Performance Analysis, Designing and Testing 512 Bit Sram Memory Chip Using Xilinx/Modelsim Tool

Monika Solanki*

Department of Electronics & Communication Engineering, MBM Engineering College, Jodhpur, Rajasthan

Research Article

Received date: 07/12/2016

Accepted date: 18/01/2017

Published date: 24/01/2017

***For Correspondence**

Monika Solanki, Department of Electronics & Communication Engineering, MBM Engineering College, Jodhpur, Rajasthan, India, Tel: 0291-2551566.

E-mail: rahulliger19@gmail.com

Keywords: Built-in self-test (BIST), Memory, Architecture, Software

ABSTRACT

In this thesis a memory chip has been designed with the size of 512 bit using Xilinx or model sim software. Memory Built-in Self-Test (MBIST) or as some refer to it array built-in self-test is an amazing piece of logic. Without any direct connection to the outside world, a very complex embedded memory can be tested efficiently, easily and less costly.

Modelling and simulation of MBIST is presented in this paper. The design architecture is written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) code using Xilinx ISE tools. Verification of this architecture is carried out by testing stuck at fault SRAM. A BIST algorithms is implemented like March C- and many more to test the faulty SRAM.

INTRODUCTION

Fast low power SRAMs are becoming the critical component of many VLSI chips. There is increasing divergence in the speed of the processors and the main memory, and the power dissipation is also increasing due to increase in the integration and the operating speed as well as increase in the battery powered devices. The SRAM helps in bridging the gap and also reducing the power dissipation. After designing the memory, we will go for testing the fault in the memory.

Memory Testing

Built-in self-test (BIST) technique is widely used to test and diagnose the random access memories (RAMs) [1]. To support the diagnosis, a BIST circuit exports the data to be diagnosed serially due to the limitation of test I/Os. Clearly, the diagnostic data exported is time consuming, because they are exported bit by bit. To reduce this time consumption, several diagnostic data compression techniques are there. Various compression methods are proposed to compress the data to be diagnosed in a pause and export way. Thus, BIST circuit is paused when fault is detected and then the fault is compressed and exported serially.

Fault Model of Sram

To Study the fault detection method of memory, we have to build fault models of memory first. There are three simplified parts where memory faults occur [2]. The three parts are: address decoder, read and write logic, memory cell array. The former two parts are equal to the latter in function; we only need to detect the memory cell array.

Following are the types of faults also called fault model:

- Stuck-at fault
- State transition fault
- Coupling fault
- Addressing fault
- Data retention fault

Stuck-at fault: Abbreviated as SAF. In this type of fault model a unit or a line of memory is being assumed to be stuck at logical “1” or at logical “0”.

State transition fault: It is a part of stuck at fault, in which a unit or a line of memory that cannot achieve 0-1 or 1-0 conversion after a writing operation. These are called the rising state transition fault and falling state transition fault separately [3].

Coupling fault: It includes two units, one unit's state changing causes the other's unit's state changing accordingly.

Addressing fault: In this type of fault model, a row or a column decoder may not access the addressing unit, or multiple addresses access the same storage unit, or one address accesses several cells simultaneously, or accessing other unit instead of the same specified unit.

Stuck open fault: In this type of fault model, a storage unit cannot be addressed. If storage unit has only one input port, it will generate only a fixed output value.

Data retention fault: In this type of fault model, a storage unit cannot effectively keep its data value unchanged within specified time.

METHODOLOGY

Ram Bist Architecture

The architecture of RAM BIST is shown below. In the first block of RAM BIST architecture "ROM based Algorithm Generator" has been shown in which all 8 algorithm are kept in an ideal state [4]. This algorithm will remain in an ideal state unless BIST_EN signal is made equal to "1" (or it can be understood as until BIST_EN signal is made equal to "1", the BIST operation would not be started). Among these 8 algorithms, one of the algorithms will be selected depending on the selection line of 8:1 MUX. Then selected algorithm is passed to "Algorithm Decoder". The read/write signal from this block is transferred to the Embedded RAM to define weather read operation or write operation will be performed on to RAM Address [5,6]. The up/down signal is passed to the "Address Generator" to define whether a Read / Write operation is performed in "Up addressing mode" or "Down addressing mode", it is decided by selecting the algorithm operation **(Figure 1)**.

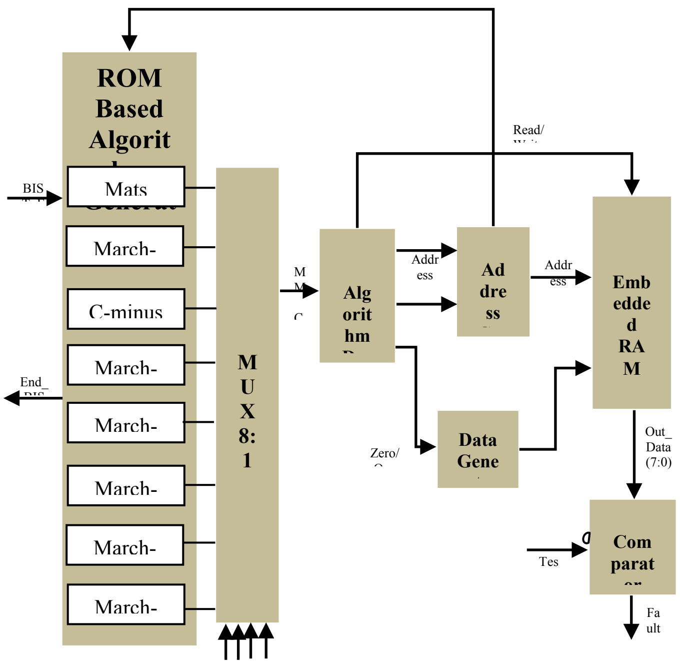


Figure 1. RAM BIST architecture.

