AUTOMATED BIST FOR REGULAR STRUCTURES EMBEDDED IN ASIC DEVICES

Duane R. Aadsen, Harold N. Scholz, and Yervant Zorian

Duane R. Aadsen *and* ***Harold N. Scholz*** *are at AT&T Bell Laboratories' Cedar Crest facility in Allentown, Pennsylvania; and* ***Yervant Zorian*** *is with AT&T Bell Laboratories' Engineering Research Center in Princeton, New Jersey. Mr. Aadsen is a distinguished member of technical staff in the Technology Implementation Department, and is responsible for the design, characterization, testability, BIST, customer support, and automatic generation of ASIC SRAMS. He joined AT&T in 1978 with a Bachelor of Physics from the University of Minnesota, Minneapolis, and an M.S. and Ph.D. in physics from the University of Illinois, Urbana-Champaign. Mr. Zorian has been with AT&T since 1987, and is a member of technical staff in the Test and Diagnosis Department, where he researches the optimization of information loss in BIST,*
(continued on page 109)

Testing complex very large-scale integrated (VLSI) circuits is increasingly difficult. It is almost impossible to achieve high fault coverages without using design for testability (DFT) approaches. In one of the most promising DFT approaches, the complex VLSI circuit is divided into various structural modules, and an appropriate built-in self-test (BIST) scheme for each module is then implemented. This approach also provides a path to board- and system-level test. This paper presents a general procedure to implement BIST for a specific set of modules, i.e., embedded regular structures such as static random-access memory (SRAM), read-only memory (ROM), and register files. We will show that fault coverages achieved with this procedure are greater than 99 percent, because the deterministic BIST algorithms and output-data compaction techniques adopted are structure-specific. Moreover, fault simulations are not needed because the fault coverages are analytically determined by an algorithmic approach. The general procedure presented here results in an automated BIST implementation. An illustration of such a procedure is presented through the BIST register file example.

The size and complexity of electronic systems is growing exponentially. An electronic system can consist of one or more subsystems or boards. Each subsystem can consist of one or more components, such as power supplies, capacitors, resistors, transistors, and integrated circuits (ICs).

Modern technology is permitting much finer structures for ICs. To take advantage of these finer structures and reduce the number of subsystems and components, more functional building components

Panel 1. Terms and Acroynms in This Paper

ASIC	application specific integrated circuit
BC	BIST complete
BF	BIST flag
BFC	BIST flag check
BIST	built-in self-test
BRIC	BIST resource interface controller
BSBRIC	boundary-scan BIST resource interface controller
B-S	boundary-scan
CAM	content-addressable memory
CMOS	complementary metal-oxide semiconductor
CP	circuit pack
DFT	design for testability
FIFO	first-in, first-out memory
GALPAT	galloping patterns
IC	integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	input/output
JTAG	Joint Test Action Group
MACLOG	Macrocell Layout Generator
ODE	output data evaluator
PBX	private branch exchange
PLA	programmed logic array
RAM	random-access memory
ROM	read-only memory
SD	select data input
SI	scan-in data input
SO	scan-out data input
SRAM	static random-access memory
TPG	test pattern generator
VLSI	very large-scale integration

are now being integrated and embedded into each IC. The structure for these functional building components can be either regular or irregular. Regular components include SRAM; dynamic random-access memory (DRAM); ROM; content-addressable memory (CAM); programmed logic array (PLA); first-in, first-out memory (FIFO); shift register; and multiplier. Irregular structures include random logic.

A regular structure is a one- or two-dimensional array of similar or identical blocks surrounded by dedicated blocks. Memory cells are typical identical blocks. Input and output data registers, address registers, and address decoders are frequently used as dedicated blocks. The block diagram in Figure 1 represents the general design of such a regular structure.

Testability issues, along with systems and ICs, have grown in complexity. Previously, several techniques were used in testing systems: test-integration, bed-of-nails, and scan-testing.

- In *test-integration*, the tests for individual units are merged together to test a larger subsystem or system. These units generally interact, and some can become deeply embedded. Thus, test-integration often is a long, tedious, and costly procedure.
- The *bed-of-nails* technique was used for direct access to an IC. An array of pins (known as *nails*) would descend and make contact with the IC pins. Introducing fine-pitched packages, with pins closer together than those on the bed-of-nails tester, makes this technique unusable.
- *Scan-testing* required at least three extra signals: a scan-in data input (SI), a scan-out data output (SO), and a select data input (SD). With SD low, some or all flip-flops on an IC, subsystem, or system could be chained together serially into a long shift register. (A *flip-flop* is a cell in a circuit for storing 1 bit of information. It has two stable states, 0 and 1.) With SD high, the flip-flops were placed back into their normal functional relationship. For scan-testing, the SD would first be set low so data could be scanned through the chain via the system clock signal into the flip-flops via SI. SD would then go high for one clock cycle to exercise the logic between the flip-flops. SD would then go low again to scan out the pattern in the flip-flops via SO while setting

up a new pattern via SI. As complexity grew, scan-testing tended to get lengthy and required an expensive tester.

