Rescuing Uncorrectable Fault Patterns in On-chip Memories through Error Pattern Transformation

Henry Duwe
University of Illinois
at Urbana-Champaign
Email: duweiii2@illinois.edu

Xun Jian
University of Illinois
at Urbana-Champaign
Email: xunjian1@illinois.edu

Daniel Petrisko
University of Illinois
at Urbana-Champaign
Email: petrisk2@illinois.edu

Rakesh Kumar
University of Illinois
at Urbana-Champaign
Email: rakeshk@illinois.edu

***Abstract*—Voltage scaling can effectively reduce processor power, but also reduces the reliability of the SRAM cells in on-chip memories. Therefore, it is often accompanied by the use of an error correcting code (ECC). To enable reliable and efficient memory operation at low voltages, ECCs for on-chip memories must provide both high error coverage and low correction latency. In this paper, we propose error pattern transformation, a novel low-latency error correction technique that allows on-chip memories to be scaled to voltages lower than what has been previously possible. Our technique relies on the observation that the number of on-chip memory errors that many ECCs can correct differs widely depending on the error patterns in the logical words they protect. We propose adaptively rearranging the logical bit to physical bit mapping per word according to the BIST-detectable fault pattern in the physical word. The adaptive logical bit to physical bit mapping transforms many uncorrectable error patterns in the logical words into correctable error patterns and, therefore, improving on-chip memory reliability. This reduces the minimum voltage at which on-chip memory can run by 70mV over the best low-latency ECC baseline, leading to a 25.7% core-wide power reduction for an ARM Cortex-A7-like core. Energy per instruction is reduced by 15.7% compared to the best baseline.**

I. Introduction

Current and future process technologies face serious power challenges. A well-known technique that effectively reduces processor power consumption is supply voltage scaling. One major challenge for voltage scaling is that the reliability of SRAM cells decreases with the supply voltage. As the supply voltage decreases, an rapidly increasing fraction of SRAM cells become faulty due to process variations [1], [2]. This problem becomes more pronounced at smaller feature sizes (Fig. 1) and is expected to get worse [3]. As such, the reliability of SRAM cells, and not the reliability of logic circuits, often determines the extent to which supply voltage can be reduced [4], [5], [6]. Although using more robust SRAM cell implementations with larger area and leakage can improve the reliability of SRAMs at low supply voltages, a significant fraction of these SRAM cells are still faulty at low voltages, as shown in Fig. 2.

Many recent works have investigated using an error correcting code, or ECC, to tolerate the high fault rates of SRAMs at low supply voltages [7], [8], [9], [10]. However, there is often a strong trade-off between error correction coverage and latency between different ECC schemes. For example, Fig. 3 shows that the four-bit-correcting BCH (127,64) ECC can tolerate many factors higher bit failure rates than weaker ECC schemes for the same coverage; however, this comes at the cost of incurring up to 20X higher latency overhead[1]. Although stronger ECC schemes provide the high error correction coverage needed to enable low supply voltage, their high error correction latencies make them unattractive for on-chip memories, where latency is often critical.

[1]BCH decoder latency is taken from [11]; the calculation assumes an oracular BCH decoder whose latency depends on the actual number of faults in a word, as opposed to having a constant worst-case latency.

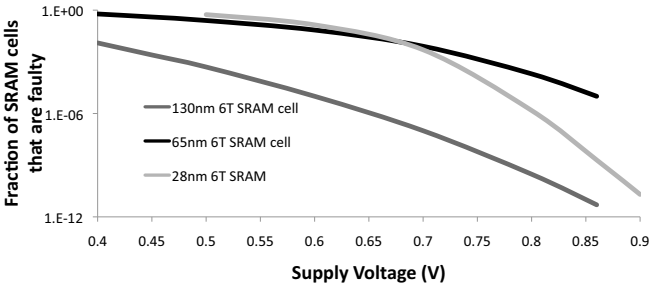


Fig. 1: **SRAM failure probability w.r.t. to voltage for different processing technologies [1], [2], [12].** Technologies with smaller feature sizes exhibit higher fault rates.

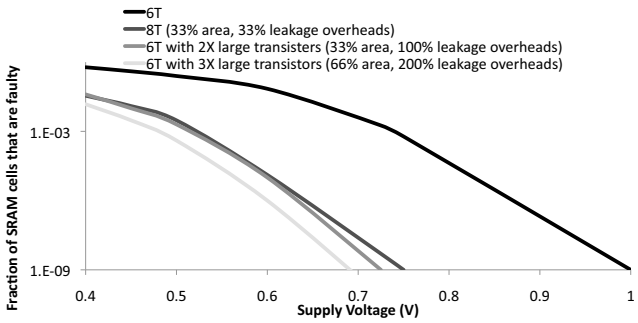


Fig. 2: **65nm SRAM failure probability w.r.t. to voltage for different SRAM cells [2].** Larger SRAMs have lower but still significant fault rates at low voltages.

In this paper, we propose error pattern transformation, a novel low-latency microarchitectural technique that allows on-chip memories to be scaled to voltages lower than what has been previously possible. We observe that although many ECCs only guarantee correction of a few memory errors, they can opportunistically correct more errors depending on the error pattern in the logical words they protect. As such, we propose transforming the uncorrectable error patterns in logical words into correctable ones by intelligently rearranging the logical bit to physical bit mapping when storing a logical word into a physical word in an on-chip memory. This improves on-chip memory reliability and, therefore, leads to a reduction in the minimum voltage at which on-chip memory can be run reliably.

Error pattern transformation (EPT) is a general technique that can be applied on top of many prior works on improving on-chip memory reliability. EPT can provide reliability benefits even in the presence of soft errors and erratic faults. Our evaluations show that EPT reduces the minimum voltage at which on-chip memory can run by 70mV

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.61

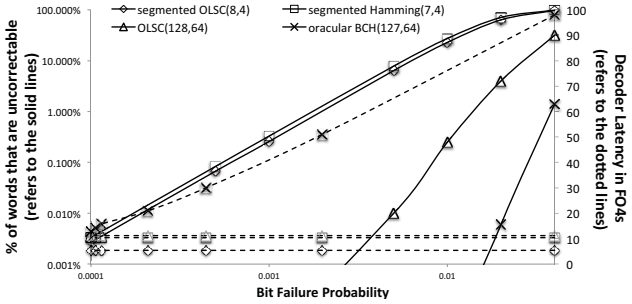


Fig. 3: **ECC latency and coverage for 64-bit data words.** Low-latency ECCs typically provide lower coverages than long-latency ECCs.

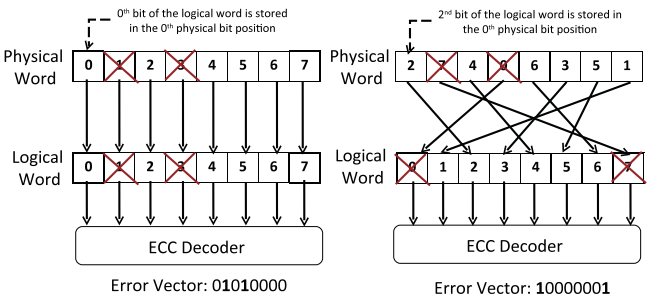


Fig. 6: **Possible logical to physical bit orderings.** The error pattern in a logical word depends on the bit ordering used to access the physical word.

over the best low-latency ECC baseline, leading to a 25.7% core-wide power reduction. Energy per instruction is reduced by 15.7% compared to the best baseline.

