Memory repair logic sharing techniques and their impact on yield

Benoit Nadeau-Dostie
Mentor, A Siemens Business
Ottawa, Canada
benoit_nadeau-dostie@mentor.com

Luc Romain
Mentor, A Siemens Business
Ottawa, Canada
luc_romain@mentor.com

2020 IEEE International Test Conference (ITC) | 978-1-7281-9113-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/ITC44778.2020.9325280

***Abstract*— Techniques for sharing memory repair logic amongst memories are described. The techniques allows reducing silicon area and loading time of repair information upon power up. The impact on yield is predicted using two different methods based on defect density and clustering or past silicon experience.**

Keywords*— *Memory repair, silicon area, repair time, yield

I. INTRODUCTION

The amount of memory embedded in integrated circuits continuously increases. In [1], it is reported that as much as 400Mbits had been observed at 28nm and 500Mbits at 16nm. This memory is distributed over thousands and even tens of thousands of memory instances depending on the application. The number of bits per instance does not necessarily increase. It is the number of instances which tends to increase in order to achieve parallel processing. For example, increasing the number of cores in a processor chip or increasing the number of channels of a telecommunications chip.

Memory repair has become essential to achieve sufficient yield on circuits with such high memory content. However, only a small percentage of memories actually need to be repaired on any one chip. This fact allows to significantly compress the memory repair information stored in a Non-Volatile Memory (NVM) as shown in [2] and [3], for example.

In [4], sharing of the repair analysis logic calculating a repair solution for each memory is described to reduce area. However, test time is not reduced because memories sharing the repair analysis logic need to be tested one at a time. Also, there is no reduction of the power up time since the number of repair registers to be loaded is not reduced. Authors in [5] address the first limitation and explain in more detail how the repair analysis logic can be shared for memories of different dimensions and different physical address mappings. This is done by using an "align and merge" logic module allowing the repair analysis logic to calculate a solution in a unified repair format. Again here, the number repair registers is not reduced and the area of the repair analysis logic increases linearly with the number of memories tested in parallel.

The purpose of our work is to further reduce the amount of repair logic and the time required to load repair registers upon power-up. We propose to achieve both goals by merging failure information coming from different memories or segments of memories tested to calculate a common repair solution which can be broadcasted. A method to limit the potential yield loss resulting from using a common repair solution which might not be optimal is also proposed.

Section II describes the hardware modifications required to share the repair logic. Section III analyzes in detail the potential impact on yield followed by some conclusions.

II. REPAIR LOGIC OPTIMIZATION

Each Memory BIST controller is typically shared by several memories. Tens or even hundreds of memories can be tested by a single controller. These memories can be different in term of sizes, port configurations (single/multi-port), physical organization (with/without column multiplexing) and repair method (row-only, column-only, row and column). Access to these memories can be direct or indirect through a shared bus such as in [6]. Memories can be tested in parallel, in series or a combination of the two. All these aspects are considered in our proposed solution for sharing the repair logic.

The proposed method does not require modifying existing partitioning algorithms used to assign memories to controllers. These algorithms are mainly driven by parameters such as power domains, clock domains, memory placement, test time, power levels, and others. The proposed method creates repair logic sharing groups after this initial partitioning. These groups are local to each controller. The groups are automatically defined based on different criteria explained later and a user-defined limit defining the maximum size of each group. For example, a group might consist of at most 100K bits of memory. This could represent several memories, a single memory or a portion of a large memory. The built-in repair analysis (BIRA) module associated to a group must be able to compute a common repair solution even if the group is composed of memories with different characteristics (size, physical organization, pipeline depth, etc…) similar to what the memory BIST controller itself can do. This means that the column address range processed by the BIRA module can be defined by one memory and the row address range defined by a different memory.

A BIRA module allocates at most one unique spare row unit and one unique spare column unit within a group. This means that if a memory has multiple segments, each with their own spare resources, the same repair solution is applied for all segments. A typical example is a memory with two segments, left and right, and each segment has one spare IO which can be allocated independently if repair sharing is not enabled.

