Post-Manufacturing ECC Customization Based on Orthogonal Latin Square Codes and Its Application to Ultra-Low Power Caches

Rudrajit Datta and Nur A. Touba
Computer Engineering Research Center
The University of Texas at Austin, Austin, TX 78712
rudrajit.datta@mail.utexas.edu
touba@ece.utexas.edu

Abstract

The paper proposes the idea of implementing a general multi-bit error correcting code (ECC) based on Orthogonal Latin Square (OLS) Codes in on-chip hardware, but then selectively, on a chip-by-chip basis, using only a subset of the code's check bits (subset of the rows in its H-matrix) depending on the defect map for a particular chip. The defect map is obtained from a memory characterization test which identifies which cells are defective or marginal. The idea proposed here is that if a general t-bit error correcting code is implemented in hardware and requires c_{full}=n-k check bits for k information bits, then once the defect map is known, the defective cells become erasures w.r.t. the ECC. This fact can be used to select only a subset of the n-k rows in the H-matrix which are sufficient to provide the desired error detection/correction capability in the presences of the known erasures. By selectively reducing the number of rows in the H-matrix, the number of check bits that are actually stored and used, c_{used}, can be restricted and the corresponding unused ECC hardware disabled. This reduces the check bit storage requirements and hence frees up more of the cache for storing data and improving performance. This strategy is applied to the problem of providing reliable cache operation in ultra-low voltage modes, and results indicate that with the proposed post-manufacturing ECC customization, a fraction of the number of check bits are required compared to using a full OLS code for handling a particular defect rate.

1. Introduction

Voltage scaling, which is one of the most effective ways to reduce power consumption, is limited by a minimum value referred to as V_{ccmin} beyond which circuits may not function reliably [Taur 98]. Voltage scaling beyond V_{ccmin} gives rise to reliability issues, most notably for the memory sub-systems. In order for V_{ccmin} to be reduced to enable ultra-low power modes in microprocessors and other circuits, some means for handling high memory bit failure rates is required.

One general approach that has been proposed in [Wilkerson 08] is to trade off cache capacity for reliable low voltage operation. The idea is that in high-voltage operation, failure rate is low, so the entire cache is available. However, in low voltage operation, many memory cells become unreliable, so the cache size is sacrificed to increase reliability. Two approaches were proposed for this in [Wilkerson 08] which are based on performing a low voltage characterization test of the memory and identifying the failing cells. One approach (called word-disable) uses two physical lines to configure one logical line where only non-failing words are used. The other approach (called bit-fix) uses 25% of the cache to store a defect map which is used to bypass failing cells. Another approach that has been proposed in [Chisti 09] is based on using part of the cache to store the check bits for an Orthogonal Latin Squares (OLS) code [Hsiao 70] when operating in low voltage mode. An OLS code is a multi-bit error correcting code which is one-step majority decodable which allows faster encoding and decoding than traditional ECC at the cost of more check bits. The OLS code can be used to encode each cache line and correct errors that may arise due to failing cells as well as due to transient errors. The amount of usable cache size depends on the number of check bits required for the OLS code which in turn depends one how many errors the OLS code can correct.

The idea proposed in this paper is that after manufacturing a chip, a memory characterization test [Chang 07] can be performed to generate a defect map that identifies which cells fail or are vulnerable in low voltage operation. Once this information is known, the marginal cells effectively become erasures (i.e., errors with known locations), and it is possible to select a subset of the rows in the H-matrix for an OLS code which are sufficient to provide the desired level of error detection/correction capability in the presence of the defective cells for that particular chip. This reduces the number of check

