International Journal of VLSI System Design and Communication Systems

ISSN 2322-0929
Vol.01,Issue.01,
August-2013,
Pages:01-08

www.semargroups.org

Advanced LFSR with BIST for Testing EMIFs for Synchronous and Asynchronous Memories

GOTTIATI RAMESH

[1]VLSI Dept, Hyderabad, AP-India, Email: rameshchowdary7@gmail.com,

Abstract: This paper involves a new random testing circuit for SOC System Design based on LFSR to test the integrated EMIF IP core with restricted random verification methods. External Memory Interface (EMIF) is a slave interface device build on Advanced High-performance Bus (AHB) mainly used for embedded SOC applications. EMIF pass requests from various masters like CPU, DSP or EDMA to control SRAM. The functional verification is used to look into that whether the design actualize a specification or not. Primarily, verification incorporates black-box, white-box, and grey-box methods. Black-box is a kind of functionality existence check irrespective of internal structure, but it's difficult to locate errors. Thus it lacks maintainability of verifications; white-box can configure test input rapidly through internal structure, however, it has poor portability and needs to be designed in much detail; the third one, grey-box method is work out on not only output validity to input but internal performance and configuration. The test result decides this circuit achieves random testing of the EMIF IP core.

Keywords: EMIF, LFSR, AHB Protocol, SOC System.

1. INTRODUCTION

In addition to development of SOC design, verification and logic inspection are becoming a mega factor. Especially, verification is being regarded as the bottle neck of the whole design cycle [1]. The problems in design of Memory system are even more critical for its slow down progress in performance contrast with processors. EMIF (External Memory Interface) is a slave interface device mainly used for embedded SOC applications. EMIF pass requests from various masters like CPU, DSP or EDMA to control SRAM, SDRAM, NORFLASH, FIFO, and other read and write operations.

The functional verification is used to look into that whether the design actualize a specification or not. Primarily, verification incorporates black-box, white-box, and grey-box methods [2]. Black-box is a kind of functionality existence check irrespective of internal structure, but it's difficult to locate errors. Thus it lacks maintainability of verifications; white-box can configure test input rapidly through internal structure, however, it has poor portability and needs to be designed in much detail; the third one, grey-box method is work out on not only output validity to input but internal performance and configuration. The grey-box verification quality depends on test stimulus which is supported by building qualify testing circuit. A direct testing circuit can be used for some simple IP cores.

Test points are added manually, then the response is inspected during simulation. Instead of direct way, by improving the verification space, random verification is becoming as a flexible to designers [3]. Additionally, it is not necessary to store every test points for the random method, which will save time, correcting the space and behavior issues. To avoid consequences, testing circuit should be operated towards a desired direction which originates from input bus protocol and functional point coverage. This may leads to tester's, commit a faults during compiling test bench. Various conditions can be generated to get better coverage, quality and efficiency [4]. This paper focused on random testing circuit design to achieve functional verification of EMIF IP core.

II. STRUCTURE OF THE RANDOM TESTING CIRCUIT

The intended random testing circuit is build on LFSR, which generates pseudo-random numbers follow the AHB protocol include the testing signals like: HWRITE, HSIZE, HBURST, HTRANS, HADDR, and HWDATA. This circuit will also results statistic testing. LFSR is a linear feedback shift register, contains a shift register and the feedback circuit. Shift register embody a series of trigger. For each clock cycle, signal transfers from one trigger to the next trigger, throughout this output is generated. The feedback circuit performs the XOR operation on several trigger

outputs, and then results to the first trigger, which is a common circuit. LFSR works as the random generator to produce pseudo-random binary bits sequence in the testing circuit. LFSR structure is shown in Figure 1.

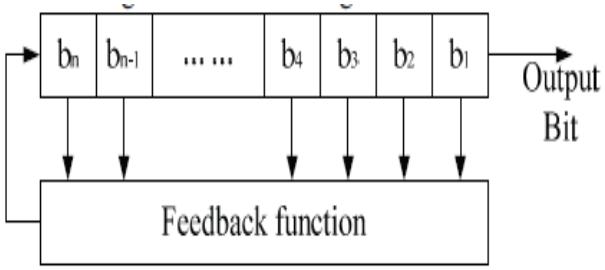

Fig 1. LFSR structure diagram

AHB bus signals contains HCLK(clock signal), HRESET(the global reset signal), HSEL(selecting signal from the device), HWRITE(reading and writing signal), HSIZE(size of the data transfered), HBURST (transmission type signals), HTRANS(transfer mode signal), HADDR(address signal), HWDATA(write data signals), HRDATA(read data signal), HRESP (transmission status feedback signal), and HREADY(ready signal). HCLK, HRESET, HSEL are rendered by the system, HRDATA, HRESP, and HREADY are rendered by the external memory interface. The association between random testing circuit and the EMIF is shown in Figure 2. the EMIF AHB signals generated by the circuit and its width are shown in Table 1.

Table 1: Description of AHB signals

Signals	Width
HWRITE	1
HSIZE	3
HBURST	3
HTRANS	2
HADDR	32
HWDATA	32

The structural representation of the random testing circuit is shown in Figure 3. Here the testing vector generating module is used to produce testing vectors from the pseudo-random numbers produced by the LFSR that meeting AHB protocol.

