Configurable Spare Database Reduction for RAMs

Marek SPURNÝ[*]

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
spurny.marek@gmail.com

Abstract. Built-in self-repair for memories is widely used in industry to increase their yield. Recently some approaches were proposed which use flexible configurable spares. However, the reconfiguration control is more complex as with traditional spares because there are many spare configurations that can be used to repair faults and the repair algorithm has to keep the database of these configurations. This database is generally very large which is not feasible in order to keep the area overhead acceptable. In this paper a method for configurable spare database reduction is proposed along with its hardware implementation.

1 Introduction

Current semiconductor process technologies cause the fault density in RAMs to increase. In present day system on a chips (SoCs) memories are dominating the chip area therefore the overall SoC yield is impacted mostly by memory yield. To increase memory yield, built-in self repair (BISR) is widely used in industry. Spare memory cells are addressed instead of faulty ones. The basic BISR scheme is shown in Figure 1. The built-in self test (BIST) block tests main memory for faults. If a fault is found, its location is sent to the built-in repair analysis (BIRA) block which determines which spare will be used to replace the fault. The final repair solution for all faults is stored in the address reconfiguration (AR) block. When a faulty cell is being addressed, the address is translated into the spare memory by AR and the spare is addressed instead of the faulty cell.

Most of existing approaches [1, 3] use the traditional spare model with rows and columns. The entire row or column which contains faulty cell(s) is replaced by a spare row or column. An example memory with 2 spare rows and columns is shown in Figure 2 (a). Spare S1 was used to replace a faulty row and spare S3 was used to replace a faulty column in the memory.

Recently some approaches were proposed which use spare model with more flexible configurable spares [4]. A spare can be configured as a traditional row or column but also as other shapes such as a square or a rectangle. These various configurations are referred to as *configuration types*. An example memory with 4 spares is shown in Figure 2 (b). Both used spares S1 and S2 were configured as rectangles and replaced the faulty areas of the corresponding shapes in the memory.

[*] Master study programme in field: Software Engineering
Supervisor: Dr. Štefan Krištofík, Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies STU in Bratislava

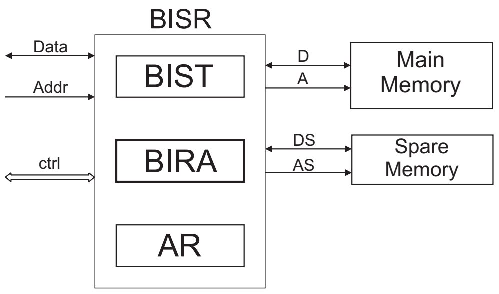

Figure 1. BISR for memories.

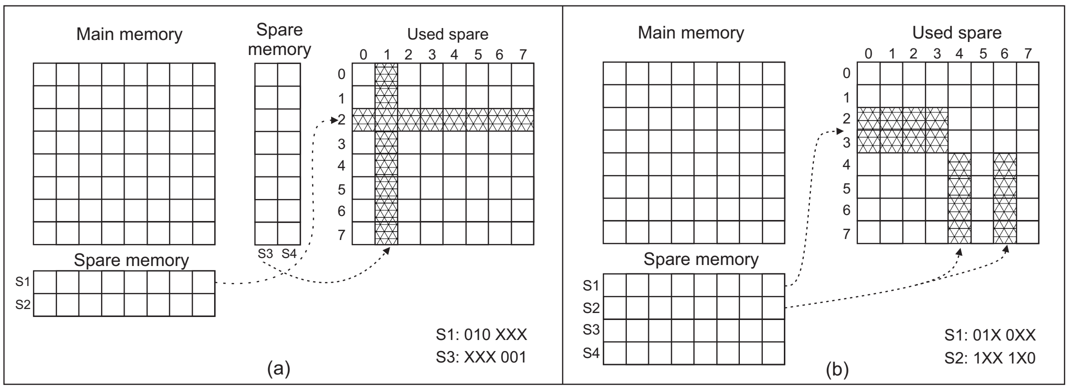

Figure 2. Spare architectures: (a) traditional, (b) configurable.

In Figure 2 the *address space* of each used spare is shown. This space contains the addresses of all faults it can possibly replace. For example, the spare row S1 in Figure 2 (a) can replace all faults in row 2 in the memory, therefore its address space is 010 XXX. Similarly, the address space of spare S2, which can replace all faults in column 1 is XXX 001. The address space of configurable spares can be obtained similarly. For example, the spare rectangle S1 in Figure 2 (b) can replace faults in rows 2 and 3 in the memory, but only in columns 0 through 3. Its address space is 01X 0XX.

