JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, 155–168, 2004

A Low-Cost At-Speed BIST Architecture for Embedded Processor and SRAM Cores

M.H. TEHRANIPOUR
Center for Integrated Circuits & Systems, Department of EE, The University of Texas at Dallas, Richardson, TX 75083,USA
mht021000@utdallas.edu

S.M. FAKHRAIE, Z. NAVABI AND M.R. MOVAHEDIN
VLSI Circuits & Systems Laboratory, Department of ECE, The University of Tehran, Tehran 14399, Iran
fakhraie@ut.ac.ir
navabi@ece.neu.edu
mov@dr.com

Received July 6, 2001; Revised June 30, 2003

Editor: Y. Zorian

Abstract. We have introduced a low-cost at-speed BIST architecture that enables conventional microprocessors and DSP cores to test their functional blocks and embedded SRAMs in system-on-a-chip architectures using their existing hardware and software resources. To accommodate our proposed new test methodology, minor modifications should be applied to base processor within its test phase. That is, we modify the controller to interpret some of the instructions differently only within the initial test mode. In this paper, we have proposed a fuctional self-test methodology that is deterministic in nature. In our proposed architecture, a self test program called *BIST Program* is stored in an embedded ROM as a vehicle for applying tests. We first start with testing processor core using our proposed architedture. Once the testing of the processor core is completed, this core is used to test the embedded SRAMs. A test algorithm which utilizes a mixture of existing memory testing techniques and covers all important memory faults is presented in this paper. The proposed memory test algorithm covers 100% of the faults under the fault model plus a data retention test. The hardware overhead in the proposed architecture is shown to be negligible. This architecture is implemented on UTS-DSP (University of Tehran and Iran Communicaton Industries (SAMA)) IC which has been designed in VLSI Circuits and Systems Laboratory.

Keywords: BIST architecture, DSP/microprocessor, UTS-DSP, bit/word-oriented memory, memory testing, march test

1. Introduction

The ASIC industry, driven by ever increasing demands for miniaturization, higher reliability, and greater speeds, continuously introduces new product innovations to the microelectronics market. As a result, the density of semiconductor μP/DSP chips has been increased dramatically. The manufacturing yield of silicon products strongly depends on the silicon area, and their performance is directly related to the delays on the critical paths. Thus, in order to build reliable and competitive products, it is essential that the testing strategy provides high fault coverage, without increasing a significant area overhead and degrading the performance. With the increasing complexity, it has been recognized that the testing of such μP/DSPs is a difficult problem, and the test time and cost for such chips are very large. To overcome the problem of large test time and cost, self-test methods have been developed.

One of the most widely researched self-testing techniques is Built-In Self-Test (BIST), which uses embedded hardware test generators and test response analyzers to generate and apply test patterns on-chip at the speed of the circuit, thereby eliminating the need for an external tester [1, 3]. Several self-test methodologies for testing microprocessors have been presented in recent years. A native mode functional test generation for processors is proposed in [20]. The generated test program can be applied to both of design validation and manufacturing test. The built-in self-test method presented in [2] combines the execution of microprocessor instructions randomly with on-chip test hardware. The authors in [5] proposed a partially-automated test program generation to test a processor core. The method examines all processor instructions. An instruction-based self-test methodology is proposed in [11] to test embedded processor cores in a system-chip based on the knowledge of its instruction set architecture and register transfer level description. The authors in [4] presented a software-based self-testing methodology for processor cores which uses a software tester embedded in the processor memory as a vehicle for applying structural tests. The software tester consists of programs for test generation and test application. A low-cost BIST architecture based on modifying instructions in the test mode is proposed in [21]. A full functional testing of a processor core in a system-chip is presented in [16]. The proposed method sends instructions serially and exercises them with a huge number of executions, which is very time consuming.

Several self-test methodologies have been reported in literature for testing DSP cores [10, 15, 18, 27]. DSP cores mostly include larger arithmetic units such as shifters and Muliply-Acumulators, compared to general processors. A built-in self-test method is implemented in a 24 bit floating point digital signal processor using pseudo-random patterns [18]. The number of random test patterns applied to the DSP under test is very large. Reference [10] presents a full scan with a dual phase level sensitive scan design (LSSD) to be implemented on a DSP core. An arithmetic BIST is proposed in [15] in which all generation and compaction functions are executed by basic building blocks such as adders, ALUs, and multipliers. Testing datapath of DSP cores based on self-test programs is proposed in [27]. During the test random patterns are loaded into the core, exercise different components of the core, and then are loaded out of the core for observation under the control of self-test program.

Since the need for self testing is most acute for high performance processors, we propose a self testing program which is stored in embedded ROM and can be used for applying the needed tests (Deterministic or random). The hardware overhead in the proposed architecture is shown to be negligible. To circumvent the low fault coverage associated with random pattern testing of processors, in this approach, we first determine the structural test requirements of processor components, which are usually much less complex than the full processor, and hence more amenable to random pattern testing. At the processor level, the instructions of the processor are used to apply the tests to each component at the speed of the processor.

Memory is an important part of a system chip. Many RAM test algorithms based on different fault models have been proposed during past years [6, 7, 12, 22]. With increasing complexity, the efficient testing of such memories has been recognized as a difficult problem. Almost all memories today tend to use BIST methodology. Hence, core providers of embedded memory typically incorporate BIST wrappers in the memory core design [28]. In this paper, we propose an efficient BIST method for testing the embedded SRAM, and show the feasibilty of fault model and test algorithm. In this method, with increasing the size of RAMs, only the test time is increased without any increasing hardware overhead [22].

UTS-DSP includes complicated functional blocks such as a fixed-point Arithmetic Logic Unit (ALU), a Multiplier/Accumulator unit (MAC), Shifter, Compare-Select-Store Unit (CSSU) and etc. [17]. With such large units it becomes immensely time-consuming to obtain test patterns having high fault coverage. The proposed BIST architecture reduces BIST area, test time and cost significantly. This BIST architecture is implemented on UTS-DSP IC which has been designed in VLSI Circuits and Systems Laboratory.

This paper is organized as follows. Section 2 describes an overview of the UTS-DSP. Section 3 discusses the testing of microprocessor and DSP cores and provides a testable design for a general processor core. The embedded SRAM testing is decribed in

Section 4. Section 5 describes practical results from implementation of BIST on the UTS-DSP. The paper ends with conclusions in Section 6.

