Area and power efficient ECC for multiple adjacent bit errors in SRAMs.

Kumar Rahul
XILINX
Hyderabad, India
kumarr@xilinx.com

Santosh Yachareni
XILINX
Hyderabad, India
santoshy@xilinx.com

Abstract—As submicron technology scales, SRAM bitcell density increases on the chip. This results in an increase of soft errors due to radiation induced multiple-bits upsets (MBUs). SRAM uses SEC-DED [Single Error Correct – Double Error Detect] code along with word interleaving or column muxing to mitigate these soft errors. The probability of MBUs in the SRAM bitcells in 16nm/7nm technology has increased considerably. Even using word interleaving with 4:1 column mux is not sufficient to mitigate these soft errors. There are powerful ECC codes which can correct these soft errors but it comes at the cost of overhead in the parity bits which eventually translates into the SRAM size increase. So here we are proposing an area efficient ECC(72,64) and ECC(39,32) code with no extra parity bit cost which can detect and correct adjacent 2-bit error and detect adjacent 3-bit error. We are also proposing an area efficient ECC(73,64) and ECC(40,32) code with one extra parity which can detect and correct both adjacent 2-bit and 3-bit errors.

Keywords - static random access memory [SRAM], Error-Correcting Code (ECC), Single Event Upset (SEU), multiple-bits upsets (MBUs), Multiple Unidirectional SEU (MSEU), SEC-DED [Single Error Correct, Double Error Detect], SEC-DED-ADEC-ATED (Single Error Correct, Double Error Detect, Adjacent Double Error Correct, Adjacent Triple Error Detect), SEC-DED-ADEC-ATEC (Single Error Correct, Double Error Detect, Adjacent Double Error Correct, Adjacent Triple Error Correct), FINFET(Fin-Shaped Field Effect Transistors)

I. INTRODUCTION
As memory bit cells of an IC get smaller and/or denser, the likelihood of an SEU impacting more than one of such memory bit cells simultaneously increases. However, increasing too is a demand for memory bandwidth, and thus the addition of more parity bits to resolve data corruption issues through use of an ECC would hamper efforts to satisfy such demand for memory bandwidth. Accordingly, it would be desirable and useful to provide an ECC that addresses both of these conflicting issues. With the scaling of submicron technology, the size of the bit cell is continuously reducing and so is the bit cell internal nodes capacitance. So the probability of the bit cell storage node getting corrupted by the SEU event increases. Study has shown that 1-5% of the SEU can cause MBUs [1]. SEU can cause several types of MBUs [2], [3], [4]. Analysis of the SRAM bit cell in 7nm FINFET further suggests that the probability of adjacent 3-bit in error has increased with respect to the planar 28nm SRAM bit cell and the 16nm FINFET SRAM bit cell.

All high-speed SRAM use SEC-DED Hamming Code [5] to encode the input data into a code word using the H-Matrix [5] and write into the memory. If any SEU upsets 1 bit of the data stored in the memory, then the ECC decode using the same H-matrix can correct the 1 bit of corrupted data. The syndromes generated using the H-matrix is used to detect which bit location is in error and correct it [6].

This paper is organized as follows. Section II explains adjacent bit errors in SRAM with different column mux scheme. Sections III & IV discuss the ECC Code Algorithm and the ECC Code Matrix and compare the Area, Power and Delay impact of different ECC techniques. Section V shows the verification methodology of these codes and section VI concludes the paper.

II. ADJACENT BIT ERROR AND COLUMN MUX IN SRAM
In SRAM, the SEU event corrupting the adjacent bit has high probability [7]. We propose a new ECC(72,64), SEC-DED-ADEC-ATED code with the optimized syndrome calculation for ECC encode and decode. ECC(72,64) uses the same number of parity bits as the SEC-DED Hamming code H(72,64). ECC(72,64) can correct adjacent 2-bit in error and detect adjacent 3-bit in error while reducing the number of 2 input XOR gates by 40% as compared to SEC-DED code. A new ECC(73,64), SEC-DED-ADEC-ATEC code is proposed to do adjacent 3-bit error correct with the optimized syndrome calculation and reducing the number of 2 input XOR gates by 30% as compared to SEC-DED code. This ECC(73,64) code uses only one extra parity bit compared to the code proposed in [8].

One way to mitigate the SEU is to use word interleaving [9], or column mux in the SRAM array design. Column mux of 2:1, 4:1 is used extensively. SRAM used in processors with no column mux could have multiple [10] adjacent SRAM bits affected by the SEU. The probability of 5 physical adjacent bits corrupted by SEU [11] is less.