II. Motivation

The key insight driving this work is that for for many ECCs, the number of errors that they can correct differs widely depending on the observed error pattern (*i.e.,* the bit locations of the individual errors). Consider, for example, a segmented ECC, which breaks a word into several independent and equally-sized segments, each with its own check bits for error correction [7]. The number of errors that a segmented ECC can correct depends on how many errors are located in each segment, as illustrated in Fig. 4.

Our second example ECC does not break a word into independent segments. An Orthogonal Latin Square Code, OLSC(128,64), protects 64 bits of data with 128-64=64 bits of redundancy. OLSC(128,64) performs error correction in multiple levels of majority voting, where each level consists of multiple majority voters processing overlapping sets of input bits in parallel for low-latency error correction. When there are too many errors in the inputs to one of the majority voters,

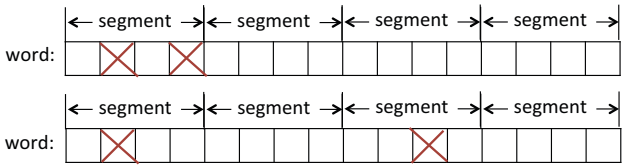


Fig. 4: **Uncorrectable (above) and correctable (below) error patterns in a segmented ECC that corrects one error per segment.** Low-latency ECCs can correct different numbers of errors for different error patterns.

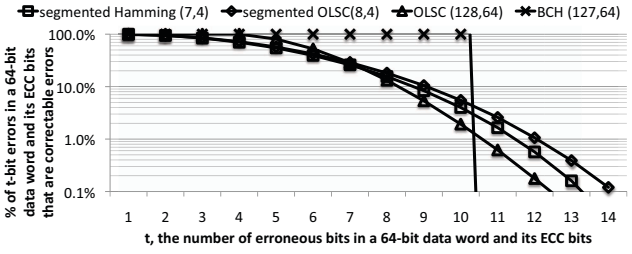


Fig. 5: **Fraction of t-bit errors that are correctable by different ECCs.** Many t-bit errors are correctable even for large values of t.

the output of the voter can flip; this can in turn flip the outputs of subsequent voting levels. As such, the number of errors that the OLSC ECC can correct again depends on the error pattern. OLSC(128,64) can correct up to 32 bad bits among the 128 bits for some specific error patterns; for all error patterns in general, however, OLSC(128,64) only guarantees correction of up to four bad bits.

In fact, it is quite common for many low-latency ECCs (the latency-coverage trade-offs of different ECCs are shown in Fig. 3) to have some error patterns for which more errors can be corrected than other error patterns. Fig. 5 shows the fraction of t-bit error patterns that are correctable under different ECCs. Fig. 5 considers 64-bit data words; it presents the unsegmented OLSC(128,64) ECC as well as a segmented Hamming(7,4) ECC and a segmented OLSC(8,4) ECC, where a segmented ECC(n,k) breaks a 64-bit data word into segments of k data bits with $n-k$ check bits per segment. Fig. 5 shows that a significant fraction of t-bit error patterns are correctable for a large range of t even though segmented Hamming(7,4), segmented OLSC(8,4), and OLSC (128,64) guarantee correction of only 1, 1, and 4 errors, respectively.

In this paper, we improve the coverage of low-latency ECCs by adaptively transforming an uncorrectable BIST-detectable t-bit error pattern in a logical word into one of the many correctable t-bit error patterns (the latency-coverage trade-offs of different ECCs are shown in Fig. 3). This allows low-latency ECCs to obtain similar coverage as high-latency ECCs (e.g., the coverage of the BCH(127,64) shown in Fig. 5), while incurring only a small fraction of the latter's long latency overhead. The improved coverage allows on-chip memories to be scaled to voltages lower than what has been previously possible.

III. Error Pattern Transformation

Consider the proposed technique in context of a cache. We observe that the same physical fault pattern in a cache word can manifest as different error patterns in the logical word presented to the ECC decoder depending on the mapping of the logical bits in a cache word to the physical bits stored in the physical cache word. For example, the logical bit to physical bit mapping from a logical word to a physical cache word, or simply *bit ordering*, shown in the left half and the right half of Fig. 6 generates error vectors '01010000' and '10000001', respectively, even though the physical fault pattern in the cache word remains the same. We seek to transform uncorrectable error patterns in the logical words into correctable error patterns by supporting multiple bit orderings per physical cache word. In comparison, conventional cache designs support only a single bit ordering, which maps bit x in a logical word (e.g., logical bit 0) to the same bit x in the physical cache word (e.g., physical bit 0).

When accessing a cache word with a known (*BIST-detectable*) physical fault pattern, our proposal uses a pre-recorded bit ordering that generates only correctable errors for the known physical fault pattern in the cache word; the pre-recorded bit ordering is selected from a pool of available bit orderings. Under our proposal, a physical fault pattern in a cache word is uncorrectable only if no bit ordering that generates only correctable error patterns for the given physical fault pattern can be found; intuitively, the larger the number of supported bit orderings, the less likely that a bit ordering that generates only correctable error patterns cannot be found for a given physical fault pattern.

Fig. 7 shows the calculated ideal fault coverages for different ECCs under different numbers of supported bit orderings[2]. Fig. 7 shows that the fraction of correctable fault patterns increases significantly as the number of supported bit orderings increases. In fact, for a sufficient number of available bit orderings (e.g., 32 to 256 bit orderings), coverage approaches, and sometimes, exceeds the high-latency BCH ECC. This improved coverage can be used to to be scale voltages lower than what has been previously possible for cache memories.

Our proposal requires addressing three main challenges. First, how to identify the appropriate bit ordering for each cache word. Second, how to record the identified bit orderings in a space-efficient manner. Third, how to enact the appropriate bit ordering during cache accesses at low latency overhead. We address these challenges in Sections III-A to III-C. Finally, Section III-D walks through examples of how to apply our proposal.

A. Bit Ordering Selection

Low voltage SRAM faults are largely hard faults that can be identified using a BIST (Built-in Self Test) routine that is run either during post-manufacture testing or at set intervals during processor lifetime [4], [8]. The following describes a general approach, applicable to different ECCs, for selecting the appropriate bit ordering that only generates correctable errors for a physical fault pattern that has been identified in a cache word.

The first step is to identify the physical fault vector of a cache word at low supply voltage using BIST. Next, the fault vector is sent to the bit reordering logic (described in detail in Section III-C), which modifies the ordering of the bits in the fault vector. The modified fault vector is then fed into a correctability checker to check whether all possible error patterns due to the modified fault vector are correctable. The correctability checker is specific to an ECC. For example, for the segmented Hamming (7,4) ECC, the correctability checker counts the number of '1's in every segment of seven adjacent bits in the fault vector under test; the checker asserts a fail signal if any segment contains more than one '1' and asserts a pass signal otherwise. If the check passes, the tested bit ordering for the cache word is recorded so that the ordering will be used to access that cache word for all future accesses. Otherwise, the bit reordering logic checks another bit ordering of the identified fault vector until one of the bit orderings passes or until all available bit orderings have been tested; in the latter case, the cache word is reported as having an uncorrectable fault pattern.

