A Controller-Based Design-for-Testability Technique for Controller-Data Path Circuits

Sujit Dey Vijay Gangaram Miodrag Potkonjak
C&C Research Laboratories, NEC USA
Prinecton, NJ 08540

ABSTRACT

This paper investigates the effect of the controller on the testability of sequential circuits composed of controllers and data paths. It is shown that even when both the controller and the data path parts are individually 100% testable, the composite circuit may not be easily testable by gate-level sequential ATPG. Analysis shows that a primary problem in test pattern generation of combined controller-data path circuits is the correlation of control signals due to implications imposed by the controller specification. A design-for-testability technique is developed to re-design the controller such that the implications which may produce conflicts during test pattern generation are eliminated. The DFT technique involves adding extra control vectors to the controller. Experimental results show the ability of the controller DFT technique to produce highly testable controller-data path circuits, with nominal hardware overhead.

I. INTRODUCTION

Several existing scan-based design-for-testability techniques use heuristics based on the topology of a circuit, like breaking all loops, except self-loops, and reduction of sequential depth, as ways to make sequential ATPG of circuits easy [4, 5, 14]. However, it has been observed that for many circuits, even when all loops are broken using scan FFs, and the sequential depth is low, the circuit remains difficult for sequential ATPG. This paper introduces a new DFT technique to supplement topology-based DFT techniques like loop-breaking, and sequential depth reduction. The proposed technique uses high-level design information, both topological and functional.

Circuits synthesized from behavioral specifications typically consist of controller(s) and data path(s). Besides techniques that modify the behavioral specification of a design to improve the testability of the resulting circuit [3, 8, 19], several behavioral and RT-level synthesis approaches have been proposed to generate easily testable data paths for both Built-In-Self-Test (BIST)-based testing methodology [1, 6, 13, 16], and Automatic Test Pattern Generation (ATPG) methods [9, 7, 15]. However, all the existing high level test synthesis techniques consider the testability of the data paths only. Either the existing techniques ignore the controller, or assume that the control signals coming to the data path are independently controllable. An exception is Genesis [2], but that system focuses on hierarchical test generation, and not gate-level sequential ATPG, which is the focus of this work.

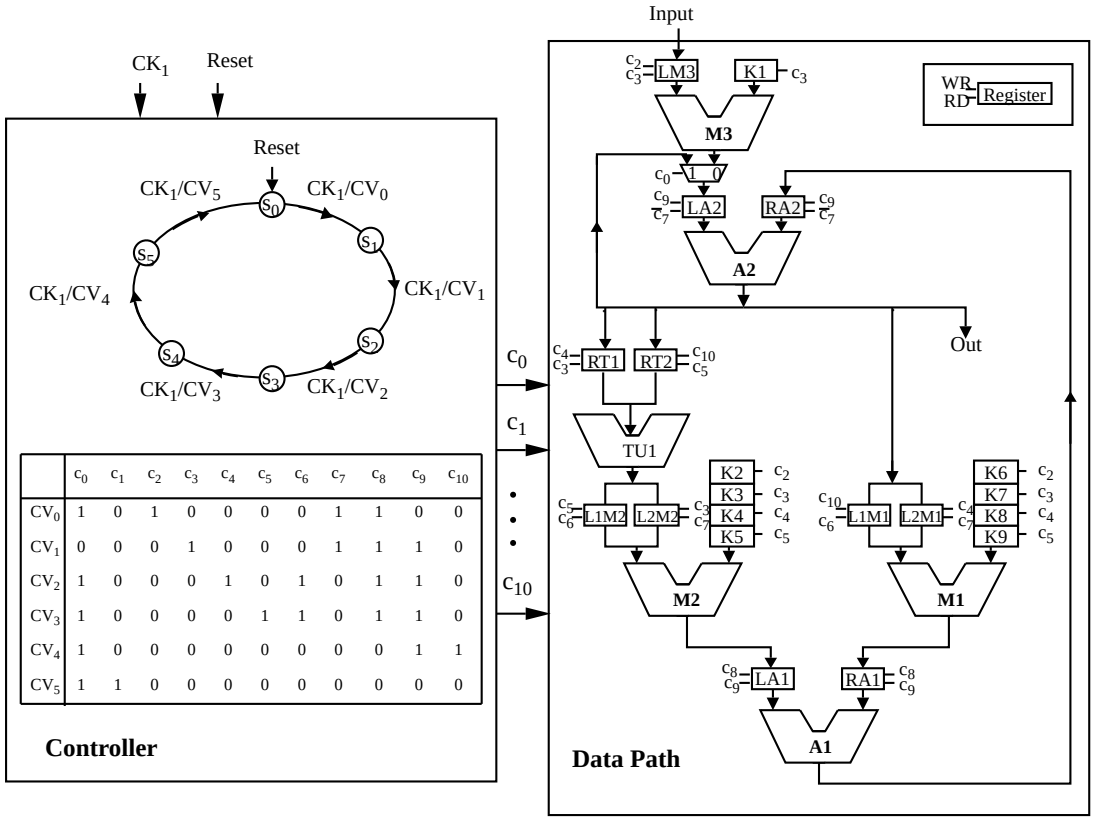
Even when the controller is made highly testable by using a gate-level DFT technique like [4, 14], and the data path is made highly testable by using either a gate-level or high-level DFT technique, a high test efficiency may not be achieved by gate-level sequential ATPG for the combined controller-data path circuit. In this paper, we address the problem of making a circuit composed of a controller and a data path easy-to-test for sequential ATPG, unlike the existing high-level techniques, which focus exclusively on data paths.

