EXPLOITING PROGRAMMABLE BIST FOR THE DIAGNOSIS OF EMBEDDED MEMORY CORES

D. Appello**, P. Bernardi*, A. Fudoli**, M. Rebaudengo*, M. Sonza Reorda*, V. Tancorre**, M. Violante*

*
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

**
ST Microelectronics
TPA Mixed Signal Test Solution Group
Cornaredo, Italy

Abstract[1]

This paper addresses the issue of testing and diagnosing a memory core embedded in a complex SOC. The proposed solution is based on a P1500-compliant wrapper that follows a programmable BIST approach and is able to support both testing and diagnosis. Experimental results are provided allowing to evaluate the benefits and limitations of the adopted solution and to compare it with previously proposed ones. The solution takes into account several constraints existing in an industrial environment, such as minimizing the cost of test development, easing the reuse of the available architectures for test and diagnosis of different memory types and minimizing the cost of the external ATE.

1. Introduction

Fast innovation in VLSI technologies makes it possible to integrate a complete system into a single chip (System-on-Chip, or SOC). In order to handle the resulting design complexity, reusable cores are being used in many SOC applications. System designers can purchase cores from core vendors and integrate them with their own User-Defined Logic (UDL) to implement SOCs. Core-based SOCs show important advantages: the cost of the end-product is decreased, and thanks to design re-use, the time-to-market is greatly reduced.

When speaking of SOCs, the test problem is a major challenge for industries as well as for the research community [1-2]. The main problem lies in the reduced accessibility of cores and UDLs. Traditional approaches [1-3] for testing core-based SOCs completely rely on additional Design for Testability (DfT) structures such as test busses for test transfer from/to the core under test. The access mechanism requires additional logic (such as a wrapper around the core) and wiring (such as a Test Access Mechanism, or TAM) to connect cores to the test source and sink. A crucial point to be solved in SOC testing is the extra cost introduced by the DfT logic, i.e., the area, delay and test application time overheads [4].

Due to the complexity of current SOCs, the adoption of standards allowing reducing the efforts for the integration, debug, and test of the cores they are composed of (as well as of the whole system) is also a major trend [5]. Moreover, the issue of devising efficient methods for the diagnosis of embedded cores is now becoming of increasing importance [6-7] due to the economical impact of faults in SOCs. The adoption of diagnostic analysis is also stimulated by the increased popularity of the Built-In Self-Repair approaches that exploit diagnosis to repair faulty chips using redundant components [8].

In this paper we focus on SRAM embedded memory core test and diagnosis. Since embedded SRAMs are the most widely used cores in SOC applications, extensive research on fault detection in embedded memories has been performed and efficient algorithms have been proposed and implemented [9-10]. BIST provides an effective way to automatically generate test sequences, compressing the outputs and evaluating the goodness of embedded memory cores, and BIST-based solutions are now very popular [11-12]. The typical memory BIST implements a March algorithm [13] composed of a sequence of March elements, each corresponding to a series of read/write operations on the whole memory. Different hardware approaches have been proposed in the literature in order to implement BIST-based March algorithms.

The *hardwired BIST* approach is the most widely used. It consists in adding a custom circuitry to each core, implementing a suitable BIST algorithm [14-15]. The main advantage of this approach is that the test application time is short and the area overhead is relatively small. Hardwired BIST is also a good way to protect the *intellectual property* contained in the core: the memory core provider needs only to deliver the BIST activation and response commands for testing the core without disclosing its internal design. At the same time, this approach provides very low flexibility: any

[1] This work has been partially supported by the Italian Ministry for University through the Center for Multimedia Radio Communications (CERCOM).

modification to the test algorithm requires redesigning the BIST circuitry.

The *soft BIST* approach [16] assumes that a processor is already available in the SOC: the processor is exploited to run a program that performs the test of the other cores. The test program executed by the processor applies test patterns to each core under test and checks for the results. The test program is stored in a memory containing also the test patterns. This approach uses the system bus for applying test patterns and reading test responses, and it guarantees a very low area overhead, limited to the chip-level test infrastructure. The disadvantage of this approach is mainly related to the strict dependence of the test program on the available processor. As a result, the core vendor needs to develop for the same core different test programs, one for each processor family, thus increasing the test development costs. Moreover, intellectual property is not well protected, as the core vendor supplies to the user the test program for the core under test. Finally, this approach can be applied only to cores directly connected to the system bus; the approach cannot be applied if the core is not completely controllable and observable.

