Efficient Repair Rate Estimation of Redundancy Algorithms for Embedded Memories

Štefan Krištofík, *Member, IAENG*

***Abstract*—One important feature of redundancy analysis (RA) algorithms is repair rate. To estimate repair rate of various RA algorithms, software simulations of the algorithms on a number of memory fault maps representing real faulty memories are needed. In order to obtain realistic estimations, the fault distribution in maps has to resemble distributions observed in real chips as much as possible. In this paper, it is shown how fault distributions affect repair rate of some RA algorithms. Also, a universal fault map generator based on random and cluster-oriented approaches suitable for repair rate estimations of RA algorithms is proposed.**

***Index Terms*— embedded memory, fault distribution, memory fault map, redundancy analysis algorithm, repair rate**

I. Introduction

According to Semico Research Corp. forecast [1], area occupied by embedded memories on systems-on-a-chip (SoC) designs is slowly growing and will approach 70 % in the next few years. SoCs are moving from logic dominant to memory dominant. Overall SoC yield is therefore dominated by memory yield. As we move deeper into nanometer technology, embedded memory density and capacity grows which results in higher susceptibility of memories to various defects causing memory cells to perform faulty. This in turn causes memory and SoC yield to decrease. Maintaining acceptable yield has become an important task.

Built-in self-repair (BISR) techniques based on using redundancy are widely used to improve yield. Redundant rows and columns are added to the memory. Faulty memory cells are replaced by redundant ones. This replacement is done according to repair solution, which is basically a mapping between faulty cells and redundancies. One important part of BISR actually responsible for finding a repair solution for memories is redundancy analysis (RA) algorithm. Over past ten years, many BISR approaches and RA algorithms for various memory and redundancy architectures were proposed [2]-[6], [10]-[16]. One important feature of RA algorithms is repair rate. Repair rate is defined according to [4] as is stated by (1).

Repair rate depends on the number of redundancies available on chip, which is a fixed value. More redundancies means higher repair rates [2]-[4]. It also depends on effectiveness of RA algorithm. With fixed number of redundancies, the RA algorithm which has higher repair rate is more efficient (i.e. it can repair more memories with available redundancies). Other important features of RA algorithms are repair time and area overhead on chip needed to implement the algorithm.

$$repair\ rate = \frac{\#\ of\ good\ memories\ after\ BISR}{\#\ of\ total\ memories} \qquad (1)$$

To estimate the repair rate of RA algorithms, typical approach is to develop a software simulation tool capable of generating fault memory maps (also termed memory maps or fault maps) and executing the RA algorithm. Memory maps model a real memory as a two dimensional array of cells arranged into rows and columns. Examples of memory maps can be found in section III.

In general, faults can be distributed across the memory map in various ways. To obtain realistic estimates of repair rates of RA algorithms, simulations need to be performed on a certain (usually high) number of memory maps with fault distributions resembling distributions seen in real faulty memories as much as possible. Wafer maps with locations of defects were previously difficult to obtain, but new techniques were introduced as early as late 80's [7]. These techniques showed that defects typically are clustered, not randomly distributed on wafer level. Many other studies (e.g. [8], [9]) confirm this observation. As there are many memory chips per wafer, this clustered distribution affects memory chips in such a way that some chips are fault free but others, located around the clusters have more faults (see Fig. 1). Fig. 1 depicts two examples of wafer maps with defect locations. The first example (a) assumes a very dense defect distribution whereas in the second example (b) the defect clusters occur mainly around the edges.

To simulate such distributions as in Fig. 1, more sophisticated defect distributions than random have to be considered in simulation tools and yield models (e.g. [8], [9]). On memory level, however, software tools able to simulate fault clustering that corresponds to wafer level defect distributions such as in Fig. 1 are needed to estimate repair rates of RA algorithms.

In this paper, we propose a universal fault memory map generator suitable for efficient estimation of repair rates of RA algorithms. It is based on random and cluster-oriented approaches.