Fig. 8 summarizes the steps described above. Note that all of the steps only have to be performed once for each cache word (e.g., once during post-manufacturing testing). They are not performed during normal operations and, therefore, do not affect runtime performance. Assuming a 50s BIST overhead to identify all fault patterns for a 2MB cache [8], our 32KB L1 cache requires 0.78s. Assuming the worst case of 2^8 OID calculations per word and two cycles per calculation (one cycle to generate a new bit ordering and one cycle to check the new ordering), a 32KB L1 cache requires only 131K cycles. Therefore, selection of OID bits would add a negligible amount of time to the BIST routine.

B. Tracking Bit Orderings

To support 2^k different bit orderings per cache word, we allocate k bits per physical cache word to record the chosen bit ordering for each physical cache word. We call these k bits per cache word the *Ordering ID* or *OID* of the physical cache word. We store the OIDs in the tag array where they are accessed at the same time as the tags. Similar to prior work [8], a copy of the OIDs is stored off-chip to allow the OIDs to be used across multiple switches to low voltage or after the chip is powered down.

[2]Coverages are determined as follows. Let $f(t)$ be the fraction of t-bit error patterns that are uncorrectable by an ECC scheme; we obtained $f(t)$ for each of the fast ECC implementations via Monte Carlo simulations. Given that there are n ways to reorder the logical bits stored in a physical word, a t-bit fault pattern is uncorrectable if the fault pattern expresses uncorrectable error patterns under all n bit orderings. The probability that a t-bit fault pattern is expressed as an uncorrectable error pattern under all n bit orderings is $f(t)^n$, assuming that the n orderings are generated independently from one another. The fraction of t-bit fault patterns that are correctable given n bit orderings is, therefore, $1 - f(t)^n$.

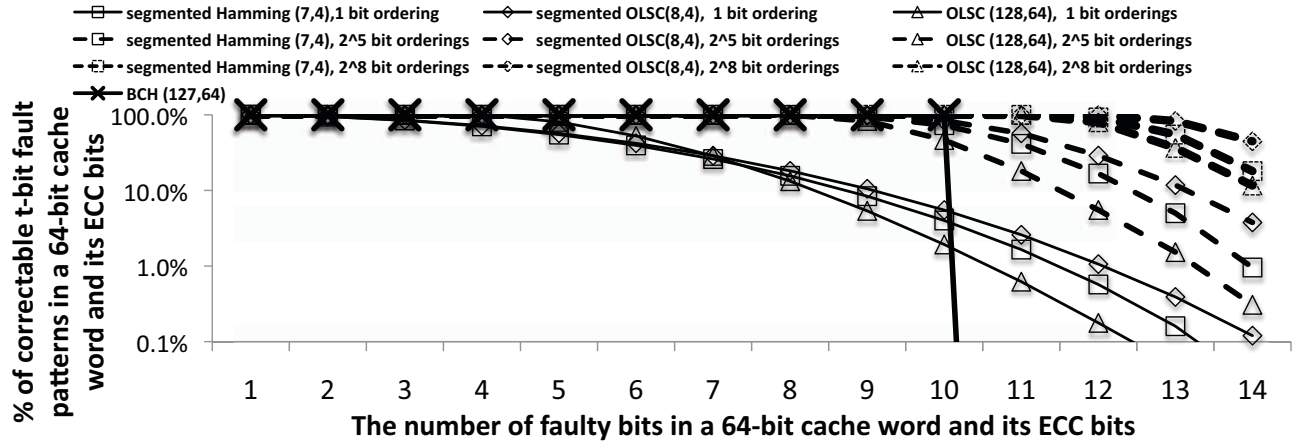


Fig. 7: **The physical fault coverages of low-latency ECCs with different numbers of bit orderings.** With multiple bit ordering options, the low-latency ECCs approach, and sometimes, exceed the slow BCH ECC in coverage.

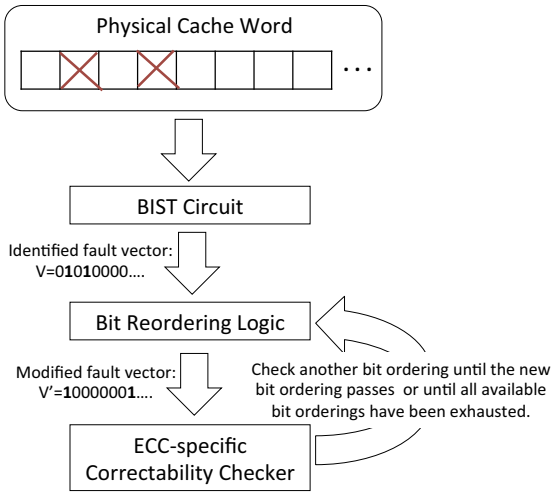


Fig. 8: **Selecting the appropriate bit ordering for a cache word.** A BIST circuit generates the fault pattern of a physical word. Logical bit reorderings are tried until one guarantees that the logical word will be correctable. In our evaluations, this takes less than two attempts on average.

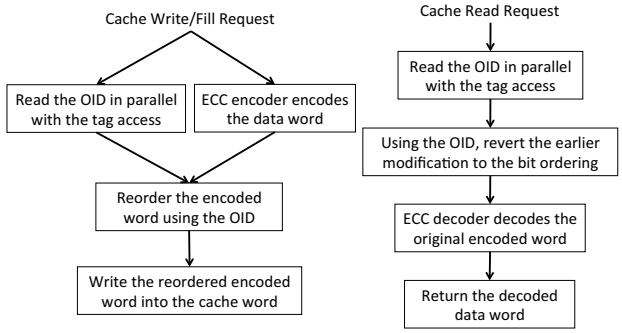


Fig. 9: **Cache access action flow.** Since bit reordering is on the critical path of a cache access, it is critical that it have low latency.

C. Bit Reordering and Order Recovery

Prior to a write to a physical cache word, we reorder the bits in the logical word to be written according to the recorded OID of the cache word. Conversely, after reading from a cache word, the earlier modification to the bit ordering has to be reversed to recover the original logical word. Fig. 9 details the steps needed for accessing a cache word. During a cache access, the OID is read at the same time as the tag; however, bit reordering and order recovery are on the critical paths of cache writes and reads, respectively. It is critical that these two steps are implemented at very low latency.

To provide low latency bit reordering and order recovery, we evenly divide each logical word into multiple small groups of bits and perform bit reordering and order recovery on all groups in parallel; bits in each parallel reordering group (or simply, *PRG*) can only be moved within the PRG but not across PRG. Intuitively, the smaller the PRG sizes, the higher the parallelism in the reordering and order recovery logic, and therefore, the lower the latency of the bit reordering and order recovery logic. However, the smaller the PRG sizes, the less flexible bit reordering becomes, which limits the correctable fault coverage because it reduces the number of correctable error patterns that can be exploited. As such, the size

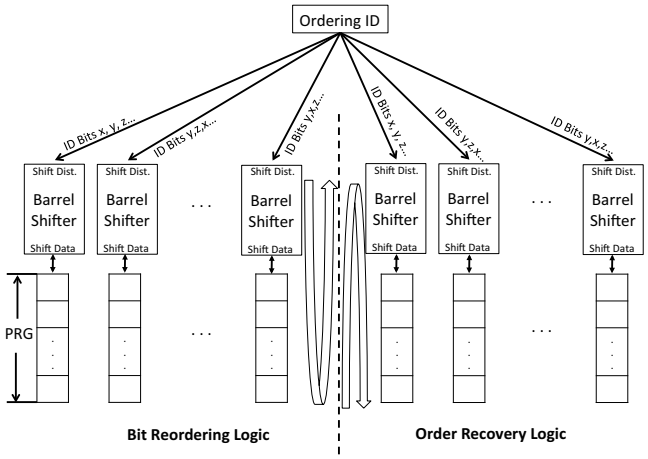


Fig. 10: **Bit reordering and order recovery logic.** PRG stands for a Parallel Reordering Group.

