Optimal Reconfiguration Functions for Column or Data-bit Built-In Self-Repair

M. Nicolaidis, N. Achouri, S. Boutobza
iRoC Technologies
38025 Grenoble, France

Abstract

In modern SoCs, embedded memories occupy the largest part of the chip area and include an even larger amount of active devices. As memories are designed very tightly to the limits of the technology they are more prone to failures than logic. Thus, memories concentrate the large majority of defects and affect circuit yield dramatically. As a matter Built-In Self-Repair is gaining importance. This work presents optimal reconfigurations functions for memory built-in self-repair on the data-bit level. We also present a dynamic repair scheme that allows reducing the size of the repairable units. The combination of these schemes allows repairing multiple faults affecting both regular and spare units, by means of low hardware cost. The scheme uses a single test pass, resulting on low test and repair time.

1. Introduction

Traditionally, memory repair is performed by using external equipment to test the memory, localize the faults, and drive a laser beam that perform the repair. Electrical fuses or anti-fuses can also be used to avoid laser beam. In this case too, the external test equipment determines the fuses to be blown. Recent developments replace external equipment by Built-In Self-Test (BIST) and Built-In Self-Repair (BISR) schemes in order to maintain at reasonable levels the test and the repair cost of embedded memories. An additional advantage is that BIST and BISR can test and repair embedded memories at any time during the product life. This reduces maintenance cost, and increases reliability and product life. Various BISR approaches have been developed. Row (or word) BISR uses a spare row (or word) to replace a faulty regular row (or word). Column BISR uses a spare column to replace a faulty regular column. Data-bit BISR uses a memory unit generating a spare data bit to replace a memory unit generating a regular data bit. Each of them has its advantages and drawbacks, and can be of interest under various circumstances. In this work we consider the data-bit BISR and column BISR schemes. Data-bit BISR can repair faulty cells, faulty columns, as well as faulty column-MUXes, faulty read or write amplifiers, and faulty data input/output registers. Repairing read amplifiers may improve yield significantly, since the sense amplifiers are very sensitive circuits and can be faulty more frequently than other parts. On the other hand, row and word repair cannot repair the above-mentioned faults.

Row/word repair is the simplest BISR approach, so, the majority of the previous works consider this scheme, although column repair was the predominant external repair scheme. Word repair was early proposed by K. Sawada et al [1]. It uses a content addressable memory storing the data and addresses of the faulty words. The work considers low numbers of faults. Also, faults in the spare parts are not considered. Subsequent work on row BISR [2] [3], [4] use the same basic scheme but improve various implementation aspects. These works too consider a low number of faults (e.g. two faults in [3]), and no faults in the spare units. A more recent work [7] innovates by using nonvolatile memory cells to fix once forever manufacturing faults.

Work on column/data BISR is more recent due to the difficulty for elaborating the reconfiguration functions. Kim et al [5] present the first work in this domain. It proposes a scheme where the reconfiguration information is generated by a controller and stored in a memory. To master the complexity of the reconfiguration process, the scheme repairs a single fault per test phase. That is, the memory is tested until the first fault is found and repaired. Then, the memory is tested again until a second fault is found and repaired, and so on. This process simplifies the work of the BISR control unit, but the test and repair time becomes very high when the number of faults increases. The paper considers a small number of faults (e.g. 2 faulty columns out-of 128 regular columns), and uses a large number of memory cells for storing the reconfiguration information.

Another paper [6] considers the combination of column and row repair. It proposes an algorithm that allocates efficiently the spare rows and columns to repair multiple faults that may affect some columns and rows. However, it does not propose BISR circuitry for reconfiguring the memory.

In the present work we present optimal reconfiguration functions for column and data-bit repair. This is the first study that derives analytically these functions. The derived functions perform repair for multiple faults affecting both the regular and spare elements, minimizes the hardware cost for implementing the repair control and for storing the reconfiguration information, and performs the repair by means of a single test pass. This optimizes BISR cost and repair efficiency. In addition, it is not required to modify the memory structure, since the repair circuitry is placed around the memory. Thus, the memory can be generated by existing memory generators.