Manuscript received March 15, 2013; revised April 5, 2013. This work was supported in part by the Slovak Science Grant Agency (VEGA 1/1008/12).

Š. Krištofík is with the Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava 842 16, Slovakia (e-mail: kristofik@fiit.stuba.sk, zastaph@gmail.com).

TABLE I
ESTIMATION OF REPAIR RATE OF RA ALGORITHMS

	year	tool name	avg. faults	fault distribution(s)	# fault maps	fault map size(s)	# redundancy	single faults %
[10]	2003	BRAVES	-	Poisson + Gamma	1552	1024x64	R 6-10 C 2-6	-
[2]	2006	-	17	random, adjustable % of 3 fault types	-	1024x64	-	adjustable
[11]	2006	-	83 to 189	random + Poisson	500	1024x1024	R 10-32 C 10-32	-
[12]	2006	eval. & verify platform	max. 10	Poisson	500	4096x128	-	0 % 50 %
[6]	2007	-	1-15	random	3000	1024x1024	R 2-5 C 2-5	20-65 %
[13]	2007	-	5-400	fixed % of each of 15 types of faults	18	32x32 to 8192x8192	R 1-30 C 1-30	-
[4]	2009	RepairSim	1-18	random	900000	1024x1024	R 5 C 5	69,32 %
[3]	2009	-	15	negative binomial	-	1024x1024	R 4-8 C 4-8	70 %
[14]	2011	-	7,8 3,3	Poisson Poisson	- 453	256x32 8192x64	R 3-6 C 3-9	20-100 % 70 %
[15]	2011	-	max. 10	random	500	8192x64 32768x64	R 0-4 C 0-4	40-100 %
[16]	2011	-	-	fixed % of each of 4 types of faults	1000	1024x128 2048x64	R 1-4 C 1	0-80 %
[5]	2012	eval. & verify platform	max.10	Poisson	3	512x1024	R 1-5 C 1-3	-

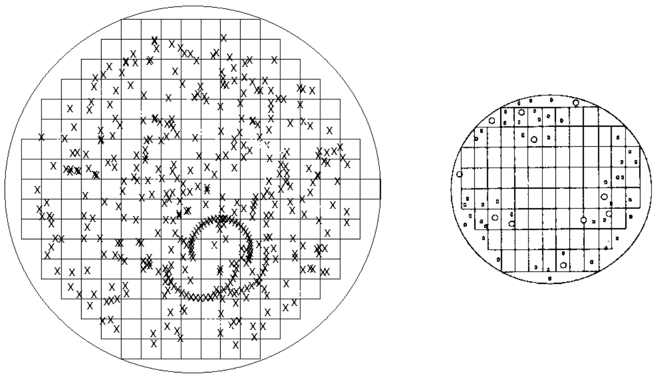


Fig. 1. Wafer level defect distribution examples. (a) [8], (b) [9].

II. RELATED WORK

The repair rates of RA algorithms are estimated in various ways. Usually, the authors implement their own software simulation tool capable of running the algorithm or in some cases more types of algorithms. Table I summarizes the various approaches for repair rate estimations found in literature.

Faults injected into memory maps are usually of various types. Single faults are most common. Usually 50 % or more of all faults in generated memory maps are single faults. Single fault is the only fault on its row and column. It is sharing neither row nor column address with any other fault. Other commonly injected faults are row and column faults (more than one fault on a row or column), clustered faults with cluster radius of 3x3 cells up to larger clusters of various shapes and other special fault types (e.g. column twin-bit fault, two adjacent faulty cells in a column). The fault distributions in fault maps used for repair rate estimations are either generated randomly or based on some theoretical distributions. The average numbers of faults in maps are varied. In some cases, they are set low, but there are cases where they are set as high as 100 per MB or even more. Fault maps are usually of various dimensions (sizes) up to 64MB (8192x8192). Often the maps with rectangular sizes (for example 1024x64) are considered rather than square ones. The numbers of redundancies (R=rows C=columns in Table I) are either set to a fixed value or experiments are conducted with varying numbers (up to 32 rows and columns per MB).

