A P1500-compatible programmable BIST approach for the test of Embedded Flash Memories

P. Bernardi, M. Rebaudengo, M. Sonza Reorda, M. Violante

Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
http://www.cad.polito.it/

Abstract

In this paper we present a microprocessor-based approach suitable for embedded flash memory testing in a System-on-a-chip (SOC) environment. The main novelty of the approach is the high flexibility, which guarantees easy exploitation of the same architecture to different memory cores. The proposed approach is compatible with the P1500 standard. A case study has been developed and demonstrates the advantages of the proposed core test strategy in terms of area overhead and test application time.

1 Introduction

Continuous technological improvements allow designing a complex system into a single chip (System-On-Chip or SOC). A SOC is composed of different reusable functional blocks, called *embedded cores*. Typical embedded cores include processors (such as CPU, DSP, and microcontrollers), memories (such as ROM, DRAM, SRAM, and flash), I/O devices, etc. System designers can purchase cores from core vendors and integrate them with their own User-Defined Logic (UDL) to implements SOCs. Core-based SOCs present important advantages: the size and the cost of the end-product is decreased, and thanks to the design re-use, the time-to-market is greatly reduced.

Conversely, testing a core-based SOC is a major challenge [1]. The main problem is that accessibility to cores and UDL is greatly reduced. Traditional approaches [1-2] for testing core-based SOCs completely rely on additional Design for Testability (DfT) structures such as test busses for test transfers 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 (TAM) to connect cores to the test source and sink. A critical 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. Some approaches have been proposed to solve this problem. A class of test approaches [3-4] adopts the reuse of existing functionalities for test access. These methods assume that every core has a transparent mode in which data can be propagated. However, these methods are not general enough to handle all the possible kinds of cores and all the possible test schemes, such as scan or BIST. Some researchers [5-10] proposed to exploit an embedded processor to test the other components of the SOC: first the processor core is tested (e.g., by means of functional patterns or full-scan test), and then a test program, executed by the embedded processor, is used to test the on-chip memories and other cores. The use of embedded processors to test cores presents many advantages: the size of the test controller is normally negligible, the test program (being in software) guarantees a high flexibility, and the testing process can often be done at-speed. Moreover, the test process is done inside the chip and the tester can work at a lower speed, thus reducing the costs for the test equipment. The main disadvantage is related to the need for an on-chip processor and to the dependence on the one possibly present in the SOC. In case of different processors, the test program has to be adapted to each of them causing an increased cost in the test development. Moreover, this approach cannot be applied to embedded cores not suitably connected to the processor.

In this paper we propose a core test strategy based on a custom processor wrapped to the embedded core under test. From the test point of view, the core complies to the P1500 standard, thus easing the connection of the core to other test resources on the chip (e.g., a 1149.1 TAP controller for accessing the SOC from outside). The advantages of this solution are mainly the low area overhead introduced, the negligible costs in terms of wirings, and (most importantly) the high flexibility that it guarantees. Although the approach is general, in this paper we focus on the test of flash memories, which are becoming widely used in many applications, especially thanks to their ability to be electrically programmed and erased, combining the advantages of non-volatile and volatile memories. A characteristic of flash memories is

1530-1591/03 $17.00 © 2003 IEEE

the significant difference among flash memories models in terms of size, programming, and timing characteristics. The proposed method is particularly oriented to minimize the design effort to customize the test solution to the flash memory model under test and to the specific test constraints. A case study has been developed to demonstrate the advantages of the proposed core test strategy in terms of area overhead and test application time.

Section 2 summarizes the possible architectural solutions for core-based SOC testing and introduces the one adopted in our work. Section 3 presents the background about flash memory architectures, the considered fault model and test algorithms; Section 4 describes the proposed test architecture in terms of internal organization, instruction set, and external layers. Section 5 describes a case study. Finally, Section 6 draws some conclusions.

2 Memory core test architectures

Testing core-based SOCs is a complex problem that can be divided in two parts: core-level and chip-level testing. Core-level testing involves making each core testable, i.e., inserting the necessary test structures and generating test sequences. Chip-level testing involves defining a test access mechanism architecture in order to apply the test sequences to the input peripheries of the cores, and to propagate the test responses from the core outputs to the chip outputs.

In this paper we mainly focus on the core-level testing problem, resorting to a custom TAM for connecting the core to a standard 1149.1 TAP controller, thus providing an easy interface between SOC and an external ATE.

Different approaches can be adopted for testing memory cores; they are normally based on BIST solutions, and grouped in the following classes:

- *Hardwired BIST*
- *Soft BIST*
- *Programmable BIST*.