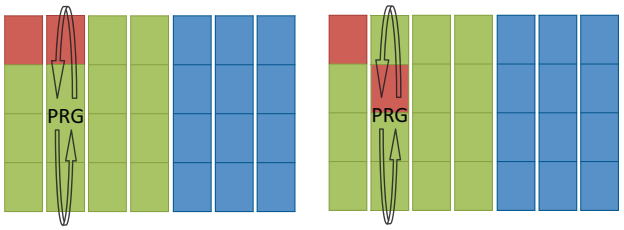
of the PRGs needs to be selected based on the underlying ECCs. Section III-D walks through some examples of PRG selection and sizing.

A straightforward way to reorder the bits in a PRG is to use a set of static circuits, each designed to reorder bits within a PRG in a specific way. However, this can lead to a large area overhead when supporting a large number of bit orderings per cache word. Instead, we use barrel shifters as a more scalable approach to reorder a PRG. A barrel shifter can generate a different bit ordering within a PRG for a different shift distance input value. In addition to providing good scalability, the barrel shifters can also reorder the bits in a PRG at low latency since the PRGs are small (*e.g.,* only 8 to 16 bits per PRG). Synthesis shows (details in Section IV-A) only < 0.3 cycle of barrel shifting latency assuming 20FO4s per cycle.

Fig. 10 illustrates the details of how to implement the bit reordering and order recovery logic using barrel shifters. The barrel shifters in the bit reordering logic and order recovery logic differ simply by the direction of rotation so that the actions taken by the former can be reversed by the latter. The shift distance inputs to the barrel shifters are taken from the OID of a cache word; this allows a logical word to be reordered/recovered according to the OID of the cache word that stores the logical word. Note that the number of bits in the shift distance input to a barrel shifter is a function of the size of each PRG; this number may or may not be equal to the size of an OID. As such, each barrel shifter takes as its shift distance input a combination of bits from the OID. Different barrel shifters take different combinations of the bits from an input OID to reduce the correlation between different PRGs since this correlation can reduce the effectiveness of bit reordering by reducing the movement of the bits in a logical word relative to each other. For our evaluation in Section V, we choose the combinations of input bits to the barrel shifters experimentally by picking the set of combinations that provides the best empirical fault coverage out of ten randomly generated sets of combinations.

D. Application Examples

Error pattern transformation (or EPT) is a general technique that can be applied on top of many prior works on improving on-chip memory reliability. In this section, we describe how to apply EPT to two different ECCs.



(a) A bit ordering that can generate uncorrectable errors for segmented Hamming(7,4).

(b) A bit ordering that only generates correctable errors for segmented Hamming(7,4).

Fig. 11: **PRGs for segmented Hamming(7,4).** Each row corresponds to one segment within the access granularity. Green and blue represent data and ECC bits, respectively. Red represent bits mapped to faulty SRAM cells. All rows are accessed at once, are decoded at once, and are combined to form one word.

1) Applying to Segmented ECCs: For segmented ECCs, we construct each PRG using logical bits from different segments to move errors in segments with more errors into segments with fewer errors; this PRG composition principle benefits all segmented ECCs. Fig. 11 shows the PRG composition for a 16-bit data word protected by the segmented Hamming(7,4) ECC; each column of squares represents a PRG while each row of squares represents a segment. Fig. 11 also illustrates how the composition of each PRG from bits in different segments improves the fault coverage of a segmented ECC.

2) Applying to Unsegmented ECCs: For unsegmented ECCs, a good PRG composition may be different for different unsegmented ECC implementations. As such, we propose a two-stage EPT that provides more freedom in bit reordering to better suit the specific needs of different unsegmented ECCs. During the first stage, each PRG is composed from bits in the same row so that the bits can be reordered along the X-axis. During the second stage, each PRG is composed from bits in the same column so the bits can be reordered along the Y-axis. Fig. 12 shows the PRG composition for a 16-bit data word protected by the OLSC(24,16) ECC.

Section VI provides more examples of application of EPT to prior works on improving on-chip memory reliability.

IV. Methodology

A. EPT-Based Cache Resilience Designs

To demonstrate the applicability of EPT to a variety of ECCs that can be used to protect deeply voltage scaled on-chip memories, we apply EPT to three different ECCs in the context of a 32KB L1 cache with a word size of 64 bits - the segmented Hamming(7,4), segmented OLSC(8,4), and OLSC(128,64). All three ECCs protect at the granularity of a 64-bit data word, which is the word size of the evaluated L1 cache. For sensitivity analysis, we evaluate EPT using both 5-bit OIDs and 8-bit OIDs; we refer to them as *5-bit EPT* and *8-bit EPT*, respectively. We apply EPT to segmented Hamming(7,4) and segmented OLSC(8,4) by adding one level of 16-bit barrel shifters and apply EPT to OLSC(128,64) by adding two levels of 16-bit barrel shifters.

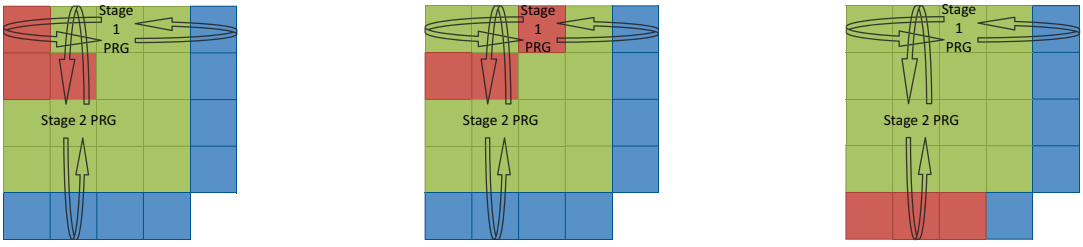
To accurately characterize the latency and area overheads of EPT, we implemented these ECCs, both by themselves and with EPT, in RTL and synthesize them using Synopsys Design Compiler [13] and the TSMC 65GP standard cell library. Cadence SoC Encounter [14] was used for physical layout. Area and latency values were verified using Encounter. Table I shows the latency in cycles and area overheads normalized to the evaluated 32KB cache (see Section IV-C); our evaluation assumes the 20FO4 cycle commonly used in prior works [4], [2]. Table I shows that all three ECCs stay within one cycle of latency, with or without EPT. By itself, each level of barrel shifters required by EPT only incurs < 0.3 cycle latency and 3% area overhead. In particular, the two segmented ECCs both stay within 0.5 cycle of latency, with or without EPT. The low-latency overheads justify our design choices in Section III. In our evaluations, we model the latency overhead of all of the ECCs in Table I as one cycle; the one exception is EPT:OLSC(128,64), which we conservatively model as two cycles of latency. Table II lists the EPT latency overheads that are used during our performance and EPT area overheads used during our power evaluations.

