Dynamic Data-bit Memory 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, they concentrate the large majority of defects and affect circuit yield dramatically. Thus, Built-In Self-Repair is gaining significant importance. This work presents a dynamic memory built-in self-repair schemer acting on the data-bit level. It allows reducing the size of the repairable units, or in other words, it allows using a single spare unit for repairing faults affecting several regular units. As a consequence, it repairs multiple faults by means of low hardware cost.*

1. Introduction

Traditionally, memory repair is performed by using external equipment to test the memory, localize the faults, and drive a laser beam to perform the repair. Electrical fuses or anti-fuses can also be used to avoid laser beam, but again an 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 a reasonable test and repair cost for embedded memories. As an additional advantage BIST and BISR can test and repair embedded memories at any time during the product life [1]. This reduces maintenance cost, and increases reliability and product life. Various BISR approaches have been developed. Row (or word) BISR replaces faulty regular rows (or words) by spare rows (or words). Column BISR replaces faulty regular columns by spare columns. Data-bit BISR uses a memo ry part generating a spare data bit to replace a memory part generating a regular data bit. In this work we consider the data-bit BISR and the column BISR schemes. Data-bit BISR can repair faulty cells and faulty columns, as well as faulty column-MUXes, faulty read and write amplifiers, and faulty data input/output latches. Repairing read amplifiers may improve yield significantly, since the sense amplifiers are very sensitive circuits and can be faulty more frequently than other parts. Column repair can repair faulty columns, and also entire faulty bits, if it disposes sufficient spare columns. Repairing faulty columns in addition to the faulty cells is important since a large number of defects may result on a faulty column. For instance, any defect creating a stuck-on fault on one of the two access transistors of an SRAM cell will result on a faulty column, since the cell is permanently connected to the bit line. On the other hand, row and word repair cannot repair these important classes of faults.

Row/word repair is the simplest BISR approach. Thus, the majority of the previous works consider this scheme, although column repair was the predominant external repair scheme used for stand-alone memories. Word repair was early proposed by K. Sawada et al [2]. It uses a content addressable memory for storing the data and addresses of the faulty words. The work considered low numbers of faults. Also, faults in the spare parts are not considered. Subsequent works on word BISR [3] [4], [5] use the same basic scheme but improve various implementation aspects. These works too consider a low number of faults (e.g. two faults in [4]), and no faults in the spare units. A more recent work [8] uses nonvolatile memory cells to fix once forever the repair of manufacturing faults. Fixing manufacturing faults is also treated in [1], where a taxonomy of various ways to repair and fix manufacturing faults and field faults is presented.

Work on column/data BISR is more recent due to the difficulty for elaborating the reconfiguration functions for this repair. Kim et al [6] present the first column repair scheme. In this scheme 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 session. That is, the memory is tested until a 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 will become 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 storage cells for storing the reconfiguration information.

Another paper [7] 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 performing the memory reconfiguration.

Finally a more recent paper [9] presents optimal reconfiguration functions for data-bit repair. The derived functions perform repair for multiple faults affecting both the regular and spare elements, minimize the hardware cost for implementing the repair control and for storing the reconfiguration information, and perform the repair by means of a single test pass. This optimises the BISR cost and the repair efficiency. In addition, the scheme does not require modifying the memory structure, since the repair circuitry is placed around the memory. Thus, the repair is compatible with standard memory compilers.

In the present paper we present a dynamic repair scheme that increases the repair efficiency of the scheme presented in [9]. This is done by using a single spare unit for repairing faults affecting several regular units. In addition the dynamic scheme is enhanced to allow using spare units of a size smaller than the regular units (the memory parts generating a data-bit). This is important for achieving low repair cost in memories having words of medium or small size. For such memories, adding even a single spare bit, as required by the basic dynamic repair approach, leads in a significant area cost. The enhanced scheme shows a dramatic decrease of the area cost.

2. Static data-bit BISR

Data-bit BISR is a repair approach that determines the data-bit positions in which the memory generates erroneous data, and replaces the defected parts connected to these bit positions by spare parts. Thus, a replaceable part is the set of memory columns connected through the column multiplexer to a data input/output of the memory. For simplicity, we will call the regular replaceable parts as regular units and the spare replaceable parts as spare units. We can use k spare units in order to be able to replace up to k faulty regular units. This scheme is very flexible, since we do not need to modify the memory structure for implementing it. Instead, we can use a standard memory compiler to generate a memory having a word length of n+k bits. In addition, we generate a BISR circuitry external to the memory, able to capture the locations of the faulty data-bits and replace them by fault-free data-bits. Such a scheme has been presented in [9].

