Complexity of Sequential ATPG

Thomas E. Marchok[1], Aiman El-Maleh[2], Wojciech Maly[1], Janusz Rajski[2]

[1] ECE Department, Carnegie Mellon University, Pittsburgh, PA 15213
[2] MACS Laboratory, McGill University, Montreal, Canada - H3A 2A7

Abstract

The research reported in this paper was conducted to identify those attributes, of both sequential circuits and structural, sequential automatic test pattern generation (ATPG) algorithms, which can lead to extremely high test generation times. The retiming transformation is used as a mechanism to create two classes of circuits which present varying degrees of complexity for test generation. It was observed for three different sequential test generators that the increase in complexity of testing is not due to those circuit attributes (namely sequential depth and cycles) which have traditionally been associated with such complexity. Evidence is instead provided that another circuit attribute, termed *density of encoding*, is a key indicator of the complexity of structural, sequential ATPG.

1. Introduction

Despite its maturity, the testing of VLSI circuits must still be viewed as an arena with a number of unsolved problems. One of these is a lack of understanding of which attributes of a sequential circuit lead to long test generation times (and therefore high test generation costs). Without such understanding, it is difficult for designers to determine whether or not design for testability (DFT) techniques need to be employed in order to meet target cost, quality, or time to market constraints. This often leads to both high testing costs and inadequate test quality.

The purpose of the research described in this paper was to identify those circuit attributes which influence the complexity of structural automatic test pattern generation (ATPG) for sequential circuits. This study focuses on control logic, because it is in practice the most difficult type of sequential logic to test. This study does not consider data path logic, which is typically more easily testable.

This work has been supported by the Semiconductor Research Corp under contract no. 94-DC-068, Intel, a Cooperative Research and Development Grant from the "National Sciences and Engineering Research Council of Canada", Bell Northern Research, and a scholarship from the Quebec "Fonds pour la Formation de Chercheurs et l'Aide a la Recherche".

Our methodology of investigation was simple. First, we found a circuit transformation which strongly impacts ATPG for sequential circuits. This transformation is retiming ([1],[2]), which as has already been reported [3], causes a (sometimes drastic - more than two orders of magnitude) increase in test generation time and a decrease in the resultant test quality. Hence retiming was used to create two classes of circuits which present varying degrees of complexity for test generation. Results of ATPG experiments were then correlated to differences between the attributes of the two classes of circuits.

The results of our investigations are presented in the following way. Section 2 provides a detailed description of the test generation experiments and the circuits used. Section 3 interprets these results. Section 4 introduces important theoretical apparatuses which are used to analyze the experimental results. Section 5 then addresses the main topic of this paper, which is to identify those circuit attributes which influence the complexity of structural, sequential ATPG.

2. Impact of retiming on testability

The objective of the experiments was to measure the increase in the required ATPG CPU time caused by retiming. In the experiments three different structural, sequential ATPGs - HITEC [4], Attest [5], and Sequential EST (SEST) [6] have been used. The performance of each ATPG was measured according to the levels of fault coverage and fault efficiency attained, and the required CPU time. Among the three test generators used the emphasis was placed on HITEC, while Attest and SEST were used to confirm the HITEC based findings, and not for the purpose of performance comparison. Details of the experiments, which consumed a total of more than 5000 CPU hours, are as follows.

2.1. Circuits used

Circuits synthesized from the MCNC finite state machine (FSM) benchmarks using the SIS sequential synthesis tool

[7] have been used as a basis in this study. Table 1 lists the number of primary inputs, primary outputs, and states for the FSM descriptions used to synthesize the circuits reported here. The versions of dk16, pma, scf, and s510 used employ an explicit reset line. The reset line circumvents the well documented problem which structural ATPGs commonly have with circuit initialization [8].

Table 1: Finite state machines used to synthesize circuits.

FSM	PI	PO	states
dk16	3	3	27
pma	7	8	24
s510	20	7	47
s820	18	19	25
s832	18	19	25
scf	27	54	121

The SIS command sequence employed to synthesize the circuits followed that suggested in the SIS documentation [9]. The *stamina* tool was first used to perform state minimization, followed by *jedi* to perform state assignment (using the minimum number of state encoding bits). The *extract_seq_dc* command was used to extract the unreachable states and store them as external don't cares. This information was used by one of two scripts distributed with SIS (*script.rugged* or *script.delay*) to synthesize the network to implement the next state and primary output functions. The networks were mapped onto a version of the *mcnc.genlib* gate library which had been modified to contain only those gate types recognized by the sequential ATPGs used to generate test sets for the circuits. The *retime* command was then used to create a retimed version of each circuit. Multiple (non-retimed) circuits were created from the same FSM description by using combinations of different *jedi* state encoding algorithms and logic synthesis scripts. Each circuit attained a different area versus delay trade-off. The name of each circuit contains multiple fields which reflect the synthesis options employed. The *.j* field denotes the *jedi* state assignment encoding algorithm used: *.jo* represents the output dominant algorithm, *.ji* the input dominant algorithm, and *.jc* a combination of the input and output dominant algorithms. The *.s* field denotes the SIS script used for logic synthesis: *.sd* indicates *script.delay,* and *.sr* denotes *script.rugged*. The presence of a *.re* field indicates the circuit is a retimed version of the correspondingly named original circuit.

2.2. HITEC results

Table 2 lists the results of the HITEC ATPG on the synthesized circuits. The columns labeled *#DFF* lists the number of edge triggered DFFs in that circuit. *#CPU seconds* reports the number of DECstation 3100 CPU seconds which HITEC required to attain the levels of fault coverage and fault efficiency reported in the columns labeled *%FC* and *%FE*, respectively, on sequential (non-scan) versions of each circuit. Fault coverage is defined as the percentage of faults which were detected. Fault efficiency is defined as the percentage of faults which were either detected or labelled redundant. The column titled *CPU ratio* lists the ratio of the CPU execution times between the retimed and corresponding original circuit.

