International conference on Advanced Techniques in Communication Networking and Automation

RESEARCH ARTICLE OPEN ACCESS

A BUILT-IN SELF REPAIR ANALYZER FOR WORD-ORIENTED MEMORIES

Kowsalya.S[1], Somu.K[2], Saravanakumar.P[3]

PG Student, VLSI Design, Department of ECE, Maha Barathi Engineering College, India[1]

(Kowsalyakowsalya54807@gmail.com)

Professor, Department of ECE, Maha Barathi Engineering College, India[2]

(somu.pgp@gmail.com)

Assistant Professor, Department of ECE, Maha Barathi Engineering College,India[3]

(yasaraa@gmail.com)

ABSTRACT

In this project a built-in self repair analyzer with the optimal repair rate for memory arrays with redundancy. The existing techniques used depth first search using a stack and a finite-state machine. The formulation of the circuit allows using the parallel prefix algorithm, it can be configured in various ways to meet area and test time requirements.. The content addressable memory is used to identify the fault address to store in must-repair analyzer. The proposed methods using the bit swapping linear feedback shift register to reduce both the transition and the power consumption, and also using precomputation content addressable memory to reduce the time delay in the must-repair analyzer. It requires only a single test, even in the worst case. By performing the must-repair analysis during the test, it selectively stores fault addresses, and the final analysis to find a solution is performed on the stored fault addresses. The infrastructure is also extended to support various types of word-oriented memories

Keywords:

depth first search, finite-state machine, BIST

1.INTRODUCTION

Today's System-On-Chip (SOC) environment requires significant changes in testing methodologies for memory arrays. The failure of embedded memories in a SOC is more expensive than that of commodity memories because a relatively large die is wasted. Due to the large die size and the complex fabrication process for combining memories and logic, SOC suffer from relatively lower yield, necessitating yield optimization techniques. At present, the area occupied by the embedded memories takes more than half of the total area of atypical SOC, and the ratio is expected to keep increasing in the future. In addition, the aggressive design rules make the memory arrays prone to defects. A SOC consists of both the hardware and software. The software controlling the microcontroller, microprocessor or DSP cores, peripherals and interfaces. The design flow for a SOC aims to develop this hardware and software in parallel. Therefore, the overall SOC yield is dominated by the memory yield, and optimizing the memory yield plays a crucial role in the SOC environment.

To improve the yield, memory arrays are usually equipped with spare elements, and external testers have been used to test the memory arrays and configure the spare elements. However, in the SOC environment, the overall test time is prohibitively increased if the test response data from the memory arrays are sent to the external testers. On the other

hand, the SOC environment, combined with shrinking technology, allows us more area for on-chip test infrastructure at lower cost than before, which makes feasible a variety of built-in self test (BIST) and built-in self-repair (BISR) techniques for reducing the test time, and also it is to test the memory arrays.

1.1 SYSTEM-ON-CHIP

A system on a chip or system on chip (SOC) is an integrated circuit (IC) that integrates all components of a computer or other electronicsystem into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functionsall on a single chip substrate. A typical application is in the area of embedded systems.

The contrast with a microcontroller is one of degree. Microcontrollers typically have under 100 KB of RAM (Random Accessable Memory) (often just a few kilobytes) and often really are single-chip-systems, whereas the term SOC is typically used for more powerful processors, capable of running software such as the desktop versions of Windows and Linux, which need external memory chips (flash, RAM) to be useful, and which are used with various external peripherals.

In short, for larger systems, the term system on a chip is a hyperbole, indicating technical direction more than reality: increasing chip integration to reduce manufacturing costs and to enable smaller systems. Many interesting systems are too complex to fit on just one chip built with a process optimized for just one of the system's tasks. Therefore, the overall SOC yield is dominated by the memory yield, and optimizing the memory yield plays a crucial role in the SOC environment. The defects are thus likely to affect the functionality of the memory arrays rather than that of logic.

When it is not feasible to construct a SOC for a particular application, an alternative is a system in package (SiP) comprising a number of chips in a single package. In large volumes, SOC is believed to

be more cost-effective than SiP since it increases.

