Programmable Memory BIST

Slimane Boutobza   Michael Nicolaidis** Kheiredine M. Lamara**   Andrea Costa**

**Synopsys Corp, 12 rue Lavoisier 38330 Montbonnot, France*
***iRoC Technologies, WTC, BP 1510, 38025 GRENOBLE, France*

Abstract: *In modern SoCs embedded memories include the large majority of defects. In addition defect types are becoming more complex and diverse and may escape detection during fabrication test. As a matter of fact memories have to be tested by test algorithms achieving very high fault coverage. Fixing the test algorithm during the design phase may not be compatible with this goal, as thorough screening inspection or customer returns may discover after fabrication unexpected fault types. A programmable BIST approach allowing selecting after fabrication a vast variety of memory tests is therefore desirable, but may lead to unacceptable area cost. In this work we present a programmable memory BIST architecture offering such flexibility at an area cost similar to traditional memory BIST schemes.*

1. Introduction

In modern SoCs, embedded memories occupy the largest part of the chip area and include an even larger amount of active devices. As memories are designed very tightly to the limits of the technology they are more prone to failures than logic. Thus, they concentrate the large majority of defects. In addition defect types are becoming more complex and diverse and may escape detection during fabrication test. The above two trends increase defect level and affect circuit quality dramatically. To cope with, memory test algorithms should evolve to cover the fault models corresponding to the target fabrication process and memory design. These fault models may not be known completely during the design of a product, so the memory test approach must be very flexible to allow selecting the memory test algorithm in silicon. This flexibility is particularly suitable for debugging a new fabrication process or memory design, as well as for testing unexpected failures discovered after design and fabrication. This flexibility has to be integrated in memory BIST, which is the mainstream test technology for embedded memories. Hence, a programmable memory BIST is required, allowing selecting memory test algorithms able to cover a large variety of faults.
This flexibility has to be achieved at low area cost, to make the approach attractive for real products.

Existing programmable BIST solutions lie in 3 main categories: micro-code based BISTs, micro-programmable based BISTs and FSM based BISTs.
Based on storage element (i.e.: RAM) to store the test algorithm, the micro-code based BIST is mainly the most used approach. One of the first examples of this method was proposed in [1]. In this method only two operations per March Element (sequence) can be performed efficiently. This was a serious drawback since almost all coupling faults as well as linked faults require more than this number of operations in order to be detected. Other approaches with greater flexibility were proposed in [2] for process monitoring and diagnosis, and in [3] optimized for symmetric algorithms with rather a reduced area overhead. But still the use of consequent storage elements set the problem of their area overhead and auto-test cost.

In the micro-programmable based BISTs the test algorithm is generated using a microprocessor. [4] gives an example of this approach. While having the most flexibility (algorithmic approach), it suffers however from a high area overhead. Furthermore, it is not suitable when the BIST hardware is intended to be an integral part of the embedded RAM to form an intellectual propriety (IP) core.

Attempts to design FSM based programmable BIST controllers have resulted in rather ineffective implementations. In [3] the test algorithm is generated in door through a combination of set of March test components, produced by an internal FSM. In addition, a 2-dimentional circular buffer is used to store test algorithm instructions. This mechanism of storage+decoding+generation is area consuming, and may not achieve a high clock frequency.
The approach described in [5] generates the test algorithm outdoor through a scan mechanism. Hence it is time consuming (serial BIST), but also the area overhead in term of scan-registers length is not optimal since it doesn't exploit the characteristics of memory test algorithms.

Hence we observe that existing programmable BIST solutions suffer at least, from one of the following weakness:

- Area overhead. In a context of built-in test this is a serious drawback.
- Low testability of the test circuitry, especially of the storage element.
- Flexibility in term of supported test algorithms achieved is not always high.

In this work, we present a new programmable memory BIST approach that eliminates the need of an expensive storage unit by an appropriate decomposition of the test algorithm.
The proposed programmable BIST solution ([6], [7] and [8]) allows testing memories for a vast variety of fault models, while requiring an area cost similar to traditional memory BIST schemes.

In the remaining sections we develop the principle of programmable BIST in section 2, in section 3 we explain the programmable BIST for March test algorithms, section 4 and 5 describe the implementation of the proposed architecture, section 6 focuses on the data background programmability and its combination with test algorithm programmability to achieve a highest fault coverage and we finish, in section 7, by some experimental results and comparison with other Programmable memory BIST approaches.

2. Test algorithms for highly flexible and low cost programmable memory BIST

Implementing memory BIST allowing test algorithms programming that target a large variety of memory fault models, may lead to unacceptable area cost. The selection of test algorithm families able to test the target fault models may have a considerable impact on this cost. Thus, this selection must be done cautiously.
The most widely fault model families encountered in memories comprise stuck-at faults, transition faults, coupling faults, dynamic faults, passive pattern sensitive faults, active pattern sensitive faults (some times also called dynamic pattern sensitive faults), static pattern sensitive faults, and retention faults [9].
An important class of memory test algorithms is March algorithms. These algorithms are very regular. For instance, the March test algorithm (described by using the notation adopted in [9]):

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

consists in three March sequences. The first sequence initializes all the memory addresses to 0. The addressing order is meaningless in this sequence. The second sequence visits all memory addresses using a certain address order (up order), and performs to each address a read 0, a write 1, and a read 1. The third sequence visits all memory addresses using the reverse address order (down order), and performs to each address a read 1, a write 0, and a read 0. If exploited, this regularity may allow low cost programmable BIST architecture. In addition such tests can be used to detect a large number of predominant memory fault models, including memory cell faults such as stuck-at-faults, transition faults, and coupling faults [10] [9]. Also, faults affecting the read/write circuitry and most of decoder faults are also tested by these test algorithms since they can be modeled as cell array faults [11]. Thus, March test algorithms are good candidates for a programmable BIST approach.

