

State Variable Extraction to Reduce Problem Complexity for ATPG and Design Validation *

Qingwei Wu and Michael S. Hsiao {qiwu3, hsiao}@vt.edu

Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA

Abstract

We present a new algorithm to extract characteristic flip-flops, which form a characteristic state set, using state correlation information. The extracted characteristic state set allows us to focus on a significantly smaller set of flip-flops while ignoring other flip-flops, thereby simplifying the target problem and reducing state explosion in very large sequential circuits. Next, partitioning is applied only on the characteristic state variables, and partial state transition graphs (STGs) are built. During test generation, test vectors are generated using a two-fold criteria: (1) whether the vector will expand the overall STGs, and (2) whether this vector will break the relationship among flip-flops within the correlated sets. Experiments showed that our extraction algorithm can reduce the original complete state set by up to 97%. In addition, with the reduced state variables, we achieve not only equal or better coverages for both stuck-at faults and design errors, the execution time is also significantly reduced due to the much smaller set of flip-flops. For some large sequential circuits, highest coverages have been obtained.

1 Introduction

Generating effective vectors is an important research problem in both testing and verification. However, developing efficient test generation methods for large sequential circuits remains very difficult due to time-frame expansion and the need for implicit state enumeration. In deterministic ATPGs [1–4], a fault is targeted at a time and the corresponding test sequence is generated for the target fault. Aborts often result for hard-to-detect faults when a large number of backtracks is encountered. Random or weighted random patterns can be used as a preprocessing step of the ATPG to first remove some faults to alleviate the computation cost. However, random-resistant faults make the approach limited, and computation for multiple weights may be needed to reach a satisfactory coverage.

Instead of deterministic methods, simulation-based ATPGs [5–10, 12] utilize information gathered by simulating the circuit to guide the test generation. Simulation-based ATPGs can be broadly divided into two categories: fault-simulation based test generation and logic-simulation based test generation. As its name implies, fault-simulation based test generators use metrics obtained during fault-simulation on target faults to guide the search. Since fault simulation can be costly, its application for large sequential circuits may be limited.

On the other hand, logic-simulation-based test generators generate vectors by exercising the fault-free circuit into expected behaviors with only logic simulation. Different criteria used to drive the fault-free circuit can yield different fault coverages. In LOCSTEP [7], it was observed that the test sequence generated by deterministic test generators generally traversed through a sequence of states, and the new states reached by the sequence usually helped in improving the fault coverage. Thus, the goal in [7] is to reach as many new states as possible. Since the number of reachable states in a large sequential circuit can be huge, the test set sizes obtained can be large. In [6], it has been observed that some measure to distinguish one new state from another is important in large circuits, as the fault coverage tends to level off even when new states continue to be added. Thus, the aim is to reach not only new, but also valuable states. To achieve this, state partitioning is used. The ATPG tries to favor those states that are deemed more valuable, and hence more likely to detect the hard faults. Because logic-simulation-based test generators don't target any faults, the fault coverage for large sequential circuits may be lower than the fault-simulation-based counter-part, such as those reported in [5, 10, 12]. Nevertheless, in [8], improved methods for state partitioning and partial STGs (State Transition Graph) have been proposed, and high fault coverages have been reported.

In design validation, formal methods using reachability analysis and model checking [13] are gaining popularity in the past decade. However, the well-known "state explosion problem", where the complexity of the model is exponential in the number of variables that

*This work is supported in part by NSF Grants CCR-0196470, CCR-0098304, and CCR-0305881.

comprise the system, is still a major hurdle in validating today's systems. An alternative approach for design verification is via simulation. However, the obvious disadvantage is that the simulation vectors may only traverse a small portion of all possible scenarios within a complex design, known as the coverage problem. Some semi-formal and abstraction techniques [11, 15–18] have concentrated on generating tests from scratch before simulation. For instance, high-quality abstract tests are extracted from a finite-state-machine (FSM) or VHDL model of a design. To explore the corner cases or complex scenarios, the concept of state exploration history was adopted in [14]. It maintained state exploration history to avoid redundant work in simulation. However, because a complete history is expensive to maintain, aggressive abstraction and approximation were used in [14]. In [20], a new model is proposed that adopts state partitioning and partial State Transition Graphs (STGs) to enhance the simulation coverage. It was observed that a good state partition helps to traverse through useful portions of the state space. By "useful portions", we mean portions of state space where hard stuck-at faults or design errors are more likely to be detected.

Our goal in this work is to develop an efficient algorithm to generate effective vectors for both stuck-at faults and design errors with only logic simulation *on a reduced circuit view* to avoid state explosion when dealing with large sequential circuits. We first use state correlation analysis to extract *characteristic* flip-flops at the gate level to form a characteristic state set. This characteristic state set is significantly smaller than the complete state set. Next, we focus only on these characteristic flip-flops and apply state partition [20] on them. Figure 1 illustrates the basic concept of our algorithm, which is dividing the characteristic state set into smaller groups of flip-flops after extraction. With partition obtained, partial State Transition Graphs (STGs) are built for each state group. Our ATPG engine tries to generate vectors that will either (1) expand the overall STGs as much as possible or (2) break the precomputed correlation relationships among flip-flops in the correlated sets. Experiments showed that our extraction algorithm can reduce the original complete state set by up to 97%. With *only* the characteristic flip-flops, equal or better coverages can be achieved for both stuck-at faults and design errors. In addition, since we target the much smaller set of flip-flops, the execution time is significantly reduced. For some large sequential circuits, highest coverages have been achieved.

