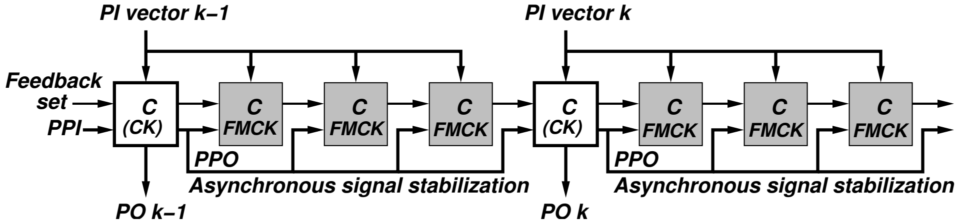


Figure 8.19: Two-phase time-frame expansion for asynchronous circuits.

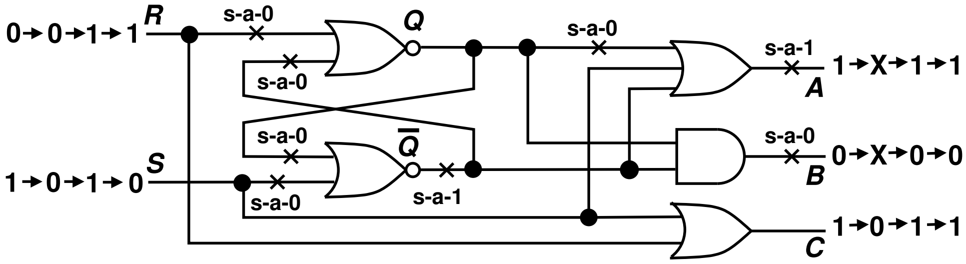


Figure 8.20: An asynchronous circuit for test generation.

2. *Fast modeling clock (FMCK) phase.* Following the system clock phase, which provides new inputs to the combinational logic, a series of "fast" time-frames exercise the logic until signals become stable. For practical reasons, a small fixed number of time-frames is used (three in Figure 8.19.) If some signal does not become stable, then an oscillation can be assumed and that signal is set to the unknown state. During this phase, the primary inputs and the clocked flip-flop states are held without change, and no primary output is examined for fault detection.

Thus, each input vector produces a new set of PPO and asynchronous states (feedback signal states.) In a simplified version of asynchronous circuit test generation, we ignore the feedback states. Each input vector then begins with feedback signals in the unknown states. The test generator favors those primary input values that uniquely determine the feedback signals. For example, for the NOR latch of Figure 8.16(b), inputs 01 and 10 will be favored. Also, the loops are preprocessed and a *sensitization value set*, i.e., a set of signal values that sensitize a path around the loop, is generated for each loop. For example, the sensitization set for the NOR latch of Figure 8.16(b) is $S = R = 0$. Then, whenever the signal states coincide with the sensitization set, outputs of all gates in the loop are set to X. This approach avoids any oscillatory evaluation of signals. However, the conservatism prevents the generation of tests where asynchronous states are essential. This approach is used in GENTEST [72], as the next example illustrates.

Example 8.7 *Consider the asynchronous circuit shown in Figure 8.20. We will discuss test generation by the memory-less model. The feedback set is $(Q, \overline{Q})$. The loop sensitization condition is $R = S = 0$. Thus, whenever this signal combination* *occurs, the test generator sets the feedback set to the unknown state,* $Q = \overline{Q} = X$*. Gentest [72] produced the following result on a SUN Sparc 2 workstation:*

```
Primary inputs = 2
Primary outputs = 3
State elements = 0
Total Faults = 23
test generation time = 33 ms
fault simulation time = 16 ms
total vectors = 4
detected faults = 15
untestable faults = 8
undetected faults = 0
0 untestable faults were potentially detected
0 undetected faults were potentially detected
faults tried = 12
time limit per fault = 0.8 ms
fault coverage = 65.2%
```

The four test vectors, corresponding outputs, and eight untestable faults are shown in Figure 8.20. We make several observations:

- *Not all faults identified as untestable are really untestable. They are really untestable by a single-vector test, which is a limitation of the combinational model. For example, the s-a-0 fault on the Q input of the OR gate A is testable by two vectors, (S,R) = (1,0), (0,0). Still, there are several faults that are either not detectable even by multiple vector tests, or can only be detected potentially or as race faults. A generally low fault coverage is quite typical of asynchronous circuits.*

- *Fortunately, the generated test sequence does not cause a race condition in the fault-free circuit, which is a requirement for useful tests but is not imposed by the test generator. For example, if we generate tests with the fault list ordered as "C s-a-0" followed by "s-a-0 on S input of A," then the two tests* $S = R = 0$ *and* $S = R = 1$*, applied in that order, will produce a race in the NOR latch in the fault-free circuit. If the asynchronous logic is embedded in a sequential circuit, the ordering of vectors cannot be arbitrarily changed. Such race conditions should be found by a simulator and the vectors causing them should be discarded or modified. Alternatively, the test generator should recognize the race producing sequences and generate alternative tests.*

Asynchronous circuits continue to be difficult to test. Tools and techniques are only adequate for small circuits. The typical situation often encountered involves large synchronous circuits with a small amount of asynchronous circuitry embedded in the combinational logic. In addition, tests for faults in the clock circuitry require asynchronous techniques. The major difficulty of finding good tests for asynchronous circuits arises due to the inadequate treatment of delays. Analysis of races and hazards can improve the tests, but requires additional computation [96, 141]. Determining the steady-state without the complete delay information can be troublesome, too. A recent method gives specific attention to the time-frames used for signal stabilization [64]. Finally, when it comes to handling of delays, logic simulators are more advanced than test generators and the early proposal of Seshu and Freeman [587] for simulation-based test generation is still attractive.