Table 2: HITEC ATPG results.

circuit	#DFF	%FC	%FE	# CPU seconds	CPU ratio
dk16.ji.sd	5	99.8	100.0	308	323.1
dk16.ji.sd.re	19	99.7	100.0	99529	
pma.jo.sd	5	99.4	100.0	791	231.5
pma.jo.sd.re	21	98.8	99.3	183145	
s510.jc.sd	6	98.2	100.0	24507	16.6
s510.jc.sd.re	20	95.3	96.0	405630	
s510.jc.sr	6	94.3	99.3	43060	9.6
s510.jc.sr.re	26	53.9	54.6	415021	
s510.ji.sd	6	99.2	100.0	2918	56.6
s510.ji.sd.re	11	98.8	99.6	165190	
s510.ji.sr	6	98.9	100.0	12460	27.6
s510.ji.sr.re	23	91.4	92.0	343420	
s510.jo.sr	6	96.2	100.0	3822	261.6
s510.jo.sr.re	28	56.5	57.0	1000000	
s820.jc.sd	5	99.4	99.9	1536	174.2
s820.jc.sd.re	14	95.3	96.6	267502	
s820.jc.sr	5	98.7	100.0	1207	6.6
s820.jc.sr.re	9	98.5	99.8	7913	
s820.ji.sr	5	98.2	100.0	8385	35.4
s820.ji.sr.re	8	97.3	100.0	296864	
s820.jo.sd	5	100.0	100.0	1282	297.7
s820.jo.sd.re	22	92.5	93.6	381636	
s820.jo.sr	5	98.6	99.8	1212	80.4
s820.jo.sr.re	13	97.3	98.8	97495	
s832.jc.sr	5	98.4	100.0	1225	405.7
s832.jc.sr.re	27	53.7	56.0	496961	
s832.jo.sr	5	98.1	100.0	1103	452.6
s832.jo.sr.re	15	96.7	99.1	499200	
scf.ji.sd	7	99.6	100.0	17262	40.0
scf.ji.sd.re	20	63.1	63.7	689651	
scf.jo.sd	7	99.6	100.0	16725	41.8
scf.jo.sd.re	23	97.8	97.9	699508	

Table 2 contains a number of interesting results. For instance, rows 13 and 14 of this table describe the circuits s510.jo.sr and s510.jo.sr.re. While HITEC required only 3822 CPU seconds to attain 96.2% fault coverage and 100% fault efficiency for s510.jo.sr, HITEC was able to attain only a 56.5% fault coverage and 57.0% fault efficiency in 1000000 (1 million) CPU seconds for s510.jo.sr.re. (One million seconds is more than eleven days.) During the final 320000 CPU seconds of HITEC execution, no additional faults were either detected or determined to be sequentially redundant. (For each retimed circuit, HITEC was manually halted after at least 12 CPU hours had expired without a single additional fault being detected or labeled redundant.) Though not reflected in Table 2, HITEC was able to initialize each circuit in less than 2 CPU seconds.

Table 3: Attest ATPG results.

circuit	%FC (orig)	%FE (orig)	%FC (re)	%FE (re)	CPU ratio
dk16.ji.sd	99.3	99.7	95.1	99.3	176.2
pma.jo.sd	99.2	99.4	96.3	98.3	18.8
s510.jc.sd	95.0	95.3	51.9	52.2	23.3
s510.ji.sr	95.6	95.6	79.9	79.9	8.0
s510.jo.sr	94.2	94.2	71.5	71.5	12.3

Table 4: Sequential EST ATPG results.

circuit	%FC (orig)	%FE (orig)	%FC (re)	%FE (re)	CPU ratio
dk16.ji.sd	98.0	99.8	97.6	99.3	3.5
pma.jo.sd	98.3	100.0	96.4	97.8	104.6
s510.jc.sd	95.4	98.2	6.7	10.4	2.1
s510.ji.sd	95.7	99.5	95.2	99.1	2.5
s510.jo.sr	92.2	94.6	63.6	65.4	2.7

2.3. Attest and Sequential EST results

Tables 3 and 4 list the Attest and SEST results, respectively, for those circuits which exhibit the most dramatic difference in the performance of that ATPG between the original and retimed circuits. The columns titled *%FC (orig)* and *%FE (orig)* list the fault coverage and fault efficiency on the original circuit, while the columns titled *%FC (re)* and *%FE (re)* list the fault coverage and fault efficiency on the corresponding retimed circuit. In order to avoid misinterpretations between the relative performance of the ATPGs, Tables 3 and 4 do not provide the execution times in terms of the absolute number of CPU seconds. (However it is worth noting that the CPU times for all retimed circuits in Tables 3 and 4 except dk16.ji.sd.re measured in the hundreds of thousands of CPU seconds.) Instead the most relevant metric, the ratio of ATPG CPU time between the retimed and corresponding original circuit, is reported in the column titled *CPU ratio*.

3. Analysis of experimental results

The results presented above demonstrate that retiming increases the amount of ATPG time, and decreases the levels of fault coverage and fault efficiency by each of the three sequential ATPGs employed. Notice that there are seven instances in Table 2 where the ratio of CPU time between the retimed and original circuits is more than two orders of magnitude, and in four instances there is a substantial difference between the levels of fault coverage and fault efficiency attained on the original and corresponding retimed circuit.

There are a number of possible causes of the observed effect of the increase in the complexity of ATPG which should be investigated. First, this effect might be due to a limitation of the particular heuristics used for sequential test generation. This is plausible given the difficulty of sequential test generation, which is much less mature than combinational test generation. (Of course, the necessity of handling the “time dimension” makes test generation for sequential circuits considerably more complex than for combinational circuits. Unlike combinational ATPGs which can find a test for any testable fault in a combinational circuit given only structural information of the circuit under test, no known practical algorithm exists for structural sequential test generation [8].)