The remainder of the paper is organized as follows. Section 2 gives an overview and motivation for the new test generation approach. Section 3 explains the algorithm for extracting characteristic flip-flops. Section 4

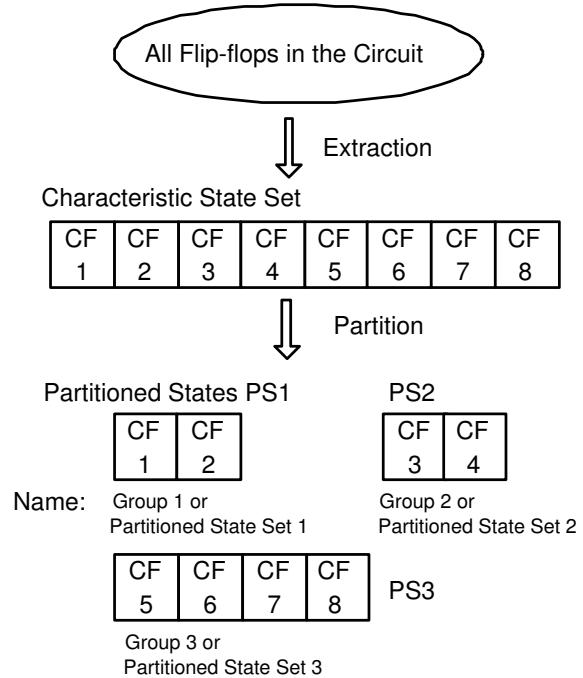


Figure 1: State Partitioning on Characteristic State Set

discusses the overlapped state grouping on characteristic flip-flops. Section 5 explains how to generate efficient vectors. Section 6 reports the experimental results, and Section 7 concludes the paper.

2 Overview and Motivation

Presently, logic-simulation based methods are gaining popularity in verifying digital system of today's scale and complexity. However, to make simulation methods efficient, state histories need to be maintained to some extent to avoid repeated visits. This can result in "state explosion problem". In [20], state partitioning and partial state transition graphs (STGs) were proposed to solve this problem. However, for very large sequential circuits, many STGs need to be stored and maintained, which complicates the computation and increases the execution time. Furthermore, a large number of STGs can bring additional noise to the test generation system so that it becomes difficult to distinguish the useful partitioned states from those which are less useful for detecting stuck at faults and design errors. Consequently, we would like to reduce the number of state variables that we need to consider if possible.

In general, the flip-flops on the data path do not contribute as much in guiding test generation as those flip-flops on the control path. We will explain this with a simple example. Figure 2 illustrates a circuit, in which flip-flops FF_1 , FF_2 and FF_3 are on the data

path while flip-flops FF_4 , FF_5 and FF_6 are on the control path. Suppose we partitioned all flip-flops into two sets, $\{FF_1, FF_2, FF_3\}$ and $\{FF_4, FF_5, FF_6\}$ and built corresponding STGs (STG_1 and STG_2) for each of them. Suppose we have two vectors V_1 and V_2 such that V_1 reaches a new state (and/or state-transition) only in STG_1 while vector V_2 reaches a new state (and/or state-transition) only in STG_2 . Which vector would be more useful if added to the test suite? Naturally, V_2 is better because V_2 allows us to explore more corner cases within the controller, while states within the data-path STG_1 are generally deemed as more controllable (less likely to be a corner-case state). To view this problem from a different angle, if we only consider flip-flops $\{FF_4, FF_5, FF_6\}$ and build one STG for this set only (flip-flops $\{FF_1, FF_2, FF_3\}$ are ignored), we can easily make the decision to choose V_2 . Another notable benefit is that the execution time can be greatly reduced since we target a much smaller set of flip-flops.

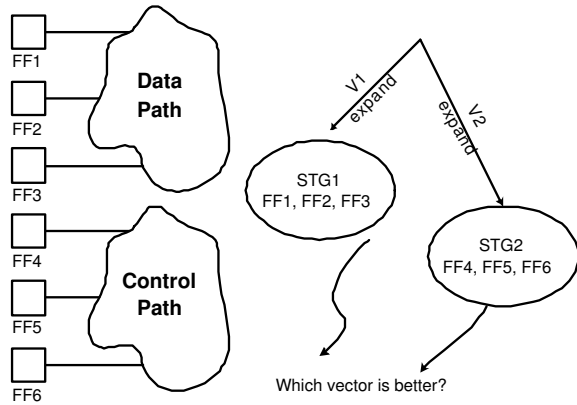


Figure 2: Differentiate between data path and control path

ATKET [21] extracts behavioral information to separate data path flip-flops from control path flip-flops. However, high level description of the circuit is needed. In this work, we perform state correlation analysis to extract the *characteristic* flip-flops from the gate level description of the circuit. The *characteristic* flip-flops are defined as a subset of the entire flip-flop set that can sufficiently represent the state characteristics of the circuit. In other words, one characteristic flip-flop is a representative of a group of flip-flops with similar properties. Then, overlapped state grouping is applied on this reduced state set and corresponding STGs are built. Finally, our ATPG engine uses spectrum information extracted from existing test set, and a Genetic Algorithm (GA) test generation framework is used to construct test vectors that will either (1) expand all the STGs as much as possible or (2) break the precomputed correlation relationships among flip-flops in the correlated sets. Fig-

ure 3 illustrates the overall framework. Details of the blocks in the figure are explained in the subsequent sections. The work described in this paper differs from our previous work [8, 20] in a number of ways. First, we only target the *characteristic* flip-flops and ignore other flip-flops in the circuit, while our previous work consider all flip-flops in the circuit. Second, in this work, the fitness function consider two factors as mentioned above, while our previous work only consider whether the vector will expand the STGs.

