Efficient Testing of Acyclic Structures in Partial Scan Designs

BY

Rajesh Gupta and Melvin A. Breuer

Technical Report No. CENG 90-13

April 1990

Electrical Engineering - Systems Department

University of Southern California

Los Angeles, CA. 90089-0781

Efficient Testing of Acyclic Structures in Partial Scan Designs*

Rajesh Gupta and Melvin A. Breuer

Department of Electrical Engineering–Systems
University of Southern California
Los Angeles, California 90089-0781
Phone: (213) 743-7258
Fax: (213) 745-7284
E-mail: gupta@usc.edu

April 1990

*This work was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract No. N00014-87-K-0861, and in part by the Semiconductor Research Corporation under Contract No. 88-DP-075. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

Abstract

It is well known that acyclic sequential structures are considerably easier to test than arbitrary sequential circuits. Hence some partial scan techniques attempt to simplify the test generation problem by ensuring that the portion of the circuit effectively under test is acyclic. In such designs the test time is dominated by the shifting of test patterns into and out of the scan path. We present a compacting technique that minimizes the number of test patterns required to detect an arbitrary fault. A modified test schedule is used in which each compacted pattern is held in the scan path until the next pattern is ready to be shifted in. An optimal test scheduling algorithm is presented which determines the minimum possible number of compacted test patterns required for an arbitrary fault based on the circuit structure. Using the optimal schedule we derive a condensed combinational test generation model (TGM) for combinational ATPG under a multiple fault model. This TGM replaces the iterative array used in traditional sequential ATPG. This model allows the detection of any arbitrary fault using a minimum possible number of distinct test patterns.

1 Introduction

Automatic test pattern generation (ATPG) for acyclic sequential structures is known to require substantially lower computation effort than for arbitrary sequential structures [1]. Partial scan approaches that make use of this fact have recently been proposed [2,3]. Essentially they select flip-flops (FFs) to be included in the scan path such that the portion of the circuit effectively under test, the **kernel**, is either acyclic or close to acyclic. Test generation is then carried out for the kernel to determine test sequences; the complexity of this computation is similar to that for combinational circuits. This approach requires that while in the test mode the clock signals for scan FFs should be controllable independently of the clock signals for the non-scan FFs. A test sequence can then be applied to the kernel using the following two steps alternately:

1. Serially shift a test pattern into the scan path while disabling the clock signal feeding the non-scan FFs (this effectively puts the non-scan FFs in a HOLD mode);
2. While disabling the clock signal feeding the scan FFs (putting them in a HOLD mode), activate the clock signal for the non-scan FFs for one clock cycle (this enables test data to propagate through one level in the kernel).

A *test sequence* for a fault in a sequential circuit consists of a set of consecutive time frames, in each of which patterns containing both specified and don't-care values may need to be applied at the various inputs. The *length* of a sequence is the total number of time frames in it. For an acyclic structure the length is related to the **depth** or the highest number of FFs in any path in the structure. If d is the depth of a structure, the test sequence length is bounded by $d + 1$. However, a given test sequence may contain unassigned or don't-care input values such that not all primary inputs need to be provided with new data at each of the $d + 1$ clock cycles. If the inputs and outputs of the structure under test are directly accessible, the time for applying the sequence is $d + 1$ clock cycles and is not affected by the presence of don't-care inputs. However, in a partial scan design many of the inputs and outputs of the structure are accessed by shifting data serially. Hence the presence of don't-care input values could potentially lead to a great saving in test time. In such circuits, where the length of the scan path is usually much higher than d, the test time is dominated by the time to shift new patterns into and out of the scan path.

An example of acyclic structures which need less than $d + 1$ input patterns are *balanced structures* [2]; these are a class of structures that require only a single-pattern test for any fault. The BALLAST methodology uses this fact by selecting scan path FFs so as to make the kernel balanced, i.e., every path between any two points in the kernel has the same number of FFs. Each test pattern is held in the scan path for $d + 1$ clock cycles. (In some cases the need to hold the test data can be eliminated by ordering the scan path appropriately and manipulating the test data [4].) This technique guarantees

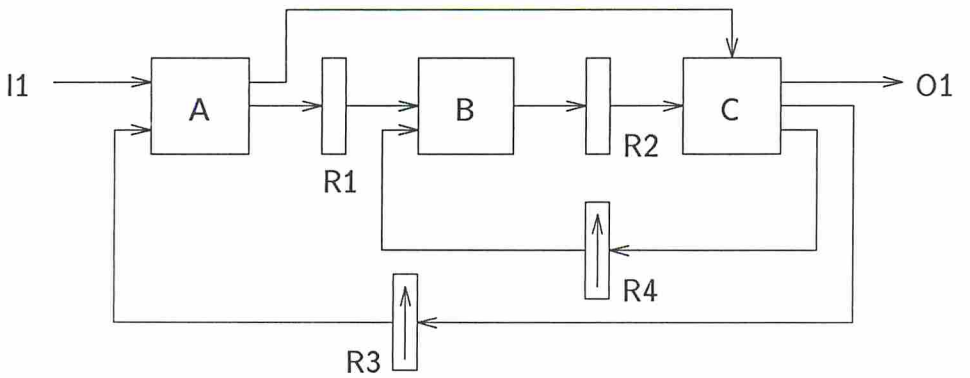
that every detectable fault is single-pattern testable. Test patterns are obtained simply by replacing all FFs in the kernel by wires and running combinational ATPG on the resulting **combinational equivalent**. The number of FFs to be placed in the scan path in this approach, however, is higher than that required to just make the kernel acyclic.

In this paper we study the implications of using a kernel that has an acyclic structure but may be unbalanced. A branch-and-bound algorithm for determining a minimal set of registers to be made scannable such that the kernel is acyclic can be found in [2]. Clearly the area overhead for this case is lower than for a balanced kernel. But a simple combinational equivalent cannot be used for ATPG. Further, any given fault may require up to $d+1$ patterns to detect it.

