SERIAL INTERFACING FOR EMBEDDED-MEMORY TESTING

BENOIT NADEAU-DOSTIE
ALLAN SILBURT
BNR

VINOD K. AGARWAL
McGill University

The authors present a serial interfacing scheme in which several embedded memories share the built-in, self-test circuit. For external testing, this approach requires only two serial pins for access to the data path. There is considerable savings in routing area and fewer external pins are needed to test RAMs with wide words, such as those in telecommunication ASICs. Even though the method uses serial access to the memory, a test pattern is applied every clock cycle because the memory itself shifts the test data. The authors have adapted this method to four common test algorithms. In their implementations of the BIST circuitry on several product chips, they found the area overhead acceptable.

Application-specific digital ICs have become increasingly memory intensive in recent years. The design flexibility and performance offered by module generators for single-port and multiport SRAMs and content-addressable memories are opening up new possibilities in system design as the constraint of standard memory dimensions and operating modes is relaxed. However, these new possibilities in design are posing several new challenges for test. For example, today's telecommunication ICs often have a variety of multiport memories on one chip, some with very wide words. External testing becomes complex because it is difficult to bring all the signals to the pins. Tester hardware and design-automation tools also impose limitations on external testing because usually neither can support the large algorithmic pattern sequences in simulation or high-level test generation. Moreover, even if an algorithmic memory test was expanded into individual vectors, they still must be stored and loaded. These requirements could easily exceed a tester's maximum pattern depth, particularly if the patterns had to be multiplexed.

It is these challenges to external testing that are making built-in self-test more attractive to chip designers and manufacturers. Additional incentives for BIST are the ability to provide full-speed tests with minimal test hardware and the reuse of these tests for diagnostics at the board and system levels.

Various BIST algorithms and techniques have been proposed,[1-8] most of which evaluate all the bits of a memory word in parallel as it is read. Some[1,4,6,7] require modification of the RAM circuitry, which may not be practical or even possible in many ASIC design environments. Others[4,6,7] perform signature analysis and then compress the data. These techniques must then consider aliasing uncertainties when calculating fault coverage.[9] We can encounter significant problems when applying these BIST schemes to chips that have multiple embedded RAMs of varying sizes and port configurations. If each memory required a dedicated BIST circuit, the chip area devoted to testing would be unacceptably high. A better approach is to share BIST circuitry among several RAM blocks. We would also want to reuse as much test circuitry as possible in the normal operation of the chip, called the mission mode. With existing BIST schemes, sharing the circuitry between BIST and mission mode can be quite difficult.

Our solution is a serial interfacing technique for embedded RAMs that allows the BIST circuit to control a single bit of a RAM's (or group of RAMs') input data path. Only one bit of the output data path is available to the BIST circuit for observation while the test algorithms are executed.

0740-7475/90/0004-52$1.00©1990 IEEE

The other bits are controlled and observed indirectly through the serial data path built using the memory itself and a set of multiplexers. We have successfully applied this serial interfacing approach to static single-port and dual-port memories in custom ICs and have observed the following benefits:

- Only a small amount of additional circuitry is required.
- Only a few lines are needed to connect the RAM to the test controller.
- Several RAM blocks easily share the BIST controller hardware.
- The serial-access mode does not compromise the RAM cycle time. We can perform a read or write cycle with new data at the full mission-mode clock rate.
- When test algorithms are adapted to the serial interface, the fault coverage is high.
- Existing memory designs do not need any modification to use the interface.

BNR has developed automatically generated BIST circuits around this serial technique. The circuits embed an algorithm suited for application to a RAM. We have also used the serial interface to provide external access to memories on cost-sensitive chips that could not justify the full BIST overhead. When used for external testing, the serial approach allows us to use the minimal number of pins yet exercise the memory at full speed.

We have used the serial interface on cost-sensitive chips that could not justify the full BIST overhead.

SERIAL ACCESS

Three types of signals must be controlled and observed during a memory test: the data lines, the address lines, and the control lines. As Figure 1 shows, the control-line signals consist mainly of the Read, Write, and Enable strobes. With the serial-access technique proposed, only one data line is controlled and observed during the test. This technique is applicable to static RAMs regardless of word width and is especially efficient when the word is very wide. We can also test dynamic RAMs in this manner, although we are still working on adapting suitable algorithms. As we show later, tests for dual-port memories require only minor modifications.

Figure 2 is a block diagram of a static RAM (inner box with dotted outline) and the additional external connections required to implement serial shifting. The address bus is not latched and is applied directly to the X (row) and Y (column) decoders. The input and output data paths are separate. When the Read strobe is high, a word is read and transferred to the memory's output. When the strobe goes back to low, the transparent latches at the output of the sense amplifier maintain the data until the next read operation. When the Write strobe is high, a word is written into memory at the location determined by the address.