2. The UTS-DSP Overview

Since our case study is performed on the UTS-DSP, in this section we briefly describe the UTS-DSP structure. UTS-DSP core is compatible at instruction set level with TI's TMS320C54x [23, 24] DSP processor family, which has CISC architecture [9]. The UTS-DSP has been designed in VLSI Circuits and Systems Laboratory [8, 17]. The UTS-DSP is a fixed-point digital signal processor (DSP). The UTS-DSP central processing unit (CPU), with its modified Harvard architecture, features minimized power consumption and a high degree of parallelism. Also, the versatile addressing modes and instruction set improve the overall system performance. The UTS-DSP block diagram is shown in Fig. 1.

Instruction sets and addressing modes have great influence on core design process, especially in pipeline structures [17, 23]. There are 187 instructions in the instruction set of the UTS-DSP core, which are compatible with the TI's TMS320C54x family of the DSP processors. Instructions are grouped in sub-categories to simplify the modeling of each stage of the pipeline. By grouping the instructions, each stage of the pipeline is designed to perform similar operations on all members of a group, not for each instruction individually. Having 7 addressing modes, UTS-DSP core can perform many required operations as a DSP processor. These addressing modes are as follows: (1) Immediate addressing, (2) Absolute addressing, (3) Accumulator addressing, (4) Direct addressing, (5) Indirect addressing, (6) Memory-mapped registers addressing, and (7) Stack addressing. UTS-DSP core consists of a six-stage pipeline. These stages are prefetch, fetch, decode, access, read and execute stages, which shortly will be described further. Then an overview of the suggested CPU architecture, which comprises from ALU, Shifter, and Multiplier/adder units is provided. We have designed an emulator board to verify our design at structural level. This board consists of one Flex10K250 and one Flex10K100 FPGA ICs. It also contains four memory banks to implement parallel RAM and ROM blocks of UTS-DSP.

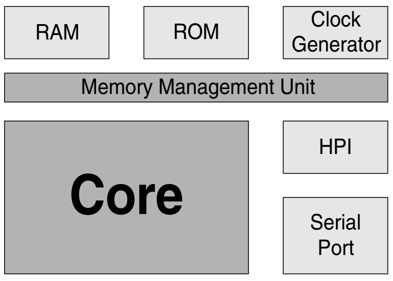

Fig. 1. The UTS-DSP block diagram.

As shown in Fig. 1, the core consists of the pipeline stages and CPU. Some other important blocks shown are internal RAM and ROM, memory management unit, HPI (for interfacing the processor to a host), different possible serial ports, and clock generator unit [13, 17, 19].

2.1. CPU Structure

This section provides an overview of the suggested CPU architecture, which comprises from ALU, Shifter, and Multiplier/Accumulator (MAC) units. The CPU can perform high-speed arithmetic operations within one instruction cycle due to its parallel and combinational architectural design. Fig. 2 is a functional block diagram of the proposed architecture, which includes the principal blocks and required input/output registers [8, 17].

2.1.1. Arithmetic and Logic Unit (ALU). The 40-bit ALU implements a wide range of arithmetic, logical, and rotate operations [8]. The result is transferred to a destination accumulator (A or B) or sent to the output bus for memory writing. There are two input busses to read dual words of memory in a cycle and write the result back to the memory. The ALU contains two separate adders to be able to perform dual 16-bit operations. The ALU can operate in a special dual 16-bit arithmetic mode that performs two 16-bit operations (for instance, two additions or two subtractions) in one cycle.

2.1.2. Shifter Unit. To maintain accuracy without dealing with complexity of a floating-point data path, fixed-point DSP processors have a good support for shifting operations. To manipulate a wide range of left and right shifts with an acceptable delay time, we use a barrel shifter with range of -16 to 31 shift count value [8]. The 40-bit shifter is fed through one of the two 40-bit accumulators, D_data and 16-bit left shifted C_data for a 16-bit data input operand, and

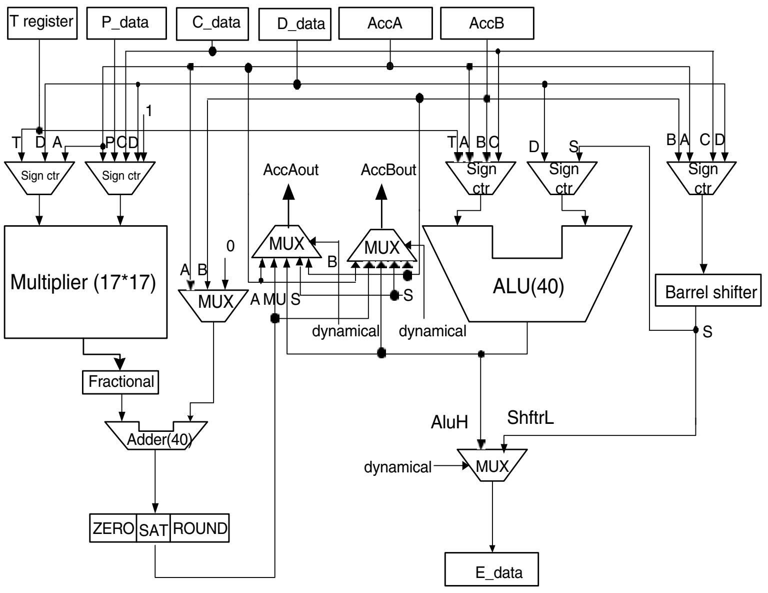

Fig. 2. Block diagram of CPU architecture [7].

concatenated C_data and D_data for a 32-bit data input operand. The output is connected to either one of the ALU inputs or the E_data bus to be sent to memory.

2.1.3. Multiplier and Accumulator Unit (MAC). The CPU architecture has a 17-bit $\times$ 17-bit hardware multiplier coupled to a 40-bit dedicated adder. This multiplier/accumulator provides multiply and accumulate (MAC) capability in one cycle. The multiplier can perform signed, unsigned, and signed/unsigned multiplication [8]. The multiplier output can be shifted left by one bit to compensate for the extra sign bit generated by multiplying two 16-bit 2s-complement numbers in fractional mode. The adder's inputs come from the multiplier's output and from one of the accumulators. Once any multiply operation is performed in the unit, the result is transferred to a destination accumulator (A or B). To maintain efficiency of the structure, instead of using adder/subtractor hardware, we have accomplished a multiplier capable of producing $X \times Y$ and $-X \times Y$ without any overhead.

