Optimal Memory Tests Coding for Programmable BIST Architecture

Alexander A. Ivaniuk

***Abstract* — Programmable memory BIST architecture is becoming a necessity for embedded memory cores. Classical memory BIST architectures use fixed algorithmic tests during the whole live of digital device. To improve the flexibility of memory BIST the programmable solution, based on finite state machine with microcode control, was invented. The requirement to use such flexibility is dictated by reason to use newest test for memory cores. In this paper a new Programmable Memory BIST architecture with small microcode memory is proposed. The analysis of existing March tests allows to code them into the optimal binary format, which cause not only small hardware overhead but also may speed-up the transferring of new test over the serial interfaces like IEEE 1149.1 and P1500.**

***Index Terms* — built-in testing, finite state machines, memory testing, microprogramming, random access memories.**

I. Introduction

THE progress in system-on-a-chip (SoC) technology makes possible to integrate huge embedded memory cores into a single chip. The testing of embedded memory cores becomes a problem, since they cannot be controlled from the outside by automatic test equipment (ATE). The built-in self-test (BIST) has become an great alternative for embedded memory cores testing [1]. Memory BIST provides high fault coverage, full speed testing, extensive diagnostics instead of expensive and sophisticated ATE. Traditionally BIST is implemented as additional hardware unit, which is an integral part of memory core. Two alternative memory BIST architectures are exist: hardwired memory BIST and programmable memory BIST (P-MBIST). The hardwired BIST customized for a given memory architecture and a predefined set of algorithmic tests. This type of BIST cause low hardware overhead and provides at-speed memory testing. The major disadvantage of hardwired memory BIST architectures is their poor design and functional flexibility. The programmable memory BIST solution provides a certain degree of flexibility to modify test algorithms at run time. There are several reasons to use different tests for selected memory core during the whole life cycle. At first, the requirements for used memory test may be change. For example different life stages of memory core may need different tests to be applied. At second, different types of embedded memory cores included in a single SoC need a distinct test algorithms.

Big amount of programmable memory BIST architectures have been proposed and developed [2]-[11]. Mostly all of them are based on microprogramming finite state machine (FSM) model. The selected algorithmic memory test is represented as a microprogram, which consists of predefined microcodes. A microprogram needs to be stored in additional memory unit, which content can be modified from the outside. The set of microcodes allows to form optional memory test to be executed by FSM unit.

The flexibility of P-MBIST architecture based on the format and a complete set of microcodes. But at the same time the capacity of this set affects on the hardware complexity.

In this paper, we propose an idea of optimal memory tests coding for programmable memory BIST architecture. The small size of memory test microprogram can reduce the hardware overhead and can speed up the transfer of new tests over the internal serial SoC interfaces.

The remainder of the paper is organized as follows. In section 2, a review of generic programmable memory BIST architecture is included. Section 3 describes original interface of memory module under test. Section 4 includes memory test analysis. Section 5 describes the general functionality of programmable memory BIST hardware. Section 6 concludes the paper.

II. Preliminaries

Let us determine the main constrains for generic P-MBIST architecture. We take bit-oriented random access memory (RAM) unit with organization bits, where is an amount of address lines. The detailed functional description of RAM unit is presented in next section.

Traditionally P-MBIST architecture consists of following hardware units: MUX -- the set of multiplexers or another wrappers, which are used to isolate RAM module under test from external devices; TAP(WSP)-controller provides the serial communication between P-MBIST hardware and external devices and ATE (usually IEEE 1149.1 or P1500

Manuscript received October 28, 2008.

Alexander A. Ivaniuk is with the Department of Software for Information Technologies of the Byelarussian State University of Informatics and Radio-Electronics, Byelorussia, 220027, Minsk, P. Brovki 6, (e-mail: ivaniuk@bsuir.unibel.by).

interfaces are used) [12], [13]; FSM -- is a central core of P-MBIST hardware, which controls all main units and executes the predefined memory test algorithm; MM -- microprogram memory unit, which stores the test in binary format; RI - additional unit, which allows FSM to communicate with RAM under test.

