Chapter 15

BUILT-IN SELF-TEST

> *"In a digital instrument designed for troubleshooting by signature analysis, this method can find the components responsible for well over 99% of all failures, even intermittent ones, without removing circuit boards from the instrument."* — Robert A. Frohwerk, in "Signature Analysis: A New Digital Field Service Method" [228].

Frohwerk ushered in a new era of determining the correctness of a circuit by examining a *signature*, which is some statistical property of the circuit. He applied the work of Peterson and Weldon [521] and Golomb [264] on error-correcting codes and shift register sequences to *built-in self-testing* (BIST) of ICs. Frohwerk also provided the first analysis for transition count testing and BIST to determine the probability of a signature indicating the circuit was good, when in fact it was faulty.

A digital system is tested and diagnosed during its lifetime on numerous occasions. Test and diagnosis must be quick and have very high fault coverage. One way to ensure this is to specify test as one of the system functions, so it becomes self-test. At the highest level of systems test, the testing function is frequently implemented in software. Many digital systems designed at AT&T circa 1987 had self-test, usually implemented in software [33]. Its most common use was in maintenance and repair diagnostics. Although this approach provided flexibility, it also had disadvantages. The fault coverage and the diagnostic resolution of those software-implemented tests were not as high as desired. The diagnostic resolution may be poor because the software must test parts that are difficult to test, and therefore it may not effectively determine which part is at fault. Also, software tests can be long, slow, and expensive to develop. Therefore, it becomes increasingly attractive to build the self-test function into the hardware [36, 37]. It is also most effective to consider testing as early in the design cycle as possible. Otherwise, costly prototyping turns (cycles of redesign and refabrication of the prototype) result, and these lead to schedule slippages for product introduction.

Systems designed without an integrated test strategy (covering all levels from the entire system to components) can best be described as *chip-wise and system-foolish*. With properly designed BIST, the cost of added test hardware will be more than balanced by the benefits in terms of reliability and the reduced maintenance cost [33].

Thus, these benefits, rather than the cost, can be passed on to the customers. The savings from BIST include reduced test generation effort at all levels, reduced test effort at chip through system levels, improved system-level maintenance and repair, and improved component repair. Gordon and Nadig [266] described the economic impact of signature analysis and BIST on the first systems that used BIST: two Hewlett-Packard digital voltmeters, one of them called the HP 3455A. Development time and costs rose roughly 1%. There was a 1% increase in parts cost due to added jumpers and extra ROM space required in the electronics for signature analysis, but total factory costs dropped, because of a 5% decrease in other materials costs. For example, it was no longer necessary to divide the product into small, replaceable modules. Also, BIST reduced service-module inventory at the factory, and decreased administrative and handling costs for failing units returned to the factory.

15.1 The Economic Case for BIST

These are some chip-level testability problems of the late 1990s [532]:

1. There is an extremely high and still increasing logic-to-pin ratio on the chip. This increasingly makes it harder to accurately observe signals on the device, which is essential for testing.

2. VLSI devices are increasingly dense and faster with sub-micron feature sizes.

3. There are increasingly long test-pattern generation and test application times.

4. Prohibitive amounts of test data must be stored in the *automatic test equipment* (ATE.)

5. There is increasing difficulty in performing at-speed (rated clock) testing using external ATE. For clock rates approaching 1 GHz, at-speed testing with an ATE is very expensive due to pin inductance and high tester pin costs.

6. Designers are unfamiliar with the gate-level structure of their designs, since logic is now automatically synthesized from the *VHDL* or *Verilog* hardware description languages. This compounds the problem of testability insertion.

7. There is a lack of skilled test engineers.

Complexity. One unfortunate property of large VLSI circuits is that testing cannot be easily partitioned. Consider two cascaded devices. There is frequently no simple way to obtain tests for the complete system from tests for the individual parts. In fact, even though each part is fully testable and has a test set that gives 100% stuck-fault coverage, the cascaded connection of the two parts will often have untestable and redundant hardware and much lower stuck-fault coverage. In other words, testing is a global problem. It is well known that there is no simple way to create tests for an entire *printed circuit board* (PCB) from tests for the chips on the board. For

design and test development effort, BIST provides a way to hierarchically decompose the electronic system-under-test, so this allows sub-assemblies to be first run through a BIST cycle, and if there are no faults, then boards in the system are run through a BIST cycle. Finally, if there are no board faults, then the entire system can be run through a BIST cycle. As an example, consider a system containing boards, which in turn contain chips. For a chip test, the system sends a control signal to the PCB, which then activates self-test on the desired chip, and sends the test result back to the system. BIST efficiently tests embedded components and interconnect, thus reducing the burden on system-level test, which now only needs to verify the synergy among the functional components [36]. When faults occur, the BIST hardware should be designed to indicate via an error signal or bus which sub-assembly is faulty. This greatly reduces repair costs.

Quality. Typical quality requirements are 98% single stuck-fault coverage or 100% interconnect fault coverage. The *reject ratio* is the percentage of faulty parts in the number of parts passing a test. The goal of testing at many companies is a low reject ratio, e.g., 1 in 10,000, at reasonable cost. The Motorola *six sigma* project has a goal of lowering this number to 1 in 100,000 [532]. In huge systems, this is attainable only through *design for testability* (DFT), and BIST is the preferred form of DFT.

Test Generation Problems. It is difficult to carry a test stimulus involving hundreds of chip inputs through many layers of circuitry to the chip-under-test, and then convey the test result back through the many circuit layers to an observable point. BIST localizes testing, which eliminates these problems.

Test Application Problems. In the past, *in-circuit testing* (ICT) [69] used a *bed-of-nails* fixture customized for the PCB-under-test. The bed-of-nails tester applied stimuli to the solder balls on the back of the PCB where the component leads were soldered to the PCB. Power was applied only to the component under test – all others in the PCB were left unpowered. It was effective for chip diagnosis and board wiring tests. However, ICT is not effective unless the PCB is removed from the system, so it is not helpful in system-level diagnosis. Also, *surface-mount technology* (SMT) components are often mounted densely on both sides of the board, and the PCB wire pitch is also too small for accurate probing of the back of the board by the bed-of-nails tester. Therefore, ICT is no longer a solution. BIST, however, solves these problems by eliminating expensive ATE, and BIST also lets us use the same tests and test circuits that are used at the system level [36, 37]. With BIST, there can be virtually unlimited circuit access via test points designed into the circuit through scan chains, resulting in an *electronic bed-of-nails* [33]. Another advantage of BIST is that the testing capability grows with the VLSI technology, whereas with external testing, the test capability always lags behind the VLSI technology capability. Logic gates and transistors are relatively cheap compared to the labor needed to develop

test programs, the cost of automatic test equipment, and the cost of real time for the tests to be run on production chips with ATE [33].

An additional benefit of BIST is lower test development cost, because BIST can be automatically added to a circuit with a CAD tool. Also, BIST generally provides a 90 to 95% fault coverage, and even 99% in exceptional cases [33]. The test engineer need no longer worry about backdriving problems of in-circuit test (where electrical stimuli provided to the middle of the circuitry damage outputs of logic gates), or how much memory is available in the ATE.

Table 15.1: Built-in self-testing costs.

Level	Design & test	Fabrication	Prod. test	Maintenance test	Diagnosis & repair	Service interruption
CHIPS	+/–	+	–			
BOARDS	+/–	+	–		–	
SYSTEMS	+/–	+	–	–	–	–

+ cost increase; – cost reduction; +/– cost increase ≈ cost saving (neutral or slight saving)

Table 15.1 [36, 37] shows the relative BIST costs at the chip, board, and system levels of packaging. BIST always requires added circuit hardware for a *test controller* to operate the testing process, *design for testability* hardware in the circuit to improve fault coverages during BIST, a hardware *pattern generator* to generate test-patterns algorithmically during testing, and some form of hardware *response compacter* to compact the circuit response during testing. We see an increase in fabrication costs at all three levels of circuit packaging. The BIST cost is frequently measured in terms of the added chip/board area required for the BIST hardware. The relative costs of added logic gates are declining, because hardware continues to become cheaper, but the relative costs of added long wires for test mode control are not really decreasing. This cost can also include added circuit delay, due to the extra device loads and delays from the test hardware. This may require a slight increase in the clock rate, and additional electrical adjustments to the design. Also, the test hardware can consume extra power, which is an additional cost. Since the BIST circuitry uses chip area, a final BIST cost is a decrease in the chip yield and chip reliability, due to the increased chip area [33]. BIST feasibility for a system must be evaluated using benefit-cost analysis, in the context of assessing total life cycle costs. Table 15.2 [33] gives more detailed metrics for evaluating BIST costs.

15.1.1 Chip/Board Area Cost vs. Tester Cost

At all three levels, BIST reduces testing costs. In order to understand why, consider the example of a 1 GHz microprocessor on a chip with 800 pins. For reliable stuck-fault and limited transition-delay fault testing, we should conduct the test at the rated clock speed. This forces us to use the Advantest Model T6682 1 GHz ATE, which can sample circuit outputs at this rate. The tester costs 800 $pins \times$

Table 15.2: Metrics for evaluating BIST.

Fault characteristics	Fault classes tested – Single stuck-at faults in functional circuitry Sequential faults in functional circuitry Delay faults Single stuck-at faults in BIST circuitry Fault Coverage – % of faults detected in functional circuitry % of faults detected in the BIST circuitry
Associated costs	Area overhead – Additional active area and interconnect Pin overhead – Additional pins. At least 1 pin is required to control whether BIST operates or not. One can design circuits that put the chip into BIST mode without using an extra pin, by applying a voltage level not normally used (e.g., 12 V instead of 0 or 5 V) to a pin. Additional input (output) pins for BIST are obtained by pin multiplexing, where input (output) pins are multiplexed into the BIST circuit when in BIST mode. The MUX adds a slight performance penalty. Performance overhead – added path delays due to BIST Yield loss – due to increased area or more chips in the system Reliability reduction – due to increased area Increased design effort and time Testability of the BIST hardware. The BIST hardware complexity increases when the BIST hardware is made testable.
Associated benefits	Reduced cost of testing and maintenance Lower test generation cost Reduced storage and maintenance of test patterns Simpler and less costly ATE Ability to test many units cost-effectively in parallel Shorter test application times Ability to test at system speed
Other characteristics	Degree to which BIST structure is function independent Diagnostic resolution Effect of engineering changes on BIST structure

$\$6,000\ per\ pin = \$\ 4,800,000$, but there is no chip area cost due to testing, because we do not use on-chip BIST hardware. Therefore, there is a huge initial capital cost for the ATE, but there is no recurring chip area cost on each chip for test hardware. If, instead, we provide BIST hardware, then the need for a very high-speed ATE is eliminated, except to test the wires from the circuit pins to the *Input MUX*, and from the circuit outputs to the output pins. The number of tests for that is very short, say perhaps 7 or 8 patterns and measurements per pin, and the cost of this can be safely ignored in this analysis. Therefore, with BIST doing all stuck-fault and transition-delay fault testing, we need a 1 GHz signal oscillator to clock the chip, and we need the ATE only to provide DC command signals to tell the microprocessor to perform BIST. Finally, we need an ATE to read out the success or failure DC signal for BIST from a circuit pin. In this case, we can use an inexpensive, 20 MHz ATE that costs roughly \$391 per pin, so our cost is $800\ pins \times \$391\ per\ pin = \$312,800$, a savings of \$4,487,200. This example is hardly far fetched. On-chip clock rates are expected to rise above 1 GHz, and at present, no ATE exists to test a circuit above 1 GHz.

For design and test development, BIST significantly reduces the costs of *automatic test-pattern generation* (ATPG), and reduces the likelihood of disastrous product introduction delays because a fully-designed system cannot be tested. Such a delay has occurred in the Intel Merced^{TM} project due to unexpected delays in inserting testability hardware into the chip, and fabrication line problems. There is a slight cost increase due to BIST in design and test development, because of the added time required to design and add pattern generators, response compacters, and testability hardware. However, our experience is that this is less costly than test development with ATPG.

15.1.2 Chip/Board Area Cost vs. System Downtime Cost

The true economic benefits of BIST show up in the last three columns of Table 15.1. Without BIST, maintenance test requires the presence of an expensive ATE at the site of the failing system, and this is a significant cost. With BIST, there is no need for an ATE, so this reduces system test cost. For boards and systems, BIST drastically reduces the diagnosis and repair cost, by quickly determining and indicating which sub-assembly or component is faulty, without the extensive labor and equipment normally required. This great reduction in diagnosis and repair time naturally leads to a major shortening in service interruption, particularly at the system level. An example of an application where service interruptions must be minimal is the # 5 ESS telephone exchange designed by Lucent Technologies. The exchange is designed to have, at most, a fraction of a second down time per year, because the lost revenue to an operating phone company when long distance calls cannot be made is quite severe. Another example is the credit card operations of American Express, which must be completed in a timely way so that the company knows which of its customers should be allowed additional credit and which should not. A final example is the Quotron system of the New York Stock Exchange for electronic trading of stocks. Down time of this system can cost billions of dollars per hour in lost trading opportunities. When viewed in this light, the added costs

of BIST hardware may be very minor compared with the severe cost of service interruption. A further benefit of BIST is its cost reduction for diagnosis and repair. Whenever any consumer home appliance breaks, the minimal cost just to have a repairman come out and look at the appliance is usually $100, and the total repair cost may exceed $175. It may be possible to justify BIST on a benefit-cost basis in personal computers and other home electronic appliances costing more than $1,500.

The reader may recall the rule of ten in testing costs from Chapter 3. If it costs $0.50 to detect a fault at the chip level, it will cost $5 to detect the same fault at the board level, $50 at the system test level, and $500 for a field repair for the same fault (because of the expense of sending out a serviceman to a customer site.) Therefore, test cost reductions due to BIST in Table 15.1 in the board and system columns are particularly important. It is most economical to detect problems early.

15.2 Random Logic BIST

15.2.1 Definitions

- BILBO – *Built-In Logic Block Observer.* This is a bank of circuit flip-flops with added testing hardware, which can be configured to make the flip-flops behave like a scan chain, a *linear feedback shift register* (LFSR) pattern generator, an LFSR-based response compacter, or merely as D flip-flops.
- Concurrent Testing – A testing process that detects faults during normal system operation.
- CUT – *Circuit-Under-Test*
- Exhaustive Testing – A BIST approach in which all 2^n possible patterns are applied to n circuit inputs.
- Irreducible Polynomial – A Boolean polynomial that cannot be factored.
- LFSR – *Linear Feedback Shift Register.* This is hardware that generates an exhaustive or pseudo-random pattern sequence of test patterns, and can also be used as a response compacter.
- Non-Concurrent Testing – A testing process that requires suspension of normal system operation to test for faults.
- Primitive Polynomial – A primitive Boolean polynomial $p(x)$ has the property that we can compute increasing powers of x *modulo* $p(x)$, and obtain all possible non-zero polynomials of degree less than $p(x)$. We compute the remainders of $x/p(x)$, $x^2/p(x)$, and so on. A primitive polynomial defines a mathematical number system that has the properties of a mathematical *field.*
- Pseudo-Exhaustive Testing – A BIST approach in which a circuit having n *primary inputs* (PIs) is broken into smaller, overlapping blocks, each with $< n$ inputs. Each of the smaller blocks is tested exhaustively.

- Pseudo-Random Testing – A BIST pattern generator that produces, via an algorithm, a subset of all possible tests that has most of the properties of randomly-generated tests. The random patterns must have a statistically high enough fault coverage to ensure a good test.
- TPG – Hardware *Test-Pattern Generator*

15.2.2 BIST Process

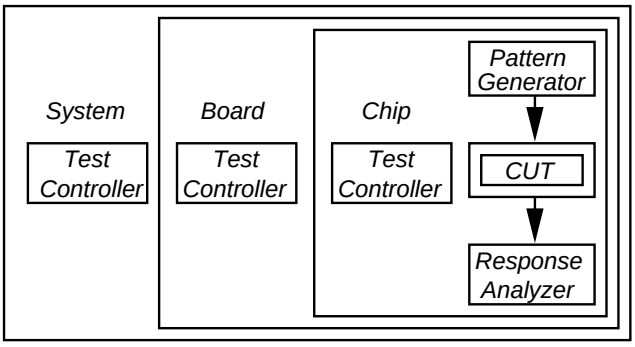


Figure 15.1: BIST hierarchy.

Figure 15.1 shows the BIST system hierarchy and all three levels of packaging mentioned earlier. The system has several PCBs, each of which, in turn, has multiple chips. The system *Test Controller* can activate self-test simultaneously on all PCBs. Each Test Controller on each PCB can activate self-test on all chips on the PCB. The Test Controller on a chip executes self-test for that chip, and then transmits the result to the PCB Test Controller, which accumulates test results from all chips on the board and sends the results to the system Test Controller. The system Test Controller uses all of these results to isolate faulty chips and boards [36, 37].

System diagnosis is effective only if the self-test procedures are thorough. For BIST, fault coverage is a major issue. Other issues are chip area overhead, its impact on chip yield, the cost of the additional chip pins required for test, the performance penalty in terms of added circuit delay, and extra power requirements. For BIST, the test engineer frequently, but not always, modifies the chip logic to make all latches and flip-flops controllable, perhaps by using the scan technique [36, 37].

BIST Implementations

Figure 15.2 shows typical BIST hardware in more detail. Note that the wires from PIs to the *Input MUX* and the wires from circuit outputs P to *primary outputs* (POs) cannot be tested by BIST. These wires, instead, require another testing method, such as an external ATE or JTAG Boundary Scan hardware. Figure 15.2 also shows how a comparator compares the signature produced by the data compacter with a reference signature stored in a ROM during BIST. This comparator and ROM hardware can frequently be implemented with a single logic gate with 32

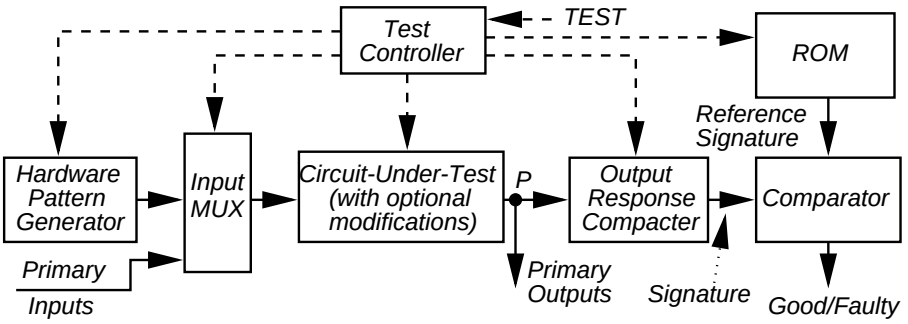


Figure 15.2: BIST process.

or fewer inputs. This is acceptable only when the comparison can occur at extremely low rates of circuit operation, since this logic gate is exceedingly slow.

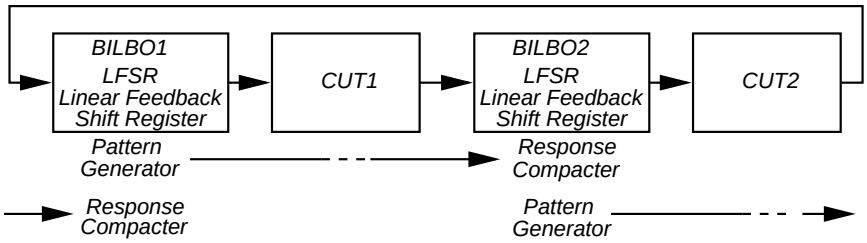


Figure 15.3: Multi-purpose registers in a BIST implementation.

