International Journal of Research
Available at https://edupediapublications.org/journals
P-ISSN: 2348-6848
E-ISSN: 2348-795X
Volume 07 Issue 06 June 2020

Low Power Parallel VLSI Architecture for Mbist

G SAI RAM RAJU (Research Scholar) [1], Mrs. K VANI (Assistant professor) [2]
Gonna institute of information Technology and Sciences

ABSTRACT

This paper will present a flexible Memory Builtin Self-Test (MBIST) designed to be easily adaptable to specific memory configurations and user requirements. Its RTL code is generated by means of programming scripts that provide an easy to read code without the use of complex compiler directives. The basic architecture can be adapted to different schemes of test such as parallel, in which all the memories are tested concurrently, or sequential, in which the memories are tested one at the time. Linear-feedback shift register (LFSR) counters however a counter design based on multiple LFSR stages is the proposed designs that have been shown to be well suited to applications requiring large arrays of counters and can improve the area and performance compared with conventional binary counters, which are used in existed design.

Index Terms— MBIST, comparator, multiplexers, linear-feedback shift register (LFSR) and counter.

I.INTRODUCTION

A concern of a digital IP provider is to be able to rapidly develop and deliver IPs with different sets of features and characteristics ensuring that only the necessary features are included in the final RTL. A secondary concern is to make the IP usable for many years to get the most value out of one's work. These seem to be contradictory objectives. Much has been said about IP reuse but the economics of the semiconductor industry has not been kind to IP providers. Ips are expensive to develop and maintain, which is perhaps why few IP providers are successful, and even in these cases the solutions are in general complete and self-contained like CPU cores, cache controller, memory compilers, etc.

One way to address these concerns would be to architect the design in which modules can be plugged-in like "LEGO blocks". Using this philosophy, a memory built-in self-test (MBIST) architecture can be used in dozens of SoCs ranging from very simple (containing a handful of memories) to very simple (containing a handful of memories) to very complex (containing thousands of memories and hundreds of MBISTs).

A) LINEAR-FEEDBACK SHIFT REGISTER (LFSR)

In computing, a linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the

register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle.

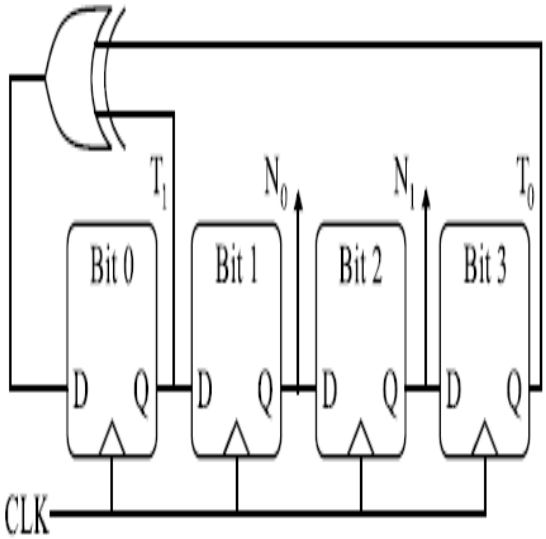

Fig1: Structure of 4bit LFSR.

II. LETERATURE SURVEY

Linear-feedback shift register (LFSR) counters have been shown to be well suited to applications requiring large arrays of counters and can improve the area and performance compared with conventional binary counters. However, significant logic is required to decode the count order into binary, causing system-on-chip designs to be unfeasible. This paper presents a counter design based on multiple LFSR stages that retains the advantages of a single-stage LFSR but only requires decoding logic that scales logarithmically with the number of stages rather than exponentially with the number of bits as required by other methods. H. Mo and M. P. Kennedy, "Masked dithering of MASH digital delta sigma modulators with constant inputs using multiple linear feedback shift registers," Paper shows that applying a linear feedback shift register (LFSR) dither to a digital delta-sigma modulator (DDSM) cannot always increase its fundamental period. For some DDSMs, the LFSR dither may reduce its period in some cases, instead of increasing it, which worsens the output spectrum. Hence, the paper calculates the dithered DDSM's period and analyzes the influence of LFSR dither on the period. Furthermore, for such kind of DDSM, the paper explains how to add the LFSR dither to increase the period for a full input range. Finally, experiment is performed to confirm the analysis.