3. Testing of Microprocessor and DSP Cores

Several BIST architectures have been presented for testing μP and DSP cores. Area, performance and cost overheads are important factors for BIST implementation. This section describes a BIST architecture for testing μP and DSP cores. Implementation of one of the BIST architectures on the general processors which has been presented previously [16], is shown in Fig. 3. As shown, a Test Control Register (TCR) is used for transferring opcodes from scan-in pin to instruction decoder logic. LFSR and MISR (LFSR for the pseudo-random pattern generator, MISR for signature analyzer) are used as test generator and signature analyzer. BIST controller generates clock and control signals. When the processor is entered into the test mode, the test opcodes (instructions opcodes) are transmitted into the instruction decoder logic through TCR and MUX. The instruction decoder logic transfers control signals for executing of input instructions. LFSR is used when test operand is required and MISR is used when the result of an operation is provided. The scan-in, scan-out, clock and control, and test mode pins are added to the microprocessor. In this architecture, BIST controller performs the following control operations: transmitting instruction opcodes to the microprocessor through scan-in pin, receiving serial output of MISR from scan-out pin, comparing outputs of scan-out with expected results, and transmiting clock and control signals. This method is very time consuming because the number of execution cycles for each instruction is huge.

BIST controller transmits a variety of instructions to processor. For reducing the number of instructions, the input instructions should be applied in a program form to scan-in pin in order to test each block within the

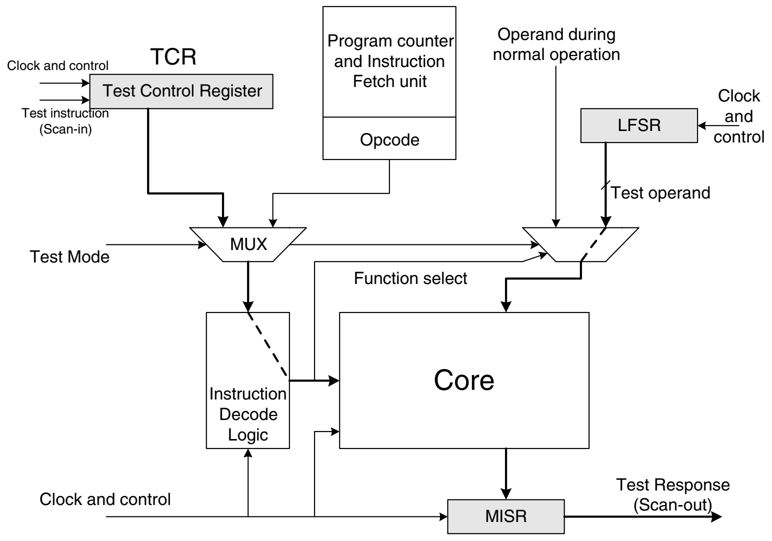

Fig. 3. Implementation of a BIST architecture on the general structure of a microprocessor.

core. BIST controller applies required clock and control signals for executing each instruction. For implementation of this architecture, we need to have enough information of the internal structure and instructions of the processor under test. It should be tried to avoide the using of repeated intructions that have the same operations. Thus, all instructions should be analyzed and classified based on their operations.

3.1. *Introduction of the Proposed BIST Architecture*

In this paper, we introduce a BIST architecture based on a testable design for processor cores (Subsection 3.2 describes a testable design for processors). In this architecture, the hardware overhead for BIST controller and elements are reduced without performance degradation. Implementation of this architecture has been treated on UTS-DSP. In this architecture, we can store the test instructions and the expected results at the embedded ROM. Thus, we do not need to enter them as inputs. Instead of using some additional elements for transfering instructions into instruction decoder logic, we can use processor controller to read them from the embedded ROM. The comparison of MISR value with expected result is achieved internally. The expected results have been precomputed from a fault-free circuit. With knowing the internal structure of each functional block, a self-test program is provided for testing that block. Operation of some instructions are modified for

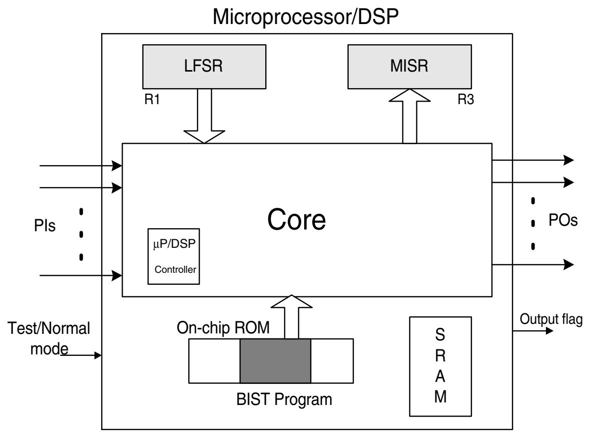

Fig. 4. The proposed architecture.

test mode for Read or Write operations from LFSR or to MISR. It requires no external BIST controller, all of the control operations are performed internally. This architecture is shown in Fig. 4. As shown, this architecture requires no external hardware for testing μp/DSP cores. Implementation of this architecture on the general structure of a processor is shown in Fig. 5.

Since the main functional blocks such as ALU and MAC are connected to the data bus, test data can be input to or output from these blocks by connecting LFSR and MISR to the data bus. By replacing some I/O registers such as Data Receive Register (DRR) and Data

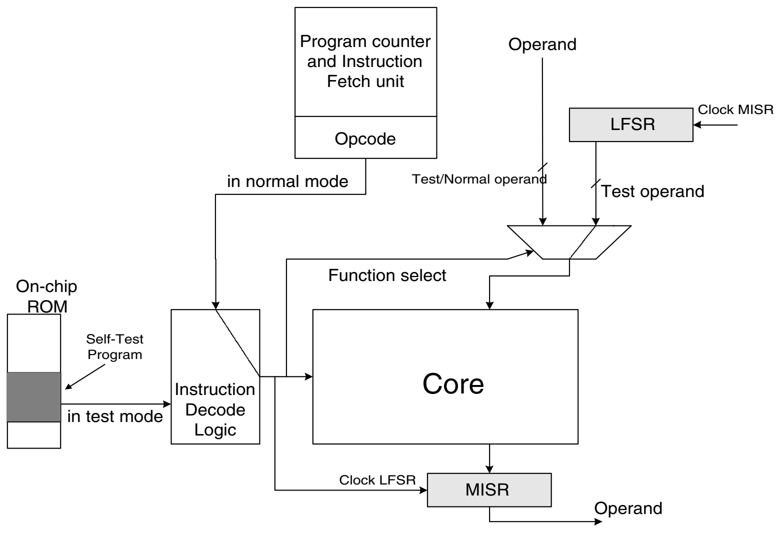

Fig. 5. Implementation of the proposed architecture on a general core.

Transmit Register (DXR) with the LFSR and MISR, the functional blocks can be self-tested by normal instructions when the DSP enters into the BIST mode. Since the I/O registers are the only registers that are used in data communication for external devices, they are chosen to be replaced. While in the BIST mode, data movement is identical to that during normal operation [21].

