Low overhead DFT using CDFG by modifying controller

M. Hosseinabady, P. Lotfi-Kamran, F. Lombardi and Z. Navabi

Abstract: A novel design-for-test (DFT) method that requires minor modifications to the controller in the register-transfer level (RTL) description of a circuit is presented. The control/data flow graph representation of an RTL circuit is used for analysing the testability of individual RTL operations within the RTL circuit. Using a non-scan arrangement, existing data paths are utilised to provide controllability and observability to RTL operations. Furthermore, additional data paths are introduced by altering the controller states or adding new transitions. This method considerably reduces the test application time by ignoring unnecessary control states in the test process. The proposed method is applied to behavioural and RTL benchmarks. The results show the effectiveness of this method when compared with some other DFT insertion methods.

1 Introduction

Test generation of VLSI circuits has reached a level of complexity that traditional gate-level automatic test pattern generation (ATPG) techniques cannot often handle. To ease the complexity of test generation, design-for-test (DFT) techniques have been proposed. The most commonly used DFT techniques for VLSI circuits – and particularly applicable to sequential circuits – utilise a scan chain [1]. These techniques rely on modifying a circuit such that ATPG tools can achieve a high fault coverage within a reasonable amount of time. However, a significant disadvantage of the scan-based technique is that at-speed testing with a complete test set is not possible (i.e. all test patterns cannot be applied at the operational speed of the circuit) [1, 2]. In addition, high test application time associated with the scan-based testing causes an increase in the cost of the test. Non-scan DFT approaches have been proposed to avoid the disadvantages of scan techniques such as lack of at-speed testing and increase in test time [3].

A top-down design methodology, which has increasingly been adopted for synthesis, offers an alternative approach to ease test generation and decrease the test application time. High-level definitions of the circuit [e.g. behavioural level or register-transfer level (RTL)] have significantly reduced the number of primitives (as a measure of complexity) compared with a gate-level representation. Consequently, if test generation can be performed at the RTL or behavioural level, the test problem may be more tractable. Test generation techniques can be accomplished easier at the behavioural level using test information from the gate-level description of the design modules. This technique is usually referred to as hierarchical test generation, that is, a hierarchical test generation uses information from different abstraction levels for generating the test vectors.

© The Institution of Engineering and Technology 2007

doi:10.1049/iet-cdt:20050133

Paper first received 1st June 2005 and in revised form 30th January 2006

M. Hosseinabady, P. Lotfi-Kamran and Z. Navabi are with the Department of Electrical and Computer Engineering, University of Tehran, Tehran 14399, Iran

F. Lombardi is with the ECE Department, Northeastern University, Boston, MA 02115, USA

E-mail: mohammad@cad.ece.ut.ac.ir

This paper presents a non-scan DFT method at the RTL for hierarchical testing using post behavioural synthesis information and pre-computed test vectors of individual modules. The proposed method also determines a mechanism for application of pre-computed test vectors to the modules of an RTL description. The RTL structure wherein all modules can obtain their pre-computed test vectors is called a hierarchically testable structure. Our method proposes a scheme to convert an RTL structure to a hierarchically testable one. This scheme adds a few gates to the datapath of the design, resulting in a small area overhead. Then, the proposed method generates a minimal control/data flow graph (CDFG) (called test-CDFG) for each execution unit (module) in the datapath by processing the information that is readily available in the CDFG of the complete design. This minimal CDFG is used for test generation and test application of the corresponding module. This technique adds a small amount of test hardware to a given RTL circuit (obtained by behavioural synthesis) to ensure that the embedded elements in the circuit are hierarchically testable. Since the added hardware is primarily in the controller of the circuit, it has a small overhead, when compared with changes that would have been required if they were done in the datapath of a circuit. In addition, this approach reduces the test application time by reducing the number of clock cycles needed to transfer parallel test data to the datapath modules.

There has been a multitude of research exploring the techniques to perform the test generation at the RTL or behavioural level. These researches can be classified into three categories. The first category [4–9] analyses the RTL hierarchical testability (justification and propagation analysis) of a circuit and on the basis of this analysis proposes methods to translate local vectors to the system vectors. The works in the second category [2, 10–12] propose DFT methodologies to generate hierarchical test vectors and ease the test generation complexity to reach full-fault coverage. Most of the works in this category modify the datapath of an RTL design. Finally, works in the third category [13–16] propose DFT methodologies to reduce test generation and test application times. These methodologies modify the datapath and the controller of the circuit.

Makris and Orailoglu [4, 5] have proposed an RTL analysis methodology that identifies the test justification and

propagation bottlenecks. They introduced two mechanisms (property-based and channel-based schemes) for capturing, without reasoning on the complete functional space, data and control module behaviour related to test translation. Makris *et al.* [6] then introduced a methodology for identifying transparency behaviour appropriate for a hierarchical test. Unlike high-level approaches that identify limited, coarse transparency behaviour, their methodology is capable of extracting a wide class of fine grained transparency functions for arbitrary sub-word bit clusters. A methodology that examines design modules and identifies appropriate vector justification and response propagation requirements for reducing the cost of hierarchical test path construction was proposed in the work of Makris and Orailoglu [7]. Using the transparency scheme [4, 5], Makris *et al.* [8, 9] proposed a hierarchical test generation method for DFT-free controller-datapath. In addition to introducing the concept of influence table, they translated the controller of the design to several tables, by which the interactions among datapath variables for each state of the controller are captured. These tables were used to find a valid test path in the datapath.

The aforementioned works of the first category primarily examine the existing datapath and controller of a design to find test justification and propagation paths and to translate the locally generated test vectors at module (of the datapath) boundaries to global design tests. In addition, the primary objectives of these works are to ease test generation and reduce test generation time. Although, the test generation time is an important factor that has impact on the time-to-market of a design, test application time is more determinative in the total test cost (i.e. consumed power, ATE cost, time-to-market). Adding a low area overhead to the datapath and the controller, our work finds the test paths in the datapath such that the total test application time will be reduced.

Ravi *et al.* [10] used datapath of a design as a search space to find the best test paths. They proposed a computational extensive algorithm to find a test path in the datapath that could be activated by the controller. Ghosh *et al.* [11] presented techniques that added test hardware to a given RTL circuit obtained by behavioural synthesis in order to ensure that the embedded elements in the circuit were hierarchically testable. They generated a system-level test set to deliver pre-computed test sets to each element in the RTL circuit. Wada *et al.* [12] introduced the strong testability and proposed a DFT method based on the strong testability for RTL datapaths to achieve 100% fault coverage. This was accomplished by introducing additional functionality (hardware) to the modules in a datapath, thus increasing the testability of the circuit under test (CUT). The additional hardwares were called *mask element* and *thru function*. This DFT method is based on hierarchical test generation and the use of a combinational ATPG tool. Nicolici and Al-Hashimi [2] proposed a BIST methodology for RTL datapaths. This BIST methodology takes advantage of the structural information of an RTL datapath and reduces the test application time by grouping same-type modules into test compatibility classes.

These works of the second category primarily examine the datapath of a design and modify it to increase the testability of the design, whereas our proposed methodology considers both datapath and controller to increase the testability of the design and also proposes a hierarchical test application scheme. The use of controller and datapath together results in more reduction in the test application time and in the area overhead of the inserted DFT. In addition, because we use the original CDFG of the design, the complexity of our method is much lower than those works.

