Journal of Theoretical and Applied Information Technology
10th July 2014. Vol. 65 No.1
© 2005 - 2014 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DEEP ENCODING WITH CLUSTERING TECHNIQUE FOR PROGRAMMABLE MBIST

[1]NURQAMARINA BINTI MOHD NOOR, [2]AZILAH SAPARON

Faculty of Electrical Engineering
UniversitiTeknologi MARA
Shah Alam, Selangor
Email: [1]nurqamarina@isiswa.uitm.edu.my , [2]azilah574@salam.uitm.edu.my

ABSTRACT

As latest trend in designing processors and system-on-chips (SoCs), they require more RAMs than logics. These embedded RAMs contribute to the high percentage of yields for these processors and SoCs. To ensure high percentage of yield is achieved, a built-in self-test (BIST) is utilized to test these RAMs. The memory BIST applies various test algorithms such as MARCH tests to detect various RAM faults. Numerous design objectives such as programmability, low area overhead, at-speed/full-speed test and multiple RAMs target are proposed in the BIST designs. These objectives must be achieved to provide best fault detection in these embedded RAMs. A technique called clustering which is applied to other architectures such as VLIW processor and FPGA architecture is utilized in this study to achieve low area programmable memory BIST (P-MBIST). The synthesis results justify that the cluster technique provides low area overhead for the programmable memory BIST controller at optimum performance.

Keywords:- *P-MBIST, MARCH Test, Cluster, Low Area, FPGA*

1. INTRODUCTION

The future trends in the processors and systems on chip (SoCs) are moving from logic and memory balanced chips to memory dominated chips in order to deal with the increasing application requirements. According to International Technology Roadmap for Semiconductor (ITRS) that is provided by Semiconductor Industry Association (SIA) in 2001, the embedded memories are expected to utilize more than 83% of the chip area after 2008. Typically, embedded memories account approximately half the area of the microprocessor [1] and their density is continuously raising. They are scattered around the device (SoCs) rather than concentrated in one location [2]. As a result, the and intra-word coupling faults are the faults that are evaluated on these RAM. These faults can be detected using the MBIST controller. The MBIST controllers can be classified as FSM-based or microcode-based. Both MBIST controllers are designed to be programmable to allow multiple MARCH test patterns to be added without requiring changes on the controller's architecture. The architectures of the available programmable MBIST (P-MBIST) controllers are studied to analyze their area-efficiency, full-speed capability and ability to be shared to test multiple RAM cores.

overall SoC yield is dominated by the memory yield. In order to achieve high memory yield, a thorough understanding of memory design, faults models and adequate tests strategies is a must.

SRAM can be organized as BO-RAM(Bit-Oriented) and WO-RAM(Word-Oriented RAM). Bit-oriented memories (BO-RAM) are the memories where the read/write operations are performed on the RAM cell arrays by bit. Word-oriented memories (WO-RAM) where the read/write operations are performed on the RAM cell arrays by word. Both BO-RAM and WO-RAM have two types of fault models: static and dynamic. There are various types of faults can occur in both types of RAMs. The static, dynamic

To achieve these factors, a technique called clustering is studied. This technique was applied widely to the datapath of the VLIW architecture [3] and FPGA architecture [4]. The area, power and speed of these two architectures are improved with the application of clustering technique on both of them.

Based on these results, the clustering technique is applied to the proposed P-MBIST controllers to achieve better area-efficiency than the available P-MBIST controllers. This paper is organized by introducing some literatures regarding RAM's fault models and test algorithms, P-MBIST designs and clustering technique. Section III and IV describe the development of clustering technique on the MARCH test algorithms. The P-MBIST architectures for the application of this clustering technique are described in Section V. The area and performance result are presented in Section III. Finally, the Section VI concludes this paper.

2. BACKGROUND AND MOTIVATION

More RAMs are embedded in today's processors and SoCs to accommodate wide range of computer applications such as gaming and Internet. These embedded RAMs use fault models to classify types of faults that may affect them during test. The RAM fault models are classified according to how the read/write operation is performed on the RAM cell arrays..

Static fault models are faults sensitized by performing at most one operation. It is divided into two: simple fault and linked fault. In [5], Van de Goor divided static faults into single-cell faults and faults between memory cells. Single cell faults are further classified into state faults (SF) and transition faults (TF). The faults between memory cells are further classified as coupling faults (CF). The dynamic faults are the faults requiring more than one operation occur sequentially in order to be sensitized. There are two types of dynamic faults: single-cell dynamic and two-cell dynamic [6]. Single cell dynamic faults are the faults that occur during read operation. Two-cell dynamic faults are basically the CF in dynamic condition.

Both static and dynamic faults that occur in the embedded RAM are detected using test algorithms. The MARCH test algorithm is the common test utilized to detect these faults. This test algorithm comprises of a sequence of MARCH elements. The MARCH element is defined as a sequence of MARCH operations applied to each cell in the memory before proceeding to next cell [7]. There are four types of MARCH operations that can be performed in each cell in MARCH test. These operations are writing 0 (*w0*), writing 1(*w1*), reading 0(*r0*) and reading 1(*r1*). The address of the next cell to be tested is determined by the ascending addressing order, $\uparrow$ or descending addressing order, $\downarrow$ or irrelevant addressing order, $\updownarrow$. Examples of earlier MARCH tests which are developed to detect static faults are MATS+, MARCH X, and MARCH C-[8]. However, these MARCH algorithms provide low fault coverage because they can only detect certain types of static faults. Nowadays, new MARCH algorithms are widely used to detect the new faults such as dynamic faults and to improve the earlier MARCH algorithms by providing high fault coverage. Examples of these MARCH algorithm are MARCH SS, MARCH SAM-opt and MARCH RAW [9, 10].

