# High Quality Simulation Tool for Memory Redundancy Algorithms

Kaname Yamasaki Said Hamdioui Zaid Al-Ars Arjan van Genderen Georgi N. Gaydadjiev
Delft University of Technology, Faculty of Electircal Engineering, Mathematics and Computer Science,
Computer Engineering Laboratory, Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: kaname@dutepp0.et.tudelt.nl

***Abstract*—This paper presents a high quality simulation tool that evaluates the efficiency of redundancy algorithms (RAs) for repairable memory devices. The tool can generate various faulty memory models to be analyzed by a given RA. Furthermore it can provide useful information for the feedback to RA under evaluation when the RA fails to repair any theoretically repairable model. The approach and the external specifications are described according to the properties required for the tool.**

**Keywords:** Redundancy algorithm (RA), efficiency, yield, built-in self repair (BISR), faulty memory model.

## I. Motivation

Modern system-on-chips (SoCs) involve increasing number of various Intellectual Properties (IPs). Most of the IPs are apt to include their own dedicated memories. This is the reason why more memories with different types, sizes and operating frequencies have been embedded in a single SoC. It is also increasing the memory share of the overall chip area budget, which is more than 50 % in some present SoCs. Consequently, the yield of a chip depends more on that of the memories it embeds. A repairable memory, or a memory equipped with the redundancy circuits such as spare rows, columns and data-bits, is effective to keep the yield at acceptable level. Small faults in the memory can be repaired by replacing with the redundancy circuits. One dimensional (1D) redundancy employs either of the above redundancy circuits. Moreover two dimensional (2D) redundancy employs the spare columns or data-bits with the spare rows. A way how to repair the faults on a memory with 2D redundancy is referred as redundancy algorithm (RA). The RA problem has been shown to be NP-complete [5] hence many heuristic approaches have been previously developed and proposed. At the same time, a built-in self repair (BISR) scheme must be implemented because it is not economical to repair many large embedded memories using external automatic test equipment (ATE). The proposed RAs for BISR are in general less efficient than the ATE based ones due to the chip area overhead limitations of the BISR schemes. Therefore more efficient and compact RA is necessary for improved yield and reduced test time.

Some simulation tools for evaluating the efficiency of RA have been developed and proposed by Virage Logic [8] [10] [9] [12], National Tsing Hua University [2] [3], and others [13] [7] [14]. However, for simplicity, they limit variety of faults in a memory model to be analyzed by RA. This limitation may cause incorrect evaluations. Moreover, their tools don't provide any feedback information to RA under evaluation. In short, they don't help the RA improve but only evaluate it.

According to Nakahara et. al [6], even a simple RA is comparable in efficiency to other superior RAs such as repair most [11] and branch & bound [5], if simple failures can be fixed effectively. Their approach was experimentally evaluated using their simple RA and hundreds of repairable faulty models sampled from identical real memory devices. Actually, the sample contained more than 75% of models with simple failures. That means models with simpler failures increase the efficiency while those with more complicated ones decrease it. In other words, an over- or under-estimation of RA may happen if simpler or more complicated failures in models are assumed than common ones in real devices. Therefore, the faults in a memory model should be realistic from the point of view of type, probability and location.

This paper presents a new simulation tool for proposing compact and efficient RAs especially for BISR, which evaluates them using realistic memory models and also provides feedback information in case of failed repairs. Section II discusses the specification and functions required for the tool. Section III discusses the ways to satisfy the above properties. Section IV describes the external interfaces of the tool. Section V concludes this paper.

## II. Tool Properties

The purpose of the tool development is to propose and evaluate an optimum RA according to the memory configuration, redundancy circuits and process technology. The following items are required for the tool.

1) The efficiency of a given RA, or RA efficiency, should be evaluated quantitatively but fast: Various faults can occur in a memory device during the manufacturing process. So many faulty memory models should be considered, and then analyzed by RA. Generally quantitative approach causes complex computations. So some techniques to simplify the computation are also required.
2) Realistic faults in a faulty memory model should be considered for more exact evaluation: The faults should be realistic in view of the types, probability and location. Because non-realistic faults may cause over- or under-estimation as described above.

