International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 3, March - 2013

Scalable Controller Based PMBIST Design For Memory Testability

M. Kiran Kumar, G. Sai Thirumal, B. Nagaveni
M.Tech (VLSI DESIGN)

Abstract

With increasing design complexity in modern SOC design, many memory instances with different sizes and types would be included. To test all of the memory with relatively low cost becomes an important issue. Providing user-defined pattern for screening out various manufacturing defects is also a major demand. To ease the trade-off between the hardware cost and test flexibility, Programmable Built-In Self-Test (P-MBIST) method is an opening approach to complete the memory testing under these circumstances. Many researchers have been focused on P-MBIST design. Processor-based architecture provides high test flexibility, but it increases the test development costs while applying to various processor families. To lower the design cost, a customized processor and instruction have been developed. It uses program memory to store the test program. To further reduce the hardware cost, the instruction can be serially input and saved in one internal register by adopting simple controller. In this project, we implement a hardware sharing architecture to test the memory with same type in parallelism. The proposed method uses only one address counter to generate the required address for March-based algorithm, including row scan and column scan. The controller can be applied to different memory types with the same read/write cycle.

Programmable Built-In Self-Test (P-MBIST) solution provides a certain degree of flexibility with reasonable hardware cost, based on the customized controller/processor. In this work, we propose a hardware sharing architecture for P-MBIST design. Through sharing the common address generator and controller, the area overhead of P-MBIST circuit can be significantly reduced. Finally, the proposed P-MBIST circuit can be automatically generated from the user-defined configuration file.

1. Introduction

With increasing design complexity in modern SOC (System-On-Chip) design, many memory instances with different sizes and types would be included. To test all of the memory with relatively low cost becomes an important issue. Providing user-defined pattern for screening out various manufacturing defects is also a major demand. To ease the tradeoff between the hardware cost and test flexibility, Programmable Built-In Self-Test (P-MBIST) method is an opening approach to complete the memory testing under these circumstances. Many researches have been focused on P-MBIST design. Processor-based architecture provides high test flexibility, but it increases the test development costs while applying to various processor families. To lower the design cost, a customized processor and instruction have been developed. It uses program memory to store the te4st program. To further reduce the hardware cost, the instruction can be serially input and saved in one internal register by adopting simple controller. In this project, we implement a hardware sharing architecture to test the memory with same type in parallelism. The controller can be applied to different memory types with the same read/write cycle.

Programmable Built-In Self-Test (P-MBIST) solution provides a certain degree of flexibility with reasonable hardware cost, based on the customized controller/processor. In this work, we propose a hardware sharing architecture for P-MBIST design. Through sharing the common address generator and controller, the area overhead of P-MBIST circuit can be significantly reduced. Finally, the proposed P-MBIST circuit can be automatically generated from the user-defined configuration file.

2. Supported Testing Algorithms

Many efficient testing algorithms have been proposed to detect different fault models. However, to implement various testing algorithms in the same P-BIST design would require high area cost. Thus, the selection of test algorithm families should be carefully considered. March-based algorithm is an important class to detect a large variety of faults. A March test consists of a finite sequence of March elements. Each element performs a

series of "reading" and "writing" operations to all memory cells. The addressing order of each element is executed in ascending ( ), descending ( ), or either ( ) way. The first element performs writing 0 to each memory cell in either addressing order. The second element performs two operations to each address in ascending order. Finally, the third element reads all the address in descending order. The complexity order is noted to 4N, where N is the number of memory addresses.

Test sequence (4N): {⇕ (w0); ⇑ (r0, w1), ⇓ r1)}

Beside, based on the March-based test sequence, the dynamic faults can be tested by repeatedly performing the same operation in the same memory cell. To efficiently implement March-based algorithm, we define the corresponding instruction for P-MBIST architecture. The advantage is that one can program the instruction to modify the testing algorithm during run time.

3. PMBIST (Programmable Memory BIST)

The word programmable indicates that we are defining user defined patterns to the memory BIST and is flexible to all types of processors.

It consists of three modules

1. Memory Module
2. Instruction Register
3. BIST Controller

4. Memory Module

Memories are one of the most universal cores. In Alpha 21264, cache RAMs represent 2/3transistors and 1/3 area. In StrongArmSA110, the embedded RAMs occupy 90% area In average SOC, memory cores will represent more than 90% of the chip area by 2010. Here we mentioned different reasons why memory testing is important.