Before the first fault is detected by BIST in the traditional spare model, it is clear that the fault could only be replaced by either row or column when it appears. However, in configurable spare model, the set of possible spare configurations that could replace the first fault can be large. The exact number of possible configurations is determined by the memory size.

In the configurable spare model, the repair algorithm has to keep a database of all configurations that can possibly repair the faults. The database is progressively reduced as more faults are detected [4]. This database can be initially very large (depending on the memory size) which can increase the BISR area overhead to unacceptable levels. To keep the area overhead feasible, a method for configurable spare database reduction is proposed along with its hardware implementation.

2 Configurable spare database reduction principles

In the configurable spare model as was shown in Figure 2 (b), before the first fault is detected by BIST any spare configuration could be used to replace it, because it is not known where the fault will occur. Therefore, if the database of spare configurations that could possibly replace the first fault (in the rest of the paper this will be referred to simply as 'database') is constructed before the first fault is detected, it has to contain all configurations. However, if the database is constructed after the first

Table 1. Database size.

		#areas	
		any	max. 2
#faults	0	1120	272
	1	70	17

fault is detected, its size can be narrowed down significantly.

In general, the database size can be reduced M times (M being the bit size of each spare) when it is constructed after the first fault is detected as opposed to when it is constructed before the first fault is detected. The reason behind this is that the incoming address of the first fault rules out the other $\frac{M-1}{M}$ configurations of each type which cannot be used to replace the fault. For example, suppose that the first fault detected in the memory in Figure 2 is (2,3). The row address of the fault is 2, therefore it can only be replaced by replacing row 2 with a spare row. Other $\frac{7}{8}$ row configurations are ruled out. Same principle is true for any spare configuration type (column, square, rectangle).

The database can be reduced further if only a subset of all configuration types is considered, i.e. only spare configurations with a set maximum number of areas. An *area* of a spare is a continuous block of spare cells. For example, spare S1 in Figure 2 (b) has 1 area and spare S2 has 2 areas.

The database reduction method proposed in this paper is based on the following principles:

1. The database is constructed after the first fault is detected.
2. Only spare configurations with a maximum of 2 areas are considered.

3 Implementation

The proposed configurable spare database reduction method was implemented for a small 1 kB memory (64x16 bits) with 8 16-bit spares. 5 spare configuration types are considered: row (1x16 bits), column (16x1 bits), square (4x4 bits), horizontal rectangle (2x8 bits) and vertical rectangle (8x2 bits). Table 1 shows the database sizes for this memory in various cases if using or not using principles 1 and 2 mentioned in Section 2. The initial size of the database when not using the principles is 1120. This means the first fault could be replaced by 1120 various spare configurations. Some reduction can be achieved by using only one of the principles, but if both principles are used, the database size can be reduced to 17 entries. This means there are only 17 spare configurations that could possibly repair the first detected fault.

Per principle 2, there are only spare configurations with a maximum of 2 areas in the reduced database. The configurations with 1 area include 1 row, 1 column, 1 square, 1 horizontal rectangle and 1 vertical rectangle configuration. The configurations with 1 area are referred to as *default*. To obtain the remaining configurations with 2 areas, the operations *RshiftN* and *CshiftN* are defined for the default square and both rectangle configuration types, as follows:

- RshiftN - the leftmost 'X' bit in the row address of the address space of the spare is shifted N positions to the left.
- CshiftN - the leftmost 'X' bit in the column address of the address space of the spare is shifted N positions to the left.

In both operations, the incoming fault address bit is inserted in the original bit position of the shifted 'X' bit. These operations are not defined for row and column spare configurations. An example of Rshift and Cshift operations for a square spare configuration (4x4 bits) is shown in Figure 3 and the following is true:

- `10XX 10XX` - address space of the default square configuration.
- `1011 1010` - address of the first detected fault (11,10).
- `1X1X 10XX` - address space produced by the Rshift1 operation.

