ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 8, August 2014

An ALU Based Online BIST for Varying Word Widths of RAM

B.Niharika[1], Rani Rajesh[2]

Student, Electronics and Communication Department, Stanley College of Engineering, Hyderabad, India[1]

Associate Professor, Electronics and Communication Department, Stanley College of Engineering, Hyderabad, India[2]

Abstract: On-line testing of word oriented memories is fast becoming a basic feature of digital systems, not only for critical applications, but also for highly-available applications. With the use of transparent BIST (built in self test) schemes for testing of RAMs, preservation of memory contents during periodic testing is assured but, it requires more hardware for signature prediction and more time for testing. Symmetric Transparent BIST skips the signature prediction phase, thus reducing the hardware and test time. Previously, one ALU was used for testing one RAM which increases the hardware of BIST circuit. The proposed method uses a symmetric transparent BIST scheme and an ALU capable of testing more than one RAM thus decreasing the hardware required to test RAM modules on a die. Different word widths of RAMs can be tested by using addition and subtraction operation of ALU with the help of series of March elements. Four RAM modules are tested using one ALU module in a roving manner. Due to the decrease in hardware overhead of proposed scheme the test time is also reduced.

Keywords: BIST (built in self test, March algorithms, memory test, Symmetric transparent built in self test (BIST), online testing, memory testing, and hardware overhead.

I. INTRODUCTION

The system on chip (soc) design's current practice is to allocate a large amount of chip area for memories, the manufacturing yield of such devices greatly depend on the yield of embedded memories [1] .The yield of these devices is greatly decreased because large number of defects occur by the use of memories. Memory arrays are more susceptible to defects as they are designed with minimal design rule tolerances. Numerous test algorithms and fault models have been introduced to detect defects in a memory. These test algorithms can be implemented effectively by the use of BIST (built in self test).BIST is capable of meeting certain requirements such as reduction in cost of testing, limited technician accessibility, high reliability, low repair cycle time, high fault coverage, full speed test application, extensive diagnostics, and on-chip test hardware thereby eliminating the need for sophisticated ATE. In online BIST the testing occurs during normal functional operating conditions, it provides real time fault detection. In critical applications such as space applications, it is impractical to shut down the system since contents of the memory must not be lost therefore online BIST is used for real time testing for fault detection, this is periodic testing in normal operation. The normal operation of RAM modules is halted then they are tested and again the normal operation is continued.

March test algorithms are capable of locating and identifying the faults, and a major advantage is that it has high fault coverage and test time is usually linear with the size of memory which is acceptable in industrial standards. Each location in RAM is tested using these algorithms, set of data is written into each memory address and read back to verify it. If all the values that are read are same as values which are written the RAM is not faulty, otherwise it is faulty. A March algorithm consists of series of March elements that perform read and write operations on RAM.

The memory which contain more than 1 bit per word is called word organized memory. This paper presents an online BIST using a symmetric transparent version of March algorithm to test various RAM modules consisting different word widths. Previously one ALU module was required to test one RAM module, in this paper one ALU module is used for testing four RAM modules as a consequence attaining reduction in hardware overhead and test time.

II. MARCH ALGORITHMS

March algorithms comprises of series of March elements. Traditional march algorithms consist of write all zero initially, here zero is written into all the RAMs this was used to make sure that the value of final signature in output compactor is known [9]. As shown in fig 1 each March element of March algorithm consists of a read operation or write operation or both of the operations. Write 0 is denoted by w_0.

Write 1 is denoted by w_1. Read 0 is denoted by r_0. Read 1 is denoted by r_1. ⇧ Denotes an increasing addressing order and ⇩ denotes a decreasing addressing order. The six march elements are denoted by m_0 to m_5 . The first March element w_0 writes 0 into all the locations of the RAM therefore, all the original contents of RAM are lost it cannot be retrieved for future use.

m_0	⇧ (w_0)
m_1	⇧ $(r_0 w_1)$
m_2	⇧ $(r_1 w_0)$
m_3	⇩ $(r_0 w_1)$
m_4	⇩ $(r_1 w_0)$
m_5	⇩ (r_0)

Fig. 1 March algorithm

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 8, August 2014

A. *TRANSPARENT MARCH ALGORITHM*

Transparent algorithm is designed in order to preserve the memory contents of ram before the testing phase. This is achieved by use of signature prediction phase.