III. PROPOSED FAULT MAP GENERATOR

The proposed fault map generator RNDCLUS is based on the random cluster generator approach proposed in [7], which is able to generate symmetric clusters of faults, using symmetric Gaussian distribution, on the wafer level. The clusters are centered in the centers of the fault maps. In next step, it randomly stretches, rotates and relocates the clusters. In last step, it adds additional clusters to the map that simulate scratches that occur during manufacturing process. We adopt this approach and use it on the memory fault map level. We however, omit the scratching simulation, but add an option to generate fault maps randomly when desired by the user. We now describe the fault map generation process of RNDCLUS.

A. Random option and centered clusters

Probability of faults occurring in memory cells is defined as follows [7]:

$$P(x,y) = Ce^{-(x^2+y^2)/2\sigma^2} \quad (2)$$

where C is a constant and σ is the standard deviation. The values of P(x,y) range from 0 to 1. The address values of x and y both range from -1 to 1. They are related to actual memory addresses in a way that is explained in Fig. 2, which shows an example for a small 8x8 memory map. Probability matrix of values of P(x,y) in Fig. 2 was obtained from (2) using values C=1 and σ=0,6.

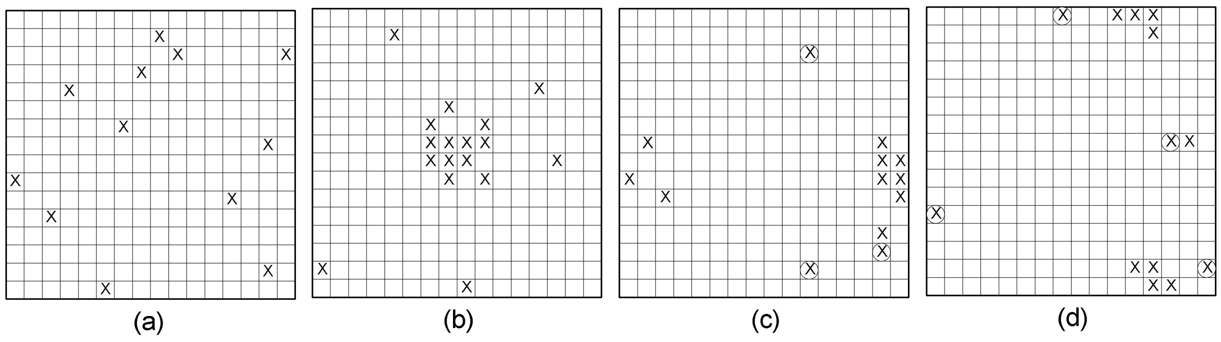
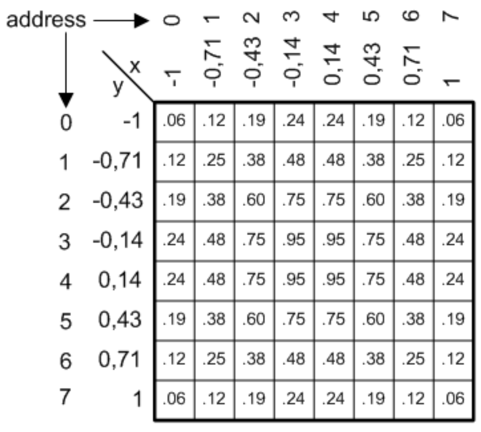


Fig. 3. Examples of fault maps: (a) random, (b) centered cluster, (c), (d) randomized clusters.

