Cost Models for Large File Memory DRAMs with ECC and Bad Block Marking

Curtis Wickman, Duncan G. Elliott, Bruce F. Cockburn
Department of Electrical and Computer Engineering, University of Alberta
Edmonton, AB T6G 2G7, Canada
{wickman | elliott | cockburn}@ee.ualberta.ca
http://www.ee.ualberta.ca/~elliott/memory

Abstract

We present cost models appropriate for large file memory DRAMs that exploit error-correcting codes, redundant elements and bad block marking in order to reduce the average cost per working bit. Many different fault-tolerance methods have been considered previously for DRAM but, because of the constraints of conventional commodity memory, only a few methods, such as redundant rows and columns, have entered wide-spread use. Our research on file memory breaks from past work by relaxing the requirements that random-access be fast and that shipped devices contain 100% of the nominal working bit capacity. We show that, under the relaxed requirements of file memory, the greater potential efficiencies of large ECC codewords and bad block marking may become cost-effective. These file memory techniques may thus be a way of accelerating the economic production of 256 Mbit and 1 Gbit DRAMs.

1. Introduction

Cost models help clarify economic trade-offs present in alternative design choices. In DRAM design, cost models have been used to guide the choice of page sizes and the selection of redundancy configurations that permit permanent on-chip repair. Previous cost models for DRAM have operated under the requirements that (1) random-access be fast, and (2) that 100% of the nominal capacity of working bits be present.

We are currently revisiting an old idea with new objectives to create what we call file memory, in which we relax the two preceding requirements in order to reduce the average cost per working bit. Specifically, file memory is designed to provide competitive page mode access performance, as opposed to fast random-access. Conventional bit-level or word-level random access is not essential for memory devices that lie lower down in the memory hierarchy, devices which typically operate in block-oriented sequential access mode. Also, many data-intensive applications access data in larger-sized blocks for which the overhead of the initial access can be amortized over the large number of data bits in the block, provided that the data bandwidth is sufficiently high. The second assumption, that 100% of the nominal memory capacity be available in all parts, is already not required of magnetic mass storage devices. System software or device firmware copes with bad sectors or blocks by keeping track of them and avoiding them by means of a software mapping from nominal memory addresses to working physical locations on the imperfect storage medium. Likewise, in file memory we consider the possibility that we can tolerate bad blocks of cells by recording the location of all bad blocks and then, possibly in cooperation with system software, redirecting memory accesses to avoid bad blocks.

In this paper we present cost models appropriate for large file memory DRAMs that exploit various combinations of error-correcting codes, redundant elements, and bad block marking to reduce the effective cost per functional bit. The relaxed requirements of file memory make it possible to reconsider known fault tolerance methods and to apply them in combination in new ways. Our proposed cost models will guide the selection of the most economical file memory architectures.

The paper is organized as follows. The next section reviews the necessary technical background. The third section presents our framework for modeling the cost per working bit. The fourth section examines a cost model that assumes the use of modified Hamming codes. The fifth section extends the analysis to consider the effects of bad block marking. The last section summarizes our results and makes some concluding remarks.

2. Background

2.1. Modified Hamming codes

Modified Hamming codes are an efficient class of error-correcting code (ECC) that has been implemented both in memory modules as well as in ECC DRAMs [8][6]. Check bits are created from the data bits and are stored with the data bits in memory. When the data is read back from memory, the check bits are recalculated and then compared to the bits that were written originally. This comparison creates syndrome bits which indicate if an error has occurred. For some codes it is also possible to determine if two errors have occurred. This type of error correction code, which can correct one error and detect two errors, is called a single-error correcting, double error detecting (SEC-DED) ECC. XOR gates are used to create the check bits and are also used to generate the syndrome bits.

Detailed descriptions of the modified Hamming code are given elsewhere [8][5]. Briefly, in a Hamming code, each codeword of size n consists of k data bits augmented with r check bits. The number n of odd-weight codewords is an even power of 2, where $n = 2^{r-1}$. The relationship between parameters k and r is given by $k \leq 2^{r-1} - r$.