978-1-4244-7207-9/10/$26.00 ©2010 IEEE

bits that need to be stored in the cache thereby freeing up more of the memory for storing data and improving performance. Note that conventional approaches that use spare rows and columns for repairing memories can only repair a small number of defects and hence are not effective for high defect rates [Kim 98]. Because OLS codes use majority decoding for error correction, it is very easy to disable portions of the code by simply masking bits at the input to the voters and adjusting the threshold of the voter. An OLS code can be selectively reduced by storing one configuration bit for each row in the *H*-matrix which indicates whether or not that row should be included when encoding and decoding. By selectively reducing the number of rows in the *H*-matrix that are used, the number of check bits that need to be stored for each word is reduced thereby reducing the amount of redundancy. The idea of post-manufacturing customization of the ECC can be applied to the problem of providing reliable cache operation in ultra-low power modes of operation. A *t*-error correcting OLS code is selected for a particular cache design based on the expected defect rate and implemented in full with on-chip hardware. It requires $c_{full}=n-k$ check bits for *k*-information bits. *t* must be selected large enough to handle the worst-case number of defects in any line of the cache. If some cache line has more than *t* defects, then the cache is unusable and the chip must be discarded. Based on the desired yield, *t* is selected appropriately. After a chip is manufactured, a memory characterization test is used to obtain a defect map. It may turn out that for a particular chip, no cache line has more than *(t-3)* defects present. Thus, the *t*-error detecting code is overkill. Even if some line has *t*-errors present, it still may not be necessary to use the entire H-matrix of the OLS code. In Sec. 3, a procedure is described for selecting a minimal number of rows in the H-matrix so that *e* additional transient errors can be corrected in addition to the permanent erasures identified in the defect map. In this way, the number of check bits that are actually used for a particular chip, c_{used}, is smaller than for using the full code, c_{full}, however, the code is still able to provide the desired level of protection. The configuration bits on the chip are set to indicate which rows of the H-matrix to use while all others are disabled. Compared to [Christi 09], the proposed method use up less of the cache for storing the check bits in low power mode, which means less caches misses and resultant memory accesses.

Even though the hardware for the full code, c_{full} is implemented on the chip with the proposed method, the functional design can be done assuming an a priori upper bound on c_{used}, the number of check bits that will actually be used post-manufacture, based on statistical analysis for the defect rate that is to be tolerated. This allows the desired level of fault tolerance to be achieved with fewer redundant memory cells storing check bits. In effect, the proposed method is exploiting the degree of freedom in selecting which portion of a larger code to use to get more leverage from a certain number of check bits in comparison to a conventional approach which uses the same code for every chip. It provides a beneficial tradeoff where a small amount of extra ECC circuitry (hardware redundancy) is added in order to reduce the number of check bits (information redundancy) that needs to be stored in the cache. This increases the performance of the cache while still achieving the desired level of reliability.

2. Related Work

Most prior work in memory ECC has focused on low failure rates present at normal operating voltages, and has not focused on the problem of persistent failures in caches operating at ultra low voltage where defect rates are very high.

For high defect rates, memory repair schemes based on spare rows and columns are not effective. Much higher levels of redundancy are required that can tolerate multi-bit errors in each cache line. In addition to the techniques in [Wilkerson 08] mentioned earlier, other prior work includes the two-dimensional ECC proposed by [Kim 07] which tolerates multiple bit errors due to non-persistent faults, but is slow and complicated to decode. Similarly the approach in [Kim 98] can tolerate as many faults as can be repaired by spare columns, which would be insufficient in the present context with high bit-error rate. In some cases, check bits are used along with spare rows and columns to get combined fault-tolerance. In [Stapper 92], interleaved words with redundant word lines and bit lines are used in addition to the check bits on each word. [Su 05] proposes an approach where the hard errors are mitigated by mapping to redundant elements and ECC is used for the soft errors. Such approaches will not be able to provide requisite fault tolerance under high bit error rates when there are not enough redundant elements to map all the hard errors.

The application of OLS codes for handling the high defect rates in low power caches as described in [Christi 09] provides a more attractive solution. While OLS codes require more redundancy than conventional ECC, the one-step majority encoding and decoding process is very fast and can be scaled up for handling large numbers of errors as opposed to BCH codes, which while providing the desired level of reliability requires multi-cycles for decoding [Lin 83]. The post-manufacturing customization approach

proposed in this paper can be used to reduce the number of check bits and hence the amount of redundancy required in the memory while still providing the desired level of reliability. Note that the proposed approach does not reduce the hardware requirements for the OLS ECC as the whole code needs to be implemented on-chip since the location of the defects is not known until post-manufacturing test is performed.

3. Orthogonal Latin Square Codes

A Latin square [Hsiao 70] of order (size) m is an m x m square array of the digits 0, 1, . . . , m - ***1,*** with each row and column a permutation of the digits 0,1, … , m - 1. Two Latin squares are orthogonal if, when one Latin square is superimposed on the other, every ordered pair of elements appears only once.

