J Electron Test (2011) 27:583–598
DOI 10.1007/s10836-011-5241-8

A New Design-for-Testability Method Based on Thru-Testability

Chia Yee Ooi · Hideo Fujiwara

Received: 21 August 2009 / Accepted: 4 August 2011 / Published online: 1 September 2011

Abstract Partial scan and non-scan techniques allow test generation of high fault coverage for sequential circuits with less area overhead and less performance degradation than full scan technique. In most of these techniques, extra logic (e.g. a multiplexer introduced by partial scan) is added to permit a data transfer from a flip-flop (or input) to another flip-flop (or output). Such additional logic function is called thru function in this paper, which plays an essential role in enhancing the testability of a circuit. In this paper, we introduce a design-for-testability (DFT) technique which modifies a given sequential circuit to a thru-testable sequential circuit with acyclic test generation complexity by adding new thru functions based on the information of thru functions that may exist in the original design and the dependency among these thru functions. Thus, thru functions of a given sequential circuit should first be extracted from its high-level description. If there is a thru function that transfers data from a flip-flop to another flip-flop, the latter is exempted from being considered in DFT insertion. This reduces the additional logic to be added. Using ITC'99 benchmark circuits, we show that the proposed DFT method takes up less area overhead compared to the previous scan methods in testing the difficult-to-test circuits like b07, b08 and b15. Besides, the test application time is shorter than that of previous scan methods while the test data volume is less too in difficult-to-test circuits.

Responsible Editor: M. A. Breuer

C. Y. Ooi (✉)
Universiti Teknologi Malaysia,
81310 UTM,
Skudai, Johor, Malaysia
e-mail: ooichiayee@fke.utm.my

H. Fujiwara
Osaka Gakuin University,
2-36-1 Kishibe-minami, Suita, Osaka 564-8511, Japan

Keywords Thru-testability · Design-for-testability · Sequential circuit · Thru function · Path dependency

1 Introduction

Owing to the high complexity of test generation for sequential circuits, design-for-testability (DFT) methods are essential in reducing test generation costs. For instance, the popular full scan technique is able to reduce the test generation of a sequential circuit into the combinational test generation. However, full scan technique requires all the flip-flops of a circuit to be augmented into scan flip-flops, which results in large area overhead. In addition, full scan technique also has drawbacks of high test application time and high test data volume. To alleviate the problems, several partial scan techniques and alternative DFT methods have been introduced. The following text summarizes the previous techniques.

1.1 Partial Scan at Gate-Level

The partial scan technique, which breaks the minimum feedback loops [14], has the remaining kernel an acyclic sequential circuit so it succeeds in reducing the number of scan flip-flops. The scan flip-flop selection is based on the concept of minimum feedback vertex set (MFVS) where only minimum number of flip-flops is selected to be scanned. Another partial scan technique whose flip-flop selection is based on empirical testability was introduced in [12] and the experiment showed that the resulting area overhead was comparable to [14]. Cost-free scan [15] identifies existing scan path composed by the existing logic and thus further reduces the area overhead compared to [14] and [12]. The scan technique also identifies the enabling

vectors for the primary inputs that enable the free scan path. The computation to identify the free scan path and the enabling vectors becomes expensive when the circuit is large because the identification algorithm is based on reduced ordered binary decision diagram (ROBDD).

Autoscan [18] is similar to conventional full scan but it does not have external input for scan-in and external output for scan-out. Thus, although the pin overhead is less, the area overhead of the circuit augmented by autoscan is still similar to that augmented by full scan technique. On the other hand, the experimental result with high fault coverage has shown that autoscan can improve the testability of the benchmark circuits. It has been shown to be effective when there is a single scan source but it permits multiple scan chains sourcing from the single scan source. However, the comparison between autoscan with multiple scan chains starting from a single source and full scan with multiple scan chains is not clear.

The above mentioned scan techniques limit the implementation of DFT to the end of design phase though they successfully reduce area overhead, delay overhead and pin overhead (in autoscan). When the use of high-level description to design large VLSI circuits has become popular, it has led to the new paradigm testing, that is DFT insertion at high-level. Improving testability during the early stages of the design flow is getting more concern because it can significantly improve the fault coverage, reduce the hardware and delay overhead and shorten the test application time. Among the previous works which are the main motivation to our work include research detailed in [2,10,11]. [2,10,11] introduced a systematic synthesis-for-testability procedure to embed test functions into a finite state machine, which is a high level modeling of a digital system, such that the synthesized gate-level implementation has improved testability. This works inspired us that testability analysis at high-level is much faster than that at gate-level.

1.2 Knowledge-Based DFT Method for Datapath

Knowledge-based design-for-testability methods select the flip-flops to be scanned based on the knowledge of the circuit description at high-level. The first knowledge-based partial scan was introduced in 1985 by Abadir and Breuer [1]. In this DFT method, a given register-transfer level (RTL) description is modeled by a graph where each vertex represents an RTL structure like multiplexer, register or combinational kernel while each arc represents an interconnect between two RTL structures. An RTL structure is said to have an identity mode (I-mode) if the structure can transfer the data from its input port to its output port. A time tag t and an activation condition tag C are associated to each I-mode where t is the time needed for data-transfer while C is the input value on the other input port required for data-transfer. If there is no I-path (a series of RTL structure with I-mode), then a scan register is used to compose the I-path. Still, scan flip-flops are used to replace the normal flip-flops accordingly and thus the test application time is still large since the test vectors have to be shifted through the scan chain.

F-path was defined in [7] as a combinational logic that permits one-to-one transformation from the input to the output of the combinational logic. The definition is more general compared to I-path because the output data is not necessary equal to the input. It could be a logical or arithmetic function of the input. H-scan [3,4] is one of the first non-scan DFT methods that have been introduced. It utilizes the existing paths between registers, which consist of a series of multiplexers, to build parallel scan paths. Some extra gates are added to the logic of the existing path so that signals transfer between the registers is enabled by a new input independent on the signals from the controller. Ordering the scan flip-flops orthogonally to the registers in the datapath, another non-scan technique called orthogonal scan [16] allows the use of datapath components like adder or multiplier with slight modification to implement the scan path besides allowing the parallel scan paths. Both of these two non-scan techniques did not consider the test generation complexity. Fujiwara et. al [8] introduced a non-scan DFT method based on cycle-unrollability concept which bound the test generation time by a linear function of the number of flip-flops in the RTL circuit. In [8], a test controller is added into the circuit to provide the control signal to activate the justification and propagation paths that send test patterns (resp. test response) to (resp. from) the module-under-test. In other words, the test controller is to isolate the datapath from the controller unit of the RTL circuit.

All the previous works [3,8,16] highlighted the reduction in area overhead. Besides, since they are non-scan DFT methods which allow at-speed testing, their test application time is shorter than the partial scan techniques. However, these knowledge-based DFT methods treat controller and datapath separately. In fact, the methods are well applied on the datapath part. DFT methods at RTL insert additional hardware like test controller or multiplexer to control the data transfer along the justification and observation paths. Instead of additional test controller, state registers in the controller could be manipulated to enable/disable the data transfer along the paths. In this paper, we introduce a new DFT method that extracts the following information from the high-level description of a given RTL circuit:

a. thru function—a logic that allows data transfer from a register to another register;
b. state register (resp. primary input) and the state values (resp. input values) that activate the thru functions.

Besides the knowledge of the circuit testability extracted at high level, the structure of the remaining circuit excluding scan flip-flops is also another factor in determining how a scan technique is to be applied. [13] identified a hierarchy of circuit structures with varying degrees of test generation complexity. There are five subclasses defined, including combinational circuits, strongly balanced sequential circuits, balanced sequential circuits [9], internally balanced sequential circuits and acyclic sequential circuits. The class of acyclic sequential circuits is the superset of internally balanced sequential circuits and so on. After scan technique is applied to a given general sequential circuits, the remaining circuits excluding scan flip-flops (called circuit kernel) can belong to one of the subclasses. The bigger the class to which the circuit kernel belongs to, the fewer flip-flops need to be scanned. Therefore, our work attempts to identify a subclass of the acyclic sequential circuits which can benefit from lower number of flip-flops to be scanned.

The paper is outlined as follows. In Section 2, we briefly review *Assignment Decision Diagram (ADD)*, which is used to model a given high-level description. Then, a circuit property called *thru function* will be described. We will show how a *thru function* is extracted from an ADD. We also define *R-graph* as a representation of a sequential circuit and introduce a new concept of testability called *thru-testability*. We depict a method of identifying the thru-testability of a sequential circuit when it is represented by R-graph. In Section 3, we discuss the implementation of our DFT method. The details include an algorithm that augments a given sequential circuit into a *thru-testable* circuit by adding new thru functions. Experimental setup and result are presented in Section 4 and Section 5 concludes our work.