To generate actual fault maps, for each map location an auxiliary value of N(x,y) ranging from 0 to 1 is randomly generated. Then if N(x,y) < P(x,y) a fault is injected into the location given by corresponding values of x and y. The result is a symmetric cluster of faults centered in the center of the fault map. The value of σ sets the radius of the cluster and the value of C sets the fault density within the cluster. An example of a fault map with a symmetric cluster is shown in Fig. 3 (b). The fault clustering can be seen around the center as well as some other faults near the edges. An example of a fault map created with random option is shown in Fig. 3 (a) for comparison. The addresses of faults are generated randomly and range from 0 to dimension-1. Random option is used exclusively with centered cluster function (i.e. fault map either has a centered cluster or it is generated randomly – see section III.F).



address	y \ x	0: -1	1: -0,71	2: -0,43	3: -0,14	4: 0,14	5: 0,43	6: 0,71	7: 1
0	-1	.06	.12	.19	.24	.24	.19	.12	.06
1	-0,71	.12	.25	.38	.48	.48	.38	.25	.12
2	-0,43	.19	.38	.60	.75	.75	.60	.38	.19
3	-0,14	.24	.48	.75	.95	.95	.75	.48	.24
4	0,14	.24	.48	.75	.95	.95	.75	.48	.24
5	0,43	.19	.38	.60	.75	.75	.60	.38	.19
6	0,71	.12	.25	.38	.48	.48	.38	.25	.12
7	1	.06	.12	.19	.24	.24	.19	.12	.06

Fig. 2. Values of P(x,y) for 8x8 memory.

B. Relocated clusters

Relocating the clusters of faults is done by generating random values x_m and y_m. Their values range from 0 to dimension-1. Then all faults are relocated to a new location given by summing their original location (row address y, column address x) with the values x_m and y_m:

$$x = x + x_m \qquad (3)$$

$$y = y + y_m \qquad (4)$$

In case the new location is out of the bounds of the fault map, the approach [7] used the cropping technique and discarded the out-of-bounds faults. We however modify this behavior and treat the fault map as a surface of a sphere and the fault re-emerges on the other side of the fault map. This is done to avoid possible high fault count losses in memory maps.

C. Shaped clusters

Shaping of clusters is done by generating random values x_s and y_s. Their values range from 0,1 to 1 meaning that the cluster is stretched by a minimum of 0 % (when x_s or y_s=1) and up to 90 % (when x_s or y_s=0,1). Next, all faults have their original location multiplied by the values of x_s and y_s:

$$x = x * x_s \qquad (5)$$

$$y = y * y_s \qquad (6)$$

By executing previous procedure, the clusters would be not only stretched, but also slightly moved towards the upper left corner of the map since their actual row and column locations are decreased. Therefore, after the procedure, we compensate this by following modifications obtained with trial and error experiments:

$$x = x + dimension * (1 - x_s)^{\frac{1}{x_s}} \qquad (7)$$

$$y = y + dimension * (1 - y_s)^{\frac{1}{y_s}} \qquad (8)$$

D. Rotated clusters

Rotation of clusters is done by generating a random value of angle α ranging from 0 to 359. The clusters are rotated by this angle counterclockwise around the center of the map. If a fault is out of the bounds of the fault map, we again do not use the cropping technique, as stated in section III.B, and the fault re-emerges on the other side of the map. Since we use the non-standard left handed Cartesian coordinate system to assign location (addresses) to faults where the row address is increased downwards instead of upwards, it is first necessary to temporarily convert the addresses to standard right handed system. Next, the center of the coordinate system is "moved" to the center of the fault map by temporarily modifying the fault addresses. Without this step, the rotation would be done around the lower left corner of the fault map and not around the center. The actual rotation follows and all faults have their locations in the map recalculated according to these standard rotation equations:

$$x = x * \cos\alpha - y * \sin\alpha \qquad (9)$$

$$y = x * \sin\alpha - y * \cos\alpha \qquad (10)$$

In the last two steps, two temporal changes made previously are reverted and addresses are reverted back to left handed coordinate system.