In general, a t-error correcting majority decodable code works on the principle that $2t + 1$ copies of each information bit are generated from $2t + 1$ independent sources. One copy is the bit itself received from memory or any transmitting device. The other $2t$ copies are generated from $2t$ parity relations involving the bit. By choosing a set of h Latin squares that are pair-wise orthogonal, one can construct a parity check matrix such that the number of 1's in each column is $2t = h + 2$. The orthogonality condition ensures that for any bit d**,** there exists a set of $2t$ parity check equations orthogonal on d_i**,** and thus makes the code self-orthogonal and one-step majority decodable. One-step majority decoding is the fastest parallel decoding method. The t-error correcting codes generated by OLS codes [Hsiao 70] have m^2 data bits and $2tm$ check bits per word.

Let the m^2 data bits be denoted by a vector:

$$D = [d_0, d_1, ...d_{m^2-1}] \quad \text{..... (1)}$$

Then the $2tm$ check-bit equations for t-error correcting are obtained from the following parity check matrix H:

$$H = \left[\begin{array}{c|c} M_1 & \\ M_2 & \\ M_3 & I_{2tm} \\ : & \\ M_{2t} & \end{array}\right] \quad \text{..... (2)}$$

$\boldsymbol{I}_{2tm}$ is an identity matrix of order $2tm$ and $\boldsymbol{M_1}, \ldots, \boldsymbol{M_{2t}}$ are submatrices of size m x m^2. These submatrices $\boldsymbol{M_1}$....$\boldsymbol{M_{2t}}$ have the form

$$M_1 = \begin{bmatrix} 11...1 & ... & \\ : & 11...1 & : \\ : & ... & 11...1 \end{bmatrix}_{mxm^2} \quad \text{..... (3)}$$

$$M_2 = \begin{bmatrix} I_m & I_m & ... & ... & ... & I_m \end{bmatrix}_{mxm^2} \text{... (4)}$$

The matrices $\boldsymbol{M_1}$....$\boldsymbol{M_{2t}}$ are derived from the existing set of orthogonal Latin squares $L_1, L_2, \ldots, L_{2t-2}$ of size m x m. Denote the set of Latin squares as,

$$L_1 = \left[l_{ij}^1\right]_{mxm}$$
$$L_2 = \left[l_{ij}^2\right]_{mxm}$$
$$\vdots$$
$$\vdots$$
$$L_{2t-2} = \left[l_{ij}^{2t-2}\right]_{mxm}$$

where $l_{ij} \in \{1, 2, \ldots, m\}$

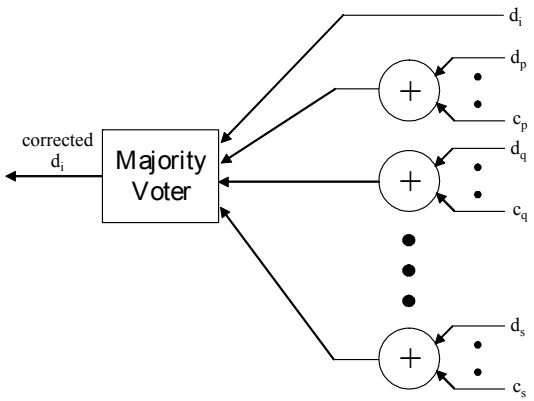

Figure 1. Decoding Data Bit d_i with Majority Voter

Once the H-matrix is constructed, the decoding for each data bit is done using a majority voter as illustrated in Fig. 1. When decoding data bit d_i, the set of bits in each of the $2t$ H-matrix rows that d_i is present in are XORed together and serve as an input to a majority voter along with d_i itself giving a total of $2t+1$ inputs. Since the set of inputs to the XOR gates are orthogonal, the OLS code will provide the correct output as long as the number of errors is less than half the number of inputs to each voter, i.e., t or less. Note that OLS coding does not need to generate a syndrome, but can "correct" errors directly from majority voting.

4. Proposed Scheme

The proposed scheme leverages memory tests [Chang 07] that can be performed at manufacture time or out in the field when the system is booted up. These tests identify which are the vulnerable bits in the cache. The defect map can then be used to select a subset of the rows from the original t-error correcting OLS matrix. The suspect bits will be referred to as erasures (i.e., errors with known locations). For any cache line, given the defect map,

the goal is to be able to correct all erasures along with one or more random errors that may occur on any other bit in that cache line.