An alternative approach is the one usually denoted as *programmable BIST* [17]. The core vendor develops a DfT logic, which wraps the core under test and includes a custom processor, which is exclusively devoted to test the core. The advantages of this architecture are manifold: the intellectual property can be protected, only one test program has to be developed, and the design cost for the test is very reduced; the technique provides high flexibility since any modification of the algorithm simply requires a change in the test program; the test application time can be taken under control thanks to the efficiency of the custom test processor, and the test can consequently be executed at-speed. Finally, each core is autonomous even from the test point of view, and its test only requires activating the test procedure and reading the results, as for hardwired BIST. The main potential disadvantage is the area overhead introduced by replicating the custom processor in each core under test. However, due to the very limited size of the processor, this problem is marginal (especially when applied to cores including medium- and large-sized memories), as shown in the following.

In this paper we describe how the programmable BIST approach is being introduced in an industrial design flow, and reports experimental results gathered on a case study which is currently being implemented on a test chip by STMicroelectronics. The proposed architecture (a preliminary version oriented to the test of flash memories is described in [18]) is based on a custom processor in charge of executing the test of embedded memory modules by running a test program which implements a March algorithm. The whole architecture has been designed in order to support not only test, but also diagnosis, and for this reason the external ATE can interact with the core to extract information and to customize the execution of the March algorithm according to the diagnosis needs. Communication between the external ATE and the core takes place through an IEEE 1149.1 TAP; the core has an external layer corresponding to a P1500-compliant wrapper [19]. The test of the core can be executed by means of IEEE 1149.1 instructions and the core test details are completely transparent to core user.

The case study has been developed using a SRAM core, which is representative of memory cores mostly used within STMicroelectronics SOCs; the core has been properly wrapped according to the new architecture. Experimental results are reported concerning the area overhead involved by the method: results are also compared with those provided by comparable hardwired BIST approaches [7][20]. The main advantage of the proposed approach lies in its high flexibility and adaptability to different memory cores and in its possibility of easily changing the implemented March algorithm as well as introducing new features. The new method is compatible, from the point of view of the test and diagnosis program, with the solution already presented in [20]. The main contribution of this paper lies in showing how the programmable BIST approach can be implemented in a P1500 compliant core, and in practically demonstrating its advantages in terms of flexibility: in fact, the proposed method is particularly suited to minimize the design effort for customizing the general solution to the memory core under test, and easily fits into an existing test and diagnosis environment.

Section 2 describes the design environment and the constraints existing when dealing with the test and diagnosis of embedded memory cores. Section 3 describes the proposed architecture in terms of hardware organization, internal processor instruction set, and wrapper description. Section 4 outlines the test algorithm and its diagnostic features; Section 5 reports some experimental results gathered on the considered case study and evaluates the cost of the approach in terms of area overhead. Section 6 draws some conclusions.

2. Background

The design flow adopted by STMicroelectronics includes BIST solutions for testing embedded memories since several years. These solutions support not only the test, but also the diagnosis of embedded memories; the goal of the diagnosis process is mainly to facilitate the tuning of the production process, and we will not address in this paper the issue of exploiting diagnosis for memory repair.

The main constraints to be fulfilled by any test solution are:

- the area overhead for supporting diagnosis can not be much higher than the one for pure test

- the analysis of extracted information required to identify error location must be performed off-line on the ATE
- the adopted ATE should be as cheap as possible
- the effort for integrating cores and for developing the test program for the resulting SOC should be minimized
- the compatibility with previous test solutions should be as high as possible.

For this purpose the adopted architecture should support a way to:

- extract from each memory core detailed information about each March test execution (not only the usual go/nogo response): as an example, the number of detected errors should be made available to the outside, as well as the address and faulty word corresponding to the last detected error
- customize the March algorithm in order to support the extraction of the above information not only at the end of its execution, but even during it. For this purpose, some mechanism similar to breakpoints is implemented, so that the test algorithm can be stopped when required, and the desired information can be extracted.

A previous solution fulfilling the above requirements was presented in [20]. The architecture was based on a layered architecture allowing ATEs extracting from each memory core executing BIST a sufficient amount of information to achieve fault classification and identification. The BIST layer implemented a March algorithm which was (possibly) repeated several times in order to extract detailed information about all the detected errors. The BIST architecture allowed the ATE to send commands specifying the number of read/write operations the algorithm was asked to execute: once the algorithm reached this number, it had to stop, so that the ATE could access information about the last (if any) detected error, and send new commands.