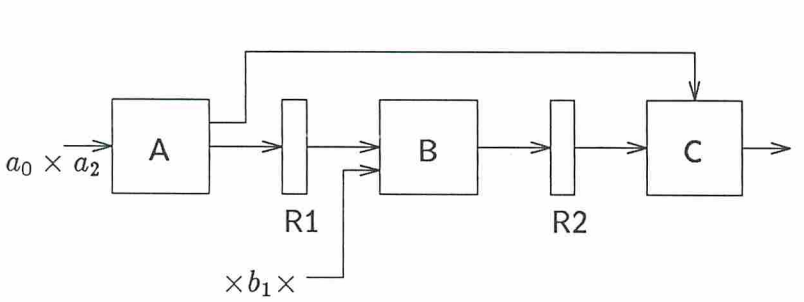
For example, consider the circuit shown in Figure 1(a) consisting of combinational logic blocks A, B, C interconnected with registers. Each connection shown may consist of any number of wires. Registers R3 and R4 are selected to be made scannable since the resulting kernel, shown in Figure 1(b), is acyclic. Note that the kernel is obtained from the original circuit simply by removing the scan registers and replacing them by pseudo-inputs/outputs. In the kernel, inputs/outputs connected to the same combinational logic block are merged together; thus the kernel effectively has one output at C which feeds scan path registers and the primary output O1, and inputs at both A and B are fed by scan registers and/or the primary input I1. The depth of this kernel is 2.

Typical sequential ATPG programs would construct an *iterative array* consisting of up to 3 copies of the kernel, each representing one time frame, and attempt to find a test sequence for a given fault (if one exists) within these time frames. Since the kernel has only one primary output at C, any test sequence for a fault must propagate an error to this output. Assume that the error is visible at the output at time frame 2. Because of the topology of the circuit this output value must depend only on the input values at A at time steps 0 and 2 and on the input value at B at time frame 1. All other input values are essentially "don't-cares" in all possible test sequences and are indicated by '$\times$' in the input sequences in Figure 1(b) (to be applied in the order from left to right). Note that each '$\times$' represents a vector of don't-care values. An ATPG program would normally assign random logic values to these inputs. This means that approximately half the test time in this case could be taken up in shifting random data into the scan path.

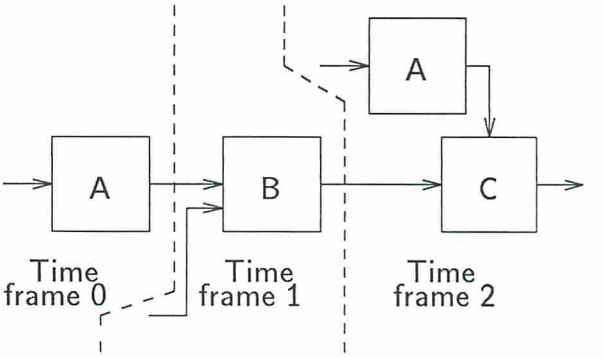
The effectiveness of this test process can be improved in two ways. First, the test pattern generator can be enhanced to fill in the unassigned input values with deterministic patterns that detect one or more additional faults, rather than with random data. In general there is no guarantee that any additional faults can be detected in this way, especially if most of the circuit faults have already been covered. Second, the test sequence can be compacted so that the shifting time is reduced without reducing its effectiveness. The latter approach is the subject of this paper. In the simple example of Figure 1, A has its input value specified at time frame 0 but not at 1, while B has its input value specified at 1 but not at 0. Hence we can combine the first two patterns by shifting a_0 and b_1 simultane-



(a) Partial scan design



(b) Acyclic kernel



(c) Test generation model

Figure 1: Example of partial scan design.

ously into the scan path, holding them there for two clock cycles instead of one, and then shifting in a_2. This is a modification of the basic test procedure described in Section 1. Thus the number of *shift cycles* of the scan path (i.e., the number of times a new pattern is shifted in) is reduced from three to two without losing any of the deterministic part of the test sequence. Note that this applies irrespective of which fault is under test. Intuitively, the fact that there are two "unbalanced" paths from A to D with unequal delays indicates that in general two distinct input patterns at A will be required to guarantee detection of an arbitrary fault.

In Section 2 we present a more formal and general discussion of how to test unbalanced acyclic structures. We use a formal model to compute a lower bound on the number of shift cycles required to test for an arbitrary fault, and present a test compaction algorithm that achieves the lower bound. In Section 3 we study the problem of simplifying the iterative array model used for ATPG by making use of the fact that the kernel is acyclic. For example, Figure 1(c) shows a test generation model (TGM) consisting of a reduced iterative array that is sufficient for any fault in the kernel. The TGM can be further condensed based on the compacted schedule that will be used for test application, as described in Section 3. Conclusions are presented in Section 4.

2 Optimal Test Scheduling

A given test sequence for a partial scan design whose kernel is an acyclic structure of depth d may consist of up to $d+1$ time frames. Our objective is to find a way of compacting the patterns in a test sequence so as to minimize the number of time frames at which new data needs to be applied. This ensures that when the test patterns are applied using the scan path, the shifting time (which usually dominates the test time) is minimized. Assume that the time frames are numbered from 0 (the earliest) to d (the latest, at which the fault gets detected). We define the **schedule** as the list of time frames in the test sequence that require new data to be shifted in, in ascending order. Thus a schedule $(0, 1, 2, \ldots, d)$ means that new data is shifted in at every time frame, while (0) represents a single-pattern test. In the example of Figure 1 presented earlier time frames 0 and 1 are combined together, hence the schedule is $(0, 2)$.

We shall refer to each element in the schedule as a **shift step**, since in the corresponding time frame a new pattern needs to be shifted into the scan path, and each element not in the schedule as a **hold step**, since it requires the contents of the scan path to be held for an additional clock cycle. Note that the test pattern scanned in during a given shift step i in the schedule $(\ldots, i, j, \ldots)$ is actually the result of compacting the test patterns for time frames i, $i+1$, $i+2$, ..., $j-1$.

The Compaction Principle

In our earlier example using Figure 1, we combined the test patterns for time frames 0 and 1 because neither of the inputs at A or B need to have a value specified in *both* frames. This compaction applies to all test sequences in this example. Before studying more complex cases we state the following principle that governs our compaction problem. The term *minimal test sequence* refers to a test sequence in which all unspecified input values are left as don't-care values.