All these techniques have been used to test regular structures embedded in an IC or a system. They are becoming less feasible as complexity increases.

By 1985, the problems and costs associated with testing at the board and system levels had become so great that the Joint Test Action Group (JTAG) was formed by several European and North American companies. Its goal was to use boundary-scan (B-S) to standardize the board and chip level interconnect tests. Standardization was viewed by all companies as extremely important because ICs from different vendors could reside on the same board. B-S can be viewed as a form of scan-testing where the chain of flip-flops reside at the boundaries of the IC chips. It would thus provide a common access to the various chip-level testability resources for test and diagnosis at all assembly levels. In 1990, JTAG's proposal was adopted by the Institute of Electrical and Electronics Engineers (IEEE) as the IEEE 1149.1 standard.[1]

Chip-level testability resources include scan-testing and BIST. BIST is a means to generate test stimuli internally and compress the results. When ICs or regular structures run faster than the machines used to test them, BIST is the only way to test them adequately. Thus, several advantages can be gained by automatically adding BIST to an embedded regular structure:

- Higher fault coverage
- Test standardization
- Better controllability and observability
- Less test development and test application time
- Added debug and repair capability
- In-system, at-speed testability
- Increased quality.

The operation of BIST for regular structures must conform to the IEEE 1149.1 standard architecture.

Proper use of BIST, coupled with its controllability (through architectures such as IEEE 1149.1), clearly affects all major downstream processes (i.e., manufacture, distribution and installation, service and maintenance) as well as customer requirements. Most of the effects are found above in the advantages of automating BIST. Further examples will be discussed in the section "Tradeoffs: Advantages and Disadvantages."

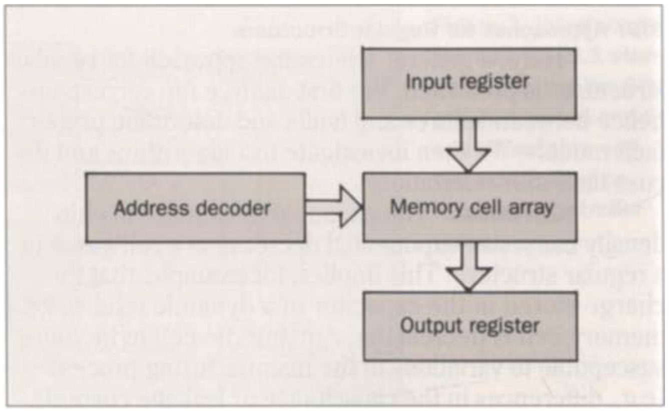

Figure 1. The general design of a memory structure, showing it as a one- or two-dimensional array of similar or identical blocks surrounded by dedicated blocks. Memory cells are typical identical blocks. Input and output data registers, address registers, and address decoders are frequently used as dedicated blocks.

At AT&T Bell Laboratories, BIST was first automatically integrated into a SRAM by using a parameterized modular-array assembler.[2] Given the RAM organization in the number of words and the number of bits-per-word, a RAM could be automatically laid out with BIST in under two minutes of computer time. This was achieved by the combined efforts of the system design, silicon IC design, and research and development areas.[3] Since then, BIST has been—and continues to be—applied to other regular structures.

In the remaining sections, we will present our BIST approach for regular structures, some BIST applications, tradeoffs on when and where to use BIST, and some future directions.

BIST Approaches for Regular Structures

Here, a general self-testing approach for regular structures is presented. We first analyze the correspondence between failures and faults and determine proper fault models. We then investigate test algorithms and discuss their BIST realizations.

Fault Models. The exponential increase in chip density causes an exponential decrease in a cell's area in a regular structure. This implies, for example, that the charge stored in the capacitor of a dynamic read-write memory cell is decreasing, causing the cell to be more susceptible to variations in the manufacturing process (e.g., differences in the capacitance or leakage current) and in use (e.g., noise or crosstalk). In addition, the cells are placed closer together. Thus, they are more sensitive to the influence of neighboring cells, which increases the likelihood of disturbances from noise, address, and data lines. Such physical failures cannot often be represented simply by the classical "stuck-at" fault model, as happens in the memory cell arrays in regular structures. (A "stuck-at" fault is a signal line in a circuit that is assumed to have its value fixed at a logical 1 or a logical 0.) Hence, the physical failures specific to memory cell arrays must be represented by a set of logical fault types (e.g., transition, coupling, pattern sensitivity, crosspoint) to constitute a realistic fault model for a regular structure.[4]

Transition, coupling, and pattern sensitivity faults are typical of a specific subset of regular structures, namely read-write memories.

- A *transition fault* occurs if a memory cell fails to undergo a 0 to 1 or 1 to 0 transition during a write operation.
- A *coupling fault* involves two cells, and also happens during a write operation if a transition in one cell changes the content of a second cell. However, if cell content or the possibility of changing that content is influenced by the contents of a group of cells (i.e., a pattern of 0s and 1s), the resulting fault is an example of pattern sensitivity.
- *A crosspoint fault* occurs mainly in read-only regular structures, e.g., PLAs and ROMs. These faults are caused by the unintentional presence or absence of features such as transistors or contacts.