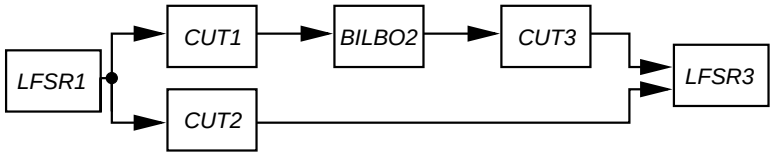


Figure 15.4: Complex BIST implementation.

Figure 15.3 shows a BIST implementation using *built-in logic block observers* (BILBOs) [364]. A BILBO is a bank of D flip-flops in the CUT that has test hardware added to make it behave in one of four modes:

- As ordinary D flip-flops.
- As a *linear feedback shift register* (LFSR) hardware pattern generator.
- As an LFSR configured to compact a circuit response.
- As a scan chain.

BILBO1 is configured as an LFSR pattern generator to test *CUT1* in the circuit, while *BILBO2* is configured as a response compacter to compact the responses of *CUT1*. During this process, the behavior of *CUT2* is ignored. *BILBO2* is configured as an LFSR pattern generator to test *CUT2* in the circuit, while *BILBO1* is

configured as a response compacter to compact the responses of *CUT2*. During this second process, the behavior of *CUT1* is ignored. For the normal system function, both *BILBO1* and *BILBO2* are configured to behave as simple D flip-flops.

Figure 15.4 shows a more complicated BIST system. Here, *LFSR1* is used to simultaneously generate patterns to test *CUT1* and *CUT2*. *BILBO2* is configured as a response compacter for *CUT1*, while *LFSR3* is configured as a response compacter for *CUT2*. In this mode, inputs to *CUT3* must be held steady so that the outputs of *CUT3* remain stable. In the second test mode, *BILBO2* is configured as a pattern generator for *CUT3*, while *LFSR3* is configured as a response compacter for *CUT3*. The outputs of *CUT1* are ignored, and LFSR1 must be set so that the outputs of *CUT2* are held steady during this second mode. Finally, Figure 15.5 shows a bus-oriented BIST implementation. The *self-test control* broadcasts test-patterns to each of the CUTs over their common bus. The self-test control then awaits bus transactions from each CUT that indicate the CUT's response to the stimulus pattern broadcast over the bus. This allows for a certain amount of concurrency during self-test. In this mechanism, pattern generation for all of the CUTs can happen in parallel, but response compaction is serialized over the bus.

The strategy for overall test control is the most difficult part of BIST design [33]. Care must be taken so that the BIST circuitry, as well as the circuit-under-test, can be tested for stuck-at faults. The BIST circuitry that must operate correctly for BIST to work is referred to as the *hard-core*. Either the hard-core must be minimized, or it must be made testable.

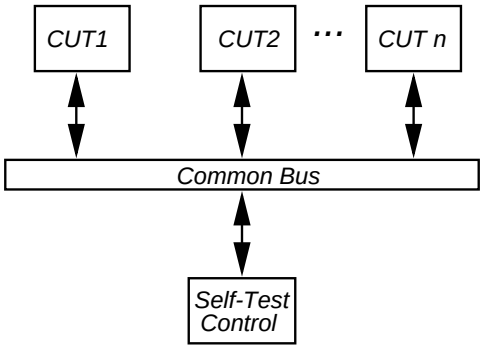


Figure 15.5: Bus-based BIST implementation.

15.2.3 BIST Pattern Generation

The following hardware pattern generation approaches have been used.

1. *ROM*. One method is to store a good test-pattern set (from an ATPG program) in a ROM on the chip, but this is prohibitively expensive in chip area, and will not be discussed further.

2. *LFSR*. Another method is to use a *linear feedback shift register* (LFSR) to generate pseudo-random tests. This frequently requires a sequence of 1 million

or more tests to obtain high fault coverages, but the method uses very little hardware and is currently the preferred BIST pattern generation method.

3. *Binary Counters.* A binary counter can generate an exhaustive test sequence, but this can use too much test time if the number of inputs is huge. For example, with 64 inputs and the test-pattern generator clocked at 100 MHz, this takes 51,240,955.8 hours of test time to generate all 2^{64} patterns, which is impractical. Therefore, this type of pattern generator must be partitioned. Also, the binary counter requires more hardware than the typical LFSR pattern generator.

4. *Modified Counters.* Modified counters have also been successful as test-pattern generators, but they also require long test sequences.

5. *LFSR and ROM.* One of the most effective approaches is to use an LFSR as the primary test mode, and then generate test-patterns with an ATPG program for the faults that are missed by the LFSR sequence. These few additional test-patterns can either be stored in a small ROM on the chip for a second test epoch, they can be embedded in the output of the LFSR, or they can be embedded in a scan chain in order to augment the stuck-fault coverage to 100%.

6. *Cellular Automaton.* In this approach, each pattern generator cell has a few logic gates, a flip-flop, and connections only to neighboring gates. The cell is replicated to produce the cellular automaton.

Exhaustive Pattern Generation

Exhaustive testing methods are intended to show that:

1. Every intended circuit state exists, and

2. Every state transition works.

For an n-input circuit, one must test the circuit with all 2^n input vectors, which causes exhaustive testing to be impractical when $n > 20$. Figure 15.6 shows an exhaustive pattern generator implemented with a binary counter.

Example 15.1 Exhaustive Test. *In the Intel 80386, exhaustive test was used for three control PLAs and for one control ROM using* multiple-input signature registers *(MISRs) for response compacters.*

Pseudo-Exhaustive Pattern Generation

We discuss four methods of pseudo-exhaustive testing:

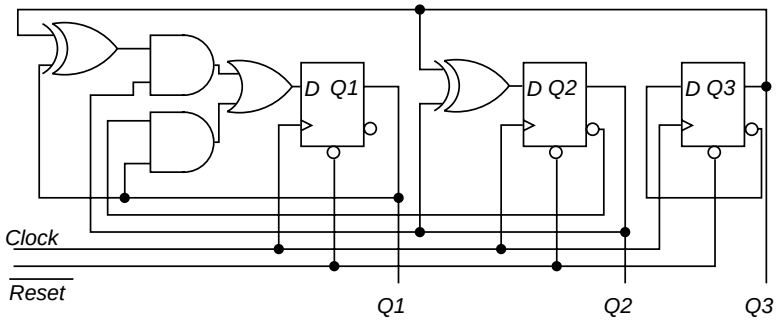


Figure 15.6: Exhaustive pattern generator.

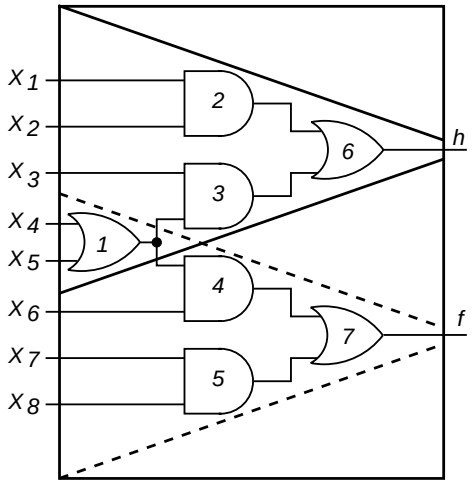


Figure 15.7: Backtracing for pseudo-exhaustive testing.

1. In Figure 15.7, exhaustive testing is made practical by *verification testing* (cone segmentation) [447]. We partition a large circuit into fanin cones by backtracing from each PO through the circuit to the input gates that influence that output. In fact, the fanin cones can often be tested in parallel to reduce test time. In this case PO h is influenced by inputs X_1, X_2, X_3, X_4, and X_5 while PO f is influenced by inputs X_4, X_5, X_6, X_7, and X_8. Therefore, we can design a pseudo-exhaustive test-pattern generator that in one mode stimulates all combinations of X_1, X_2, X_3, X_4, and X_5 while in a second mode it stimulates all combinations of X_4, X_5, X_6, X_7, and X_8. The benefit of pseudo-exhaustive testing is that the number of patterns is reduced from 256 to $2 \times 32 = 64$, which represents a testing cost reduction. However, observe that testing of the stuck-faults on the fanout branch from gate *1* to gate *4* will be missed in the first mode, but will instead be done in the second mode, so we will still obtain 100% stuck-fault coverage. Now, if we delete inputs X_4 and X_5 from the second testing epoch, in the interest of shortening the test pattern length by 24 patterns, then observe that this fanout branch fault will no longer be tested. This is because it still is not tested in the first test

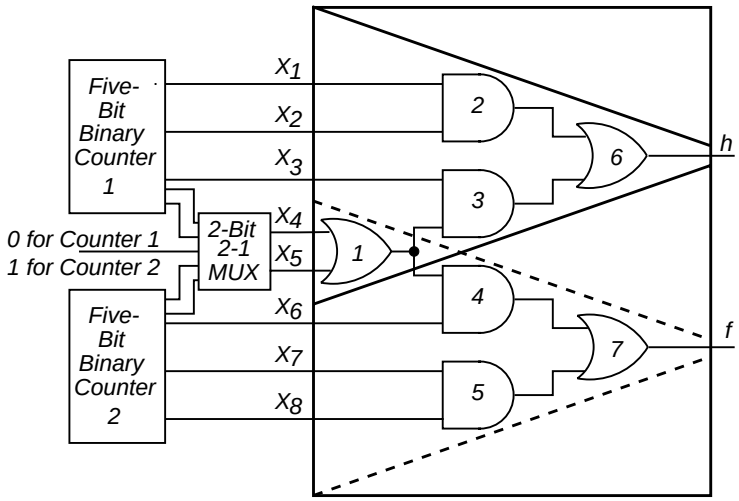


Figure 15.8: Pseudo-exhaustive pattern generator.

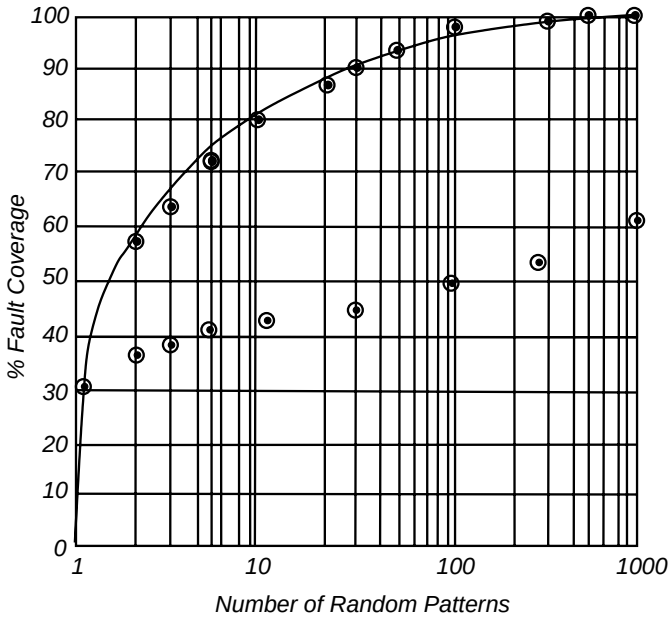
mode, and since the second test mode no longer controls inputs X_4 and X_5, it cannot be tested there, either. Finally, note that nearly any digital ATE can be configured to generate pseudo-exhaustive tests, apply them to a CUT, and observe the circuit responses. This is *ex situ* testing, in which the patterns are applied externally. BIST is a form of *in situ* testing, in which the patterns are applied internally by hardware in the circuit. However, ex situ testing lacks many of the economic benefits of BIST, such as a field-test capability and reduced tester cost.

2. A second method is *hardware partitioning* (physical segmentation) [448] in which we add extra circuit logic in order to divide the CUT into smaller subcircuits, each directly controllable and observable. Each of these is tested exhaustively.

3. A third method is *sensitized path segmentation* [136, 147, 448, 681], in which the circuit is partitioned so that sensitizing paths are set up from PIs to the partition inputs, and then from the partition outputs to the POs. Each partition is tested individually while the remaining partitions are simulated, so that non-controlling signals are set in the CUT to sensitize and propagate signals in the partition-under-test.

4. A final method is *partial hardware partitioning* [682], which combines *hardware partitioning* for observing signals in the partition-under-test, and *sensitized path segmentation* to control inputs to the partition-under-test.

Example 15.2 Pseudo-Exhaustive Pattern Generator. *Figure 15.8 shows a pseudo-exhaustive pattern generator using binary counters for the circuit of Figure 15.7. Pseudo-exhaustive testing is practical when each partitioned CUT element*



(a) Top curve -- random pattern testing with acceptable fault coverage.
(b) Bottom curve -- unacceptable random pattern testing.

Figure 15.9: Random-pattern testing and fault coverages.

can be made controllable from no more than k elements, where $k < n$, the number of PIs.

Testability vs. Random Pattern Count

Random pattern testing was investigated by Agrawal and Agrawal [15, 16, 20], Parker and McCluskey [512], and Eichelberger and Lindbloom [209]. Figure 15.9(a) shows how the stuck-fault coverage rises in a logarithmic fashion towards 100%, but at the cost of enormous numbers of random patterns. However, certain circuits, notably *programmable logic arrays* (PLAs), instead exhibit the behavior shown in Figure 15.9(b), and these are called *random-pattern resistant* circuits. Such circuits either require extensive insertion of testability hardware, or a modification of *random pattern generation* (RPG) with *weighted pseudo-random pattern generation* in order to obtain an acceptable fault coverage. At present, industry appears to require at least 98% stuck-fault coverage for reliability.

One can estimate the number of pseudo-random patterns needed from the desired fault coverage and either the set of hard-to-detect faults [67] or the circuit testability [591]. A million or more patterns is common for BIST, so fast fault simulation is essential. An appropriate technique for combinational circuits is the *parallel-pattern, single-fault propagation* (PPSFP) technique [210]. If the test sequence is too long for fault simulation, one can instead use deterministic ATPG to create test patterns for the hard-to-test faults, and embed these patterns in a ROM on the chip for a second episode of BIST. Alternatively, one can modify the circuit

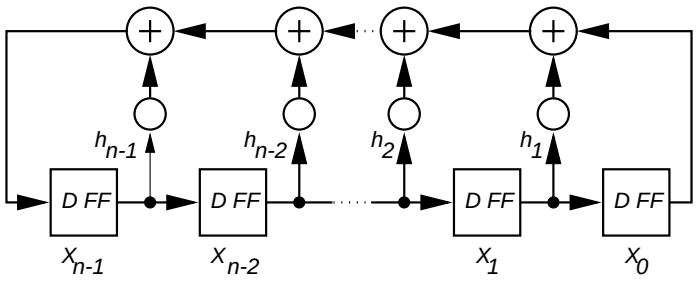


Figure 15.10: Standard linear feedback shift register.

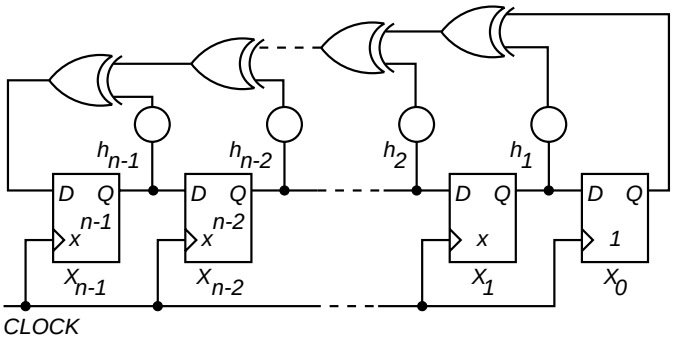


Figure 15.11: Actual digital circuit for standard LFSR.

to improve testability [210].

Pseudo-Random Pattern Generation

The *linear feedback shift register* (LFSR) pattern generator is most commonly used for pseudo-random pattern generation [67]. These patterns have all of the desirable properties of random numbers, but are algorithmically generated by the hardware pattern generator and are therefore repeatable, which is essential for BIST. We no longer cover all 2^n input combinations, but long test-pattern sequences may still be necessary to attain sufficient fault coverage. In general, pseudo-random pattern generation requires more patterns than deterministic ATPG, but fewer than exhaustive test [33].

Standard LFSR and Equations. Figure 15.10 shows a *standard, external exclusive-OR*, or *Type 1 linear feedback shift register* (LFSR) [206, 264]. It consists of D flip-flops and linear *exclusive-OR* (XOR) gates. It is an external exclusive-OR LFSR because the feedback network of XOR gates feeds externally from X_0 to X_{n-1}. The theory of these devices is described in Appendix A, which should be read before attempting this section. There are n flip-flops $(X_{n-1}, ..., X_0)$, so this is called an n-stage LFSR. A properly-designed LFSR can function as a near-exhaustive test-pattern generator, as it can cycle through $2^n - 1$ distinct states, the only omitted state being the all 0 state in the flip-flops. This is known as a *maximal length* LFSR.

Maximal-length sequences from such LFSRs have these additional properties:

- There is one pattern of n consecutive ones and one pattern of $n-1$ consecutive zeroes.
- There is an *autocorrelation property.* Any two sequences, the original and the circularly shifted sequence (in the same LFSR), will be identical in $2^{n-1}-1$ bit positions and will differ in 2^{n-1} positions [532].

In Figure 15.10, each tap coefficient h_i indicates the presence or absence of feedback from that particular flip-flop position into flip-flop position X_{n-1}. This is indicated by setting h_i $(0 \leq i \leq n-1)$ to 1 if the feedback exists, and to 0 if there is no feedback in that particular position. Figure 15.11 shows the actual hardware implementing this circuit. In the actual hardware, if an h_i is 0, then there is no XOR gate in the feedback network for that bit position; otherwise, the XOR gate is included. Remember that in this Galois field number system, multiplication by x is equivalent to a right shift in the LFSR register by one bit, and the addition operation is the XOR ($\oplus$) operator. Therefore, addition is equivalent to XOR subtraction, so $0-0=0$, $0-1=1$, $1-0=1$, and $1-1=0$. This is because there are no carries or borrows in XORing arithmetic. The following matrix system of equations describes the system:

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & h_1 & h_2 & \cdots & h_{n-2} & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} \tag{15.1}$$

This system is written as:

$$\boldsymbol{X(t+1) = T_S X(t)} \tag{15.2}$$

The first column of $\boldsymbol{T_S}$ is 0, except for the last row, to indicate that the flip-flops shift right. The 2nd through nth columns and 1st through $n-1$st rows are the identity matrix, to indicate that X_0 receives input from X_1, etc. Finally, the nth element in the first column is 1 to indicate that X_0 always feeds back into X_{n-1} through the XOR feedback network. The remaining elements in the nth row are the feedback coefficients h_i, which indicate whether the remaining flip-flops feed back into X_{n-1} or not. We also see why this LFSR cannot be initialized to all zeroes. If that were done, the feedback network and the right shifts of the flip-flops would always produce all zeros, and the LFSR would hang in the all-zero state. Note that the + operator implied in this matrix system is actually the XOR ($\oplus$) operator.

