Low Test Application Time Resource Binding for Behavioral Synthesis

MOHAMMAD HOSSEINABADY, PEJMAN LOTFI-KAMRAN,
and ZAINALABEDIN NAVABI
University of Tehran

Recent advances in process technology have led to a rapid increase in the density of integrated circuits (ICs). Increased density and the need to test for new types of defects in nanometer technologies have resulted in a tremendous increase in test application time (TAT). This article presents a test synthesis method to reduce test application time for testing the datapath of a design. The test application time is reduced by applying a test-time-aware resource sharing algorithm on a scheduled control data flow graph (CDFG) of a design.

Categories and Subject Descriptors: B.5.3 [**Register-Transfer-Level Implementation**]: Reliability and Testing—*Testability; test generation*; B.5.1 [**Register-Transfer-Level Implementation**]: Design—*Data-path design*

General Terms: Design, Reliability

Additional Key Words and Phrases: Testability, test synthesis, CDFG, high-level synthesis

ACM Reference Format:

Hosseinabady, M., Lotfi-Kamran, P., and Navabi, Z. 2007. Low test application time resource binding for behavioral synthesis. ACM Trans. Des. Autom. Electron. Syst. 12, 2, Article 16 (April 2007), 22 pages. DOI = 10.1145/1230800.1230808 http://10.1145/1230800.1230808.

1. INTRODUCTION

Recent advances in process technology have led to a rapid increase in the density of integrated circuits (ICs). This increased density and the need to test for new types of defects in nanometer technologies have resulted in a tremendous increase in *test generation time* (TGT) and *test application time* (TAT).

To ease the complexity of test generation, design-for-testability (DFT) techniques have been proposed. The most commonly used DFT techniques for VLSI

This research was supported by the Nanotechnology-Center of Excellence, University of Tehran, Iran.

Authors' addresses: M. Hosseinabady (contact author), P. Lotfi-Kamran, and Z. Navabi, Electrical and Computer Engineering Faculty, University of Tehran, 14399 Tehran, Iran; email: mohammad@cad.ece.ut.ac.ir.

circuits – particularly applicable to sequential circuits – utilize a scan chain. These techniques rely on modifying a circuit such that automatic test pattern generation (ATPG) tools can achieve a high fault coverage. Scan-based techniques have the disadvantage that the test application time is very large compared to that of nonscan designs.

A top-down design methodology that has increasingly been adopted for synthesis offers an alternative approach to ease test generation and to reduce test application time. In addition, high-level definitions of a circuit (e.g., behavioral-level or register-transfer-level (RTL)) have significantly reduced the number of primitives (as a measure of complexity) compared to that of a gate-level representation. Consequently, if test generation and testability enhancement can be performed at RT- or behavioral level, the test problem may be more tractable. Considering test application time during high-level synthesis not only provides an edge for testability-related optimizations, but also enhances the scope of critical timing path analysis. Towards this direction, test application time analysis information constitutes a precious resource for innovative high-level synthesis solutions.

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

There are two parameters that affect test application time: *the number of test vectors* and *the number of clocks* that are needed to apply a test vector to the design. Note that increasing the testability (i.e., controllability and observability) of different parts of a design makes the design more easily testable, and consequently decreases the number of test vectors. On the other hand, decreasing the depth of a circuit during test decreases the number of clocks that is needed to apply a test vector to the design.

1.1 Previous Work

There are many works in the literature that propose methodologies for test synthesis and DFT methods to increase the testability of a design.

Makris et al. [2001] introduce a methodology for identifying the transparency behavior to be used in a hierarchical test scheme. Using the transparency scheme, Makris et al. [2002] propose a hierarchical test generation method for DFT-free controller-datapath design. In addition, introducing the concept of an influence table, they translate the controller of a design to several tables by which the interactions among datapath variables for each state of the controller are captured. These tables are used to find a valid test path in the datapath.

These works 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. These works result in reducing test generation time. Although test generation time is an important factor impacting on the time-to-market

of a design, test application time is more determinative of the total test cost (i.e., consumed power, ATE cost, time-to-market). Adding only a very low area overhead DFT hardware to the datapath and controller, our work will find the test paths in the datapath such that the total test application time will be reduced.

Bhatia and Jha [1994] propose a behavioral synthesis technique that guarantees high and easy testability of data paths, even in the presence of loop constructs in the behavioral description. Given a test set for each module, the aim of their behavioral synthesis system, called Genesis, is to make it possible to obtain a system-level test set with 100% test coverage of each embedded module. To examine the testability of a design, they define three testability measures, called controllability, observability, and verifiability, for different nodes in the DFG (data flow graph) of the design. Based on the obtained testability parameters, if it is not possible to identify a test environment for any of the operators mapped to a module, they add a multiplexer, map testable operators to the untestable module, and modify the register allocation or unroll the DFG to handle loops in the DFG. Whereas these techniques increase the testability of modules in an RTL design, they do not change the test environment for a testable module. Not only do we increase the testability of an RTL module in a design using a few low-overhead DFT techniques, but also we try to minimize the number of clocks needed to test a module, even that module which is testable in the original CDFG. For this purpose, adding extra transitions in the controller of the design, we bypass the extra clocks in justification and propagation paths during test.

Inoue et al. [2003] propose a test synthesis method for RT-level designs. In this scheme, to increase the testability of a design, the augmented RTL structure of the design is generated. An augmented RTL structure is a strongly testable datapath and controller structure capable of providing test plans for each module. Controller modification is performed by adding transitions, states, and primary inputs. For the datapath, a hold function is added to the registers, while a thru function is added to the modules. This method generates control sequences for justification and propagation at the register-transfer level using the existing RTL design. This reference considers only the increasing testability of a design and does not address the number of clocks needed to apply test vectors to the design. In addition, to realize the strong testability, they added multiplexers, mask elements (a few controllable gates), and registers to the existing datapath. The extra hardware may have a timing penalty on the normal operation of the circuit, whereas in large designs in which the added DFT can have a timing penalty and large area overhead, the conditions for test justification and propagation (called strong testability) can be achieved by altering resource binding and allocation in the early design stage (i.e., during the synthesis algorithm).

1.2 Contribution and Overview of the Proposed Method

This article addresses both of the mentioned parameters (i.e., the number of test vectors and number of test clocks) simultaneously. For increasing the testability

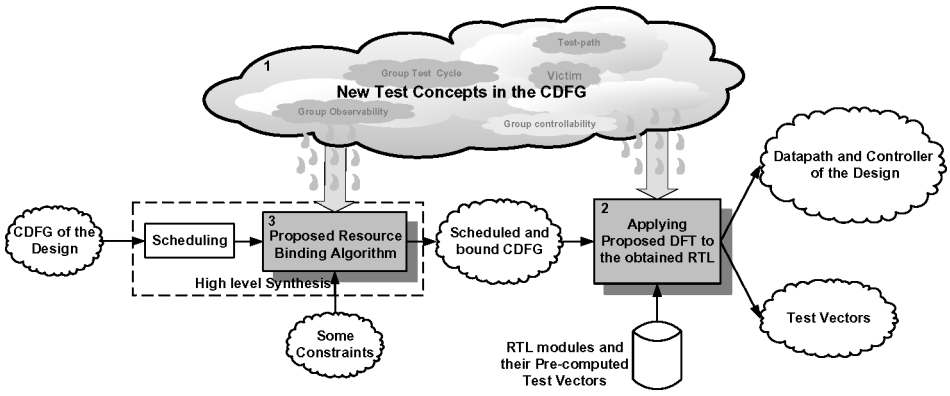


Fig. 1. Overview of our synthesis algorithm.

of modules in an RTL design, we first impose a transparency that takes advantage of the functionality of RTL modules, which is called *functional transparency*. Then, we change the controller so that each module can be more easily reachable. In addition, a heuristic resource binding algorithm as a part of the test synthesis method is proposed to decrease the number of clocks needed to test a module under test in the design.