2.2. Synergistic fault tolerance between ECC and redundancy

The IBM 16-Mb ECC DRAM [9] used a Hamming code to increase the reliability against soft errors and to correct hard errors. In addition to providing on-chip SEC-DED coding and decoding circuitry (based on an ECC similar to a modified Hamming code), the IBM design kept with conventional practice and also provided both redundant rows and columns. A key observation was that using redundancy together with error correcting circuitry (ECC) produced a far greater fault tolerance that just a sum of the fault tolerance provided by either redundancy or ECC alone. This is illustrated below in Figure 1. Redundancy alone allowed a 50% expected yield to be achieved in the presence of an average of 28 failing cells. Using ECC alone, 50% yield could be achieved in the presence of 428 failing cells. However, with both redundancy and ECC, an average of 2725 failing cells could be tolerated to achieve a 50% yield. The synergism between redundancy and ECC is due to the complementary abilities of the two fault tolerance methods: redundancy can be used to “break up” clusters of faulty bits so that it is rare that an ECC codeword has more than 1 faulty bit. The ECC can then readily detect and correct the resulting isolated faulty bits. The synergistic effect was so

strong that, in the opinion of the IBM authors, the fault-tolerance of their DRAM greatly exceeded the foreseeable fault tolerance requirements of any fabrication process. We, however, are trying to define methods that can be used in order to bring forward the economic cross-over point between current mass manufactured 64 Mbit DRAMs and larger next generation (256 Mbit and 1Gbit) DRAMs.
The IBM chip used ECC codewords containing 137 bits. Each codeword contained 128 data bits and 9 check bits. The codeword length appears to have been selected mainly to produce acceptably fast random-access times. The extra area required to store the check bits and the extra area occupied by the ECC circuitry increased the size of the ECC DRAM by about 8% over the size of a conventional DRAM with only row and column redundancy.

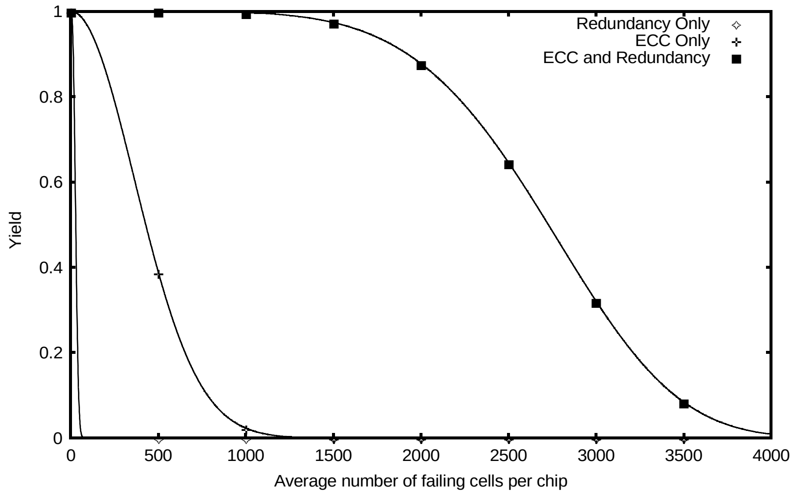

FIGURE 1. IBM redundancy, ECC, and redundancy/ECC plots [9]

The majority of our work presented here (with the exception of Section 4.2) was done using the relatively well documented IBM ECC DRAM as a reference point.

3. Modeling the cost per working bit

In the past work described above in Section 2.2, yield was used to compare redundancy methods. However using yield to measure the effectiveness of a redundancy method misses the key economic goal of reducing the average cost per working bit. To take area into account, we will be using the ratio of area to yield to compare redundancy techniques. The area is nearly proportional to the total number of bits on a die and can thus be replaced with the total bits per die. Dividing the ratio by the number of working data bits on the die we can obtain the cost per working bit (cpwb) as given below in Equation 1.

$$cpwb = \frac{TotalBitsPerDie}{Yield \times WorkingBitsPerDie} \qquad \text{(EQ 1)}$$

The yield was calculated using a Poisson model as described in [9]. This model was used to calculate the yields for this work and was also used to create Figure 1. We are considering single cell failures that are uncorrelated.

4. Modified Hamming codes in DRAM

There are several aspects of using ECC in DRAM that we wanted to examine that were not considered in previous work. The first aspect is the effect that the ECC word size has on the cost per working bit. The second is an analysis of the amount of area needed to do the detection and correction.

4.1. Effect of word size

If a modified Hamming code is used in a DRAM, then the first question is how long should the ECC codeword be? A 137-bit ECC codeword (128 data bits) was used in the IBM ECC DRAM. In [1] it was stated that “comparisons of 72-, 137-, and 266-bit ECC systems show that maximum density is at the 137-bit word length and the change in speed from 72-bit system to a 137-bit system is a fraction of a nanosecond”. It was not clearly explained what was meant by maximum density. It appears likely that they were referring to the maximum density of the Hamming code ECC circuitry.
We modeled the yield of a chip using the same equations from [9]. In [9] a codeword length of 137 bits was used. To revisit this choice we varied the size of the codeword and then plotted the resulting cost in order to see what effect changing the codeword length would have on the cost per working bit. The results are shown below in Figure 2.