8.3 Simulation-Based Sequential Circuit ATPG

The application of a fault simulator for test generation was suggested by Seshu and Freeman in the early 1960s [587]. They used a compiled-code simulator and the faults were *serially* injected. Random vectors were used in the 1970s with fault simulation to select only those vectors that increased the fault coverage [26]. While this strategy was quite successful with some combinational circuits, for hard to test circuits it had to be backed up with algorithmic (non-random) vectors. Breuer [89] devised a simulation-based method for sequential circuits. In his method, several randomly generated vectors were simulated for some "present state" of the circuit and the best vector (according to specified criteria) was included in the test sequence. The circuit state was then advanced before simulating a new set or random vectors. Schuler *et al.* [573] were the first to use a *concurrent fault simulator* (CFS) for test generation. They simulated a set of random vectors. Each vector was simulated for the same given starting state of the circuit. The vector that detected the largest number of faults was selected. The states of the good and all faulty circuits were changed corresponding to the selected vector. The test generator then advanced to the selection of the next vector from a new set of random vectors. Parker [508] reported an adaptive method of making the random vector source circuit-specific.

One of the greatest advantages of these methods is that before a vector is selected as a test, it is simulated. As an event-driven simulator analyzes both logic and timing behavior of the circuit, the selected vector is guaranteed to be free from harmful races or hazards. Many other test generators completely neglect timing information and produce hazardous tests.

Several observations were made by Schuler *et al.* [573]. They experienced a serious shortage of available memory required to simulate a large number of faults. They suggested using a small subset of faults. However, the problem of finding a proper subset had no existing solution. They also reported that for a given computing time, the fault coverage remained somewhat low unless extra observation points were inserted in the circuit. Their circuits contained up to 1,000 gates and were small by today's standard.

The problem of low fault coverage when no extra observation points are inserted has been reported by other workers as well [343, 361]. These authors did not use CFS. However, the difficulty lies not in the simulation algorithm, but in the way vectors are selected. The vector that detects either the target fault or the largest number of faults at primary outputs is the natural choice. When the faults are very difficult to detect, none of the trial vectors may detect anything. Selection of a test vector from a reasonable number of random vectors is very inefficient in this situation. Also, when the circuit is sequential, not every test vector may produce a fault effect at primary outputs. In general, several vectors are required to bring the circuit to a state such that the fault can be activated. Again, several vectors may be needed to propagate the effect of the fault to a primary output. Thus, a test for a fault may consist of a sequence of vectors where only the last vector produces a fault effect at a primary output. Generation of such a sequence is highly improbable by a process that only considers single vectors and relies on fault detection information at primary outputs. Output observation can produce better results if vector sequences, like those produced by a genetic algorithm, are used instead of single vectors (see Subsection 8.3.2.)

The above observations motivated further exploration. A fault simulator computes the fault activity at all internal nodes of the circuit. This information is frequently ignored since we use the simulator only to gather fault detection data. Takamatsu and Kinoshita [649] have used CFS for generating tests for combinational circuits. They use the test generation algorithm, PODEM (see Chapter 7), to generate a test for some target fault. PODEM involves a series of backtrace and forward implication operations. The backtrace determines a value for some primary input to accomplish some objective like fault activation or fault propagation. Forward implication ascertains that the input value determined by the backtrace does not contradict the objective. In case of a contradiction, the input value must be changed via backtracks. For an algorithm like PODEM, which enumerates primary input values until a test is found, the CPU time of test generation largely depends upon the number of backtracks. Takamatsu and Kinoshita find that a backtrack can often be avoided by changing the target fault. In their CONT-2 algorithm, the forward implication is similar to CFS. Thus, fault activity information about all undetected faults is available. When a contradiction occurs, CONT-2 will abandon the current target fault and select some other target fault that has a greater chance of being detected. The new target can be a fault that is already active and whose effect is present at some signal close to a primary output. PODEM and CONT-2, which is based on PODEM, are test generation algorithms applicable only to combinational circuits.

The CONTEST algorithm, devised by Agrawal, Cheng, and Agrawal [31, 153], uses a directed-search approach for generating tests for sequential circuits. This algorithm works in a closely knit fashion with CFS. The basic idea is to obtain test vectors by successive modification of primary input bits based upon cost functions that are computed by the simulator. More advanced test generators use genetic algorithms [445].