Compaction Principle: *A set of consecutive time frames in a minimal test sequence may be compacted together into a single shift step in a schedule only if no input is required to be assigned values in more than one of these time frames.*

We will apply this compaction principle before test pattern generation is actually carried out. Note that the principle does not make use of the actual values of the test patterns; it uses only the information, derived from the circuit structure, about which input values can be specified and which must be don't-cares in various time frames. In a single-output acyclic structure, such as our previous example, the required information about the input values can be derived using the following rule: An input can be assigned a value in time frame x if there exists a path from that input to the output that passes through $d - x$ FFs (assuming the error is first observed at the output in time frame d). Later in this section we describe how to determine an optimally compacted schedule that satisfies the compression principle based on this information.

Modeling Schedule Constraints

Let us now consider the multi-output structure in Figure 2 in which blocks A, B, C, etc. are combinational and unlabeled blocks are registers. Its depth d is 4 and it has three inputs and four outputs, all of arbitrary width. As before the inputs and outputs are accessible only through a scan path which is not shown. Given an arbitrary fault in the circuit, a test sequence may propagate the fault to any of the outputs. At some outputs it may be possible to detect a fault at a time frame earlier than d. However, note that the same test sequence displaced in time can be used to detect a fault at different time frames. Hence we shall assume without loss of generality that a fault is to be detected at time frame d. This is justified since any fault that is propagated into the scan path at time frame d will be observed during the first shift step for the subsequent test sequence. This assumption will lead to a simplified test generation model discussed in Section 3.

The patterns shown at each input in Figure 2 indicate the time frames at which an input may possibly need to be specified in order to detect a fault at one of the outputs at time frame d. This information is determined in the same way as in the single-output case, except that for a given input, paths to all outputs have to be taken into account. Thus

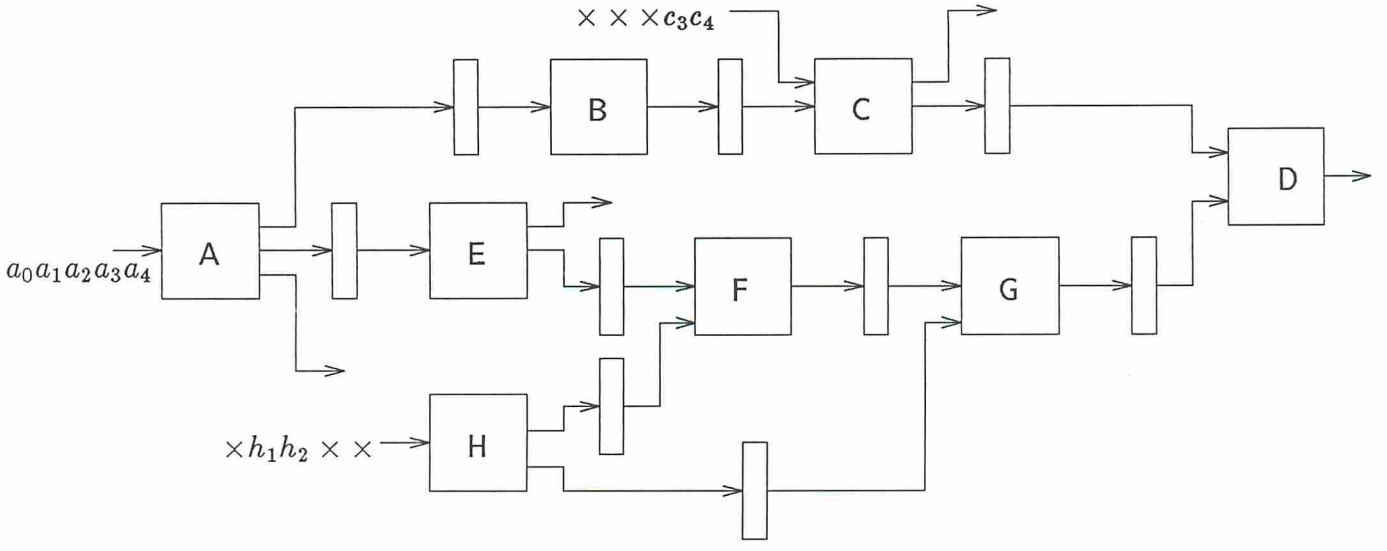


Figure 2: Example of acyclic kernel.

Output	Input values on which the output depends		
	Sequence at A	Sequence at C	Sequence at H
A	× × × × a_4	—	—
C	× × a_2 × ×	× × × × c_4	—
D	a_0 a_1 × ××	× × ×c_3×	×h_1 h_2 × ×
E	× × ×a_3×	—	—

Table 1: Relationships among inputs and outputs.

for example the input to C has values at time frames 3 and 4 but is always unspecified at all other times. The input to A may need specified values at any time frame, because corresponding to each time frame there is some path with the appropriate number of FFs ending in time frame d at one of the outputs. Hence it appears at first glance that the compaction principle will not allow a reduced test schedule.

However, to detect any fault it is sufficient to propagate it to just one of the outputs. If some test sequence for a fault propagates an error to more than one output, this implies that there may exist a reduced form of the same sequence that propagates it to only one output. Table 1 shows, for each output, what input values need to be specified such that the fault is observed at that output at the end of the 4th time frame. The table shows that no test sequence would require all 5 values at A to be specified. Also it is clear that all 5 frames cannot be compacted together, since the output at D (which we shall refer to as D for short) requires two different values at A and also two different values at H. Under these constraints it seems intuitively clear that a bare minimum of two shift steps will be needed in the schedule for an arbitrary test sequence in order to satisfy the compaction principle stated earlier. A model for representing the schedule constraints is described below.