1) Memory testing is a more and more important issue

- RAMs are key components for electronic systems
- Memories represent about 30% of the semiconductor market
- Embedded memories are dominating the chip yield.

2) Memory testing is more and more difficult

- Growing density, capacity, and speed
- Emerging new architectures and technologies
- Embedded memories: access, diagnostics & repair, heterogeneity, custom design, power & noise, scheduling, compression, etc.

3) Cost drives the need for more efficient test methodologies

- IFA, fault modeling and simulation, test algorithm development and evaluation, diagnostics, DFT, BIST, BIRA, BISR, etc.

4) Test automation is required

- Failure analysis, fault simulation, ATG, and diagnostics
- BIST/BIRA/BISR generation

5) Embedded memory testing is increasingly difficult

- High bandwidth (speed and I/O data width)
- Heterogeneity and plurality
- Isolation (accessibility)
- AC test, diagnostics, and repair

BIST is considered the best solution. Because, for Embedded memories accessibility of Pins is not sufficient for testing out side the chip.SOC consists of many memory models. Like, SRAM, FLASH, and ROM etc.

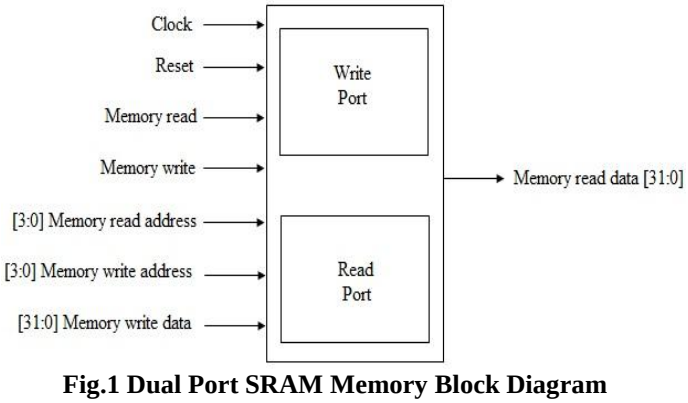

Fig.1 Dual Port SRAM Memory Block Diagram

Description

As our processor/controller is customized, it can support memory types include single/dual/two-port SRAM, one/two-port register file and ROM.

In the project, dual-port SRAMs is used, with same read and write cycles. One port is designed for writing the data and the other port is designed for reading the data.

Implementation in Detail

- Fig.1 shows the block diagram of dual-port SRAM.
- When memory write is high, it enable the processor to write the data to the memory through memory write data, and to a particular address through memory write address.
- When, memory read is high, it enable the processor to read the data from the memory through memory read data, and from a particular address through memory read address.

5. Instruction Register

Description

Instruction register stores the instruction currently being executed or decoded. In simple processors each instruction to be executed is loaded into the instruction register which holds it while it is decoded, prepared and ultimately executed, which can take several steps.

Instructions are serially shifted into instruction register. The output of Instruction Register is available to control circuits which generate the timing signals that controls the various processing elements involved in executing the instruction.

It acts as an interface between processor and BIST Controller.

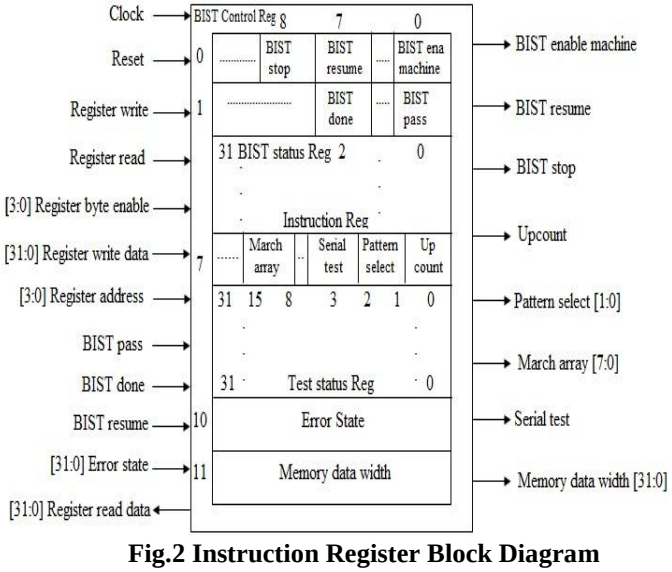

Fig.2 Instruction Register Block Diagram

Our BIST Instruction Register acts as a decoder wherein it stores the instructions given by the processor and BIST Controller obeys these instructions for further testing process.