8.3.1 CONTEST Algorithm

The test generation process can be subdivided into three phases. In Phase 1 initialization vectors are generated. The purpose of these vectors is to bring flip-flops in the circuit to known states irrespective of their starting state. Phase 2 begins with vectors that are either supplied by the designer or generated in Phase

1. A fault list is generated in the conventional manner. For example, this list may contain all single stuck faults or a subset of such faults. These faults are simulated using a fault simulator. If the coverage is adequate, the test generation would stop. Otherwise, tests are generated with all undetected faults as targets. In the initial stages of test generation, the fault list is usually long and the objective of this phase is to generate tests by concurrently targeting all undetected faults. At the end of Phase 2, if the fault coverage has not reached the required level then Phase 3 is initiated. In this phase, test vectors are generated for single faults targeted one at a time.

Phase 1: Initialization. Here, the cost is defined simply as the number of flip-flops that are in the unknown state. Initially, the cost may be equal to the number of flip-flops in the circuit. The goal in the initialization phase is to reduce this cost to 0. This cost function is derived only from good circuit simulation and is not related to the faulty circuit behavior. If the circuit is hard to initialize, one may relax the criterion for exiting to the next phase by allowing a small number of flip-flops, say 10%, to remain uninitialized.

Using this cost function, the circuit is driven to the easiest initialized state instead of any specified state. All flip-flops are assumed to be in the unknown state at the beginning and the cost function is equal to the number of flip-flops in the circuit. Before applying a trial vector, signal states are saved for restoration in case the trial vector is not accepted. To start the process, any trial vector (a randomly generated or user-supplied vector) can be used. This is called the "current vector." Subsequent trial vectors are generated by changing the bits of the current vector. The integer n is used to denote the bit position in the current vector that is changed. The clock bits (only in the synchronous mode) are treated separately. The user specifies the clock sequence. During simulation, the input data bits are kept fixed whenever the given clock sequence is applied. In combinational or asynchronous sequential circuits, all input bits are treated as data.

After simulation of a trial vector, the "trial cost" is computed as the number of flip-flops that are in the unknown state. If the trial cost is lower than the current cost, then the trial vector is saved. If the trial cost is zero, then the initialization phase is complete. Otherwise, the current cost is updated, signal states are saved, the accepted trial vector becomes the current vector, n is set to 1, a new trial vector is generated by changing the nth data bit, and the process of simulation, cost calculation, etc., is repeated.

A trial vector is not accepted as an initialization vector if the corresponding trial cost is not lower than the current cost. In that case, the bit number n is advanced and the process is repeated with a new trial vector. When all bits of a current vector have been changed without lowering of the cost, this process will stop, indicating that initialization is impossible with this scheme, i.e., a local cost minimum is reached. One possible strategy is to restart with a new randomly-selected current vector.

Phase 2: Concurrent fault detection. The initialization vectors may already have detected some faults. Some others may have been activated but not detected. As a result, effects of active faults will be present at internal nodes of the circuit. For

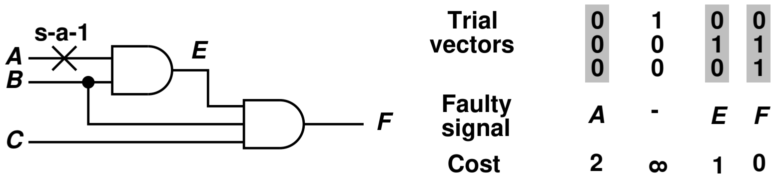


Figure 8.21: An example of the distance cost function used in Phase 2.

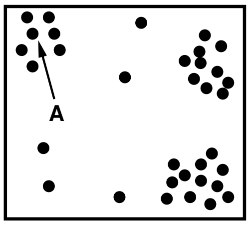
an active fault, a suitable cost function is the shortest distance to a primary output from any fault effect caused by the fault. The distance here is simply the number of logic gates on the path. The smaller this cost, the closer the fault is to being detected. When a fault is detected, its cost will be zero. The objective in test generation is to reduce the cost by propagating the fault effect forward, gate by gate, until it reaches a primary output. If the fault is not activated, i.e., no fault effect is present anywhere in the circuit, then the cost is defined as infinite.

Figure 8.21 gives a simple example to illustrate how the *distance* cost function works. The given fault is signal A stuck-at-1 and the initial vector is 000. The fault effect appears at signal A; thus, initial cost is 2. After simulating three trial vectors, the search terminates and a test is found.

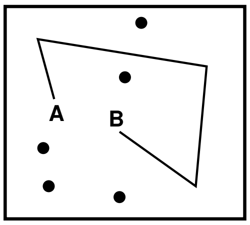
When there are several undetected faults, cost C_i is computed for each fault i for some input vector and internal state. Similarly, the cost C'_i is obtained for a candidate trial vector. A comparison of C_i and C'_i determines whether to accept the candidate vector or reject it. Since there can be several undetected faults, there are two *lists* of cost functions instead of just two numbers. The search for tests should be guided by a group of faults instead of a single target fault. One can devise simple rules to determine the acceptance of a vector. For example, if the combined cost of 10% of the lowest-cost undetected faults is found to decrease, then the new vector may be accepted. Experience has shown that for many circuits, the test vectors for all stuck-at faults are usually clustered instead of being evenly distributed in the input vector space. Figure 8.22 shows the input vector space with dots representing tests for undetected faults from the fault list. In the beginning there are many undetected faults and the vector space may have large clusters of tests as shown in Figure 8.22(a). Starting at any initial vector A, the cost function will steer the search toward large clusters. When only a few faults are left, their tests will be a few isolated vectors. In Figure 8.22(b), test generation in Phase 2 has followed the path from A to B. At B, the combined cost provides very little "direction." Hence, a single target fault strategy may be needed.