We study the effect of the controller on the testability of the data path. We observe that because of the correlation of the control signals due to implications imposed by the controller specification, the complete circuit may not have high test efficiency even when both the controller and data path are individually highly testable. We develop techniques to identify the control signal implications which may create conflicts during sequential ATPG. A technique is developed to re-design the controller such that the identified implications are eliminated. The technique involves adding a few extra control vectors to the existing control vectors which are outputs of the controller. The controller DFT technique has been applied to several controller-data path circuits. Experimental results show the ability of the proposed DFT technique to produce highly testable controller-data path circuits, with only marginal area overhead.

II. DFT OF CONTROLLER-DATA PATH CIRCUIT

Figure 1 shows a circuit composed of a controller and a data path. The data path implements the operations of the design, while the controller implements the schedule of the operations. The data path has a register-transfer (RT) level structural specification consisting of execution units (like adders, multipliers, ALUs, transfer units), registers, multiplexors, and interconnects. The controller has two inputs, Reset and CK1, while it has 11 outputs, the control signals $\{c_0, \ldots, c_{10}\}$. The control signals are inputs to the data path, and are used to control the various units of the data path. The state transition graph (STG) of the controller, and the control vectors $\{CV_0, \ldots, CV_5\}$, representing the outputs of the controller and expressed in terms of the control signals $\{c_0, \ldots, c_{10}\}$, are shown.

The controller-data path circuit has been synthesized using a behavioral test synthesis tool BETS [9] such that all loops in the data path, except self-loops, pass through the register RA2, and scanning RA2 breaks all loops. Assuming full and independent controllability of the control signals, as is done by the existing high level test synthesis techniques including BETS [9], the data path with the partial scan register RA2 is easily testable, as indicated by the row *Scan DFT - Datapath* in Table 1. The sequential ATPG program HITEC [17] can achieve almost 100% test efficiency on the scan data path. Similarly, after scanning all the FFs of the controller, the controller is highly testable, with both the fault coverage as well as the test efficiency being 100%.



	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
CV_0	1	0	1	0	0	0	0	1	1	0	0
CV_1	0	0	0	1	0	0	0	1	1	1	0
CV_2	1	0	0	0	1	0	1	0	1	1	0
CV_3	1	0	0	0	0	1	1	0	1	1	0
CV_4	1	0	0	0	0	0	0	0	0	1	1
CV_5	1	1	0	0	0	0	0	0	0	0	0

Figure 1: An example circuit IIR filter consisting of a controller and data path. The controller shows the state transition graph, and the control vectors composed of control signals

However, when the controller and the data path are actually put together, the independence assumption of the control signals is no more valid, and the test efficiency achieved for the full circuit is only about 84%, as shown by the row *Scan DFT - Contr + DP* in Table 1. Note that the composite controller-data path circuit, whose test efficiency is only about 84%, has all the non-self loops broken by scanning register RA2 and the controller FFs.

A. Conflicts Due to Implication of Control Signals

When the controller and data path shown in Figure 1 are considered together, conflicts may arise in justification and propagation of values during test pattern generation. These conflicts stem from the correlation of the control signals imposed by the controller specification.

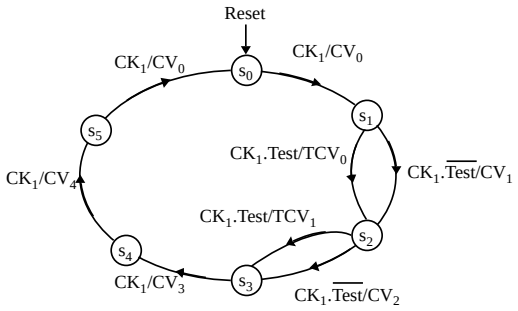
Let us consider justification of a value at the output of the execution unit A1. Let us say that in the next time frame, two different values will be required to be justified at the two registers LA1 and RA1. There exists a reconvergence from the output of A2 to the output of A1, leading to a possible conflict. However, since the paths from Out(A2) to LA1 pass through 2 registers, while the paths from Out(A2) to RA1 pass through 1 register, the fanout conflict can be avoided by requiring that in two successive time frames, two different values x and y are generated at the output of A2. It can be shown that satisfying the above condition requires simultaneously that the READ signal of register LA2, $\bar{c_7} = 1$, and the multiplexor signal and the LA2 WRITE signals $c_0 = 0$ and $c_9 = 1$.

