A decoupled bit shifting technique using data encoding/decoding for DRAM redundancy repair

Kyu Hyun Choi\textsuperscript{1}, JaeYung Jun\textsuperscript{1}, Hokwon Kim\textsuperscript{1}, Seon Wook Kim\textsuperscript{1}, and Youngsun Han\textsuperscript{2a)

\textsuperscript{1} School of Electrical Engineering, Korea University, Seoul, Korea
\textsuperscript{2} Department of Electronic Engineering, Kyungil University, Gyeongsan, Korea
a) youngsun@kiu.ac.kr, Corresponding Author

Abstract: Redundancy repairs are commonly used to support fault tolerance in DRAM systems and recently, the processor performance has been greatly improved, so DRAM access latency has become an important issue. However, existing redundancy repairs using shift logic have difficulty in further reducing the latency due to their design limitations. In this paper, we propose a novel, decoupled bit shifting technique that uses data encoding and decoding to resolve this limitation. Our technique decouples the conventional shifting logic into two units, a bit selection vector generator (BSVG) and a data manipulation unit (DMU), to reduce the latency overhead of the shifting logic. Our technique can apply the BSVG in parallel with other logic consuming long latency operations, thereby reducing the total latency compared to conventional shifting logic. We implement both serial and parallel approaches to demonstrate that the parallel approach performs significantly better than the serial one in terms of delay, area, and dynamic power consumption. The experimental results show that our bit shifting technique is applicable for redundancy repair technique in state-of-the-art DRAM architectures.

Keywords: bit shifting, redundancy repair, encoding, decoding, DRAM

References

[1] W. K. Huang, \textit{et al.}: “New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. \textbf{9} (1990) 323 (DOI: 10.1109/43.46807).

[2] M. Horiguchi: “Redundancy techniques for high-density DRAMs,” Proc. Second Annual IEEE International Conf. Innovative Systems in Silicon (1997) 22 (DOI: 10.1109/ICISS.1997.630243).

[3] J. Hirase: “Yield increase of VLSI after redundancy-repairing,” Proc. 10th Asian Test Symposium, 2001 (2001) 353 (DOI: 10.1109/ATS.2001.990308).

[4] M. Horiguchi, \textit{et al.}: \textit{Nanoscale Memory Repair} (Springer, New York, 2011) 19 (DOI: 10.1007/978-1-4419-7958-2).
1 Introduction

Redundancy repair techniques are widely employed in modern DRAM systems to recover from various faults and to prevent system failures [1, 2, 3, 4]. Such repair techniques are designed to substitute spare elements for faulty elements, particularly permanent faults due to manufacturing defects. Several implementations are possible for such redundancy repair, but one of the most effective methods is to employ bit shifting with switch circuits [5]. Fig. 1 depicts the basic concept of a conventional shift redundancy technique using a switching method. The bitline of each data block is selected according to the column select line (CSL), and the switch control logic chooses fault-free bitlines from the selected bitlines based on the result of the address comparator to connect them to I/O lines. Although the conventional technique is intuitive, it has a significant drawback in that the total column access latency increases because the critical path includes the process of accessing the shift point registers and comparing a column address with the registers. To overcome the latency limitation of the conventional technique, we propose decoupled redundancy repair that employs encoding and decoding methodology. We decouple the conventional shifting logic into two different units: a bit selection vector generator (BSVG) and a data manipulation logic (DMU). The BSVG generates a bit selection vector by comparing a column address with the fault addresses, and the DMU manipulates the data according to the vector. Due to the decoupled design, the latency of the BSVG could be hidden within the latency...
of column address decoding. The DMU encodes/decodes the original input/output data to avoid storing/loading data to/from the defective cells. Our technique is more applicable to general DRAM designs than the conventional technique because the circuit for each unit is quite simple and easy to synthesize. Our proposed technique is fairly scalable because its delay is linearly proportional to the number of redundant elements, regardless of the size of the data width. We suggest two possible implementations for the DMU, a serial approach and a parallel approach. We then evaluate both serial and parallel implementations with their synthetic results.

In summary, this paper makes the following contributions:

- We propose a decoupled bit shifting logic for redundancy repair to reduce the total latency of the DRAM access.
- We introduce two possible implementations of shifting logic that employ an encoding/decoding method and are quite simple and synthesizable.
- We confirm that the parallel implementation is easily adaptable to state-of-the-art DRAM design in terms of delay, area and dynamic power consumption.