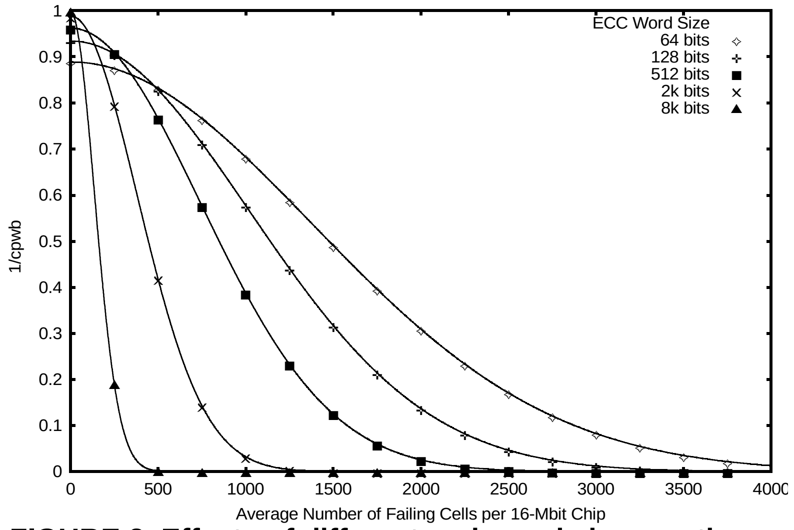

FIGURE 2. Effects of different codeword sizes on the cpwb

Figure 2 shows that as the ECC word size increases, the cost per working bit decreases for low numbers of defects. This is because as the word size is increased, the number of bits needed to hold the ECC bits decreases overall and hence gives us a higher yield as well as a greater ratio of data bits to total bits. Although increasing the word size decreases the cost per working bit when the average number of faults is low, it also means that the dies cannot tolerate more than a very small number of defects. Overall, shorter codewords produce a more graceful degradation in yield as the expected number of failing cells increases. For any average number of faults there is always an optimal size for the ECC word. There are two practical limitations to how large the ECC word can be made, however. The first is that all of the codeword bits need to be read in order for an error to be detected and corrected. This implies that the upper limit on the size of the ECC word may be dictated by the size of a DRAM page. The second limitation is that as the ECC word becomes larger the area of the circuitry to detect and correct the error increases.

4.2. Area of the Hamming code circuitry

Our analysis has not yet taken into account the amount of area needed to do detection and correction. Since processes and feature sizes are always changing, the area needed by the circuitry can be expresed as the number of 2-input, 1-output, 4-transistor gates. Since ECC circuitry typically contains XOR gates, one could argue that a 2-input, 1-output XOR gate cannot be built with 4 transistors. Fast 4-input XOR gates can be built in 16 transistors using a DCVS XOR gate as described in [3]. This comes close to the assumed 12 transistors and is probably close enough for a first-order approximation. Since there are many different ways in which the hardware could be implemented, we chose a straightforward hardware architecture from [8], as shown below in Figure 3 and Figure 4.

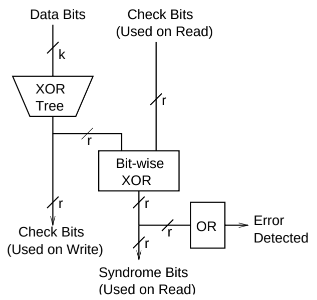
Data Bits
Check Bits (Used on Read)
k
XOR Tree
r
Bit-wise XOR
OR
Error Detected
Check Bits (Used on Write)
Syndrome Bits (Used on Read)

FIGURE 3. Hamming code error detection hardware [8]

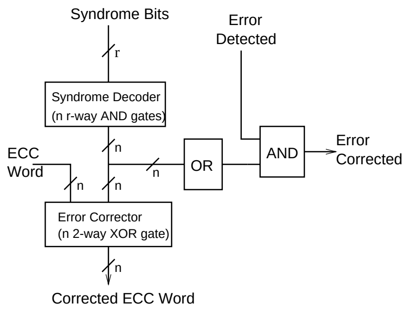

FIGURE 4. Error correction hardware [8]

Figure 5 shows how the normalized area of the assumed modified Hamming code circuitry grows with codeword length. The number of check bits was varied from 8 to 10 bits and we can see, in all cases, that the area grows roughly linearly with the number of data bits in the codeword. We can also see that the area needed for detection and correction is significantly larger than the area required for detection only. This is an important factor to note because, if we are not as concerned with performance, then we can consider off-loading error correction (an infrequent operation) to a microprocessor to do, and thus reduce the cost even further. This would in turn allow the ECC word to be made larger while reducing the ECC hardware. The penalty is a slower access time, depending on the frequency of the errors.