3.1.1. Test Sequence for the Architecture. This architecture can be implemented on all general processor cores. In this paper, this architecture has been implemented on UTS-DSP. The test process for testing the UTS-DSP is as follows (see Figs. 4 and 5 as well): First, DSP is arrived into the test mode by asserting test/normal mode pin, then the DSP is reset. DSP controller jumps to boot routine at the on-chip ROM. The operation mode is checked here. In the test mode, Program Counter (PC) jumps to self-test program address at the on-chip ROM. Self-test program is read by the DSP controller and is executed by execution unit. The needed tests are as deterministic or as provided by random pattern generator (LFSR). The results of tests are transmitted into test response analyzer (MISR). The generated signature for each block by MISR is read by BIST program and compared with the expected result precomputed for a fault-free block.

3.1.2. BIST Controller for the Architecture. DSP controller and BIST program perform the following control operations: checking the test or normal mode, reading the self-test program from the on-chip ROM, executing the instructions of self-test program, reading the test data from LFSR or operand, writing the results into MISR, reading the content of MISR and comparing with the pre-computed signature of a fault-free circuit, transmitting the final result (pass/fail) to output flag. BIST program and DSP controller perform all required control operations. There is no need to use additional hardware for BIST controller and thus, area and cost overhead is significantly reduced.

3.2. *Testable Processor Core Design*

To accommodate our proposed test methodology, minor modifications should be applied to processor controller and to some instructions within its test phase. In that case, we will be able to generate and apply test patterns at the speed of the processor under test. That is, we modify the controller to interpret some of the instructions differently only within the initial test mode. The VHDL code of UTS-DSP is available in VLSI Circuits and Systems Laboratory and can be used for changing the instructions and DSP core controller. Because the testing of DSP is performed in off-line mode, some components such as Data Receive Register (DRR) and Data Transmit Register (DXR) of DSP chip can be used for self-testing. LFSR is used only for read operation, which is used as test generator that is, an instruction reads the contents of LFSR (a random pattern). MISR

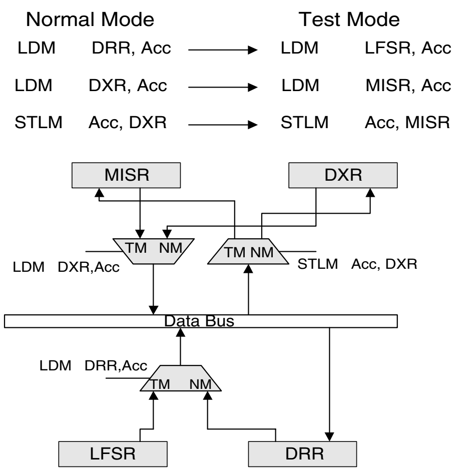

Fig. 6. Modifications of instructions and DSP controller.

is used for Read/Write operations. When the MISR is used as signature analyzer, the result of execution of an instruction (which is in Accumulator) is loaded into MISR (write operation). When the value of MISR is compared with the expected results, the content of MISR must be loaded into Accumulator (Read operation). Instructions for reading from LFSR/MISR and writing to MISR are created (see Fig. 6). To avoide defining some new instructions, the existing instructions can be modified for this purpose. Some instructions of DSP are not used in the test mode thus, we utilize these instructions for the test purpose. In fact, we change the interpretation of these instructions for the test mode. For example, the normal instruction "LDM DRR, Acc" is changed to "LDM LFSR, Acc" in test mode. For this instruction, in normal mode, the content of DRR is loaded into Accumulator and when this instruction is used in self-test program, first LFSR is clocked and then the new content of LFSR is loaded into Accumulator. The same operations are performed for instructions of "LDM DXR, Acc" and "STLM Acc, DXR." These instructions are changed to "LDM MISR, Acc" and "STLM Acc, MISR," respectively. All of these changes are applied in VHDL code of the UTS-DSP. Fig. 6 shows the changes of instructions and DSP controller.

A minor modification for checking the test/normal mode has to be made in the initialization of the μP/DSP at the reset situation. For this purpose, if μP/DSP is entered into normal mode, PC jumps to normal routine, otherwise jumps to test routine for execution of self-test program from the on-chip ROM.

3.3. BIST Program

Actually, the test program consists of the normal instructions that are executed by the system, while the data are provided as deterministic patterns or random patterns by the LFSR. In this step, tests are developed for individual components of the processor, such as the ALU and the Shifter. Structural faults are targeted during component test generation. Component tests can be provided as deterministic or random patterns. We use only deterministic tests for Shifter unit and for other blocks such as ALU both deterministic and random tests are used. Deterministic tests are stored into the embedded ROM and the random tests are provided by the random generator (LFSR) and can be read by the modified instruction "LDM LFSR, Acc."

If random test set is chosen, we use the modified instructions in our self test program to read random tests from LFSR and create a signatur by output response analyzer for each block. The number of needed test patterns for each component relies on desired fault coverage and test time overhead. To achieve a high fault coverage, the number of random tests are intensively increased, therefore the test time is increased. We have considered the primitive polynomials for 16-bit LFSR and MISR. A signature for each block has been obtained from a fault free circuit and all signatures are stored into the embedded ROM. If deterministic test set is chosen, tests are loaded along with BIST program into ROM. Note carefully that set of BIST data/program consists of self-test program, deterministic tests and expected signatures, that can be loaded into ROM as shown in Fig. 7 and no external tester is required. As

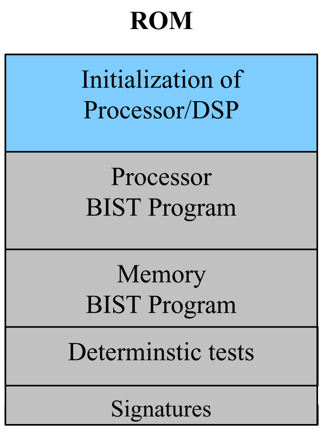

Fig. 7. A schematic of the embedded ROM.

shown, there is no random pattern generation and signature analysis program stored in ROM because these tasks are performed by the modified instructions. Unlike random patterns, reading of the deterministic tests are done by the normal instructions.

By targeting the structural test requirement of individual components, our methodology has the fault coverage advantage of determiniatic structural testing. Since component test application and response collection are performed with instructions instead of with scan chains, it imposes no area and performance overhead, and the test application is performed at the speed of the processor. Based on the above, a complete self test program has been written in UTS-DSP assembly language. BIST program includes two parts: (1) Self-Test program for testing core functional blocks and (2) BIST controller program for controlling of Read/Write/Compare operations on memories. BIST program consists of 350 words for self test program and signatures of functional blocks.