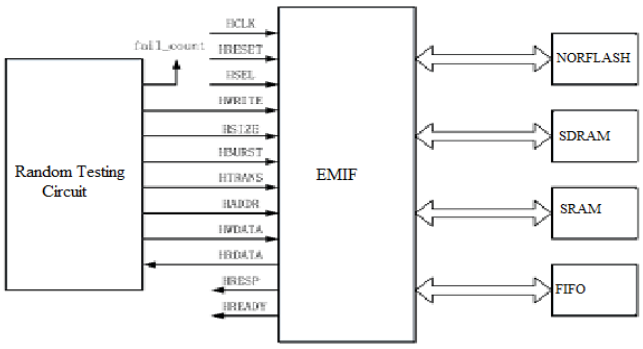

Fig 2. Connection between random testing circuit and

The data is tested, before read it from the external memories and written it to the address in testing module. If both contain same data, the testing module will raise a success flag signal, otherwise it will raise a fail flag signal. The testing process will also record the number of the failures during the test.

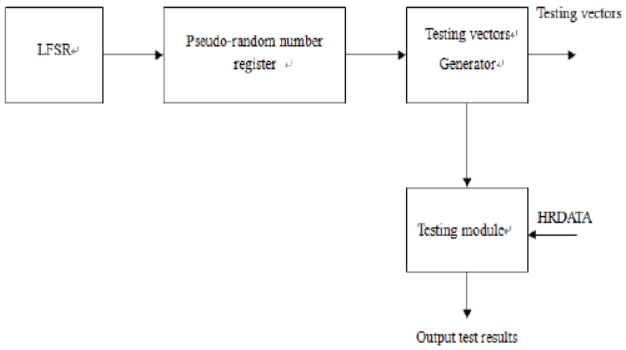

Fig 3. Structure of the random testing circuit

BIST Architecture: It is very important to choose the proper linear feedback shift register architecture for achieving the appropriate fault coverage. Every architecture consumes different power even for same polynomial. A typical BIST architecture contains a test pattern generator; usually implemented as a linear feedback shift register, a test response analyzer; implemented as a multiple input shift register and a BIST control unit. All executed on the chip (Figure1). Another problem associated with choosing linear feedback shift register is design issue, which includes linear feedback shift register partitioning. In this the linear feedback shift register are differentiated on the basis of hardware cost and testing time cost. This approach allows applying at-speed tests and eliminates the need for an external tester. The BIST architecture is show in below.

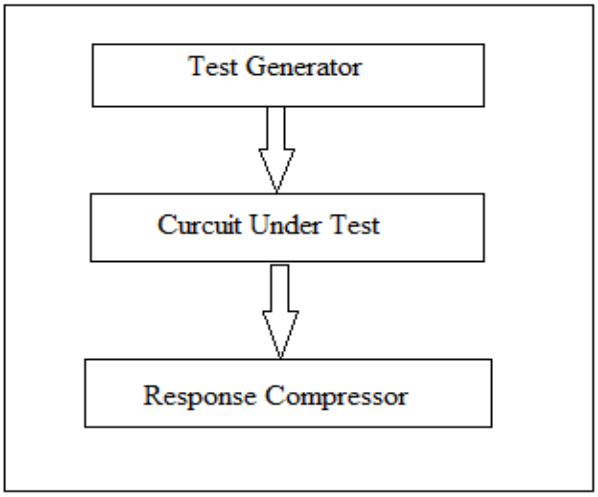

Fig 4. BIST Basic Idea

III. DETAILS OF THE RANDOM TESTING CIRCUIT

The detailed view of the random testing circuit is shown in Figure 5. As discussed before on LFSR, the write generator produces a bit of pseudo-random number from 0 to 1 by attaining the remainder of the pseudo-random number from LFSR that divided by 2, to assess whether this operation is to write data into the EMIF or read the data from the EMIF. The size generator produces 3 bits of pseudo-random number from 0 to 2 by attaining the remainder of the pseudo-random number from LFSR that divided by 3, to fix the transfer size. The burst generator produces 3 bits of pseudo-random number from 0 to 7 by attaining the remainder of the pseudo-random number from LFSR divided by 8, to fix on transfer type. The address and the data are generated according to the pseudo-random number from LFSR and the specification. Address decoder translate or decodes the address, to judge the access is to NORFLASH and check whether it is necessary or not to start up a command generator.

Once the enable signal received, the command generator initiate to work, then assess the type of the operation according to the burst. Finally write signal used to determine which command gives to output. The writing data recorder is used to record data that written to EMIF, and to output the writing data when compare with the data read from EMIF is going to be executed in the test module. The verification logic is used to justify whether the read and write operation are executed perfectly or not by comparing HRDATA (data read from EMIF) and re_data (the writing data recorded in the writing data recorder). If HRDATA and re_data data is same, then the result is correct, the output result is represent as 0; if not equal, then this result is incorrect , the output result is represent as 1. The testing counter will accumulate all result values. The final result will be the total number of failures tests. At the end of the test, if the total number of failures is 0, all the tests pass; otherwise, problems still exist in the circuit so that further inspection is needed.