Testing dynamic faults may require repeating many times the same operation over the same memory address [12]. Due to this repetition, some authors do not classify test algorithms for dynamic faults to the family of March test algorithms. However, their structures are similar as for March test algorithms; see for instance [12]. Thus, they can be implemented by the same programmable BIST architecture as March test algorithms.

We observe that despite their regularity, March tests detect a large variety of faults. This is not unexpected since each fault in a fault model may affect any of the memory cells. Thus, it is natural to repeat the same operations to each cell, since we need to test the same faults at all the cells. However, this regularity has some limitations because within a March sequence a given write operation writes the same data to all the memory cells and creates a limited set of data patterns in the cell array. As a consequence, neighborhood pattern sensitive faults cannot be covered as they require more complex data patterns in a set of neighborhood cells. These faults include static, dynamic, active and passive pattern sensitive faults. Many of the test algorithms proposed to test such faults are quite complex and the adoption of such algorithm families in a programmable BIST approach may lead on significant area cost. Instead of implementing such families, we adopt an approach that maintains the structure of March test algorithms but combines it with test data that create any possible background in any cell neighborhood of a given size. Let us consider the case of passive neighborhood pattern sensitive faults (PNPSF), under such a fault, the basic cell of a neighborhood cannot perform a transition ($0 \rightarrow 1$, or $1 \rightarrow 0$), when the set of neighboring cells are in a given state. Since, any of the possible states of the neighboring cells can be on the origin of the fault; we need to test the two transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ of the basic cell for all the possible states of the neighboring cells. We can detect such faults by loading each of the required backgrounds in the memory and execute for each background a March algorithm that detects the transition faults.

Another important family of faults is retention faults. For detecting if the memory retains its state after a certain amount of time we need to introduce in the test algorithm a wait state of a given duration. This state can be obtained by programming a “March” test algorithm of a certain length in which all operations are non-operations.

Following to this discussion, the kernel of our BIST architecture will implement a programmable BIST for March test algorithms. As a matter of fact we need to develop a low cost programmable BIST for March test algorithms. We also need a low cost scheme for supplying all possible data patterns to all cell neighborhoods of a given size. The resulting BIST architecture will allow us to program test algorithms for covering stuck, transition, coupling, pattern sensitive, dynamic and retention faults. In addition to these fault classes some faults may require to use some particular address orders. As a matter of fact, the programmable BIST architecture is expanded to allow selecting several address modes. A low cost implementation of this architecture is presented in the following sections.

3. Programmable BIST for March Test Algorithms

The present work describes a new programmable BIST architecture, which uses a single instruction per March sequence. The size of each instruction is small. In addition, the instructions describing a March test algorithm are loaded to the Instruction Register one at a time by means of a scan path. This eliminates the need for using a specific memory for storing the micro-program, and allows performing March test algorithms containing any number of March sequences. This scheme requires a hardware cost similar to traditional memory BIST, making it attractive for any design.
To illustrate this scheme, let us consider the following March test algorithm:

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

Memory BIST schemes for March test algorithms follow two basic architectures, the serial architecture introduced in [13] and [14] and the parallel BIST architecture introduced in [15]. In this work, we selected the parallel BIST architecture since the serial one imposes some restrictions to the memory test algorithm.
Following to the architecture presented in [15], we can decompose the March test algorithms into several levels of hierarchy. In the highest level we have the whole March test algorithm. In the next level we have the March sequences composing it. In the next level we have the addressing order used within the March sequence. Then, we have the number of operations performed at each cell within the March sequence. Finally, at the lowest level, we have the type of each individual operation and the data used by the operation. Our programmable BIST uses this decomposition. This representation leads to efficient hardware architecture by associating a simple hardware block to each level of hierarchy. Following this scheme, the programmable BIST hardware for March test algorithms comprises the following blocks:

- a register which holds at any given time the instruction that determines the March sequence under execution (Instruction Register);
- an address generator that is able to generate all the memory addresses in increasing (up) and decreasing (down) order. This generator can be an up/down binary counter, an up/down LFSR, or any other counter able to generate all the addresses in increasing and decreasing order. A binary counter will be preferred for memories with incomplete address space;
- a comparator or a signature analyzer for test response verification;
- a controller that sequences the operation of the BIST hardware;
- a data register (to be used only in word-oriented memories) in which we load the data word to be used during a March sequence.