B. Baseline Cache Resilience Designs

To show that the proposed approach can lead to lower on-chip memory voltages than what has been previously possible, we compare against seven baselines. We compare against the SECDED (72,64) ECC commonly used in existing processors [15]. We compare against the primary implementation of MS-ECC [7], which uses the OLSC(128,64) ECC. We also compare against the circuit-level

	Cycle	Area	Gate Count
Segmented Hamming(7,4)	0.22	0.44%	528
EPT:segmented Hamming(7,4)	0.43	1.53%	2009
Segmented OLSC(8,4)	0.16	0.40%	573
EPT:Segmented OLSC(8,4)	0.46	1.60%	2095
OLSC(128,64)	0.68	8.89%	6394
EPT:OLSC(128,64)	0.98	10.10%	9187

TABLE I: **Synthesized decoder latency and area.** Area overheads is w.r.t a $0.76mm^2$ L1 cache.



(a) A bit ordering with uncorrectable errors.

(b) Bit ordering after stage 1 reordering.

(c) Bit ordering after 2D reordering.

Fig. 12: **PRGs for OLSC(24,16).** Green and blue represent data and ECC bits, respectively. Red grid represents bits mapped to faulty SRAM cells.

Baselines	Latency Overhead (Cycles)	Usable Cache Size	Decoder Area Overhead	Metadata Overhead	Total Area Overhead
5-bit EPT+Segmented Hamming(7,4)	1	16KB	3.06%	0%	3.1%
5-bit EPT+Segmented OLSC(8,4)	1	16KB	3.2%	0%	3.2%
5-bit EPT+OLSC(128,64)	2	16KB	20.20%	0%	24.2%
8-bit EPT+Segmented Hamming(7,4)	1	16KB	3.06%	4%	7.1%

TABLE II: **Evaluated EPT overheads.**

Baselines	Latency Overhead (Cycles)	Usable Cache Size	Decoder Area Overhead	Metadata Overhead	Total Area Overhead
Oracular BCH(127,64) (Hi-ECC)	2 - 21 (3.2 avg)	100%	70%	0%	70%
VS-ECC	1-7 (1.9 avg)	16KB	37%	4%	42%
MS-ECC	1	16KB	9.4%	0%	9.4%
SECDED	1	32KB	10.9%	0%	10.9%
8T SRAMs	0	32KB	0%	0%	33.3%
WD–Original	1	16KB	¡1%	43.2%	44.2%
WD–IsoArea	1	16KB	¡1%	4%	5%

TABLE III: **Evaluated baseline overheads.**

technique of using robustified 8T SRAM cells, instead of the usual 6T SRAM cells.

We compare against an Oracular BCH(127,64) ECC that guarantees correction of up to 10 bits of error per word. We consider this baseline oracular because we optimistically assume a BCH decoder that incurs the minimum latency for the exact number of errors currently present in a logical word. The oracular decoder has the low-latency benefit of weaker BCH decoders that correct fewer errors per word while having the high error coverage benefit of always guaranteeing correction of 10 bits in a word. As such, this oracular BCH baseline represents the best coverage and latency obtainable by prior works that propose protecting caches with the BCH ECC, such as Hi-ECC [16].

We compare against word disable (*WD*), the best resilience scheme presented in [4] for L1 caches. WD is previously evaluated in the context of caches with 512-bit access granularity; in this context, WD combines two 512-bit cache lines into an error-free logical line by dividing the two lines into 32 chunks and remapping data to fault-free chunks among the 32 available chunks. When evaluating WD in the context of our L1 cache with a 64-bit word size, we note that dividing two 64-bit words into 32 chunks can result in a high storage overhead for recording the 32 chunks. As such, we evaluate two versions of WD, WD–Original and WD–IsoArea; the former divides two 64-bit words into 32 chunks, while the latter divides the two 64-bit words into fewer chunks such that the implementation requires the same number of extra storage bits as an 8-bit OID EPT scheme.

We compare against a VS-ECC [8] implementation where each 64-bit data word is divided into four 16-bit segments. The 64-bits of ECC are shared as 6-bit ECC chunks between the four segments. Each segment can have up to four 6-bit ECC chunks allocated to it; each additional 6-bit chunk guarantees correction of up to an additional error. For this VS-ECC implementation, we assume that the four segments each have a dedicated decoder that corrects up to a single error but shares two 4-bit correcting decoders; the single-bit correcting decoders are sufficient for segments with a single error while the multi-bit decoders are needed in case multiple errors are present in a segment. We use two instead of four 4-bit correcting decoders to optimize the decoder area for VS-ECC since the segments more commonly experience single-bit errors than multi-bit errors in our evaluation.

We calculate the latency of SECDED according to [11], which conservatively estimates decoder latency in FO4s by ignoring the wire delay. We calculate the decoder area overhead of SECDED(72,64) by counting the number of gates required according to [11] and assuming that each gate is equal in area as two SRAM cells; the latter assumption is consistent with prior works [8]. When calculating the latency of Oracular BCH(127,64), we use the formula in [11] to calculate the latency required to correct *t*-bit errors and weight that calculate latency by the probability of encountering *t* bits of errors in a word. For the decoder area overhead of Oracular BCH(127,64), we assume it is the same as the area overhead of a constant latency BCH(127,64), which we calculate according to [11]. We extrapolate the latencies and area overheads of MS-ECC and WD from their respective papers. We follow the methodology for the parallel BCH decoder in VS-ECC to calculate the latency and decoder area overheads of our VS-ECC adaptation; we assume that the latency overhead of VS-ECC when a cache word is accessed is determined by the segment with the most errors among the four segments in the cache word, since this segment takes the longest to correct.

C. Experimental Setup

Keeping with the context of this work - energy efficient computing, we evaluate a two-issue in-order core similar to the ARM Cortex-A7 [15]; the detailed micro-architectural parameters are enumerated in Table IV. The baseline core operating at nominal voltage (1.2V) uses no ECC for L1 accesses.[3] For each design that requires correction, we deepen the pipeline to accommodate the minimum correction latency of that design. The core uses a common voltage rail for both its L1 caches and the rest of the core; as such, the L1 caches determine the minimum operating voltage of the entire core, in accordance with many prior works [5], [4], [7], [17]. We only voltage scale the core, but not the rest of the system, such as the memory system and lower-level caches, which we assume are on different voltage rails. To model voltage scaling, we scale frequency and leakage power with respect to voltage using the detailed 70nm scaling given in [18]; we use the common quadratic scaling to model dynamic power with respect to voltage. We model the SRAM failure probabilities using data reported in [2]. During low voltage

[3]Without any loss of generality, we can easily assume that some other default ECC is used at the nominal voltage

operations, we account for the added cache access latency due to error correction by stalling each load instruction by the number of cycles needed to perform error correction. We simulate seventeen benchmarks from the SPEC2000 [19] and SPEC2006 [20] benchmark suites[4] using Gem5 [21]; we fast-forward each benchmark for 1 billion instructions and execute in detailed timing mode for 1 billion instructions. We model processor power with McPAT [22] using simulation outputs from Gem5. We use Cacti [23] to model cache latency and power.

We choose a simple cache organization that is commonly used by prior works [4], [7], [17] to evaluate the relative merits of EPT and the various baselines. This simple cache organization stores the check bits of a data cache word (i.e., a cache word assigned to storing data during low voltage operation) in the same cache word in a different way in the same cache set, as these two cache words are typically always accessed in parallel in L1 caches both with or without support for voltage scaling. Unlike an alternative cache organization that uses one ECC way to protect multiple data ways, this simple cache organization that allocates a dedicated ECC way for each data way does not require expensive read-modify-write operations when updating the ECC cache word. We do not investigate new approaches to store ECC bits because it is orthogonal to the core idea of EPT. All the evaluated cache resilience designs have almost equal amount of check bits as data bits in each word, and thus all effectively utilize the available space in the dedicated ECC way for each data way. The only exception is the SECDED(72,64) ECC, which only provides $72-64=8$ bits of check bits per 64-bit data word. Instead of reserving half of the ways as ECC ways, we leave all four ways as data ways and expand the size of each cache word statically to store the check bits for SECDED(72,64).