If flip-flops are modeled as functional primitives, they may be treated differently from individual gates such as AND or OR. Propagating a fault through a gate only needs setting appropriate values at the inputs of the gate. In contrast, propagation through a flip-flop requires first setting the appropriate value at its data input and then activating the clock signal. In cost computation, therefore, a large constant, say 100, is assigned to a flip-flop as its distance contribution.

Phase 2 begins with fault simulation of initialization vectors. The faults thus



(a) Many undetected faults.



(b) Few undetected faults.

Figure 8.22: Directed search in Phase 2 and the need for Phase 3.

detected are eliminated from consideration. Generation of trial vectors is performed in a manner similar to the initialization phase. Costs for trial vectors are obtained from the result of CFS. Prior to simulation, gates are levelized starting from primary outputs. Thus the level of a gate directly gives its distance from primary outputs. The cost of a trial vector is easily computed using the levels of the faulty circuit gates. The concurrent phase stops when all single bit changes in a current vector produce no cost reduction. This will normally happen when the number of faults left in the target set is small. The test vectors for these faults sparsely populate the vector space and, therefore, the collective cost function does not provide any significant guidance.

Phase 3: Single fault detection. The cost function in this phase is based on a SCOAP-like testability measure. In SCOAP (see Chapter 6), each signal is assigned three measures, 1-controllability, 0-controllability, and observability, respectively. All measures are integer-valued and a higher value of the measure for a signal indicates that it might be difficult to control or observe. In Phase 3, testability measures are dynamically computed. Their values depend upon the circuit structure as well as on input vectors. The dynamic nature is essential in this application since the measures are used to compare the suitability of vectors for detecting a target fault.

$DC1(i)$ and $DC0(i)$, are defined as dynamic 1 and 0 controllabilities for node i. These are related to the *minimum* number of primary inputs that must be *changed* and the *minimum* number of additional vectors needed to control the value of node i to 1 or 0. The number of inputs required to be changed is further defined as the *dynamic combinational controllability* (DCC) and the number of required vectors is called the *dynamic sequential controllability* (DSC.) In order to keep the test sequence short, DSC is weighted heavier than DCC. For example, $DC1(i)$ and $DC0(i)$ could be the weighted sums of DCC and DSC, say, DSC times 100 plus DCC. If the current logic value of node i is 1, then $DC1(i)$ is defined as:

$$DC1(i)\ |_{V(i)\,=\,1} \quad = \quad 0 \tag{8.1}$$

where $V(i)$ is the logic value on node i. Similarly, if the current logic value on node i is 0,

$$DC0(i)\ |_{V(i)\,=\,0} \quad = \quad 0 \tag{8.2}$$

This definition follows from the fact that no input change is needed to justify a 1(0) on node i if the value is already 1 (0.) Under other conditions, $DC1(i)$ and $DC0(i)$

will assume nonzero values. For example, for the output line i of an AND gate with m inputs, $DC1(i)$ and $DC0(i)$ are computed as:

$$DC1(i)\mid_{V(i)\,=\,0\ or\ X} \;=\; \sum_{j=1}^{m} DC1(k_j) \tag{8.3}$$

$$DC0(i)\mid_{V(i)\,=\,1\ or\ X} \;=\; \min_{1\le j\le m} DC0(k_j) \tag{8.4}$$

where k_j is the jth input line of the gate. Here, min means the minimum of m quantities and its use is similar to that in the SCOAP testability measures. Primary input controllabilities are set to 0 or 1 depending on their current state. As explained, the controllability of sequential elements is weighted heavier. Dynamic controllabilities for a flip-flop output i are defined as:

$$DC1(i)\mid_{V(i)\,=\,0\ or\ X} \;=\; DC1(d)\ +\ K \tag{8.5}$$

$$DC0(i)\mid_{V(i)\,=\,1\ or\ X} \;=\; DC0(d)\ +\ K \tag{8.6}$$

where d is the input data signal of the flip-flop and K is a large constant, say, 100.

In order to detect a stuck fault, the test generator must first find a sequence of vectors to activate the fault, i.e., set the appropriate value (opposite of the faulty state) at the fault site, and then find another sequence to sensitize a path to propagate the fault effect to a primary output. Thus, the cost function should reflect the effort needed for activating and propagating the fault. The activation cost, $AC(i_{s-a-j})$ of a node i stuck-at-j fault is defined as:

$$AC(i_{s-a-j}) \;=\; DCv(i) \tag{8.7}$$

where v is 0 if node i is stuck-at-1 and v is 1 if node i is stuck-at-0. This follows from the consideration that the cost of activating a stuck-at-0 (stuck-at-1) fault is the cost of setting up a 1 (0) at the fault site. The propagation cost is basically a dynamic observability measure. For a fanout stem i with n fanout branches, it is:

$$PC(i) \;=\; \min_{1\le j\le n} PC(i_j) \tag{8.8}$$

where i_j is the jth fanout branch of i. For an input signal i_a of an m-input AND gate whose output signal is i, we have