Then third signal Zero /One is passed to the "Data Generator" to define that read/write operation is performed on "0" bit or "1" bit [7]. Hence according to the state of selected algorithm, read/write operation is performed onto the RAM. Whenever "Read operation" performed then the data comes out of RAM and that data is transferred to the "Comparator". In comparator there is another input port (that is test data) in which test input is given as same as that of data written onto "Embedded RAM".

The comparator will check that both inputs are same or not, if there is some fault on (like stuck at fault or coupling fault), then the output data coming out of "Embedded Memory", is different from the input given in to the comparator. So it will show a fault by making " fault detect" signal "equal to 1", if there is no fault in memory then "fault detect" will be equal to the "0" On the other hand if "Write" is being performed then comparator will be in an ideal state.

Algorithm for Ram Testing

There are many types of test algorithms that are used for RAM testing [8]. These algorithms are rather simple for BIST implementations. Table shown below includes various test algorithms for RAM testing along with the instructions which are performed by those algorithms **(Table 1)**.

Table 1. Algorithms for ram testing.

No.	Algorithm	March Elements Code
000	MATS+	{↕(w0); ↑(r0,w1); ↓(r1,w0)}
001	March X	{↕(w0); ↑(r0,w1); ↓(r1,w0), ↕(r0)}
010	March C-	{↕(w0); ↑(r0,w1); ↓(r1,w0); ↕(r0,w1); ↓(r1,w0), ↕(r0)}
011	March A	{↕(w0); ↑(r0,w1,w0,w1); ↑(r1,w0,w1); ↓(r1,w0,w1,w0); ↓(r0,w1,w0)}
100	March B	{↕(w0); ↑(r0,w1,r1,w0,r0,w1); ↑(r1,w0,w1); ↓(r1,w0,w1,w0); ↓(r0,w1,w0)}
101	March U	{↕(w0); ↑(r0,w1,r1,w0); ↑(r0,w1); ↓(r1,w0,r0,w1); ↓(r1,w0)}
110	March LR	{↕(w0); ↓(r0,w1); ↑(r1,w0,r0,w1); ↑(r1,w0); ↑(r0,w1,r1,w0); ↑(r0)}
111	March SS	{↕(w0); ↑(r0,r0,w0,r0,w1); ↑(r1,r1,w1,r1,w0); ↓(r0,r0,w0,r0,w1); ↓(r1,r1,w1,r1,w0); ↕(r0)}

Among all these test algorithms, March C- algorithm is practical to offer the highest fault coverage. The capabilities of the fault detection of March test algorithms are summarized in Table below [9]. All these test algorithms are planned for RAMs with one data bit per word, but the multiple bits per word can be applied as well. To increase the fault detection rate, modification on the algorithm is required for sensitivity and coupling faults in RAM **(Table 2)**.

Table 2. The summarized fault detection of march test algorithms.

Algorithm	Struck-at	Address	Transition	Coupling
MATS	yes	some	no	no
MATS+	yes	yes	no	no
MATS++	yes	yes	yes	no
March X	yes	yes	yes	some
March Y	yes	yes	yes	some
March C-	yes	yes	yes	yes

RESULTS AND DISCUSSION

There are various types of tools which can be used as a simulator for executing the step to test the design. This is done by the use of the test bench [10]. A test bench is a set of the test stimuli timing corresponding to the circuit design. The response of the circuit which is under test is then can be read out in the form of waveforms.

The test bench file is responsible for providing the input stimuli to the memory under test (MUT) like clock and various other test control signals. Then the response will be produced in terms of the failure signals which are detected and displayed in the form of waveforms [11]. In this project we have used Model sim simulator. The result of the designing is shown in the following section in the form of screenshots of the software. First of all we are showing the screenshot of the RTL schematic which is written in the Xilinx software **(Figure 2)**.