$\Uparrow (r_a); \Uparrow ((r_a)^c); \Downarrow (r_a); \Downarrow (r_a); \Downarrow (r_a);$
$\Uparrow (r_a w^c{}_a);$
$\Uparrow (r_a w_a);$
$\Downarrow (r_a w^c{}_a);$
$\Downarrow (r_a w_a);$
$\Downarrow (r_a);$

$\boldsymbol{r_a}$ Read the contents of a word of the RAM, expecting to read the initial contents of the RAM word (i.e., before the beginning of the test).

$\boldsymbol{r_a^c}$ Read the contents of a word of the RAM, expecting to read the complement of the initial contents of the RAM word.

$(r_a)^c$ Read the contents of a word of the RAM expecting to read the initial word contents and feed the complement value to the compactor.

$\boldsymbol{w_a}$ Write to the memory word; the value that was stored in this memory word at the beginning of the test is (assumed to be) written to the word.

$\boldsymbol{w_a^c}$ Write to the memory word; the inverse of the value that was stored in this memory word at the beginning of the test is (assumed to be) written to the word.

By default, the data driven to the compactor with the $(r_a)^c$ operation are identical to the data driven by the r_a^c [2], [10].The importance of the $(r_a)^c$ operation is the following: during the signature prediction phase the contents of the RAM are equal to the initial contents (since no write operation has been performed); therefore, in order to simulate the r_a^c operation we invert these contents prior to driving them to the compactor[11].

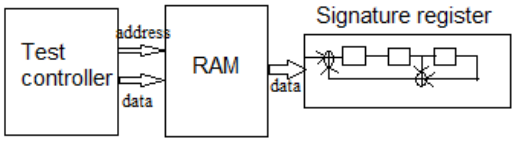

Fig. 2 Transparent BIST

The first March element is the signature prediction phase as shown in fig 2 is needed for calculating the learnt signature which depends on the memory contents, and the latter phase is used for RAM's testing. During the testing process after each Read operation the obtained data is fed into the signature register, and at the end of the test session the final signature is compared with the learnt signature. The signature prediction phase consists of all Read operations of the complete march algorithm [8],[10]. One third of the test time is required for signature prediction. To cope with this problem symmetric transparent BIST algorithm is used.

B. *SYMMETRIC TRANSPARENT MARCH ALGORITHM*

The symmetric march algorithm produces a symmetric test data string D. The march elements of the transparent algorithm are modified in such a way that the result obtained in the register is equal to a known value i.e. all ones [5]. In this way the need for signature prediction is eliminated and the test time is also reduced.

$\Uparrow ((r_a)^c);$
$\Uparrow (r_a w^c{}_a);$
$\Uparrow (r_a w_a);$
$\Downarrow (r_a w^c{}_a);$
$\Downarrow (r_a w_a);$
$\Downarrow (r_a);$

By the first element it is evident that the signature prediction phase of transparent BIST is eliminated. The utilization of accumulator modules for output data compaction in symmetric transparent BIST for RAMs is used. If the March algorithm is symmetric (as in the case of symmetric transparent BIST) then the number of ra elements equals the number of ra^c elements plus the number of $((ra)^c)$elements (without taking into account the addressing order of the march element) [3],[4]. The accumulator-based compaction of the responses holds the order-independent property (i.e., the final signature is independent of the order of the incoming vectors. If a symmetric transparent march algorithm is applied to a word-organized memory whose word length is n and the responses are captured in an n-stage accumulator comprising a 1's complement adder (starting from the all-0 state), then the final content of the accumulator is equal to the all-1 state [7].