The instruction determining the current March sequence is stored in the Instruction Register (see figure 1) It comprises the following fields:

- An up/down field (U/D). This is a single-bit field. It determines the address order to be used during the March sequence,
- A March sequence size field (NO). It determines the number of operations performed during the current March sequence. This field has $\lceil \log_2 m \rceil$ bits, where m is the maximum number of operations of a March sequence allowed by a given implementation of our programmable BIST. In our convention, the binary value 0 of this field will determine that we will execute a single operation in the March sequence (the operation O0). The decimal value t of this field determines that we will execute t+1 operations during the March sequence,
- m operation fields (O0, O1, ... Om-1). They determine the type of each individual operation of the March sequence. Each operation field will be a single-bit field if we want to implement only March sequences using two types of operations (write and read). It will have more than one bit if we want to implement March sequences using more than two operations (e.g. read, write, read-modify-write, no operation, read don't care…),
- m polarity fields (P0, P1 ... Pm-1). Each polarity field is a single-bit field. It determines the polarity of the data to be written in the memory during a write, and the polarity of the data expected to be read from the memory during a read. The polarity for the read data is used if the test response evaluation is done by a comparator. In bit-oriented memories, the polarity bit is equal to the value of the data bit. In word-oriented memories, a separate field holds the data word used during the current March sequence. The polarity bit determines at each cycle of the current March test algorithm if this word data is used in its direct or complemented form. Only the contents of the first t+1 pairs of operation and polarity fields are meaningful during

the current March sequence, where t is the binary value specified in the field NO. The values of the fields Ot+1...Om, Pt+1...Pm, are don't care, since only t+1 operations are performed during the current March sequence.

- A data word field (Data). This field is used for word oriented memories only. It determines the value of the word to be used (in its direct or complementary form) during the March sequence.
- A wait field (W). This is a single-bit field. It determines if the operations (e.g. read, write) specified in the operation fields have to be executed during the present March sequence, or if instead, this operations will be replaced by non-operation (or idle) cycles. This field is optional and is used if the programmable BIST has to offer the option of retention fault testing. This field is used for implementing a March sequence composed only of wait cycles. However, for March test sequences where only some cycles are wait cycles, each wait cycle is considered as an operation, the non-operation, and is coded within the respective operation field.
- An address mode field (@mode). It is used to specify the mode of the address Counter, if the address Counter has some other modes in addition to the up and down modes (for instance fast-X, fast-Y, fast Z…).
- A test end field (TE). A single-bit field that determines if the current March sequence is the last sequence of the test algorithm. So, at the end of a March sequence having its TE field active, the BIST controller activates the March test algorithm completion signal. The TE field is optional, because our BIST hardware asserts the LM signal, at the completion of each March sequence. Since the global test controller of the chip knows that the BIST executes its last March sequence, it could use this March sequence completion signal, asserted during the last March sequence, as the March test completion signal.

4. Programmable BIST Implementation

We can describe the control part of the BIST circuitry as an FSM and synthesize it by using a generic synthesis tool. However, a BIST architecture is supposed to be used in a large number of designs. So, it is interesting to define a custom architecture that saves costs with respect to a synthesized circuit. Figure 1 shows the detailed implementation of the programmable BIST circuitry using such custom control part. For sake of clarity, we assigned for each block in this figure a specific ID (number) that will be used as a reference later on. Figure 1 depicts the following modules:

Instruction Register: As said earlier, this register (1) holds the current instruction that determines the current March sequence and includes the fields specified earlier.

Cycle Controller: This circuit (30) comprises the control MUX (32), the cycle counter (42), a comparator (43), a flip-flop (44) and an OR gate (45) connected as explained below. The control MUX has m sets of inputs coming from the m operation fields of the Instruction Register (1), m sets of inputs coming from the m polarity fields of the Instruction Register, one set of outputs O and a second set of outputs P. During t+1 consecutive cycles, the successive values of the first t+1 operation fields (O1, O2,…, Ot+1) of the Instruction Register are provided at the output O of the control MUX, and the successive values of the first t+1 polarity fields of the Instruction Register (P1, P2,…, Pt+1) are provided at an output P of the control MUX, where t is the binary value stored in the NO field of the Instruction Register. The Cycle Controller repeats the above operation for each address generated by the address counter (73).

The comparator (43) compares the value stored in the field NO of the Instruction Register against the value present on the outputs of the control counter. When the comparison matches the output of the comparator is activated high. This signal indicates the completion of the test operations performed over the current address. The D flip-flop (44) delays the output of comparator 43. The output of this flip-flop initializes the control counter to the 0 binary state, which is now ready to resume operations for the next memory address. A second signal TS can be combined with the output of flip-flop 44, by means of a logic gate 45, to generate the signal that initializes the binary counter 42. The cycle counter is also initialized at the beginning of the test session.

Address Counter: During each clock cycle of the test process, the Address Counter provides on its outputs the address of the current memory operation. The size of the Address Counter is equal to the number n of the address bits of the memory. The Address Counter can be implemented by any sequential circuit able to generate on a set of n signals all the possible 2^n binary values. The Address Counter generates these values at a first order (up order), or at a second order (down order) which is the reverse of the first order. We can use for instance an up-down binary counter, an up-down LFSR, an up-down Gray code generator, etc. However, when the memory is incomplete, that is, it uses only a subset of the possible 2^n address values to select all the existing memory locations, a binary counter is more convenient since it will address at a first time the existing memory locations. Thus, a circuit decoding the maximum address of the actual memory location can be used to stop the current March sequence when all the actual memory locations have been tested (addressed).