Given an input x and an output y, let $\sigma(x,y)$ be defined as the ordered list of time frames at which the input sequence at x for output y can have specified values. Thus for example Table 1 shows that $\sigma(A,D) = (0,1)$ and $\sigma(H,C) = (\,)$. $|\sigma(x,y)|$ denotes the number of elements in $\sigma(x,y)$. We shall attempt to find a minimal schedule by constructing a **schedule constraint graph** (SCG). We define an SCG as a directed graph $G = (V,A)$ where $V = (0,1,2,\ldots,d)$ represents the set of time frames and an arc (f_1,f_2) in A implies that frame f_1 must occur strictly before frame f_2 in any compacted test sequence. An SCG is constructed using the following procedure, which takes as input the values $\sigma(x,y)$ for all inputs x and all outputs y.

procedure constructSCG (σ)**:** Returns schedule constraint graph, $G = (V,A)$.

```
{
  V ← {0,1,2,...,d}, where d = depth of the circuit;
  A ← { };
  For all input-output pairs (x,y) of the circuit such that |σ(x,y)| ≥ 2 do:
  /* Add constraints corresponding to this input-output pair */
  {
    L ← σ(x,y);
    While |L| ≥ 2 do:
    {
      i ← first element of L;
      j ← second element of L;
      Remove i from L;
      /* Time frames i and j cannot be compacted together */
      For each k, 0 ≤ k ≤ i, do:
        A ← A ∪ {(k,j)};
      For each k, j < k ≤ d, do:
        A ← A ∪ {(i,k)};
    }
  }
}
```

□

For the circuit of Figure 2 the construction of the SCG is illustrated in Figure 3. We begin with the set of nodes $V = \{0,1,2,3,4\}$ and no arcs in A. Referring to Table 1, there are two input-output pairs that may contribute to arcs in the SCG: $\sigma(H,D) = (1,2)$ and $\sigma(A,D) = (0,1)$. The fact that $\sigma(H,D) = (1,2)$ implies that there must be a shift step separating time frames 1 and 2 since distinct test patterns may be required at input H. In terms of constraints on the schedule, this implies that:

1. All time frames up to and including time frame 1 must occur before time frame 2 in the schedule; and

2. All time frames including 2 and beyond must occur after time frame 1 in the schedule.

(a) Constraints for (H, D) only (b) Constraints for (H, D) and (A, D)

Figure 3: Construction of schedule constraint graph.

The first item above contributes arcs (0, 2) and (1, 2), while the second contributes arcs (1, 3) and (1, 4). Thus the constraints due to the input-output pair (H, D) translate into the arcs shown in Figure 3(a), which is the result of the first iteration of the outer 'for' loop in the procedure. In the second iteration the constraints due to the pair (A, D) are added, resulting in the completed SCG shown in Figure 3(b). Note that in the above example it is not sufficient to have only the arcs (0, 1) and (1, 2) in the SCG. By adding the other arcs we are explicitly encoding the fact that although some time frames represented by nodes in V may be compacted with others, they can never be scheduled in reverse order.[1]

In the procedure **constructSCG**, the outer 'for' loop may be repeated for all N_I inputs and all N_O outputs. Within the loop the time complexity is $O(d^2)$, hence the overall time complexity is $O(N_I N_O d^2)$.

Picking a Schedule

The SCG is essentially a representation of information on which time frames may be compacted together and which may not. Based on the SCG we are in a position to make the following statements about the schedules resulting from compaction.

Lemma 1 *Given a sequence of frame numbers, $S = (f_1, f_2, \ldots, f_n)$, where $0 = f_1 < f_2 < \cdots < f_n \leq d$, the compacted schedule denoted by S satisfies the compaction principle if for any arc (a, b) in the SCG, there is some f_i in S such that $a < f_i \leq b$.*

Proof Assume that for all arcs (a, b) in the SCG, there is some f_i in S such that $a < f_i \leq b$. Assume for the purpose of contradiction that the compaction principle is violated by S. Then there must be some input of the circuit that needs distinct values in some time frames a and b that are compacted into the same shift step in S. This implies that the SCG has an arc (a, b). But the fact that a and b are in the same shift step also implies that there is no f_i in S such that $a < f_i \leq b$, which is a contradiction. □

[1] With these constraints encoded explicitly, our problem is actually a special restriction of the equal execution time job scheduling problem [5][p. 402] with the number of processors not less than the number of jobs.

The above lemma essentially means that a given schedule S is valid, i.e., does not violate the compaction principle, if no two time frames that have an arc between them in the SCG are merged within the same shift step.

Lemma 2 *The number of nodes in the longest directed path in the schedule constraint graph is a lower bound on the number of steps in any schedule that satisfies the compaction principle.*

Proof Let P be a longest path in the SCG and let it consist of the δ nodes f_1, f_2, ..., f_δ in sequence. From the construction of the SCG, every arc (f_i, f_{i+1}) implies that if the frames f_i and f_{i+1} were compacted into the same shift step, the compaction principle would be violated. Hence there cannot be less than δ shift steps in any valid schedule. □

In Figure 3(b) the path consisting of nodes 0, 1, 2 is the longest, hence at least three shift steps are required in any compacted schedule.

Our problem is now to find a schedule $(f_1, f_2, \ldots, f_\delta)$ of minimum length that satisfies the following condition: given any arc (a, b) in A, the nodes a and b must not be compacted into the same shift step in the schedule, i.e., there must be an f_i in the schedule such that $a < f_i \leq b$. We present below a greedy algorithm that achieves the lower bound of Lemma 2. It essentially places the nodes of the SCG in levels such that the nodes in the longest path lie in consecutive levels, and all arcs that begin at a particular level end at some higher-numbered level. Then all nodes (time frames) at the same level can be compacted into the same shift step in the schedule.

algorithm schedule ($G = (V, A)$: schedule constraint graph): Returns S, a schedule of minimum length satisfying G.

```
{
  l ← 0;
  While |V| > 0 do:
  {
    l ← l + 1;
    R_l ← nodes in V having no incoming arcs;
    /* R_l consists of consecutively numbered time frames starting with the
         lowest-numbered time frame in V; see proof of correctness */
    Remove the nodes in R_l, along with adjacent arcs, from G;
  }
  /* Final value of l represents number of steps in schedule */
  Return schedule S = (m_1, m_2, ..., m_l) where
    m_i = lowest-numbered time frame in R_i, 1 ≤ i ≤ l.
}
```

□

The sets R_l determined by this algorithm for the SCG of Figure 3 are {0}, {1} and {2, 3, 4}, hence the schedule is (0, 1, 2). The computation involved in computing R_l in

each iteration is of order $O(d^2)$ assuming that an adjacency matrix is used to represent the SCG. Since the number of iterations is bounded by d, the overall complexity is $O(d^3)$. Below we demonstrate that the algorithm **schedule** works correctly in all cases.