This paper is organized as follows: In Section 2, we explain our selective bit shifting and its implementation. The performance of the proposed technique is evaluated in Section 3. Finally, the conclusions are presented in Section 4.

2 Decoupled bit shifting technique

In this section, we describe the details of our decoupled bit shifting technique by using encoding and decoding for redundancy repair in DRAM systems.

2.1 Overall architecture

Fig. 2 shows the overall architecture of our proposed technique to replace faulty elements with redundant ones. The decoupled bit shifting technique consists of three main components: a fault address table (FAT), a bit selection vector generator (BSVG), and a data manipulation unit (DMU).
First, the FAT contains defective row/column addresses, that are found during a post-manufacturing test with ATE, in a sorted order [6, 7]. The FAT provides defective addresses to the BSVG, similarly to the shift point register of the conventional technique [5]. Second, the BSVG uses defective addresses from FAT to generate bit selection vectors that are used to manipulate the input data. The bit selection vector is a chunk of bits, each of which indicates whether its corresponding bit of input data is shifted by 1 or not in the serial approach or indicates which bit should be selected in the parallel approach, respectively. Finally, the DMU is designed to transform the input data using the bit selection vectors to recover from the faults. The DMU consists of two parts: an encoder used when writing data, and a decoder used when reading data. The encoder of the DMU encodes the input data so that all input data can be stored in non-defective cells. The decoder of the DMU eliminates the data bits from the defective columns using the bit selection vector, and thus provides fault-free data. The decoupled design of the BSVG and DMU increases the design flexibility and reduces the total DRAM access latency by arranging each unit separately. The address comparison process for BSVG consumes a relatively long time in the whole redundancy repair process. By conducting the address comparison process in parallel with other long latency processes, i.e., column access, the BSVG does not incur any additional latency overhead. For instance, if there are faults in the second and fifth of six resource elements, the defective row/column addresses are 1, 4, respectively, and the address is stored in the FAT. When the device tries to read the fault address 1, in the first stage, the FAT delivers the address to the BSVG, and the BSVG generates the bit selection vector using the address. The latency of this processes could thus be hidden because it can be simultaneously performed with the decoding process for the column address. When the I/O gate outputs the read data from the sense amplifier in the second stage, the DMU decodes the read data using the bit selection vector to generate the final output data. In the following subsections, we describe serial and parallel approaches to implement our bit shifting technique.
2.2 Serial approach

In the serial approach, the bit selection vector indicates which bit has a fault. The DMU consists of a chain of 1-bit shifters, and each shifter shifts the input data according to the bit selection vector. If there is a fault in an element, the 1-bit left shifter that corresponds to the element performs a shift. Otherwise, it does not shift the input data. By selectively shifting data according to the bit selection vector, all bits corresponding to fault elements are redundantly copied so that the entire data are stored to clean elements. Fig. 3 shows an example of encoding 6-bit input data into 8-bit encoded data using the serial approach when faults occur at the second and fifth bit. In this example, the serial approach employs six 1-bit left shifters, and each of them is serially deployed. The bit selection vector contains six bits; each bit indicates whether its corresponding element is faulty or not. The input data $D_5 D_4 D_3 D_2 D_1 D_0$ is encoded as $D_5 D_4 D_3 D_2 D_1 D_0$ by selectively shifting bits to prevent the loss of $D_1$ and $D_3$. Thus, the encoded data can be generated to avoid faults. The decoding process is similar to encoding, but proceeds in the opposite way, by shifting bits to the right by 1.

Although the serial approach can be implemented intuitively, it suffers from a serious limitation in that its delay of the DMU is linearly proportional to the bit width of the data. Therefore, in this paper, we present a parallel approach to alleviate the delay problem by shifting bits in parallel.