We use the multiplexers along the I/O data path to implement the shifting. First, either the normal inputs or the test inputs are selected, depending on the value of the Test-Mode signal. The test input applied to input i of the memory is simply the i–1 output—or, in the case of the least significant bit, a signal controlled directly by the BIST circuit. To illustrate, suppose M_i and M_{i+1} are two logically consecutive bits of the same word. They may or may not be physically adjacent in memory depending on the layout. To move the contents of M_i to M_{i+1}, we first perform a read operation of the word containing these two bits. This brings the bit in M_i to the corresponding output latch. We then perform a write operation at the same location to store the bit in M_{i+1}. This

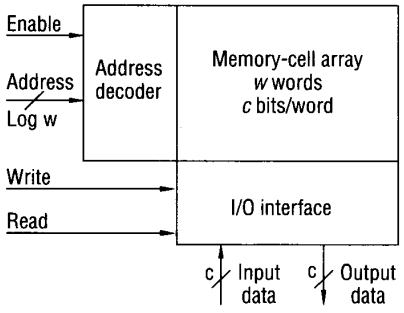

Figure 1. *Input and output signals to control and observe a memory test.*

The serial output of the memory, is the only output available to the BIST circuit.

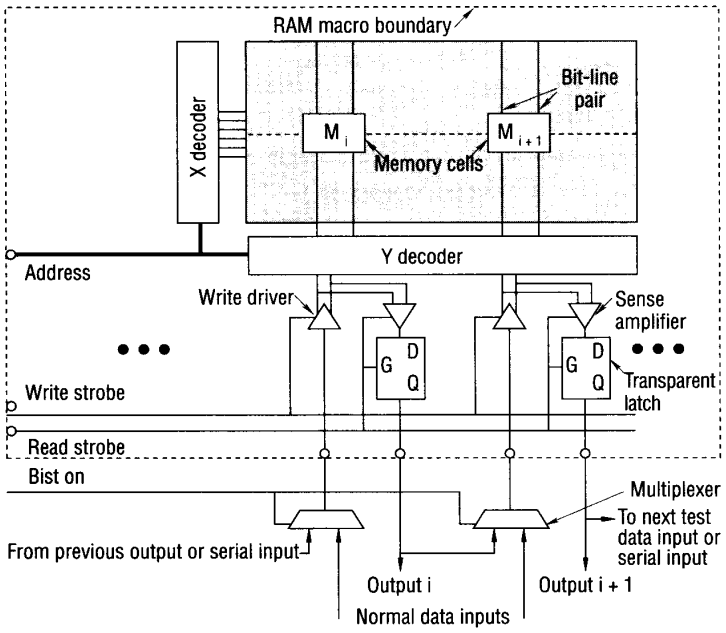

Figure 2. *Memory showing serial-data-path connections in the BIST mode.*

procedure shifts the full word by one position. The memory cell storing the least significant bit of the word has been written with the data provided by the BIST circuit (serial input). The bit that was originally in the cell that stored the most significant bit is now only at the corresponding output latch. The BIST circuit must examine this bit before the next read operation at any location, including the one just read.

This output, called the serial output of the memory, is the only output available to the BIST circuit. Successive read/write, read/write,... operations on a given address serially shift out the contents of the word, replacing it with new data from the serial input. Changing one bit at a time increases the coverage of coupling faults between physically adjacent bits of the same word. Also, a single comparator analyzes the response of the memory under test.

This serial mode of operation is similar to that proposed by You and Hayes[5] except that in our implementation, the RAM requires no modification nor do we make any special assumptions about its operation.

The memory's ability to shift is also useful in testing circuitry connected to the memory. We can link the memory onto a regular scan path or build a linear-feedback shift register with one word of the memory. We can also adapt algorithms in a word- or bit-oriented manner.

Figure 3 shows the shifting capability of the memory. In this example, the contents of an initially clear four-bit word are shown after each memory access. The BIST circuit holds the address lines constant, while applying a succession of four read and write cycles. It stores the data presented at the serial input port (SI) at the beginning of the procedure—1 in this case—successively in all memory cells of that word and finally shifts it out at the serial output port (SO). Only the leftmost bit of a word is directly controlled and only the rightmost bit is directly observed.

To apply external tests to an embedded memory with this scheme, we need only two serial pins to access the data path. The result is consid-

Time	Operation	Serial in	Word contents	Serial out
0		x	0 0 0 0	x
1	R0	x	0 0 0 0	0
2	W1	1	1 0 0 0	0
3	R0	1	1 0 0 0	0
4	W1	1	1 1 0 0	0
5	R0	1	1 1 0 0	0
6	W1	1	1 1 1 0	0
7	R0	1	1 1 1 0	0
8	W1	1	1 1 1 1	0

Figure 3. *Shifting capability of the memory illustrated by the contents of one word during a group of serial shift operations—(ROW1)*[4].

erable savings in routing area and fewer external pins for RAMs with wide words—data ports more than 16 bits wide are not uncommon in telecommunication chips, for example. Since each shift operation has a read and write cycle, the serial technique does not compromise the speed at which we apply new data to the memory. Thus, using test algorithms adapted to the serial interface we can apply high-coverage external tests at full speed.