Second, the effect might be caused if retiming introduces a large number of undetectable (sequentially redundant) faults into the circuit. The taxonomy of redundant faults provided in [10] reports that the most commonly occurring types of sequentially redundant faults (SRFs) are invalid-SRFs and equivalent-SRFs. Invalid-SRFs are those faults for which no valid excitation state exists, and equivalent-SRFs involve the interchange of equivalent states. Retiming adds D flip-flops to a circuit as well as states to the state transition graph. [11] illustrates that retiming can create and add equivalent states. Therefore the possibility exists that retiming introduces equivalent-SRFs. For each D flip-flop added, the number of possible states which the state bits can represent is doubled. However, the number of valid states (those states which are valid and can be traversed) does not necessarily increase accordingly. Hence retiming introduces the potential for a large number of invalid states, and therefore the potential for invalid-SRFs.

A large percentage of sequentially redundant faults in retimed circuits would account for the vast increase in the amount of CPU time required for test generation. In terms of the amount of computation time required, it is more expensive to determine if a fault is redundant than to construct a test for a detectable fault [11]. To label a fault redundant it must be proven that the fault cannot be

detected by any combination of input vectors and attainable states. This requires that the entire primary input and state space be covered, which is computationally prohibitive for practical circuits.

Third, the effect could be caused by some structural attribute which differs between the original and retimed circuits. Unlike behavioral test generation techniques, structural test generation techniques do not possess a knowledge of the state transition information of the machine which the circuit implements. Structural test generators have knowledge only of the structural nature of the circuit. Differences in the structural nature of circuits can therefore affect the performance of a structural ATPG as well.

The remainder of this paper is used to determine which of the above options is responsible for the observed increase in the complexity of ATPG caused by retiming.

4. Reason for high complexity of sequential ATPG

Among the three possible reasons which may be responsible for the high complexity of ATPG in retimed circuits, the least likely is the inadequacy of the test generators employed. Two facts support this observation. First, even though each ATPG differs in the test generation heuristics employed and specific implementation details, all three exhibited greater difficulty in deriving test sets for the retimed circuits. Second, all three ATPGs produced a few extremely "bad" results (however on different retimed circuits). This leads to the conclusion that the high complexity of sequential ATPG must be attributed to the circuit itself rather than to particular features of the ATPGs employed. It is therefore necessary to consider the other two options in more detail.

4.1. Sequentially redundant faults and testability preservation

In moving registers to the periphery of reconvergent regions, retiming may transform combinationally redundant faults into sequentially redundant faults [12]. However, based upon the knowledge that for each of the non-retimed circuits considered in this study less than 1% of the total number of faults were combinationally redundant, this alone could not account for the vast increase in test generation time required.

Converse to the arguments presented in Section 3, which postulate that retiming might add a large number of redundant faults to the circuit, it was shown in [13] and [14] that the retiming transformation preserves single stuck-at fault testability. For any given input sequence, such as a set of test vectors, retiming does not alter the sequence of logic values which propagate through each node in the circuit. Retiming can only alter the clock cycle at which the logic values arrive at affected nodes. The logic values themselves are not changed. Because the same logic values are propagated through each node, the same paths are sensitized, and therefore the same (single stuck-at faults) which those vectors detect in the original circuit are expected to be detected in the retimed circuit[1].

These results lead to the important conclusion that retiming preserves single stuck-at fault testability. Theorem 1 formalizes this concept.

Theorem 1: *The retiming transformation preserves single stuck-at fault testability.*

The reader is referred to [14] for the accompanying proof and further analysis of this concept. This consideration leads to the important conclusion that the retimed circuits do not contain a large number of extra redundant faults, and that the retimed circuits can be efficiently tested with the test sets generated for the original circuits. This in turn implies that the experimentally observed increases in ATPG might instead be caused by some change in the structural nature of the retimed circuits.

4.2. Structure of the circuits

The widely held belief (presented for instance in [8]) is that the complexity of structural, sequential test generation is linearly proportional to the maximum sequential depth of a circuit, and is exponentially related to the number of cycles present, and the maximum length of any cycle. The sequential depth of a path through a circuit (from a primary input to a primary output) is defined as the number of D flip-flops encountered along that path, in which each node is visited at most a single time. Maximum sequential depth refers to the greatest number of D flip-flops encountered in a traversal from any primary input to any primary output. A cycle exists when the same node can be revisited after starting from that node and traversing the circuit, not traversing any other node more than a single time within the traversal. The length of the cycle is said to be the number of D flip-flops encountered in the traversal.

The above reasoning makes intuitive sense for structural ATPGs, which are based on the iterative array model [15] in which a separate copy of the circuit is maintained to represent the value of each node at each cycle in time. The

[1] However, it is shown in [14] that there are faults in the retimed circuit that might not have equivalent single stuck-at faults in the original circuit. Furthermore, a synchronizing sequence (or a test) for a single stuck-at fault in a circuit K might not synchronize (or test) the corresponding fault(s) in a circuit K' resulting from a retiming of K. This can occur when retiming moves sequential elements (D flip-flops) forward across nodes (gates and fan-out stems). Furthermore, it is also shown in [14] that if T is a test for a single stuck-at fault in the circuit K, then the sequence P ∪ T is a test for the corresponding fault(s) in the retimed circuit K'. Here P is a sequence of arbitrary vectors of length equivalent to the maximum number of forward retiming moves across any node in the circuit.

presence of cycles increases the dependency of logic values on specific nodes in the circuit across time. The greater the number of cycles, the greater this dependency, and the more complex the task of establishing the desired logic values needed on certain nodes to sensitize a path. The same can be said about the maximum length of any cycle. The greater the maximum length of any cycle, the greater the number of time frames across which the dependency exists, and thus the greater the complexity of test generation.

