Enhancing High-Level Control-Flow for Improved Testability

Frank F. Hsu **Elizabeth M. Rudnick** **Janak H. Patel**
Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL

Abstract

In this study, we present a controllability measure for high-level circuit descriptions and a high-level synthesis-for-testability technique. Unlike many recent studies in the area of high-level synthesis for testability that focus on improving the testability of data paths, the objective of our approach is to improve the testability of synthesized circuits by enhancing the controllability of the control flow. Experimental results on several high-level synthesis benchmarks show that when this approach is used prior to logic synthesis, a shorter ATPG time, a smaller test set, and better fault coverage and ATPG efficiency are often achieved. Implementation of this technique requires minimal logic and performance overheads and allows test vectors to be applied at clock-speed.

1 Introduction

A majority of current design-for-testability (DFT) and synthesis-for-testability (SFT) approaches employ full scan or partial scan techniques to improve the testability of VLSI circuits. Although the scan approach greatly reduces the difficulties of sequential circuit test generation, it also has many disadvantages. Besides the area overhead required to implement the scan chain and increased time needed for test application, scan-based solutions may have limited capabilities for at-speed test. Scan tests targeted at stuck-at faults cannot be applied at the operational speed of the circuit due to scanning in and out of flip-flop values. Work done by Maxwell *et al.* [1] has shown that a stuck-at fault test set applied at clock-speed is able to identify more defective chips than a test set having the same fault coverage but applied at a slower speed.

Furthermore, most conventional DFT and SFT techniques are applied near the end of the design cycle, when the gate-level structure of the circuit is known. Unfortunately, as the size and complexity of VLSI circuits increase, the testing process also becomes increasingly complicated and costly. Many studies [2, 3, 4] have been conducted recently on the analysis of testability and the application of SFT schemes early in the design cycle. By integrating the testability techniques prior to logic synthesis, the designer is able to obtain an easily testable circuit with reasonable overhead [5]. SFT techniques presented in [6, 7, 8] utilize information at the Register Transfer (RT) level to generate easily testable designs. Dey *et al.* [9] presented a method to break data flow loops by exploiting hardware sharing to minimize the usage of scan registers. In these approaches, the testability enhancement is achieved at the expense of employing scanned memory elements.

A nonscan DFT approach that uses RT-level structural information to produce testable data paths was presented in [10]. This approach utilizes multiplexers to implicitly break feedback loops in the data path and redirect data to improve controllability and observability of logic modules. Similarly, the technique proposed in [11] provides a means of augmenting data flow paths by inserting test statements into the high-level description prior to logic synthesis. These high-level approaches improve the data path testability by adding buses and multiplexers to the circuit; however, high area overhead for routing these buses result.

Instead of focusing on data path testability, we introduce a new testability measure based on the controllability of branch conditions in the control-data flow graph (CDFG). The entry and exit conditions of some loops in a CDFG are often hard to control; hence, they may cause difficulties during automatic test generation. We will concentrate on these hard-to-control (HTC) loops in the system. The proposed controllability measure is first employed to identify HTC loops in a CDFG. Then a nonscan SFT scheme is introduced to augment controllability of the exit-condition node within each of the HTC loops. The purpose of adding controllability to the control flow is to achieve efficient fault activation and facilitate fault-effect propagation to the primary outputs (PO's), while using the data paths that already exist in the system.

In [12], Lee *et al.* presented a data path scheduling algorithm for easily testable systems. Unlike the approach in [12], our approach performs the behavioral modifications at the high level, such that any implementation of this behavior is *inherently* testable. The advantage of this technique is that it can use any synthesis tool, since the technique does not require any modification in the synthesis procedures.

The design flow used in this work is illustrated in Figure 1. The high-level description is analyzed, and

*This research was supported in part by DARPA under Contract DABT63-95-C-0069, in part by the Semiconductor Research Corporation under Contract SRC 95-DP-109, and by Hewlett-Packard under an equipment grant.