FAULT MODEL

The fault model we used to evaluate algorithms adapted to the serial interface consists of the basic stuck-at fault model for the memory array, the address decoder, and the read/write circuitry. In addition, we consider two types of coupling faults in the memory cells, C1 and C2. C1 coupling faults are static (or state) coupling faults in which the state of cell i forces a particular state on cell j. C2 coupling faults are dynamic coupling faults in which a read or write operation to cell i forces a particular state on cell j.

We use the static-coupling (or state-coupling) fault model for coupling between bits of the same word, since the cells of the same word are simultaneously accessed. We use the dynamic-coupling fault model for all coupling faults between bits of different words.

The model also takes into account two types of sequential faults. S1 faults are the stuck-open faults observed in static address decoders. S2 faults are inaccessible cells, such as open-access transistors or stuck word lines.

Because the outputs are latched, we had to design the algorithms so that data latches do not mask out sequential S2 faults.[10] Consequently, we have transitions on every data latch at each address. Conventional march-like tests described in the literature[11,12] do not force such transitions and may not detect S2 faults.

The model also considers two additional faults for dual-port memories, which Figure 4 illustrates. Fault type D1 consists of shorts between bit lines belonging to different ports in the same column. Fault type D2 consists of shorts between word lines belonging to different ports in the same row.

ALGORITHMS

We considered four algorithms for implementation using this serial access method: SMarch, SMarchdec, SGalpat, and SWalk. As the names suggest, they are adaptations of well-known algorithms. We consider the first three algorithms to be word-oriented because the entire word is shifted out at each address. The fourth, SWalk, is bit-oriented because a single shift operation is performed for each address change.

We use a special notation is used to describe the algorithms. The address space has w words, and each word contains c bits. A read operation is denoted by R0, R1, or Rx, depending on the expected value at the serial output (X=don't care). For a write operation, the terms W0 or W1 are used. Only the serial input is forced to the value indicated.

Figure 3 shows the notation used for repetitive operations on a particular word. For example, $(R0W1)^c$ means that the operations R0 followed by W1 are repeated c times. However, if we decompose this set of operations into the actual sequence performed on each bit of the word, we get $(R0W1)(R1W1)^{c-1}$ for the first bit, $(R0W0)(R0W1)(R1W1)^{c-2}$ for the second one, and so on. The last one will be $(R0W0)^{c-1}(R0W1)$. Thus, each memory cell has only two nonredundant operations, and these are the ones causing a data transition. We can view this procedure as part of a

The fault model we used consists of the basic stuck-at fault model for the memory array, the address decoder, and the read/write circuitry.

The algorithm covers all the faults described in the model except for the decoder stuck-open faults.

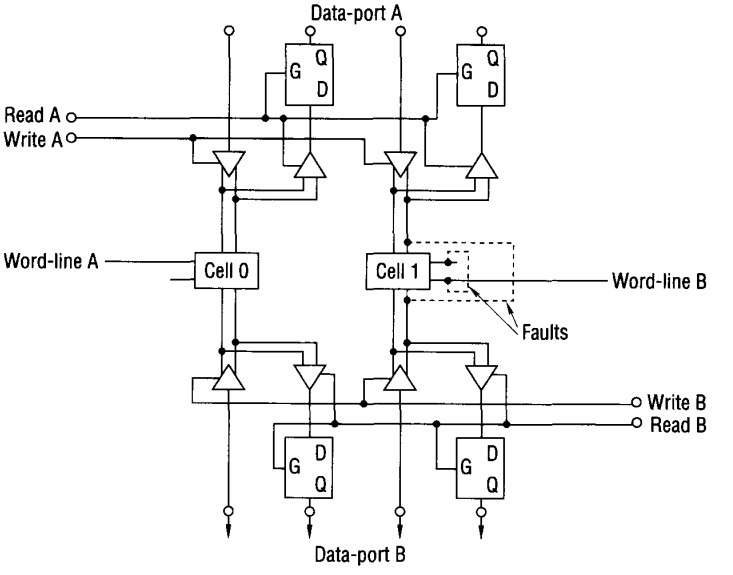

Figure 4. *Simplified schematic of a dual-port RAM showing the additional faults obtained from analyzing the layout.*

simple march test that will pick up coupling between the bits of a word within each word of the RAM. In a byte-wide memory, we do not have to repeat algorithms with different byte-data patterns, since all possible two-cell data combinations are established within and between words.

In some algorithms, we add a subscript to indicate which word is affected by the operation described. For example, $(R0W1)^c_{refadd}$ means that the operation is performed at the reference address refadd.