As shown in Fig.2, it contains few inputs, given as

- **Register write & Register read:-** it enables the instructions to write into the instruction register using register write data and read from the instruction register using register read data, through processor.
- **Register byte enable:**- it loads the instructions into the instruction register as per the processor requirement. For example, if the processor is of 8 bit and 32 bits of instructions are to be send to IR, then the 32 bits are partitioned to four 8 bits of instructions and loaded into the IR. Or if the processor is of 32 bit, then all bits can be send at a time.

In our project, we enable the register byte enable as
0001 – 1 – 0 to 8 bits
0010 – 2 – 8 to 16 bits
0100 – 4 – 16 to 24 bits
1000 – 8 – 24 to 32 bits
1111 – F – all bits (32 bits)

- **Register write data:**- instructions are send to IR through this port, so that the BIST Controller is enabled and the testing is performed as per our requirement.
- **Register address:**- instructions are written at any required address in IR.
- **BIST pass:**- it is high when the memory completes the testing process, that indicates that the controller has passed the testing.
- **BIST done:**- it is high when the memory completes the testing process, that indicates that the controller has completed the testing.
- **Clear resume:**- whenever an error occurs it pauses the BIST Controller for a while, so that the error data sent to the processor. Clear resume enable the BIST Controller to start the process again from the pause state.
- **Register read data:**- shows the error data to the processor.
- **Error State:**- which displays the error to the Register read data, when an error occurs.

Implementation in Detail

- In IR, by enabling the register read, register write and making the register byte enable to 1111 (sending all instructions at a time) as we are using 32 bit processor, we are sending instructions through register write data at some address using register address.
- Instructions should be in such a way that, it enables the controller and sets the testing as per our requirement.
- Instructions are internally applied to the register blocks, as shown in Fig.3, so that it can be retained by the controller.

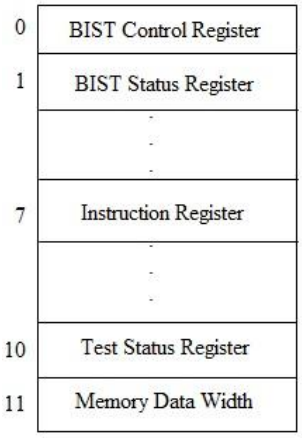

Fig.3 Internal Register Block Diagram in IR

➢ By these register blocks, the outputs of the instruction register are instructed.

• **BIST Control Register:-** Controls the BIST Controller internal operations like, BIST enable machine, BIST resume, BIST stop, etc, as shown in Fig.4.

Fig.4 BIST Control Register Block Diagram

• **BIST Status Register:-** It indicates the status of the Instruction Register, operations like, BIST pass and BIST done, as shown in Fig.5.

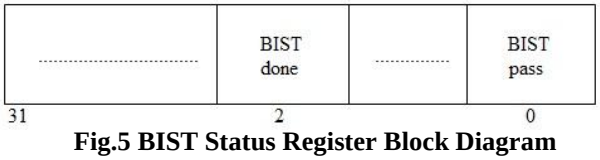

Fig.5 BIST Status Register Block Diagram

• **Instruction Register:-** It enable the instruction like march array, serial test , pattern select and up count, in the BIST Controller, as shown in Fig.6.

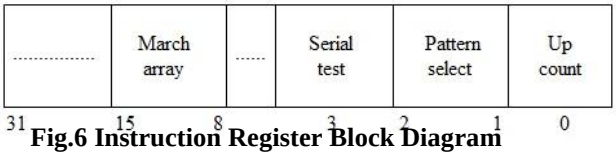

Fig.6 Instruction Register Block Diagram

• **Test Status Register:-** All 32 bits are indicated for the error state to display the fault data.

• **Memory Data Width:-** It sends the size of the instruction.

➢ In the Instruction register, first the instructions are written into the register and, then read by the BIST Controller for testing.

➢ BIST pass and BIST done are high when the testing is completed and clear resume is high when a fault is detected and the testing is started again.