The external layer of each memory core is a P1500 wrapper; which is in charge of interacting with the ATE (e.g., by receiving commands and sending diagnostic information) by means of the TAP controller through a suitable TAM. Different cores are allowed to exist in the same SOC thanks to the adoption of a serial TAM connecting all of them. An environment including a set of software applications and tools (like the one described in [21]) was developed and successfully evaluated within STMicroelectronics for such architecture.

In order to guarantee the compatibility with such environment and to allow the continuous usage of the software applications developed for it, the general test interface of the core must remain unchanged. In the following, a brief outline of the core test interface is provided.

The external ATE controls the test and diagnosis execution by means of a *Test Access Port* (TAP) IEEE 1149.1 standard compliant interface. The TAP allows the ATE to send the commands for executing the test and diagnosis algorithm and for reading the results. A *TAP Controller* decodes the commands sent by the ATE through the TAP and sends them to the BIST module through the Wrapper, playing the role of interfacing the ATE with the wrapper, and hence with the BIST module.

The Wrapper is in charge of interacting with the BIST module and to provide higher-level facilities, which can be activated from the outside through the TAP interface. These facilities correspond to TAP instructions and allow starting, pausing, resetting, and continuing the March algorithm, and extracting the information concerning the last detected error. Thanks to these facilities, the ATE can gather all the diagnostic information that are usually accessible when the memory is not embedded (i.e., the output value produced by a read operation executed by the March algorithm).

3. The programmable BIST architecture

The new approach modifies the hardware architecture presented in [20] by substituting the circuitry implementing the March algorithm with a custom test processor able to execute a test program. The new architecture, shown in Fig. 1, is still fully controlled by means of an external interface compliant with the current test standards (IEEE 1149.1 and P1500).

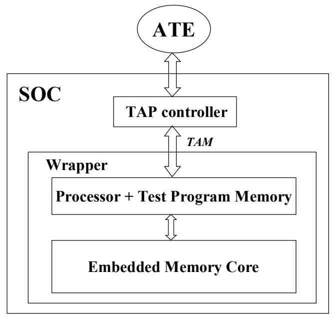

Fig. 1: General Test Architecture.

3.1. The processor

The processor *internal architecture*, as shown in Fig. 2, is divided into 2 functional blocks: a *Control Unit* to manage the test algorithm and a *Memory Adapter* to apply it to a specific memory core. Splitting the processor in two parts allows reducing the cost for its extension to new cores.

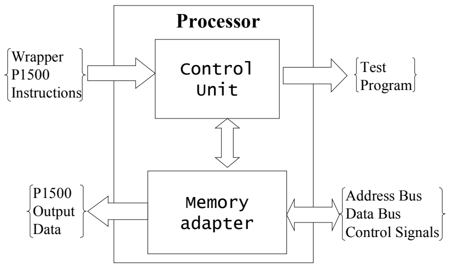

Fig. 2: Processor internal architecture.

The *Control Unit* manages the test program execution; it receives the start command and generates the control signals that the Memory Adapter executes on the memory core under test. The Control Unit includes an *Instruction Register* (IR) and a *Program Counter* (PC). By means of control commands, the Control Unit allows the correct update of some registers located in the Memory Adapter and devoted to customize the test and diagnosis procedures. This choice simplifies the processor reuse in different applications without the need for any re-design, e.g., the execution of a different test program, the test of a memory with a different size or the test of different memory models.

The *Memory Adapter* includes all the test and diagnosis registers used to customize and correctly execute the March algorithm:

- the Control Address register (*Current_address)*: it contains the address of the currently accessed memory cell;
- The Control Memory registers:
  - *Current_data*: it contains the data to be written into the memory during the current read/write operation;
  - *Received_data*: it contains the data read from the memory;
- The Control Test registers:
  - *Dbg_index*: it contains the index to access to the databackground register file
  - *Step*: it contains the number of steps to be executed. The size of this register is $log_2 M$, being M the total number of read/write operations executed by the March algorithm
  - *Direction flag*: a bit specifies the direction (*forward* or *backward*) of the March Element
  - *Timer*: it contains the number of waiting clock cycles in order to introduce pauses into the test algorithm execution.
- The Result registers:
  - *Status* (Status Register): it contains 2 bits; `E` is active when the March algorithm has reached its end, `S` is active when the BIST algorithm reaches the number of steps to be executed
  - *Err* (Error Register): it counts the number of times a fault is detected in a cell (i.e., when at least a bit in the cell is faulty); its size is equal to $log_2 M$ bits.
  - *Result* (Result Register): it contains the information concerning the last detected fault. The stored data are:
    - `STEP`, the ordinal number of the operation which has just been executed; its size is $log_2 M$ bits
    - `DATA`, the logical exor between the read and the expected words; its size is n bits, being n the parallelism of the memory.

The Memory Adapter is also composed of a set of registers containing constant values defined at the design step according to the characteristics of the memory under test and to the test algorithm:

- *Add_Max* and *Add_Min*: the first and the last addresses of the memory under test, respectively;
- *DataBackGround*: it contains the databackground set of values in use during the test cycle;
- *Dbg_max*: it contains the reference to the databackground value in use.

The functional modes of the processor are the following:

- *normal*, the processor is inactive; this is the default mode during the system normal mode.
- *reset*: it is entered when the `RESET` P1500 instruction is sent; this instruction activates the processor, so that it becomes active and ready to run the program
- *run*: it is entered when the `RUNBIST` P1500 instruction is sent, which forces the processor to start the program execution.

The instruction set has been designed to support the widest range of March algorithms and to guarantee high flexibility and adaptability to the processor. Detailed information about the set of instructions supported by the processor are reported in Table 1.

Instruction	Meaning
SET_ADD	Current_address ⇐ Add_Max Direction flag ⇐ BACKWARD
RST_ADD	Current_address ⇐ Add_Min Direction flag ⇐ FORWARD
STORE_DBG	Current_data ⇐ DataBackGround [Dbg_index] Dbg_index ⇐ Dbg_index + 1
INV_DBG	Current_data ⇐ NOT (Current_data)
READ	Current_data ⇐ Memory[Current_address]
WRITE	Memory[Current_address] ⇐ Current_data
BNE Offset	if (Direction flag = BACKWARD) then { if (Current_address <> Add_Min) then { Program Counter⇐Program Counter-Offset Current_address⇐ Current_address - 1 } else Program Counter = Program Counter + 1 } else { if (Current_address <> Add_Max) then { ProgramCounter⇐ProgramCounter-Offset Current_address⇐ Current_address +1 } else Program Counter ⇐ Program Counter + 1 }
LOOP Offset	Dbg_index = Dbg_index+1 if (Dbg_index < Dbg_max) then Program Counter = Program Counter – Offset else Program Counter + Program Counter + 1
SET_TIME data	Timer ⇐ data
PAUSE	Timer = Timer - 1 if (Timer = 0) then Program Counter = Program Counter + 1
END_CODE	Functional Mode ⇐ Normal

Tab. 1: Processor Instruction Set.

3.2. Test program memory

A test program memory exists in the wrapper, storing the code to be executed by the processor to implement the adopted March algorithm on the memory under test.

3.3. The wrapper

The wrapper, shown in Fig. 3, contains the necessary circuitry to interface the processor with the outside in a P1500 compliant fashion, supporting the commands for running the BIST and accessing to its results. The wrapper is compliant with the suggestions of the P1500 standardization group.

In addition to the mandatory components, the wrapper architecture includes the following Wrapper Data registers:

- *Wrapper Control Data Register* (*WCDR*): through this register the TAP controller sends the commands to the processor (e.g., the processor reset, the test program start, the result registers read, etc.).
- *Wrapper Data Register* (*WDR*): it is an I/O buffer register. The TAP Controller can read the diagnostic information stored in the result registers (*Status*, *Err* and *Result*). According to the command written in WCDR the outside world may execute one of the following operations involving WDR:
  - read from WDR the diagnostic information (i.e., the number of detected errors, the step corresponding to the last detected error and the faulty word) stored into the result registers
  - write in WDR the number of steps to be executed by the March algorithm to be written into the *Step* register.