If $\boldsymbol{X}$ is the LFSR initial state, the LFSR will progress through the states: $\boldsymbol{X}$, $\boldsymbol{T_S X}$, $\boldsymbol{T_S^2 X}$, $\boldsymbol{T_S^3 X}$, etc. The *matrix period* is the smallest integer k such that:

$$\boldsymbol{T_S^k = I} \tag{15.3}$$

where $\boldsymbol{I}$ is the identity matrix, k is the LFSR cycle length (note that $k = 0$ for $\boldsymbol{X} = \boldsymbol{0}$), and $\boldsymbol{T_S}$ is known as the *companion matrix*. Recall that multiplication by x is equivalent to shifting a bit through the D flip-flop register of this LFSR. Therefore, we view X_0 as the constant 1, $X_1 = x \times X_0 = x$, $X_2 = x \times X_1 = x^2$, ... $X_n = x \times X_{n-1} = x^n$, etc. This hardware system can be described by the *characteristic polynomial*:

$$\begin{aligned} f(x) &= |\boldsymbol{T_S} - \boldsymbol{I}\ \boldsymbol{X}| \\ &= 1 + h_1 x + h_2 x^2 + \cdots + h_{n-1} x^{n-1} + x^n \end{aligned} \tag{15.4}$$

Thus, polynomial algebra can be used to predict the LFSR behavior.

Pattern Length and Detection Probability for LFSRs. Seth *et al.* [591] developed a method that can estimate both the fault coverage of a testing process, as well as the test length. The fault *detection probability* is the probability of detecting the fault by a random vector, and is represented by the distribution $p(x)$ of detectable faults:

$$\begin{aligned} p(x)\ dx &= \text{Fraction of detectable faults with probability} \\ &\quad \text{of detection between } x \text{ and } x + dx. \end{aligned} \tag{15.5}$$

$p(x)$ is non-zero and positive only when $0 \leq x \leq 1$. Also, $\int_0^1 p(x)\ dx = 1$. This assumes that all faults have non-zero detection probabilities. If redundant faults are present, then the formulation can be modified by adding a *Dirac delta function*, $\delta(x)$, to $p(x)$ such that the integral still evaluates to 1. The magnitude of the delta function will then be the fraction of redundant faults. For BIST pseudo-random vectors, there are $p(x)\ dx$ faults with detection probability x. The mean coverage among those faults by a pseudo-random vector is $x\ p(x)\ dx$. The mean fault coverage of the first pseudo-random vector is:

$$y_1 = \int_0^1 x\ p(x)\ dx$$

After removing the fault detected by the first vector, the detection probability distribution becomes $(1 - x)\ p(x)$. This leads to the mean fault coverage of n vectors:

$$y_n = 1 - \int_0^1 (1 - x)^n\ p(x)\ dx = 1 - I(n) \tag{15.6}$$

where $I(n)$ represents the above integral. If vectors are generated deterministically, and Y is the total number of faults, then the fault coverage after the first vector is generated is:

$$y_1 = \frac{1}{Y} + \left(1 - \frac{1}{Y}\right) \int_0^1 x\ p(x)\ dx$$

where the first term is the coverage due to the targeted fault, and the second term is the random coverage from the remaining faults. Proceeding recursively, one obtains an approximation for the coverage after n vectors are generated:

$$y_n \approx 1 - I(n) + \frac{n}{Y} \tag{15.7}$$

where $1 \ll n < Y$. This is valid for a large number of vectors, provided that the vector set is significantly smaller than the fault set.

Fault coverage and vector length can now be estimated by a fault simulation of a sampled set of faults, with fault dropping. The *random-first-detection* (RFD) variable of a fault is the vector number at which that fault is first detected. We obtain w_i as the number of faults whose RFD value is i. If N random vectors are generated, then the complete detection probability distribution is:

$$p(x) = \frac{1}{n_s} \sum_{i=1}^{N} w_i \; p_i \; (x)$$

where n_s is the number of faults in the sample that is simulated, and:

$$w_0 \triangleq n_s - \sum_{i=1}^{N} w_i.$$

This expression is valid when N is large. The integral can now be evaluated as:

$$I(n) = \frac{w_0(N+1)}{n_s(n+N+1)} + \frac{1}{n_s} \sum_{i=1}^{N} \frac{i(i+1)w_i}{(n+i)(n+i+1)}. \tag{15.8}$$

Accuracy improves as N increases, so one obtains the w_is from fault simulation using fault sampling, and then $I(n)$ is computed. The *fault coverage* is estimated using Equation 15.6 for random vectors and Equation 15.7 for deterministically-generated vectors. The *test length* to obtain a prespecified fault coverage can be predicted by inverting Equation 15.6 or 15.7, and solving it numerically. The functions $p(x)$ and $I(n)$ represent the circuit testability. This can be determined topologically (see Chapter 6) where it represents the testability by random vectors, but it can also be determined from the fault coverage data from fault sampling given a particular BIST vector set, using the above method. This may be more useful to the designer than a topological estimate.

Wagner *et al.* [702] provide a method of estimating the test length needed to achieve a given test confidence, which is the probability of detecting a fault with L pseudo-random vectors. However, that requires an exhaustive fault simulation, with no fault dropping, to determine the confidence accurately. Rajski and Tyszer [532] derive a test length for purely random testing, using assumed fault detection probabilities, which are generally not known. This would again require a lengthy fault simulation. Williams [723] and Savir and Bardell [567] also provide methods to estimate the test length.

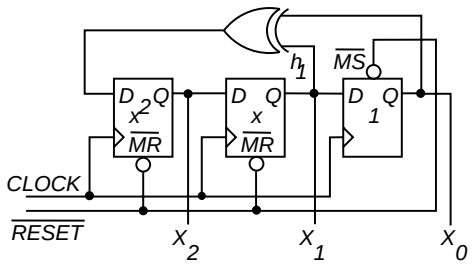


Figure 15.12: Example external-XOR standard LFSR.

Table 15.3: Pattern sequence for LFSR of Figure 15.12.

X_0	1	0	0	1	0	1	1	1	0	
X_1	0	0	1	0	1	1	1	0	0	...
X_2	0	1	0	1	1	1	0	0	1	

Example 15.3 Standard LFSR. *Figure 15.12 [732] shows an example external XOR standard LFSR with characteristic polynomial* $f(x) = 1 + x + x^3$. *Table 15.3 shows the pattern sequence generated by this LFSR when* $X_2X_1X_0$ *are set to "001." The characteristic polynomial* $f(x)$ *of the external-XOR LFSR is read from right to left, so since the rightmost flip-flop is always tapped, this polynomial has a* $1(x^0)$ *and since the middle flip-flop is tapped, it also has an* x *term, since* $h_1 = 1$. *However, there is no* x^2 *term, because the leftmost flip-flop is not tapped. There is always an* x^n *term in the characteristic polynomial for an* n*-bit standard LFSR, so we include the* x^3 *term. So,* $f(x) = 1+x+x^3$. *We see the pattern repeating after the first seven patterns are generated. The student should convince him/herself of the correctness of the pattern sequence by simulating the circuit logic. This LFSR implements the equation system:*

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \end{bmatrix} \tag{15.9}$$

During BIST, as will be explained below, it is essential that the circuit be excited once and only once with a particular pattern. This is because a given pattern causes an *error vector* to appear at the faulty circuit outputs, which are read by the BIST response compacter, and repeating the pattern later causes the same error vector to appear again. Since the response compacter is an XORing system as well, the two erroneous responses from that error vector will cancel and leave the BIST system with only the good-machine response. This will cause the testing hardware to accept a faulty circuit as a good circuit. Therefore, we must avoid repeating any of the LFSR patterns more than once, and we must not initialize the LFSR to all zeros, or it will hang indefinitely in the all zero state.

Modular LFSR and Equations. The *modular*, *internal exclusive-OR*, or *Type 2* LFSR is described by a companion matrix $\boldsymbol{T_M} = \boldsymbol{T_S}^\top$, which is the transpose of

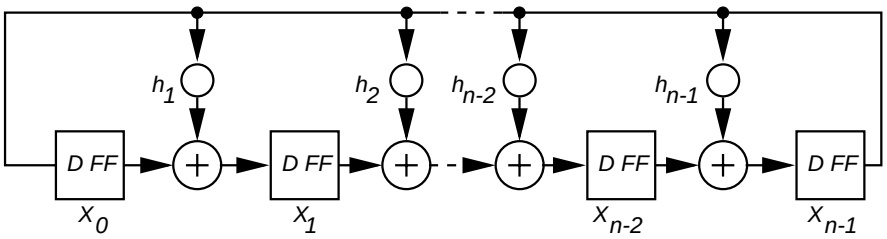


Figure 15.13: Modular LFSR example.

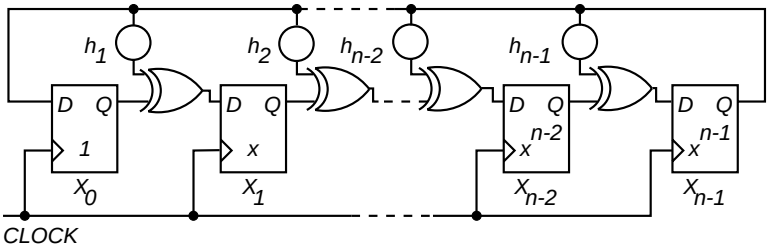


Figure 15.14: Digital circuit for modular LFSR example.

$\boldsymbol{T_S}$. It is called an internal XOR LFSR because the feedback XOR gates are located between adjacent flip-flops. The modular LFSR can run somewhat faster than the standard LFSR, because it has at most one XOR gate delay between adjacent flip-flops. However, this is usually not a serious consideration in testing, because actual circuits always have more logic gates between flip-flops than there are XOR gates in the feedback network of the external XOR LFSR.

Figure 15.13 shows the modular LFSR, and Figure 15.14 shows the actual LFSR circuit implementation. This hardware implements these equations:

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & h_1 \\ 0 & 1 & 0 & \cdots & 0 & 0 & h_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & h_{n-3} \\ 0 & 0 & 0 & \cdots & 1 & 0 & h_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & 1 & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} \tag{15.10}$$

This system is written as:

$$\boldsymbol{X(t+1) = T_M X(t)} \tag{15.11}$$

This hardware system can be described by the *characteristic polynomial*:

$$\begin{aligned} f(x) &= |\mathbf{T}_M - I\ X| \\ &= 1 + h_1 x + h_2 x^2 + \cdots + h_{n-1} x^{n-1} + x^n \end{aligned} \tag{15.12}$$

In this LFSR of Figure 15.13, a right shift is equivalent to multiplying the register contents by x, and then dividing its value by the characteristic polynomial and

storing the remainder. Every LFSR can be realized either in standard or modular form. Both use m XOR gates, where m is the number of non-zero h_i feedback coefficients in the LFSR.

Example 15.4 Modular LFSR. *Figure 15.15 shows an example modular LFSR and its characteristic polynomial, which implement this equation system:*

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ X_3(t+1) \\ X_4(t+1) \\ X_5(t+1) \\ X_6(t+1) \\ X_7(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ X_3(t) \\ X_4(t) \\ X_5(t) \\ X_6(t) \\ X_7(t) \end{bmatrix} \tag{15.13}$$

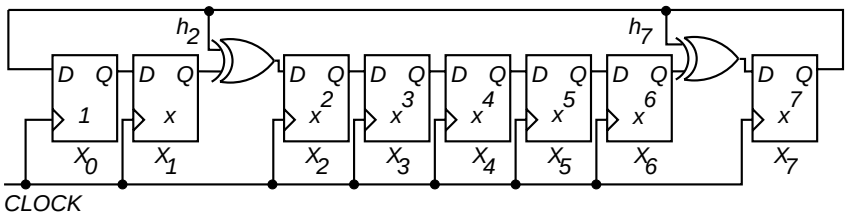


Figure 15.15: Modular LFSR for $f(x) = 1 + x^2 + x^7 + x^8$.

Primitive Polynomials. It is highly desirable that the LFSR generate all possible $2^n - 1$ patterns. Peterson and Weldon [521] have established the conditions necessary to satisfy this requirement, or to have a *primitive* polynomial for the LFSR:

1. The polynomial must be *monic*, which means that the coefficient of the highest-order x term of the characteristic polynomial must be 1. For the modular LFSR, this means that all D flip-flops must right shift through XOR gates from X_0 through X_1, ..., through X_{n-1}, which must then feed back directly into X_0. For the standard LFSR, this means that all D flip-flops must right shift directly from X_{n-1} through X_{n-2}, ..., through X_0, which must then feed back into X_{n-1} through the XORing feedback network.

2. The characteristic polynomial must divide the polynomial $1+x^k$ for $k = 2^n - 1$, but not for any smaller value of k.

Bardell *et al.* [67] have published tables of characteristic polynomials that are primitive, and therefore suitable for LFSRs (see Appendix B.)

To conclude, pseudo-random is the most frequently used method for BIST pattern generation [33]. However, it may require detailed fault simulation to evaluate the fault coverage.

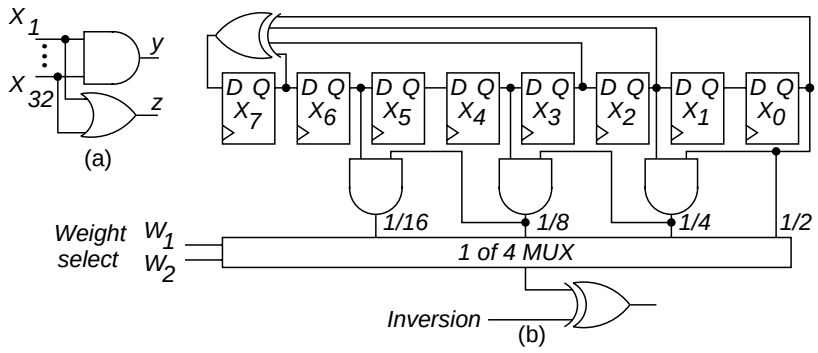


Figure 15.16: Weighted pseudo-random pattern generator.

Weighted Pseudo-Random Pattern Generation

The desire to achieve higher fault coverages with shorter test lengths, and therefore shorter test times, led to the invention of the weighted pseudo-random pattern generator. With this sort of pattern generator, one can adjust the probabilities of generating 0s or 1s at each input, in order to improve the fault coverage while shortening test length. Figure 15.16(b) [532, 706] shows a weighted pseudo-random pattern generator implemented with programmable probabilities of generating zeroes or ones at the PIs. This pattern generator effectively deals with hard-to-detect faults. In pseudo-random test, each input bit has probability 0.5 of being either a 0 or a 1. In weighted pseudo-random BIST, the probabilities are adjusted to make it more likely that tests will detect hard-to-detect faults. One method uses software to determine a single or multiple-weight set by probabilistically analyzing the hard-to-detect faults [210]. Another method uses a heuristic to generate an initial weight set, and relies on ATPG systems to produce additional weight sets. Weights are realized either by logic or are stored in a ROM. Over 98% fault coverage was obtained [210] for 10 designs, the same coverage as deterministically-generated vectors.

For the 32-bit arithmetic generate-propagate circuit of Figure 15.16(a), in order to detect the fault y stuck-at-0, all inputs must be 1. If uniformly-distributed pseudo-random patterns are used, the detection probability is 2^{-32}, because there are 32 inputs. This will lead to unacceptably long testing times. Pseudo-random patterns rarely achieve 100% fault coverage. If, instead, we could set each input to 1 with probability of 31/32, then the fault y stuck-at-0 could be detected with probability $(31/32)^{32} = 0.362$ [532]. This means that, on average, only three vectors would have to be generated to detect the fault. However, then each AND gate input stuck-at-1 fault would be detected with probability $(1/32)(31/32)^{31} = 0.01168$, so on average 86 vectors would be needed to detect such faults [532]. Figure 15.16(b) shows an adjustable weight pseudo-random pattern generator [532]. If each LFSR stage has probability of 0.5 of being either 0 or 1, and is statistically independent of the values on the other LFSR bits, then ANDing k LFSR signals results in a 1 value at the AND gate output with probability 0.5^k. OR gates and INVERTERS can be used to obtain other probabilities. In this example, the rightmost bit entering

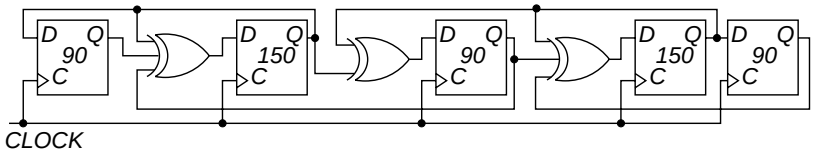


Figure 15.17: Five-stage hybrid cellular automaton pattern generator.

the MUX has a probability of 1/2 of being 1, and the second bit from the right has a probability of 1/4 of being 1, since it is the AND of two bits each with probability 1/2 of being 1. The third bit from the right has probability of 1/8 of being 1, since it is the AND of three bits each with probability 1/2 of being 1, and so on. If we program the *Weight select* lines to select the 1/16 probability of generating a 1, but we set the *Inversion* signal to 1, then we instead get a signal with 15/16 probability of being 1. This is because the XOR gate functions here as a programmable inverter. For the fault y stuck-at-0, each input should receive a 1 with probability of $1 - (31/32) = 1/32$ [532]. There is no common weight set for both faults, so two different weights must be stored for each circuit input. The main problem with weighted pseudo-random BIST is that several different weight sets are required in order to obtain 100% stuck-fault coverage, and for each set one must generate a number of random patterns [74, 347, 517, 706, 739, 740]. The multiple weight sets lead to excessive pattern generator hardware overhead.

Cellular Automaton Pattern Generation

Cellular automata (CA) are excellent for pattern generation, because they have a better randomness distribution than LFSRs [307]. A cellular automaton is a cell collection with regular connections [306, 353, 532, 691]. Each cell can only connect to its local neighbors. The connections are expressed as rules, which determine the next state based on the state of the cell's neighbors. If cell c can talk only with its neighbors, $c-1$ and $c+1$, then the following rule, called *rule 90*, can be established based on the following state transition table:

$x_{c-1}(t)x_c(t)x_{c+1}(t)$	111	110	101	100	011	010	001	000
$x_c(t+1)$	0	1	0	1	1	0	1	0

$$2^6 + 2^4 + 2^3 + 2^1 = 90$$

The term *rule 90* comes from the decimal equivalent of the binary code for the next state of cell c [738]. In this case, $x_c(t+1) = x_{c-1}(t) \oplus x_{c+1}(t)$. Another relation, *rule 150*, is implemented as $x_c(t+1) = x_{c-1}(t) \oplus x_c(t) \oplus x_{c+1}(t)$. Figure 15.17 [532] shows a hybrid cellular automaton alternately using rules 90 and 150 in its cells. Serra *et al.* [585] demonstrated an isomorphism between a one-dimensional linear hybrid cellular automaton and the LFSR having the same irreducible characteristic polynomial. However, state sequencing may still differ between the CA and the LFSR. CAs have no shift-induced bit value correlation, whereas LFSRs do. A LFSR pattern generator can be made more random, however, by using linear phase

shifters [182].

Test Pattern Augmentation

In certain applications, notably telephone exchange testing, 100% stuck-fault coverage is required. This, in turn, requires one to augment the set of patterns generated by an LFSR in a BIST testing method, in order to include test-patterns for the faults missed by the LFSR pattern sequence. We now describe the more frequently used methods for augmenting test-patterns.

Adding a Secondary ROM. The simplest way to increase stuck-fault coverage to 100% for a BIST pattern generator is to generate the patterns with an ATPG tool for the random-pattern resistant faults in the CUT that the BIST pattern generator misses, store the generated patterns in a ROM, and set up a second on-chip test pattern generation epoch. The Input MUX needs an extra control mode, and can connect the input pins, the BIST pattern generator, or the ROM output to the PIs. For ROM pattern generation, a simple hardware counter cycles the ROM through all of its addresses. The disadvantage of this method is the hardware overhead. With this method, it is extremely important to use a test-pattern compaction program to compact the test-patterns from the ATPG program into the minimum number needed to detect the random-pattern resistant faults.

