A Statistical Wafer Scale Error and Redundancy Analysis Simulator

Atishay Atishay, Ankit Gupta, Rashmi Sonawat, Helik Kanti Thacker, B. Prasanth

A Statistical Wafer Scale Error and Redundancy Analysis Simulator

Atishay[1], Ankit Gupta[1], Rashmi Sonawat[1], Helik Kanti Thacker[1], Prasanth B[2]

[1] DRAM Solutions, Samsung Semiconductor India Research and Development
{atishay.1,ankit.g2,rashmi.s,h.thacker}@samsung.com
[2] Host Software, Samsung Semiconductor India Research and Development
prasanth.b@samsung.com

Abstract. Manufacturing a DRAM chip involves multiple steps. External impurities, faulty deposition, or manufacturing errors in any of these steps could generate chips with faulty memory cells, rendering the chip unusable. To overcome these faulty memory cells, redundancies are included in the memory, allowing mapping of faulty memory cells to these redundant cells. The process of mapping faulty cells to redundant cells is called Redundancy Analysis (RA). Different RA algorithms have been developed and are often tested on randomly generated defect to test their efficiency and execution time. But we observed that, the defect pattern of a chip is not completely random, it follows a distribution pattern and the algorithms should be tested on chips with similar error distribution patterns. So, in this paper, we propose a Statistical Wafer Scale Error and Redundancy Analysis Simulator to generate defects on the chips similar to defects on the manufacturing line. The simulated errors on the chips are based on statistical models derived from real data. After generating defects on the chip, execution, comparison and benchmarking of algorithms based on yield and execution time is done. The simulator gives insights on algorithm behavior with different kinds of memory architectures and defect patterns. This allows designers of memory architecture and RA algorithm to simulate, predict and improve the wafer yield for different RA algorithm designs and memory architectures before manufacturing a new memory device.

Keywords: redundancy analysis algorithm, defect simulation, error analysis, statistical modeling, wafer simulation.

1 Introduction

Manufacturers have been able to keep up with the increasing semiconductor demands by producing them in large quantities on a single wafer. During manufacturing of the wafer, there might be surface flaws due to different external conditions. Since these flaws cannot be avoided, the wafer ends up with some defective chips. The increase in memory densities and the decrease in node sizes have led to an increased probability of chip defects. These factors reduce the wafer yield. Along with increasing the quality of production of wafers, several measures have been taken to improve the overall yield.

adfa, p. 1, 2011.

Beginning with 64Kbit generation of DRAM chips, manufacturers have included redundancies in the chips to repair them. Different operations are performed on the chip in different phases of manufacturing. Using wafer tests, the exact address of the defect can be located. This defect address is used to perform memory repair in the Laser Repair step while manufacturing. The process of memory repair is called Redundancy Analysis (RA). RA is a process of allocating spare rows and columns to the defective address detected in the chip, as illustrated in Figure 1.

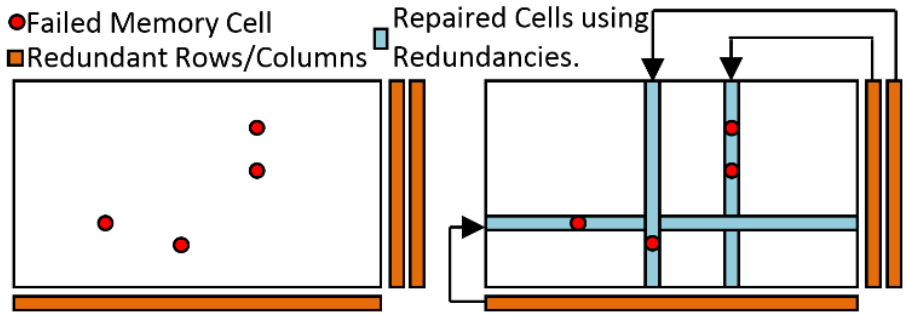

Fig. 1. Memory repair using redundant rows and columns

To a certain extent RA allows the repair of defective chips, but if the chip is unrepairable due to a large number of errors on, it would be discarded. Adding redundancies on the chip and allocating them to the defective addresses improves the yield but, the process of allocation of spares or Redundancy Analysis is an NP-complete problem [1]. So, with an increasing number of chip errors or problem size, time required to repair the chip or solve the problem using known algorithms increases exponentially. This means that only when time complexity of the algorithm is high, maximum yield can be achieved.

Some of the proposed RA algorithms have been discussed in Section 2, but most of these algorithms have been tested on errors that have been randomly generated, or generated based on Binomial or Polya-Eggenberger distribution. But these random defects or these distributions do not represent the defect on the chip in a wafer on the manufacturing line. These defects depend on the wafer lot, the wafer, position of the chip on the wafer and many other factors. All these dependencies have been explored in Section 3 along with the results of the same.

If the existing algorithms are benchmarked on random error distributions which don't take into account these factors, the yield and run time approximations are not similar to that on the manufacturing line. So, in this paper we have proposed a Statistical Wafer Scale Error And Redundancy analysis Simulator which uses statistical models for different stages of device manufacturing to simulate chip error. It considers various factors which affect the defect distribution that allows generation of chips with error distribution similar to the actual data. It also allows to run different RA algorithms on the simulated data to provide insights on algorithm behavior with respect to various factors described in the paper.

The RA algorithm behavior insights provided by the simulator would help in designing new RA algorithms and improving the memory architecture design. The behavior insights include timing, efficiency and spare allocation of different algorithms. Wafer efficiency comparison allows analysis of yield with respect to different sizes of redundancies. The simulator would allow improvement in algorithms and memory design before the chip is manufactured which would improve the wafer yield.

The outline of the paper is as follows. Section 2 gives an explanation of Redundancy Analysis and existing Simulators and their features. Section 3 gives the detailed explanation of the statistical models used in the simulator. Section 4 describes the implementation of the simulator. Finally, Section 5 gives details about the experimental setup and the results are described in Section 6. In Section 7 conclusions are drawn from these results.

2 Background

2.1 Redundancy Analysis Algorithms

An ideal Redundancy Analysis (RA) Algorithm should find a repair solution whenever one exists, execute in a reasonable length of time and abort at the earliest sign of unreparability. The repair rate of an RA algorithm determines its ability to obtain a correct repair solution. The definitions of the repair rate and the normalized repair rate [1] are as follows:

$$R_W = {}^{C_{Repaired}}\Big/_{C_{Total}} \tag{1}$$

$$R_{NW} = {}^{C_{Repaired}}\Big/_{C_{Repairable}} \tag{2}$$

Where R_w is defined as the repair rate of the wafer, $C_{Repaired}$ is the number of chips repaired and, C_{Total} is the number of chips on the wafer. R_{NW} is defined as the normalized repair rate and $C_{Repairable}$ is the number of theoretically repairable chips. The total number of tested chips includes the number of chips that are unrepairable. These chips degrade the effectiveness of the RA algorithm, causing it to have a low repair rate. However, the normalized repair rate is independent of these unrepairable chips, thus making it a more appropriate for estimating the ability of an RA algorithm to obtain a correct repair solution.

Exhaustive search algorithms are required for determining if the chips are theoretically repairable. But the memory repair problem also can be solved using heuristic algorithms. Proposed heuristic algorithms like Repair-Most [2] and OSP [3] can find a solution quickly, but they may not find a solution of a theoretically repairable chip. So, the heuristic algorithms may not achieve an optimal repair rate. Whereas exhaustive search algorithms like Branch-and-Bound [1], PAGEB [1], Fault-driven [4] will certainly reach the optimal repair rate, but the time and space complexity of these algorithms grow exponentially with the number of errors. Therefore, the RA algorithm for chip repair must be chosen carefully as they provide varying yield and time complexity.

RA algorithms are generally divided into the must-repair and the final-repair phase. The must repair phase repairs the faults when there is no choice between invoking a redundant row or a column. If there are more errors in a row than spare columns, it must be repaired using a spare row. The final-repair phase analyses the remaining faults for reparability.

This paper uses the existing heuristic and exhaustive algorithms to test the repair rate and time. The time complexities of the discussed algorithms have been listed in Table 1. The Broadside Algorithm [2] is a heuristic algorithm which uses a greedy approach to perform a repair. It assigns a spare row or column based on whichever is present in excess when it repairs an error. In case of same number of spare row and column, assignment is based on the algorithm design and device requirement.

In [4] an exponential Fault Driven Comprehensive algorithm is defined. It uses a full solution tree to try all possible combinations of spare row and columns to repair an error and generates all possible solutions. In [5] a Faulty Line Covering Algorithm (FLCA) is defined. This is based on the principal that a faulty row with k faults can be covered either by a spare row or k spare columns. This eliminates branches with parents as faults which have already been repaired. Hence it is an improvement over the naive fault driven algorithm.

In [5] a Largest Effective Coefficient Algorithm (LECA) is defined. This heuristic algorithm uses Effective Coefficients (EC) to rank the rows and column of a chip in the order in which they have to be repaired. The EC considers both fault counters and complements of a faulty line.