The *hardwired BIST* approach is the most widely used for testing embedded memories. It consists in adding a custom circuitry to each core, implementing a suitable BIST algorithm. This approach deeply exploits the many research efforts spent in developing memory BIST architectures [8-9]. The main advantage of this approach is that the test application time is short and the area overhead is relatively small: as an example, by adopting the approach proposed in [9], with a 16 Mbit DRAM embedded memory the area overhead is less than 0.3%. 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 a very low flexibility: any modification of the test algorithm requires a re-design of the BIST circuitry. Moreover, since each memory core has its own BIST circuitry, in the case of multiple cores on the same SOC the overall area overhead corresponds to the sum of all the areas occupied by the BIST circuits.

The *soft BIST* approach [10] exploits a processor already available in the SOC (if any) to execute the test of all the other cores and the UDL. A test program, executed by the processor, applies a test pattern sequence to the cores under test and possibly checks for the results. The entire test is stored in a memory containing the test program and possibly the test patterns. This approach uses the system bus for applying test patterns and reading test responses, and for this reason 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 on the available (if any) processor. The test program has to be adapted to the available processor: the core vendor needs to develop for the same core different test programs, one per each processor family, thus increasing the test development costs. Moreover, the intellectual property is not well protected, as the core vendor supplies to the user the test program for the core under test. The test application time may also depend on the test processor: if its clock frequency is not fast enough, at-speed test cannot be performed, and the overall test application time is also increased. Finally, this approach can be applied only to cores directly connected to the system bus: the approach cannot be applied if the core is connected to different system buses, or if the processor accesses the cores via other logic (e.g., when a memory controller exists) and the core is not completely controllable and observable.

An alternative approach, which is a mix of the previous two, is the one usually denoted as *programmable BIST*. The core vendor develops a DfT logic, which wraps the core under test and includes a very simple custom microprocessor, which is exclusively devoted to test the core. The advantages of this architecture are manifold: the intellectual property can be protected (such as in the hardwired case), only one test program has to be developed, and the design cost for the test is very reduced; the technique is significantly flexible and any modification of the algorithm simply requires a change in the test program; the test application time can be reduced thanks to the efficiency of the custom test processor, and the test can consequently be executed at-speed. 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 can be considered marginal, as shown in this paper.

The approach we proposed improves the one described in [11], since the test processor is in charge of the test execution, without the need of any additional logic for test. Despite being based on a microprocessor, the method is compatible with the IEEE P1500 [12] standard. As a first step, we applied the method to the test of a flash memory core, although its extension to embedded RAM cores is already under way. Our architecture is particularly efficient in minimizing the temporal overhead introduced by the program execution. Moreover, the processor (described in Section 4), has been designed in such a way that can be modified and adapted to the test different flash memory models. Finally, the proposed architecture is accessible through the test standard interfaces (IEEE 1149.1 and P1500) in order to make the test completely accessible from the outside: the test of the core can thus be executed by means of 1149.1 instructions and the core test details are completely transparent to core designers.

3 Flash memory background

A generic Flash Memory is organized as a NOR or NAND array (like conventional RAMs) with a grid of row and column lines connected by Floating Gate transistors. A detailed description of different cells and their own structures is presented in [13].

In addition to the classical fault models considered in the RAM memory test, such as Stuck-At Faults (SAFs), Coupling Fault (CFs), and Data Retention Faults (DRFs), flash memories can suffer disturbances that do not conform to any of these fault models. During programming, typical faults or disturbances in flash memories occur to cells that share a row or a column with another cell that is being programmed. Possible disturb mechanisms for NOR-type flash memories [14-16] include:

- Gate Program Disturbance (GPD) and Gate Erasure Disturbance (GED): a cell program operation causes the erroneous programming or erasing, respectively, of a second cell in the same word line;
- Drain Program Disturbance (DPD) and Drain Erasure Disturbance (DED): a cell program operation causes the erroneous programming or erasing, respectively, of a second cell in the same bit line;
- Read Disturbance (RD): a cell read operation causes the erroneous programming of the read cell;
- Over Erase (OE): a cell program operation has no effect on the interested cell.

Disturbance faults occur only into the same row or column of the programmed/erased cell. In [15] a simplified coupling fault model is presented, where flash disturbances are modeled as One-Way Coupling faults, since they can be sensitized using a ↓ transition.

Considering this fault model allows the adoption of test algorithms for coupling faults able to detect all types of flash disturbances. A *March Test for Flash Memory (March FT)* for NOR-type stacked-gate flash memories [16] can be summarized as follows:

{(f);↓(r1,w0,r0);↕(r0);(f);↑(r1,w0,r0);↕(r0)}

In order to cover intra-word coupling faults and intra-word Gate Program/Erase Disturbances different data values, called *databackground*, must be considered.