Step 1. FOR address 1 to w
{count address forward}
$(RxW0)^c$ $(R0W0)^c$
{initialize memory with 0's}

Step 2. FOR address 1 to w
$(R0W1)^c$ $(R1W1)^c$
{read 0's and replace with 1's}

Step 3. FOR address 1 to w
$(R1W0)^c$ $(R0W0)^c$
{read 1's and replace with 0's}

Step 4. FOR address w to 1
{use reverse address sequence}
$(R0W1)^c$ $(R1W1)^c$
{read 0's and replace with 1's}

Step 5. FOR address w to 1
$(R1W0)^c$ $(R0W0)^c$
{read 1's and replace with 0's}

Step 6. FOR address w to 1
$(R0W0)^c$ $(R0W0)^c$
{only first read operation important}
{final RAM state is selectable}

Figure 5. *Pseudocode for the SMarch algorithm.*

SMARCH

Figure 5 is the pseudocode for the SMarch algorithm. It is a march-like test similar to algorithm *C* described by Marinescu,[13] covering all stuck-at faults and the coupling fault types C1 and C2. At each step, we perform two groups of *c* serial operations on each word. The second group of operations is to ensure each word has a transition occurring on all *c* data latches. This transition prevents the fault masking mentioned earlier. Strictly speaking, only *c*+1 operations are required per address, but we used 2*c* operations for symmetry because it allows us to minimize BIST hardware, albeit at the expense of longer test time.

In the first step of the algorithm, the BIST circuit initializes the memory. Unknown values are expected and 0's are injected in each word serially. During the second step, the circuit reads all 0's and replaces them with 1's, where $(R0W1)^c$. Step 3 is similar to step 2, except that the circuit reads 1's and replaces them with 0's in the same forward address order. Steps 4 and 5 are similar to 2 and 3 except that the operations are done by reversing the address sequence. Finally, in step 6, only the first read operation is of interest and any value can be written back in the memory.

The total number of read plus write operations is 24*cw*, that is, 24 times the number of bits in the memory. By performing only the necessary *c*+1 operations per address and by shortening the initialization step, we can cut that number roughly in half. During initialization,

the circuit writes only the first word serially. It writes all the other words in parallel. The finite-state machine, or FSM, is slightly more complex, but this optimization may be necessary for larger memories to reduce test time.

The algorithm covers all the faults described in the model except for the decoder stuck-open faults. The proof for this is similar to that shown by Nicolaidis.[2]

SMARCHDEC

This algorithm extends the SMarch test just described by testing stuck-open faults observed in static address decoders, which are used in some CMOS memories. Since the SMarchdec algorithm adds about 30% to the BIST area overhead, we use it as an extension to SMarch only when the memory requires it.

Figure 6 is the pseudocode for the algorithm. We assume that the memory is filled with 0's after the first six steps of SMarch are executed. The strategy is then to write a foreground pattern of 1's at the first address and write 0's at an address that is at a Hamming distance of one. That is, only one bit of this second address is different from the reference address. The reference address is then read back to see if the foreground pattern is still there. This is repeated for each bit of the address. The reference address is restored to the background pattern, and a new reference address is selected. The number of additional read and write operations performed is $2cw(1+2\log_2 w)$.

```
Step 1. Execute SMarch

Step 2. For refadd = 1 to w
        {RAM assumed to be initially clear}
          (R0W1)^c_refadd
          {write 1's at reference address}
          FOR j = 1 to log2w
          {for each bit of the address}
            newadd = refadd ⊕ 2^(j-1)
            {flip one bit of the address}
            (R0W0)^c_newadd
            {write 0's at neighboring address}
            (R1W1)^c_refadd
            {check contents of reference address}
          END
          (R1W0)^c_refadd
          {reset reference address to 0's}
        END
```

Figure 6. *Pseudocode for the SMarchdec algorithm.*

SGALPAT

The third algorithm we use is a serial implementation of the classic Galpat test. Figure 7 shows the algorithm pseudocode. After an initialization sequence, a reference address is picked and the corresponding word is filled with the foreground value, say, all 1's, one bit at a time. Then, another address is read to see if the word still contains the background value, all 0's, in this case. The reference address is read again. This is done for all addresses. At the end, the reference address is changed to the background value and another reference is picked. The entire step 2 is repeated, interchanging the value of the background and foreground patterns.

The test time of SGalpat is proportional to cw^2, or $O(cw^2)$. However, because this algorithm can use any unidirectional counting sequence, it forms a more compact and versatile BIST circuit. Thus, counters are more easily shared with mission-mode circuitry. On the other hand, the SMarch counters must be able to count up and down, and the SMarchdec counters must implement the Hamming count sequence. Therefore, SGalpat is quite useful for small embedded RAMs. Like the common Galpat test, SGalpat covers all the faults in our fault model as well as many multiple-cell coupling faults, which we do not formally consider in our model. No supplementary test is needed to cover decoder stuck-open faults, since all address transitions are exercised.

```
Step 1. FOR add = 1 to w
          (RxW0)^c_add
          {initialize memory to 0's}

Step 2. FOR refadd = 1 to w
          (R0W1)^c_refadd
          {write 1's at reference address}
          FOR newadd = 1 to w
            IF newadd ≠ refadd
              (R0W0)^c_newadd
              {check if 0's are still there}
            END
            (R1W1)^c_refadd
            {check if 1's are still at reference}
          END
          (R1W0)^c_refadd
          {reset reference address to 0's}
        END

Step 3. Repeat steps 1 and 2, interchanging the 1's
        and 0's in the different operations.
```

Figure 7. *Pseudocode for the SGalpat algorithm.*

SWALK

SWalk is a serial implementation of the walking 1's and 0's test. However, its adaptation to serial format is different from the other tests because it is bit-oriented instead of word-oriented. That is, a single 1 (or 0) propagates through the memory, which behaves as a single circular shift register with as many elements as there are memory cells.