$$PC(i_a) \;=\; PC(i)\ +\ \sum_{\substack{1\le k\le m\\ k\ne a}} DC1(i_k) \tag{8.9}$$

where i_k is the kth input line of the gate. Similar formulas are easily derived for other types of gates. The cost function for test generation for a single target fault is derived from the activation cost and the propagation cost defined above. For an undetected fault F, line i stuck-at-v, that is not activated, the cost is defined as:

$$Cost(F)\ |_{F\ not\ activated} \quad = \quad K1 \ \times \ AC(i_{s-a-v}) \ + \ PC(i) \tag{8.10}$$

where $K1$ is a large constant that determines the relative weighting of the two costs. If the fault has been activated, and N_F is the set of the nodes where the fault effect appears, then:

$$Cost(F)\ |_{F\ activated} \quad = \quad \min_{i\ \in\ N_F} PC(i) \tag{8.11}$$

Notice that in this cost function reconvergent fanouts are ignored. This approximation provides computational simplicity but may occasionally result in failure to detect a fault. Once activation and propagation costs are computed for each fault in the list, the lowest-cost fault is targeted for detection. New input vectors are created to further lower the cost of the target fault until it is detected or the search is abandoned due to a local cost minimum. As new vectors are added to the sequence, CFS eliminates any other detected faults from the list. This phase ends when either an adequate coverage is achieved or all faults that were left undetected at the end of Phase 2 have been processed.

A program implementing the CONTEST algorithm (CONcurrent TEst generator for Sequential circuit Testing) [31, 153] accepts a logic-level circuit description in a hardware description language. The test generator works in two modes: *synchronous* and *asynchronous*. In the synchronous mode, clock signals and their transition sequence within a period must be specified. The test generator follows each change in primary inputs by a clock sequence. In the asynchronous mode, no clock signal is identified and the test generator treats all primary inputs alike. For circuits that are largely synchronous with a limited amount of asynchronous circuitry, test generation in the synchronous mode works better initially. If the coverage by this mode is inadequate, then, for the remaining faults, asynchronous mode can be used. This is because the speed of test generation depends upon the number of primary inputs that must be manipulated. In the synchronous mode, clock signals are prespecified and are not manipulated.

An optional fault list is an input to the test generator. If this is not given, the system generates a list of collapsed single stuck-at faults. CONTEST contains an event-driven CFS. Race analysis in feedback structures is automatic and is performed through special modeling features. By default, potentially detectable faults (that produce an unknown faulty output) are considered detected. This option can be turned off by the user. If the number of changes in a signal for the same input vector exceeds a prespecified number, then the simulator assumes oscillation and sets the signal to the unknown state.

In Phase 1, the user can specify the acceptable percentage of uninitialized flip-flops. The default is 10 percent. Also, Phase 2, which normally follows Phase 1, can be independently run if the user supplies functional vectors or initialization vectors. Experience has shown that Phase 2 can achieve a coverage of 65 to 85 percent. Phase 3 can also be independently run if the size of the given fault list is small.

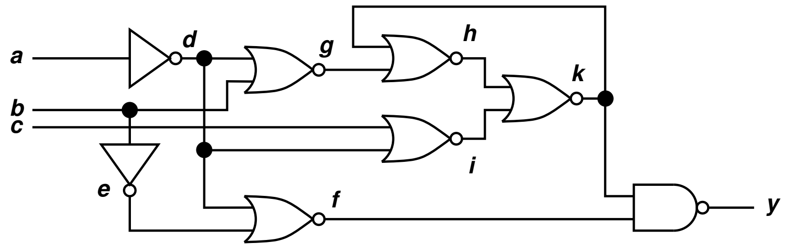


Figure 8.23: An asynchronous circuit for test generation by CONTEST.

Table 8.2: CONTEST results (CON: CONTEST, GEN: GENTEST.)

Circuit data				Fault coverage (%)		Fault eff. (%)	Number of test vectors		VAX 8650 CPU s	
Name	Gates	FFs	Faults	CON	GEN	GEN	CON	GEN	CON	GEN
MANNY	26	7	67	100.00	83.95	83.95	32	219	5	NA
SSE	207	6	454	83.26	83.20	99.50	561	676	291	1134
MULT4	382	15	540	97.04	92.78	93.15	364	148	838	1490
TLC	355	21	772	94.69	94.64	94.64	1256	5340	3312	32590
PLANET	690	6	1582	95.13	57.71	61.00	1439	132	3120	19388
MI	779	18	1629	94.53	NA	NA	1358	NA	1261	NA
CHIP-A	1112	39	1643	93.73	84.11	88.48	1031	384	98432	NA
CHIP-B	1539	73	2533	91.28	NA	NA	1034	NA	77904	NA

Example 8.8 *Asynchronous circuit. Consider the circuit of Figure 8.23. This is an asynchronous sequential circuit. Muth [481] used this example to illustrate the necessity of a nine-value path sensitization algorithm when a test for the fault "d stuck-at-1" is attempted. CONTEST produced four vectors: 000, 100, 101, 111. The last two vectors are the same as given in Muth's paper. Two extra vectors are generated because CONTEST starts with an arbitrary 000 vector and then brings the circuit to the appropriate state.*