The complete P-MBIST architecture shown in Fig. 1.

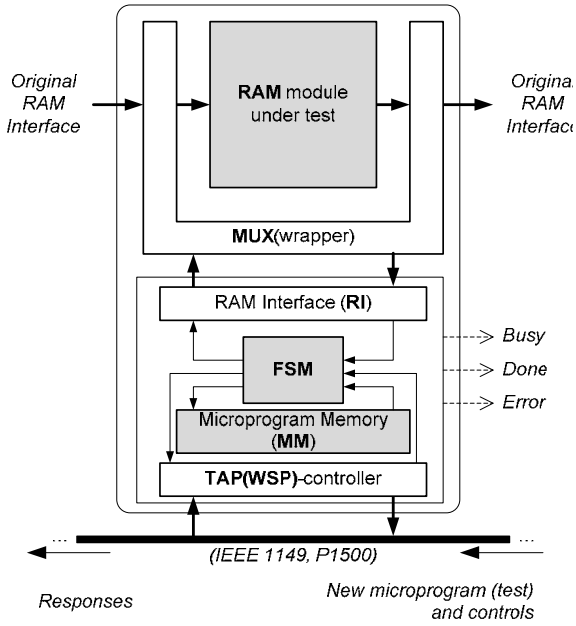

Fig. 1. Generic P-MBIST architecture

Splitting the P-MBIST architecture in several units allows reducing the cost for its extension to new memory cores. During normal operation of RAM module the wrapper schemes are transparent to the original memory interface until the P-MBIST hardware will use it. The self-test procedure of RAM module is initiated outside by external ATE or embedded test core. Specified start instruction or signal is translated over the serial interface, decoded by TAP-controller and allows FSM unit to block original RAM interface. Then FSM starts to fetch the microcode instructions from MM unit, decode and execute them. This allows applying predefined memory test algorithm to the RAM module by controlling the RI unit of P-MBIST architecture. During the whole test execution FSM keeps in active the output *Busy* signal, which can be analyzed by externals cores over additional wire or over the serial test interface. When test procedure is finished FSM generates *Done* signal and appropriate value of *Error* signal.

In this paper we focus on the problem of memory test coding, which binary representation can be effectively store in MM unit. The compact memory test microprogram allows to reduce MM hardware overhead and to speed-up the transfer of new test over the serial interfaces like IEEE 1149.1 and P1500.

III. Original Memory Interface Representation

Let us consider the main ports of RAM module under test: AB - m-lines address bus; DI - data input line; DO - data output line; CS - chip select input line; CLK - synchronization input line; WR - operation type input line (WR=0 denotes write operation, WR=1 - read operation). Let us define a notation for RAM module operations:

$$\{Inputs\} \Rightarrow \{Output\ \_\ Data\}, \quad (1)$$

where *Inputs* denotes a set of logic values or transitions of these values on input ports $\{CS, CLK, AB, DI, WR\}$, which change the logic value on output port $\{DO\}$ (Fig.2).

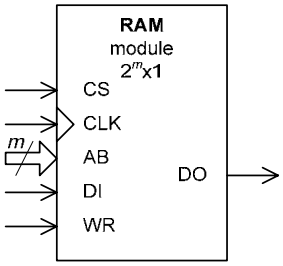

Fig. 2. Entity of RAM module

The RAM module has three main operation modes: storage mode, writing mode and reading mode (Fig.3).

