Analyzing the Impact of Joint Optimization of Cell Size, Redundancy, and ECC on Low-Voltage SRAM Array Total Area

Nam Sung Kim, Stark C. Draper, Shi-Ting Zhou, Sumeet Katariya, Hamid Reza Ghasemi, and Taejoon Park

***Abstract*—The increasing power consumption of processors has made power reduction a first-order priority in processor design. Voltage scaling is one of the most powerful power-reduction techniques introduced to date, but is limited to some minimum voltage V_{DDMIN}. Below V_{DDMIN} on-chip SRAM cells cannot all operate reliably due to increased process variability with technology scaling. The use of larger SRAM cells, which are less sensitive to process variability, allows a reduction in V_{DDMIN}. However, since the large-scale memory structures such as last-level caches (LLCs) often determine the V_{DDMIN} of processors, these structures cannot afford to use large SRAM cells due to the resulting increase in die area. In this paper we first propose a joint optimization of LLC cell size, the number of redundant cells, and the strength of error-correction coding (ECC) to minimize total SRAM area while meeting yield and V_{DDMIN} targets. The joint use of redundant cells and ECC enables the use of smaller cell sizes while maintaining design targets. Smaller cell sizes more than make up for the extra cells required by redundancy and ECC. In 32-nm technology our joint approach yields a 27% reduction in total SRAM area (including the extra cells) when targeting 90% yield and 600 mV V_{DDMIN}. Second, we demonstrate that the ECC used to repair defective cells can be combined with a simple architectural technique, which can also fix particle-induced soft errors, without increasing ECC strength or processor runtime.**

***Index Terms*—Error correction coding (ECC), low-voltage SRAM, redundancy, voltage scaling.**

I. Introduction

As technology scaling allows us to integrate more transistors, reducing overall power consumption has become a critical design priority. Although voltage scaling is one of the most effective techniques for minimizing power consumption, processors do not operate reliably at low supply voltage. This is because random uncorrelated process variations, which increases with technology scaling, result in mismatches between the transistors in SRAM cells. This make the cells unstable at low voltages and increase the cell failure probability; a failure mode addressed by imposing some minimum operating voltage, V_{DDMIN}. Processors typically contain very large-scale SRAM structures, such as on-chip caches, and the V_{DDMIN} needed by these structures often determines the V_{DDMIN} of the whole processor.

To lower V_{DDMIN} one can increase the size of the constituent transistors in 6-transistor cells, which reduces the amount of random process variations (i.e., mismatches) [1], or one can adopt 8- or 10-transistor cells [2], [3]. However, these approaches substantially increase the area of large on-chip caches. On the other hand, one can use redundant columns of cells or error-correction coding (ECC). A small number of extra (redundant) columns of cells are made available to replace a column of cells that contains a cell (or cells) with manufacturing defects (e.g., stuck-at-faults) [4]. Following manufacturing, a test for defective cells is performed. The SRAM array is then reconfigured to access a redundant column instead of the column containing the defective cell(s). Although ECC was originally adopted to protect memory from particle-induced soft errors [5], it can also be used to correct errors arised from low-voltage operation.

SRAM arrays composed of smaller cells, which exhibit higher failure probability than larger ones, require more redundancy and stronger ECC to achieve a target V_{DDMIN}. Although the overall cell area allocated for data bits of this solution is small, the total SRAM area can be very large due to substantial area that must be allocated for the redundant and ECC bits. On the other hand, increased transistor size (i.e., larger cell size) reduces the amount of random process variation, and results in a substantially lower cell failure probability. This greatly lowers the number of redundant or ECC bits required to achieve a V_{DDMIN} target. However, using large cells results in a significant increase in total SRAM array area. In one way or another we must pay for a low V_{DDMIN} through increased area. However, balancing cell size with the degree of fault tolerance (i.e., redundancy and/or ECC) will allow us to minimize the total area of SRAM arrays.

Prior studies of V_{DDMIN} reduction through the use of redundancy, ECC, or other fault-tolerance techniques, did not take advantage of the trade-off between SRAM cell size and failure probability. The approach we take in this paper is to combine the proven design techniques discussed above in a novel hybrid manner that delivers a quantitative boost in effectiveness when compared to the application of any single technique. First, for SRAM cells, we explore the change in single-cell failure probability as the size of the constituent transistors is varied. Second, we apply a necessary amount of fault tolerance needed, through redundancy and/or ECC to achieve a target V_{DDMIN} for the given cell failure probability, which is determined by the cell size. As we show, a joint optimization that includes transistor sizing can quite substantially impact the overall failure probability of large on-chip caches. Finally, we demonstrate that the ECC used to repair defective cells can be combined with a simple architectural technique to fix particle-induced soft errors without increasing ECC strength or processor runtime.