The MARCH tests cannot be applied to the embedded RAM through the chip's I/O pins, because the address, data, and control signals are not directly available through these I/O pins. Hence, the best test solution for the embedded RAM is by using built-in self-test (BIST). Typically, the BIST is designed based on the deterministic patterns such as MARCH test. This MARCH test is generally programmed inside the BIST engine. This BIST engine is basically divided into two types: state-machine and micro-code [11]. A state-machine (also known as finite state-machine (FSM)) memory BIST (MBIST) is generally used in the industry to generate a single MARCH test. However, a better memory test solution requires a set of multiple MARCH tests. This certainly increases the complexity of the MBIST design. A state-machine BIST, as the name implies, uses a number of states, to decode the MARCH test [12]. However, the state-machine MBIST is less flexible as modifying the patterns requires major changes in the MBIST design. As for the micro-code BIST, the test patterns are inserted into the controller in the form of instruction sets. This type of MBIST is highly flexible because different MARCH tests can be utilized. It can also be used in both manufacturing and in a system environment. Both FSM-based and microcode-based MBIST controllers must be able to generate different types of MARCH tests [12]. This can be accomplished by using a programmable architecture. Both types of MBIST: state-machines and micro-code can be designed to be a programmable memory built-in self-test (P-MBIST).

CLUSTERING MARCH OPERATIONS OF MARCH ELEMENTS IN MARCH TESTS

As described in Section II., a MARCH algorithm comprises of MARCH element. This element comprises of a sequence of read/write operations and test data. This sequence is called MARCH operation. Clustering the MARCH operation of MARCH elements in a MARCH test is the operation where one or two read/write operations and test data in a MARCH element are clustered into one major operation. To understand more of how this cluster is created, the MARCH tests is segregated by their elements and operations as shown in Table 1.

Table 1: MARCH Operations in their Clusters

Elements	R/W Operation in the Element			
	Original (before clustering)	**New (after clustering)**		
	No Cluster	**C 1**	**C 2**	**C 3**
e0	w0	w0		
e1	r0,w1	r0,w1		
e2	r1,w0	r1,w0		
e3	r0	r0		
e4	r0,w0,r0,r0,w1,r1	r0,w0,	r0,r0,	w1,r1
e5	r1,w1,r1,r1,w0,r0	r1,w1,	r1,r1,	w0,r0
e6	r0,w1,w0,w1	r0,w1,	w0,w1	
e7	r1,w0,w1,w0	r1,w0,	w1,w0	
e8	r0,w1,w0	r0,w1,	w0	
e9	r1,w0,w1	r1,w0,	w1	
e10	r0,w1,r1,w0,r0,w1	r0,w1,	r1,w0,	r0,w1
e11	r0,w1,r1,w0	r0,w1,	r1,w0	
e12	r1,w0,r0,w1	r1,w0,	r0,w1	
e13	r0,r0,w0,r0,w1	r0,	r0,w0,	r0,w1
e14	r1,r1,w1,r1,w0	r1,	r1,w1,	r1,w0

It can be seen that there are 15 distinct MARCH elements which are used repetitively in the MARCH tests. The read/write operations and test data of the MARCH elements are divided in two sub-columns; **original** and **new**. The **original** column represents the original read/write operations before the cluster method is applied while the **new** column embodies the read/write operation after the cluster method is applied. The **new** column is further split into numbers of sub-columns depend on the numbers of clusters generated. From the original read/write MARCH operations, it can be seen that the *e4*, *e5* and *e10* have the maximum numbers of MARCH operations which are six. The *e13* and *e14* have five MARCH operations while the *e6*, *e7*, *e11* and *e12* have four MARCH operations. Both *e8* and *e9* consist of three MARCH operations. For *e1* and *e2*, they both share two MARCH operations. The *e0* and *e3* has the minimum number of MARCH operation which is only one operation.

The numbers of clusters for all fifteen MARCH elements are determined by the numbers of MARCH operations in an element. For example, six MARCH operations in *e10* generate three clusters where each cluster comprises of two MARCH operation. After every two MARCH operations in a MARCH element were grouped, any single MARCH operation left can be grouped as one cluster. Let's take *e8* and *e9* as examples. Both elements have three MARCH operations. The first two MARCH operations in these elements are clustered in one cluster and the last single operation is also clustered as one cluster. The read/write operation in a cluster is then translated into the microcode as shown in Table 2.

Table 2: MARCH Operation and their Microcodes

Elements	R/W Operation in the Element		
	Cluster 1	**Cluster 2**	**Cluster 3**
e0	0000	-	-
e1	1001	-	-
e2	1100	-	-
e3	1010	-	-
e4	1000	1010	0111
e5	1101	1111	0010
e6	1001	0001	-
e7	1100	0100	-
e8	1001	0000	-
e9	1100	0101	-
e10	1001	1100	1001
e11	1001	1100	-
e12	1100	1001	-
e13	1010	1000	1001
e14	1111	1101	1100

The 4-bit microcode represents two read/write operations in an element (e.g. *1001* and *1100* in *e1* and *e2* respectively). The odd bits of microcode represent the read/write operation where '*1*' indicates read and '*0*' signifies write. The even bits represent the test data where '*0*' indicates data of value zeros and '*1*' signifies data of value ones. The *e6*, *e7*, *e8*, *e9*, *e11* and *e12* generate two clusters of 4-bits microcode because these elements have three or four read/write operations. For *e4*, *e5*, *e10*, *e13* and *e14*, they produce three clusters of 4-bits microcode as they have five or six read/write operations.

The *e0* and *e3* have only a single operation and are supposed to be assigned as 2-bit microcode. However, these single operations are repeated right after the 2-bit microcode of the original single operations because the clustering technique requires two MARCH operation to be clustered in a cluster. The redundancy causes the single operations in the *e0* and *e3* to be assigned as 4-bit microcode. The single operations in the last cluster of the MARCH elements with three or five operations such as *e8*, *e9*, *e13* and *e14* are also subjected to this redundancy. However, this redundancy resulted in longer test time. Furthermore, the microcode in the clusters only applicable for only two kinds of DBS: zeros and ones. Thus, it is not suitable for complex MARCH test algorithm such MARCH SAM-opt which has multiple DBS. To overcome these limitations, these micro-codes in the clusters must be deep-encoded.

3. DEEP-ENCODING THE CLUSTERS FOR MARCH TEST WITH MULTIPLE DBS

Originally, the maximum number of clusters in a MARCH element is three. Each cluster can support up to two MARCH operations. Thus, the maximum numbers of MARCH operations supported by these clusters are six. However, the MARCH SAM-opt algorithm has more than six MARCH operation in an MARCH element. It also has four different test data. These test data are tabulated in Table 3.