However, due to the design requirements, the controller has been specified and synthesized such that the control signal $c_7 = 0$ implies that $c_0 = 1$, as can seen from the control vectors specification in Figure 1. Consequently, $\bar{c_7} = 1$ and $c_0 = 0$ cannot be satisfied simultaneously in the same clock cycle, leading to a justification conflict. In general, two control signals (c_i, c_j) are said to be *correlated* if there exists an implication $c_i \Rightarrow c_j$ in the controller specification. The control signal implications, and the consequent correlations between corresponding control signals, are responsible in producing conflicts during test pattern generation, leading to poor test efficiency even when scan DFT has broken all loops in the circuit, and the individual controller and data path parts are 100% testable.

B. Redesigning the Controller to Improve Testability

The correlations imposed by the implications of the control signals can be broken by adding extra control vectors to the controller. The state transition graph of the redesigned controller, and its table of control vectors in terms of the control signals, are shown in Figures 2(a) and (b) respectively. Two new control vectors, TCV_0 and TCV_1 have been added. The output functions of the controller, when in state 1 and 2 have been changed. In state 1, if $Test = 1$, the output of the controller is the vector TCV_0, else it remains CV_1. Similar change has been effected in state 2. Note that the "Test" pin is set to 0 in the functional mode, hence the controller modification preserves the functionality of the circuit.

The control vector TCV_0 has been designed such that it breaks the implication $\bar{c_7} \Rightarrow c_0$, by having $c_7 = 0$ and $c_0 = 0$ simultaneously in the same vector. The two test control vectors eliminates a number of other control signal implications that would most likely produce conflicts during test pattern generation. After the controller DFT,



(a)

(b)

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c8	c_9	c_{10}
CV_0	1	0	1	0	0	0	0	1	1	0	0
CV_1	0	0	0	1	0	0	0	1	1	1	0
CV_2	1	0	0	0	1	0	1	0	1	1	0
CV_3	1	0	0	0	0	1	1	0	1	1	0
CV_4	1	0	0	0	0	0	0	0	0	1	1
CV_5	1	1	0	0	0	0	0	0	0	0	0
TCV_0	0	0	1	1	0	0	1	0	1	1	0
TCV_1	1	0	0	1	1	1	1	1	1	1	1

Figure 2: The controller **after DFT**: (a) state transition graph, (b) control vectors

again all the FFs of the controller are scanned. The data path remains the same, with register RA2 scanned. The composite controller-data path circuit becomes significantly more testable than the composite circuit with the same scan FFs but no controller DFT. The last row *Contr + DP* in Table 1 shows the results of running Hitec on the circuit containing the controller with controller DFT. A very high test efficiency (99.7%) can be achieved for the circuit derived by controller DFT + scan DFT, as opposed to 83.7% test efficiency achieved for the controller-data path circuit derived by just scan DFT.

In the next two sections, we describe the two phases of the controller-based DFT technique. The first phase identifies which control signal implications/correlations should be eliminated. The second phase derives a minimal set of control vectors to eliminate the identified implications/correlations. The control vectors are added to the controller so that they can be activated only in the test mode, thus preserving the functionality of the design.

III. Identifying Which Control Signal Implications to Eliminate

As explained in the previous section, control signal implications may cause conflicts during test pattern generation, even in the absence of topological features which have been traditionally identified to be problematic for ATPG, like reconvergences, loops, and sequential depth. In this section, we first show how control signal implications can be extracted from the given controller specification. Since there are typically a large number of such implications, trying to eliminate all of them can be very costly. We show how to identify a small subset of the implications, termed inhibiting implications, that need to be eliminated to facilitate easy sequential ATPG.

A. Control Signal Implications

Given the controller specification, we first form the control vector table shown in Figure 1. Next, we extract all the implications of each control signal, c_i and $\overline{c_i}$. Consider the controller specification and the corresponding control vector table shown in Figure 1. Let us find the implications of the control signal c_7 and $\overline{c_7}$. $(c_7 = 0)$ implies $(c_0 = 1), (c_2 = 0)$ and $(c_3 = 0)$. Hence, $\overline{c_7} \Rightarrow c_0, \overline{c_2}, \overline{c_3}$.
Similarly, $c_7 \Rightarrow \overline{c_1}, \overline{c_4}, \overline{c_5}, \overline{c_6}, c_8, \overline{c_{10}}$.
The implications of the other control signals are as follows:
$\overline{c_0} \Rightarrow \overline{c_1}, \overline{c_2}, c_3, \overline{c_4}, \overline{c_5}, \overline{c_6}, c_7, c_8, c_9, \overline{c_{10}}$
$c_2 \Rightarrow c_0, \overline{c_1}, \overline{c_3}, \overline{c_4}, \overline{c_5}, \overline{c_6}, c_7, c_8, \overline{c_9}, \overline{c_{10}}$
$c_3 \Rightarrow \overline{c_0}, \overline{c_1}, \overline{c_2}, \overline{c_4}, \overline{c_5}, \overline{c_6}, \overline{c_7}, \overline{c_8}, \overline{c_9}, \overline{c_{10}}$
$c_4 \Rightarrow c_0, \overline{c_1}, \overline{c_2}, \overline{c_3}, \overline{c_5}, c_6, \overline{c_7}, c_8, c_9, \overline{c_{10}}$
$c_5 \Rightarrow c_0, \overline{c_1}, \overline{c_2}, \overline{c_3}, \overline{c_4}, c_6, \overline{c_7}, \overline{c_8}, c_9, \overline{c_{10}}$
$c_6 \Rightarrow c_0, \overline{c_1}, \overline{c_2}, \overline{c_3}, \overline{c_7}, c_8, c_9, \overline{c_{10}}$
$c_8 \Rightarrow \overline{c_1}, \overline{c_{10}}$
$c_9 \Rightarrow \overline{c_1}, \overline{c_2}$
$c_{10} \Rightarrow c_0, \overline{c_1}, \overline{c_2}, \overline{c_3}, \overline{c_4}, \overline{c_5}, \overline{c_6}, \overline{c_7}, \overline{c_8}, c_9$