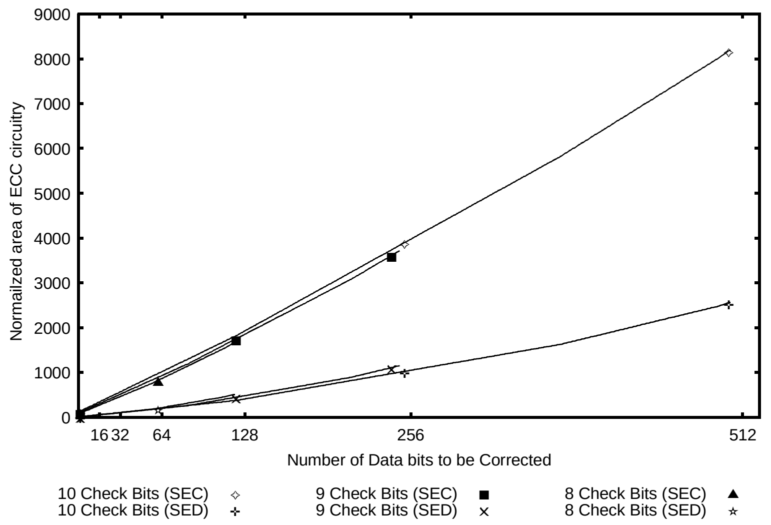

FIGURE 5. Comparison of ECC modified Hamming code circuitry area

Table 1 shows the normalized area needed for doing detection and correction, the normalized area needed for detection only, and the area ratio between the two. The first three rows show that as the number *k* of data bits increases, the area ratio decreases. The last four rows show what effect changing the number of check bits has on the area ratio. Since we would typically increase the number of check bits in order to accommodate a larger word size, this comparison was done with each check bit having the largest power of 2 number of data bits that it can correct. We can see that this ratio varies within the tight range of 3.8-4.0. This implies that a significant area (about 75%) can be saved if the syndrome bits are made available for performing the ECC calculation off-chip.

TABLE 1. Area of SEC-DED vs. SED

		Area		
r	**k**	**SEC-DED**	**SED**	**Ratio**
9	64	936	206	4.54
9	128	1855	485	3.82
9	247	3711	1151	3.22
8	64	863	215	4.01
9	128	1855	485	3.82
10	256	3975	1049	3.79
11	512	8519	2243	3.80

4.3. Estimated performance imact of doing off-chip correction

If we off-load the error correction task to the host CPU then we can save silicon area compared to full on-chip ECC. If we then consider the total time needed to access a DRAM, for very small number of defects, it may be acceptable to do off-chip correction.

Figure 6 shows the total normalized access time for a 64 Mbit DRAM. The plots are normalized against a DRAM that has no ECC. For this plot it was assumed that having error detection and error correction on chip would increase the delay by 0.5% (this would correspond to

50 ps for a PC100 DRAM). As expected if the correction is done on chip then the access time does not depend on the number of errors. For off-chip correction we assumed that the 150 MHz processor was 1.5 times faster than the memory bus speed, and we calculated the number of clock cycles needed to do the correction as follows (assuming that we are dealing with the same 128 data bit, 9 check bit Hamming code, and a 32-bit memory bus):

$$ClockCycles = \frac{4 \times 1.5}{1} + (8 \times 128) + 4 = 1034 \qquad \text{(EQ 2)}$$

We need 4 memory accesses at the memory bus speed and we have 9 check bits to decode, which require 8 instructions to decode for every one of the 128 data bits. We then have an additional 4 instructions in order to perform the bitwise XOR with the mask and the data.
We assume that half of the on-chip correction time was spent on detection and an equal time was also included in the off-chip correction. We can see from the figure that off-chip correction gives us a faster average access time if there are few errors.