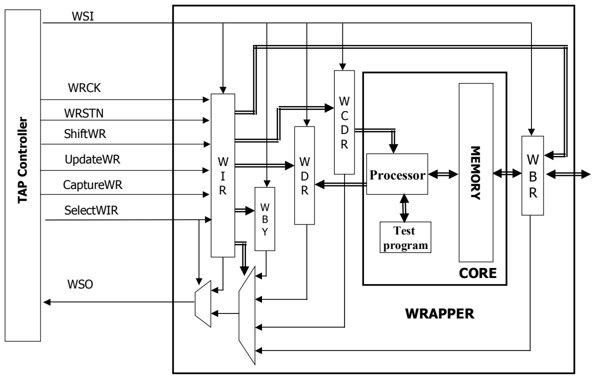

Fig. 3: The proposed Wrapper Architecture.

3.4. The TAP Controller

The IEEE 1149.1 compliant TAP Controller plays the role of interfacing the ATE with the wrapper, and hence with the test module. The TAP Controller supports the following instructions (beyond the standard ones, such as BYPASS):

- `RESET`: puts a core into the reset state;
- `RUNBIST`: executes a complete run of the March program;
- `LOADSTEPS`: loads the number of operations to be executed by the March algorithm into the *Step* register; by default this number is equal to the number of operations required by a complete execution of the adopted March algorithm
- `READSTATUS`: reads the *Status* register and verifies whether the March algorithm finished its task (either because it reached its end, or because it executed the specified number of operations)
- `READRESULT`: reads the *Result* register containing the information about the last error detected by the March algorithm
- `READERROR`: reads the *Err* register containing the number of errors detected by the March algorithm.

4. The Test and Diagnosis Program

As reported in Section 2, one of the main constraints considered in the test architecture selection was its compatibility with a low cost ATE.

4.1. ATE software architecture

The ATE software module performs a test or a diagnosis session. According to the scheme shown in Fig. 4, the ATE is composed of 2 modules: a Test Program and a Bitmap Generator. The Test Program is in charge of controlling the test and diagnosis execution. It is composed of 2 tasks: a Run BIST task and a Response Analysis one. The Bitmap Generator program represents a higher-level procedure in charge of identifying fault(s) based on the collected information e.g., by exploiting the approach described in [21].

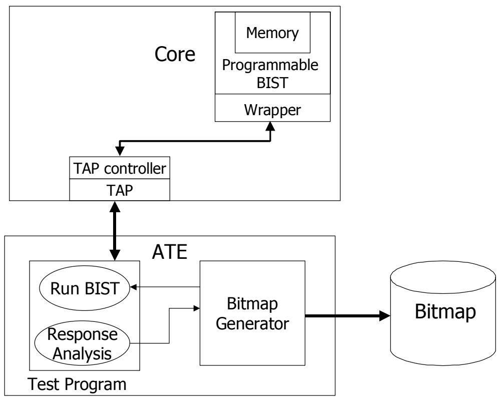

Fig. 4: ATE General Software Architecture.

4.2. Typical test and diagnosis procedure

When test is the only goal, using the retrieved information about the number of detected errors one can separate fault-free from faulty chips, and the process ends. In the case of testing:

- the Run BIST task sends the `RESET` instruction, which initializes the processor in the core and the `RUNBIST` instruction, which starts the execution of the test program.
- The Response Analysis task waits until the completion of the test program by polling the status of the BIST module using the `READSTATUS` instruction and accesses the result through the `READERROR` instruction.

When diagnosis is the concern, the ATE gathers all the information about each error detected by the test program on the faulty embedded memory. The ATE attains this goal by executing the following operations:

1. the Run BIST task sends the `RESET` and `RUNBIST` instructions as before.
2. the Response Analysis task executes the procedure shown in Fig.5:
   a. it waits until the completion of the test program by polling the status of the BIST module using the `READSTATUS` instruction
   b. it accesses the result through the `READERROR` instruction.
   c. the information about the last detected error is retrieved through `READRESULT`.
   d. the Bitmap Generator receives the information stored into the *Result* register and computes the address of the faulty memory cells and their faulty bit cells, updating the memory bitmap
3. The Run BIST task executes the `RESET` instruction and the `LOADSTEPS` instruction updating the number of operations to be executed by the March algorithm according to the data stored into the STEP field of the *Result* Register; the program execution is again launched through the `RUNBIST` instruction.
4. Steps 2 and 3 are iteratively repeated for all the detected errors, and the related information are then extracted. Thanks to the algorithm adopted in the Bitmap Generator module the diagnosis process normally reaches its goal (i.e., identifying the fault type and location) before extracting the information about all the detected errors.

```
REPEAT
    wait
    READSTATUS
UNTIL S=FALSE
ERR = READERROR
if ( ERR > 0 )
    {
    RESULT = READRESULT
    Insert_Bitmap(RESULT)
    step = RESULT.STEP - 1
    LOADSTEPS
    }
```

Fig. 5: Response Analysis task basic procedure.