To build this circular shift register, we added a flip-flop between the serial output (SO) and the serial input (SI) as shown in Figure 8. This

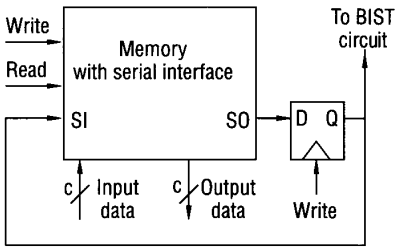

Figure 8. *Connection of SI and SO to implement the SWalk algorithm.*

```
Step 1. FOR address = 1 to w
            (RxW0)^c
            {initialize all words to 0's}

Step 2. FOR address = 1
            (R0W1)
            {set first bit of word to 1}

Step 3. FOR bit = 1 to n-1
        {total number of bits n=wc}
            FOR address = 1 to w-1
            {shift each word by one bit}
              (RxW)
              {the value read at SO is not compared
                but saved in the flip-flop and used
                as the serial input SI for the next
                word}
            END
            FOR address = w
              (R0W)
              {expect 0 on last bit of last address}
            END

Step 4. FOR bit = n
        {total number of bits n=wc}
            FOR address = 1 to w-1
            {shift each word by one bit}
              (RxW)
              {the value read at SO is not compared
              but saved in the flip-flop and used as
              the serial input SI for the next word}
            END
            FOR address = w
              (R1W)
              {the 1 finally comes out}
            END

Step 5. FOR address = 1 to w
            (R0W1)c
            {initialize all words to 1's and verify that
            the 0's are still there in case of backward
            coupling}
        END

Step 6. Repeat steps 2-5, interchanging 0's and 1's
        in all operations.
```

Figure 9. *Pseudocode for the SWalk algorithm.*

additional flip-flop allows the last bit of one word to propagate to the first bit of the following address. The concept is the same as that used by You and Hayes.[5] Similarly, the last bit of the last address is fed back to the first bit of the first address completing the register loop.

As Figure 9 shows, the serial output is ignored until the last bit of the last word is read, which constitutes the final output of the shift register. Thus, the foreground value we inject at one end of the shift register eventually comes out at the other end. After initializing the test and injecting the seed bit, we do not need to generate any other particular value to write to the memory, since the circular register loop is closed. We could detect faults earlier by observing the last bit of every word, but we would have to add considerable BIST hardware to predict the intermediate values.

SWalk covers all the faults in our model except the decoder stuck-open faults. The test length, like that of the SGalpat test, is $O(cw^2)$, but SWalk does not cover as many multiple-cell faults. Therefore, this test cannot be used for memories that are sensitive to decoder stuck-open faults.

The novelty of the SWalk test is the simplicity of its BIST controller and the ease with which a single BIST circuit is shared among multiple memories. Its BIST implementation is the most compact of all the schemes.

BIST ARCHITECTURE / IMPLEMENTATION

Figure 10 shows the general architecture for a BIST circuit that uses the serial technique. It has the following components:

- data-path multiplexers to set up the serial shifting mode
- multiplexers on the address and control lines to switch between mission-mode and test-mode access to the RAM
- a set of counters whose length depends on the RAM's dimensions and the algorithm being used; in all cases, we need a counter that has the same number of states as memory addresses (address counter) and another with the same number of states as the bits in a single word (position counter)
- a finite-state machine, or FSM, that embodies the actual test algorithm and controls the counters and generates the serial data stream and expected data for comparison with the RAM output
- a memory control timing generator

The circuit is ordered hierarchically. Blocks that could be useful in mission mode, such as counters, are separate. The FSM, which determines the algorithm to be used, is also in a block of its own. To more easily and quickly prototype the algorithms, we constructed the first generation of BIST FSM circuits using the Synful FSM synthesizer, which was developed at BNR. We also wrote a netlist generator for the counters, multiplexers, and timing circuitry, which adapts to the single- and dual-port RAM module generators available.

The generated netlists for a particular embedded RAM BIST circuit are automatically routed in a standard-cell or gate-array technology that is compatible with the target memory. Future versions will use custom-optimized layouts that can be merged with the RAM module generators to reduce the area of the overall circuit. All the BIST circuitry is synchronous and scan testable.

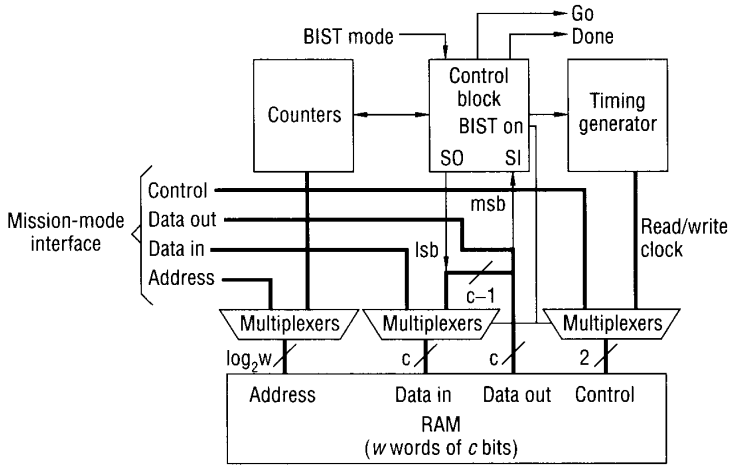

Figure 10. *Architecture of a BIST circuit.*

The most commonly used schemes for sharing the BIST circuit are daisy chaining and test multiplexing.