3) Configuration of the memory array and redundancy circuits in a memory model should be variable: The efficiency is defined only for a same type of memory device that has same architecture and configuration. Therefore the numbers of regular rows and columns, and spare rows and columns should be based on a repairable memory device that employs RA.
4) A handmade faulty memory model should be also analyzed by a given RA: There are several well-known faulty memory models as difficult problems for most RAs [1]. A proposed RA should be applied to such models in order to check the superiority.
5) When RA fails to repair a faulty memory model, additional information should be also available: First, the fault bit map is definitely necessary to know whether it has really complicated but repairable faults. Second, the locations of consumed spare rows and columns are also useful to know whether they are effectively allocated. Moreover, the ratio of repaired fault classes to all fault ones is defined as the repair rate for class (Rc), which is useful to know how many classes RA recognizes. Similarly, the ratio of repaired fault bits to all fault ones is defined as the repair rate for bit (Rb), which is useful to know whether RA miss large fault classes.

## III. Approach

The ways to satisfy the above properties are discussed in this section.

### A. Concept and Definition of RA Efficiency

Figure 1 shows the categorization of a memory device based on the faultiness and repairability. All memories set represents many identical memory devices with the redundancy circuits, which have the same architecture and configuration, and are fabricated by the same process technology. All memories are first categorized into fault-free ones and faulty ones. Furthermore, the faulty ones are categorized into theoretically unrepairable ones and theoretically repairable ones based on the redundancy configuration. General RAs sometimes fail to repair theoretically repairable ones because of the NP-complete problem as described in I. Therefore, theoretically repairable ones are categorized into unrepaired ones and repaired ones based on the analysis result of a given RA. The RA efficiency or $E$ is expressed as the following equation.

$$E = \frac{\text{Number of Repaired Memories}}{\text{Number of Theoretically Repairable Memories}}$$

As various faults can cause randomly in a memory device, enormous numbers of faulty memory models are required for the exact evaluation. Additionally, the order of detecting fault bits in a memory device affects the analysis results of some RAs especially for a repair system without enough memory resources, such as BISR and low-end ATE. In other words, the border between the set of repaired memories and that of unrepaired ones is not so clear as shown in Figure 1. Because such RAs start to allocate a spare element on some fault bits

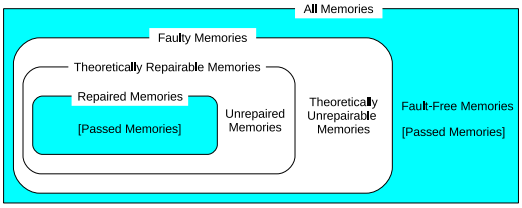


Fig. 1. Categorization of Memory Device

if the memories or registers in the system for storing fault bits are full even before recognizing all of them. The order usually depends on the used test algorithm, the behavior of fault bits etc. Therefore, $E$ should be independent of the order. All considerable orders of handling fault bits for each faulty memory model must be analyzed by RA. Needless to say, it is impossible to apply such analysis to enormous number of faulty memory models. In order to reduce the computational complexity, sampling is applied to the evaluation. Furthermore, all fault bits in a fault class are assumed to be detected not discretely but sequentially. In other words, some memory models are randomly selected as a sample from all memory models, and the order of detecting fault classes for each model is determined in the same fashions described in III-C. Figure 2 shows an example of categorization in the case of 20 models. Note that all fault-free models are represented as different points for the countability although they are the same from the view of fault bit map. The number of theoretically repairable ones is 10, and that of repaired ones is 6. Therefore $E$ is 60 %. This method will provide approximate evaluation when the number of models is small, but more exact evaluation when larger.

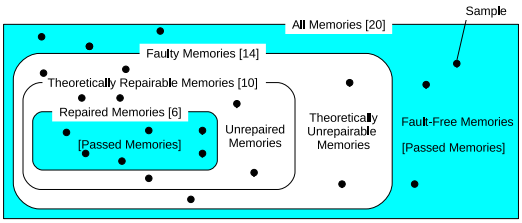


Fig. 2. Example of Categorization

Consequently, RA evaluation is composed of four procedures; sampling, ordering, categorization and calculation. In the first procedure, some memory models are randomly selected as a sample from all memory models with or without faults. The configuration of the models is based on the memory device that employs RA. A designated model is also available instead of random selection. The number of models is changeable according to the exactness. In the second procedure, the order of detecting fault bits in each model is determined. In the third procedure, each model is categorized based on the repairability. When a given RA fails to repair a model, information about the faults and analysis result is provided. In the final procedure, $E$ is calculated according

to the categorization data. The detail of each procedure is described in the following sections.