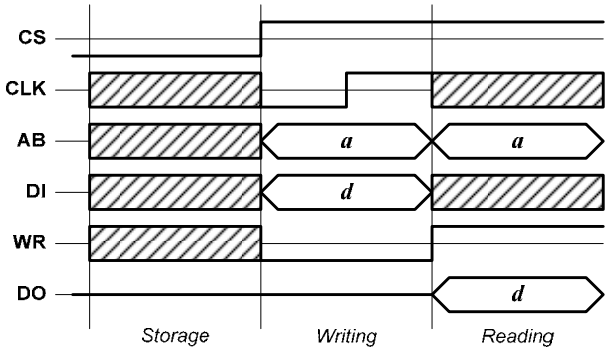

Fig. 3. Timing diagrams of main RAM functional modes

In the context of previous notation 1 we can determine storage mode as

$$\{0, X, X, X, X\} \Rightarrow \{Z\}, \quad (2)$$

where 0 - inactive value of signal on *CS* line, *X* - arbitrary value of signal, *Z* - high impedance on output port *DO*.

The notation of synchronous write operation of binary value *d* into the memory cell with address *a* looks like

$$\{1,0 \rightarrow 1, a, d, 0\} \Rightarrow \{Z\}, \quad (3)$$

where transition $0 \rightarrow 1$ denotes the rising edge on *CLK* line.

Next equation represents the asynchronous read operation of stored binary value *d* from the memory cell with address *a*:

$$\{1, X, a, X, 1\} \Rightarrow \{d\}. \quad (4)$$

Equations 2, 3, 4 can be used to design the behavior of FSM and RI modules of P-MBIST architecture.

IV. MEMORY TESTS ANALYSIS

Memory test is a set of basic operations performed on a RAM module to determine functionality. There is wide range of functional memory tests [14]. One type of tests that has proven to be practically effective in time and complexity is the March test [15]. Any March test for bit-oriented RAM can be defined by the set of primitives of MTL language [16]:

1) the set of basic operations $r0, r1, w0, w1$, where r means read operation and w - write operation of predefined values 0(1) for the memory cell with current value of address a;

2) march element (test phase) - the concrete finite sequence of basic operations applied for current memory cell: $(r0, w1, r1)$;

3) each march element has addressing order, which denotes the direction of address space transmission: symbol $\Uparrow$ denotes addressing order from 0 to $2^m - 1$, symbol $\Downarrow$ denotes backward addressing order from $2^m - 1$ to 0 and symbol $\Updownarrow$ is used when the addressing order is irrelevant; for example the first march element of all tests looks like $\Updownarrow(w0)$;

4) the finite set of different march elements forms complete march test; for instance, march test *MATS++* can be written as $\{\Updownarrow(w0); \Uparrow(r0, w1); \Downarrow(r1, w0, r0)\}$.

Let us determine the basic operations in terms of notation 1. It is necessary to note that each read operation $r_{dt}(dt \in \{0,1\})$ consists of two micro operations: reading the value d from selected memory cell and comparison the value d with the reference value dt:

$$\{1, X, a, X, 1\} \Rightarrow \{d\}, CMP(d, dt). \quad (5)$$

The basic write operation $w_{dt}(dt \in \{0,1\})$ will be written as

$$\{1,0 \to 1, a, dt, 0\} \Rightarrow \{Z\}. \quad (6)$$

Each basic operation belongs to specified march element. All operations from current march element are performed sequentially for selected memory cell. When last operation from march element will be completed then the current value of address will be changed according to the specified addressing order. For this reason we denote two markers for all basic operations: *AO* - Address Order and *LO* - Last Operation. Also we add these markers to the notation of basic operations 1:

$$[AO, LO]\{Inputs\} \Rightarrow \{Output\_Data\}. \quad (7)$$

Classical March tests are designed to detect different types of faults. All March tests are able to detect single-cell faults of different multiplicity, but not all of them are able to detect single faults, which affect more then one cell [15]. At first case only two types of addressing order can be used: $\Uparrow$ and $\Downarrow$. All three types of addressing order are used by multi run March tests to detect multiple-cell faults, where type $\Updownarrow$ is used to set up different initial backgrounds [17].