2 Preliminaries

2.1 Assignment Decision Diagram (ADD)

ADD is a modeling that has been proposed previously for high-level synthesis [5] to model a behavioral description. The unique feature of ADD is its capability to represent conditions and computations in a consistent data flow fashion. Therefore, operations in ADD are ordered by data dependencies, which is free of control dependency that is introduced in the description. With this capability, ADD can represent the most parallel representation of a given description.

The ADD representation consists of read nodes, operation nodes, write nodes and assignment decision nodes (ADN). Read node represents a storage element or an input; write node represents a storage element or an output; operation node can be a logical or arithmetic unit; ADN is an assignment decision part which selects a value from a set of values that are provided at its value inputs. If one of the conditions to the ADN evaluates to true, then the corresponding input value is selected. The assignment target is represented by a write node. The write node is associated with the selected value from the corresponding ADN. A value will be assigned to the write node only if one of the condition inputs to the ADN evaluates to true and only one value can be assigned to a target at a time. For instance, the ADD for ex1 circuit of the VHDL code in Fig. 1 is as illustrated in Fig. 2. The control flow of the design can be captured by tracing its *ps* values while the data flow of the design is represented by the logical and arithmetic operation nodes.

2.2 Thru Function

ADD can represent a non-separable RTL circuit. Thus, the data transformation in the original design that is useful in justification and fault propagation and the variables that activate the data transformation can be more clearly illustrated using ADD. In the following text, we denote the logic that allows the data transformation under certain assignment of some variables as *thru function*.

Definition 1. A thru function type $P, f{:}X \times Y' \rightarrow Z$ is a partial function from X and Y to Z with $Y' \subseteq Y$ which is injective where every (x,y) is mapped to a unique $z \in Z$, and $y \in Y'$.

Definition 2. A thru function type $J, f{:}X \times Y' \rightarrow Z$ is a partial function from X and Y to Z with $Y' \subseteq Y$ which is surjective where for every $z \in Z$, there exists a pair of (x,y) such that $z = f(x,y)$ and $y \in Y'$.

x is called input data variable; z is called output data variable; y is called an activating variable or activator. Activator is the set of inputs that enables the transformation from X to Z.

Definition 3 An ultimate thru function is a thru function that can function as both thru function type P and thru function type J.

Based on the ADD description of a design, a condition can be defined for the value assignment that, when assigned to the activating variable, establishes onto or one-to-one transformation from the input data variable to the output data variable. This condition is called activation condition. It can normally be described in Boolean expression.

Example 1: Referring to ADN *M1* in Fig. 3, data from read node *A* will transfer to write node *rega* when the present state is *s0*. Therefore, the ultimate thru function can be written as *f(A,ps)=rega with activation condition* (*ps=s0*).

Example 2: In Fig. 3, there exists a direct connection from read node *rego* to write node *O*. This implies the direct data transfer from *rego* to *O* which is represented by *f(rego,1)=O*

Fig. 1 VHDL listing for *ex1*

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ex1 is
 port(
  clk : in std_logic;
  rst : in std_logic;
  A : in std_logic_vector(7 downto 0);
  B : in std_logic_vector(7 downto 0);
  C : in std_logic_vector(7 downto 0);
  D : in std_logic_vector(7 downto 0);
  E : in std_logic;
  O : out std_logic_vector(7 downto 0));
end ex1;

architecture RTL of ex1 is
 type state_type is (s0,s1,s2,s3);
 signal ps : state_type;
 signal rega : std_logic_vector(7 downto 0);
 signal regb : std_logic_vector(7 downto 0);
 signal regc : std_logic_vector(7 downto 0);
 signal regd : std_logic_vector(7 downto 0);
 signal rege : std_logic;
 signal regf : std_logic_vector(7 downto 0);
 signal regg : std_logic_vector(7 downto 0);
 signal rego : std_logic_vector(7 downto 0);
begin --RTL

 process(clk, rst)
  begin
   if rst='1' then
    ps <= s0;
   elsif clk'event and clk = '1' then
    case ps is
     when s0 =>
      rega <= A;
      regb <= B;
      regc <= C;
      regd <= D;
      rege <= E;
      regg <= "00000000";
      ps <= s1;
     when s1 =>
      if regc < regd then
       regf <= rega + regb;
       ps <= s2;
      else
       regf <= rega - regb;
       ps <= s3;
      end if;
     when s2 =>
      regg <= regf + regg + regf;
      if rege = '0' then
       ps <= s1;
      else
       ps <= s3;
      end if;
     when s3 =>
      rego <= regg - 3;
      ps <= s0;
     when others => null;
    end case;
   end if;
  end process;