The fault model of each regular structure includes a subset of the previously mentioned fault types. Identifying a fault model is based on analyzing the corresponding structure to find all possible fault types.

Test Algorithms. The conventional approach to test regular structures includes many well-known tests such as GALPAT ("galloping patterns"), checkerboard, and sliding diagonal for read-write memories,[5] and exhaustive tests for PLAs and ROMs.[4] These tests are not based on fault models and, therefore, do not allow estimates of test quality in terms of fault coverage. Some of the read-write memory tests require test times of about $O(n \log n)$ or $O(n^2)$, where n is the number of words in the memory. Because of the deficiencies in these conventional approaches, the subject of memory testing has been a major concern, and has resulted in much research in the subject area. Thus, a structured approach has emerged to determine test algorithms based on fault models of regular structures, allowing faults of the particular model for each algorithm to be found. By carefully selecting the fault models, the execution times of many test algorithms become of the order $O(n)$. Such test algorithms are well documented for read-write memories[3,5–6] and PLAs.[7]

This approach can also be considered structure-specific. That is, without fault simulation, it exploits the functional and structural characteristics of the regular structure in question, and yields a proper test algorithm with guaranteed fault coverage.[5] We discuss an example of such an algorithm for register files in the BIST applications section.

BIST Realization. Because of BIST's nature, test patterns must be generated and evaluated on-chip. The input patterns are generated by a block called a test pattern generator (TPG), and the resulting output data is compacted and evaluated on-chip by a block called the output data evaluator (ODE). As a rule in BIST, a TPG either generates pseudo-random or exhaustive test

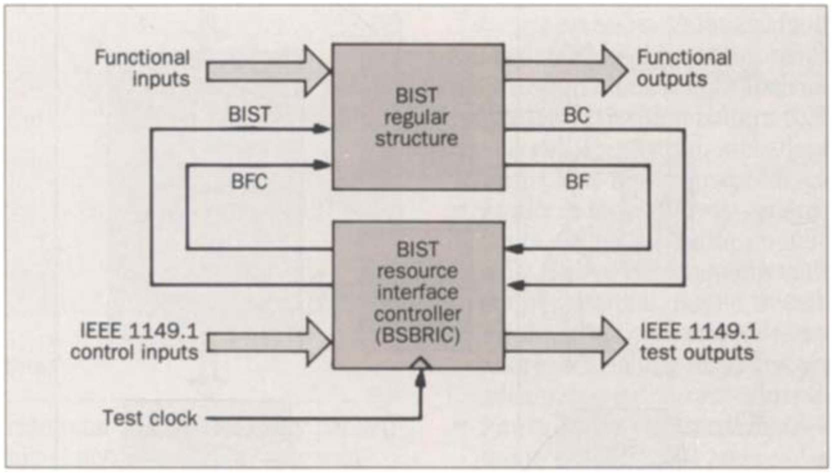

Figure 2. Interface to the IEEE 1149.1 standard showing the BIST and BFC inputs and BC and BF outputs and the BIST Resource Interface Controller (BRIC).

patterns, because from the hardware implementation standpoint, it remains expensive to generate deterministic test patterns.[8] But for regular structures, the deterministic test algorithms tend to have regular input patterns and therefore can be easily generated by simple hardware blocks. The register file example will also substantiate this.

But compacting output data in an ODE generally results in an information loss in terms of coverage. This loss of information, known as *aliasing*,[9] causes a subset of the pre-compaction detected faults to have the same final signature as the fault-free signature, and therefore to go undetected. Specific compaction functions based on the output data to be compacted are used in our regular structure case. These compaction functions either eliminate or reduce this information loss. Such a technique for the register file example is presented in our section on BIST applications. In all our regular structure cases, the final signature is compacted to single-bit information (GO/NOGO) stored in the BIST flag.

The BIST realization of a regular-structure test algorithm also contains a BIST controller, i.e., a state machine that executes the entire sequence of operations in an algorithm. The area overhead needed for this block depends on the complexity of the algorithm adopted.

BIST Access. As a rule, an on-chip run of a BIST algorithm must be controlled and observed through a specific protocol. Because our regular structures contain embedded controllers to perform the BIST operations sequence, and because their final signature consists of a single-bit flag, a simple protocol is needed just to begin the BIST execution and verify the BIST flag. The protocol adopted for our regular structures[10] adds two input and two output lines to each regular structure under BIST. The two inputs are BIST and BFC (BIST flag check); the two outputs are BC (BIST complete) and BF (BIST flag); see Figure 2.

The protocol starts by sending a signal on the BIST line to start the execution of the BIST algorithm. When the algorithm has executed, a signal appears on the BC line, and the test result (GO/NOGO) can be verified on BF. To insure that the BIST flag is not stuck at GO, an additional step is needed: the flag is forced to NOGO status by using the BFC line, and BF is verified again.