Also similar to prior works [4], [17], the cache organization we evaluate uses the larger but more reliable 10T cells [2][5], as the tag array is only a small fraction of the total cache area. Similar to these prior works, we also use the more reliable tag array to store any metadata needed for the evaluated cache resilience schemes, such as the OID bits for EPT, the disable bits for WD, and the bits to identify how many ECC chunks are allocated to each segment for VS-ECC. Note that since half of the ways in the cache are used to store ECC bits during low voltage operation, half of the tags in the tag array are unused during low voltage operation. Therefore, we reuse the unused tag bits to store the metadata during low voltage operation. When the metadata bits do not completely fit in the unused tags (*i.e.,* more than 5-bit OIDS), we expand the size of the tag entries in the ECC ways to accommodate the remaining metadata bits. The area overhead due to expanding the size of the tag entries are given in Table III and Table II. We do not study OIDs smaller than 5-bits because they do not reduce area and latency overheads, but reduce coverage.

V. Results

In this section, we demonstrate that the proposed technique allows memory to run at lower voltages than was previously possible for a given yield target. We also discuss the core-wide power and energy benefits this entails compared to the different baselines.

A. Yield

We define cache yield as the fraction of caches where all of the data cache words in the data ways (i.e., the two data ways in our evaluated 4-way set-associative caches) are functional and error-free, similar to prior works [4], [7][6]. Fig. 13 shows the cache yield at low supply voltage for the different cache resilience designs. Fig. 13 shows that for a 99.9% yield target, which is commonly used in prior works [4], [7], 8-bit EPT + Segmented Hamming(7,4) reduces the latter's V_{min} by 134mV. For the same yield target, 8-bit EPT + Segmented Hamming(7,4) reduces the V_{min} of the two weakest baselines in terms of V_{min} (i.e., SECDED(72,64) and MS-ECC) by 70mV or more.

After 8-bit EPT + Segmented Hamming(7,4), the next closest resilience design in terms of achieving the lowest V_{min} is the 10-bit correcting Oracular BCH(127,64) ECC (the strongest version of a capacity-efficient code whose redundant bits fit in the disabled way). While 8-bit EPT + Segmented Hamming(7,4) achieves only 1mV lower V_{min} than Oracular BCH(127,64), the former's main advantage over the latter is having lower latency overhead; 8-bit EPT + Segmented Hamming(7,4) incurs a constant one cycle latency (see Table I), while the Oracular BCH(127,64) incurs 2 to 21 cycles of latency overhead per cache access (3.2 cycles, on average). A second benefit of 8-bit EPT + Segmented Hamming(7,4) over Oracular BCH(127,64) is having much lower decoder area. The area overhead of a BCH decoder increases rapidly with the word size and the number of errors to correct [11]. 8-bit EPT + Segmented Hamming(7,4) requires both minimal word size (i.e., only seven bits per segment) and minimal correction strength(i.e., only corrects one error per segment), as compared to BCH (127,64) which uses a word size that is roughly 16X as large and corrects 10X as many errors per word.

Following after Oracular BCH(127,64), the next closest baseline in V_{min} is VS-ECC. 8-bit EPT + Segmented Hamming(7,4) only reduces V_{min} by 13mV compared to VS-ECC. However, the main benefit of 8-bit EPT + Segmented Hamming(7,4) over VS-ECC is against for having lower latency overhead; VS-ECC incurs one to seven cycles of latency overhead per cache access (1.9 cycles, on average). 8-bit EPT + Segmented Hamming(7,4) also incurs much lower decoder area overheads, since the latter uses a word size 8X as large and corrects 4X as many errors per word.

After VS-ECC, the next closest baseline in V_{min} is WD–Original. 8-bit EPT + Segmented Hamming(7,4) reduces the V_{min} of WD–Original by 33mV. However, WD–Original requires 4X as many metadata bits as 8-bit EPT, which incurs a 43% overhead in total cache area. On the other hand, the alternative WD–IsoArea implementation we evaluate, which requires the same number of metadata bits as 8-bit EPT, only provides a V_{min} of 769mV, which is nearly

Core Type		In-order, 7-Stage
Register File		32 Int, 32 FP
Fetch/Decode/Issue Width		2/2/2
BTB Size		4096 entries
RAS Size		16 entries
Branch Predictor		Tournament
ALUs/FPUs/MDUs		2/1/1
Cache Line Size		64B
L1 I$	Nominal (1.2V)	32KB, 4-way, 2-cycle
	V_{min}	16KB, 2-way, 2-cycle
L1 D$	Nominal (1.2V)	32KB, 4-way, 2-cycle
	V_{min}	16KB, 2-way, 2-cycle
L2 Unified $		1MB, 8-way, 20ns latency
Memory Configuration		1066 MHz DDR3

TABLE IV: **Microarchitectural parameters.**

[4]We evaluated ammp, applu, apsi, art470, facerec, gromacs, lbm, libquantum, lucas, mcf_2006, mesa, mgrid, milc, omnetpp, soplex, swim, and wupwise.

[5]Our reliability calculations also account for the failure probability of the 10T cells.

[6]Cache word error rates are calculated analytically where possible (*e.g.,* Oracular BCH(127,64)). For the rest of the designs, in particular, EPT, word error rates are determined through Monte Carlo simulations.

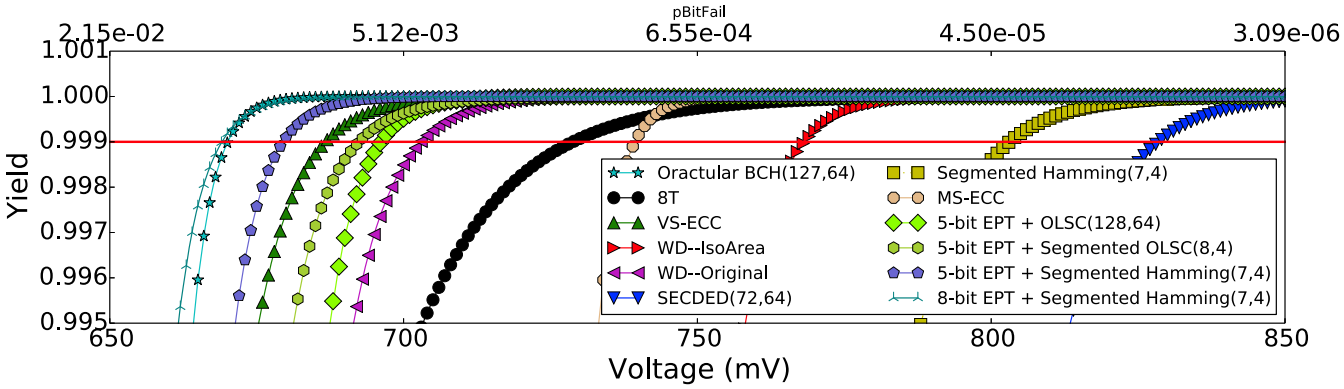


Fig. 13: **Yield for 32KB 4-way cache implemented in 6T SRAM cells.** Cache yield vs voltage for the various cache resilience schemes. The red line depicts the 99.9% yield.