  O <= rego;
 end RTL;
```

without any activation condition needed. This thru function is also called *direct thru function.*

Example 3: ADN *M2* connects two arithmetic operation nodes, i.e. addition and subtraction. When all the paths starting from read nodes *rega* and *regb* to write node *regf* are examined carefully, two thru functions are identified. They are

i. *f(rega,(ps,regb,regc,regd)) = regf with activation condition (ps = s1)∧(regb = any value)*;
ii. *f(regb,(ps,rega,regc,regd)) = regf with activation condition (ps = s1)∧(rega= any value)*.

In this example, the activating variable is decomposed into a concatenation of few variables, such as *(ps,regb,regc,regd)* in thru function *i*.

2.3 R-Graph

Although thru function extraction is done at high-level as a pre-processing in our DFT approach, the circuit augmentation is still done after design synthesis. This is because all the feedback loops formed by flip-flops can only be identified completely at gate-level. Thus, the flip-flop selection method we use in our method is based on MFVS concept. To analyze the testability of the circuit, we introduce a circuit representation called *R-graph*, which contains the information of connectivity between combinational components and flip-flops, and thru functions. R-graph is defined such that it can be used to represent a gate-level circuit as well as a RTL circuit.

Definition 4: A circuit representation called *R-graph* is a directed graph $G = \{V,A,t,a,h\}$ that has the following properties.

a. $v \in V$ is a primary input, a primary output or a register;
b. $(u,v) \in A$ denotes an arc if there exists a combinational path from the primary input or register corresponding to u to the primary output or register corresponding to v;
c. $t: A \rightarrow T \cup \{1, \varnothing\}$ (T is a set of Boolean expressions) defines activation condition that establish the thru functions along arc $(u,v) \in A$ where vertex u represents an input data variable while v represents the output data variable. $t(u,v) = \varnothing$ if there is no thru function for $(u,v) \in A$. If $t(u,v)=1$, the signal values are transferred from u to v through a wire logic (not a gate logic) directly via a direct thru function.

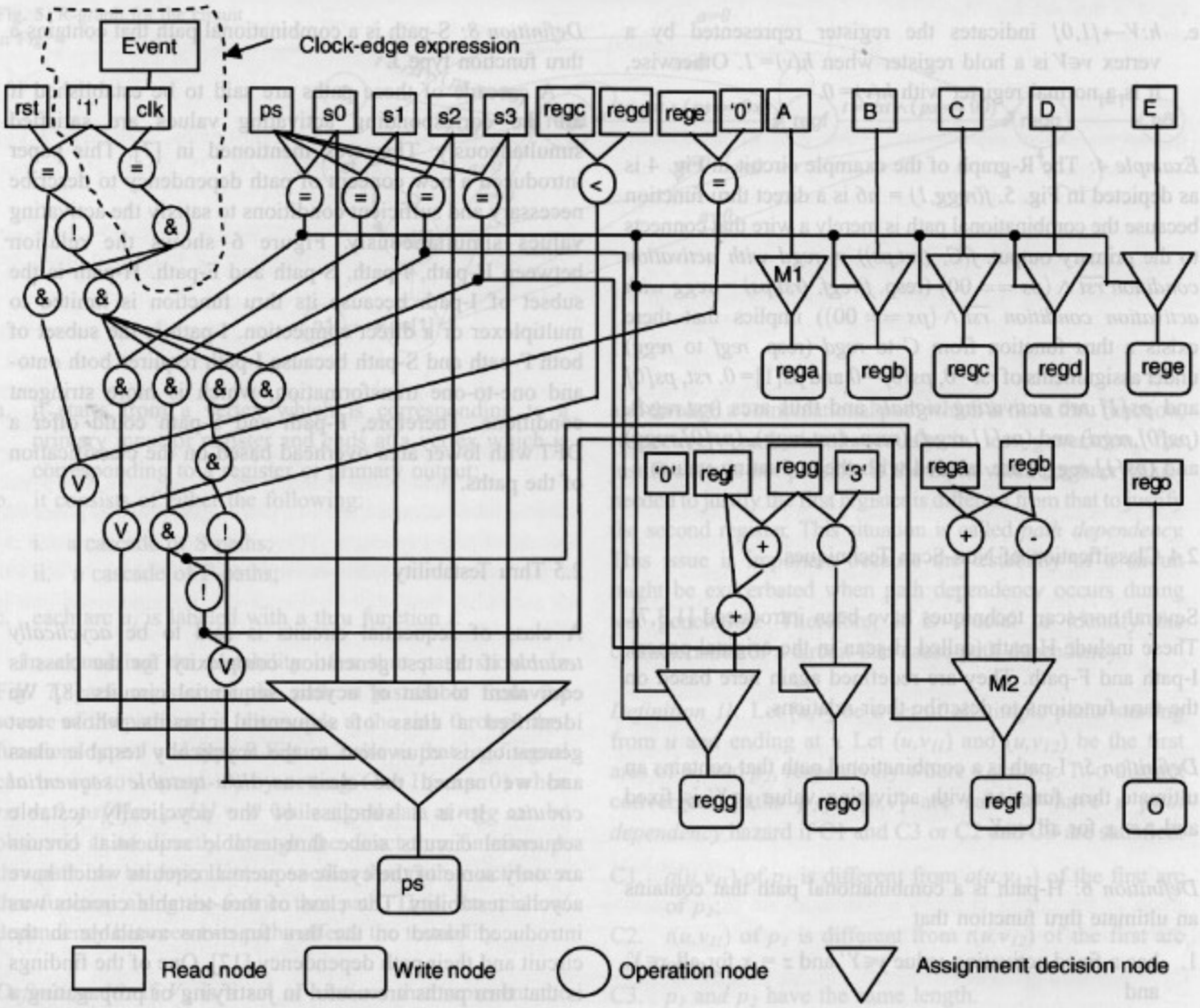
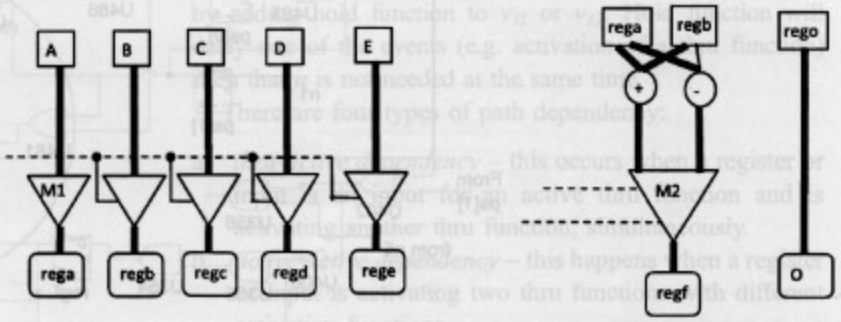


Fig. 2 Assignment decision diagram of *ex1*

d. $a: A \rightarrow \{1,0,\varnothing\}$ defines the Boolean value of the input/register u that activates the thru function whose output data variable is represented by vertex v. $a(u,v) = \varnothing$ means the combinational path (u,v) is not a logic that activates the thru function with output data variable v.

Fig. 3 Thru function extraction



e. $h:V \rightarrow \{1,0\}$ indicates the register represented by a vertex $v \in V$ is a hold register when $h(v)=1$. Otherwise, it is a normal register with $h(v)=0$.

Example 4: The R-graph of the example circuit in Fig. 4 is as depicted in Fig. 5. $f(regg,1) = n6$ is a direct thru function because the combinational path is merely a wire that connects to the primary output. $f(C, (rst,ps)) = regd$ with activation condition $\overline{rst} \wedge (ps == 00)$ (resp. $f(regf, (rst,ps) = regg$ with activation condition $\overline{rst} \wedge (ps == 00)$) implies that there exists a thru function from C to $regd$ (resp. $regf$ to $regg$) under assignments of $rst=0$, $ps[0]=0$ and $ps[1]=0$. rst, $ps[0]$ and $ps[1]$ are *activating signals* and thus arcs $(rst,regd)$, $(ps[0],regd)$ and $(ps[1],regd)$ (resp. $(rst,regg)$, $(ps[0],regg)$ and $(ps[1],regg)$) are labeled with the activating values.

2.4 Classification of Non-Scan Techniques

Several non-scan techniques have been introduced [1,3,7]. These include H-path (called H-scan in the original paper), I-path and F-path. They are redefined again here based on the thru functions to describe their relations.

Definition 5: I-path is a combinational path that contains an ultimate thru function with activating value $y \in Y'$ is fixed and $z = x$ for all $x \in X$.

Definition 6: H-path is a combinational path that contains an ultimate thru function that

1. has a fixed activating value $y \in Y'$ and $z = x$ for all $x \in X$; and
2. is either a multiplexing function, or a direct data transfer without an activating function ($Y' = \varnothing$).

Definition 7: F-path is a combinational path that contains a thru function type P.

Definition 8: S-path is a combinational path that contains a thru function type J.

A cascade of these paths are said to be established if all the corresponding activating values are satisfied simultaneously. This was mentioned in [7]. This paper introduced a new concept of path dependency to describe necessary and sufficient conditions to satisfy the activating values simultaneously. Figure 6 shows the relation between H-path, I-path, S-path and F-path. H-path is the subset of I-path because its thru function is limited to multiplexer or a direct connection. I-path is the subset of both F-path and S-path because I-path requires both onto- and one-to-one transformation, which is more stringent conditions. Therefore, F-path and S-path could offer a DFT with lower area overhead based on the classification of the paths.

2.5 Thru Testability

A class of sequential circuits is said to be *acyclically testable* if the test generation complexity for the class is equivalent to that of acyclic sequential circuits [8]. We identified a class of sequential circuits whose test generation is equivalent to the acyclically testable class and we named the class as *thru-testable sequential circuits*. It is a subclass of the acyclically testable sequential circuits since thru-testable sequential circuits are only some of the cyclic sequential circuits which have acyclic testability. The class of thru-testable circuits was introduced based on the thru functions available in the circuit and their path dependency [17]. One of the findings is that thru paths are useful in justifying or propagating a signal in the circuit.

Definition 9. Let R-graph $G = \{V,A,t,a,h\}$ represent a given sequential circuit S. A thru path is a simple path of the R-graph such that

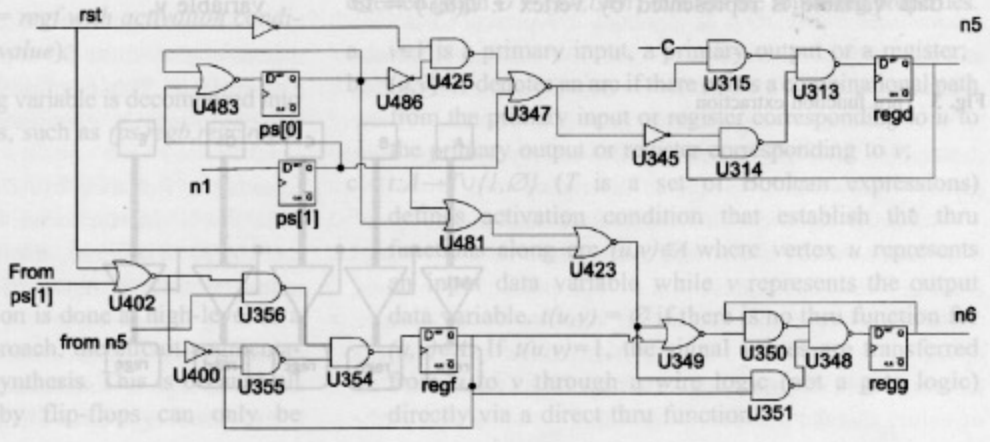


Fig. 4 An example circuit

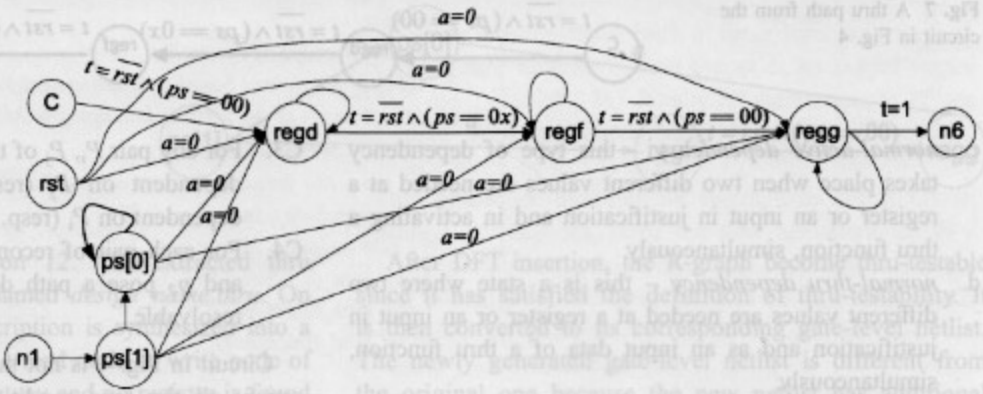


Fig. 5 R-graph for the circuit in Fig. 4

a. it starts from a vertex which is corresponding to a primary input or register and ends at a vertex which is corresponding to a register or primary output;
b. it consists of either the following:
   i. a cascade of S-paths;
   ii. a cascade of F-paths;
c. each arc a_i is labeled with a thru function t.

In examining the testability of a thru path shown in Fig. 7, each register on the path is justifiable from the source of the path and is observable at the sink through thru functions type J and type P. Figure 7 shows that *regd* signal can be set to 1 (resp. 0) by setting C to 1 (resp. 0) when $rst = 0$, $ps[0] = ps[1] = 0$ while signal data at *regg* can be observed at *n6* directly through the direct thru function. A thru path can be dependent on another thru path to activate a thru function along the former thru path. Thus, the relation/dependency between two paths affects the testability.

Definition 10. If V_{ti} is a set of vertices that represent the activator of a thru function t_i in a thru path P_m, P_m is said to be dependent on V_{ti}. Furthermore, if V_{ti} contains a vertex in a thru path P_n, P_m is said to be *dependent* on P_n.

For instance in Fig. 7, V_{ti} = {*rst, ps[0], ps[1]*} is the variables of activator for the ultimate thru function *f(regd, (rst,ps))* = *regf*. When an input (or a register) in a given circuit is used to justify two registers through a thru function