For ease of explanation, two terms will be defined with respect to each information bit d_i – a "good row" and a "bad row". A "good row" for information bit d_i is a row of the OLS H-matrix that does not have a '1' in any bit position where there is an erasure in any line in the set of considered cache lines *C* except for erasures in bit *i* itself. Such a row can be used to unambiguously decode information bit d_i in the presence of the considered erasures. A "bad row" for an information bit d_i is one row of the OLS H-matrix which has a '1' in one or more bit positions where one or more erasures exist in the set of considered cache lines *C*. A small example is shown in Fig. 2 where the set of considered cache lines contains two cache lines where 'E' denotes a vulnerable cell which is treated as an erasure. In this example, *H-row1* shown in Fig. 2 would be a good row for any information bit because it does not intersect with any erasure bit. *H-row2* intersects with the erasure in d_1, so it would only be a good row for d_1, but would be considered a bad row for all other information bits. *H-row3* intersects with erasures in both d_3 and d_5, so it would not be considered a bad row for all information bits.

Each information bit is generated by a majority voter whose inputs correspond to rows in the *H*-matrix plus the information bit itself. In a *t*-error correcting OLS code, each information bit is generated by a *2t+1* input majority voter. By construction of the code, at most *t* inputs to the majority voter can be bad rows at any given time, so the voter will always produce a correct output as long as the number of errors in any cache line is *t* or less. In the proposed approach, the goal is to tolerate *e* transient errors on top of the known erasures identified in the defect map. This can generally be accomplished with much fewer than *2t+1* inputs to the voter, and hence some inputs can be disabled (i.e., masked off). For information bit d_i, the set of good and bad rows can be identified for the considered erasures. Inputs to the voter for d_i can be disabled provided the following relationship is maintained:

"good rows" – "bad rows" $\geq 2(e+1)$ (Condition 1)

This ensures that if *e* transient errors occur which could cause *e* good rows to become bad rows, the voter will still produce a correct output even if d_i itself has an erasure. For example, if *e*=1, then the voter needs a minimum of 5 inputs with 4 of them coming from good rows and one input coming from d_i itself. In the worst case where d_i has an erasure and one row has a transient error, the 3 remaining good rows would out vote the two erroneous inputs. Alternatively, a 7-input voter could be used with inputs coming from 5 good rows, one bad row, and d_i itself. Any size voter can be used provided the number of good rows is larger than the number of bad rows by a sufficient amount as per *Condition 1*.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
line1	-	E	-	-	-	E	-	-
line2	-	-	-	E	-	-	-	-
H-row1	1	0	0	0	1	0	1	1
H-row2	0	1	1	0	1	0	0	1
H-row3	1	0	0	1	0	1	1	0

Figure 2. Example where H-row1 good for all d_i, H-row2 good for only d_1, and H-row3 bad for all d_i.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
row1	G	-	-	-	G	-	G	G
row2	-	G	B	-	B	-	-	B
row3	B	-	-	B	-	B	B	-

Figure 3. Covering matrix for example in Fig. 2

The problem of selecting a minimal set of rows so that the inputs to the voter for every information bit satisfies condition 1 for a given set of erasures can be formulated as a covering problem. A covering matrix can be formed where each column corresponds to an information bit, and each row corresponds to a row in the *H*-matrix. Each entry in the matrix is a 'G' if the row is a good row for the corresponding information bit, a 'B' if it is a bad row, and a '-' if the H-matrix row does not intersect with the information bit. The covering matrix for the small example in Fig. 2 is shown in Fig. 3. There are 8 information bits and three H-matrix rows. Note that this is not a complete OLS code, but only a small fragment to illustrate how the covering matrix is formed.

Once the covering matrix is formed, then a sufficient number of rows need to be selected to satisfy condition 1 for all information bits. As long as the following condition is satisfied for a *t*-error correcting OLS code, there will always exist a solution to the covering problem, i.e., if nothing else, the solution where all rows in the OLS H-matrix are included will work.

(Max erasures in any line) $+ e \leq t$ (Condition 2)

While the covering problem is NP-complete, good heuristic algorithms can be employed. For example, a greedy procedure that first selects rows with the maximum number of G's weighted by the difference between the G's and B's for each column could be used. Rows are iteratively selected until a valid

solution is found that satisfies Condition 1. Other heuristic covering algorithms can be used as well [Vazirani 04]. Another strategy would be to start with all rows selected and iteratively remove rows as long as condition 1 is satisfied.