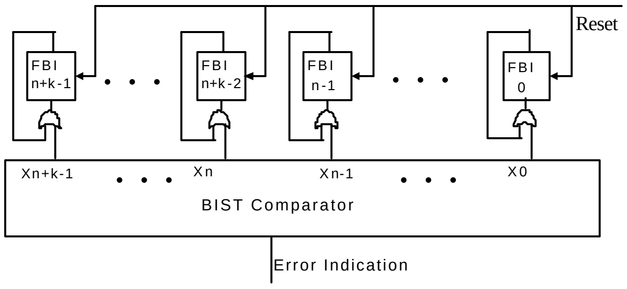

Figure 1: Generation of the state of the FBI latches by means of the BIST comparator

Figure 1 shows the circuit used for locating the faulty-bit positions. It consists on n+k latches (faulty-bit indication - FBI- latches) and n+k OR gates. The input of each latch is generated by an OR gate that receives as inputs the output the latch and the output of one XOR gate of the BIST comparator. During the test phase, the whole memory (regular and spare bits) is tested and the latch corresponding to any of these bits is set to 1 when an error is detected on this bit by the corresponding XOR gate of the BIST comparator. In this scheme we used n+k faulty-bit indication latches. This way, we can locate both the regular and spare faulty bits, and use only fault-free spares to replace the faulty regular bits.

We number the functional units (regular bits) from 0 to n-1 (U0, U1, …, Un-1) and the spare units (spare bits) from n through to n+k-1 (Un, Un+1, …, Un+k-1). To perform repair, a set of MUXes is used to replace the faulty bits by spare ones. The reconfiguration can be done in a local manner. In this case, a faulty unit is replaced by its left-side closest fault-free unit, as illustrated in figure 2, with n=4, and k=3, where the MUX of position i is connected to the memory output of position i, and to the memory outputs of positions i+1, i+2, …, i+k. A distant repair is also possible as illustrated in figure 3.

To generate the control signals of the MUXes in figures 2 and 3, we need to implement the reconfiguration functions that generate these signals in response to the states of the faulty-bit indication latches of figure 1. To implement a dynamic scheme, multiple copies of these functions have to be used, as we will see later. Thus, it is important to dispose low cost reconfiguration functions, to make the dynamic scheme practical. [9] proposes compact implementations for these functions. For the local repair, these functions are described by the recursive equations (1) and (2), where FBI_i is the state of the faulty-bit indication latch of position i, and M_j^i is the control signal of input j, of the MUX of position i, and the MUXes are implemented without internal decoders (the signals M_j^i are already decoded).

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

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

The reconfiguration functions for the distant repair, are described by the recursive equations (3), (4), (5), and (6). In this case we the regular units are numbered RU_0, RU_1, …RU_{n-1}, the spare units are numbered SU_1, SU_2, …, SU_k. The states of the corresponding fault location latches are noted RF_0, RF_1, …, RF_{n-1} and SF_1, SF_2, …, SF_k. The variables F_j are intermediate variables used for convenience.

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

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

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

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

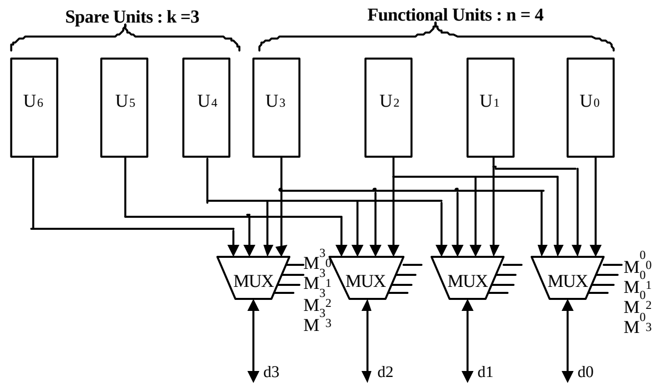

Figure 2: The local repair scheme.