Determining the number of cycles in a circuit is an NP-hard problem [16]. Fortunately cycle counting algorithms of reasonable complexity do exist. In order to investigate the suppositions mentioned above, the algorithm presented in [17] was implemented to measure the number and maximum length of any cycle. A separate algorithm was implemented to measure the maximum sequential depth of a circuit. Table 5 lists the obtained results, showing both the maximum sequential depth and cycle information of each of the circuits in Table 2. Columns titled *max seq depth* list the maximum sequential depth. Columns titled *max cycle length* and *#cycle*s list the maximum length of any cycle and the number of cycles according to the algorithm presented in [17], respectively. Column names which include *(orig)* refer to the non-retimed circuit, and those which include *(re)* refer to the corresponding retimed circuit.

Table 5: Structural attributes of each circuit.

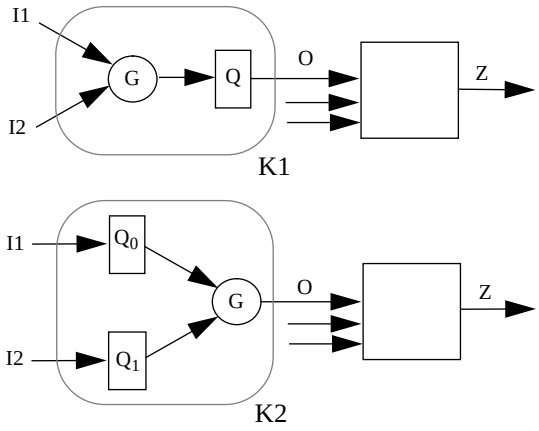
circuit	max seq depth (orig)	max cycle length (orig)	#cycles (orig)	max seq depth (re)	max cycle length (re)	#cycles (re)
dk16.ji.sd	4	4	10	4	4	19
pma.jo.sd	5	5	12	5	5	18
s510.jc.sd	6	6	15	6	6	26
s510.jc.sr	6	6	16	6	6	32
s510.ji.sd	6	6	18	6	6	21
s510.ji.sr	6	6	18	6	6	33
s510.jo.sr	6	5	15	6	5	28
s820.jc.sd	5	5	14	5	5	19
s820.jc.sr	5	5	14	5	5	18
s820.ji.sr	5	5	12	5	5	14
s820.jo.sd	5	5	14	5	5	24
s820.jo.sr	5	5	13	5	5	19
s832.jc.sr	5	5	11	5	5	25
s832.jo.sr	5	5	14	5	5	22
scf.ji.sd	7	6	22	7	6	32
scf.jo.sd	7	6	19	7	6	27

The results in Table 5 clearly indicate that retiming does not affect the sequential depth of a circuit. Such an observation can be generalized in the following way.

Theorem 2: *Retiming does not affect the sequential depth of any path through the circuit.*

Retiming of a network can be thought of as a collection of atomic transformations where an atomic transformation moves registers forward or backward across combinational logic gates, or forward or backward across a fanout stem. For simplicity and without loss of generality, an atomic retiming transformation that moves registers forward across a combinational logic gate is shown in Figure 1, where the circuit K1 is transformed into circuit K2. I1, I2 and Z are assumed to be functions of the primary inputs and the state variables. Z is assumed to be a primary output, G is a combinational gate, and Q is a sequential element (edge triggered D flip-flop). Note that all the nodes in K2 have structurally equivalent nodes in K1 except inside the marked (dotted) region. To show that retiming does not affect the sequential depth of any path through the circuit, it is sufficient to show that atomic retiming transformations do not affect the sequential depth of any path through the circuit.

Figure 1. Atomic retiming transformation.

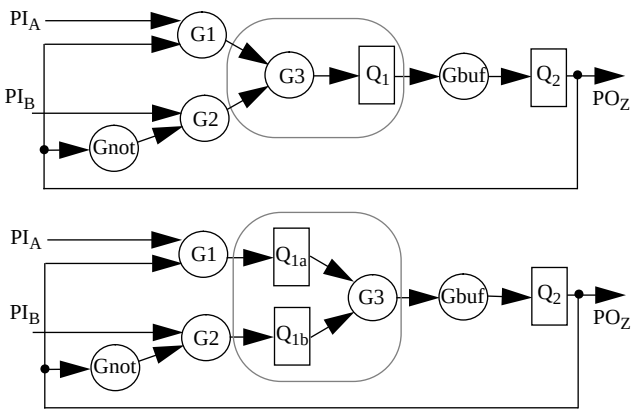


Proof: No two fan-ins to the same logic gate can be traversed without traversing the output of that gate more than a single time. Thus, according to the definition of sequential depth, at most a single fan-in stem to any gate can be traversed in any given path through the circuit. Referring to the atomic retiming transformation illustrated in Figure 1, the same number of D flip-flops will be traversed through the dashed region in Figure 1 whether the D flip-flops are at the input nodes to or output node of the gate. Since the rest of the circuit is unaffected by this atomic retiming operation, it is clear that a single atomic retiming operation that moves D flip-flops either forward or backward across combinational logic gates does not alter the sequential depth of any path through the circuit. Because retiming is a

collection of these atomic operations, this statement can be generalized to multiple and/or serial instances of this atomic operation. The same argument also holds for both forward and backward moves across fanout stems. Thus retiming cannot alter the sequential depth of any path through the circuit. ■

The second result to note from Table 5 is that retiming increases the number of cycles, according to the algorithm used. However one should note that, as pointed out in [17], the number of cycles computed varies according to the algorithm used. Before the difference in the number of cycles computed is used to account for the increase in the complexity of sequential test generation for retimed circuits, one should consider how the algorithm employed counts cycles. This counting is done according to the identity of the D flip-flops in the cycle - where at most a single cycle can exist for any unique subset of D flip-flops, regardless of the number of combinational paths which connect those D flip-flops.