In [3] a One Side Pivot algorithm (OSP) is defined. It uses Pivot fault properties to find repair priorities reducing the analysis time even when the error rate is high. Faults are classified into 3 types of faults Pivot faults, intersection faults and OSP faults. Where pivots fault is defined as a fault in a faulty line which is not included in any other faulty lines, an intersection fault is defined as a fault which is included in both faulty column or faulty row and one side pivot (OSP) fault is defined as a pivot fault which is not included in a faulty line which does not have an intersection fault. Row OSP faults (pivot in its column) will be solved using spare rows and column OSP faults (pivot in its row) will be solved using spare column. To repair pivot faults, if fault is pivot in its row it is solved using spare column and if fault is pivot in its column then it is solved using spare row.

Table 1. Algorithms time Simulation Parameters

Algorithm	Time Complexity	Remarks
Broadside Algorithm	$O(n)$	n is number of errors
FLCA	$O\left(2^{\left(\frac{T_F - S_F}{min}+1\right)} - 1\right)$	T_F is number of total faults and S_F is number of single faults
LECA	$O(\max\{\mathrm{R_A}, C_A\}^2 \log\max\{\mathrm{R_A}, \mathrm{C_A}\})$	$\mathrm{R_A}$ is redundant row and $\mathrm{C_A}$ is redundant columns
OSP	$O(\max(n, n_P.n))$	n_P is number of pivot fault

2.2 DRAM Manufacturing and Architecture

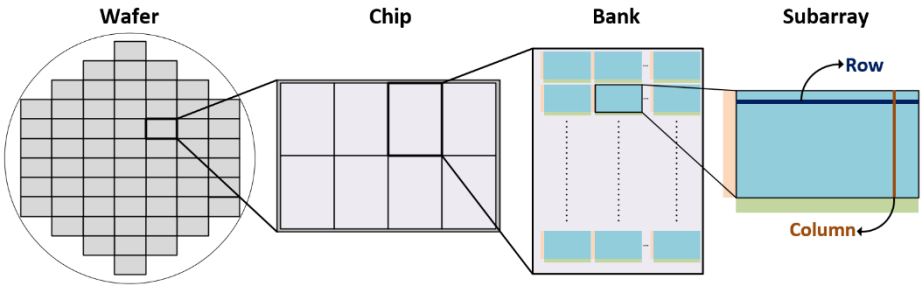

Fig. 2. Wafer, Chip, Bank and Subarray Arrangement in a DRAM Chip

In Figure 2 the relationship between the wafer, chip, bank and the subarrays has been explained. During the manufacturing process, multiple chips are manufactured on a circular wafer. Each chip contains multiple banks which is an independent two dimensional array of storage cells. In Figure 2 each chip has 8 banks associated with it. Only one bank is accessed in a single read or write operation. Starting with DDR4 memory, banks are organized into bank groups for continuous data rate scaling without necessitating longer burst lengths.

These banks are not monolithic structures, they are divided into subarrays. Read and Writes to the DRAM are performed through row activation command which when issued, a group of storage cells gets activated in parallel. Such a group of cells is called a row, often referred to as a page. The storage cells store the binary information in the form of a charge stored on a capacitor. A row activation command caches a row or a page of memory in the sense amplifier. But the whole row is not read at a time, column or the bit line is the smallest addressable unit of memory. Row decoder extracts the row number from the address and activates the corresponding word line, and the sense amplifier compares the voltage of the bit lines to V_{ref} and thus accesses an entire word line amplifying the voltages stored in the memory cells. Then the column decoder which is a multiplexer is used to access particular bit lines.

2.3 Existing Simulators

A literature survey of some existing RA simulators has been described in this subsection. In [6] a simulator for evaluating RA Algorithm which can calculate repair rate and allow user to manipulate memory configuration has been proposed. It uses a single Poisson, Gamma or Negative binomial distribution to find defect per dies and has fixed row, column, cluster and single fault probability. The simulator also incorporates the overhead of spare elements for evaluation of algorithms which is very useful for the user to find the better configurations of the spare elements under a certain redundancy structure and area constraint. The tool described in [6] allows the user to develop the built-in redundancy analysis (BIRA) algorithms and circuits for built-in self-repair (BISR) of embedded memories. This work has been continued in Raisin [7] where the RTL memory behavioral model are tested with the BISR circuit.

An improved version called Raisin was proposed in [7], which also allowed development of Built-In Redundancy Analysis (BIRA) algorithms and introduced a fault detection sequence. But with the increasing DRAM sizes the DRAM memory is not implemented as a monolithic block [11], we need multiple distributions to accurately simulate the error conditions of current DRAM architectures. Adding a few more parameters like the row and column error distributions as suggested in this paper would allow more realistic results.

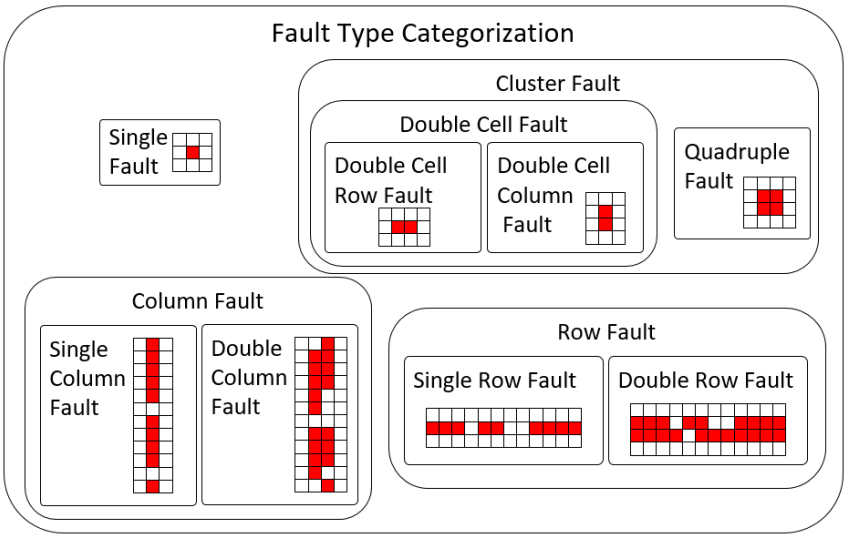

Fig. 3. Categorization of Fault types used in [8]

In [8] a bank model based simulator is proposed which allows creation of realistic faults in banks. It introduces errors in the bank based on categorization of memory device such as fault-free, faulty, theoretically repairable, and repairable memories. It also, introduces various fault types as shown in Figure 3, derived from defects that are generated statistically. It allows evaluation of RA efficiency with the help of the RA evaluation tool proposed in the paper. But the memory device categorization in [8] does not involve wafer lot distributions and wafer specific errors.

In [9] various defect models have been described. Fault analysis, combinations and distributions are used to generate faults in the wafer. Chip defects are then simulated to perform fault analysis like shorts, opens, new gate material device and open devices. Although the simulator takes into account the wafer lot and wafer simulation details, it cannot take into account newer DRAM architectures and specific chip errors.

The current simulators are either confined to a chip which hampers their ability to consider errors in the silicon wafer or do not take into account changes due to evolving DRAM architecture. Chip based simulation is one of the major drawback of the existing simulators. To the best of our knowledge, a statistical simulator with the ability to perform wafer level defect injection and chip fitting providing insights on algorithm performance has not yet been proposed.

3 Statistical Error Model

In this section we will discuss the statistical models used to determine the defects generated on the chip. These models are used to inject errors described in Section IV.

3.1 Wafer Lot Distribution

The manufacturing of the wafers involves cutting and polishing silicon ingots into wafers. These wafers are divided into wafer lots. Multiple distribution of errors are observed in these wafer lots because they were manufactured at different time and external conditions. So the error in a wafer is determined by wafer lot and the wafer distribution. An example of the distribution of error in a wafer lot is illustrated in Figure 4.

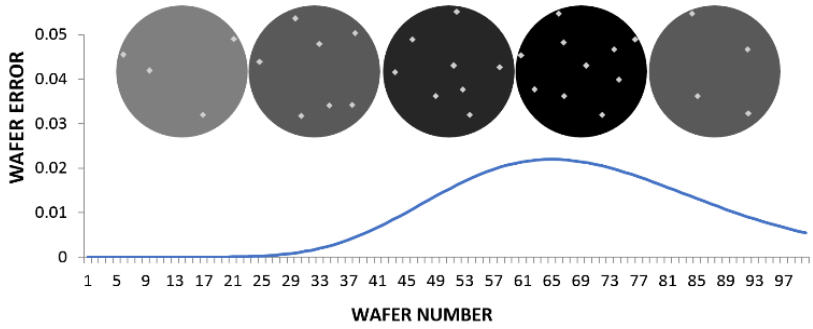

Fig. 4. Example of distribution of errors in a wafer lot

In Figure 4, the following negative binomial distribution has been used for wafer error distribution:

$$Pr_{WL}(x = k) = \binom{\alpha+k-1}{\alpha-1}\left(\frac{n}{\mu}\right)^{k}\left(1-\frac{n}{\mu}\right)^{n-k} \quad (3)$$

Where Pr_{WL} is defined as the defect probability among wafer lots, k is the wafer lot number, n is number of wafer lots and, μ is mean of the negative binomial distribution. Defect distribution probability among wafers (Pr_{WF}) also follows the probability distribution described in equation (3). We use a two level negative binomial distribution similar to [9], for distribution of errors across wafers and wafer lots. The first level distribution allows the modelling of variance of defects among lots and the second level depicts the variance between wafers in a particular lot.