To control and observe the execution of BIST sessions in an application-specific integrated circuit (ASIC), some additional logic is needed to interface the external accessing port on one hand, and the embedded modules

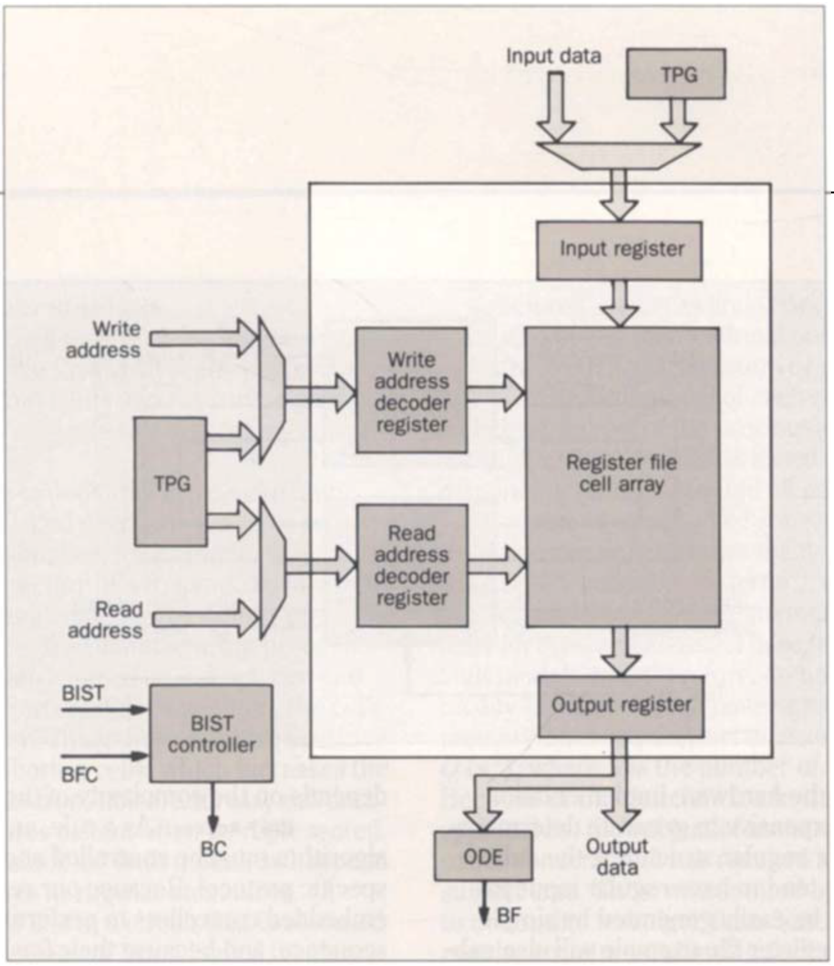

Figure 3. BIST register file realization, showing the dual port read-write memory, with synchronized read capability.

with BIST on the other. A dedicated interface block is required for the ASICs that have regular structures with BIST and contain the standard boundary-scan, IEEE 1149.1 test access port for external access.[11] This block already exists in the standard complementary metal-oxide semiconductor (CMOS) cell library, and is known as the BIST resource interface controller (BRIC),[12] shown in Figure 2. A BRIC communicates with the B-S port on one side and, through the above protocol on the other side, operates the modules with BIST.

BIST Applications

This section discusses the procedure for developing automatic generation of BIST blocks for embedded regular structures. Our example is the BIST register file, including the fault model determined from analyzing its design blocks, the register file's test algorithm and realization, and BIST's autonomous execution in a register file.

Overview of Regular Structures. BIST's automatic generation capability is provided through the Macrocell Layout Generator (MACLOG), the tool that also generates all the regular structures for the AT&T CMOS standard cell library. The BIST offerings by MACLOG have been used in automated mask layout generators for 1.25 micrometer (μm) and 0.9μm CMOS SRAMs; are now in place for the 1.25μm and 0.9μm CMOS register files, 0.9μm CMOS CAMs, 1.25μm CMOS ROMs; and are being extended to cover all other regular structures it contains (i.e., FIFOs, multipliers, PLA, shifters). The goal is that MACLOG will provide the versions of these regular structures using BIST by the end of 1990. Given the structure organization in words by

$\uparrow W^0$			
$\uparrow R^0$	W^1	W^0	W^1
$\uparrow R^1$	W^0	R^0	W^1
$\downarrow R^1$	W^0	W^1	W^0
$\downarrow R^0$	W^1	R^1	W^0
$\rightarrow W^0 \rightarrow$	$R^0 \rightarrow$		
$\rightarrow W^1 \rightarrow$	$R^1 \rightarrow$		

Figure 4. The BIST algorithm.

bits-per-word, the generator can output a densely packed structure and its customized BIST circuitry in only minutes. About 50 ASIC devices containing regular structures with BIST currently are being manufactured by AT&T. By automating BIST implementation in these devices, we have made significant development savings in both circuit design and test design.