The remainder of the paper is organized as follows. In Section II, we discuss the relationship between failure probability and size of SRAM cells. In Section III, we jointly explore the size of SRAM cells, the degree of redundancy, and the strength of the ECCs used to minimize the total SRAM area. In Section IV, we describe how to use a mask error correction technique in combination with an ECC that has both error correction and detection capabilities, to correct errors from both particle strikes and defects more efficiently. Section V concludes the study.

II. Failure Probability Versus SRAM Cell Size

We estimate the failure probability of SRAM cells in our study using the method described in [6]. The degree of transistor mismatch is quantified by the standard deviation of each transistor's threshold voltage V_{TH}, i.e., σV_{TH}, and as is discussed in [1], σV_{TH} increases as transistor size decreases. The σV_{TH} for an nMOS (pMOS) transistor with width equal to the minimum length in a high-performance 32-nm predictive technology model is 24 mV (29.2 mV) [7]. For each statistical sample, we apply random V_{TH} values to individual transistors in an SRAM cell and check read and write failures for a 128 by 256 SRAM array using SPICE.

Our base-line cell (C1) has the minimum width $(\mathrm{W} = 3\lambda)$ for all 6 transistors. To analyze the relative area of 6 different SRAM cells (C1–C6), we create cell layouts using a TSMC 0.18-μm technology design rule and Cadence Virtuoso layout editor since the design rule

Manuscript received February 06, 2011; revised June 17, 2011; accepted August 12, 2011. Date of publication November 15, 2011; date of current version August 02, 2012. This work was supported in part by an NSF Grant (CCF-1016262), by a Multidisciplinary Research Award from the University of Wisconsin-Madison Graduate School, and by NSF CAREER Awards (CCF-0953603 and CCF-0844539).

The authors are with the University of Wisconsin-Madison, WI 53706 USA (e-mail: nam.sung.kim@gmail.com).

Digital Object Identifier 10.1109/TVLSI.2011.2173220

1063-8210/$26.00 © 2011 IEEE

TABLE I
W_{PD}, W_{PU}, AND W_{PG} FOR C1~C6 CELLS AND THEIR AREA RELATIVE TO C1 SIZE

	C1	C2	C3	C4	C5	C6
W_{PD}	3λ	5λ	8λ	11λ	14λ	16λ
W_{PU}	3λ	4λ	4λ	4λ	4λ	5λ
W_{PG}	3λ	5λ	8λ	11λ	14λ	16λ
Relative Area	1.00	1.12	1.23	1.35	1.46	1.58

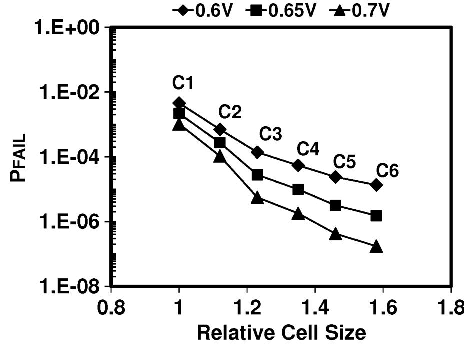

Fig. 1. Failure probabilities as a function of cell size at $V_{\mathrm{DD}} = 0.6$, 0.65, and 0.7 V.

for very advanced technology is not yet available. An SRAM cell consists of a pair of pull-down (PD), pass-gate (PG), and pull-up (PU) transistors. The widths of PD, PG, and PU transistors are represented by W_{PD}, W_{PG}, and Wpu, respectively. The cell height is usually fixed while $W_{\mathrm{PD}}+W_{\mathrm{PU}}$ determines the width of the cell. The minimum size cell has $W_{\mathrm{PD}} + W_{\mathrm{PU}}$ equal to 6λ plus the fixed width cost. Second, for each cell, we sweep the size of W_{PD}, W_{PG}, and W_{PU} to find the minimum failure probability of each cell for a given constraint, i.e., $W_{\mathrm{PD}} + W_{\mathrm{PU}} \leq$ 6, 9, 12, 15, 18, and 21λ for C1, C2, C3, C4, C5, and C6, respectively. In Table I we tabulate the optimized W_{PD}, W_{PG}, and W_{PU} for each cell and its cell area relative to the area of C1. All the possible combinations of W_{PD}, W_{PG}, and W_{PU} are exhaustively searched to minimize the failure probability at 600 mV with the step value equal to 0.5λ for a given cell size.