Table 3:Test Data for MARCH SAM-opt Test Algorithm

Notation	Test data in two bit form
d0	2'b01
d1	2'b11
d2	2'b10
d3	2'b00

To accommodate the MARCH SAM-opt algorithm into these clusters, the MARCH element for MARCH SAM-opt must be reorganized into few sub-elements. These sub-elements are arranged according to types of the test data. The MARCH operations that share the same test data are grouped into one sub-element. For example, the first element, ↕ *(wd0, rd0, wd0, rd0, rd0, wd1, rd1, wd1, rd1, rd1)* is divided into two sub-elements: *wd0, rd0, wd0, rd0, rd0* and *wd1, rd1, wd1, rd1, rd1*. There should be eight sub-elements derived from all the MARCH elements for MARCH SAM-opt algorithm.
These numbers of sub-elements can be further reduced by applying concept of true-complement data. From Table 4, it is known that *d1* is the complement of the *d3* and *d2* is the complement of the *d0*. Now, the numbers of test data in a MARCH SAM-opt algorithm can be made generic by assigning the test data as true data, *d* and complement data, *d'*. By applying this concept, the numbers of sub-elements in MARCH SAM-opt algorithm are reduced to only four sub-elements. The test data of value *d0* and *d3* in Table 3 are now assigned as the true data, *d* whilst the test data of value *d1* and *d2* are assigned as the complement data, *d'*. Table 4 shows the application of this true-complement data concept to the original 15 MARCH elements in Table 1. The extra four MARCH elements: *e15*, *e16*, *e17* and *e18* are originated from the MARCH SAM-opt algorithm.

These read/write operations and test data are deep-encoded to avoid redundancy and to accommodate the MARCH test with multiple test data. The new 4-bit microcode for these read/write operations and test data is named as deep-code. Table 5 tabulates the deep-codes for all the MARCH operations in the clusters. It can be seen for the table that the double true operations are symbolized as *D3*, *D2*, *D1* and *D0*. The single true operations are symbolized by the *S3*, *S2*, *S1* and *S0*. The single operations are divided into the repeated single operation (*S3* and *S2*) and non-repeated single operation (*S1* and *S0*). All the double and single operations have their complements. These complements are symbolized as same as the true operations except that a bar is added at the top of the symbols.

Table 4: MARCH Operations using True-Complement Test Data Concept

True Cluster						**Complement Cluster**					
	Op	**Deep-codes[3:0]**					**Op**	**Deep-codes[3:0]**			
		3	**2**	**1**	**0**			**3**	**2**	**1**	**0**
D3	wdrd	1	0	0	0	$\overline{D3}$	wd'rd'	1	1	1	0
D2	rdwd	1	0	0	1	$\overline{D2}$	rd'wd'	1	1	1	1
D1	wdwd'	1	0	1	0	$\overline{D1}$	wd'wd	1	1	0	0
D0	rdwd'	1	0	1	1	$\overline{D0}$	rd'wd	1	1	0	1
S3	wdwd	0	0	0	0	$\overline{S3}$	wd'wd'	0	1	1	0
S2	rdrd	0	0	0	1	$\overline{S2}$	rd'rd'	0	1	1	1
S1	wd	0	0	1	0	$\overline{S1}$	wd'	0	1	0	0
S0	rd	0	0	1	1	$\overline{S0}$	rd'	0	1	0	1

Table 5: MARCH Operations and Their Deep-Codes

Elements	R/W Operation in the Element			
	Original (before clustering)	New (after clustering)		
	No Cluster	C 1	C 2	C 3
e0	Wd	wd		
e1	rd,wd’	rd,wd’		
e2	rd’,wd	rd’,wd		
e3	Rd	rd		
e4	rd,wd,rd,rd, wd’,rd’	rd,wd,	rd,rd,	wd’,rd’
e5	rd’,wd’,rd’,rd’,wd,rd	rd’,wd’,	rd’,rd’,	wd,rd
e6	rd,wd’,wd, wd’	rd,wd’,	wd,wd’	
e7	rd’,wd,wd’, wd	rd’,wd,	wd’,wd	
e8	rd,wd’,wd	rd,wd’,	wd	
e9	rd’,wd,wd’	rd’,wd,	wd’	
e10	rd,wd’,rd’, wd,rd,wd’	rd,wd’,	rd’,wd,	rd,wd’
e11	rd,wd’,rd’, wd	rd,wd’,	rd’,wd	
e12	rd’,wd,rd,wd’	rd’,wd,	rd,wd’	
e13	rd,rd,wd,rd, wd’	rd,	rd,wd,	rd,wd’
e14	rd’,rd’,wd’,rd’,wd	rd’,	rd’,wd’,	rd’,wd
e15	wd,rd,wd,rd,rd	wd,rd	wd,rd	rd
e16	wd’,rd’,wd’,rd’,rd’	wd’,rd’	wd’,rd’	rd’
e17	wd,rd	wd,rd		
e18	wd’,rd’	wd’rd’		

The bit 3 of the deep-codes is high to signify a double operation or low to signify a single operation. The read/write enable is activated by the bit 0 of the deep-codes. The read operation takes place if the bit 0 is high and if otherwise, the write operation is performed. The bit 2 and bit 1 of deep-codes represent the operation class. This operation class is mainly used to determine the test data for the operations in the clusters. It is also utilized to classify the repeated and non-repeated single operations and decode the correct read/write operations. Table 6 shows the operations in the clusters in the form the symbols that are introduced in Table 5. These symbols are used to simplify the process of determining what types of operations in a cluster.

Table 6: MARCH Operations using Symbols for Deep-Codes

Elements	R/W Operation in the Element			
	Original (before clustering)	New (after clustering)		
		C1	C 2	C 3
e0	wd	S1	-	-
e1	rd,wd’	D0	-	-
e2	rd’,wd	$\overline{D0}$	-	-
e3	Rd	S0	-	-
e4	rd,wd,rd,rd,wd’, rd’	D2	S2	$\overline{D3}$
e5	rd’,wd’,rd’,rd’, wd,rd	$\overline{D2}$	$\overline{S2}$	D3
e6	rd,wd’,wd,wd’	D0	D1	-
e7	rd’,wd,wd’,wd	$\overline{D0}$	$\overline{D1}$	-
e8	rd,wd’,wd	D0	S1	-
e9	rd’,wd,wd’	$\overline{D0}$	$\overline{S1}$	-
e10	rd,wd’,rd’,wd,rd,wd’	D0	$\overline{D0}$	D0
e11	rd,wd’,rd’,wd	D0	$\overline{D0}$	-
e12	rd’,wd,rd,wd’	$\overline{D0}$	D0	-
e13	rd,rd,wd,rd,wd’	S0	D2	D0
e14	rd’,rd’,wd’,rd’, wd	$\overline{S0}$	$\overline{D2}$	$\overline{D0}$
e15	wd,rd,wd,rd,rd	D3	D3	S0
e16	wd’,rd’,wd’,rd’, rd’	$\overline{D3}$	$\overline{D3}$	$\overline{S0}$
e17	wd,rd	D3	-	-
e18	wd’,rd’	$\overline{D3}$	-	-

The deep-codes of the read/write operation and test data for the MARCH test with multiple test data are now complete. In the following section, the P-MBIST architectures that apply these deep-codes are designed.