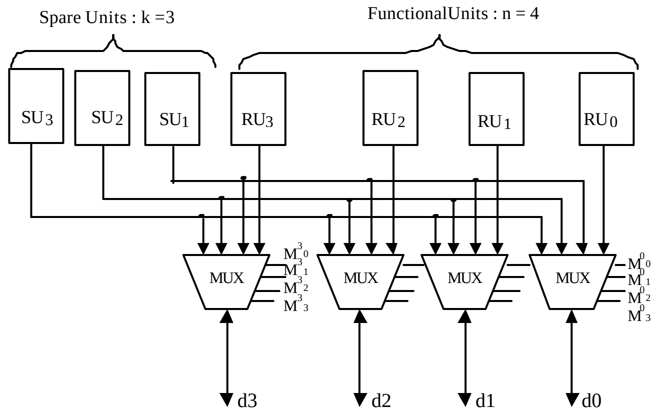

Figure 3: The distant repair scheme.

3. Dynamic data-bit BISR

A major attribute of a repair scheme is the size of the replaceable units. This size impacts the extra area required for repairing a fault. If the replaceable unit is too small, then, we will have a large number of such units, and the area occupied by the reconfiguration logic and the interconnections added for localizing and replacing the faulty units will be very large. For instance, if the replaceable unit is the cell, then the number of possible faulty locations is very high, requiring a large amount of memory for storing this information, and a large amount of routing area together with a very complex reconfiguration logic. On the other hand, if the size of the replaceable unit is very large, then, to repair a single faulty-cell within a replaceable unit will require to use a spare unit of large size, making high the repair cost per faulty-cell. Therefore, there is an optimal size of the replaceable unit for each memory size and for the target number of repairable faults. However, once this optimal size is determined, it is not always easy to implement the repair scheme using this replaceable unit size, due to topological constraints of a memory design. In fact, memories have a very regular and compact layout. Inserting within this layout extra routing and extra logic for performing the repair may break this regularity and increase the memory size considerably. In addition, this approach requires a considerable effort to modify the memory compilers. Thus, it is suitable to repair the memory by using access only to the memory inputs and outputs. The static data-bit repair scheme presented [9] has this advantage, but it uses a large spare unit to repair a single fault (a spare block includes a number of cells equal to the number of memory words). To reduce the size of repairable units, while performing the reconfiguration by accessing only the external inputs/outputs of the memory, we propose a dynamic repair. It modifies dynamically the control signals of the MUXes, so that they use a spare bit to replace a faulty regular bit only for a subset of the memory addresses and other faulty regular bits for other subsets of the memory addresses. To explain this scheme let us first consider the static scheme of figure 4.

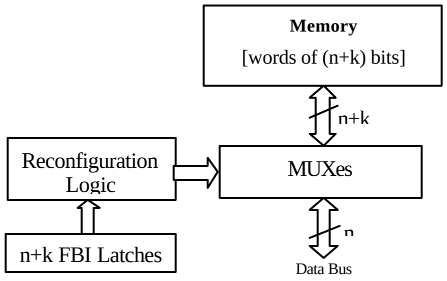

Figure 4. Static data-bit repair scheme

In this figure, the memory includes *n+k* blocks of cells, each corresponding to one functional or one spare data bit. The reconfiguration circuitry includes n+k latches storing the fault location information (n+k FBI latches), n MUXes allowing to connect the n bits of the data BUS to the fault-free regular and spare data bits of the memory, and a reconfiguration logic that generates the control signals of the MUXes. The repairable unit is the block of cells connected to a data-bit input/output. Thus, its size is equal to N_{mw} cells, N_{mw} is the number of memory words. To divide the size of the repairable units by a factor *R*, the new scheme uses *R* blocks of n+k FBI latches as shown in figure 5. In this figure, during the test phase, we use r address bits (A_1 through to A_r) to select a different set of latches for each value of these address bits. Thus, for each of the $R = 2^r$ values of these bits, a different set of FBI latches stores the fault-location information.

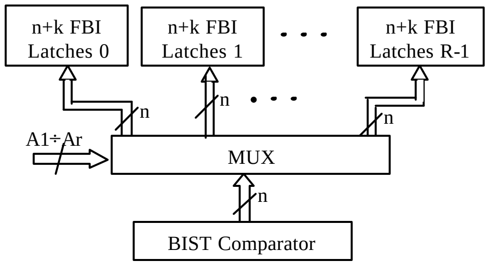

Figure 5: Selecting a block of FBI latches during the test phase

During the regular operation of the system, the address bits A_1, A_2, …, A_r, are monitored to determine which block of FBI latches will be selected to drive the reconfiguration of the memory during each memory access cycle, as shown in the figure 6.