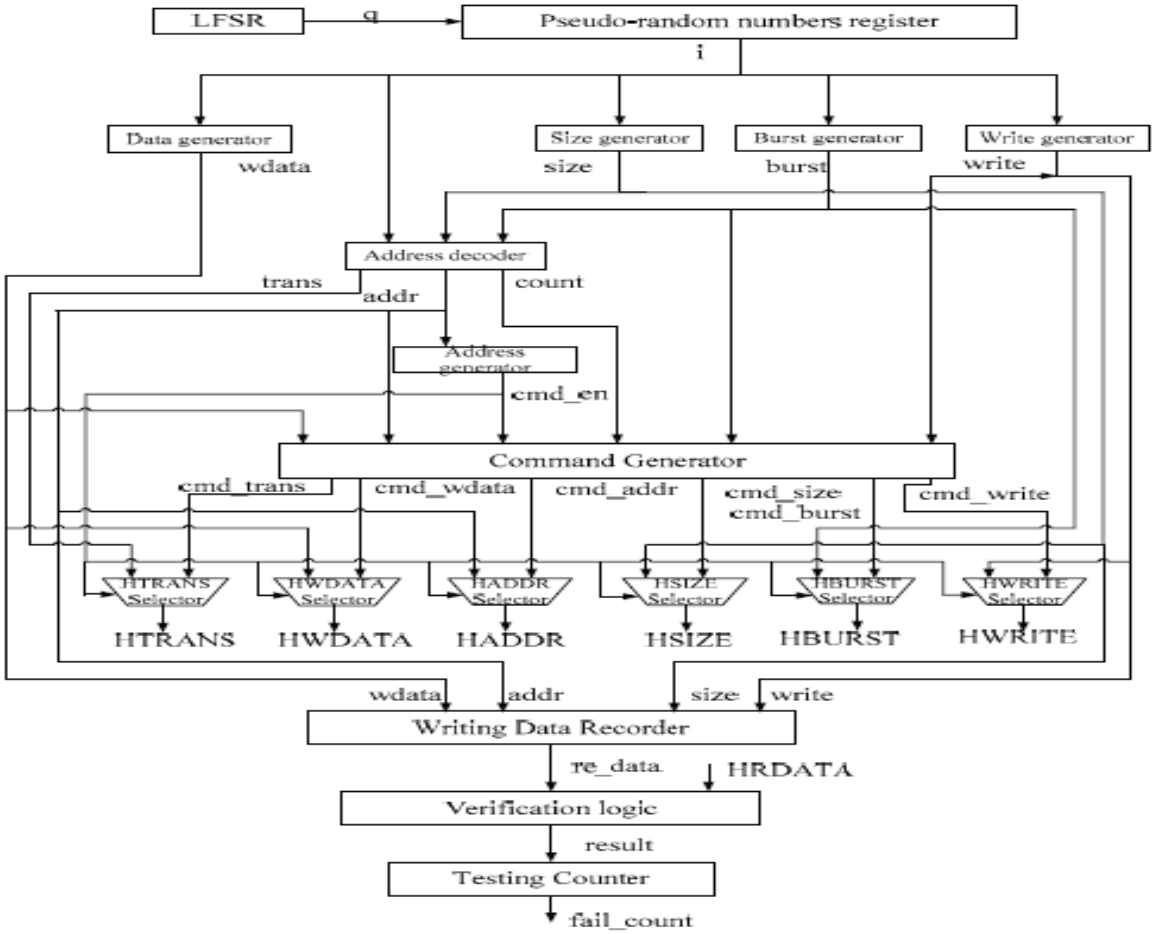

Fig 5. The detailed structure of the random testing circuit.

IV. SYSTEM ON CHIP

A. The System Onchip Era

The concept of a system on chip (SoC) device has evolved over the past decade, influenced by significant advances in semiconductor design and manufacturing technologies. Most notably, the consistent scaling of Moore's Law continues to extend the capabilities and the levels of function integration (i.e., for integrated circuits or ICs). The business and economic dynamics of the electronics industry are equally, if not more significant an influence on how complex chips are designed and manufactured. More digitally-enabled products are proliferating through all levels of increasingly interconnected economies and geographies. Consumer markets are becoming the main driver for this proliferation and the consumer market is fast-paced, with a thirsty for new products on a regular basis. The electronics industry that services this proliferation must become more nimble and efficient to keep up. At its essence an SOC represents the natural progression of the industry to design and build things that perform more functions in smaller, more efficient ways. The functionality that a decade ago may have been contained on a printed circuit board (PCB) and executed by multiple discrete chips (e.g., memory, processors, interface, graphics, etc.), can now be integrated on a single IC, commonly called an SoC.

This integration is achieved largely through the emergence of compact and robust techniques to represent the important functions or applications contained in an SoC. These techniques leverage high-level design languages and synthesis technology to deliver complete, yet flexible descriptions of SoC components in synthesizable or "soft" form. This new level of design abstraction to represent critical building blocks, known as semiconductor intellectual property (IP), provides access to micro components designed to optimally perform specific functions. These IP blocks are then integrated or assembled to form a working SoC.