2. Data-bit or column Built-In Self- Repair

Data-bit repair localizes the faulty data bit positions, and replaces the memory parts generating these bit positions by spare parts. Thus, the replaceable unit will be

1530-1591/03 $17.00 © 2003 IEEE

the block of the memory cells connected to a single input/output. In order to be able to repair k faults we will add k such spare blocks (spare units).

We consider that we dispose a set of latches that store the information of the faulty bit positions (Faulty Bit Indication – FBI latches). The FBI latch of position i (FBI_i) contains a 0 if the data bit di is fault-free or a 1 if it is faulty. The number of FBI latches is equal to the number of regular and spare units (n + k). One possible way to compute the contents of the FBI latches is shown in figure 1. The comparator used in the BIST circuitry for comparing the read data against the expected data, is also used to provide the signals X_i that determine the state of the FBI_i latches. The BIST comparator uses an XOR gate at each bit position i. During the test session, this gate compares the data read at the output di against the value expected to be read at this output. These XOR gates are not shown in the figure. An OR compacts the outputs of the XOR gates to generate a unique error detection signal (this gate is not shown in the figure). The signals X_i are not exploited by the go/no go BIST approach. However, in the case of the BISR approach we use the signals X_i as shown in figure 1. An OR gate combines the signal X_i with the output of the latch FBI_i to generate the input of the latch FBI_i. Thus, if at any cycle of the test session a 1 is generated on X_i (detection of an error on the bit position di), this value is memorized in the FBI_i latch for the rest of the time, indicating a fault in the bit position di. Note that the spare units may also be faulty. The reconfiguration functions developed in this work will be implemented in a manner that only the fault-free spare units will be used to repair the faulty regular units. For doing so, both the regular and the spare units are controllable and observable by the BIST circuitry. This circuitry will use a comparator of length n+k, and will provide inputs to n+k FBI latches, as shown in figure 1.

To perform repair we use a set of reconfiguration MUXes to replace the faulty units by the spare ones. A logic bloc implementing the reconfiguration functions will receive at its inputs the outputs of the FBI latches, and will provide on its outputs the control signals of the MUXes. We can use MUXes having control signals coded into the binary code or into the 1-out-of-(k+1) code.

We can use a left-side or a right-side repair. In the following we will consider a left-side repair. The reconfiguration can be done in a local manner, or in a distant manner. In the local repair (figure 2), a faulty unit is isolated and its left-side closest fault-free unit is used to replace it. Such a replacement will also require shifting to the left the connections of the bit positions being to the left of the repaired position. The repair starts from the faulty unit of lower order (the right most one). Then, we proceed to the repair of the next lower order faulty unit, and so on. We number the functional units from 0 to n-1 (U0, U1, …, Un-1) and the spare units from n through to n+k-1 (Un, Un+1, …, Un+k-1). Figure 2 shows an example of the local left-side repair using four functional units and three spare units. We observe that for each regular unit we use one MUX of the type 1-out-of-(k+1), to be able to connect each data bit to the corresponding regular unit or to one of its k left-side neighbor units.

In the distant repair, the functional and spare units form two distinguished sets in the repair process. Each faulty functional unit is replaced by a spare one rather than by its closest fault-free one. In this case there is no shift on the connections of the fault-free regular units. For this scheme we need to use the same number of MUXes as for the local repair, but the signals controlling the MUXes are computed by using different reconfiguration functions. Also, we use a smaller number of interconnection lines, but each of these lines is longer. An example of distant repair for n = 4 and k = 3 is shown in figure 3.

In the above we consider that the reconfiguration is done at the data-bit level. However, the reconfiguration functions derived in the next sections can also be used to perform the repair at the memory column level. In this case the FBI latches have to indicate the faulty columns instead of the faulty data input/outputs. The problem with a pure column repair is that it requires introducing the reconfiguration MUXes and the routing within the memory layout. This can be difficult because each 1-out-of-(k+1) MUX must fit within the width of a single column. To avoid modifying the memory layout, but reduce the size of the repairable units at the size of a memory column or even less, we have developed a dynamic reconfiguration scheme described in section 5.