4. PROPOSED P-MBIST ARCHITECTURES FOR THE UTILIZATION OF THE DEEP-CODES

The FSM-based and microcode-based P-MBIST controllers are proposed for the utilization of the deep-codes. The proposed deep-encoded FSM-based P-MBIST also utilizes the same macro-commands [13,14] for selecting the test algorithm as the ones used in the clustered FSM-based P-MBIST controller. As for the deep-encoded microcode-based P-MBIST, the numbers of instruction bits for the read/write operations and test data are still set to be fixed regardless of how many the MARCH operations are in a MARCH element. This is different from the microcode-based P-MBIST controller in [15,16] where their length of micro-codes are varied with the numbers of MARCH operation in a MARCH element.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.1. Deep-Encoded FSM-based P-MBIST Architecture

The deep-encoded FSM-based P-MBIST controller has five component blocks: *def_algel*, *def_addec*, *def_elecl*, *def_opsdt* and *rtb.* Figure 1 portrays the block diagram of the controller. The *def-algel* block encodes the MARCH element into 12-bit deep-codes. The *def_addec* block generates the address for the RAM under test (*RUT*) based on the addressing order provided by *def_algel* block. The *def_elecl* block is utilized to segregate the 12-bit deep-codes into three clusters of 4-bit deep-codes. The *def_opsdt* block is employed to decode the read/write operation and test data according to the deep-codes. The *rtb* block is used to compare the injected data and the acquired data from the RAM under test *RUT.*

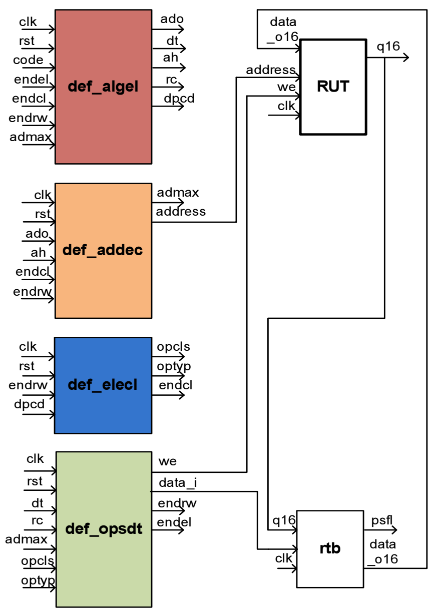

Figure 1:. Block diagram of the proposed deep-encoded FSM-based P-MBIST.

The *def_algel* block comprises of *clk*, *rst*, *code*, *endel*, *endcl*, *endrw* and *admax* as the input signals and *ado*, *dt*, *ah*, *rc* and *dpcd* as the output signals. This block starts to receive the 3-bit *code* if the *endel*, *endcl* and *endrw* signals are high at positive edge *clk*. The state machines in this block decode the MARCH elements according to the *code*. The numbers of states in the *def_algel* block depend on the data types used in the MARCH SAM-opt algorithm. There are four different data types repeated twice in the MARCH SAM-opt algorithm as shown in Table 3. Thus, maximum of nine states are used for encoding all nineteen MARCH elements.

Figure 2 shows that the states for the *def_algel* block. The *code 0* represents the MARCH SAM-opt. All nine states; *S0*, *S1*, *S2*, *S3*, *S4*, *S5*, *S6*, *S7* and *S8* are triggered if *code 0* is sent by the ATE. There are four MARCH elements: *e16*, *e17, e17* and *e18* (refer Table 4.2) associated to the *code 0.* Each state encodes the MARCH element and addressing order as a 12-bit deep-code, *dpcd* and 1-bit signal, *ado.* Unlike the *cf_algel* block, there are three new signals are decoded in these states. These signals are the test data type, *dt*, address hold, *ah* and repeat cluster, *rc*. The *dt* signal determines the types of the test data. The *ah* signal is utilized to hold the address for the sub-elements in the main MARCH elements of the MARCH SAM-opt test algorithm. The *rc* signal distinguishes between the MARCH algorithms with only one data type or multiple data types.

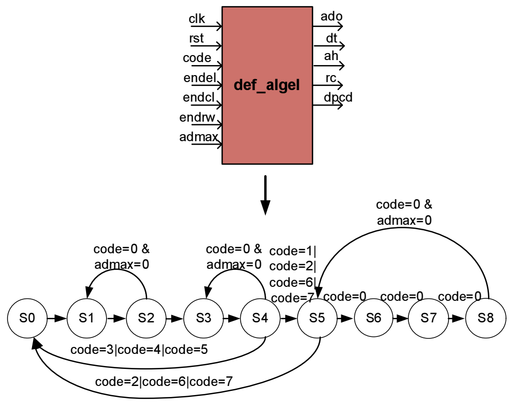

Figure 2: Nine states for encoding MARCH elements

The address decoder is triggered by the *clk*, *rst*, *endcl*, *endrw*, *ah* and *ado* signals. The output signals for this block are *admax* and *address*. The *address* signal signifies the generated address. The addresses are generated when the *ah* signal is low and the *endcl* and *endrw* are high at the positive edge of clk. The sequences of generated addresses depend on ascending or descending addressing order of the MARCH element under test. The active *admax* signal means the MARCH element under test reaches the last address. However, in a case of the MARCH SAM-opt test algorithm, the *admax* signal works together with the *rc* signal to generate new starting address for all the MARCH elements.