Inoue *et al.* [14] proposed a test synthesis method for datapaths. This method generates control sequences for justification and propagation at the RTL while assuming that for each module in the datapath, there are paths P_1, $P_2, \ldots, P_k$ from the primary inputs to the data input ports $i_1, i_2, \ldots, i_k$ such that any pair of P_i, P_j $(i \neq j)$ is disjoint or there is a register appearing in only one of these paths. Dey *et al.* [13] studied the testability of the controller and the datapath. Initially, the controller–datapath circuit is synthesised using behavioural test synthesis tools such that all loops in the datapath, except self-loops, can be broken with a minimal number of scan FFs and sequential depth. Then, the controller is modified by adding some control vectors. This technique tries to reduce the area overhead by redesigning the controller; in this case, the conflicts that may arise during the test pattern generation process are avoided. While this work breaks the loops in the datapath, our method handles the loops translating a test vector of the module in the loop into two different test vectors. This way, the area overhead due to the DFT will be decreased. Flottes *et al.* [15] proposed controller modifications for providing the maximum testability for the datapath of a design. Those modifications concern the next stage logic as well as the decoder part of the controller. This method is based on the results of testability analysis of the datapath at RTL and on finite automata theory. Ravi *et al.* [16] proposed a controller resynthesis technique to enhance the testability of RTL controller/datapath circuits. Their technique exploits the fact that the control signals in an RTL implementation are don't care under certain stated conditions. They use this don't care information in the controller specification to improve the overall testability.

These works of category three consider both controller and datapath of a design to propose hierarchical test schemes for the design. The benefits of our algorithm, which is of category three, with respect to the works mentioned earlier are 2-fold: (1) because we use CDFG, we can easily find optimum paths using different well-known shortest path algorithms in the literature; (2) using the CDFG, our methodology can optimally modify the controller to realise extra paths in the datapath.

The benefits of our method can be summarised as follows. It reduces the test application time as well as the area overhead due to the inserted DFT. In addition, because the proposed scheme uses the original CDFG, it can be merged with a synthesis methodology to decrease the whole design time of the circuit.

2 DFT procedure

We begin our DFT procedure using the CDFG and its corresponding synthesised RTL description of a circuit. We assume pre-computed tests are available for the individual datapath modules of this circuit. The steps for this procedure are outlined in what follows.

1. We consider CDFG of CUT as a framework for considering RTL module faults and tests.
2. We apply some implication rules to operators in a CDFG to allow passing test vectors and fault effects to a module under test (MUT). These rules may require adding hardware to facilitate testing of an MUT by giving pseudo-transparency to others.
3. Select an operation of the CDFG to represent the RTL module being tested (i.e. MUT). This is called the *victim*

operator. If there is more than one representative operator for an MUT, select one with the minimum test time.
4. On the basis of the selected victim and the pseudo-transparencies available through other operators, trim down the CDFG to obtain a *pruned-CDFG*. Applying a scheduling algorithm on the pruned-CDFG, a *test-CDFG* with a proper order of activation of operators is obtained that can be used for hierarchical test generation and test application. The portion of controller that can be used during the test can be extracted from a test-CDFG.
5. With the pruned-CDFG and the given victim (representative operator of MUT), modify controller of CUT to propagate pre-computed tests to MUT and propagate its response.

The mechanisms for performing these steps will be discussed in the sections that follow.

CDFG of a CUT contains datapath and controller information. The controller identifies paths of flow of data in the datapath. Other paths in the datapath that are not utilised can be added by altering the controller. The controller can be modified to ease controllability and observability of an MUT.

For passing test vectors through the CDFG operators and consequently datapath modules, we propose several implication rules (lemmas) for various types of CDFG operators. These rules generate pseudo-transparency for operators that are used for facilitating testing of an MUT. Pseudo-transparencies will be used for justification and propagation of test vectors through the CDFG to and from an operator being tested. With the pseudo-transparencies identified, the part of a CDFG that relates to a specific MUT will be identified. Such partial CDFGs (pruned-CDFGs) can be used for applying pre-computed test vectors to specific RTL modules. For finding a pruned-CDFG, this work finds a corresponding operator for each RTL module in the CDFG. Because an RTL module may be responsible for multiple CDFG operations, only one operation is selected (called victim) to be faulty as a representative of the RTL module. The selected operator (best victim) in the CDFG has the least test application time among other operators that are mapped to the same RTL module. The pruned-CDFG is constructed based on this operator. To find the best victim, testability metrics are defined, which can be used for computing the test application time of each victim. Our procedure finds enough victims to cover every RTL module. For every victim, there will be a pruned-CDFG. With this information, control states are adjusted to realise the pruned-CDFG and cause propagation of data to and from the victim operator. The proposed method adds a very small amount of test hardware to a given RTL circuit (obtained by behavioural synthesis) to ensure that the embedded elements in the circuit are hierarchically testable. Since the added hardware is primarily in the controller of the circuit, it has a small overhead. Also, this approach reduces the test application time by reducing the number of clock cycles needed to transfer test data to the datapath modules.

3 CDFG in test applications

The behavioural description of a circuit is usually provided using a hardware description language like VHDL. This description is compiled into a CDFG [2], which is a directed graph with operation vertices, data variable arcs, conditions and loops. CDFG can be used for extracting control and data information during synthesis.

In the sequel, we use a simple example synthesised at the architectural level to demonstrate the proposed DFT insertion technique. The original CDFG of this example along with the datapath and controller generated by the synthesis process will be used for producing individual pruned-CDFGs for the circuit operators.

Example: Consider the CDFG of Fig. 1*a*. This CDFG has six multipliers, four adders and one subtractor (these are the operators). Assume that after applying scheduling and resource sharing, the scheduled and bound CDFG shown

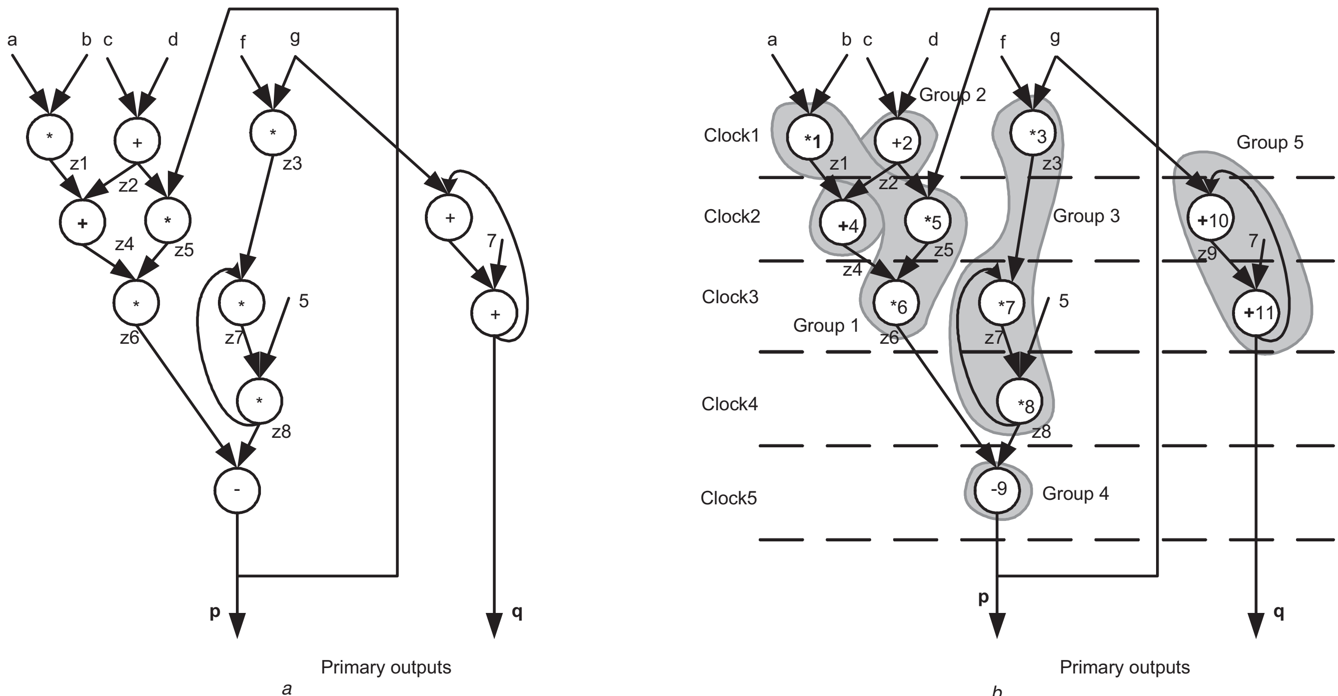


Fig. 1 *CDFG*

a CDFG of an example
b Scheduled and bound CDFG

in Fig. 1*b* is obtained. The horizontal lines in the figure represent the clock boundaries and the groupings (shaded areas) show group of operators that are mapped into the same module. Fig. 2*a* shows the datapath of this circuit, it has five modules that correspond to operations of Fig. 1*b*. The controller specifications are given in the state transition graphs of Fig. 2*b*. As shown in this example, the multiplier on the left of Fig. 2*a* performs the multiplication operations 1, 5 and 6 in Fig. 1*b*.