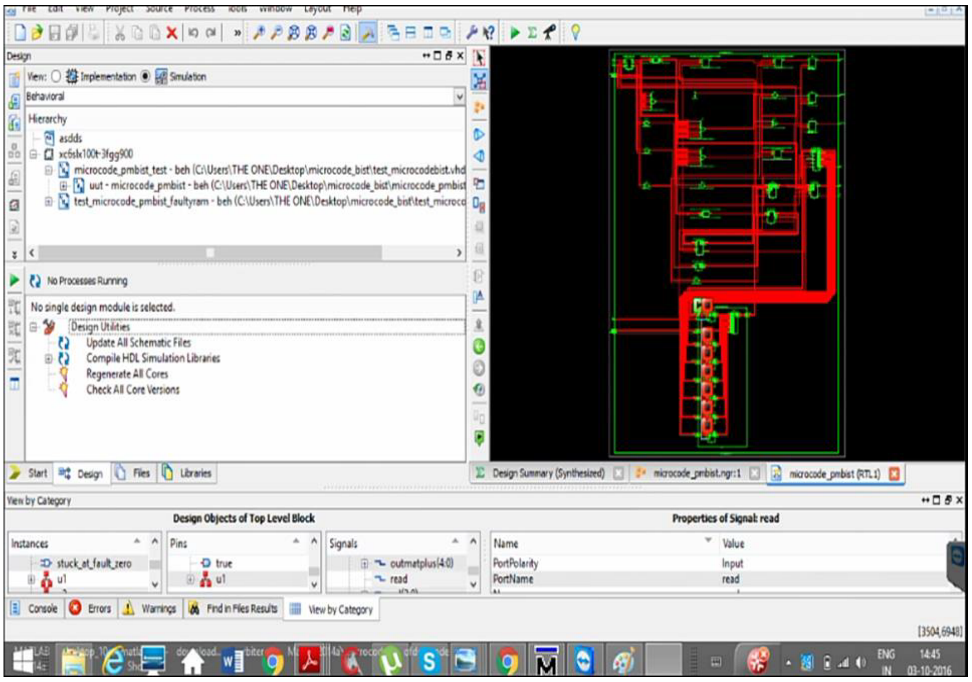
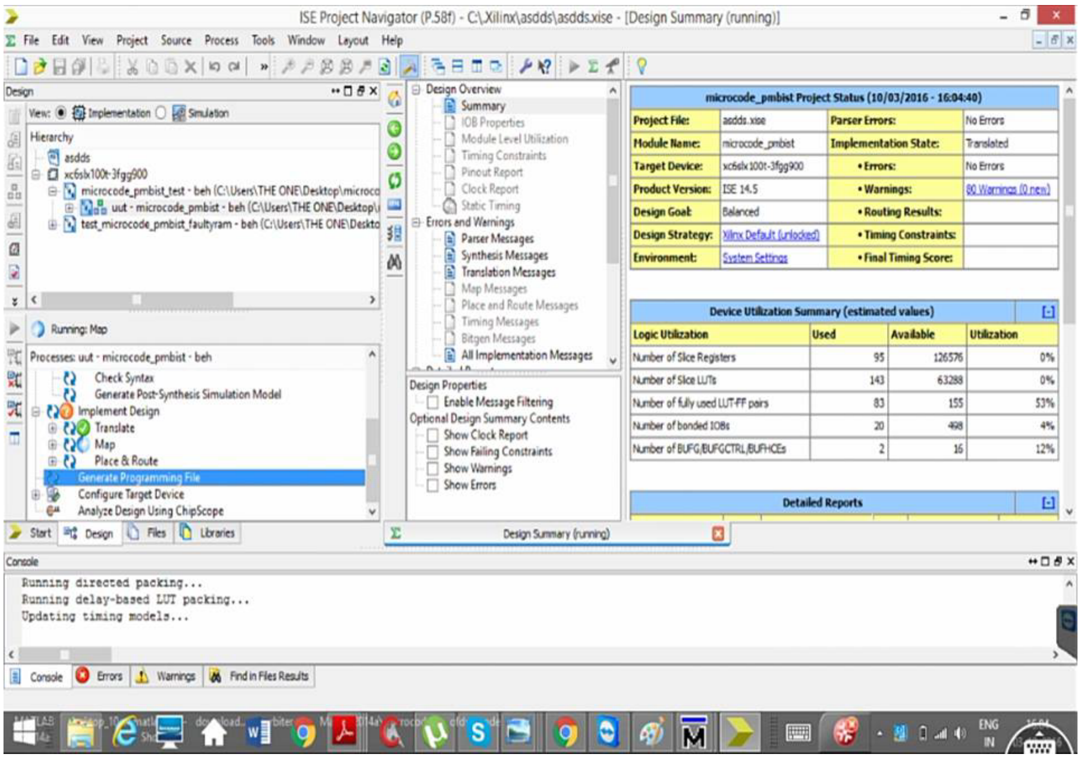

Design
View: Implementation Simulation
Behavioral
Hierarchy
asdds
xc6slx100t-3fgg900
microcode_pmbist_test - beh (C:\Users\THE ONE\Desktop\microcode_bist\test_microcodebist.vhd
uut - microcode_pmbist - beh (C:\Users\THE ONE\Desktop\microcode_bist\microcode_pmbist
test_microcode_pmbist_faultyram - beh (C:\Users\THE ONE\Desktop\microcode_bist\test_microco
No Processes Running
No single design module is selected.
Design Utilities
Update All Schematic Files
Compile HDL Simulation Libraries
Regenerate All Cores
Check All Core Versions
Start
Design
Files
Libraries
Design Summary (Synthesized)
microcode_pmbist.ngr:1
microcode_pmbist (RTL1)
View by Category
Design Objects of Top Level Block
Properties of Signal: read
Instances
stuck_at_fault_zero
u1
Pins
true
u1
Signals
outmatplus(4:0)
read
Name
Value
PortPolarity
Input
PortName
read
Console
Errors
Warnings
Find in Files Results
View by Category

Figure 2. RTL schematic in xilinx software.