Let D Î {0, 1}2n be a data string. D is called symmetric, if there exists a data string d Î {0, 1}n with D = (d, d*) or D = (d, d*c). A transparent march test is called symmetric if it produces a symmetric test data string D. In general, it cannot be expected, that an arbitrary

Transparent march test is symmetric [6]. However, typical transparent test algorithms contain symmetric subsequences and can be easily extended to fully symmetric versions. Consider the March C- algorithm as an example. It is originally defined as {c(w0); Ý(r0, w1); Ý(r1, w0); ß(r0, w1); ß(r1, w0); c(r0)} leading to the transparent version {Ý(ra, wac); Ý(rac, wa); ß(ra, wac); ß(rac, wa); c(ra)}. The test data stream fed to the signature analyzer is not symmetric . For example the 4-bit memory of provides the sequence (1101, 0010, 1011, 0100, 1011) with a decreasing addressing order in the last phase. If this sequence is extended to (**0010**, 1101, 0010, 1011, 0100, 1011)), then it can be written as (d, d*c) with the symmetry axis after the first 12 bits. Such an extended symmetric sequence is for example produced by the extended test {Ý**((ra)c)**; Ý(ra, wac); Ý(rac, wa); ß(ra, wac); ß(rac, wa); ß(ra)}, which guarantees at least the same fault coverage as the original test. Although the

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 8, August 2014

extension will increase the test time from 9n to 10n, there is still a considerable gain in efficiency compared to the conventional 14n approach with signature prediction. Similarly, for any of the known transparent algorithms it is possible to identify a potential symmetry axis and to add some additional read sequences to obtain symmetric versions of the algorithms. Table 1 shows the resulting test lengths for some commonly used transparent march tests.

TABLE 1
COMPARISON OF TEST TIMES FOR TRANSPARENT AND SYMMETRIC TRANSPARENT BIST

Algorithm	Transparent BIST			Symmetric Transparent BIST
	Signature prediction	Test	Total Time	Total time
MATS +	2n	4n	6n	4n
March C-	5n	9n	14n	10n
March A	4n	14n	18n	16n
March B	6n	16n	22n	18n
March Y	3n	5n	8n	6n

It can be observed that in all cases the symmetric versions of the considered transparent algorithms require a considerably shorter test time than the original versions.

The properties of the proposed BIST technique with respect to error masking and fault coverage are summarized in the following observations. The stuck-at fault (SAF) can be described as follows: The logic value of a stuck-at (SA) cell or line is always 0 or 1. (It is always in state 0 or in state 1 and cannot be changed to the opposite state.) A test that has to detect and locate all stuck-at faults should satisfy the following requirement: From each cell or line, a 0 and a 1 must be read. A special case of the SAF is the transition fault (TF). It is defined as follows: A ceil or line that fails to undergo a 0 + 1 transition when it is written is said to contain an up transition fault, for which the notation T-T will be used. Similarly, a T-J fault indicates the impossibility of making a 1 to 0 transition. The Coupling Fault (CF) used here and by is the 2-coupling fault. It involves two cells and is defined as follows: A write operation that generates a 0 or a 1 transition in one cell changes the contents of a second cell. Two different types of coupling faults can be distinguished, idempotent coupling fault and inversion coupling fault. The idempotent coupling fault is the one in which a transition in one cell forces the contents of second cell to a certain value 0 or 1. Inversion coupling fault is the one in which transition of one cell inverts the contents of second cell. A test that has to detect and locate all coupling faults should satisfy the following requirement: For all cells that are coupled cells, each cell should be read after a series of possible coupling faults may have occurred (by writing into several coupling cells), with the condition that the number of transitions in the coupled cell is odd. This requirement ensures that all coupled cells are read while they are in a state opposite to the expected state.

TABLE 2
SIMULATION RESULTS (FAULT COVERAGE IN % FOR SINGLE FAULTS) FOR A 32 KBIT MEMORY

Algorithm	Fault Model			
	SAF	TF	CFid	CFin
March C Transparent symmetric	99.9992 100	99.991 100	99.995 100	99.996 100
March C- transparent symmetric	99.992 100	99.991 100	99.996 100	99.997 100
March X transparent symmetric	100 100	100 100	49.993 49.996	99.992 99.997