3.1 Paths in the CDFG and RTL

The role of a controller is to map the CDFG of a circuit to its datapath. Consequently, all paths in a CDFG exist in the datapath of a circuit. However, there may be additional paths in a datapath that do not necessarily exist in a CDFG that the datapath is generated from. This is because, the datapath hardware has shared registers that through their connecting busses have created new paths and data routing operations.

Some of these extra paths in a datapath can be activated for passing data by the existing controller of the circuit. We refer to these paths as *controller-supported* paths. Other paths of the datapath that are not specified by the CDFG and are not activated by the existing controller are referred as the *controller-unsupported* paths. To recognise and thus utilise these paths, changes may be needed in the circuit's controller. Examples of controller-supported paths are those from Operator 2 to Operators 4 and 5 as shown in Fig. 1*b*. An example of a controller-unsupported path is the direct path from group 4 to group 2. This path is controller unsupported because it exists in the datapath of Fig. 2, but not in the CDFG of Fig. 1*b*, which means that no control signals exist to activate this path.

Another issue of importance in the paths of a CDFG is the existence of loops. Two different types of loops are possible: a local-loop and a global-loop. A loop is a local-loop if it is within a single group of the operators in the CDFG; otherwise, it is a global-loop. The CDFG of Fig. 1*b* has two different loops: two local-loops ($*_7 \rightarrow *_8 \rightarrow *_7$) and ($+_{10} \rightarrow +_{11} \rightarrow +_{10}$), and a global-loop ($-_9 \rightarrow *_5 \rightarrow *_6 \rightarrow -_9$).

3.2 CDFG testability

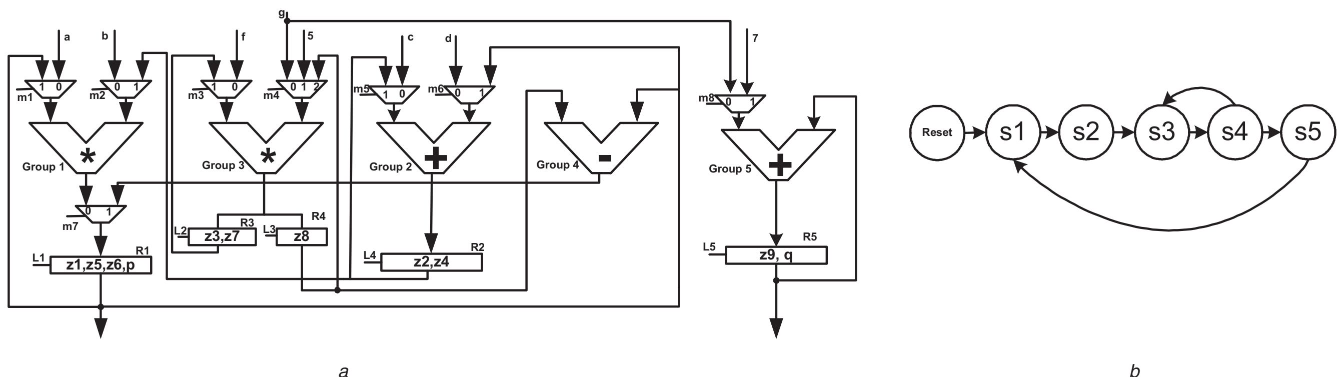
Consider a faulty module in the datapath of a circuit, and assume some test vectors for detecting faults in this module have already been generated. To activate these pre-computed test vectors, we have to find a path from the primary inputs of the circuit to the inputs of this module. So, it becomes possible to deliver pre-computed test vectors to the inputs of the module by applying the correct values at the primary inputs of the datapath. This also requires a path from the output of the faulty module to the primary outputs of the datapath to propagate the effects of the fault. The path from the primary inputs of the circuit to the faulty module and from this module to the primary outputs is referred to as a test-path. A test-path consists of two justifications (one for each input of the faulty module) and a propagation path. A test-path must be supported by the controller. The procedure for finding such paths on the datapath of a circuit is complicated. However, it can be significantly simplified (at no loss of correctness) by the use of the CDFG as a search space. Furthermore, this guarantees that the path found is supported by the controller. For faults of datapath modules, we consider their corresponding CDFG operators, and we assume that each operator in the CDFG may be faulty.

A test-path in a CDFG goes through several operators. For each such operator, the input not in the test-path is assumed to have a fixed known value. This assumption makes the test justification and propagation through the test-path feasible. Our manipulation of test data only affects operation inputs that are on the test-path. This is done by adding a few extra transitions and states to the controller, which cuts off all reconvergent fanouts and loops and generates the desired test-path.

For example, if the adder module of Group 2 in the datapath of Fig. 2*a* is faulty then Operator 2 of Fig. 1*b* will be faulty. In this case, the corresponding test-path is ($c \rightarrow +_2,\ d \rightarrow +_2,\ +_2 \rightarrow *_5 \rightarrow *_6 \rightarrow -_9 \rightarrow p$); where the first two ($c \rightarrow +_2$ and $d \rightarrow +_2$) are justification paths and the last one ($+_2 \rightarrow *_5 \rightarrow *_6 \rightarrow -_9 \rightarrow p$) is the propagation path. This test-pass goes through Operators 5, 6, 9. Left input of Operator $*_5$, right input of Operator $*_6$ and left input of Operator $-_9$ are not in the test path. Thus, we assume they have fixed values during testing of Operator $+_2$.

4 Accessing MUT

Operations in a CDFG are performed by datapath modules. For testing these modules other datapath modules have to allow tests to reach the MUT and to allow the responses to reach a primary output. For this purpose, many researchers propose various transparency properties [4, 5, 7] for RTL modules. Our method can use such schemes for finding test-paths for RTL modules, but we do not depend on a specific one. However, we propose several transparency rules for RTL modules that are more adaptable to our method.



a
b

Fig. 2 *CDFG*

a Datapath
b Controller

Rules for forward and backward implication of CDFG operators (and consequently, RTL modules) are determined by the lemmas that follow. These implications implement transparency for the operators of the CDFG in the test-path. We refer to such transparencies as pseudo-transparency. Using the lemmas for the operators in a test-path, the operator that the test-path is generated for will become testable. In contrast, these implications lead to determination of hierarchical tests in a test-path for pre-computed test vectors of the faulty operator. Note that, these pseudo-transparency rules are applicable to modulo-2^n operators. For other operators, such as floating point or fixed point operators, we can use the transparency rules in the literature or hold and thru functions and mask element concepts of the work of Wada *et al.* [12]. Fig. 3 shows examples of the lemmas that follow.

Lemma 1: If an add (subtract) operator has a specified value on one of its inputs, then its output can be set to any desired value by setting its other input to an appropriate value (this guarantees value justification along the justification path). Moreover, if an add (subtract) operator has a specified value on one of its inputs then different values on the other input generate different values at the output (this guarantees fault propagation along the propagation path) (Fig. 3*a*).