Note that the covering problem is solved w.r.t. a set of considered erasures. Ideally, the set of considered erasures (and hence the covering matrix) should be recomputed for each cache line and the covering problem for every cache line should be simultaneously solved. However, the computation complexity for this is infeasible. One solution to simplifying the problem would be to consider all erasures in the cache as occurring in the same line which would allow forming a single covering matrix to solve. However, this would badly over constrain the problem. The proposed procedure is to first consider only the erasures in the worst-case cache line (i.e., the cache line that has the most erasures). Solve the covering problem for that, and then check if it is a valid solution for all other cache lines. If not, then the erasures for one of the cache lines that it is not a solution for are added to the set of considered erasures, and the procedure is repeated. This is done iteratively until a solution that works for all cache lines is found. Typically the worst-case cache lines are the limiting factor, so solving for them typically solves for all cache lines.

5. Implementation

One way to implement the proposed approach is as follows. The maximum number of erasures that needs to be tolerated in any line of the cache to achieve a desired yield is statistically computed based on the expected memory cell defect probability, the word size of the cache, and the number of lines in the cache. The OLS code is then selected so that it satisfies Condition 2 where the number of bits that the OLS code can correct is equal to the maximum number of erasures in any line plus the number of transient errors that are to be tolerated. Let c_{full} be the number of check bits for the selected OLS code. The maximum number of check bits that the proposed method will require to achieve the desired yield, c_{used}, can then be determined through Monte Carlo simulation. The memory is then designed so that it has c_{used} redundant columns for each line to store the check bits. For the application to low power caches (as described in [Christi 09]), the cache would be reconfigured in low power mode so that it could store c_{used} check bits for each line.

The full OLS code is implemented on the chip for generating all c_{full} check bits when writing to the cache, and for performing the decoding with all c_{full} check bits when reading from the cache. Configuration circuitry is added so that the full OLS code can be reduced down to c_{used} check bits based on the configuration bits that are set for each chip after performing a memory characterization test as illustrated in Fig. 4. There is one configuration bit for each of the c_{full} check bits for the code. The configuration bit is a '1' if that check bit is to be used, and is a '0' if that check bit is to be disabled. The configuration bits can either be stored with fuses if they are to be one-time programmed at manufacture time, or they can be stored in flip-flops if they are to be programmed by software each time the chip is powered up after performing a memory BIST. Note that the switch networks in Fig. 4 can be simplified by placing constraints on the ways that c_{full} can be mapped to c_{used}.

The decoding process for each data bit is based on majority voting between the H-matrix rows associated with the data bit. The decoding process can be configured as shown in Fig. 5. Since some H-matrix rows may not be included in the reduced code, so inputs to the voter associated with those rows are masked off with AND gates. The majority voter is replaced with a threshold voter that gives a '1' output if the number of inputs that are equal to '1' is greater than or equal to the threshold T. The value of T is configured so that it is equal to $\lceil$(number of non-masked inputs)/2$\rceil$. For example, if there are 7 inputs to the voter, but 2 of them are not part of the reduced code, then they are masked off, and the threshold of the voter is set to 3. As long as no more than 2 out of the 5 non-masked voter inputs are correct, the output of the voter will be correct, so two errors can be tolerated. The control lines in Fig. 5 are generated by control logic that decodes the configuration bits (as shown in Fig. 4).