### B. Sampling

Faulty memory models generated randomly can be regarded as a sample. However, some models may be fault-free or theoretically unrepairable because it is difficult to generate theoretically repairable models selectively for evaluating RA. Therefore the following methods are proposed for generating the models.

1) The number of defects causing faults in a memory is determined statistically. Here a defect means like a crystal defect in Si or a cluster on Si, which has a potential to cause any type of fault. The number is assumed to depend on a probability distribution, or Poisson distribution as a default. Others such as Polya-Eggenberger distribution, gamma distribution, negative binominal distribution, uniform distribution should be also available according to the practical application. The number can be determined from the used cumulative distribution function and uniform random number generator.
2) The transformation from a defect to a fault class is performed also statistically. In fact, it can be done according to the probability of each fault type as well as using a uniform random number generator. As many as 8 types shown in Figure 3 are considered as a fault class. Figure 4, 5 and 6 shows physical topology of each fault type. Each linear fault class like RS, RD, LS and LD includes full or partial linear fault. A partial one is assumed to consist of sequential faulty cells in line from the beginning.

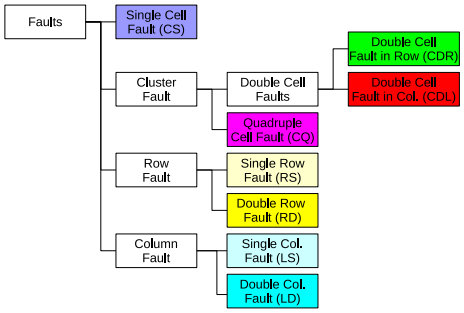


Fig. 3. Categorization of Fault types for Memory Circuit

3) The length of each linear fault class is determined randomly or using a uniform random number generator. The minimum length is as long as must repair is possible, and the maximum one is equal to the length of row or column. For example, when the number of spare rows (or columns) is $m$, the minimum length of a column (or row) fault is $m+1$.
4) The location of each fault class is determined randomly, or using a uniform random number generator. An overlap of 2 or more fault classes is allowed because it can be occurred in a real device. Figure 7 shows a model after allocating faults.

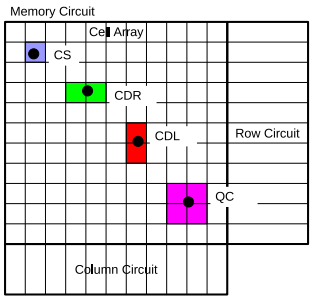


Fig. 4. Topology of single and cluster faults

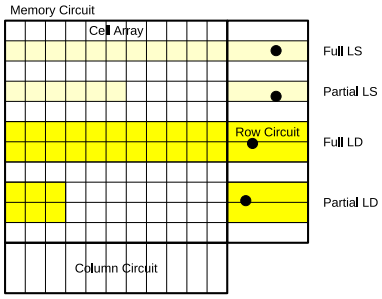


Fig. 5. Topology of row faults

### C. Ordering

A model generated automatically in the above procedures or by hand provides only the location of fault classes, which is not sufficient for evaluating RA. Here the order of detecting fault bits for each model is determined in the following fashion for simplicity. First, all fault bits in a class are assumed to be always detected not discretely but sequentially. Furthermore, all fault classes of the same type are assumed to be always detected sequentially and randomly. Generally, the earlier detection of large fault classes makes it easier to repair the model, which may cause an over-estimation of RA efficiency or $E$. On the contrary, the later detection of them makes it more difficult, which may cause the under-estimation. Both of them are not preferable, but are useful to guess the range of $E$. Additionally if a fault class is randomly detected, it will provide the average of $E$. Therefore the following ways are proposed for ordering fault bits.