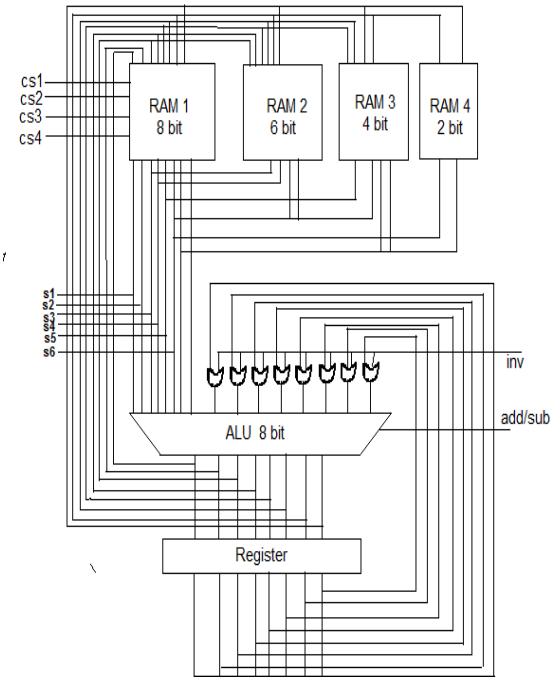

Fig.3. Proposed ALU based BIST for varying word widths of RAM

The RAM module implemented consists of two modes of operations such as Read & Write. The RAM module during Write operation address is given and the data that has to be stored is also given. The data will be stored in the specified address. During Read operation the address is specified. The data that is present in the specified address is given on the output data signal. In the architecture four RAMs are implemented with different word widths. As shown in fig 3 the RAMs can store the 8bits, 6bits and 4bits and 2bits respectively. The RAMs are selected depending upon the chip select signals cs1, cs2 and cs3 and cs4. When CS1 is enabled RAM1 is selected for testing. When CS2 is enabled RAM2 is selected for testing. When CS3 is enabled RAM3 is selected for testing. When CS4 is enabled RAM4 is selected for testing. Stuff signals are connected to the output signals of the RAMs that are passing through the ALU. If RAM of 4 bits is being accessed then the Stuff values must be given in order to make it 8 bits. If RAM of 8 bits is being

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 8, August 2014

accessed then the Stuff values must not be given because it is already 8 bits..). When RAM 4 is selected then the stuff values are given as "000000", because the ALU that we have implemented is of 8-stage. When RAM2 is selected then the stuff values are '0000'. There is no stuff value when we have selected RAM4.

B. *OPERATION OF ALU MODULE:*

The ALU module in the architecture stands for Arithmetic and Logic Unit which performs arithmetic and logical operations on the data. The ALU implemented in this architecture is a 1's complement ALU. For the description of the project, we will denote with n the number of stages of the ALU that can perform one's complement addition and with k the number of bits of the RAM word (hence k<n) .The purpose of the proposed scheme is to assure that the contents of the register will be equal to a specific value (i.e. '11…1') at the end of the test. In order to assure this, the (n-k) high-order inputs of the ALU are appropriately driven by the all-1 or all-0 value. It should be noted that, if the march algorithm is symmetric, then the inputs driven to the march algorithm n is symmetric, then the inputs driven to the response verifier and data generator are complementary during consecutive march elements. In order to expand this for the case where the width of memory is smaller than the number of ALU stages, we add a series of (n-k) High-order inputs of exactly the half elements added to the ALU. This stems from the fact that if $a \leq 2^k$, since

$$a + a^c = 2^k - 1,$$

Then we also have that

$$(\sum_{i=k}^{n-1} 2^i) + a + a^c = 2^n - 1$$

For example, if k=6 and n=8, let us take the case where a= $(000101)_2 = (5)_{10}$ *and* $a^c = (111010)_2 = (58)_{10}$.

then we have that

$$(\sum_{i=k}^{n-1} 2^i) + a + a^c = 2^6 + 2^7 + 5 + 58$$

$$= 64+128+5+58$$

$$=255$$

$$= 2^8 - 1 = 2^n - 1$$

Using the above observation, we can extend the scheme in order to handle the case where the ALU has, more stages than the RAM word width. More precisely, we can stuff the high-order bits with a signal that has the value '1' during half the cycles and '0' during the other half. Input is driven by the inv signal in Fig 3. Therefore, the inverse of the read vector appears at the outputs the adder/subtractor and applied to the RAM inputs.