100mV higher than the best EPT solution.

The next closest baseline in V_{min} are caches built using 8T SRAMs. 8-bit EPT + Segmented Hamming(7,4) reduces V_{min} by 38mV and 59mV, respectively. However, 8T incurs high area (and thus power) overheads of 33%. Fig. 13 shows that, although 8T SRAM cells provide a reduction in V_{min}, EPT solutions provide lower V_{min} (*e.g.,* EPT(8-bit OID) applied to segmented Hamming(7,4) provides a reduction of 59mV over 8T). However, 8T incur high area overheads (i.e., 33% [2], respectively).

For sensitivity analysis, we also evaluated three 5-bit EPT design points. Among these three designs, 5-bit EPT + Segmented Hamming(7,4) provides the highest yield; it provides higher yield than 5-bit EPT + Segmented OLSC(8,4) because the former requires fewer check bits than the latter while correcting the same number of errors per segment as the latter. Having a fewer number of bits per word reduces the average number of faulty bits per word and, thereby, improving yield. On the other hand, 5-bit EPT with Segmented OLSC(8,4) provides higher yield than 5-bit EPT + OLSC(128,64); this is because the fraction of t-bit error patterns that are correctable by OLSC(128,64) decreases more rapidly with respect to t than the fraction of t-bit error patterns that are correctable by Segmented OLSC(8,4), as shown in Fig. 5. When the fraction of t-bit error patterns that are correctable is small, it becomes less likely to find a bit ordering that always generates one of the correctable t-bit error patterns in a logical word given a t-bit fault pattern in a cache word.

All of the above results hold irrespective of the core microarchitecture (*e.g.,* in-order vs out-of-order, issue-width, etc.) and workloads. The next section describes how the improved yield due to EPT translations into improved power and performance characteristics for the chosen core microarchitecture and workloads.

B. Power and Performance

Table V summarizes the core-wide power and performance of the EPT designs and the baselines. While 8-bit EPT + Segmented Hamming(7,4) operates at the same V_{min} and, therefore, similar power as Oracular BCH(127,64), the former provides 21% higher IPC than the latter. This performance improvement is due to having much lower latency overhead than Oracular BCH(127,64) ECC. This shows that EPT is better than imply using a stronger ECC due to decoder latency and area overheads. Compared to 8-bit EPT + Segmented Hamming(7,4), VS-ECC only increases V_{min} by 13mV. However, the higher decoder latency and area overhead of VS-ECC results in a high EPI overhead. 8-bit EPT + Segmented Hamming(7,4) reduces EPI by 40.2% compared to VS-ECC.

WD-Original and WD-IsoArea provide similar IPC as EPT since WD also only incurs a constant one correction cycle latency. However, the high area overhead of WD-Original (i.e., 44.2%) results in a high power overhead; 8-bit EPT + Segmented Hamming(7,4) reduces power by 20.5% compared to WD-Original. Although WD-IsoArea has similar area overhead as 8-bit EPT + Segmented Hamming(7,4), it has a much higher V_{min}; 8-bit EPT + Segmented Hamming(7,4) reduces power by 31.1% over WD-IsoArea. MS-ECC also incurs similar latency and area overheads as EPT. However, the V_{min} of MS-ECC is 70mV higher than that of 8-bit EPT + Segmented Hamming(72,64). As a result, 8-bit EPT + Segmented Hamming(72,64) reduces core-wide power by 25.7% compared to MS-ECC.

Using robustified SRAM cells can provide significant V_{min} reduction without incurring any error correction latency overhead since they do not require error correction; however, robustified SRAM cells incur significant area. Due to the high area overhead as well as the higher V_{min} of 8T cells, 8-bit EPT + Segmented Hamming(7,4) reduces overall core power by 28.1% compared to using 8T SRAM cells. Due to this power overhead, 8-bit EPT + Segmented Hamming(7,4) has a 19.1% lower EPI than 8T cells. Note that unlike architectural techniques where most of the circuits (*e.g.,* decoders) needed for low voltage operations can be power gated during nominal voltage operations, caches with robustified SRAM cells continue to incur high overheads at nominal operation. For example, caches with the 8T SRAM cells incur a 12.0% core-wide EPI overhead at nominal voltage operation, whereas 8-bit EPT + Segmented Hamming(7,4) incurs only a 7.8% EPI overhead at nominal voltage (due to the overheads of the added correction pipeline stages).

The above results show that EPT is very effective at reducing the V_{min} and, therefore, power of low latency caches by increasing the coverage of an ECC-based error resilience scheme. The results also show that EPT does not have some of the limitations that hamper the efficacy of other comparable techniques (*e.g.,* high power and area overheads during nominal and low voltage operation). This makes EPT a promising technique to allow deeply voltage scaled processors and on-chip memories.

C. Additional Sensitivity Analysis

1) BIST-Undetectable Faults: EPT targets faulty SRAM cells that can be identified during BIST. However, some SRAM faults, such as soft errors, erratic faults, or aging related faults, cannot be detected by BIST. EPT does not protect against these faults because they cause errors to occur randomly in different locations. For L1 caches, existing processors typically only detect these errors using parity or

	Vmin (mV)	Freq (GHz)	IPC	Power (%)	EPI (%)
Oracular BCH(127,64)	671	0.931	1.035	24.3	50.5
VS-ECC	683	0.960	0.854	22.4	54.8
8T SRAMs	729	1.061	1.236	26.6	40.5
WD–IsoArea	769	1.141	1.171	27.7	41.5
WD–Original	703	1.006	1.228	24.0	38.9
SECDED	823	1.256	0.879	34.5	62.6
Segmented Hamming(7,4)	804	1.210	1.147	31.5	45.4
MS-ECC	740	1.083	1.191	25.7	39.8
EPT-Based Designs					
5-bit EPT+ OLSC(128,64)	697	0.993	0.918	22.0	48.3
5-bit EPT+ Seg OLSC(8,4)	693	0.984	1.229	20.7	34.3
5-bit EPT+ Seg Ham(7,4)	680	0.953	1.242	19.7	33.2
8-bit EPT+ Seg Ham(7,4)+	670	0.929	1.252	19.1	32.8

TABLE V: **Performance, power, and energy normalized to nominal execution of the baseline design.**

protect against these errors using SECDED to guarantee correction of a single random error per word [15]. One can also guarantee correction of a single random error per word in caches with EPT by adding an independent layer of random error correcting code.

As an example, when storing a 64-bit dataword, besides protecting it with EPT + Segmented Hamming(7,4), one can also protect it using an independently calculated single-symbol-correcting (SSC) Reed-Solomon (RS) ECC with 5-bit symbols; each adjacent four data bits in the 64-bit dataword is mapped to a symbol in the SSC ECC when calculating the SSC ECC. When a random error occurs in a cache word, EPT + Segmented Hamming(7,4) may not be able to correct the segment in which the error resides. However, since only a single segment and, therefore, a single symbol contains the random error, the erroneous segment is guaranteed to be correctable via the SSC ECC if the random error is the only random error in the cache word. The overhead required by this additional layer random error correction is small. The RS ECC requires 10 additional ECC bits[7], which fit within the $64-(7-4)\cdot 16=16$ unused bits per ECC way of Segmented Hamming(7,4). The area overhead of the RS ECC encoder [24] and a ROM-based decoder, which stores the correctable syndromes and the error pattern corresponding to each correctable syndrome, is only 1.8% of the L1 cache area. Our evaluation shows that including this additional layer of SSC ECC to 8-bit EPT + Segmented Hamming(7,4) only increases the latter's core-wide power by 6.8%. Since random errors only need to incur any correction latency when an random error occurs [15], which is rare, the energy overhead of adding the SSC ECC is minimal (i.e., 4.4%). In comparison to VS-ECC, the best prior scheme that can provide built-in soft error protection, EPT + Segmented Hamming(7,4) provides a 40% energy reduction and a 25% total cache area reduction. Despite incurring a 4.4% energy overhead and a 2% cache area overhead from SSC ECC, EPT's relative benefits remain high.