Proof of Correctness We need to prove two assertions: first, that S is a schedule satisfying the compaction principle; second, that the resulting schedule is optimal.

Consider the first iteration of the 'while' loop. By construction, the lowest-numbered node in G (i.e., 0 for $l = 1$) cannot have incoming arcs, hence it must be included in R_l. Let this node be r_1. Let the highest-numbered node in R_l be r_2. We will now show that all nodes r such that $r_1 < r < r_2$ must be in R_l. Assume that there is in fact a node v, $r_1 < v < r_2$, that is not in R_l. Then there must be a node $u < v$ with an arc (u, v) in G. Then by construction of G, u must have outgoing arcs to all nodes $v' \geq v$. Hence there must be an arc (u, r_2) in G, which is a contradiction since r_2 is in R_l. Thus R_l represents a group of consecutively numbered time frames starting with the lowest-numbered one currently in V.

After the nodes in R_l are removed from G, the resulting graph is similar in form to G since only a consecutive set of lowest-numbered nodes has been removed. Hence the arguments above can be applied recursively to the resulting graph for the subsequent iterations. Thus every set R_l consists of consecutive time frames. Note also that for any arc in G, the two adjacent nodes cannot be in the same R_l. From Lemma 1 it follows that S is a valid schedule satisfying the compaction principle.

Since all nodes with no incoming arcs are removed in each iteration of the 'while' loop, the length of the longest path must decrease by 1 each time. Thus the number of iterations is equal to the number of nodes in the longest path. According to Lemma 2, this is in fact a lower bound on the number of steps in any valid schedule. Hence the schedule S returned by the algorithm is optimal. □

In this section we have shown how to determine an optimally compacted schedule based on the structure of the acyclic circuit under test. This schedule can be utilized in two ways. First, it can be used in conjunction with a traditional sequential ATPG program to compact each test sequence produced before random data is used to fill in unspecified input values. In the sequences produced by ATPG, the time frame at which the fault is detected may be treated as frame d, and the sequence can be compacted according to the schedule. Some test sequences produced by ATPG may propagate a fault to more than one output. Such sequences should be preprocessed by selecting any one of those outputs and then forcing any input value to don't-cares if the value in that time frame does not influence the selected output at the time of detection.

The second and more efficient way to utilize the schedule is to use it as a guide for test generation itself. In the following section we will show how to construct a restricted test generation model to replace the traditional iterative array used in sequential ATPG.

Test generation on this model will directly result in compacted test sequences for the desired schedule.

3 Test Generation Model

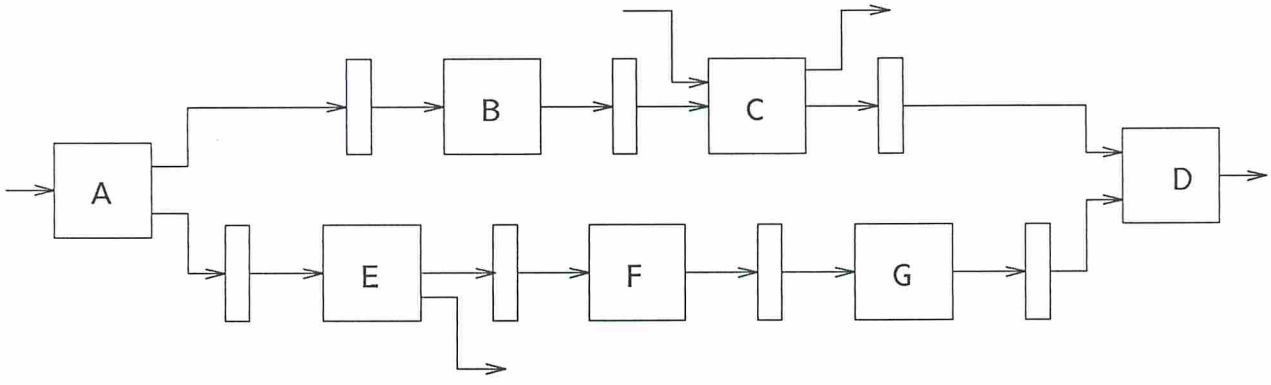
We now turn to the problem of test pattern generation for an acyclic structure. Given an optimized schedule with the smallest number of shift steps, we shall use it to influence the test generation process and simplify it if possible.

In test generation for general cyclic circuits, sequential ATPG programs typically construct an iterative array containing repeated copies of the circuit in order to represent the behavior of the circuit in different time frames [6]. With cyclic circuits the size of the iterative array required to detect an arbitrary fault may grow exponentially with the number of FFs in the circuit. However, in an acyclic circuit every irredundant fault must be detectable within $d + 1$ clock cycles, where d is the depth of the structure, and the complexity of the test generation process is comparable to that for combinational circuits [1]. In fact a simple combinational **test generation model** (TGM) can be derived from the circuit structure, and any combinational ATPG program capable of dealing with multiple faults can be used. Not only is a sequential ATPG program unnecessary, this also avoids the execution overhead in maintaining iterative arrays of various lengths.