Figure 1a. SEU corrupting 5 adjacent bits. No Column mux (a0a1…a7 is same word0)
As shown in Fig. 1a, SEU event corrupts 5 adjacent bits with no column mux. Fig. 1b with column mux of 2:1, 5 physical adjacent bits corrupted is mapped to 3 adjacent bits in error in word0. ECC(73,64) can be used to mitigate all the possible 3 adjacent bits in error. For the word size of 73, there are 71 such possible 3 adjacent bits in error. As shown in Fig. 1c, SEU event corrupting 5 adjacent bits, with column mux of 4:1, 5 physical adjacent bits corrupted is mapped to 2 adjacent bits in error in word0 and single bit error in word1, word2, word3. ECC(72,64) can be used to mitigate all the 2 adjacent bits in error. There is also a very low probability of MSEU. MSEU could lead to scenario where two separate 5 physical adjacent bits get corrupted. This could lead to a case where these two MBUs are aligned and one starts right after the other physically. This is a scenario of 10 physical adjacent bits corrupted. As shown in Fig. 1d, word0 and word1 has 3 adjacent bits in error and word2, word3 has 2 adjacent bits in error.

III. ECC CODE IMPLEMENTATION

The generation of the ECC code matrix also depends on how the data and the parity bits form a 72/73 bit word. Depending on how the parity is placed w.r.t to the data the ECC(73,64) or ECC(72,64) Code Matrix changes. Some of the examples of data and parity arrangement are shown in Fig. 2.

<table>
<thead>
<tr>
<th>Case1</th>
<th>p0</th>
<th>p1</th>
<th>...</th>
<th>p7</th>
<th>d0</th>
<th>d1</th>
<th>...</th>
<th>d62</th>
<th>d63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case2</td>
<td>d0</td>
<td>d1</td>
<td>...</td>
<td>d62</td>
<td>d63</td>
<td>d0</td>
<td>d1</td>
<td>...</td>
<td>d62</td>
</tr>
<tr>
<td>Case3</td>
<td>d0</td>
<td>...</td>
<td>d7</td>
<td>p0</td>
<td>p1</td>
<td>...</td>
<td>d8</td>
<td>...</td>
<td>d15</td>
</tr>
<tr>
<td>Case4</td>
<td>d0</td>
<td>...</td>
<td>d15</td>
<td>p0</td>
<td>...</td>
<td>p3</td>
<td>d16</td>
<td>...</td>
<td>d31</td>
</tr>
<tr>
<td>Case5</td>
<td>d0</td>
<td>...</td>
<td>d31</td>
<td>p0</td>
<td>...</td>
<td>p7</td>
<td>d32</td>
<td>...</td>
<td>d63</td>
</tr>
</tbody>
</table>

Figure 2. Data and Parity arrangement

Parity bits first followed by the data bits or vice versa. 8 bits of data followed by 2 bits parity, 8 bits of data and so on. 16 bits of data followed by 4 bits of parity, 16 bits of data and so on. ECC(72,64) or ECC(73,64) code Matrix will be separate for each of the case. Even with the code matrix change the algorithm to optimize the syndrome calculation holds true.

Code for generating the ECC code Matrix

Chose any number a and b
Initialize the array synd1[], synd2[] and synd3[] to 0. While ( N< Wordsize) {
    synd1_flag = true, synd2_flag = true, synd3_flag = true
    Choose any number i.( between 1 to 2^8 -1)
    synd1_flag = synd1_flag (and) synd1[i-1]
    if (synd1_flag=true )
    {  Match not found  }
    else
    {
        synd2bit(b, i)
        synd3bit(a, b, i)
        synd2_flag = synd2_flag (and) synd2[synd2bit-1] ;
        synd3_flag = synd3_flag (and) synd3[synd3bit-1] ;
        if (synd2_flag=true)
        {  Match not found  }
        else
        {
            if (synd3_flag=true )
            {  Match not found  }
            else
            {
                synd1[i-1], synd2[synd2bit-1], synd3[synd3bit-1] =true;

                Output the value i, synd2bit, synd3bit
                }
        }
    }
    if ( N=73)
    {ECC code Matrix generated}
    else { chose another a and b} and repeat the While loop
}

Figure 3. ECC(73,64) Code Matrix Algorithm

The proposed ECC(72,64) or ECC(73,64) code Matrix algorithm has to satisfy certain conditions. The events 1 bit in error, 2 adjacent bits in error, 3 adjacent bits in error are mutually exclusive. This algorithm is true for all possible combinations of events which are mutually exclusive to each other. ECC(73,64) code matrix algorithm shown in Fig. 3. should satisfy the following conditions

1) The syndrome of each 1 bit error should be unique.
2) The syndrome of each 2 adjacent bit error should be unique.
3) The syndrome of each 3 adjacent bit error should be unique.
4) The 1 bit error syndrome can share the common space with the 2 adjacent bit error syndrome, p0 distinguishes between the even or odd number of bits in error.
5) The 3 adjacent bit error syndrome can share the common space with the 2 adjacent bit error syndrome, p0 distinguishes between the even or odd number of bits in error.
6) The 1 bit in error syndrome, 3 adjacent bit error syndrome should not share the common space.
IV. ECC Code Matrix and Comparison