The implications of each control signal are stored as an implication graph, shown in Figure 3(b) for the control signal $\overline{c_7}$.

B. Topological Relations of Control Signals in Data Path

Since the number of control signal implications is usually very large, a large number of control vectors may have to be added to eliminate all the implications, making the controller very expensive as well as difficult to test. Using structural information about how control signals control the different modules in the data path, we derive information regarding which pairs of control signals can affect the flow of data in k time-frames, for an user specified k. We will eventually consider eliminating implications between those pairs of control signals which affect data flow directly over k time-frames, and hence can lead to conflicts during ATPG.

Each control signal c_i controls the flow of data through one or more modules in the data path like a multiplexor and/or a register. We say that control signal c_j is *k time-frame module dependent* on control signal c_i if a module controlled by c_j has a path of no more than k time frames from a module controlled by c_i. For each control signal c_i, we construct a k-time-frame module dependency graph as follows. Initially, the graph has only one node c_i. A node c_j and a directed edge from node c_i to node c_j are added to the graph if control signal c_j is k time-frame module dependent on control signal c_i. This is determined by traversing the data path from all modules controlled by c_i, and including the corresponding control signals of all modules reached in k time-frames.

The 2-time-frame module dependency graph for the control signal $\overline{c_7}$ is shown in Figure 3(a). For each edge $(\overline{c_7}, c_j)$ in the graph, there is a path of not going across more than 2 time frames from the register controlled by $\overline{c_7}$ to a module controlled by signal c_j. For instance, there is a path of 1 time frame from LA2 controlled by $\overline{c_7}$, and RT1 controlled by c_4, hence the edge $(\overline{c_7}, c_4)$ in Figure 3(a).

C. Forming the Dependency-Implication Graphs

For a control signal c_i given its k-time-frame module dependency graph, and its control implication graph, a new graph, the *dependency-implication graph*, is formed which consist of those edges which are common to both the k-time-frame module dependency graph and the control implications graph of signal c_i. The formation of the dependency-implication graph for control signal $\overline{c_7}$ is shown in Figure

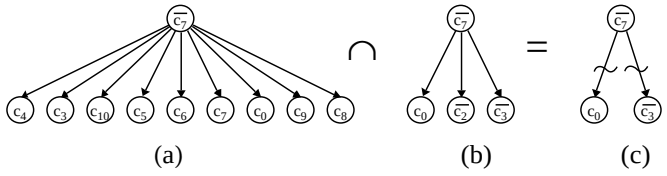


Figure 3: Derivation of Dependency-Implication graph for control signal $\overline{c_7}$: (a) 2-time-frame module dependency graph for $\overline{c_7}$, (b) control signal implications of $\overline{c_7}$, (c) Dependency-Implication graph for $\overline{c_7}$

3(c). Each edge in the dependency-implication graph, (c_i, c_j), correspond to an implication $c_i \Rightarrow c_j$ which exists due to the controller specification, and may affect data flow in the circuit since the module controlled by c_j is dependent on module controlled by c_i. Hence, each implication in the dependency-implication graph can potentially cause a conflict during ATPG, and hence is a candidate for removal by the DFT technique described later. The dependency-implication graphs of the control signals of the controller in Figure 1 are shown in Figure 4.

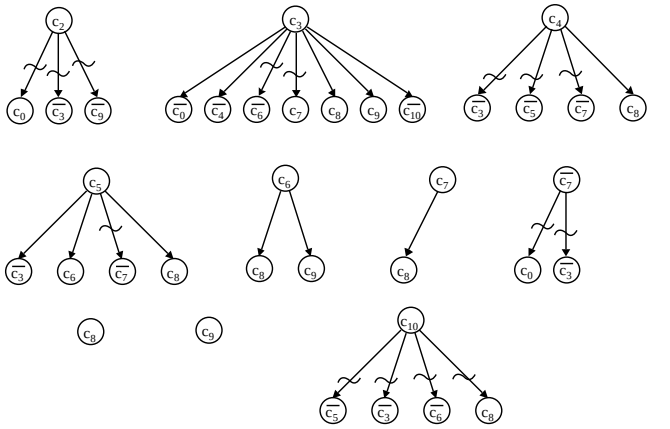


Figure 4: Dependency-Implication graphs for the control signals in Figure 1. A dashed edge denotes an inhibiting implication.