The above procedure guarantees that faults can be identified with the maximum diagnostic capability allowed by the March algorithm implemented by the embedded processor. This means that all faults that could be diagnosed by the adopted March algorithm when directly implemented by an ATE having full access to the memory are still diagnosable by the described architecture.

4.3. Test program flexibility

The Programmable BIST approach allows to easily adapting the test algorithm to the specific test requirements.

The Instruction Set allows executing all the possible March Test algorithms. The instructions `SETTIME` and `PAUSE` can be inserted into the test program to detect data retention faults as reported in the example shown in Tab. 2.

Different databackground (0,1, checkerboard, etc.) can be adopted: a suitable set of values into the *DataBackGround* register file have to be defined at the design level.

5. Experimental evaluation

The core test architecture described above is currently being evaluated on a sample chip including a 16 K × 16 bits embedded memory. The chip is currently being manufactured by STMicroelectronics using a mixed/power 0.18 μm library.

In the current implementation, the adopted March algorithm is the same as in the previous version of the core [20]. The test program resides in a ROM memory, whose size is strictly dependent on the selected March algorithm. The size of the test program for the 12N adopted March algorithm is 43 4-bit words.

The core implementing the proposed test processor architecture was modeled in VHDL for an amount of about 3,000 lines of code. We then synthesized it with Synopsys Design Compiler.

The total area occupied by the additional logic for test and diagnosis is reported in Tab. 3. The Memory Adapter introduces the largest overhead due to the test registers it includes. The total area overhead introduced by the programmable BIST amounts to about 2.1 % of the memory area. In Tab. 3 the TAP Controller and the TAP have not been considered since they are not related to a single core, but shared among multiple cores present in the SOC. Anyway, their size amounts to about 800 gates.

It is interesting to note that the programmable BIST approach proposed in this paper requires a negligible extra area overhead with respect to the one introduced by the hardwired BIST approach [20] which requires 6,913 gates for the same memory type with the same technology.

The proposed approach is also comparable in terms of area overhead with a different BIST and self diagnosis approach [7], where for a 128 Kb SRAM memory the area overhead amounts to 2.4% of the memory area.

The programmable BIST approach does not introduce any temporal overhead and it guarantees an at-speed test with a 40Mhz clock frequency.

6. Conclusions

We described a solution exploited by STMicroelectronics for test and diagnosis of embedded memories. The proposed solution is based on a P1500-compliant wrapper, which implements a programmable BIST approach. Being programmable, the wrapper can be easily reused to implement different test algorithms. Diagnosis is supported by allowing the external ATE to interact with the wrapper and force the test algorithm to start, stop, pause, reset, etc., and to extract information about the detected errors. The proposed solution has been evaluated on a case study, and experimental results are reported comparing its benefits and limitations with alternative approaches. The main novelty of the proposed solution lies in the integration of a programmable BIST architecture in a P1500 wrapper, showing how to effectively implement diagnosis on the resulting core with a low-cost tester and with minimum efforts in terms of test program development.

March symbol	Instruction	Address	Code
⇑	`RST_ADD`	N	0110
{	`INV_DBG`	N+1	1000
R1	`READ`	N+2	1001
	`INV_DBG`	N+3	1000
W0	`WRITE`	N+4	1010
}	`BNE -5`	N+5	1101
		N+6	0101
	`SETTIME 255`	N+7	0001
		N+8	1111
		N+9	1111
	`PAUSE`	N+10	1111
⇓	`SET_ADD`	N+11	1110
{R0	`READ`	N+12	1001
	`INV_DBG`	N+13	1000
W1	`WRITE`	N+14	1010
	`INV_DBG`	N+15	1000
}	`BNE -5`	N+16	1101
		N+17	0101

Tab. 2: A portion of the Test Program.

Component	Programmable BIST [# of equivalent gates]
Wrapper	2,944
Control Unit	760
Memory Adapter	4,027
ROM	220
TOTAL	7,951

Tab. 3: Area overhead evaluation.