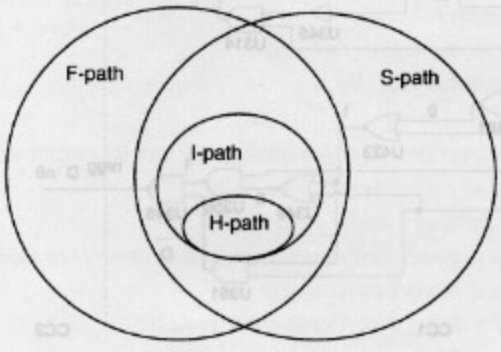


Fig. 6 Relation of F-path, S-path, I-path, and H-path

and through a combinational path which is not a thru function, respectively and simultaneously in test generation, the justification is not possible if the input value (register value) needed to justify the first register is different from that to justify the second register. This situation is called *path dependency*. This issue is important because the testability of a circuit might be exacerbated when path dependency occurs during test generation. Therefore, it is crucial to identify the characteristics of a circuit that cause path dependency.

Definition 11. Let $[u,v]$ be a set of all simple paths starting from u and ending at v. Let (u,v_{11}) and (u,v_{12}) be the first arcs of p_1 and p_2, respectively where $v_{11} \neq v_{12}$. Two distinct convergent paths $p_1,p_2 \in [u,v]$ are said to have a *path dependency* hazard if C1 and C3 or C2 and C3 are satisfied.

C1. $a(u,v_{11})$ of p_1 is different from $a(u,v_{12})$ of the first arc of p_2;
C2. $t(u,v_{11})$ of p_1 is different from $t(u,v_{12})$ of the first arc of p_2;
C3. p_1 and p_2 have the same length.

C1 implies that the signal value required at u to activate the thru function whose output is v_{11} is different from that to activate the thru function whose output is v_{12}. C2 means that at least one of the combinational paths (u,v_{11}) and (u,v_{12}) is a thru function. According to C3, p_1 *and* p_2 are two convergent paths. When C1 and C3 are satisfied, it's likely that u is needed simultaneously to activate two different thru functions during test generation. If a pair of paths is identified to have path dependency hazard, it can be resolved by adding hold function to v_{11} or v_{12}. Hold function will delay one of the events (e.g. activation of a thru function) such that u is not needed at the same time.

There are four types of path dependency:

a. *thru-active dependency* – this occurs when a register or input is an input for an active thru function and is activating another thru function, simultaneously.
b. *active-active dependency* – this happens when a register or input is activating two thru functions with different activating functions.

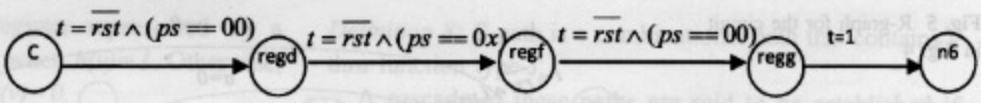


Fig. 7 A thru path from the circuit in Fig. 4

c. *normal-active dependency* – this type of dependency takes place when two different values are needed at a register or an input in justification and in activating a thru function, simultaneously.
d. *normal-thru dependency* – this is a state where two different values are needed at a register or an input in justification and as an input data of a thru function, simultaneously.

One example of path dependency occurring in the circuit of Fig. 4 is illustrated in the time expansion model in Fig. 8. In Fig. 8, let the output of *regf* be the fault location of a stuck-at-0 fault. We need to assign *1* to *regf* to excite the fault and then we need to propagate the fault effect *D* to output *n6*. To do so, *ps[1]* at time frame *CC0* is used to justify *ps[0]* at the next time frame *CC1* and to activate the thru function that transfers *1* from *regd* at *CC0* to *regf* at *CC1* to activate the fault, simultaneously. The problem in this case is that *ps[1]* needs to be assigned *1* and *0* at the same time in order to accomplish the above-mentioned justification and thru function activation. Therefore, any path dependencies have to be resolved to improve the testability. Based on several aspects discussed above, we define a class of circuits which is thru-testable and thus easily testable.

Definition 12. A circuit is said to be *thru-testable* if the R-graph of the circuit contains a minimum set of thru paths such that the following conditions are satisfied.

C1. The thru paths cover all the vertices of the minimum vertex feedback set (MFVS).
C2. For any thru path P_i, P_i is not dependent on itself.
C3. For any pair P_i, P_j of the thru paths, if P_i (resp. P_j) is dependent on P_j (resp. P_i), P_j (resp. P_i) is not dependent on P_i (resp. P_j).
C4. For each pair of reconvergent paths p_1 and p_2, if p_1 and p_2 pose a path dependency hazard, it must be resolvable.

Circuit in Fig. 4 is not thru-testable because it does not satisfy all the four conditions defined in Definition 12. From the R-graph of the circuit, MFVS includes vertices *ps[0]*, *regd*, *regf* and *regg*. *regd*, *regf* and *regg* are on the thru path in the R-graph but *ps[0]* does not belong to any thru path. This violates C1. In addition, C4 is not satisfied too because there is a pair of paths which poses the hazard of path dependency. Figure 9 shows the paths and its dependency. In the next section, the circuit in Fig. 4 is used to exemplify how a given circuit can be augmented into a thru testable circuit.

3 Design-for-Testability Method Based on Thru-Testability

We introduce a new DFT approach based on thru-testability. Figure 10 illustrates our DFT methodology. The input to our DFT insertion methodology includes RTL description which can consist of a non-separable datapath and controller.

Firstly, an RTL description is transformed into a ADD description for thru function extraction. A thru function whose activating signals are the same as the thru function output or input is not selected. This is to prevent the

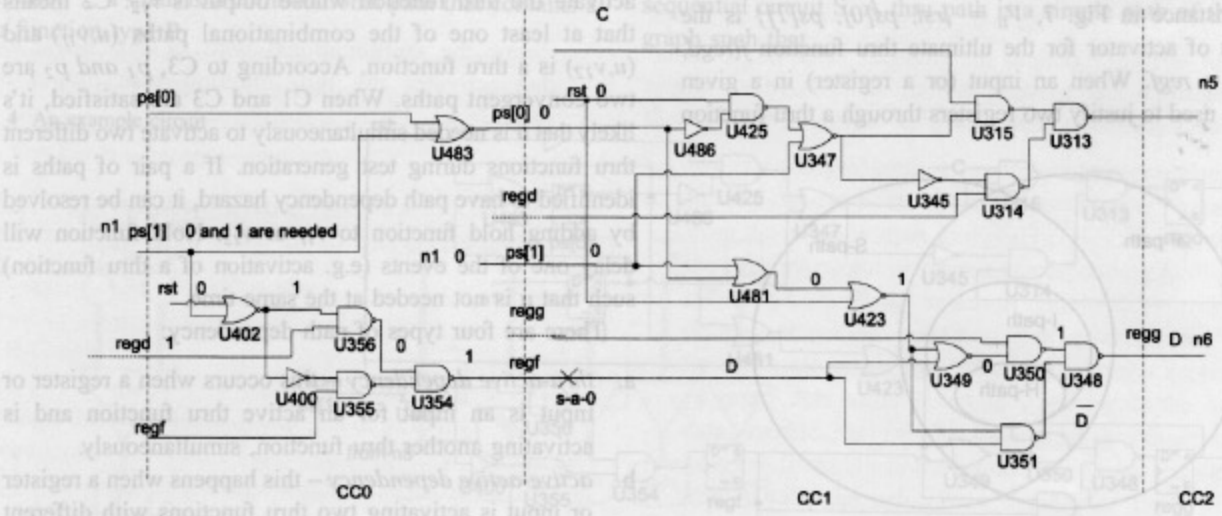


Fig. 8 Path dependency

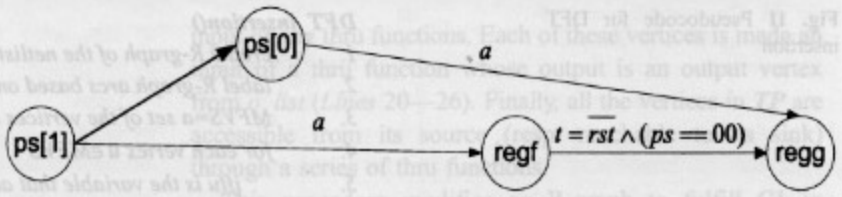


Fig. 9 Path dependency between path *ps[1]→ps[0]→regg* and path *ps[1]→regf→regg*

violation of C2 in Definition 12. The extracted thru functions are kept in a file named *design_name.thru*. On the other hand, the RTL description is synthesized into a gate-level netlist. Sometimes, a read node or write node of the ADD maybe synthesized away and no register is found to be corresponding to the node. If such node that is also the input or output of an extracted thru function exists, the extracted thru function cannot be considered. Prior to DFT insertion, the gate-level netlist is transformed into its R-graph. Then, during DFT insertion, the connectivity of the R-graph will be analyzed based on a set of existing thru functions in *design_name.thru* to add minimum number of new thru functions and new hold functions to make the circuit thru-testable. There are three tasks under DFT insertion (Fig. 11):

a. identifying vertices belonging to MFVS (*Line 3)*;
b. adding new thru functions (*Lines 8–9)*; and
c. adding new hold functions (*Line 10)*.

After DFT insertion, the R-graph become thru-testable since it has satisfied the definition of thru-testability. It is then converted to its corresponding gate-level netlist. The newly generated gate-level netlist is different from the original one because the new netlist has additional gates to realize the new thru functions and some flip-flops are changed to the flip-flop with hold function. We elaborate the algorithms for DFT insertion in the following sub-sections.

3.1 Minimum Feedback Vertex Set

We are using the exact algorithm introduced in [6] to identify the flip-flops that belong to minimum feedback vertex set (MFVS). This algorithm involves graph transformations, a partitioning scheme used in the branch and bound procedure and pruning techniques based on an integer linear programming. The author has shown the effectiveness of the algorithm through a reasonable com-

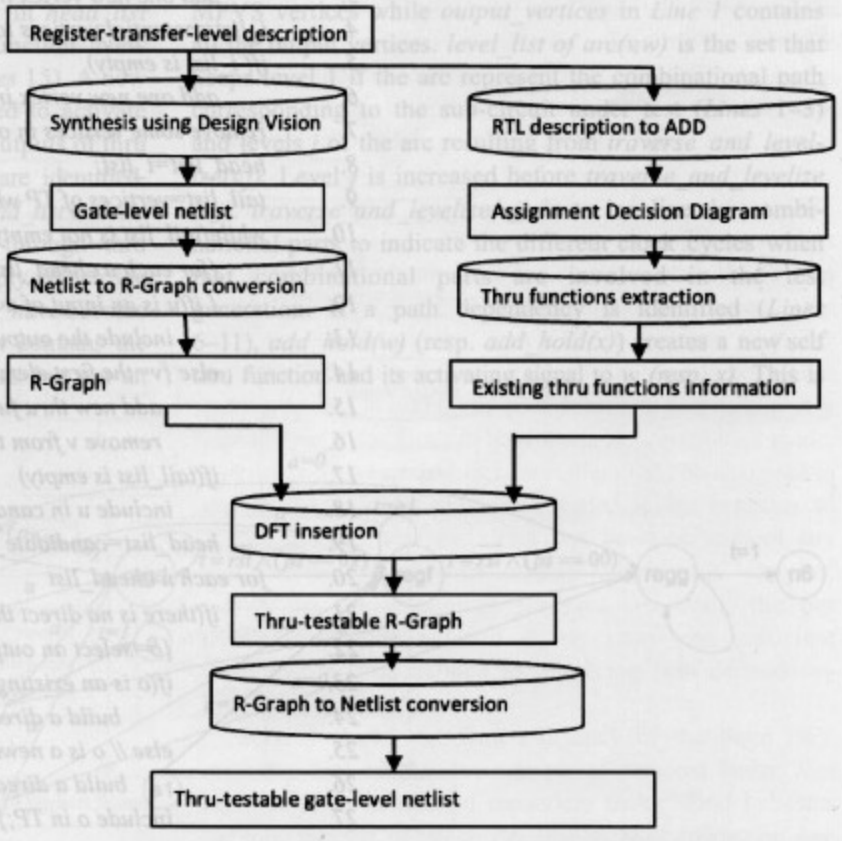


Fig. 10 Thru-testable designfor-testability methodology

Fig. 11 Pseudocode for DFT insertion