In this paper we assume that *AO*=0 denotes two types of addressing order $\Uparrow$ and $\Updownarrow$, when *AO*=1 denotes only one type - $\Downarrow$. The marker *LO*=1 corresponds to the operation, which is the last operation in current march element. Other operations from current march element have got marker *LO*=0. For instance, the first march element of test *MATS++* will be represented as $[0,1]\{1,0 \to 1, a, 0, 0\} \Rightarrow \{Z\}$.

Let us consider the difference between basic operations in the context of their notations. If operations belong to the same march element then all of them have identical marker *AO*. If two read operations belong to the same march element they can differ to following values:

$$[-, LO]\{-,-,-,-,-\} \Rightarrow \{d\}, CMP(d, dt), \quad (8)$$

where symbol '-' means the coincidence of appropriate values. Two write operations, which are belong to the same march element can differ to following values:

$$[-, LO]\{-,-,-, dt, -\} \Rightarrow \{-\}. \quad (9)$$

Two polytypic operations belonging to one march element can be differ to following values:

$$[-, LO]\{-, CLK, -, dt, WR\} \Rightarrow \{DO\}. \quad (10)$$

It is necessary to notice that values *CLK*, *dt*, *WR* and *DO* are identical for one-type operations. For expediency of the subsequent reasoning we will enter a new marker *OT* - Operation Type. Let value *OT*=0 corresponds to all write operations and value *OT*=1 - to all read operations. Let us add marker *OT* to the notation of basic operations. By excluding general values we will get a new form of notation:

$$[AO, LO, OT]\{a, dt\}. \quad (11)$$

Each basic operation of given march test uses all possible values of address $a = (0, \ldots, 2^m - 1)$, which are generated by a part of RI module of P-MBIST architecture. The best way to design RI module is to use the binary up-down counter, which estimates address space bounds (0 or $2^m - 1$). If one of these bound was obtained and current operation has marker *LO*=1 then in means the end of current march element. So, the value of address a can be excluded from notation 11, because the value of a automatically generated by counter, which is controlled by values of two markers *AO* and *LO*. The current value of a will be incremented only if *AO*=0 and *LO*=1 and decremented when *AO*=1 and *LO*=1. In this case the march test operation can be described by next notation:

$$[AO, LO, OT]\{dt\}. \quad (12)$$

For instance, test *MATS++* will be represented as it shown in Table I. The marker *AO* was used only for the first operations of march element.

TABLE I
MATS++ TEST DESCRIPTION

MTL-notation	[AO,	LO,	*OT*]	{*dt*}
$\Updownarrow (w0);$	[0,	1,	0]	{0}
$\Uparrow (r0$	[0,	0,	1]	{0}
$, w1);$	[	1,	0]	{1}
$\Downarrow (r1$	[1,	0,	1]	{1}
$, w0$	[	0,	0]	{0}
$, r0);$	[	1,	1]	{0}

The presented binary format also needs such marker as the end of the test. Any data, which can be added to the end of March test binary format will be decoded in terms of markers *AO*, *LO*, *OT* and *dt*. That is why we propose to use a new marker *NOE* that represents the Number Of march Elements of the test. For instance, *MATS++* has three march elements and the value of *NOE* will be equal to 11 (3 in binary format). In general to code any march test we need *V* bits,

$$V = \lceil \log_2 n_e \rceil + n_e + 3n_o, \quad (13)$$

where n_e is the number of march elements, n_o is the number of operations. For example, test *MATS++* has got next values: $n_e = 3$, $n_o = 6$ and $V = 23$.

Let us show that *dt* marker can be excluded from the march test notation. If the current operation is write operation w_{dt} and next operation is read operation then it uses the same value of *dt*. We describe the given property in the form of the following operator

$$(w \to r) \Rightarrow (dt \to dt), \quad (14)$$

where $w \to r$ denotes the sequence of adjacent operations and $dt \to dt$ means that the value of *dt* will be not changed for these operations.