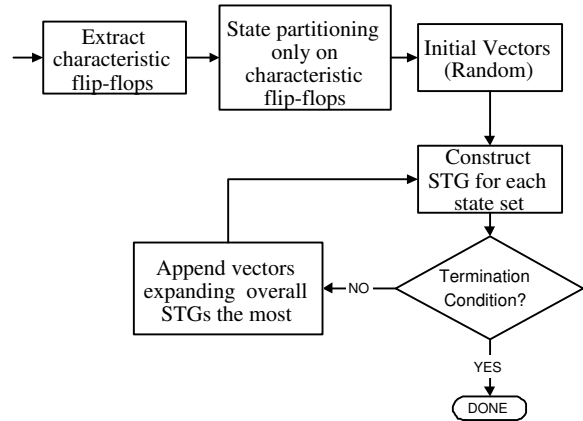


Figure 3: Overall Framework

3 Extracting Characteristic Flip-flops

The key step of our algorithm is the extraction of characteristic flip-flops. Two flip-flops FF_i and FF_j are said to be strongly correlated if in the presence of a particular logic value (0 or 1) in FF_i , the probability of FF_j having a specific logic value (0 or 1) is very high. Furthermore, if these two logic values are the same, we say FF_i and FF_j are positively correlated; otherwise, we say they are negatively correlated. For example, the bit streams listed in Figure 4 illustrate this concept. In case (a), the bit stream FF_i traversed is (10010011) and the bit stream FF_j traversed is also (10010011). We can see that FF_i always has the same logic value as FF_j for the entire sequence; therefore, they are positively correlated. Conversely, in case (b), the bit stream FF_j traversed is (01101100) and we can conclude that FF_i and FF_j are negatively correlated.

3.1 Review of signal correlation

In [22], exhaustive simulation of each combinational cone was performed to gain accurate signal correlation and a metric to determine the degree of correlation caused by a set of reconvergent fan-out paths is introduced. However, this step is very time consuming. In

FF _i	FF _j
1	1
0	0
0	0
1	1
0	0
0	0
1	1
1	1

positively correlated
(a)

FF _i	FF _j
1	0
0	1
0	1
1	0
0	1
0	1
1	0
1	0

negatively correlated
(b)

Figure 4: Positive and negative correlation

this work, to trade off between accuracy and computation effort, we only logic simulate the circuit using T random vectors (in our work, T equals 10,000) and record the bit stream each flip-flop traversed. In order to compute the correlation between two flip-flops, we first apply a transformation to each bit-stream b_i for flip-flop FF_i . That is, if any bit in the bit-stream is a logic 1 in the original sequence, it remains as 1 in b_i . If any of them is a logic 0, we use -1 instead. Then we use the following equations to determine if flip-flops FF_i and FF_j are correlated:

$$Corr_{ij} = \sum_{k=0}^T b_{ik} \times b_{jk}$$

$$Norm_Corr_{ij} = \frac{|Corr_{ij}|}{T} - P_r$$

where b_{ik} is the k^{th} bit of bit stream b_i , $Corr_{ij}$ is the correlation measure between flip-flops FF_i and FF_j , and P_r is a predetermined threshold. If $Norm_Corr_{ij}$ is not less than 0, then FF_i and FF_j are either positively or negatively correlated depending on the sign of $Corr_{ij}$. Using the previous example again, since the original bit-stream is (10010011), the actual values used in b_i will be (1 -1 -1 1 -1 -1 1 1). Bit stream b_j can be obtained in a similar fashion. Thus, we can see that, by this transformation, whenever b_{ik} and b_{jk} are different, the product will be negative, and it will decrease the correlation measure. On the other hand, if the two bits are the same, the product will be positive, increasing the correlation. In case (a) of previous example, the length $T = 8$ and let us assume that the threshold $P_r = 0.85$. Using the equations, we obtain $Corr_{ij} = 8$ and $Norm_Corr_{ij} = 0.15$. Since $Norm_Corr_{ij}$ is greater than 0, FF_i and FF_j are positively correlated according to the sign of $Corr_{ij}$. Similarly, in the second

case, we can determine that FF_i and FF_j are negatively correlated.

3.2 Our method to compute state correlation

A naive way to perform state correlation analysis is by calculating the correlation measurement for every pair of flip-flops. However, this would require too much computation and contain redundant work. If we consider only 100% (or -100%) correlation, then the correlation relationships have the transitive property; in other words, if $\{FF_i, FF_j\}$ are correlated and $\{FF_j, FF_k\}$ are correlated, then $\{FF_i, FF_k\}$ are correlated as well. On the other hand, if $\{FF_i, FF_j\}$ are correlated and $\{FF_j, FF_k\}$ are not correlated, $\{FF_i, FF_k\}$ are not correlated. Furthermore, suppose $\{FF_i, FF_j\}$ are positively correlated and $\{FF_j, FF_k\}$ are negatively correlated, then $\{FF_i, FF_k\}$ would be negatively correlated. Note that the above analysis is always true only if the predetermined threshold P_r is set to be 1. Nevertheless, even if flip-flops are not perfectly correlated, the transitive property most likely still holds. All correlation possibilities are listed in Table 1, where $Corr_{mn}$ represents the correlation relationship between flip-flops FF_m and FF_n ($m, n \in \{i, j, k\}$), '1' means positive correlated, '-1' means negatively correlated and '0' means not correlated.

Table 1: Transitive Properties of Correlation

		$Corr_{ik}$		
		$Corr_{jk}$		
$Corr_{ij}$	1	1	-1	0
	-1	-1	1	0
	0	0	0	0

The correlation analysis algorithm exploiting the transitive property is outlined below. As mentioned above, the transitive property is always true only if the threshold is 1. In practice, we use 0.98. Thus, the result of correlation analysis we obtained becomes approximate.