D. Identifying Inhibiting Implications to Eliminate

Even after the dependency-implication graphs have been formed for each control signals, there are many implications to possibly break. However, eliminating some of the implications in the dependency-implication graphs will not help in justification/propagation of values. As an example, consider the implication $(c_3 \Rightarrow \overline{c_0})$ in the dependency-implication graph of c_3 in Figure 4. The consequence of the implication on the data path is that when in a particular clock cycle, the value of register LM3 is read, the result of the multiplication in M3 is passed on through the multiplexor to be stored in LA2. This implication will not lead to any conflict during the justification/propagation of values, and hence is not needed to be broken to reduce conflicts during ATPG.

On the other hand, some implications may potentially cause conflicts during ATPG, like the implication $(\overline{c_7} \Rightarrow c_0)$ shown in Section A. We use a heuristic to identify implications that may cause conflicts. We say an implication is an *inhibiting implication* if it prevents a desired action from taking place. We choose to eliminate only the inhibiting implications in the dependency-implication graphs.

Consider the dependency-implication graph shown in Figure 4. The implication $(\overline{c_7} \Rightarrow c_0)$ is inhibiting because it prevents data from M3 to go through the multiplexor to LA2 in the same time frame that data is read from LA2. Similarly, the implication $(c_2 \Rightarrow \overline{c_9})$ prevents data to be written to register LA2 in the same time frame that data is being written to LM3. The implication $(c_3 \Rightarrow \overline{c_6})$ prevents data from being read from register L1M2 in the same time frame as data is read from register RT1. The above implications which restrict/prevent data flow are termed inhibiting implications. The inhibiting implications for the example controller-data path circuit of the IIR filter are shown marked in the dependency-implication graphs in Figure 4.

The restrictions imposed by the inhibiting implications are due to the design functionality. However, they will reduce concurrency during ATPG, and may lead to conflicts. Hence, each inhibiting implication is selected for removal during the next step of the DFT technique.

IV. Eliminating Control Implications by Adding Test Control Vectors

The second phase of the proposed DFT technique is to add control vectors to the controller specification such that the inhibiting implications identified in the first phase are eliminated. In this section, we show how to derive such control vectors, termed test control vectors (TCV), as they will be generated by the controller only in the test mode. We provide a technique to merge the test control vectors to minimize the number of test control vectors needed, so as to minimize the DFT overhead. We outline the process of augmenting the controller with the test control vectors such that the functionality of the controller, and hence the whole design, is preserved.

A. Creating Test Control Vectors

For each control signal which has inhibiting implications that need to be eliminated, a test control vector is created to eliminate the inhibiting implications. An implication $(c_i = x) \Rightarrow (c_j = y)$, $x, y \in \{0, 1\}$, can be eliminated by adding a new control vector which has $c_i = x$ and $c_j = \overline{y}$. In general, for all the inhibiting implications of control signal c_i, $\{(c_i = x) \Rightarrow (c_j = y), \ldots, (c_k = z)\}$, a test control vector is formed with $c_i = x$, and each implied signal $c_j, \ldots, c_k$ having the complement of the value implied, that is, $(c_j = \overline{y}), \ldots, (c_k = \overline{z})$.

Consider the inhibiting implications from control signal c_2, denoted by dashed edges in Figure 4. The implications $c_2 \Rightarrow c_0, \overline{c_3}, \overline{c_9}$, due to the existing control vectors, can be eliminated by adding a new control vector $T_1 : c_2 = 1, c_0 = 0, c_3 = 1, c_9 = 1$ to the controller. Similarly, a new test control vector $T_2 : c_3 = 1, c_6 = 1, c_7 = 0$ will eliminate the inhibiting implications from c_3. Figure 5 lists the test control vectors $T_1, \ldots, T_6$ created to break the inhibiting implications marked in Figure 4 for each control signal.

B. Minimizing Number of Test Control Vectors

Two test control vectors are defined to be *compatible* if and only if for each control signal c_k that is defined in both the control vectors, the value of c_k is identical in both the control vectors. A pair of compatible control vectors can be merged into a single vector while still eliminating the implications that each control vector was created to eliminate. For example, the vectors T_1 and T_2 in Figure 5 are compatible because the only control signal existing in both T_1 and T_2, c_3, has the same value 1 in both the vectors. Similarly, the test control vector T_5 is compatible with both T_1 and T_2, and hence all

To Break Implications from	TCV	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
c_2	T_1	0		1	1						1	
c_3	T_2				1			1	0			
c_4	T_3				1	1			1			
c_5	T_4						1		1			
$\bar{c}_7$	T_5	0			1				0			
c_{10}	T_6				1			1		1		1
$c_2, c_3, \bar{c}_7$	TCV_0	0	0	1	1	0	0	1	0	1	1	0
c_4, c_5, c_{10}	TCV_1	1	0	0	1	1	1	1	1	1	1	1

Figure 5: Test control vectors to break inhibiting implications shown in Figure 4

three vectors can be merged to form a single test control vector, TCV_0, shown in Figure 5. Note that TCV_0 eliminates all the implications from c_2, c_3, $\bar{c_7}$ that the original TCVs T_1, T_2, and T_5 eliminate.