```
DFT_insertion()
1.  create R-graph of the netlist;
2.  label R-graph arcs based on existing thru functions and activating function;.
3.  MFVS=a set of the vertices belonging to MFVS;
4.  for each vertex u ∈MFVS
5.      if(u is the variable that activates a thru function)
6.          include u in thru path TP1;
7.      else include u in thru path TP2;
8.  add_thru_function(TP2);
9.  add_thru_function(TP1);
10. add_hold_functions();
```

putation time when this algorithm is applied on ISCAS'89 benchmark circuits. Applying this algorithm on the R-graph in Fig. 5, vertices *ps[0], regd, regf* and *regg* are identified as MFVS vertices.

3.2 New Thru Functions Insertion

After recognizing the vertices belonging to MFVS, we divide the MFVS vertices into two groups; those activate a thru function and the others. For instance, *ps[0]* is a MFVS vertex that activates thru functions so it is included in *TP1*. The rest of the MFVS vertices are included in *TP2*. Therefore, *TP1* = *{ps[0]}* and *TP2* = *{regd, regf, regg}*. *TP1* and *TP2* contain a set of vertices that has a potential to form a set of thru paths, respectively. For each *TP1* and *TP2*, new thru functions are added based on the existing thru function. Figure 12 shows the detailed procedure.

Fig. 12 Pseudocode for *add_thru_function(TP)*