3.2 Chip Distribution

The two level negative binomial distribution used in the previous step allows us to determine the quality of wafer. Using Pr_{WL} and Pr_{WF} the wafer lot error and wafer level error are determined. Multiple chips are manufactured on a single wafer but the distribution of errors among the chips is not random.

In our experimental data, it is observed that chips near the edge of the wafer had a higher rate of failure than chips at the center. This is consistent with the Edge Effect observed in [9] and [10]. Because of different error distributions at the center and circumference of the wafer, it is difficult to fit a single distribution over the whole wafer.

So, for error distribution on chips, a Bimodel distribution has been used which was verified with Hypothesis testing of the sample data. An average p-value of 0.833 was obtained. A Bimodel distribution is observed for errors on the wafer. This distribution consists of a binomial distribution with mean at radial center and Poisson distribution at circumference of wafer with mean greater than radius of wafer.

$$Pr_A(x = k) = \binom{n}{k}\left(\frac{n}{\mu}\right)^k \left(1 - \frac{n}{\mu}\right)^{n-k} + \beta \cdot \frac{\lambda^k e^{-\lambda}}{k!} \tag{4}$$

Where Pr_A is defined as the defect probability at length k, n is normalized radial distance, μ is mean of binomial distribution, λ is mean of Poisson distribution and β is a constant, directly proportional to λ. The λ value chosen for the Poisson distribution has to be greater than n for a correct fit.

An example of the distribution described in equation (4) has been illustrated in Figure 5. In this example the radial distance has been divided into 20 divisions, so each division represents error distribution in 5% of the radial length. From equation (4) the values used are n is 20, μ is 0.6, λ is 32 and β is 15. The radial distribution has been mirrored to represent the distribution of error on the wafer diameter. The first division, represented by R1 has a probability distribution of 0.27, calculated by keeping k value as 20. R2 represents the division with the probability distribution of 0.015 corresponding to k = 14. Similarly division R3 represents the lowest probability distribution of 4.3 x 10-6 corresponding to k = 7. R4 in the center of the chip has a probability distribution value of 0.15 corresponding to k = 0.

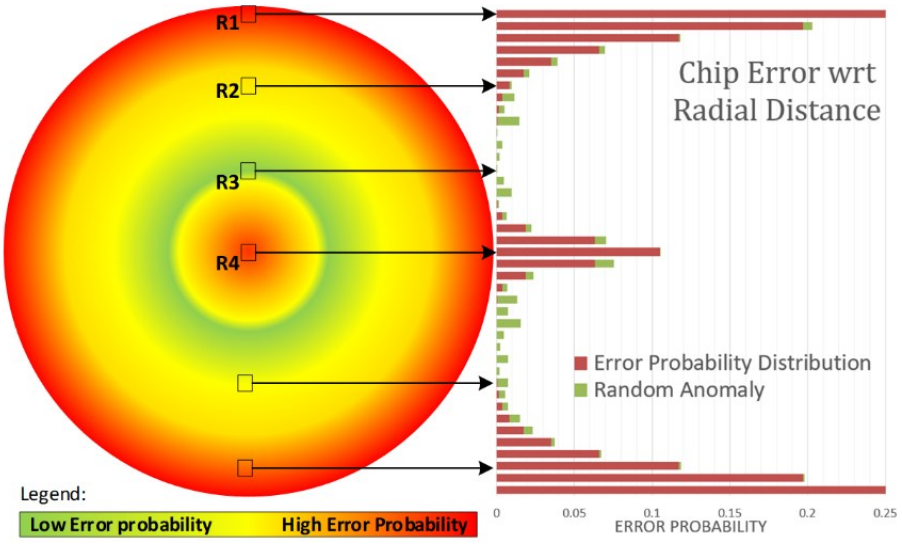

Fig. 5. Wafer Error Distribution example

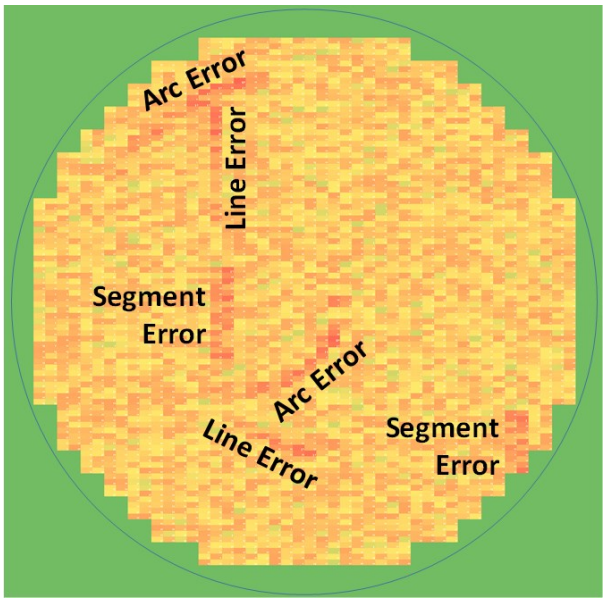

Fig. 6. Specific Errors on a Wafer

Other than the radial distribution, the wafer also experiences some specific faults. Specific errors consist of arc error, line error, and segment errors. Each of them have high error densities and an independent distribution. An example with all the 3 specific errors on a small wafer has been illustrated in Figure 6. In the simulator Poisson distributions are used for specific errors which allows errors in chips lying on the line, arc or segment to have different error rates depending in the distribution parameters.

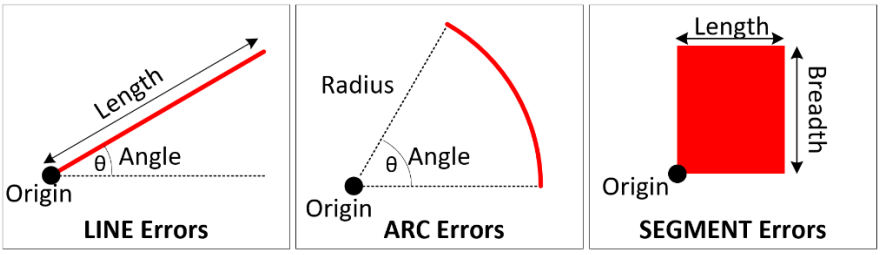

Fig. 7. Parameters used to determine specific errors simulated on a single wafer

As illustrated in figure 7, different parameters are used to determine the location, orientation and size of the specific errors on the wafer. For a Line Error, the origin coordinate, the angle of the line with the horizontal and the length of the line is used. For arc type error the Radius origin and the arc angle is used to determine arc errors. Segment errors are defined with an origin coordinate, length and a breadth. The parameters are defined in terms of Wafer Radius. E.g.: 0.2 x Wafer Radius. Origin is defined as (x, y) coordinates and the Angle is defined in radians.

3.3 Subarray specific Distribution

After the distribution of errors on the wafer has been decided, the quality of the chip has to be determined. This means that the number of errors on the chip is known but, on the chip, these errors are not randomly distributed.

A chip is divided into banks which are further divided into subarrays [11]. The DRAM bank is not implemented as a monolithic structure because it would require very long bit-lines and word-lines. The huge parasitic capacity of these bit-lines would severely affect the access latency. Due to this the modern DRAM bank is divided into a 2-D array of tiles [11]. An example of the tile division is shown in Figure 8.

There is a main row decoder which drives multiple sub word line decoders which in turn drive the local memory arrays. Extra decoders might be present in between the main row decoder and the sub word line decoder depending on the architecture. In the architecture we also observe sense amplifiers which are local to each subarray. And we also have a global column decoder which allows selection of columns to be read from the memory. Multiple such decoders also might be present between these local sense amplifiers and the column decoders depending on the architecture.

In the experimental data, we observed that although the error distribution on the chip could be expressed with a Poisson equation, there were some anomalies. When these error distributions were observed, peaks in the number of errors were discovered in both row and column error count. The peak in the values were observed at multiples of the tile size. The faults on the wafer can also be in the area of the row decoder, column decoder or the sense amplifier as shown in Figure 8. This would render the whole row or column in that tile unusable and would return incorrect values while testing.

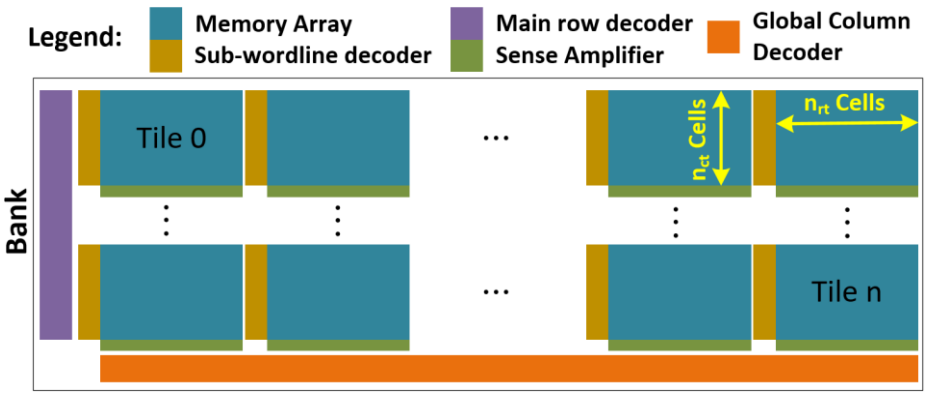

Fig. 8. Division of Bank into Tiles

In the examples of distribution of errors explained below, n_{rt} is number of cells in a row per tile and n_{ct} is number of cells in column per tile. We assume that the tiles are numbered from Tile 0 to Tile n for the simplicity in the explanation of the example.