4. Embedded SRAM Testing

In this paper we show the feasibility of fault model and test algorithm. Defects in the layout of memory are modeled as faults in the corresponding transistor diagram. The electrical behavior of each defect is analyzed and classified, resulting in a fault model at SRAM cell level. Only spot defects are considered for memory testing. These defects result breaks and shorts in the circuit. Word-oriented memories contain more than one bit per word; i.e., $B \geq 2$, that B represents the number of bits per word and usually is a power of two [25]. Read operation reads the B bits simultaneously and write operation writes data into the B bits of memory. Many different data backgrounds (DBs) are used for testing of word-oriented memories [26]. Once the testing of the processor core is completed, this core is used to test the embedded SRAM. The proposed architecture is also implemented for testing an embedded SRAM but there is no need in using LFSR, MISR and modified instructions. The test vectors and the test algorithm that are written in normal UTS-DSP assembly language are stored at the on-chip ROM.

4.1. The SRAM Fault Model

A widely used fault model for RAM devices is the one presented in [12]. In this model, a RAM circuit is divided into three blocks, i.e., memory cell array, address decoder circuit and the sense amplifier or the read-write circuit. These blocks differ in structure hence they are analyzed separately. Defects in the address decoder and the R/W logic are mapped onto functionally equivalent faults in the memory array. This method has the advantage that all faults can be considered to be in the memory array.

A cell may have stuck-at-1/0 or coupling with other cells faults. In the decoder circuit, a decoder may not access the addressed cell; it may access a nonaddressed cell or multiple cells. The read-write circuit may have stuck-at-1/0 faults, which appear as memory cell stuck-at faults. Actual fault mechanism based upon physical defects in memory devices have been investigated. The proposed fault model with the addition of state transition and data retention faults could cover all faults in the memory [7].

4.1.1. Memory Array. Some defects can occur in the layout schematic (manufacturing process). Defect analysis is done in two steps. The first step is the translation of defects in the layout to defects in the transistor circuit. The second step is classifying defects at transistor level based on equivalent faulty memory cell behavior. This implies a fault model at SRAM cell level. Thus a more general fault model for SRAMs includes: (1) Memory cell stuck-at-1/0 faults, (2) Memory cell stuck-open fault, (3) Memory cell state transition 1-to-0 and 0-to-1 faults, (4) Memory cell state coupling faults to another cell. (5) Memory cells multiple access and wrong addressing faults. (6) Data retention faults. Corresponding defects of faults at the transistor diagram of a memory cell are shown in Fig. 8.

4.1.2. Address Decoder and R/W Logic. A general fault model for the address decoder is: (1) More than

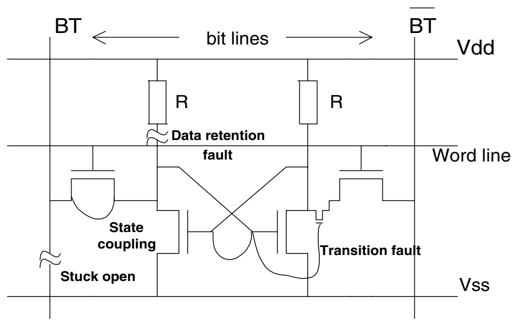

Fig. 8. Examples of circuit defects for several fault classes.

Table 1. DB and DBbar for 16-bit SRAMs.

No. of DB	Normal (DB)	HEX	Inverse (DBbar)	HEX
1	0000 0000 0000 0000	0000	1111 1111 1111 1111	FFFF
2	0101 0101 0101 0101	5555	1010 1010 1010 1010	AAAA
3	0011 0011 0011 0011	3333	1100 1100 1100 1100	CCCC
4	0000 1111 0000 1111	0F0F	1111 0000 1111 0000	F0F0
5	0000 0000 1111 1111	00FF	1111 1111 0000 0000	FF00

one cell is accessed by one address (*multiple access*), (2) An address accesses no cells (*stuck-open*). All faults in the address decoder can be converted as memory array faults. A general fault model for R/W logic has been introduced, namely: (1) One or more of the m bits is stuck-at, (2) One or more of the m bits is stuck-open, (3) A pair of bits is state coupled. All faults in the R/W logic can be viewed as faults in memory array.

4.2. *Test Algorithm and Fault Model for Word-Oriented SRAMs*

Several innovative test algorithms for RAMs have been reported in the recent years. In this paper, marching 1/0 test algorithm is used as a basis of test method. With the slight modifications in this algorithm, very high fault coverage can be obtained. The fault models for word-oriented memories can be divided into the following classes.

4.2.1. Single-Cell Faults. this class can be included the following faults: (a) Stuck-at faults, (b) Transition faults, (c) Data retention faults.

4.2.2. Fault Between Memory Cells. this class of faults consists of coupling faults.

March tests for bit-oriented memories can be converted to march tests for word-oriented memories by taking into account that in the bit-oriented memory tests, the 'Rd0', 'Rd1', 'Wr0' and 'Wr1' operations are applied to a single bit. In case of word-oriented memories, an entire word of B bits has to be read or written; the data value of this word is called *Data Background* (*DB*). Word-oriented SRAMs introduce the problem of state coupling faults between two cells at one address. To detect these faults all states of two arbitrary cells at the same address must be checked. This is only possible if several data backgrounds are used. A minimum of K data backgrounds will be needed where $K = [\log_2 B] + 1$. In many memories B is power of two, then the formula simplified to: $K = \log_2 B + 1$.

For a memory with $B = 16$, the DBs of Table 1 could be used; it can easily be verified that for any two cells all 4 states occur. In this case state coupling between any of two cells in the same address is checked.

4.3. *The 9N Test Algorithm*

A length 9N test algorithm is presented, where *N* is the number of addresses. A data retention test is added to this algorithm that is shown in Fig. 9. A Rd0 instruction represents reading from the memory array and expect the logical 0 from the addressed cell. A Wr0 instruction represents writing a logical 0 to an addressed cell. The address is indicated in the first column of the Fig. 9. The proposed wait-time in the data retention test depends on the nodal capacitance and the leakage current in a memory cell. In the Philips 8K8 memory, a wait-time of 100 *msec* was estimated. Other cell designs

Address	Initialization	March element1	March element2	March element3	March element4	Wait	March element5
0	Wr0	Rd0Wr1	Rd1Wr0	Rd0Wr1	Rd1Wr0		Rd0
1	Wr0	Rd0Wr1	Rd1Wr0	Rd0 Wr1	Rd1Wr0	Disable	Rd0
2	Wr0	Rd0Wr1	Rd1Wr0	Rd0 Wr1	Rd1Wr0	RAM	Rd0
N	Wr0	Rd0Wr1	Rd1Wr0	Rd0Wr1	Rd1Wr0		Rd0

Fig. 9. The 9N test algorithm for SRAMs. A data retention test is added.

or processes may result in other wait-times. It can be proven that the 9N test algorithm detects all faults of the fault model. The test algorithm detects all stuck-at, transition, state coupling, multiple access and stuck-open faults [6].