Now we detail the correlation computation process. First, we compute and store the correlation information in a correlation matrix M_{corr} which is an $n \times n$ square matrix (n is the number of flip-flops in the circuit). An element M_{ij} in correlation matrix M_{corr} indicates the correlation relationship between flip-flops FF_i and FF_j . It has three possible values as explained in Table 1. By noting that the element on the diagonal is always 1 and M_{corr} is symmetric, we only need to consider the upper triangular elements. In addition, if two flip-flops FF_i and FF_j are positively (negatively) cor-

related, then the corresponding two rows in M_{corr} (after column j , suppose $i < j$) will be identical (negated). This allows us to reduce the computation overhead significantly. The pseudo-code is listed below.

```

// step 1: Logic simulation
logic simulation with  $T$  random vectors;
record bit stream for each flip-flop;

// step 2: correlation analysis
calculate the first line in  $M_{corr}$ ;
for ( $i$  from 2 to  $n - 1$ ) {
    if ( $M_{i-1,i} \neq 0$ )
        // using transitive property
        copy values from line  $i - 1$  with the
        corresponding sign;
    else {
        for (all non-zero value in line  $i - 1$ )
            set to 0 in line  $i$ ;
        for (all rest flip-flops)
            calculate correlation measurement;
    }
}

```

We will illustrate how this algorithm works with a simple example. Suppose there are 5 flip-flops in the circuit, and flip-flop FF_1 is negatively correlated with FF_2 and positively correlated with FF_5 . So the first row in M_{corr} is (1 -1 0 0 1) as illustrated in Figure 5 (a). We already set all the elements on the diagonal to '1'. When we calculate the second row, since $M_{12} = -1$, we can propagate this to the second row, but we need to negate the sign on all elements in the row, as illustrated in Figure 5 (b). For the third row, since $M_{23} = 0$, we cannot propagate any non-zero value from the second row; instead, we set them to 0 in the third row, i.e. $M_{35} = 0$. Then we need to calculate the correlation measurement for the remaining flip-flops. In this case, we need to calculate M_{34} . Suppose flip-flops FF_3 and FF_4 are positively correlated as illustrated in Figure 5 (c). For the fourth row, since $M_{34} = 1$, we can simply copy the values from the third row to fill M_{45} . The lower triangular elements of the matrix can be filled in simply by copying from the upper triangular elements, and we will obtain the final correlation matrix as illustrated in Figure 5 (d).

Once we have obtained the correlation matrix M_{corr} , we can divide all flip-flops into several correlated sets, each of which is a transitive closure on some given flip-flops in M_{corr} . Starting from the first row of M_{corr} , for all the elements with non-zero values, all the corresponding flip-flops form a transitive closure, indicating that they are all related (either posi-

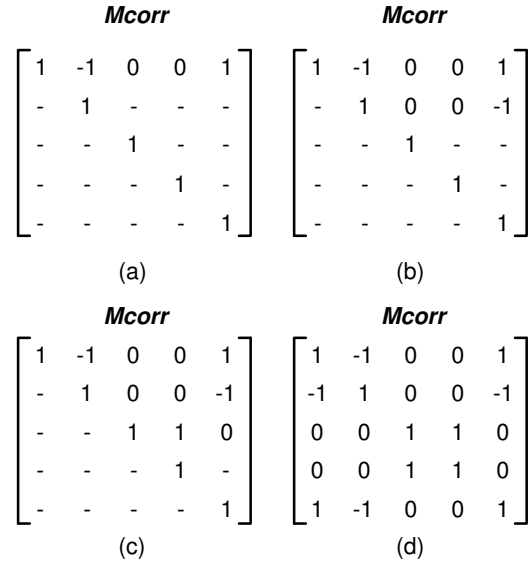


Figure 5: Correlation Analysis Using Transitive Property

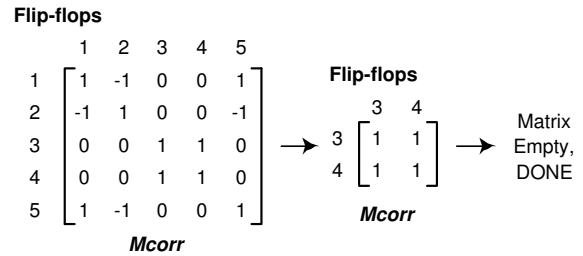


Figure 6: Transitive Closure in Correlated Matrix

tively or negatively). Then, the rows and columns corresponding to these flip-flops are deleted from M_{corr} . For the reduced matrix, repeat the above process until the matrix becomes empty. Using the example in Figure 5 (d), we start from the first row, since M_{11} , M_{12} and M_{15} have non-zero values, the corresponding flip-flops $\{FF_1, FF_2, FF_5\}$ form a cluster, thus composing a correlated set. Then we delete rows and columns corresponding to these 3 flip-flops from M_{corr} as shown in Figure 6. For the reduced matrix, starting from the first row again, we get another cluster, which contains two flip-flops FF_3 and FF_4 . Then we delete the corresponding rows and columns and the matrix becomes empty, ending the procedure. Finally we have 2 correlated sets: $\{FF_1, FF_2, FF_5\}$ and $\{FF_3, FF_4\}$. If we pick one representative flip-flop from each correlated set, these representative flip-flops are characteristic flip-flops of the circuit and form the characteristic state set.

Given a correlated state set, the way we select the representative flip-flop from the set must be clever. Because we want the representative flip-flop of one cor-

related set to interact with the flip-flops in *other* correlated sets as much as possible, we pick the flip-flop whose structural fan-out cone contains the most number of flip-flops in other correlated sets. Note that the flip-flops in the same correlated set with this flip-flop should not be over-counted. To do so, we add the following constraint: any flip-flop belonging to the same correlated set can only be counted at most once. The above algorithm works as follows. Suppose Figure 7 gives the circuit of previous example. Both FF_1 and FF_5 have one flip-flop in their fan-out cone. However, FF_5 will be picked as the representative flip-flop because FF_2 , the flip-flop in the fan-out cone of FF_1 , is in the same correlated set as FF_1 . The number of flip-flops in the fan-out cone of FF_3 is 2 because FF_1 and FF_5 belong to the same correlated set and they can only be counted once. FF_4 has no flip-flops in its fan-out cone, thus FF_3 will be selected as the representative flip-flop for the correlated set $\{FF_3, FF_4\}$.