Data Register: The contents of the data register (53) come from the Data field of the Instruction Register. When the current value of the polarity bit P provided by the control MUX is 1, the Data field is inverted by a set of XOR gates (52). During each write operation of the test process, the Data Register provides on its outputs the data to be written in the memory. During each read operation of the test process, this register provides the data value expected to be read from the memory. This data value is compared through a comparator (100) against the actual data value read from the memory (200) to verify its correctness. Another possibility is to verify the read data by using a signature analyzer. In this case the values present on the outputs of the Data Register during the read operations are useless. Very often the data words used in memory testing have a periodic structure. In this case the Data Register can have a number k of bits, which is lower than the number w of bits of the memory words. The k bits of the Data Register will be expanded into w bits as shown in figure 1.

Operation Control Register: This register (63) provides during each cycle of the March sequence the control signals that determine the operation to be executed by the memory (e.g. a read, a write). These signals come from the outputs O of the control MUX (32). The binary values used to code the memory operation in the operation fields (O1, O2 … Om+1) of the Instruction Register can use the same coding as the control signals of the memory. For instance, for a memory having an R/W and a "Memory Enable" (ME) control signal, R/W=1, ME = 1 may code a read cycle, R/W=0, ME = 1 may code a write cycle, and R/W=x, ME = 0 may code a cycle with no memory access. In this case, we can use two bits per operation field of the instruction register and use the same coding as the control signal of the memory, for the read operation, the write operation and the non operation. It is also possible to use another coding. For instance, it is possible to code only the read and write operations in the operation fields of the Instruction Register. Thus, we will use a single bit per operation field. In this case the operation code O selected by the control MUX (32) must be modified to generate the actual coding of the memory control signals. A Formatting block 62 is used in this case to perform this modification. Another problem concerns the timing characteristics of the address, the data, and the control signals of the memory. It may be necessary to modify the BIST hardware in order to conform to the timing constraints of each memory design. These aspects are not considered here. However, the hardware implementation presented in figure 1 provides to the memory under test, address, data, and control signals that are ready from the beginning of each clock cycle. Thus, they should conform to a majority of memory designs (synchronous and asynchronous as well).

Control signals of the Address Counter:
A control signal U/D determines the order in which the Address Counter generates the memory addresses. This signal comes from the U/D field of the Instruction Register.
At the beginning of the test session (TS =1), the Address Counter is initialized either to the first address of the down order or to the first address of the up order by means of signals S and R. These signals are generated by a logic (75) that combines the signal TS with the signal U/D coming from the input of the Instruction Register.

Hold signal of the Address Counter: For each address visited by the current March sequence, the contents of the Address Counter are held unchanged until all operations are executed over this address. For doing so, the Address Counter is controlled by a Hold signal. To generate this signal we use the output of the flip-flop (44) of the Cycle Controller. As we have seen earlier, this output is active during all but the last operations executed over a memory address. The state of the Address Counter must also be held unchanged when we finish a March sequence and we start a new one, and these sequences use different address order. For doing so, the value stored in the U/D field of the Instruction Register is loaded to a flip-flop (76) at each clock cycle. An EXCLUSIVE OR gate (77) receives the input and the output of this flip-flop. The output value 1 of this gate will activate the Hold signal of the Address Counter. As a matter of fact, the hold signal of the address counter is generated by an OR gate, which combines the output of XOR gate (77) and the output of the flip-flop (44).

Hold signal of the Instruction Register: This signal holds unchanged the contents of the Instruction Register until the end of the current March sequence. To generate this signal we use a decode logic (74), which decodes the last address of the up sequence of the Address Counter and the last address of the down sequence of the Address Counter and activates respectively signals LU and LD when these addresses occur. A logic circuit (82) combines these signals with the U/D field of the Instruction Register, the signal TS, which activates the test phase, and the output of a comparator (43) to generate the hold signal of the Instruction Register. Thanks to this signal, the Instruction Register is loaded at the beginning of the test session (TS active) and at the end of each March sequence.

Programmable Retention Sequences: To program a retention test of programmable duration, the Instruction Register 1 can be extended to include a wait field W. When the level of this field is inactive (e.g. value 0), the BIST provides on the control signals of the memory the values coming from the output O of the control MUX. When the value of the wait field W is active the BIST provides on the control signals of the memory test a value that corresponds to a non operation cycle of the memory. For instance, if the memory uses a "Memory Enable" signal, the active level of the wait field W disables this signal and forces the memory in a non operation (or idle) cycle. The wait signal can be used to program wait phases of any given duration for detecting retention faults. This can be done by loading the Instruction Register several times with well selected values of the NO field and the W field.

Other Address Modes: In the above description, the address counter implements a given address sequence generated in one order (up) and in the reverse order (down), but some defects may require several types of address sequences (e.g. fast-X, fast-Y, fast-Z…). To allow such options, the Address Counter is implemented to have different operation modes, corresponding to such address sequences. In addition, the Instruction Register is expanded to include an address filed (denoted as @mode in figure 1). This field controls the Address Counter to select the type of the address sequence generated during the current March sequence.

5. Instruction Register Load Modes

To start the March test algorithm we load in the Instruction Register the instruction corresponding to the first March sequence of the March test algorithm. This is done by activating the signal TS during one clock cycle. Then, at the end of the first March sequence and of the consecutive March sequences, a new instruction is loaded automatically, which codes the next March sequence. In the BIST hardware described previously, these loads are activated by the BIST hardware itself, by means of the hold signal of the Instruction Register. Another option is to control the load of the Instruction Register by using the test controller of the SOC.