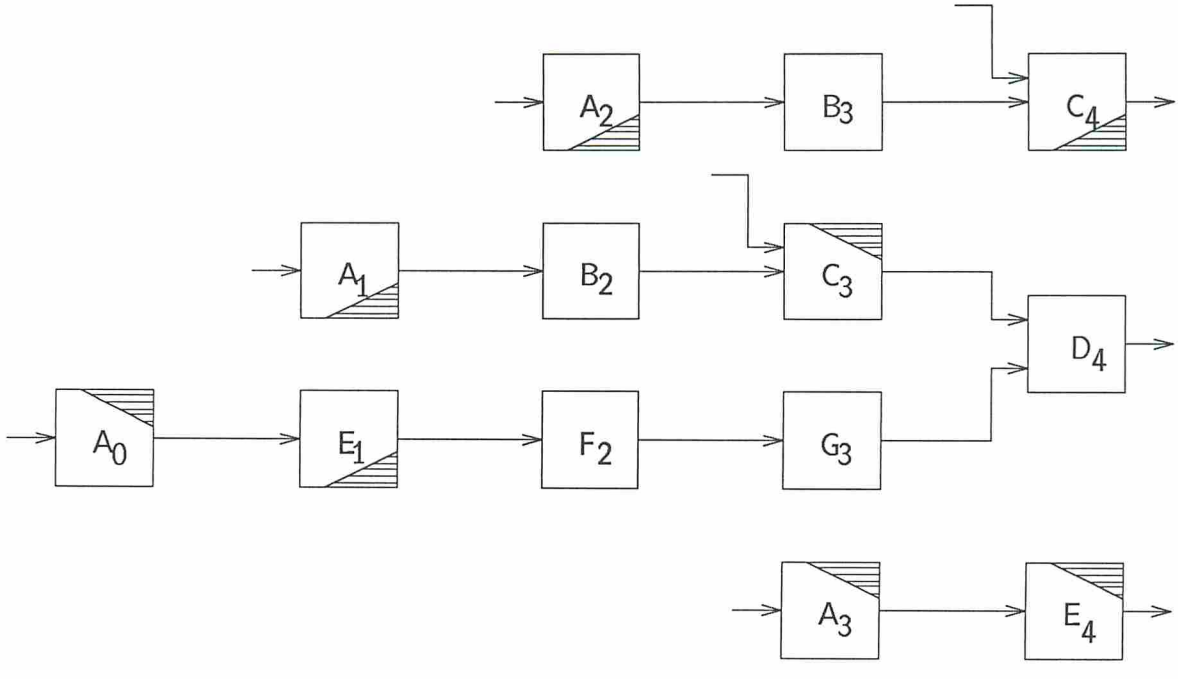
The concept of combinational TGMs is illustrated in Figure 4. Figure 4(a) shows a simplified version of the structure in Figure 2. It has three outputs at C, D and E respectively. We assume that any fault under test will be detected at one of the outputs at time frame 4. Figure 4(b) shows both outputs placed in time frame 4, and the portion of the circuit that feeds each output is laid out in a levelized fashion corresponding to the time frames. Blocks that are required to be in more than one time frame are replicated; thus for example A occurs at several different time frames in the expanded structure since the output values may depend on the behavior of A in various time frames.

Each copy of a repeated block has been pruned to remove any logic that will not be used for test generation; this is indicated by shaded regions but will not be explicitly shown from now on. The subscripts on A_0, E_1, etc. refer to the time frames in which the corresponding instances of the logic blocks exist; the highest subscript is clearly the depth $d = 4$. All registers in the expanded structure have been replaced by wires and the resulting TGM is combinational. This is the general form of the TGM before any compaction; we shall refer to it as the **basic TGM** and it represents the schedule $(0, 1, 2, \ldots, d)$.

In order to generate a test for a fault in the original sequential circuit, the fault must first be mapped to the set of corresponding fault instances in the combinational TGM. (This is analogous to the modeling of faults in iterative arrays.) Ordinary combinational ATPG can now be carried out on the TGM. The test pattern obtained can be transformed into a test sequence for the sequential circuit using the following rule: all input patterns

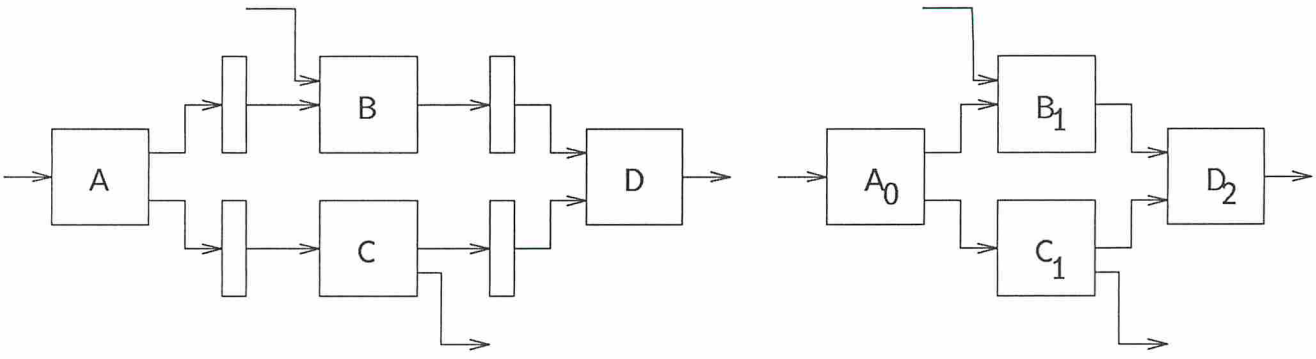


(a) Acyclic structure



(b) Basic TGM

Figure 4: Basic test generation model.



(a) Balanced structure (b) Combinational equivalent

Figure 5: Balanced structure and its test generation model.

at logic blocks with subscript i must be used as the ith pattern in the test sequence. This of course applies if the schedule $(0, 1, 2, \ldots, d)$ is used with no compaction.

Condensing the Test Generation Model

When the test schedule is compacted as described in Section 2, not only is the test time per test sequence minimized, but we can also take advantage of the compacted schedule to condense the TGM. For the special case of *balanced structures* [2] such as the one shown in Figure 5(a) it has been shown that a single pattern is always sufficient for detecting any fault, i.e., the optimal schedule is always (0). The TGM for this class of structures is simply the combinational equivalent of the structure formed by replacing all FFs by wires as shown in Figure 5(b). Thus each logic block appears only once in the TGM, and only single faults need to be considered during combinational ATPG. However, this is not the case with general unbalanced structures. Given the schedule to be used, we shall show how a maximally condensed TGM can be derived. We shall prove that provided the schedule satisfies the compaction principle, the condensed TGM is sufficient for complete test pattern generation.

In condensing the TGM we begin with the basic TGM for the schedule $(0, 1, 2, \ldots, d)$ and modify it based on the schedule provided. We essentially utilize the fact that each input pattern applied to the kernel at a shift step in the schedule is also applied during the subsequent hold steps. The condensation process can be carried out by repeating the following two steps which are illustrated in Figure 6 for the circuit of Figure 4. The term **schedule interval** refers to a shift step and its subsequent hold steps.

Step 1: Repeat the following for each schedule interval. If any input signal occurs at more than one time frame in the same interval, connect the different copies together by fanning out the earliest copy of the signal (i.e., the one

occurring in the lowest-numbered time frame) to the other copies so that only one copy of the input signal remains within the interval.