The analysis of existing March tests has shown that additionally there properties are exist:

$$(r \to w) \Rightarrow (dt \to \overline{dt}), \quad (15)$$

$$(r \to r) \Rightarrow (dt \to dt), \quad (16)$$

$$(w \to w) \Rightarrow (dt \to \overline{dt}), \quad (17)$$

where $\overline{dt}$ means the inverted value of *dt*.

Two properties 14 and 15 are typical for all March tests. Property 16 belongs for tests *Marching 1/0*, *March Y*, *March C* when property 17 belongs only for *March A*, *March B* and for *Algorithm B*.

In view of that fact that all tests begin with operation $w0$ (initial value $dt = 0$) and properties 14-17 cover all possible combinations of two adjacent operations the marker *dt* can be excluded from the notation of March test operations (12):

$$[AO, LO, OT]. \quad (18)$$

The complete Extended Backus-Naur form [18] of proposed representation of March tests looks like

$$\begin{aligned} bd &= 0 \mid 1 \\ NOE &= 3 * bd \\ AO &= bd \\ LO &= bd \\ OT &= bd \\ el &= "[" [AO", "]LO", "OT"]" \\ test &= "(" NOE ")" \{el\{el\}\}, \end{aligned} \quad (19)$$

where nonterminals *test* and *el* represent complete March test and march element respectively.

In a case of new notation test MATS++ will be represented as it shown in Table II.

TABLE II
MODIFIED DESCRIPTION OF MATS++ TEST

MTL-notation	[AO,	LO,	*OT*]	{*dt*}	*NOE*
	[0,	0,	0]	$dt = 0$	3
$\Updownarrow (w0);$	[0,	1,	0]	dt	2
$\Uparrow (r0$	[0,	0,	1]	dt	2
$, w1);$	[	1,	0]	$\overline{dt}$	1
$\Downarrow (r1$	[1,	0,	1]	$\overline{dt}$	1
$, w0$	[	0,	0]	dt	1
$, r0);$	[	1,	1]	dt	0

The value of *dt* is changing according to the current value of marker *OT*. At the same time value of *NOE* is automatically decremented at the end of each march element when *LO*=1 and *EOP*=1. The zero value of *NOE* indicates the end of March test.

In general case to code any march test using new notation 18 we need V' bits,

$$V' = \lceil \log_2 (\max(n_e)) \rceil + n_e + 2n_o, \quad (20)$$

where $\max(n_e) = 7$ for existing March test.

Table III represents values of n_e, n_o and V' for classical March tests.

TABLE III
CHARACTERISTICS OF CLASSICAL MARCH TESTS

Name	n_e	n_o	V'
MATS	3	4	14
MATS+	3	5	14
MATS++	3	6	18
Marching 1/0	6	14	37
March X	4	6	19
March Y	4	8	23
March C	7	11	32
March C-	6	10	29
March A	5	15	38
March B	5	17	42
Algorithm B	5	17	42

The value of $V' = 42$ is enough to store any existing March test in microprogram memory of P-MBIST architecture.

V. The functionality of P-MBIST hardware

Proposed P-MBIST architecture consists of control logic as FSM module, microcode memory unit (MM), RAM interface model (RI) and TAP-controller.

Executable microcode of March test have been stored in MM, which has two ports. The write-port is used to receive binary data from TAP-controller. Another read-port is used to transmit binary data to FSM module.

RI module provides connections of P-MBIST hardware to RAM module under test. It generates sequences of control signals, which allow to produce complete read and write operations for RAM.

Binary counter is the part of RI, which generate the predefined sequences of RAM addresses during March test execution.

FSM module controls the functionality of all units of P-MBIST hardware. The control flow of FSM module is shown in Fig. 4. The FSM module includes next registers: *PC* - program counter, which addresses the binary data in MM module; PC^* - register to store a copy of current *PC* value; *NOE* - register to store number of march elements, *AO*, *LO*, *OT* - 1-bit registers to store March test markers.

