Asian Journal of Engineering and Applied Technology
ISSN 2249-068X Vol. 2 No. 1, 2013, pp.21-27

Implementation of ReBISR Scheme for RAMs Using Spare Elements

Shweta Meena

[1]PG Scholar, Aurora's Scientific Technological and Research Academy, Hyderabad - 500 005, Andra Pradesh, India

Email: mail2shwetameena@gmail.com

(Received on 18 February 2013 and accepted on 25 May 2013)

Abstract – **Key components of SOCs are memories which come with different sizes and different configurations. Memories usually constitute a major portion of the chip area. By improving the yield of RAM improves the yield of chip. Diagnostics for yield improvement of the memories thus is a very important issue. This paper presents a Built-in Self Repair scheme to repair the memories for yield improvement of the chip using redundancy analysis algorithm. The proposed BISR scheme has three phases. In the first phase BIST is used to detect the faulty location in the memory. In order to determine a correct repair solution, spare memories are allocated in the second phase using BIRA circuitry. Finally, in the third phase the actual repair process is carried out using BISR circuitry. Experimental results show that the proposed BISR algorithm achieves optimal repair rate and low area cost.**

Keywords: **Memory built-in self test (MBIST), Built-in redundancy analysis (BIRA), Writing0/ writing1 algorithm, Built-in self repair**

I. Introduction

As VLSI devices are becoming more and more complex, there is a need for high density memories for implementation of system chip. As SOC size is shrinking, the major area on SOC is occupied by embedded memories. Thus memories in chip will decide the yield of the SOC. Since chip area is a major constrain, RAM's are subjected to manufacturing defects due to dense packing. That is, RAMs have more serious problems of yield and reliability in an SOC. Generally much of the low yield of the chip is due to faults in the memory. Hence there is a need to repair these memories for yield improvement. For such purpose, usually redundancy analysis is implemented using spare elements i. e. spare rows and/or spare columns. Conventionally, redundancy analysis (RA) is performed using Automatic Test equipment (ATE). But ATE is a time taking process since RA algorithms using ATE are complex, costly and the memories that implement the redundancies are usually large. Most SOCs adopt a built-in self-test and built-in redundancy analysis to test and repair their embedded memories instead of using external ATE because this method is more cost-effective and less time consuming. 2-D redundancy approach is proposed in this paper for carrying out the repair process.

Many BISR schemes have been proposed for repairing RAMs. BIRA algorithm is one key component of a BISR scheme, and it is responsible for allocating redundancies for defective RAMs. The BIRA algorithm called Fault Checking Algorithm is presented in this paper. The BIRA modules can be used in parallel to allocate redundancy. However, the numbers of BIRA modules will be increased with respect to the number of redundancies of the defective RAMs. This results in very high area cost. The solution to the above problem is to use a single BISR circuit. The ReBISR can be shared by multiple RAMs with different sizes and redundancy organizations. This can reduce the area cost of the BISR circuits in an SOC.

II. BISR Architecture

The memory BISR (MBISR) contains an interface between memory BIST (MBIST) and redundancy wrapper for storing faulty addresses. The MBIST controller output must provide three signals to the wrapper logic during test. A fail signal to store data in the fuse register, the expected data that is compared to the results of RAM data and the failing address.Fig.1shows the architecture of MBISR scheme for a RAM, which consists of four major components.

1) ***Repairable RAM:*** A RAM with redundancies, address decoder and repair register is called a repairable RAM. Figure 2 depicts an example of an 8*8 bit-oriented RAM with 1 spare row and column. Address decoder is used to select the column and row where our desired data lies. Sense amplifier assists the charging of the lines so that the memory cell doesn't discharge and lose its data. Row repair address and column repair address are the address of the defective row and column respectively. When RAE is high, the RRA is decoded using decoder into control signal to replace the faulty row. Similarly when the CAE is high, the CRA used to repair the faulty column.

2) ***BIST Circuit:*** It generates test patterns for repairable RAM using TPG during testing. If a fault is detected in the defective RAM by the BIST circuit, then the faulty information is sent to BIRA circuit.