SHARING THE BIST CIRCUIT

As mentioned earlier, one of the most attractive features of the serial BIST scheme is the ease with which several memories can share the test circuit. It is common to see several small RAM blocks embedded on custom telecommunication chips, for example. Unless the blocks can share the BIST circuit in these cases, the silicon overhead associated with testing the RAMs will be unacceptably large. Many factors determine how easily a BIST circuit can be shared. The key questions we ask are

- How much interconnection is required for the BIST circuit and the RAM blocks, since they may not be located near one another on the chip's floorplan?
- How similar must the memories be to share the controller?
- How much additional circuitry is required?
- Are the connections simple enough that ASIC designers can assemble the network from simple guidelines?

The serial data path inherently reduces the interconnection complexity, particularly when the RAMs have very wide words. For all the algorithms, we apply the address and control signals simultaneously to all memories to be tested. These lines are then common to all RAM blocks, again simplifying the interconnection. We can adapt all the algorithms to test several memories each with a different number of words and bits per word with only a few added gates to reconfigure counters.

The most commonly used schemes for sharing the BIST circuit are daisy chaining and test multiplexing. Daisy chaining is simply connecting the serial output of one memory to the serial input of another so that they appear as one larger block, as Figure 11 shows. As we just mentioned, we apply the address and control signals simultaneously to all RAMs. For the SMarch, SMarchdec, and SGalpat algorithms, all memories must have the same number of words, although the number of bits per word is not restricted. We need to set only the position counter to the word-width sum in this case.

For SWalk, no restrictions apply to the memory dimensions with daisy chaining. The address-counter length is equal to the number of words of the largest memory. Each memory receives the same address, and a

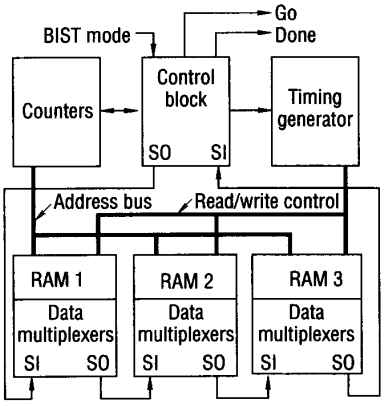

Figure 11. *Daisy chain connection for sharing a BIST circuit among three memories.*

small circuit associated with each memory checks to make sure the address is within its range. If not, the memory is not accessed, and its serial input is passed to the next memory along the chain. We must set the position counter equal to the sum of the number of bits of all the memories. This modification is the only one we need to make to the BIST circuit, making SWalk the simplest BIST algorithm to use for a cluster of small dissimilar RAMs.

Figure 12 shows the second method of sharing the BIST circuit, which is test multiplexing. In this method, the controller tests one memory at a time. Any of the algorithms can use this technique without restriction on memory dimensions. We can apply the address, control, and serial data outputs from the BIST circuit simultaneously to all memories. That is, no multiplexing is required on these lines, although the contents of the idle memories will be disturbed if we do not explicitly disable them from selection. A small additional control circuit identifies which RAM is to be tested. We use the output of this circuit to select which serial line returning from the memories we should apply to the BIST error-detection circuit. It is also used to reconfigure the address and position counters to the appropriate length. These counters need be only as long as the greatest memory dimension we must accommodate. Generally speaking, this method is slightly more complex to implement, but, depending on the individual memory dimensions, we often get shorter test times and use less chip area.

In addition to daisy chaining and test multiplexing, we use two methods of sharing BIST hardware among several memories through the serial interface. One scheme, called address windowing, allows us to treat a group of memories that have an identical number of bits per word as a single block. Another scheme, which is suitable for testing memories with very wide words or identical memories in parallel, is used when test times must be as short as possible. With this method, a given word is serialized into more than one bit stream, introducing a degree of parallel processing into the test.

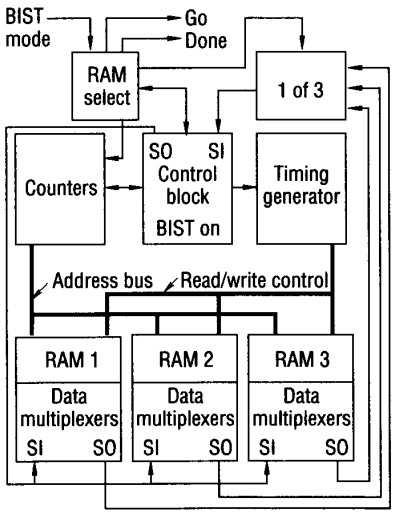

Figure 12. *Parallel connections for sharing a BIST circuit among three memories using test multiplexing.*

DUAL-PORT MEMORIES

For dual-port memories the tests are performed twice, once on each port. These tests also cover shorts between bit lines and shorts between word lines from opposite ports (fault types D1 and D2 mentioned earlier). We do not have to modify the algorithms, although a the BIST circuit must perform a special write operation on the port opposite the one under test.