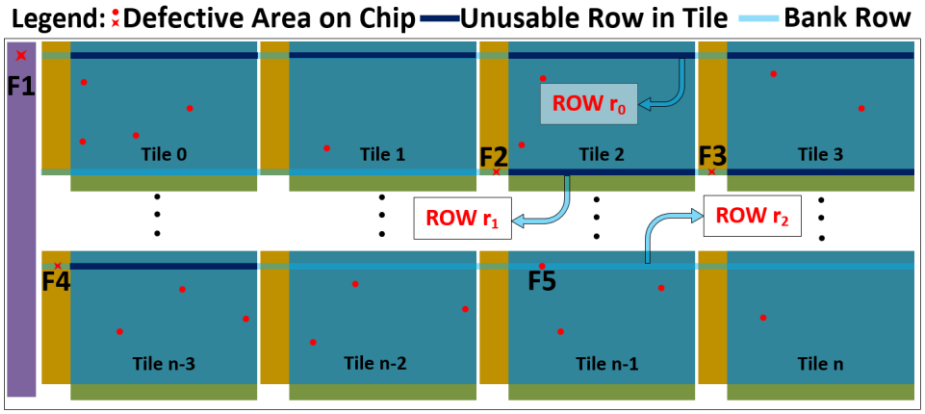

Fig. 9. Row error due to defective decoder and local row buffer

Figure 9 represents a bank with errors in different areas. Faults F_1, F_2 and F_3 are present at sub word-line decoder side which render the whole row in a tile useless. Therefore, the row error count for row r_1 will be number of cells in a row of tile 2 and tile 3 i.e., $2 \cdot n_{rt}$. Similarly row error count for row r_2 will be the sum of errors in tile n-3 due to F_3 and the cell error F_4 in tile n-1 i.e., $n_{rt} + 1$.

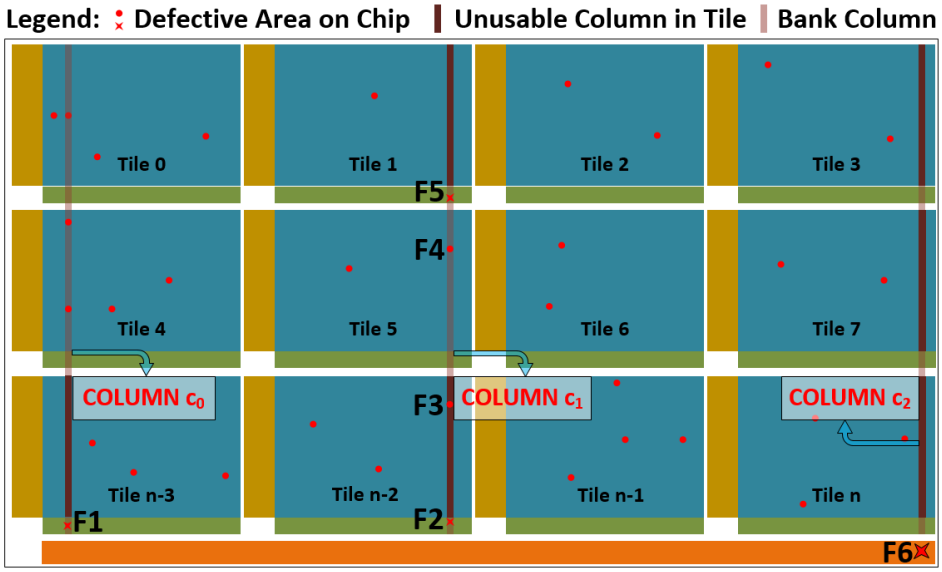

Fig. 10. Column error due to defective sense amplifier and global column decoder

Similarly column errors as represented in Figure 10, can occur when the error on the chip is present in the local sense amplifier (F1, F2 and F5) or the row buffer (F6). Number of column errors on such a fault can be similarly calculated as $n_{ct} + 3$, $2 \cdot n_{ct} + 1$ and $3 \cdot n_{ct}$ for column c_0, c_1 and c_2 respectively.

Due to these faults, the number of row and column faults will have anomalies at $(r \cdot n_{rt}) \pm x$ and $(c \cdot n_{ct}) \pm y$ Where r and c lie between 1 and the number of tiles in a row or column of the 2D array of tiles in the bank. To the best of our knowledge, this is the first RA simulator to include tile size specific faults in the DRAM chip. This anomaly in the distribution was corrected by adding a binomial distribution with a multiplier and its mean value as the tile size to the original Poisson distribution of row and column errors in the DRAM chip. Multiple such distributions with different multiplier values can be added to the original Poisson distribution with mean values as a multiple of tile size to accurately depict the row and column error distribution.

$$Pr_{Lm}(x = k) = \frac{\lambda^k e^{-\lambda}}{k!} + \sum_{i=1}^{i=\frac{a_n}{t_n}} \gamma_i \binom{n}{k} \left(\frac{n}{\mu_i}\right)^k \left(1 - \frac{n}{\mu_i}\right)^{n-k} \quad (5)$$

Where Pr_{Lm} is defined as the defect probability of line having k faults, λ is mean of Poisson distribution, a_n is number of cells in the array, t_n is the number of cells in a tile, γ_i is damping factor of ith binomial distribution, n is tile size and μ_i is mean of i^{th} binomial distribution. Pr_{Lm} can be the defect probability of a row or a column (Pr_{LR} or Pr_{LC}).

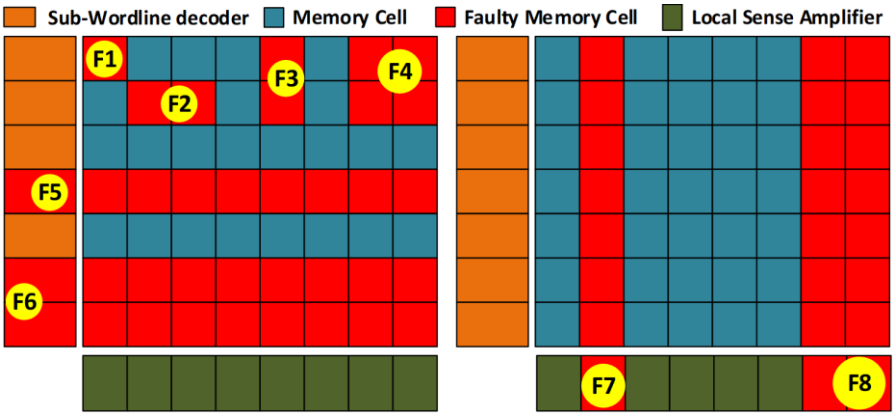

Fig. 11. Fault types in a DRAM tile

The probability distribution described in equation (5) allows distribution of errors on the chip in form of row and column errors in each tile. In [8], different faults occurrences have been considered at bank level, but in the simulator, faults are also considered on a tile level. Faults can occur in the DRAM due to a manufacturing defect. Faults are the reason for occurrence of errors in the memory. As illustrated in Figure 11, the *F1* fault only introduces a single cell error in the memory array. Faults *F2* and *F3* due to their occurrence on the boundary of 2 cells, cause 2 adjacent errors. Fault *F4*, a fault of a larger size might cause an error in 4 adjacent cells. Faults *F5* and *F6*, occur in the sub-word line decoder, which causes an error in a single row or 2 adjacent rows. Similarly, faults *F7* and *F8* in the local sense amplifier might cause a single column or 2 adjacent columns to be faulty. The fault types are also a part of the error model which is incorporated into the simulator with the help of the Defect Injector described in Section 4.

3.4 Final Distribution

The final probability of error in the chip is calculated using the probability values of Pr_{WL}, Pr_{WF} and Pr_{A} which are the probability of error in a wafer lot, a wafer and a region of the wafer respectively.

These 3 values determine the probability of an error at a particular location in a wafer which decides the error in the chip that is manufactured at that location. This error probability needs to be distributed among the banks in that chip as row errors, column errors and other faults. Pr_{RN} and Pr_{CN} is the probability of occurrence of a particular number of row errors in a row or column errors in a column of a bank.

After these probabilities are decided, similar to [8] the row and column errors are divided into single and double row errors and single and double column errors respectively. Rest of the errors are divided into single, double and quad errors.

4 Implementation

The overall architecture of the simulator is described in Figure 12. Simulator parameters can be input manually or can be derived from fitting the error distributions on processed chip errors. Change in the wafer or chip size changes the distribution parameters for the models in the Defect Injector. The distribution parameters are used by the defect injector module to define errors in a particular region of the wafer.