3) ***BIRA Circuit:*** If BIST detects a fault, then the faulty information is exported to the BIRA circuitry and then BIRA allocates redundancies according to the faulty information provided by the BIST circuit using the redundancy analysis algorithm.

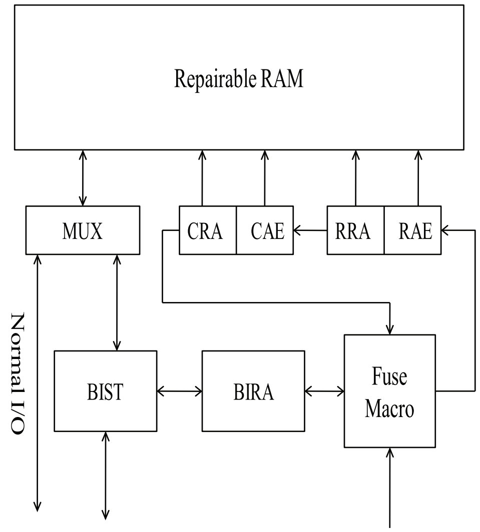


Fig. 1 Architecture of BISR Scheme for RAMs

The overall RAM BISR flow as depicted in Fig.1.is described as follows. Firstly, the BIST tests the repairable RAM. If BIST detects a fault, then the fault information is exported to the BIRA circuit. Then, the BIRA circuit collects the faulty information provided by the BIST circuit and allocates redundancies to replace defective elements. When the repair process finished, the repair signatures are loaded into the repair register in the repairable RAM.

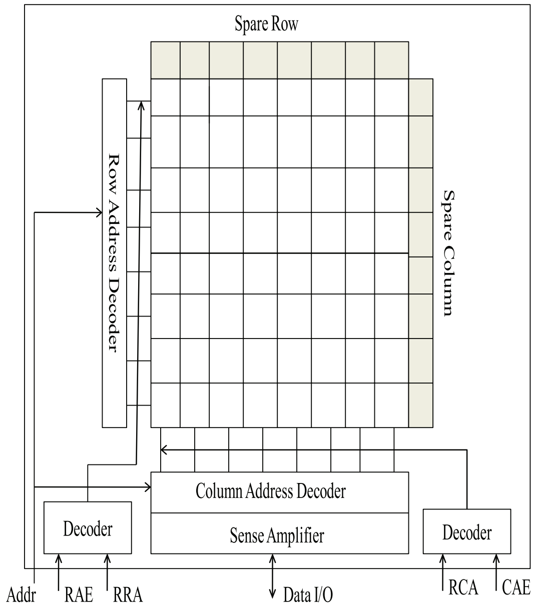


Fig. 2 An 8 * 8 bit oriented repairable RAM

A. Memory Bist

MBIST is used to test the on chip memories, where the on chip memories can be RAM, SRAM, DRAM and flash memory. The block diagram of the proposed MBIST is shown in fig. 3.TPG is used to generate the test sequences which are applied to RAM during testing. Comparator is used to compare the output responses with the expected responses and decision is made whether the RAM is faulty or fault free. A controller is a hardware realization of a memory test algorithm, usually in the form of an FSM. The memory test algorithm here is writing0/writing1algorithm.

This algorithm involves successive writing and reading of 0's and 1's into the RAM. The fault location details like fault present, fault row and fault column addresses are given at the output pins. The input pin start is made one when the BIST starts the testing process. The output pin done and progress are resulted to one and zero respectively, after the MBIST process is completed, so that the ReBIRA circuit can start analysis. The MBSIT FSM is shown in fig.4.

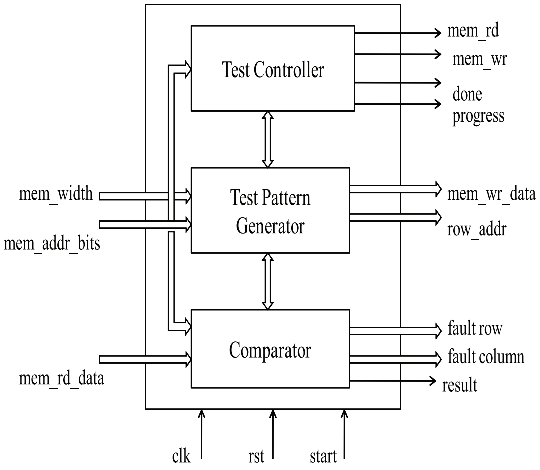


Fig. 3 Typical block diagram of proposed MBIST

State: Idle

When the input pin start is asserted to zero, the MBIST process is not yet started so it remains in the idle state. Else, the MBIST process enters into the next.