2.AN ADVANCED BIRA FOR MEMORIES WITH AN OPTIMAL REPAIR RATE AND FAST ANALYSIS SPEED BY USING A BRANCH ANALYZER

As memory capacity and density grow, a corresponding increase in the number of defects decreases the yield and quality of embedded memories for systems-on-chip as well as commodity memories. For embedded memories, built-in redundancy analysis (BIRA) is widely used to solve quality and yield issues by replacing faulty cells with healthy redundant cells.Many BIRA approaches require extra hardware overhead in order to achieve optimal repair rates, or they suffer a loss of repair rate in minimizing the hardware overhead.An innovative BIRA approach is proposed to achieve optimal repair rates, lower area overhead, and increase analysis speed. The proposed BIRA minimizes area overhead by eliminating some storage coverage for only must-repair faulty information. The defective cells detected by the BIST circuit are replaced by the cells of the spare SRAM.

In the case of embedded memories for systems-on-a-chip (SOC), built-in redundancy analysis (BIRA) is widely used as a solution to solve quality and yield issues by replacing faulty cells with extra good cells. The proposed BIRA analyzes redundancies quickly and efficiently by evaluating all nodes of a branch in parallel with a new analyzer which is simple and easy-to-implement. Experimental results show that the proposed BIRA allows for a much faster analysis speed than that of the state-of-the-art BIRA, as well as the optimal repair rate.

3.MUST REPAIR ANALYZER

The Must Repair Analyzer(MRA)circuit diagram is as shown in Fig.3.1. It consists of a pair ofCAMs for fault addresses, called the fault-list, and a pair ofCAMs for a repair solution, called the solution record. The memory is repaired during testing by storing faulty addresses in registers. These addresses can be streamed out after test

completion. Furthermore, the application can be started immediately after the memory BIST passes.In the fault-list, each CAM has one extra valid bit for eachword, and the valid bits are initialized to “0” in the beginning.Since the CAMs assert “1” at the valid bit position for write andmatch operation, only written entries can be matched. Duringthe test, if the BIST engine detects a fault, it sends the faultaddress to the MRA on the fly through BIST_R_DUTAddr andBIST_C_DUTAddr, continues the test. The row (column)fault address is compared against row (column) CAM entries.

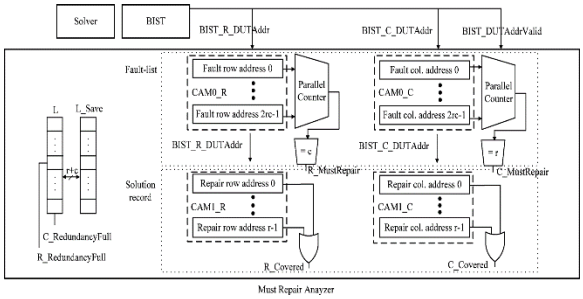

Fig3.1 Must Repair Analyzer

And the number of matched entries is efficiently counted by a parallel counter. If the number of the matched entriesequals in the row (column) CAM, the row (column) indicatedby the fault address satisfies the must-repair condition andR_MustRepair (C_MustRepair) signal is asserted. If the faultaddress triggers neither the row nor column must-repair condition,MRA writes the row and column address in the row andcolumn CAMs, respectively. Due to Corollary 2, we can limitthe size of the fault-list as 2, and if the overflow of the fault-listoccurs, the memory array can be determined as unrepeatable,and the test can be terminated early.If a particular row or column is identified as must-repair, therow or column address must be part of the solution. Thus theMRA writes the row or column address in the solution record.The L registers are used as valid bits for the solution record in fig 3.1.

Since a must-repairrow and a must-repair column can be identified by a faultat the same time, the MRA should be able to write a pair of rowand column addresses simultaneously. Once a row or columnaddress is stored as part of solution by the must-repair condition,then all solution candidates considered by the SOLVER include the address, and faults on the address do not affect the final analysis any more. Therefore, such faults do not need to be stored, and we can collect all necessary information for the final analysis during a single test. Once the test is completed (thus the must-repair analysis is done), BIST_Done signal is asserted and the final analysis is started. In the final analysis, the SOLVER module controls the MRA.

3.1SOLVER

The SOLVER has a register to store the cost of the current repair strategy, orUsed Repair Elements. The SOLVER has a register to store the cost of the current repair strategy, or Used Repair Elements. The SOLVER also has registers to store the repair strategy with minimum cost so far and the minimum cost, or Repair Strategy Opt and Used Repair ElOpt.The current cost is compared against the minimum cost so far, which generates the Better signal. If the Better signal goes down duringthe evaluation of the current repair strategy, the SOLVER immediatelyasserts the RESTART signal and moves on to the nextrepair strategy. If the Better signal stays at “1” until the end of the evaluation, the SOLVER saves the current repair strategy and its cost. Since the SOLVER continues to search for a better solution even after finding a solution, the MRA may not have the optimal solution after the last repair strategy is evaluated. The Must-Repair Analysis repairs rows and columns for which there is no other choice.