The procedure adopted for an automated implementation of BIST is:

- Establish an appropriate fault model, given the design of a regular structure.
- Develop a corresponding BIST algorithm for high-quality test.
- Design on-chip testing facilities to implement the BIST algorithm.
- Design an autonomous BIST controller by adopting the generic external-control protocol to execute BIST.
- Parameterize and integrate all BIST facilities to the corresponding regular structure generator, MACLOG.

An application of this procedure is provided in the next section.

Specific Application: BIST Register File. The register file is second only to SRAM as the most used regular structure available through MACLOG. It consists of a dual port read-write memory (illustrated in the non-shaded part of Figure 3, with a synchronized read capability).

Fault Model. A logical fault model of the register file is determined based on structural analysis of the file's design blocks. This analysis also results in a customized initialization pattern to find physical faults in the layout. The fault model adopted for the 0.9μm and 1.25μm CMOS BIST register file consists of:

- *Faults in the cell array*—memory element stuck-at faults, transition faults, coupling between any two cells, linked two coupling faults (i.e., two interacting coupling faults), linked transition and coupling faults.
- *Faults in the address decoders*—accessing multiple addresses, failing to access appropriate address, addressing a different address.
- *Faults in the read-write logic*—stuck-at faults in input/output (I/O) and read-write register.
- *Faults in the additional BIST circuitry*—stuck-at faults in TPG and ODE blocks.

Test Algorithm. Figure 4 shows the BIST register file algorithm. It is a modified march algorithm[5] with a test length of $17n$, where n is the number of words in the memory. (A *march algorithm* is a deterministic sequence of read and write operations, usually of order n.) The algorithm is represented with a shorthand notation. Each row is a set of specific read (R) and write (W) operations applied to each address in incrementing (↑) or decrementing (↓) order, or to just one address. The superscript is the data value: all 0s, all 1s, walking zeroes (0→) and walking ones (1→).

For example, the second row means *read all 0s from the lowest address*, followed by *write all 1s*, then *write all 0s*, and *write all 1s into the same address*. Identical read-write operations are repeated for successive addresses until the highest possible address is reached. The last two rows of the algorithm represent read-write operations of walking 0s and 1s, through the data word → applied to a single address word only.

Each read-write operation is intended to cover one or more faults. For instance, the last write operation of the second row and the read operation on the third row verify that no cells of the memory cell array are stuck at 0, by forcing each cell to a write 1 and then a read 1. Detailed analysis of each operation and its correspondence to each fault type is beyond the scope of this paper, as are the theoretical proofs that guarantee the algorithm's 100-percent fault coverage.

BIST Realization. The input data patterns (all 0s, all 1s, walking sequences) and address patterns (i.e. ↑, ↓), are simple and regular because of the algorithm's regularity. Hence, the TPGs—the hardware needed to generate these patterns—are realized by simple logical blocks.[13] Therefore, these blocks are realized in one of two ways:

- *By modifying the existing registers.* Under the BIST mode, the external inputs will be blocked and the input registers will become test vector generators. The input data patterns are generated by modifying the input data register into to a rotating shift register. However, the address patterns can be generated by modifying the address register to an up-down counter under the BIST mode.
- *By adding extra TPG blocks with multiplexers to block external inputs.* A single TPG for address patterns is satisfactory for both read and write address sides, because the algorithm simultaneously applies the address patterns into both read and write address decoders.

Because of the test output data's regularity (all 0s, all 1s), the output data compactor adopted in the ODE is also composed of simple logic. It is designed not to lose any information after compaction; hence, the aliasing probability is zero. The output data modification concept[14] has been used to obtain the specific output data compactor. The design of the compactor consists of two gates: AND and OR. Both outputs of these two gates provide a zero (0) signal for all 0s, and a one (1) signal for all 1s. Therefore, monitoring the two outputs is adequate to avoid aliasing.

Automated Generation of BIST Regular Structures. The shaded boxes in Figure 3 show the BIST facilities needed for the register file. Although the exact area overhead depends on the parameters of a register file, it has been known to cause a hardware increase averaging 4 to 12 percent.

Autonomous execution of BIST in a register file is provided by the BIST controller block shown in Figure 3. This block consists of a four-state state machine that performs the exact sequence of operations in the algorithm. To automate BIST block generation, the TPGs and ODE need to be parameterized. The parameters here are the size of the data and the size and number of address words.

The final step is integrating all BIST blocks with the CMOS register file generator. Note that the above procedure has been simplified by taking advantage of the regularity of regular structures. Such a procedure has been utilized for other regular structures, including RAMs,[3] CAMs, and ROMs[15].

Tradeoffs: Advantages and Disadvantages

To make effective tradeoffs between BIST and non-BIST, one must first understand the advantages and disadvantages of both approaches. Here we will discuss the advantages gained at the chip level. Then, the disadvantages will be presented, followed by some examples showing the advantages and disadvantages at the printed-circuit board and system levels. The section will conclude with some recommendations.