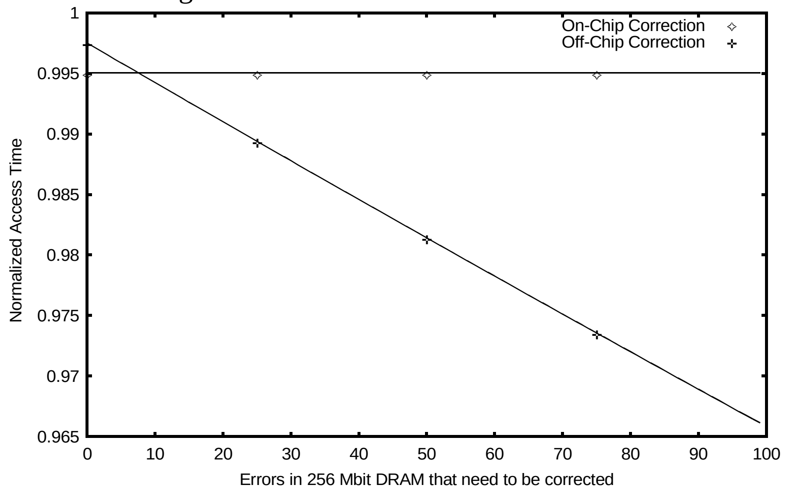

FIGURE 6. Access time for on-chip vs. off-chip error correcting

We can calculate the worst-case access time for off-chip correction as:

$$AccessTime = \frac{1034}{150MHz} = 6.89us \qquad \text{(EQ 3)}$$

The access time for on-chip correction as:

$$AccessTime = \frac{4}{100MHz}1.005 \cong 40ns \qquad \text{(EQ 4)}$$

The factor of 4 in Equation 4 accounts for the four accesses that are needed in order to get the same 128 bits. This means that off-chip correction takes about 170 times longer than on-chip correction. Whether this penalty is acceptable depends on the probability of errors and the access patterns of the application.

5. Other fault tolerance techniques

5.1. Bad block marking

Marking bad blocks was also examined as a method to reduce the cost per working bit. Currently hard drives record the bad sectors and then the product is sold with near-nominal storage capacity. We can recover more of the manufacturing cost if blocks are marked as bad and

the chip is sold. The cost per working bit assuming bad block marking is shown below in the results section. Some specific method of marking a block bad remains to be determined.
Any memory that contains redundant or bypassed elements must also contain a mapping, stored in nonvolatile memory, of where the defects are and how the elements are to be used. Such a "defect list" for hard disks can be kept on the disk itself since the media is nonvolatile. The deployment information for redundant rows and columns in conventional DRAM is stored in arrays of fuses, which are selectively blown using a laser. Conventional DRAMs have the additional constraint that a continuous range of addresses must be mapped, within a given access time, to functional memory cells. File memory applications don't have the hard constraints on providing continuous good memory with a constant access time. If file memories, however, are manufactured as part of early production, the required capacity to store a map of defects would be much larger than for conventional DRAM. For this reason, we propose using nonvolatile memory cells such as EPROM, EEPROM or flash for marking blocks bad. We approximate the non-volatile memory cell size as 10 times the size of a DRAM cell.

5.2. Redundancy

Since we are now examining the cost per working bit, we need to attribute a cost to the redundant columns and rows. This was done by assuming that the number of bits in the redundant elements are bits needed to make the DRAM array work. Redundant columns and rows also have some type of fuse or EEPROM in order to map where the redundancy is used. We also assumed that flash memory or EEPROM could be used to control the mapping for the redundant columns and rows. Nonvoltatile EPROM memory can be created with a single layer of polysilicon interconnect and thus be added to a DRAM process [4].

6. Results

As the yield goes down the cost per working bit tends to infinity. We found it more convenient to plot 1/*cpwb* or bits per unit cost instead of *cpwb*. A bits-per-cost of 1.00 can be interpreted as the lowest cost that can be achieved and would be achieved by using no ECC or redundancy and have 100% yield.

