An Adaptive Code Rate EDAC Scheme for Random Access Memory

Ching-Yi Chen and Cheng-Wen Wu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan

Abstract—As the VLSI technology scaling continues and the device dimension keeps shrinking, memories are more and more sensitive to soft errors. Memory cores usually occupy a large portion of an SOC and have significant impact on the chip reliability. Therefore error detection and correction (EDAC) techniques are commonly used for protecting the system against soft errors. This paper presents a novel EDAC scheme, which provides adaptive code rate for random access memories (RAMs). Under a certain reliability restriction, the proposed design allows more error bits than a conventional EDAC design.

Index Terms—Error correction codes, memory, Hsiao code, fault tolerance, reliability

I. INTRODUCTION

The continuous scaling of VLSI technology and increase of memory density worsen the problem of soft errors. Soft errors are transient errors caused by system noise or radiation events such as alpha particles. In memories, these radiation events cause upsets in memory cells while sufficient charge is deposited [1]. Among the components in a system, memories are one of the parts most sensitive to soft errors [2], [3]. Moreover, in submicron technologies, memory cells can be upset by less charge, there may consequently be higher upset rate than those reported in earlier technologies [4], [5].

For protecting a system against transient errors, error correction codes (ECCs) are commonly used on random access memories (RAMs), where parity bits are appended to each data word and form a codeword. Codewords are written into memories (RAMs), where parity bits are appended to each data word and form a codeword. Codewords are written into RAMs, where parity bits are appended to each data word. A read codeword is decoded and a parity generation matrix (denoted as the $H$-matrix) to check if there are error bits in the data. We can use the corresponding $H$-matrix to check if there are error bits in the data read from memory. An output word $y$ is correct if and only if

$$H \cdot y^T = 0,$$

while the one with an error bit results in a non-zero $r$-bit vector. The non-zero vector, known as the syndrome, can be used to locate the error bit in the output word.

$$p = d \cdot G^T$$

The parity and data form an $n$-bit codeword, i.e., $n = k + r$. We can use the corresponding $r$-by-$n$ parity check matrix (known as the $H$-matrix) to check if there are error bits in the data read from memory. An output word $y$ is correct if and only if

$$H \cdot y^T = 0,$$

while the one with an error bit results in a non-zero $r$-bit vector. The non-zero vector, known as the syndrome, can be used to locate the error bit in the output word.

Fig. 1. An H-matrix for the (22,16) SEC-DED Hsiao code.
The proposed EDAC scheme adopts the Hsiao code, which is considered a better solution in performance and cost among the SEC-DED codes. A Hsiao code’s H-matrix has to meet the following constraints: 1) There is no all-0 column. 2) Every column is distinct. 3) Every column contains an odd number of 1’s [7]. With these constraints satisfied, a distance-4 code is guaranteed, and thus the double-error detection is achieved. An H-matrix for the (22,16) code is shown in Fig. 1, where every column is distinct and contains one or three 1’s. As the figure shows, an H-matrix of the (n,k) code is an r-by-n matrix, which is composed of an r-by-k G-matrix and an identity matrix of size r. For SEC-DED, the length of parity can be determined by the rule

\[ 2^{r-1} \geq k + r. \] (1)

While constructing the G-matrix and the H-matrix for an (n,k) code, Hsiao code suggests two more constraints for less hardware and minimum delay: 1) The total number of 1’s in each column (as known as the column weight) should be a minimum. 2) The number of 1’s in each row (known as the row weight) should be made as balanced as possible. For implementation, the G-matrix is used in encoding and the H-matrix in decoding, while the 1’s in these matrices represents fan-ins to XOR gates for the encoding and checking process. Accordingly, more 1’s cost more hardware, and larger Hamming distance between two rows in G-matrix/H-matrix results in timing skew between the parity generating paths in the encoder/decoder circuits. As a result, these constraints lead to the least XOR gates in the XOR tree and a balanced number of gates in each path of the encoder. That is the reason why a Hsiao code codec has lower area overhead and timing latency than a conventional Hamming code codec.