Advantages at the Chip Level. The advantages gained by adding BIST at the chip level include:

- Higher fault coverage
- Test standardization
- Better controllability and observability
- Shorter test development time
- Shorter test application time
- Added debug/repair capability
- In-system, at-speed testability
- Increased quality.

When BIST is not used, good access to the I/O of the regular structures is a major concern. If the access is impaired, it limits the ability to apply a standard set of tests, leading to different tests that tend to be much longer. Thus, more test development and test application time are needed. Limited access to the regular structure I/O can also hamper the testing of certain faults. Also, the inability to do an in-system, at-speed test can allow some performance-related faults to go undetected. Thus, the quality of the product can be seriously compromised when BIST is not used.

Another advantage of BIST is its ability to parallel-test multiple regular structures. Without this ability, a block of I/O pins must be used to test a regular structure. When multiple regular structures are present, either several blocks of I/O pins must be used, or the regular structures must be tested serially. Obviously, a serial test is much longer. Scan-testing can greatly reduce the number of I/O pins needed to test any or all regular structures, but the test becomes even more extensive. When the standard protocol is used with BIST, the control pins can be shared with all regular structures. Multiple regular structures can then be tested in parallel, greatly reducing the test length for the whole chip. The only disadvantage to parallel testing is in designs that put temporarily unused regular structures on standby to reduce power consumption. Here, one would want to take the test time penalty and serially test the regular structures with BIST.

Disadvantages at the Chip Level. The disadvantages or penalties paid depend on the regular structure and BIST implementation. The SRAM generator integrates the BIST into the layout. This integration gives the added advantage of some fault tolerance due to a controlled layout. The penalties for the SRAM are: 0.5 to 5 percent extra chip area, 1 to 5 percent extra chip power, 20 to 40 percent extra setup time, negligible extra output propagation delay, and the possibility of extra chip pins.

The extra chip area depends on both the size of the SRAM and the size of the chip. For a 250 mil *x* 250 mil 1.25μm CMOS chip, the extra area overhead is less than 2 percent per SRAM. The extra chip power has a similar dependence. The setup time is small to begin with. By adding BIST, one is only adding one multiplexer or gate delay to the inputs. If the BIST is accessed through the IEEE 1149.1 standard interface or an existing microprocessor interface, there is no extra chip-pin penalty. Otherwise, the minimum extra chip-pin penalty is two pins.

If the BIST were implemented outside the SRAM as random logic instead of being integrated in the SRAM, the chip area overhead would be 1 to 10 percent. However, if several SRAMs could share this BIST random logic, the chip area overhead could be cut back. Because of the extra routing capacity (and the possible use of larger buffers to handle it), up to 50 percent more chip power and setup time may be needed than when integrating the BIST into the layout. Also, the circuit timing for implementing BIST with random logic cannot be precharacterized. IC designers or users must simulate, rebuffer, and characterize their use of the random logic. Thus, integrating BIST into the layout of the regular structure is most desirable for the user whenever its overhead is not shared between multiple regular structures.

The cost to integrate BIST into the regular structure layout also is important. This integration increases the layout's complexity, development time, and characterization time for the regular structure. The added costs for these increases must be shared among all users. We must also consider the unknown cost associated with regular structures that are not designed while the regular-structure designer is integrating BIST into an existing regular structure. Thus, the integration of BIST into the layout is done only for the most frequently used regular structures. The number of users required to make this effort cost effective is still being debated, but it is at least ten.

Examples of Using Non-BIST and BIST Regular Structures. It is difficult to make quantitative cost estimates for use or non-use of BIST on regular structures. To do this correctly, comparison must be made between two systems, where the only difference is BIST's presence or absence. Costs taken into consideration must include:

- Circuit designs for the integrated circuits, the printed

circuit boards, and the system
- Writing (and rewriting) of the test stimuli
- Simulations
- Testing time at the integrated circuit, printed-circuit board, and system levels
- Test equipment needed
- Inventory levels required to meet production goals
- Personnel required to meet production goals
- Time for debugging and maintenance
- Any field-returns.

Because of the high overhead cost involved in each system, the only practical way to do this might be to add BIST to an existing non-BIST system. But this is becoming impractical because the lifetime of systems is so short. We are left, therefore, with examples and estimates.

Non-BIST Regular Structures. In the first example, an IC with non-BIST RAMs was manufactured. The RAMs were so embedded in this device that the accepted RAM test could not be applied, and much time was spent writing test vectors for them. Unfortunately, some faults still slipped through, but were not detectable until the IC was in the field.

Additional time was then spent to find the particular test pattern that made the part fail, to identify the type of failure that resulted, to write an additional bank of test patterns to add to the test at manufacture, and to replace the parts in the field. The result was a much larger added expense in debugging, test pattern writing, and part replacements. Also, 30,000 additional test vectors or stimuli were needed, compared to the 10,000 test vectors used by the accepted test algorithm in the standard RAM test, which could not be applied.

The customer perhaps still has the uneasy feeling that another missed set of faults may show up in the field. No information is available on the time or money lost in each of the operations. If such information were known, it would be meaningless because it would depend on the IC chip itself, the total amount of chips involved, and when the problem was detected. The important point here is that the amounts are rarely small.