2.3 Parallel approach

The basic idea of the parallel approach is that an input bit is shifted by as many as the number of redundant elements at most, thus output bits are determined with input bits by as many as the number of redundant elements. In the parallel approach, the BSVG generates multiple bit selection vectors using the algorithm, and the DMU selects the appropriate output data. Fig. 4 depicts an example of the DMU implementation in which 6-bit input data are encoded in parallel to repair
faulty elements at the second and fifth bit positions, as in the previous serial example. Unlike in the serial approach, all bits of encoded data are concurrently calculated by employing multiple bit selection vectors. The number of bit selection vectors is equal to the sum of the data width and the number of redundant elements. The output encoded bits are produced by performing bitwise AND and bit-reduction OR operations between a bit selection vector and the partial input data. The bit width of the partial input data to calculate an output bit is completely dependent on the number of additional redundant elements occupied for redundancy repair. Specifically, both $N+1$ bits of the bit selection vector and the partial data are required to compute an output bit where $N$ is the number of redundant elements. In this case, the partial input data for an output bit of index $i$ are derived by concatenating the input bit of index $i$ and its previous $N$ bits. For example, in the case of two redundant elements, if the index of an encoded output bit is 3, the partial input data consists of $D_3D_2D_1$. Therefore, the critical path of the logic design of the DMU in the parallel approach is determined by the number of redundant elements, not the bit width of the input data. In general, the number of redundant elements is considerably smaller than the data width, so the parallel approach can significantly reduce the delay compared to the serial approach.

Algorithm 1 describes the generation of a bit selection vector. $FP$ is a list of the defective column addresses in FAT, $V$ is a list of bit selection vectors, $i$ is the index of the bit selection vector to be calculated, and $EN$ specifies whether to perform encoding or decoding. The subscript in the algorithm refers to the corresponding element of the list. At first, the algorithm initializes $pos$ and $k$ to $i$ and 0, respectively. If $EN = 1$, then $k$ is incremented by 1 when $pos$ is greater than or equal to the $k$-th $FP$ ($FP_{k-1}$). Otherwise, $k$ is incremented by 1 if $pos$ is greater than or equal to $FP_k - k$. The operations to increment $k$ are repeated while $FP_k$ remains valid. Finally, after the loop completes, $V_i$ is derived by shifting 1 to the left $k$ times. The logical equations for encoding and decoding the bit of index $i$ with the bit selection vector $V_i$ are as follows:

![Fig. 4. An example of parallel encoding from Fig. 3.](image-url)
Algorithm 1 Bit Selection Vector Generation Algorithm

**INPUT**
- FP: fault addresses,
- i: Index,
- EN: Encode/Decode signal

**OUTPUT**
- V: Bit selection vector

1: procedure BITSELVECTORGEN(FP, V, i, EN)
2:   pos ← i
3:   k ← 0
4: loop:
5:   if FP<sub>k</sub> is valid then
6:     if EN then // encoding
7:       if pos ≥ FP<sub>k</sub> then
8:         k ← k + 1
9:     else // decoding
10:    if pos ≥ FP<sub>k</sub> − k then
11:       k ← k + 1
12: goto loop
13: Vi ← 1 ⇐ k

\[ E_i = \sum_{j=0}^{n} D_{i−j} \cdot V_{i,j} \quad \text{(where } i < j, D_{i−j} = 0) \quad (1) \]
\[ D_i = \sum_{j=0}^{n} E_{i+j} \cdot V'_{i,j} \quad (2) \]

where \( E_i \) is the encoded bit of index \( i \), \( D_i \) is the decoded bit of index \( i \), and \( V_{i,j} \) is the bit of index \( j \) in the bit selection vector \( V_i \) for the output bit of index \( i \). In addition, \( n \) is the number of redundant elements. By employing these equations, multiple data bits can be encoded and decoded with an extremely low delay in the parallel approach.

Fig. 5 summarizes the previous example of encoding and decoding 6-bit input data using (1) and (2) when there are two faulty elements. In Fig. 5(a), \( D_0 \) is located in front of the first faulty element, so we do not shift the bit left by employing \( V_0 \) of \( \{0, 0, 1\} \). The bit selection vectors \( V_2 \), \( V_3 \), and \( V_4 \) should be \( \{0, 1, 0\} \) to shift \( D_2 \), \( D_3 \), and \( D_4 \) left by 1, respectively, since they exist between the first and second faulty elements. In a similar way, \( V_5 \), \( V_6 \), and \( V_7 \) should be \( \{1, 0, 0\} \). The encoded data is finally obtained by applying (1) with the generated bit selection vectors. Similarly, as shown in Fig. 5(b), the bit selection vectors for decoding indicate the amount of shift to the right needed to restore the encoded data to the original data. In the example, \( D_0 \) has been shifted left by 0, so the bit selection vector \( V'' \) for decoding has to be \( \{0, 0, 1\} \). The vectors of \( D_1 \) and \( D_2 \) are \( \{0, 1, 0\} \) to give a right shift of 1, since they were previously shifted left by 1 to repair the first defective element. The vectors for \( D_3 \), \( D_4 \) and \( D_5 \) become \( \{1, 0, 0\} \).