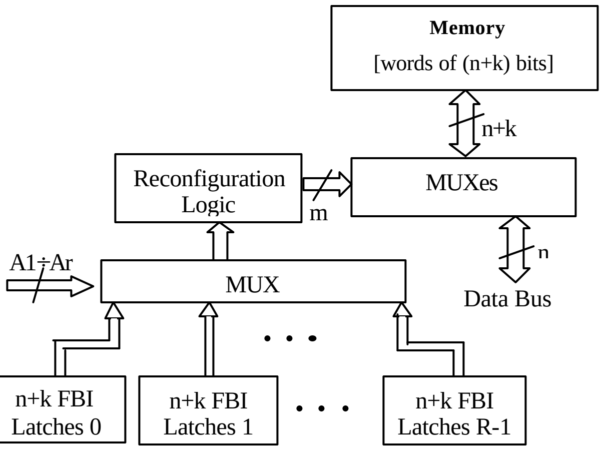

Figure 6: Selecting a block of FBI latches during normal operation

In figure 4, we note that during the normal operation, the outputs of the reconfiguration functions are constant, since the inputs of these functions are fixed at constant values during the repair phase. Thus, there is no delay added to the memory operation due to the computation of the output values of these functions. On the other hand, in figure 6, the inputs of the reconfiguration function are not fixed, introducing extra delay in the memory operation. To avoid this delay, we replicate the reconfiguration functions R times (blocks RFL_0, RFL_1, … RFL_{R-1}), and place the multiplexers controlled by the address bits A1, A2, …, Ar on the outputs of these blocks (see figure 7).

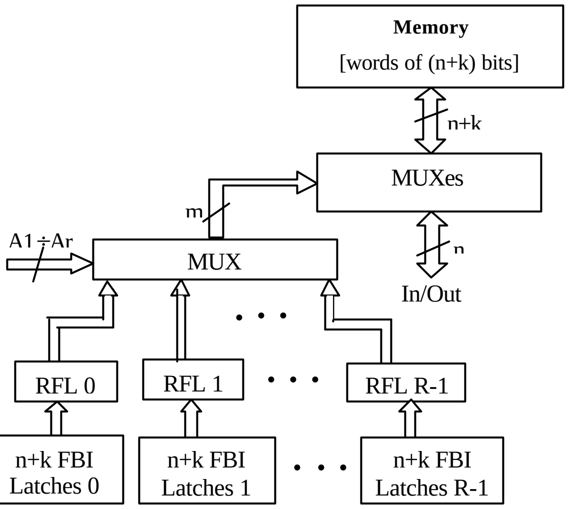

Figure 7: A faster scheme for dynamic repair

In this case, the hardware cost is higher, since we use R reconfiguration logic blocks instead of 1, but the reconfiguration speed is higher, since the inputs and the outputs of the reconfiguration functions are fixed during the repair phase. Thus, the longest path of the reconfiguration circuit includes the two MUXes, while in figure 7 it includes the two MUXes, and the reconfiguration functions.