```
add_thru_function (TP)
1.  o_list=potential vertices to be the sink of the thru path TP;
2.  if(o_list is empty)
3.      add one new vertex into o_list;
4.  i_list=potential vertices to be the source of the thru path TP;
5.  if( i_list is empty)
6.      add one new vertex into i_list;
7.  remove some vertices in o_list or i_list until their size are equal;
8.  head_list=i_list;
9.  tail_list=vertices of TP which are not thru function outputs;
10. while(tail_list is not empty)
11.     {for each u ∈head_list
12.     { if(u is an input of a thru function)
13.         include the output of the thru function v in candidate_tail_list;
14.     else {v=the first element in tail_list;
15.         add new thru function t=(activator==1) to arc (u,v);
16.         remove v from tail_list;}
17.     if(tail_list is empty)
18.         include u in candidate_tail_list;
19.     head_list=candidate_tail_list;}
20. for each u ∈head_list
21.     if(there is no direct thru function connecting u to an output vertex)
22.         {o=select an output vertex from o_list;
23.         if(o is an existing output)
24.             build a direct thru function t=(activator==1) to (u,o);
25.         else // o is a newly added output
26.             build a direct thru function t=1 to (u,o);
27.         include o in TP;}
```

```
add_hold_function()
1.    for each u∈MFVS∪output_vertices
2.      {for each v∈direct_predecessor(u)
3.        {include 1 in level_list of arc (v,u);
4.        j=2;
5.        traverse_and_levelize(v,j);}
6.    for each v∈MFVS
7.      for each pair w,x∈direct_successor(v)
8.        if( (level_list of arc (v,w)∩ level_list of arc (v,x)≠ϕ)
9.          if( a(v,w)≠a(v,x) || t(v,w)≠t(v,x) )
10.           {add_hold(w);
11.           add_hold(x);}
12.   clear_levelization(); }