3. Reconfiguration functions for local repair

The idea for deriving optimal reconfiguration functions is to implement a circuit that counts the number Shift(i) of positions that each signal di must be shifted. This count can be performed by a combinational or by a sequential circuit. To compute the number Shift(i) [10] we need to count the number of ones in a first set of FBI latches composed of the FBI latches of positions 0 through to i, and then determine the minimum integer q such that the set of FBI latches of positions i+1, i+2, …, i+q (second set) contains a number of 0's equal to the number of 1's in the first set. Shift(i) is the equal to q. In an equivalent manner, we can count the number of 1's in the first and second set of FBI latches, and set Shift(i) equal to this number. This counting can be done more easily by a sequential circuit. For a combinational circuit, counting the number of 0's in the second set of FBI latches is more complex since this set has a variable size (it depends on the contents of the FBI latches of the first set and on the contents of the second set itself). As an illustration, consider the case where two latches in the positions 0 through to i contain a 1, the latches in positions i+1 and i+4 contain a 0, and the latches in positions i+2 and i+3 contain a 1. In this situation, to determine by how many positions we have to shift the position i, we have to count the number of 1's in positions 0 through to i (first set), and the number of 1's in positions i+1, i+2, i+3, and i+4. This is equal to 4, and the position i must be shifted to the position i+4. However, if positions i+1 and i+3 contain a 0 and positions

i+2and i+4 contain a 1, then, if we consider again the positions i+1, i+2, i+3, and i+4 to compose the second set, we will count again four 1's and we will shift position i to position i+4, while the correct decision is to shift it to the position i+3. In fact, in the present case, the second set should include only the positions i+1, i+2, and i+3. Thus, to count the number of 1's in the second set we need to determine this set. But for determining it we need to count its number of 1's. This results on complex reconfiguration functions that are costly to implement, especially as combinational functions. In order to simplify the complexity of the combinational functions, we make the count into the 1-hot code instead of the binary code. In this code, the jth variable will determine if the ith position will be shifted to the i+j position. This is determined by the contents of a constant set of latches (those of positions 0 through to i+j), simplifying considerably the complexity of the reconfiguration functions. Also the 1-hot code will control directly the MUXes without using a decoder, as is the case for the binary code. In addition, in order to further reduce the hardware cost, we have adopted a recursive approach. It leads to recursive equations that are implemented as an iterative combinational or sequential logic array. The hardware complexity of such arrays is low but their delay is large, since it is linear to the number of input variables. However, these arrays will compute the values of the reconfiguration signals only once (at the end of the repair phase). Then, these signals will remain stable during the circuit operation. Thus, the delay of the reconfiguration functions is meaningless.

We have developed two types of iterative networks for computing the reconfiguration signals: sequential iterative functions and combinational iterative functions.

3.1 Sequential local reconfiguration functions

To count the number of positions that the data-bit di has to be shifted, we implement a sequential circuit that generates a signal R_i which takes the value 1 during Shift(i) consecutive clock cycles [8][9]. A counter is associated to each signal R_i. The counter is incremented each time R_i is 1. If we use reconfiguration MUXes that are controlled by signals belonging to the 1-out-of-(k+1) code, then the counter will be a shifter of size k+1, initialized to the value 000…01. If the reconfiguration MUXes are controlled by a binary code, then, a binary counter will be used.

We use the FBI latches and some logic to implement the sequential circuit that generates the signals R_i. The signal R_i depends on the value of latch FBI_i, but also on the FBI_j latches, with j<i. This is because when a position j is shifted to the left, any position i with i>j has also to be shifted to the left. The value of R_i also depends on the values of the FBI_j latches with j>i. This is because each time the position i is shifted to the left it is connected to a unit being on the left of position i, which can be faulty or fault-free. The information indicating the state of this unit is stored in a latch FBI_j having j>i.

As said earlier, to simplify the equations of the reconfiguration functions for an arbitrary position, we have adopted a recursive approach. We obtain these equations thanks to the following observations:

a) $FBI_j = 1 \Rightarrow R_j = 1$, since for a faulty unit Uj, R_j must counts at least once.

b) Each time a signal R_j is activated it also forces the signal R_{j+1} to the active state (i.e. logic 1), since each-shift left of dj implies a shift-left of dj+1.

c) From a) and b) we obtain $R_i = FBI_i + R_{i-1}$.

d) When the signal R_i is activated (R_i=1) the value of latch FBI_{i+1} is transferred to the latch FBI_i. In any other case the value of FBI_i becomes 0. This is because the signal di is now connected to the unit on which it was connected previously the signal di+1. Thus, the FBI latch of position di should now indicate the state indicated previously by the latch FBI of position di+1.