### 3 Performance evaluation

In this section, we evaluate the performance of our serial and parallel implementations. Tables I and II summarize the result of the performance in terms of the delay,
area, and dynamic power. We evaluated the BSVG and DMU separately, because the units exist in different stages. For the evaluation, we implemented the architecture using Verilog HDL and synthesized it with Synopsys Design Compiler [8] using the Samsung 65 nm CMOS process library. The dynamic power consumption was measured using Synopsys PrimeTime [9]. For accuracy, we applied the switching activities from the simulation to the power estimation. The static leakage power was omitted, because it accounts for less than 1%. We measured the performance by changing the data size from 8 to 64 bits and the number of redundant elements from 1 to 8 bits.

The delay in the BSVG is almost constant (0.937–1.000 ns) in all configurations for both approaches, but the area of the BSVG in the serial approach is only about one-third of that in the parallel one. The sorted fault addresses allow us to process BSVG in parallel, so the delay in both the serial and parallel approaches is constant. Note that the delay of BSVG could be hidden within the latency of the column address decoding, which is significantly longer than 1 ns because the BSVG can work independently regardless of the column access [10] in our decoupled design.

Both the delay and area of the DMU are better in the parallel approach than in the serial one. Unlike BSVG, the DMU delay is critical because the DMU is located on the critical path for the column access logic in DRAM. The parallel approach is 12 times on average and up to 14 times faster than the serial implementation, and the area of the parallel DMU is just about 15% of the serial one. In the parallel approach, we achieved extremely low delays of 0.087–0.244 ns for 1–8 spare columns with a 64-bit data size that is generally used in modern DRAM [11, 12].

\[
i : \text{Bit Index, } j : \text{Vector Index}
\]

| Faulty Element | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
|----------------|----------|----------|----------|----------|----------|----------|
| **Input Data (D)** | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
| **Encoded Data (E)** | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
| **Bit Selection Vectors (V) for Encoding** | \( j \) | \( i \) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

(a) encoded data

| Faulty Element | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
|----------------|----------|----------|----------|----------|----------|----------|
| **Input Encoded Data (E)** | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
| **Decoded Data (D)** | \( D_5 \) | \( D_4 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) |
| **Bit Selection Vectors (V) for Decoding** | \( j \) | \( i \) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(b) decoded data

Fig. 5. An example of encoded/decoded data and bit selection vector for 6-bit data with two redundant elements.
### Table I. Performance of the serial approach.

| Data size [bit] | # of spare elem. | Delay [ns] | Area [µm²] | Dynamic power [mW] |
|-----------------|------------------|------------|------------|-------------------|
|                 |                  | BSVG       | DMU        | Total             |
|                 |                  |            |            | BSVG              | DMU         | Total     |
| 8               | 1                | 0.985      | 0.683      | 1.668             | 61          | 377       | 438       | 0.076   |
|                 | 2                | 0.937      | 0.604      | 1.541             | 120         | 334       | 454       | 0.085   |
|                 | 4                | 0.998      | 0.683      | 1.681             | 266         | 377       | 643       | 0.117   |
|                 | 8                | 1.000      | 0.850      | 1.850             | 563         | 696       | 1,259     | 0.238   |
| 16              | 1                | 0.955      | 0.889      | 1.844             | 76          | 986       | 1,062     | 0.154   |
|                 | 2                | 0.988      | 0.937      | 1.925             | 146         | 1,114     | 1,260     | 0.229   |
|                 | 4                | 0.992      | 1.072      | 2.064             | 319         | 1,181     | 1,500     | 0.339   |
|                 | 8                | 0.998      | 1.228      | 2.226             | 680         | 1,723     | 2,403     | 0.616   |
| 32              | 1                | 0.999      | 1.650      | 2.649             | 104         | 3,229     | 3,333     | 0.472   |
|                 | 2                | 0.996      | 1.700      | 2.696             | 211         | 3,166     | 3,377     | 0.640   |
|                 | 4                | 1.000      | 1.838      | 2.838             | 457         | 3,651     | 4,107     | 1.052   |
|                 | 8                | 1.000      | 2.025      | 3.025             | 959         | 4,290     | 5,249     | 1.654   |
| 64              | 1                | 0.996      | 3.238      | 4.234             | 211         | 10,782    | 10,993    | 1.598   |
|                 | 2                | 0.999      | 3.294      | 4.293             | 289         | 10,547    | 10,836    | 1.970   |
|                 | 4                | 1.000      | 3.367      | 4.367             | 689         | 11,104    | 11,793    | 2.964   |
|                 | 8                | 0.999      | 3.621      | 4.620             | 1,410       | 11,827    | 13,237    | 4.406   |