E. Randomized clusters with added random faults

By combining the procedures from sections III.A – III.D, the resulting fault distribution can be randomized even more. Lastly, to add some more final randomization to resulting fault distributions, a small number of faults is added at random locations. This number is generated randomly and its value range from 0 to 5 meaning that a maximum of 5 randomly located faults are added to the distributions obtained by procedures from sections III.A – III.D. Two examples of randomized clusters with added random faults are shown in Fig. 3 (c) and (d). For example, the fault map in Fig. 3 (c) was obtained from the fault map in Fig. 3 (b) by using values α=124, x_s=0,28, y_s=0,52, x_m=8, y_m=1 and number of randomly added faults was 3. The circled faults are the ones added randomly.

The results from Fig. 3 (c) and (d) are very similar when compared to results in [17] and [18]. Both studies show random fault map examples similar to that in Fig. 3 (a) and clustered fault map examples similar to those in Fig. 3 (c) and (d).

F. Parameters

Based on the observations in section II, we have set the basic parameters of RNDCLUS according to Table II. It is able to generate a large number of square fault maps of sizes up to 1024x1024. Gauss distribution was chosen because it generates sufficient ‘starting’ clustering of faults and then by modifying it (sections III.A – III.D) we are able to achieve similar results to those reported in [17] and [18]. Therefore there is no need to use more complex theoretical distributions. The average number of faults for small memories (16x16) was set to 10. For large memories (1024x1024), it was first set to 15 as in [3]. Then we tried to set the parameters C and σ of generator so that the average number of faults in generated maps is 15, but were able to approximate it only to 17. For 512x512 memories, the approximation was also done and the average number of faults was 16. For other fault map sizes, the average number of faults obtained was 15. The approximations were done on a trial and error basis while setting the values of C and σ and running the simulations until desired average numbers were obtained. The resulting parameters C and σ for each fault map dimension are listed in Table III. All the example fault maps on Fig. 3 were created using the parameters from Table III for dimension 16. The procedures from sections III.A – III.D are used randomly with a certain probability given by values in Table IV, for each generated memory map. Most of these values are user adjustable.

TABLE II
BASIC PARAMETERS

fault map size	avg. faults	# fault maps	fault distribution
16x16	10	1-100000	Gauss + random
32x32 – 256x256	15		
512x512	16		
1024x1024	17	1-10000	

TABLE III
PARAMETERS C AND σ

dim.	16	32	64	128	256	512	1024
C	1	0,4	0,05	0,05	0,05	0,05	0,05
σ	0,15	0,1	0,018	0,009	0,0045	0,0023	0,0012

TABLE IV
ADVANCED PARAMETERS

parameter	range	description
cluster_chance	0-1	A prob. there is a cluster in fault map. If there is not, random option is invoked.
cluster_reloc	0-1	A probability that if there is a cluster in fault map, it will be randomly relocated.
cluster_shp	0-1	A probability that if there is a cluster in fault map, it will be randomly shaped.
cluster_rot	0-1	A probability that if there is a cluster in fault map, it will be randomly rotated.
rndcnt_max	0-5	Sets the max. of randomly added faults in case there is a cluster in fault map.
rndcnt_max_nc	-	Sets the max. of randomly added faults in case there is not a cluster in fault map. These values are fixed to 2*(avg. faults) column from Table II.

G. Function

The functional flow of RNDCLUS is shown in Fig. 4. Output is stored into text file containing generated fault maps in the form of a list of fault location addresses.

IV. EXPERIMENTAL RESULTS

We now show how various fault distribution types can affect estimation of repair rate of RA algorithms. The modified essential spare pivoting (MESP) algorithm [3] was selected for implementation because it is targeted specially on cluster faults. It targets the block-based redundancy architecture with divided word and bit line techniques. Memory is divided into several quadrants of same size and redundancies are divided into several blocks of same size.

We estimate the repair rate of MESP on small (dim. 16), medium (dim. 128) and large (dim. 1024) memories. Maximum number of generated maps from Table II was selected. The number of quadrants of MESP is assumed to be 16. RNDCLUS generator is used in 6 various configurations shown in Table V.