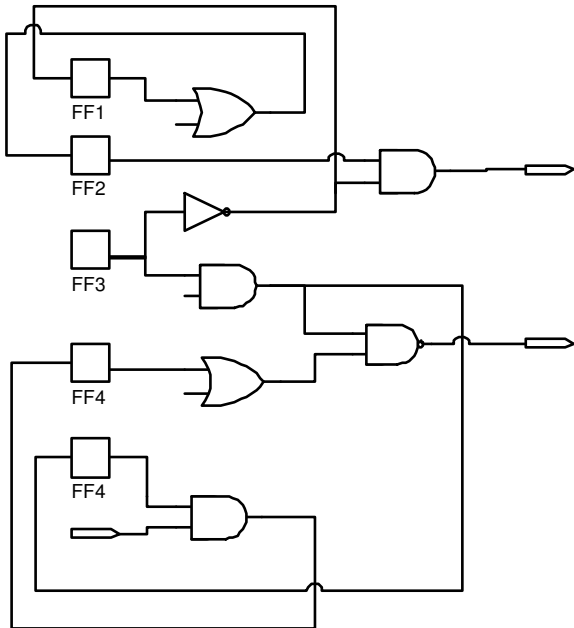


Figure 7: Choose Representative Flip-flop

4 Overlapped State Grouping on Characteristic Flip-Flops

Although the number of characteristic flip-flops is significantly smaller than the original number of flip-flops, it can still be expensive, sometimes infeasible, to maintain the complete state histories on the characteristic flip-flops for very large sequential circuit. In addition, in order to generate efficient vectors, we need to differentiate among the newly visited states on characteristic flip-flops. Hence, state partitioning is applied. We limit

the maximum size each state group can be as k (in this work, k equals 8). However, different from the previous work [20], our state partitioning is performed on the characteristic flip-flops only. [20] proposed a partition method based on "Common Paths Graph" and "triangle-based algorithm". The "Common Paths Graph" is a complete weighted graph, where each node represents a flip-flop and the weight of each edge indicates how many common paths to primary outputs and flip-flops its endpoints (two flip-flops) share. Then, a triangle-based algorithm is used to cut the graph, forming state partition. Readers are referred to [20] for the "Common Paths Graph" and "triangle-based algorithm".

In our work, we need to modify the original "Common Paths Graph" to form a reduced graph for the characteristic flip-flops. Since a given characteristic flip-flop represents the flip-flops of the correlated set, one simple way is to update the weights of the edges between two characteristic flip-flops with the average weight of the non-zero edge weights between their corresponding correlated sets as represented by the following equation:

$$new_w_{ij} = \frac{\sum old_w_{mn}}{\text{number of such edge}}$$

$$m, i \in CS_p, j, n \in CS_q, old_w_{mn} \neq 0$$

where CS_p is the correlated set containing flip-flop i and CS_q is the correlated set containing flip-flop j . This is illustrated in Figure 8. $FF_2, FF_3, FF_5, FF_7, FF_9$ and FF_{10} are characteristic flip-flops after the correlation analysis. $\{FF_1, FF_2, FF_6\}$ belong to correlated set CS_1 and $\{FF_3, FF_4, FF_8\}$ belong to correlated set CS_2 . The new edge weight between characteristic flip-flops FF_2 and FF_3 in the reduced graph is the average weight of edges $\{E_{13}, E_{14}, E_{23}, E_{68}\}$ in the original graph. With the reduced graph, we can use the triangle-based algorithm [20] to cut the graph, thus forming the state partition on characteristic flip-flops.

In the above state partitioning, each flip-flop in the circuit can belong to exactly one partitioned set. However, by combining the spectral information embedded in the characteristic flip-flops, we can construct overlapped state grouping; in other words, some characteristic flip-flops may belong to multiple state groups. Thus, we can adopt frequency decomposition using Hadamard transformation [19] to extract the spectral information embedded in the flip-flops. Note that, since we're only interested in characteristic flip-flops, spectral information is extracted *only* on characteristic flip-flops. In addition, since we already recorded the bit stream each flip-flop traversed during the process of the correlation analysis, extracting embedded spectral information on characteristic flip-flops with dominant frequency [8] will not introduce too much overhead.

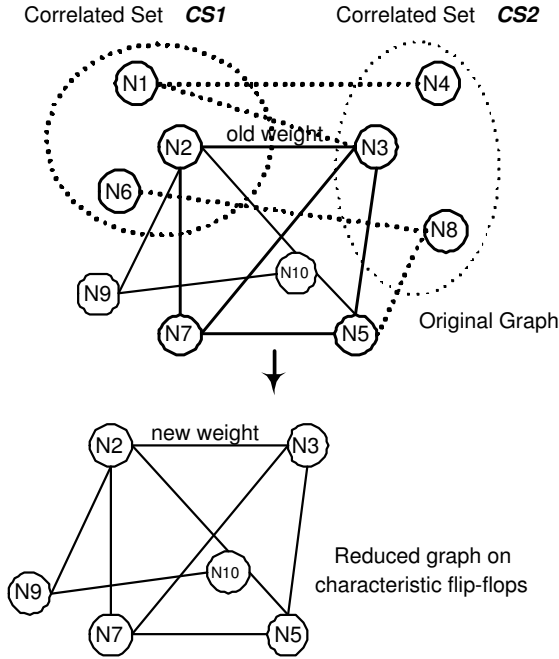


Figure 8: Update edge weight in reduced graph

The overlapped state grouping is illustrated in Figure 9. Suppose we have two state sets from the previous example, $\{FF_2, FF_9, FF_{10}\}$ and $\{FF_3, FF_5, FF_7\}$. If two flip-flops have the same dominant frequency but do not belong to the same state set, we add a *directed* edge between them. The direction is decided by using the reduced graph on characteristic graph discussed above. In Figure 9, suppose FF_2 and FF_3 have the same dominant frequency, and the average weight of the edges connecting FF_3 and the nodes in partitioned state set PS_1 is greater than that of the edges connecting FF_2 and the nodes in partitioned state set PS_2 , then the direction of the edge between FF_2 and FF_3 is pointed toward FF_2 . For each such edge, we add the original node into the directed state set. By doing this, we obtain overlapped state grouping incorporating both structural and spectral analyses.