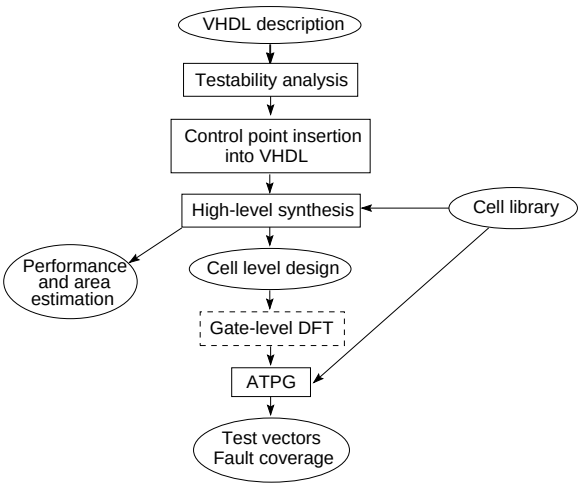

Figure 1: **High-level SFT design flow.**

control points are inserted. A high-level synthesis tool is then used to obtain a gate-level implementation. Gate-level DFT tools may be used if desired, and then an automatic test pattern generator (ATPG) is used to obtain test vectors. Performance and area estimates are made by the synthesis tool.

We will introduce our testability measure that uses the CDFG information available from the high-level description in Section 2. Then the proposed nonscan SFT approach will be discussed in Section 3, followed by experimental results for several high-level synthesis benchmarks in Section 4.

2 Testability Measure

A typical CDFG consists of operation nodes, decision nodes, and transition arcs connecting these nodes. As an example, the high-level description and flow graph for the Greatest-Common-Divider (GCD) circuit are shown in Figure 2. The rectangle processing function contains serial operations that are to be executed by the circuit. The diamond shape decision function denotes the decision node with branching conditions. Our GCD example contains a *while* loop, as described in the high-level program. The nodes in the *while* loop are connected by bold lines in the corresponding CDFG, as shown in Figure 2. Before we study the characteristics of the CDFG, some definitions are given.

DEFINITION 1: *A node within a control-data flow graph is the* **locus of execution** *for the system if it is currently being executed by the system.*

DEFINITION 2: *A decision node within a control-data flow graph is* **K-controllable** *if the direction of the branch taken can be controlled directly or indirectly by the input values* **K** *clock cycles before the* **locus of execution** *reaches the node, where* **K** *is the smallest such integer.*

DEFINITION 3: *A decision node within a control-data flow graph is* **non-controllable** *if the direction of the branch taken cannot be controlled directly or indirectly by the primary inputs within any predetermined number of clock cycles prior to the* **locus of execution** *reaching the node.*

In the remaining text, the term *locus* will mean **locus of execution**. During normal operation, the GCD circuit first reads values from the primary inputs (PI's). Then depending on the input values, the system either sends the result to the PO's, or spends several iterations within the *while* loop before the result is ready for output. Notice that while the *locus* stays within the loop, the PI's are ignored and PO's are held constant. Since each iteration is triggered by a rising edge of the system clock, the system becomes uncontrollable and unobservable for several clock cycles until the *locus* exits the *while* loop.

By applying the proposed testability measure as described in Definition 2, the controllability of a loop-exit node can be determined, where a *loop-exit* node is the decision node that controls the exit condition of a loop. In the GCD example shown in Figure 2, decision node **A** is 1-controllable, since its decision can be directly controlled by the PI's within one clock cycle before the *locus* arrives at node **A**. Decision nodes **B** and **C** are not easily controllable, because the direction of the branch taken cannot be controlled by the PI's once the *locus* enters the while loop. Thus, decision nodes **B** and **C** are marked as HTC nodes in the control flow. By identifying the HTC decision nodes at the high level, the designer is able to pinpoint the hard-to-test portions of the synthesized circuit. Then suitable SFT or DFT techniques may be applied to improve the overall testability of the system.

It should be noted that the testability measure defined above is used for guidance, and therefore, the accuracy of **K** is not extremely important. When behavioral synthesis is used, the clock is not exactly known before synthesis. Therefore, one needs to approximate the above measure. One approximation is to count the number of register transfer statements in place of clock cycles.