Additional Methods. One method, *test pattern diffraction* [532] involves computing a good test pattern and then using a *hardware diffractor* to generate a cluster of vectors in the neighborhood of the pattern, which is stored in a ROM. Other approaches embed the deterministic patterns to detect the random-pattern resistant faults in the LFSR pattern sequence [676, 741]. Additional approaches [144, 675] transform the patterns produced by a pseudo-random pattern generator into a new test vector set providing the desired fault coverage. The STAR-BIST scheme embeds a small number of patterns to test random-pattern-resistant faults in the scan chain [678].

15.2.4 BIST Response Compaction

During BIST, it is necessary to reduce the enormous number of circuit responses to a manageable size that can be stored on the chip. For example, consider a circuit with a hardware pattern generator that computes 5 million test patterns during testing, and where there are 200 POs. The total number of responses will be $5,000,000 \times 200 = 1,000,000,000$ *bits*! This amount of information cannot be economically stored on the CUT, so the circuit responses must be compacted.

Definitions

- Aliasing – During circuit response compaction, because of the information loss, it is possible that a signature of a bad machine may match the good machine

signature, which is called aliasing. In such cases, a failing circuit will pass the testing process.

- Compaction – A method of drastically reducing the number of bits in the original circuit response during testing in which some information is lost.
- Compression – A method of reducing the number of bits in the original circuit response during testing in which no information is lost, so the original output sequence can be fully regenerated from the compressed sequence.
- Signature – A statistical property of a circuit, usually a number computed for a circuit from its responses during testing, with the property that faults in the circuit usually cause the signature to deviate from that of the good machine.
- Signature Analysis – A method of circuit response compaction during testing, whereby the entire good circuit response is compacted into a *good machine signature*. The actual circuit signature is generated during the testing process on the CUT, and then compared with the good machine signature to determine whether the CUT is faulty.
- Transition Count Response Compaction – A method of response compaction in which the number of transitions from 0 to 1 and 1 to 0 at circuit POs are counted to create a testing signature.

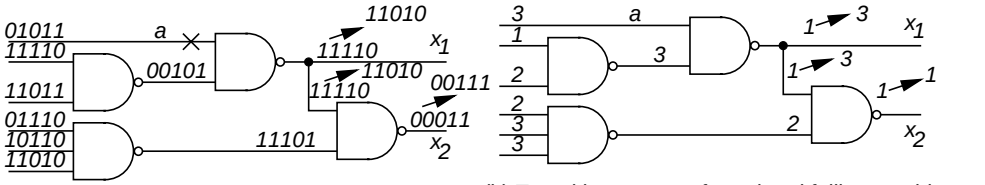
In this matter, we must distinguish between *compression* and *compaction*. Circuit response compression is lossless, because the original output sequence (10^9 bits in the above example) can be completely regenerated from the compressed sequence. Compaction, however, results in information loss, so regenerating the original circuit response information is not possible. Compression schemes, at present, are impractical for BIST response analysis, because they inadequately reduce the huge volume of data, so we use only compaction schemes. In mathematical words, compression functions are *invertible*, but compaction functions are not. *Signature analysis* is the process of compacting the circuit responses into a very small bit length number, representing a statistical circuit property, for economical on-chip comparison of the behavior of a possibly defective chip with a good machine chip. Frohwerk [228] invented *signature analysis* in 1977 at Hewlett-Packard. Also, the signature must preserve as much of the fault information contained in the circuit output response before compaction as possible, and the circuitry used to implement the compacter should be small [33]. All compaction techniques require that the fault-free circuit signature be known.

Some trivial schemes for response compaction are:

1. Parity checking, where we form parity across all circuit responses, and
2. Ones counting, where we count the number of ones in the output responses from the circuit. Savir [562] pioneered *syndrome testing*, in which pattern generation must be exhaustive, and ones counting is used for response compaction.

Aliasing occurs when the compacted response of the bad machine matches the compacted response of the good machine, and is always a problem with compaction because information is lost. In parity checking, aliasing frequently happens. Also, with ones counting, it is possible to permute the placement of ones in the circuit's Karnaugh map, and still obtain a correct ones count, so it is also very prone to aliasing and also requires significant arithmetic hardware.

Transition Count Response Compaction



(a) Logic simulation of good machine and fault a stuck-at-1.

(b) Transition counts of good and failing machines.

Figure 15.18: Example of transition counting response compaction.

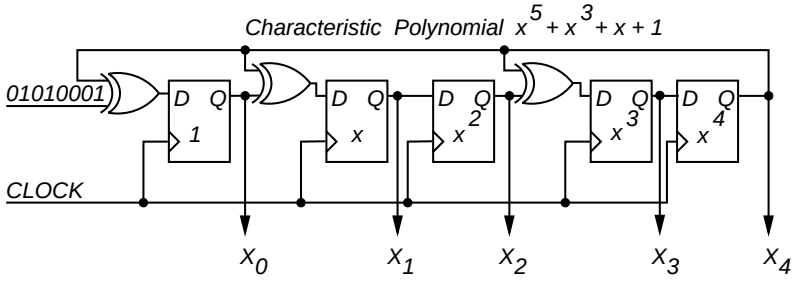
Hayes [285] described *transition count testing*. The *transition count* $C(R)$ is the number of times signals in the circuit response R change during BIST. Figure 15.18 shows an example circuit with the fault a stuck-at-1. In Figure 15.18(a), we see the circuit responses to five test-patterns, where the faulty machine response is shown above the good machine response. Figure 15.18(b) shows the sum of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions, with those of the faulty machine shown above those of the good machine. In Figure 15.18(b) at PO x_1, the good machine has a transition count of 1, but the faulty machine count is 3. One advantage of transition count compaction is that $|C(R)|$, the number of bits to represent $C(R)$, is $|C(R)| \leq \lceil log_2|R| \rceil$. $|R| = \#\ bits\ in\ R$. r_i is the circuit output response at time i. Then:

$$C(R) = \sum_{i=1}^{m} (r_i \oplus r_{i-1})\ for\ R = r_1 r_2 \ldots r_m \tag{15.14}$$

In order to maximize the test set fault coverage, in transition count testing we must make $C(R_0)$, the transition count of the good machine, as large or as small as possible. Transition count testing aliases less than ones counting, because it not only checks for the correct number of ones and zeroes in the circuit output response, but also partially tests for the correct *ordering* of the ones and zeroes in the response.

LFSR for Response Compaction

Frohwerk [228] introduced the LFSR for response compaction by signature analysis. The *signature* is any statistical property of the circuit that is used for checking its correct operation. He used the data compaction method of the *cyclic redundancy check* (CRC) code generator, which requires an LFSR hardware device (see Appendix A.) In this method, the circuit output data stream is treated as a descending order coefficient polynomial. The output response compacter LFSR performs



Inputs	X_0	X_1	X_2	X_3	X_4
Initial State	0	0	0	0	0
1	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
1	1	0	0	0	1
0	1	0	0	1	0
1	1	1	0	0	1
0	1	0	1	1	0

(Remainder $= 1 + x^2 + x^3$)

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 \cdot x^0 & +1 \cdot x^1 & +0 \cdot x^2 & +1 \cdot x^3 & +0 \cdot x^4 & +0 \cdot x^5 & +0 \cdot x^6 & +1 \cdot x^7 \end{array}$$

$x + x^3 + x^7 =$ Data Stream Polynomial

$$\begin{array}{r|l} & x^2 \quad +1 \\ x^5 + x^3 + x + 1 & \begin{array}{llllll} x^7 & & +x^3 & & +x & \\ x^7 & +x^5 & +x^3 & +x^2 & & \\ \hline & x^5 & & +x^2 & +x & \\ & x^5 & +x^3 & & +x & +1 \\ \hline & & x^3 & +x^2 & & +1 \end{array} \leftarrow \text{remainder} \end{array}$$

Figure 15.19: Modular LFSR as a response compacter.

polynomial division of this data stream polynomial by the characteristic polynomial of the LFSR. The final state of the modular LFSR is the polynomial remainder of this division. The final state of the standard LFSR is not always the polynomial remainder of this division, but is related to the true remainder through a different state assignment. The error detection hypothesis is that a faulty data stream changes the output data stream, and hence the remainder of this polynomial division, which is used as the signature in this compaction method. The LFSR must be initialized to the *seed* value, and after data compaction, the signature must be observed and compared with the known good-machine signature [33]. The signature analyzer circuit is easily testable.

Modular LFSR Response Compaction. Figure 15.19 shows a modular LFSR that has an extra XOR gate at the input to the flip-flop driving the least significant

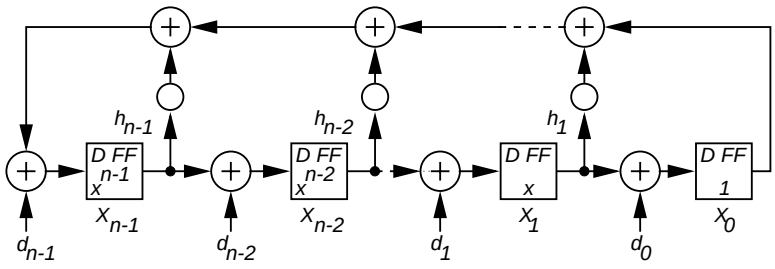


Figure 15.20: Example multiple input signature register.

bit X_0. This XOR gate XORs the circuit output response stream, "01010001" in this case, into the least significant bit of the modular LFSR. Before response compaction occurs, the LFSR flip-flops must be initialized to all zeroes. Here, "01010001" is interpreted as $0\times x^0+1\times x^1+0\times x^2+1\times x^3+0\times x^4+0\times x^5+0\times x^6+1\times x^7 = x+x^3+x^7$. Reading the LFSR tap coefficients from left to right in Figure 15.19, we see that the characteristic polynomial of this modular LFSR is $1+x+x^3+x^5$. The figure shows how eight clock periods are simulated after the LFSR is initialized to "00000." It also shows the long division of the reversed data stream polynomial by the reversed characteristic polynomial of the LFSR. The remainder of the division, $1+x^2+x^3$, also matches the remainder left after eight clock periods in the LFSR, because only X_0, X_2, and X_3 are ones. Thus, we have agreement between the signature predicted by polynomial division and the signature produced by logic simulation.

Multiple Input Signature Register. In the example of Figure 15.19 [186] we see that one primary circuit output required an LFSR for signature analysis with 5 flip-flops and 3 XOR gates. However, consider the case where the above circuit has 200 outputs. Then, we would need $200 \times 5 = 1000$ flip-flops and more than $200 \times 3 = 600$ XOR gates. This is a serious hardware overhead. Fortunately, we can exploit the fact that the hardware pattern generation and response compaction system using LFSRs is a *linear* system, obeying the equation $\boldsymbol{X(t+1) = T_S X(t)}$. Therefore, because of its linearity, this system also obeys the *superposition principle*. If we superimpose all of the responses of the 200 circuit outputs in the *same* LFSR for response compaction, then the final remainder will be the sum (under XOR logic arithmetic) of the remainders due to all of the circuit outputs. This is highly advantageous, as it reduces the flip-flop count from 1000 to 200 and the XOR gate count from more than 600 to approximately $3 + 200$. The 200 added XOR gates are needed to XOR all of the circuit outputs into different bits of the LFSR, where there must be one bit for each circuit PO, called d_i. This new response compacter is known as a *multiple-input signature register* (MISR), and an example is shown in Figure 15.20. The alternative to using the MISR structure is to provide only one simple LFSR for one circuit output, but multiplex it among the 200 different outputs. This then requires 200 different testing epochs, where for each epoch the LFSR compacts the response from a different circuit output. It is much more attractive to use the MISR, because it eliminates a 200 to 1 MUX, and also because

the response compaction time with the MISR is 200 times less than the time with a multiplexed LFSR. The MISR can be represented by a system of equations:

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ 1 & h_1 & \cdots & h_{n-2} & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} + \begin{bmatrix} d_0(t) \\ d_1(t) \\ \vdots \\ d_{n-3}(t) \\ d_{n-2}(t) \\ d_{n-1}(t) \end{bmatrix} \tag{15.15}$$

The vector of $d_i(t)$ values represents the circuit outputs at time t on PO i.

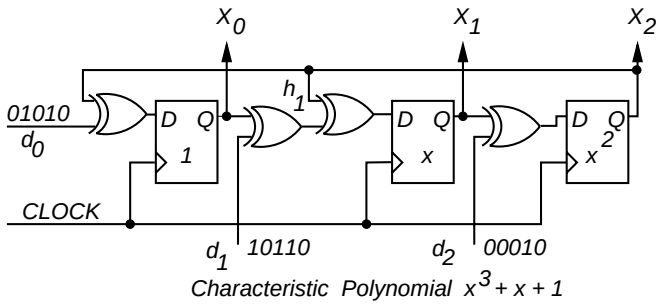


Figure 15.21: Multiple input signature register.

Example 15.5 MISR Example. *Figure 15.21 shows a modular LFSR converted into a MISR, by XORing a different circuit primary output into each flip-flop position. The resulting signature, since this system is linear, is the XORing of the three different signatures due to the polynomial division from each of the three POs. It implements the following equation system:*

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \end{bmatrix} + \begin{bmatrix} d_0(t) \\ d_1(t) \\ d_2(t) \end{bmatrix} \tag{15.16}$$

Multiple Signature Checking. Hassan and McCluskey [280] have proposed using multiple signatures to reduce the likelihood of aliasing. Two separate testing epochs are run, one with a MISR with one particular primitive polynomial, and the second using the same MISR but with a different primitive polynomial implemented in the feedback network. The hardware cost of this is low, usually a few XOR gates for a second MISR feedback network and a 2-to-1 MUX to select which feedback network will feed back into the External-XOR MISR. In some cases, it may be necessary to increase the bit width of the MISR in order to obtain a different primitive polynomial. This method is highly effective, since it is unlikely that the eigenvectors of the two different feedback networks would coincide, so aliasing is unlikely.

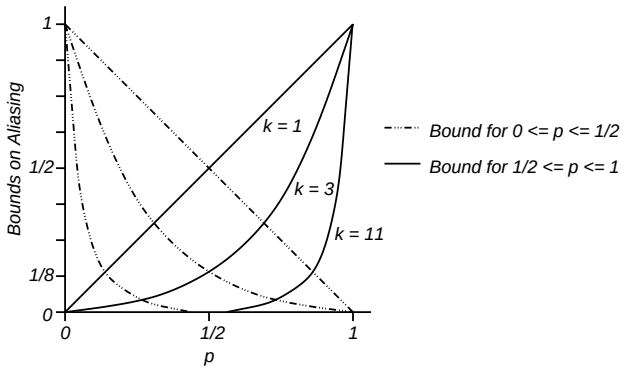


Figure 15.22: Aliasing error probability.

Aliasing Analysis

Williams *et al.* [726, 727, 728, 730] have extensively analyzed the aliasing probabilities of BIST LFSR response compacters. Earlier analyses also exist [114, 612, 630]. Williams *et al.* consider only the error sequence $e(n)$ at a circuit PO, which is obtained by *exclusive-XORing* (XORing) the output sequences of the good and faulty CUTs. A 1 in the error sequence indicates a manifestation of a fault. Let p be the probability of a 1 in $e(n)$. Let P_{al} be the probability of aliasing. If $0 < p \leq 1/2$ then $p^k \leq P_{al} \leq (1-p)^k$, but if $1/2 \leq p \leq 1$, then $(1-p)^k \leq P_{al} \leq p^k$ [729]. If all error sequences are equally likely, which is usually not true, then $P_{al} = 2^{-k}$. Figure 15.22 shows the weak aliasing bounds as a function of p, the probability of an error, and k, the number of bits in the response compacter. Williams *et al.* analyzed the dynamic properties of aliasing to confirm that primitive polynomials in response compacters alias less than non-primitive polynomials.

Theorem 15.1 *Assuming that each circuit PO d_{ij} has probability p of having an error, and that all outputs d_{ij} are independent, in a k-bit MISR, the aliasing probability is $\frac{1}{2^k}$, regardless of the initial condition of the MISR.*

The assumption of independent outputs is not true in the general case since circuit outputs are correlated, but the correlated outputs have little impact on this analysis. In the situation where each circuit PO d_{ij}, $j = 1, 2, ..., k$ had a unique probability of error p_j, the result still holds [730].

Theorem 15.2 *Assuming that each circuit PO d_{ij} has probability p_j of having an error, where the p_j probabilities are independent, and that all outputs d_{ij} are independent, in a k-bit MISR, the aliasing probability is $\frac{1}{2^k}$, regardless of the initial condition of the MISR.*

The aliasing probability is $1/2^k$, and therefore the error detection probability is:

$$1 - \frac{1}{2^k} \tag{15.17}$$

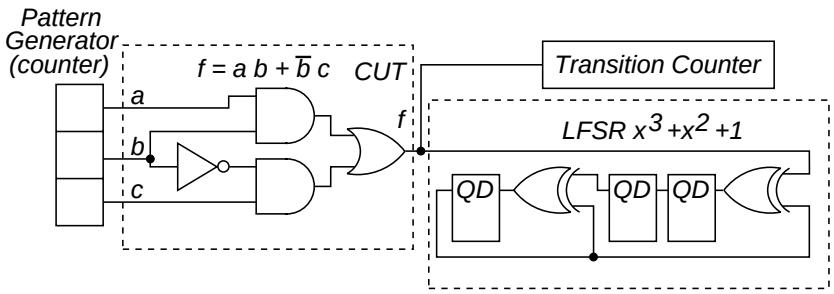


Figure 15.23: Example circuit with transition counter and LFSR.

These theorems say that it is just as good for testing purposes to just look at the effect of the last pattern, which is an unfortunate conclusion for testing.

Note that a MISR has an additional aliasing source, compared with a single-output signature analyzer. An error in CUT output d_j at time t_i followed by an error in output d_{j+h} at time t_{i+h} has no effect on the signature, if there is no feedback tap in the MISR between the outputs Q_j and Q_{j+h}. The probability of this aliasing can be reduced, by using a different primitive polynomial so that there is a feedback tap between outputs Q_j and Q_{j+h} [33]. Others have also analyzed aliasing probabilities [179, 180, 325, 349].

Example 15.6 *In the circuit of Figure 15.19 [186], we see that one primary circuit output required an LFSR for signature analysis with 5 flip-flops. The aliasing probability here would be* $1/2^5 = 3.125$ %. *If we add 3 more bits to the LFSR, the probability reduces to* $1/2^8 = 0.39$ %, *which is much more acceptable.*

Example 15.7 Comparison of Transition Count and LFSR Compaction. *Figure 15.23 shows a BIST system where both a transition counter and an LFSR with characteristic polynomial* $f(x) = x^3 + x^2 + 1$ *are used to compact the testing responses. The circuit function is* $f = a \cdot b + \overline{b} \cdot c$, *and the hardware pattern generator is an exhaustive three-bit binary counter. Table 15.4 shows the results of testing. The column labeled abc gives the test-pattern, and the other columns give the fault-free response and the responses for faults a stuck-at-one, f stuck-at-one, and b stuck-at-one. The table clearly shows the LFSR aliasing for the fault f stuck-at-one, and the transition count compacter aliasing for the fault a stuck-at-one.*

15.2.5 Built-In Logic Block Observers

As mentioned earlier, the BILBO combines the functionality of the D flip-flop, a testing hardware pattern generator (for the circuit portion driven by the BILBO Q outputs), a testing response compacter (for the circuit portion driving the BILBO D inputs), and a scan chain function. The scan chain BILBO can be reset to zero by shifting in an all-zero pattern into the BILBO in serial scan chain mode. Figure 15.24 shows a BILBO with characteristic polynomial $f(x) = 1 + x + \ldots + x^n$. Note the use

Table 15.4: Comparison of transition count and LFSR response compacters.