III. MEMORY BIST ARCHITECTURE

Semiconductor memories are dedicated circuits designed to store digital information, they are the most used IP in modern SoCs. Memories incorporate the greatest concentration of transistors per square area for a given semiconductor technology, pushing the chip fabrication process to its limit. Consequently, memories are more failure prone than logic. Testing such embedded memories can be a challenging task as each type of memory has peculiarities that make it more susceptible to distinct types of faults. The objective of this paper is not to discuss various fault models a memory can present but instead to show how to implement a Built-In Self- Test (BIST) Architecture that could cover them all.

The focus of the discussion in this paper will be static RAM memories (SRAM), which are the most common

memory type used in SoCs. The memory test is in general a well understood process. It consists of identifying all physical processes that can lead a memory to malfunction; these are usually known as physical faults. The list of such faults is then mapped to observable behaviors that do not correspond to normal (expected) memory behaviors; this is called functional-faults or fault-models. Note that many different physical-faults can exhibit themselves as the same malfunction. We could think of the physical-faults as the disease while the malfunctions as the symptoms. Finally, specific sequences of operations are designed to sensitize and detect these symptoms. In most cases (except when fault analysis is intended) the reasons for the symptoms are irrelevant. It is enough to detect the misbehavior to conclude that the memory has problems.

In the case of SRAMs, the sequence of operations designed to detect faults is called a "March sequence" or "march algorithm". Some professionals may consider the March algorithm as being the whole memory test, but for this paper, we prefer to consider a March algorithm to be only the sequences of reads and writes performed in the memory. The SRAM test consists of the application of the designed sequences or March algorithms to the target memory, and comparing the results to the expected values. This is shown in the diagram of figure 2.

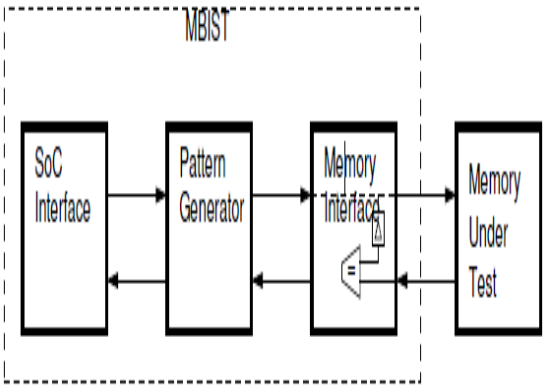

Fig. 2. Basic Architecture.

The figure 1 shows the Pattern Generator that generates the test sequences (march algorithm) and the comparison block that compares the values read from the memory to the expected (golden behavior) values. If any difference is detected a fault has been found. Also shown in figure 1 is a system interface block which interfaces to the SoC. This is necessary to control the memory BIST and to allow tester or software get information about the test. Another point to be noted is that the comparison block is inside a block called "Memory Interface". The Memory Interface block is divided in two main sections: the first translates the patterns generated by the pattern generation block into memory signals; the other is a comparison section which compares the values read from the memory. This basic structure can be applied to all memories in the SoC but this can be very costly for a system that includes a large number of memories.

IV. PARALLEL ARCHITECTURE

Many optimizations can be applied to this basic architecture. The most obvious would be sharing the system interface block among several BISTs. Devising a very

simple protocol it is possible to control several basic BISTs and collect important feedback from the tests using a common SoC interface. On the other hand if we could improve the intelligence of the "Memory Interface" block it would be possible to share the core BIST components including the Pattern Generator among several memories.

To test several identical memories, the patterns from Pattern Generator block would be translated to each memory individually and the read back values compared individually on each respective Memory Interface block. That alone would improve the BIST area significantly compared to the initial option of replicating the entire basic architecture for each memory. This is a preferred alternative over just sharing the SoC Interface and this is the basis for a "Parallel BIST".

The parallel BIST idea can easily be expanded to memories of different geometries if the pattern generation is dimensioned to the biggest memory being tested. In this case, the respective Memory Interface would also be responsible to decide if a particular address applies to its memory. This is easily done making the Memory Interface aware of the address range and data width of its memory.

The main advantage of the Parallel BIST architecture is that it can test all memories at the same time, and at speed with no interval between operations. Its basic structure is shown in figure 3.Making the Pattern Generator a common structure in the architecture brings another advantage. It is possible to make it more powerful without increasing too much the overall BIST area. Interesting features that can be added are selectable or programmable algorithms and configurable address ranges.