3 Synthesis for Testability

Once the HTC nodes have been identified using the above testability measure, the controllability of these nodes can be augmented. Our SFT technique utilizes one or two extra test input pins to control the outcome of the conditional branches, directly guiding the control flow and indirectly affecting the values of the variables, as shown in Figure 3. We use control points of various types, depending on the situation. Control points of type T1 are **AND**'ed with the original branch condition to allow the condition to be forced *false*.

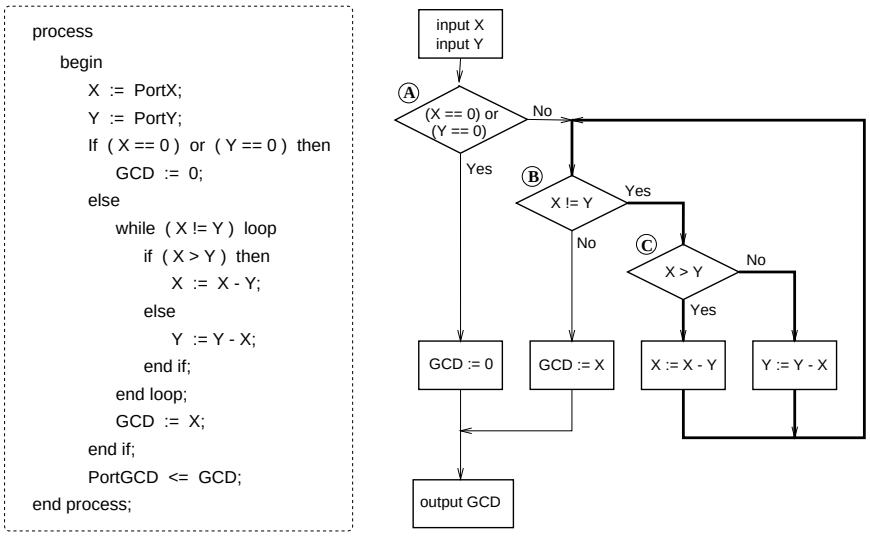

Figure 2: **High-level description and control-data flow graph for circuit GCD.**

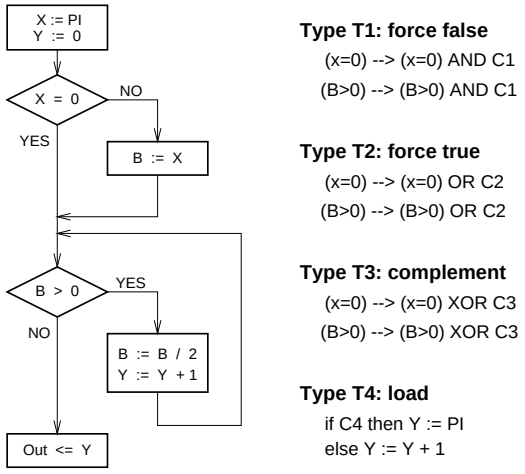

Figure 3: **Controllability insertion in the control-flow.**

Control points of type T2 are **OR**'ed with the original branch condition to allow the condition to be forced *true*. Control points of type T3 enable the branch condition to be complemented through an **exclusive-OR** function. If HTC variables remain in the circuit after control points of type T1, T2, or T3 have been used, a control point of type T4 is added to enable the variables to be loaded from existing PI's. Each test pin can be connected to various decision nodes in the flow graph as long as only one node with augmented controllability is being executed in any clock cycle. The added controllability not only reduces the difficulties of sequential circuit test generation, but it also increases the fault coverage when testing the synthesized circuits. The advantage of controlling the data path through the control of conditional branches is that the area overhead is small and is independent of the width of data registers. Direct control of data register through multiplexers becomes increasingly costly with the increase in data width. From a testability point of view, conditional branches that control loops are far more important than other branches. Therefore, we discuss the loop control in more detail.