A substantial portion of a SoC may contain parts of previous generation chips that are reused to save time and ensure reliability. Often, critical functions not within the core competence of the system developer can be obtained elsewhere in the SoC ecosystem. Indeed, the SoC era has spawned a cottage industry within the semiconductor business of companies whose sole business is to provide pre-built IP blocks to the developers of the complete SoC. Suppliers such as ARM (microprocessors), MIPS (video processing), Rambus (memory), Imagination Technologies (graphics) and CEVA (digital signal processing) have been successful in mastering the implementation of specific functions and licensing that knowledge through their IP that is delivered in "soft" or synthesizable form. Modern SoCs are true systems that save product developer's time, money and product real estate thanks to their efficiency. As the concept of SoC becomes more widely adopted throughout the electronics industry, both the opportunities this approach offers, as well as the challenges it presents, are becoming clearer. SoCs are having a similar effect as ASICs, except the scale is larger. The promise of an efficient "mix-and-match" approach to product development, enabled at a new level of design abstraction, holds the potential to dramatically reduce the time, costs and resources required to introduce new products at consumer-market pace. But the economic and technical challenges of SoCs are daunting and increasing. The complexity of modern devices requires new approaches to design and a re-thinking of supply chain models. Manufacturing costs for leading-edge semiconductor processes add to the challenge. And the pitfalls of design reuse -sourcing and integrating building blocks from a variety of places - introduce more risks still. More than ever, the most cost-, time- and risk-efficient methodologies are needed.

B. The Soc Development Process

SoCs hold the key to continued scaling of innovation in electronics and are a linchpin to growth in all sectors. But significant challenges exist on both the technical and business levels of SoC development. The inability to meet some of these fundamental challenges threatens to limit the vast potential for growth of the entire industry. First and foremost, SoCs, which require the most advanced manufacturing process and design methodologies, are expensive to develop. Market researchers estimate that the total cost of a design at 22nm is more than double that of a 45nm-based design. A significant portion of that cost is related to time-consuming iterations to ensure the design functions properly and at the right specifications (timing, area and power being the most critical constraints). At 32nm it is estimated that a typical design is likely to meet just 50% of its objectives; at 28nm that number drops to 30%. This is a very high risk game!

As mentioned, historically the electronics industry has addressed the need to manage greater complexity by moving design representation up in abstraction. This continues to be true, but perhaps not through new languages or formats, as has previously been the practice. The great hope of next-generation SoC development is that design reuse at an abstraction level beyond logic gates or RTL – whether through pre-existing internally developed blocks, or the use of the vast 3rd party market of IP available – will save time and reduce risk in putting together multi-function chips. The areas for consideration will be modeling, analysis, simulation and optimization. In concept that is true, but as with most things the devil is in the details and IP reuse has no shortage of pitfalls that can trip engineering teams up. To understand where the greatest opportunities for improvement exist, it's important to view the big picture of SoC development. Today's SoC development process progresses essentially through three main steps.

• **System Realization:** The development of a complete hardware/software platform that will provide all necessary

support for end-user applications. The platform includes one or more SOCs and adds an embedded software infrastructure that typically includes an OS and middleware reference applications. This step is increasingly driven by the software applications that will run on the completed system.

• SOC Realization: The new and critical "bridge" step to take a system concept to a real design that can be implemented at a detailed level. It involves the all important steps of choosing the right architecture and IP, validation of the IP, assembling it all on a workable platform, and performing a variety of analysis and verification tasks to ensure that it can be implemented as a viable, working SoC device.

• Silicon Realization: This represents the set of steps necessary to take a verified SoC design consisting of hardware and software functions, and implement it in working silicon. This is the domain of traditional EDA providers, who work closely with semiconductor manufacturers such as Taiwan Semiconductor Manufacturing Corporation (TSMC). We often call this "EDA Classic".

There are three critical areas driving today's IP-based SoC Realization solution:

• Soc Creation: Is the architecture chosen for the SoC the "right" one? Will it allow the entire system to be synchronized, hardware and software? Will it be scalable across at least two generations of systems products?

• SOC Handoff: Once the IP and platform has been chosen, is the resulting SoC ready for implementation based on the desired specifications? Can the impact of decisions associated with cost, time and market be measured?

There is also an important modeling issue that spans all three areas.

• Design Coherence: As one moves up and down the design hierarchy and associated levels of abstraction, is there confidence that the accuracy and completeness of the system model is maintained? Is system intent maintained in the translation to more silicon specific implementations?