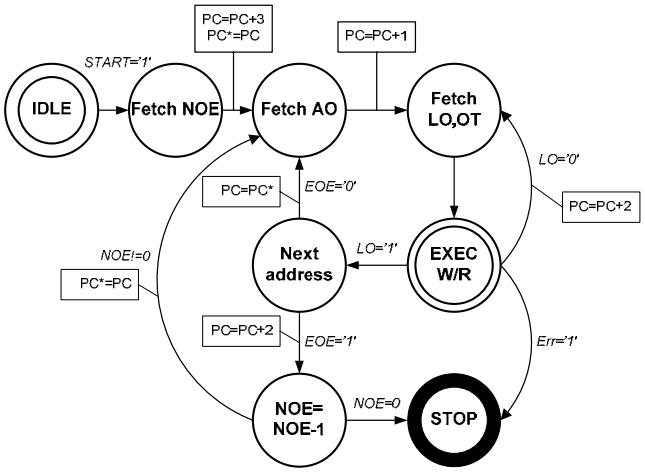

Fig. 4. FSM Control Flow

At the very beginning FSM module is waiting the *START* signal to be active. After *START* signal is rising high the FSM fetches *NOE* value into the counter from MM module. The value of program counter will be modified *PC*=*PC*+3 and stored in PC^* register. In the next state FSM module fetches *AO* marker from microprogram memory and increments the *PC* value. Next state is separated from the previous state to produce basic operation until the end of current march element (*LO*=1). The execution of basic operation holds in *EXEC W/R* state. If read operation fails then FSM generates output error signal *Err*=1 and permanently stalls in *STOP* state. When current march element is finished (marker *LO*=1) FSM turns into the *Next address* state. If the boundary of address space is not reached FSM restores program counter value from *PC** register and goes to *Fetch AO* state. In other case FSM modifies program counter value *PC*=*PC*+2 to be able to fetch *AO* marker of next march element. Then FSM decrements the value of *NOE* and checks it to zero value. If *NOE* equals to zero then FSM stops by turning into the *STOP* state. Otherwise FSM stores *PC* value and starts to fetch another markers of next march element.

The proposed programmable memory BIST architecture was designed for the synchronous single port RAM using Xilinx ISE WebPack 9.1i. The design was verified through the functional simulation using basic March tests. The complete P-MBIST circuit for 1Mb SRAM module (*m*=20) has next hardware overhead estimation: MUX: 11%, RI: 35%, FSM: 37%, MM: 2%, TAP: 15%.

VI. Conclusion

An efficient microcode-based programmable memory BIST architecture is introduced in this paper. The main goal of investigation was to minimize the binary microcode of march tests, which are stored in P-MBIST internal memory. It was shown that the test and reference data can be excluded from the binary representation of march test elements and basic operations. The proposed P-MBIST architecture can be widely used for the self-testing of embedded memory cores, especially under the system on a chip design environment.

References

[1] Y. Zorian and S. Shoukourian, "Embedded-memory test and repair: Infrastructure ip for soc yield," *IEEE Design and Test of Computers*, vol. 20, no. 3, pp. 58–66, May/Jun. 2003.

[2] K. Zarrineh and S. J. Upadhyaya, "On programmable memory built-inself test architectures," in *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE'99)*, Munich, Germany, Mar. 1999, pp. 708–813.

[3] D. Appello, P. Bernardi, A. Fudoli, M. Rebaudengo, M. S. Reorda, V. Tancorre, and M. Violante, "Exploiting programmable BIST for the diagnosis of embedded memory cores," in *Proc. IEEE International Test Conference (ITC'03)*, Charlotte, NC, USA, Sep. 2003, pp. 379–385.

[4] M. Kume, K. Uehara, M. Itakura, H. Sawamoto, T. Kobayashi, M. Hasegawa, and H. Hayashi, "Programmable at-speed array and functional BIST for embedded DRAM LSI," in *Proc. IEEE International Test Conference (ITC'04)*, Charlotte, NC, USA, Oct. 2004, pp. 988–996.

