficiency is 99.99% because the combinational ATPG tool can not generate a test pattern for a fault in a multiplier of the circuit in the method of [15] and the proposed method. Test application time of the proposed method is shorter than the method of [15]. Test application time of the proposed method is drastically reduced compared with that of the full-scan design.

The area and pin overheads of the full-scan design, the method of [15] and the proposed method are shown in Table 7. Columns "C", "DP", "$\mathcal{TPG}$", "TMR,TPR" and "MUX" in columns "Prev. work" and "This work" of column "Area overhead" denote the area overhead of controllers, data paths, $\mathcal{TPG}$s, registers TMR and TPR and multiplexers on control signals, status signals and primary outputs of data paths. In the proposed method, area overhead of $\mathcal{TPG}$ is less compared to the method of [15]. Furthermore, for data paths, however a fixed-control testable data path can achieve simpler test plans compared to strong testable one, the difference between the area overhead of the proposed method and that of the method of [15] is not large. Especially, for RISC, the proposed method can reduce the area overhead compared with the method of [15]. In these results, area overhead of the proposed method is less than that of the method of [15] for all circuits except JWF. The area overhead of the proposed method is larger than that of the full-scan design but the difference between the area overhead of the proposed method and that of the full-scan design is not large. The pin overhead of the proposed method is the same as the method of [15] and larger than that of the full-scan design. For the circuit RISC, the pin overhead of the method of [15] and the proposed method are larger than the other because two select signals for concatenation of two multiplexers are needed on the primary outputs of the data path to observe the control signals and the status signals from the primary outputs. In return for these disadvantages, the proposed method allows at-speed testing.

In the proposed method, masks, multiplexers and bypass registers are appended to a controller/data path circuit. First, let us consider performance degradation of the circuit caused by multiplexers appended to the controller, the signals between the controller and the data path, and the primary outputs of the data path. Multiplexers are appended in front of the state register of the controller in the circuit, on control signals between the controller and the data path in the circuit, and in front of the primary outputs of the data path. The multiplexer in front of the state register is the same as the full-scan design. The multiplexer on the control signals does not affect performance of the circuit because the control signals are not generally included in the critical path of the circuit [11]. The multiplexer in front of the primary outputs of the data path does not also affect performance of the circuit because, in general, there exist registers in front of the primary outputs of the data path and delay of multiplexers are less than operational modules.

For the data path, the performance might be degraded by appended masks, test multiplexers and multiplexers of bypass registers. In the full-scan design, multiplexers are added to all registers to make each register a scannable register. On the other hand, in the method proposed here, masks and multiplexers are added to some (not all) operational modules and combinational paths between two registers, respectively. Further-

Table 6: Test generation results.

Circuit	Test generation time(*sec.*)				Test application time(*cyc.*)				Fault efficiency(%)			
	Original	Full-scan	Prev. work	This work	Original	Full-scan	Prev. work	This work	Original	Full-scan	Prev. work	This work
GCD	18055.53	171.51	0.69	0.69	9	6629	504	504	4.92	100.00	100.00	100.00
JWF	2348.24	2.88	0.37	0.27	488	20519	1497	1621	98.14	100.00	100.00	100.00
LWF	171.68	0.47	0.27	0.27	322	4066	517	443	99.64	100.00	100.00	100.00
PAULIN	20362.55	4.68	2.11	2.20	283	16187	2193	2172	97.01	100.00	100.00	100.00
RISC	288102.05	51740.92	71.50	72.29	4298	1006154	9674	7768	62.31	99.97	99.99	99.99

Table 7: Hardware overheads.

Circuit	Area overhead(%)														Pin overhead(#)		
	Full-scan	Prev. work						This work							Full-scan	Prev. work	This work
			C	DP	$\mathcal{TPG}$	TMR,TPR	MUX		C	DP	$\mathcal{TPG}$	TMR,TPR	MUX				
GCD	26.6	39.7	1.1	2.6	19.1	4.1	12.8	32.8	1.1	2.6	12.2	4.1	12.8		3	5	5
JWF	26.7	37.1	0.4	5.2	20.5	1.4	9.7	41.9	0.4	9.3	20.7	1.4	10.0		3	5	5
LWF	33.4	48.6	0.8	18.1	17.2	3.9	8.6	44.6	0.8	18.1	13.2	3.9	8.6		3	5	5
PAULIN	7.4	8.1	0.2	1.2	4.9	0.4	1.4	7.1	0.2	2.5	2.6	0.4	1.4		3	5	5
RISC	16.7	27.3	0.1	10.9	12.3	0.2	3.6	21.0	0.1	9.6	7.4	0.2	3.7		3	6	6

more, delays of masks is less than that of multiplexers for scan and multiplexers added in the proposed method is the same as multiplexers for scan. Therefore, performance degradation of the proposed method is not so much compared to that of full-scan design.