The chips are fit onto the wafer depending on the wafer and the chip size. Once the chip fitting is done, the defect injector module injects error in these chips by fitting the distribution parameters onto the equations described in Section 3. The defect injector logs the defects in the bank for future use and debugging, and the RA simulation module outputs simulation results for further analysis as described in Section 6. A detailed description of the chip fitting module and defect injector has been done in this section.

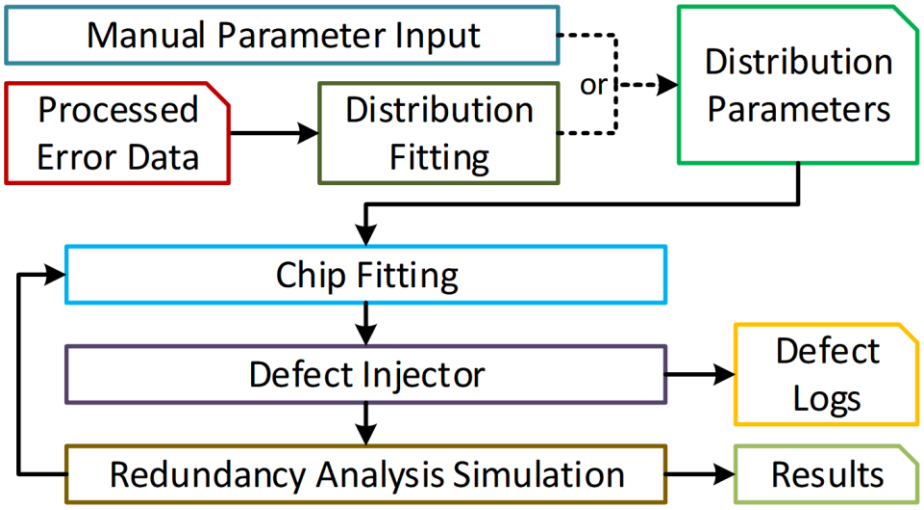

Fig. 12. Simulator block diagram

4.1 Chip Fitting

The Chip Fitting module takes into consideration parameters like wafer size and chip size to iteratively fit different memory architectures in the wafer. Defect injection is performed and RA algorithms are applied on these chips while measuring the repair rate. This results in a wafer efficiency peak with respect to different memory architectures as illustrated in Figure 13.

For a given memory configuration and RA Algorithm, as the number of spares increases, the repair rate and the chip area also increases. Increase in chip area means a decrease in the total number of chips on the wafer. So in this paper, a solution for finding an optimized memory architecture that accounts for the repair rate and the number of chips on the wafer has been proposed.

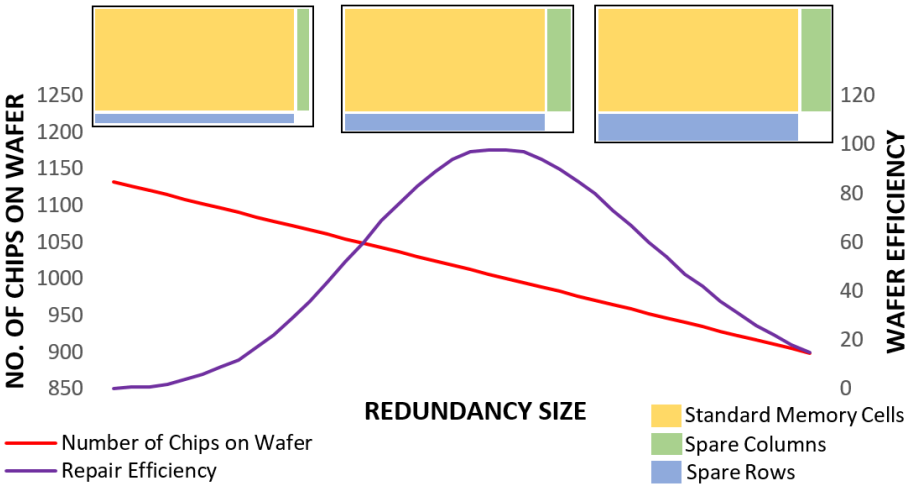

Fig. 13. Number of Chips and Wafer Efficiency with respect to the Redundancy Size

The chip fitting module uses the bank and chip configuration to find out the area occupied by each chip. Chips are fit side by side optimally on the wafer taking into consideration the radius of the wafer. Figure 14 illustrates how different chip sizes lead to difference in the number of chips on the wafer. In the example illustrated in the figure, a steady decrease in the number of chips with increasing redundancies is observed. But the number of repairable chips on the wafer is not maximum when the number of chips are maximum, rather when the number of redundancies are an optimal number the number of usable chips on a wafer is maximum.

Various user adjustable mechanisms for chip fitting were introduced in the simulator to maximize the number of chips that could fit in a wafer. Initially a fixed coordinate was used to determine the starting position of the chip, but it would lead to inefficient chip placement on the wafer. Depending on the chip configuration i.e.: the number of rows, columns, spare rows and spare columns, starting coordinate of the top left chip is determined and the rest of the chips are arranged accordingly to maximize the number of chips on the wafer.

The same bank and chip configuration have been used in both the wafers but the number of spare rows and columns have been increased in (A). The specifications considered for the illustration are a chip with 8 banks in a 4x2 configuration. Each of the bank is 256x128 bits. The wafer on the left (A) has 8 spare rows and 4 spare columns with a total of 24 chips on the wafer. Even though the wafer size is small, a considerable decrease of 4 chips in the number of chips on the wafer was observed. The wafer on the right (B) has 20 chips as it has 32 and 16 spare rows and columns respectively. Although wafer (B) might have a higher yield, but it is possible that wafer (A) might have more usable chips even with lower yield.

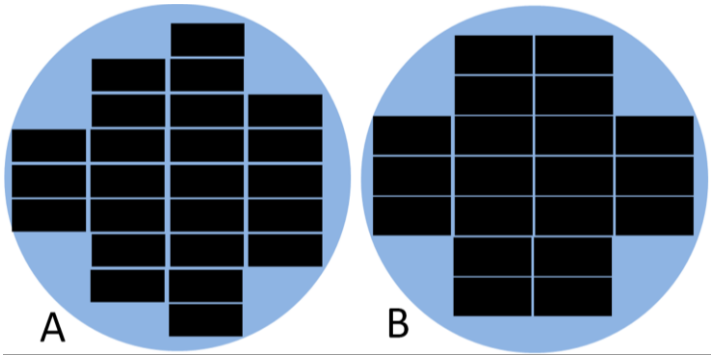

Fig. 14. Illustration of Chip Fitting on a Wafer

4.2 Defect Injector

The Defect Injector uses the distributions described in Section III to inject errors into chips as shown in Figure 15.

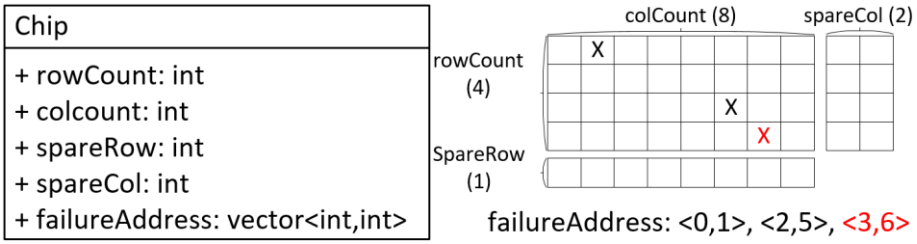

Fig. 15. Chip implementation in the Defect Injector

The Chip object in the Defect Injector contains the row, column count and the number of spare rows and columns in that chip. To store the errors, a *failureAddress* vector is used which stores the failed bit addresses in form of a <row, column> pair. When a new error is generated with the help of the distributions in Section III, it is added to the failed bit address vector of the chip object. As shown in Figure 15, when a chip is created, the *failureAddress* vector is empty. For example, when a 3rd error is injected in row 3 and column 6 represented as a red X, a <3, 6> pair is added to the vector.

Due to memory constraints, the chip objects are created one at a time and tested with different RA algorithms sequentially during the simulation. After testing with RA algorithms and logging the necessary parameters, this chip object is deleted. The probability distributions used to determine the wafer lot, wafer and chip errors are stored in maps for faster access and probability value insertion.