We have selected these configurations to answer the following questions:

1. Is repair rate of MESP higher when dealing with clustered faults than with random faults, as is expected [3]? We observe the differences in repair rate between configuration RND and others.
2. How is repair rate of MESP affected by the percentage of clustered faults? We observe repair rate while decreasing parameter cluster_chance from 0,75 to 0,5 and then to 0,33.
3. How is repair rate of MESP affected by the number of randomly added faults? We observe repair rate while

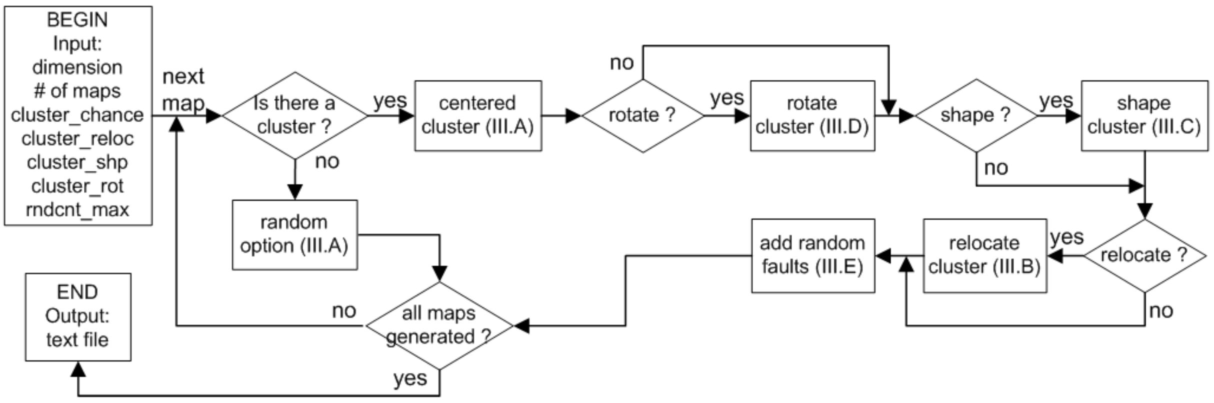


Fig. 4. Flow diagram of RNDCLUS.

TABLE V
CONFIGURATIONS OF RNDCLUS

configuration type	random	cluster-oriented						
configuration name	RND	C 0,75	C 0,5	C 0,33	add3	noadd	C 0,75	C 0,5
cluster_chance	0	0,75	0,50	0,33	0,75	0,75	0,75	0,50
cluster_reloc	-	0,75	0,75	0,75	0,75	0,75	0,75	0,75
cluster_shp	-	0,50	0,50	0,50	0,50	0,50	0,50	0,50
cluster_rot	-	0,75	0,75	0,75	0,75	0,75	0,75	0,75
rndcnt_max	-	5	5	5	3	0	5	5

decreasing parameter rndcnt_max from 5 to 3 and then to 0.

4. Will repair rate of MESP estimated by RNDCLUS be similar to repair rate reported in [3]? If not, what are the possible causes and what can be done to obtain more similar results?

Table VI shows the repair rate of MESP using all 6 RNDCLUS configurations from Table V. The number of redundancies ranged from 3 row and column blocks (3/3) to 12 row and column blocks (12/12). In the last column of the table, the resulting repair rate obtained by RNDCLUS with the results reported in [3] is compared where available. However, results for RNDCLUS were obtained for average number of faults equal to 17 whereas results in [3] are for average number of faults equal to 15. Also, it is unknown how the numbers of faults were generated for each memory map (Were they generated equally or using some distribution?) and what was the total number of generated fault maps. By analyzing the results in Table VI, the aforementioned questions can be answered:

1. Yes. In small memories this becomes evident when the number of redundancies reaches 4 and for medium and large memories when it reaches 7.

2. Repair rate slightly increases when the numbers of redundancies are small and it begins to decrease with increasing the number of redundancies. This is an expected result since the larger the map, the thinner are the generated clusters and the percentage of single faults increases which in turn has negative impact on repair rate.

3. Repair rate increases greatly with all sizes of memories with decreasing the number of added random faults. This suggests that the initial value of rndcnt_max equal to 5 was set too high.