8. Conclusion

This paper presented a non-scan DFT method for controller/data path circuits designed at RTL and that for data path based on fixed-control testability. The proposed method can achieve 100% fault efficiency and allows at-speed testing. We reduced the hardware overhead in the presented method compared to the method of our previous work [15]. The hardware overhead of the method is slightly more than that of the full-scan design. Since the hierarchical test generation can be applied to the data path part of the circuits, test generation time of the proposed method is shorter than that of the full-scan design. Furthermore, since the proposed method uses no traditional scan path, test application time is very low.

Acknowledgments

This work was supported in part by Semiconductor Technology Academic Research Center (STARC) under the Research Project. Authors would like to thank Toshimitsu Masuzawa and Michiko Inoue of Nara Institute of Science and Technology and Tomoo Inoue of Hiroshima City University for their valuable discussions.

References

[1] H. Fujiwara: *Logic Testing and Design for Testability*, The MIT Press, 1985.

[2] P. C. Maxwell, R. C. Aitken, V. Johansen and I. Chiang: "The effect of different test sets on quality level prediction: when is 80% better than 90%?," in *Proc. of International Test Conference*, pp. 358–364, 1991.

[3] S. T. Chakradhar, S. Kanjilal and V. D. Agrawal: "Finite state machine synthesis with fault tolerant test function," *Journal of Electronic Testing: Theory and Applications*, Vol. 4, pp. 57–69, February 1993.

[4] S. Ohtake, T. Masuzawa and H. Fujiwara: "A non-scan DFT method for controllers to achieve complete fault efficiency," in *Proc. of the 7th Asian test symposium*, pp. 204–211, 1998.

[5] D. K. Das, S. Ohtake and H. Fujiwara: "New DFT techniques of non-scan sequential circuits with complete fault efficiency," in *Proc. of the 8th Asian test symposium*, pp. 263–268, 1999.

[6] R. B. Norwood and E. J. McCluskey: "Orthogonal scan: Low overhead scan for data paths," in *Proc. of Int. Test Conf.*, pp. 659–668, 1996.

[7] R. B. Norwood and E. J. McCluskey: "High-level synthesis for orthogonal scan," in *Proc. of 15th VLSI Test Symp.*, pp. 370–375, 1997.

[8] S. Battacharya and S. Day: "H-SCAN: A high-level alternative to full-scan testing with reduced area and test application time," in *Proc. of the IEEE VLSI Symp.*, pp. 74–80, 1996.

[9] B. T. Murray and J. P. Hayes: "Hierarchical test generation using pre computed tests for modules," *IEEE Trans. on CAD*, Vol. 9, No. 6, pp. 594–603, June 1990.

[10] S. Bhatia and N. K. Jha: "Genesis: A behavioral synthesis system for hierarchical testability," in *Proc. of European design and test conference*, pp. 272–276, 1994.

[11] I. Ghosh, A. Raghunath and N. K. Jha: "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," in *Proc. of IEEE Int. Conf. on Computer Design*, pp. 173–179, 1995.

[12] I. Ghosh, A. Raghunath and N. K. Jha: "A design for testability technique for RTL circuits using control/data flow extraction," in *Proc. of IEEE/ACM Int. Conf. on CAD*, pp. 329–336, 1996.

[13] I. Ghosh, A. Raghunath and N. K. Jha: "Hierarchical test generation and design for testability methods for ASPP's and ASIP's," *IEEE Trans. on CAD*, Vol. 18, No. 3, pp. 357–370, March 1999.

[14] H. Wada, T. Masuzawa, K. K. Saluja and H. Fujiwara: "Design for strong testability of RTL data paths to provide complete fault efficiency," in *Proc. of International Conf. on VLSI Design*, pp. 300–305, 2000.

[15] S. Ohtake, H. Wada, T. Masuzawa and H. Fujiwara: "A non-scan DFT method at register-transfer level to achieve complete fault efficiency," in *Proc. of Asian South Pacific Design Automation Conference (ASP-DAC)*, pp. 599–604, 2000.