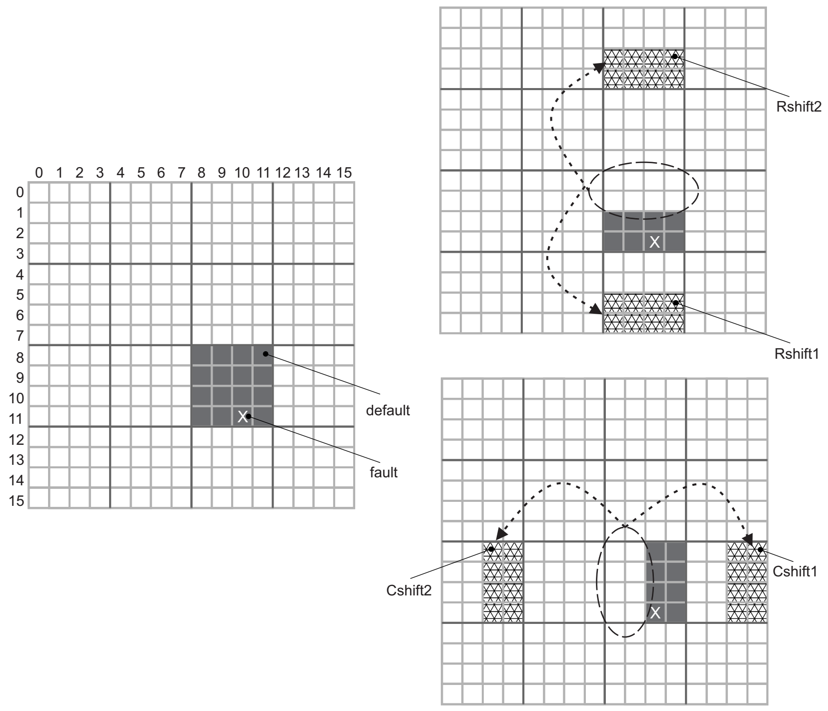

Figure 3. Rshift and Cshift operation examples for a 4x4 spare configuration.

- `X01X 10XX` - address space produced by the Rshift2 operation.
- `10XX 1X1X` - address space produced by the Cshift1 operation.
- `10XX X01X` - address space produced by the Cshift2 operation.

Similarly, the Rshifted and Cshifted address spaces are produced for the horizontal and vertical rectangle spare configurations and the reduced database is constructed. All entries of the reduced database are shown in Table 2. The database has 17 entries, comprising 1 row, 1 column, 5 square, 5 horizontal rectangle and 5 vertical rectangle configurations including all default, Rshifted and Cshifted entries. Note that the database is generalized for any fault, therefore the incoming fault row address bits are noted as $R_1, \ldots, R_4$ and the column address bits as $C_1, \ldots, C_4$.

4 BIRA hardware implementation

The first fault in the 1 kB memory described in Section 3 can be replaced by 17 spare configurations. The subsequent detected fault can further reduce the set of configurations that could possibly repair both faults. In the worst case, none of the 17 configurations could repair both faults. In this case the next spare has to be activated and used to repair the second fault. On the other hand, several faults may be detected that could still be repaired using the first spare.

To progressively rule out the various spare configurations that could no longer repair the incoming faults, for each spare one FGEN circuit shown in Figure 4 is used. The address of the first detected fault (A1) is compared with all subsequent addresses of detected faults (AN) by the CMP block. 17 comparisons are performed by the CMP block to check which of the 17 spare configurations in the database (according to Table 2) could still be used to repair the incoming faults. The flag bits (the first column in Table 2) are set to 0 by the MEM block for configurations that can no longer repair faults and to 1 for configurations that can still repair faults. The flag bits are updated on the rising edge of the *update* signal but only if the *det* signal is active. One AND gate is used for this purpose.

Table 2. Reduced database.

bit flag #	type	dim.	row address				column address			
1	default	1x16	R_1	R_2	R_3	R_4	X	X	X	X
2	default	16x1	X	X	X	X	C_1	C_2	C_3	C_4
3	default		R_1	R_2	X	X	C_1	C_2	X	X
4	Rshift1		R_1	X	R_3	X	C_1	C_2	X	X
5	Rshift2	4x4	X	R_2	R_3	X	C_1	C_2	X	X
6	Cshift1		R_1	R_2	X	X	C_1	X	C_3	X
7	Cshift2		R_1	R_2	X	X	X	C_2	C_3	X
8	default		R_1	R_2	R_3	X	C_1	X	X	X
9	Rshift1		R_1	R_2	X	R_4	C_1	X	X	X
10	Rshift2	2x8	R_1	X	R_3	R_4	C_1	X	X	X
11	Rshift3		X	R_2	R_3	R_4	C_1	X	X	X
12	Cshift1		R_1	R_2	R_3	X	X	C_2	X	X
13	default		R_1	X	X	X	C_1	C_2	C_3	X
14	Rshift1		X	R_2	X	X	C_1	C_2	C_3	X
15	Cshift1	8x2	R_1	X	X	X	C_1	C_2	X	C_4
16	Cshift2		R_1	X	X	X	C_1	X	C_3	C_4
17	Cshift3		R_1	X	X	X	X	C_2	C_3	C_4

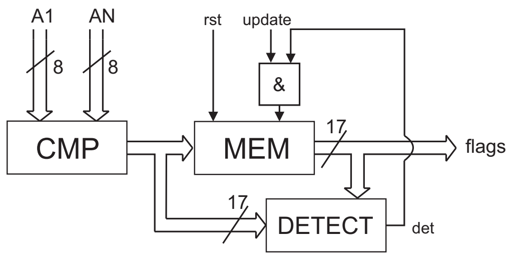

Figure 4. Flag bit generator - FGEN.