BIST Regular Structures. In the second example, BIST was chosen for use on each of the embedded SRAMs in the AT&T Definity® PBX system. Once the decision was made, BIST and testability features were added to the rest of the system. In estimating the cost savings, the Definity system's management found that this BIST approach:
- Required less expensive test and repair equipment
- Reduced misdiagnosis of ICs
- Minimized circuit pack (CP) or board scrap rate
- Reduced CP and system labor costs
- Reduced CP and system inventory
- Substantially eased testing efforts
- Reduced system troubleshooting
- Needed fewer service technicians.

The lowered cost for test and repair equipment was the largest impact.

If we assume an annual production of 1,000 CPs and January 1987 costs, the savings were estimated at several million dollars. Though using BIST on the embedded SRAMs did not contribute to all these savings, it was needed to make them possible.

When to Use BIST. BIST should be used with regular structures under any of the following conditions:
- Reduced IC or system controllability
- Reduced IC or system observability
- Nonavailability of test equipment for in-system, at-speed testing.

BIST should be considered when any of the following is considered important:
- Reduced design cycle time
- Reduced test development time
- Reduced test application time
- Added system debug and repair capability.

Non-BIST should be used when the regular structures can be fully controlled and observed, and test equipment is available for in-system, at-speed testing. Note that all these conditions *must* be present. If any condition is not met, quality will be affected. It is not always clear that the non-BIST approach will be the least expensive. This was demonstrated by our two examples.

Directions

Current work in BIST for regular structures centers around test algorithms and the logic to fulfill them. Future work will minimize the cost of using BIST. Cost reductions will be seen by minimizing chip area, chip design interval, and test time. Future developments should make it not only possible, but also desirable, to include BIST for all regular structures.

Minimizing Chip Area. BIST's required chip area can be minimized by optimized logic, more efficient layout techniques, or BIST overhead logic shared between multiple regular structures. The current implementations of the logic are well optimized, and little chip area can be saved. However, a custom layout of the BIST logic can reduce the BIST area by about 10 to 20 percent over the standard-cell layout. The disadvantage of a custom layout is increased design time and higher costs to change the algorithm.

Another approach to reducing BIST area is to share BIST structures among similar blocks. The large potential savings for this approach increases as the number of blocks sharing the same BIST circuitry increases. For example, if two register files share the same BIST circuitry, the area required per register file to implement BIST is reduced by nearly 50 percent (some extra logic is needed to allow sharing). This sharing technique has been used in several designs to reduce the area penalty of BIST.

There are two tradeoffs involved in sharing BIST logic. Tests can be run simultaneously on all the blocks to reduce test time. But this increases the BIST logic area, because each block will require its own data compaction circuitry. The second technique is to test each block sequentially, reducing area but increasing the time to test all the blocks. Both techniques are useful and are being investigated.

Minimizing Chip Design Interval. The time needed to design an ASIC is constantly being reduced. Adding BIST for regular structures relieves the designer of worry about test algorithms. However, the BIST logic still needs to be interfaced to the rest of the design.

The boundary-scan BIST resource interface controller (BSBRIC) provides an interface to the IEEE 1149.1 test port, simplifying the design job. The current BSBRIC is limited to one BIST block.

A controller ("super BRIC") is needed to interface multiple BIST blocks. Not only will the super BRIC have to control multiple blocks, but it also will have to sequence them. Often, blocks can be tested simultaneously, but because of power dissipation and power supply noise issues, sequential testing is often required. The need for both simultaneous and sequential control makes an efficient super-BRIC design both interesting and important for the next generation of ASICs.

Minimizing Test Time. Test time becomes an issue when an ASIC goes into manufacture. The cost of testing goes up as the test time increases. As noted, the choice of simultaneous or sequential testing can noticeably effect test time. Test times can also be reduced by improving the test algorithms to reduce the total number of clock cycles needed to detect all known faults.

Another technique to reduce test time would be to provide programmable BIST controllers. These controllers would run a subset of the full test suite available. The subset selected could depend on where in the production cycle the device is being tested. At chip test, for example, all tests may be performed. In a board test, however, only a partial set would be enough to determine that the chip is still healthy. Thus, test time could be optimized for each step in the production cycle.

Future Developments. Several developments could aid in more efficient design of the BIST logic for a new block. A library of parameterizable BIST components—such as controllers, pattern generators, and compressors—are being developed. These components could be reused in new BIST designs.

Another more ambitious approach is to design programmable BIST elements that can produce various test algorithms. These could then be programmed to test the new block. The drawback of this approach is increased overhead. But it may significantly reduce the design time for a new block or a special block that was designed for use on only one ASIC.

Much work remains to be done to completely optimize the use of BIST for regular structures on ASIC devices. Because the optimum conditions may vary for each ASIC, tests and interfaces to BIST must be flexible.

Summary

An application of implementing BIST to the register file was presented. Examples were given that showed the impacts of both using and not using BIST. BIST often should be mandatory for regular structures.