The *def_elecl* block consists of four input signals; *clk*, *rst*, *endrw* and *dpcd* and three output signals;

opcls, *optyp* and *endcl*. This block receives the 12-bit deep-codes, *dpcd* from the *def_algel* block once the *endrw* signal is high at positive edge of *clk*. The *def_elecl* block plays the key role in the deep-encoding method because the state-machine in this block is used to segregate the 12-bit deep-codes into three clusters of 2-bit *opcls* and 2-bit *optyp* signals. This block produces the *endcl* signal if the 2-bit *opcls* and 2-bit *optyp* signals are the last operations for the MARCH element under test. Figure 3 shows the state-machines of the *def_elecl* block. Three states are used to segregate the 12-bit deep-encoded microcode into three 2-bit *opcls* and 2-bit *optyp* signals.

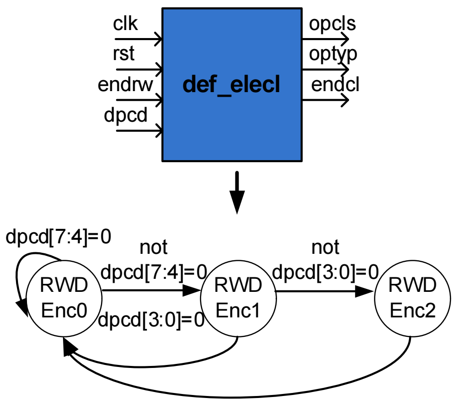

Figure 3: States for assigning deep-codes to clusters

The *def_opsdt* block shares the same clock as the *RUT*. Figure 4 shows the state diagram of the *def_opsdt* block. It has *clk*, *rst*, *dt*, *rc*, *admax*, *opcls* and *optyp* signals as the input signals and *we*, *data_i*, *endrw* and *endel* as the output signals. The *rc* and *admax* signals play important roles in generating the *endel* signal which signifies the next MARCH element in a test algorithm to be tested. The *opcls* signal and the MSB of the *optyp* signal activates the states in the *def_opsdt* block. The actual read/write operations for the MARCH operations in the clusters are mainly decoded by the *optyp* signal with the help of the *opcls* signal. The *opcls* signal also works with the *dt* signal to determine the test data type for the MARCH operations in the clusters. The *we* and *data_i* signals represent the actual read/write operations and test data that are decoded from the *opcls* and *optyp* signals. This block produces the *endrw* signal to signify last decoding of the read/write operation and test data for the MARCH operations in the clusters.

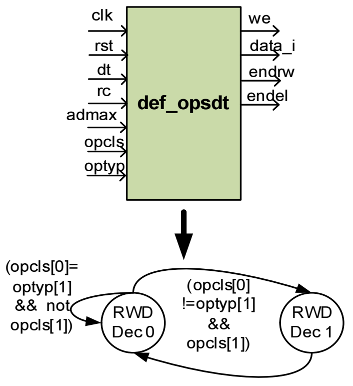

Figure 4: States for decoding operation according to deep-codes

In the *rtb* block, the 2-bit test data, *data_i* signal is replicated according the number of bits of input data of the *RUT*. The replicated *data_i* signal is fed to *RUT* as the *data_o16* signal. This feature allows the BIST controller to test different *RUTs* that share same address width but different data width simultaneously. This replicated *data_i* signal is also registered internally in the *rtb* block to be compared to the output from the RUT, q16. The *psfl* signal is high if the *q16* is not equal to the registered *data_i* signal.

4.2. Deep-Encoded FSM-based P-MBIST Architecture

The complete microcode for one MARCH element can be rearranged as following 17-bit microcode as shown in Figure 5. The bit 16 and bit 15 of the instruction represent the *ado* signal. The bit 14, bit 13 and bit 12 of the instruction signify the *ah*, *rc* and *dt* signals .The remaining 12-bit wide are further divided into three 4-bit wide: bit 11 to bit 8, bit 7 to bit 4 and bit 3 to bit 0. The 4-bit microcode is divided into three parts. The first part is the operation class, *opcls* which is represented by bit 2 and bit 1. The second part is the operation type, *optyp* which is characterized by bit 3 and bit 0.This 4-bit microcode is assigned to each cluster.

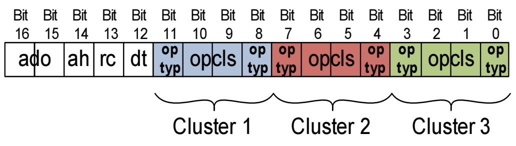

Figure 5: The instruction's microcode for deep-encoded method

The controller comprises of three component blocks; *deu_addec* block to decode the address, *deu_elecl* block to encode the instruction, *deu_opsdt* block to decode the read/write operation and test data. The *rtb* block has the same function as the *rtb* block in the deep-encoded FSM-based P-MBIST controller. Figure 6 portrays the block diagram of the controller.

It can be seen from the figure that the input and output signals for *deu_addec* and *deu_opsdt* blocks are as same as the input and output signals for *def_addec* and *def_opsdt* blocks respectively. This is because both *deu_addec* and *deu_opsdt* blocks function exactly as the *def_addec* and *def_opsdt* blocks. The *rtb* block is the same one that is used in the proposed deep-encoded FSM-based P-MBIST. As for the *deu_elecl* block, it has the same input signals as the *def_elecl* block except the *dpcd* signal is now replaced with the *inst* signal. Its output signals are as same as the *def_elecl* block. The deep-codes are received by this block through the *inst* signal.

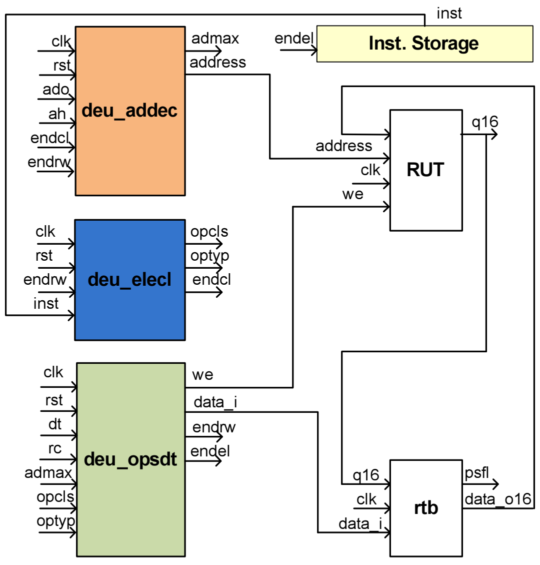

Figure 6: Block diagram of the proposed deep-encoded microcode-based P-MBIST

The bit [11:0] of this *inst* signal is then segregated into three cluster. The bit 11 of the *inst* signal determines whether the operation in the first cluster is a single or double. The bit 10 and bit 9 of the *inst* are the operation class and they are also used to determine the test data for the first cluster. The bit 8 of the *inst* signal determines the read/write operations for the first cluster. The bit 7 of the *inst* signal decides whether the operation in the second cluster is a single or double. The bit 6 and bit 5 of the *inst* signal are the operation class and they are also used to decide the test data for the second cluster. The bit 4 of the *inst* signal decides the read/write operations for the second cluster. The bit 3 of the *inst* signal determines whether the operation in the third cluster is a single or double. The bit 2 and bit 1 of the *inst* signal are the operation class and they are also used to determine the test data for the third cluster. The bit 0 of the *inst* signal determines the read/write operations for the third cluster. This 4-bit microcode in each cluster is represented by the *opcls* and *optyp* signals.