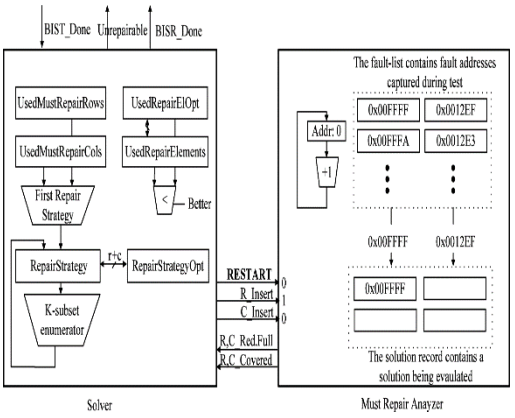

Fig 3.2 Solver Circuit

To reduce area, we have stored the optimal repair strategy instead of the optimal solution. If the solution is directly stored, the size of the solution record should be doubled. Thus we need to recover the solution from the repair strategy, and the SOLVER goes into recovery phase. Since the SOLVER continues to search for a better solution even after finding a solution, the MRA may not have the optimal solution after the last repair strategy is evaluated. The solver and MRA circuit operation on shown in Fig 3.2, the MRA gives the fault address to solver. In the recovery phase, the optimal repair strategy, stored in the Repair Strategy Opt, goes into the Repair Strategy register and the strategy is evaluated again and the final analysis ends up with the optimal solution. The Final Analysis is a branch and bound search in the space of partial solutions, each node records the spare row and column assignments.

4.RESULT AND SIMULATION OUTPUT

4.1SIMULATION OUTPUT

4.1.1 Error Detection

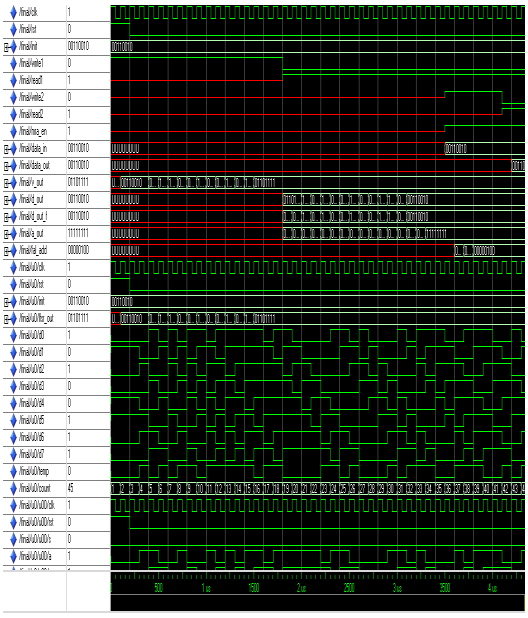

Fig 4.1 Simulation Output for Error Detection

The MODELSIM simulation output of error detection as shown in Fig.5.1. First the input data to write give write-‘1’ to enable. After to read the input data give read-‘1’to enable and write-‘0’to disable. In the three data the last bit is error occurs, the fault data bits are shown in d-out-f and the input data bits are shown in d-out.

4.1.2 Error Correction

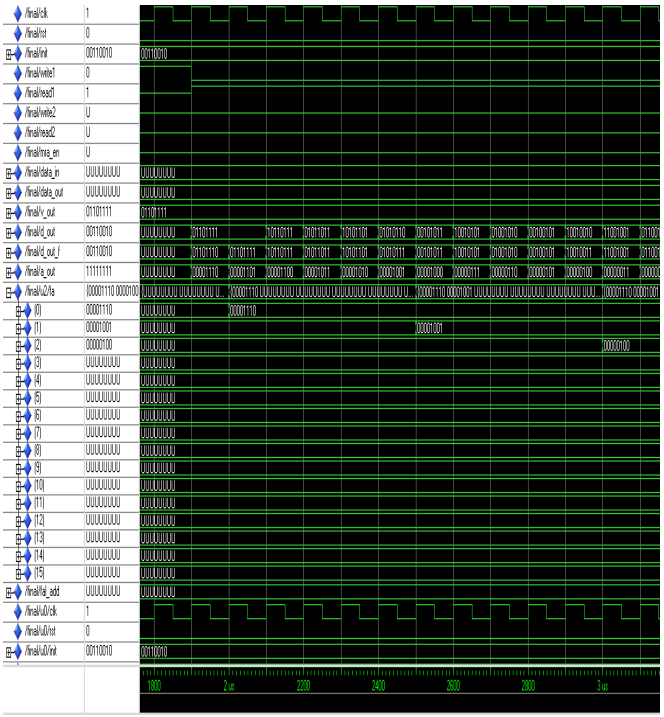

Fig 4.2 Simulation Output for Error Correction

The MODELSIM simulation output of error correction as shown in Fig.5.2. The fault data addresses are store in fault/u2/fa. Using the spare wires to correct the error bits and also again to read and write the data using give to write2-‘0’ and read2-‘1’. Finally, to check the error correction output data bits shown in d-out.