Some results are shown in Table 8.2 [31, 153]. The circuit MANNY is asynchronous. All others are synchronous. SSE and PLANET are finite state machines with combinational logic implemented as a programmable logic array. MI is another finite state machine with random logic implementation. MULT4 is a four bit Booth multiplier circuit synthesized by an automatic synthesis system and TLC is a traffic light controller circuit. CHIP-A and CHIP-B are CMOS custom chips. CHIP-B contains one asynchronous flip-flop. For comparison, the results of a time-frame type of sequential circuit test generator (GENTEST [72]) are included in Table 8.2, CONTEST consistently produced better fault coverage and required less CPU time as compared to GENTEST. In some cases, due to circuit model incompatibilities, GENTEST aborted. The unavailable data are shown as "NA" in the table. GENTEST was ineffective for several circuits because of their complex sequential structure and asynchronous behavior. In other cases, GENTEST could identify redundant faults that were used to obtain fault efficiencies. A simulation-based technique like CONTEST cannot identify redundancies and hence does not provide the fault efficiency.

Consider the TLC Circuit to examine the run time performance of the directed-search method. Even though there are only 21 flip-flops, it is a highly sequential circuit with an internal counter and the ratio of logic to primary inputs is high (only 4 primary inputs.) For some faults, more than 200 vectors are needed to initialize the circuit to appropriate states to activate and propagate the fault effects to primary outputs. That means that more than 200 copies of the combinational portion are created in the iterative-array model. This is one of the reasons why GENTEST required significantly more CPU time than CONTEST for the TLC circuit.

As evident from Table 8.2, CONTEST frequently generated more vectors than the GENTEST program. This is a consequence of the one-bit change heuristic. In general, neighboring vectors in a CONTEST-generated sequence have fewer bit changes than in sequences generated by GENTEST. As a result, more vectors are needed to take the circuit to a desired state, starting from some given state. In some cases, this may be desirable because too many simultaneous input changes can produce hazards in the logic or produce power supply fluctuation due to current surge. The single-bit change strategy has also been used by other workers [587]. In general, the vector sequence length directly affects the test time and long sequences may have to be compacted to reduce the testing cost.

When the final fault coverage is lower than the desired goal, two options are possible. The first option is to start with a different (randomly selected) vector and attempt generation of tests for the undetected faults. The second option is to expand the one-bit change heuristic to include two-bit, three-bit, . . . changes. One should, however, expect a rapid increase in the amount of computations. Lioy *et al.* have taken a different trial vector approach [403]. They implemented the directed search approach in the MOZART concurrent fault simulator [236]. When a single vector with one-bit change does not provide cost reduction, their program examines the cost with multiple vectors. This strategy has the advantage of being able to get out of some local minima. They were able to obtain excellent results for some ISCAS '89 sequential benchmark circuits [99]. However, their conclusion was that the simulation-based technique was not very efficient for circuits with a complex feedback structure. Such structures are, in fact, known to be troublesome for other test generation algorithms also (see Subsection 8.2.6.) Among other possible strategies for trial vectors is the method based on genetic algorithms.

8.3.2 Genetic Algorithms

The process of test generation in CONTEST is *evolutionary*, in the sense that a test sequence is evolved by accepting and rejecting vectors according to their fault detection characteristics. Improved results are possible if trial vectors are generated by some "learning" process. For example, we can probabilistically favor the generation of the type of vectors that were more successful in the past. That is the basic idea of *genetic algorithms*, introduced by Holland [303]. An interested reader will also find the book by Goldberg to be useful [261]. A recent book by

Mazumder and Rudnick [445] discusses VLSI design and test applications of genetic algorithms.

Test generators based on genetic algorithms resemble CONTEST in several ways [445]. One uses the three-phase process. The cost function is replaced by a *fitness function*, which is now maximized instead of being minimized. Various types of fitness functions are computed via true-value or fault simulation. The basic difference, however, lies in the method of generating trial vectors. The procedure works with a set of vector sequences, called the population, which is improved iteratively. Each iteration is called a new generation. Vectors of a generation are produced from those of the previous generation, using operations known as *crossover*, *mutation*, and *selection*. In crossover bits from two vectors of the old generation are combined to construct two vectors for the new generation. In mutation, bits of a vector from the old generation are manipulated to create a vector for the new generation. In selection two individuals are selected, with selection biased toward more highly fit individuals. The fitness of the new generation is evaluated by simulation of the required characteristics such as initialization or fault detection. Creation of vectors in later generations is biased toward higher fitness.

One of the earliest simulation-based programs that used *genetic algorithms* (GA) was CRIS [556]. A simple fitness function, evaluated from true-value simulation, was used. It would favor those vector sequences that increased the signal activity in the circuit. This program had only limited success, perhaps because increased signal activity only improves controllability but does not necessarily increase observability. A later version of the program included fault simulation. Another program used *adaptive GA* [631]. Here, crossover and mutation probabilities were dynamically reduced for more fit individuals. The authors showed advantages of the adaptive scheme though their implementation was only for combinational circuits.