In the 32-nm technology with the given σV_{TH} for nMOS and pMOS transistors, the failure probabilities are fairly high for very small cells (e.g., C1 and C2) at low voltages. This is because transistors with smaller W_{PD}, W_{PG}, and W_{PU} exhibit larger σV_{TH} and thus their on-current characteristics are more sensitive to V_{TH} variability. However, as the cell size increases, the failure probability of a single cell decreases consistently across different voltages. This is shown in Fig. 1 where the failure probabilities are plotted versus cell size at $V_{\mathrm{DD}} = 0.6$, 0.65, and 0.7 V. Note that the cell failure probabilities for the largest two cells—C4 and C5 are very close to the Intel's data based on 45-nm technology [8].

III. Joint Optimization of Cell Size, Redundancy, and ECC

A. Redundancy

Various redundancy techniques have been widely adopted in both SRAM and DRAM to repair defective cells caused by stuck-at-faults [4], [9]. More recently, such techniques have also been used to reduce V_{DDMIN} for SRAM. A column redundancy implementation in an SRAM array using a "shift redundancy" scheme is developed in [9]. Basically, if any cell in a column fails, the next column replaces the failed column. In turn, the next column is replaced with the column next to it, and so on. This is repeated until a redundant column replaces the last non-redundant column in a segment of SRAM columns. Note that only one redundant column is made available for each set of m columns due to the increasing complexity (in m) of the steering logic circuit [10]. For example, consider the case when the data I/O width is 32 and there are two redundant columns. The first redundant column can only fix a single column in the first 16-bit segment and the second one in the second segment. Thus, we cannot repair two failing columns in the first 16-bit segment even with two redundant columns unless a much more complex and expensive data steering mechanism is implemented.

Let the probability of failure of an individual cell be P_{cell}. When a column in an array has i cells the column-failure probability, P_{col}, is

$$P_{\mathrm{col}} = 1 - (1 - P_{\mathrm{cell}})^{i}. \quad (1)$$

Next let the number of redundant columns be equal to k in an array with j columns, i.e., one redundant column per $m = j/k$ columns. Finally, let a segment be a set of m consecutive columns. Then, the failure probabilities of a block, an array, and a cache are

$$P_{\mathrm{segment}} = 1 - (1 - P_{\mathrm{col}})^{m} - m \cdot P_{\mathrm{col}} \cdot (1 - P_{\mathrm{col}})^{m-1} \quad (2)$$

$$P_{array} = 1 - (1 - P_{\mathrm{segment}})^{k} \quad (3)$$

$$P_{\mathrm{cache}} = 1 - (1 - P_{array})^{l} \quad (4)$$

where l is the number of arrays in an L-MB cache—$(L \cdot 8 \cdot 2^{20})/(i \cdot j)$. Further discussion of the area and delay impact of the use of redundancy is addressed in [6].

B. Error Correction Coding (ECC)

ECC adds redundant *parity* or *check bits* to detect and repair errors in stored data. Single-error-correction, double-error-detection (SECDED) codes are widely used in both DRAM and SRAM, primarily to protect them from particle-induced soft errors [5]. However, due to the increasing probability of SRAM cell failure as technology is scaled, multi-bit error correction codes are now being used in large on-chip caches to satisfy yield targets at desired V_{DDMIN} values. With a q-bit error correction capability requiring r check bits, the failure probability of a k-bit data block is given by

$$P_{\mathrm{block}} = 1 - \sum_{i=0}^{q} \binom{n}{i} \left((1 - P_{\mathrm{cell}})^{n-i} \cdot P_{\mathrm{cell}}^{i} \right) \quad (5)$$

where n is the length of the code $n = k + r$. The failure probability of a cache with a total of j data block is

$$P_{\mathrm{cache}} = 1 - (1 - P_{\mathrm{block}})^{j}. \quad (6)$$

Due to the substantially higher failure rates of SRAM cells at very low voltages, we need to implement multiple-bit error correction capability per data block to meet yield targets. Conventional multiple-bit ECCs such as BCH codes do not have encoding and decoding complexity that scales in a simple way with correction capability. For this reason in this study we use Orthogonal Latin Square (OLS) codes to implement ECCs for $q \geq 3$ [11]. While OLS codes require more check bits than Hamming or BCH codes, they have modular correction hardware and lower coding complexity, often allowing for fast encoding/decoding for multiple-bit error corrections. When k is equal to p^2, an OLS code requires $2 \cdot p \cdot q$ check bits. For example, n becomes $256 + 32 \cdot q$ for each 256-bit data block and q-bit error correction capability. Further discussion of the area and delay impacts of the use of ECCs is addressed [6].