Figure 1 shows different parts of the proposed methodology. Based on this figure, the proposed test synthesis methodology has three main portions: new high-level test concepts based on the CDFG of a design (label 1 of Figure 1), a DFT methodology at the RT level (label 2 of Figure 1), and a new resource binding algorithm (label 3 of Figure 1).

—*High-level test concepts*: We propose several concepts in the behavioral description of a design (CDFG) for considering test and testability issues by inserting faults, the design's behavioral elements (operators).

—*RTL DFT*: The proposed DFT methodology prepares test paths for RTL modules to which the precomputed test vectors of the modules are applicable. The steps for the DFT method are outlined next:

(a) First, we perform some modifications in the RTL modules to prepare functional transparency and guarantee the fault propagation and value justification in the test path of the CDFG.
(b) We assign faults to the operators in the CDFG. Each faulty *operator* in the CDFG we refer to as a *victim*, and can be a representative of a faulty RTL *module*.
(c) Because a faulty RTL module may have multiple victims in the CDFG, the best victim that leads to a minimum test time for that RTL module is selected.
(d) Based on the selected victim, the original CDFG is trimmed down, adding a few transitions or states to the controller.

—*New resource binding algorithm*: We develop a resource (operator) binding algorithm in a synthesis method so that the test application time of the

faulty operator becomes minimal. The steps for the test synthesis method are outlined next:

(a) First, we define two testability metrics (group controllability and group observability) for groups of CDFG operators that map to the same RTL module.
(b) Based on the defined testability metrics of each group, we estimate the number of test clock cycles (group test cycle) which is needed to test the RTL module that corresponds to the group.
(c) Based on the group Test Cycle, a heuristic algorithm groups operations in the CDFG so that the test clock cycles of the design are minimized.

In this work, we assume that the controller of a design has a reset signal. This assumption is usually satisfied for real-life circuits. We also assume that all variables in a CDFG are mapped to registers of the same bit width. However, this assumption of fixed bit-width can easily be relaxed.

Section 2 describes the CDFG test concepts. Our test synthesis algorithm is explained in Section 3. Section 4 demonstrates our experimental results. Finally, Section 5 concludes our article and points to some future work.

2. TEST CONCEPTS AND THE PROPOSED DFT

Our proposed synthesis algorithm uses high-level test and testability concepts that are fully explained in Hosseinabady [2006]. In this section, we briefly explain these concepts and show how they can be used to reduce test application time.

2.1 CDFG Testability

Using an example, we explain the behavioral test concepts.

Example. Consider the scheduled and bound CDFG of Figure 2(a). This CDFG is comprised of two multipliers and three adders. It also contains three groups of operations and the horizontal lines in this figure represent the clock boundaries. Figure 2(b) shows the datapath of this circuit. The corresponding controller specifications are given in the state transition table and state transition graph of Figure 2(c).

The operators of the CDFG in Figure 2(a) that have been mapped into the same hardware in the datapath (shown in Figure 2(b)) are put into the same *group*, shown by the shaded areas in Figure 2(a). For example, the module Add1 of Figure 2(b) performs the add operations 2 and 3 in Figure 2(a).

The role of a controller is to map the CDFG of a circuit into its datapath. Consequently, all paths in a CDFG exist in the datapath of a circuit, but the opposite is not always true, that is, the obtained datapath has additional paths which are translated into extra edges in the CDFG. To realize and thus utilize these edges, changes may be needed in the circuit's controller.

Consider a faulty operator in the CDFG of a circuit, and assume that some test vectors for detecting faults for an individual instance of this operator have already been generated. To utilize these precomputed test vectors, we have to find a path from the primary inputs of the circuit to the inputs of this operator.

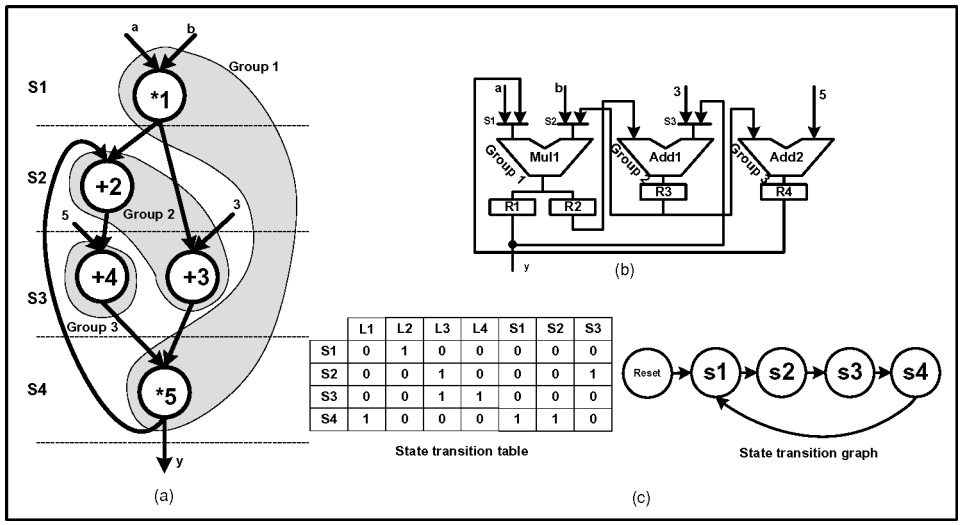


Fig. 2. An example CDFG: (a) scheduled and bound CDFG; (b) datapath; (c) controller.

Hence, it is possible to assign precomputed test vectors at inputs of the operator by applying appropriate values at primary inputs of the CDFG. In addition, a path from the output of the faulty operator to primary outputs of the CDFG must propagate the effect of a fault. The path from primary inputs of the circuit to the faulty operator, and from this operator to the primary outputs, is referred to as a *test path*. A test path for an operator in the CDFG is also a test path for the related module at the datapath to which that operator is mapped during allocation. A test path consists of two justification paths and one propagation path. For example, a test path for Operation 1 of Figure 2(a) is ($a \rightarrow *_1$, $b \rightarrow *_1$, $*_1 \rightarrow +_3 \rightarrow *_5 \rightarrow y$), where the first two parts are justification paths and the third is the propagation path.