traverse_and_levelize(v,j)
1.    for each u∈direct_predecessor(v)
2.      if((u∈MFVS && ( a(u,v)≠ϕ || t(u,v)≠ϕ)) || u∈MFVS)
3.        {include j in level_list of arc(u,v);
4.        j++;
5.        traverse_and_levelize(u,j);}
```

Fig. 13 Pseudocode for *add_hold_function()*

The procedure might generate a set of thru paths (not only a single thru path) from the vertices in ***TP***. The number of thru paths to be generated is determined by the size of *i_list* or *o_list*, which is less (*Lines* 1–7). *head_list* initially take the input vertices as its members while *tail_list* includes the *MFVS vertices* which are not outputs for any thru functions in *design_name.thru* (*Lines* 8–9). Lines 10–19 link each vertex in *head_list* with a vertex in *tail_list* through a thru function. New thru function is introduced if necessary (*Lines* 15). A new activator (an activating signal) is introduced to activate all the new thru functions for ***TP***. After the outputs of thru functions with input vertices from *head_list* are identified, these vertices will be treated as members in *head_list* and the process to link the vertices in *head_list* through thru functions is repeated until the *tail_list* is empty. When the *tail_list* is empty, it means all the vertices in ***TP*** have become thru function outputs. At this stage, *head_list* contains the vertices of ***TP*** which are thru function outputs but not an input of any thru functions. Each of these vertices is made an input of a thru function whose output is an output vertex from *o_list* (*Lines* 20—26). Finally, all the vertices in ***TP*** are accessible from its source (resp. reachable to its sink) through a series of thru functions.

This procedure modifies an R-graph to fulfill C1 in Definition 12. Besides, since the new activators for *TP1* and *TP2* are different, this makes sure that any two paths are not mutually dependent as stated in C3 of Definition 12. The following sub-section describes another task to guarantee C4 of Definition 12.

3.3 New Hold Functions Insertion

To resolve any potential path dependencies, it is sufficient to add hold functions (also called self thru functions) to the vertices which are direct successors to the source vertex of any two paths which have path dependency hazards. Figure 13 shows the pseudocode to select vertices for new hold function insertion.

To identify which registers to be augmented with hold function, we first levelize R-graph starting from a MFVS vertex or an output vertex. Then, the second vertices of two paths with path dependency hazard are augmented with hold functions or self thru functions. The activating signals for these two hold functions are different. *MFVS* in *Line 1* is a set that contains all the MFVS vertices while *output_vertices* in *Line 1* contains all the output vertices. *level_list of arc(v,w)* is the set that keeps level 1 if the arc represent the combinational path corresponding to the sub-circuit under test (*Lines* 1–3) and levels *j* of the arc resulting from *traverse_and_levelize(v,j)*. Level *j* is increased before *traverse_and_levelize (v,j)*. *traverse_and_levelize(v,j)* is to levelize the combinational parts to indicate the different clock cycles when that combinational parts are involved in the test generation. If a path dependency is identified (*Lines* 6–11), *add_hold(w)* (resp. *add_hold(x)*) creates a new self thru function and its activating signal to *w (resp. x)*. This is

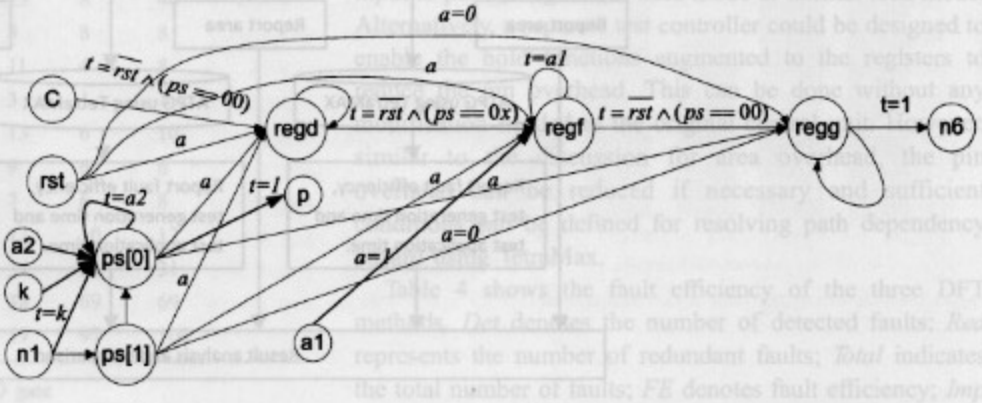


Fig. 14 An example of thru-testable R-graph

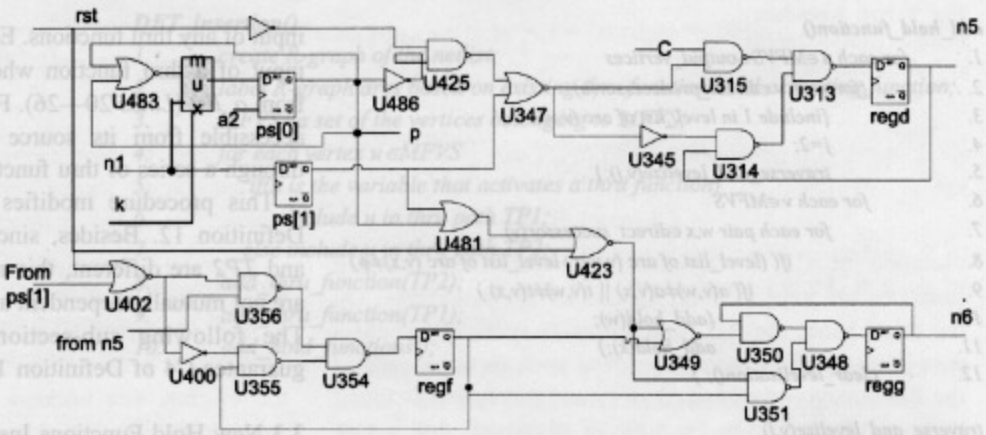


Fig. 15 Thru-testable circuit after applying TT DFT methodology on circuit in Fig. 4

equivalent to adding a hold function to register *w* (resp. register *x*). Then, the levels kept in each arc is removed before the procedure of *add_hold_function()* is executed for other fault excitation graphs.

Example 5: Figure 14 shows the thru-testable R-graph obtained from the augmentation of R-graph in Fig. 5 based on thru-testable DFT. New thru functions $f(n1,k) = ps[0]$ and $f(ps[0],1) = p$ are added to form a new thru path $n1 \rightarrow ps[0] \rightarrow p$ to cover the only MFVS vertex not covered by the thru path in Fig. 6. A self thru-function or hold function is added to *ps[0]* to resolve the path dependency hazard. Based on the thru-testable, the circuit in Fig. 4 can be augmented into thru-testable as shown in Fig. 15.

To show the effectiveness of our DFT method, we have set up an experiment and augmented some benchmark circuits of ITC'99. The following section explains our experiment setup and result.

4 Experiment Setup and Result

4.1 Experiment Setup

Figure 16 shows the planning and flow of our experiment in order to show the effectiveness of thru-testable DFT methodology. For a circuit that has path dependency hazard, the path dependency may or may not happen depending on the test generation model used in an ATPG. Therefore,

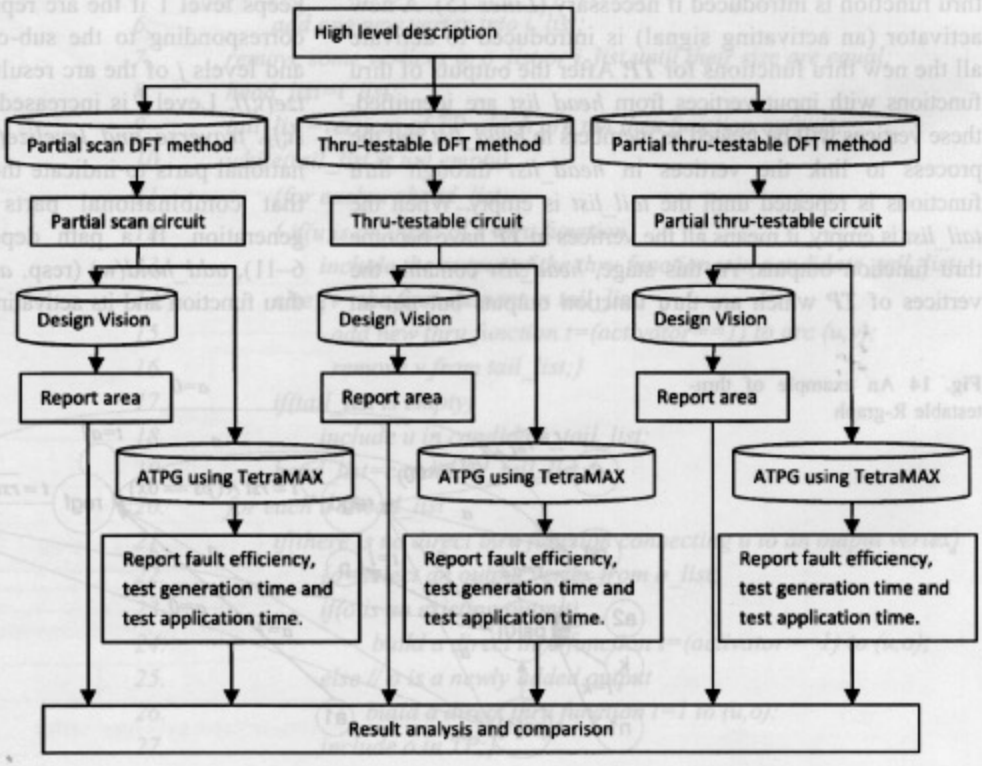


Fig. 16 Experiment flow

adding hold functions is just a sufficient condition to resolve any path dependency that might happen in a sequential circuit.

We used Synopsys TetraMax to run ATPG on ITC'99 benchmark circuits. Since it is difficult to determine whether a path dependency will happen or not, we have also conducted experiment on the partially thru-testable circuits which is a thru-testable circuit without hold functions to show that the partially thru-testable DFT is good enough if path dependency does not occur much during test generation. Partial thru-testable DFT methodology can be simply developed based on thru-testable DFT methodology by removing *Line10* in *DFT_insertion()* procedure. We have also conducted experiment on the benchmark circuits that have been modified by partial scan DFT methodology for comparison purpose. The partial scan selection is based on minimum feedback vertex set (MFVS). Our method is not compared with other non-scan techniques [1,3,7,8] because those DFT techniques are implemented on the structural RTL circuits which have separate datapath and control units. ITC'99 benchmark circuits are only available at functional RTL whereby its datapath and control units are non-separable.

Table 1 presents the characteristics of the benchmark circuits and *ex1* used as an example for this paper. The benchmark consists of small circuits like b03 and also large circuits like b14 and b15. Column *#FF* shows the number of flip-flops in the circuit while Column *Area* indicates the size of the circuit in unit of the size of an inverter. Column *IO pins* shows the number of primary inputs (*PI*) and primary outputs (*PO*) while the last column shows the number of thru functions exists in the original benchmark circuits before any modification. These thru functions are extracted from the high level descriptions using the method mentioned in our thru-testable DFT methodology. Notice that some circuits like b04 has many thru functions available in their original design.

Table 1 Characteristics of the benchmark circuits

Circuit	#FFS	Area	IO pins		# Thru functions
			PI	PO	
ex1	59	450.5	35	8	8
b03	30	211	6	4	17
b04	66	589.5	13	8	40
b07	45	397.5	3	8	8
b08	21	175	11	4	8
b09	28	198	3	1	1
b10	17	172	13	6	10
b11	31	394	9	6	6
b12	121	1054.5	7	6	8
b13	51	388.5	12	10	12
b14	215	5325.5	34	54	31
b15	417	6405	38	69	69
b17	1317	19811.5	39	97	100

one unit area = the size of a NAND gate

4.2 Experimental Result

Table 2 shows the area overhead resulted from partial scan DFT, partially thru-testable DFT (*PTT DFT*) and thru-testable DFT (*TT DFT*). For all these three DFT methods, area overhead is caused by the new thru functions added to the circuit during DFT insertion. In addition, the area overhead for *TT DFT* also comes from the newly added hold functions. For all the benchmark circuits, the area overhead in *PTT DFT* method is less compared to partial scan DFT. This is the advantage of considering and utilizing the existing thru functions during DFT insertion. However, it is not so for *TT DFT* in the cases of *ex1, b09, b11, b12* and *b14* because of the area overhead of newly added hold functions. This can probably be improved since adding hold functions is a sufficient condition. Figure 17 shows the area overhead caused by the hold functions denoted by *TT DFT (HF)* and the new thru functions *TT DFT (TF)* explicitly. If we can reduce hold functions, it's possible that the area overhead for these cases become less than the area overhead in partial scan DFT (*PS DFT*) which is contributed by thru functions only. Overall, the area overhead incurred in *TT DFT* is less than that in partial scan DFT.

Table 3 shows the pin overhead of these three DFT methods. Partial scan DFT needs one extra pin as a select pin to switch a circuit from test mode to normal and vice versa. The proposed *PTT DFT* has more pins because the activator for thru paths from *TP1* is different from the activator for thru paths from *TP2*. *TT DFT* needs even more extra pins due to additional hold functions that need a new input to put the register in hold mode or normal load mode. Alternatively, a separate test controller could be designed to enable the hold functions augmented to the registers to reduce the pin overhead. This can be done without any modification needed to the original control unit. However, similar to the discussion for area overhead, the pin overhead can be reduced if necessary and sufficient conditions can be defined for resolving path dependency hazard using TetraMax.

Table 4 shows the fault efficiency of the three DFT methods. *Det* denotes the number of detected faults; *Red* represents the number of redundant faults; *Total* indicates the total number of faults; *FE* denotes fault efficiency; *Imp*

Table 2 Area overhead comparison

Circuit	Area	Partial scan			PTT DFT			TT DFT		
		#TF	Area	AO(%)	#TF	Area	AO(%)	#TF	Area	AO(%)
ex1	450.5	59	657	45.84	52	629	39.62	52	687	52.50
b03	211	30	316	49.76	13	256.5	21.56	13	271.5	28.67
b04	589.5	66	820.5	39.19	26	680.5	15.44	26	713.5	21.03
b07	397.5	44	551.5	38.74	36	527	32.58	36	549.5	38.24
b08	175	21	248.5	42.00	13	220.5	26.00	13	231	32.00
b09	198	21	271.5	37.12	20	268	35.35	20	278.5	40.66
b10	172	17	231.5	34.59	7	196.5	14.24	7	205	19.19
b11	394	31	502	27.41	25	488	23.86	25	503.5	27.79
b12	1054.5	117	1464	38.83	109	1443	36.84	109	1501.5	42.39
b13	388.5	50	563.5	45.05	39	521.5	34.23	39	546.5	40.67
b14	5325.5	213	6071	14.00	184	5969.5	12.09	184	6076.5	14.10
b15	6405	414	7854	22.62	348	7623	19.02	348	7831.5	22.27