An evaluation of the different flash memory test algorithms has been provided in [16]. The March FT algorithm covers all the SAF, OE, DPD, DED, RD and almost all the GPD and GED faults. The complexity is equivalent to $2F + 2N*P + 6N*R$, where R, P and F represent the number of read, program and flash operations, respectively, and N is the number of words. Usually, the read operation is the fastest one, while the flash operation is the slowest one. Conversely, the word-oriented March FT Test covers all the possible faults with the considered fault models and its test length is equivalent to $(2F + 2N*P + 6N*R) + (2F + 2N*P + 2N*R) * (DBG - 1)$, where DBG corresponds to the databackground set length.

4 The microprocessor-based approach

The approach proposed in this paper is based on a custom microprocessor in charge of executing a pseudo-March algorithm oriented to flash memory test. The proposed microprocessor has been designed to allow high flexibility in order to achieve two goals: on the one hand we want to simplify the introduction of any modification in the test program; on the other hand we aim at adopting the same architecture to a large set of flash memory models. The proposed architecture exploits the current test standard interface in order to simplify the design effort. The processor is not directly visible from the outside, since the core is P1500 compliant, and includes a P1500 wrapper.

4.1 The microprocessor

The microprocessor *internal architecture* is divided into the following functional blocks:

- a *Control Unit* to manage the test algorithm;
- a *Memory Adapter* to manage the main test registers;
- a *Flash Manager* to manage the memory access timing.

This organization allows reducing the re-design operations and supports the reuse of the internal structures: only the Flash Manager module has to be adapted to a different flash memory model while letting the rest of the processor unchanged.

The **Control Unit** manages the test program

execution; it gets the start command and generates the signals for the Flash Manager and the memory containing the test program. This module manages the *Instruction Register* (IR) and the *Program Counter* (PC). By means of control signals, the Control Unit allows the correct update of the Memory Adapter test registers. This choice allows its easy reuse without any re-design in different applications like the execution of a different test program or the test of a memory with a different size or the test of different Flash memory models.

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

- the Control Address registers:
  - *Current_address*: it contains the current memory address value;
  - *Add_Max* and *Add_Min*: constant values storing the first and the last addresses of the memory under test.
- The Control Flash registers:
  - *Current_data*: it contains the data to be written into the memory during a program operation;
  - *Received_data*: it contains the data read from the memory;
  - *DataBackGround*: a register file containing the databackground set in use during the test cycle;
  - *Dbg_max*: a constant value containing the reference to the databackground value in use;
  - *Dbg_index*: it contains the index used to access to databackground register file.
- The Result register: a *Status_register* contains 2 flags signaling that at least an error has been detected and the end of the test program, respectively.

The size of the Control Address and Control Flash registers and the constant values (such as *Add_Max*, *Add_Min* and *DataBackGround*) are set according to the characteristics of the memory under test.

The **Flash Manager** needs to be customized to the flash memory under test, due to its strict dependence on the memory model used. It manages the specific control and access timing signals of the memory, while its behavior is controlled by the Memory Adapter. This protocol is suitable for using two distinct clock signals: the *external clock* connected to the Control Unit and Memory Adapter and an *internal clock* connected to the Flash Manager. While the external clock is provided by the SOC and can present higher frequency in order to speed up the program execution, the internal clock is used during memory access operations and must be chosen to properly satisfy the timing requirements of the memory under test. This approach increases the design flexibility enabling the execution of the test operation at a frequency not dependent on the memory model.

4.2 The instruction set

The instruction set has been designed to support all pseudo-March algorithms and to guarantee high flexibility and adaptability to the microprocessor.

The set of 11 instructions can be grouped in the following classes:

- Instructions working on the *Current_address* register:
  - SET_ADD: it loads into *Current_address* the constant value *Add_Max*;
  - RST_ADD: it loads into *Current_address* the constant value *Add_Min*;
- Instructions working on the *Current_data* register:
  - STORE_BG_DATA: it loads into *Current_data* the value contained into the *DataBackGround* register file and pointed by *Dbg_index*. After that, the *Dbg_index* value is incremented;
  - INV_BG_DATA: it inverts the current value of *Current_data*;
- Instructions executing flash memory I/O operations:
  - READ_FLASH: it executes a read operation at the word addressed by *Current_address*; the extracted data is stored in the *Received_data* register and compared with the expected value: if these two values are different, the *Status_register* is updated;
  - PROG_FLASH: it executes a program operation writing the *Current_data* value at the word addressed by *Current_address*;
  - ERASE_FLASH: it executes a flash operation on memory;
- Instructions executing the program control flow:
  - BNE_ADD: it executes a jump if *Current_address* has not reached the address limit (stored in *Add_Max* or *Add_Min*, according to the direction of the current March Element). A 4-bit operand stores the jump offset. If the condition is not verified, *Current_address* is increased (or decreased) according to the current March Element direction and the Program Counter is modified according to the operand value, otherwise the Program Counter is increased in order to execute the following instruction
  - LOOP_DBG: it increments *Dbg_index* value and executes a jump if the *Dbg_index* value hasn't reached the *Dbg_max* value. 2 4-bit operands store the jump offset. If the condition is not verified the *Dbg_index* value is incremented and the Program Counter is modified according to the operand value; otherwise the Program Counter is increased in order to execute the following instruction
  - END_CODE: it concludes the test program, changing the processor state to *idle*.

4.3 Wrapper module

The wrapper, shown in Fig. 1, contains the circuitry necessary to interface the test processor with the outside in a P1500 compliant fashion, supporting the commands for running the BIST operation and accessing to its results. The wrapper is compliant with the suggestions of the P1500 standardization group [12].

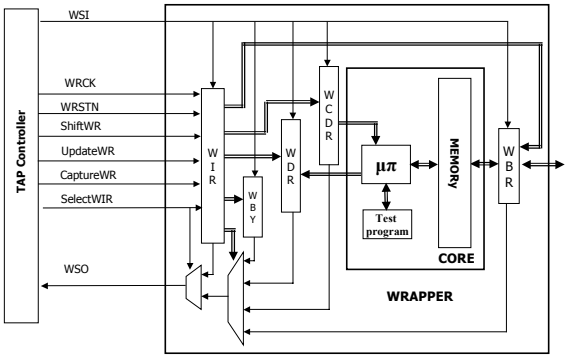

Fig. 1: The proposed Wrapper Architecture.

In addition to the mandatory components we propose the introduction of the following Wrapper Data registers:

- *Wrapper Control Data Register* (*WCDR*): through this register the TAP controller sends the commands to the microprocessor (e.g., the microprocessor reset, the test program start, the Status register read, etc.).
- *Wrapper Data Register* (*WDR*): it is an output register. The TAP Controller can read the test information from the microprocessor stored into the status register.

5 The case study

In order to evaluate the feasibility of the proposed architecture, the M50FW040 device produced by STMicroelectronics has been considered as a case study: the size of this flash memory model is 4 Mbit and it is divided into 8 blocks with a 8 bit word parallelism. The M50FW040 device presents some particularities that imply a specific Flash Manager design:

- specific codes are needed to access the memory;
- the addressing phase is divided into two distinct steps due to the presence of only 10 address bits: the address is provided by sending first the 10 less significant bits, then the 9 remaining bits.

The test algorithm chosen to test the M50FW040 device is the word-oriented March-FT one, introduced in Section 3.

In the proposed example, the test program resides in a EPROM memory: this kind of approach permits an on-the-fly test algorithm modification.

As far as the processor architecture design is considered, we have that the Control Unit is independent from the memory model. The size of its registers has been chosen according to the general microprocessor architecture:

- IR on 4 bits, according to the instruction format;
- PC on 8 bits, according to the test program length.

The Memory Adapter has been tuned according to the M50FW040 device characteristics as following:

- data registers (*Current_data*, *Received_data*) on 8 bits according to the memory under test data parallelism;
- address registers (*Current_address*, *Add_Max* e *Add_Min*) on 19 bits according to the memory under test size;
- *DataBackGround* is a vector of 4 (*Dbg_max*) bytes with *Dbg_index* on 2 bits
- the clock frequencies of the external and internal clocks have been set to 20 MHz.

In order to test the additional DfT logic a full scan approach has been adopted.

The total area occupied by the DfT additional logic is reported in Tab. 1, considering 2 different test algorithms: a word-oriented March FT and a March FT. We modeled the core implementing the proposed test processor architecture in VHDL for an amount of about 3,000 lines of code. We then synthesized it with Synopsys Design Compiler using a simple in-house developed technological library.

The TAM logic (which includes the Wrapper module) represents a fixed cost necessary to manage the chip-level test. Its area overhead can be quantified as the 30% of the global cost of the additional core-level test logic.

The analysis of the microprocessor units underlines the difference among them: the Memory Adapter introduces a larger overhead due to the included test registers, while the Flash Adapter is the smallest component. The simplicity of the Flash Adapter helps the modification and the adaptation of the proposed architecture to new specific designs.