We have indicated that BIST has been implemented on several types of embedded regular structures in automated, parameterizable ways. This automation can come about either by implementing BIST with random logic, or by integrating BIST directly into the layout of the regular structure. Moreover, the BIST algorithm and implementation depend strongly on the type and internal architecture of the regular structure. With some small penalties, more than 99 percent fault coverage can be achieved for the regular structures.

These BIST implementations also can be easily accessed through the IEEE 1149.1 standard protocol. Thus, one immediately obtains testability access to the embedded regular structure up to the subsystem and system levels. The ability to reduce greatly the assembly and test costs while increasing the quality of the system or product is now attainable.

More efforts are under way to increase the BIST usage. Making BIST easier and even more efficient to use will go a long way toward that goal.

Acknowledgements

We are very grateful to our AT&T colleagues who supported us in this work, especially the following: Edwin Brown, Scott Davidson, Sunil Jain, Bahman Keramati, Stefanie Lam, Larry Roybal, Charles E. Stroud, Rod E. Tulloss, Shianling Wu, and Chi W. Yau.

References

1. IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture, IEEE Standards Office, Piscataway, New Jersey, May 1990.
2. D. R. Aadsen and S. K. Jain, "Automation of BIST for Embedded RAM," *Proceedings of the IEEE 1987 Custom Integrated Circuits Conference*, Portland, Oregon, May 4-7, 1987, IEEE Publishing Services, pp. 66-67.
3. S. K. Jain and C. E. Stroud, "Built-In Self-Testing of Embedded Memories," *IEEE Design and Test of Computers*, IEEE, Piscataway, New Jersey, Vol. 3, No. 5, October, 1986, pp. 27-37.
4. A. G. van de Goor, *Testing Semiconductor Memories: Theory and Practice*, John Wiley & Sons, Chichester, England, to be published September 1990.
5. M. S. Abadir and H. K. Reghbati, "Functional Testing of Semiconductor Random Access Memories," *Computing Surveys*, Vol. 15, No. 3, September 1983, pp. 175-198.
6. M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," IEEE International Test Conference, Philadelphia, Pennsylvania, IEEE Computer Society Press, Washington, D. C., November 1982, pp. 236-239.
7. V. K. Agarwal, "Easily Testable PLAs," *VLSI Testing*, T. W. Williams (ed.), New York, North Holland Publishing Company, 1986, pp. 65-94.
8. E. J. McCluskey, "Built-In Self-Test Techniques," *IEEE Design and Test of Computers*, Vol. 2, No. 2, April 1985, pp. 21-28.
9. Y. Zorian and V. K. Agarwal, "A General Scheme to Optimize Error Masking in Built-In Self-Testing," *Digest of Papers: The 16th Annual International Symposium on Fault-Tolerant Computing Systems (FTCS-16)*, Vienna, Austria, IEEE Computer Society Press, Washington, D. C., July 1986, pp. 410-415.
10. AT&T 1.25 CMOS Cell Library, "Standard Cells and Function Blocks," AT&T Marketing, Allentown, Pennsylvania, 1989.
11. Y. Zorian and N. Jarwala, "A Structured Approach to Complex VLSI Testing Using BIST and Boundary-Scan," *Proceedings of ATE and Instrumentation Conference West*, Anaheim, California, Miller Freeman, Boston, Massachusetts, January 1990, pp. 69-74,
12. H. N. Scholz, R. E. Tulloss, C. W. Yau and Wach, "ASIC Implementations of Boundary-Scan and Built-In Self-Test (BIST)," *Journal of Semicustom ICs*, Elsevier Science Publishers, Ltd., London, Eng-

land, Vol. 6, No. 4, 1989, pp. 30-38.
13. Y. Zorian, "A Structured Approach to Macrocell Testing Using Built-In Self-Test," *Proceedings of IEEE Custom Integrated Circuits Conference (CICC)*, Boston, Massachusetts, IEEE Publishing Company, New York, May 1990.
14. Y. Zorian and V.K. Agarwal, "Optimizing Error Masking in BIST by Output Data Modification," *Journal of Electronic Testing: Theory and Applications (JETTA)*, Kluwer Academic Publishers, The Netherlands, March 1990, pp. 59-71.
15. Y. Zorian and A. Ivanov, "EEODM: An Effective BIST Scheme for ROMs," *Proceedings of IEEE International Test Conference (ITC)*, to be published September 1990.

Biographies (continued)

develops BIST methodologies for regular structures, and consults with device designers. He holds an M.Sc. in computer engineering from the University of Southern California, Los Angeles, and a Ph.D. in electrical engineering from McGill University, Montreal, Canada. Mr. Scholz is a supervisor in the Technology Implementation Department, and is responsible for the Boundary Scan Library, MACLOG, and third-party tool libraries and interfaces. He has a B.S. and M.S. in physics from the University of Akron (Ohio), and a Ph.D. in physics from Ohio State University, Columbus. He joined AT&T in 1981.

(Manuscript received January 9, 1990)