Logic built-in self-test (LBIST), is a mechanism that lets an (IC) test the integrity of its own digital logic structures. LBIST operates by stimulating the logic-based operations of the IC and then detecting if the logic behaved as intended. The main advantage of LBIST is that it provides test capability without an external tester. In particular, safety-critical designs need to be tested and retested in a system or board.

To initiate testing, an LBIST engine inside the chip requires only an access mechanism like a TAP (test access port). When a device is powered on, LBIST can also check that the logic is working properly before starting any functional tests. LBIST also decreases test cost by shortening test-cycle duration and by reducing the complexity of the test/probe setup, since the number of I/O signals that must be driven and examined under tester control is usually smaller.

**LBIST Implementation**

LBIST can have many implementations, but most depend on generating pseudo-random patterns as stimulus for the logic. The response to this stimulus is captured in a MISR (Multiple Input Shift Register). If all the registers that hold state in an IC are on one or more internal scan chains, then the registers and the combinational logic between them can be used to generate a unique CRC (cyclic redundancy check) signature over a large enough sample of random inputs. The IC stores the expected CRC signature and tests for it after collecting a large enough pattern set from a PRPG (pseudo-random pattern generator). The result of the CRC comparison with the expected signature is typically accessed via a test port based on the JTAG IEEE 1149.1 standard. The CRC signature is unique in the sense that a defect-free DUT (device under test) always generates the same signature value at the end of the test process, and each failure in the device will lead to a different signature value.

The most popular LBIST architecture is called the STUMPS (Self-Test Using a MISR and Parallel Shift register sequence generator). See Figure 1. Test inputs are generated by a PRPG combined with a phase-shifter circuit. The output response analysis block includes the MISR and a signature analyzer. LBIST schemes based on the STUMPS architecture can generate test stimuli and analyze test responses with little or no help from external ATE (automatic test equipment). Patterns generated by the PRPG are applied to multiple scan chains in parallel and the output (test responses) of the scan chains are compacted into a signature by the MISR (see part 1 of this series for a description of scan chains). Any corruption in the output signature indicates a defect in the device.
Figure 1. A typical Logic BIST architecture.

One PRPG method is to use an LFSR (linear feedback shift register) to generate pseudo-random patterns (Figure 2). LBIST typically requires a sequence of 50K to 100K tests to obtain high fault coverage, but the LFSR method uses very little hardware and is consequently one of the preferred LBIST pattern generation methods.

Figure 2. A third degree standard LFSR.

You can improve on this basic approach by using an LFSR for the primary test mode. Some faults can be missed by the LFSR patterns, so to increase stuck-fault coverage to near 100%, you can generate test-patterns with an ATPG program. A typical LBIST controller includes the critical functions shown in Table 1.

Table 1. LBIST Controller Features
During LBIST execution, all the scan-chain flops in a circuit must capture a known (expected) state for each pattern. Any unknown values (referred to as Xs) that can occur in the circuit and captured into the scan chains will corrupt the MISR and LBIST results. If Xs are discovered during design, they can be eliminated by suitable initialization or masking techniques in the circuit. For Xs that are only discovered after fabrication, however, you need to mask corrupted scan chain results so they don’t affect the MISR signature.

Consequently, LBIST tools include chain masking features so you can effectively test the rest of the circuit using logic BIST in the presence of a bad chain.

**Memory BIST**

**Memory BIST**

Most designs today include a large number of embedded memories. You typically need additional circuitry to allow an external tester to access each embedded memory through the external I/O pins of the IC. This can negatively impact chip routability, area, performance, and design cycle time.

MBIST (memory BIST) eliminates the design effort needed to make embedded memories accessible for external test and provides at-speed self-test using algorithmically generated memory testing patterns. A serial data interface enables communication between multiple embedded random access memories (RAMs) and an MBIST controller. For testing very large memories or to achieve very short test times, MBIST can employ a parallel data interface (Figure 3).

**Figure 3. A typical Memory BIST architecture.**

An MBIST DFT (design-for-test) tool generates a test structure (MBIST control logic) for one or more embedded memories. It also includes a simulation test bench in Verilog for design verification and
shell scripts to drive synthesis of the RTL (resistor-transistor logic). Some MBIST tools automate the entire process of inserting the MBIST controller logic into the design as well as the interface circuitry between the controller and all memories located anywhere in the design hierarchy.

Most memory BIST solutions now support some form of BISR (built-in self-repair). Having the ability to repair embedded memories by swapping out defective rows or columns is increasingly important to achieve adequate yield levels as the size of embedded memories shrinks and their and densities continue to grow. A typical self-repair architecture consists of a fuse array to store repair information, a repair register placed next to each repairable memory for locally storing the necessary repair data, and a fuse controller for transferring data between the fuse array and each of the repair registers. All of the repair registers throughout the chip are typically placed on a serial chain in order to minimize routing. When the device is powered up, the fuse controller reads the repair info from the fuse array and scans it into all of the repair registers. This technique can be applied in the field, as well as during production test.