Figure 2. Example circuit to demonstrate behavior of cycle counting algorithms.



Careful discretion must be exercised to differentiate between the actual cyclical nature of a circuit and the number of cycles measured by standard algorithms. Consider the simple retiming example illustrated in Figure 2. The circuit on the top represents the circuit before retiming, and the circuit on the bottom the retimed circuit. The algorithm used to report the results in Table 5 reports two cycles of length two in the retimed circuit on the bottom of Figure 2 - one through {G1, Q_{1a}, G3, Gbuf, Q_2} and a second through {Gnot, G2, Q_{1b}, G3, Gbuf, Q_2}. This algorithm counts cycles involving any one set of D flip-flops a single time, and therefore counts only a single cycle of length two in the original circuit on the top of Figure 2, even though two such cycles exist - one through {G1, G3, Q1, Gbuf, Q2} and the second passes through {Gnot, G2, G3, Q_1, Gbuf, Q_2}.

Though retiming may alter the number of cycles which are counted by various cycle counting algorithms, retiming does not alter the (actual) number of cycles. The following theorem has been formulated to support this claim.

Theorem 3: *Retiming does not change the number of cycles in a circuit.*

Proof: The atomic operation illustrated in Figure 1 preserves the connectivity involving the input and output nodes of the dotted region. Paths are not added which alter the set of nodes which can be reached from some other node. The atomic retiming operation does not create any new paths between any two nodes outside the dotted region, nor does it break any paths between any two nodes outside of the dotted region. Furthermore, retiming neither creates any new cycles nor breaks any existing cycles within the dotted region. In preserving the exact connectivity of nodes outside of the dotted region, and in neither adding nor deleting any new cycles within this dotted region, an atomic retiming operation that moves D flip-flops either forward or backward across combinational logic gates preserves the exact number of cycles. The same argument can be applied to atomic operations which move D flip-flops either forward or backward across fanout stems. This result can be generalized to the retiming transformation since retiming is a collection of these atomic operations. ■

The results in Table 5 also indicate that retiming does not increase the maximum length of any cycle, thereby offering experimental data to support the following Theorem.

Theorem 4: *Retiming does not affect the length of any cycle in a circuit.*

Proof: Any cycle beginning and ending at the output node of the gate G in Figure 1 can traverse at most a single input to Gate G. The same number of D flip-flops are encountered in a traversal through the dotted region regardless of whether the D flip-flops are at the output node of gate G or if the retiming transformation has been used to move D flip-flops to the input nodes of that gate. Because a single atomic retiming operation would not affect any other portion of the circuit outside of the dotted region in Figure 1, a single atomic retiming operation that moves D flip-flops either forward or backward across combinational logic gates does not change the length of any cycle, regardless of whether or not that cycle traverses the dotted region. The same argument can be applied to moves across fanout stems. This result can be generalized to multiple atomic retiming operations, and therefore the retiming operation does not change the number of D flip-flops encountered within a cycle traversal. ■

Theorems 2 and 4 can also be shown based on Lemma 1 of [2].

4.3. General observations

At this point is it useful to summarize the results presented so far with a couple of more general observations.

Recall that experimental results and theorems have been presented which show that neither extra sequentially redundant faults nor such structural circuit attributes as the basic characteristics of sequential cycles could be responsible for the observed retiming related increase in ATPG complexity. This leads to the more general conclusion that the high complexity of ATPG tasks for any circuit should not be explained (or quantified) using solely notions proposed in the past literature and commonly accepted by both test generation software developers and IC designers. The remainder of this paper suggests a circuit attribute which can be used to explain the observed increase in ATPG complexity.

5. Complexity of sequential ATPG

The difference between the original and retimed circuits which causes the increase in complexity is most likely related to the (three separate) tasks necessary to generate a test to detect a fault in a sequential circuit. In the first task the values of the machine state and primary input values which excite the fault must be determined. Next a justification sequence must be derived in order to attain the value of the excitation state on the state bits. Finally, the effect of the fault must be propagated to a primary output [11]. The task of exciting the fault is on the order of complexity of combinational test generation. However the tasks of state justification and fault propagation both involve traversing distinct states of the circuit.

For structural test generation, the complexity of state traversal must be correlated to the size of the state space which must be traversed. The total number of possible states in a circuit is $2^{\#\text{D flip-flops}}$. However not all states are necessarily valid, meaning that not all states can be traversed. (Valid states are those states which can be reached from the reset state of the machine. A state which cannot be reached from the reset state is called an invalid state. The circuit is known to be in a reset state after either a hardware reset or a synchronizing sequence of inputs is applied [18].) Structural test generators do not possess a knowledge of the state transition information, and thus at the beginning of test generation have no knowledge of which states (combinations of D flip-flop values) are traversable. The presence of invalid states is known to increase the difficulty of state traversal [19]. Based upon this knowledge, some sequential ATPG algorithms employ state learning techniques to eliminate duplicate searches in the invalid state space, thereby increasing their efficiency [20], [21]. These techniques have proven to decrease the amount of ATPG time which is required for some circuits by an order of magnitude. However state learning cannot completely eliminate the increase in complexity due to the presence of invalid states, it can only increase the efficiency with which circuits which contain invalid states are processed.