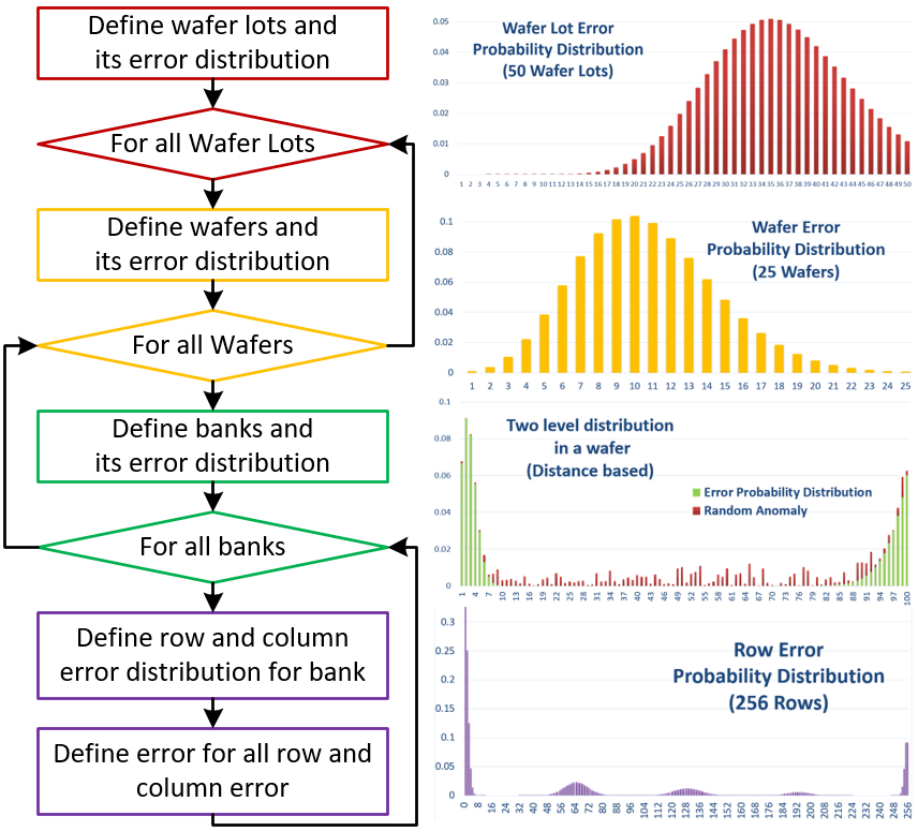

Fig. 16. Working of the Defect Injector

The Defect Injector as shown in Figure 16 is divided into four distributions as described in Section III. The Defect Injector begins by defining the error distribution in between the wafer lots. In the example shown in Figure 16, the error is distributed among 50 wafer lots using the distribution from equation (3) with x as wafer lot number, α as 50, μ as 103 and n as 60. In the second step, for each of the wafer lots, errors are distributed among the wafers. In this example, 25 wafers were generated using the negative binomial distribution from equation (3) with x as the wafer number, α and n as 25 and, μ as 35. The distributions are stored in an errorDistribution map as a <key, value> pair. The keys are the wafer lot or wafer number and the values are their respective error probabilities.

Then the error is divided among the chips which are placed on the wafer. A bimodal distribution which takes into account the Edge Effect [10] is used for radial error distribution of chips on the wafer. Before injecting different chip errors, specific errors are also injected into the wafer which bring in some anomalies in the distribution data. The following values are used in equation (4): n is 150, μ is 8.3x103, β is 60 and λ is 125. The final step of the distribution is to distribute the errors in the chip. The distribution used here is described in equation (5). The λ value used in this equation is 1.5. It is then added to four different binomial distributions as a_n/t_n is 4. Assuming that the tile size in this example is 64 bits, the mean values for the distributions, μ_1, μ_2, μ_3 and μ_4 are 64, 128, 192 and 256 respectively which are multiples of 64. The γ_1, γ_2, γ_3, and γ_4 values used are 0.4, 0.25, 0.1 and 0.25 respectively. This allows addition of defects at row level which takes into account the tile based architecture explained in Section III.

5 Experimental Setup and Results

Multiple algorithms were tested on the simulator. This included an implementation of the Broadside [2], FLCA [5], LECA [5] and OSP [3] algorithm. In the experimental setup subsection we have also discussed the data structures and optimization steps used to implement the algorithms as these impact the yield and algorithm run time results.

5.1 Experimental Setup - Algorithm

The fault address are stored in the fail bit address vector which can be decoded into the row and column addresses. This vector along with the redundancy architecture details are input to the algorithm for finding a solution. As explained in Figure 17, one of the common operations in an algorithm is iterating and decoding the failed bit vector to row and column address, followed by the algorithms steps. Next, depending on the availability of spare rows and columns or by user preferences, spares are assigned to that faulty bit by inserting into the row and column solution vector. This helps to easily compare the performance of the algorithms on a particular chip. All algorithms invoke must-repair and also check for early abort conditions before starting their respective steps.

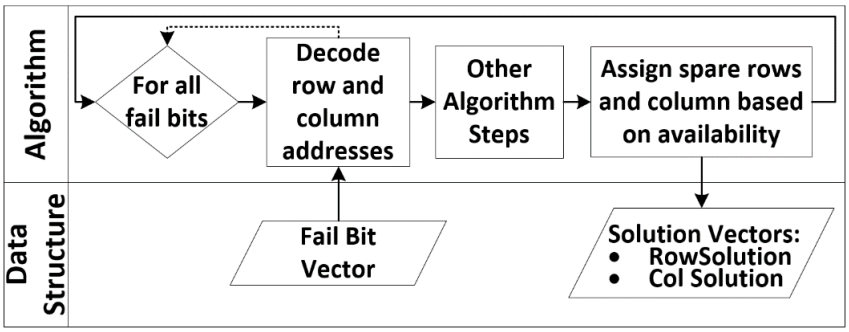

Fig. 17. General implementation of algorithms

The Broadside Algorithm was implemented as described in [2]. For each fault that has not yet been repaired, a spare row or column is assigned depending on whichever is available more at that stage of the algorithm execution.

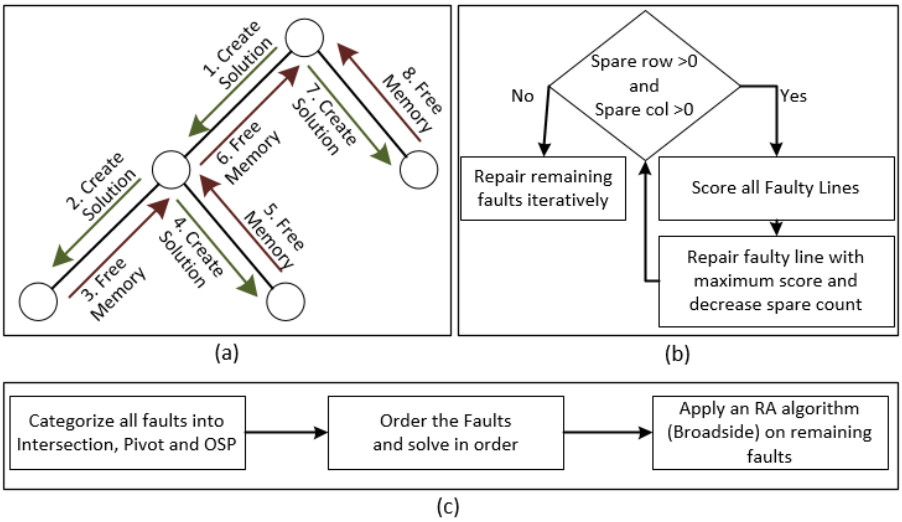

Fig. 18. Implementation details of the FLCA, LECA and the OSP algorithm

Implementation of the FLCA algorithm [5] involves an object-oriented approach where solution node object contains total errors left, spare rows, columns left and the number of single faults. To save memory, as illustrated in Figure 18 (a), Depth First Search (DFS) is used while generating the solution tree so that objects generated can be cleared when not needed. Also, since the single faults can be solved in any manner, they are handled at the very end of the algorithm execution as mentioned in [5]. It is sometimes advantageous to find all the possible solutions for a particular configuration. But in this case since only the yield is important, we stop the execution as soon as one solution is found. This helps in further reducing the amortized time complexity over various chips because all the possibilities in the search space are not explored.

In LECA algorithm [5] implementation (Figure 18 (b)), score maps are maintained. After one spare is allocated, the scores of some faulty lines need to be modified. Only the affected maps are renewed with each pass of the algorithm step while keeping in check the maximum score values. Counter values used to calculate the scores are maintained in vectors for faster access.

In the implementation of the OSP algorithm [3] described in Figure 18 (c), vectors are used to classify and keep track of Pivot, Intersection and OSP faults. This allows quickly solving and removing faults based on priority. After priority based repair, any RA algorithm can be used to repair the remaining faults. Broadside algorithm was selected since it was experimentally observed that the yield improvement for the other algorithms was not justifiable, given the increase in execution time.

5.2 Experimental Setup – Defect Injector

The parameters for different simulations performed have been described in Table 2. In this table, under Wafer Lot and Wafer subsection, n_{WL} is the number of wafer lots and n_W is number of wafers. α and μ can be referred from equation (3). Under chip Specification, r_n and c_n are the total number of rows and columns. sr_n and sc_n are the total number of spare rows and spare columns. n_{rt} and n_{ct} represent the total number of rows and columns in a tile. Under Wafer Area section, N represents the number of chips in a wafer and rest of the parameters are mentioned in equation (4). Row and column error distribution for the subarray section can be referred from equation (5). Here two parameters have been mentioned for row and column distribution. Some parameters in the simulation like μ in the Wafer subsection of simulation S_2 lies within a range of 15 to 20 and is not fixed. This is represented by [15, 20]. Similarly in the subarray section, row and column distributions can have different μ_i values.