C. Joint Use of Redundancy and ECC

Since column redundancy and ECC are distinct techniques, we can apply them jointly. Combining column redundancy with ECC allows

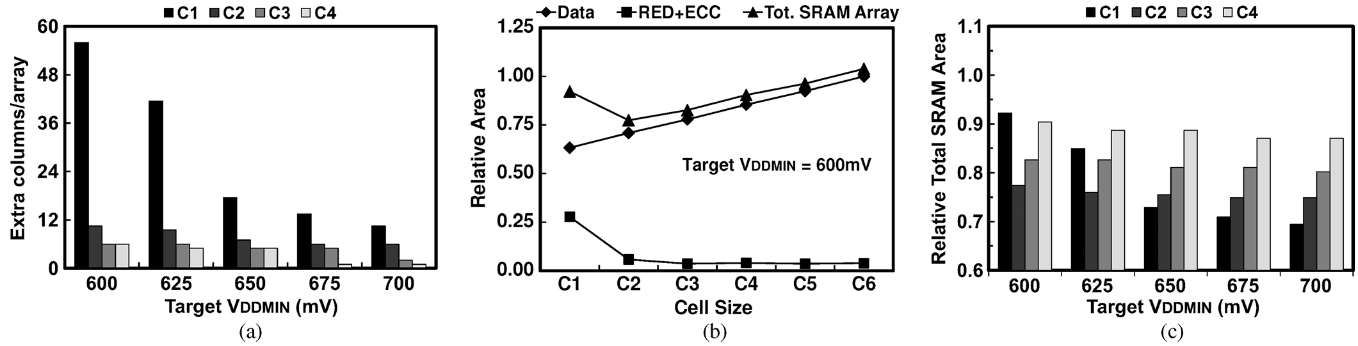

Fig. 2. (a) Optimal number of extra columns per array to implement both redundancy and ECC as a function of V_{DD}, (b) the relative area of data and extra bits as a function of cell size at $V_{\rm DDMIN} = 600$ mV, and (c) the relative total SRAM array area including the overhead of extra bits as a function of target $V_{\rm DDMIN}$. The plots in (a), (b), and (c) are for 90% yield and a 4 MB cache. The area numbers in (b) and (c) are normalized to the area of an array implemented with the largest C6 cells where neither of redundancy nor ECC is used.

for finer adjustment of overall cache failure probability to meet a target yield. The optimum mixture of the two techniques depends on cell size. For example, the largest C6 cell has very low failure probability. But, they still require either at least one redundant column per array or single-bit error correction capability per 256-bit data block to satisfy 90% yield for 4 MB caches at 600 mV. Single-bit error correction capability per 256-bit data block requires 10 check bits, i.e., 4% overhead. On the other hand, a single redundant column per array incurs only 0.8% overhead. In both cases the overhead is calculated in terms of the total number of extra bits is required per array. However, as the failure rate of cells increases (as individual transistors scale down), ECC becomes a more efficient method, i.e., less area overhead is required to protect the caches against multiple-bit failures.

The problem of analyzing the exact failure probabilities of a scheme where column redundancy and ECC are combined is mathematically complex. In fact, error calculations turn into integer programs. In this paper, we suggest a heuristic approach to calculating such failure probabilities. The approach yields good estimates of failure probabilities, with the advantage of being mathematically tractable. Our approach is as follows. Say that the number of redundant columns is k, the error correction capability is q, and the probability of a cell failure is $P_{\rm cell}$.

1) We compute the failure probability of a cache, $P_{\rm cache-red}$ which only uses k redundant columns per array using (1)–(4).
2) We calculate the "equivalent failure probability" of a cell, $P_{\rm cell-eq}$. This is the cell failure probability that will make the failure probability of the cache equal to $P_{\rm cache-red}$ if there were no redundant columns.
3) We estimate the failure probability $P_{\rm cache}$ for a cache with an error correction capability of q bits using (6) where the equivalent cell failure probability $P_{\rm cell-eq}$ is used in (5) instead of $P_{\rm cell}$.