In other words, an n-bit binary adder provides a bijective function along the test path of other modules.

Lemma 2: If a multiply operator has a specified odd value on one of its inputs, then its output can be set to any desired value by setting its other input to an appropriate value (this guarantees value justification along the justification path). Moreover, if a multiply operator has a specified odd value on one of its inputs then different values on the other input generate different values at the output (this guarantees fault propagation along the propagation path) (Fig. 3*b*).

In other words, an n-bit binary multiplier provides a bijective function along the test-path of other modules if

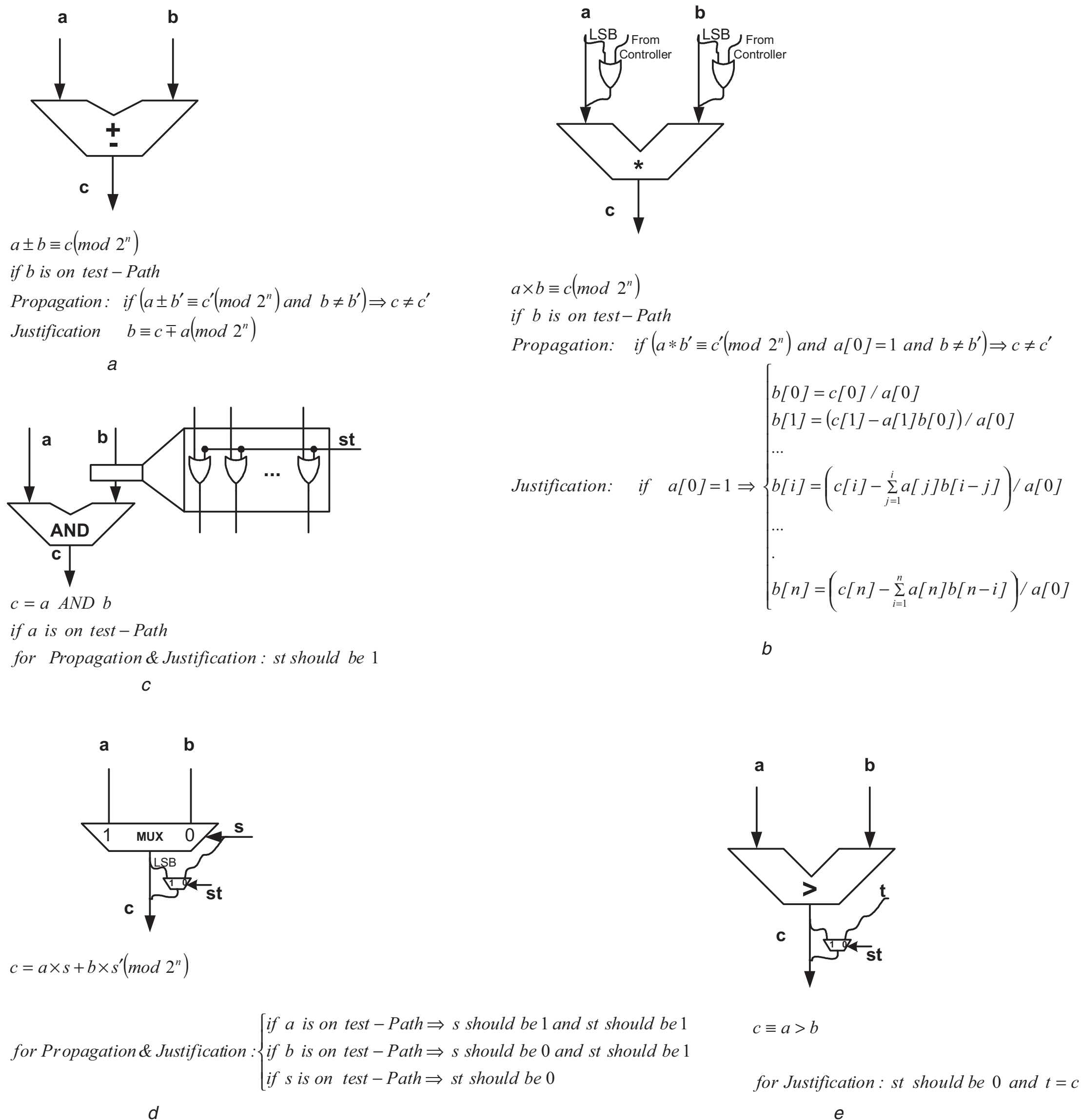


Fig. 3 *RTL module pseudo-transparency*

a Adder
b Multiplier
c Logical AND
d Multiplexer
e Comparator

the input that is not along the test-path has an odd value. To guarantee the existence of an odd value on the inputs of a multiplier, the LSB of each input of multiplier must be ORed with a control signal.

Lemma 3: If a multi-bit AND/OR operator has a string of 1/0 on one of its inputs, then its output can be set to any desired value by setting its other input to an appropriate value (this guarantees value justification along the justification path). Moreover, if a multi-bit AND/OR operator has a string of 1/0 on one of its inputs then different values on the other input generates different values at the output (this guarantees fault propagation along the propagation path).

If there are paths to AND/OR inputs to set these inputs to string of 1/0, Lemma 3 guarantees the existence of justification and propagation path. If such a path does not exist, adding a masking element [12] creates such a path (Fig. 3*c*).

Lemma 4: The output of a conditional operator (multiplexer) can be adjusted to any desired value by setting its select input and one of its data inputs to an appropriate value (this guarantees value justification along the justification path). Moreover, if a conditional operator has a specified value on one of its two inputs then different values on the other input generates different values at the output, given that the select input is set to an appropriate value (this guarantees fault propagation along the propagation path).

Equal values on both inputs of a multiplexer block the fault effect on its select input. Thus, an extra one-bit multiplexer has been added to solve the problem of propagating faults on the select signal of the original multiplexer. Adding this extra multiplexer makes the original multiplexer transparent, that is the existence of the propagation path from its select input to its output is guaranteed. Fig. 3*d* depicts the process of making the select input of multiplexer observable.

Lemma 5: Controlling the output of a relational operator (comparator) by a signal from the controller can satisfy the justification along the justification path. (Fig. 3*e*).

A comparator may not propagate fault effects from its inputs to its output. In this case, the propagation of fault effects on comparator inputs should be transferred to another path during test-path finding. If there is not such a path, adding an extra multiplexer at each input of the comparator creates this path.

We refer back to our example to illustrate some of the above lemmas. If the Operator 2 of Fig. 1*b* is the faulty operator and its test-path is ($c \rightarrow +_2$, $d \rightarrow +_2$, $+_2 \rightarrow +_4 \rightarrow *_6 \rightarrow -_9 \rightarrow p$) then applying Lemma 2 on Operators $+_4$ and $*_6$ lead to propagation fault effects of Operator 2 through the test-path. According to Lemma 1, Operator $-_9$ (a subtraction operator) can propagate the fault effects.

Note that only datapath modules that are along the justification or propagation path of other modules need the transparency property. As an example, consider the multiplier module *3 of Fig. 2. This module is not in the propagation and justification paths of other modules (Fig. 1*b*). Thus, adding DFT hardware to prepare the transparency for this module is not necessary.

5 Faulty operator in a CDFG

As mentioned earlier, all operators in a group of a binding of a CDFG are mapped to one module in the datapath. It is enough to select one operator of a group and use it as a representative faulty operator for a faulty module. Selecting one of the corresponding operators of a faulty module as the faulty operator avoids multiple CDFG faults. Such an operator is referred to as a victim operator. Once a victim is selected, other operators of the group are removed from the CDFG resulting in a pruned-CDFG.