**BIST for Low-Power Devices**

The persistent growth of mobile computing is driving an increasing need to manage power consumption within semiconductor devices. Low-power requirements affect test in two ways. First, you need to ensure that any functional power constraints are met during test execution. Second, the test solution must be compatible with whatever low-power design techniques are being used.

The first requirement has generally meant ensuring circuit switching activity levels are held below a defined threshold during test pattern application. This is accomplished by generating the test patterns in such a way as to control the number of 1 to 0 and 0 to 1 transitions within each pattern. The transition frequency corresponds directly to circuit toggle activity and thus to average power. Controlling the number of transitions is relatively straightforward for deterministically generated patterns. In the case of LBIST however, because patterns are generated on-chip using a PRPG, controlling the number of transitions within each pattern is more complicated. Some form of processing of the pattern data generated by the PRPG is therefore required.

One approach is illustrated in Figure 4. Each PRPG output produces a stream of pseudo-random bits. These bit streams are fed into a phase shifter to produce a much larger number of pseudo-random bit streams to feed each of the individual scan chains within the circuit under test. To reduce the inherent toggle rate of each bit stream, a holding register is placed between each PRPG output and the phase shifter. A low-power LBIST module individually controls each of these holding registers. This module accepts a target transition frequency as input, and based on probabilistic techniques, periodically forces each of the holding registers to maintain its current value for a certain number of cycles. This produces bit streams that together produce an average transition frequency over the entire circuit under test equal to the desired target.
In addition to having BIST solutions operate under functional power constraints, be aware that these solutions operate correctly in the presence of low-power design features. A good example of where this requirement is important is in relation to on-chip memory repair. The basic self-repair architecture breaks down when voltage islands or power domains are used. This popular power management approach involves using a separate supply voltage for each core (or, possibly, group of cores) within a design. Each resulting power domain can then be shut down when not in use and re-activated when needed. This powering up and down activity has a direct effect on repairable memories. When a sleeping power domain is re-activated, the repair information for the repairable memories in that domain will have been lost and will need to be reloaded. The challenge here is that the reloading has to occur without disrupting the already active domains, and the reloading can’t be affected by the fact that some domains may still be inactive.

To handle these constraints, the self-repair architecture described above has to be augmented to provide at least one repair shift chain for each power domain, as illustrated in Figure 5. Each shift register can be of arbitrary length. A functional power management unit indicates to the fuse controller which shift register(s) need(s) to be loaded. The other shift registers are kept in a stable state as they might contain repair information for active power domains.
Figure 5. Power-aware built-in self-repair uses shift registers.

When multiple domains are reactivated, the controller will generally need to load them sequentially according to a default priority defined at design time. The operation is sequential because all repair information is typically stored in the same fuse array. If the loading order needs to be changed, the power management unit simply needs to reactivate each island one at a time in the desired order. The functional power management unit and the fuse controller must both be in an always-on power domain while the various MBIST controllers and repair registers are placed within the same power domains as the memories they service. Power domains can span multiple physical regions and a physical region can also contain multiple power domains.

Hybrid ATPG-LBIST

Because automotive and medical designs need high reliability in the field, they must be being tested with a combination of both pseudorandom LBIST patterns and deterministic ATPG (automatic test-program generation). With this hybrid ATPG/LBIST test technique, you can use ATPG to achieve very low DPM (defects per million), checking for small delay, cell-aware, and path delay types of defects. LBIST can also be used in post-manufacturing environments such as burn-in, system-level test, and in-field self-test. The ability of a device to periodically test itself in the field is a necessity in many safety critical applications, and is required to satisfy the reliability requirements specified within the ISO 26262 automotive safety standard.

Hybrid ATPG/LBIST solutions can share on-chip DFT resources such as scan chains and clock control logic. Also, the on-chip controller logic for both ATPG compression and LBIST can be integrated into a single block that is significantly smaller than the two separate implementations. The combined architecture provides the ability to apply combinations of compressed ATPG and random LBIST patterns. A single DFT automation flow enables both a flat as well as hierarchical integration of the hybrid capabilities. Such a flow can incorporate design rule checking, hybrid controller insertion and verification, scan insertion, and fault simulation integrated across both pattern types. A hybrid ATPG/LBIST controller can be accessed through a standard IEEE 1687 network, allowing easier access to the embedded test capabilities from anywhere in the system.

Also see
Design for test boot camp, Part 3: Advanced fault models and cell-aware test
Design for test boot camp, part 2: Test compression
Design for test boot camp, part 1: Scan test