CDFG loops are important in finding a test path. Two different types of such loops are *local loop* and *global loop*. A loop is a local loop if it is within a single group of operators in the CDFG, otherwise, it is a global loop. The CDFG of Figure 2(a) has one global loop, namely, ($+2 \rightarrow +4 \rightarrow *5 \rightarrow +2$).

Note that for a faulty operator, a test path with the following conditions guarantees justification of precomputed test vectors and propagation of fault effects: (1) In the presence of a reconvergent fanout, only one input of an operation in the test path should be influenced by the test path and its other input should have a known value; (2) all modules in the test path should be transparent to propagate and justify values on their inputs and output, respectively.

By adding a few extra transitions or states to the controller so as to cut off some paths, the first condition is satisfied. To satisfy the second condition, the next section represents a few modifications to the modules of an RTL design library (as the proposed DFT) and considerations during the synthesis algorithm. The experimental results show that these modifications and considerations add low area and delay overhead to the datapath.

2.2 Proposed DFT

Applying the backward implication to each module through the justification path of the module-under-test justifies the precomputed test vector on the primary inputs. Furthermore, forward implication of each module along the propagation path of the module-under-test propagates the fault effects to a primary output.

The behavior of a two-input operator Op of the CDFG can be modeled as $y \equiv F(a, b) (\text{mod } 2^n)$, where F is the functionality of the operator, a and b are two inputs of the operator, and n is the bit width of the output. Assume that operator Op is on the defined test path, that its a input is affected by the test path, and its b input has a known value during the test. The justification and propagation conditions in our defined test path are as follows.

—*Justification*: For any desired value on output y, there is an appropriate value on a so that $y \equiv F(a, b) (\text{mod } 2^n)$.

—*Propagation*: Two different values on the a input leads to two different values on the output y.

An element of the RTL design library that satisfies the aforementioned conditions has the functional transparency property. Some elements of the RTL design library already have functional transparency. Modifying the elements of the RTL design library or adding a multiplexer to the datapath provides functional transparency for other RTL design library elements.

Adder, Inverter, and Buffer are three elements in the RTL design library that already satisfy the preceding conditions. A multiplier satisfies them if its b input is an odd number. Thus, to guarantee the existence of an odd value on the inputs of the multiplier, the LSB of each input of the multiplier will be ORed with a controller signal.

A multibit AND/OR operator that has a string of 1/0 on its b input or a path to set a string of 1/0 on this input satisfies the previous properties. Using controllability and observability metrics, the proposed synthesis algorithm supports the existence of such a path. In the absence of such a path, adding a multiplexer in the datapath creates the necessary path.

A relational operator satisfies the justification condition, but may not satisfy the propagation condition. In this case, propagation of fault effects on comparator inputs should be transferred to another path during the test mechanism. Using the controllability and observability metrics, a synthesis algorithm creates such a path. If this path does not exist, adding a multiplexer to the datapath creates it.

While the aforementioned operators have only inputs for data, a conditional operator has a select input (as control inputs) as well as the data inputs. The conditional operator satisfies the justification and propagation conditions if its select input is set to the appropriate value. Thus, test controller should set this value during the test.

Note that 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.

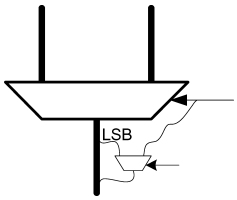


Fig. 3. Transparent multiplexer.

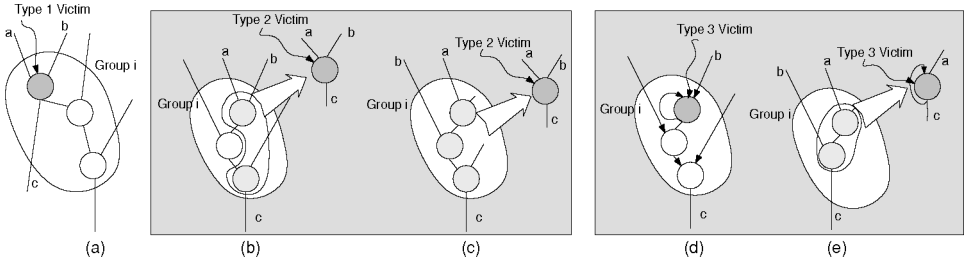


Fig. 4. Types of victim.

Adding this extra multiplexer makes the original multiplexer transparent, so the propagation path from its select input to its output is guaranteed. Figure 3 depicts the process of making the select input of a multiplexer observable.

2.3 Faulty Operator in the CDFG

When an RTL module is faulty, all operators in the CDFG that are mapped to this RTL module are faulty. Thus, we would have multiple faults in the CDFG operators. To handle this issue, all faulty operators in the CDFG, except one (called the victim), will be removed, and a trimmed down CDFG with one faulty operator is constructed. The trimmed down CDFG has only one victim and is used for finding tests for the faulty module of the RT-level description.

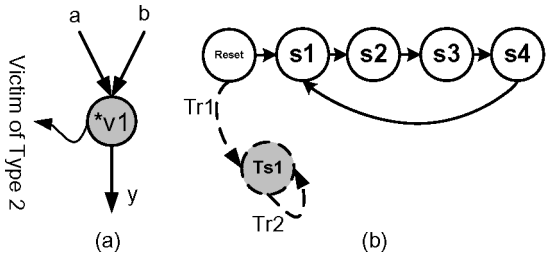
The victim is an existing or new operator that receives its inputs from outside its group and delivers its output to the outside of its group. The new operator is constructed by merging a few existing operators so as to inherit the best controllability and observability of the merged operators.

Victims can be classified into three different types based on their group structures. A victim of Type 1 is an existing operator in the CDFG, while other victims are new.

—*Type 1*: A victim of this type is an existing operator that receives both its inputs from outside of the group and delivers its output to the outside of the group (see Figure 4(a)).

—*Type 2*: A victim of this type is a new operator constructed by merging two or three operators. These merging operators provide the inputs and output of the victim (see Figures 4(b) and (c)).

—*Type 3*: The existence of two consecutive operators (Figure 4(e)) or a local loop (Figure 4(d)) in the group confirms the existence of a self-loop path around



	L1	L2	L3	L4	S1	S2	S3
S1	0	1	0	0	0	0	0
S2	0	0	1	0	0	0	1
S3	0	0	1	1	0	0	0
S4	1	0	0	0	1	1	0
Ts1	1	0	0	0	0	0	0

(c)

Fig. 5. Test path and modified controller to test group 1 of Fig. 2.

the corresponding RTL module. Thus, the RTL module can be fed by one input from 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. Thus, in the case of two consecutive operations, a victim of Type 3 is an operator constructed by merging these two consecutive operators. If a self-loop exists in the group, the victim is the operator with the self-loop.

Extending of the preceding idea, global loops in the CDFG can be handled. 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.