1) The order of detecting fault classes in a model is determined from 3 options; order of size starting with the smallest class, that of size starting with the largest class, or at a random. The 1st option, the easiest order to repair the model, will provide the highest range of $E$. The 2nd option, the most difficult order, will provide the lowest range. The 3rd option will provide the average. Figure 8 shows an order of size starting with the smallest fault class, which is ordered as below: (1) CS, (2) CR, (3) CL, (4) CQ, (5) RS, (6) LS, (7) RD and (8) LD.

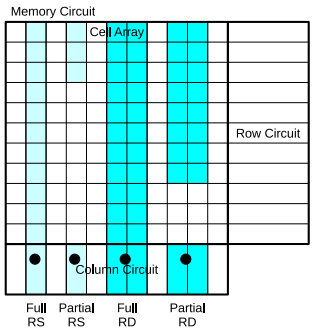


Fig. 6. Topology of column faults

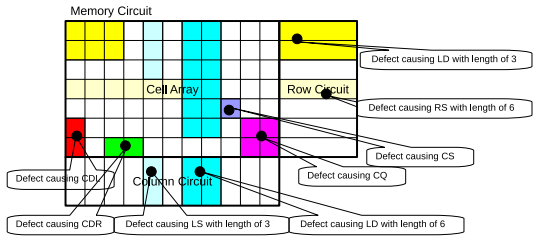


Fig. 7. Memory Circuit after Allocating Faults

2) Each fault class is decomposed into fault bits because RA analyzes every model bit by bit. A model after the decomposition is shown in Figure 9 This is generally called as a fault bit map (FBM).
3) All fault bits are assigned to an array data in the determined order of fault classes. It may be better for the tool implementation to transform their 2-dimensional (2D) addresses to 1-dimensional (1D) ones or cell addresses before the assignment.

### D. Categorization

Models generated in the above procedures are categorized here. They may include fault free ones as described above. Therefore the following ways are proposed for categorizing models.

1) Only faulty models are selected.
2) Only theoretically repairable models are selected from the above faulty ones according to the redundancy configuration. The selection is based on a golden redundancy algorithm such as comprehensive exhaustive search method [4].
3) Each theoretically repairable model is also analyzed by a given RA if it is repaired or not. The RA is given as a source code in C language. The memory resources on a repair system that employs the RA must be considered. For example, on-the-fly RA on a BISR scheme may only handle fault bits as given because the scheme does not have so many memory resources. On the other hand, RA on ATE can analyze after storing all fault bits because ATE has a lot of the resources.
4) If the given RA fails to repair any model, the feedback

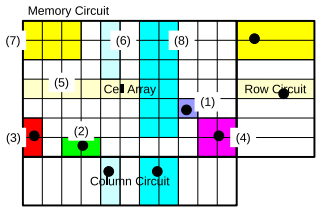


Fig. 8. Order of Size starting with the Smallest Fault Class

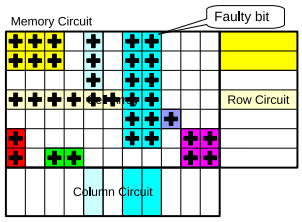


Fig. 9. FBM for RA Evaluation

information is provided for improving RA efficiency. It contains the repair rates of Rc and Rb, the location of all fault classes, and the location of replaced rows and columns. The repair rates are expressed as the following equations.

$$Rc = \frac{\text{Number of Repaired Fault-Classes}}{\text{Number of All Fault-Classes}}$$

$$Rb = \frac{\text{Number of Repaired Faulty Bits}}{\text{Number of All Faulty Bits}}$$

Note that a fault class is not exclusively but only randomly allocated. Therefore it may be partly or fully included by 1 or more other fault classes. In the case of counting fault classes, a partly included class is countable, but a fully included class is not countable. Similarly, in the case of fault bits, 2 or more bits on the same location must not be counted again.

### E. Calculation

The RA efficiency or $E$ is expressed as the following equation.

$$E = \frac{\text{Number of Repaired Models}}{\text{Number of Theoretically Repairable Models}}$$

It is calculated as the transient RA efficiency every time a theoretically repairable model is analyzed and categorized. The exactness of the transient efficiency is low when the number of models is small, but becomes higher when larger.