3.1 Single Loop

After the controllability of each decision node is computed, nodes that are HTC and nodes that have high *K-controllable* values are possible candidates for the proposed scheme. Although the technique can augment any decision node in the CDFG, our presentation of this SFT approach will concentrate on improving the controllability of HTC exit nodes of functional loops. In the GCD example, decision node **B** is chosen to demonstrate the advantages of our approach. The *non-controllable* node **B** causes the system to have very low controllability and observability while the *locus* stays within the loop. Implementation of the proposed testability scheme using a control point **C1** of type T1 **AND**'ed with the original loop-exit condition **(X!=Y)** allows the *locus* to exit the *while* loop using the extra control point **C1**. With the added controllability, faults activated within the loop can be quickly propagated to the PO's. Test vectors at the PI's can also be applied more efficiently because less time is spent performing operations within the loop.

Additional improvements in controllability can be made by adding a control point of type T2. Control point **C1** of type T1 is **AND**'ed with the original loop-exit condition **(X!=Y)**; then the result is **OR**'ed with control point **C2** of type T2. While control point **C1** allows the *locus* of the system to escape the *while* loop without completing the computation, control point **C2**

enables the *locus* to stay within the loop even after the result has been calculated. The purpose of **C2** is to activate certain faults within the loop when those faults cannot be activated under normal circumstances. Thus, by using two extra test signals to guide the direction of control flow, the test generator is able to create vectors that activate more faults.

Another variation of this approach is to use a single control point, **C3** of type T3, that is **exclusive-OR**'ed with the original branch condition. The use of control point **C3** allows the *locus* of the system to escape the *while* loop early or remain in the loop longer than it normally would. This modified approach is especially useful for reducing the number of test pins when multiple decision nodes are to be controlled independently.

Even after the controllability of branch conditions has been improved, some variables may still be hard to control for certain data value ranges. Direct control of these variables is then necessary to make the circuit testable. Rather than adding multiple extra input pins, the variables are loaded from existing PI's under control of a single control point, **C4** of type T4. Control point **C4** is implemented using one extra test pin or existing test pins of type T1, T2, or T3.

3.2 Multiple Loops

In the GCD example, there is only one *while* loop in the high-level description. However, the proposed technique can be extended for cascaded loops and nested loops. Pictorially, each loop represents a closely connected group of states in the total state space of the system. Figure 4(a) shows a pseudo state transition graph of a sequential circuit that has four cascaded loops. Each oval represents a state space occupied by states traversed within a loop. The normal transition into a state space and out of a state space is shown using bold lines. During normal operation, the *locus* of the system enters a state space at a specific entry state and exits at a specific termination state. When the *locus* stays within one state space, the functions of other state spaces lie idle. Implementation of the proposed SFT scheme provides extra transitions from the middle of a state space to the initial point of the next state space, as shown by dashed lines. Thus, the time required to traverse through all state spaces is reduced because the extra test signal is able to efficiently pass the *locus* through the loops. As shown in [13], reducing the distance among states of a sequential circuit leads to higher fault coverage, and fewer test vectors are required to test the circuit.

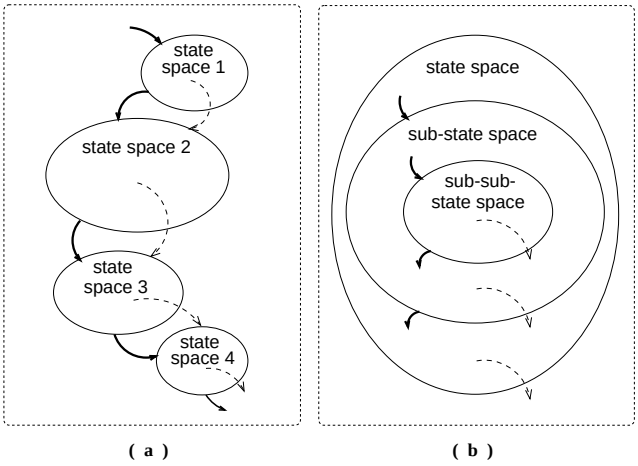

Figure 4: State graphs for (a) cascaded loops and (b) nested loops.