III. PROPOSED ADAPTIVE CODE RATE EDAC SCHEME

Most RAMs have I/O data word width of 16-bit or 32-bit, and conventional ECC techniques in general are not suitable for such memories since the data word width is short and the corresponding parity penalty is high. For example, according to (1), a 16-bit data word requires six parity bits for SEC-DED. The parity overhead is over 27%, which is unacceptable for commercial RAMs. For such case, a better way is to apply the EDAC process to the long length memory data words instead of the 16-bit I/O data words. An EDAC design proposed in [14] provides a (72, 64) code codec for the memory chip whose memory data word width is 64-bit and the I/O data word width is 16-bit. In this case, a memory data word is composed of four I/O data words, and only part of the memory data word is accessed for each time. The EDAC process is applied for the long memory data word and only eight parity bits are required for each 64-bit data word. The area overhead is therefore reduced to 11%, which seems more acceptable. Based on the EDAC architecture in [14], we develop an EDAC circuit which can further serve for the (104,96), (72,64), and (40,32) codes. It provides different codes for data words in length of 96 bits, 64 bits, and 32 bits, such that we can help memories’ reliability by adapting the length of encoded data words.

We propose an EDAC scheme as shown in Fig. 2, where the memory arrays have long data word width, and the data word width of chip I/O is only 16-bit. To explain the scheme precisely, we assume there is an internal memory controller dealing with the data flow between the memory arrays and the I/O buffer. There are three operation modes with different code rates. Code rate is defined as the ratio of the data word length to overall codeword length, i.e., CR = k/n. In the three modes, the EDAC circuit applies the (104,96) code, the (72,64) code, and the (40,32) code respectively. It protects data words in different length against one bit error by 8-bit parities, and both the data words and parities are stored in the memory arrays.

According to the specified operation mode, the internal memory controller fetches a data word in certain length (among 96 bits, 64 bits, and 32 bits) with its 8-bit parity. For a write operation, certain 16 bits of the fetched data word are updated into the 16-bit new word input from the I/O buffer, and the EDAC circuit encodes a parity for the new data word. Then the internal memory controller passes the new data word and parity to the memory arrays. For the read operation, the EDAC circuit checks the correctness of the fetched data word, and certain 16 bits of it would be output to the I/O buffer by the internal memory controller.

For normal cases we specify the mode of the (104,96) code, in which the EDAC process is applied for data words in length of 96-bit. When memories are of low yield, we specify the mode of the (40,32) code instead, such that each eight parity bits are responsible for fewer data bits. The memory system is consequently tolerant of more errors and more reliable.

The provided code rates are determined based on the parity length. For simplifying the memory implementation, we fix the parity length to eight bits, which is of the power of two and is an acceptable length for redundancy. According to (1), for the parity length r=8, the data length k is restricted to 120. Then we assign k to be 32, 64, and 96 for the three operation modes in the proposed scheme, which can be easily aligned with the I/O data word width, 16. The codewords for each mode are as Fig. 3 shows. For three operation modes, the internal memory controller fetches data words of 96 bits, 64 bits, and 32 bits respectively, as well as their 8-bit parity. The codewords are therefore in lengths of 40 bits, 72 bits, and 104 bits respectively. Accordingly, the data percentage of the memory arrays will be different. Take Mode-32 as an example, every 32 bits require eight parity bits for better correction...
capability, therefore, we use more user storage for parity. As a result, the data percentage of the memory arrays is only 80%. Even though the EDAC circuit deals with codewords in different length, its input wires are fixed to 96-bit width. Hence for Mode-32 and Mode-64, the memory controller fills the unused data wires by 0’s to avoid signal fluctuation.

IV. EDAC CIRCUIT AND PARITY GENERATION MATRIX DESIGN

The EDAC circuit is based on the architecture introduced in [14]. To serve different data word lengths for different modes, we designed a specific parity generation matrix and made modifications to the circuit.