4.4. BIST Structures

The 9N test algorithm is written in assembly language program only with normal instructions. Although, we used the 9N algorithm, any memory test algorithm can be used for this purpose. When the processor is entered into test mode, the embedded RAM is tested by BIST program. The 9N test algorithm is implemented on the 32K word 16-bit memory of the UTS-DSP. In this method, there is no additional hardware overhead and it covers all important memory faults. There is no need to hardware BIST controller. A complete test for a word-oriented SRAM will thus proceed in the following way: First run the test algorithm with data background 1, then run it again with data background 2 and etc. Finally the data retention test is run (only once). There is no need to run the data retention test for all data backgrounds. The BIST program and DSP controller are used for controlling the test process like the processor core testing. BIST program and DSP controller perform the reading of deterministic tests, writing of test data into memory cells, reading back test data from memory cells, and comparison between the read data from memory and the written data into it. In this method using separate hardware as BIST controller is not necessary. In BIST program, Read instruction is used to read data from memory, Write instruction is used to write data into memory and Compare instruction is used to compare between the content of addressed memory space and the correct value. The read, write and compare process in the BIST program is performed for each memory space. BIST program can detect the fail-bit locations. Hence, the diagnosis capability is also provided.

4.4.1. The Flow of BIST Program. BIST program implements the 9N test algorithm for word-oriented memories. This algorithm is written in processor assembly language. The flow of BIST program for $B = 16$ is shown in Fig. 10. This algorithm can be implemented on all RAM cores on a system-on-a-chip. This program consists of only normal instructions which allocates 88 words (each 16 bits wide) of embedded ROM.

As shown in Table 1, the number of DBs for $B = 16$ is 5. The initialization step from 9N algorithm is shown in lines of 7 to 14. The march elements 1 and 2 are shown in lines of 15 to 32. These steps cover all single faults. The march elements 3 and 4 are shown in lines of 33 to 48. These steps cover all coupling faults. For

```
1                    /* BIST Program */
2                 .  /* See figure 9, the 9N test algorithm*/
3    for j=1  to  5  DO;
4              /* 5 : Number of DBs for 16-Bit SRAM,
5                      as shown in Table 1 */
6  {
7    for i=1  to  N  DO;
8             /* N: Total address space for 32K SRAM,
9                N=32768 */
10 {
11     Write DBj  to address word(i)
12            /* Initialization, write DBj to all address
13                    of memory space */
14   }
15              /*  March element 1 */
16   for i=1  to  N  DO;
17 {
18    Read DBj     from  address word(i)
19    Compare DBj with RDBj
20              /* Compare read value with DBj */
21     if (NotEqual) GoTo Error
22    Write DBjbar  to     address word(i)
23 }
24              /*  March element 2 */
25   for i=1  to  N  DO;
26 {
27        Read DBjbar  from  address word(i)
28       Compare DBjbar with RDBjbar
29               /* Compare read value with Dbjbar */
30       if (NotEqual)   GoTo Error
```

Fig. 10. The flow of BIST program.

(*Continued on next page.*)

```
31      Write DBj    to    address word(i)
32 }
33              /* March element 3 */
34   for i=N  to  1  DO;
35 {
36        Read DBj    from  address word(i)
37        Compare DBj with RDBj
38        if (NotEqual) GoTo Error
39        Write DBjbar  to    address word(i)
40 }
41              /* March element 4*/
42   for i=N  to  1  DO;
43 {
44        Read DBjbar  from  address word(i)
45        Compare with Dbjbar with RDBjbar
46        if (NotEqual)   GoTo Error
47        Write DBj    to    address word(i)
48  }
49 }
50              /* Data Retention Test  ,  j=5  */
51        Pause for 100 msec
52   for i=1  to  N  DO;
53 {
54        Read DB(final)    from  address word(i)
55        Compare DB5 with RDB5
56        if (NotEqual) GoTo Error
57  }
58              /* End of test
59   No faulty : Reset the output flag
60   Error : Set the output flag
/* RDBj : Read DBj from addressed space of memory.
   RDBjbar : Read DBJbar */
```

Fig. 10. (*Continued*).

each DB, lines 3 to 49 are repeated to detect all single and coupling faults. A data retention test is added to BIST program to cover the data retention faults. The BIST program covers all of the single and coupling faults. This program is written in UTS-DSP assembly language [8, 17].

5. Practical Results

5.1. The UTS-DSP Core Testing

A complete self-test program for CPU testing of UTS-DSP has been provided. To evaluate the fault coverage of a test program on the processor under test, we have established the test evaluation framework shown in Fig. 11. The compiler takes the code of the test program written in processor assembly language and prepares a .lst file. A convertor program (written in C language) converts .lst to .dat file (memory readable format). This file containing the initialized instruction memory and data memory which is used as a test bench for VHDL simulator. The VHDL simulator takes the design description, run the test bench and captures the input signals to the processor. The fault injection into core VHDL code [14] is done by a C program. The fault coverage is determined by a comparator. For practical testing of BIST program and determining the actual test-time, a DSP emulator can be used. The TMS320C548 DSP (Texas Instruments) has been used on the examined emulator board. After running the BIST program on the DSP emulator, the obtained test time from execution of the self-test programs for different functional blocks and CPU are shown in Table 2. The total area overhead is comprised of one 16-bit LFSR, one 16-bit MISR, 3 MUX (2×1) and 350 words of ROM. This area overhead in contrast to the size of processor core is negligible. Table 2 also lists the size of each block and CPU (equivalent NAND gates) and the size of each functional block self-test program. As shown in the table, the CPU fault coverage, 87%, is less than the fault coverage of functional blocks, 95%, because the controller blocks in the CPU are hard to test by random patterns. They need to be tested by higher number of deterministic patterns extracted by ATPG to achieve

Table 2. Practical results.