Pattern	Responses			
abc	Fault-free	a stuck-at-one	f stuck-at-1	b stuck-at-1
000	0	0	1	0
001	1	1	1	0
010	0	1	1	0
011	0	1	1	0
100	0	0	1	1
101	1	1	1	1
110	1	1	1	1
111	1	1	1	1
Signatures				
Transition count	3	3	0	1
LFSR	001	101	001	010

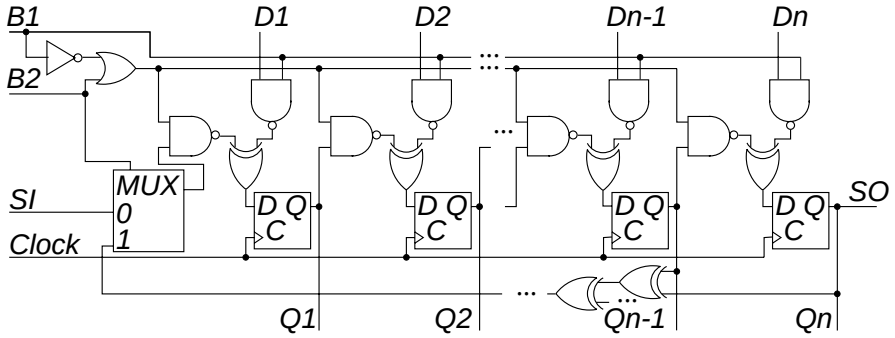


Figure 15.24: BILBO example.

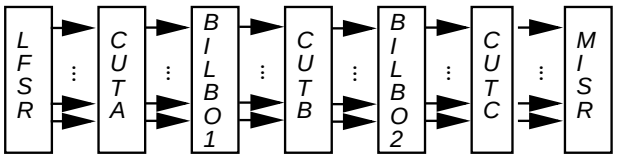
of NAND gates in this BILBO in order to accelerate its speed over implementations with AND and OR gates. Table 15.5 shows the control line modes for this BILBO.

Table 15.5: Control modes for the BILBO of Figure 15.24.

$B1$	$B2$	Mode	$B1$	$B2$	Mode
0	0	Serial scan chain	1	0	Normal D flip-flop
0	1	LFSR pattern generator	1	1	MISR response compacter

Figure 15.25(a) shows a testing configuration with three subcircuits to be tested, *CUTA*, *CUTB*, and *CUTC*; two BILBOs, *BILBO1* and *BILBO2*; an input *LFSR*; and an output *MISR* for testing hardware. Figure 15.25(b) shows how the testing hardware is used to test the three parts of the circuit.

Figure 15.26 shows the effective BILBO hardware in serial scan mode $(B1B2) = (00)$, Figure 15.27 shows the hardware in LFSR mode $(B1B2) = (01)$, Figure 15.28 shows the hardware in D flip-flop mode $(B1B2) = (10)$, and Figure 15.29 shows the hardware in MISR mode $(B1B2) = (11)$. Bold lines show the enabled data path.



(a) Example test configuration.

CUT	Mode BILBO1	Mode BILBO2
A, C	MISR	LFSR
B	LFSR	MISR

(b) Circuit modes during test sessions.

Figure 15.25: Circuit configured with BILBOs.

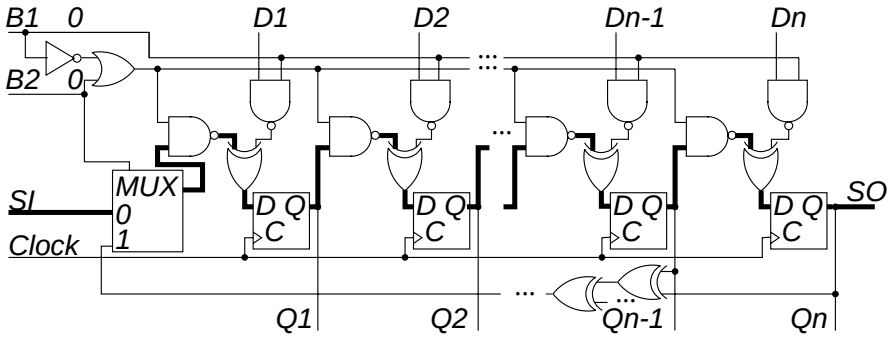


Figure 15.26: Example BILBO in serial scan mode.

15.2.6 Test-Per-Clock BIST Systems

In a *test-per-clock* BIST system, some new set of faults is tested during every clock period. The advantage of this BIST system is that it has the shortest possible pattern length, which may or may not be important. Consider a BIST pattern sequence of 10 million vectors, applied at the system operating speed of 200 MHz. Testing takes only $10,000,000/200 \times 10^6 = 0.05$ s. However, a major issue for BIST pattern length is fault simulation time. We can only know the actual fault coverage for a BISTed system from a stuck-fault simulation. The lengthy computation time worsens as the BIST pattern sequence increases, because the entire sequence must be simulated for the good machine and all failing machines. Figure 15.30(a) shows a test-per-clock system. In Figure 15.30(b) there are large numbers of PIs, so it is feasible to apply an LFSR to a subset of these inputs and then serially shift the most significant bit coming out of the LFSR into a shift register to provide pattern stimulation for the remaining inputs. The saving in test hardware is extremely minor. The shift register portion of the circuit has the same flip-flop hardware as the LFSR, but it lacks the few XOR gates used to form a LFSR feedback network.

15.2.7 Test-Per-Scan BIST Systems

In a *test-per-scan* BIST system, each new set of faults that is tested requires one clock to conduct the test and a series of shifts of the scan chain to complete that test and read out all of the test results. Test-per-scan, therefore, takes significantly more time than a test-per-clock method to detect the same number of faults in a given circuit. The advantage of test-per-scan systems over test-per-clock systems is that a judicious combination of scan chains and a MISR can lead to a significantly smaller

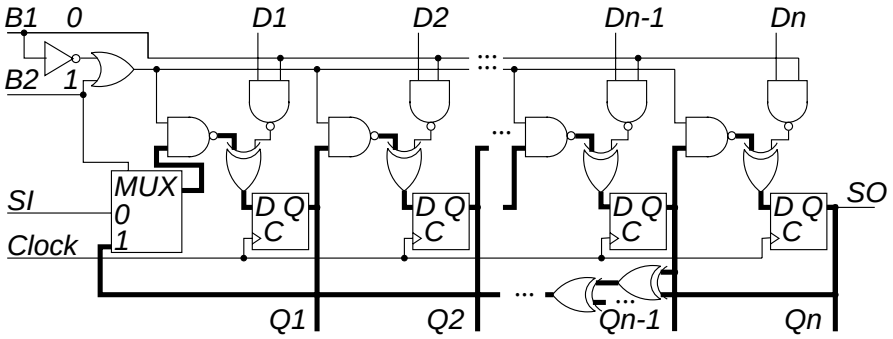


Figure 15.27: Example BILBO in LFSR mode.

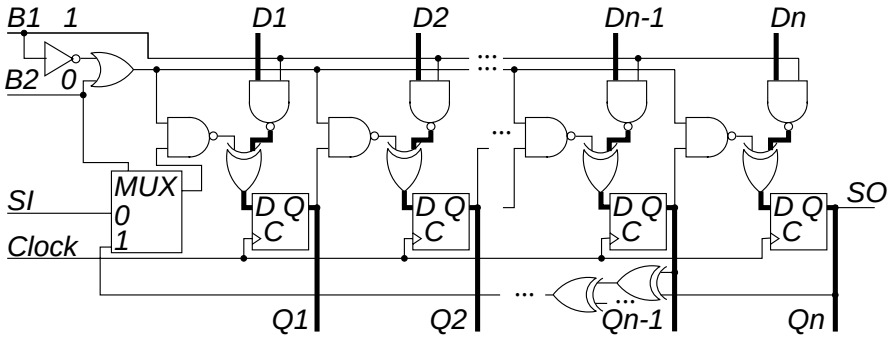


Figure 15.28: Example BILBO in normal D flip-flop mode.

MISR than in a test-per-clock system. However, this saving occurs at the expense of greatly increased BIST test-pattern length, which can cause a major computational bottleneck in fault simulation. Also, the advantages of hardware savings in the MISR are far less important now than they were a decade ago.

One problem in test-per-scan systems is that usually a new set of circuit input patterns is generated using a pseudo-random or exhaustive technique. However, because the input patterns are time shifted and repeated to the circuit through the scan chain, the patterns become correlated. This, in turn, reduces the pattern effectiveness for fault detection, so very frequently it is necessary to provide an input network of XOR gates to phase shift the inputs and de-correlate them.

Figure 15.31 shows a test-per-scan system called STUMPS* [66], in which the LFSR generates pseudo-random patterns, which are then fed through full-scan chains (*SR1*, *SR2*, ..., *SRn*) on various chips in the system-under-test. The scan chains drive the inputs of these chips. The chip outputs are collected in another scan chain, which is serially driven by all of the chip outputs. The chip output scan chains then drive a MISR. The advantage of this system is that if there are collectively 5,000 chip outputs, but they are sampled by 25 scan chains, each of length 200, then the output MISR needs to have only 25 bit positions in it, one for each scan chain, rather than 5,000. A scan chain for the chip outputs requires only 1

**Self-Test Using* a *MISR* and *Parallel Shift* register sequence generator.

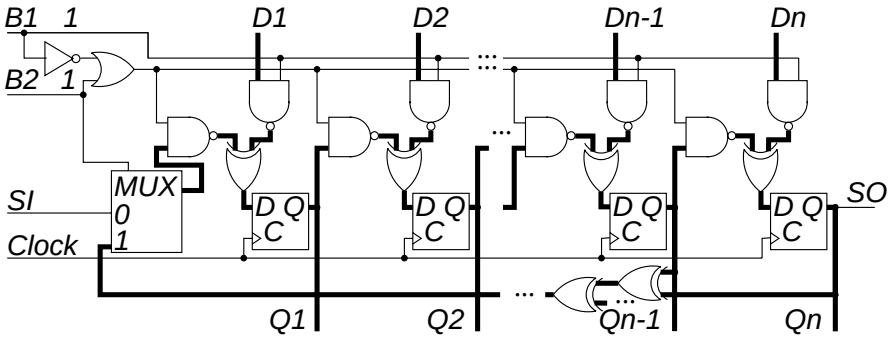
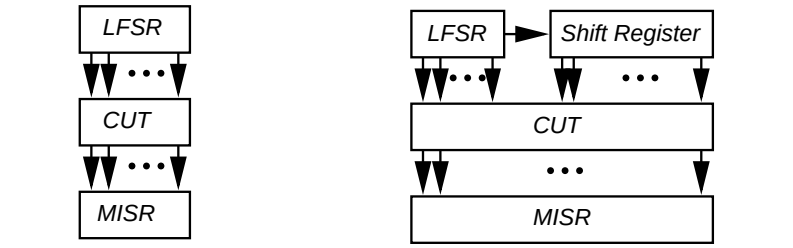


Figure 15.29: Example BILBO in MISR mode.



(a) Test-per-clock system. (b) Large input count test-per-clock system.

Figure 15.30: Test-per-clock scheme.

flip-flop and one MUX per output. However, a MISR for the chip outputs requires 1 flip-flop per output, 1 XOR gate per output, reset hardware for the flip-flops, and a number of XOR gates for the MISR feedback network. Again, because of the recent drastic decrease in hardware cost, the hardware savings of STUMPS over a MISR may be less important than they were a decade ago. A single test requires that we scan in patterns from the LFSR into all of the scan chains, and then we switch the system into normal functional mode and clock it once with the *system clock*. Finally, we scan out the contents of the scan chains into the MISR, where the scan chain contents are compacted. Scan out of the prior test results can be

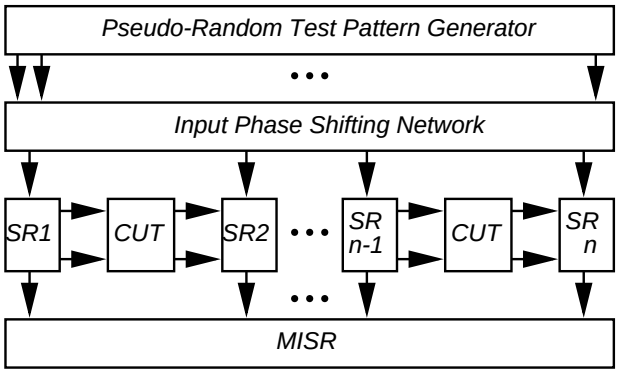


Figure 15.31: STUMPS test-per-scan testing system.

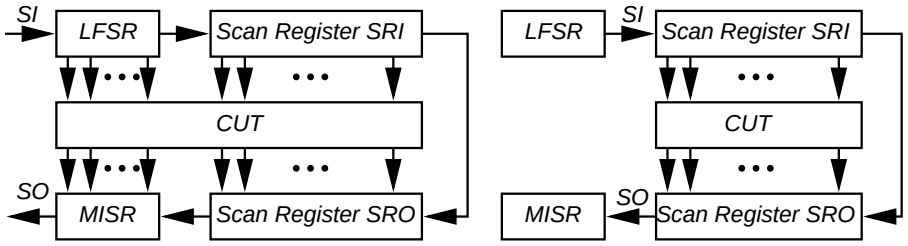


Figure 15.32: Test-per-scan systems.

overlapped with scan in of the next test pattern into the scan chains. This test method requires that every system input be driven by a scan chain, and requires that every system output either be caught in a scan chain or drive another chip in the system, where the other chip is also participating in the STUMPS testing system. Figure 15.32(a) shows an alternative test-per-scan system and Figure 15.32(b) shows a final alternative test-per-scan system.

Example 15.8 BILBO Versus STUMPS Versus External ATE at 325 MHz. *We now compare the performance of a BILBO system, a STUMPS system, and a conventional testing system using an external ATE. Here,* level-sensitive scan design *(LSSD) is assumed, where each flip-flop in the system is modified so that it cannot generate hazards at its outputs, and all clock lines are rigidly controlled during design so that there are no gated clocks and so that there are distinct clocks for normal mode operation and scan chain operation. We define the following variables:*

P = *number of patterns* $\quad L$ = *maximum scan chain length*
CP = *Clock period* $\quad k$ *ratio* = *self-test speed / LSSD tester speed*

We assume a system clock rate of 325 MHz, *and an LSSD tester speed also of* 325 MHz, *so* $k = 1.0$. *We assume that there are 2,000,000 BIST patterns and a maximum scan chain length of 100 bits. Table 15.6 shows the results of this gestalt experiment. Here, we see that the External ATE and STUMPS take the same amount of test time, but the BILBO system only takes 1/100 as much time, because it obtains a test-per-clock, rather than a test-per scan. This is a test cost reduction. Although fixturing setup times for chip testing are long, most production ATE has multiple chip fixtures and is capable of applying tests to one chip while several other chips are being loaded into or unloaded from other fixtures. Therefore, this reduced test time could possibly lead to an increase in chip testing throughput at the ATE, provided that there are enough different fixtures in the ATE to fully exploit this test time reduction. Also, fault simulation time will be 100 times longer for STUMPS and the External ATE system than for the BILBO system.*

Example 15.9 BILBO Versus STUMPS Versus External ATE at 1 GHz. *Now let us consider another example involving IBM's experimental 50 million transistor*

Table 15.6: BILBO, STUMPS, and external ATE comparisons.

	Testing method		
	BILBO (test-per-clock)	STUMPS (test-per-scan)	External ATE (test-per-scan)
Test time at 325 MHz	$P \times CP$ 0.00615 s	$P \times L \times CP$ 0.61538 s	$P \times L \times CP \times k$ 0.61538 s
Test time at 1 GHz	$P \times CP$ 0.002 s	$P \times L \times CP$ 0.2 s	$P \times L \times CP \times k$ 0.615384 s

chip, which operates a 1 GHz *clock rate. No existing ATE can operate at this speed, so we will assume that the ATE still operates at* 325 MHz. *Now,* $k = 3.07692$ *and* $CP = 10^{-9}$ s *for the BILBO and STUMPS methods, and* $CP \times k = 3.076923 \times 10^{-9}$ s *for the external ATE approach. We continue to assume that* $P = 2,000,000$ *and* $L = 100$. *Table 15.6 shows the results of this gestalt experiment. We see here that external testing with the ATE takes 307 times longer than with the BILBO, in addition to requiring a far more elaborate and expensive ATE than the BILBO system requires. We also see that STUMPS takes 100 times longer in testing time than the BILBO, because of the need to conduct a test-per-scan, rather than being able to conduct a test-per-clock.*

Example 15.10 *IBM RISC/6000 Pseudo-Random STUMPS Test. This testing system was used for embedded RAM test and transition-delay fault testing. The STUMPS system was hierarchical. Delay testing was performed using two clock phases, and operated at-speed, between scans of the scan chain.*

15.2.8 Circular Self-Test Path System

Figure 15.33 [367] shows the *circular self-test path* (CSTP) BIST configuration. In this testing system, the hardware pattern generator and response compacter are combined into a single hardware device, which is the entire circular flip-flop path. Therefore, this is a non-linear mathematical BIST system, so superposition no longer holds. Some of the flip-flops are converted into self-test cells (see Figure 15.33(a)), where in *TEST* mode, the cell XORs its D input with the state from the immediately prior flip-flop in the CSTP chain. After initialization of the registers, in the *TEST* mode, the circuit runs for a number of clock cycles and then the signature is read out of the circular register path. The entire path can be regarded as a MISR with characteristic polynomial $f(x) = x^n + 1$. However, the non-linear nature of this system makes it difficult to compute the fault coverage.

Example 15.11 Circular BIST. *At Lucent Technologies circular BIST has been used to test application-specific integrated circuits (ASICs) [37]. This example has four ASICs. On three, everything was tested using BIST except for the input/output buffers and the Input MUXes. For all four devices, the average logic gate count was*

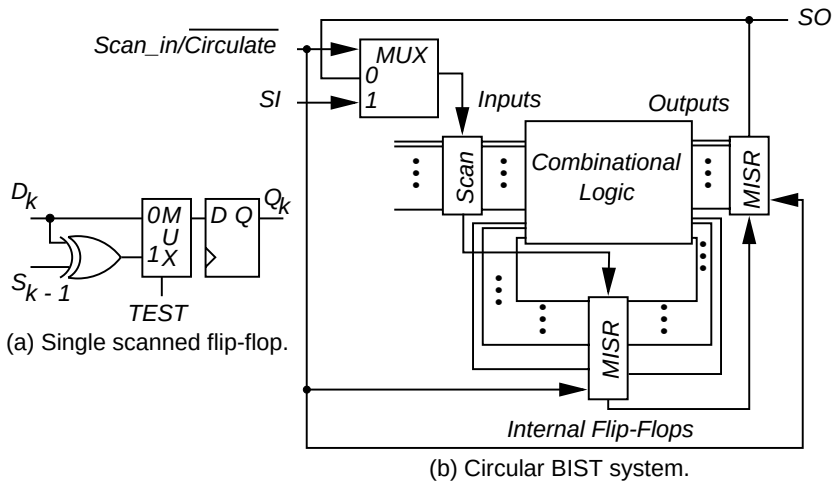


Figure 15.33: Full-circular BIST configuration.

14,150, the BIST logic overhead (per cent testing logic gates) was 20%, and the chip area overhead was 13%. The BIST stuck-fault coverage was 92%.