4.2 POWER CONSUMPTION

4.2.1 Existing Power

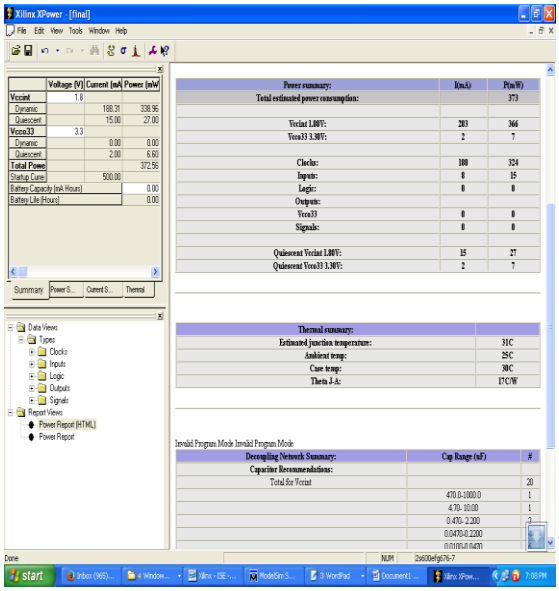

Fig 5.3 Existing Power Consumption

The Xilinx Power Summary of Existing and Proposed System is as shown in Fig 5.3 and 5.4. Here before using the linear feedback shift register required high transition and 373mW power. By using Bit swapping –LFSR require low transition and also require only 113mW power.

5.2.2 Proposed Power

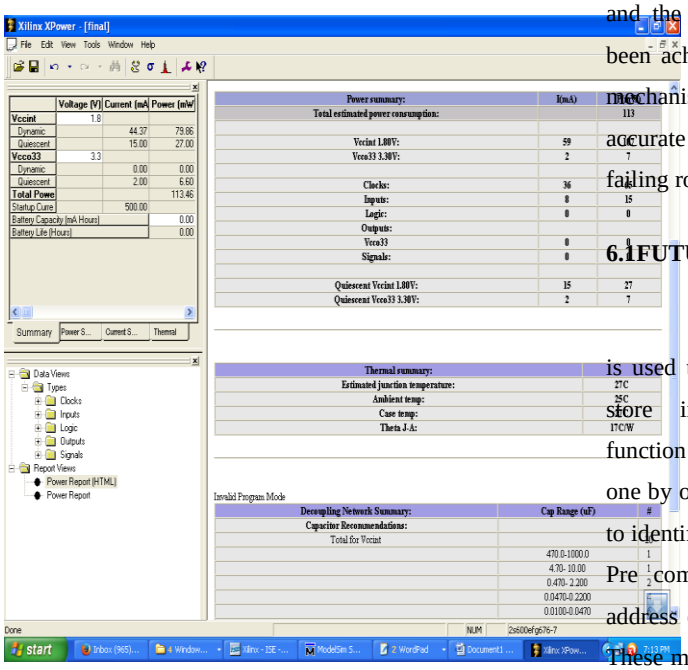

Fig 5.4 Proposed Power Consumption

5.3 COMPARISON FOR OUTPUT POWER

Table 5.1 Comparison Tablefor Power Consumption

S.NO	PROCESS	POWER CONSUMPTION (mW)
1.	Existing	373
2.	Proposed	113

CONCLUSION

In proposed an on-chip infrastructure for repair analysis with the optimal repair rate for word oriented memories.As part of repair analyzer, and also developed a novel combinatorial circuit formulation allows to use the parallel prefix algorithm; it can be configured in various ways to meet area and test time requirements. The built in self repair analyzer used to detect and correct the errors in word oriented memories with single test. The linear feedback shift register used to count the next state and also requires high transitions in BIST architecture, required 373mW power. This system proposed methods using the bit swapping linear feedback shift register to reduce both the transition and the power consumption as 113mW. This has been achieved by using low-cost on-chip selection mechanisms, which are instrumental in very accurate and power reduction identification of failing rows, columns, and word oriented memories.