Similarly, Figure 4(b) shows a pseudo state transition graph of a circuit that has three nested loops. The state space of an inner loop is embedded within the state space of its outer loop. The normal transitions into the inner loop and exit from the inner loop are shown using bold lines. The outer loop halts its execution when it initiates the execution of its inner loop; it only resumes its execution after the inner loop returns the *locus* of the system. Notice that when the *locus* is within an inner loop, the other functionalities of the parent loop remain idle. Thus, by providing the extra transition to exit from the inner loop's state space, as shown with dashed lines, the outer loop is able to exercise its own state space more effectively.

4 Experimental Results

Experiments were conducted to demonstrate the effectiveness of the proposed SFT techniques. The high-level synthesis benchmarks HLSynth92 and HLSynth95 were used in the experiments. The circuit descriptions are written in VHDL code, and they were translated into a synthesizable subset of the VHDL language in order to evaluate their testability. Then the HTC nodes were identified, and the VHDL code was modified for improved testability. Finally, gate-level implementations were obtained for the original circuits and circuits with enhanced testability using a commercial logic synthesis system. Characteristics of the high-level circuit descriptions are listed in Table 1, including the number and types of functional loops that exist in each high-level description, the number of HTC loops, the number of HTC variables, and the number of conditional branches. Circuit GCD calculates the greatest common divisor value of two numbers; Diffeq solves differential equations; Barcode processes signals received

Table 1: Circuit Characteristics

Circuit	Functional Loops	HTC Loops	HTC Variables	Conditional Branches
GCD	1 single	1	0	3
Diffeq	1 single	1	1	1
Barcode	2 nested	2	1	5
DHRC	2 cascaded	2	0	2

Table 2: Area and Performance Impact of Proposed SFT Techniques

Circuit	Control Points	Test Pins	Est. Area	Est. Delay	Primitives
GCD	-	0	1156	98	1167
GCD1	T1	1	1159	95	1194
GCD2	T1,T2	2	1146	98	1180
GCD3	T3(2)	2	1135	99	1174
Diffeq	-	0	31,776	93	33,800
Diffeq1	T1	1	30,842	105	31,783
Diffeq2	T1,T2	2	30,843	105	31,786
Barcode	-	0	678	47	598
Barcode1	T1	1	729	44	627
Barcode3	T3(2),T4	2	769	48	655
DHRC	-	0	4619	99	4211
DHRC1	T1	1	4745	99	4259
DHRC2	T1,T2	2	4775	99	4323
DHRC3	T3(2),T4	2	4562	99	4252

from a barcode reader; and DHRC performs differential heat computation.

Several circuit implementations were synthesized for each high-level description. Characteristics of the synthesized circuits are shown in Table 2. One implementation uses the original description, and another uses the description with control point C1 of type T1 AND'ed with the branch condition. For all circuits except Barcode, a third implementation uses the description with two control points, C1 of type T1 and C2 of type T2, where C1 is AND'ed with the branch condition and C2 is OR'ed with the result. For some of the circuits, the original description with two control points of type T3 were also used; in this case, the control points were exclusive-OR'ed with the branch conditions for two decision nodes. In two circuits, one extra control point, C4 of type T4, was added to allow the HTC variables to be controlled directly from existing PI's.

For the Barcode circuit, one of the test pins used to implement the T3 functions was also used for the T4 functions; thus, the number of test pins required was two. During logic synthesis, the optimization directive was set to minimize circuit area, and the timing constraint (clock cycles) was set to a fixed constant across different variations of a given design. The estimated area and delay are reported by the synthesis tool and are shown in the table. The small variations among the sizes and delays are caused by the heuristics used in the circuit synthesis process, which are far from optimum. As shown in Table 2, the proposed SFT technique produces almost no area overhead, while the circuit delays are kept within a reasonable range. The total number of primitives in each gate-level circuit is also shown in the table; primitives include gates such as **and**, **or**, **not**, **mux**, and **dff**, as well as primary inputs and primary outputs. After the gate-level circuit descriptions were obtained, partial scan and full scan were added to the original circuits for comparison. The OPUS partial scan tool [14] was used to select flip-flops to break all cycles in the circuit graph.