15.2.9 Circuit Initialization

It is very important in random logic BIST to initialize all flip-flops in the circuit when BIST is used with partial scan. Otherwise, X logic values in a 3-valued logic system will be clocked into the MISR. In the real hardware, different chips will randomly initialize their flip-flops to different values. Initialization problems can be discovered by setting all flip-flops initially to the X state, running the BIST cycle, and simulating the system in a 3-valued logic simulator. If the MISR or other response compacter finishes the test session with bits in the X state, then initialization is not correct. All such uninitializable flip-flops must then be initialized by adding master set or reset lines to them. Another approach is to break all cycles (loops of flip-flops) in the circuit, and then apply a partial BIST pattern sequence that synchronizes all flip-flops to a known state. Then, the response compacter can be turned on to compact the circuit's response. If the BISTed circuit uses a full-scan chain, then it is important to initialize the flip-flops in scan mode before initiating BIST. The initialization hardware significantly increases the chip area overhead of BIST.

15.2.10 Device Level BIST

It is necessary to isolate the BIST circuitry and the circuit-under-test from normal system data during testing. Inputs to the CUT can be isolated by multiplexers (known as the *Input MUX*) or by blocking gates. The Input MUX switches input to the CUT from the normal system inputs (port 0) to the BIST pattern generator (port 1.) When blocking gates, such as AND gates, are used, a constant '0' would

be applied to the second AND gate input, to block the normal system input arriving from the first AND gate input. Most importantly, note that neither the Input MUX nor the blocking gate will be thoroughly tested by the BIST hardware. The untested Input MUX or blocking gate hardware can be tested by additional, short external testing sequences, or by use of the Boundary Scan Standard (see Chapter 16.)

One useful technique during design simulation and verification of the BIST circuitry is to initiate BIST in an uninitialized circuit and apply unknown values to the CUT inputs during the simulation. If there is a logical "leak" of the system data into the BIST circuitry, this usually results in a signature full of unknown values [33]. If this is observed, then the isolation mechanism must be redesigned.

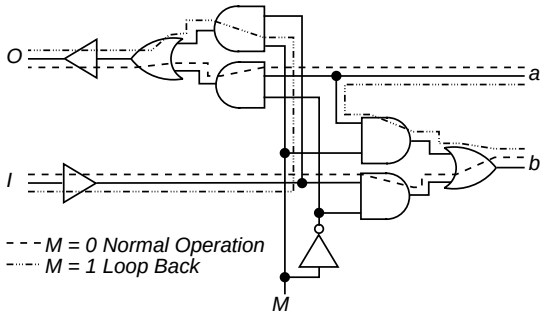


Figure 15.34: Circuit pack input/output loop back.

Another possibility for isolation is to loop back circuit outputs into circuit inputs [33]. A signal from outside the circuit indicates that the circuit should perform a self-test. The test controller sets a series of loop backs on the chip to allow data to circulate through the VLSI chip and not interfere with the rest of the system, as in Figure 15.34. The BIST procedure would exercise each VLSI chip using the loopback circuit. Then, the boundary scan chain (see Chapter 16) can be used to test the interconnections between the VLSI chips. Alternatively, the interconnections can also be tested by sending data from the test controller circuit through the subcircuits, looped I/O, and back to the test controller, as shown in Figure 15.35.

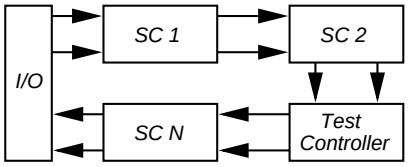


Figure 15.35: Functional test path.

It may be necessary to distribute BIST capabilities throughout the chip due to mixtures of general sequential logic, RAMs, ROMs, PLAs, etc., on a single chip. The BIST capabilities will differ significantly, and incorporating the results of the various BIST schemes into a single pass/fail result status is complex but highly desirable. Additional circuitry and test development effort is needed. One can instead provide the capability for routing the signatures of the individual BIST functions to the

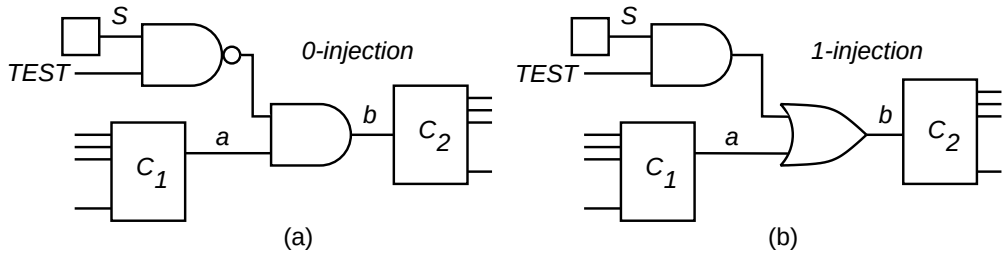


Figure 15.36: Control points to force 0 and 1.

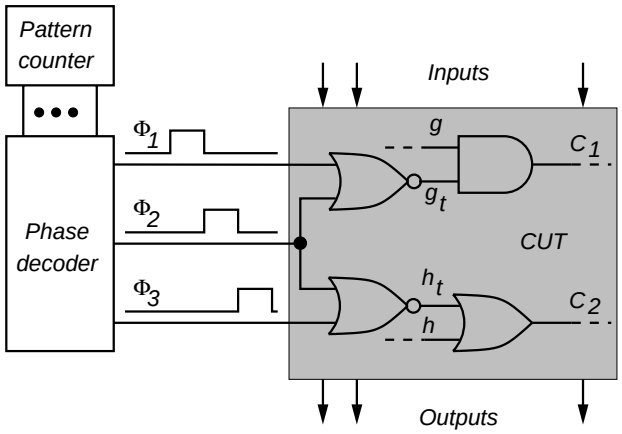


Figure 15.37: Test point activation.

device-under-test outputs [33]. If multiple VLSI devices on the same circuit pack are using BIST, then the hardware to incorporate the signatures from the various devices into a single BIST result can be included on one of the VLSI chips [33].

BIST can come close to, but not quite meet, the benefits of *scan design* in terms of fault coverage and area overhead [33]. One method uses an LFSR pseudo-random pattern generator, an LFSR for signature analysis, and LSSD. Fault coverage of BIST using this scheme can be greater than 98% [735]. Totten describes BIST approaches for general sequential logic [673].

15.2.11 Test Point Insertion

When random-logic BIST is used in a circuit, often not all of the faults are detected, either because of aliasing or because the test-pattern set from the hardware pattern generator is not rich enough to excite all faults. Touba and McCluskey [674] proposed modifying the circuit during logic synthesis to make it fully testable, using a two-level circuit representation. However, practical circuits are multi-level, and have already been optimized for function, timing, and power consumption. Re-synthesis would ruin the prior optimizations. Post-synthesis methods, instead, introduce observation points into the BISTed circuits to achieve 100% fault coverage. These methods use either exact fault simulation [101, 326, 676] or approximate testability measures [157, 561, 581]. Fault simulation finds signal correlations that

block fault propagation, but is extremely expensive in computation time. Rajski and Tyszer [532] show how to improve fault detection probabilities by inserting logic gates to improve signal controllabilities in Figure 15.36 [532]. The Part (a) circuit controls signal b to 0 when both $TEST$ and S are 1, while the Part (b) circuit controls b to 1 when both $TEST$ and S are 1. Observation points are inserted by tapping a signal with poor observability, and routing the wire from the signal to an extra flip-flop in the scan chain BILBO. Tamarapalli and Rajski [650] partition the entire random-logic BIST test set into multiple phases, each contributing to the goal of 100% stuck-fault coverage. Figure 15.37 shows how each phase activates a group of control and observation points [532]. They use probabilistic fault simulation to select the optimal set of control and observation points that gives the maximum improvement in fault coverage using the fewest test points.

Example 15.12 *Test point activation. In Figure 15.37, there are four test epochs, ϕ_0, ϕ_1, ϕ_2, and ϕ_3. A phase decoder enables different test points during each phase, during which a specific number of test-patterns are applied. Here, signal g_t is 0 in phases ϕ_1 and ϕ_2, forcing C_1 to be 0, independently of g. During phases ϕ_0 and ϕ_3, g_t is 1, so g controls C_1 during those phases. Similarly, h_t is 1 during phases ϕ_0 and ϕ_1, forcing C_2 to 1, independently of h. During ϕ_2 and ϕ_3, h_t is 0, allowing h to control C_2.*

The probabilistic fault simulator determines which signals receive control and observation points. It finds, for each circuit node, the fault list that propagated to it, along with the fault detection probabilities (called the *propagation profile*.) The detection probability is computed from analytical equations, and represents for all cases whether the fault effect is D or $\overline{D}$ at that node. The probabilistic fault simulation is set up to meet a user-specified threshold for the detection probability of a maximum number of faults. A greedy heuristic selects observation points first, and then an estimation technique computes two estimates, E_0 and E_1, of the number of faults that could potentially be detected by placing a 0-controllability or 1-controllability point at each circuit node. Nodes with E_0 or E_1 greater than a threshold are considered further, by injecting, for each candidate node, a 0 or 1 determining the improvement in the detection profile. Candidates are ranked by the number of additional faults that propagate to POs or observation points, as a result of control point insertion. Results of experiments show that near-complete fault coverage can be achieved by inserting very few test points and using a two-phase testing scheme. Even better results occur when a multi-phase testing scheme is used.

15.3 Memory BIST

We have entered an era of integration of various layouts or *cores*, from different companies, onto a single chip. For example, one custom VLSI chip may now contain an embedded RAM, a microprocessor, a DSP processor, and various analog circuit layouts. Embedded RAM memories are perhaps the hardest type of digital circuit to

test, because memory testing requires delivery of a huge number of pattern stimuli to the memory and the readout of an enormous amount of cell information. The difficulty and time required to propagate all of that information through the various glue logic and busses in an embedded core chip almost forces the use of memory BIST. The reader may refer to Chapter 9 for a complete discussion of memory march tests and the various memory fault models, as we will not repeat that information. With memory *design for testability* (DFT), the most time-consuming part of a memory test algorithm is implemented on-chip, and reduces the memory test time by an order of magnitude [688]. Kraus *et al.* [368] give a 1% area overhead for memory DFT for a 4 Mb DRAM. With memory BIST, the entire memory testing algorithm is implemented on-chip, and operates at the speed of the circuit, which is 2 to 3 orders of magnitude faster than a conventional memory test [688]. A 2% chip area overhead for memory BIST can be expected. Most memory BIST schemes exploit the parallelism within the memory device to achieve a massive reduction in test time (and therefore cost.) This is done by a test mode where more than one memory cell is accessed with each address, usually by accessing the entire row of cells on a word line for a single read or write operation. For n cells in the memory, with $\sqrt{n}$ rows and $\sqrt{n}$ columns, this reduces test time by a $\sqrt{n}$ factor. However, the parallel mechanism makes it difficult to test for memory coupling faults between cells in the same row, so it may not be appropriate. The regularity of the march tests makes them most suitable for memory BIST. We will not discuss random or pseudo-random memory BIST here, because the march tests achieve higher fault coverages with shorter pattern sequences than random or pseudo-random memory tests. However, sometimes the complexity of march test BIST implementation is too great, so longer memory BIST test algorithms are used, because they have less chip area overhead. ROM testing, as described in Chapter 9, can easily be extended to ROM BIST.

15.3.1 Definitions

- Concurrent BIST – A memory test mechanism where the memory can be tested concurrently with normal system operation.
- Non-Concurrent BIST – A memory test mechanism that requires interruption of the normal system function in order to perform the testing. The original memory contents are lost.
- Transparent Testing – A memory test mechanism that requires interruption of the normal system function for testing. The original memory contents are preserved in the memory after testing is finished.

Memory BIST requires an *address generator* or stepper (often an LFSR) and a *data generator*. As pointed out by van de Goor [688], an LFSR is better for march test BIST than a binary counter, because it uses substantially less area, and can easily be made self-testable [494]. Furthermore, the LFSR can be adjusted to provide the all-zero pattern and the forward and exact reverse LFSR

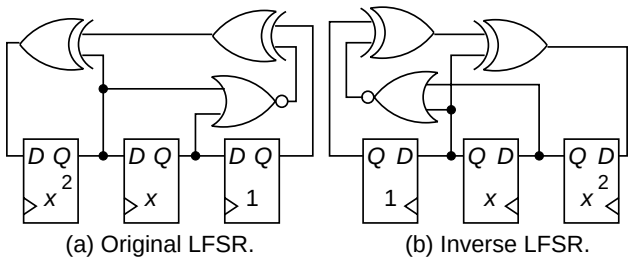


Figure 15.38: LFSRs that count up/down in inverse order.

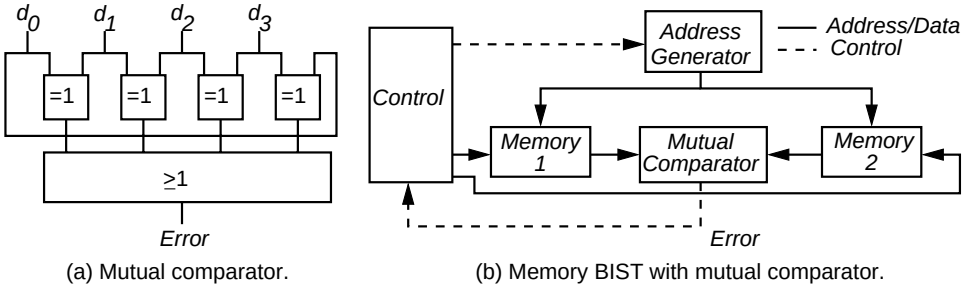


Figure 15.39: Mutual comparator for memory BIST.

sequences [67, 688]. Therefore, it satisfies all of the address ordering conditions for detecting address decoder faults with march tests. A reverse sequence LFSR generator has to have a characteristic polynomial that is the *reciprocal characteristic polynomial* of the LFSR, and it must shift in the opposite direction from the original LFSR. This is achieved by numbering the cells of the LFSR in the reverse order. Figure 15.38(a) shows an LFSR with a characteristic polynomial $G(x) = x^3 + x^2 + 1$, while Figure 15.38(b) shows the inverse LFSR with inverse characteristic polynomial $G(x) = x^3 + x + 1$, which also can generate the all-0 pattern because of the extra NOR gate and XOR gate. The Part (a) LFSR generates the sequence $1 \rightarrow 0 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 5 \rightarrow 2$ when initialized to 1, while the second generates the sequence $1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 0$ when initialized to 1. The NOR gate forces the LFSR into the all-zero state. These two LFSRs can be combined into a single LFSR, by adding a few additional logic gates. Another advantage of the LFSR over a counter is that the probability of an address bit changing is equal for all address bits. This enables detection of *write recovery faults* (see Chapter 9.) The test data can either be produced by a finite state machine or from the address. Response data evaluation is often carried out by deterministic comparison.

The *mutual comparator* [688] (see Figure 15.39(a)) is useful in memory BIST when the memory system has multiple arrays. We test two or more arrays (in this case 4) simultaneously, by applying the same test commands and addresses to all 4 arrays. The mutual comparator asserts the *Error* signal when one of d_0 through d_3 disagrees with the other data coming out of the memory arrays. The comparator eliminates the need to generate the good machine response, and implicitly assumes that only a minority of the memory array outputs are incorrect at any given time.

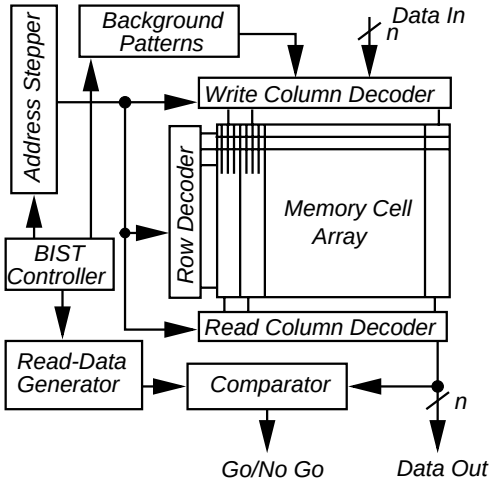


Figure 15.40: Parallel memory BIST.

The march tests are appropriate for SRAM testing. However, for DRAM testing, a *neighborhood pattern sensitive fault* (NPSF) testing model is more appropriate, since it provides better DRAM fault coverage [688]. Since the operation count is much longer for NPSF tests than for march tests, the benefit of BIST is greater when the NPSF tests are implemented on-chip. However, no NPSF test can detect address decoder faults, whereas all march tests can. Therefore, an appropriate scheme would be to put both test algorithms in the BIST hardware. We discuss march test BIST and NPSF BIST here.

15.3.2 March Test SRAM BIST

Nadeau-Dostie *et al.* [483] proposed a method to handle BIST of n-bit word RAMs. They provide a serial access design by adding multiplexers to the inputs of the write drivers. Each MUX selects between normal data input and a latch driven by the sense amplifier output of its left neighbor bit. This creates a shift register in memory BIST mode where a write causes each cell to copy data from its left neighbor, so only the leftmost bit in the memory word is directly controlled. A read causes only the rightmost bit in the word to be observed, while all other cells in the row are copied to the right.

Any march test can now be generalized to test an n-bit word memory row (called *line-mode* testing.) In the SMARCH test below, which is derived from MARCH C (see Tables 9.14 and 9.15), $(x)^n$ refers to operation x being repeated n times.

$$\{\Updownarrow (w0)^n (r0, w0)^n; \Uparrow (r0, w1)^n (r1, w1)^n; \Uparrow (r1, w0)^n (r0, w0)^n;$$
$$\Downarrow (r0, w1)^n (r1, w1)^n; \Downarrow (r1, w0)^n (r0, w0)^n; \Downarrow (r0, w0)^n (r0, w0)^n\}$$

Memories can be daisy-chained together, so that the serial output of one memory drives the serial input of the next, such that the memories appear to be one very

Table 15.7: Signals needed for march test BIST.

Signal	Type	Meaning
$WRITE/\overline{READ}$	OUTPUT	If 1, write *Data_In* to memory, else read memory contents into *Data_Out*
Count_Up	OUTPUT	If 1, count up in the *Address Stepper*; otherwise, count down
COUNT	OUTPUT	If 1, let the *Address Stepper* increment as specified by *Count_Up*, else hold the *Address Stepper* steady
First_Address	INPUT	If 1, the memory *Address Stepper* is currently at the first memory address
Last_Address	INPUT	If 1, the memory *Address Stepper* is currently at the last memory location
CLEAR	OUTPUT	If 1, clear the *Address Stepper* to the first address; otherwise, do nothing

large block. We present parallel memory BIST in Figure 15.40 [532]. The memory must be equipped with this test hardware:

1. A memory *BIST Controller*.
2. An *Address Stepper*.
3. A MUX circuit feeding the memory during self-test from the controller.
4. A *Comparator* for response checking.
5. A *Background Pattern* inserter or *Data Generator* for inserting test patterns into memory columns.