State: W1

In this state, all the memory locations are filled with 1's using the mem_wr_data when mem_wr signal is high, until the row address is equal to zero. When row address is equal to zero, the MBIST process enters into next state called R1state.

State: R1

In this state, when mem_rd is 1, all the memory locations are read through mem_rd_data until row address is equal to zero. When row address is equal to zero, the MBIST process enters into next state.

State: W0

In this state, when mem_wr is 1, all the memory locations are filled with 0's using mem_wr_data until row address is equal to zero. When row address is equal to zero, the MBIST process into next state.

State: R1

In this state, when mem_rd is 1, all the memory locations are read through mem_rd_data until row address is equal to zero. When row address is equal to zero, the MBIST process into next state called done.

State: Done

At this stage, the MBIST circuit completes the testing of the memory and it again enters into the idle state.

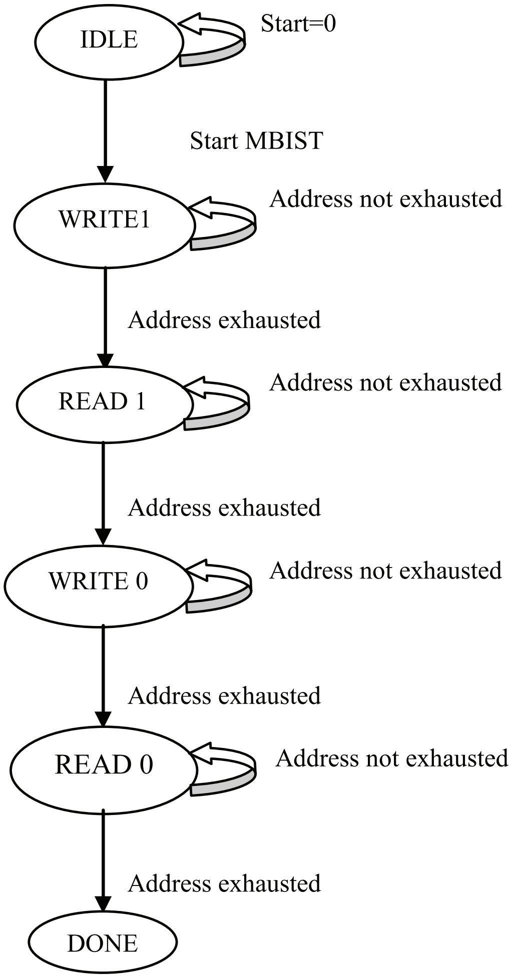


Fig. 4 MBIST FSM

1. Proposed Memory ReBIRA

If separate BIRA circuit is used for repairing individual RAM in an SOC, then the total area in the SOC is large, since there are many RAMs in a SOC. In the proposed BISR scheme single BIRA circuit is used for repairing many RAMs. Hence the area cost is reduced.

1.1 Architecture of The ReBIRA

Fig. 5 shows the simplified block diagram of ReBIRA used for repairing multiple RAMs in an SOC. If the BIST detects a fault, then the fault information is exported to the ReBIRA circuitry, and then the ReBIRA performs redundancy allocation based on the redundancy algorithm. The redundancy algorithm used here is Fault Checking Algorithm. Fault information includes fault row and fault column addresses and the fault_present. The proposed scheme uses a local bitmap (fault_table) of size 4*64 to store fault information detected by the BIST circuit. Once the local bitmap is full, the MBIST is paused and the ReBIRA allocates redundancies according to the fault information. After, the ReBIRA allocates a redundancy to repair a corresponding faulty row/column, the local bitmap is updated and the MBIST is resumed. This process continues until the test and repair process is completed. The repair information i.e. repair_register_write,repair_register_data,repair_register_address is sent to the repair register in the repairable RAM once one spare element is allocated. Fig.6. shows the simplified block diagram of repairable RAM. After the ReBIRA circuit allocates redundancies, the repair signatures rep_reg_wr, rep_reg_data and rep_reg_addr are sent to the repair registers present in the repairable RAM. Then, the RAM is repaired using this information as follows.