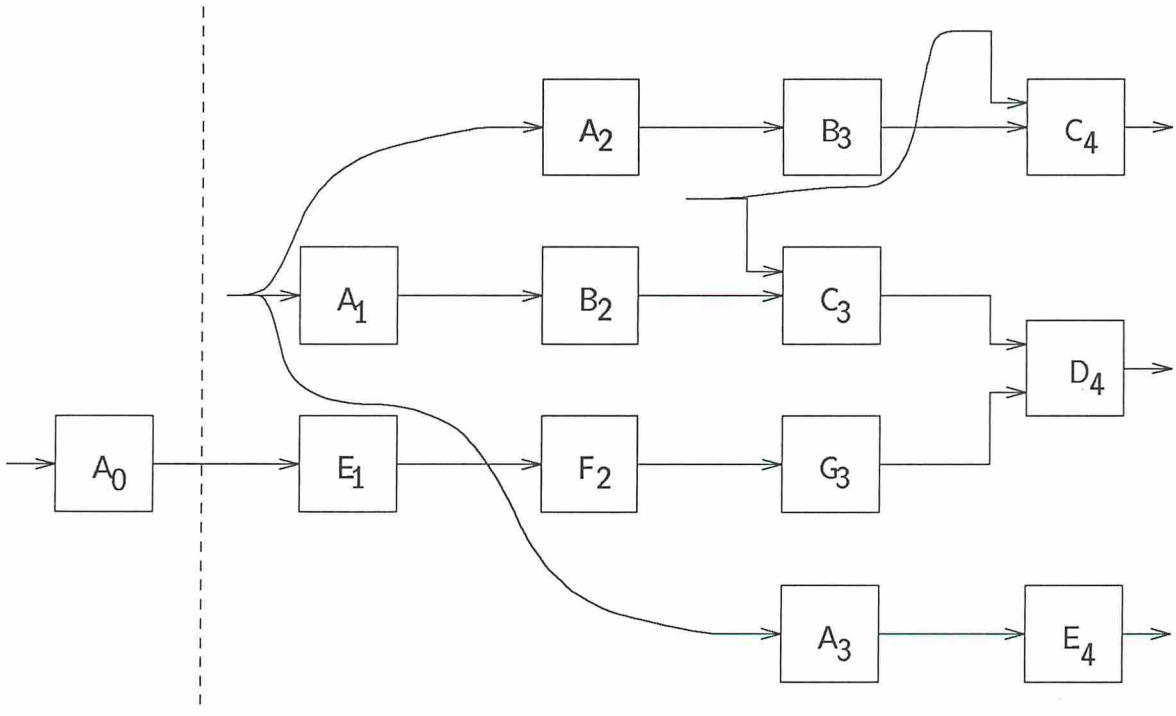
For example, consider the circuit of Figure 4(a) for which (0, 1) is an optimal schedule. Figure 4(b) shows the basic TGM which is to be condensed. The schedule (0, 1) has only one interval containing more than one step, namely the one consisting of time frames 1, 2, 3 and 4. In this interval the input feeding logic block A occurs three times, hence these inputs are connected together as shown in Figure 6(a). Similarly the inputs to C in time frames 3 and 4 are connected together. Note that A_1, A_2 and A_3 now receive identical inputs and in fact they represent exactly the same behavior extended over three clock cycles. The following operation will replace them with one merged copy in A_1.

Step 2: Repeat the following operation until no further changes can be made in the TGM. Let β be a logic block in the circuit and let $\beta_{i_1}, \beta_{i_2}, \ldots, \beta_{i_n}$ be different copies of β such that $i_1 < i_2 \cdots < i_n$ and every signal feeding an input of β_{i_1} also fans out to the corresponding input of each of $\beta_{i_2}, \ldots, \beta_{i_n}$. Then remove $\beta_{i_2}, \ldots, \beta_{i_n}$ from the TGM and fan out each output of β_{i_1} to all the signals originally fed by the corresponding outputs of $\beta_{i_2}, \ldots, \beta_{i_n}$.

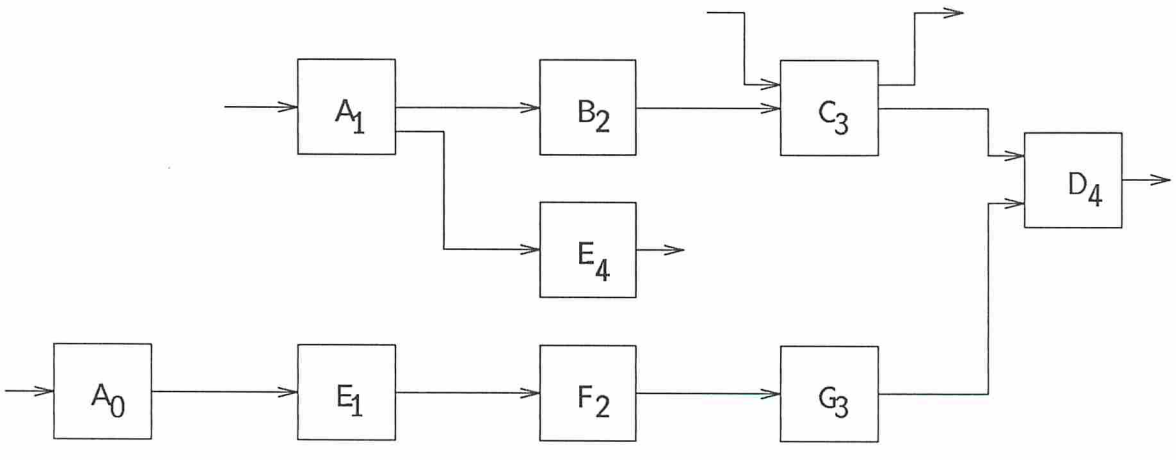
At first this step can be applied to remove A_2 and A_3 and fan out the output of A_1 to B_3 and E_4 as well. Because block A must have exactly the same behavior in time frames 1, 2 and 3, we have simply combined the three copies for the purpose of ATPG, and fanned out the outputs appropriately. Note that this does not alter the execution of the test in any way; it only incorporates some information present in the test schedule into the test generation process, reducing the amount of analysis carried out during ATPG. In the resulting structure both B_2 and B_3 are fed by A_1, hence they can be merged into B_2. Finally in a similar way C_4 can be merged into C_3. No further merging is possible and the final condensed TGM for the schedule (0, 1) is shown in Figure 6(b).