	No. of gates	No. of patterns deterministic & random	No. of words	Fault coverage
ALU	4105	32×10^3	130	>93.7
MAC	3756	40×10^3	87	>92.3
Shifter	663	1.6×10^3	59	=99.2
CSSU	383	4.4×10^3	31	=95.5
Functional blocks	8907	78×10^3	307	>95%
CPU	10110	81×10^3	350	>87%

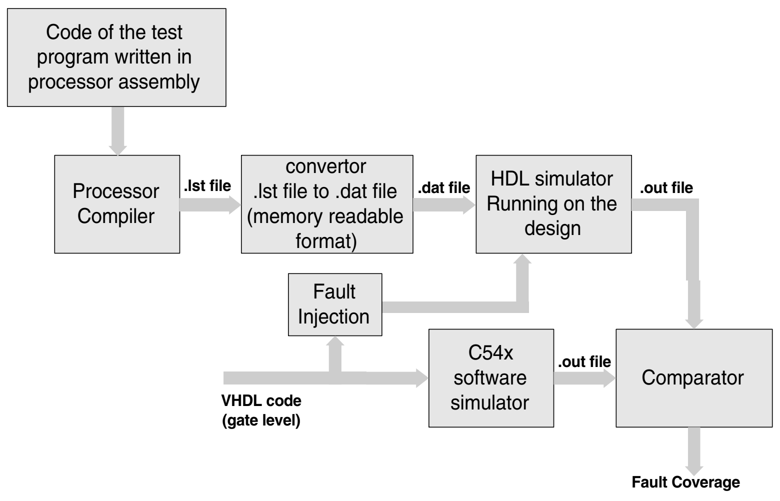
Code of the test program written in processor assembly
Processor Compiler
.lst file
convertor .lst file to .dat file (memory readable format)
.dat file
HDL simulator Running on the design
.out file
Fault Injection
VHDL code (gate level)
C54x software simulator
.out file
Comparator
Fault Coverage

Fig. 11. Test evaluation framework.

a higher fault coverage. The overall test time for executing the test program is 2.5 msec in the working frequency $f = 65$ MHz. Increasing the number of random test patterns increases the fault coverage and the test time as well, but the size of the test program is fixed.

5.2. *The Embedded SRAM Testing*

The same process is also performed for embedded SRAM testing with a BIST program and the fault model as has been described in Section 4. The BIST program that is written in assembly language allocates 88 words of embedded ROM. Additionally, only 10 test patterns need to be stored in the ROM. The achieved results from implementation of this algorithm and their comparison with other methods are shown in Table 3. The table also includes the total number of words occupied by each algorithm in ROM. The 9N test algorithm covers all single and coupling faults. However, the other methods as shown in Table 3, cannot cover all the faults under the defined fault model [22]. In the proposed method, with increasing the size of RAMs, only the test time is increased without any increasing the hardware overhead or self-test program size.

6. Conclusion

This paper presents a low cost BIST architecture. This architecture is an efficient method for testing of all processor and embedded SRAMs in system-on-chip. This architecture has been implemented on UTS-DSP. A complete self-test program has been written in processor assembly language and it is stored at the embedded ROM. There is no need of additional hardware for BIST controller. DSP controller and BIST program control the BIST process. This test method covers >95% faults of the functional blocks of UTS-DSP in a very small test time, test cost and negligible hardware overhead.

As well, a systematic approach for constructing marching test (9N algorithm) for word-oriented

Table 3. Comparison between several methods (n = Number of bits).

Test method	Complexity	Stuck-at	State transition	Decode address	State coupling	32K SRAM ($f = 65$ MHz)	No. of words
GALPAT algorithm	$4n^2$	Yes	Yes	Yes	Yes	16 hr	68
Checker pattern	$4n$	Yes	50% Yes	No	No	51 msec	49
Marching 1/0	$6n$	Yes	Yes	Yes	No	72 msec	72
9N algorithm	$9n$	Yes	Yes	Yes	Yes	81 msec	88

memories has been presented in this paper. The 9N test algorithm is an efficient method for testing the embedded SRAMs. The test algorithm covers 100% of faults under the defined SRAM fault model. The proposed test algorithm shows excellent performance in test time and fault coverage, and it is independent of row, column and cell arrangements in the memory array. The given test algorithms are both suitable for bit and word-oriented SRAMs and there is no additional hardware overhead. This algorithm has been implemented on 32K word SRAM of UTS-DSP. With increasing the size of RAMs, only the test time is increased without any increasing hardware overhead.

Acknowledgment

The authors would like to thank Professor Krishnendu Chakrabarty of ECE Department of Duke University for his useful suggestions.