The size of the ROM storing the program is dependent on the selected test algorithm: the size of the test program for the word-oriented March-FT algorithm is 53 4-bit words, and for the basic March-FT the test program length is 33 4-bit words.

Considering the word-oriented March FT algorithm, with respect to M50FW040 model, the overall area overhead stemming from the additional DfT logic is about 0.2% of the full memory area.

Tab. 1 shows also that the application of the basic March FT algorithm causes a marginal reduction of the introduced area overhead, due to the decrease of the test program length and to the use of a single databackground value. This result underlines the high flexibility of the proposed test architecture: a new test algorithm can be introduced by changing only the

databackground description in the Memory Adapter unit and modifying the test program stored in ROM.

Component	Word-oriented March FT [# of gates]	March FT [# of gates]
TAP	786	786
TAP controller	14	14
Wrapper	992	992
Control Unit	624	624
Memory Adapter	1,258	1,242
Flash Manager	307	307
ROM	265	190
TOTAL	4,246	4,155

Table 1: Area overhead evaluation.

In order to evaluate the test application time overhead with respect to an alternative hardwired approach, we executed the test program and then we simulated the application of the same March algorithm directly to the pins of a hypothetical stand-alone chip.

The complete test program execution time requires 6.97 seconds for each block. This time is heavily conditioned by the specific access timing of the M50FW040 model, especially by the erase operation time, which requires about 0.75 sec per block.

The simulation of the application time to test a stand-alone chip equivalent to the same memory flash model requires 6.59 seconds. Comparing the times we can state that the time overhead is limited to about 6% of the complete test time.

In a second set of experiments, considering the March FT algorithm, the internal time overhead grows to about 12% of the complete test time, as a consequence of the reduced number of erase operations executed during the test. We thus experimentally proved that the time overhead does not limit the efficiency of the test, and we can state that an at-speed execution can be supported by our proposed architecture.

6 Conclusions

In this paper we presented a novel technique for efficient testing of embedded memory cores. The proposed approach is based on a custom microprocessor that is only in charge of executing a test algorithm. The proposed microprocessor has been designed to allow high flexibility in order to easily adopt the same processor to a large set of different memory models and easily allow any modification in the test program. The proposed architecture exploits the current test standards interface (IEEE 1149.1 and P1500 standards) in order to simplify the design effort. The test architecture is transparent to the core user, thus guaranteeing the protection of the intellectual property. A case study has been presented, for which we developed a test for an embedded flash memory core environment. The synthesized core adopting the proposed architecture presents a negligible overhead in terms of additional area with respect to the memory size. The test application time is also marginally increased with respect to the stand-alone chip, guaranteeing an at-speed memory test.

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] 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

[3] F. Bouwman, S. Oostdijk, R. Stans, B. Bennetts, and F. Beenker, *Macro Testability: the results of production device applications*, IEEE International Test Conference, 1992, pp. 232-241

[4] I. Ghosh, N. Jha, and S. Dey, *A Fast and Low-Cost Testing Technique for Core-Base System-Chips*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 8, Aug. 2000, pp. 863-877

[5] C.A. Papachristou, F. Martin, and M. Nourani, *Microprocessor based testing for core-based system on chip*, in Proc. Design Automation Conference, 1999, pp. 586-591

[6] R. Rajsuman, *Testing a System-On-a-Chip with Embedded Microprocessor*, in Proc. International Test Conference, Oct. 1999, pp. 499-508

[7] C.-H. Tsai, C.-W. Wu, *Processor-Programmable Memory BIST for Bus-Connected Embedded Memories*, in Proc. of the ASP-DAC 2001, Asia and South Pacific Design Automation Conference, 2001, pp. 325-330

[8] 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

[9] 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

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

[11] 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

[12] 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

[13] B. Pavan, R. Bez, P. Olivo, and E. Zanoni, *Flash Memory cell-An overview*, Proc. IEEE, vol. 85, no. 8, Aug. 1997, pp. 1248-1271

[14] M.G. Mohammed, K.K. Saluja, and A. Yap, *Testing Flash Memories*, in Proc. Int. Conference on VLSI Design, 2000, pp. 406-411

[15] M.G. Mohammed, and K.K. Saluja, *Flash Memory disturbances: modelling and test*, in Proc. Int. Symposium on VLSI Test, VTS 2001, pp. 218-224

[16] J.-C. Yeh, C.-F. Wu, K.-L. Cheng, Y.-F. Chou, C.-T. Huang, C.-W. Wu, *Flash memory built-in self-test using March-like algorithms*, IEEE Int. Workshop on Electronic Design, Test and Applications, 2002, pp. 137-141