The problem of merging the test control vectors to produce a minimal number of test control vectors can be solved by mapping the problem to the well known problem of graph clique-partitioning. We begin by defining the clique-partitioning problem. A graph is complete if and only if every pair of its nodes has an edge connecting them. A clique of a graph G is a complete subgraph of G. The problem of clique partitioning is to partition a graph into a minimal number of cliques such that each node belongs to exactly one clique. The clique partitioning problem is NP-Complete for partitioning into 3 or more cliques [11].

Given a set of TCVs, we construct a compatibility graph, where each node corresponds to a test control vector, and there is an edge (T_i, T_j) if the two vectors T_i and T_j are compatible. A solution to the minimal clique partitioning problem of the compatibility graph gives a solution to the problem of finding a minimal set of test control vectors. Partitioning the compatibility graph into a minimal number of cliques $\{K_i\}$, and merging the test control vectors corresponding to nodes of clique K_i to form the test control vector TCV_i, results in a minimal set of test control vectors $\{TCV_i\}$, with the number of test control vectors equal to the number of cliques. Note that the number of TCVs to be minimized is bounded by the number of control signals, which is in the order of the number of modules (registers, multiplexors, execution units) in the data path. Hence the number of nodes in the compatibility graph is usually small. We use the clique partitioning algorithm in [20], which has a running time complexity of $O(n^2)$ for a compatibility graph with n nodes, and guarantees an optimal solution with a very high probability.

Figure 6(a) shows the compatibility graph for the set of test control vectors $\{T_1, \ldots, T_6\}$ listed in Figure 5. A possible solution to the minimal clique partitioning problem is illustrated in Figure 6(b), which shows two cliques, $K_0 = \{T_1, T_2, T_5\}$, and $K_1 = \{T_3, T_4, T_6\}$, covering all the nodes of the graph. Hence, the six TCVs $\{T_1, \ldots, T_6\}$ can be merged into only two TCVs: $TCV_0 = (T_1 \cup T_2 \cup T_5)$, and $TCV_1 = (T_3 \cup T_4 \cup T_6)$, as shown in Figure 5.

C. Adding Control Vectors to Controller

After deriving the new test control vectors to eliminate the inhibiting implications, and hence the undesirable control signal correlations, the new vectors are added to the controller specification in a way that they are generated only in the test mode, hence preserving the behavior of the circuit. This is achieved by using a new "Test" pin, which

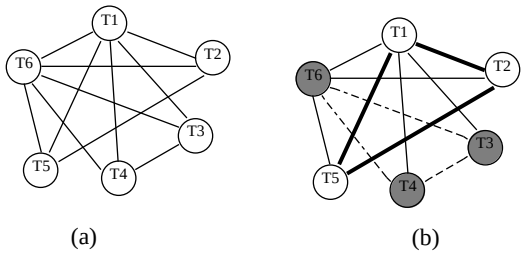


Figure 6: (a) Compatibility graph of the test control vectors, (b) a two-clique partition

is set to 0 in the functional mode. Also, to minimize area overhead, a test control vector TCV_i is added to a state with control vector CV_j such that TCV_i and CV_j have the minimum distance, that is the minimum number of control signals which have a value x in TCV_i and a different value $\bar{x}$ in CV_j. This would ensure the minimal change in minterms of the output function of any control signal c_k after the addition of the new test control vectors.

The state transition graph, and a table of control vectors, of the redesigned controller derived from the original controller are shown in Figures 2(a) and (b) respectively. The two test control vectors TCV_0 and TCV_1, derived in the previous section, are added as follows. TCV_0 has a minimum distance with CV_1 (= 3), and hence TCV_0 is added to state 1 with CV_1. If $Test = 1$, the output of the controller is the new vector TCV_0, else it remains CV_1. The test control vector TCV_1 is similarly added to state 2, since its distance with vector CV_2 is minimum. Since the "Test" pin is set to 0 in the functional mode, the functionality of the circuit is preserved.

V. Experimental Results

We applied the proposed controller-based DFT technique, consisting of the two phases described in Sections III and IV, on several controller-data path circuits, synthesized using the behavioral test synthesis system BETS [9] from behavioral descriptions [12]. In this section, we report the results obtained for the following circuits implementing: (1) a 4th order IIR filter with a word size of 8 bits (4IIR.8), (2) a 4th order IIR filter with a word size of 14 bits (4IIR.14), (2) a Speech filter of word size 12 bits (Speech.12), and (5) a Wave Digital filter of word size 8 bits (WDF.8). The designs generated by BETS consist of a structural VHDL specification of the data path and a functional VHDL specification of the controller. The proposed DFT technique modifies the functional VHDL specification of the controller; the data path specification remains unchanged. The VHDL specifications are synthesized to gate-level circuits using OASIS [10]; one-hot encoding is used to synthesize the controller.

The results of applying scan-based DFT and the proposed controller-based DFT techniques on the different design examples are reported in the Tables 1, 2, 3, and 4. The Tables show the different kind of circuits for each design (column *Circuit*), the DFT technique used to derive the circuits (column *DFT*), and the area in number of transistor pairs (column *Area*) after technology mapping using the SIS technology mapper [18], and the *lib2.genlib* standard cell library.