When we test a dual-port memory externally, we test these shorts by initializing a memory cell to a value (0 or 1) and reading back this memory cell from one port while writing a cell in the same column with the opposite value, from the second port. If a short exists, the read access is perturbed and returns incorrect data. In a BIST environment, it is harder to provide two addresses, one for each port, and make sure that the corresponding memory cells are in the same column. Instead, we provide the same address to both ports and use what we call a "shadow" write operation.

A shadow write is an attempt to write a memory cell with the row selector disabled. The powerful write drivers then drive only the bit lines. If the memory has no faults, the memory cell will not be modified. If, however, we perform a normal read operation of the same cell from the port under test and a short (of the type just described) does exist, the result of the read access is affected.

The data needed to perform the shadow write is easy to obtain, assuming we are allowed simultaneous read operations of the same memory cell from both ports. For example, with the SMarch algorithm, we do the simultaneous read access while the second group of c serial operations is being executed on each word, say $(R1W1)^c$, for the port under test. We can then load the output latches of both ports with 1's and use this data to perform the shadow write at the next address, since the first group of c serial operations performed is $(R0W1)^c$.

CHOICE OF ALGORITHM

The choice of algorithm depends on the following factors:

- fault coverage required
- area available
- test or simulation time restrictions
- number of memories to be tested
- counters to be shared with mission mode

Table 1 lists the various attributes of the four algorithms. We estimate the coverage of unmodeled faults on the basis of our test chips and by partially analyzing multiple faults. The nominal area overhead (without BIST sharing) is about 30% larger for the SMarchdec test than for the SMarch test for a typical memory of 4 Kbits. However, the overhead is considerably lower when the BIST circuit is shared among several blocks and the counters are reused in mission mode. Sharing BIST hardware was a key objective of this work, and most memory-intensive chips recently designed at BNR have exploited this capability.

Generally speaking, we see the SMarch algorithm as the most widely applicable. However, when the test time is not prohibitive, SGalpat is better, because it covers more multiple faults, although we did not quantify these in the model.

IMPLEMENTATION

Figure 13 illustrates the area overhead for the BIST circuit implementing SMarch. The areas are for an automatically routed standard-cell implementation of BIST, which includes all the circuitry we listed earlier. The upper curve is for a BIST circuit dedicated to one memory. We save

Our technique is particularly powerful for testing telecommunications ASICs, which commonly have a variety of small RAMs.

Table 1. *A comparison of serial test algorithms.*

	SMarch	**SMarchdec**	**SGalpat**	**SWalk**
Test/simulation time	O(cw)	O(cw)	$O(cw^2)$	$O(cw^2)$
Decoder stuck-open fault coverage	No	Yes	Yes	No
Unmodeled fault coverage	Low	Low	High	Medium
Area overhead (no BIST sharing)	Low	High	Medium	Low
Ease of sharing BIST among RAMs	High	High	High	High
Ease of reuse for BIST counters	Medium	Low	Medium	High

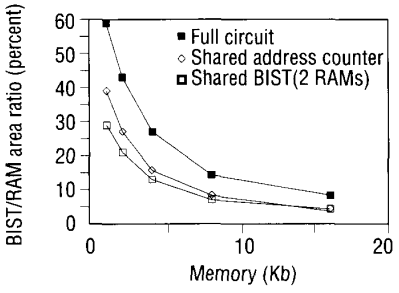

Figure 13. *Overhead with a BIST circuit based on the SMarch algorithm.*

significant BIST area if we can reuse the address and position counters in mission mode, as the middle curve (black diamonds) shows. The bottom curve (black squares) shows the improvement in efficiency if two identical RAMs share the BIST circuit. This curve is valid if the memories are nearby or if they share an address bus. Otherwise, we must take into consideration an extra routing factor. For all curves, the BIST area overhead decreases rapidly with the size of the memory.

We verified the effectiveness of the test algorithms and our BIST methodology using a test chip. We have since applied the scheme to several product chips implemented in BNR's 1.5- and 1.2-micron CMOS processes. Table 2 shows three specific examples, which used the SMarchdec algorithm to test memories with fully static decoders. We chose the SMarch algorithm because decoder stuck-open fault coverage was required and the test time had to be minimized. The BIST counters were not needed in mission mode so the entire BIST area is considered in the fifth column. Chip 3 used the daisy chaining method of sharing the BIST circuit.