A. EDAC Architecture

In the EDAC scheme, a parity is generated from a 96-bit, 64-bit, or 32-bit data word, which is composed by several 16-bit I/O words, and only 16 bits of the data word are written for each time. For possible error in the non-updated data bits, we have to check the fetched data before encoding the new parity, or the parity might be generated according to the wrong data and written, then the error will be masked. However, running the encoding process after a decoding process makes a significant timing penalty for the memory cycle. Therefore we adopt a parallel encoding and decoding architecture.

The data access flow is shown in Fig 4, here we use the Mode-96 case for example. While writing a 16-bit new word in to a memory data word of 96-bit, both the encoding and the decoding phases are applied. The EDAC scheme composes a 96-bit new data word by updating certain 16 bits of the data word, and then generates a new parity by the encoder. Simultaneously, the original data word is checked by the decoder. If the syndrome shows there is an error bit in the non-update 80 bits, the new parity will be modified then. Finally, the 96-bit new data word with a correct 8-bit new parity are stored into memory. Reading a 16-bit word from memory is relatively simple, for which only the decoding phase is applied. A 96-bit data word containing the specified 16-bit word is fetched and then checked. If there is an error bit in the fetched data word, the EDAC circuit will correct it.

Corresponding EDAC architecture is shown in Fig 5. There are two main paths in the architecture. The right one is for parity check, while the left one is for new data composition and new parity generation. In the right path, the parity generator \( P_{Generator} \) 1 and a syndrome generator form a decoder. The parity generator is an XOR tree developed according to the G-matrix of (104,96) code. A \( k \)-bit data word and its 8-bit parity are fetched by the internal controller, where \( k \) is different in each operation mode. During the read phase, the decoder checks the data and parity. The results will be recorded as an 8-bit syndrome. \( Error_{Locator} \) locates the error bit according
to the syndrome, and $D_{Corrector}$ outputs a corrected data word. If the EDAC scheme detects two error bits occur in the $k$-bit data word, $DE_{Indie}$ signals this situation. As to the left path, the $D_{Selector}$ module is for composition of a new data word. During the write phase it updates certain 16 bits in the $k$-bit prefetched data word according to the specified address. Another parity generator, $P_{Generator2}$, on the left path is an encoder, which is an XOR tree the same with $P_{Generator1}$. As $P_{Generator2}$ generates a new parity for the new data word, the prefetched data and parity are checked in parallel by the decoder in the right path at the same time. If there is an error bit within the non-updated $(k - 16)$-bit data, $P_{Corrector}$ will modify the new parity. Then the new parity and the $k$-bit data will be written into memory. To keep the access transparent to users, performance of the EDAC circuit is important. Adopting Hsiao code for the parity generators leads to balanced XOR tree. As a result, the latency due to the EDAC circuit can be acceptable.

B. Hardware Sharing of Encoders

For the three operation modes of the (104,96) code, the (72,64) code, and the (40,32) code, the codec requires the parity generators implemented according to an 8-by-96, an 8-by-64 and an 8-by-32 G-matrix respectively. Constructing these G-matrices according to the Hsiao code rules in Section II, we can derive each G-matrix with least 1’s. The corresponding parity generators are XOR trees with eight outputs and respectively 96 inputs, 64 inputs and 32 inputs. However, in practice we do not implement three pairs of different parity generators for encoding and decoding to serve each mode, which cost a great area overhead.

In our proposed design, three modes share the same hardware of parity generators, that is, only a pair of XOR trees are implemented in our design for encoding and decoding. First we construct an 8-by-96 G-matrix for the (104,96) code according to the Hsiao code rules. As to the (72,64) code and the (40,32) code, we directly use part of the 8-by-96 G-matrix for the 8-by-64 G-matrix and the 8-by-32 G-matrix, as shown in Fig 6. Here we defined the 8-by-96 G-matrix into three sections for explanation. The first section of the inputs $m1$ to $m32$ is excerpted for the 8-by-32 G-matrix of the (40,32) code, and the sections of inputs $m1$ to $m64$ are excerpted for the 8-by-64 G-matrix of the (72,64) code. Though the G-matrices for the (72,64) code and the (40,32) code are not of the least 1’s as the conventional Hsiao code rules suggest, it does not matter since their physical hardwares share the existing XOR tree of the (104,96) code instead of making extra overhead.