The DETECT block detects a possible situation when all flag bits would be set to 0 (indicating a new spare is needed) and prevents this from happening by setting the *det* signal to 0. This is important to keep the information on the flag bits active even when a new spare is needed.

The overall scheme of the proposed BIRA architecture for all 8 spares is shown in Figure 5. It contains one FGEN circuit and one A1GEN circuit for each spare and the BIRA CONTROL block. FGEN circuit was already described. A1GEN circuit is used to distinguish between the first fault and the other faults and generates the first fault address A1 for the corresponding FGEN circuit. A1 is generated on the rising edge of the *update* signal but only if the corresponding *ff* signal is active. The *ff* signals indicate that the incoming fault is the first for the corresponding A1GEN circuit. The BIRA CONTROL block controls the whole process by generating the appropriate values of *rst, update (upd.)* and *ff* signals. Incoming fault addresses from BIST are sent to BIRA through the 8-bit *A* signal.

After the repair process is finished, the final repair solution is indicated on the flag bits of FGEN circuits.

5 Conclusions and future work

The proposed BIRA architecture is based on the CSA algorithm [4] and utilizes a new method for database reduction. It is a part of the BISR architecture which contains a BIST block capable to

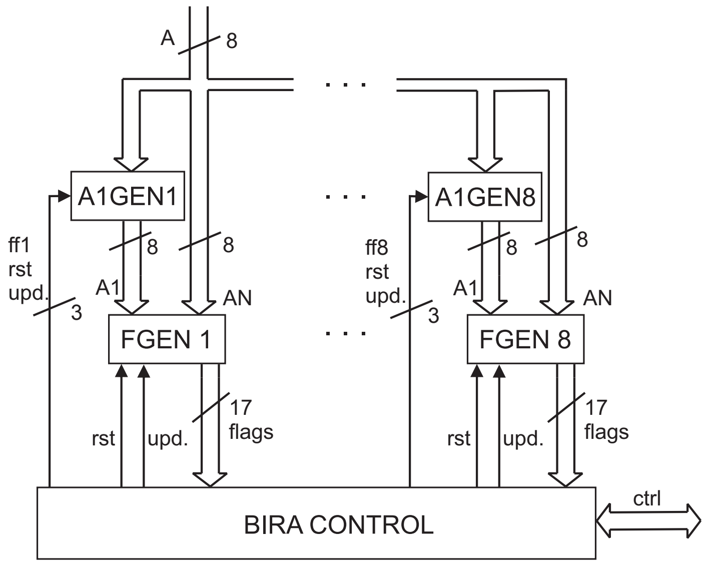

Figure 5. Proposed BIRA architecture.

test the main memory with MARCH C- and MARCH B test algorithms [2], 1 kB (64x16 bits) main memory and 8 16-bit spares.

Future work includes the design of the BIRA control block as well as the AR block. The control block will generate the necessary control signals as was described in Section 4. The AR block will process the final repair solution stored on the flag bits of FGEN circuits as was described in Section 4.

The entire BISR architecture will be described in the VHDL language and will be fully synthesizable. The area overhead will be evaluated for various memory sizes and number of spares to verify its feasibility.

Acknowledgement: This work has been supported by Slovak national project VEGA 1/0616/14.

References

[1] Chen, T.J., Li, J.F., Tseng, T.W.: Cost-Efficient Built-In Redundancy Analysis With Optimal Repair Rate for RAMs. *Trans. on Computer-Aided Design of Integrated Circuits and Systems, IEEE*, 2012, vol. 31, no. 6, pp. 930–940, doi: 10.1109/TCAD.2011.2181510.

[2] Fischerova, M., Gramatova, E.: *Chapter 7: Memory Testing and Self-Repair, In: Design and Test Technology for Dependable Systems-on-Chip*. IGI Global, Hershey, Pennsylvania, 2010, doi: 10.4018/978-1-60960-212-3.

[3] Jeong, W., Kang, I., Jin, K., Kang, S.: A Fast Built-in Redundancy Analysis for Memories with Optimal Repair Rate Using a Line-Based Search Tree. *Trans. on VLSI Systems, IEEE*, 2009, vol. 17, no. 12, pp. 1665–1678, doi: 10.1109/TVLSI.2008.2005988.

[4] Lee, M., Denq, L.M., Wu, C.W.: A Memory Built-In Self-Repair Scheme Based on Configurable Spares. *Trans. on Computer-Aided Design of Integrated Circuits and Systems, IEEE*, 2011, vol. 30, no. 6, pp. 919–929, doi: 10.1109/TCAD.2011.2106812.