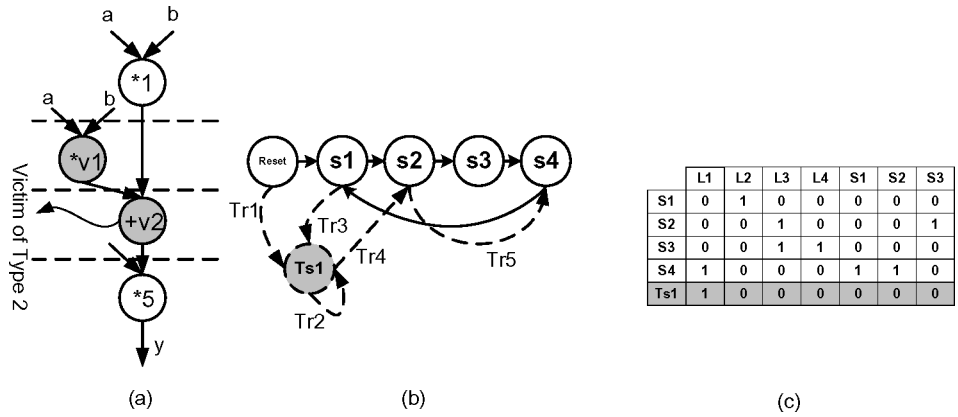
Note that a new victim can be realized by adding a few extra transitions or states to the controller. After finding a victim for a given group according to victim's type, the CDFG is trimmed by deleting all other operators in the group and their connecting nodes.

2.4 Example

Using the example of Figure 2(a), this subsection explains our DFT methodology. Thus, consider the example of Figure 2(a). In group 1 of Figure 2(a), operator *1 has a good controllability and operator *5 has a good observability, thus, merging these two operators, we can generate a victim of Type 2 (i.e., *$v1$ of Figure 5(a)) that is a new operator. Adding a new state to the controller realizes this victim. Figure 5 shows the reduced CDFG for testing the *Mul1* RTL module and the corresponding modified controller. In this case, the test path in the datapath is (a→*v1, b→*v1, *v1→y) and the controller during test has only the new state *Ts1* in which the operator *$v1$ is activated. Therefore, one clock cycle is enough to apply a precomputed test vector to the module-under-test.

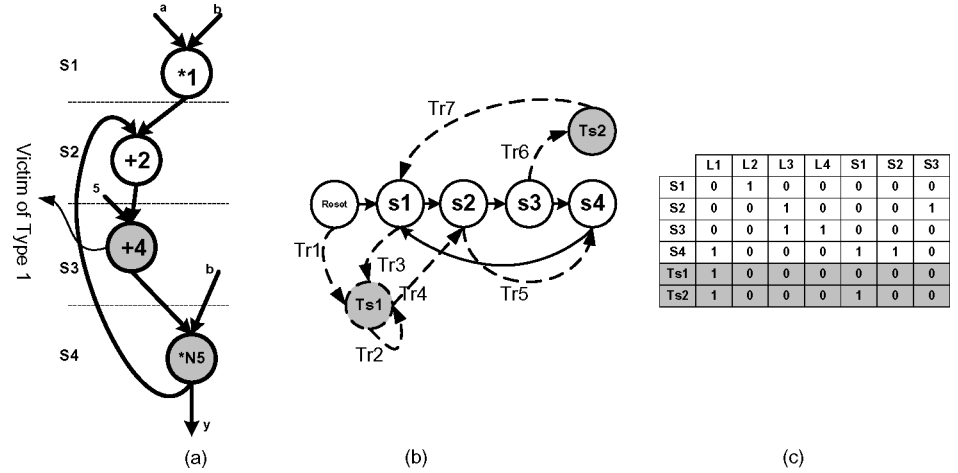
Group 2 of Figure 2(a) has a victim of Type 2. Since both inputs of operator +2 come from outside of the group and the output of operator +3 goes to the outside of the group, the victim of this group satisfies the conditions of the victim of Type 2. Because the outputs of the two operators +2 and +3 are bound to the same register, an extra transition that bypasses state 3 of Figure 2(a) realizes the victim of this group (i.e., transition *Tr5* from state 2 to state 4 of Figure 6(b)).

Using the added operator and related state to test group 1 (i.e., operator *$v1$ and state *Ts1* of Figure 5), and state $S1$ of the controller, the victim can get



	L1	L2	L3	L4	S1	S2	S3
S1	0	1	0	0	0	0	0
S2	0	0	1	0	0	0	1
S3	0	0	1	1	0	0	0
S4	1	0	0	0	1	1	0
Ts1	1	0	0	0	0	0	0

Fig. 6. Test path and modified controller to test group 2 of Fig. 2.



	L1	L2	L3	L4	S1	S2	S3
S1	0	1	0	0	0	0	0
S2	0	0	1	0	0	0	1
S3	0	0	1	1	0	0	0
S4	1	0	0	0	1	1	0
Ts1	1	0	0	0	0	0	0
Ts2	1	0	0	0	1	0	0

Fig. 7. Test of group 3.

its test vectors in two clock cycles (see Figure 6). Furthermore, state *S4* of the controller propagates the test response to the primary output. Figure 6(a) shows the reduced CDFG to test group 2 of operators. Based on this reduced CDFG, four clocks are needed to test group 2 operators. The modified controller is shown in Figures 6(b) and (c). As shown in Figure 6, the test path in the reduced CDFG is (a $\rightarrow$ *1 $\rightarrow$ +v2, a $\rightarrow$ *v1 $\rightarrow$ +v2, +v2 $\rightarrow$ *5 $\rightarrow$ y) and the corresponding path of the controller is (s1 $\rightarrow$ Ts1 $\rightarrow$ s2 $\rightarrow$ s4).

Group 3 of Figure 2(a) has only one member. Its inputs are fed from primary inputs a and b through operators *1 and +2. In addition, its output propagates to the primary output y through operator *5. Note that there are reconvergent paths (i.e., paths *1 $\rightarrow$ +2 $\rightarrow$ +4 $\rightarrow$ *5 and *1 $\rightarrow$ +3 $\rightarrow$ *5) with which to apply test vectors to the operator of this group. To resolve this problem, we can modify state 4 such that operator *5 is fed by primary input b and the output of group 3. The modified state and operator are shown in Figure 7 as state *Ts2* and

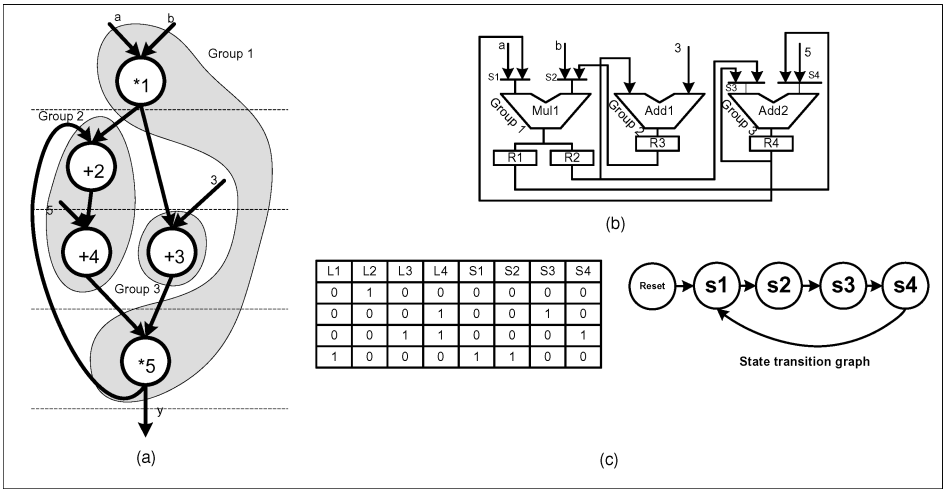


Fig. 8. Alternative operator grouping for design of Fig. 2.