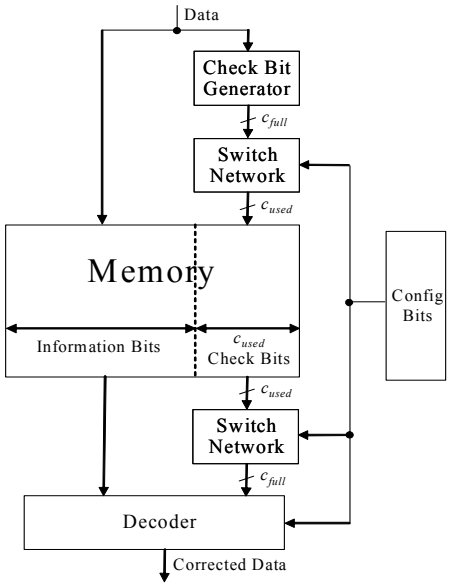

Figure 4. Block Diagram of Scheme

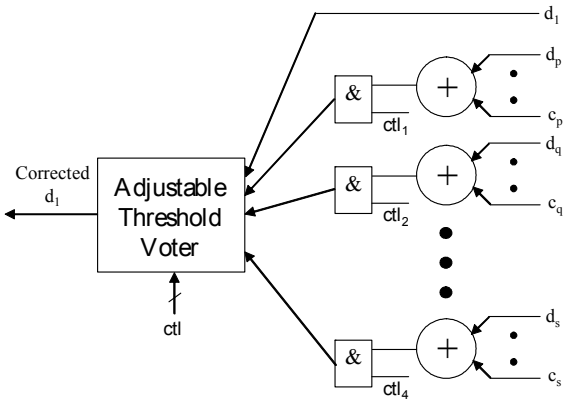

Figure 5. Decoding Logic for Data Bit

6. Experimental Results

Table 1 shows the results for a constant cache size of 16KB where simulations were performed for 1000 caches. The first column shows the word size, the second column shows the bit error rate (probability that a bit is defective), and the next two columns show the average and maximum number of check bits required for tolerating all defects among 1000 caches using a conventional OLS code. The last two columns show the same data using the proposed method where the OLS code is customized for each cache for withstanding the effect of one transient error and all erasures.

As can be seen from the results in Table 1, significant reduction in the number of check bits can be achieved. The reduction becomes larger for higher bit-error probabilities and larger word sizes. The relatively less improvement for lower bit-error probabilities is due to the small number of erasures that are present. Also, smaller word sizes have less error correction capability. An OLS code for 64 data bits can correct up to 4 errors, an OLS code for 256 data bits can correct up to 8 errors, and an OLS code for 484 data bits can correct up to 11 errors.

Table 2 shows the results for different size caches with a word size of 256 bits, for tolerating one transient error on top of all the erasures present in the defect map. As can be seen, with the proposed method, the maximum number of check bits that are required is reduced by 30% to 42% which allows more of the cache to be used for storing data.

Fig. 6 shows the distribution of check bits required per cache for the proposed scheme compared to conventional OLS. If a threshold on the number of check bits is set at some point, it can be seen that the yield for proposed method would be much higher.

Table 1. Results for Different Bit-Error Rates and Word Sizes across Constant Cache Size of 64KB

Word Size (Bits)	Bit-error Rate	Check bits for conventional OLS		Check bits for customized OLS	
		Avg	Max	Avg	Max
64	10^{-3}	63	64	55	64
	10^{-4}	41	64	35	56
	10^{-5}	32	32	32	32
256	10^{-3}	177	224	138	157
	10^{-4}	98	128	84	107
	10^{-5}	66	102	64	68
484	10^{-3}	295	396	198	230
	10^{-4}	143	176	117	139
	10^{-5}	92	132	89	115

Table 2. Results for Different Cache Sizes with Word Size of 256 Bits and Bit-Error Rate of 10^{-3}

Cache Size (Bytes)	Check bits for conventional OLS		Check bits for customized OLS		Percentage reduction in Max. Check Bits
	Avg	Max	Avg	Max	
16 KB	155	224	117	145	35.27
32 KB	166	256	125	148	42.19
64 KB	175	256	134	156	39.06
128 KB	208	256	163	177	30.81