978-1-7281-9113-3/20/$31.00 ©2020 IEEE

However, when repair sharing is enabled, the BIRA module is modified to combine the error signals coming from individual IOs to calculate a common repair solution. In this example, the repair inputs are assigned the following codes:

Right segment: IO[0] to IO[31] → 00000 to 11111

Left segment: IO[32] to IO[63] → 11111 to 00000

The error signals of IOs sharing the same code are ORed together before being applied to the input of an encoder. This encoder generates a repair code and detects if errors occurred on multiple IOs in the same clock cycle. A multi-bit error would immediately result in a non-repairable condition because there is no spare row block available in this example. Once a single-bit error is detected, the column repair sub-module stores the repair code. Subsequent errors are compared against the previously allocated repair address and ignored if they match. If not, then the memory is declared non-repairable. Note that a memory could have been repairable if repair sharing was disabled. The potential impact on yield is analyzed in Section III.

A slightly more complex example is shown in Fig. 1 where 4 memories of different dimensions both in terms of number of addresses and number of IOs are tested in 2 steps. Each memory has a single segment. The repair inputs are assigned the following codes:

64-bit memories: IO[0] to IO[63] → 000000 to 111111

32-bit memories: IO[0] to IO[31] → 00000 to 11111

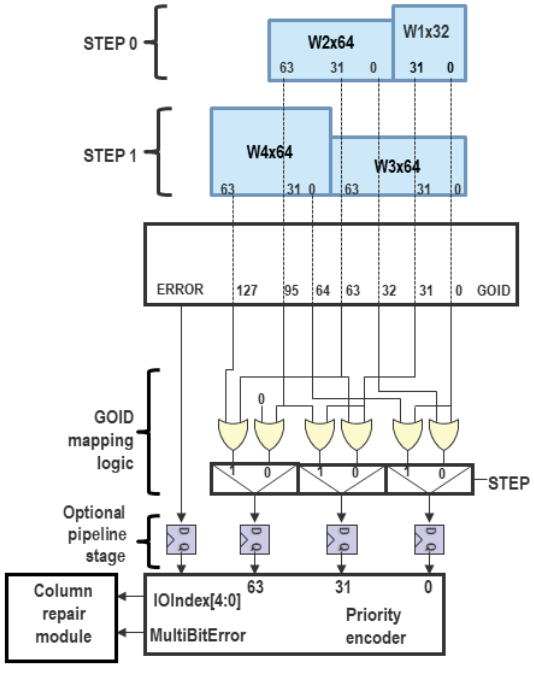

Fig. 1. Circuit calculating a common repair solution for several memories over multiple steps

These codes are provided by the memory vendor. Only a subset of the IOs is shown for simplicity. Again here, IOs sharing the same code are ORed together. This is first done within each step and then across steps. In step 0, IO[32] to IO[63] of memory W2x64 are the only contributors to the corresponding inputs of the priority encoder since memory W1x32 only has 32 codes. If the first error occurs on one of the most significant bits (IO[32] to IO[63]) of W2x64, a different IO of W1x32 is repaired. For example, if the first error occurs on IO[40] corresponding to code 101000 of W2x64, then IO[8] corresponding to code 01000 of W1x32 will be allocated as only the least significant bits of the code are applied.

The same type of "aliasing" occurs for memories using row repair or a combination of row and column repair. Take for example a small group where memory M1 has 64 rows and 64 IOs and memory M2 has 128 rows and 16 IOs. Each memory has one spare IO and one spare row unit of two rows. The BIRA module is virtually operating on a memory view composed of 128 rows and 64 IOs when calculating the repair solution. This solution is stored in the built-in self-repair (BISR) register located between the BIRA module and the memories. Memory M1 receives all 6 bits of the IO repair code but only the 5 least significant bits of the row address. Memory M2 receives all 6 bits of row address but only the 4 least significant bits of the IO repair code.