Compared to the early versions, significantly improved results were achieved by the GATEST program developed by Rudnick *et al.* [553]. In that program accurate fault simulation data was used for evaluating the fitness function. In a GA framework, new populations via crossover, mutation, and selection can be very quickly generated. However, fault simulation of large populations consumes enormous computing resources. GATEST simulates 100 to 300 randomly sampled faults to compute the fitness. These strategies worked well and the results compared favorably with the time-frame expansion program HITEC. GATEST used less CPU time than HITEC, produced shorter test sequences and sometimes (though not always) obtained higher fault coverages. More importantly, the results established the practicality of GA-based test generation and showed the necessity of fault simulation for fitness assessment.

Another recent program, GATTO [174], targets one single fault at a time. The circuit is assumed to be already initialized and a target fault is chosen from among those already active. The GA then generates vectors to propagate the fault effect toward POs.

Several strategies of the previous programs are combined in the STRATEGATE program developed by Hsiao *et al.* [310, 311]. This program uses GA in multiple

Table 8.3: Sequential ATPG by STRATEGATE [311].

Circuit name	Number of faults total	Number of faults detected	Fault coverage	Number of vectors	HP J200 (256MB) CPU time
s1423	1,515	1,414	93.3%	3,943	1.3 hours
s5378	4,603	3,639	79.1%	11,571	37.8 hours
s35932	39,094	35,100	89.8%	257	10.2 hours

phases to activate the fault in the combinational circuit (time-frame 0), justify the required state through previous time-frames, and propagate the fault effect through later time-frames. Table 8.3 shows a sample of results obtained for three benchmark circuits by STRATEGATE on a HP J200 computer with 256MB RAM. The test generator starts with the circuit in a completely unknown state. Only coverages are given since the simulation-based ATPG cannot identify redundant faults. Although CPU times are large, the fault coverages are significantly higher than those reported for time-frame expansion programs such as GENTEST [72, 160] and HITEC [497]. The reader may compare the result for the circuit s35932 in Table 8.3 with that obtained by GENTEST in Section 8.2.4.

Years of research on genetic algorithms for sequential ATPG has produced some highly improved programs. In the true sense of the word, this evolution will continue and more improvements may be forthcoming. Similarly, the quest for improved implementations of time-frame expansion algorithms also continues.

8.4 Summary

Sequential ATPG is practical for arbitrarily large circuits with adequate testability properties such as good initializability and cycle-free or limited-cycle structure. The University of Illinois program, HITEC [497], has been the basis for a commercial ATPG system, available for several years. GENTEST [72, 160] has been used within Lucent Technologies. At least two other sequential ATPG systems have been in use at IBM and NEC, respectively. IBM's program incorporates sophisticated branch and bound techniques originally developed at Rutgers University [150]. It has been used to generate tests for their partial scan microprocessor [151]. It is also commercially available to the users of IBM's TestBench system. NEC's program, SATURN [120], is based on various neural network and graph theoretic algorithms [127], and has been used within the company to generate tests for VLSI chip sets containing several million transistors. These programs employ the time-frame expansion technique.

In view of the large CPU times required for sequential ATPG, multi-processing has been explored. A popular technique is to distribute the fault list over a network of workstations that independently generate tests. This procedure is known as *fault-parallelism*. The inter-processor communication is minimized by only sharing the generated tests [63]. Significant speedups have been reported for the GEN-

TEST [603] and ESSENTIAL [369] programs. Interestingly, in cases where the detection of hard-to-detect faults is strongly influenced by the circuit state, even *superlinear speedup* is possible [18]. Superlinear speed up refers to the speed up of the program by a factor greater than the number of processors used. Sienicki [603] has analyzed the conditions for such speed up and has given adaptive techniques to obtain the best advantage of parallelization. Krauss *et al.* [370], using a distributed system of 100 workstations, first divide the fault list among workstations. When only the hard-to-detect faults are left over, they use several processors to cooperatively explore the vector space for tests targeting one fault at a time. This procedure is known as *search-space parallelism.* They observed speed ups between 42 and 92 for various circuits. They also reported that parallelization of test generation produced more vectors, which had to be compacted.

Simulation-based methods of test generation derive their efficiency from fault simulators such as a concurrent fault simulator (CFS.) Since the selection of a test vector depends upon the cost comparison, several trial vectors have to be simulated before a decision is made. A CFS implementation simulates the trial vectors in series. A more efficient implementation will be to use MDCCS (*multi-domain concurrent and comparative simulation*) [684] such that costs for many trial vectors are concurrently evaluated. The simulation-based method is applicable to all types of circuits, combinational or sequential. Its best advantage is in sequential, particularly asynchronous, circuits. In such circuits, timing of signals cannot be neglected and, therefore, the time-frame expansion methods run into difficulties (see Subsection 8.2.9.) With the simulation-based method, any circuit that can be simulated, can be tested. Among simulation-based techniques genetic algorithms have produced the best results.

Problems

8.1 *Race condition.* Suppose that all gates in the flip-flop circuit of Figure 8.2 have one unit of delay. Analyze all signal waveforms when a falling transition at D and a rising transition at CK occur, simultaneously. What timing condition should data (D) and clock (CK) signals satisfy for race-free operation?

8.2 Show that a test for any fault on the primary output of the serial adder circuit of Figure 8.3 can be obtained with at most two time-frames. *Hint:* Note that the primary output fault does not interfere with the initialization for the flip-flop which can be set in either 0 or 1 state by a single vector.