Table 2. Simulation Parameters

Simulation		**S_0**	**S_1**	**S_2**	**S_3**	**S_4**
Wafer and Wafer Lot (IIIA)	n_{WL}	-	-	9	7	-
	α_{WL}	-	-	12	20	-
	μ_{WL}	-	-	35	55	-
	n_W	-	-	21	15	-
	α_W	-	-	[10,14]	[10,15]	-
	μ_W	-	-	[15,20]	[25,35]	-
Chip Specification	r_n x c_n	512 x 256	512 x 256 256 x 128	1024 x 512	128 x 64	128 x 64
	sr_n	[16,256]	[4,256] [4,128]	128	16	16
	sc_n	[8,128]	[2,128] [2,64]	64	8	8
	n_{rt} , n_{ct}	512, 256	512, 256 256,128	256, 256	64,32	64,32
Wafer Area (III B)	N		3700	1250	1200	153047
	n		270	41	7	Random Chip Error (mt19937)
	μ		9000	[10,30]	[15,50]	
	β		0	3	[2,3]	
	λ	-	-	20	12	
Subarray (III C)	λ	-	-	1.5	1.5	Random distribution of single Errors (mt19937)
	i	-	-	[1,4],[1,2]	2	
	μ_i	-	-	[256,1024], [256,512]	[64,128], [32,64]	
	γ_i	-	-	[0.1,0.3]	[0.15,0.25]	

In Simulation S_0 and S_1, only a single wafer has been simulated, so the wafer lot and wafer parameters have not been used. Here tile size is assumed to be the same as chip size as sr_n and sc_n sizes are being varied in the simulation. The number of spare rows and columns are varied in the range as described in sr_n and sc_n rows of Table 2. These values are varied in multiples of 2 and the result for the same has been discussed in the next section.

In S_2, the whole wafer lot was simulated, but since the number of errors were very high (up to 5000) FLCA was not implemented as it is an exponential algorithm. This simulation was done for runtime comparisons between OSP, LECA and Broadside algorithms. In S_3, a very small chip size was simulated with the full wafer lot simulation to see the execution time and normalized repair rate using the FLCA algorithms. In S_4 random error on the chip and random single error distribution were simulated. For comparison, the number of chips simulated were the same as simulation S_3. The chip error and the position of chip errors were determined using a Mersenne Twister pseudorandom number generator [12].

5.3 Results

In simulation S_0, a single wafer with different chip configurations was simulated to observe the pattern of usable chips on the wafer with respect to the redundancy size. The simulation was done with the Broadside algorithm, the maximum yield was obtained at 112 spare rows and 8 spare columns, as shown in Figure 19. Similar simulations can be run by chip designers and manufacturers with their respective RA algorithm to predict the number of spares to be included in the new memory architecture. The useful insight gained from this simulation is that just increasing the number of spare rows and columns does not guarantee an improved wafer yield i.e., the number of usable chips obtained per wafer.

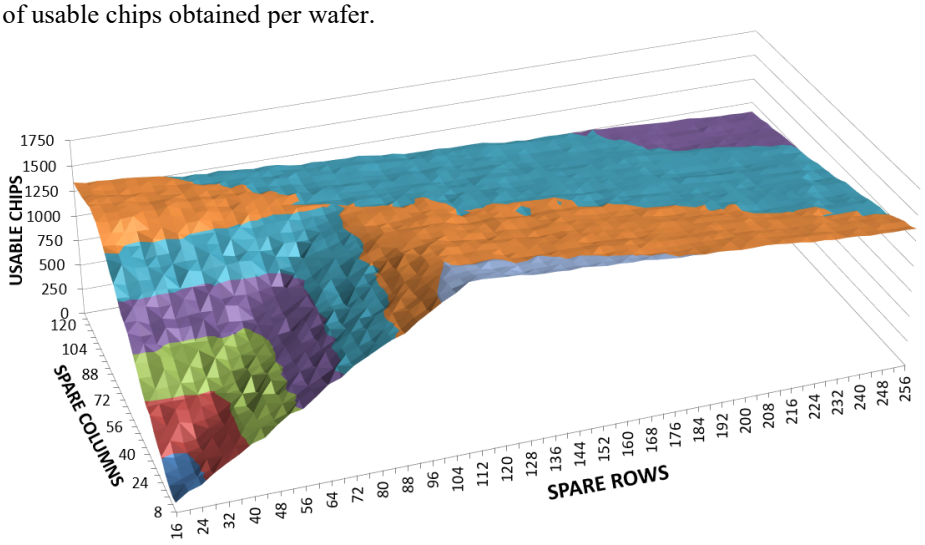

Fig. 19. Usable chips on wafer vs spare size

In Figure 20, results of simulation S_1 with two different executions on a row and column configuration of 512 x 256 (Figure 20 (a)) and 256 x 128 (Figure 20 (b)) was simulated. Difference in the slope to achieve the maximum wafer yield was observed. There was also a difference in the initial yield with low spare columns and rows where in case of a larger chip size, the number of solvable chips were close to 0.

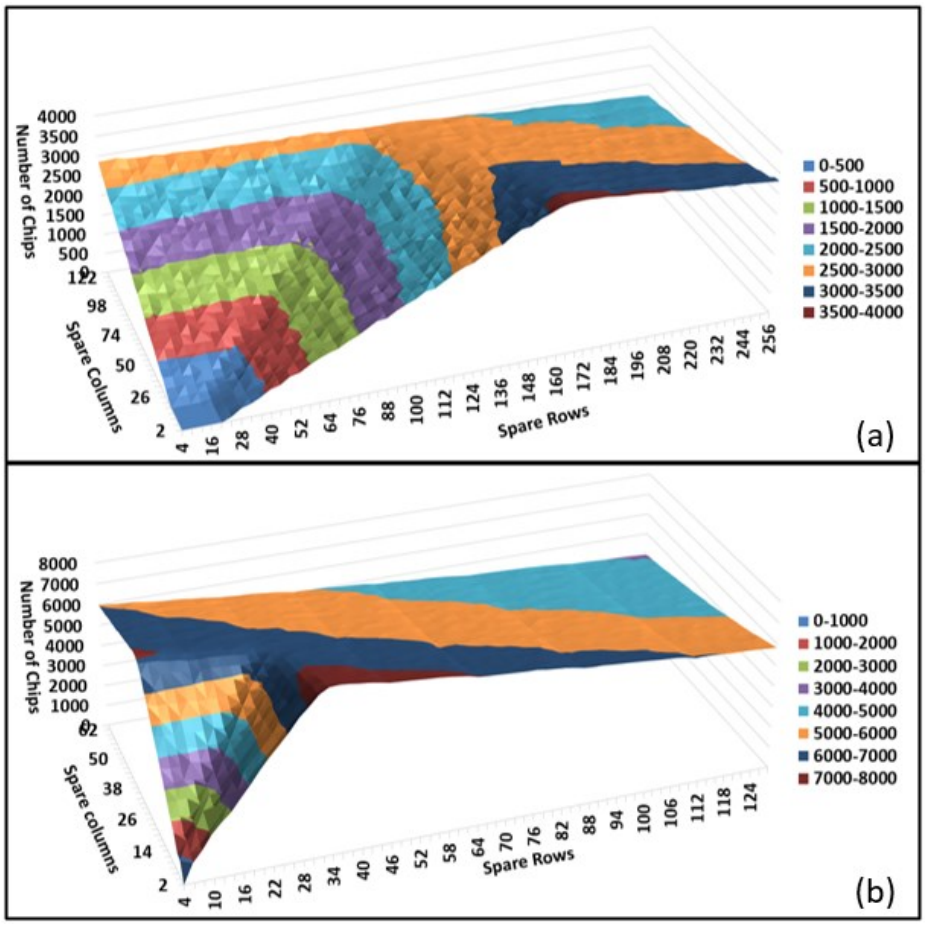

Fig. 20. Variance of Usable Chips on a wafer with configuration

In simulation S_2, the whole wafer lot was simulated and three algorithms were compared. The number of chips repaired and time taken by different algorithms for wafers have been plotted. The average error of the wafers for all wafer lots are also plotted in the lower part of the graph in Figure 21. In S_2, LECA repaired 72.9% and 24.9% more chips than Broadside and OSP algorithms respectively. The Broadside algorithm had a constant execution time and was at least 15 times faster than other algorithms executed. On an average, LECA was 24.5% slower than OSP in the entire simulation.

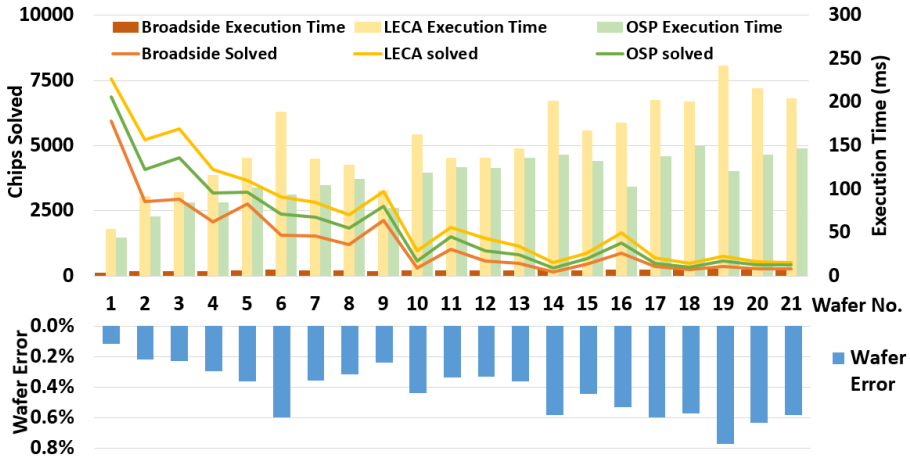

Fig. 21. Algorithm Yield and Wafer Error

In simulation S_3, a small chips size is simulated to observe the normalized yield of the algorithms from equation (2). The Broadside algorithm has a 0% normalized repair rate if chip error is more than 1%, whereas LECA and OSP still maintain a normalized repair rate of ~20% and ~5% respectively. LECA and OSP have a high (>90%) normalized repair rate up to a chip error rate of 0.5%, so these algorithms can be used at a low error rate for determining the chip reparability.