Fig. 2 plots (a) the optimal number of extra columns required per array to implement both column redundancy and ECC as a function of target $V_{\rm DDMIN}$, (b) the relative area of data and extra bits as a function of cell size for $V_{\rm DDMIN}$ equal to 600 mV, and (c) the relative total SRAM array area including extra columns as a function of target $V_{\rm DDMIN}$. The target yield used in all three plots is 90% yield and the cache size considered is 4 MB. The area numbers plotted in (b) and (c) are normalized to the area of an array implemented with the largest C6 cells when no redundancy or ECC is used.

To obtain Fig. 2(a), we explore all the possible redundancy and ECC parameter pairs (k, q) that satisfy 90% yield with each cell at each target $V_{\rm DDMIN}$ and chose the combination that results in the minimum extra column overhead. For each of the four cell sizes, C1, C2, C3, and C4, and at each $V_{\rm DDMIN}$ (600, 625, 650, 675, and 700 mV), respectively the best choice of design parameters is {(8, 3), (1, 2), (1, 1), (1, 1)}, {(32, 2), (0, 2), (1, 1), (1, 0)}, {(8, 2), (2, 1), (0, 1), (0, 1)}, {(4, 2), (1, 1), (0, 1), (1, 0)}, and {(1, 2), (1, 1), (2, 0), (1, 0)}. While the column redundancy is implemented per array, the ECC is applied to the cells at the same row across multiple arrays. Thus, we assume that we distribute the ECC check-bits evenly across the multiple arrays to calculate "the number of extra columns per array" in Fig. 2(a). Note that in many layouts, separate arrays are used to store the ECC check bits. However, for the purposes of this study, we calculated the overhead on per array basis.

Examining Fig. 2(b) we see that as the cell size decreases the number of extra columns to implement both column redundancy and ECC increases. Although C1 is the smallest cell size, when C1 cells are used at 600 mV the overall overhead for redundancy and ECC to satisfy a yield of 90% is quite considerable. Meanwhile, while C2 cells are larger than C1, they are smaller than C3 or C4, and the choice of C2 minimizes total SRAM area. This is because the failure probability of the C2 cells is not too high and allows redundancy and ECC to be maximally effective.

We now contrast the improvement in efficiency of the redundancy-only, the ECC-only, and the redundancy plus ECC designs. First, compare the ECC-only results with the redundancy plus ECC of Fig. 2(c) for a 90% target yield. Concentrate on $V_{\rm DDMIN}$ of 600 mV and cache size of 4 MB. The ECC-only results from [6] show a reduction in total SRAM area of 7% when C3 cells are used. In contrast ECC plus redundancy achieves a 23% reduction, achieved by C2 cells. In other words, joint application of redundancy and ECC shift the optimal cell size to a smaller size. Table II summarizes the reduction in total SRAM array area of optimized 4, 8, and 16 MB caches. The optimal cell size choices for each target $V_{\rm DDMIN}$ are indicated by the shaded cells.

IV. ECC Technique for Fixing Failures From Defects and Particle Strikes

Error correction was originally introduced in memories to repair particle-induced errors. However, if all ECC capability is used to fix errors due to defective cells failing at low voltages, we either lose the ability to protect memory cells from particle strikes or we need to increase the ECC strength by one to retain that capability. To give a sense of the magnitude of the overhead, increasing the error correction capability from double-error to triple-error corrections requires 47% more check bits; this translates into an increases in ECC check bit overhead, relative to the number of data bits, from 7% to 14%.

In order to provide both low $V_{\rm DDMIN}$ and protection from particle strikes, while using ECC cost-effectively, we make two observations. First, only a small number of blocks in an on-chip cache will require the full error correction capability to combat errors due to defective cells at