The testability enhancements were evaluated using a commercial ATPG system that uses a deterministic, fault-oriented algorithm. Results are given in Table 3, including the number of faults detected, the number of faults found to be untestable, the number of test vectors generated, the fault coverage, the ATPG efficiency, and the execution time for each circuit. Fault coverage is the percentage of faults detected, and ATPG efficiency is the percentage of faults either detected or identified as untestable. Results for partial scan and full scan circuit implementations are included in the table. The number of test pins required for scan is not given, since this depends on the scan implementation. The use of scan elements introduces extra logic gates (primitives) and approximately seven additional faults per scan flip-flop, which accounts for the larger number of total faults in the scan versions. The proposed technique is able to increase the fault coverage and fault efficiency of the synthesized circuits by increasing the effectiveness of the test generation process. The number of aborted faults is greatly reduced by the augmented controllability in the circuits, and the time required to perform automatic test generation is often reduced.

The progression of testability improvement is shown by the results for the *GCD* circuits in Table 3. The testability in general for *GCD* is very low, and many faults are aborted during test generation. By adding a control point of type T1, we are able to increase the fault coverage from 75.0% to 92.3%, and the time for test generation drops by 62%. Adding a control point of type T2 further increases the fault coverage to 98.1%, and the test generation time drops by 85%. Adding controllability to the "**if (X > Y)**" decision node in addition to the *while* loop using two test pins of type T3 results in improvements in fault coverage similar to those for a single control point of type T1. The increasing number of test vectors for the circuits with testability enhancements is justifiable by the improving fault coverage of these vectors. The fault coverage is comparable to the partial scan (GCD_ps) and full scan (GCD_fs) fault coverages, but the scan logic would be expected to have significant area and performance overheads. In summary, the testability of *GCD* is improved by the proposed scheme, and the impact on circuit area and performance is negligible.

Unlike *GCD*, which has low fault coverage initially, *Diffeq* is highly testable even without implementing any testability enhancement scheme. However, the proposed SFT approach can still be applied to further improve its testability. As shown in Table 3, adding the extra control points enables the test generator to produce smaller test sets that achieve higher fault coverage and are generated within a shorter period of time. The fault coverages are as high as the partial scan (Diffeq_ps) and full scan (Diffeq_fs) fault

Table 3: ATPG Results for Proposed SFT Techniques

Circuit	Control Points	Test Pins	Scan DFFs	Total Faults	Detected Faults	Unt. Faults	Test Vectors	Fault Cov.(%)	ATPG Eff.(%)	ATPG Time
GCD	-	0	0	2101	1576	8	200	75.01	75.39	1.87h
GCD_ps	-	-	32	2357	2310	5	583	98.01	98.22	10.9m
GCD_fs	-	-	49	2493	2455	2	338	98.48	98.56	8.96m
GCD1	T1	1	0	2101	1939	5	543	92.29	92.53	43.1m
GCD2	T1,T2	2	0	2111	2071	5	999	98.11	98.34	16.4m
GCD3	T3(2)	2	0	2152	1957	12	713	90.94	91.50	46.7m
GCD1′	T1	1	0	2101	1513	9	143	72.01	72.44	1.79h
Diffeq	-	0	0	86,886	86,778	13	1062	99.88	99.89	52.9m
Diffeq_ps	-	-	96	87,654	87,621	12	987	99.96	99.98	16.5m
Diffeq_fs	-	-	289	89,198	89,161	15	559	99.96	99.98	13.8m
Diffeq1	T1	1	0	82,000	81,981	5	921	99.98	99.98	41.8m
Diffeq2	T1,T2	2	0	82,002	81,990	5	868	99.99	99.99	33.0m
Diffeq1′	T1	1	0	82,000	81,927	6	759	99.91	99.92	40.0m
Barcode	-	0	0	1033	717	18	1977	69.41	71.15	57.1m
Barcode_ps	-	-	26	1243	1236	3	342	99.44	99.68	1.08m
Barcode_fs	-	-	46	1403	1401	2	206	99.86	100	16.5s
Barcode1	T1	1	0	1199	1038	6	888	86.57	87.07	30.7m
Barcode3	T3(2),T4	2	0	1231	1091	2	885	88.63	88.79	28.2m
Barcode1′	T1	1	0	1199	742	88	3930	61.88	69.22	1.36h
DHRC	-	0	0	9546	8939	93	1176	93.64	94.62	1.80h
DHRC_ps	-	-	9	9626	9281	135	1619	96.42	97.82	47.5m
DHRC_fs	-	-	202	11,115	10,914	135	272	98.19	99.41	17.1m
DHRC1	T1	1	0	9491	8887	87	645	93.64	94.55	1.85h
DHRC2	T1,T2	2	0	9539	9036	142	1216	94.73	96.22	1.43h
DHRC3	T3(2),T4	2	0	9556	9104	84	1788	95.27	96.15	1.71h
DHRC1′	T1	1	0	9491	8862	126	1160	93.37	94.70	2.19h