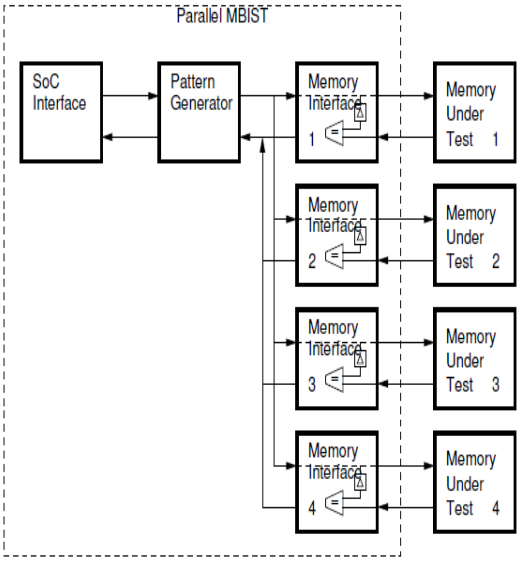

Fig. 3. Parallel Architecture.

V. SEQUENTIAL ARCHITECTURE

Configurability to individual basic blocks can enable new possibilities in terms of architecture choices. Suppose that instead of only being aware of the memory's geometry, the Memory Interface is able to adjust itself to several types of memories. The Pattern Generator could generate specific patterns for different memories, signaling to each memory whether the it is valid on a cycle by cycle basis. In this case, besides the enhanced Memory Interface and Pattern Generator blocks, only a multiplex layer would be required to connect the enhanced Memory Interface to all memories. This structure would be like the diagram in figure 4, and it is referred to as "Sequential BIST".

The Sequential BIST tests one memory at a time. It selects the first memory in its list and applies the march algorithm adjusted to that memory, selecting the

appropriate address range and other requirements. When the first memory is completely tested (i.e. the entire algorithm has been executed on this memory), it proceeds to the next memory, until all the memories are tested. In this case the test time required by the Sequential BIST will be larger compared to the Parallel BIST, but the BIST area will be much smaller if the number of memories is sufficiently large.

Note that in figure 4 the "Pattern Generator" block has been replaced by a new block called "Test Control". Note that in figure 4 the "Pattern Generator" block has been replaced by a new block called "Test Control".

What was previously called Pattern Generator is no longer a simple state machine that generates the read/write sequences for the test, now it also selects one memory at a time and then applies the March algorithm test, adjusting itself to the geometry of the selected memory.

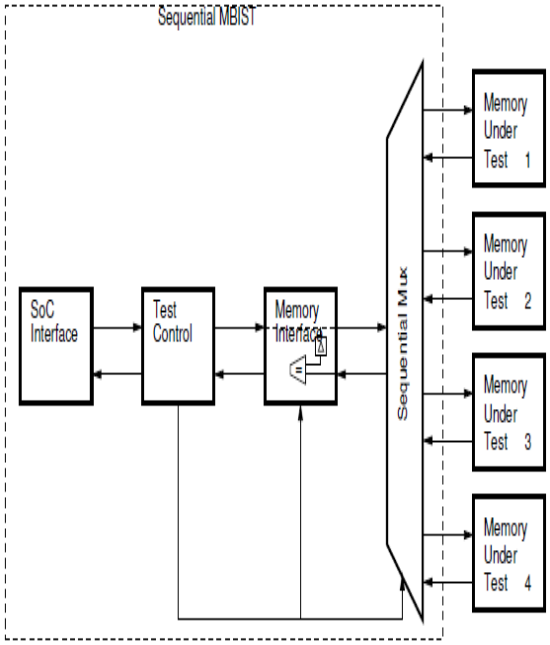

Fig. 4. Sequential Architecture.

A. SYSTEM INTERFACE AND REGISTER MODEL

The SoC Interface is an important part of the overall BIST architecture because it will determine whether BIST integration, verification and programming will be easy or cumbersome. Many solutions propose a serial interface, occasionally integrated to the DFT processes, others prefer more elaborate parallel protocols, making use of the already existent infrastructure of proprietary buses inside the SoC. All solutions have pros and cons; the important question is "how to enable all these choices". Figure 4 shows a breakdown of the SoC Interface block.