Another significant aspect of the hardware implementation is the handling of the variable pipeline depth which can vary as a function of the operation set selected and the step for each group sharing repair logic [6]. The presence/absence of the optional pipeline stage of Fig. 1 must also be taken into account. This stage is required when the number of steps and/or the number of inputs of the encoder is large for performance reasons. For the address, the issue is further complicated by the presence of address mapping/scrambling.

We determined four main conditions for defining repair groups. Automation was built into our tools to verify that the conditions are met and extract the information necessary to construct the hardware as described in the previous section. Memories in a group must:

Condition 1: share comparators if IO/column repair is used. For this type of repair, the BIRA module needs to be located near the comparators to minimize routing. For the same reason, comparators might have to be located close to the memories which might restrict the group size. For larger groups, comparators should be located in the controller. A shared bus such as the one in [6] might be advantageous in this case to connect memories to the controller. Note that for row-only or word-only type of repair, the BIRA module does not need to follow the comparators as only the global error signal needs to be sent to the BIRA module. This is advantageous because the comparators can stay close to the memories to minimize routing but the BIRA module can be moved to the controller to maximize the group size.

Condition 2: use the same type of repair. Our tools support row/word-only, column/IO-only and a combination of

the two. However, only one type can be used within a group. This is usually not a severe limitation as memories within a design block tend to use the same repair type.

Condition 3: use the same spare size. For a spare row unit, this means a same number of rows. In our example of Fig. 1, both memories were using spare units of two rows and were therefore compatible. The least significant bit of the repair inputs correspond to row address bit 1 for both memories. Row address bit 0 is ignored. Note that the number of columns in the row is not important. For a spare column block, it is the number of rows which is not important. For IO repair, the number of rows and columns is not important. Also, multiple IOs can be replaced as long as they are associated to a same repair code.

Condition 4: comply with user requirements. The user can define the maximum number of memory bits of the group or limit the sharing to be done within logical or physical memory boundaries.

The reduction in repair logic area and repair time upon power up are approximately linear with the number of memory segments sharing a same repair solution. The segments can be part of the same or different memories. For example, 5 memories with 2 segments each sharing a repair solution will provide about an order of magnitude reduction for both the repair logic area and repair time. As indicated earlier, the area reduction might be less for the case of IO/column repair due to the extra routing required in some cases. However, the reduction in repair time is the same for all repair methodologies. The area and repair time reduction are not analyzed in more detail due to space restrictions.

III. YIELD IMPACT ANALYSIS

Repair logic sharing should not be done at the expense of yield. This section provides some theoretical basis to choose group sizes in a way to minimize the yield impact.

We use an equation from [7] to calculate the probability of having a number of defects on a chip.

$$\text{Probability}(x) = \frac{\Gamma(\alpha + x)}{x!\,\Gamma(\alpha)} \cdot \frac{(Ad/\alpha)^x}{(1+Ad/\alpha)^{\alpha+x}} \qquad (1)$$

Where:
Probability (x) = Probability (number of defects on chip = x)
d = defect density
A = chip area
α = clustering parameter
Γ is the gamma function
α = 0, $p(x)$ is a delta function (maximum clustering)
α = ∞ , $p(x)$ is Poisson distr. (no clustering, William/Brown)

Using this equation, we calculate the yield assuming that 1) all defects are in memories and 2) all memories are repairable. The first assumption is reasonable since more than 50% of a chip is memory and the defect density of memories is 1 to 2 orders of magnitude higher than random logic due to the analog nature of its circuitry.

The second assumption might appear somewhat optimistic but should be acceptable since we are mostly interested in analyzing yield sensitivity to repair logic sharing and not so much in the actual yield.

Fig. 2 and Fig. 3 show some examples of estimated yield. Fig. 2 is for a small size chip of 140mm^2 and Fig. 3 is for a large chip of 560mm^2. The defect densities and clustering factors are selected to cover ranges found in the literature. The figure indicates that the number of defects is relatively small even for large chips and high defect densities considering the number of memories in the circuit. This is why it is possible to compress and store memory repair solutions in eFuse macros of a few thousand bits.