We use the following two options for applying instructions on the Instruction register.

Shift Mode: An interesting characteristic of the programmable BIST proposed here is that it uses only one instruction per March sequence. Thus, we can use a shift mode (e.g. a scan path (101)) to shift the first instruction until the inputs of the Instruction Register, and activate the TS signal to load it in the Instruction Register. Then, during the execution of the first March sequence, we can use the shift mode to shift in the second instruction until the inputs of the Instruction Register 1. This instruction is maintained at the inputs of the Instruction Register until the end of the second March sequence, where it is loaded in the Instruction Register. When the second instruction is loaded in the Instruction Register the third instructions starts to be shifted through the scan path, and so on until the end of the test algorithm. As shown in figure 3 the adopted protocol prevents from delays between test sequences, ensuring an interrupted at-speed testing (even between sequences), which in turn allows a maximum optimization of the overall test time.

The synchronization between the serial shift of the scan register and parallel load of the instruction register is possible thanks to the pre-computation of the exact load instances (in figure 3, pre-computed instance t1 indicates that the sequence 0 has been finished and sequence 1 will be loaded) or through handshaking for ATE supporting it.

We observe that using the shift mode, we have a maximum flexibility (test algorithms comprising any number of March sequences can be executed) at minimal cost, i.e. no memory is needed for storing the instructions of the test sequence.

Preset Mode: Companies may prefer to fix during the design phase the test algorithm used during fabrication test, and reserve the programmable test mode to experts for performing debug, failure analysis of field failures, or in cases where it is discovered after design and production that the memory includes unexpected faults not covered by the pre-selected test algorithm. In this case, the fixed test algorithm is implemented by using some logic that implements the instructions corresponding to the preset test algorithm(s). A counter that drives the inputs of this logic is incremented at the end of each March sequence to select the instruction corresponding to the next March sequence of the pre-selected test algorithm. During fabrication test, the preset mode is activated by default, while the experts activate the expert mode that enables programming the test algorithms of their choice.

Theoretically, the user can generate as many preset algorithms as he wants. Practically however, this number should be kept relatively low to ensure a good area overhead / BIST control tradeoff. Therefore, in order to maximize BIST sharing, only well proven and general purpose test algorithms can be set as preset algorithms, the remaining ones (e.g. specific for a given memory type and/or a given memory operating mode) can be performed using the Shift Mode.

Note that except for the Instructions generator module (not shown in figure 1) that replace the Scan register, all the remaining modules of the Preset Mode are shared with those of the Shift Mode architecture (see figure 1). As mentioned before, these modules already beneficiate from an efficient implementation, which results in an area optimized implementation of the overall Preset Mode.

6. Programmable Data Backgrounds

A limitation of the March test algorithms is the use of a single data word and its reverse during a March sequence. More complex data backgrounds can be required to test for some fault models. Some of these backgrounds are simple, such as for instance checker-board. To implement simple data backgrounds, the instruction register is expanded to add a data background field. This field determines the data background used during the current March sequence. Some logic combines the value of this field with the Data field to determine the test data. For instance, when the data background field checks for checkerboard, the data are reversed (in case of interleaved word organization and/or odd number of data bits) when the parity of the less significant bit of the column address and of the row address is 1.

More complex data backgrounds can be implemented by means of an auxiliary memory of small size (figure 2), to create periodic data backgrounds in the memory cell array (figure 2). Such data backgrounds are useful to cover neighborhood pattern sensitive faults. Before applying the test algorithm, the auxiliary memory is loaded in a shift mode (e.g. by means of a scan path) with a given background. Then, the BIST circuitry controls the memory under test and the auxiliary memory and performs the test algorithm. The auxiliary memory is addressed by the less significant row and column address bits generated by the BIST hardware (test address signals, TAS in figure 2). The data read from the auxiliary memory, are expanded and supplied as write data to the memory under test (TDS signals in figure 2), and as read data to the BIST comparator of figure 1. After the test algorithm is executed with a given data background, a new data background is loaded to the auxiliary memory and is used by the BIST to execute the test algorithm with this background. For instance, consider a memory using a 1-out-of-4 or higher column multiplexing. To test this memory for any data background covering any neighborhood of 4-by-4 cells, we can use an auxiliary memory of 16 words of 1 bit, and address this memory by using the two less significant row address bits and the two less significant column address bits generated by the BIST hardware. Note however that in memories using address scrambling, this approach may require implementing a descrambling on the memory address signals in order to map the logic addressing order generated by the BIST circuit, into the addressing corresponding to the physical positions of the cells. Usually the descrambling logic is quite simple.

7. Programmable Memory BIST Tool and Experimental Results

The described programmable BIST architecture is integrated in the memory BIST generation and insertion tool. The tool offers a large flexibility to the designer to select:

- minimal programmable BIST configuration
that includes shift mode for loading the instructions register, binary mode of addressing, fixed data backgrounds,
- expanded programmable BIST configurations using extra modes selected by designer. Some of the possible options for these configurations are: preset algorithms specified by the designer, fast-X, fast-Y, fast-Z addressing modes, programmable backgrounds based on auxiliary memory, sharing of the same BIST controller among a group of memories and test data expansion period.