5.1 Selecting a victim

A victim is an/a existing/new operator that receives its inputs from outside of its group, and delivers its output to the outside of its group. If the victim is a new operator, it is constructed by merging several of the existing operators while inherits the best controllability and observability of the merged operators. Victims can be classified into four different types on the basis of their group structures.

Type 1: A victim of Type 1 (Fig. 4*a*) is an existing operator which receives both its inputs from outside of the group and delivers its output to the outside of the group through existing/controller-supported paths in the CDFG. Hence, controllability and observability of the corresponding RTL module are supported by the original CDFG.

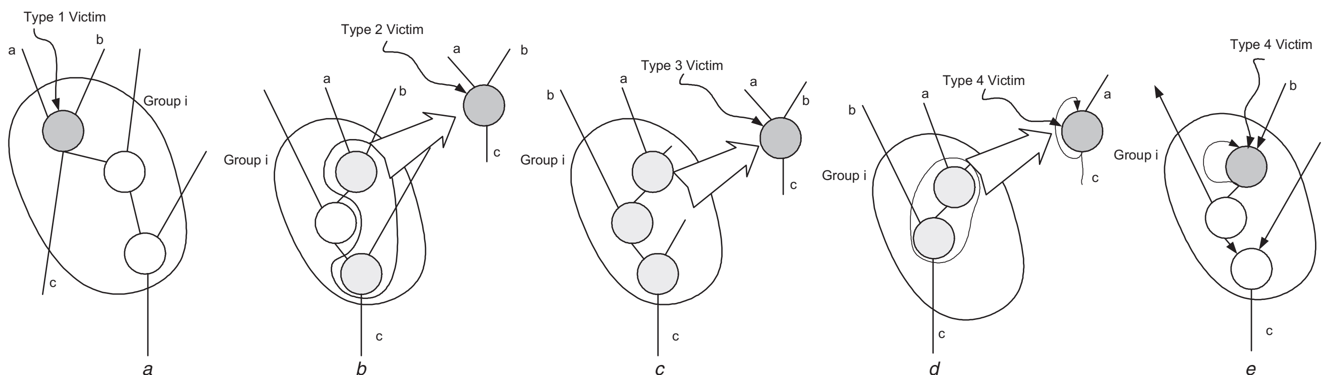


Fig. 4 *Different types of victim*

a Type 1 victim
b Type 2 victim
c Type 3 victim
d Type 4 victim
e Type 4 victim

Type 2: A victim of Type 2 (Fig. 4*b*) is a new operator constructed by merging an existing operator which receives both its inputs from outside of the group (called input-provider), and an existing operator which delivers its output to the outside of the group (called output-generator). These two merging operators are called complementary operators. This victim is realised by adding an extra transition/state to the controller to cause flow of data from inputs of the input provider to the output of the output provider.
Type 3: A victim of Type 3 (Fig. 4*c*) is a new operator constructed by merging two existing operators that receive one of their inputs from outside of the group (input-providers), and an existing operator that delivers its output to the outside of the group (output-generator). These merging operators are complementary operators. This victim is realised by adding an extra transition or state to the controller to cause flow of data from inputs of the input providers to the output of the output provider.
Type 4: The existence of two consecutive operators (Fig. 4*d*) or a local-loop (Fig. 4*e*) in the group confirms the existence of a self-loop path around the corresponding RTL module. Thus, the RTL module can be fed by one input from the outside of the related group. Since in this scheme, the feedback input of the RTL module receives its value from the other input through the RTL module, we should verify the correctness of this value by propagating it to an observable point. For this purpose, some controller-unsupported/controller-supported paths of the design need to be utilised. Thus, in the case of two consecutive operators, a victim of Type 4 is an operator constructed by merging these two consecutive operators. In the case that a self-loop exists in the group, the victim is the operator with self-loop.

Note that, pre-computed test vectors should be modified for Type 4 victims. In such cases, before applying the pre-computed test vector to an MUT, an extra test vector should be added to the pre-computed test set. For example, Fig. 5 shows the corresponding RTL module of a victim of Type 4. One of these multiply RTL module inputs is *g* and its other input is fed from its output. To apply a pre-computed test vector to this module, first, the feedback input must be set to a desired value by setting the controllable input to an appropriate value based on the current value of the feedback register, and next, the non-feedback input must be set to its corresponding value in the pre-computed test vector. For example, assuming test vector (3, 5) should be applied to the multiplier inputs of Fig. 5*a*. Assume that the current value of the feedback register is 1, to set this register to 5, the controllable input of the multiplier (input *g*) must be

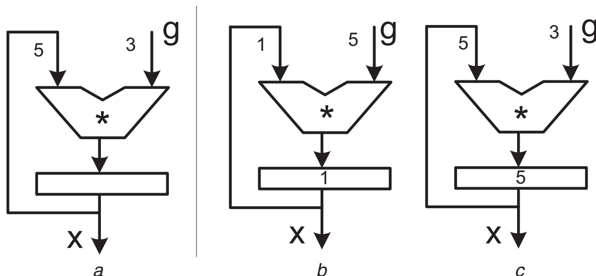


Fig. 5 *Applying a desired test vector to a self-loop*
a Multiplier inputs
b Multiplier module
c Test application

set to 5 ($5/1 = 5$). After applying values 1 and 5 to the multiplier module (Fig. 5*b*), the multiplier output must be propagated to a primary output to verify the correct functionality of the module. In the next step, applying the value 3 to the controllable input, the multiplier module can be tested with test vector (3, 5). Fig. 5*c* shows the last step of test application. In Fig. 5, test vector (5, 1) is added to precede test vector (3, 5) in the pre-computed test set.

Extending the above idea, global loops can be handled in the CDFG. If a global loop exists around a victim, a test vector can be applied to the victim in two steps and justifying twice, guarantees the fault detection.

6 Finding test-CDFGs

Extracting a minimal test-CDFG for a given group from the CDFG of a CUT is elaborated in this section. For this purpose, traditional controllability and observability metrics are assigned to input and output nodes of a group of operators, which are defined based on the number of clocks along the shortest path of the nodes to primary input and primary outputs, respectively. In addition, the minimum controllability of group inputs and minimum observability of group outputs are considered as group controllability and group observability, respectively. After deciding on an MUT, three algorithms implement the methodology of finding the test-CDFG: *FindVictim( )*, *PruneCDFG( )* and *ScheduleCDFG( )*.

- *Find Victim:* The victim of a group is found by the *FindVictim( )* algorithm. In this *FindVictim( )* algorithm, all inputs and outputs of a group are examined and inputs with the best controllability and the output with the best observability are selected to form the best victim. Note that, using the group controllability and observability, this function computes the number of clocks needed for testing the group. This algorithm also constructs a new victim in a CDFG by adding a new state or new transitions to the controller. Additionally, it detects the existence of global feedbacks around the victim which is handled in the *PruneCDFG( )* algorithm.
- *Prune CDFG:* A pruned-CDFG that contains the test-path of the victim will be constructed by the *PruneCDFG( )* algorithm. The *PruneCDFG( )* algorithm marks test-paths for a group for which a victim is found by *FindVictim( )*. The justification and propagation paths of the test-path are found by running the shortest path algorithm on the CDFG. After finding a test-path, all unnecessary clocks in the CDFG are eliminated. This algorithm also cuts all reconvergent fanouts and feedbacks in the test-path by adding states or transitions to the controller.
- *Scheduling a CDFG:* The *ScheduleCDFG( )* algorithm determines the sequence of controller states to apply a test vector to a pruned-CDFG. The final reduced CDFG with the scheduling information is called test-CDFG.

Using two examples, the next section illustrates the victim concept and the finding test-path methodology.

7 Examples

In what follows, we apply the previous methodology to two designs. The first design is the example of Fig. 1*b* and the second design is a bigger design called fifth-order wavelet filter [18].