Table 6: HITEC ATPG state traversal information.

circuit	#states HITEC trav	#valid states	% valid states trav	total #states	density of encoding
dk16.ji.sd	27	27	100	32	0.84
dk16.ji.sd.re	89	105	85	5.24E5	2.0E-4
pma.jo.sd	27	27	100	32	0.84
pma.jo.sd.re	27	27	100	2.09E6	1.3E-5
s510.jc.sd	47	47	100	64	0.73
s510.jc.sd.re	47	47	100	1.04E6	4.5E-5
s510.jc.sr	47	47	100	64	0.73
s510.jc.sr.re	18	148	12	6.71E7	2.2E-6
s510.ji.sd	47	47	100	64	0.73
s510.ji.sd.re	69	70	99	2048	3.4E-2
s510.ji.sr	47	47	100	64	0.73
s510.ji.sr.re	64	202	32	8.38E6	2.4E-5
s510.jo.sr	47	47	100	64	0.73
s510.jo.sr.re	22	490	5	2.68E8	1.8E-6
s820.jc.sd	24	24	100	32	0.75
s820.jc.sd.re	100	164	61	16384	1.0E-3
s820.jc.sr	24	24	100	32	0.75
s820.jc.sr.re	42	47	89	512	9.1E-2
s820.ji.sr	24	24	100	32	0.75
s820.ji.sr.re	40	50	80	256	3.9E-3
s820.jo.sd	24	24	100	32	0.75
s820.jo.sd.re	77	297	26	4.19E6	7.1E-5
s820.jo.sr	24	24	100	32	0.75
s820.jo.sr.re	46	48	96	8192	5.9E-3
s832.jc.sr	24	24	100	32	0.75
s832.jc.sr.re	23	273	8	1.34E8	2.0E-6
s832.jo.sr	24	24	100	32	0.75
s832.jo.sr.re	47	54	87	32768	1.6E-3
scf.ji.sd	94	94	100	128	0.73
scf.ji.sd.re	41	209	20	1.04E6	2.0E-4
scf.jo.sd	94	94	100	128	0.73
scf.jo.sd.re	93	94	99	8.38E6	1.1E-5

Note that each D flip-flop that retiming adds doubles the size of the state space which an ATPG must search. Furthermore, the number of valid states grows at a rate lower than the total number of states. Hence, not only does retiming increase the size of the search space, it also decreases the fraction of the total number of states which are valid. It is therefore worthwhile to investigate differences in the characteristics of the state spaces between the original and

retimed circuits. Information concerning the nature of the state space in each circuit can be gained by considering the number of valid states, the total number of states, and the actual number of distinct states traversed during test generation. This information is presented in Table 6 for the HITEC test results for those circuit pairs reported in Table 2. In Table 6, the column titled *# states HITEC trav* lists the number of states which HITEC traversed during test generation. The column titled *# valid states* lists the actual number of valid states for that circuit[2]. The column titled *% valid states trav* lists the percentage of valid states which HITEC traversed in deriving the test set for that circuit, and the column titled *total #states* lists the total number of possible states ($2^{\#D\ flip\text{-}flops}$).

The information in Table 6 suggests what might cause the dramatic increase in the complexity of test generation in the retimed circuits. The data supports the conjecture that this increase is related to both the explosion in the size of the state space which the ATPG must traverse and the percentage of states which are valid. Because structural test generators do not possess state transition information, the smaller the percentage of states which are valid, the greater the chance that the test generator will spend time attempting to traverse to an invalid state. Thus the smaller the fraction of the total number of states which are valid, the greater the difficulty of state traversal, and the higher the complexity of sequential ATPG. Table 6 reveals that the percentage of states which are valid is considerably less in the retimed circuits than in the original circuits. For example, in deriving the test set for the original circuit dk16.ji.sd, HITEC searched a state space where 27 of a total of 32 possible states were valid. However to derive the test set for the corresponding retimed circuit dk16.ji.sd.re, HITEC was forced to navigate a state space where only 105 out of 5.24e+05 (2^{19}) states were valid. Comparing the test generation results in Table 2 with the state space information in Table 6, it is clear that test generation is least complex in those circuits which use the minimum number of state bits to encode the possible machine states (the original circuits), and is more difficult in circuits which use more than the minimum number of state bits necessary to encode the states of the machine (the retimed circuits). We will use the term *density of encoding* to describe the fraction of the total number of possible states which are valid.

[2] The SIS command *extract_seq_dc* was used to determine the actual number of valid states. The state minimization procedure employed during logic synthesis minimized the total number of states in all non-retimed s820, s832, and scf circuits to 24, 24, and 94, respectively. SIS added three states to pma.jo.sd to specify edges which were previously unspecified in the pma FSM, thereby resulting in a better implementation of that circuit (which contains 27 states). The original circuits synthesized from all other FSMs contain the number of valid states listed in Table 1.

The right-most column of Table 6 lists density of encoding information for each circuit. This column reveals a number of interesting results. Comparing the original and retimed circuits (and referring to the test generation results in Table 2), HITEC is able to attain higher levels of test coverage (fault coverage and fault efficiency) for those circuits with higher densities of encoding, and lower levels of test coverage for those circuits with lower densities of encoding. Furthermore, HITEC attained the lowest levels of test coverage for the three retimed circuits for which the densities of encoding are the lowest (s510.jo.sr.re, s832.jc.sr.re, and s510.jc.sr.re). It is difficult to draw conclusions from a direct comparison of the density of encoding values and the test generation results of the retimed circuits because of the non-linear relationship between the amount of CPU time which is required to attain given levels of test coverage. In addition, we do not claim that the density of encoding is the only circuit attribute which impacts the complexity of sequential ATPG.