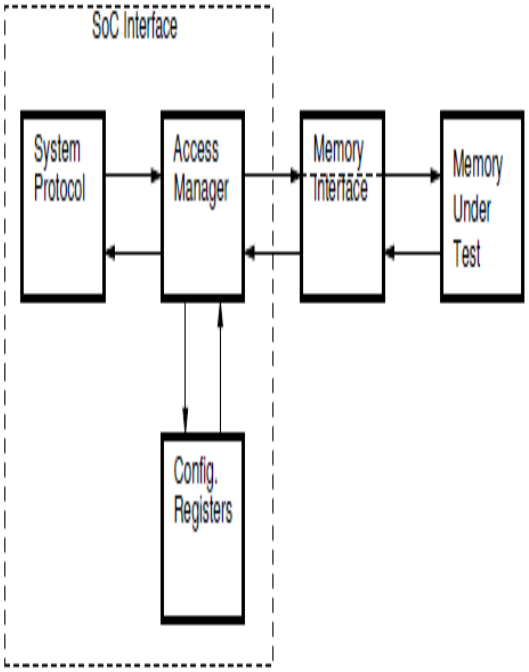

Fig. 5. SoC Interface.

The SoC Interface block is composed of three major blocks: The System Protocol, The Access Manager, (responsible for the internal accesses in the BIST) and the Configuration Registers. The SoC Interface is partitioned this way in order to decouple the internal logic from the

SoC. The System Protocol is designed to work as a bridge between the SoC and the internal BIST logic, in principle it can be replaced by any other access protocol, such as the serial IJTAG, or a parallel protocol such as ARM's APB [4].

The advantage of using a parallel interface, beyond the low latency transactions, is the possibility of accessing the memory directly independent of the functional path. This is called "backdoor access" and can be seen in figure 4 by the connection of the Access Manager block and the Memory Interface.

Other important aspect of the SoC Interface is the definition of a common set of registers for all possible BISTs in the family and all configurations. The purpose of these registers is to configure, program and collect feedback for the memory test, and control major aspects of BIST's operation. The use of a common set of configuration registers makes the test vector generation a much simpler task as test vectors can be easily ported among several different SoCs.

The register set is defined in a standard spreadsheet where address offset of each register from a base address, its bit-fields and the characteristics of this bit-field are specified. If a register or bit field is associated to a particular feature, this is also annotated in the spreadsheet. It contains textual descriptions of the registers and their bit-fields. In summary, the spreadsheet is the center specification of all software visible aspects of the BISTs architecture.

The spreadsheet is used to generate the RTL for the internal Configuration Register module, sections of the block guide, and many different machine register descriptions used by verification and test generation tools. All of these are derived from the register description spreadsheet.

B.RAM TEST:

In the Memory BIST Architecture section, a high level description of the RAM test was provided. It is time now to dive a little deeper into the details of each block to complete the picture of this architecture. Instead of discussing the component blocks, we will discuss specific aspects of the RAM test and then assign the implementation to the specific block. This way we will gradually construct a specification for each component blocks.

A. March Algorithm and Pattern Generation We saw there is a Pattern Generator block, which is responsible for the generation of read/write sequences for a particular march algorithm. We can expand its role now to select different address ranges and different march algorithms. Finally, to enable the Sequential BIST behavior, the block needs to also select one memory at a time and repeat the test multiple times to test all the memories. We see here that the original purpose of the state machine has expanded to encompass more than just patterns for the memories but also different aspects of the test. The block can now be named as the Test Control block. The Test Control block can be implemented in many ways.

It basically generates a sequence of operations. In circuit terms, it is just a state machine (FSM). There are many ways to implement this FSM. The obvious approach would be to consider all variations and implement a single FSM, but this would not

be efficient since some of the features might not be used resulting in a waste of logic. Having this in mind, a hierarchical design is implemented which allows insertion and removal of features as needed. The main idea is illustrated in figure 5.

The SRAM test consists of a sequence of operations that repeat themselves for every memory address. Having this in mind, the Test Control block is designed as a set of nested state machines, each one taking care of one aspect of the test. The lower FSMs in the figure 5 are the ones generating the March algorithm. The inputs of these FSMs are configurable Configuration Registers or are hard coded when RTL is generated. March Element (ME) and March Phase (MP) state machines are implemented as simple counters, while the Address Generator is a programmable up/down counter. Each state machine has its "start/done" signal linked to the upper FSM in the hierarchy. It works this way: when initiated (started), the ME Selector indicates the first March element of the test and then starts the Address Generator.

This points to the first memory location to be accessed and starts the MP Selector. The MP Selector applies all the March phases programmed/selected for that March element. When all the March phases have been executed, the MP Selector signals "done" to its predecessor. Receiving a "done" from the MP Selector, the Address Generator increments/decrements the current address to generate the next memory address and starts MP Selector again. This operation is repeated until all programmed/selected address range is covered.