References

1. M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital System Testing and Testable Design*, New York: Computer Science Press, 1990.
2. K. Batcher and C. Papachristou, "Instruction Randomization Self Test for Processor Cores," in *Proc. VLSI Test Symp. (VTS'99)*, 1999, pp. 34–40.
3. M. Bushnell and V. Agrawal, *Essentials of Electronic Testing*, Kluwer Publishers, 2000.
4. L. Chen and S. Dey, "Software-Based Self-Testing Methodology for Processor Cores," *IEEE Trans. On CAD of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, 2001.
5. F. Corno, M. Sonza Reorda, G. Squillero, and M. Violante, "On the Test of Microprocessor IP Cores," in *Proc. Design Automation and Test in Europe (DATE'01)*, 2001, pp. 209–213.
6. R. Dekker, F. Beenker, and L. Thijssen, "Fault Modeling and Test Algorithm Development for SRAMs," in *Proc. Int. Test Conf., (ITC'88)*, 1988, pp. 343–351.
7. R. Dekker, F. Beenker, and L. Thijssen, "A Realisitic Fault Model and Test Algorithm for Static Random Access Memories," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 9, no. 6, pp. 567–572, 1990.
8. M. Fayyazi , M. Movahedin, Z. Navabi, P. Riahi, and A. Dezfoli, " An Efficient CPU Architecture for DSP Processor," in *Iranian Conf. On Electrical Engineering (ICEE), Computer Proceedings*, 1999, pp. 63–70.
9. S. Heath, *Microprocessor Architectures, RISC, CISC and DSP*, 2nd Edition, 1995.
10. G. Hetherington, G. Sutton, K. Butler, and T. Powell, "Test generation and Design for Test for a Large Multiprocessing DSP," in *Proc. Int. Test Conf. (ITC'95)*, 1995, pp. 149–156.
11. N. Kranitis, D. Gizopoulos, A. Paschalis, and Y. Zorian, "Instruction-Based Self-Testing of Processor Cores," in *Proc. VLSI Test Symp. (VTS'02)*, 2002, pp. 223–228.
12. R. Nair and S. Abraham, "Efficient Algorithm for Testing Semiconductor Random-Access Memories," *IEEE Trans. on Computers*, vol. C-27, pp. 572–576, 1978.
13. O. Nasibi, M. Nourani, M. Fakhraie, and A. Dezfoli, " Multi-Access Integrated Memory Management for Delay Pipelined Processors," in *International Conference on Microelectronic*, 1999, pp. 214–220.
14. Z. Navabi, *VHDL: Analysis and Modeling of Digital Systems*, McGraw-Hill: New York, 1993.
15. K. Radecka, J. Rajski, and J. Tyszer, "Arithmetic Built-In Self-Test for DSP Cores," *IEEE Trans. on CAD*, vol. 16, no. 11, pp. 1358–1369, 1997.
16. R. Rajsuman, "Testing a System-on-a-Chip with Embedded Microprocessor," in *Proc. Int. Test Conf.*, 1999, pp. 499–508.
17. P. Riahi, M. Movahedin, M. Fayyazi, and A. Dezfoli, "UTS-DSP IC Core," in *Iranian Conf. on Electrical Engineering (ICEE), Computer Proceedings*, 1999, pp. 71–79.
18. N. Sakashita and H. Sawai, "Built-In Self-Test in A 24 bit Floating Point Digital Signal Processor," in *Proc. Int. Test Conf.,* 1990, pp. 880–885.
19. M. Samady, M. Movahedin, M. Fakhraie, and A. Dezfoli, "Implementation of Serial Port Interconnections for Integrated Circuits," in *International Conference on Microelectronic*, 1999, pp. 291–294.
20. J. Shen and J. Abraham, "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation," in *Proc. Int. Test Conf. (ITC'98)*, 1998, pp. 990–999.
21. M.H. Tehranipour, Z. Navabi, and S.M. Fakhraie, "A Low Cost BIST Architecture for Processor Cores," in *Proc. IEEE Electronic Circuits and Systems Conference (ECS'01),* 2001, pp. 11–14.
22. M.H. Tehranipour, Z. Navabi, and S.M. Fakhraie, "An Efficient BIST for Embedded SRAM Testing," in *Proc. Int. Symp. on Circuits And Systems (ISCAS'01),* 2001, vol. 5, pp. 73–76.
23. *TMS320C54x DSP: CPU and Peripherals*, Texas Instruments, 1996.
24. *TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set*, Texas Instruments, 1996.
25. A.J. Van de Goor and I. Tlili, "March Test for Word-Oriented Memories," in *Proc. Design Automation and Test in Europe (DATE'98)*, 1998, pp. 501–508.
26. A.J. Van de Goor, I. Tlili, and S. Hamdioui, "Converting March Test for Bit-Oriented Memories into Tests for Word-Oriented Memories," in *IEEE Design and Test*, 1998, pp. 46–51.
27. W. Zhao and C. Papchristou, "Testing DSP Cores Based on Self-Test Programs," in *Proc. Design, Automation and Test in Europe (DATE'98)*, 1998, pp. 166–172.
28. Y. Zorian, "System-Chip Test Strategies," in Proc. *Design Automation Conference (DAC'98)*, 1998, pp. 752–757.

Mohammad H. Tehranipour received his B.Sc. degree in Electrical Engineering from Amirkabir University of Technology (Tehran Polytechnic University) in 1997 and M.Sc. degree in Electrical Engineering from the University of Tehran, Tehran, Iran in 2000. He has worked in the VLSI Circuits and Systems Laboratory in ECE Department of the University of Tehran from 1998 to 2002 as a Research Assistant and Lab Administrator. Since Jan. 2002 he has been on the Ph.D. student in Department of Electrical Engineering in the

University of Texas at Dallas. He is currently a Research Assistant in the Center for Integrated Circuits and Systems (CICS). His Current research interests include: VLSI Testing, Design-for-Testability (DFT), Signal Integrity Fault Modeling and Test, Test Resource Partitioning, Low-Power Testing and DSP Architectures.

Sied Mehdi Fakhraie was born in Dezfoul, Iran, in 1960. He received his M.Sc. degree in electronics from the University of Tehran, Tehran, Iran, in 1989 an the Ph.D. degree in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada in 1995. Since 1995, he has been an Assistant Professor with the Department of Electrical and Computer Engineering, University of Tehran, where has been the founding Director of the VLSI Circuits and Systems Laboratory. From September 2000 to April 2003, he was with Valence Semiconductor Inc. and has worked in Dubai, UAE, and Markham, Canada offices of Valence as Director of ASIC/SoC Design and also technical lead of Integrated Broadband Gateway and Family Radio System baseband processors. During the summers of 1998, 1999, and 2000, he was a visiting professor at the University of Toronto, where he continued his work on efficient implementation of artificial neural networks. He is coauthor of the book VLSI-Compatible Implementation of Artificial Neural Networks (Boston, MA: Kluwer, 1997). He has also published more than 60 reviewed conference and journal papers. He has worked on many industrial IC design projects including design of network processors and home gateway access devices, DSL modems, pagers, and one- and two-way wireless messaging systems, and digital signal processors for personal and mobile communication devices. His research interests include system design and ASIC implementation of integrated systems, novel techniques for high-speed digital circuit design, and system-integration and efficient VLSI implementation of intelligent systems.

Dr. Zainalabedin Navabi is an adjunct professor of electrical and computer engineering at Northeastern University and an associate professor of ECE at the University of Tehran. Dr. Navabi is the author of several textbooks and computer based trainings on VHDL, Verilog and related tools and environments. He has published several books on VHDL and Verilog and is a trainer and consultant to EDA companies. He is an active member of IEEE societies working on hardware description languages and related standards. Since 1981, Dr. Navabi has been involved in the design, definition and implementation of Hardware Description Languages (HDL). He has written numerous papers on the application of HDLs in simulation, synthesis and test of digital systems. His research interests are in digital system test and utilization and design of hardware description languages. Dr. Navabi received his M.S. and Ph.D. from the University of Arizona in 1978 and 1981, and his B.S. from the University of Texas at Austin in 1975. He is a senior member of IEEE, a member of IEEE Computer Society, member of ASEE, and ACM.

Mohammad-Reza Movahedin received his B.Sc., M.Sc. and Ph.D. in Electrical Engineering in 1989, 1991 and 2000 all from University of Tehran. During his Ph.D. study, he was involved in design, verification, fabrication and field testing of several digital integrated circuits. During this period, he proposed novel methods of design for error tolerant digital chips. In August 2000 Dr. Movahedin joined Valence Semiconductor headquartered in Irvine, California and was in charge of development of technical infrastructure of their branch in Dubai Internet City, United Arab Emirates. In addition, he was responsible for digital design and implementation of HomePlug 1.0 base-band chip with close to 5 million equivalent gates. The chip contains a whole system with processor, memory and peripheral systems and a huge DSP coprocessor for signal processing operations needed for HomePlug 1.0 standard. Dr. Movahedin is currently involved in design of several SoC's based on digital signal processing applications for companies and research institutes in Tehran, Iran and Dubai, UAE.