3.1 Trade-offs in repair parameters, and experiments

Using the dynamic reconfiguration scheme described above, we can increase the repair efficiency since the size of the repairable unit is divided by $R = 2^r$. Thus, the size of the repairable units can become arbitrarily small. For instance, by using all the address bits of the memory to drive the MUX used in the dynamic reconfiguration, the size of the repairable unit is reduced to a single cell. From this point of view, the repair becomes optimal, but the cost of the reconfiguration circuitry becomes excessive since it increases linearly with R. There is an optimal R that gives the best cost trade-offs. Consider for instance that we need to repair up to 8 faults in a 1 Mbit memory having a 32-bit word size. The block of cells connected to each data input/output includes 32K cells. This is the size of the repairable unit used by the static reconfiguration scheme. Thus, to repair 8 faults we need 8 spare repairable units corresponding to 256K spare cells. By selecting r = 1, we divide by 2 the size of the spare units , so we will need 128 K spare cells (a gain of 128 K spare cells,) but we will need 2x40 FBI latches instead of 40 FBI latches (an increase of 40 FBI latches) for the scheme of figure 6 (and also a second block of reconfiguration logic for the scheme of figure 7). The improvement is significant since we have a gain of 128 K memory cells by paying 40 extra latches and a block of reconfiguration logic. If we use r = 2, then, with respect to the case of r = 1 we gain 64 K memory cells and we pay 80 extra latches (and also two extra blocks of reconfiguration logic for the scheme of figure 7). The gain is still important, but it is divided by 2, while the cost of the extra circuitry is multiplied by 2. This happens each time we increment r. So, at some point the gain becomes less important than the extra cost. This point corresponds to the optimal implementation of the scheme. Determining this point will enable an optimal repair. This analysis is pertinent for memory technologies affected by very high defect densities, as expected for nano-technologies [10] [11], where we will need to repair huge numbers of faults. However, for the moderate numbers of faults affecting CMOS ICs, the optimal point will correspond to k=1 and R=Nf (Nf being the maximum target fault multiplicity). Table 1, shows the area overhead required for applying the dynamic scheme for various values of k and r, as well as for the static scheme (r=0) for various values of k. The considered memory is a 64K X 32 SRAM. The area overhead is estimated for a commercial 0.18 micron CMOS technology. The area cost of the reconfiguration functions and of the routing is estimated using the

AMBIT tool. The maximum number of repairable faults is equal to $k2^r$. Thus, we can use k=1, r=3, to repair a memory including up to 8 faults, at an extra area of 3.4%, while using k=2, r=2, will repair the same number of faults at a much higher area cost (6.1%). We also observe that the static scheme for r=0, k =6, can repair only 6 faults at a drastically higher area cost (17.44%).

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

r	***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

Table 2. Area overhead of various BISR implementations for a 256K x 8 SRAM

r	k = 6	k = 5	k = 4	k = 3	k = 2	k = 1
0	74.27	61.93	49.55	37.13	24.72	12.27
1	74.37	62.01	49.63	37.19	24.77	12.31
2	74.56	62.17	49.78	37.3	24.87	12.39
3	74.96	62.49	50.08	37.52	25.07	12.55

Table 2 shows the area cost for a memory using 8-bit words. We observe that in this case, the minimum cost, required for repairing a single fault (k=1, r=0) is 12.27%. This cost is much higher, than the cost required for the 32-bit word width memory, even when we use the later to target a much higher fault multiplicity. When using the dynamic repair approach, this cost increases very slightly for repairing a much higher number of faults (e.g. 12.55% for repairing 8 faults, when k=1, r=3).

4. Dynamic repair using spares of reduced size

In the previous data-bit repair schemes, a spare unit includes a number of cells equal to the number of memory words N_{mw}. Thus, the minimum area overhead (use of a single spare unit) is equal to (#spare unit cells)/(#regular memory cells) %= N_{mw} /(N_{mw} x N_{wc}) % = 1/ N_{wc} %, where N_{wc} is the number of regular cells per memory word. Thus, the minimum area overhead can be significant for memories with medium or small word size. To reduce this cost, we need to reduce the size of the spare units. This section describes an extension of the dynamic repair scheme that allows achieving this goal. The scheme is illustrated in figures 8 and 9. We use n regular units and k spare ones. As for figures 6 and 7, we use r address bits to perform dynamic repair, but we use spare units with size equal to the $1/2^r$ of the size of the regular units. We illustrate this scheme by considering that each spare unit is composed of one memory column. In this case we select r to be equal to the number of the address bits of the column decoder (bits A1÷Ar), but the scheme is valid for any other value of r. From this choice, each regular unit includes R= 2^r columns. We use R sets of FBI latches for the regular positions, but only one set of FBI latches for the spare positions (since each spare position is composed of a single column). The scheme is described for local repair. A similar approach can be used for distant repair.

Figure 8 illustrates the operation during the test phase. Similarly to the dynamic scheme of figure 5, during this phase, the current value of the address signals A1÷Ar connects one set of FBI to the outputs of the XOR gates of the BIST comparator. This is done by means of a MUX controlled by the signals A1÷Ar. This arrangement is used only for the regular positions. On the other hand, no MUX is employed for the k spare positions, since we use only one set of FBI latches for these positions.