### F. Techniques for Fast Evaluation

The simulation time for evaluation may be still long due to the following reasons even if only an order of detecting fault bits is considered for each model.

1) Much more than the optimal number of models is always required because it is impossible to estimate the optimal number in advance according to the exactness.

2) Models for a large memory require much memory resource on a computer in the categorization because a linear fault class contains many fault bits according to the size of memory. Furthermore they require much computation in the sampling and categorization because there are more faulty bits on a large memory.
3) A memory with many redundancy circuits requires much computation in the categorization because comprehensive exhaustive search method as a golden RA is composed of permutations of allocating all spare rows and columns. Especially a theoretically unrepairable model wastes most computation because it is always analyzed by all solutions.

The 1st and 2nd problems can be solved. The final problem is not perfectly avoidable because it exactly originates from its NP-complete problem. Therefore the following solutions are proposed.

1) The number of models can be automatically determined according to the convergence of the transient efficiency, which is calculated from the cumulative models. The target for the convergence degree or change of transient efficiency can be given as one of the input parameters. If it is low convergence, small numbers of models generated provide fast and rough evaluation. It is useful for relative evaluation, for example, judging if a new proposed algorithm is better than conventional ones. On the other hand, if it is high convergence, large numbers of models generated provide slow but more exact evaluation. In order to avoid the infinite generation, the maximum number should be also given as one of the input parameters. Furthermore, in order to avoid stopping the simulation at a temporary convergence, the change of the transient efficiency is evaluated after analyzing 100 or 1000 repairable models. In other words, the current transient efficiency should be always compared with the last 100 or 1000 transient efficiency to check if the change reach the target.
2) The dimension of a memory model can be reduced independent of that of a memory device to which RA is applied, but only according to the number of spare rows and columns. The reduced model is called as Fault Array Graph (FAG) and is defined as the array of $m*(n+1)\times n*(m+1)$, which has been proposed by Y. Zorian et. al for their RA evaluation method [10] [9]. $m$ and $n$ mean the number of spare rows and columns, respectively. This model can be much smaller in dimension than the original model, and can solve the computation problems for a large memory. However, a fault class includes partly or fully another one more often, which may give less exact evaluation.
3) The number of models to be categorized can be reduced if clearly unrepairable models are removed in advance. A simple detection method is to count the number of independent fault classes, which don't share row or column with another fault class. If the number is more than the number of all spare elements, the model is unrepairable. Another method is to count the number of faulty rows (or columns) after detecting independent row (or column) fault classes, which are not partly included by another linear fault class in parallel. If the number is more than that of all spare rows (or columns), the model is unrepairable.

## IV. External Specification

The specifications such as architecture and interfaces of the tool are determined below according to the above sections.

### A. Tool Organization

Figure 10 shows a block diagram of the tool. The tool is composed of 4 logical procedures as described above. A model to be analyzed by a given RA is generated and categorized each iteration. All inputs and outputs are composed in text format. The input 1, input 2 and input 3 specify all input parameters, the RA and external model data, respectively. When the RA fails to repair any theoretically repairable model, the repair rates and FBM are added to the output 1. When the RA analyzes any theoretically repairable model, the updated RA efficiency $E$ is added to the output 2. The iteration continues until the exactness $\Delta E$ reaches the target.