Once the last address position is covered, the "done" signal from the MP Selector will also generate a "done" signal from the Address Generator to the ME Selector. At this point the ME Selector goes to a new ME selection starting the Address Generator again. Only when all March elements have been applied, the "done" signal from the ME Selector is generated to the next higher FSM in the hierarchy.

The example in figure 5 shows other levels in the control hierarchy that might be useful for the RAM test. There is a Background Selector above the March Algorithm FSM, that could select multiple data backgrounds for the test like checkerboard, solid 0/1, 55/AA, etc. There is also the Fast mode Selector that could be used to select address progression such as row-fast, column-fast or block-fast.

And finally, the Array Selector that selects the memory to apply the test. Other FSMs could easily be added to the hierarchy, the one shown in figure 5 is the Loop Counter that runs the test multiple times, which can be useful during fault analysis. The Test Control presented is suited for the Sequential BIST because it has the Array Selector integrated. For the Parallel BIST, the Array Selector can be eliminated, the chain of "start/done" signals bypass the unused Array Selector. The same principle applies to all hierarchical levels. For example, if no fast-modes are necessary the Fast-mode Selector block can be eliminated and the "start/done" chain could be go straight from the Array Selector to the Background Selector.

The opposite is also true: if a new level or hierarchy is required it can easily be

inserted in the Test Control block. For example, suppose it is necessary to include an automated dual-port switch to test a dual-port SRAM by both the primary as well as the secondary port. A new Port Selector could be inserted between Background Selector and ME Selector connecting the respective "start/done" signals according to the hierarchy. Note that all control levels are fully configurable and/or programmable to provide the most flexibility.

The March Algorithm blocks, for example, are adaptable to different algorithms that can be programmed in the Configurable Registers block or selectable pre-defined table. This configurability is controlled by the number of MEs and MPs that the selectors are expected to execute and the direction the Address Generator should progress.

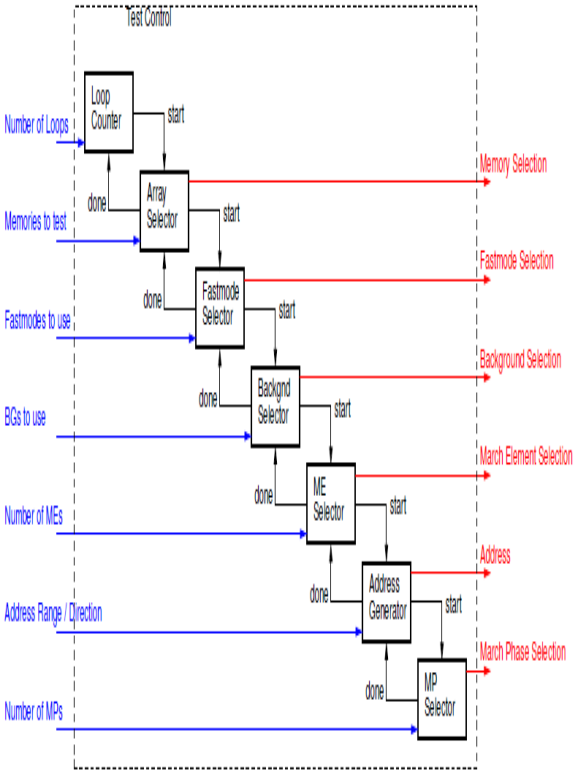

Fig. 6. Test Control.