Average				36.26			25.9			31.63

one unit area = the size of a NAND gate

tabulates the fault efficiency improvement in percentage compared to partial scan technique. The last row of Table 4 shows the average fault efficiency. For *PTT DFT* and *TT DFT*, the last row also displays the average improvement of fault efficiency compared to partial scan technique. All the fault efficiency of thru-testable sequential circuit are comparable or less than fault efficiency in partial scan designed circuits. For the cases including *ex1, b03, b04, b09, b10, b11, b12, b13 and b14* which show the comparable results, this maybe because only little path dependency has occurred during test generation using TetraMax. Thus, adding new hold functions to resolve path dependency cannot improve further the fault efficiency of these circuits. The point to be emphasized here is that *TT DFT* can improve the testability of difficult-to-test circuits effectively. Moreover, the average fault efficiency is the highest among all the DFT methods. This is proved by the experimental result of fault efficiency for circuits of *b07, b08* and *b15*. This can further be supported by the experimental result of test generation time shown in Table 5 where test generation time for the thru-testable circuits of *b07, b08* and *b15* is much shorter than the test generation time for the partial scan designed *b07, b08* and *b15*. The test generation time is also shorter even for the other circuits like *b10* and *b13* whose thru-testable version has fault coverage comparable to its partial scan designed counterparts. Table 6 shows the test application time in

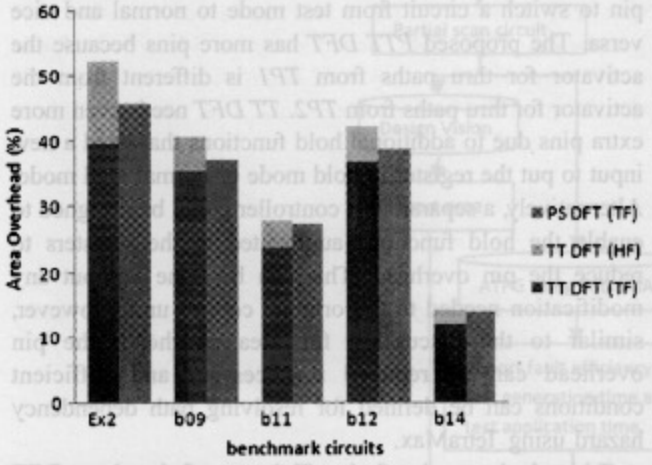


Fig. 17 Comparison between area overhead of PS DFT and area overhead of TT DFT

Table 3 Pin overhead comparison

Circuit	Partial Scan			PTT DFT			TT DFT		
	PI	PO	total	PI	PO	total	PI	PO	total
ex1	1	0	1	2	1	3	5(3)	1	6
b03	1	0	1	3	1	4	5(2)	1	6
b04	1	0	1	2	1	3	5(3)	1	6
b07	1	0	1	4	1	5	7(3)	1	8
b08	1	0	1	2	1	3	6(4)	1	7
b09	1	0	1	3	1	4	5(2)	1	6
b10	1	0	1	2	1	3	7(5)	1	8
b11	1	0	1	2	1	3	5(3)	1	6
b12	1	0	1	3	1	4	7(4)	1	8
b13	1	0	1	2	1	3	7(2)	1	8
b14	1	0	1	3	0	3	6(3)	0	6
b15	1	0	1	3	4	7	7(4)	4	11

Table 4 Fault efficiency (in percentage %)

Ckt.	Ori.	Partial Scan				PTT DFT					TT DFT				
		Det	Red	Total	FE	Det	Red	Total	FE	Imp	Det	Red	Total	FE	Imp
ex1	69.35	4220	0	4228	99.83	4048	0	4060	99.78	−0.05	4117	0	4302	97.78	−2.05
b03	69.58	1703	0	1704	99.94	1016	3	1340	75.99	−23.95	1450	3	1468	98.98	−0.96
b04	83.39	4742	36	5040	94.76	3710	36	4168	89.79	−4.97	4121	36	4446	93.45	−1.31
b07	4.11	2303	4	3300	69.9	2590	4	3136	82.77	12.87	2866	4	3318	86.56	16.66
b08	92.62	1352	0	1528	88.48	1277	0	1360	93.97	5.49	1441	0	1452	99.24	10.76
b09	88.18	1417	0	1421	99.51	1344	0	1412	95.18	−4.33	1491	0	1502	99.27	−0.24
b10	94.32	1485	0	1486	99.93	1268	6	1274	99.53	−0.4	1350	2	1352	99.85	−0.08
b11	81.1	3393	15	3432	99.3	3269	15	3308	99.27	−0.03	3395	15	3438	99.18	−0.12
b12	13.75	7968	2	8962	88.95	7705	5	8796	87.66	−1.29	8135	5	9272	87.8	−1.15
b13	34.23	3081	67	3170	99.29	2834	67	2914	99.54	0.25	3051	67	3124	99.8	0.51
b14	63.08	47104	359	48024	98.82	46253	362	47390	98.35	−0.47	47191	362	48252	98.54	−0.28
b15	4.54	42307	234	59888	70.97	31026	120	58446	53.20	−17.77	54039	200	60146	90.15	19.18
Ave.	58.19				92.47				89.59	−2.88				95.88	3.41

clock cycles for original circuits, partial scan designed circuit, partially thru-testable circuits and thru-testable circuits where thru-testable circuits has longer test application time.

The advantages of using thru testable DFT methodology is expected to become more obvious if larger and more complicated circuits with more path dependency hazards are used in the experiment. However, the experiment in the current work is limited to the benchmark circuits up to *b15* because the current version of script cannot support hierarchical design of the benchmark such as *b17, b18, b19, b22* and so on. In this paper, thru testable DFT method is performed at gate level, which is a complicated process. If the method is extended to be applied for RTL circuits, larger circuits can be used to show the effectiveness of the method.

5 Conclusion

A new design-for-testability method called thru-testable DFT method has been introduced in this paper based on thru-testability. For all the benchmark circuits, the fault efficiency of the thru-testable circuits is comparable to that of the partial scan designed circuit. In addition, our method has also shown higher fault efficiency, less hardware area overhead, and shorter test generation time compared to partial scan technique in difficult-to-test circuits.

Table 5 Test generation time comparision (in seconds)

Circuits	Original	Partial Scan	PTT DFT	TT DFT
ex1	9263.78	108.93	90.49	1658.54
b03	2229.25	48.10	354.68	236.14
b04	1341.99	227.48	922.45	456.55
b07	16821.17	11385.35	9220.63	4608.07
b08	326.28	878.09	245.88	69.61
b09	516.66	42.80	689.31	920.43
b10	514.10	88.95	85.54	51.45
b11	4999.26	64.58	98.94	243.25
b12	23725.42	13360.27	32471.66	17217.38
b13	10848.61	113.88	95.80	96.18
b14	6540.87	3890.99	4698.42	5279.06
b15	104084.22	80521.65	80168.35	72405.00

Table 6 Test application application (in clock cycles)

Circuits	Original	Partial Scan	PTT DFT	TT DFT
ex1	321	1467	970	1533
b03	340	1649	617	2633
b04	1448	1784	2156	2682
b07	8	2606	2663	5564
b08	1284	463	1211	2186
b09	2652	5958	4456	6773
b10	1013	998	1478	1664
b11	1022	2039	2412	2815
b12	132	6204	11492	15904
b13	465	1751	3866	5226
b14	1335	15518	21539	21623
b15	50	3555	2094	19978

References

1. Abadir MS, Breuer MA (1985) A knowledge based system for designing testable VLSI chips. IEEE Des Test Comput 2 (4):56–68
2. Agrawal VD, Cheng KT (1990) Finite state machine synthesis with embedded test function. J Electron Test Theory Appl 1 (3):221–228
3. Asaka T, Bhattacharya S, Dey S, Yoshida M (1997) H-scan+: a practical low-overhead RTL design-for-testability technique for industrial designs. Proc International Test Conference: 265–274
4. Bhattacharya S, Dey S (1996) H-scan: a high level alternative to full-scan testing with reduced area and test application overheads. Proc IEEE 14th VLSI Test Symposium (VTS'96): 74–80
5. Chaiyakul V, Gajski DD (1992) Assignment decision diagram for high level synthesis. Technical Report: 5–50
6. Chakradhar ST, Balakrishnan A, Agrawal VD (1994) An exact algorithm for selecting partial scan flip-flops. Proc. of 31st ACM/ IEEE Design Automation Conference: 81–86
7. Freeman S (1988) Test generation for datapath logic: the F-path method. IEEE Trans Solid State Circuits 23(2):421–427
8. Fujiwara H, Iwata H, Yoneda T, Ooi CY (2008) A nonscan design-for-testability method for register-transfer-level circuits to guarantee linear-depth time expansion models. IEEE Trans CAD Integr Circuits Syst 27(9):1535–1544
9. Gupta R, Gupta R, Breuer MA (1990) The BALLAST methodology for structured partial scan design. IEEE Trans Comput 39 (4):538–548
10. Kanjilal S, Chakradhar ST, Agrawal VD (1995) Test function embedding algorithms with application to interconnected finite state machines. IEEE Trans CAD 14:1115–1127
11. Kanjilal S, Chakradhar ST, Agrawal VD (1995) A partition and resynthesis approach to testable design of large circuits. IEEE Trans CAD 14:1268–1276
12. Kim KS, Kime CR (1995) Partial scan flip-flop selection by use of empirical testability. J Electron Test Theory Appl 7:47–59
13. Kim YC, Agrawal VD, Saluja KK (2005) Combinational automatic test pattern generation for acyclic sequential circuits. IEEE Trans CAD 24:948–956
14. Lee DH, Reddy SM (1990) On determining scan flip-flops in partial-scan designs. Proc Int Conf on Computer-Aided Design: 322—325
15. Lin C, Marek-Sadowska M, Lee MT, Chen K (1998) Cost-free scan: a low-overhead scan path design. IEEE Trans CAD Integr Circuits Syst 17(19):852–861
16. Norwood RB, McCluskey EJ (1996) Orthogonal scan: low overhead scan for data paths. Proc International Test Conference: 659–668
17. Ooi CY, Fujiwara H (2006) A new class of sequential circuits with acyclic test generation complexity. International Conference on Computer Design: 425–431
18. Pomeranz I, Reddy SM (2005) Autoscan: a scan design without external scan inputs or outputs. IEEE Trans Very Large Scale Integr Syst 13(9):1087–1095

Chia Yee Ooi (S'04–M'07) received the B.E. and M.E. degrees in electrical engineering from the Universiti Teknologi Malaysia, Skudai, Malaysia, in 2001 and 2003, respectively, and the Ph.D. degree in information science from the Nara Institute of Science and Technology, Kansai Science City, Japan, in 2006. She is currently a Lecturer with the Universiti Teknologi Malaysia. Her research interests include VLSI design and testing.

Hideo Fujiwara (S'70–M'74–SM'83–F'89) received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. From 1974 to 1985, he was with Osaka University. From 1985 to 1993, he was with Meiji University, Tokyo, Japan. From1993 to 2011, he was with the Nara Institute of Science and Technology, Kansai Science City, Japan. Since April 2011, he has been with Osaka Gakuin University where he is currently a Professor. He served as an Editor of the Journal of Electronic Testing: Theory and Application and the Journal of Circuits, Systems and Computers from 1989 to 2004 and the VLSI Design: An Application Journal of Custom-Chip Design, Simulation, and Testing from 1992 to 2005. He also served as a Guest Editor for the Special Issues of the IEICE Transactions of Information and Systems. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). His research interests include logic design, digital systems design and test, VLSI CAD, and fault-tolerant computing, which includes high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. Dr. Fujiwara is currently an Advisory Member of the Institute of Electronics, Information and Communication Engineers of Japan (IEICE) Transactions on Information and Systems. He is a Golden Core Member of the IEEE Computer Society, a Fellow of the IEICE, and a Fellow of the Information Processing Society of Japan. He served as an Editor of the IEEE TRANSACTIONS ON COMPUTERS from 1998 to 2002. He was the recipient of the 1977 Institute of Electronic Communications Engineers Young Engineer Award, the 1991, 2000, and 2001 IEEE Computer Society Certificate of Appreciation Awards, the 1994 Okawa Prize for Publication, the 1996 and 2005 IEEE Computer Society Meritorious Service Awards, the 2001 IEEE Computer Society Outstanding Contribution Award, and the 2005 IEEE Computer Society Continuing Service Award.