7.1 Example of Fig. 1

In the CDFG of Fig. 1*b*, Group 1 consists of three multiply operators whose outputs are bound to a single register. The first step is to find a victim of this group. For this purpose, we consider *a* and *b* as inputs of the victim because they have the best controllability of all other group member inputs and are assigned to the opposite sides of the corresponding multiplier of Fig. 2*a*. In contrast, $z1$, $z5$ and $z6$ are outputs with best observability (because they share a register with primary output *p*, they can be interpreted as primary outputs during test). In this case, $z1$ is selected as the output of the victim of this group. Therefore Operator 1 is chosen as the victim of Type 1 for this group. The corresponding pruned-CDFG is shown in Fig. 6*a*. This test-CDFG adds a transition to the controller (*t*1 of Fig. 6*f*) for applying continually test vectors to the faulty module (the chosen victim).

Group 2 of Fig. 1*b* consists of two add operators whose outputs have been bound to a single register. *c* and *d* are inputs of this group with best controllability, which are assigned to opposite sides of the corresponding adder of Fig. 2*a*. Because two outputs $z2$ and $z4$ of Group 2 drive two signals $z6$ and $z5$ through a level of multiplier and two signals $z6$ and $z5$ are assigned to the same register, two outputs $z2$ and $z4$ have the same observabity. Therefore Adder 2 becomes the victim of this group. The corresponding test-CDFG is shown in Fig. 6*b*. This test-CDFG is realised by adding transition *t*2 to the controller as shown in Fig. 6*f*. Addition of this edge causes Adder 2 to operate in state *s*1, the result to be used by Multiplier 5 in *s*2, and the result of multiplication to become observable on *p* in *s*2.

The test-CDFG in Fig. 6*c* corresponds to Group 3 of Fig. 1*b*. The extra state *ts*1 and transitions *t*3, *t*4 and *t*5 realise this test-CDFG. Because the single operator of Group 4 exists in the test-CDFG of Group 3, this CDFG can also be used as the test-CDFG of Group 4 (Fig. 6*f*).

Finally in Fig. 1*b*, Group 5 consists of two add operators whose outputs are bound to a single register. *g* is the only input to this group. The victim of this group is of type 4. Fig. 6*e* shows the corresponding pruned-CDFG for this group. The extra transition *t*6 of Fig. 6*f* realises this test-CDFG.

Fig. 6*f* shows the modified controller. In this figure, the extra state and transitions for test purposes are shown by the dotted lines. Furthermore, an input signal in the design selects between paths of the original controller and paths of the test controller.

7.2 Elliptical wave filter

The rest of this section deals with applying the proposed algorithm on a fifth-order elliptical wave filter (fifth-order EWF) [18] whose scheduled and bound CDFG is shown in Fig. 7*a*. This CDFG consists of 25 add and 8 multiply operations. Constant inputs of operators are not shown in this figure. The corresponding datapath has three adders (*A*1, *A*2, and *A*3) and three multipliers (*M*1, *M*2 and *M*3). The latencies of adders and multipliers are one and two clock cycles, respectively. An uppercase label on the left side of each operation in the CDFG represents the corresponding bound module in the datapath.

In this CDFG, the corresponding registers are shown in lowercase letters near the signal. To identify the left and right inputs of the related RTL module of each operation in this CDFG, the ‘L’ and ‘R’ labels are used.

Let us consider the group of operators that are bound to adder *A*1 of the RTL datapath. This group consists of operators 13, 19, 22, 25, 26, 27 and 30. Signals *r*0, *r*3 and *r*4 are assigned to the left input of the adder, and signals *r*1, *r*6 and *r*9 are assigned to the right input of the adder. Because the group controllability of operators bound to *A*3 is 0 (i.e. a primary input directly feeds module *A*3), the controllability of *r*3 and *r*6 are 2 and 1, respectively, which have the best controllability on the opposite sides of adder *A*1. Adder *A*1 drives two signals r_0 and *r*1. The observability of the signal *r*0 is 3, which is the best observability for the output of this RTL module (i.e. *A*1). Using signals *r*3, *r*6, as the victim inputs and *r*0 as the victim output, FindVictim( ) function constructs victim of Type 3 that is shown Fig. 7*b* and referred to as Operator 25*. The next step deals with finding the test-path for this victim in the CDFG using the pruneCDFG( ) function. Bold lines of Fig. 7*b* show this test-path. Finally, applying the scheduleCDFG( ) algorithm, the sequences of controller states during which a test is applied to the module-under-test are obtained. The corresponding test-CDFG is shown in Fig. 8. Fig. 9 shows the corresponding modified controller. The added transitions and state are shown with dotted lines.

As shown in Fig. 8, the test-CDFG for module *A*1 has seven states, only one of which new. The other states are the original states of the controller of the fifth-order EWF. Because the victim is of Type 3 and consequently a new operator is needed, an extra state should realise it. Therefore State 11* (shadowed state of Fig. 8 and dotted state of Fig. 9) identifies the victim in the test-CDFG. As shown in Fig. 8, activating the test path of module *A*1 requires six extra transitions to the controller: from reset

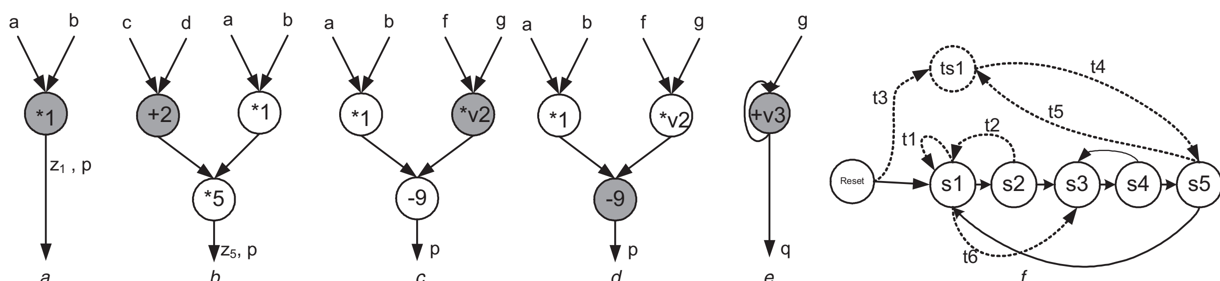


Fig. 6 *Test-CDFGs and modified controller of Fig. 1*

a Type 1 victim
b Type 2 victim
c Type 2 victim
d Type 1 victim
e Type 4 victim
f Modified controller