8.3 Show that any single stuck-at fault on primary inputs of the circuit in Figure 8.3 can be detected by two vectors when the initial state of the flip-flop is unknown.

8.4 Determine a test sequence for the s-a-0 fault on the output line of the flip-flop in the circuit of Figure 8.3.

8.5 Show that a test for the fault A s-a-0 in the circuit of Figure 8.24 cannot be obtained using the five-valued logic of the D-calculus. Obtain a test for this fault using the nine-valued logic.

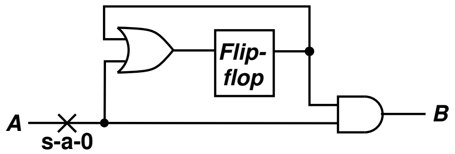


Figure 8.24: Circuit for Problem 8.5.

8.6 *Initialization fault.* Derive a test for the A s-a-1 fault in the circuit of Figure 8.25. Does the test provide a definite or a potential detection?

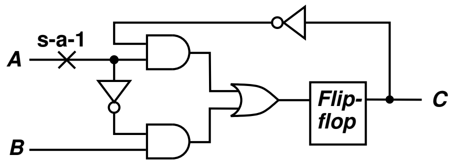
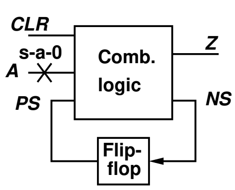


Figure 8.25: Circuit for Problems 8.6 and 8.7.

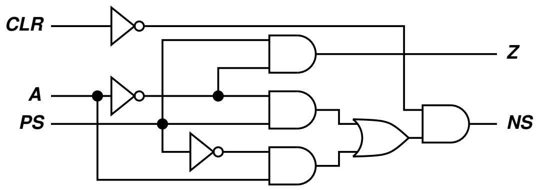
8.7 Devise a multiple observation test for the fault shown in Figure 8.25. Is a multiple observation test still possible if the inverter in the feedback path was shorted?

8.8 Compute drivabilities for all lines in the circuit of Figure 8.9 for the fault B s-a-0.

8.9 *Approximate test.* The single clock synchronous sequential circuit in Figure 8.26(a) has two inputs CLR and A. $CLR = 1$ initializes the flip-flop to 0. Using only the combinational part shown in Figure 8.26(b), derive a test vector (CLR, A, PS) to detect the A s-a-0 fault at the output Z. Find a justification sequence, assuming the combinational logic to be fault-free in previous time-frames. Verify whether this test sequence will work when the fault is present in all time-frames. If the test does not work, then derive an alternative test assuming the fault to be present in all time-frames.



(a) Sequential circuit.



(a) Combinational logic.

Figure 8.26: Circuit for Problem 8.9.

8.10 Prove that a fault that is untestable in the stand-alone combinational logic is also untestable in the sequential circuit.

8.11 Prove that a fault in the combinational logic of a synchronous sequential circuit is untestable if no combinational test vector can be justified using fault-free time-frames. *Hint:* See the paper by Agrawal and Chakradhar [30].

8.12 *Pseudo-combinational circuit.* Derive a combinational circuit by replacing all flip-flops by shorting wires in the circuit of Figure 8.9. This is known as the *pseudo-combinational* transformation, which can be applied to any cycle-free clocked sequential circuit [463]. Derive a test for the fault D s-a-0 in the pseudo-combinational circuit. Verify that the vector sequence obtained by repeatedly applying this vector four times will detect the D s-a-0 fault in the original sequential circuit. Note that the number of repetitions equals *sequential depth* $+1$.

8.13 Prove that if a combinational test vector can be obtained for a fault in the pseudo-combinational circuit, then that vector repeated as many times as *sequential depth* $+ 1$ will always detect the corresponding fault in the sequential circuit. *Hint:* See the paper by Min and Rogers [468]

8.14 Prove that a synchronous sequential circuit that is not initializable, must be cyclic.

8.15 *Cyclic circuits.* Redefine the s-graph by including PIs and POs as additional vertices. Levelize the graph starting from PI vertices using the minimum distance rule. Draw the new types of levelized s-graphs for circuits of Figures 8.9 and 8.13. What do the depths of these graphs represent in terms of the length of test sequences?

8.16 *Race fault in asynchronous circuit.* Derive a test for the s-a-1 fault at the output of the NOT gate in the circuit of Figure 8.27. Is this a race fault?

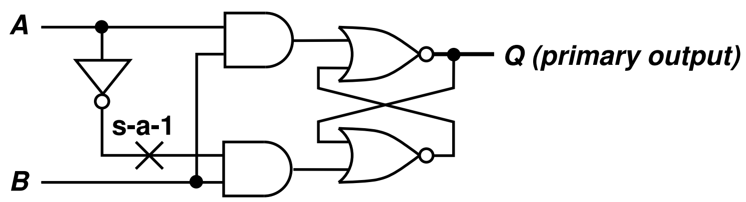


Figure 8.27: Circuit for Problem 8.16.

8.17 *Oscillation fault.* The asynchronous circuit of Figure 8.28 is designed to have no memory state. Derive a test for the s-a-1 fault on the C input of the NAND gate and show that it is an oscillation fault. Redesign the fault-free function as a combinational circuit.