This analysis gives the following sequential equation for the FBI latches, during the repair phase.

$(FBI_i)t_{q+1} = (R_i \cdot FBI_{i+1})t_q$ where tq and tq+1 are two consecutive time instances.

We obtain two very simple equations implementing the sequential circuit that generates the signals R_i.

(1) $R_i = R_{i-1} + FBI_i$, $\forall$ i : n+k-1 ≥ i ≥ 0, $R_{-1} = 0$

(2) $(FBI_i)t_{q+1} = (R_i \cdot FBI_{i+1})t_q$, $\forall$ i : n+k-1 ≥ i ≥ 0, $FBI_{n+k} = 0$.

From figure 1, during the test phase the values of the FBI latches can be computed by using the equation $FBI_i = FBI_i + X_i$. Where X_i is the signal indicating the status of a read operation at position di (X_i = 0 for correct read data, X_i = 1 for false read data). Then, the equation (2) can be replaced by the equation (3), where the signal REP is 1 during the repair phase and 0 during the test phase:

(3) $(FBI_i)t_{q+1} = (\bar{}REP \cdot (FBI_i + X_i) + REP \cdot R_i \cdot FBI_{i+1})t_q$, $\forall$ i : n+k-1 ≥ i ≥ 0, $FBI_{n+k} = 0$.

The scheme described by the equations 1) and 3) is shown in figure 4. We note that a chain of OR gates interconnected in series implements the equation (1). This will result on high delays. This may involve a low operation frequency of the repair phase. However, this does impact neither the frequency of the test phase nor the frequency of the normal operation of the memory. In the implementation we can use, during the repair phase, an independent clock signal having a low frequency (e.g. obtained from a frequency divider), or use the same clock signal as for the other phases, but reduce the frequency of this signal during the repair phase.

We dispose n MUXes, one for each data-bit di. The MUX of position i connects the data input/output di to the regular unit Ui, if this unit is fault-free, and to one of the k units Ui+1, … Ui+k if Ui is faulty. Thus, each of the signals $R_0, R_1, \ldots R_{n-1}$ is used to enable the counter of one of the MUXes and it is also used as entry to the equations (1), (3). On the other hand, the signals $R_n, \ldots R_{n+k-1}$ are only used as entries to the equations (1), (3), but no counters are associated to them. The signal R_{n+k-1} can be used to indicate the success or the failure of the repair process. If $R_{n+k-1} = 0$ after k clock cycles of the repair

phase, the repair was successful. However, if $R_{n+k-1} = 1$ after the k clock cycle, then the repair has failed because there were more than k faulty units. R_{n+k-1} can also be used as a repair completion signal. That is, the repair phase can be finished as soon as R_{n+k-1} becomes 0.

3.2 Combinational local reconfiguration functions

With this scheme [9] we use reconfiguration MUXes that have inputs coded into the 1-out-of-(k+1) code). In this case, the control signals of the MUX of position i will count into the 1-out-of-(k+1) code the number of positions that we have to shift the data bit di. We have to derive the function of a combinational circuit that performs this counting. The input variables of this function are the states of the FBI latches. For the MUX of position i, the control signal M_j^i determines if di is shifted to the unit Ui+j. We have seen that the number of positions that a data di has to be shifted is a function of a variable set of FBI latches, resulting on complex reconfiguration functions. This problem appears when the control signals of the reconfiguration MUXes, belong to the binary code. However, when these signals belong to the 1-out-of-(k+1) code, this problem is eliminated. In fact, the signal M_j^i is a function of a fixed set of FBI variables (the variables FBI_0, FBI_1, …, FBI_{i+j}). Thus, the reconfiguration functions are simplified considerably. To derive the functions of the control signals of the reconfiguration MUXes in a recursive manner, we first consider the control inputs of the MUX of the position 0. The signal M_0^0 is 1 iff the unit U_0 is fault free ($FBI_0 = 0$), that is $M_0^0 = \neg FBI_0$. The signal M_j^0 is 1 iff the unit U_0 has to be shifted by j positions in order to be connected to the first fault-free unit. That is, when $FBI_0 = FBI_1 = \ldots = FBI_{j-1} = 1$ and $FBI_j = 0$. Thus, we obtain:

(4) $M_0^0 = \neg FBI_0$, $M_1^0 = \neg FBI_1 \cdot FBI_0$, …, $M_k^0 = \neg FBI_k \cdot \ldots FBI_1 \cdot FBI_0$.

The equations describing the signals M_j^i of an arbitrary position i involve the variables on the left and on the right of position i in a complex manner. To simplify these equations, we have adopted a recursive approach where the control signals of position i+1 are expressed as functions of the control signals of position i. The signal M_j^{i+1} is equal to 1 if the unit Ui+1 has to be shifted by j positions. If M_r^i is 1 then M_j^{i+1} will be 1 only if the units Ui+r+1, Ui+r+2, …, Ui+j are faulty and the unit Ui+j+1 is fault-free. Thus, we obtain the equations:

(5) $M_j^{i+1} = \neg FBI_{i+j+1} (M_j^i + M_{j-1}^i \cdot FBI_{i+j} + M_{j-2}^i \cdot FBI_{i+j-1} \cdot FBI_{i+j} + \ldots + M_0^i \cdot FBI_{i+1} \cdot FBI_{i+2} \ldots \cdot FBI_{i+j})$, $0 \le j \le k$, $0 \le i \le n-2$.

4. Combinational distant reconfiguration functions

With this scheme [9], a regular faulty unit is replaced by a spare fault-free unit and not by its closest fault-free unit. Thus, a regular unit, which is fault-free, is not shifting its position. The regular units are labeled as RU_0, RU_1, …RU_{n-1} and the spare units are labeled as SU_1, SU_2, …, SU_k. The corresponding states of the FBI latches will be noted RF_0, RF_1, …, RF_{n-1} and SF_1, SF_2, …, SF_k.

The equations of the control signals of the MUX of position 0 are the simpler to obtain. The signal M_0^0 is 1 if RU_0 is fault-free. The signal M_j^0 is 1 if RU_0, SU_1, SU_2, …, SU_{j-1} are faulty and SU_j is fault-free. Thus, we obtain the equations:

(6) $M_0^0 = \neg RF_0$, $M_1^0 = \neg SF_1 \cdot RF_0$, …, $M_k^0 = \neg SF_k \cdot SF_{k-1} \ldots SF_1 \cdot RF_0$.

To reduce the complexity of the equations for an arbitrary data bit dj we have again adopted a recursive approach. However, the analysis for determining variables M_j^i is trickier than in the case of local repair. In fact, in the present case, when computing the variables M_j^{i+1} we can not use the value of the variables M_r^i , $0 \le r \le k$, as indicators of the number of spare units occupied after the repair of some of the units RU_0, RU_1, …RU_i.This is because in the present case, if RU_i is fault-free, then, the functions M_1^i, M_2^i ,…, M_k^i are all 0, although some of the units RU_0, RU_1, …, RU_{i-1} can be faulty. Thus, we introduce some intermediate variables F_j^i, $0 \le i \le k$, that count the number of spare units occupied by the faulty units RU_r, $0 \le r \le i$. These variables are determined as follows.

For position 0, the variables F's are equal to the variables M's. Thus we have:

(7) $F_j^0 = M_j^0$, $\forall j \in \{0, 1, \ldots, k\}$.

For position i+1, the variables F_j^{i+1} are equal to the variables F_j^i if the unit RU_{i+1} is fault-free otherwise it is equal to M_j^{i+1}. Thus we have:

(8) $F_j^{i+1} = F_j^i \cdot \neg RF_{i+1} + M_j^{i+1} \cdot RF_{i+1}$, $0 \le i \le n-2$, $0 \le j \le k$.