For this the software first asks for the selection of the elements. After selecting the elements we have to click on the tab Create Schematic [12]. As soon as we click on this tab, we see the RTL schematic of the program **(Figure 3)**.



ISE Project Navigator (P.58f) - C:\Xilinx\asdds\asdds.xise - [Design Summary (running)]

microcode_pmbist Project Status (10/03/2016 - 16:04:40)			
Project File:	asdds.xise	Parser Errors:	No Errors
Module Name:	microcode_pmbist	Implementation State:	Translated
Target Device:	xc6slx100t-3fgg900	• Errors:	No Errors
Product Version:	ISE 14.5	• Warnings:	80 Warnings (0 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)

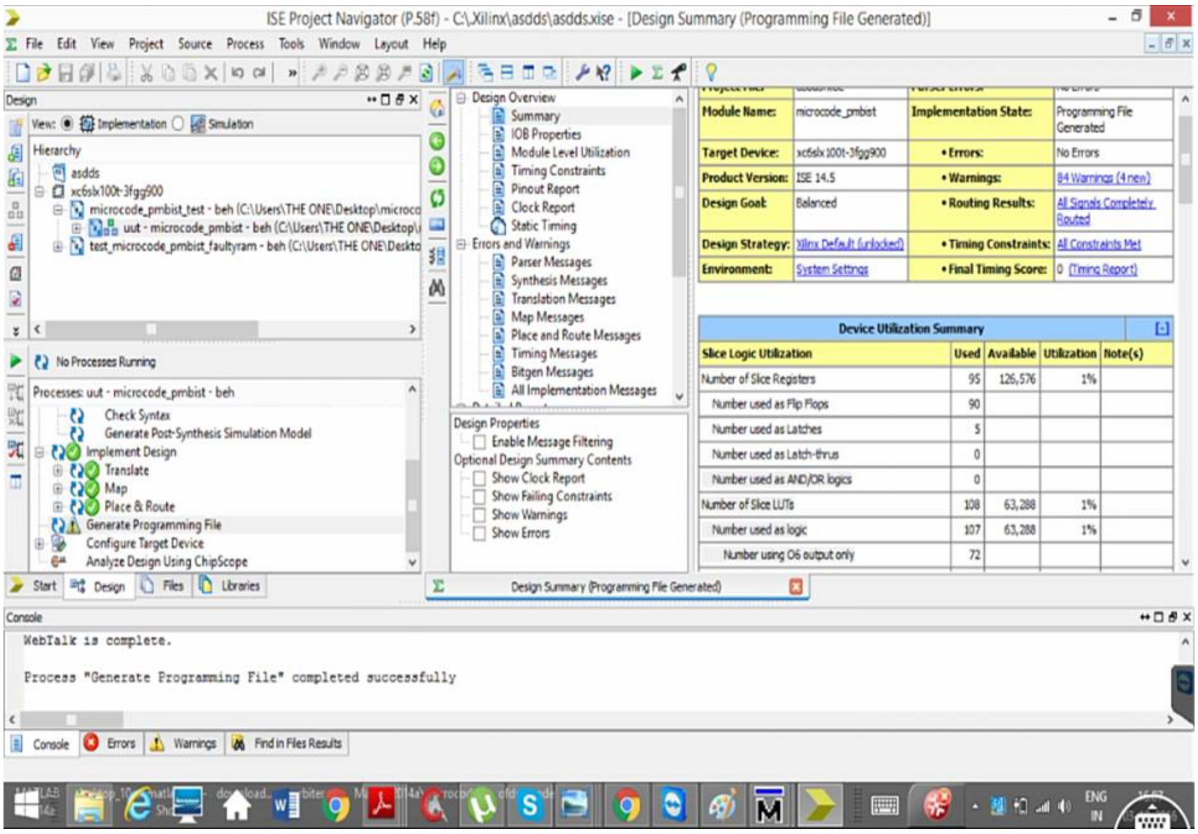
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	95	126576	0%
Number of Slice LUTs	143	63288	0%
Number of fully used LUT-FF pairs	83	155	53%
Number of bonded IOBs	20	498	4%
Number of BUFG/BUFGCTRL/BUFHCEs	2	16	12%

Detailed Reports

```
Running directed packing...
Running delay-based LUT packing...
Updating timing models...
```

Figure 3. Design summary in xilinx software.

We can see all the details of the program in the design summary. The detail is shown in the form of table. This picture is shown in the following **(Figure 4)**.