VI. RESULTS

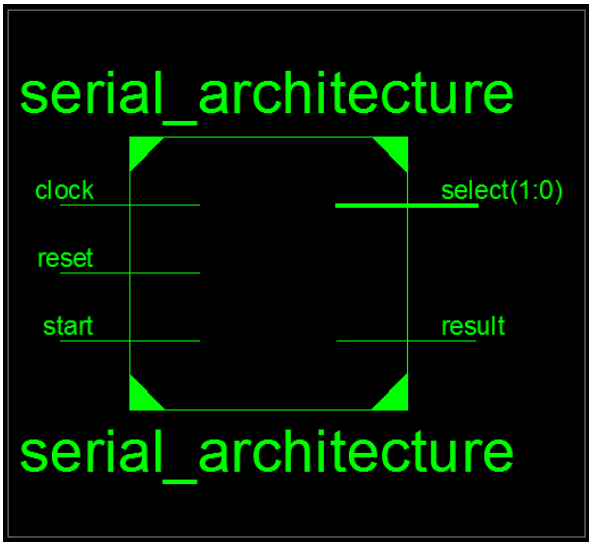

Fig7: RTL schematic view of serial architecture

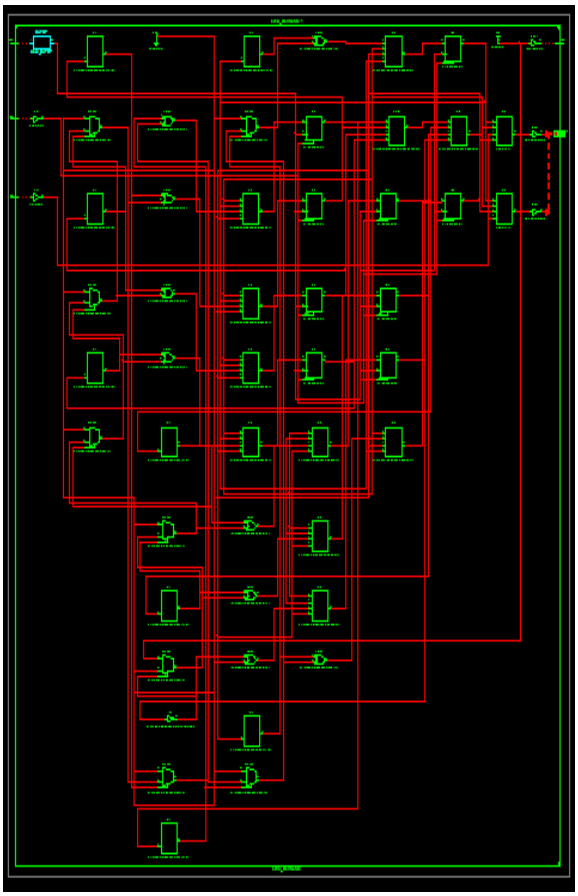

Fig 8: VIEW Technology Schematic of serial architecture

International Journal of Research
Available at https://edupediapublications.org/journals
P-ISSN: 2348-6848
E-ISSN: 2348-795X
Volume 07 Issue 06 June 2020