[5] D. Youn, T. Kim, and S. Park, "A microcode-based memory BIST implementing modified march algorithm," in *Proc. 10th Asian Test Symposium (ATS'01)*, Kyoto, Japan, Nov. 2001, pp. 391–395.

[6] X. Du, N. Mukherjee, C. Hill, W.-T. Cheng, and S. Reddy, "A field programmable memory BIST architecture supporting algorithms with multiple nested loops," in *Proc. 15th Asian Test Symposium (ATS'06)*, Fukuoka, Japan, Nov. 2006, pp. 287–292.

[7] P.-C. Tsai, S.-J. Wang, and F.-M. Chang, "FSM-based programmable memory BIST with macro command," in *Proc. IEEE International*

Workshop on Memory Technology, Design, and Testing (MTDT'05), Taipei, Taiwan, Aug. 2005, pp. 72–77.

[8] X. Du, N. Mukherjee, W.-T. Cheng, and S. Reddy, "Full-speed fieldprogrammable memory BIST architecture," in *Proc. IEEE International Test Conference (ITC'05)*, Austin, TX, USA, Nov. 2005, pp. 1165–1173.

[9] K. Zarrineh and S. J. Upadhyaya, "Programmable memory BIST and a new synthesis framework," in *Proc. IEEE The 29th Annual International Symposium on Fault-Tolerant Computing (FTCS'99)*, Madison, Wisconsin, USA, Jun. 1999, pp. 352–355.

[10] P. McEvoy and R. Farrell, "Built-in test engine for memory test," in *Proc. IEEE IC Test Workshop (ICTW'04)*, Limerick, Ireland, Sep. 2004, pp. 15–21.

[11] M. Zhang, D. Tao, and B. Wei, "A programmable BIST for embedded SDRAM," in *Proc. IEEE International Symposium on VLSI Technology, Systems, and Applications(VTSA'01)*, Hsinchu, Taiwan, Apr. 2001, pp. 244–248.

[12] L.-T. Wang, C. E. Stroud, and N. A. Touba, *System-on-Chip Test Architectures*. Burlington, MA, USA: Morgan Kaufmann, 2007.

[13] P. Bernardi, M. Rebaudengo, M. S. Reorda, and M. Violante, "A 1500- compatible programmable BIST approach for the test of embedded flash memories," in *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE'03)*, Munich, Germany, Mar. 2003, pp. 720–725.

[14] Z. Al-Ars, S. Hamdioui, and A. J. van de Goor, "A fault primitive based analysis of linked faults in RAMs," in *Proc. 11th IEEE International Workshop on Memory Technology, Design, and Testing (MTDT'03)*, an Jose, CA, USA, Jul. 2003, pp. 33–39.

[15] A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice*. Chichester, UK: John Wiley & Sons Inc., 1991.

[16] A. J. van de Goor, A. Offerman, and H. Schanstra, "Towards a uniform notation for memory tests," in *Proc. IEEE European Design and Test Conference (EDTC'97)*, Paris, France, Mar. 1996, pp. 420–427.

[17] S. Yarmolik and V. Yarmolik, "Memory address generation for multiple run march tests with different average hamming distance," in *Proc. IEEE East-West Design & Test Workshop (EWDTW'06)*, Sochi, Russia, Sep. 2006, pp. 212–216.

[18] *International standard ISO/EIC 14977, Information Technology – Syntactic metalanguage – Extended BNF*, ISO/EIC First edition, Rev. 1996- 12-15, 1996.

Alexander A. Ivaniuk received the MSc degree in 1995 from the Computer Science Department of the Belarussian State University of Informatics and Radio-Electronics in Minsk, Belarus. At the same university, he completed his PhD thesis on Methods and Tools for Concurrent RAM Testing and Checking and was awarded the PhD degree in 1999. From 1999 to 2002 he has been an assistant professor. Since 2002 he became an associated professor at the Department of Software for Information Technologies.