Two different types of experimentations have been conducted.
The first one focused on the comparison of our solution against existing benchmarks. We chosen for this purpose the best in class programmable Memory BIST solutions proposed in [3] and [2] (referred as PBIST1 and PBIST2 respectively). Table 1 summarizes the results in terms of area size and number of sequential/combinatorial units to match the representation of the original benchmarks.
The number of operations per sequence parameter was set to 3 (i.e. the architecture supports a maximum of 8 operations per sequence, but no other limitation on the test length exists), which covers the whole majority of well known test algorithms (e.g. March C, B, LR, LA, X…)

Hardware resources	Proposed PBIST	PBIST1	PBIST2
Area (mm2) for 0.35 micro and 1 M bits RAM	0.145	0.191	-
Sequential units /logic units (256kx8 RAM)	49/160	-	450/2500

Table 1: Area overhead comparison

Among the chosen benchmarks, PBIST1 gives the lowest area overhead. We observe that even if its area is based upon a particular case of symmetric algorithms, and thus it is rather an optimal estimation, our approach improves it by saving up to 22 %.
Furthermore, our architecture is at most 1/10th of the PBIST2 area. This factor is rather pessimistic since we haven't taken into account the area of the 32x14 bits RAM used as a storage module. Although PBIST2 provides slightly more functionalities, we still convinced however, that even with equivalent features our BIST still offer the lowest area.
Therefore, these experimental results confirmed the important save in area overhead achieved by the proposed programmable BIST solution in comparison to the traditional micro-code programmable BIST approaches.

The second set of experiments focused on the evaluation of our solution in real SoC environment. For this purpose, three BIST architectures have been considered: a BIST architecture using only Preset Algorithm (as described in section 5), a programmable BIST (data and algorithm programmability) and a customer internal dedicated BIST solution, using a tradition parallel BIST implementation. Our Preset BIST and the Customer BIST both implemented the March LR algorithm (non programmable BISTs). The experiment was made for a Telecom SoC that contains about 450 memories instances (169 different RAMs) using a 0.13 micro technology.

All BIST architectures have been synthesized and simulated with Synopsys tools (Design Compiler and VCS) using a distributed BIST approach.
Figure 4 gives the area overhead of the three BIST architecture for set of 169 memories (for confidentiality reasons the results are given in term of unit X of the area overhead). Table 2 gives the cumulative area overhead for the entire SoC memory instances.

BIST Architecture	Preset BIST	Prog BIST	Customer BIST
Area Overhead (% of cumulative areas of memory instances)	1X	1.9X	2.1X

Table 2: Soc Cumulative area overhead for the three architectures

We observe that due to our hierarchical algorithm decomposition, the preset BIST is half the traditional BIST area.

The programmable BIST, although with the programmability feature, results in area less or equal to the traditional dedicated BIST.

These results confirmed that the proposed original programmable BIST approach results in area overhead comparable (and even equal) to those of non-programmable BIST.

8. Conclusion

An original programmable BIST solution has been presented. In comparison to the traditional micro-code based programmable BIST, our solution offers a set of benefits (Table 3). The hardware resources allocation is not the same in the two approaches, resulting in limitations of the number of operations per sequence in proposed solution, and the test length in the other approaches. As a consequence, some dynamic fault requiring a high number of at-speed repetitive operations cannot be tested by the later. Also, our programmable BIST consumes much less hardware, while ensuring (at least) equivalent flexibility. And finally, the testability of the proposed solution is higher, since it ensures full scan testing (the traditional approaches use a RAM that decreases their overall testability).

Criteria	Proposed PBIST	Existing PBIST
Resources	1 instruction/ sequence	1 instruction / operation
Limitation	#Operations/ sequence	Test length
Area Overhead	Low	Medium to high
Flexibility (for comparable area)	Medium to high	Low to medium
Testability	High	Medium

Table 3: Proposed Programmable BIST Vs micro-based BISTs

Therefore, we presented an original programmable BIST solution allowing for good flexibility on both algorithm and test data programmability. This flexibility is particularly suitable for debugging a new fabrication process or memory design, as well as for testing unexpected failures discovered after design and fabrication

A very important aspect of the proposed solution is that it allows for testing of vast variety of memory fault models while maintaining a very compact implementation, which results in an area cost similar to traditional memory BIST schemes.

9. Acknowledgments

The authors would like to thanks M. Issam Nofal for providing help in the evaluation of the presented solutions.

10. References

[1] H. Koike, T. Takeshima, and M. Takada. “A BIST Scheme Using Microprogram ROM for large Capacity Memories”. In Proc. Of International Test Conference, 1990, pp 815-822

[2] I. Schanstra, D. Lukita, Ad, J. van de Goor, K. Veelenturf, P.J. Van Wijnen “Semiconductor Manufacturing Process Monitoring Using Built- In Self Test for Embedded Memories”. In Proc. Of International Test Conference, 1998, pp 872-881.

[3] k. Zarrineh and S.J. Upadhyaya. “On Programmable Memory Built-In Self Test Architectures”. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 1999.