The variable M_0^{i+1} is 1 iff RU_{i+1} is fault-free. The variable M_{j+1}^{i+1} is 0 if RU_{i+1} is fault-free. If RU_{i+1} is faulty ($RF_{i+1} = 1$), then, M_{j+1}^{i+1} becomes 1 if SU_{j+1} is fault-free ($SF_{j+1} \cdot = 0$) and the following two conditions hold for any integer r , $0 \le r \le j$:

- the first r spare units are already occupied either because they are used to repair some units RU_q $q \le i$ or because they are faulty (which means $F_r^i = 1$),
- the spare units SU_{r+1}, SU_{r+2}, …, SU_j are faulty (i.e. $SF_{r+1} = SF_{r+2} = \ldots = SU_j = 1$).

This analysis leads to the equations

(9): $M_0^{i+1} = \neg RF_{i+1}$, $M_{j+1}^{i+1} = \neg SF_{j+1} \cdot RF_{i+1} (F_j^i + F_{j-1}^i \cdot SF_j + F_{j-2}^i \cdot SF_{j-1} \cdot SF_j + \ldots + F_0^i \cdot SF_1 \cdot SF_2 \cdot \ldots \cdot SF_j)$, $0 \le j \le k-1$, $0 \le i \le n-2$.

A sequential reconfiguration functions for distant repair can also be derived. The circuit has a similar complexity as the sequential reconfiguration functions for local repair.

5. Dynamic Repair

In order to reduce the size of the units used to repair each fault, we have developed a dynamic repair approach With this scheme [10]. The idea is to reconfigure the memory inputs/outputs in a dynamic manner, instead of static one. In fact, instead of shifting permanently a data input/output to a fixed position, we shift it dynamically to various positions. That is, a data input/output is shifted only when a part of a bloc containing a faulty cell is selected by the memory addresses. For doing so, we implement $R = 2^m$ blocks of FBI latches, each block containing n+k FBI latches. We decode the value of m address bits to select one of these blocks at each cycle of the test phase. In fact, at each cycle of the test phase, the signals X_i (i.e. the outputs of the XOR gates of the BIST comparator shown in figure 1), are connected to the inputs of the selected set of FBI latches. This is done by means of a MUX controlled by these address bits. During each cycle of the regular operation, the value of the m address bits selects, through another MUX, the block of FBI latches that drives the inputs of the Reconfiguration Logic. This logic reacts to the state of this bloc of FBI latches and reconfigures the memory according to the fault-location information stored there. At another cycle, another value of the m address bits reconfigures the memory differently, by selecting another bloc of FBI latches to drive the Reconfiguration Logic. Thus, a fault-free part of one block of the memory can be selected at a cycle to replace a faulty part of a second memory bloc, and another non-faulty part of the former bloc can be selected at another cycle to replace a faulty part of a third memory bloc. Thus, for repairing a fault we use only a part of a memory bloc instead of the whole block. The higher the m the smaller is the bloc part used to repair each fault, improving the repair efficiency. Since the dynamic scheme uses multiple copies of the reconfiguration functions, the low cost functions derived previously improve drastically this scheme.

6. Implementation and Results

A BISR generator was developed to automate the implementation of the schemes described in this work. To avoid implementing all the schemes, we performed a preliminary cost analysis. This analysis shown that the hardware used for the local and the distant repair is of similar complexity. Since the distant repair involves longer interconnects it introduces larger delays. Therefore, we have selected the local repair approach for implementation. For this approach, the combinational scheme requires less hardware for small and moderate values of k. For larger values, the extreme simplicity of the logic part of the sequential reconfiguration functions (five gates per memory input/output), and the logarithmic increase of the binary counter with the increase of k, make the sequential scheme more economic. Thus, both schemes were implemented. The dynamic repair scheme is also implemented since it reduces the hardware cost drastically for a given repair efficiency. The tool was used to evaluate the hardware cost of the schemes for a 64K X 32 SRAM, implemented in a commercial 0.18-micron process. The results are presented in table 1. In this table, k represents the number of spare blocks and m the number of address bits used to perform dynamic repair (m = 0 corresponds to the static repair). The table presents the extra area as a percentage of the area of a memory that do not use repair. The extra area includes the area of the spares, the area of the reconfiguration functions and the area of the reconfiguration MUXes. We observe that the extra area increases slightly as we increase m. This is because the area of the optimal reconfiguration functions derived in this work is very small. Thus, even the cost of multiple copies of these functions remains low. As a result, the BISR cost is basically the cost of the spares. Thus, by increasing the value of m we can increase significantly the number of repairable units (and thus the number of repairable faults), by paying a small amount of extra area. For instance, for k = 2 and m = 0, we can repair 2 faults by paying a 6.37% of extra area. On the other hand, for k = 2 and m = 3 we can repair 16 faults by paying a slightly higher extra area (7.06%). We also see from the same table that for m = 0 and k = 6 the hardware cost is 18.94%. This option performs static repair (m = 0) and thus is very inefficient (it can repair only 6 faults for a much higher area cost). Thus, the low-cost reconfiguration functions combined with the dynamic scheme allow repairing a large number of faults by means of low area overhead.