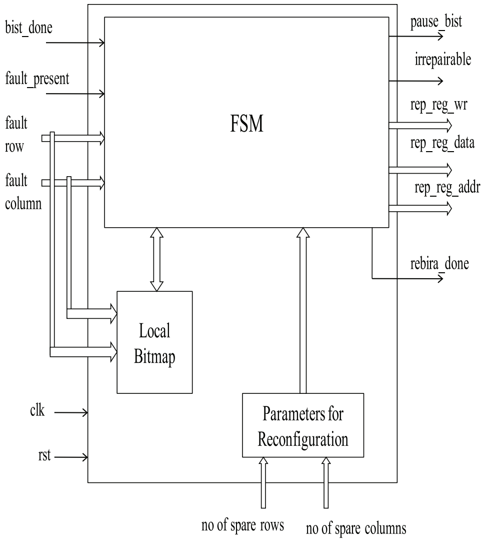


Fig. 5 Simplified block diagram of ReBIRA

When, either Clk signal or rst signal is high, the row repair registers and the column repair registers are filled to zeroes. When rep_reg_wr signal is high, if rep_reg_addr is less than the number of spare rows, then the rep_reg_data (defective row addresses) is written into the row_repair_address_reg, else if rep_reg_addr is less than the number of sum of spare rows and spare columns, the rep_reg_data is written into the column_repair_address_reg.

1.2 Redundancy Analysis Algorithm

In this section the proposed redundancy algorithm known as Fault-checking algorithm is described. The Fault-Checking algorithm can be used for RAM with different redundancy organization.Fig.6 shows RAM with local spare column and global spare row. In order to carry out the redundancy analysis a local bitmap of size 4X64 is used which is shown in fig.7. The local bitmap is updated with the faulty information detected by BIST circuit. Once the local bit map is full, the Fault-Checking algorithm does the analysis based on the faulty informationthe local bitmap.

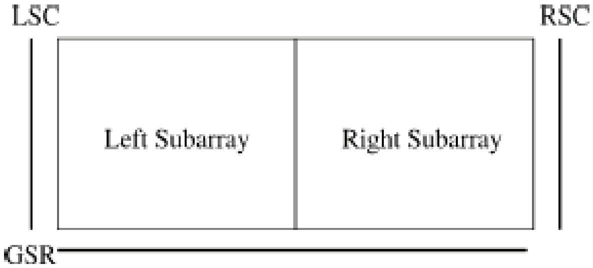
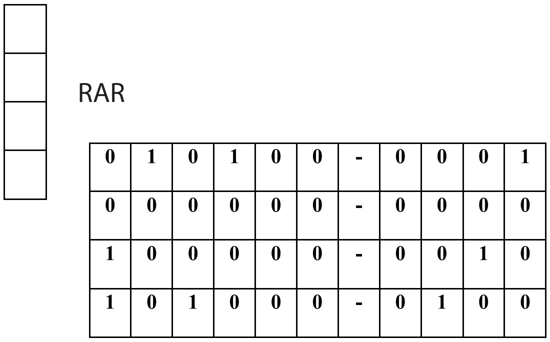


Fig.6. RAM with one global spare row and two local spare columns



0	1	0	1	0	0	-	0	0	0	1
0	0	0	0	0	0	-	0	0	0	0
1	0	0	0	0	0	-	0	0	1	0
1	0	1	0	0	0	-	0	1	0	0

Fig. 7 A 4x64 Local Bitmap

A row address register is used for each row in order to store the faulty row address. Here 64 columns are used. Since there are 64 columns in the memory and 64 columns in the local bitmap, there is a one to one mapping between memory and local bitmap. The Fault-checking algorithm checks the number of rows and columns with fault in the local bitmap. If the number of rows with faults (NRF) is more than the number of columns with faults (NCF), then a defective column with maximum number of faults (CMF) is replaced with a spare column. If the number of columns with faults (NCF) is more than the number of rows with faults (NRF), then a defective row with maximum number of faults (RMF) is replaced with a spare row. This process continues till all the faults in the local bitmap are repaired.