Figure 4. Proposed ECC(72,64) SEC-DED-ADEC-ATED Code Matrix

Figure 5. Proposed ECC(73,64) SEC-DED-ADEC-ADEC Code Matrix

Figure 6. Proposed ECC(39,32) SEC-DED-ADEC-ATED

Figure 7. Proposed ECC(40,32) SEC-DED-ADEC-ADEC

Fig. 4 shows the proposed ECC(72,64) Matrix for SEC-DED- ADEC-ATED code. Fig. 5 shows the proposed SEC-DED-ADEC-ADEC ECC(73,64) Code Matrix. Fig. 6 shows the proposed ECC(39,32) Matrix for SEC-DED-ADEC-ATED code. Fig. 7 shows the proposed SEC-DED-ADEC-ADEC ECC(40,32) Code Matrix.

Figure 8. Parity computation for ECC(72,64)
The number of 2 input XOR gates for each syndrome calculation depend on the numbers of 1 in each row shown in the ECC matrix in Fig. 4 and Fig. 5. Optimizing the parity equation into separate minterms is shown in Fig. 8. The proposed algorithm uses the approach where there is overlap of minterms between syndromes. Term t1 is a part of computed syndrome S1. t1 is unique to S1. Term t2 is a part of computed syndrome S2 and so on. t1t2 means its common between S1 and S2. t1t2t3 means these terms are common to S1, S2, S3. Once all these t1, t2, t3, t4, t5, t6, t7 is broken in minterms, then S1 is computed by doing the XOR of all the terms which has t1 in common. All the common minterms are computed separately, and then used in the parity generation equation. It helps in reducing the number of 2 input XOR gates required in the proposed code as well as in the Hamming code. ECC(72,64) encode/decode compared to the Hamming Code H(72,64) and other codes is shown in Table 1. As the wordsize increases, the method shown in Fig. 8 helps in considerable reduction of 2 input XOR gates and hence the area and static power of ECC.

Table 1. Comparison of proposed SEC-DED-ADEC-ATED codes with the other codes.

<table>
<thead>
<tr>
<th>(n,k)</th>
<th>Codes</th>
<th>2-input XOR Gates</th>
<th>Max logic depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>(39,32)</td>
<td>SEC-DED (IBM 8130) [40,32]</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Hsiao Code</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>SEC-DED-ADEC in [Abramson]</td>
<td>132</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Proposed SEC-DED-ADEC-ATED</td>
<td>82</td>
<td>5</td>
</tr>
<tr>
<td>(72,64)</td>
<td>SEC-DED (IBM 3081)</td>
<td>256</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Hsiao Code</td>
<td>208</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>SEC-DED-ADEC in [Abramson]</td>
<td>296</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Proposed SEC-DED-ADEC-ATED</td>
<td>154</td>
<td>7</td>
</tr>
</tbody>
</table>

There is considerable reduction in the total number of internal nets as well, thereby reducing the total routing and the internal capacitance and the overall delay. Extra logic required to correct 2 adjacent bit in error and detect 3 adjacent bit in error results in increase in the total area of SRAM by 0.15%. ECC(73,64) encode/decode compared to the H(72,64) has 1 extra parity bit to be generated. Extra logic is required to correct 2 adjacent bit in error and correct 3 adjacent bit in error. This increases the total area of SRAM by 0.25%.

V. VERIFICATION

For ECC(73,64) code, 64 bit of data is input to the encoder. Output of the encoder is 9 bits of parity.

Table 2. Error flags

<table>
<thead>
<tr>
<th>Error</th>
<th>sbit</th>
<th>dbit</th>
<th>tbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>No error</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 bit error</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Output of the ECC decode is 9-bit syndrome. If all the 9 bit syndromes are zero, then there is no error. If syndrome S0 is 1, then it’s a single bit error. If the syndrome S0 is 0 and if any of the other syndrome bits are 1 then it’s a double bit error. Table 2 shows the status of the flag for all the possible cases of bits in error.

VI. CONCLUSIONS

The ECC code proposed in this paper can correct multiple adjacent bit in error with minimal area and performance impact on the SRAM. The proposed code is easy to implement and verify. These codes are useful especially for submicron 16nm/7nm FINFET design where the probability of adjacent MBU due to SEU is high. These codes can be used with the word interleaving to mitigate the adjacent MBU. The need of these specially designed codes will increase with the scaling submicron technology. The proposed scalable ECC can be designed for SRAM with any kind of data bit arrangement in the word.

References