Table 7: Density of encoding sensitivity analysis.

circuit	delay (nsec)	#DFF	#valid states	total #states	density of encoding
s510.jo.sr	43.87	6	47	64	0.73
s510.jo.sr.re.v1	42.51	8	71	256	0.28
s510.jo.sr.re.v2	42.04	16	150	65536	2.3E-3
s510.jo.sr.re.v3	41.55	22	233	4.19E6	5.6E-5
s510.jo.sr.re	41.51	28	490	2.68E8	1.8E-6

A more effective way to isolate the effect of the density of encoding on the complexity of test generation is to consider multiple retimed versions of the same original circuit. Each circuit would have a different density of encoding, however all circuits would implement the same functionality, and have identical values of sequential depth, number of cycles, and maximum length of any cycle (according to Theorems 2, 3, and 4). Table 7 lists one such set of circuits. In addition to the already existing retimed circuit s510.jo.sr.re, SIS was used to synthesize three additional retimed circuits (s510.jo.sr.re.v1, s510.jo.sr.re.v2, s510.jo.sr.re.v3) from the original circuit s510.jo.sr. Each retimed circuit has a different number of D flip-flops (and therefore area) and achieves a different level of performance (delay). The columns in Table 7 titled *#DFF* and *delay* list the number of D flip-flops and the cycle time (in nano-seconds) for each circuit. The column titled *# valid states* lists the number of valid states for that circuit, the column titled *total #states* lists the total number of possible states ($2^{\#D\ flip\text{-}flops}$), and the right-most column lists the density of encoding for each circuit. Figure 3 plots the amount of (DECstation 3100) CPU time required to attain the level of fault efficiency listed on the vertical axis for

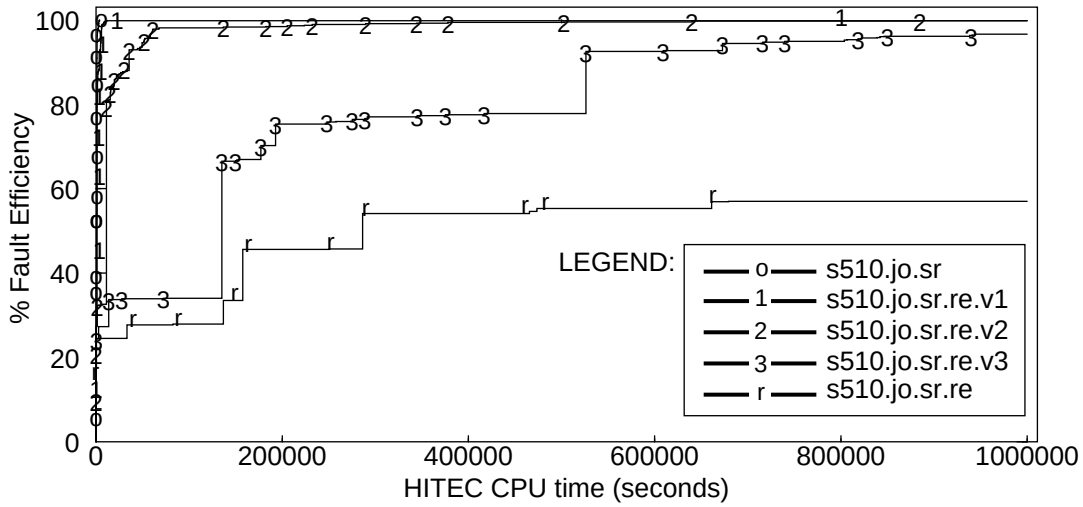


Figure 3. ATPG performance as a function of density of encoding.

each of these circuits. Figure 3 clearly demonstrates that as the density of encoding decreases, the greater the amount of ATPG CPU time required to achieve a given level of fault efficiency.

Circuits with a low density of encoding reduce the test generator's ability to traverse the state space. Examination of state traversal data underscores the importance that the test generator be able to traverse the state space. According to the data in the column titled *% valid states trav* in Table 6, while HITEC was able to traverse all of the valid states in each of the original circuits, the same is not true of the retimed circuits. While traversing all valid states does not guarantee 100% fault efficiency (since to detect some faults a specific sequence of states might have to be traversed), there is a definite trend that when the ATPG is able to traverse a majority of the valid states it consistently attains a level of fault efficiency near 100%. HITEC is able to traverse at least 80% of the valid states for each retimed circuit for which it is able to attain greater than 96% fault coverage. HITEC was unable to traverse more than 20% of the valid states in those retimed circuits for which it did not attain a fault coverage above 65%. The lowest percentage of valid states traversed corresponds to those 4 retimed circuits (s510.jc.sr.re, s510.jo.sr.re, s832.jc.sr.re, and scf.ji.sd.re) for which HITEC attained the lowest levels of fault coverage and efficiency.

It is possible to determine the number of valid states which HITEC would have to traverse in these circuits to attain higher levels of fault coverage by fault simulating the test set derived for the original circuit on the retimed circuit. When the test set derived for the original circuit s832.jc.sr is fault simulated on the retimed circuit s832.jc.sr.re using PROOFS [22], the test set attains a 98.2% fault coverage and traverses 69 states. HITEC traversed only 23 of the possible 273 valid states in deriving the test set for s832.jc.sr.re, which attains only a 53.7% fault coverage. These results indicate that HITEC was not able to traverse an adequate number of states to attain the higher levels of fault coverage and efficiency which are achieved by fault simulating the test sets derived for s832.jc.sr on s832.jc.sr.re.

Table 8: Number of states which would have to be traversed to attain higher fault coverage.

circuit	%FC	%FE	#states HITEC trav	#valid states	#states trav by orig test set	%FC orig test set
s510.jc.sr.re	53.9	54.6	18	148	72	94.6
s510.jo.sr.re	56.5	57.0	22	490	102	96.2
s832.jc.sr.re	53.7	56.0	23	273	69	98.2
scf.ji.sd.re	63.1	63.7	41	209	147	99.5

Table 8 presents the results of this same fault simulation experiment for each of the four retimed circuits for which HITEC did not attain a fault coverage greater than 65%. The columns titled *%FC* and *%FE* list the fault coverage and fault efficiency, respectively, attained by HITEC for each circuit. The column titled *# states HITEC trav* lists the number of states which HITEC traversed during test generation. The column labelled *#valid states* lists the number of valid states in the circuit. The column titled *#states trav by orig test se*t and *%FC orig test set* list the number of states traversed and the fault coverage attained, respectively, when the test set derived for the corresponding original circuit is fault simulated on that retimed circuit. In each instance the test set for the original circuit achieves at least a 94.6% fault coverage and traverses at least three times as many states than HITEC traversed during test generation for the retimed circuit. Thus for the retimed circuits listed

in Table 8 HITEC was not able to traverse a sufficient number of states to attain greater than 90% levels of fault coverage and efficiency. The previous analysis suggests that the cause of this must be the low density of encoding in each of those retimed circuits. For these circuits the low density of encoding not only increased the amount of CPU time needed for sequential ATPG, but it also resulted in low levels of test coverage.