The rows *Controller*, *Datapath*, and *Contr + DP* refer to the controller circuit, the data path circuit, and the combined controller-data path circuit for each design. The rows under **Scan DFT** show the circuits with scan FFs used as follows. All the FFs of the controller are scanned. For the data path circuit, the scan FFs correspond to the

Circuit	DFT	Area	Faults		FC%	TE%	Vec	Tgen
			Total	Abt				(secs)
Scan DFT								
Controller	6 scan/6 FFs	186	279	0	100.00	100.00	68	0.7
Datapath	8 scan/88 FFs	3300	4572	12	94.72	99.73	399	1207.8
Contr + DP	14 scan/94 FFs	3448	4805	785	77.19	*83.66*	210	17917.0
Scan DFT + Controller DFT								
Contr + DP	14 scan/94 FFs + 2 CVs	3502	4846	14	93.60	*99.71*	504	933.8

Table 1: 4IIR.8: Effect of scan-based and controller-based DFT schemes

Circuit	DFT	Area	Faults		FC%	TE%	Vec	Tgen
			Total	Abt				(secs)
Scan DFT								
Controller	6 scan/6 FFs	186	279	0	100.00	100.00	68	0.7
Datapath	14 scan/154 FFs	6912	9767	220	93.76	97.75	463	8433.8
Contr + DP	20 scan/160 FFs	7099	9996	1222	82.28	*87.78*	349	31890.9
Scan DFT + Controller DFT								
Contr + DP	20 scan/160 FFs + 2 CVs	7109	10027	42	94.43	*99.58*	807	3263.1

Table 2: 4IIR.14: Effect of scan-based and controller-based DFT schemes

Circuit	DFT	Area	Faults		FC%	TE%	Vec	Tgen
			Total	Abt				(secs)
Scan DFT								
Controller	6 scan/6 FFs	88	175	0	100.00	100.00	38	0.4
Datapath	12 scan/132 FFs	5294	7443	342	91.28	95.48	200	1014.2
Contr + DP	18 scan/138 FFs	5379	7535	782	83.79	*89.62*	332	2622.0
Scan DFT + Controller DFT								
Contr + DP	18 scan/138 FFs + 2 CVs	5441	7610	103	93.36	*98.65*	402	2271.8

Table 3: Speech.12: Effect of scan-based and controller-based DFT schemes

Circuit	DFT	Area	Faults		FC%	TE%	Vec	Tgen
			Total	Abt				(secs)
Scan DFT								
Controller	7 scan/7 FFs	63	135	0	100.00	100.00	42	0.2
Datapath	None	3586	4689	15	94,62	99.68	578	218.2
Contr + DP	7 scan/119 FFs	3638	4802	308	87.69	*93.54*	780	1569.9
Scan DFT + Controller DFT								
Contr + DP	7 scan/119 FFs + 2 CVs	3673	4849	69	93.81	*98.57*	1577	705.4

Table 4: WDF.8: Effect of scan-based and controller-based DFT schemes

registers selected by BETS as scan registers, and are the minimum number of FFs needed to break all non-self loops by gate-level partial scan tools [4, 14, 5]. The scan Control+DP circuit is obtained by just putting together the full scan controller and the partial scan data path, and hence does not contain any non-self loops. The scan FFs, and the total number of FFs, used by the three circuits are indicated in the column DFT. For instance, for the design example 4IIR.8, the DFT column shows that all 6 FFs of the controller are scanned, while 8 out of the 88 FFs used in the data path are scanned. The scan Contr+DP circuit has a total of 14 scan FFs out of a total of 94 FFs.

The application of the proposed controller-based technique on the scan Contr+DP circuit is shown under **Scan DFT + Controller DFT**. The *Contr + DP* row shows the same scan data path, with the controller redesigned by adding test control vectors, synthesized using OASIS [10] in exactly the same way as the original controller, and scanning all the FFs of the controller. Note that the controller redesign technique does not increase the number of states, and hence the number of FFs, of the controller. The scan FFs used by the redesigned Contr+DP circuit is the same as the scan Contr+DP circuit. The only difference is the test control vectors added to the redesigned controller. For the IIR.8 example, the column DFT shows that the number of scan FFs remains the same at 14, and two test control vectors are added to the controller, as shown in Figure 2.

The tables show the result of running the sequential test pattern generator HITEC [17] on the circuits in column *Circuit* for each design example. The total number of faults (*Total*) and the number of faults aborted (*Abt*) by Hitec are shown. A fault is aborted if either a backtrack limit of 100000 or a CPU limit of 20 seconds has been exceeded. Columns *FC%* and *TE%* show the percentage fault coverage, and the percentage test efficiency achieved by HITEC. The number of test vectors needed (*Vec*), and the test generation time taken (*Tgen*) in CPU secs on a Sparc10 are also reported.

The tables show that the use of scan DFT can make the controller and data path circuits highly testable, when considered separately. However, for the combined controller-data path circuit, the test efficiency deteriorates significantly from that can be achieved for the controller and data path individually. For example, for the IIR.8 example, the full scan controller has 100% fault coverage as well as 100% test efficiency. The data path using 8 scan FFs (to break all loops) has close to 100% test efficiency. However, the combined controller-data path circuit has only 83.7% test efficiency.