ISE Project Navigator (P.58f) - C:\Xilinx\asdds\asdds.xise - [Design Summary (Programming File Generated)]

Module Name:	microcode_pmbist	Implementation State:	Programming File Generated
Target Device:	xc6slx100t-3fgg900	• Errors:	No Errors
Product Version:	ISE 14.5	• Warnings:	84 Warnings (4 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary

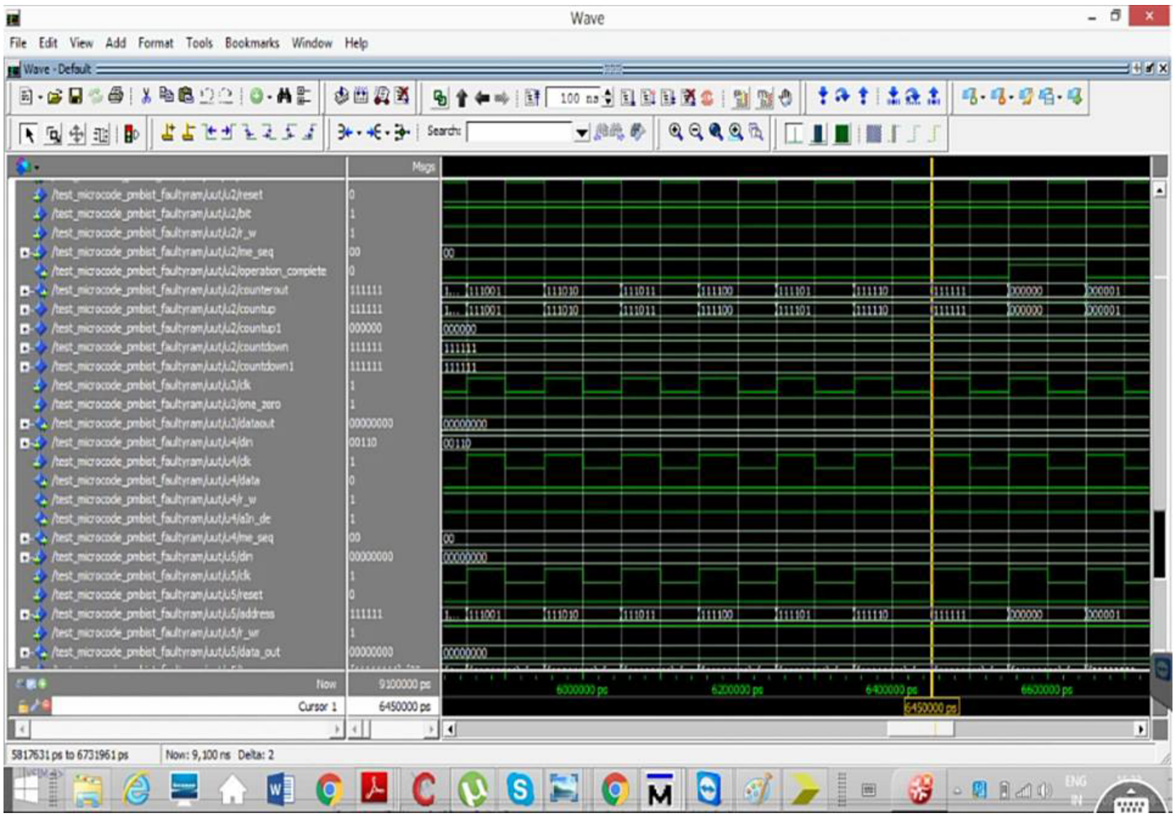
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	95	126,576	1%	
Number used as Flip Flops	90			
Number used as Latches	5			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	108	63,288	1%	
Number used as logic	107	63,288	1%	
Number using O6 output only	72			

```
WebTalk is complete.

Process "Generate Programming File" completed successfully
```

Figure 4. Design summary in xilinx software.

After doing all this we will simulate the VHDL code using the ISim simulator or modelsim software [13]. The waveforms of all the input and output parameters are displayed on the screen. We can then provide the input and observe the output **(Figure 5)**.