operator *N5, respectively. Figure 7(a) shows the reduced CDFG to test group 3, and Figures 7(b) and (c) show the modified controller of the design. The test path of the CDFG is (a→*1 → +2 → +4, 5 → +4, +4 → *N5 → y). Based on the reduced CDFG of Figure 7(a), four clock cycles are needed to apply a test vector to the corresponding RTL module of group 3.

As the number of precomputed test vectors for a 32-bit adder and multiplier based on ATALANTA [Lee and Ha 1993] (a public domain combinational test generator tool) are 23 and 89, respectively, the total number of required clock cycles to test the datapath of this implementation becomes (4+4)*23 + 1*89 + 3 = 276, in which 4*23 clock cycles are needed to test groups 2 and 3, 1*89 clock cycles are needed to test group 1, and 3 clock cycles are needed to reset the controller to switch between groups during test.

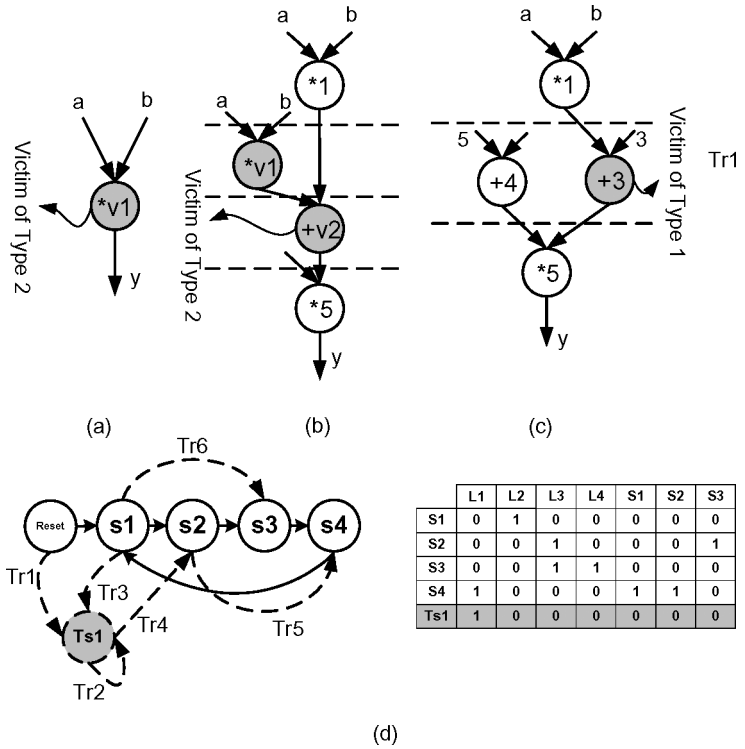
2.5 Effects of Resource Binding on TAT

Synthesis has a high impact, not only on increasing the chance of finding a victim in a group, but also on reducing test application time. As mentioned, our aim is to reduce the test application time by finding a suitable module binding in the CDFG of a design. In this section, the effects of module binding on the testability and test application time of the design are shown using an example.

Example. Figure 8(a) shows an alternative operator grouping for the given scheduled CDFG of Figure 2. Figures 8(b) and (c), respectively, show the datapath and controller for the new implementation.

The victim of group 1 in the new operator binding (i.e., Figure 8) is the same as that of group 1 in Figure 2. The reduced CDFG to test this group is shown in Figure 9(a). Therefore, one clock cycle is needed to apply a test vector to this group.

Group 2 of operators in Figure 8(a) consists of two operators +2 and +4. Inputs of operator +2 come from outside the group and the output of operator +4 goes to the outside of the group. Hence, the victim of this group is of Type 2



	L1	L2	L3	L4	S1	S2	S3
S1	0	1	0	0	0	0	0
S2	0	0	1	0	0	0	1
S3	0	0	1	1	0	0	0
S4	1	0	0	0	1	1	0
Ts1	1	0	0	0	0	0	0

Fig. 9. Test paths for group of Fig. 8.

and generated by merging operators +2 and +4. Using the new operator to realize the victim of group 1 (i.e., **v1* of Figure 9(a)), Figure 9(b) shows the reduced CDFG to test group 2. Based on this figure, four clock cycles are needed to apply a test vector to the module-under-test.

Group 3 of Figure 8(a) consists of one operator and this operator is the victim of Type 1. Bypassing clock cycle 2 of Figure 8(a) (*Tr6* of Figure 9(d)), we can use the reduced CDFG of Figure 9(c) to test this group. Based on this reduced CDFG, three clock cycles are needed to apply a test vector to the module-under-test. Figure 9(d) shows the modified controller.

Therefore, using the precomputed test vectors generated by ATALANTA, the total number of required clock cycles to test the datapath of the given design is 4*23 + 3*23 + 1*89 + 3 = 253, of which 4*23 and 3*23 clock cycles are needed to test groups 2 and 3, respectively, 1*89 clock cycles are needed to test group 1, and 3 clock cycles are needed to reset the circuit to switch between groups during test.

Comparing the total number of required test clock cycles of the two different synthesized structures in Figures 2 and 8 (276 versus 253 clock cycles) shows the impact of group binding during the synthesis process on test application time.

3. HIGH-LEVEL SYNTHESIS

Our aim in this section is to consider testability during operator grouping while minimizing test application time. For this purpose, we propose a heuristic algorithm for operator grouping. The heuristic algorithm used for operator grouping may be coupled with any high-level synthesis tool for reducing test application time regarding testability metrics. This algorithm is based on the group testability measurement defined in the following subsection. The input to the algorithm is a scheduled control data flow graph (CDFG) of the design.

3.1 Testability Analysis

In this section, a model is presented for calculating the test application time. For this purpose, we assign two metrics to each group in the CDFG of a design. These metrics are called *group controllability* (GC) and *group observability* (GO).

—*Group controllability (GC)*: Controllability of a group (GC) is the difficulty of setting the inputs of the group victim to any desired value. In other words, the GC of a given group is the minimum number of clock cycles during which both of the group victim inputs can be set to the desired values.
—*Group observability*: Observability of a group (GO) is the difficulty of observing the values of the group victim output on a primary output of the design. In other words, the GO of a given group is the minimum required number of clock cycles for propagating the group victim output to a primary output of the design.
—*Group test cycle*: Test cycle of a group (GTC) is defined as the sum of the GC and GO of that group. In other words, GTC is the number of required clock cycles to apply a test vector to that group.