7. Conclusions

This work presents low cost reconfigurations functions for memory BISR on the data-bit level, that repair multiple faults affecting both the regular and spare units. We also introduced a dynamic repair scheme that allows splitting each regular and spare unit into multiple repairable units. By combining the low cost of the reconfiguration functions with the low size of the repairable units obtained by the dynamic scheme, we achieve low repair cost for multiple faults. Also, the scheme repairs faults affecting both the functional and the spare units. An additional advantage is that the BISR circuitry is external to the memory. Thus, we can repair a memory without modifying its layout. Also, the repair scheme uses a single test pass, reducing drastically the test and repair time with respect to previous schemes.

Table 1. Area overhead of various BISR implementations for a 64K X 32 SRAM

m	***k = 6***	***k = 5***	***k = 4***	***k = 3***	***k = 2***	***k = 1***
3	18,63	15,6	12,58	9,33	6,46	3,4
2	17,97	14,98	12,02	8,91	6,1	3,11
1	17,63	14,67	11,72	8,7	5,87	2,97
0	17,44	14,51	11,57	8,6	5,77	2,89

References

[1] Sawada K., Sakurai T., Uchino Y., Yamada K., “Built-In self repair circuit for High Density ASMIC”, IEEE 1999 Custom Integrated Circuits Conference.

[2] Tanabe A. et al " A 30-ns 64-Mb DRAM with Built-in Self-test and Self-Repair Function", IEEE Journal Solid State Circuits, pp. 1525-1533, Vol 27, No 11, Nov. 1992.
[3] Bhavsar D. K., Edmodson J. H., "Testability Strategy of the Alpha AXP 21164 Microprocassor", I994 IEEE International Test Conference.
[4] Benso A. et al "A Family of Self-Repair SRAM Cores", 2000 IEEE International Test Conference. 2000 In Proc. IEEE International On-Line Testing Workshop, July 3-5, 2000.
[5] Kim I., Zorian Y., Komoriya G., Pham H., Higgins F. P., Newandowski J.L. "Built-In self repair for embedded high-density SRAM" Proc. Int. Test Conference, 1998, pp1112-1119
[6] Kim H. C., Yi D.S., Park J.Y., Cho C.H., "A BISR (Buil-In Self-Repair) circuit for embedded memory with multiple redundancies", 1999 IEEE International Conference on VLSI and CAD, Oct. 26-27, 1999, Seoul, Korea, pp 602-605
[7] V. Schober, S. Paul, O. Picot, "Memory Built-In Self-Repair using redundant words", 1991 IEEE International Test Conference.
[8] M. Nicolaidis "Symbiom: a Methodology for BIST Synthesis of Memories", 2nd Consulting Report, Mentor Graphics Corporation, December 1996.
[9] M. Nicolaidis, "Dispositif de reconfiguration d'un ensemble mémoire présentant des défauts" French patent, no 2 820 844
[10] M. Nicolaidis, "Reconfiguration device for a faulty memory" US patent pending.