4. Yes, in most cases. Repair rates are similar to those reported in [3] when cluster-oriented distributions are considered. They are slightly lower with most of the RNDCLUS configurations. This may be caused by higher average fault count than in [3]. In case random option is used, the repair rate is significantly lower for any number of redundant blocks.

V. CONCLUSIONS AND FUTURE WORK

The goal of this work is to offer the most exact estimations of repair rates of RA algorithms which can only be done if simulations are performed on memory fault maps with fault distributions that resemble fault distributions in real memory arrays as closely as possible. But to obtain such information from industry is not an easy task and one can only rely on other published approaches.

Various known approaches to repair rate estimation problem were reviewed and based on that, a universal, user-adjustable fault map generator RNDCLUS was proposed. According to experimental results, it is suitable for estimation of repair rate of RA algorithms. By setting the values of various parameters of RNDCLUS, one can modify the output and is able to select whether the distributions are more random or more cluster-oriented.

Experiments have shown some interesting results as well as proving some expected results. They also proved that the repair rate of RA algorithms is very heavily dependent on fault distributions in fault memory maps. It is worth further studying, with different algorithms and sets of parameters to obtain more results. Future research work will be invested to further study this on other algorithms as well as to further improving the proposed generator with features such as adding new distributions or new cluster-generating approaches i.e. more than one cluster per map, random cluster sizing, random cluster positioning in small quadrants of memory maps and so on.

TABLE VI
Repair rate % of MESP with different configurations of RNDCLUS

dim.	# redundancy	RND	C 0,75	C 0,5	C 0,33	add3	noadd	[3]
16	3/3	32,91	26,60	28,88	30,28	34,59	52,53	-
	4/4	45,22	52,90	50,43	49,02	64,28	77,33	-
	5/5	58,10	76,66	70,49	66,51	83,44	87,79	-
	6/6	71,77	89,71	83,74	79,98	91,70	92,43	-
	7/7	85,75	95,81	92,52	90,40	96,02	96,10	-
	8/8	96,33	98,82	98,04	97,53	98,90	98,86	-
128	3/3	20,27	5,46	10,41	13,75	5,71	6,96	-
	4/4	26,93	9,70	15,44	19,34	11,03	16,47	-
	5/5	33,87	18,84	23,97	27,25	22,88	34,34	-
	6/6	40,88	34,03	36,31	37,88	41,39	55,44	-
	7/7	47,81	52,72	51,42	49,94	61,22	72,81	-
	8/8	54,81	69,69	64,75	61,26	76,29	83,26	-
	9/9	61,91	81,89	75,33	70,59	85,56	88,60	-
	10/10	69,07	88,91	82,40	77,81	90,42	91,54	-
1024	3/3	18,28	4,57	9,25	11,38	4,57	5,80	-
	4/4	24,47	7,39	13,22	16,22	8,24	12,38	-
	5/5	30,36	14,36	20,09	22,46	18,02	28,37	-
	6/6	36,35	27,68	30,73	31,36	34,77	49,32	-
	7/7	42,10	45,77	44,29	42,69	55,35	67,45	-
	8/8	47,96	63,10	58,14	54,13	71,60	78,85	65,50
	9/9	53,91	76,29	68,79	63,42	82,10	85,06	83,00
	10/10	59,62	84,45	76,09	70,36	87,54	88,25	93,00
	11/11	65,49	88,97	80,69	75,88	90,68	90,09	96,00
	12/12	71,36	91,55	84,41	80,18	92,58	91,75	98,00

REFERENCES

[1] Semico Research Corp., “Semico: System(s)-on-a-Chip A Braver New World”, 2007. [Online]. Available: http://www.semico.com/press/press.asp?id=200

[2] S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang and C.-W. Wu, “Efficient Built-In Redundancy Analysis for Embedded Memories With 2-D Redundancy”, *IEEE Trans. VLSI Systems*, vol. 14, no. 1, pp. 34-42, 2006.