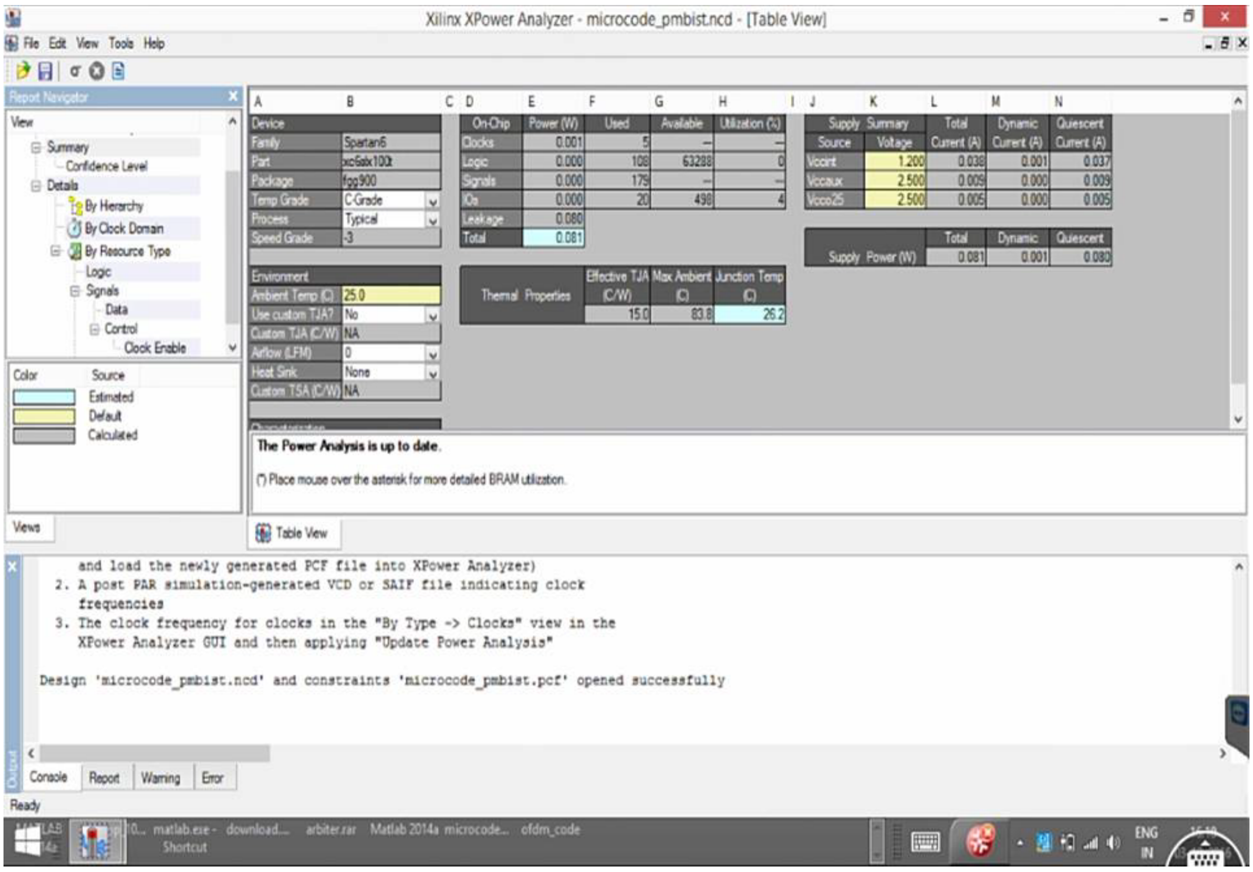
```
/test_microcode_pmbist_faultyram/uut/u2/reset                 0
/test_microcode_pmbist_faultyram/uut/u2/bit                   1
/test_microcode_pmbist_faultyram/uut/u2/r_w                   1
/test_microcode_pmbist_faultyram/uut/u2/me_seq                00
/test_microcode_pmbist_faultyram/uut/u2/operation_complete    0
/test_microcode_pmbist_faultyram/uut/u2/counterout            111111
/test_microcode_pmbist_faultyram/uut/u2/countup               111111
/test_microcode_pmbist_faultyram/uut/u2/countup1              000000
/test_microcode_pmbist_faultyram/uut/u2/countdown             111111
/test_microcode_pmbist_faultyram/uut/u2/countdown1            111111
/test_microcode_pmbist_faultyram/uut/u3/clk                   1
/test_microcode_pmbist_faultyram/uut/u3/one_zero              1
/test_microcode_pmbist_faultyram/uut/u3/dataout               00000000
/test_microcode_pmbist_faultyram/uut/u4/din                   00110
/test_microcode_pmbist_faultyram/uut/u4/clk                   1
/test_microcode_pmbist_faultyram/uut/u4/data                  0
/test_microcode_pmbist_faultyram/uut/u4/r_w                   1
/test_microcode_pmbist_faultyram/uut/u4/aln_de                1
/test_microcode_pmbist_faultyram/uut/u4/me_seq                00
/test_microcode_pmbist_faultyram/uut/u5/din                   00000000
/test_microcode_pmbist_faultyram/uut/u5/clk                   1
/test_microcode_pmbist_faultyram/uut/u5/reset                 0
/test_microcode_pmbist_faultyram/uut/u5/address               111111
/test_microcode_pmbist_faultyram/uut/u5/r_wr                  1
/test_microcode_pmbist_faultyram/uut/u5/data_out              00000000

Now       9100000 ps
Cursor 1  6450000 ps

5817631 ps to 6731961 ps    Now: 9,100 ns  Delta: 2
```

Figure 5. Waveform in xilinx software.

The power report of the memory designed is shown below which will be obtained in the Xilinx Xpower analyzer **(Figure 6)**.



Xilinx XPower Analyzer - microcode_pmbist.ncd - [Table View]

Device	
Family	Spartan6
Part	xc6slx100t
Package	fgg900
Temp Grade	C-Grade
Process	Typical
Speed Grade	-3

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.001	5	---	---
Logic	0.000	108	63288	0
Signals	0.000	179	---	---
IOs	0.000	20	498	4
Leakage	0.080			
Total	0.081			

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.038	0.001	0.037
Vccaux	2.500	0.009	0.000	0.009
Vcco25	2.500	0.005	0.000	0.005

	Total	Dynamic	Quiescent
Supply Power (W)	0.081	0.001	0.080

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0
Heat Sink	None
Custom TSA (C/W)	NA

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	15.0	83.8	26.2

The Power Analysis is up to date.