The experimental results show the effectiveness of the controller-based DFT technique to significantly improve the testability of the scan controller-data path circuits. When the controller is redesigned by adding the test control vectors by the proposed technique, followed by full scan of the control FFs, the redesigned controller-data path circuit becomes significantly more testable than the scan Contr+DP circuit. For the IIR.8 example, Hitec could achieve close to 100% test efficiency for the redesigned controller-data path circuit (last row of Table 1), up from 83.7%. Similarly, the controller DFT technique could enhance the test efficiency of other scan Cntr+DP circuits: from 87.8% to 99.6% for 4IIR.14, from 89.6% to 98.7% for Speech.12, and from 93.5% to 98.6% for WDF.8.

The proposed technique does not have routing overhead associated with scan clock and scan chain, or routing inputs/outputs to/from test points, associated with other scan and non-scan DFT schemes. Experimental results show that the active area overhead due to adding the test control vectors, measured by the increase in number of transistor pairs, is also very low. For the IIR.8 example, the controller DFT increases the number of transistor pairs from 3448 to 3502, which represents a marginal area overhead of 1.57%. The average active area overhead for the circuits reported is only 1.03%

VI. Conclusions

We have presented a new controller-based DFT technique to facilitate ATPG of sequential circuits consisting of controller and data path specifications. The proposed DFT technique uses both the functional specification of the controller, and the structural specification of the data path, to derive a few test control vectors which are added to the controller to offset control signal correlations that can cause conflicts during ATPG. We have demonstrated application of the technique, with nominal area overhead, to significantly improve the testability of controller-data path circuits.

References

[1] L. Avra and E. McCluskey. Synthesizing for Scan Dependence in Built-In Self-Testable Designs. In *Proceedings of the International Test Conference*, pages 734 – 743, October 1993.

[2] S. Bhatia and N. K. Jha. Genesis: A Behavioral Synthesis System for Hierarchical Testability. In *Proc. of the European Design and Test Conference*, Feb 1994.

[3] C.-H. Chen, T. Karnik, and D.G.Saab. Structural and Behavioral Synthesis for Testability Techniques. *IEEE Transactions on Computer-Aided Design*, 13(6):777–785, June 1994.

[4] K.T. Cheng and V.D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Transactions on Computers*, 39(4):544 – 548, April 1990.

[5] V. Chickermane and J. H. Patel. An Optimization Based Approach to the Partial Scan Design Problem. In *Proceedings of the International Test Conference*, pages 377 – 386, September 1990.

[6] S. S. K. Chiu and C. Papachristou. A Built-In Self-Testing Approach For Minimizing Hardware Overhead. In *Proceedings of the International Conference on Computer Design*, 1991.

[7] S. Dey and M. Potkonjak. Non-Scan Design-for-Testability of RT-Level Data Paths. In *Proceedings of the International Conference on Computer-Aided Design*, pages 640 – 645, November 1994.

[8] S. Dey and M. Potkonjak. Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs. In *Proceedings of the International Test Conference*, October 1994.

[9] S. Dey, M. Potkonjak, and R. Roy. Exploiting Hardware Sharing in High Level Synthesis for Partial Scan Optimization. In *Proceedings of the International Conference on Computer-Aided Design*, pages 20 – 25, November 1993.

[10] K. Kozminski (ed.). *OASIS Users Guide*. MCNC, MCNC, Research Triangle Park, N.C. 27709, 1991.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, 1979.

[12] R.A. Haddad and T.W. Parsons. *Digital Signal Processing: Theory, Applications and Hardware*. Computer Science Press, New York, NY, 1991.

[13] I.G. Harris and A. Orailoğlu. Microarchitectural Synthesis of VLSI Designs with High Test Concurrency. In *Proc. Design Automation Conference*, pages 206–211, June 1994.

[14] D.H. Lee and S.M. Reddy. On Determining Scan Flip-Flops in Partial-Scan Designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 322 – 325, November 1990.

[15] T. C. Lee, N. K. Jha, and W. H. Wolf. Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and Partial Scan Environments. In *Proc. Design Automation Conf.*, pages 292–297, 1993.

[16] N. Mukherjee, M. Kassab, J. Rajski, and J. Tsyzer. Arithmetic Built-In Self Test for High-Level Synthesis. In *Proc. of the 13th IEEE VLSI Test Symposium*, April 1995.

[17] T. M. Niermann and J. H. Patel. HITEC: A Test Generation Package for Sequential Circuits. In *Proc. EDAC*, pages 214–218, 1991.

[18] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design using Synthesis and Optimization. In *Proceedings of the International Conference on Computer Design*, October 1992.

[19] T. Thomas, P. Vishakantaiah, and J.A. Abraham. Impact of Behavioral Modifications For Testability. In *Proceedings of the 12th IEEE VLSI Test Symposium*, pages 427–432, April 1994.

[20] J.S. Turner. Almost All k-Colorable Graphs Are Easy to Color. *Journal of Algorithms*, 9(1):63 – 82, 1988.