To measure GC and GO metrics, first the controllability and observability of each signal in the CDFG of a design must be measured. The controllability of each signal in the CDFG is the number of times various operations must be activated to control that signal. Similarly, the observability of each signal in the CDFG is the number of times various operations must be activated to observe that signal at a primary output. Then, based on the obtained controllability and observability metrics for each signal of the group and based on the victim type of the group, the group controllability and observability metrics are measured.

3.2 Module Binding Based on the Testability Metrics

As mentioned earlier, suitable operation grouping during synthesis can reduce the test application time of a design. In this section, we propose a heuristic grouping method based on our testability metrics. In our method, we assume that a design is specified by a scheduled CDFG.

The aim of the proposed algorithm is to incrementally group the compatible operations (nonoverlapping operations of the same type) in order to reduce the test application time of the design. Our algorithm consists of two stages: computing the testability metrics and grouping appropriate operations. Algorithm 1

shows the pseudocode of the proposed algorithm. As can be seen in this algorithm, individual operations are first interpreted as individual groups and their corresponding testability metrics measured (lines 3–5 in Algorithm 1). In the following steps, the groups are sorted based on their corresponding group test cycle (GTC) (line 6), and that group with the greatest group test cycle (G_{worst}) is selected (line 7). Then, considering the impact of merging G_{worst} with each group compatible with G_{worst}, we find the best candidate (the group which implies the greatest improvement on the GTC of G_{worst}) to be merged with G_{worst} (lines 8–22) and generate a new group. Consequently, this new group replaces the two merging groups. This process is repeated until the number of groups does not change.

Algorithm 1: Operator Grouping Algorithm.

```
1   GroupAlgorithm(CDFG)
2   {
3     Assign each operator of the CDFG to an individual group (G={Gi| i = 0,...,n})
4     n=number of groups
5     Compute the testability metrics (GC, GO, GTC) of each group
6     Sort the groups based on their GTC
7     Gworst = Group with the largest GTC
8     For (i = 1; i <= n; i++)
9                 If (Gworst is compatible with Gi)
10                {
11                  Gworst,i=Merge(Gworst, Gi);
12                  GTCworst,i=GTC(Gworst,i);
13                }
14    If (Gworst is not compatible with any Gi)
15    {
16      Remove Gworst from G
17      If (G is empty)
18            Exit();
19      Else
20            Goto Line 5;
21    }
22    J = Index of a merged group with minimum GTC
23    Replace Gj and Gworst with Gworst,j
24    n = n – 1;
25    Goto Line 5;
26  }
```

3.3 Controller Modification and Test Generation

After applying the synthesis method, a scheduled and bound CDFG will be obtained. Based on the obtained groups of operators in the CDFG, the datapath and controller of the design are modified by the DFT algorithm. The DFT algorithm realizes the test paths and their minimal CDFG. Based on test path definition, only one of the inputs of operators in the test path is unknown. Thus, by inserting precomputed test vectors to the inputs of a victim operator and backward tracing, appropriate test vectors at the primary inputs of the circuit are obtained.

Table I. Circuits Statistics and Their Area Overhead of Lemmas

Circuit	# of Inputs	# of Outputs	# of ANDs	# of NANDs	# of ORs	# of NOTs	# of FFs	Over %
Paulin	66	64	4112	10153	77	21	236	0.31
3rd Order IIR	34	32	4109	9810	17	30	207	0.25
5th Order IIR	34	32	6136	14797	69	54	369	0.21
7th Order IIR	34	32	6172	14951	73	76	435	0.19
5th order Gray_MLF	34	34	2419	5682	65	58	377	0.31
6 Tap Wavelet Filter	34	64	2458	6089	17	43	371	0.47
5th Order Elliptic Filter	34	32	5238	12681	189	92	615	0.43
6th Order Parallel IIR	34	32	8275	23387	67	82	526	0.19
8th Order Parallel IIR	34	32	12273	29584	131	96	687	0.16
9th order wave digital filter	34	32	8817	26165	257	134	906	0.27

4. EXPERIMENTAL RESULTS

We implemented our proposed methods (i.e., the DFT of Section 2.2 and heuristic operator binding of Section 3) using several designs to show the effectiveness of this process. This section explains these results in two parts. Using several benchmarks, in the first part we evaluate our DFT method, and in the second part we examine the proposed operator binding algorithm. The number of transistors is used as a measure of area in the following results. We have considered standard CMOS technology that uses four transistors for NAND and NOR gates, and six for AND and OR gates. An inverter uses two transistors, and a flip-flop is implemented using eight transistors (i.e., C2MOS flip-flop). To evaluate our works, we have used ten benchmark circuits. Table I gives the statistics of information about these 32-bit benchmarks. Paulin [Jha et al. 2002] is a differential equation solver. The other examples, 3rdOrderIIR, 5thOrderIIR, 7thOrderIIR, 6TapWaveletFilter [Kin 1999], 5thOrderEllipticalFilter [Potkonjak et al. 2004], 5thOrderGray_MLF [Kirovski et al. 1999], 6thOrderParallelIIR [Kirovski and Potkonjak 1999], 8thOrderParallelIIR, and 9thOrderWaveDigitalFilter [Kollig and Al-Hashimi 1999] are filter circuits.

4.1 Evaluation of DFT Algorithm

We modify module elements in the RTL design library to satisfy the justification and propagation conditions of Section 2.2. Then, these modified modules are applied to ten benchmark circuits. The last column of Table I shows the area overhead of the modified RTL for the benchmark circuits. As expected, modifying the RTL modules has a relatively low area overhead.

We have implemented the DFT algorithm described in Sections 2.3 and 2.4 using C++. In the sequel, we compare our algorithm with the full scan method that is the most frequently used DFT algorithm in the industry. The full scan versions of these benchmarks are obtained by OPUS [Chickermane and Patel 1991] (a partial scan package). Then, ATALANTA generates their test vectors, and ATPG parameters are considered in Table II.