Circuits GCD1′, Diffeq1′, Barcode1′, and DHRC1′ have inactive test pins but are structurally identical to GCD1, Diffeq1, Barcode1, and DHRC1, respectively.

coverages. However, the test application time is much shorter for the proposed approach. For example, Diffeq2 requires 868 test cycles, while Diffeq_fs with full scan requires more than 100,000 test cycles due to scanning of flip-flop values. Even though the performance is lowered for the testability-enhanced *Diffeq* circuits synthesized in this experiment, the implementation of the proposed technique requires virtually no area overhead. Thus, designers may take further optimization steps to improve the performance of the circuit without significant impact on circuit area.

While *GCD* and *Diffeq* each contain a single loop only, *Barcode* contains two nested loops. The inner loop makes at least 255 iterations each time it is invoked. With one test pin of type T1 controlling the exit condition of the inner loop, the system is able to return the *locus* to the outer loop prior to completion of the iterations. As shown in Table 3, the fault coverage is improved from 69.4% to 86.6%, and the number of aborted faults is greatly reduced. Adding a control point of type T2 allowing the *locus* to stay in the inner loop after the 255 iterations would not be helpful, since reaching the end of 255 iterations is already difficult for a gate-level ATPG. However, using two control points of type T3 to control two different decision nodes through exclusive-OR functions is effective. Furthermore, Barcode contains an embedded loop counter, and embedded counters are notorious for making any circuit hard to test at the gate level. Therefore, it may be necessary to make the loop counter directly controllable to get a high fault coverage. Thus, a control point of type T4 was added to load the counter directly from existing PI's. One of the two test pins added as type-T3 control points was also used as a type-T4 control point to minimize the number of test pins. Further improvements in fault coverage resulted, at the cost of an estimated 7.5% increase in area. The partial scan and full scan implementations had significantly higher fault coverages; therefore, additional gate-level DFT techniques may be necessary for this circuit.

Unlike *Barcode*, which has two nested loops, *DHRC* has two cascaded loops. The first loop initializes the memories in the design; then the second loop performs the computations. The high-level description of *DHRC* is a modified version of the *DHRC* benchmark, because the original description is incomplete. The problem size has been reduced so that logic synthesis can be done within a reasonable period of time. The design without testability enhancement is fairly testable. The use of a single control point of type T1 does not improve the fault coverage or test generation time, but the test set size is reduced. Addition of control points

of types T1 and T2 results in fewer aborted faults, shorter test generation time, and higher fault coverage. Use of two control points of type T3 (which requires only one test pin) combined with a control point of type T4 (to load the counter directly from existing PI's) results in further improvements in fault coverage; the fault coverage is nearly as high as the partial scan and full scan fault coverages, while the impact on area and performance is negligible. The proposed SFT approach may be a better choice than partial scan or full scan for this circuit due to the area and performance overheads of scan and the higher test application time.