The condensation steps can be applied to any basic TGM, given a schedule, to yield a condensed TGM. Figure 5(a) shows an acyclic structure that has the balanced property, for which the optimal schedule is (0); Figure 5(b) shows the condensed TGM resulting from the procedure described above. As expected the TGM is a simple combinational equivalent of the original structure.

The computation complexity of the condensation steps depends on the implementation and on the level of description of the circuit. The computation in Step 2 can be minimized by forming maximally connected clusters of combinational logic blocks, and carrying out the circuit manipulations on this form of the circuit. Later the lower-level logic descriptions can be filled in for each high-level block, pruned where necessary as described earlier.



(a) Condensation step 1



(b) Condensation step 2

Figure 6: Condensation of test generation model.

Test Pattern Generation

Given an acyclic structure C, a schedule $S = (\phi_1, \phi_2, \ldots, \phi_n)$, and a condensed TGM T_C for C based on the schedule S, the following procedure can be used to generate a test for an arbitrary fault f in C. Let f_C be the corresponding fault (possibly a multiple fault since some logic may be replicated) in T_C. Let us assume that f_C is detectable in T_C; then ordinary combinational ATPG can be used to derive a test pattern for f_C in T_C. Note that due to the nature of the condensation process, any given input signal to C can occur in T_C only at time frames ϕ_1, ϕ_2, etc. in S. For each ϕ_i, let p_i be an input pattern containing the values of all inputs that occur in time frame ϕ_i in T_C, and containing don't-care values for all other inputs. Then the sequence of patterns $(p_1, p_2, \ldots, p_n)$, if applied according to the schedule S, will detect f in C.

In order to justify the use of the condensed TGM we need to validate the assumption that if f is detectable in C then f_C is detectable in T_C. This is done by the following theorem.

Theorem 1 *Given a fault f detectable in an acyclic circuit C, and given a schedule S that satisfies the compaction principle, the corresponding fault f_C (possibly multiple) is detectable in the condensed TGM T_C.*

Proof Let T_B be the basic TGM of C and let f_B be the fault (possibly multiple) corresponding to f in T_B. Since f is detectable in C, f_B must be detectable in T_B using some test pattern τ_B. Suppose the error is propagated to output Ω_d in T_B. Then the cone of logic feeding Ω_d in T_B has certain input values in τ_B that constitute a sufficient test pattern τ'_B for f_B, irrespective of the other input values. Note that in τ'_B, no input signal takes on more than one distinct value within the same schedule interval, otherwise the compaction principle would be violated by the schedule S. Hence for every input signal in the condensed TGM T_C there is a unique value that can be applied to it in every schedule interval in order to simulate the behavior of T_B. Let the input pattern formed by these values be τ_C; note that it is a condensed form of τ'_B. Since τ'_B detects f_B, it must cause different output values at Ω_d in the good and faulty versions of T_B. Hence τ_C must cause different output values at Ω_d in the good and vaulty versions of T_C. Thus f_C is detectable in T_C. □

The above theorem proves that for any detectable fault in C, a test sequence that follows the schedule S can be generated using combinational ATPG on T_C. This leads to the following corollary.

Corollary *Given an acyclic circuit C and a schedule S that satisfies the compaction principle, a complete test pattern set for the condensed TGM T_C results in a complete test sequence set for C using the schedule S.* □

We have thus shown that the condensed TGM derived in this section is a sufficient and complete model for test generation. The size of the TGM is lower than the corresponding iterative array used by traditional sequential ATPG programs. By condensing the TGM for the given schedule to be used in applying the test, some redundant computations during the test generation process are eliminated. If the schedule is minimal, the model guarantees that an arbitrary fault can be detected using the smallest possible number of shift steps.

Note that in this paper we have focussed on the problem of generating minimal test sequences for given faults. Test generation programs typically assign random values to unspecified values in the input patterns, and then run fault simulation to drop additional faults. The TGM derived here is not suited to fault simulation, except when the schedule is (0), since it does not consider errors propagated to the scan path in intermediate shift steps. Note however that sequential fault simulation is not as hard a problem as test generation and any sequential fault simulation tool can be used in conjunction with ATPG using the condensed TGM derived here.

4 Conclusion

In this paper we have studied the problem of testing acyclic structures in partial scan designs. We have presented a new approach to test sequence compaction in which the objective function is the number of distinct patterns to be shifted into the scan path per test sequence. In our approach, each test sequence is compacted into the smallest number of patterns needed to be shifted into the scan path. This leads to the lowest test time to detect an arbitrary fault. An algorithm for determining the optimal schedule, based on the structure of the circuit, was presented.

We have also presented a specialized test generation model (TGM) for acyclic structures. Like the iterative array model, this model reduces the ATPG problem to that of combinational ATPG with multiple faults. A special feature of this model is that it uses the optimal schedule determined separately in order to derive a condensed TGM. This leads to fewer redundant computations during ATPG. However, a separate sequential fault simulator is required. The optimal scheduling algorithm and the test generation model can be used to significantly reduce the testing costs in partial scan designs, especially for signal processing and pipelined circuits.

Acknowledgement

The authors wish to thank Kuen-Jong Lee and Rajiv Gupta for helpful suggestions.

References

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

[2] Rajesh Gupta, Rajiv Gupta, and M. A. Breuer. BALLAST: a methodology for partial scan design. In *Proceedings, Fault-Tolerant Computing Symposium (FTCS-19)*, pages 118–125, June 1989.

[3] K.-T. Cheng and V. D. Agrawal. An economical scan design for sequential logic test generation. In *Proceedings, Fault-Tolerant Computing Symposium (FTCS-19)*, pages 28–35, June 1989.

[4] Rajesh Gupta, Rajiv Gupta, and M. A. Breuer. An efficient implementation of the BALLAST partial scan architecture. In *Proceedings, International Conference on Very Large Scale Integration (VLSI-89)*, pages 133–142, Munich (West Germany), August 1989.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

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