Figure 10 compares the test application time (i.e., the number of clocks needed to test a circuit) of the two DFT algorithms. Note that a logarithmic

Table II. ATPG Parameters [Lee and Ha 1993]

Parameter	Value
Test pattern generation mode	RPT + DTPG + TC
Limit of random patterns (packets)	16
Backtrack limit	10
Test pattern compaction mode	REVERSE + SHUFFLE
Limit of suffling compaction	2
Number of shuffles	12

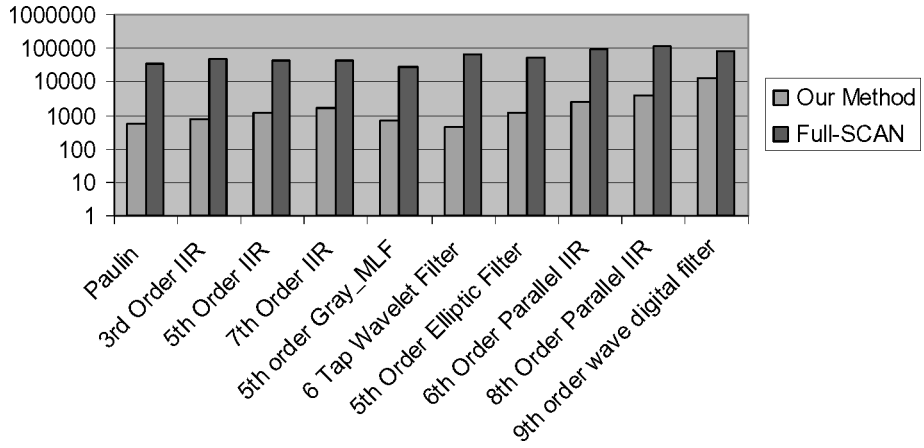


Fig. 10. Test time comparison of our DFT with full scan algorithm.

scale is used in this figure to allow for a comparison of results of the two algorithms. The proposed algorithm has significantly less test time. Figure 11 depicts the area overhead resulting from the two algorithms. In this figure, a logarithmic scale is also used so as to show the results. Our algorithm also has very low area overhead.

Table III compares the number of test vectors and fault coverage of the two algorithms. For the proposed algorithm, the fault coverage numbers were obtained by applying precomputed test vectors to modules of datapaths of the circuit through their test paths, and test vectors of the controller were obtained by applying HITEC [Niermann and Patel 1991] (a sequential test generator tool) to the gate-level description of the design. Our system-level test vectors are concatenations of the datapath and controller test vectors. The fault coverage of the proposed method is slightly less than that of the full scan method. This is because our method only increases the testability of the datapath; if we apply a technique to make the controller more testable, the fault coverage of our method would reach that of a full scan design.

Comparison of the proposed DFT with other DFT methods in literature can be found in Hosseinabady et al. [2006].

4.2 Evaluation of Resource Binding

In this subsection, we compare the proposed synthesis method with the conventional graph coloring synthesis method. Note that datapath allocation and

Table III. Fault Coverage Comparison

Circuit	Our DFT Method		Full-SCAN	
	# of TV	FC %	# of TV	FC %
Paulin	259	99.99	115	99.99
3rd Order IIR	248	99.99	167	99.99
5th Order IIR	385	98.57	107	99.99
7th Order IIR	395	98.30	95	99.99
5th order Gray_MLF	247	98.24	64	99.84
6 Tap Wavelet Filter	231	88.69	149	90.50
5th Order Elliptic Filter	470	99.48	79	99.90
6th Order Parallel IIR	537	99.21	167	99.56
8th Order Parallel IIR	756	99.15	183	99.41
9th order wave digital filter	854	97.84	284	98.45

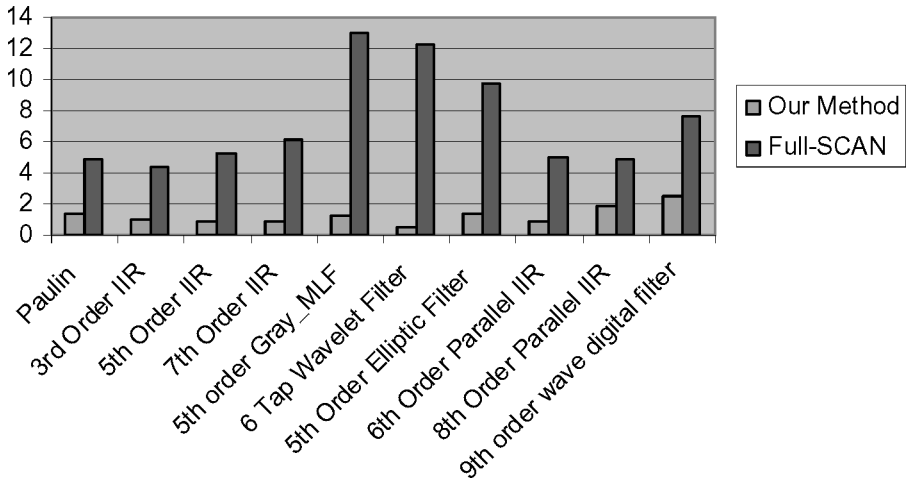


Fig. 11. Area overhead comparison of our DFT with full scan algorithm.

binding consist of register, module, and interconnection allocation. In both the graph coloring and our proposed synthesis, we assume that these three problems are solved separately and sequentially.

Table IV compares the test application time obtained by using the conventional synthesis method with that of the proposed synthesis method. Columns 2–6 show the number of inputs, outputs, add/subtract operators, multiply operators, and clock cycles of the scheduled CDFG. Column 7 shows the number of clocks to test the circuit using the conventional synthesis algorithm. Column 8 shows the same using our proposed synthesis algorithm. Finally, the last column shows the percentage of TAT reduction for our proposed algorithm. For small circuits for which there are no opportunities to alter the resource sharing, the two methods provide the same test application time (e.g., Paulin benchmark). However, for large circuits with many operators, the proposed method shows a significant reduction in test application time.

Table IV. Comparison of the Proposed Method with Conventional Synthesis

Circuit	Statistics of CDFG					Performance		
	# of Input	# of Output	# of +/−	# of *	# of Clock	Conventional Synthesis	Our Method	TAT Reduction %
Paulin	2	2	4	6	4	578	578	0
3rd Order IIR	1	1	6	8	7	1091	735	32.6
5th Order IIR	1	1	10	12	9	2415	1608	33.4
7th Order IIR	1	1	14	16	11	2458	1648	33.0
5th order Gray_MLF	1	1	20	11	16	675	514	23.8
6 Tap Wavelet Filter	1	2	10	6	11	2633	2050	22.1
5th Order Elliptic Filter	1	1	26	8	15	4135	3216	22.2
6th Order Parallel IIR	1	1	14	12	7	2521	1941	23.0
8th Order Parallel IIR	1	1	19	16	8	3857	3279	15.0
9th order wave digital filter	1	1	28	10	35	23083	17736	23.2

Table V. Area Overhead Comparison

	Conventional Synthesis				Our Method				
Circuit	+/−	∗	R	# of Trans.	+/−	∗	R	# of Trans.	Overhead %
Paulin	2	2	7	67676	2	2	7	67676	0
3rd Order IIR	1	2	6	65712	1	2	6	66860	+1.75
5th Order IIR	2	3	11	99478	2	3	10	93366	−6.14
7th Order IIR	2	3	13	100760	2	3	13	100306	−0.45
5th order Gray_MLF	2	1	11	40760	2	1	11	41972	+2.97
6 Tap Wavelet Filter	2	1	11	42260	2	1	11	43332	+2.54
5th Order Elliptic Filter	3	3	17	88390	3	3	17	88002	−0.44
6th Order Parallel IIR	3	4	16	147972	3	4	14	148700	+0.49
8th Order Parallel IIR	4	6	21	198448	4	6	20	200630	+1.10
9th order wave digital filter	8	5	27	166620	10	5	30	167964	+0.81

Table V shows the area overhead of the two aforementioned synthesis algorithms in terms of the required number of transistors for implementation of each design.