5. EXPERIMENTAL RESULTS

The synthesis of the proposed deep-encoded FSM-based and microcode-based P-MBIST controller is performed using Synopsis DesignCompiler tool. The synthesis library used is the 0.18 um Silterra library. The previously designed FSM-based [13,14] and microcode-based [15] P-MBISTs are also synthesized for comparison purpose. Three factors such as area overhead, timing closure and power consumption are analyzed from the synthesis result of the P-MBIST controller. In this research, the main factor to be analyzed is the area overhead of the P-MBIST. However, the timing closure and the power consumption of the BIST controller are also examined to ensure that the low area overhead is achieved without compensating the speed and power of the controller.

5.1. Area Result

The FSM-based P-MBIST controller in [13] has three component blocks: *BIST controller*, *access unit* and *memory interface unit*. The *BIST controller* block is comparable to the *algel* block of the proposed clustered FSM-based P-MBIST controller. This block is used to encode the MARCH elements according to the test algorithms. The *access unit* block is comparable to the *elecl*, *opsdt* and *addec* blocks because this block decodes the read/write operation and test data according to the MARCH elements and also generates the addresses for the MARCH elements. The *memory interface unit* block is comparable to the *rtb* block that is utilized as the response analyzer to compare the memory output and the test data.

As for the FSM-based P-MBIST controller in [14], it has four component blocks: *algorithm generator*, *read/write signal generator*, *address generator* and *comparator*. The *algorithm generator* block is analogous to the *algel* block as it is used to encode the MARCH elements according to the test algorithms. The *read/write signal generator* block is analogous to the *elecl* and *opsdt* blocks because this block decodes the read/write operation and test data according to the MARCH elements. The *address generator* block is analogous to the *addec* block where it is intended to generate the addresses for the MARCH elements. The *comparator* block is analogous to the *comp* block that is used as the response analyzer to compare the memory output and the test data.

Figure 7 shows the area comparison of the component blocks for the FSM-based controllers. The area for the *elecl*, *opsdt* and *addec* are combined because the FSM-based P-MBIST controller in [13] merged these blocks in its *access unit* block. It can be seen from the graph that the proposed deep-encoded FSM-based P-MBIST controller has the lowest area overhead in the *top* and *elecl+opsdt+addec* blocks compared to the other two FSM-based P-MBISTs in [13,14]. The area for the *algel* block for the deep-encoded P-MBIST controller is the highest among all. This is resulted from the complex nature of the newly-added MARCH SAM-opt algorithm. New control signals are also added in the *algel* block to accommodate the complexity of the MARCH SAM-opt algorithm. The area of *rtb* block for the proposed deep-encoded FSM-based P-MBIST is now significantly lower than FSM-based P-MBIST controller in [13]. However, its area is still higher than the area for the *rtb* block of the FSM-based P-MBIST controller in [14].

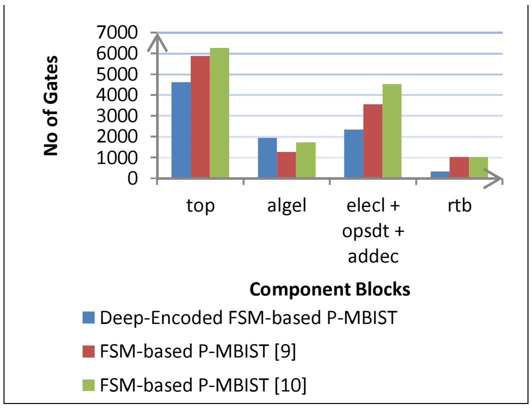

Figure 7: Area Comparison Between FSM-Based P-MBIST Controllers Under Deep-Encoded Method

The microcode-based P-MBIST controller in [15] comprises of *cycle controller*, *instruction logic*, *operation control logic*, *data generation logic*, *address generation logic* and *comparator*. The *cycle controller* comprises of the *cntmux*, *cyclecnt*, *cyclecmp*, *dff1* and *orgate3*. The *instruction logic* comprises of the *insreg*, *addrdclg* and *lg82*. The *operation control logic* comprises of the *opcntreg*. The *data generation logic* comprises of *dtreg*, *xorgt1*, *xorgt2*, *xorgt3* and *xorgt4*. The *address generation logic* comprises of *addrcount*, *lg75*, *orgt2*, *notgt3*, *dff0* and *xorgt0* and the *comparator* comprises of the *psflcomp.* For comparison purpose, the *cycle controller*, *instruction logic*, *operation control logic* and *data generation logic* are analogous to the *elecl* and *opsdt* blocks in the proposed microcode-based P-MBIST controller. The *address generation logic* is analogous to the *addec* block and the *comparator* is analogous to the *comp* block.

Figure 8 shows the area comparison of the component blocks for the microcode-based P-MBIST controllers. It can be seen from the graph that the proposed deep-encoded microcode-based P-MBIST controller has lowest area overhead in all the component blocks except for the *comp* block. The area of the *rtb* block is higher than the area of the *rtb* block for the microcode-based P-MBIST in [15]. The area of the *rtb* block for the microcode-based P-MBIST in [15] is very low because it only contains combinational logics. However, the *rtb* block for the deep-encoded microcode-based P-MBIST contains both non-combinational and combinational logics.