Fig 7 shows the hardware sharing of each mode. The 1’s on each row of the G-matrix are the fan-ins to the XOR gates which result in one parity bit, and there are eight paths in the whole XOR tree for generation of eight parity bits. Here we only shows one path for explanation. The XOR tree is divided into three blocks according to the three sections of the G-matrix. For Mode-96, the whole tree is used for parity generation. As to Mode-64 and Mode-32, data bits come only from partial inputs, and only a partial tree is running for parity generation. We keep the inputs of unused blocks to ground, thus those blocks do not influence the resulted parity and consume no dynamic power. For further power reduction, we can also introduce a MUX to the tail of the XOR tree, and switch the result among the root nodes of each block, as shown in the figure. Then the unused part can be switched off by designing the three blocks into different power domains. As a result, the unused blocks consume no power during Mode-64 and Mode-32.

Now the issue of importance is that the path delays of the three blocks can be unbalanced. If we directly use part of the 8-by-96 G-matrix for the 8-by-64 G-matrix and the 8-by-32 G-matrix, there can be a situation that most 1’s in one row
are concentrated at certain section in the matrix. An example can be found in the first row of the G-Matrix in Fig 6, where most of 1’s are concentrated at the first section. Consequently, the corresponding block of XOR tree experiences a longer depth, and the critical delay for the whole tree is increased. Row weight for a G-matrix of the (104,96) Hsiao code is 46, and the average row weight for each section should be 16. Hence, to avoid the timing skew problem, we have to make the first two blocks in depth of 4 levels and the last block in depth of 5 levels. Accordingly, after constructing the 8-by-96 G-matrix for the (104,96) code by the Hsiao code rules, we rearrange the columns for balanced row weights of each section. If certain row weight of the first or the second section is above the average, some columns with extra 1’s should be arranged to the last section, which is allowed longer depth to the corresponding block of the XOR tree. As a result, the path delays of the three parts of the XOR tree can be balanced.

C. Fast Decision for PC Enabler

In the EDAC architecture as shown in Fig 5, there is a P_Corrector module used to modify the new generated parity when the decoder detects an error bit in the non-updated data bits. As the left path in the architecture is generating the new parity, the right path is checking the fetched data simultaneously. After the error bit is located, the PC_Enabler module will decide whether to enable the P_Corrector according to the location of the error bit. The procedure is necessary to avoid error masking in the non-updated data bits. However, even the parity generating process and the data checking process are parallel, correcting new parity after locating the error bit makes the parity generating path excessively long. It can harm the performance of memory access.

Accordingly, we further design the G-matrix to advance the decision of PC_Enabler module. The 8-by-96 G-matrix is divided into six subsections, as shown in the Fig. 8. Each 16 columns form one subsection for 16 bits of input data m. We specify certain patterns of the first four entries of each column for each subsection. For example, the first four entries of the columns in the first subsection have to be either 0110 or 1001. In addition, consider the Hsiao code construction rules, every column should contain an odd number of 1’s. Thus the other four entries of each column are restricted. The number of possible patterns for each subsection is listed in the figure.

<table>
<thead>
<tr>
<th>P1</th>
<th>m1-m16</th>
<th>m17-m32</th>
<th>m33-m48</th>
<th>m49-m64</th>
<th>m65-m80</th>
<th>m81-m96</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>000...111</td>
<td>000...111</td>
<td>00...1...11</td>
<td>000...111</td>
<td>000...111</td>
<td>000...111</td>
</tr>
<tr>
<td>P2</td>
<td>111...000</td>
<td>011...000</td>
<td>011...001</td>
<td>000...111</td>
<td>000...110</td>
<td>000...110</td>
</tr>
<tr>
<td>P3</td>
<td>111...000</td>
<td>000...111</td>
<td>011...001</td>
<td>111...000</td>
<td>110...000</td>
<td>000...111</td>
</tr>
<tr>
<td>P4</td>
<td>000...111</td>
<td>000...111</td>
<td>000...111</td>
<td>000...110</td>
<td>000...110</td>
<td>000...110</td>
</tr>
</tbody>
</table>

Fig. 8. An 8-by-96 G-matrix design for fast decision on parity modification.