Fig. 7 *Fifth-order EWF*

a CDFG of the fifth-order EWF
b Test path of module A1

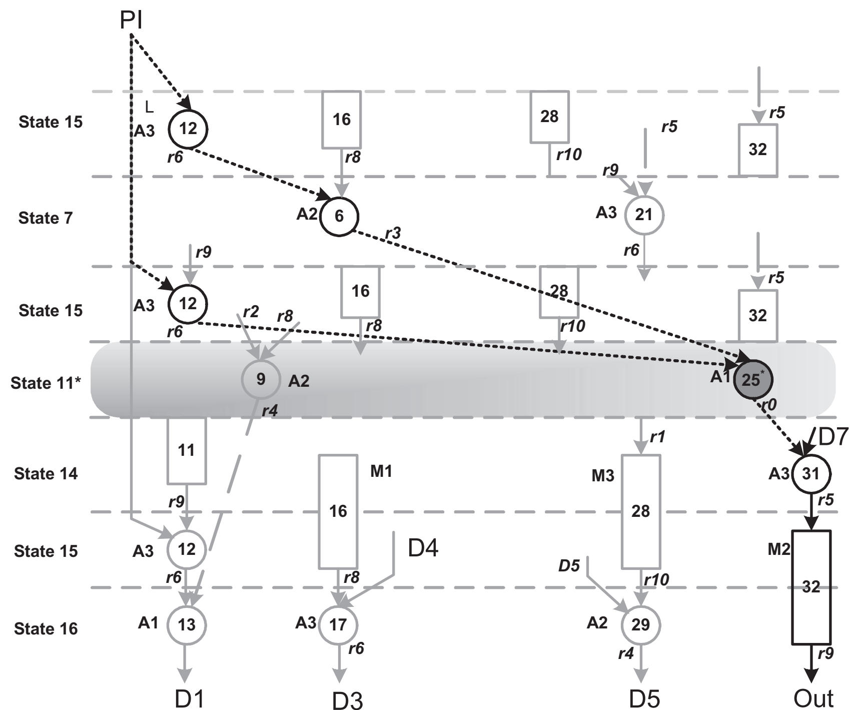


Fig. 8 *Test-CDFG*

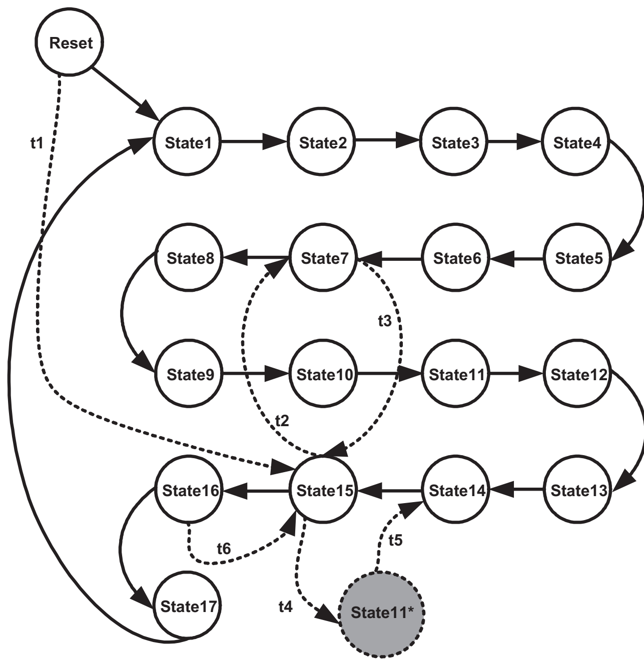


Fig. 9 *Fifth-order EWF controller with test path of module A1*

to State 15 (*t*1 of Fig. 9), from State 15 to State 7 (*t*2 of Fig. 9), from State 7 to State 15 (*t*3 of Fig. 9), from State 15 to State 11* (*t*4 of Fig. 9), from State 11* to State 14 (*t*5 of Fig. 9) and from State 16 to State 15 (*t*6 of Fig. 9). The last transition is for applying the next test vector to the MUT.

The test-CDFG of other groups can be obtained similarly. The experimental results show that the test application time of the proposed method is 1586 clock cycles, that is, about 34 times faster than the full scan. Also, the area overhead of this method is 1.4%, whereas the area overhead of full-scan is 9.74%.

8 Experimental results

In this section, we present experimental results obtained by applying our algorithm to 11 circuits. The proposed algorithm accepts a scheduled and bound CDFG of a design and its RTL circuit in the form of separated controller and datapath. First, the algorithm is performed on the scheduled and bound CDFG to find the victim of each group. Then, test path of that victim is found. Finally, the corresponding test-CDFGs of each victim is constructed and the controller is modified to activate these test-CDFGs.

We have applied the lemmas of Section 4 to 11 benchmark circuits and synthesised them by a commercial synthesis tool with a library consisting of AND, NAND, OR, NOT and flip-flop cells. Table 1 shows the number of inputs, outputs, gates and flip-flops of these 11 32-bit benchmarks. Area overhead results of applying the lemmas of Section 4 are also shown. These overheads are estimated by the computation of the transistor count in the circuits. For this purpose, we have considered the standard CMOS technology that uses four transistors for NAND and NOR gates, and uses six transistors for AND and OR gates. An inverter uses two transistors, and a flip-flop is implemented using eight transistors (i.e. C^2MOS flip-flop). Note that, the area overhead due to the use of transparency schemes in the literature other than those of lemmas of Section 4 should be added to these overheads.

In the following, we compare our method with other test methods in the literature. In these comparisons, the test application time (TApp) is the number of clocks that are needed for applying test vectors to CUT. Fault coverage is computed as the ratio of detected faults to total single stuck-at faults in the CUT. In addition, area overhead is estimated by the number of transistors in the circuits using the standard CMOS technology.

Table 2 shows the results of applying proposed algorithm to these 32-bit benchmarks and their full-scan version. For the proposed high-level DFT insertion algorithm, fault coverage was obtained on the basis of the pre-computed test vectors of modules in the datapath of the circuit and the controller test vectors which were generated by applying HITEC to the gate-level description of the design. System-level vectors are given by the concatenation of the datapath and controller test vectors. The full-scan version of these benchmarks is obtained by OPUS [19] (a public domain partial scan package). Then, ATALANTA [20] generates their test vectors. We assume that the scan chain is constructed by using a multiplexer at the input of each flip-flop of the circuit. In addition, the multiplexer is implemented by 14 transistors.

Fault coverage results obtained by applying our proposed algorithm are very close to those obtained for full-scan testing of the benchmarks. As shown, our method of DFT significantly reduces test application times. Furthermore, the proposed algorithm requires less area overhead than full-scan.

We also examined the efficiency of the proposed DFT method using gate-level test generation tools. HOPE [21]

Table 1: Circuits statistics and their area overhead of lemmas in Section 4

Circuit	Number of inputs	Number of outputs	Number of AND	Number of NAND	Number of OR	Number of NOT	Number of FF	Over, %
GCD	66	33	177	983	38	81	105	0.23
Tseng	98	64	2281	604	238	50	203	0.37
Paulin	66	64	4112	10 153	77	21	236	0.31
Barcode	37	66	292	765	293	98	177	0.45
Simple RTL	36	32	1997	4890	129	15	103	0.34
Chain_mult	34	32	2024	4829	101	11	103	0.29
4 Point DCT	130	128	2762	7144	17	33	269	0.35
Third-order IIR	34	32	4109	9810	17	30	207	0.25
Sixth-order FIR	34	32	2916	6688	99	23	297	0.38
6 Tap wavelet filter	34	64	2458	6089	17	43	371	0.47
Fifth-order elliptic filter	34	32	5238	12 681	189	92	615	0.43

Table 2: Comparison of our method's results with full-scan

	Our method			Full scan		
	TApp	FC, %	Over, %	TApp	FC, %	Over, %
Tseng	215	99.98	2.3	41 100	100	14.8
Simple RTL	412	99.99	2.8	17 526	100	4.34
Paulin	478	99.99	1.4	34 615	99.99	4.88
GCD	263	99.97	3.6	25 840	98.97	23.6
Chain_mult	354	99.99	2.9	14 824	99.99	4.38
Barcode	215	99.10	14.57	62 451	100	30.29
Third-order IIR	758	99.99	0.95	165 840	99.99	4.41
Sixth-order FIR	628	86.41	0.54	145 747	87.98	8.80
6 Tap wavelet filter	316	88.69	0.49	61 004	90.50	12.29
4 Point DCT	396	86.40	0.54	56 516	86.60	7.93
Fifth-order elliptic filter	1586	99.48	1.4	53 784	99.90	9.74

Table 3: Gate-level test generation results

	Hope		HITEC			
	Original	Modified	Original		Modified	
	FC, %	FC, %	FC, %	Test time	FC, %	Test time
Barcode	34.10	95.47	31.01	41	98.42	54
Paulin	99.12	99.54	89.62	122	99.98	131
Chain_mult	49.14	86.36	85.33	117	99.99	115
Tseng	77.28	98.24	78.48	182	99.97	174
Simple RTL	97.68	99.34	98.20	359	99.83	322

is a sequential gate-level fault simulator. After applying 10 000 random test vectors, fault coverage for the individual circuits were computed. HITEC [22], which is a sequential gate-level test generator, was also used for evaluation of the proposed method. Table 3 shows the obtained results. Columns 2 and 3 show fault coverages obtained by running HOPE on the original circuits and modified circuits by our scheme, respectively. Columns 4 and 6, 5 and 7 show the fault coverage and test application time obtained by running HITEC on the original and modified circuits, respectively. Fault coverage for modified circuits is higher

Table 5: Comparison with TAO

	TAO			Our method		
	FC, %	TApp	Over, %	FC, %	TApp	Over, %
Paulin	99.5	507	1.8	99.98	251	1.7
GCD	99.1	278	6.4	99.81	145	4.3
Chain_mult	99.2	381	1.4	99.97	194	3.2
Simple RTL	99.9	422	3.5	100	312	3.3

in all cases, whereas the test time has been decreased. While the proposed method targets testability improvement of datapath only, the results reported in Table 3 for the faults in the controller show that it significantly improves the testability of the whole circuit.