An SoC Realization environment addresses each of these issues with a set of tools, methods and integrated approaches that simplifies the process from a higher level design to a ready to implement design. The result transformed through a SoC Realization environment, that the design is targeted optimally for silicon implementation and that the developer can be assured that it will work as intended once passed to the next phases of the process – design implementation and manufacturing

C. Stuck-At Fault

A stuck-at fault is a specific model used by fault simulators and automatic test pattern generation (ATPG) tools to mimic a manufacturing defect within an integrated circuit. Assume individual signals and pins are *stuck* at Logical '1', '0' and 'X'. For example, an output is tied to a logical 1 state during test generation to assure that a manufacturing defect with that type of behavior can be found with a specific test pattern. In the same way the output could be tied to a logical 0 to model the behavior of a defective circuit that cannot switch its output pin. All the faults could be analyzed by using the stuck-at fault model. The stuck-at model also used to gain for static hazards, namely branching signals, can render a non testable circuit. Single stuck line is a fault model used in digital circuits. It is feasible for post manufacturing testing, not for design testing. The line or node in the digital circuit is a stuck at logic high or logic low. There struck t is called as fault. Digital circuits can be divided into:

1. Gate level or combinational circuits which contain no storage (latches and/or flip flops) but only gates like NAND, OR, XOR, etc.
2. Sequential circuits which contain storage.

This fault method applied to gate level circuits, or a group of a sequential circuit which can be separated from the storage elements. Preferably a gate-level circuit would be completely tested by applying all possible inputs and inspecting that they produce the right outputs, but this is totally impractical: an adder to add two 32-bit numbers would require $2^{64} = 1.8*10^{19}$ tests, a test that detects any single fault, can find multiple faults.

To use this fault model, each input pin on each gate is assumed to be grounded, and a *test vector* is used to notice the faulty circuit. The test vector is a array of bits to apply to the circuit's inputs, and a array of bits expected at the circuit's output. Once attaining the test vectors for grounded pins, then each pin is connected in turn to a logic one and another. Combination of test vectors is used to find faults falling under these conditions. If the considered gate pin is grounded, and this test vector is applied to the circuit, the corresponding output bit in the test vector will not agree by at least one of the output bits.

This fault model also fails to detect common faults among adjacent signal lines, placing in pins that drive bus connections and array structures. However, the concept of single stuck-at faults is widely used with some more tests which allowed industry to ship an acceptable low number of failure circuits and poor quality circuits.

The testing based on fault model is aided by several things:

1. A test implemented for a single stuck-at fault often finds a number of other stuck-at faults.

2. A series of tests for stuck-at faults will happened often, purely its by serendipity, find a large number of other faults, such as stuck-open faults. This also can be called as "windfall" fault coverage.

In addition *IDDQ testing*, measures the way the power supply of a CMOS integrated circuit changes when a small number of slowly changing test vectors are supplied. Since CMOS uses a very low current when its inputs are static, or else any increase in that current indicates a potential problem.

V. VERIFICATION

At the time of verification of the external memory interface, the random testing circuit supplies the testing vectors to inaugurate reading and writing tasks of whose signals to meet AHB protocol. During the test, an output contains complete records of each access time, controlling signals, address, data and test results will be taken in order to locate errors. After random testing, the circuit will output the number of test failures. If there is no circuit failure, all tests pass; if there any failures during the test, those problems could be in the EMIF or the memories, so that again we should inspect the test to find out the problems and find a way to solve them. An EMIF designed by our team has been tested by this random testing circuit without failures and allegations.