(*) Place mouse over the asterisk for more detailed BRAM utilization.

```
   and load the newly generated PCF file into XPower Analyzer)
2. A post PAR simulation-generated VCD or SAIF file indicating clock
   frequencies
3. The clock frequency for clocks in the "By Type -> Clocks" view in the
   XPower Analyzer GUI and then applying "Update Power Analysis"

Design 'microcode_pmbist.ncd' and constraints 'microcode_pmbist.pcf' opened successfully
```

Figure 6. Power analysis in xilinx software.

Now after the overall designing and testing of the memory, we will connect it to the FPGA kit through Xilinx software [14]. The screenshot is shown below **(Figures 7 and 8)**.

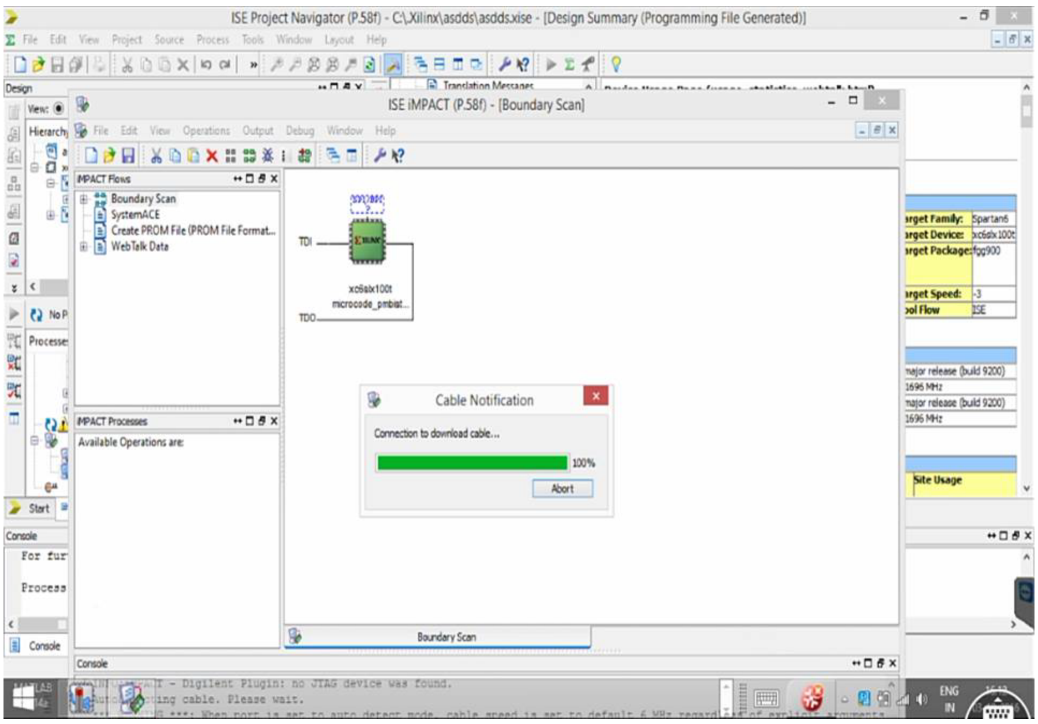

ISE Project Navigator (P.58f) - C:\Xilinx\asdds\asdds.xise - [Design Summary (Programming File Generated)]
ISE iMPACT (P.58f) - [Boundary Scan]
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
TDI
TDO
xc6slx100t
microcode_pmbist...
Cable Notification
Connection to download cable...
100%
Abort
Boundary Scan
Console

Figure 7. Connection with FPGA 1.