A Fault-Checking Algorithm for allocating spare elements

1. Start BIST; stop and go to Step 2 when fault is detected.
2. If the fault is repaired jump to step1, else go to Step 3.
3. If the local bitmap is full, go to the next step, else, go to Step 1.
4. If number of faulty rows is greater than the number of faulty column, replace the maximum faulty column with a spare column. Similarly if number of faulty columns is greater than the number of faulty rows, replace the maximum faulty row with a spare row.
5. Check if the BIST is completed. If so, go to step 6. Otherwise, go to Step 1 when the local bitmap is not full; go to Step 4 when the local bitmap is full.
6. If the local bitmap is empty export the repaired addresses and then stop. Otherwise, go to Step 4.

1.3 Design Of ReBISR Scheme

Fig.8 shows simplified block diagram of the proposed ReBISR scheme for repairing multiple RAMs in an SOC. RAM details table is used for storing the configurations of RAMs which includes the memory data width and depth, number of spare rows and number of spare columns. FSM is the main block that acts as a BIST controller for generating the control signals during testing and repairing. The BIST Controller controls the BIST and ReBIRA circuit. The BIST controller reads the RAM detail table and sends the RAM configurations like number of rows and number of columns to BIST and number of spare rows and number of spare columns to ReBIRA circuit. Program_ram_details and ram_details signals are used to make entries into the RAM detail table using the write pointer. Once the RAM detail table is full, the BIST and ReBIRA circuits start the testing and repairing process of RAMs one after other. If the BIST circuit detects a fault, then the fault information is exported to the ReBIRA circuitry, and then the ReBIRA performs redundancy allocation. After the ReBIRA allocates a redundancy to repair a corresponding faulty row or column, the local bitmap is updated and the BIST is resumed. This process continues until the test and repair process is finished. The repair signatures from the ReBIRA circuit are then sent to the repair registers present in the repairable RAMs. The BIST tests the RAMs once again (after the testing and repairing processes) to ensure that there are no faults present.Fig.9 shows the ReBISR FSM.

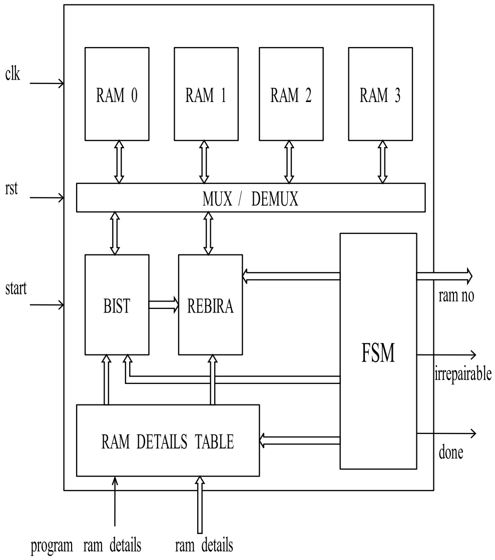


Fig. 8 Simplified Block diagram of the proposed ReBISR scheme for repairing multiple RAMs