Finally we select 16 patterns from those candidates for each subsection, by considering the balance among weights of all rows as introduced in II. Then we rearrange the sequence of the six subsections for further balance among the row weights of three sections as we discuss in IV-B. The final G-matrix is shown in Fig. 9. For the purpose of fast decision by the PC_Enabler module, the matrix is not of the least number of 1’s. It makes a little area overhead for about eight extra XOR gates, however, we can therefore advance the procedure of correcting new parity and consequently shorten the parity generating path.

V. EXPERIMENTAL RESULTS

A. Reliability Evaluation

To evaluate the reliability gain from the proposed adaptive code rate EDAC scheme, in this section we provide the reliability evaluation of memories with the proposed EDAC design. The reliability of a memory with a codec can be expressed as

$$R(N_E) = \begin{cases} 1, & N_E = 0; \\ \prod_{n=1}^{N_E} \left(1 - \frac{n-1}{N_{CW}}\right), & N_E \geq 1. \end{cases}$$

where $R(N_E)$ is the reliability of a memory chip with $N_E$ error bits, and $N_{CW}$ is the number of codewords in this memory. Fig 10(a) shows the reliability plot of a 256Mb memory chip for an example, which results from simulation according to the above expression. The dot-dashed line shows the reliability of a memory without a codec, whereas the curve A, B, and C show the reliabilities of the memories with the (104,96) code, the (72,64) code, and the (40,32) code codec respectively. The proposed adaptive rate EDAC design can provide each capability of the three codes, while a conventional EDAC design provides only one of them. Fig 10(b) provides a room-in plot. For the case that the restriction on reliability of chips is 0.96, if the memory chips with less than 460 error bits, we adopt EDAC scheme in Mode-96. As to the situation with more than 460 error bits, we can specify Mode-64 or even Mode-32, such that each-8 bits are responsible for only 64 bits or 32 bits. The correction capability of the EDAC circuit is stronger, and the memory consequently tolerates 281 extra error bits.

Compare to conventional EDAC design for RAMs, the proposed adaptive code rate EDAC design costs the same hardware with a conventional design of the (104,96) code, however, it has the flexibility in reliability among curves A, B, C in the Fig.10, while the conventional design provides the reliability limited to curve A.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Encoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>9030um²</td>
<td>5703um²</td>
</tr>
<tr>
<td>1330</td>
<td>840</td>
</tr>
<tr>
<td>1.86ns</td>
<td>1.89ns</td>
</tr>
</tbody>
</table>

B. Synthesis Result

We have developed an EDAC circuit which can serve for each of the (104,96), (72,64), and (40,32) codes, based on the architecture shown in Fig 5. It provides adaptive codes for memory data words in different length among 96 bits, 64 bits, and 32 bits, while the chip I/O data word width is only 16-bit.
The design is synthesized by TSMC 0.13 μm technology cell library, and the synthesis result is shown Table I. The design costs 1.89 ns latency for each memory access cycle, which is acceptable for DRAM and some emerging memories such as the Magnetic RAM (MRAM) and the phase-change memories (PCM). The area of the design is 14732 μm², which is very small compared to the memory arrays.

VI. CONCLUSIONS

In this paper, we proposed an adaptive code rate EDAC scheme for RAMs. We developed the codec which provides the (104,96), (72,64), and (40,32) SEC-DED Hsiao codes for memories with long data word width while an I/O word is restricted to 16 bits. When the yield of a memory chip is lower than expected, changing the code rate of the EDAC scheme can help its reliability. For the case that the restriction on the reliability of a 256Mb chip is 0.96, the proposed scheme allows 281 more error bits than a conventional codec of the (104,96) code, while both codecs cost the same hardware overhead. In the case of TSMC 0.13 μm process, the area of the EDAC circuit is 14732 μm² while it costs 1.89 ns latency.

REFERENCES