6. BIST Controller

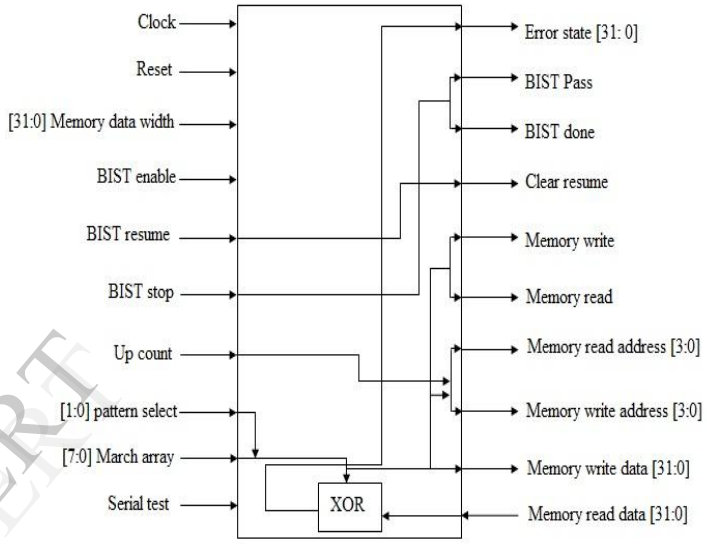

Fig.7 BIST Controller Block Diagram

Description

Typically the **BIST controllers** that are going to be generated will have test algorithms built into them. (E.g. Marching 0's, 1's, Checker - Board ...). To summarize BIST Logic is going to test your RAM.
BIST Controller is internally built by Finite State Machine (FSM) which consists of March algorithm, counter, etc.
It takes instructions from the IR, test the memory and detect the faults.
As shown in Fig.7, inputs of BIST Controller are

• **Up Count:-** Its enables that the memory should be tested upwards or downwards as per our requirement.

• **Pattern Select:-** It selects that which pattern should be selected to test the memory through the March algorithm.

Examples,

00 – Write zeros and Read zeros

01 – Write zeros and Read ones

10 – Write ones and Read zeros

11 – Write ones and Read ones

• **March Array:-** It is selects that which format need to be followed by the controller to test the memory which are instructed by the IR.

Examples,

0000 – State m0, only read

0001 – State m1, write then read

0010 – State m2, read then write

0011 – State m2, read, write and read

• **Serial Test:**- If multiple memories are to be tested, it specifies that, memories are to be serially tested or at a time..

• **BIST Enable Machine:**- Enable the BIST Controller for testing.

• **BIST Resume:**- Makes the BIST Controller start, while it is stopped when an error occurred.

• **BIST Stop:**- Stops the BIST Controller when the testing is completed.

The outputs of BIST Controller are connected to the Memory to run the test algorithm.

Implementation in Detail

➢ Instructions from IR are sourced to BIST controller, wherein, if BIST enable is high, the instruction enable up count, selects the patterns select as 11 (i.e., write one and read 1), march array as 0011 (i.e., read, write and read) and, BIST resume and BIST stop is high, then it indicates that, data is loaded from 0000 to 1111, as it's an up count. In memory, all the initial data is read (i.e., 0's) from the memory, then 1's are written into the memory, which is known to be the valid data, and finally the written values are read from the memory which is known to be the expected data. Then, an Ex-or operation takes place, which ex-or the valid and expected data, if the result is all 0's, then the memory has no faults or if the result has any 1's, then the memory has a fault which is sent by the Error state.

➢ If the testing process is completed, BIST pass, BIST done is high.

7. Programmable Memory BIST (Architecture)

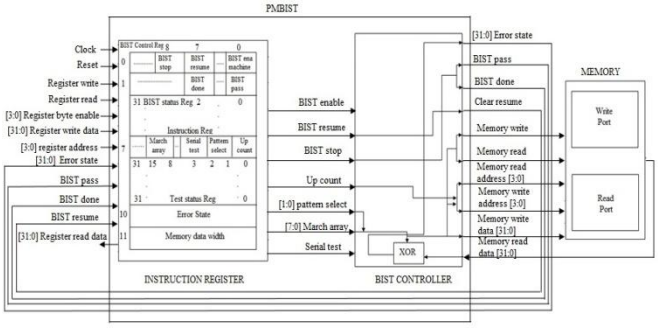

Fig.8 Programmable Memory BIST Architecture Connected To Memory

Description

Finally, it is the top module, combination of both **Instruction Register** and **BIST Controller,** as shown in Fig.8. When instructions are given to the Instruction Register by the processor, it stores the instructions currently being decoded and forwards it to the BIST Controller, where, the controller is connected to the memory. As per the instructions, controller tests the memory, and checks for the faults. If any fault is detected, then the data and address are sent to the processor by register read data. And if there is no fault and testing is completed, then BIST pass and BIST done become high.