TABLE II
RELATIVE TOTAL SRAM ARRAY AREA TO SATISFY 90% YIELD FOR 4, 8, AND 16 MB CACHES AND DIFFERENT TARGET V_{DDMIN}

		Redundancy only						ECC only						Redundancy + ECC					
Cache Size	*VDDMIN*	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6	C1	C2	C3	C4	C5	C6
4MB	600mV	—	—	—	—	1.17	1.14	1.03	0.97	0.84	0.92	0.96	1.04	0.92	0.77	0.83	0.90	0.96	1.04
	625mV	—	—	—	1.08	0.97	1.03	0.95	0.76	0.81	0.89	0.96	1.04	0.85	0.76	0.83	0.89	0.96	1.03
	650mV	—	—	0.99	0.92	0.95	1.02	0.95	0.76	0.81	0.89	0.96	1.04	0.73	0.75	0.81	0.89	0.95	1.02
	675mV	—	—	0.84	0.87	0.95	1.02	0.87	0.76	0.81	0.89	0.96	1.04	0.71	0.75	0.81	0.87	0.95	1.02
	700mV	—	—	0.80	0.87	0.95	1.02	0.87	0.76	0.81	0.89	0.96	1.04	0.69	0.75	0.80	0.87	0.95	1.02
8MB	600mV	—	—	—	—	—	1.26	1.03	0.97	0.84	0.92	0.96	1.04	0.96	0.77	0.83	0.90	0.96	1.04
	625mV	—	—	—	—	1.00	1.04	0.95	0.97	0.84	0.89	0.96	1.04	0.85	0.77	0.83	0.89	0.96	1.04
	650mV	—	—	—	0.97	0.95	1.02	0.95	0.76	0.84	0.89	0.96	1.04	0.77	0.76	0.81	0.89	0.95	1.02
	675mV	—	—	0.89	0.88	0.95	1.02	0.87	0.76	0.81	0.89	0.96	1.04	0.73	0.75	0.81	0.88	0.95	1.02
	700mV	—	—	0.81	0.87	0.95	1.02	0.87	0.76	0.81	0.89	0.96	1.04	0.70	0.75	0.81	0.87	0.95	1.02
16MB	600mV	—	—	—	—	—	1.26	1.03	0.97	0.84	0.92	1.00	1.04	0.96	0.77	0.83	0.90	0.98	1.04
	625mV	—	—	—	—	1.05	1.07	0.95	0.97	0.84	0.92	0.96	1.04	0.85	0.77	0.83	0.90	0.96	1.04
	650mV	—	—	—	1.08	0.97	1.02	0.95	0.76	0.84	0.89	0.96	1.04	0.77	0.76	0.83	0.89	0.96	1.02
	675mV	—	—	0.99	0.89	0.95	1.02	0.95	0.76	0.81	0.89	0.96	1.04	0.73	0.75	0.81	0.89	0.95	1.02
	700mV	—	—	0.84	0.87	0.95	1.02	0.87	0.76	0.81	0.89	0.96	1.04	0.71	0.75	0.81	0.87	0.95	1.02

low voltage. As an example, consider C3 cells and a target of 600 mV for V_{DDMIN}. This combination requires double-bit error corrections per data block. The probability that a particular data block requires the two bits of error correction is

$$P_2 = \frac{\binom{n}{2}\left((1-P_{\mathrm{cell}})^{n-2}\cdot P_{\mathrm{cell}}^2\right)}{(1-P_{\mathrm{block}})}. \quad (7)$$

The renormalization by $(1-P_{\mathrm{block}})$ follows from a simple application of Bayes' rule as the caches we are considering are those that passed the manufacturing test and do not have any blocks with more than two defective cells. The probability P_2 for C3 cells at 600 mV is 7.1776×10^{-6}; the expected number of blocks with two defective cells is $P_2 \times j$, where j is the number of blocks in the cache. For an 8 MB cache with C3 cells operating 600 mV the expected number of 256-bit blocks that require the full double-error correction capability is less than two. In other words, zero or at most one defective cell exists in all other 256-bit blocks. For these blocks per-block double-error correction is sufficient to repair errors due both to both defective cells and to particle strikes.[1] Second, particle strikes occur infrequently; less than two per 10^6 hours of operation of a 1 Mb on-chip SRAM [12]. The probability of any particular bit having a soft error in one cycle, P_{soft}, is 1.015×10^{-25} for a processor operating at 3 GHz [12]. Then, the per-block probability of an error due to particle strikes occurring in a single cycle is $n \times P_{\mathrm{soft}}$; and the expected number blocks with two defective cells and a soft error in one cycle is $(P_2 \times j) \times (n \times P_{\mathrm{soft}})$. For an example, it takes 1.904×10^{22} cycles for a block with two defective cells is struck by a particle. From the above discussion we observe that facing a situation in which we need both to correct a soft error and two hard errors will be very rare. Therefore, if we have an auxiliary procedure to correct such events, even if computationally more complex, the performance impact will be practically zero, due to the rarity of the events. We now describe such a technique.

Let us consider using a double-error correction code that also has a triple error detecting capability. If we use such a code, we can repair a block with two defective cells and one soft error using a *mask error correction* technique [13]. The procedure is the following.

[1]A particle strike can upset multiple neighboring bits in a SRAM array implemented with a sub-100-nm technology. However, SRAM array uses a column multiplexing scheme, where neighboring bits belong to different addresses. This results in a single error per address.