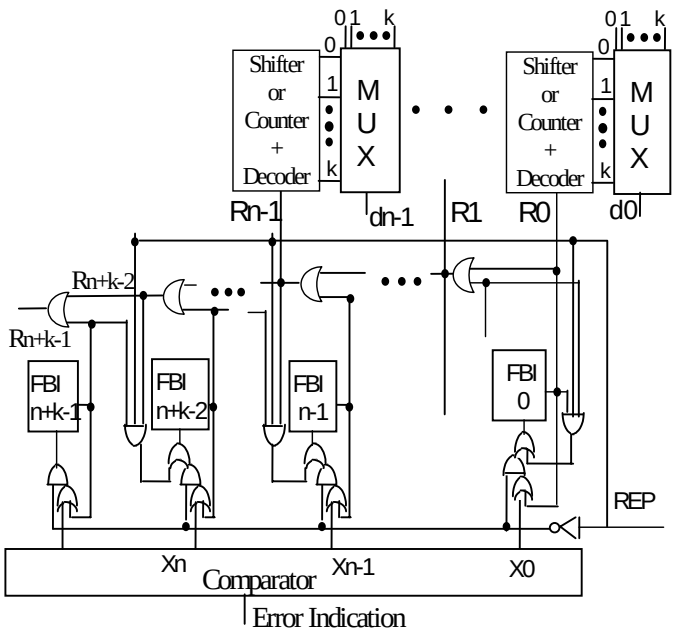

Figure 4: The sequential scheme for local repair

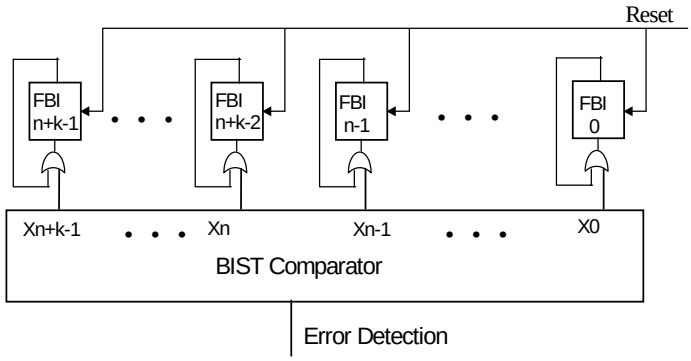

Figure 1: Generation of the state of the FBI latches

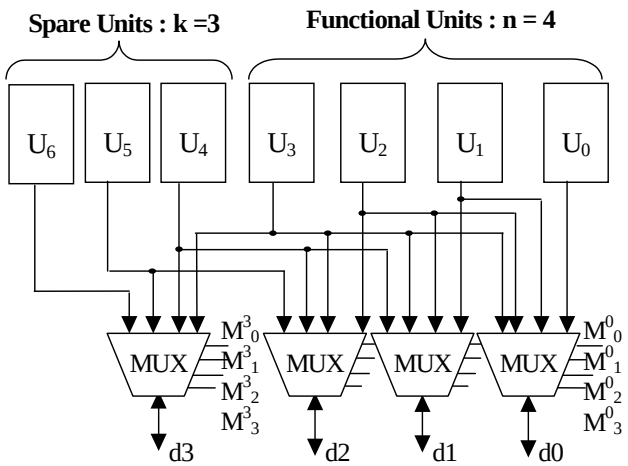

Figure 2: The local repair scheme

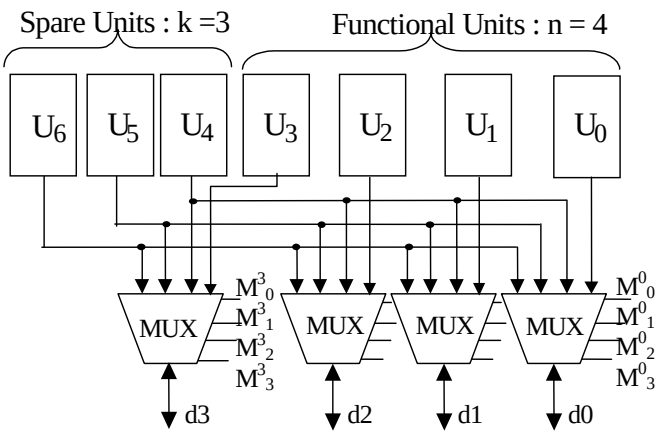

Figure 3: The distant repair scheme.