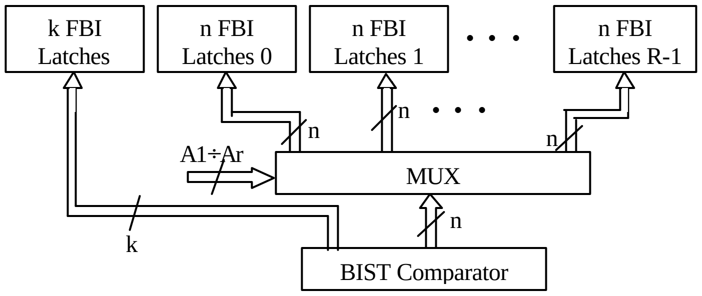

Figure 8: Selection of the FBI latches during test

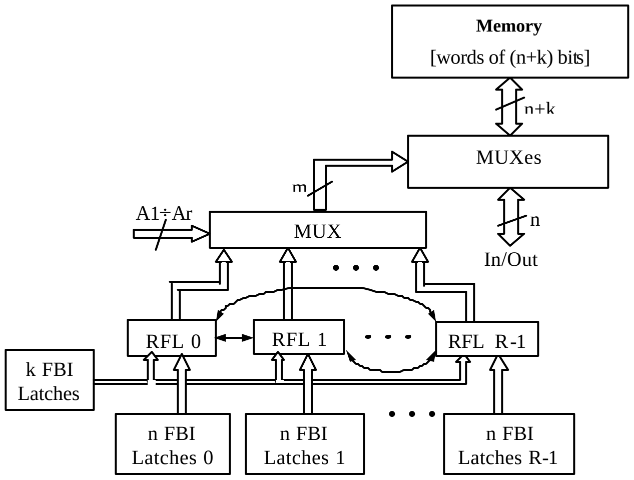

Figure 9: Interactions of reconfiguration functions in dynamic repair using single-column spares

Figure 9 illustrates how the repair is performed during the circuit operation. This operation is similar as for the dynamic scheme of figure 7. However, in figure 7, any two different blocks of reconfiguration logic (blocks LFL i) have inputs coming from two different sets of FBI latches. On the other hand, in figure 9, the different blocks of reconfiguration logic share some inputs (i.e. the outputs of the k FBI latches of the spare positions). Also, in figure 7, the blocks of the reconfiguration logic do not exchange any information (they implement mutually independent functions. On the other hand, in figure 9, the blocks of reconfiguration logic implement interdependent functions. Thus, they exchange some information.

Let us now determine the interdependent reconfiguration functions used in figure 9. Due to the interdependencies, deriving these functions becomes quite complex. To simplify this task we introduce some intermediate variables. Thanks to these variables we are able to use the reconfiguration functions described in the section 2. Let FBi be the state of the FBI latch of the spare position i ($n \le i \le n+k-1$). Let FBi^{q} be the state of the FBI latch used for the column q of the regular position i ($0 \le i \le n-1$). Let $Mj^{y,x}$ be the control variable of the MUX of position y that indicates if the column x of the position y has to be shifted by j positions. $Mj^{y,x}$ is computed by the reconfiguration logic RFLy.

We introduce the intermediate variables $FBi^{b,q}$. This variable is used only by the reconfiguration logic RFLq. For positions that dispose the complete set of columns (i.e. for the regular positions), we have $FBi^{b,q} = Fbi^{q}$ **(7)**.

For positions that dispose only one column (i.e. the spare positions) $FBi^{b,q}$ is defined as follows. $FBi^{b,q}$ takes the value 1 to indicate that the position i is not available for a possible shift of the column q of position b, due to one of the following reasons:

- the position i is faulty,
- the position i is already occupied by a faulty column of any position lower than b, excepting the columns of rank q, or by a column of the position b, having rank lower than q.
- In all other cases the value of $FBi^{b,q}$ is 0. In this definition of $FBi^{b,q}$ we have not considered the occupation of the position i by a columns of rank q, because this occupation is treated by the functions $Mj^{y,x}$ as defined in section 2. From this definition of $FBi^{b,q}$ for the spare positions, we obtain:

(8) $$FBi^{b,q} = FB_i + \sum_{\substack{x=1 \\ x\ q}}^{R} \sum_{y=i-k}^{b-1} Mi\text{-}y^{y,x} + \sum_{x=1}^{q-1} Mi\text{-}b^{b,x}$$

Then, by replacing the variables FBi^{b} by the variables $FBi^{b,q}$ in the equations (1),(2) of the variables Mj^{y} for the static reconfiguration scheme described in section 2, we obtain the equations of the variables $Mj^{y,q}$, generated by the reconfiguration functions of figure 9. These equations are:

(9) $M_0{}^{1,q} = \neg FB_1{}^{1,q}$, $M_1{}^{1,q} = FB_1{}^{1,q} \neg FB_2{}^{1,q}$,
$M_2{}^{1,q} = FB_1{}^{1,q} FB_2{}^{1,q} \neg FB_3{}^{1,q} \ldots\ldots$
$M_k{}^{1,q} = FB_1{}^{1,q} FB_2{}^{1,q} \ldots\ FB_{k-1}{}^{1,q} \neg FB_k{}^{1,q}$
$M_j{}^{i+1,q} = M_j{}^{i,q} \neg FB_{i+j+1}{}^{i+1,q} + M_{j-1}{}^{i,q} FB_{i+j}{}^{i+1,q}$
$\neg FB_{i+j+1}{}^{i+1,q} + M_{j-2}{}^{i,q} FB_{i+j-1}{}^{i+1,q} FB_{i+j}{}^{i+1,q} \neg FB_{i+j+1}{}^{i+1,q}$
$+ \ldots + M_0{}^{i,q} FB_{i+1}{}^{i+1,q} FB_{i+2}{}^{i+1,q} \ldots FB_{i+j}{}^{i+1,q} \neg FB_{i+j+1}{}^{i+1,q}$

The equations 7, 8, and 9, define the reconfiguration functions used in figure 9. A last problem concerns the fact that the spare columns are not using any column decoder. Thus, they are not isolated from the write amplifiers and will be accessed during each read and write cycle. While the access during a read cycle may not be a problem, this access may destroy the contents of the cells of the spare columns during the write cycles. To cope with we can use one of the following two solutions:

- Use the signals $Mj^{y,x}$ enabling the connection of a spare column to the data input/outputs to activate the write amplifier. In this case the write amplifier is activated only when the spare column is connected to some data input/output,
- Place a write amplifier to each branch of a MUX connecting a data input to the spare columns, but before the transistors of the MUX that create this connection. Thus, each spare column will be driven by a write amplifier only when one of these transistors is on (for activating the connection of the spare column). This is illustrated in figure 10, where the data input d3 is connected through a MUX controlled by $M^3{}_0$, $M^3{}_1$, $M^3{}_2$, and $M^3{}_3$, to the regular unit U3 and the spare units U4, U5, and U6.

The first solution is preferable since it uses a lower number of write amplifiers and avoids an increase of power dissipation. The read amplifiers of the spare columns can also be activated selectively in a similar manner, but this is not necessary as said above.

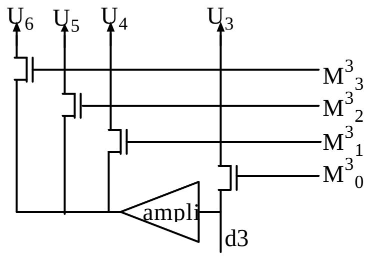

Figure 10. Write amplifier for the spare units

The scheme presented in figures 8, 9, allows using a single column as spare unit. However, with this scheme, each spare column can be used to repair a fault affecting a single column. To enable using a single spare column for repairing faults affecting several columns, we can add a second level of dynamic repair, as illustrated in figure 11. For doing so, we use $Q=2^{q}$ copies of the k FBI latches used in figures 8, 9 for the k spare units, and Q copies of the R sets of n FBI latches used in these figures for the n regular units. We also use Q copies of the R blocks of the reconfiguration functions shown in figure 9 and described by equations 7,8,9. Each of these copies is fed by the outputs of one of the Q copies of the sets of FBI latches, and provide Q copies of Rxm output signals. A MUX controlled by the address bits A1÷Ar reduces these signals into Q copies of m signals. A second MUX controlled by the address bits Ar+1÷Ar+q, reduce these Qxm signals into the m signals that control the multiplexers used to reconfigure the n+k regular and spare units. Note that the two MUXes controlled by the address bits A1÷Ar and Ar+1÷Ar+q, can be combined into a single MUX controlled by the address bits A1÷Ar+q.