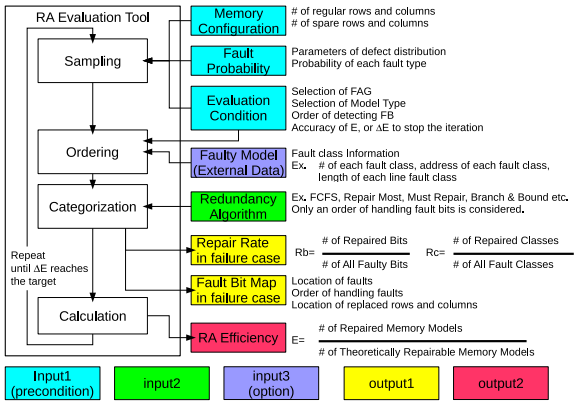


Fig. 10. Block Diagram of Tool

### B. Input

Figure 11 shows an example form of input 1 as all input parameters, which indicates memory configuration, fault probability, evaluating condition and assignment of output files. $delta\_e$ and $nrsef$ are parameters for the fast or exact evaluation. The former is for the change of transient efficiency, and the latter for the intervals between the results to be compared in transient efficiency. The form of input 2 as a given RA describes a source code of RA in C language. The resources of registers and memories on a repair system that employs the RA are considered here. Figure 12 shows an example form of input 3 as an external model data, which indicates number and address of each fault class, and length of each linear fault class.

```
void input (int nrow, int ncol, ...)
{
// Memory Configuration
        nrow    =   256         // Number of rows
        ncol    =   256         // Number of columns
        nsrow   =   2           // Number of spare rows
        nscol   =   2           // Number of spare columns
// Fault Probability
        type_d  =   1           // Distribution type of defects (0: Uniform, 1:Poisson, ...)
        av_d    =   3.8         // Average of defects
        dp_d    =   9.5         // Dispersion of defects
        pf[0]   =   45.0        // Probability of single cell fault (%)
        pf[1]   =   10.0        // Probability of double cell fault in row (%)
        pf[2]   =   10.0        // Probability of double cell fault in column (%)
        pf[3]   =   5.0         // Probability of quadruple cell fault (%)
        pf[4]   =   10.0        // Probability of single row fault (%)
        pf[5]   =   10.0        // Probability of single column fault (%)
        pf[6]   =   5.0         // Probability of double row fault (%)
        pf[7]   =   5.0         // Probability of double column fault (%)
// Evaluating Condition
        sel_sam =   0           // Selection of model type (0:Auto-Generated, 1:Handmade)
        order_FC=   0           // Order of detecting FC (0:from smaller, 1:from larger, 2:at random)
        seed_r  =   0           // Seed for random function
        sel_FAG =   1           // Selection of FAG (0:Full Size, 1:FAG)
        delta_e =   0.1         // Change of transient efficiency (%), Absolute Difference of E
        nrsef   =   100         // # of intervals between the results to be compared in transient efficiency
        nsam    =   1000        // Maximum number of models to avoid infinite iteration
// Assignment of redundancy algorithm and output files
        file_o1 =   "ra1.log"   // File name of output 1 for feedback information
        file_o2 =   "ra1.rst"   // File name of output 2 for RA efficiency
}
```

Fig. 11. Example of Input 1

```
int handmade_model( int nfc, int afc[ ], int len_lfc[ ])
{
// Number of each fault class
        nfc[ ]          =   { 3, 2, 1, 0, 1, 2, 2, 1 }

// Address of each fault class
        afc[ ]          = {   { 1, 3, 0, x, ...},       // Cell address of CS
                              { 2, 100, x, ...},        // Smaller Cell address of CDR
                              { 8, x, x, x, ...},       // Smaller Cell address of CDL
                              { x, x, x, x, ...},       // Smallest Cell address of CQ
                              { 15, x, x, ...},         // Smaller Row address of RS
                              { 4, 7, x, x, ...},       // Smaller Column address of LS
                              { 9, 5, x, x, ...},       // Smaller Row address of RD
                              { 3, x, x, x, ...}    }   // Smaller Column address of LD

// Length of each line fault class
        len_lfc[ ]      = {   { 5, x, x, x, ...},       // Length of each RS
                              { 100, 19, x, ...},       // Length of each LS
                              { 254, 7, x, ...},        // Length of each RD
                              { 42, x, x, x, ...}   }   // Length of each LD
}
```

Fig. 12. Example of Input 3

### C. Output

Figure 13 shows an example form of output 1, which always indicates categorization of each model, and additionally repair rates of Rc and Rb, location of replaced rows and columns, and location of each fault class when RA fails to repair a model. Figure 14 shows an example form of output 2, which indicates the serial number for all models, transient efficiency, categorization, and number of fault classes.