5 Generation of Vectors

In [8,20], test vectors are generated such that each partitioned state space is maximally traversed. In this work, in addition to manipulating the STGs for each partitioned state set, we also observe the flip-flops within each correlated flip-flop set. Vectors that maximally traverse the STGs are added to the test set. However, recall that all the flip-flops belonging to the same correlated set have a precomputed correlation. If a new vector breaks this relationship among the correlated flip-flops, it is also added to the test set as this vector may help

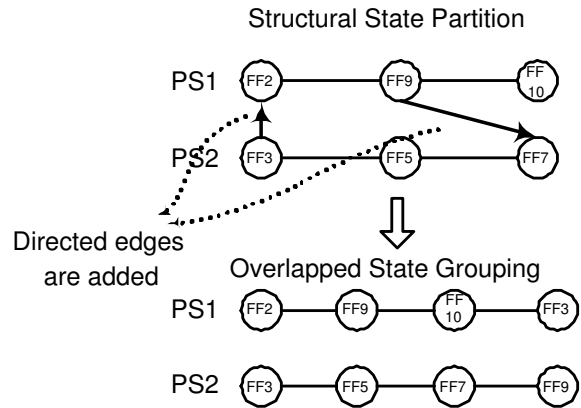


Figure 9: Overlapped State Grouping

to reach some new corner cases or exercise new activities of the circuit. This vector may help to detect additional hard faults or design errors. So, when we choose from candidate vectors or use Genetic Algorithm (GA) to generate new vectors, the criteria used is two-fold: (1) whether the vector will expand the overall STGs at most together with (2) whether this vector will break the correlation relationship in correlated sets. This process is shown in Figure 10.

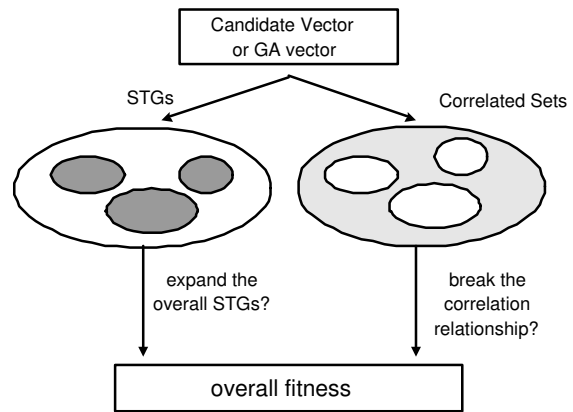


Figure 10: Manipulate STGs and Correlated Sets.

6 Experimental Results

The proposed algorithm is implemented in C++ and the experiments were conducted on a number of ISCAS89 [23] and ITC99 [24] benchmark circuits on a 2.4 GHz Pentium 4 with 512MB RAM, running the Linux operating system.

Table 2 reports the results of our correlation analysis algorithm for some large benchmark circuits. For each circuit, the number of original flip-flops is first given, then the number of characteristic flip-flops under vari-

Table 2: Size of Characteristic State Set in Large Sequential Circuits

Ckt.	orig. # FFs	Threshold									
		0.80	Frac	0.85	Frac	0.90	Frac	0.95	Frac	0.98	Frac
s1423	74	18	24%	20	27%	23	31%	25	34%	26	35%
s5378	179	49	27%	54	30%	57	32%	59	33%	59	33%
s35932	1728	29	1.7%	31	1.8%	35	2.0%	41	2.4%	45	2.6%
b12	121	5	4.1%	5	4.1%	7	5.8%	9	7.4%	12	10%
b15	447	71	16%	76	17%	80	18%	83	18.6%	86	19.2%
b22	709	112	16%	117	16.5%	125	17.6%	128	18.1%	136	19.2%

ous thresholds are reported. For each threshold, we report the number of characteristic flip-flops as a fraction of the original number. Intuitively, as the threshold increases, the size of the characteristic flip-flops will also increase. Nevertheless, even for very high threshold (0.98), the size of the characteristic state set is still significantly smaller than the original number of flip-flops. For circuit s35932, this is only 2.6% of the original flip-flops. This provides the foundation to significantly reduce the execution time of generating vectors for both stuck-at faults and design errors.

Next, we report our test generation results. For circuits with less than 50 flip-flops, extracting characteristic flip-flops is not very meaningful. However, we include results for some small circuits to show the effectiveness of the algorithm on small circuits as well. We allow for a maximum number of 20,000 vectors to be generated for each circuit, and we list the test generation time as the point where the maximum fault coverage was first reached. The threshold we used for correlation analysis is 0.98 and the maximum size each state group is set to 8.

In Table 3, the stuck-at results of our test generator is compared with HITEC [1], a deterministic test generator, LOCSTEP [7] and the results from [8], both of which are logic-simulation based test generators. For each circuit, the original number of flip-flops is first reported, followed by the number of characteristic flip-flops extracted. Then, the number of faults detected, test set size, and ATPG time are reported for each of the three ATPGs. Because we limit the maximum test set as 20,000, the results listed under [8] in the table for b12 and b15 are slightly different from those originally reported in [8]. Note that only our approach uses the characteristic flip-flop information, the other techniques targeted all flip-flops in the circuit. For LOCSTEP, execution times are not available.