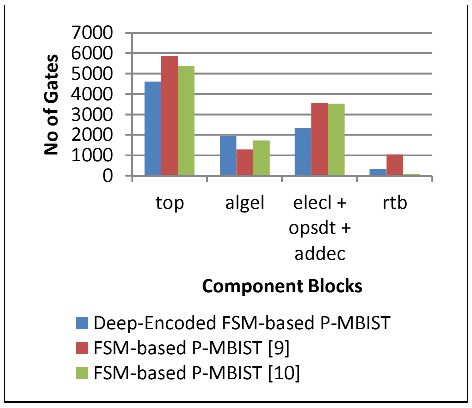

Figure 8: Area Comparison between FSM-based P-MBIST Controllers under Deep-Encoded Method

5.2. Timing Result

The timing analysis is performed by obtaining the time slacks for all three FSM-based P-MBIST controllers at different clock periods. It can be seen from Figure 9 that the time slack for the deep-encoded P-MBIST controller is almost similar to time slack for the FSM-based P-MBIST controller in [13]. The time slacks for the both FSM-based P-MBIST controllers are 12.6 to 4.6 ns from clock periods of 20 ns to 12 ns. As for the FSM-based P-MBIST controller in [14], its slack value is still almost 2ns higher than the deep-encoded P-MBIST controller.

However, when the clock periods go below 10ns, the time slacks for the FSM-based P-MBIST controller in [9] are slightly higher than the deep-encoded FSM-based P-MBIST controller until both of them reach a slack value of 0ns at clock period of 7ns. The slack values for the FSM-based P-MBIST controller in [14] are still significantly higher than the other two controllers when the clock periods go below 10ns. This indicates that the deep-encoded FSM-based P-MBIST controller operates at slower clock period compared to the FSM-based P-MBIST controller in [14].

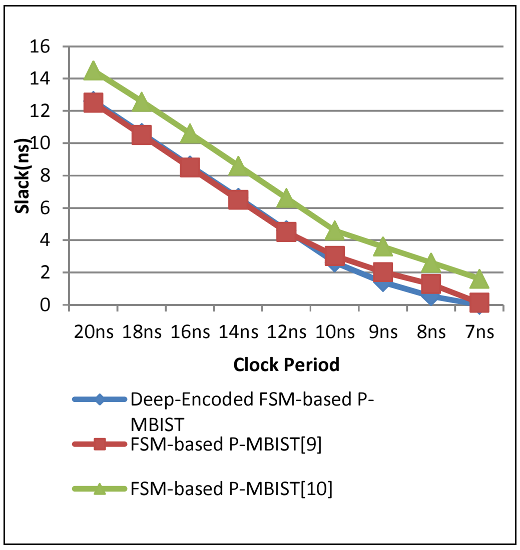

Figure 9: Timing Slack Comparison between FSM-based P-MBIST Controllers under Deep-Encoded Method

As for deep-encoded microcode-based P-MBIST, its timing slack is compared to microcode-based P-MBIST in [15]. Figure 9 shows the time slacks for the proposed deep-encoded microcode-based P-MBIST controller and the microcode-based P-MBIST controller in [15]. From clock periods of 20 ns to 10ns, the time slacks for the proposed deep-encoded microcode-based P-MBIST controller are lower than the time slack for microcode-based P-MBIST controller in [15]. At the clock period of 7 ns, the deep-encoded microcode-based P-MBIST controller settles at much lower slack value than the microcode-based P-MBIST controller in [15]. It is obvious that the slack value for the clustered P-MBIST controller is significantly lower than the microcode-based P-MBIST controller in [15]. This indicates that the deep-encoded microcode-based P-MBIST controller operates at slower clock period compared to the microcode-based P-MBIST controller in [15].

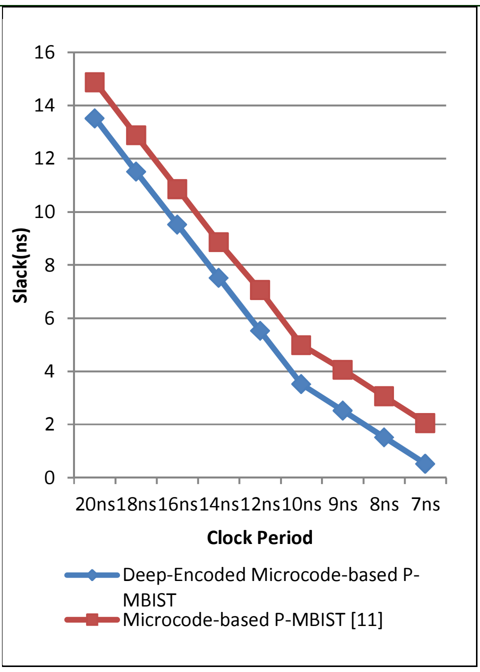

Figure 9: Timing Slack Comparison between microcode-based P-MBIST Controllers under Deep-Encoded Method

5.3. Power Consumption Result

Dynamic power and static power consumption are two types of power that can be evaluated from the Synopsys Design Compiler tool. Dynamic power is the power consumed when the circuit in operating mode. The dynamic power comprises of cell internal power and the net switching power. The cell internal power is the power when only the inputs of the circuit change whilst the net switching power is the power when the both inputs and outputs of the circuit change. Static power is the power consumed when the circuit in idle mode. This power is resulted from the transistors used to built the circuit and is also known as the cell leakage power.

Figure 10 portrays the power consumption for all three FSM-based P-MBIST controllers. It can be seen from the figure that the dynamic power of the proposed deep-encoded FSM-based P-MBIST is lower than the dynamic power for both FSM-based P-MBIST controllers in [13,14]. This indicates that the deep-encoding method is able to reduce the dynamic power of the FSM-based P-MBIST controller compared to the cluster method. In term of static power, the proposed deep-encoded FSM-based P-MBIST controller has the lowest cell leakage power because it has the lowest area overhead.