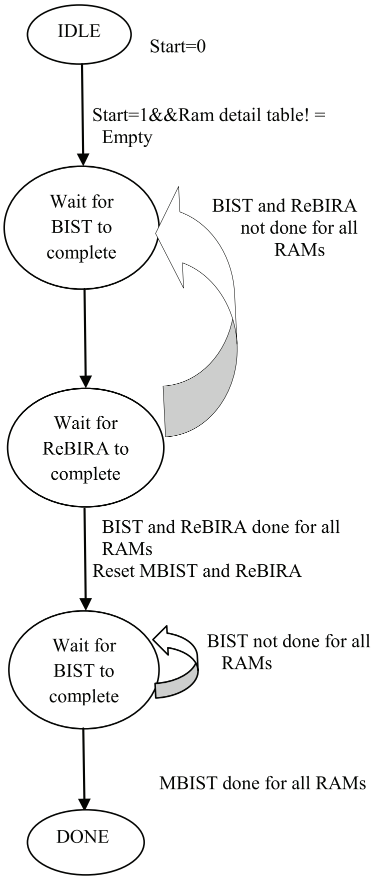

IDLE
Start=0
Start=1&&Ram detail table! = Empty
Wait for BIST to complete
BIST and ReBIRA not done for all RAMs
Wait for ReBIRA to complete
BIST and ReBIRA done for all RAMs
Reset MBIST and ReBIRA
Wait for BIST to complete
BIST not done for all RAMs
MBIST done for all RAMs
DONE

Fig. 9 ReBISR FSM

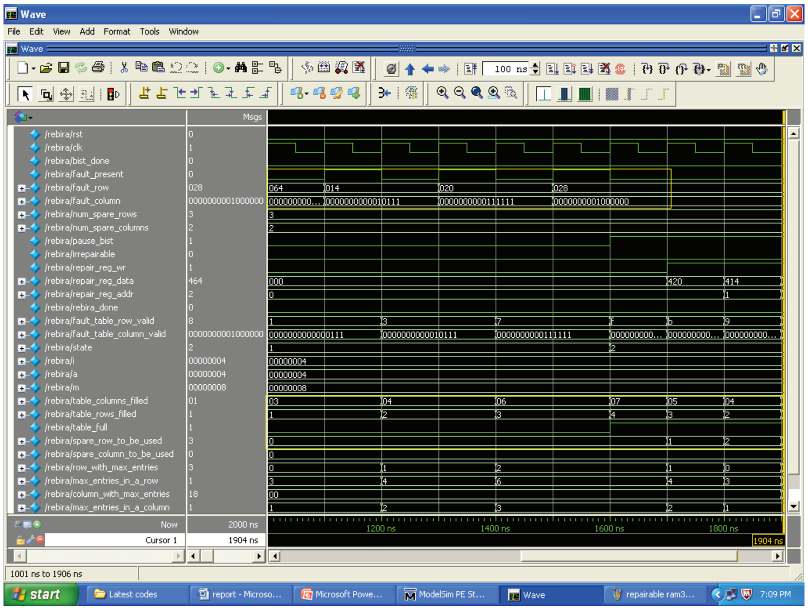
Table I Example of Four RAMs with Different Configuration

RAM No.	RAM Size (Row X Column)	Redundancies (r,c)
RAM0	16X32	(2,2)
RAM1	32X64	(2,3)
RAM2	128X64	(3,2)
RAM3	256X64	(0,0)

Table I shows four RAMs with various sizes and different redundancy configurations, where (Row X Column) denotes the row address width and column address width; (r, c) denotes the numbers of spare rows and spare columns. Therefore, design parameters of the ReBIRA for these four RAMs as shown in Table I are (Row X Column) = (256 X 64) and (r,c) = (3,3).The proposed ReBISR scheme tests and repairs each RAM serially.