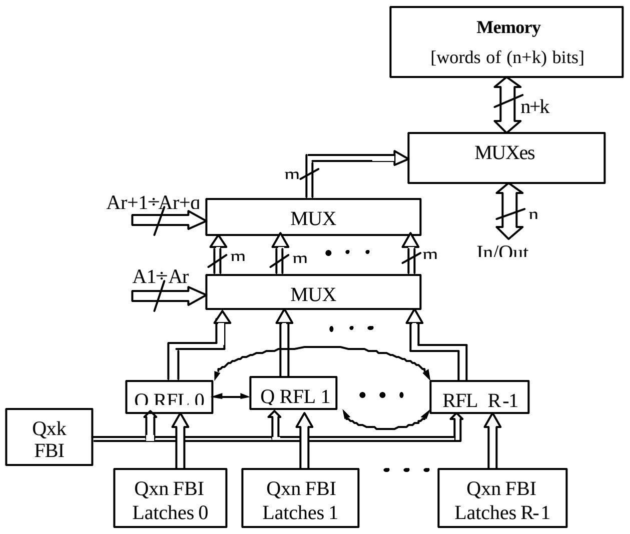

Figure 11. Dynamic repair with a second-level multiplexing

Table 3 shows the area overhead for a memory using 8-bit words. This scheme repairs $k2^r2^q$ faults. We observe a dramatic reduction of the area cost with respect to the dynamic scheme of section 3 (table 2). For instance, for repairing up to 64 faults, this scheme requires an area overhead of 5.13% (k=1, r=3, q=3), while for the same memory the dynamic scheme of section 3 requires an area overhead of 12.27% for repairing only one fault (k=1, r=0).

Table 3. Area overhead of various BISR implementations using spares of reduced size, for a 256K x 8 SRAM

k	r	q = 1	q = 2	q = 3
1	1	6.47	6.69	7.13
1	2	3.56	4.01	4.90
1	3	2.45	3.34	5.13
2	1	12.78	13.06	13.62
2	2	6.81	7.37	8.49
2	3	4.24	5.36	7.6
3	1	19.05	19.35	19.97
3	2	9.98	10.59	11.82
3	3	5.91	7.14	9.6
4	1	25.42	25.84	26.68
4	2	13.34	14.18	15.86
4	3	7.93	9.61	12.97
5	1	31.69	32.13	33.01
5	2	16.51	17.4	19.2
5	3	9.6	11.39	14.97

5. Conclusions

This paper presents a data input/output dynamic Built-In Self-Repair scheme able to repair memories at the data input-output level. The dynamic approach allows to use each spare unit for repairing faults affecting multiple regular units, thus, increasing drastically the repair efficiency. A drawback of the scheme is that the size of each spare unit is equal to (#memory cells)/(#word cells). It results in a significant area overhead for memories with small and medium word width. Therefore, an extension of the dynamic BISR scheme is also presented. It allows using spare units of any desired size. It reduces dramatically the area overhead for any word width, and more particularly for small and medium word widths.

References

[1] Zorian Y., "Embedded Memory Test & Repair: Infrastructure IP for SOC Yield", 2002 IEEE International Test Conference.

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

[3] 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.

[4] Bhavsar D. K., Edmodson J. H., "Testability Strategy of the Alpha AXP 21164 Microprocassor", I994 IEEE International Test Conference.

[5] 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.

[6] 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

[7] 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

[8] V. Schober, S. Paul, O. Picot, "Memory Built-In Self-Repair using redundant words", 2001 IEEE Intl Test Conference.

[9] M. Nicolaidis, N. Achouri, S. Boutobza, "Optimal Reconfiguration Functions for Column or Data-bit Built-In Self-Repair", 2003 Design Automation and Test in Europe (DATE'03), March 3-7, 2003, Munich, Germany.

[10] Heath J.R., Kuekes P.J., Snider G.S., Stanley Williams R., "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology", SCIENCE, Vol. 280, June 12, 1998

[11] M. Nicolaidis, N. Achouri L. Anghel, "Memory Built-In Self-Repair for Nanotechnologies", 2003 IEEE International On-Line Testing Symposium.