The serial interfacing technique for testing embedded RAMs is applicable to external as well as built-in test. The amount of interconnection compared with that for parallel test schemes reduces the complexity and silicon area dedicated to testing as well as the number of pins required for external access. Our technique is particularly powerful for testing telecommunications ASICs, which commonly have a variety of small RAMs with different sizes and port configurations, some with very wide words. Since BIST circuitry is easily shared among several embedded RAMs and in many cases reusable in mission mode, this method enables on-chip testing with low area overhead compared with other schemes. We have adapted several algorithms to the serial format. The selection of the algorithm within a standard BIST architecture allows the designer to make trade-offs among fault-coverage, test time, reusability of the BIST circuitry, and area overhead. Even though the memory is tested serially, a unique test pattern is applied every clock cycle since the memory itself is used to shift the test data. In our implementations of the BIST circuitry on several product chips, we found the area overheads acceptable and the method quite suitable for testing chips with embedded RAMs. As a result, we use the method regularly within BNR. ❖

Table 2. *Implementation examples; S = single-port static RAM, D = dual-port static RAM, test-clock rate = 2 MHz.*

Chip	RAM Configuration	Total Bits	BIST Circuits	Ratio of BIST to Chip Area	Test Time
1	1 × 64 × 32 (S)	2,048	1	1.8%	25 ms
2	1 × 512 × 10 (D) 1 × 512 × 5 (S)	7,680	2	3.4%	200 ms
3	2 × 512 × 8 (S) 2 × 512 × 16 (D) 2 × 64 × 8 (D)	25,600	2	4.3%	400 ms

ACKNOWLEDGMENTS

We thank Hojjat Salemi for his valuable help during the layout of the first test chips and Larry Sasaki for pertinent implementation hints.

Part of this work appeared previously in "A Serial Interfacing Technique for Built-In and External Testing of Embedded Memories," *Proceedings of the Custom Integrated Circuits Conference*, 1989, pp. 22.2.1-22.2.5.

REFERENCES

1. Z. Sun and L. Wang, "Self-Testing of Embedded RAMs," *Proc. Int'l Test Conf.*, Oct. 1984, pp. 148-156.
2. M. Nicolaidis, "An Efficient Built-In Self-Test Scheme for Functional Test of Embedded RAMs," *Proc. Int'l Test Conf.*, Nov. 1985, pp. 118-123.
3. P. Bardell, W. McAnney, "Self-test of Random Access Memories," *Proc. Int'l Test Conf.*, Nov. 1985, pp. 352-355.
4. T. Sridhar, "A New Parallel Test Approach for Large Memories," *IEEE Design & Test of Computers*, Aug. 1986, pp. 15-22.
5. Y. You and J. Hayes, "A Self-Testing Dynamic RAM Chip," *Proc. MIT Conf. Advanced Research in VLSI*, Jan. 1984, pp. 159-168.
6. S. Jain and C. Stroud, "Built-In Self Testing of Embedded Memories," *IEEE Design & Test of Computers*, Oct. 1986, pp. 27-37.
7. S. Han and M. Malek, "Two-Dimensional Multiple-Access Testing Technique for Random Access Memories," *Proc. Int'l Test Conf.*, Nov. 1986, pp. 248-251.
8. K. Saluja, S. Eng, and K. Kinoshita, "Built-In Self-Testing RAM: A Practical Alternative," *IEEE Design & Test of Computers*, Feb. 1987, pp. 42-51.
9. J. Smith, "Measure of the Effectiveness of Fault Signature Analysis," *IEEE Trans. Computers*, Vol. C-29, No. 6, June 1980, pp. 510-514.
10. R. Dekker, *Fault Modeling and Test Algorithm Development for Static Random Access Memories*, master's thesis, Dept. of EE, 1-68340-28(1987)25, Delft Univ. of Technology, The Netherlands, Sept. 1987.
11. R. Nair, S. Thatte, and J. Abraham, "Efficient Algorithms for Testing Semiconductor Random Access Memories," *IEEE Trans. Computers*, Vol. C-27, No. 6, June 1978, pp 572-576.
12. D. Suk and S. Reddy, "A March Test for Functional Faults in Semiconductor Random Access Memories," *IEEE Trans. Computers*, Vol. C-30, No. 12, Dec. 1981, pp. 982-985.
13. M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," *Proc. Int'l Test Conf.*, Nov. 1982, pp. 236-239.

Direct questions or comments on this article to B. Nadeau-Dostie, BNR, PO Box 3511, Stn. C, Ottawa, Ontario K1Y 4H7

Benoit Nadeau-Dostie is a member of the scientific staff at BNR, where he is developing new design-for-testability techniques for communication systems. Previously, he taught digital design and microelectronics at the Université Laval in Quebec. He holds a PhD from the Université Sherbrooke for work on microelectronic devices used in biomedical applications. He is a member of the IEEE and the Ordre des Ingénieurs du Quebec.

Allan Silburt is manager of the memory development group at BNR. Previously, he was with BNR's design-for-testability group, where he worked on fault modeling and BIST circuits for embedded memories. He also worked for Mosaid, Inc., in Ontario, where he developed device-modeling and circuit-simulation tools for use in MOS memory design. Silburt holds a BASc in electrical engineering from the University of Waterloo and an MEng in electronics from Carleton University.