From Table 3, we can observe that our approach is very efficient in obtaining high fault coverages together with small test sets. Our execution time is *significantly smaller* than the other approaches. This is because we target a significantly smaller number of flip-flops. Note that for most of the circuits, very high cov-

erages have already been achieved by [8], but the detection of a few extra faults is note-worthy, since these are very *hard-to-detect* faults, which most ATPGs spend most of their time on. For example, in the circuit s5378, HITEC detected 3238 faults, LOCSTEP detected 3059 faults, 3642 faults were detected in [8], and we were able to detect 3643 faults. Finally, for circuits b12, b15, and b22 (all very hard-to-test circuits), our approach achieved higher coverage than the previously reported approaches. In b12 and b15, highest coverages have been obtained by our method.

Tables 4 and 5 report our results when targeting design errors. In the experiment of Table 4, we injected 500 random gate substitution errors, where a gate is changed to a different gate type with the same number of inputs. The results of our test generator are compared with (1) randomly generated vectors and (2) random state partitioning. Random state partitioning is performed by simply grouping k flip-flops together (k is the maximum size of each state group). The same test generator is used to traverse the partial STGs in each of the random partitions. For each circuit, the number of flip-flops is first reported, then the number of characteristic flip-flops, followed by the number of errors detected, test set size and test generation time (in seconds) reported for each test generator. The test set size is reported as the point at which it detects all the detected errors. The last two columns report the ratios of the error coverage and tests sizes from our approach over those from random vectors.

From Table 4, we can observe that our approach is very effective in obtaining much higher gate substitution coverages together with smaller test sets and short execution times. For all the circuits, our approach can detect at least 34% more errors compared with random vectors. For some circuits, 2.58 times more defects can be detected. For the ITC99 circuit b15, random vectors can only detect 172 errors, with random partitioning, 283 errors (111 more) were detected; with our extraction algorithm, we were able to detect 386 errors while the test set is less than 70% of the random test set.

Table 5 reports the results for 500 randomly injected wiring errors. For all circuits, our method can detect at

Table 3: Experimental Results for Stuck at Faults

Circuit	# of FF_s	# of charac. FF_s	Deterministic ATPG			Logic-Simulation-Based ATPG								
			HITEC			LOCSTEP [7]		[8]			Our Approach			
			# Det	# Vec	Exe Time	# Det	# Vec	# Det	# Vec	Exe Time	# Det	# Vec	Exe Time	
s382	21	7	301	1,463	90.0	364	5,354	364	568	7.2	364	498	1.7	
s400	21	11	382	4,309	72.0	374	5,354	383	610	9.5	384	463	2.6	
s1423	74	26	776	177	834.0	1,274	9,616	1,416	1,334	180.0	1,416	956	56.3	
s5378	179	59	3,238	941	1104.0	3,059	7,545	3,642	1,479	240.0	3,643	769	82.5	
s35932	1,728	45	34,902	240	-	34,812	2,408	35,100	215	30.0	35,100	176	4.7	
b12	121	12	-	-	-	-	-	1,665	6,758	124.0	1,689	3,276	45.3	
b15	447	86	-	-	-	-	-	18,286	19,318	4,978	18,302	12,724	1,538	
b22	709	136	-	-	-	-	-	30,516	19,592	14,575	30,701	14,746	7,427	

* '-' indicates that the data was not available; execution times were not available for LOCSTEP

* Time reported in seconds

Table 4: Experimental results for gate substitution errors

Circuit	# of FF_s	# of charac. FF_s	Random Vectors		Random Partitioning			Our Approach			Cov Ratio	Vector Length Ratio
			# Det	# Vec	# Det	# Vec	Exe Time	# Det	# Vec	Exe Time		
s382	21	7	131	1,641	302	980	0.53	469	626	0.27	3.58	0.38
s400	21	11	174	1,732	217	1,517	0.65	437	347	0.42	2.51	0.20
s444	21	10	142	3,547	193	2,973	2.76	486	615	1.24	3.42	0.17
s1423	74	26	297	8,764	279	4,531	5.12	479	4,215	2.32	1.61	0.48
s5378	179	59	348	5,156	403	4,532	42.7	468	2,258	19.7	1.34	0.43
s35932	1,728	45	269	628	251	723	203.6	415	315	91.3	1.54	0.50
b04	66	19	233	3,592	258	2,538	10.3	398	1,548	2.1	1.70	0.43
b09	28	12	174	2,598	269	1,205	5.4	372	521	1.1	2.13	0.20
b12	121	12	140	7,234	147	5,683	20.4	295	4,639	5.1	2.10	0.64
b15	447	86	172	8,492	283	7,329	304.1	386	5,826	127.6	2.24	0.69
b22	709	136	187	9,513	232	9,047	725.3	369	5,264	203.2	1.97	0.55

least 30% more errors than random vectors. For some circuits, 3.87 times more errors can be detected. For instance, in circuit b12, we were able to detect 403 (out of 500) wiring errors in only 5.1 seconds. This is almost twice the error coverage compared with random vectors, in less than 25% of the execution time of random partitioning on all flip-flops.

7 Conclusion

In this paper, we have presented a new algorithm that uses state correlation analysis to extract characteristic flip-flops. Then overlapped state grouping that combines structural and spectral information is applied on the extracted characteristic flip-flops. Because the number of characteristic flip-flops is significantly smaller than the original number of flip-flops, our experimental results showed that very high fault coverages on both stuck-at faults and design errors can be achieved with smaller test sets and shorter execution times. In the future, we plan to continue to experiment on strategies to select different representative flip-flops from each correlated set and see their effect on the overall test gener-

ation process.