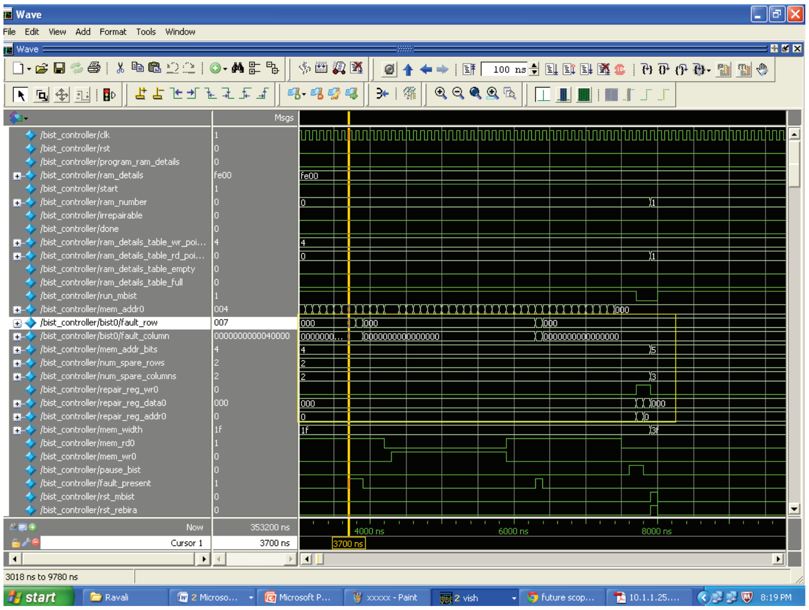
III. Experimental Results

Four RAMs with different number of redundancy configurations are simulated. The ReBISR and Dedicated BISR schemes have been synthesized and found the area cost and time cost. Table 2 summarizes simulation and comparison results of the ReBISR scheme and the dedicated BISR (DeBISR) scheme.



```
/rebira/rst                          0
/rebira/clk                          1
/rebira/bist_done                    0
/rebira/fault_present                0
/rebira/fault_row                    028
/rebira/fault_column                 0000000001000000
/rebira/num_spare_rows               3
/rebira/num_spare_columns            2
/rebira/pause_bist                   1
/rebira/irrepairable                 0
/rebira/repair_reg_wr                1
/rebira/repair_reg_data              464
/rebira/repair_reg_addr              2
/rebira/rebira_done                  0
/rebira/fault_table_row_valid        8
/rebira/fault_table_column_valid     0000000001000000
/rebira/state                        2
/rebira/i                            00000004
/rebira/a                            00000004
/rebira/m                            00000008
/rebira/table_columns_filled         01
/rebira/table_rows_filled            1
/rebira/table_full                   1
/rebira/spare_row_to_be_used         3
/rebira/spare_column_to_be_used      0
/rebira/row_with_max_entries         3
/rebira/max_entries_in_a_row         1
/rebira/column_with_max_entries      18
/rebira/max_entries_in_a_column      1
Now                                  2000 ns
Cursor 1                             1904 ns
1001 ns to 1906 ns
```

Fig.10 Output waveforms of ReBIRA.



```
/bist_controller/clk                         1
/bist_controller/rst                         0
/bist_controller/program_ram_details         0
/bist_controller/ram_details                 fe00
/bist_controller/start                       1
/bist_controller/ram_number                  0
/bist_controller/irrepairable                0
/bist_controller/done                        0
/bist_controller/ram_details_table_wr_poi... 4
/bist_controller/ram_details_table_rd_poi... 0
/bist_controller/ram_details_table_empty     0
/bist_controller/ram_details_table_full      0
/bist_controller/run_mbist                   1
/bist_controller/mem_addr0                   004
/bist_controller/bist0/fault_row             007
/bist_controller/bist0/fault_column          0000000000040000
/bist_controller/mem_addr_bits               4
/bist_controller/num_spare_rows              2
/bist_controller/num_spare_columns           2
/bist_controller/repair_reg_wr0              0
/bist_controller/repair_reg_data0            000
/bist_controller/repair_reg_addr0            0
/bist_controller/mem_width                   1f
/bist_controller/mem_rd0                     1
/bist_controller/mem_wr0                     0
/bist_controller/pause_bist                  0
/bist_controller/fault_present               1
/bist_controller/rst_mbist                   0
/bist_controller/rst_rebira                  0
Now                                          353200 ns
Cursor 1                                     3700 ns
3018 ns to 9780 ns
```

Fig.11 Output waveform of BISR.

Fig.10 shows the output waveform of ReBIRA module. The ReBIRA waveform shows bit 1 as fault information and bit 0 as fault free information. Address of the faulty row and faulty column are also shown.

Fig.11 shows output waveform of BISR module. If the BIST circuit detects a fault, the corresponding bit of fault_present is made high and the fault information i.e. address is exported to the ReBIRA circuit and the BIST is paused simultaneously.