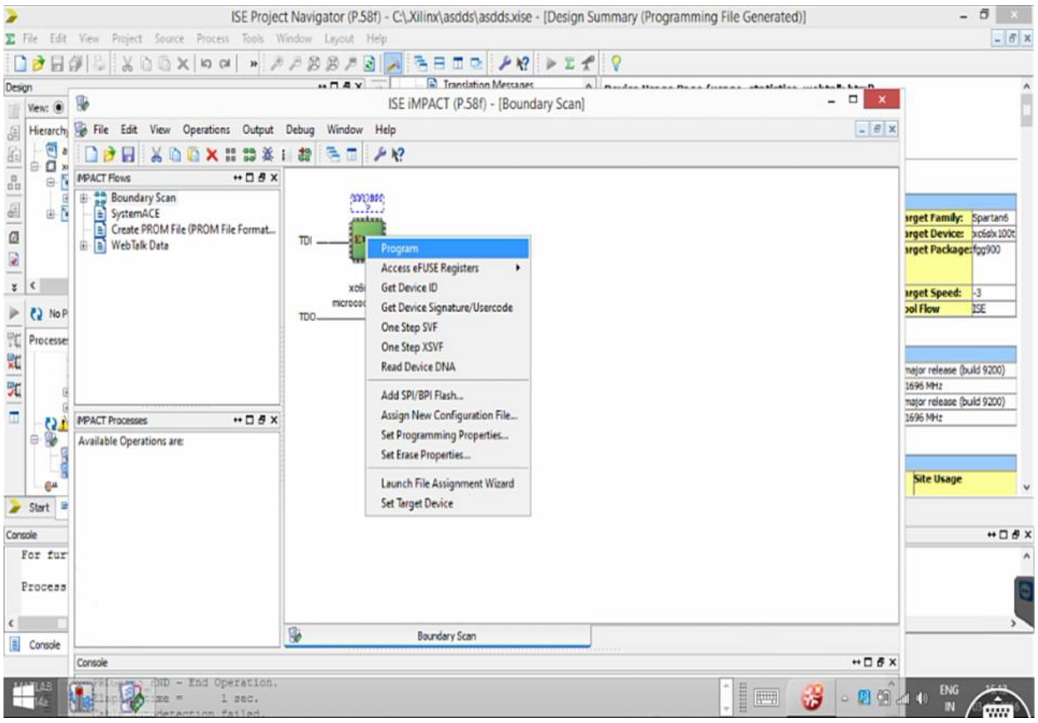

ISE Project Navigator (P.58f) - C:\Xilinx\asdds\asdds.xise - [Design Summary (Programming File Generated)]
ISE iMPACT (P.58f) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
Program
Access eFUSE Registers
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Read Device DNA
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Set Target Device
iMPACT Processes
Available Operations are:
Boundary Scan
Console

Figure 8. Connection with FPGA 2.

CONCLUSION

This paper presents the design of 512 bit SRAM memory. We chose 6T SRAM as memory bit cell and made an array designed with that bit cell. We came to know many things about the low power implementation of this 6T SRAM by going through many papers like making a 7T SRAM or 8T SRAM. The work in this respect can be taken forward to make a low power implementation. We have grasped many important concepts and learned tools that will help us in future.

In brief, a random access memory with built in self-test has been successfully designed. The BIST techniques are divided as online and offline testing. This project is designed at the entry level of VHDL so there are lots of modifications that can be done and additional architecture can be added in order to increase the robustness of the design.

- A March C- algorithm can be used in the design by simply modifying the decoder in our project. This is done because March C- algorithm is better able to cover the fault as compared to the other algorithms.
- We can increase the bit size so that more memory locations can be tested by us.
- BIST is not able to insert the fault model. So future work can be done in order to insert and then detect a set of fault models.
- We can reduce the hardware coverage of the project by replacing the comparator and counter which we have used. Thus performance of our project will be improved.
- We can replace VHDL with Verilog HDL to reduce the command line for future work.
- Due to non-availability of resources, we were not able to program this project into actual FPGA hardware. So the future works can be done to run all these testing through FPGA hardware device to verify our work.

REFERENCES

1. Haron NZ, et al. Modeling and Simulation of Microcode Memory Built-in Self-Test Architecture for Embedded Memories. Communications and Information Technologies, 2007. International Symposium; 2007.
2. Yadav S, et al. Low Power SRAM Design with Reduced Read/Write Time. International Journal of Information and Computation Technology. 2013;3:195-200.
3. Kanoujia D and Moyal V. Survey on Various Works Done In Reducing Static Power In Various SRAM Cells. International Journal Of Technology Enhancements and Emerging Engineering Research, 2014;2:27-29.
4. Rao KS and Suman JV. Low Power Design of A SRAM Cell for Embedded Memory. International Journal of Research in Computer and Communication Technology, 2013;2:1222-1228.
5. Bellerimath PS and Banakar RM. Implementation of 16X16 SRAM Memory Array using 180nm Technology. International Journal of Current Engineering and Technology. 2013.
6. Chih-Sheng H, et al. A BIST Scheme with the Ability of Diagnostic Data Compression for RAMS. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014;33:2020-2024.
7. Chuanpei X, et al. BIST Method of SRAM for Network-on-Chip. Electronic Measurement & Instruments (ICEMI), 12th IEEE International Conference; 2015.
8. Husin MH, et al. Built in Self-test for RAM Using VHDL. IEEE Colloquium on Humanities, Science & Engineering Research; 2012.
9. Goor JVD, et al. Generic, Orthogonal and Low-cost March Element based Memory BIST. Test Conference (ITC), 2011 IEEE International; 2012.
10. Fischerova M and Martin Simlastik M. MemBIST Applet for Learning Principles of Memory Testing and Generating Memory BIST. Digital System Design, Proceedings. 8th Euromicro Conference; 2005.
11. Bosio A, et al. An Effective BIST Architecture for Power-Gating Mechanisms in Low-Power SRAMs. Quality Electronic Design (ISQED), 17th International Symposium; 2016.
12. Gadde P and Niamat M. FPGA Memory Testing Technique using BIST. Circuits and Systems (MWSCAS), IEEE 56th International Midwest Symposium; 2013.
13. Manikandan P, et al. A Programmable BIST with Macro and Micro codes for Embedded SRAMs. Design & Test Symposium (EWDTS), 9th East-West; 2011.
14. Mishra B, et al. Low Power BIST based Multiplier Design and Simulation using FPGA. Electrical, Electronics and Computer Science (SCEECS), IEEE Students Conference; 2016.