VI. SIMULATION RESULTS

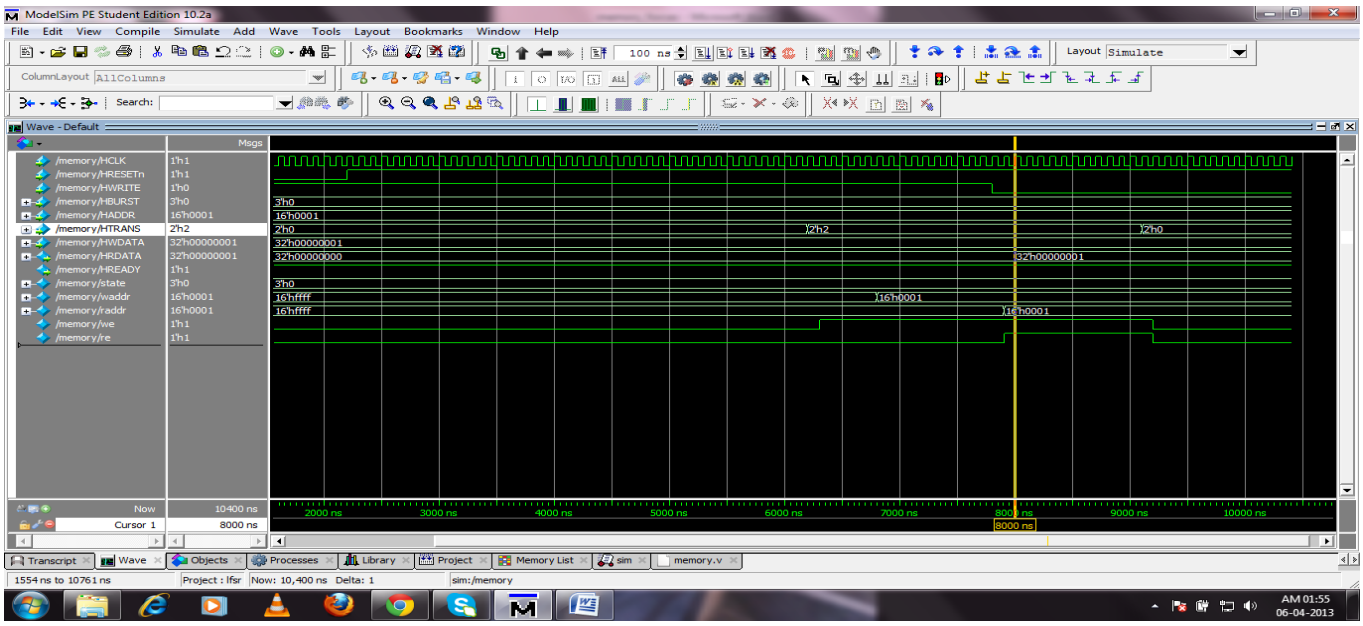

Fig 6. Simulation Result Of MEMORY

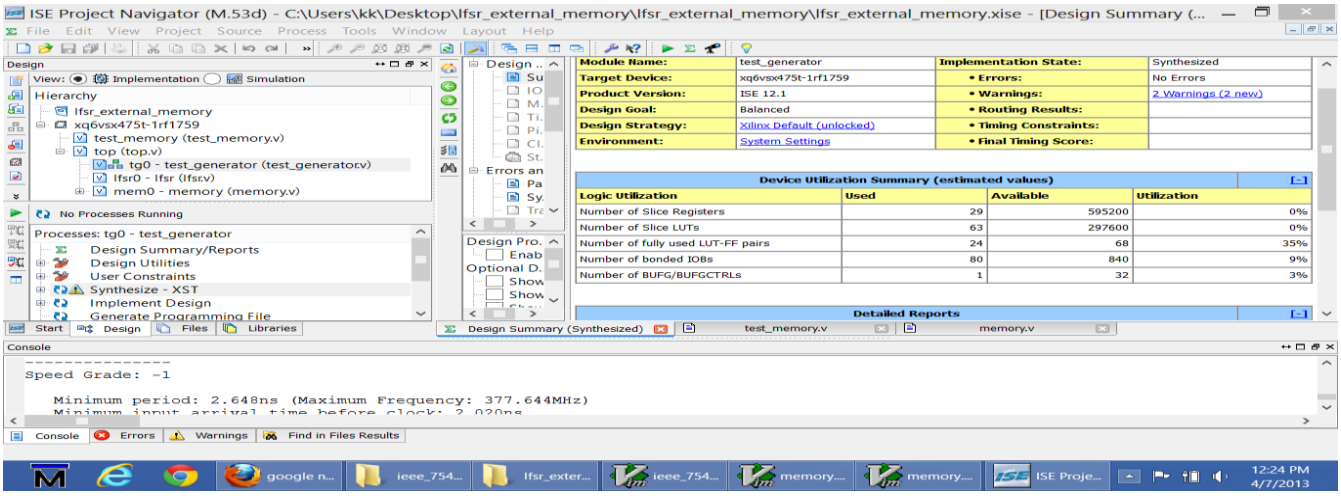

Fig 7. Synthesis Result of TEST GENERATOR.