When sharing a BISR register amongst several memories or segments of a memory with several spares, it is very unlikely that multiple defects occurring in two or more memories or segments of a group would require the same repair values. In most cases, this would result in an un-repairable chip. Consequently, we need to limit the group size so that the probability of having more than one defect in each repair group is essentially zero. We divide the chip into tiles and apply formula 1 to each tile but only include the probability of having at most one defect in each tile. The new estimated chip yield is then given by:

$$\text{Yield}_{chip} = (1 - \text{Prob}(x > 1)_{tile})^{Ntiles} \qquad (2)$$

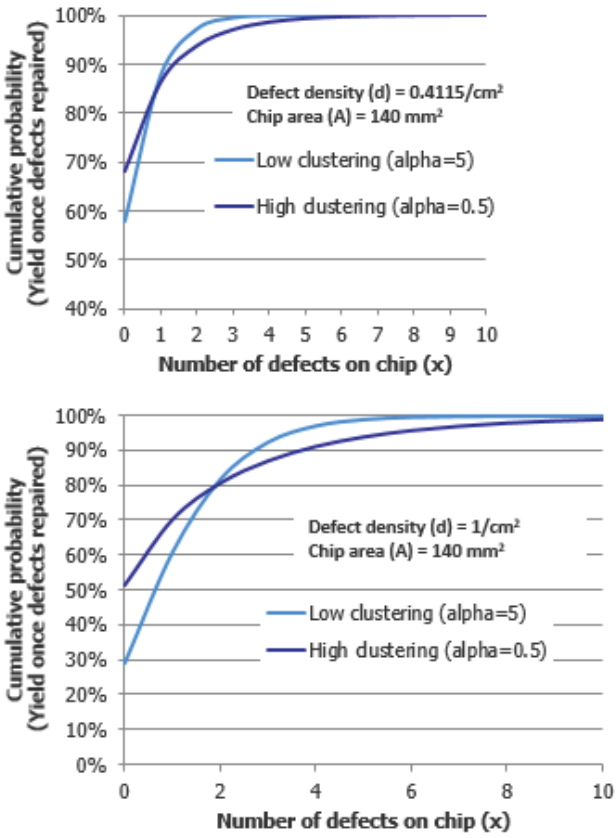

Fig. 2. Yield vs number of defects (small chip area)

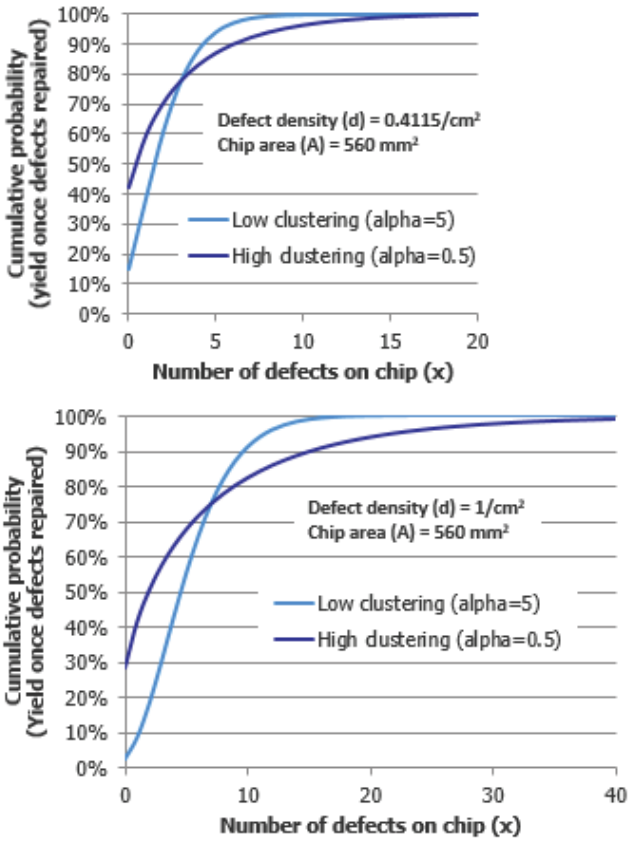

Fig. 3. Yield vs number of defects (large chip area)