1) Read the block with two errors from defective cells and one error from a particle strike (e.g., 101101 where the first two bits have errors due to defective cells and the last one bit has an error due to a particle strike).
2) Write the bits back to the block in the on-chip cache after complement the bits (e.g., 010001).
3) Re-read the block from the on-chip cache (e.g., 100010 where the values of the first two bits are unchanged since they are stored in the defective cells).
4) Re-write the bits back to the block in the on-chip cache after re-complement bits (e.g., 011101) and repair the bit error due to the particle strike with double-error correction code (e.g., 011100).

In general, the *read-complement-read-complement* procedure illustrated in step 1–4 allow us to fix a single error due to a particle strike in a block with q defective cells using q-error correction $q+1$ detection code. Once the error due to a particle strike is repaired, the block with q errors for the subsequent accesses can be fixed using q-error correcting capability. The total number of on-chip accesses to repair a block with q defective cells and one error due to particle strikes is four; the number of total cycles for a repair is four multiplied by the latency of on-chip cache access. For example, an 8 MB cache with $q = 2$ requires 13 cycles per access [6] and thus it takes 52 cycles for repairing a block with two defective cells and one error due to particle strikes. As we only need to repair such errors every 5.236×10^{24} cycles, the performance overhead is negligible.

V. CONCLUSION

In this paper, we discuss how the size of individual SRAM cells used in an on-chip cache should be chosen jointly with the amount of redundancy and the strength of ECC implemented to minimize total SRAM area. Although the failure probability of SRAMs decreases with increasing cell size, using large SRAM cells to implement on-chip caches is often not economical due to the area cost. Instead redundancy and/or ECC should be used to repair failing cells to meet target yields thus enabling reliable operation at lower voltages. In this paper we combine redundancy, ECC, and appropriate cell sizing, to yield a jointly optimized SRAM design with minimum area for target yields and V_{DDMIN}. As an example of our final designs, consider our experiments for 32 nm technology and design parameters of cache sizes from 4 to 16 MB,

a yield target of 90% and a V_{DDMIN} of 600 mV. Our final design reduces the total required SRAM area by 27% while minimally impacting the performance of multi-core processors. Further, we demonstrate that the ECC used for repairing defective cells does not need an additional correction capability to cope with particle-induced errors. Finally, our analysis is performed using 6-transistor SRAM cells. However, the proposed method is extendable to on-chip caches that can be implemented with other types of SRAM cells (i.e., 8- or 10-transistor SRAM cells) since these cells also exhibit the same fundamental trade-off between cell size and failure probability.

REFERENCES

[1] J. Croon, S. Decoutere, W. Sansen, and H. Maes, "Physical modeling and prediction of the matching properties of MOSFETs," in *Proc. IEEE Euro. Solid-State Device Res. Conf.*, 2004, pp. 193–196.

[2] L. Chang, Y. Nakamura, R. Montoye, J. Sawada, A. Martin, K. Kinoshita, F. Gebara, K. Agarwal, D. Acharyya, W. Haensch, K. Hosokawa, and D. Jamsek, "A 5.3 GHz 8T-SRAM with operation down to 0.41 V in 65 nm CMOS," in *Proc. IEEE VLSI Circuit Symp.*, 2007, pp. 252–253.

[3] I. Chang, J. Kim, and K. Roy, "A 32 kb 10 T subthreshold SRAM array with bit-interleaving and differential read scheme in 90 nm CMOS," in *Proc. IEEE Int. Solid-State Circuit Conf.*, 2008, pp. 388–389.

[4] S. Schuster, "Multiple word/bit line redundancy for semiconductor memories," *IEEE J. Solid-State Circuits*, vol. 13, no. 5, pp. 276–287, May 1978.

[5] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices*, vol. 26, no. 1, pp. 2–9, Jan. 1979.

[6] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim, "Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC," in *Proc. IEEE Int. Symp. Comput. Design*, 2010, pp. 112–117.

[7] ITRS. 2009.

[8] C. Wilkerson, H. Gao, A. Alameldeen, Z. Chishti, M. Khellah, and S.-L Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *Proc. IEEE Int. Symp. Comput. Arch.*, 2008, pp. 203–214.

[9] A. Ohba, S. Ohbayashi, T. Shiomi, S. Takano, K. Anami, H. Honda, Y. Ishigaki, M. Hatanaka, S. Nagao, and S. Kayano, "A 7-ns 1-Mb biCMOS ECL SRAM with shift redundancy," *IEEE J. Solid-State Circuits*, vol. 26, no. 4, pp. 507–512, Apr. 1990.