Using S_3 a compound RA solution for a wafer can be designed which would use multiple algorithms, say LECA and OSP. This simulation would help in determining when to use the algorithms. From Figure 22, it can be observed that till a chip error rate of 0.64%, OSP provides better normalized repair rates, but after that LECA outperforms OSP. Also, algorithms like Broadside can be included in the compound RA solution to solve chips below an error rate of 0.34%.

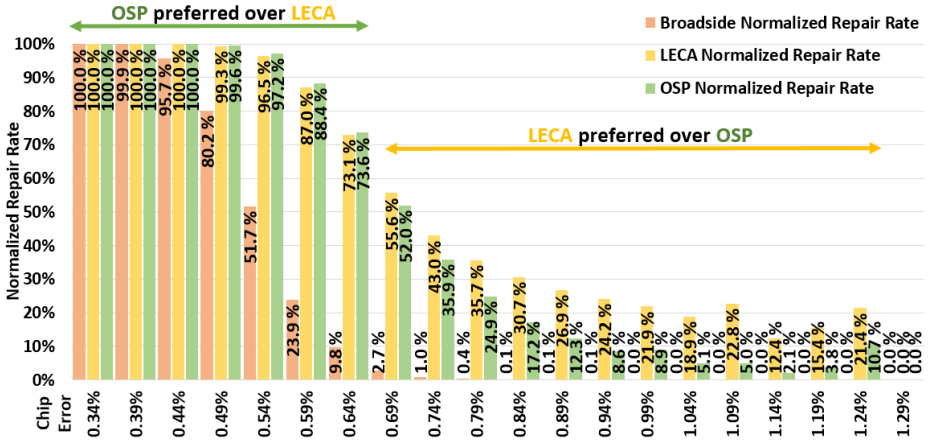

Fig. 22. Normalized Algorithm performance with Chip Error

Figure 23 is a plot of the execution time with respect to the chip error rate. With respect to the error rate, FLCA has an exponential increase in execution time, whereas LECA and OSP have a linear increase in the execution time. A decrease is seen in FLCA execution time after 1.34% chip error because of early abort conditions being satisfied.

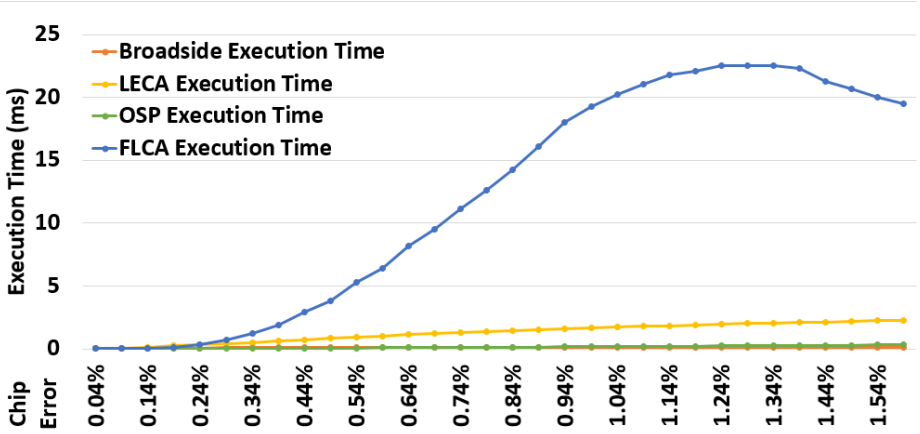

Fig. 23. Execution time with Chip Error

In Figure 24, the simulation results of S_3 are compared with S_4, a random defect injector. In S_4 13.6% of the chips were repairable compared to 59.2% by this simulator in S_3. Not only is the average chip error rate constant in S_4, but also algorithms are not able to solve chips with more than 0.6% error. In S_3, exhaustive and heuristic algorithm can solve chips with a maximum error rate of 1.6% and 1.3% respectively which is much closer to real life errors.

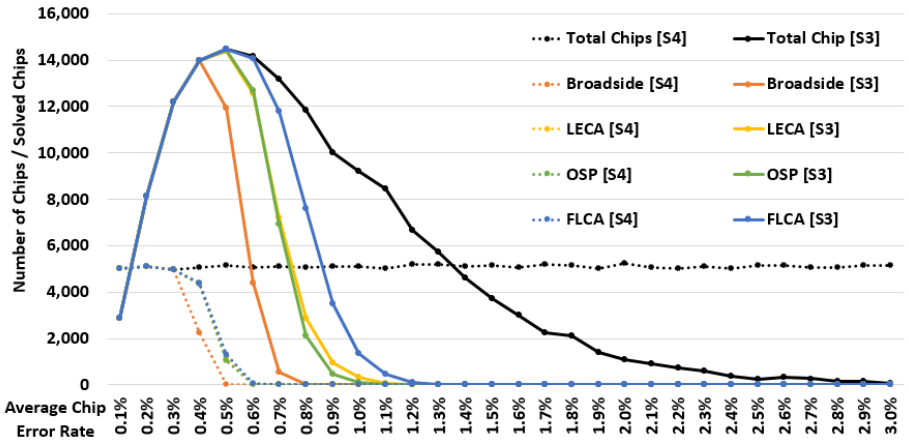

Fig. 24. Comparison of this Simulator [S_3] with Random Defect Injector [S_4]

6 Conclusion

In this paper we presented a Statistical Wafer Scale Error And Redundancy analysis Simulator, a statistical model based approach which provides various insights into RA algorithms. Algorithm designers can use these insights to simulate errors similar to those on the manufacturing line before actually manufacturing the chip and improve their algorithm design. Memory architects would be able to gain insights on the impact of memory design and RA algorithms on wafer yield, which would allow them to make changes in the memory design and improve the yield.

For upcoming DRAM technologies like DDR5 and LPDDR5, with even higher memory densities, the simulator would provide insights which would allow designers of both memory architecture and RA algorithm to simulate, predict and improve the wafer yield before actually manufacturing or finalizing the design of the new memory device.

References

1. S Keewon Cho, Wooheon Kang, Hyungjun Cho, Changwook Lee, and Sungho Kang. 2016. A Survey of Repair Analysis Algorithms for Memories. ACM Computer Survey.
2. M. Tarr, D. Boudreau, R. Murphy, "Defect analysis system speeds test and repair of redundant memories", Electronics, vol. 29, no. 12, pp. 175-179, Jan. 1984.
3. J. Kim, K. Cho, W. Lee and S. Kang, "A new redundancy analysis algorithm using one side pivot," 2014 International SoC Design Conference (ISOCC), Jeju, 2014, pp. 134-135.
4. J. R. Day, "A fault-driven comprehensive redundancy algorithm," IEEE Des. Test Comput., vol. 2, no. 3, pp. 35–44, Jun. 1985.
5. F. Lombardi and W. K. Huang, "Approaches for the repair of VLSI/WSI RRAMs by row/column deletion," International Symposium on Fault-Tolerant Computing., 1988, pp. 342-347
6. R.-F. Huang et al., ''A Simulator for Evaluating Redundancy Analysis Algorithms of Repairable Embedded Memories,'' Proc. IEEE Int'l Workshop Memory Technology, Design and Testing, 2002, pp. 68-73.
7. R.-F. Huang et al., "Raisin: Redundancy analysis algorithm simulation," IEEE Des. Test Compute, pp. 386–396, Jul.–Aug. 2007.
8. K. Yamasaki, S. Hamdioui, Zaid Al-Ars, A. V. Genderen, G. N. Gaydadjiev, "High Quality Simulation Tool Memory Redundancy Algorithms", ProRISC 2008, pp. 133-138, Nov. 2008.
9. H. Walker and S. W. Director, "VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 541-556, 1986.
10. G. O'Donoghue and C. A. Gomez-Uribe, "A Statistical Analysis of the Number of Failing Chips Distribution," in IEEE Transactions on Semiconductor Manufacturing, vol. 21, no. 3, pp. 342-351, Aug. 2008.
11. Y. Kim, V. Seshadri, D. Lee, J. Liu and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," International Symposium on Computer Architecture, 2012, pp. 368-379.
12. Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. 8, 1998, 3-30.
13. Atishay, A. Gupta, R. Sonawat, H. K. Thacker and P. B., "SEARS: A Statistical Error and Redundancy Analysis Simulator," 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), Cuzco, Peru, 2019, pp. 117-122.