[3] S.-K. Lu, C.-L. Yang, Y.-C. Hsiao and C.-W. Wu, “Efficient BISR Techniques for Embedded Memories Considering Cluster Faults”, *IEEE Trans. VLSI Systems*, vol. 18, no. 2, pp. 184-193, 2009.

[4] W. Jeong, I. Kang, K. Jin and S. Kang, “A Fast Built-in Redundancy Analysis for Memories With Optimal Repair Rate Using a Line-Based Search Tree”, *IEEE Trans. VLSI Systems*, vol. 17, no. 12, pp. 1665-1678, 2009.

[5] T.-J. Chen, J.-F. Li and T.-W. Tseng, “Cost-Efficient Built-In Redundancy Analysis With Optimal Repair Rate for RAMs”, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 930-940, 2012.

[6] P. Öhler, S. Hellebrand and H.-J. Wunderlich, "An Integrated Built-in Test and Repair Approach for Memories with 2D Redundancy", in *Proc. 12th IEEE European Test Symposium*, 2007, pp. 91-96.

[7] C. H. Stapper, “Simulation of Spatial Fault Distributions for Integrated Circuit Yield Estimations”, *IEEE Trans. Computer-Aided Design*, vol. 8, no. 12, pp. 1314-1318, 1989.

[8] R. Allison, “SAS/Graph Wafer Maps”, in *Robert Allison’s SAS/Graph Examples*, 2004. [Online]. Available: http://robslink.com/SAS/democd10/waferx.htm

[9] W. Kuo and T. Kim, “An overview of manufacturing yield and reliability modeling for semiconductor products”, in *Proc. IEEE,* 1999, vol. 87, no. 8, pp. 1329-1344.

[10] C.-T. Huang, C.-F. Wu, J.-F. Li and C.-W. Wu, “Built-In Redundancy Analysis for Memory Yield Improvement”, *IEEE Trans. Reliability,* vol. 52, no. 4, pp. 386-399, 2003.

[11] H.-Y. Lin, F.-M. Yeh and S.-Y. Kuo, "An Efficient Algorithm for Spare Allocation Problems", *IEEE Trans. Reliability,* vol. 55, no. 2, pp. 369-378, 2006.

[12] T.-W. Tseng, J.-F. Li *et al.*, “A Reconfigurable Built-In Self-Repair Scheme for Multiple Repairable RAMs in SOCs”, in *Proc. IEEE International Test Conference*, 2006, pp. 1-9.

[13] S. Bahl, “A Sharable Built-in Self-repair for Semiconductor Memories with 2-D Redundancy Scheme”, in *Proc. 22nd IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems,* 2007, pp. 331-339.

[14] C.-L. Su, R.-F. Huang *et al.,* ”A Built-in Self-Diagnosis and Repair Design With Fail Pattern Identification for Memories”, *IEEE Trans. VLSI Systems*, vol. 19, no. 12, pp. 2184-2194, 2011.

[15] T.-W. Tseng and J.-F. Li, “A Low-Cost Built-In Redundancy-Analysis Scheme for Word-Oriented RAMs With 2-D Redundancy”, *IEEE Trans. VLSI Systems*, vol. 19, no. 11, pp. 1983-1995, 2011.

[16] Y.-J. Chang, Y.-J. Huang and J.-F. Li, “A Built-In Redundancy-Analysis Scheme for RAMs with 3D Redundancy”, in *Proc. International Symposium on VLSI Design, Automation and Test,* 2011, pp. 1-4.

[17] A. Pelc and D. M. Blough, “A clustered failure model for the memory array reconfiguration problem”, *IEEE Trans. Computers,* vol. 42, no. 5, pp. 518-528, 1993.

[18] M. Choi, N. Park, F. J. Meyer, F. Lombardi and V. Piuri, “Reliability measurement of fault-tolerant onboard memory system under fault clustering”, in *Proc. 19th Instrumentation and Measurement Technology Conference*, 2002, vol.2, pp. 1161-1166.