[10] T. Kirihata, Y. Watanabe, H. Wong, J. DeBrosse, M. Yoshida, D. Kato, S. Fujii, M. Wordeman, P. Poechmueller, S. Parke, and Y. Asao, "Fault-tolerant designs for 256 Mb DRAM," *IEEE J. Solid-State Circuits*, vol. 31, no. 4, pp. 558–566, Apr. 1996.

[11] M. Hsiao, D. Bossen, and R. Chien, "Orthogonal latin square codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 390–394, 1970.

[12] J. Suh, M. Manoochehri, M. Annavaram, and M. Dubois, "Soft error benchmarking of L2 caches with PARMA," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, 2011, pp. 85–96.

[13] T. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[14] T. N. Miller, R. Thomas, J. Dinan, B. Adcock, and R. Teodorescu, "Parichute: Generalized turbocode-based error correction for near-threshold caches," in *Proc. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2007, pp. 351–362.

Complexity of Computing Convex Subgraphs in Custom Instruction Synthesis

Joseph Reddington and Kubilay Atasu

***Abstract*—Synthesis of custom instruction processors from high-level application descriptions involves automated evaluation of data-flow subgraphs as custom instruction candidates. A subgraph S of a graph D, is convex if no two vertices of S are connected by a path in D that is not also in S. An algorithm for enumerating all convex subgraphs of a directed acyclic graph (DAG) under input, output, and forbidden vertex constraints was given by Pozzi, Atasu, and Ienne. We show that this algorithm makes no more than $O(|V(D)|^{N_{\rm in}+N_{\rm out}+1})$ recursive calls, where $|V(D)|$ is the number of vertices in D, and $N_{\rm in}$ and $N_{\rm out}$ are input and output constraints, respectively. Therefore, when $N_{\rm in}$ and $N_{\rm out}$ are constants, the algorithm is of polynomial complexity. Furthermore, a convex subgraph S is a maximal convex subgraph if it is not a proper subgraph of some other convex subgraph, assuming that both are valid under forbidden vertex constraints. The largest maximal convex subgraph is called the maximum convex subgraph. There exist popular algorithms that enumerate maximal convex subgraphs, which all have exponential worst-case time complexity. This work shows that although no polynomial-time maximal convex subgraph enumeration algorithm can exist, the related maximum convex subgraph problem can be solved in polynomial time.**

***Index Terms*—Complexity, subgraph enumeration.**

I. INTRODUCTION AND RELATED WORK

Automated synthesis of custom instruction processors from high-level application descriptions has been an active research domain for the last decade. The automated techniques typically rely on compiler infrastructures to extract the source-level data flow graphs. Subgraphs of these graphs are evaluated as custom instruction candidates. There are potentially an exponential number of possible subgraphs, so much work in the area has focused on restricting the size of the set of "candidate instructions" without reducing the potential quality.

The traditional approach in custom instruction synthesis enumerated the subgraphs by restricting the maximum number of input and output operands that custom instructions can have to the available register file ports [1], [6]–[11]. Additional heuristic methods that generate custom instructions without exhaustive enumeration exist [12]–[14]. In [15], Ahn *et al.* describe a dynamic-programming method that can compute input/output constrained subgraphs that optimize an additive merit function in polynomial-time.

Recently, a class of subgraphs, called "maximal convex subgraphs" received considerable attention [2]–[5]. It is shown in [3] that the subgraph that offers the highest speed-up can be found by enumerating maximal convex subgraphs and applying a resource constrained scheduling on each such subgraph to pipeline the register file accesses. The number of subgraphs that should be enumerated are of exponential order and the scheduling problem is NP-Hard in the general case. Therefore, a heuristic scheduling algorithm is used in [3] to reduce the complexity. Computing the maximum [4], [5], [16], convex subgraphs is motivated by the experimental evidence provided in, where experiments on benchmarks with relatively large basic blocks

Manuscript received April 19, 2011; revised August 22, 2011; accepted October 13, 2011. Date of publication November 10, 2011; date of current version August 02, 2012.

J. Reddington is with Royal Holloway College, University of London, Egham TW20 0EX, U.K. (e-mail: joseph@cs.rhul.ac.uk).

K. Atasu is with IBM Research-Zurich, Ruschlikon CH-8803, Switzerland (e-mail: kat@zurich.ibm.com).

Digital Object Identifier 10.1109/TVLSI.2011.2173221

1063-8210/$26.00 © 2011 IEEE