[4] J. Dreibelbis et al. “Processor-based Buil-In Self Test for Embedded DRAM”. IEEE J. Solide State Circuits, vol 33, No 11, Nov 1998, pp 1731-1740.

[5] C.T Huang, J.R Huang, C.F Wu, C.W Wu, T.Y Chang “A Programmable BIST Core for Embedded DRAM”. IEEE Design & Test of Computers, 1999, pp 59-69.

[6] BOUTOBZA SLIMANE, “Tools for Memory BIST/BISR generation”, Dec, 2002, TIMA Laboratory INPG, Grenoble France.

[7] NICOLAIDIS MICHAEL and BOUTOBZA SLIMANE, European Patent No EP1343174, 2003-09-10

[8] NICOLAIDIS MICHAEL and BOUTOBZA SLIMANE, US Patent No US2003167431 , 2003-09-4

[9] A.J. van de Goor, "Testing Semiconductor Memories, Theory and Practice", John Wiley & Sons publisher, 1991

[10] SUK D. S., REDDY S. M. - "A March test for functional faults in semiconductor random access memories", IEEE Transactions on Computers, vol. C-30, n° 12, December 1981

[11] NAIR. R THATTE S. M, ABRAHAM J. A. "Efficient algorithms for testing semiconductor Random-Access Memories", IEEE Transactions on Computers, Vol C-27, pp 572-576, June 1978).

[12] : A.J. Van de Goor, J. de Neef “Industrial Evaluation of DRAM Tests”, Design, Automation and Test in Europe, Munich, Germany, March 1999, pp. 623-631.

[13] Y. You and J. Hayes, “A Self-Testing Dynamic RAM Chip,” ***Roc.*** *MIT Conf Advanced Research in* WI, Jan. 1984, pp. 159-168.

[14] : B. Nadeau-Dostie et al, “Serial Interfacing for Embedded Memory testing”, IEEE Design and Test of Computers, April 1990.

[15] : M. Nicolaidis, “An efficient Built in Self Test Scheme for Functional Test of Embedded RAM”, In Proc IEEE , Fault Tolerant Computer Systems, Ann Arbor, June 1985, pp. 118-123.