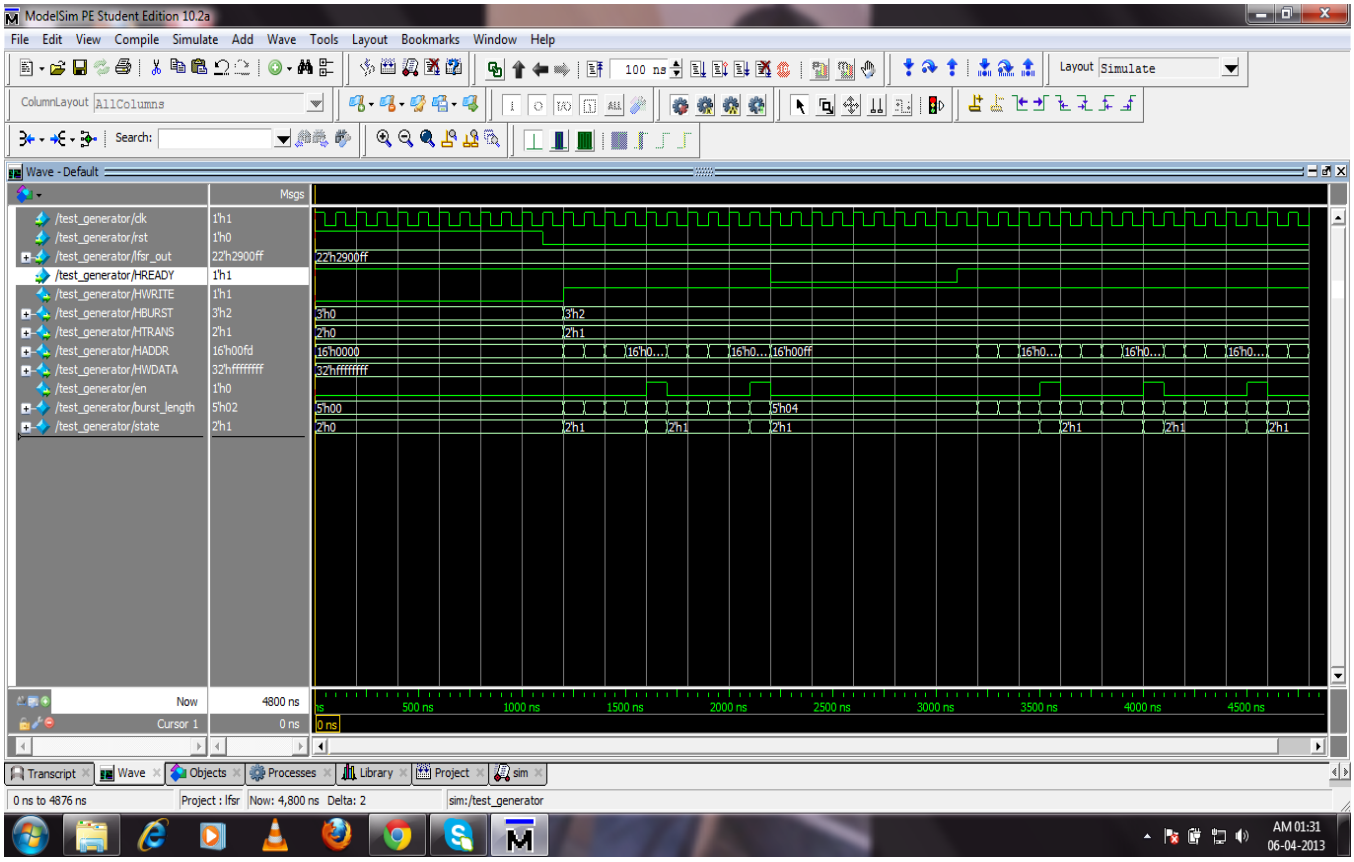

Fig 8. Simulation Results of Test Generator.

SYNTHESIS RESULTS OF TOP MODULE

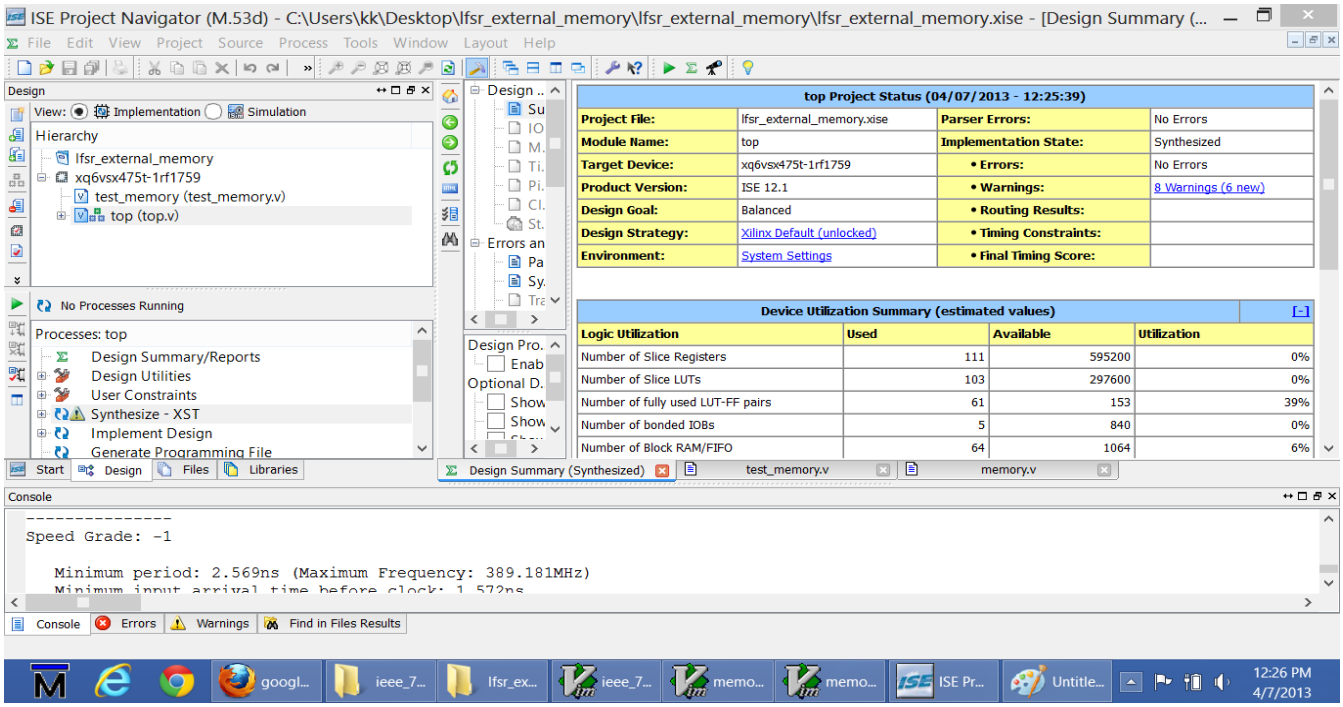

Fig 9. Synthesis Results of Top module.