References

- [1] M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Design Automation Conf.*, 1991, pp. 214-218.
- [2] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," *IEEE Trans. on Computers*, vol. 42, no. 11, pp. 1361-1371, Nov. 1993.
- [3] I. Hamzaoglu and J. H. Patel, "Deterministic test pattern generation techniques for sequential circuits," *Proc. Int'l Conf. Computer-Aided Design*, 2000, pp. 538-543.
- [4] M. Henftling, H. C. Wittmann and K. J. Antreich; "A single-path-oriented fault-effect propagation in digital circuits considering multiple-path sensitization", *Proc. Int'l Conf. Computer-Aided Design*, 2002, pp. 304-309.
- [5] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal," *Proc. European Design and Test Conf.*, 1997, pp. 22-28.

Table 5: Experimental results for wiring errors

Circuit	# of FFs	# of charac. FFs	Random Vectors		Random Partitioning			Our Approach			Cov Ratio	Vector Length Ratio
			# Det	# Vec	# Det	# Vec	Exe Time	# Det	# Vec	Exe Time		
s382	21	7	142	709	198	312	0.53	459	291	0.27	3.87	0.41
s400	21	11	157	2,425	228	587	0.65	483	438	0.42	3.08	0.18
s444	21	10	139	3,721	264	893	2.76	492	513	1.24	3.54	0.14
s1423	72	26	368	8,216	410	6,217	5.12	495	3,052	2.32	1.35	0.37
s5378	179	59	361	3,478	402	5,126	42.7	489	1,367	19.7	1.35	0.39
s35932	1,728	45	347	362	425	296	203.6	493	138	91.3	1.42	0.38
b04	66	19	374	586	371	328	10.3	488	128	2.1	1.30	0.22
b09	28	12	223	5,740	341	8,593	5.4	476	2,015	1.1	2.13	0.35
b12	121	12	203	5,629	239	7,346	20.4	403	2,729	5.1	1.99	0.48
b15	447	86	231	7,428	264	7,428	304.1	399	5,328	127.6	1.72	0.71
b22	709	136	201	8,936	241	8,458	725.3	386	6,216	203.2	1.92	0.70

- [6] S. Sheng and M. S. Hsiao, "Efficient Sequential Test Generation Based on Logic Simulation", *IEEE Design and Test of Computers*, vol. 19, no. 5, pp. 56-64, 2002
- [7] I. Pomeranz and S. M. Reddy, "LOCSTEP: A Logic-Simulation-Based Test Generation Procedure," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 16, no. 5, May 1997, pp. 544-554.
- [8] Q. Wu and M. S. Hsiao, "Efficient Sequential ATPG Based on Partitioned Finite-State-Machine Traversal", *Intl Test Conf.*, 2003, pp. 281-289.
- [9] E. M. Rudnick et al., "Application of Simple Genetic Algorithms to Sequential Circuit Test Generation," *Proc. European Design and Test Conf.*, 1994, pp. 40-45.
- [10] R. Guo, I. Pomeranz, and S. M. Reddy, "A Fault Simulation Based Test Pattern Generator for Synchronous Sequential Circuits," *Proc. VLSI Test Symp.*, 1999, pp. 260-267.
- [11] V. M. Vedula, J. A. Abraham, J. Bhadra, "Program Slicing for Hierarchical Test Generation", *Proc. VLSI Test Symp.*, 2002, pp. 237-243.
- [12] A. Giani et al., "Efficient Spectral Techniques for Sequential ATPG," *Proc. IEEE Design Automation and Test in Europe Conf.*, 2001, pp. 204-208.
- [13] J. R. Burch, E. M. Clarke, K. L. McMilian, D. L. Dill and L. J. Hwang, "Symbolic model checking: 10₂₀ states and beyond," *IEEE Symp. Logic in CS*, 1990.
- [14] C. N. Ip, "Simulation Coverage Enhancement Using Test Stimulus Transformation," *Proc. Int'l Conf. CAD*, 2000.
- [15] R. C. Ho, C. H. Yang, M. A. Horowitz and D. L. Dill, "Architecture validation for processors," *Int'l Symp. Computer Architecture*, 1995.
- [16] D. Geist, M. Farkas, A. Landver, Y. Lichtenstein, S. Ur and Y. Wolfsthal, "Coverage-directed test generation using symbolic techniques," *Int. Conf. CAD*, 1996.
- [17] M. Benjamin, D. Geist, A. Hartman, Y. Wolfsthal, G. Mas and R. Smeets, "A study in coverage-driven test generation," *Prof. DAC*, 1999.
- [18] D. Moundanos, J. A. Abraham and Y. V. Hoskote, "Abstraction techniques for validation coverage analysis and test generation," *IEEE Trans. Computers*, 47(1):2-14, 1998.
- [19] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Discrete-Time Signal Processing*, Englewood Cliffs, New Jersey, Prentice Hall, Inc., 1999.
- [20] Q. Wu and M. S. Hsiao "Efficient ATPG for Design Validation Based On Partitioned State Exploration Histories," *Proc. VLSI Test Symp.*, 2004.
- [21] P. Vishakantaiah, J. Abraham and M. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 273 - 278
- [22] Q. Zhang and I. Harris, "Partial BIST Insertion to Eliminate Data Correlation", *Intl. Conf. CAD*, 1999, pp. 395 - 398
- [23] F. Brglez, D. Bryan, and K. Kozminski, Combinational profiles of sequential benchmark circuits, *Int. Symp. on Circuits and Systems*, 1989, pp. 1929-1934.
- [24] S. Davidson and Panelists, ITC 99 benchmark circuits-preliminary results, *Proc. Int'l Test Conf.*, 1999, pp. 1125.