Let us consider the 6-bit RAM1 presented in Fig 3. The outputs of the memory are driven to an n = 8-stage ALU comprising a 1's complement adder. For the implementation of the $(ra)^c$ march element, the subtraction operation of the accumulator can be utilized. In order to apply march elements of the form (r_a, w_a^c) or (r_a^c, w_a) the output of the RAM must be inverted and then fed back to its inputs; with the proposed scheme, this can be done by forcing the all-1 vector to one input of the adder/subtractor and perform a subtract operation. This is done with the OR gates whose one whose one input is driven by the inv signal in Fig. Therefore, the inverse of the read vector appears at the outputs the adder/subtractor and applied to the RAM inputs. The addition operation is done by a formula given as (A-1+B(MOD 2^n-1)+1). The MOD operations can be seen in Table 3.

TABLE 3
OBSERVATION OF MOD VALUES IN ADDITION OPERATION

A-B+1	Result of MOD operation
0	0
1	1
2	2
3	3
. . .	
255	255
256	0
257	1
258	2
. . .	
510	254
511	255

Subtraction is given as

If (A>B) then not (A-B)

If (A<B) then not (B-A)

Let A= 000**010** i.e (ra = 010)

B= 000000

A-B =0 00010

not(A-B)= 111**101** i.e$((ra)^c$ = 101)

Let B =000**010** i.e (ra = 010)

A= 000000

B-A =0 00010

not(B-A)= 111**101** i.e$((ra)^c$ = 101)

Thus, each March element in March algorithms can be implemented using the addition and subtraction operations of ALU module.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	374	4656	8%
Number of Slice Flip Flops	399	9312	4%
Number of 4 input LUTs	480	9312	5%
Number of bonded IOBs	60	232	25%
Number of GCLKs	5	24	20%

Fig.4. Device utilization summary of BIST using single ALU for varying word widths of RAM

The device utilization summary of BIST using single ALU for four RAMs of different word widths is shown in fig.4.the implementation is done using Spartan3 FPGA(Xc3s500e-5fg320). The number of slices available

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 8, August 2014

is 4656 of which 374 are used. The logic utilization can be seen in fig.4 along with the percentage use of the logic devices. The previous schemes used for testing RAMs required one ALU module for Each RAM therefore the logic utilization for 4 RAM modules is four times more than the logic utilization of proposed scheme. There is a decrease in the hardware in the proposed scheme, the hardware utilization is 84% less when it is compared with previous schemes.

III. SIMULATION RESULTS

In this work we have designed a ALU based scheme for online BIST of RAMs. The code is written in VHDL language. Xilinx ISE 13.2 is the tool used to convert the VHDL language to hardware blocks and implemented on Xilinx Spartan 3e based Xc3s500e FPGA device. The benefits associated with FPGA such as flexibility, shorter time to market and reconfigurability make them a very attractive choice for implementing the designs. A full software simulation approach is useful in two aspects. It offers full visibility into the design and allows the test bench or design to be changed and re-verified in a rapid manner. The challenges, however, are getting an accurate simulation model of the external memory module and achieving a reasonable simulation speed. In contrast, running the design in hardware addresses these problems, but at the cost of reduced visibility into the design. It is also very complex to set up and change the test bench in hardware. I Sim hardware co-simulation gives flexibility to run a portion of design in hardware while simulating the rest in software. The memory controller and external memory are, for example, good candidates to put in hardware so that they are modelled accurately and simulated quickly. The test bench and the application logic in your design, which are under development, should be simulated in software so you can change, verify and debug them easily and rapidly. In the Xilinx simulation environment, we create an ise file, and then we specify the features of the device such as the device family, device, package, speed grade, synthesis tool, simulation tool, top level module type and the language. Then we add the source files i.e. the code and its test bench. Thus the ise file is created. Test bench with several test cases were setup to verify the expected results. After building the files we could able to see the design overview of the top level file, device utilization summary and performance summary. The figure 5 shows the result of our top module simulation, where the register of 8 bits is holding the all-1 value. If the register holds some other value then it results that there is a fault in the RAM module. These faults are shown as fault 1, fault 2, faults 3, fault 4, of RAM 1,2,3,4 respectively (the RAMs as shown is the fig.3).