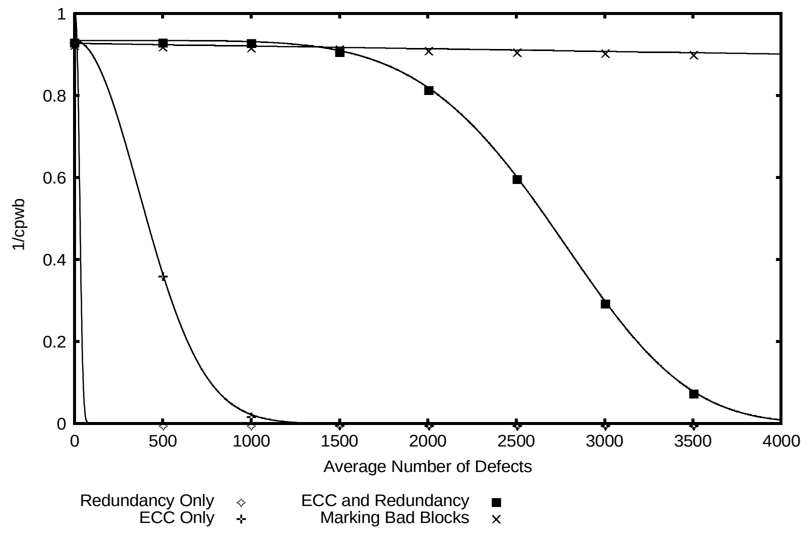

FIGURE 7. Cost per working bit using Marking Bad Blocks in a 16-Mbit DRAM

Figure 7 shows the results. We can see that this plot is very similar to Figure 1 except that the cost per working bit is used and that marking of bad blocks has been included. The cost per working bit for marking of bad blocks was calculated using a method of partial products [10]. We can see from the results that marking bad blocks has a linear effect on cost per working bit over this average number of defects. The slope and the intercept of the line depend on the granularity of the markable block size. If the granularity was the highest (and we could mark single defects as bad without any cost for marking) then the cost per working bit of the die would be the same as the cost per working bit of a single word. This is impractical however due to the increase in area to mark the bits as bad. For this work we marked 128-bit words as bad and this gives us the result as shown in Figure 7.

7. Conclusion

We have examined the performance of several techniques for decreasing the cost of memory that can tolerate higher manufacturing defect rates than conventional DRAM. In this study we applied a new measure, the relative cost per working bit, which is more meaningful here than yield. The techniques used for tolerating faults are Hamming ECC and marking bad blocks, with or without column redundancy. While these techniques may interfere to some extent with conventional random accesses to memory, file-oriented applications of memory can more easily accept these limitations. These limitation are: extended access time, non-uniform access times and discontinuous regions of good memory. As an example of non-uniform access time, significant overhead can be saved if the memory chips only detect errors in the ECC codeword and defer correction, in the rare cases when it's needed, to a processor. File-memory applications are in need of denser memory chips. The fault tolerance techniques described could hasten the economic crossover for higher density DRAM generations (e.g. 256 Mbit and 1 Gbit).

8. References

[1] J.A. Fifield, “A High-Speed On-Chip ECC System Using Modified Hamming Code”, in Sixteenth Euro. Solid-State Circuits Conf. Proc., Sept. 1990, pp. 265-268.

[2] J.A. Fifield and C.H. Stapper, “High-Speed On-Chip ECC for Synergistic Fault-Tolerant Memory Chip”, IEEE Journal of Solid-State Circuits, Vol. 26, No. 10, pp. 1449-1452, Oct. 1991.

[3] Heller et al., “Cascode Voltage Switch Logic,” ISSCC Dig. Tech. Papers 1984, p. 16-17.

[4] D. H. K. Hoe, et. al., “Cell and Circuit Design for Single-Poly EPROM”, IEEE Journal of Solid-State Circuits, Vol. 24, No. 4, Aug. 1989, pp. 1153-1157.

[5] M.Y. Hsiao, “A Class of Optimal Minimum Odd-weight-column SEC-DED Codes”, IBM J. R.&D., July 1970, pp. 395-401.

[6] H.L. Kalter, et. al., “A 50-ns 16Mb DRAM with a 10-ns Data Rate and On-Chip ECC”, IEEE Journal of Solid-State Circuits, Vol. 25, No. 5, Oct. 1990, pp. 1118-1128.

[7] B. Prince, “Semiconductor Memories”, John Wiley & Sons, 2nd Edition, 1991.

[8] D.P. Siewiorek, et al., “Reliable Computer Systems”, A. K. Peters, Natick, Massachusetts, 3rd Edition 1998, pp. 771-781.

[9] C.H. Stapper, “Synergistic Fault-Tolerance for Memory Chips”, IEEE Transactions on Computers, Vol. 41, No. 9, Sept. 1992, pp. 1078-1087.

[10] C.H. Stapper; A.N. McLaren; M. Dreckmann, “Yield Model For Productivity Optimization of VLSI Memory Chips With Redundancy and Partially Good Product”, IBM J. R.&D. Vol. 24, No. 3, pp. 398-409, May 1980.