### Table II. Performance of the parallel approach.

| Data size [bit] | # of spare elem. | Delay [ns] | Area [µm²] | Dynamic power [mW] |
|-----------------|------------------|------------|------------|-------------------|
|                 |                  | BSVG       | DMU        | Total             |
|                 |                  |            |            | BSVG              | DMU         | Total     |
| 8               | 1                | 0.998      | 0.087      | 1.085             | 157         | 60        | 217       | 0.018   |
|                 | 2                | 1.000      | 0.114      | 1.114             | 411         | 87        | 498       | 0.031   |
|                 | 4                | 1.000      | 0.192      | 1.192             | 923         | 124       | 1,047     | 0.051   |
|                 | 8                | 1.000      | 0.230      | 1.230             | 1,825       | 256       | 2,081     | 0.079   |
| 16              | 1                | 0.999      | 0.087      | 1.086             | 276         | 124       | 399       | 0.044   |
|                 | 2                | 1.000      | 0.119      | 1.119             | 661         | 288       | 949       | 0.096   |
|                 | 4                | 1.000      | 0.199      | 1.199             | 1,357       | 562       | 1,919     | 0.172   |
|                 | 8                | 1.000      | 0.243      | 1.243             | 2,993       | 897       | 3,890     | 0.271   |
| 32              | 1                | 1.000      | 0.087      | 1.087             | 423         | 491       | 914       | 0.134   |
|                 | 2                | 0.999      | 0.119      | 1.118             | 865         | 575       | 1,440     | 0.184   |
|                 | 4                | 1.000      | 0.200      | 1.200             | 2,023       | 1,155     | 3,178     | 0.346   |
|                 | 8                | 1.000      | 0.244      | 1.244             | 4,206       | 1,779     | 5,985     | 0.499   |

© IEICE 2017
DOI: 10.1587/elex.14.20170385
Received April 14, 2017
Accepted May 23, 2017
Publicized June 14, 2017
Copyedited July 10, 2017
We also confirmed that the delay of the DMU in the parallel approach does not depend on the data size, but on the number of spare elements. On the contrary, the delay of the DMU in the serial approach depends on the data size. Furthermore, additional dynamic power consumption is negligible compared to that of read/write operations [13, 14]. Note that our evaluations are result of the synthesis tool using standard cells, and if we make our design fully custom, the result could be much better.

4 Conclusion

One of the primary reasons to employ hardwired repair techniques is that redundancy repair is on a timing-critical path in DRAM. For the conventional shifting technique with the switching method, it is difficult to solve problem where the latency increases because its address comparator is placed in the critical path. To resolve this limitation, we proposed a novel redundancy repair technique that decouples the shifting logic using data encoding and decoding, i.e., decoupled bit shifting. We introduced both serial and parallel implementations, and demonstrated that the parallel approach outperforms the serial one in terms of delay, area, and dynamic power consumption. Also, we confirmed that our parallel approach is applicable in modern DRAM systems because the long latency of the BSVG can be hidden within the latency of column address decoding, and the latency of the DMU is extremely low. Furthermore, our experimental results are derived from synthesis tools using standard cells, so the performance of our proposed technique can improve significantly if using a fully custom design. To the best of our knowledge, our encoding and decoding method for redundant repair is the first programmable approach that provides a decoupled architecture that minimizes access latency.

Acknowledgments

This work was supported by the IT R&D program of MOTIE/KEIT. [10052653, Development of processing in memory architecture and parallel processing for data bounding applications]