Example 15.13 MATS+ March Test RAM BIST. *We wish to design BIST hardware for the memory system of Figure 15.40 to implement the MATS+ march test* $\{ M0 : \Updownarrow (w0); M1 : \Uparrow (r0, w1); M2 : \Downarrow (r1, w0) \}$ *in hardware. Assume that, during test, the memory address input MUX is connected to the Address Stepper and not to the memory address line. Table 15.7 shows the necessary signals that the hardware of Figure 15.40 must provide. Note that the Up/Down LFSR Address Stepper of Figure 15.40 must also include an address MUX, to switch the memory address control lines from their normal external pin inputs to the outputs of the Address Stepper. Generally, the number of states required for a hardware implementation of a march test in a finite state machine is equal to* $2 \times$ *the number of march elements* $+ 3$. *The extra three states are a START state, where the controller sits until the TEST command is asserted to start a test; an ERROR state, where the controller sits when a memory error is detected (and the Address Stepper shows the location in error); and a CORRECT state, where the controller ends up if the memory passes the march*

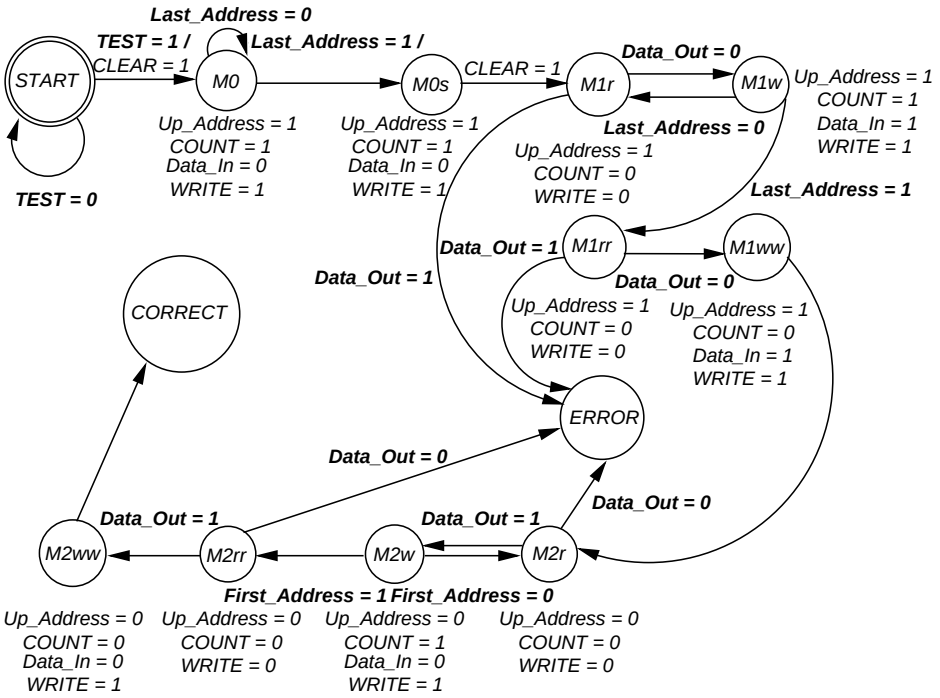


Figure 15.41: State transition diagram for the MATS+ memory BIST test.

test. Figure 15.41 shows the state transition diagram for the march test BIST controller. Because memories are huge, and the march test controller is very simple, the area overhead of memory BIST with march tests is one to two per cent, so memory BIST is widely used.

Example 15.14 March Test RAM BIST with Mutual Comparater. *Nicolaidis [494] partitioned a memory into two arrays, and tested it with the BIST system in Figure 15.39(b). The Address Generator is an Up/Down LFSR. Memories 1 and 2 each contain either half of the original number of rows (columns.) The control block steps through the states for the selected march test (see Figure 15.41.) The address generator can be tested with a parity generator [688], and the mutual comparator is tested with generated vectors (see van de Goor [688] Section 12.2.2), which are often embedded in the memory BIST RAM output sequence.*

15.3.3 SRAM BIST with MISR

A memory BIST scheme due to Jain and Stroud [337] accounts for the memory layout, and the memory address scrambling from the logical address space to the physical memory address space. The memory location is written with the value of its address or the bitwise complement of its address, so a data generator is not needed. Each location contains unique data. A series of up and down marches are performed using the Up/Down LFSR to detect stuck-at faults and transition faults. Another scheme uses a binary counter to provide addresses, input test data, and testing control signals.

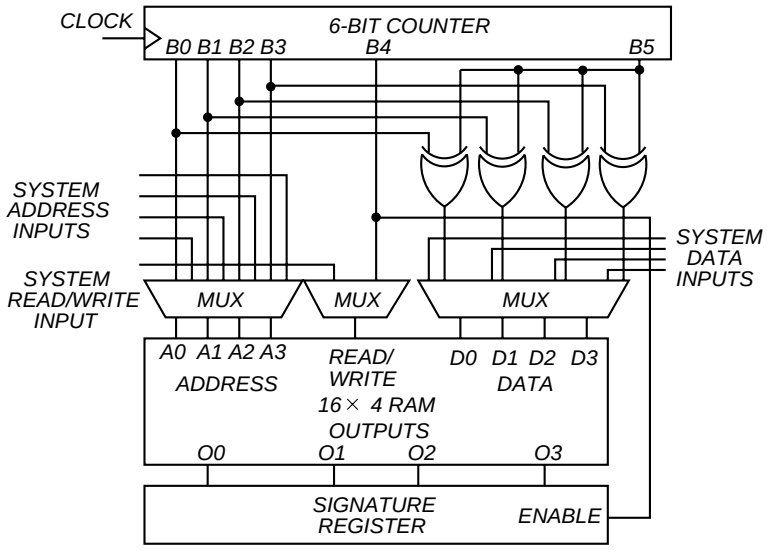


Figure 15.42: Counter test technique for RAM BIST.

Example 15.15 BIST of embedded RAM *[33]. Figure 15.42 shows the test hardware and RAM together. Here, rather than directly comparing the memory output to the expected output during testing, instead a MISR is used to compact the output from the memory. The MISR must be initialized before BIST, and must be disabled when shifting or loading new data into the RAM, since the output of the RAM is unknown during writes to the RAM. The same counter bit that enables writing also disables the LFSR in the MISR. Aliasing is controlled either by: (1) Providing a second set of feedback taps on the MISR, resulting in division by two different polynomials, or (2) Repeating the RAM test patterns in reverse order. This is done by using a counter bit to invert the RAM address during one of the multiple read sequences. This reduces aliasing, because the MISR is most apt to alias when output errors lie along a diagonal in the output pattern array. By reversing the RAM test patterns, the diagonal error pattern will lie along an orthogonal diagonal during testing, thus ensuring detection of the fault.*

Table 15.8 shows characteristics of six memory chips tested with memory BIST. They used a 6-bit binary pattern generation counter, in which the 4 LSBs were used for data and address generation during test, the 5th bit controlled reading or writing of the RAM, and the 6th bit inverted the RAM data during test. We extend march test notation to describe the actual test performed. In the following, (w *Address*) means write the address lines into the addressed location as data, and (r *Address*) means read the address lines from the addressed location without checking. However, all read-out data checking is done by a MISR, and not by the march test. The system implements the following march test: $\{\Uparrow$ (w *Address*)$;\Uparrow$ (r *Address*)$;\Uparrow$ (w $\overline{\textit{Address}}$)$;\Uparrow$ (r $\overline{\textit{Address}}$)$;\Downarrow$ (r $\overline{\textit{Address}}$)$;\Downarrow$ (w *Address*)$;\Uparrow$ (r *Address*)$;\Downarrow$ (r *Address*)$\}$. Note that this method is not proven to detect coupling faults and address decoder faults, but that march test BIST is

Table 15.8: Memory chips designed with BIST.

Aspect	Explanation
BIST application:	Embedded RAM, ROM, PLA
Test speed:	System operational speed
Benefits:	Vertical testing capability, less diagnostic run time, less run time, less test code development, less test complexity
Method:	Use extended counter as pattern generator, LFSR as response compacter, write memory address as data, read it back, then write & read back complemented address.
Example RAM size:	16 4-bit locations
Faults tested:	Stuck-faults and transition faults
Technology:	CMOS 2.5 μm
Aliasing probability:	0.000015 in a 16-bit LFSR

Device	Size	Type	# Flip-flops	Area overhead (%)		
				Logic area	Active area	Total area
A	8K	DRAM	427	9.8	5.5	3.4
B	8K	DRAM	171	18.0	7.6	5.0
C	16K	DRAM	284	20.0	5.5	3.8
D	4K	SRAM	210	11.0	4.2	2.9
E	–	–	–	11.8	6.8	4.2
F	4K	DRAM	239	6.8	4.6	3.2

able to do that, because march tests are proven to cover both types of faults.

15.3.4 Neighborhood Pattern Sensitive Fault Test DRAM BIST

Kinoshita and Saluja [358, 359] present an algorithm to test DRAMs for *static neighborhood pattern sensitive faults* (SNPSFs) (see Chapter 9.) The test has two parts: a MATS+ test (see Section 15.3.2) to test the address decoder, and a test for SNPSFs in a Type-1 neighborhood using the two-group method. The algorithm does not use parallelism, and can be also implemented on an ATE. The response compacter uses three simple count functions, described below The operation count is $58 \cdot n$, and the chip area overhead is 0.09% for a 1 Mb DRAM. This test is preferred over a pure march test for DRAMs, because its fault model better matches the actual faults encountered in production.

Fault Model. The algorithm uses a static *weight-sensitive fault* (WSF) model. This fault changes the content of the base cell, depending on how many 1s exist in the deleted neighborhood. A *t-WSF* is a WSF which occurs when any deleted neighborhood pattern has t cells at logic one and the remaining $k - t - 1$ cells at logic 0, assuming neighborhood size k. A *positive* (*negative*) WSF only allows the base

Step 0: {Assume all cells are initialized to 0};
Step 1: {Deleted neighborhood p2}
write 1 to all cells-A and all cells-B of group-1;
read all base cells 'b' of group-1;
write 0 to all cells-B of group-1;
Step 2: {Deleted neighborhood p3}
write 1 to all cells-D of group-1;
read all base cells 'B' of group-1;
write 0 to all cells-A of group-1;
Step 3: {Deleted neighborhood p5}
write 1 to all cells-C of group-1;
read all base cells 'b' of group-1;
write 0 to all cells-C of group-1;
Step 4: {Deleted neighborhood p6}
write 1 to all cells-B of group-1;
read all base cells 'b' of group-1;
write 0 to all cells-D of group-1;
Step 5: {Deleted neighborhood p4}
write 1 to all cells-C of group-1;
read all base cells 'b' of group-1;
write 0 to all cells-B of group-1;
Step 6: {Deleted neighborhood p1}
write 1 to all cells-A of group-1;
read all base cells 'b' of group-1;
write 0 to all cells-A and all cells-C of group-1;
Step 7-12: **Repeat** Steps 1-6 for group-2;

Figure 15.43: Algorithm to locate all positive, static 2-WSFs.

cell to change in the direction $0 \rightarrow 1$ ($1 \rightarrow 0$) due to a fault. A test that detects all positive and negative static t-WSFs (for $0 \leq t \leq 4$) also detects all SNPSFs [688].

SNPSF Algorithm. This algorithm detects and locates SNPSFs, and has a separate part for each value of t to locate the static t-WSFs (see Figure 15.43.) The part for $t = 0$ has been deleted because it is trivial, and we only show the algorithm for locating positive, static WSFs, as the algorithm for negative ones is very similar. It uses the 2-group method (see Figure 9.28.) See van de Goor [688] for the detailed operation analysis of this algorithm [358, 359].

Test Response Compression. Kinoshita and Saluja used three *count functions* to compress the responses of the DRAM from read operations. Here, r_i represents the value returned by read operation i, and c represents the number of times a read

Table 15.9: Count function values for good/failing machines.

Entry number	Permutation of response string	Count function		
		$C1(\mathbf{R})$	$C2(\mathbf{R})$	$C3(\mathbf{R})$
1 (good machine)	0011	2	1	1
2 (bad)	1100	2	0	1
3 (bad)	1010	2	1	3
4 (bad)	0101	2	2	3

has been performed by a specific algorithm.

$$C1(\mathbf{R}) = \sum_{i=1}^{c} r_i \tag{15.18}$$

$$C2(\mathbf{R}) = \sum_{i=1}^{c-1} \overline{r_i} \cdot r_{i+1} \tag{15.19}$$

$$C3(\mathbf{R}) = \sum_{i=1}^{c-1} r_i \oplus r_{i+1} \tag{15.20}$$

A count function detects a memory fault if, for the given test, the value of the count function differs from the reference, fault-free value. For $C1(\mathbf{R})$, the reference count values are 0 for 0-WSFs through 4-WSFs, and value n for *address decoder faults* (AFs.) The $C2(\mathbf{R})$ and $C3(\mathbf{R})$ reference counts are 1 for AFs. AFs are detected by $C1(\mathbf{R})$, $C2(\mathbf{R})$, and $C3(\mathbf{R})$. $C1(\mathbf{R})$ counts 1s, $C2(\mathbf{R})$ counts $0 \rightarrow 1$ transitions, and $C3(\mathbf{R})$ counts both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. Table 15.9 shows all 3 count function values when a MATS+ test is applied to a memory with $n = 2$.

Implementation. This BIST method requires no changes to the memory cell array.

1. It has been implemented using a microcoded ROM control having 665 6-bit words, a row and column address counter, a $\log_2(n)+1$ bit counter for $C1(\mathbf{R})$, a 2-bit counter for $C2(\mathbf{R})$, and a 2-bit counter for $C3(\mathbf{R})$ [359, 559]. The area overhead for a 64 *kb* RAM is 1.85% [559].

2. It was also implemented with a custom logic control, instead of the control ROM [559]. The area overhead decreases rapidly with the memory size n, because only the address counter size grows with increasing memory size. The area overhaed for a 6 *kb* RAM was 1.21%, for a 256 *kb* RAM it was 0.32%, and for a 1 *Mb* RAM it was 0.09%.

Several additional NPSF BIST algorithms exist. Mazumder *et al.* contributed *restricted* SNPSF algorithms for a Type-2 neighborhood [443, 444]. Static and dynamic *neighborhood pattern sensitive faults* (NPSF) in the Type-2 nine-cell neighborhood are detected in a parallel BIST environment. Multiple bit lines are selected in the BIST mode by a modified column decoder, and the same data is simultaneously written to several cells of the same word line. During reading, an additional

multi-bit comparator checks the contents of the bit lines. Mazumder *et al.* also gave an algorithm for ANPSFs and SNPSFs in a Type-1 neighborhood [444], and another BIST method based on pseudo-random techniques [441].

15.3.5 Transparent Memory BIST Tests

Transparent BIST eliminates the trouble of restoring the RAM contents after the system function has been interrupted for a periodic memory testing episode. The basic principle is that during testing, the memory stored data is complemented an even number of times. Assume that cell c contains bit v. Nicolaidis [495] then modifies the memory testing algorithm as follows:

1. Add initial memory read operations to the original algorithm.
2. Replace every $write - x$ operation on cell c with a $write - (x \oplus v)$ operation.
3. If the last write on cell c stored $\overline{v}$, add an extra read and an extra write operation to complement the cell data.

The sequence of values produced by the read operations is compacted into a signature. First, the test is run without any write operations, to calculate the signature. Then, the test is rerun with both the read and write operations. After all test patterns are applied, the actual signature is compared with the reference one.

The BIST controller must be augmented to generate not only the test sequence, but also the signature predicting sequence before the actual test. The controller must switch between the test application and signature production modes, to avoid write cycles during signature production. An extra register is needed to store the contents of addressed cells, which are used for test data generation. The transparent BIST area overhead for a 32 Kbyte RAM with the MARCH C test was 1.2% [495], only 0.2% more than conventional memory BIST with MARCH C.

15.3.6 Complex Examples

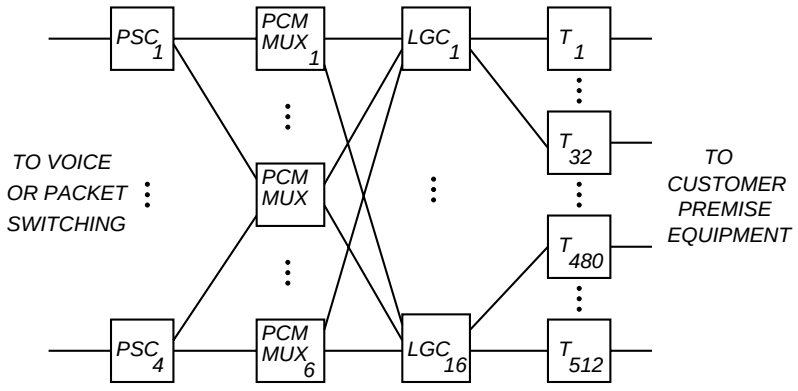


Figure 15.44: BIST of telephone exchange switch.

Example 15.16 Subsystem testing of a switching system at Lucent Tech. *This subsystem provides the Integrated Services Data Network (ISDN) capability in a telephone exchange [33]. Figure 15.44 shows a block diagram of this hardware. The inputs on the left side come from the digital voice or packet switching units. BIST was incorporated into the four PSCi chips, along with a hardware loopback capability so that any output data bus on the right side of the diagram could be looped back into any data input on the left side. In addition, intermediate points in the switch could be selected for loop back. More hardware is tested when the loop back includes the PSCi and Ti units. The BIST increased the PSC chip active area by 19.7%, but since the chip size had been increased to the minimum size for manufacturing, BIST did not increase its overall size. In terms of the total circuitry in this system, the overhead was only 4%. However, the extra hardware caused a minor decrease in yield, though in actuality, the cost was minor. The BIST area overhead at the circuit pack level was 1%. The system provided a 60% stuck-fault coverage, which was significantly better than what could be achieved by sourcing patterns at its inputs. This improvement in fault coverage simplified the writing of diagnostics, and reduced diagnostic run time by a factor of 8.*

Example 15.17 Combined sequential circuit and embedded RAM BIST [33]. *Table 15.10 shows an example of BIST used in a telephone exchange. The control RAM (CRAM) provides a control function. Figure 15.45 shows the block diagram of the CRAM memory system, and Figure 15.46 shows the automated circular BIST approach used. The* Pseudo LFSR *is a modified BILBO used simultaneously for both pattern and response compaction, built out of system flip-flops with added logic to XOR in the shifted bits from the prior BILBO stage during test. The method achieved nearly 98% fault coverage of stuck-at, transition, and neighborhood pattern sensitive faults for the memory and stuck-at faults for the random logic. Although this is outstanding, there is no guarantee here that the more exotic memory coupling faults and data retention faults will be detected, since the pattern sequence has not been proven to cover these.*

15.4 Delay Fault BIST

It is also possible to test circuits for timing delays using BIST. A delay fault BIST testing system has the standard BIST architecture, but with a hybrid pattern generator optimized to test both stuck-faults and delay faults replacing the standard LFSR pattern generator.

Motivation. For delay fault built-in self-testing, one of the problems that must be addressed is hazards in the circuit. Figure 15.47 illustrates the nature of this problem. Here, we are testing for distributed path delay for a falling transition (from logic 1 to 0.) The delay is caused by wire delays from A to the AND gate, delay in the AND gate, and additional wire delay from the AND gate to F. The

Table 15.10: BIST of embedded RAMs.