IV. CONCLUSION

In this paper it is shown that need for using BIST for memory is observed that today's all the SOC chips contain most area occupied by memory and BIST reduces the complexity, and thereby decrease the cost and reduce reliance upon external (pattern-programmed) test equipment.

By utilizing symmetric transparent algorithm the hardware used for testing is considerably decreased because of skipping the signature prediction phase previously used in transparent algorithms. The fault coverage and test times are also compared and it is observed that symmetric transparent algorithm technique used in this paper required least test time. The fault coverage of symmetric transparent algorithm is also very high. Four RAM modules of word widths 2,4,6,8 are tested using a single 8 bit ALU instead of using one ALU module for single RAM or similar word width of RAM modules thus the ALU modules are also decreased.

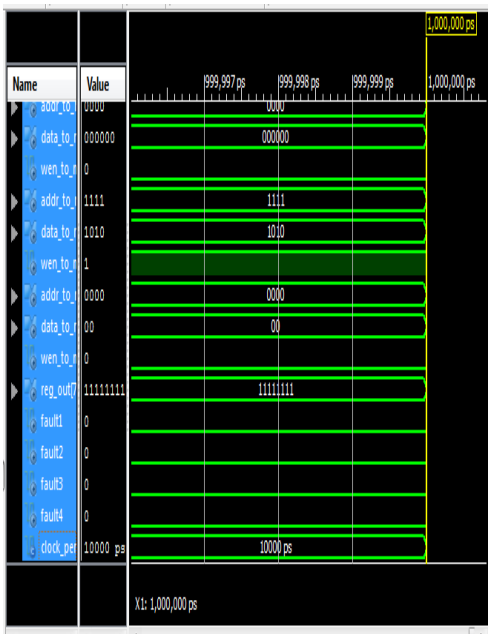

Fig.5. Result of simulation.

REFERENCES

[1] X.DU,N.Mukherjee, W.T.Cheng and S.M Reddy, "Full speed field programmable memory BIST architecture", Proc. Of Int. Test conference, pp.1173-1182, 2005.

[2] Jin-Fu Li, "Transparent test methodologies for Random access memory without/with ECC", Computer aided design of Integrated circuits and systems, IEEE transactions on page: 1888-1893, volume: 26 Issue: 10, Oct.2007.

[3] R. Dorsch, H-J. Wunderlich, "Accumulator-Based Deterministic BIST", International Test Conference, pp.412-421, 1998.

[4] I. Voyiatzis, "An Accumulator –based compaction scheme with reduced aliasing for on-line BIST of RAMs", IEEE Transactions on VLSI Systems, vo.16, no, 9, September 2008, pp.1248-1251.

[5] V. N. Yarnolik, S. Hellebrand, H.-J. Wunderlich, "Symmetric Transparent BIST for RAMs", in Date 1999, Munich, Germany, March 9-12, 1999.

[6] I. Voyiatzis, " Accumulator –based compression in Symmetric Transparent RAM BIST", in IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Technology, 2006.

[7] Stroele, "BIST pattern Generators using Addition and Subtraction operations", Journal of Electronic Testing: Theory and Applications, vol. 11, pp. 69-80,1997.

[8] V. N. Yarnolik, I.V. Bykov, S. Hellebrand, H.-J. Wunderlich, "Transparent Word oriented memory BIST based on symmetric March algorithms", in European dependable computing conference, April 2005.

[9] A.J.Van de goor "using march test to test SRAMS" IEEE design and test of computers.1993.

[10] M. Nicolaidis, "Theory of transparent BIST for RAMs" IEEE transactions on computers, vol.45, no. 10, October 1996,pp.1141-1156.

[11] [11] K. Thaller and A. Steininger, "A transparent online memory test for simultaneous detection of functional faults and soft errors in memories," IEEE Trans. Rel.,vol. 52, no. 4, pp.413-422, Dec.2003.