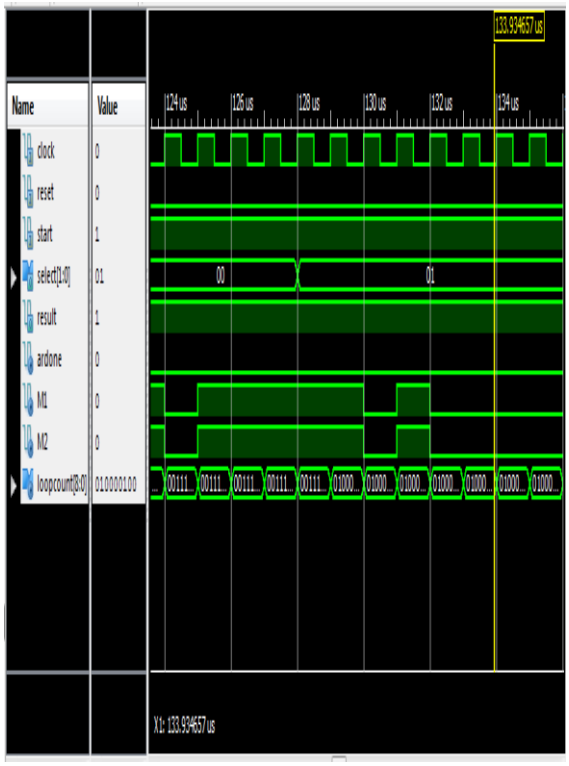

Fig 9: Simulated wave form of serial architecture

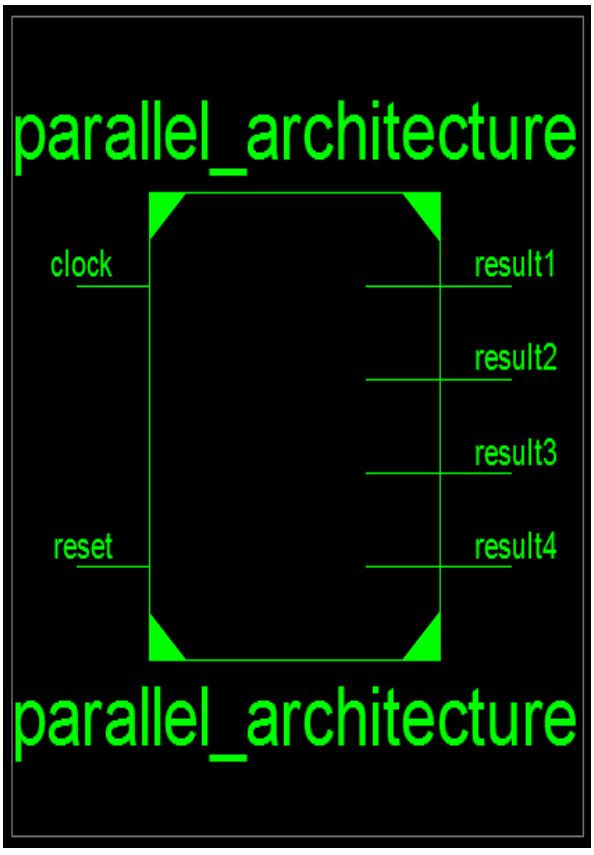

Fig10: RTL Schematic view of parallel architecture

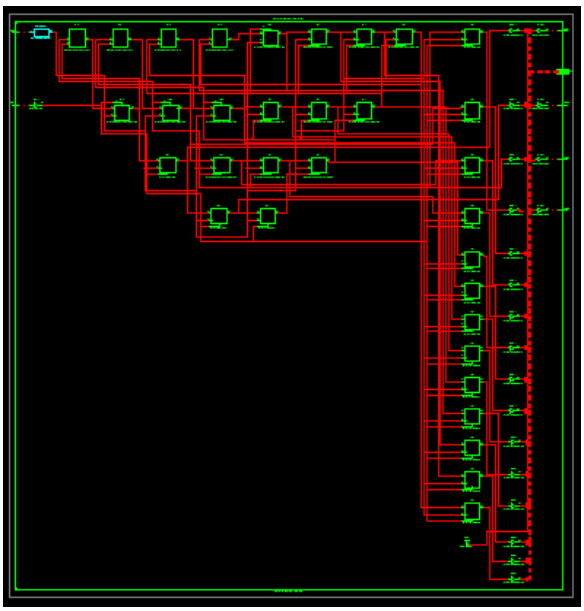

Fig11: View Technology Schematic of parallel architecture

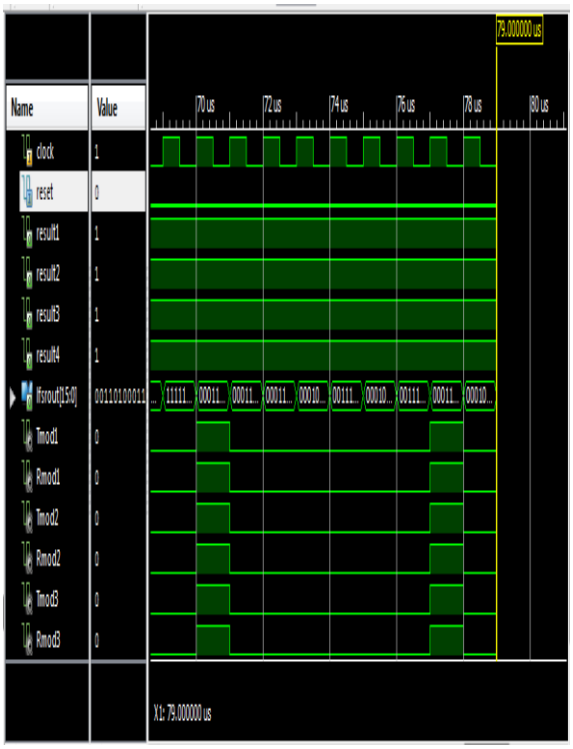

Fig12: simulated wave form of parallel architecture

Table .1: parameter comparison table

Parameter	Sequential Architecture	Parallel Architecture
Delay (ns)	6.236	4.394
Power (mW)	0.179	0.008