7. References

[1] Y. Zorian, E.J. Marinissen, and S. Dey, *Testing embedded-core based system chips*, in Proc. International Test Conference, Oct. 1998, pp. 130-143

[2] E.J. Marinissen, Y. Zorian, *Challenges in Testing Core-based System ICs*, IEEE Communications Magazine, Vol. 37, June 1999, pp. 104-109

[3] N. Touba, and B. Pouya, *Using Partial Isolation Rings to test Core-Based Designs*, IEEE Design and Test of Computers, vol. 14, Oct.-Dec. 1997, pp. 52-59

[4] S.K. Goel, E.J. Marinissen, *Effective and efficient test architecture design for SOCs*, in Proc. International Test Conference, Oct. 2002, pp. 529-538

[5] E.J. Marinissen, S.K. Goel, M. Lousberg, *Wrapper design for embedded core test*, in Proc. International Test Conference, Oct. 2000, pp. 911-920

[6] T. Bergfeld, D. Niggemeyer, E. Rudnick, *Diagnostic testing of embedded memories using BIST*, IEEE Conference on Design, Automation and Test in Europe 2000, pp. 305 –309

[7] C.W. Wang, C.-F. Wu, J.-F. Li, C.-W. Wu, T. Teng, K. Chiu, H.-P. Lin,, *A Built-In Self Test and Diagnosis Scheme for Embedded SRAM*, IEEE Asian Test Symposium, 2000, pp. 45-49

[8] H.Kim, Y. Zorian, G. Komoriya, H. Pham, F. Higgins, J. Lewandowski, *Built-In Self-Repair for Embedded High Density SRAM*, IEEE International Test Conference, 1998, pp.1112-1119

[9] A.J. van de Goor, Using March Tests to Test SRAMs, IEEE Design & Test of Computers, Vol. 10, No. 1, pp. 8-14, 1993

[10] B.F Cockburn, *Tutorial on semiconductor memory testing, Journal of Electronic Testing: Theory and Application*, Vol. 5, pp. 321-336, 1994

[11] Hetherington et al., *Logic BIST for large industrial designs: Real issues and case studies*, IEEE International Test Conference, 1999, pp. 358-367

[12] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, T.-Y. Chang, *A programmable BIST core for embedded DRAM, IEEE Design and Test of Computers*, Vol. 16, No. 1, pp. 59-70, 1999

[13] A.J. van de Goor, *Testing Semiconductor memories: Theory and Practice*, ComTex Publishing, Gouda, The Netherlands, 1998

[14] R. Treuer, and V.K. Agarwal, *Built-In Self Diagnosis for Repairable Embedded RAMs*, IEEE Design and Test of Computers, Vol. 10, No. 2, June 1993, pp. 24-33

[15] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, T.-Y. Chang, *A Programmable BIST Core for Embedded DRAM*, IEEE Design and Test of Computers, Vol. 16, No. 1, Jan.-March 1999, pp. 59-70

[16] C.-H. Tsai, C.-W. Wu, *Processor-Programmable Memory BIST for Bus-Connected Embedded Memories*, in Proc. Design Automation Conference, 2001, pp. 325-330

[17] J. Dreibelbis, J. Barth, H. Kalter, R. Kho, *Processor-based Built-In Self-Test for Embedded DRAM*, IEEE Journal of Solid-State Circuits, Vol. 33, No. 11, Nov. 1998, pp. 1731-1740

[18] P. Bernardi, M. Rebaudengo, M. Sonza Reorda, M. Violante, *A P1500-compatible programmable BIST approach for the test of Embedded Flash Memories*, Design, Automation and Test in Europe Conference, 2003, pp. 720-725

[19] E.J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, L. Whetsel, *Towards a Standard for Embedded Core Test: An Example*, IEEE International Test Conference, 1999, pp. 616-627

[20] D. Appello, F. Corno, M. Giovinetto, M. Rebaudengo, M. Sonza Reorda, *A P1500 compliant architecture for BIST-based Diagnosis of embedded RAMs*, IEEE Asian Test Symposium, 2001, pp. 97-102

[21] D. Appello, A. Fudoli, V. Tancorre, F. Corno, M. Rebaudengo, M. Sonza Reorda, *A BIST-based Solution for the Diagnosis of Embedded Memories Adopting Image Processing Techniques*, IEEE On-Line Testing Workshop, 2002, pp. 112-116