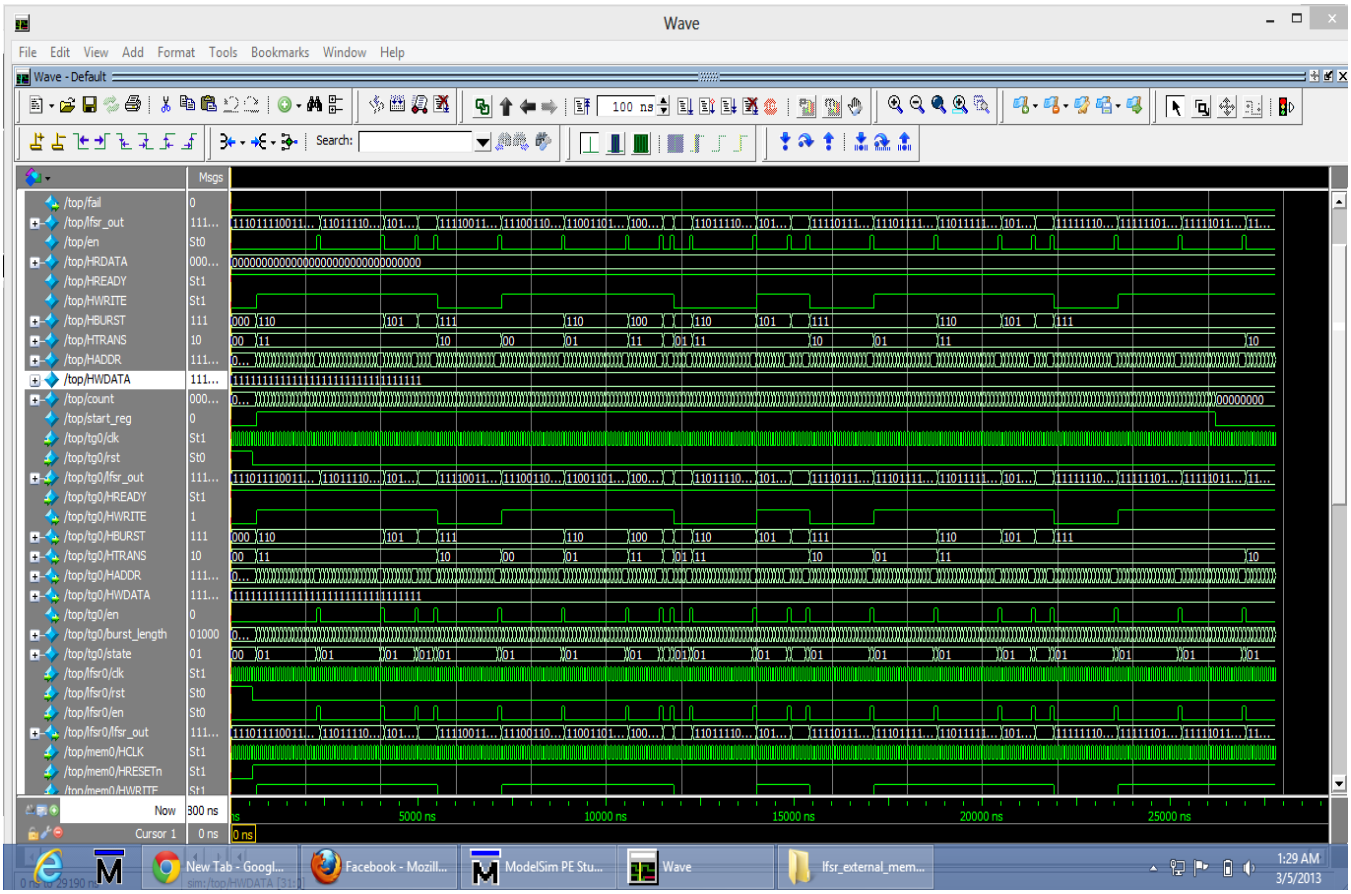

Fig 10. Simulation results of Top module.

VII. CONCLUSION

In this paper the design of a random testing circuit based on LFSR for the external memory interface is discussed. Here the structure of circuit, principle of operation, and the verification process are also elaborated. The random test circuit strengthens testing efficiency, and reduces the artificial dependency in testing process. Finally, this random testing circuit is of much versatility and adaptability because of the wide use of the AHB.

VIII. REFERENCES

[1] Y. Wu, L. Yu, K. Xue, W. Zhuangr, "Functional verification of memory controller based on the hierarchical test bench," Microelectronics and computer, pp. 25-28, 1998(2).

[2] Janick Bergeron. Writing testbenches functional verification of HDL Models(2nd Edition) [M] .Springer-Verlag-2003.

[3] Prakash Rashinkar, Peter Paterson, Leena singh. System-on-a-chip Verification Methodology and Techniques [M]Kluwer Academic Publishers-2001.

[4] Qingdong Meng, Zhaolin Li, Fang Wang. Functional Verification of External Memory Interface IP Core Based on Restricted Random Testbench. International Conference on Computer Engineering and Technology. 2010: (7) 253-256.

[5]WANG Yu-hua, GUAN Ai-hong, HOU Zhi-qiang, ZHAN Jing, ZHANG Huan-guo. Evolutionary Random Sequence Generator Based on LFSR, Computer Engineering, Vol.35 No.6.