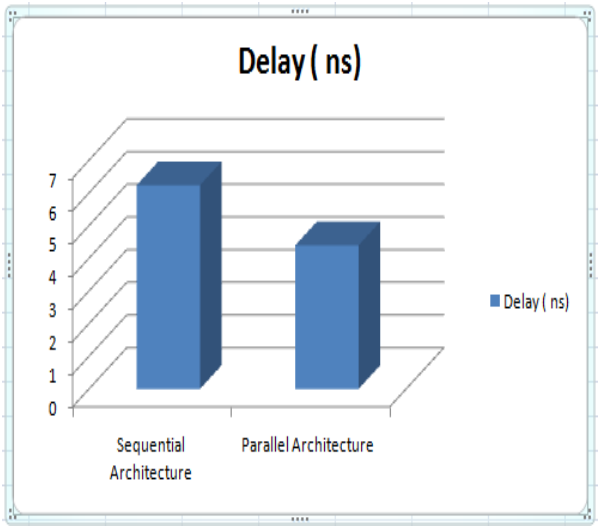

Fig 13: delay comparison bar graph

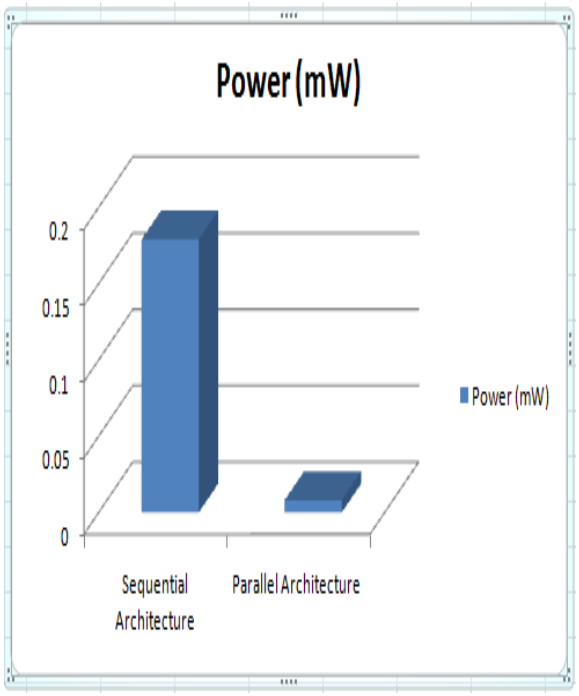

Fig 14: power comparison bar graph

VII. CONCLUSION AND FUTURE SCOPE

In this paper proposed A flexible memory BIST parallel architecture has been presented that allows the same basic architecture to be optimized for test time (by implementing a parallel BIST)configurable is better than the sequential architecture for MBIST those results are shown in table1. Additional optimizations are possible with ganging memories in either in address space or in data space. The principles presented in this paper may enabled a design team to quickly develop application specific BISTs such as ARM PORT BISTs intended to test high speed memories in ARM cores using ARM MBIST interface. The ARM PORT BIST utilizes one of BIST implementations described earlier that accesses memories through the ARM MBIST interface. TCAMBIST is another instance in which the BIST architecture discussed earlier was quickly modified to test Ternary Content Addressable Memories with very highest coverage. These BIST instances were developed rapidly by exploiting the common BIST state machine architecture described earlier.

REFERENCES

[1] W. B. Jone and D. C. Huang and S. C. Wu and K. J. Lee, An efficient BIST method for small buffers, Proceedings 17th IEEE VLSI Test Symposium (Cat. No.PR00146), 246-252, 1999.

[2] A. J. van de Goor - Testing Semiconductor Memories: Theory and Practice, John Wiley & Sons, 1991.

[3] IEEE Standard 1687-2014 - IEEE Standard for Access and Control of

Instrumentation Embedded within a Semiconductor Device, IJTAG - Internal Joint Test Action Group, IEEE Computer Society. https://standards.ieee.org/findstds/standard/1687-2014.html

[4] ARM Limited - AMBA™3 APB Protocol, ARMIHI 0024B. http://www.arm.com

[5] Free scale Semiconductor - SRS IP Interface, SRS V4.0 01 Dec 2007.

[6] Allen C. Cheng - Comprehensive Study on Designing Memory BIST: Algorithms, Implementations and Trade-Offs, Project Report, Advanced Computer Architecture Lab, University of Michigan, 2002.

[7] Mohamed Azimane and Ananta K. Majhi - New Test Methodology for Resistive Open Defect Detection in Memory Address Decoders, Proc. Of the 22nd IEEE VLSI Test Symposium, 2004.

[8] Manoj Sachdev - Open Defects in CMOS RAM Address Decoders, IEEE Design & Test of Computers, 1997.

[9] M. Miyazaki, T. Yoneda and H. Fujiwara - A Memory Grouping Method for Sharing Memory BIST Logic, Asia and South Pacific Conference on Design Automation, 2006.

[10] ARM Cortex-A15 MP Core Processor, Technical Reference Manual, ARM-DDI-04381 ID062913 4.0, 2013, http://www.arm.com.