Columns 2, 3, and 4 show the number of add/subtract operators, multiply operators, and registers in the RTL design obtained by the conventional synthesis method, respectively. Column 5 shows the number of transistors in the final implementation of that design. Columns 6–9 show the same information as those of Columns 2 to 5, but as done by our synthesis method. The last column shows the area overhead of the proposed synthesis method with respect to the conventional synthesis method. In some cases, the area of our method is less than that of the conventional method (e.g., in 5th Order IIR the area of our method is significantly less) and in others, the area overhead of our method is slightly (less than 3%) more than that the conventional method. Since only test application time is considered during the proposed operator binding algorithm, the negligible area overhead justifies our method.

4.3 Comparison with Other Methods

Table VI(a) compares the proposed algorithm with two approaches of Ghosh et al. [1997] and Wada et al. [2000]. The bit width of the circuit of this table is 16-bit. Ghosh et al. [1997] present techniques that add test hardware to a given RT-level circuit obtained by behavioral synthesis in order to ensure that the embedded elements in the circuit are hierarchically testable. They generate a system-level test set to deliver precomputed test sets to each element in the RTL circuit. Wada et al. [2000] introduce strong testability and propose a DFT method based on this for RTL datapaths to achieve 100% fault coverage. This is accomplished by introducing additional functionality (hardware) to the modules in a datapath, thus increasing the testability of the circuit-under-test. The additional hardwares are called the mask element and thru function. Table VI(b) compares the proposed algorithm with TAO, the DFT approach presented in Ravi et al. [1998]. The bit width of the circuit of this table is 16-bit. Exploring datapath and adding hardware to the datapath, as well as modifying the controller, TAO increases the testability of an RTL circuit.

Table VI. Test Application Time Comparison

	[Ghosh et al. 1997]				[Wada et al. 2000]		Our Method	
	Without C-testable		Without C-testable					
	FC %	TAT	FC %	TAT	FC %	TAT	FC %	TAT
5^{th} Order Elliptic Filter	99.46	5783	99.51	2612	N/A	N/A	99.48	3216
Tseng	99.59	1276	99.71	840	N/A	N/A	99.98	215
Paulin	99.70	2111	99.72	874	100	2512	99.99	478
GCD	N/A	N/A	N/A	N/A	100	760	99.97	263

(a)

	[Ravi et al. 1998]		Our Method	
	FC %	TAT	FC %	TAT
Chain_mult	99.2	381	99.97	194
Paulin	99.5	507	99.98	251
GCD	99.1	278	99.81	145

(b)

(a) 32-bit circuits (b) 16-bit circuits

Based on these tables, our method, which synthesizes the CDFG and uses RTL modification to decrease the test application time, is more efficient than the other algorithms discussed herein.

5. CONCLUSIONS AND FUTURE WORK

This article presents a novel test synthesis method to reduce the test application time for sequential circuits. This method requires minor modifications to the controller in the RT-level description of a circuit. The control data flow graph (CDFG) representation of an RTL circuit is used for analyzing the testability of individual RT-level operations. Additional datapaths are introduced by altering the controller states or transitions. This method considerably reduces the test application time by ignoring unnecessary control states in the test process.

Resource binding in a CDFG maps variables within it to registers (register allocation), binds operations to modules (module allocation), and offers interconnection among the registers and modules. In our proposed method, we assume that these steps are solved separately. We can consider register allocation during module binding to reduce test application time in future work.

REFERENCES

BHATIA, S. AND JHA, N. K. 1994. Genesis: A behavioral synthesis system for hierarchical testability. In *Proceedings of the European Design and Test Conference* (Feb. 28–Mar. 3). 272–276.

CHICKERMANE, V. AND PATEL, J. H. 1991. A fault oriented partial scan design approach. In *Proceedings of the International Conference on Computer-Aided Design* (Nov.). 400–403.

GHOSH, I., RAGHUNATHAN, A., AND JHA, N. K. 1997. Design for hierarchical testability of RTL circuits obtained by behavioral synthesis. *IEEE Trans. Comput. Aided Des. Int. Circ. Syst. 16*, 9 (Nov.), 001–1014.

HOSSEINABADY, M., LOTFI-KAMRAN, P., LOMBARDI, F., AND NAVABI, Z. 2006. Low overhead DFT using CDFG by modifying controller. *IEE Proc. Comput. Digital Technol.* to appear.

INOUE, M., SUZUKI, K., OKAMOTO, H., AND FUJIWARA, H. 2003. Test synthesis for datapaths using datapath-controller functions. In *Proceedings of the 12th Asian Test Symposium (ATS)* (Nov). 294–299.

KIN, H. B. 1999. High-Level synthesis and implementation of built-in self-testable data path intensive circuit. Ph.D. dissertation, Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

KIROVSKI, D. AND POTKONJAK, M. 1999. Localized watermarking: Methodology and application to behavioral synthesis. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)* (Nov.). 596–599.

KIROVSKI, D., POTKONJAK, M., AND GUERRA, L. 1999. Improving the observability and controllability of datapaths for emulation-based debugging. *IEEE Trans. Comput. Aided Des. Int. Circ. Syst. 18*, 11, 1529–1541.

KOLLIG, P. AND AL-HASHIMI, B. M. 1999. Reduction of latency and resource usage in bit-level pipelined data paths for FGPAs. In *Proceedings of the ACM / SIGDA 7th International Symposium on Field Programmable Gate Arrays (FPGA)*. 227–234.

LEE, H. AND HA, D. 1993. On the generation of test patterns for combinational circuits. Tech. Rep. 12_93, Department of Electrical Engineering, Virginia Tech.

MAKRIS, Y., PATEL, V., AND ORAILOGLU, A. 2001. Efficient transparency extraction and utilization in hierarchical test. In *Proceedings of the 19th VLSI Test Symposium (VTS)* (Apr. 29–May 3). 246–251.

MAKRIS, Y., COLLINS, J., AND ORAILOGLU, A. 2002. Fast hierarchical test path construction for circuits with DFT-free controller-datapath interfaces. *J. Electron. Test. Theory Appl. 18*, 1, 29–42.

NIERMANN, T. M. AND PATEL, J. H. 1991. HITEC: A test generation package for sequential circuits. In *Proceedings of the European Design Automation Conference* (Feb.). 214–218.

POTKONJAK, M., DEY, S., AND WONG, J. L. 2004. Optimizing designs using the addition of deflection operations. http://trix.cs.ucla.edu/jenni/papers/HotPot_TR.pdf.

RAVI, S., LAKSHMINARAYANA, G., AND JHA, N. K. 1998. TAO: Regular expression based high-level testability analysis and optimization. In *Proceedings of the International Test Conference* (Oct.). 331–340.

WADA, H., MASUZAWA, T., SALUJA, K. K., AND FUJIWARA, H. 2000. Design for strong testability of RTL data paths to provide complete fault efficiency. In *Proceedings of the 13th International Conference on VLSI Design* (Jan.). 300–305.

Received September 2005; revised June 2006; accepted October 2006