Table 4 compares the fault coverage, area overhead and test application time of the proposed methods against the three previously proposed methods. In this table, faults of the 16-bit circuits of the benchmarks are considered and the fault coverage shows the percentage of the detected faults in the datapath. Scan_dec uses scan chains in the controller outputs while scan_sec uses scan chains on the next-state flip-flops of the controller [15]. Flottes *et al.* [15] have proposed the controller re-synthesis for improving datapath testability whose results are also included in this table. The proposed technique compares very favourably with other techniques with the advantage of a reduced test time and lower area overhead. This occurs because these approaches consider only the controller for increasing testability while our proposed approach considers modifications both in controller and datapath. Furthermore, the added hardware is primarily in the controller of the circuit.

Table 5 compares the proposed algorithm with TAO [10]. In this table, the bit-width of all circuit is 16. TAO increases the testability of an RTL circuit by exploring the datapath and adding hardware to the datapath and modifying the controller. As mentioned previously, our RTL modification to increase the testability is based on the controller and then the datapath (if necessary). So our approach incurs in a smaller area overhead. In addition, in the modification of the controller, unnecessary clocks during the test application are eliminated (Table 5 shows the test application time).

9 Conclusions

This paper has presented a novel DFT method that requires a small modification to the controller in the RTL description of a circuit. The CDFG representation of an RTL circuit is used for analysing the testability of individual RTL operations within a hierarchical test technique. Using a non-scan arrangement, existing data paths are utilised to provide controllability and observability to RTL operations.

Table 4: Comparison with three other strategies

	Scan_dec			Scan_seq			Controller resynthesis			Our method		
	FC, %	Over, %	Test time	FC, %	Over, %	Test time	FC, %	Over, %	Test time	FC, %	Over, %	Test time
Tseng	100	12.8	2440	96.37	0.5	10.91	98.11	14.2	803	99.98	3.1	125
Diffeq	100	14.3	4305	95.78	0.5	802	96.25	12.4	1683	99.98	2.4	123
Simple RTL	99.88	1.8	3935	97.03	0.5	5270	98.98	1.5	2366	100	3.3	312

Furthermore, additional data paths are introduced by altering the controller states or adding new transitions. Experimental results show that this method considerably reduces the test application time by ignoring unnecessary control states in the test process.

10 References

1 Ohtake, S., Wada, H., Masuzawa, T., and Fujiwara, H.: 'A non-scan DFT method at register-transfer level to achieve complete fault efficiency'. Proc. ASP-DAC, 2000, pp. 599–604

2 Nicolici, N., Al-Hashimi, B., Brown, A.D., and Williams, A.C.: 'BIST hardware synthesis for RTL data path based on test compatibility classes', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2000, **19**, (11), pp. 1375–1385

3 Dey, S., and Potkonjak, M.: 'Nonscan design-for-testability techniques using RT-level design information', *IEEE Trans. Comput.-Aided Des.*, 1997, **16**, pp. 1488–1506

4 Makris, Y., and Orailoglu, A.: 'DFT guidance through RTL test justification and propagation analysis'. Proc. Int. Test Conf. (ITC'98), 1998, pp. 668–677

5 Makris, Y., and Orailoglu, A.: 'RTL test justification and propagation analysis for modular designs', *J. Electron. Test. Theory Appl. (JETA)*, 1999, pp. 105–120

6 Makris, Y., Patel, V., and Orailoglu, A.: 'Efficient transparency extraction and utilization in hierarchical test'. Proc. 19th VLSI Test Symp. (VTS'01), 2001, pp. 246–251

7 Makris, Y., and Orailoglu, A.: 'Test requirement analysis for low cost hierarchical test path construction'. Proc. 11th Asian Test Symp. (ATS'02), 2002, pp. 134–139

8 Makris, Y., Collins, J., and Orailoglu, A.: 'Fast hierarchical test path construction for DFT-free controller-datapath circuits'. Proc. 9th Asian Test Symp. (ATS'00), 2000, pp. 185–190

9 Makris, Y., Collins, J., and Orailoglu, A.: 'Fast hierarchical test path construction for circuits with DFT-free controller-datapath interfaces', *J. Electron. Test., Theory Appl. (JETA)*, 2002, pp. 29–42

10 Ravi, S., Lakshminarayana, G., and Jha, N.K.: 'TAO: regular expression based high-level testability analysis and optimization'. Int. Test Conf., 1998, pp. 331–340

11 Ghosh, I., Raghunathan, A., and Jha, N.K.: 'Design for hierarchical testability of RTL circuits obtained by behavioral synthesis', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1997, **16**, (9), pp. 1001–1014

12 Wada, H., Masuzawa, T., Saluja, K.K., and Fujiwara, H.: 'Design for strong testability of RTL data paths to provide complete fault efficiency'. Proc. 13th Int. Conf. on VLSI Design, 2000, pp. 300–305

13 Dey, S., Gangaram, V., and Potkonjak, M.: 'A controller redesign technique to enhance testability of controller-data path circuits', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1998, **17**, (2), pp. 157–168

14 Inoue, M., Suzuki, K., Okamoto, H., and Fujiwara, H.: 'Test synthesis for datapaths using datapath-controller functions'. Proc. 12th Asian Test Symp. (ATS'03), 2003, pp. 294–299

15 Flottes, M.L., Rouzeyre, B., and Vople, L.: 'A controller resynthesis based method for improving datapath testability'. Proc. Int. Symp. on Circuits and Systems, 2000, pp. 346–350

16 Ravi, S., Ghosh, I., Roy, R.K., and Dey, S.: 'Controller resynthesis for testability enhancement'. Proc. 11th Int. Conf. on VLSI Design, 1998, pp. 193–198

17 De Micheli, G.: 'Synthesis and optimization of digital circuits' (McGraw-Hill Inc., 1994)

18 Potkonjak, M., Dey, S., and Wong, J.L.: 'Optimizing designs using the addition of deflection operations'. Available at http://trix.cs.ucla.edu/jenni/papers/HotPot_TR.pdf

19 Chickermane, V., and Patel, J.H.: 'A fault oriented partial scan design approach'. Proc. Int. Conf. Computer-Aided Design, 1991, pp. 400–403

20 Lee, H., and Ha, D.: 'On the generation of test patterns for combinational circuits'. Technical Report 12-93, 1993, Department of Electrical Engineering, Virginia Polytechnic Institute and State University

21 Lee, H.K., and Ha, D.S.: 'HOPE: an efficient parallel fault simulator'. Proc. Design Automation Conf., 1992, pp. 336–340

22 Niermann, T.M., and Patel, J.H.: 'HITEC: A test generation package for sequential circuits'. Proc. European Design Automation Conf., 1991, pp. 214–218