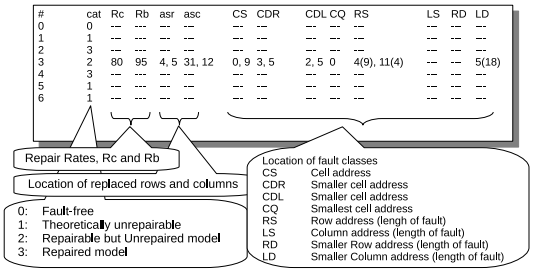


Fig. 13. Example of output 1

## V. Conclusion

This paper presents a high quality simulation tool that evaluates the efficiency of RA applied to repairable memory devices. According to the tool properties, the approach and external specifications are discussed and determined. More

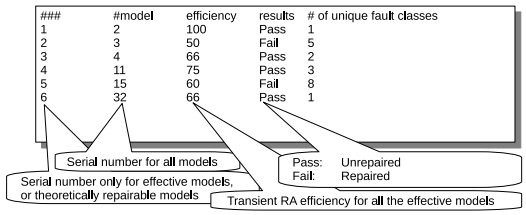


Fig. 14. Example of output 2

various fault types in a memory model enable a more exact evaluation of RA. The 3 different orders of detecting fault bits in a model give upper-limit, lower-limit and average efficiency of RA. The information such as Rc and Rb is provided for the feedback to a given RA. A software tool implementing the proposed approach is being developed currently. We will also present the validation and application results of the tool in our future work.

## References


[1] J. R. Day, *A fault-driven, comprehensive redundancy algorithm*, IEEE Design and Test of Computers, vol. 2, 1985, pp. 35–44.

[2] R.-F. Huang, J.-F. Li, J.-C. Yeh, and C.-W. Wu, *A simulator for evaluating redundancy analysis algorithms of repairable embedded memories*, Proc. Of IEEE intl. workshop on Memory Technology, Design and Testing, 2002.

[3] R.-F. Huang, J.-C. Yeh, J.-F. Li, and C.-W. Wu, *Raisin: Redundancy analysis algorithm simulation*, IEEE Design and Test of Computers, vol. 24, Jul. 2007, pp. 386–396.

[4] T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada, and H. Hidaka, *A built-in self-repair analyzer (cresta) for embedded drams*, Proc. Of IEEE Intl. Test Conference, 2000, pp. 567–574.

[5] S.-Y. Kuo and W. K. Fuchs, *Efficient spare allocation for reconfigurable arrays*, IEEE Trans. on Computers, vol. 41, 1992, pp. 221–226.

[6] S. Nakahara, K. Higeta, M. Kohno, T. Kawamura, and K. Kakitani, *Built-in self-test for ghz embedded srams using flexible patterngenerator and new repair algorithm*, Proc. Of IEEE Intl. Test Conference, 1999, pp. 301–310.

[7] P. Ohler, S. Hellebrand, and H.-J. Wunderlich, *An integrated built-in test and repair approach for memories with 2d redundancy*, Proc. Of IEEE European Test Symposium, 2007.

[8] S. Shoukourian, V. Vardanian, and Y. Zorian, *An approach for evaluation of redundancy analysis algorithms*, Proc. Of IEEE Intl. Workshop on Memory Technology, Design and Testing, 2001, pp. 51–55.

[9] _____, *A methodology for design and evaluation of redundancy allocation algorithms*, Proc. Of IEEE VLSI Test Symposium, 2004, p. 249.

[10] _____, *Soc yield optimization via an embedded-memory test and repair infrastructure*, IEEE Design and Test of Computers, vol. 21, 2004, pp. 200–207.

[11] M. Tarr, D. Boudreau, and R. Murphy, *Defect analysis system speeds test and repair of redundant memories*, Electronics, Jan. 1984, pp. 175–179.

[12] V. Vardanian and G. Gabrielyan, *A tool for evaluation and comparison of redundancy allocation algorithms for static random access memories*, Proc. Of Intl. Conference on Computer Science and Information Technologies, 2003, pp. 391–394.

[13] C.-L. Wey and F. Lombardi, *On the repair of redundant ram's*, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 6, 1987, pp. 222–231.

[14] P. hler, S. Hellebrand, and H.J. Wunderlich, *Analyzing test and repair times for 2d integrated memory built-in test and repair*, IEEE Design and Diagnostics of Electronic Circuits and Systems, 2007.