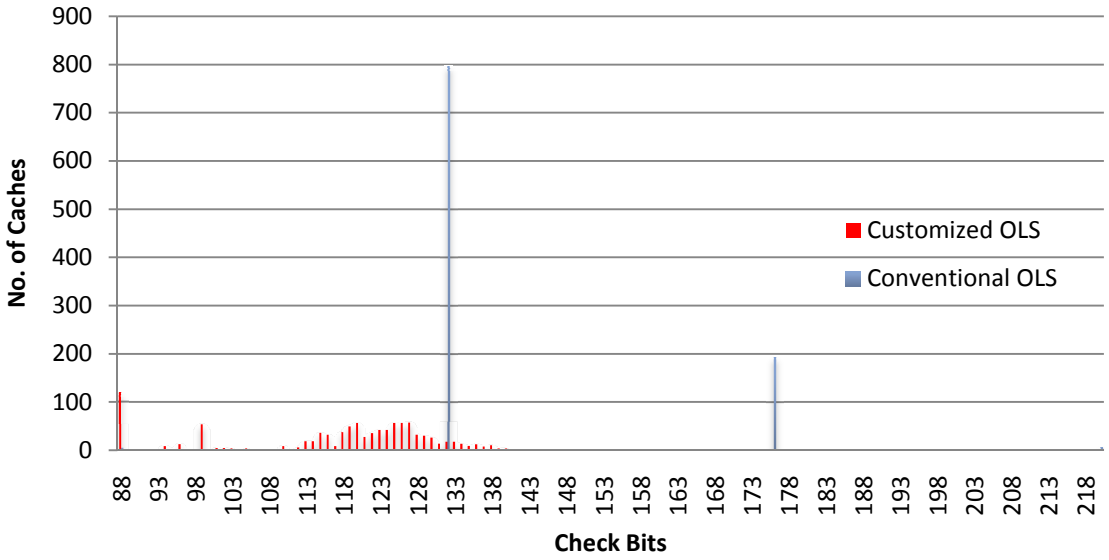

Figure 6: Distribution of 484-bit Word, 64 KB Cache at 10^{-3} Bit-Error Rate

7. Conclusions

The check bit overhead for tolerating large numbers of marginal cells in ultra-low power caches is quite large. The proposed method can be used to significantly reduce the check bit overhead while still providing the same level of reliability especially when bit-error rates are higher.

The idea of post-manufacture ECC customization can be applied to other problems to provide efficient fault tolerance when high defect rates are present.

Acknowledgements

The authors would like to thank Chris Wilkerson and Shih-Lien Lu of Intel Corporation for introducing us to this problem and for many helpful technical discussions. This research was supported in part by the National Science Foundation under Grant No. CCR-0426608.

References

[Chang 07] Chang, J., Ming Huang; Shoemaker, J., Benoit, J., Szu-Liang Chen, Wei Chen, Siufu Chiu, Ganesan, R., Leong, G., Lukka, V., Rusu, S., Srivastava, D., "The 65-nm 16-MB Shared On-Diie L3Cache for the Dual-Core Intel® Xeon Processor 7100 Series,," *IEEE Journal of Soliid-state Circuits,* Vol.. 42, no. 4,,pp. 846-852, April,, 2007.

[Chisti 09] Chishti Zeshan, Alameldeen Alaa R., Wilkerson Chris, Wu Wei and Lu S.-L., "Improving Cache Lifetime Reliability at Ultra-low Voltages", *Proc. Of International Symposium on Microarchitecture*, Dec. 2009

[Hsiao 70] Hsiao, M. Y., Borren, D.C., and Chien, R.T., "Orthogonal Latin Square Codes", *IBM Journal of Research and Development*, Vol. 14, No. 4, pp. 390-394, July 1970.

[Kim 07] Jangwoo Kim, Hardavellas, N., Ken Mai, Falsafi, B., Hoe, J.C, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding", *Proc. Of International Symposium on Micro-architecture*, December 2007.

[Kim 98] Kim, I., Y. Zorian, G. Komoriya, H. Pham, F.P. Higgins, and J.L. Lewandowski, "Built In Self Repair for Embedded High Density SRAM," *Proc. of International Test Conference*, pp. 1112-1119, 1998.

[Lin 83] Lin, S. Costello, D., .*Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983

[Stapper 92] Stapper, Charles H., Hsing-san Lee, "Synergistic Fault-Tolerance for Memory Chips", *Proc of IEEE Transactions on Computers*, Vol. 41, No. 9, pp 1078-1087, Sep. 1992.

[Su 05] Su, Ching-Lung, Yeh, Yi-Ting, Wu, Cheng-Wen, "An Integrated ECC and Redundancy Repair Scheme for Memory Reliability Enhancement," *Proc. of Defect and Fault Tolerance in VLSI Systems*, pp. 81-89, 2005

[Taur 98] Taur, Y., Ning, T.H., *Fundamentals of Modern VLSI Decices*, Cambridge University Press, 1998

[Vazirani 04] Vazirani, V.V., *Approximation Algorithms*, Springer, 2004..

[Wilkerson 08] Wilkerson, C. Hongliang Gao, Alameldeen, A.R., Chishti, Z., Khellah, M., Shih-Lien Lu, "Trading of Cache Capacity for Reliability to Enable Low Voltage Operation", *Proc. of International Symposium on Computer Architecture*, pp.203-214, June 2008.