The goal is to reduce the number of tiles while minimizing the impact on yield. Since only one BISR register is needed for each tile, the repair logic area and repair time are both reduced proportionally. Table I and Table II show the results for the chips of Fig. 2 and Fig. 3 using the same defect densities and clustering factors. Assume that the number of tiles of 10000 corresponds to the case that there is no sharing of repair registers so that each memory or memory segment can be repaired independently of each other. The yield is close to 100%. However, when reducing the number of tiles and BISR registers by a factor of 10, there is virtually no yield loss for a moderate defect density for both chips. For a high defect density, the yield loss is moderate, less than 0.3% for the first chip but more significant (4.5%) for the second one especially for a low clustering factor. Trying to reduce the number tiles by another order of magnitude can still be considered for a moderate defect density but not practical for a high one.

The defect density and clustering parameter required in equation 1 are not always known by the designer who needs to make a decision about the level of repair logic sharing acceptable for a new design. However, data collected from previous designs of similar size without sharing can be used to anticipate the potential yield impact of sharing repair logic. The first two columns of Table 3 shows an example of such data for a lot of 1000 chips of a fictitious circuit. The first column indicates the number of repairs needed and the second column the number of chips requiring that number of repairs.

TABLE I. YIELD VS NUMBER OF TILES (CHIP SIZE=140MM^2)

	d=1/cm^2		d=0.1/cm^2	
# of tiles	α = 5	α = 0.5	α = 5	α = 0.5
100	98.85%	97.23%	99.99%	99.97%
1000	99.88%	99.71%	100.00%	100.00%
10000	99.99%	99.97%	100.00%	100.00%

TABLE II. YIELD VS NUMBER OF TILES (CHIP SIZE=560MM^2)

	d=1/cm^2		d=0.1/cm^2	
# of tiles	α = 5	α = 0.5	α = 5	α = 0.5
100	83.63%	67.39%	99.81%	99.54%
1000	98.15%	95.49%	99.98%	99.95%
10000	99.81%	99.53%	100.00%	100.00%

Based on this data, a simple formula can be applied to calculate the yield impact as a function of the sharing factor. In our example, the circuit includes 14286 repair registers of 7 bits on average (100000 bits total). Each repair register is associated to one spare resource either spare row or column. Each memory can have more than one spare. The goal is to reduce the number of repair registers while minimizing the yield impact.

This is done by using a same repair register for a group of spares. The group size selected for the calculations of Table 3 is 10. The third column represents the probability that the number of defects in any repair group is at most 1 knowing that that there is exactly k defects in the chip. This can be expressed as:

$$\text{Yield}(k) = (N*(N-1)*(N-2)*\ldots*(N-(k-1))/N^k \quad (3)$$

Where N is the number of groups and k is the number of defects.

TABLE III. YIELD IMPACT PREDICTION BASED ON EXISTING SILICON

Number of defects	Number of chips	Yield(k)	Weighted Yield(k)
0	484	100.00%	48.40%
1	327	100.00%	32.69%
2	133	99.93%	13.24%
3	42	99.79%	4.17%
4	11	99.58%	1.12%
5	3	99.30%	0.27%
6	1	98.95%	0.06%
Final yield			99.98%
Yield impact			0.02%

To simplify the calculations, we assumed that all repair registers were of the same size (7 bits). The fourth column represent the contribution of Yield(k) for the entire chip modulated by the number of chips that required k defects. It can be seen that even if Yield(k) is relatively low for a large value of k, the impact on the overall yield is low due to the small number of chips with that number of defects.