The analysis presented here investigates the properties of the broad class of heuristics used for sequential, structural ATPG. By comparing the test generation results (in Tables 2, 3, and 4) for any given pair of original/retimed circuits, it is evident that the density of encoding impacts each of the ATPGs to a different degree. These differences are understandable, as there are many differences between the algorithms which are employed by the ATPGs considered. The degree to which the density of encoding impacts these various algorithms motivates further research.

6. Conclusion

The research reported in this paper was conducted to identify those attributes, of both sequential circuits and structural, sequential ATPG algorithms, which can lead to extremely high test generation times. The retiming transformation was used as a mechanism to create two classes of circuits which present varying degrees of complexity for test generation. It was determined that the increase in complexity of testing was present in three sequential test generators, thus eliminating the possibility that the effect was specific to a single ATPG. The difference in this complexity was then related to well controlled differences in the structural nature of the original and retimed circuits, and it was determined that this difference was not due to differences in either sequential depth or cycles - those circuit attributes which have traditionally been associated with the complexity of sequential ATPG. Evidence was then given that another circuit attribute, termed density of encoding, is a key indicator of the complexity of structural, sequential ATPG. Density of encoding seems to capture well the essence of the difficulty with which structural ATPGs must struggle to produce adequate test sets.

The experimental and theoretical results presented here permit the complexity of sequential ATPG to be viewed from a new perspective. This new perspective should ultimately help to affect the way ATPG algorithms should be constructed and to optimize design for testability (DFT) solutions which can be applied to target both reduction in cost and improvements in the quality of IC testing.

References

[1] C.E. Leiserson, F.M. Rose, and J.B. Saxe, "Optimizing Synchronous Circuitry by Retiming", *Advanced Research in VLSI: Proceedings of the Third Caltech Conference*, pp. 86-116, 1983.

[2] C.E. Leiserson and J.B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, No.1, 1991, pp. 5-35.

[3] T.E. Marchok and W. Maly, "Automatic Synthesis and the Cost of Testing", *Proceedings of the 1994 Custom Integrated Circuits Conference*, pp. 132-135, May, 1994.

[4] T.M. Niermann and J.H. Patel, "HITEC: A Test Generation Package for Sequential Circuits", *Proceedings of the European Design Automation Conference*, pp. 214-218, 1991.

[5] "TDX Product Description Sheets", Attest Software, Inc., 4677 Old Ironsides Drive, Suite 100, Santa Clara, CA 95054.

[6] X. Chen and M.L. Bushnell, "SEST - A Sequential Circuit Automatic Test Pattern Generator at Rutgers - User's Guide", Rutgers University, Piscataway, NJ, March, 1994.

[7] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization", *Proceedings of the International Conference on Computer Design*, pp. 328-333, 1992.

[8] A. Miczo, *Digital Logic Testing and Simulation*, Harper & Row Publishers, New York, 1986.

[9] E.M. Sentovich, et.al., "*SIS: a System for Sequential Circuit Synthesis*", U.C. Berkeley Memorandum No. UCB/ERL M92/41, May, 1992.

[10] S. Devadas, H.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "Irredundant Sequential Machines via Optimal Logic Synthesis", *IEEE Transactions on Computer Aided Design*, pp. 8-16, January 1990.

[11] A. Ghosh, "Techniques for Test Generation and Verification of VLSI Sequential Circuits", Ph.D. Dissertation,. U.C. Berkeley Memorandum No. UCB/ERL M91/73, Sept., 1991.

[12] S. Dey and S.T. Chakradhar, "Retiming Sequential Circuits to Add Testability", *Proceedings of the VLSI Test Symposium*, pp. 28-34, 1994.

[13] T.E. Marchok, A. El-Maleh, J. Rajski, and W. Maly, "Test Set Preservation Under Retiming Transformation", Carnegie Mellon University Technical Report CMUCAD 94-23, May, 1994. Presented at the *First International Test Synthesis Workshop*, May, 1994, Santa Barbara, CA.

[14] A. El-Maleh, T.E. Marchok, J. Rajski, and W. Maly, "Behavior and Testability Preservation Under the Retiming Transformation", McGill University Technical Report #94-R3, December, 1994.

[15] M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, 1976.

[16] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co., New York, 1979.

[17] A. Lioy, P.L. Montessoro, and S. Gai, "A Complexity Analysis of Sequential ATPG", *Proc. of ISCAS*, 1989, pp. 1946-9.

[18] F. Hennie, *Finite State Models for Logical Machines*, John Wiley, New York, 1986.

[19] K.R. Bowden, "A Technique for Automatic Test Generation for Digital Circuits", *IEEE Intercon Conference Record*, Paper 15/1, 1975.

[20] N. Gouders and R. Kaibel, "Test Generation Techniques for Sequential Circuits", *Proceedings of the VLSI Test Symposium*, pp. 221-226, 1991.

[21] X. Chen and M.L. Bushnell, "Dynamic State and Objective Learning for Sequential Circuits Automatic Test Generation Using Decomposition Equivalence", *Proceedings of the Fault Tolerant Computing Symposium*, 1994.

[22] T.M. Niermann, W.T. Cheng, and J.H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator", *Proceedings of the 27th Design Automation Conference*, pp. 535-540, 1990.