8. Simulations

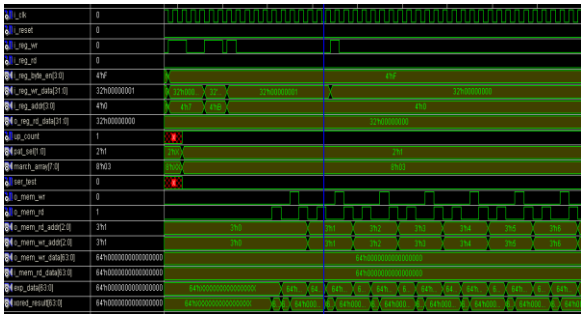

Fig.5.4. Simulations of PMBIST

➢ In here, we can see that, instructions are sent to the PMBIST using register write data, at some address using register address, by which we are selecting up count, pattern select as 01, march array as 0011, wherein, we are operating the march operation as read, write, read, which can be viewed at memory write and memory read. These pins are

enabled when ever the march test is conducted in a particular pattern as mentioned in IR.

- Memory read address and memory writer address are incremented to write the binary values and test the memory.

- Finally, an ex-or operation is done between the memory read data (which is read from memory) and expected data (data which undergoes March operations), so that we get the xored result as, 0 ⊕ F = F and 0 ⊕ 0 = 0 (32 bit or 64 bit data), indication for no fault memory.

- If any fault is detected, that can be viewed by register read data in IR via, error state in BIST Controller.

9. Conclusion

An Area Efficient Programmable Memory Built-in-Self-Test, proposed in our paper gives users the flexibility to select test algorithm patterns (User defined configuration file). The test complexity can be easily adjusted as a result. Compared with earlier Memory BIST designs, our method achieves more the level of flexibility. The proposed method will be very useful in SOC testing, since many different memory core modules (e.g., DRAM, SRAM and ROM) may be employed in SOC and they required different same test algorithm with different patterns. Moreover, the controller can be extended to different memory types in the same read/write cycle condition, without increasing any state. The proposed design greatly simplifies the reconfiguring process when a new testing pattern is selected, and thus reduces the overall test time. Thus, the hardware cost can be greatly reduced. The advantages of the proposed architecture include reduced test time, re-programmability, easy and simple to control test procedure.

9. References

1. Chung-Fu Lin and Yeong-Jar Chang, "An area efficient design for programmable Memory built in self test," IEEE transaction, 2008.

2. V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self-test. I. Principles," *IEEE Design & Test of Computers, Vol. 10, No. 1*, pp.73-82, Mar. 1993.

3. (No)D. Appello, V. Tancorre, P. Bernardi, M. Grosso, M. Rebaudengo and M.S. Reorda, "Embedded Memory Diagnosis: An Industrial Workflow," in *Proc. ITC*, pp:1 - 9, Oct. 2006.

4. S. Boutobza, M. Nicolaidis, K.M. Lamara and A. Costa, "Programmable memory BIST," in Proc, ITC, pp:45.2, Nov.205.

5. D. Xiaogang, N Mukherjee, C. Wu-Tung: S.M. Reddy, "Full speed field-Programmable memory BIST architecture," in Proc. ITC,pp:45.3,Nov.2005.

6. A.J. Van de Goor and J. de Neef "Industrial Evaluation of DRAM Tests," in *Proc. Design, Automation Test in Europe*, pp. 623-631, March 1999.

7. C.F. Wu, C.T. Huang and C.W. Wu, "RAMSES: a fast memory fault simulator," in *Proc. Defect and Fault Tolerance in VLSI Systems Symposium*, pp. 165 - 173, Nov. 1999.

8. www.searchmobilecomputing.techtarget.com/definition/RAM

9. Michael D. Ciletti "Advanced Digital Design with Verilog HDL," Prentice-Hall, Inc.(PHI) publishers.

10. http://en.wikipedia.org/wiki/Xilinx_ISE