Aspect	Explanation					
BIST application:	Full chip random sequential logic & embedded RAM					
Levels of testing:	VLSI device, circuit pack, & system test					
Product:	Electronic Switching System					
Specific function:	Control RAM					
Detail:	Controls 8K bit RAM read sequentially by a counter RAM written into/read from a processor interface					
Technology:	Old: 2.5 μm CMOS, New: 1.25 μm CMOS					
Redesign reason:	Technology update					
BIST control:	Only 1 pin					
Faults tested:	Stuck-at, RAM transition, RAM NPSF					
BIST approach:	LFSR in a BILBO, used simultaneously for pattern generation & response compaction – non-linear circular BIST system, with loopback circuit					
Approach reason:	Lower overhead than with linear BIST					
Problem:	Signal correlation in pattern generation					
Input isolation:	MUXes at input buffers driving combinational logic Blocking gates at input buffers driving flip-flops					
Test sequencing	Two BIST sequences, combined into a single one					
BIST patterns:	RAM initialized using RAM address as data, Signals from each LFSR propagate through circular chain – combined in Output Data Register. The same LFSRs compact RAM BIST & random logic BIST results					
Signature:	Read from Output Data Register by processor					
Signature Requirement:	*Good machine* signature must match old product. Done by seeding Output Data Register before BIST					
Logic overhead:	29%					
Area overhead:	CRAM function	RAM BIST	Logic BIST	Input isolation	Inte-gration	Total
(% Active area)	90.3 %	0.6 %	6.5 %	0.8 %	1.7 %	9.7 %
Fault coverage:	BIST 94 %	I/O Buffer external test 3.7 %			Total 97.7 %	Prior 96 %

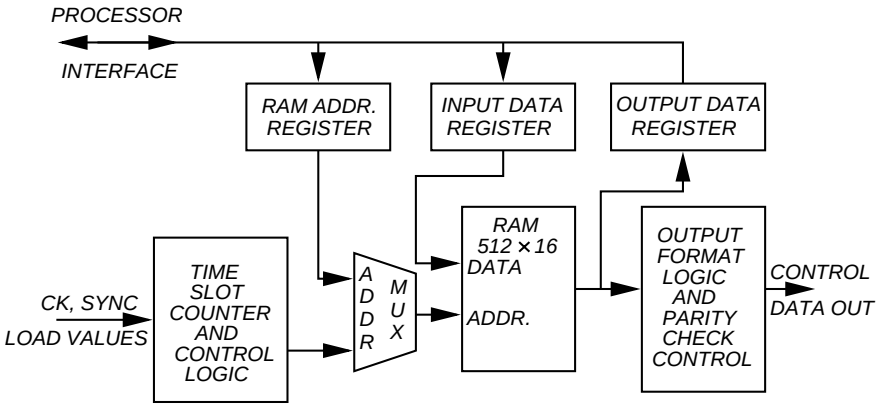


Figure 15.45: CRAM block diagram.

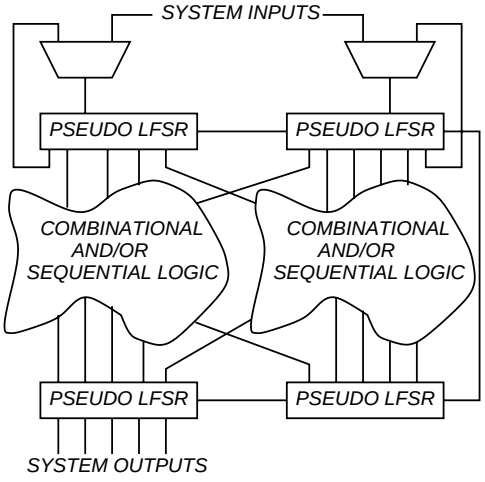


Figure 15.46: Automated circular BIST approach.

testing method is to apply two patterns, separated by the δt shown between the dotted lines. The path-delay fault testing requirement is that the off-path input (the OR gate output) must be a 1 in both time-frames. In the good machine timing diagram, the output F is sampled at a time given by the sum of δt and the *Path Delay Specification*. However, in the failing machine, where the signal is late propagating from A, a hazard appears at the logical sampling time, caused by the switch in the logic 1 value from signal B to signal C. Since the hazard arrives at the sampling time, testing is fooled into accepting a circuit with a timing defect as good. This *test invalidation problem* can be alleviated by using a delay fault BIST pattern generator that reduces the circuit hazard activity.

Delay Fault Testing Pattern Generation. One way to avoid hazards is to use a hardware pattern generator that creates *single-input changing* (SIC) patterns, so only one input changes during each clock period. This reduces the likelihood of generating hazards during testing, although some will still occur [593, 594]. Breuer and

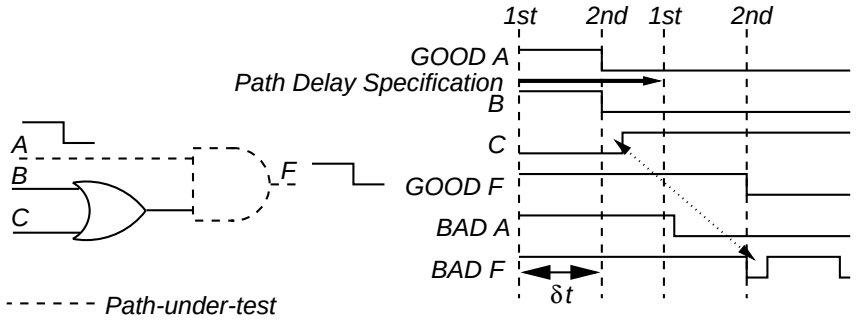


Figure 15.47: Test invalidation by hazards during delay fault testing.

Nanda [97] used a Gray code pattern generator, which must be partitioned, because otherwise for n PIs, this pattern generator goes through a cycle of 2^n patterns, which is intractable for large n. Figure 15.48 shows a hybrid pattern generator patented by Shaik and Bushnell [593, 594]. When the MUX is set to the top feedback path, this is an ordinary External-XOR LFSR pattern generator. When the MUX is set to the lower feedback path, this uses a Johnson or moebius counter of length n, which leads to $2 \times n$ patterns.

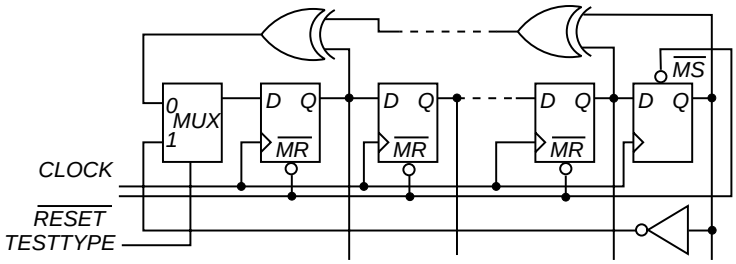


Figure 15.48: Hybrid delay fault testing hardware pattern generator.

15.5 Summary

BIST is now becoming more accepted as the preferred method of VLSI circuit testability insertion. This is because BIST hardware overheads have come down, particularly for memory BIST (1-3%), but also because BIST enables partitioning of the testing problem for large hardware systems. At present, memory BIST is widely used. Linear feedback shift registers, multiple-input shift registers, and built-in logic block observers are the most commonly used schemes to provide pattern generation and response compaction for BIST. Also, random logic BIST, while not yet fully accepted, has overheads of 13 to 20%, and experimental random logic BIST [111, 515, 516] has a chip area overhead of 6.5%. Random logic BIST has been used by IBM and Lucent Technologies. Finally, experimental systems for path-delay fault BIST also exist [515].

Problems

15.1 *Test length.* If $N = 15$ patterns are produced by an LFSR, and 2 of those patterns detect a given fault, say e stuck-at 0, what is the average test length T to detect e stuck-at-0? *Hint:* See the paper by Wagner *et al.* [702].

15.2 *Standard LFSR.* Implement a standard LFSR for the characteristic polynomial $f(x) = x^8 + x^7 + x^2 + 1$. Write the system of equations with the *companion matrix* for this LFSR.

15.3 *Modular LFSR.* Implement a modular LFSR for the characteristic polynomial $f(x) = x^3 + x + 1$. Write the system of equations with the *companion matrix* for this LFSR.

15.4 *Standard LFSR.* Compute the first eight patterns generated by the standard LFSR with characteristic polynomial $f(x) = x^8 + x^7 + x^2 + 1$ and an initialization of "00000001," with the one in the least significant bit.

15.5 *Modular LFSR.* Compute the first eight patterns generated by the modular LFSR with characteristic polynomial $f(x) = x^3 + x + 1$ assuming that the LFSR was initialized to "001" with the one in the least significant bit.

15.6 *MISRs.* Figure 15.49 shows a multiple-input signature register of the STANDARD (external XOR) type. This MISR takes outputs from the circuit, labeled as A and B, and compacts their responses. Please convert this signature register into the equivalent MODULAR (internal XOR) type and draw this equivalent signature register. Although the equations representing these two response compacters are the same, the signatures will be different, so explain the relationship between the two signatures.

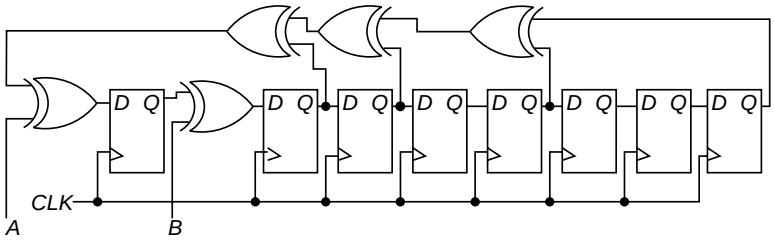


Figure 15.49: MISR for Problem 15.6.

15.7 *Weighted random patterns.* Apply four bits of the weighted pseudo-random pattern generator of Figure 15.16(b) to the four-input circuit: $f = (a \oplus b) \cdot (c \oplus d)$. For each of the four inputs, you can either choose one of X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0 (1/2 probability), $X_2 \wedge X_0$ (1/4 probability), $X_4 \wedge X_2 \wedge X_0$ (1/8 probability), or $X_6 \wedge X_4 \wedge X_2 \wedge X_0$ (1/16 probability.) If necessary, you may activate the *Inversion* signal to invert the probability. How many weight sets are needed to obtain 100% stuck-fault coverage for the faults in this circuit?

15.8 *Weighted random pattern generator.* Design a weighted pseudo-random pattern generator with programmable weights 1/2, 1/4, 11/32, and 1/16.

15.9 *Cellular automaton.* Build a four flip-flop rule 150 *cellular automaton* (CA), and compute its pattern sequence. Seed the CA pattern generator with "0001." What is the period of the cellular automaton? Compute the pattern sequence of the four flip-flop LFSR with characteristic polynomial $f(x) = 1 + x^4$. What is the LFSR's period? Is the CA better than the LFSR, and if so, then why?

15.10 *Maximal LFSR.* Design a 3-bit maximal LFSR, but please add hardware to map the test-pattern "010," which is not useful, into the pattern "000," which detects several circuit faults.

15.11 *Aliasing probability.* Using Figure 15.22, please compute the probability of aliasing for an error vector e with error probability $p = 0.3$, where a 15-bit LFSR is used for response compaction.

15.12 *Fault detection.* In Figure 15.23, can the transition counter detect the multiple stuck-at fault both b and c stuck-at-0? Can the LFSR detect this fault?

15.13 *LFSR enhancement.* Design test pattern embedding hardware to control an LFSR with the characteristic polynomial $f(x) = 1 + x + x^3$ to produce an all-zero test pattern. This problem also requires you to design the actual LFSR. Is this less hardware than just implementing a 3-bit binary counter?

15.14 *Aliasing analysis.* Consider the BIST system in Figure 15.50. Circuit inputs are A, B, and C, which are generated by the LFSR, and the outputs are Y and Z, which are compacted by the output MISR. The LFSR is initialized to 001 (i.e., the bottom LFSR flip-flop is set to 1 and the other two are cleared) and the MISR is initialized to 000. The circuit is clocked for eight periods to produce this test sequence:

L_1	0 1 0 1 1 1 0 0
L_2	0 0 1 0 1 1 1 0
L_3	1 0 0 1 0 1 1 1

The LFSR and the MISR are wired to the same clock line, and are fault-free. Explain why aliasing does not occur for the fault e stuck-at 0, even though it is expected since the test 001 for this fault is applied twice. What are the final good machine and bad machine signatures for the fault e stuck-at-0, eight clock periods after the LFSR and the MISR were initialized?

15.15 *Fault detection.* For Problem 15.14, find which of these faults are detected:

A sa0	A sa1	B-e sa0	B-e sa1	C-e sa0	C-e sa1

15.16 *Fault detection.* For Problem 15.14, find which of these faults are detected:

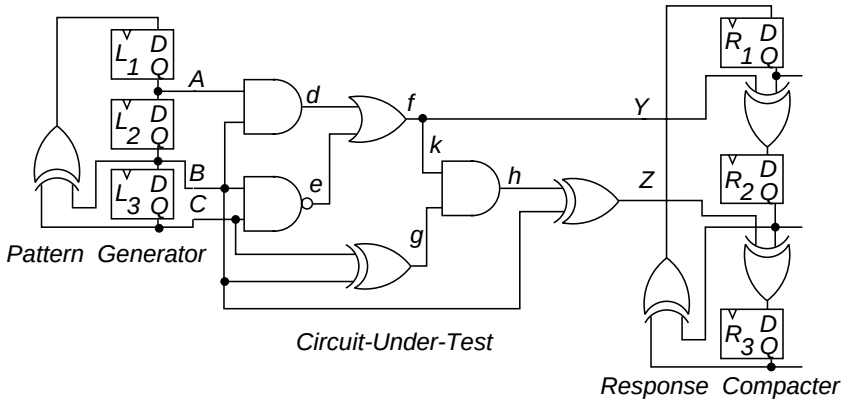


Figure 15.50: BIST system for Problem 15.14.

B sa0	B sa1	B-g sa0	B-g sa1	f sa0	f sa1

15.17 *Fault detection.* For Problem 15.14, find which of these faults are detected:

C sa0	C sa1	C-g sa0	C-g sa1	f-Y sa0	f-Y sa1

15.18 *Fault detection.* For Problem 15.14, find which of these faults are detected:

B-d sa0	B-d sa1	B-Z sa0	B-Z sa1	f-k sa0	f-k sa1

15.19 *Signature computation.*

(a) For the circuit in Figure 15.51, please design an external-XOR LFSR pattern generator implementing the characteristic polynomial $1+x^2+x^3$ and an Input MUX for testing.

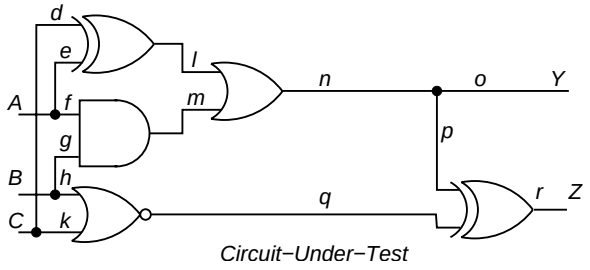


Figure 15.51: Circuit for BIST Problem 15.19.

(b) Express the linear system of matrix equations describing this pattern generator.

(c) Now, assume an exhaustive counter-based pattern generator and the given response compacter for the circuit in Figure 15.52. Compute the good machine signature for the circuit and design a single NAND gate

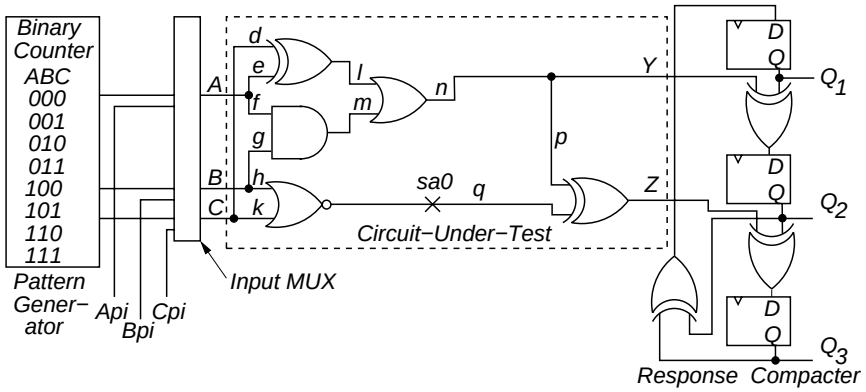


Figure 15.52: Circuit for Problem 15.19 with BIST hardware.

signature comparator that outputs a logic 0 on the $\overline{GOOD}$ signal when the circuit is good, and a logic 1 when it is faulty. The output MISR is initialized to "000" before testing.

(d) For the same circuit, compute the bad machine signature for the fault q stuck-at-0. Does the test hardware alias for this fault?

15.20 *STUMPS.* For the circuit of Figure 15.53, devise a STUMPS testing system. Drive the three scan chains with a maximal 3-bit LFSR, initialized to "001," and compact the output of the scan chains with a 3-bit MISR, initialized to "000." Simulate the system for 12 clock periods and provide the final signature. Discuss the pros and cons of STUMPS.

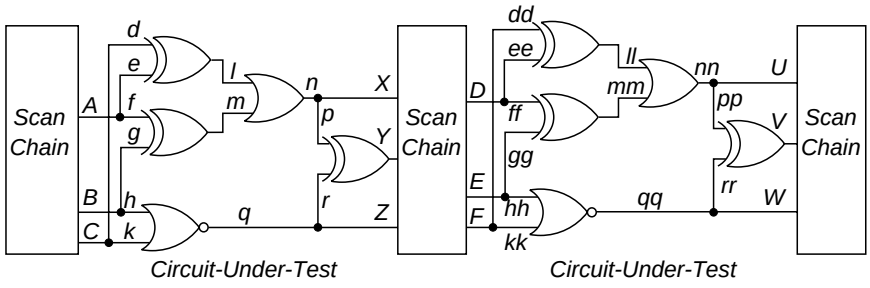


Figure 15.53: Circuit for BIST Problem 15.20.

15.21 *MATS+ memory BIST.* Implement the logic level hardware realizing the test controller documented in Figure 15.41 for the MATS+ memory BIST system.

15.22 *MARCH X memory BIST.* Implement the state transition diagram for the memory BIST controller for the MARCH X test described in Table 9.15.

15.23 *BIST system.*

(a) For the circuit in Figure 15.54, please design a 4-bit internal-XOR (modular) LFSR pattern generator implementing the characteristic polynomial

$1 + x + x^4$ and an Input MUX for testing. Note that one of the bits of the pattern generator will be wasted, as there are only 3 circuit inputs. Please include appropriate reset hardware.

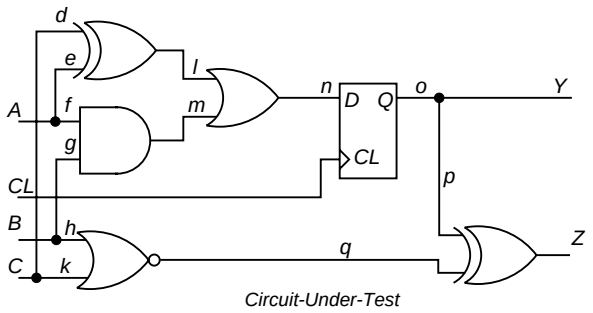


Figure 15.54: Circuit for built-in self-testing.

(b) Express the linear system of matrix equations describing this pattern generator.

(c) Now, design an external-XOR (standard) MISR for this circuit, using the same characteristic polynomial $1+x+x^4$. Note that only two of the MISR bits take circuit outputs (since the circuit has only two outputs.) Add appropriate initialization hardware for the MISR. What is the advantage of using a 4-bit MISR, rather than a 2-bit one?

(d) Express the linear system of matrix equations describing this MISR.

(e) In production testing, it was found that a significant number of these circuits failed due to the incorrect signature. Yet, *failure effect analysis* indicated that there were no defects (nothing was wrong) in these rejected circuits. Please explain what is wrong with this BIST testing process, and how to fix it.

15.24 *Up/Down LFSR.* Design a 4-bit LFSR using the tables in Appendix B and the up/down LFSR counting method of Section 15.3. The LFSR should take an $Up/\overline{Down}$ input signal to indicate whether it counts up or down. Also, design the LFSR to initialize to 0001.