6.1FUTURE WORK

The content addressable memory (CAM) is used to identify the fault address and it can be store in Must-Repair-Analyzer (MRA). The function of CAM is to search the fault address in one by one in MRA; it requires high searching time to identify the fault address. In future work by using Pre computational-CAM is to identify the fault address easily, and also reduce the searching time. These modifications are implementing to the FPGA kit.

APPENDIX

```
library ieee;
use ieee.std_logic_1164.all;
entity final is
   port(     clk,rst  : in std_logic;
          init   : in std_logic_vector(7 downto 0);
         Write1   : in std_logic;
          Read1   : in std_logic;
         Write2   : in std_logic;
          Read2   : in std_logic;
         mra_en   : in std_logic;
         Data_in  : in std_logic_vector(7 downto 0);
                  Data_out     :  out  std_logic_vector(7
downto 0);
         v_out    : inout std_logic_vector(7 downto 0);
         D_out    : inout std_logic_vector(7 downto 0)

    );
     end final;

architecture str of final is
component bs_lfsr is
  port( clk,rst  : in std_logic;
        init    : in std_logic_vector(7 downto 0);
       lfsr_out  : out std_logic_vector(7 downto 0));
end component;

component memory is

port(       Clock : in std_logic;
     Write : in std_logic;
     Read  :  in std_logic;
              Data_in :              in   std_logic_vector(7
downto 0);
              Data_out:              out std_logic_vector(7
downto 0);
             Data_out_f:      out      std_logic_vector(7
downto 0);
              add_out  : out std_logic_vector(7 downto
0)

);
end component;

component mra is
port
  ( clk     : in std_logic;
    ad      : in std_logic_vector(7 downto 0);
    f,f1    : in std_logic_vector(7 downto 0);
    mra_en   : in std_logic;
    fa_add   : out std_logic_vector(7 downto 0)


   );
end component;


component solver is

port(       Clock  :              in std_logic;
     Write  :                     in std_logic;
     Read   :  in std_logic;
              f_add  :             in   std_logic_vector(7
downto 0);
              Data_in :            in   std_logic_vector(7
downto 0);
             Data_out :            out std_logic_vector(7
downto 0)

);
end component;

signal      D_out_f  : std_logic_vector(7 downto 0);
signal a_out    : std_logic_vector(7 downto 0);
signal fal_add   : std_logic_vector(7 downto 0);

begin

u0:bs_lfsr port map(clk,rst,init,v_out);
u1:memory                                    port
map(clk,Write1,read1,v_out,D_out,D_out_f,a_out);
u2:mra                                       port
map(clk,a_out,D_out,D_out_f,mra_en,fal_add);
uu3:solver                                   port
map(clk,Write2,read2,fal_add,Data_in,Data_out);


end str;
```

REFERENCES

1. W. Jeong, J. Lee, T. Han, K. Lee, and S. Kang, "An advanced BIRA for memories with an optimal repair rate and fast analysis speed by using a branch analyzer," IEEE Trans. Comput.-Aided Design Integr. CircuitsSyst., vol. 29, no. 12, pp. 2014–2026, Dec. 2016.
2. W. Jeong, I. Kang, K. Jin, and S. Kang, "A fast built-in redundancyanalysis for memorieswith optimal repair rate using a line-based searchtree," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no.12, pp. 1665–1678, Dec. 2015.
3. P. Oehler, S. Hellebrand, and H.-J. Wunderlich, "Analyzing test and repair times for 2D integrated memory built-in test and repair," in Proc.Design Diag. Electron. Circuits Syst., 2014, pp. 1–6.
4. C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancyanalysis formemory yield improvement," IEEE Trans. Reliab., vol. 52,no. 4, pp. 386–399, Dec. 2013.
5. S.-Y. Kuo andW. Fuchs, "Efficient spare allocation for learrays," IEEE Design Test Comput., vol. 4, no. 1, pp. 24–31, Feb. 1987.C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancyanalysis formemory yield improvement," IEEE Trans. Reliab., vol. 52,no. 4, pp. 386–399, Dec. 2003.
6. X. Du and W.-T. Cheng, "At-speed built-in self-repair analyzer toembedded word-oriented memories," in Proc. Int. Conf. VLSI Design,2004, pp. 895–900.
7. B. Fitzgerald and E. Thoma, "Circuit implementation of fusible redundant addresses on RAMs for productivity enhancement," IBM J. Res.Develop., vol. 24, no. 3, pp. 291–298, 1980.