Table II Simulation and Comparison Results of The Rebisr Scheme and Dedicated BISR Scheme

BIRA Scheme	RAM Type	BIRA Architecture	No. of slices used	Area Cost	Time Cost	Clock Cycles	% of Area Used
DeBIRA	WOM	Dedicated	7068	High	Short	2,06,700	81%
ReBIRA	WOM	Shared	2873	Low	Long	3,53,200	33%

IV. Conclusion

We have proposed a Reconfigurable BISR scheme to repair multiple RAMs with different depths and redundancy configurations. BIRA algorithm for redundancy analysis has been presented. Simulation results show that results the ReBISR scheme incurs low area cost when compared with the dedicated BISR. Also, the reconfigurable BISR scheme has greater flexibility than the dedicated BISR scheme as the former one supports the repair of multiple memories. Therefore, our ReBISR scheme has low area cost compared with the other BISR scheme for general applications.

References

[1] Tsu-Wei Tseng, Jin-Fu Li, Member, IEEE, and Chih-Chiang Hsu, “A Reconfigurable Built-In Self-Repair Scheme for Random Access Memories in SOCs”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol.18, No. 6, JUNE 2010.

[2] R. Rajsuman, “Design and test of large embedded memories: An overview,” *IEEE Des. Test Comput.* Vol.18, No.3, pp.16-27, May 2001.

[3] Y. Zorian, “Embedded memory test&repair: Infrastructure IP for SOC yield,” in *Proc. Int. Test Conf. (ITC),* Baltimore, MD, pp. 340-349, Oct. 2002.

[4] S. Nakahara, K. Higeta, M. Kohno, T. Kawamura, and K. Kakitani, “Built-in self-test for GHz embedded SRAMs using flexible pattern generator and new repair algorithm,” *in Proc. Int. Test Conf.* (ITC), pp. 301–310, 1999.

[5] Jin-Fu Li, “Memory Built-In Self-Repair”, Advanced Reliable Systems (ARES) Lab., Department of Electrical Engineering, National Central University, Jhongli, Taiwan.

[6] Joohwan Lee, Kihyun Park, and Sungho Kang, “An Area-efficient Built-in Redundancy Analysis for Embedded Memories with Optimal Repair Rate using 2-D Redundancy” Dept. of Electrical & Electronic Engineering, Yonsei University ,Seoul, Korea.

[7] S. K. Thakur, R. A. Parekhji, and A. N. Chandorkar, “On-chip test and repair of memories for static and dynamic faults,” in Proc. Int. Test Conf. (ITC), Santa Clara, CA, pp. 1-10, Oct. 2006.

[8] S.Y. Kuo and W.K.Fuchs, “Efficient spare allocation in reconfigurable arrays,” *IEEE Des. Test Comput.,* Vol.4, No.1, pp.24-31, Feb.1987.

[9] D. Xiaogang, S. M. Reddy, W.T. Cheng, J. Rayhawk, and N. Mukherjee, “At-speed built-in self-repair analyzer for embedded word-oriented memories,” *in Proc. Int. Conf. VLSI Des.,* pp.895–900, 2004.

[10] D. K. Bhavsar, “An algorithm for row-column self-repair of RAMs and its implementation in the Alpha 21264,” *in Proc. Int. Test Conf.* (ITC), Atlantic City, NJ, pp. 311–318, Sep. 1999.

[11] T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada, and H. Hidaka, “A built-in self-repair analyzer (CRESTA) for embedded DRAMs,” in *Proc. Int. Test Conf.* (ITC), pp. 567-574, 2000.

[12] J.-F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu, “A built-in self-repair design for RAMs with 2-D redundancies,” *IEEE Trans. Very Large Scale Integr.* (VLSI) Syst., Vol.13, No.6, pp.742-745, Jun. 2005.

[13] P. Ohler, S. Hellebrand, and H.-J. Wunderlich, “An integrated built-in self-test and repair approach for memories with 2D redundancy,” in Proc. *IEEE Eur. Test Symp.* (ETS), Freiburg, pp.91-99, May 2007.

[14] C.-D. Huang, J.-F. Li and T.-W. Tseng, “ProTaR: An infrastructure IP for repairing RAMs in SOCs,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* Vol.15, No.10, pp.1135-1143, Oct. 2007.

[15] R.-F. Huang, J.-F. Li, J.-C. Yeh, and C.-W. Wu, “Rainsin: Redundancy analysis algorithm simulation,” *IEEE Des. Test Comput.,* Vol.24, No.4, pp.386-396, Jul- Aug. 2007.