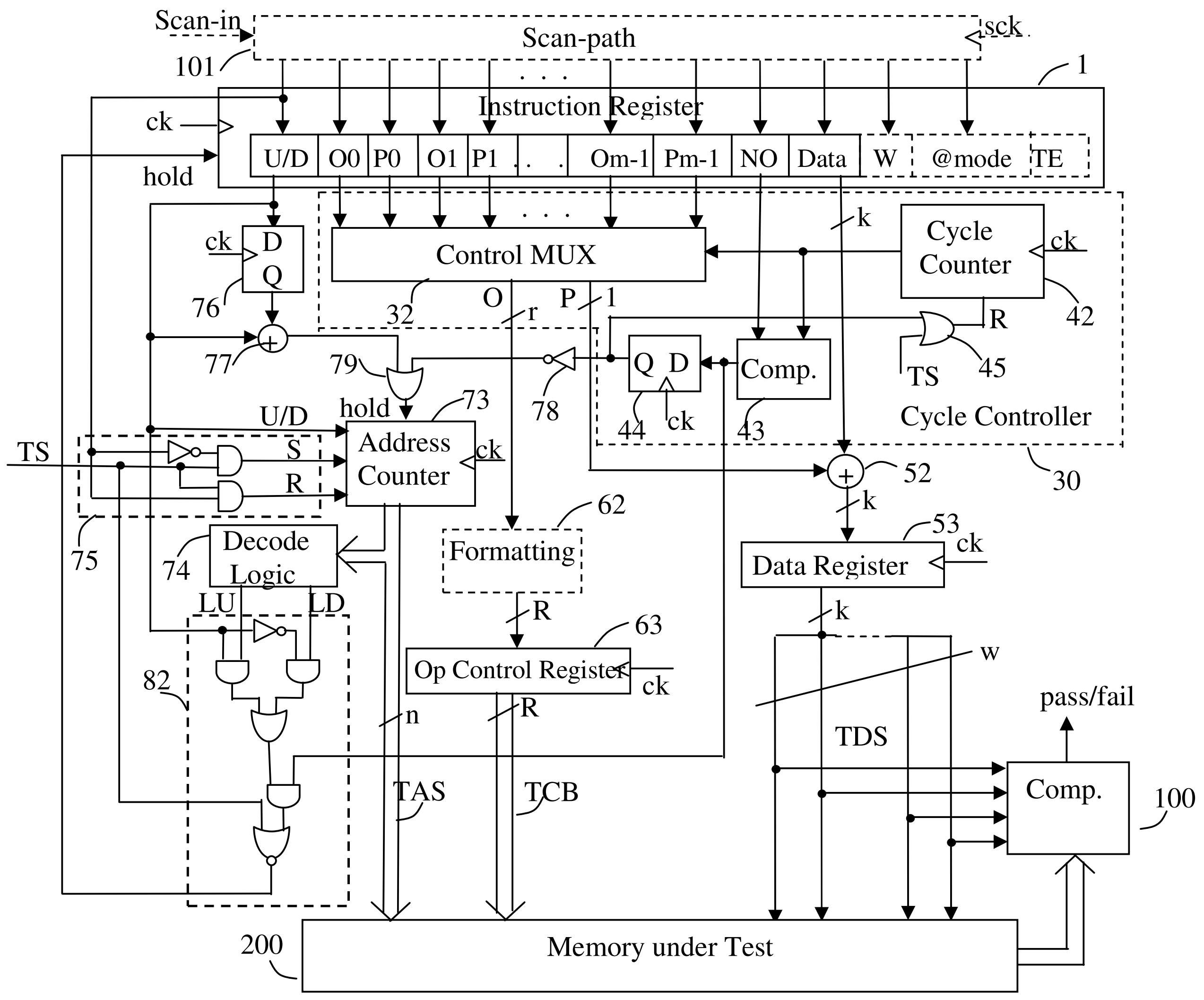

Figure 1: Programmable BIST architecture

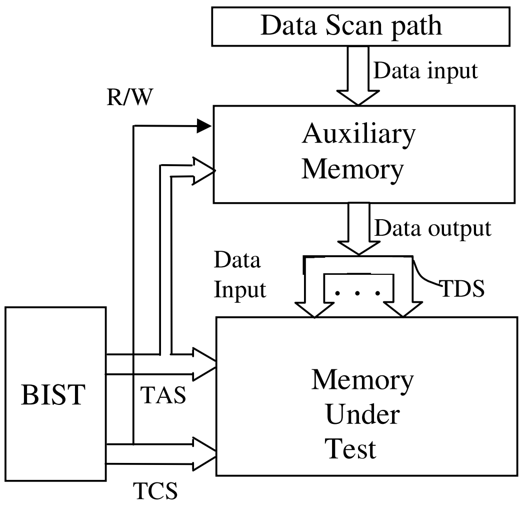

Figure 2. Auxiliary memory for data backgrounds

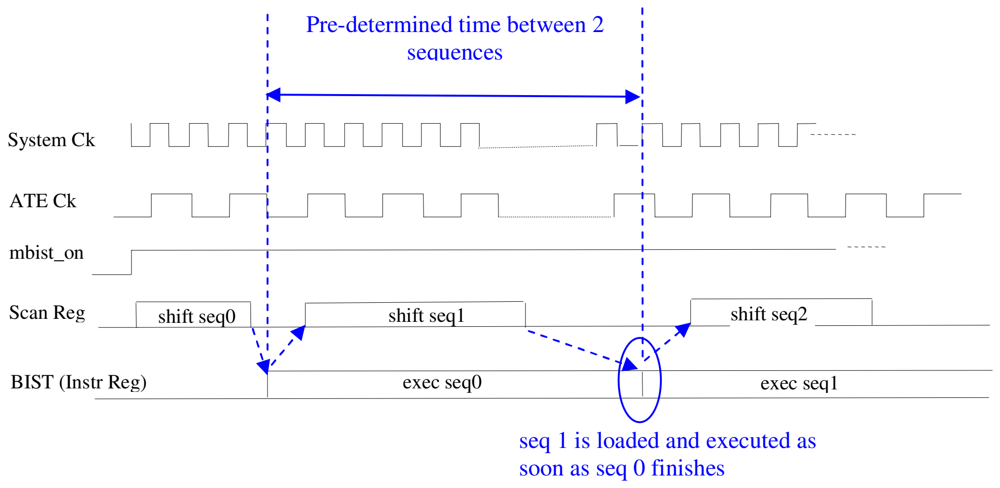

Figure 3: Programmable BIST protocol for the Shift Mode

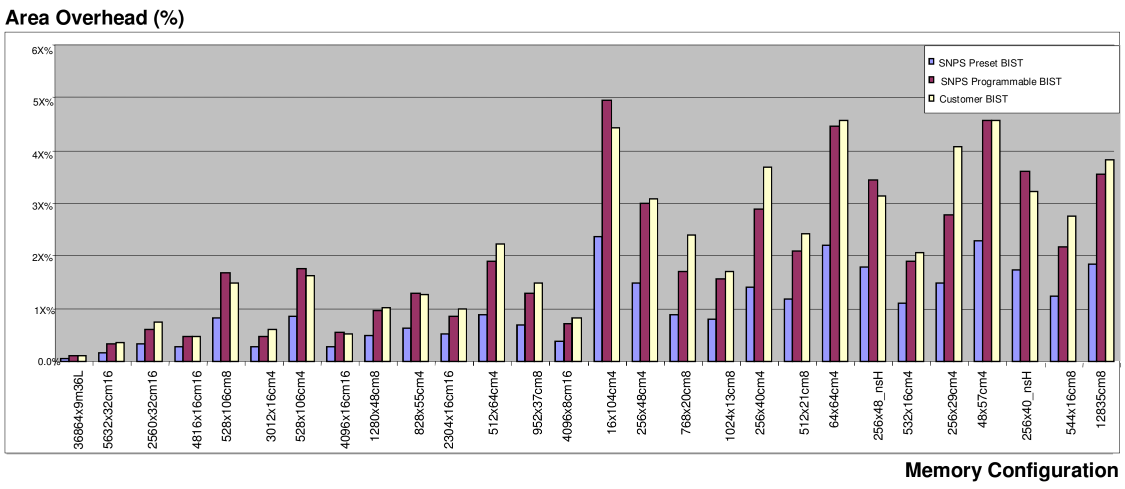

Figure 4: Area Overhead for different RAM configurations