Although various types of circuits are synthesized for the same design, one might argue that they differ in testability because they have different circuit structures. A further experiment was conducted to demonstrate the effect of the extra test inputs on circuits having the same structure. Using circuits enhanced with test pins of type T1 produced in the previous experiment, we tied the test pins to a constant value so that they are not functional during test generation and test application. In other words, the functions of these circuits are identical to the original circuits. The resulting circuits are named GCD1$'$, Diffeq1$'$, Barcode1$'$, and DHRC1$'$. We then performed test generation on the modified circuits. Results are shown in Table 3. The testability of circuits with inactive test pins is shown to be much lower than the testability of circuits with functional test pins. In most cases, the testability of circuits with inactive test pins is similar to the testability of circuits generated without test pins, i.e., their fault coverage, ATPG time, and test length are similar.

5 Conclusions

We have presented a high-level testability measure to evaluate the controllability of functional loops in a high-level circuit description. The proposed testability measure is first applied to identify hard-to-control nodes in the control-data flow graph. Then the proposed synthesis-for-testability technique adds controllability to these hard-to-control loops to assist the activation and propagation of faults during test generation and test application. Implementation of this approach does not utilize any scan design, and the logic synthesis can be performed by any high-level synthesis system. Experimental results on several high-level synthesis benchmarks show that circuits generated using this approach often require shorter ATPG times to produce test sets that achieve better fault coverage and fault efficiency. The test vectors can be applied at clock-speed, and the testability is improved at the cost of one or two extra input pins, while the area and performance overheads are minimal.

References

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

[2] S. Chiu and C. A. Papachristou, "A partial scan cost estimation method at the system level," *Proc. IEEE Int. Conf. Computer Design*, pp. 146–150, 1993.

[3] T. C. Lee, N. K. Jha, and W. H. Wolf, "A conditional resource sharing method for behavioral synthesis of highly testable data paths," *Proc. Int. Test Conf.*, pp. 744–753, 1993.

[4] M. Potkonjak, S. Dey, and R. K. Roy, "Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Trans. Computer-Aided Design*, vol. 14, no. 9, pp. 1141–1154, Sept. 1995.

[5] T. Thomas, P. Vishakantantaiah, and J. A. Abraham, "Impact of behavioral modifications for testability," *Proc. IEEE VLSI Test Symp.*, pp. 427–432, 1994.

[6] V. Chickermane, J. Lee, and J. H. Patel, "A comparative study of design for testability methods using high-level and gate-level descriptions," *Proc. Int. Conf. Computer-Aided Design*, pp. 620–624, 1992.

[7] S. Bhattacharya, F. Brglez, and S. Dey, "Transformations and resynthesis for testability of RT-level control-data path specifications," *IEEE Trans. VLSI Systems*, vol. 1, no. 3, pp. 304–318, Sept. 1993.

[8] H. Harmanani and C. Papachristou, "An improved method for RTL synthesis with testability tradeoffs," *Proc. Int. Conf. Computer-Aided Design*, pp. 30–35, 1993.

[9] S. Dey, M. Potkonjak, and R. Roy, "Exploiting hardware sharing in high-level synthesis for partial scan optimization," *Proc. Int. Conf. Computer-Aided Design*, pp. 20–25, 1993.

[10] S. Dey and M. Potkonjak, "Non-scan design-for-testability of RT-level data paths," *Proc. Int. Conf. Computer-Aided Design*, pp. 640–645, 1994.

[11] C. Chen, T. Karnik, and D. G. Saab, "Structural and behavioral synthesis for testability techniques," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 6, pp. 777–785, June 1994.

[12] T. C. Lee, W. H. Wolf, and N. K. Jha, "Behavioral synthesis for easy testability in data path scheduling," *Proc. Int. Conf. Computer-Aided Design*, pp. 616–619, 1992.

[13] F. Hsu and J. H. Patel, "A distance reduction approach to design for testability," *Proc. IEEE VLSI Test Symp.*, pp. 158–163, 1995.

[14] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," *Proc. Int. Test Conf.*, 1990, pp. 377–386.