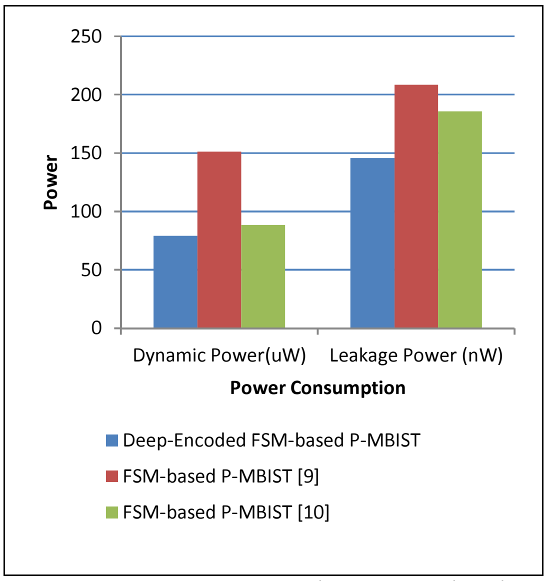

Figure 10: Power Comparison between FSM-based P-MBIST Controllers under Deep-Encoded Method

Figure 11 depicts the power consumption analysis between the proposed deep-encoded microcode-based P-MBIST controller and the microcode-based P-MBIST controller in [15]. It can be seen from the figure that the dynamic and leakage power of the proposed deep-encoded microcode-based P-MBIST are lower than microcode-based P-MBIST in [15].

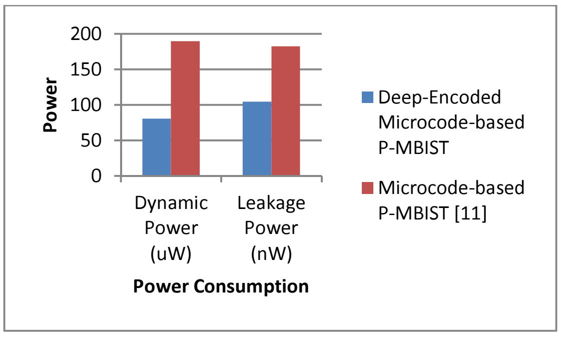

Figure 11: Power Comparison between microcode-based P-MBIST Controllers under Deep-Encoded Method

From these area, slack and power results, it can be concluded that the proposed deep-encoded FSM-based-based P-MBIST controller achieves the lowest area overhead compared to the FSM-based P-MBIST controllers in [13,14]. However, it compensates the speed in order to achieve the lowest area overhead. In term of power consumption, the dynamic and leakage power of the proposed deep-encoded FSM-based P-MBIST controller are improved than the FSM-based P-MBIST controllers in [13,14]. As for the deep-encoded microcode-based P-MBIST controller, it also achieves the lowest area overhead compared to the microcode-based P-MBIST controllers in [15]. But it still compensates the speed in order to achieve lowest area overhead. However, both dynamic and leakage power of the proposed deep-encoded microcode-based P-MBIST controller are improved than microcode-based P-MBIST controller in [15].

6. CONCLUSION AND FUTURE WORKS

The deep-encoding of the read/write operation and test data is developed to distinguish between the single and double operation in the cluster method. Deep-encoding method maintains the division of the MARCH operation of a MARCH element into three clusters but a specific 4-bit deep-code is developed to replace the simple microcode of the read/write operations and test data of a MARCH element. This technique is able to handle the complexity of MARCH SAM-opt algorithm. The application of the deep-encoding method is able to produce low area and robust FSM-based and microcode-based P-MBIST controllers. The synthesis results of these controllers justify that by utilizing the deep-encoding method, the low area overhead are still achieved at optimum performance

REFERENCES

[1] S. Hamdioui, G. Gaydadjiev and A.J. van de Goor, "The state-of art and future trends in testing embedded memories," in *Proceedings of 2004 IEEE International Workshop on Memory Technology, Design and Testing,* 2004, pp. 54-59.

[2] J. Bhadra, M. S. Abadir, D. Burgess and E. Trofimova, "Bottom-up approach in automated embedded memory model generation for high-performance microprocessors," *IEEE Proceedings-Computer and Digital Techniques*, vol. 153, pp. 302-312, 2006

[3] A. Terechko, M. Garg, and H. Corporaal, "Evaluation of speed and area of clustered VLIW processors," in *Proceedings of 18th IEEE International Conference on VLSI Design*, 2005,pp. 557-563.

[4] A. Singh and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field Programmable Gate Array (FPGA 2002),* 2002, pp. 59-66.

[5] A. J. van de Goor and Z. Al-Ars, "Functional Memory Faults: a formal notation and a

taxonomy," in *Proceedings of the 18th* IEEE VLSI Test Symposium (VTEST), 2000, pp. 281-289.

[6] S. Hamdioui, R. Wadsworth, J. Delos Reyes and A.J van de Goor, "Importance of Dynamics Faults for New SRAM Technologies," in *Proceedings of the Eighth IEEE European Test Workshop,* 2003, pp. 29-34.

[7] Laung-Terng Wang, Cheng-Wen Wu and Xiaoging Wen, *VLSI Test Principles and Architecture.* Morgan Kaufmann , Elsevier, 2006.

[8] A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice*. John Wiley & Sons, 1998.

[9] S. Hamdioui, Z. Al Ars, A.J. van de Goor and M. Rodgers, "Dynamic Fault in Random-Access-Memories: Concept, Fault Models and Test," *JOURNAL OF ELECTRONIC TESTING: Theory and Application,* vol. 19, pp. 195-205, 2003.

[10] S. Hamdioui and J.E.Q.D. Reyes, "New data-background sequences and their industrial evaluation for word-oriented random-access memories," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 892-904, 2005.

[11] R. Dean Adams, *High Performance Memory Testing*. Kluwer Academic Publisher, 2003.

[12] Laung-Terng Wang, Cheng-Wen Wu and Xiaoging Wen, *VLSI Test Principles and Architecture.* Morgan Kaufmann , Elsevier, 2006.

[13] Po-Chang Tsai, Sying-Jyan Wang and Feng-Ming Chang, "FSM-based programmable memory BIST with macro-command," in *Proceedings of 2005 IEEE International Workshop on Memory Technology, Design and Testing,* 2005, pp. 72-77.

[14] WonGi Hong, JungDai Choi and Hoon Chang, "A programmable memory BIST for embedded memory," in *Proceedings of IEEE International SoC Design Conference (ISOCC 08)*, 2008, pp. 195-198.

[15] S. Boutobza, M. Nicolaidis, K. M. Lamara and A. Costa, "Programmable Memory Bist" in *Proceedings of IEEE International Test Conference (ITC 2005)*, 2005, pp 1155-1164.

[16] Chung-Fu Lin, Jen-Chieh Ou, Meng-Hsueh Wang, Yu-Sen Ou and Ming-Hsin Ku, "An area-efficient design for Programmable memory Built-In Self-Test," in Proceedings of IEEE *International Symposium on VLSI Design, Automation and Test,* 2008, pp. 17-20.