Final yield indicates the percentage of repairable chips which are still repairable after applying sharing. The yield impact indicated does not consider the potential yield gain due to the area reduction implied by sharing the repair logic which could partly offset the yield loss. Remember also that the non-repairable chips are not considered throughout this paper so that the yield impact due to sharing would be reduced further.

The sensitivity of the yield loss was analyzed with respect to 3 parameters: total BISR chain length, average repair register length and sharing factor. The analysis was done for a 560 mm^2 circuit. Two defect densities were compared. One is relatively low ($d=0.1395/cm^2$) and the other one is relatively high ($d=1.0/cm^2$). A low clustering factor ($\alpha = 5$) was used in both cases. Table IV shows the results. The values highlighted in orange are used as reference. When the value of one of the parameters is changed, the other parameters take their reference value.

A number of conclusions can be drawn from the results. The yield loss is directly proportional (within rounding errors) to the sharing factor but inversely proportional to the total BISR chain length. The yield loss is also proportional to the average repair register length but since the range of values is relatively small, the impact is limited. For low defect densities, a sharing factor of 10 or even 20 might be considered. However, for high defect densities, the sharing factor would definitely be less than 10 and should only be considered for circuits with a large number of memories.

TABLE IV. YIELD IMPACT AS A FUNCTION OF PARAMETERS

Parameter	Value	d=0.1395/cm2	d=1.0/cm2
Total BISR chain length	25000	0.10%	5.00%
	50000	0.05%	2.63%
	100000	0.03%	1.39%
	200000	0.01%	0.76%
	400000	0.01%	0.44%
Average repair register length	6	0.02%	1.21%
	7	0.03%	1.39%
	8	0.03%	1.57%
	10	0.04%	1.93%
	12	0.04%	2.28%
Sharing factor	2	0.01%	0.38%
	5	0.01%	0.76%
	10	0.03%	1.39%
	20	0.05%	2.63%
	40	0.10%	5.00%
	100	0.25%	11.40%

IV. CONCLUSIONS

We presented techniques reducing the amount of logic dedicated to memory repair and reducing the time to repair memories at power up. This is accomplished by sharing repair logic calculating a common solution for a group of memories and/or group of segments within a memory in a concurrent fashion. The implementation of the architecture was implemented in a commercial tool. Area and repair load time reductions are directly proportional to the group size which is under user control. A theoretical model was developed to predict the impact on yield as a function of the group size where the defect density and clustering parameter are known. An alternative method only relying on repair data collected from previous chips was also presented. The proposed equations help designers making the trade-off which is most appropriate for their situation.

REFERENCES

[1] Brian Bailey, Memory Design at 16/14nm, Semiconductor Engineering, May 28th 2015, https://semiengineering.com/memory-design-at-1614nm/

[2] Bruce Cowan, Owen Farnsworth, Peter Jakobsen, Steve Oakland, Michael R. Ouellette, Donald L. Wheater, On-Chip Repair and an ATE Independent Fusing Methodology, International Test Conference 2002, paper 7.3

[3] Harsharaj Ellur and Kalpesh Shah, A Tag based solution for efficient utilization of efuse for memory repair, International Test Conference 2014, paper 17.3

[4] Swapnil Bahl, A Sharable Built-in Self-repair for Semiconductor Memories with 2-D Redundancy Scheme, 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (2007), pp 331-339

[5] V. R. Devanathan and Sumant Kale, A Reconfigurable Built-in Memory Self-repair Architecture for Heterogeneous Cores with Embedded BIST Datapath, International Test Conference 2016, paper 18.2

[6] Teresa McLaurin, Frank Frederick, Rich Slobodnik, "The DFT Challenges and Solutions for the ARM Cortex-A15 Microprocessor, ITC International Test Conference 2012, paper 1.2

[7] Akhil Garg, Prashant Dubey, Fuse Area Reduction based on Quantitative Yield Analysis and Effective Chip Cost, Proceedings of the 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06)