Note that in our evaluation in Sections V-A and V-B, we dedicate all the error correction resources of all evaluated resilience schemes to protecting against BIST-detectable faults; in other words, additional overheads will be required for all the evaluated baselines to guarantee correction of a single random error per word while maintaining the same strength of protection against BIST-detectable faults that the baselines enjoy in Sections V-A and V-B. As such, although we do not evaluate adding random error protection for the baselines, we expect the merits of EPT relative to the various baselines to remain roughly the same as those in Sections V-A and V-B even in scenarios where protection against random errors are needed.

[7]For simplicity, we protect against known bit faults among these 10 bits per cache word by only using EPT to move the faulty bits away from them but not storing Hamming(7,4) ECC bits for them.

	Vmin (mV)	Freq (GHz)	IPC	Power (%)	EPI (%)
Oracular BCH(767,512)	701	1.002	0.956	28.0	58.5
MS-ECC	758	1.119	0.996	29.0	52.0
8-bit EPT+ Seg OLSC(128,64)	689	0.974	0.983	24.0	50.2
One ECC way for seven data ways					
VS-ECC	754	1.111	0.967	30.8	57.3
8-bit EPT: Seg BCH(80,64)	742	1.087	0.930	29.0	57.4

TABLE VI: **L2 6T performance, power, and energy normalized to nominal execution.**

2) L2 Caches: While EPT is primarily a technique to allow deeply voltage scaled on-chip memories, it can also be viewed as a technique to reduce the latency of strong error correction since it allows low-latency ECCs to attain the coverage of high-latency (Sections II and III). While latency is less of a concern for L2 caches, we demonstrate that EPT can still provide some benefits. Table VI shows the performance, power, and energy evaluations for several designs where the L2 is co-scaled with L1s and the core. For these designs, the L2 determines the V_{min}. The values in the table include the power consumed by the L2. BCH, MS-ECC, and EPT + Segmented OLSC(128,64), continue to use an ECC way for each data way, which is consistent with our evaluation of the L1 cache. Compared against MS-ECC, 8-bit EPT + Segmented OLSC(128,64) reduces combined core and L2 power by 17%.

Since the capacity overhead requirement may be more stringent at the L2 level, we also demonstrate the benefits of EPT while disabling only one of 8 ways. For this capacity overhead requirement, we compare EPT against a VS-ECC solution which has the same number of redundant bits as 8-bit EPT. This VS-ECC solution breaks each cache line into four words each of which can correct up to four faults. Across these four words, up to eight total faults may be corrected, requiring 72 bits of redundancy. We do not allocate any correction resources to BIST-undetectable faults. Results show that the energy characteristics of EPT and VS-ECC are comparable. However, EPT may still be preferable since VS-ECC decoders have high area overhead–10% of the L2 cache area. High decoder area overhead comes from implementing four 4-bit correcting BCH decoders in a fully combinational manner (*i.e.,* lowest latency BCH decoder). 8-bit EPT + Segmented BCH(80,64), on the other hand, only incurs a 3.5% total area overhead.

3) 28nm/32nm Technology Node: The performance, power, and energy benefits provided by fault tolerant mechanisms for low-V_{min} caches are strongly tied to the characteristics of a particular technology process (*i.e.,* how fault rates, frequency, and power scale with voltage). In this section, we explore the benefits for a 28nm fault rate [12][8] and 32nm frequency and power rates [6]. 8-bit EPT + Segmented Hamming(7,4) has 17.7% lower power and 34.5% lower energy than VS-ECC (the next lowest power correction technique). 8-bit EPT + Segmented Hamming(7.4) has 23.9% lower power and 11.8% lower energy than MS-ECC (the next lowest energy correction

[8]We could not find any 28nm or 32nm 8T SRAM fault rates.

	Vmin (mV)	Freq (GHz)	IPC	Power (%)	EPI (%)
Oracular BCH(127,64)	674	0.326	0.977	15.5	44.6
VS-ECC	688	0.344	0.896	14.7	43.5
WD–IsoArea	738	0.415	1.157	17.3	32.9
WD–Original	701	0.361	1.185	15.5	33.0
SECDED	777	0.469	0.851	21.3	48.8
MS-ECC	719	0.388	1.165	15.9	32.3
EPT-Based Designs					
5-bit EPT+ OLSC(128,64)	696	0.355	0.873	14.2	41.8
5-bit EPT+ Seg OLSC(8,4)	693	0.351	1.186	13.3	29.3
5-bit EPT+ Seg Ham(7,4)	682	0.336	1.186	12.6	28.9
8-bit EPT+ Seg Ham(7,4)	673	0.324	1.202	12.1	28.5

TABLE VII: **28nm/32nm performance, power, and energy at V_{min} normalized to those at nominal voltage (1.2V).**

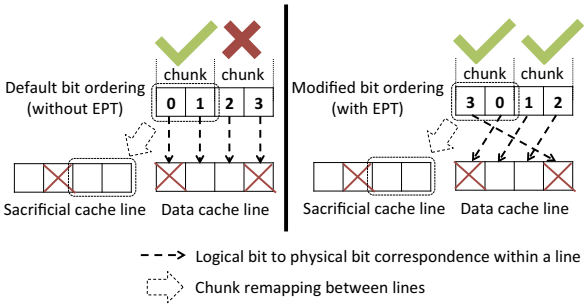


Fig. 14: **Applying EPT to Archipelago [25].** Left: two chunks need to be remapped while the sacrificial line only has room for one chunk. Right: Only one chunk needs to be remapped after bit reordering.

technique). These 28nm/32nm results demonstrate that EPT can provide voltage scaling benefits across technology nodes.

VI. Related Work

This section describes several additional related works. Parichute [9] explores how to protect low-level caches operating at low voltages using Turbo codes. By themselves, Turbo codes under-utilize cache capacity because they do not increment in size in powers of two. To better utilize cache capacity, Parichute fills up the ECC ways by constructing additional check bits to the original Turbo codes and, thereby, improves the strength of the original Turbo codes as well. However, the latency overheads of Turbo codes are still significant (e.g., >4 cycles, on average [9]). In addition, Turbo codes can correct a different number of errors depending on the error pattern in the word. As such, EPT can be applied on top of Parichute and is, therefore, orthogonal to Parichute.

Archipelago [25] avoids accessing faulty bits by remapping groups of logical bits (called *chunks*) from one physical cache line to a second cache line, called a *sacrificial line*. However, remapping chunks across lines require significant design complexity. For each remapped chunk, Archipelago records the original location of the chuck, its corresponding sacrificial line, and the corresponding position within the sacrificial line; this requires adding two additional address translation tables to the critical path of cache access [25]. In addition, a complete cache access now has to wait for two cache line accesses to complete. Since the sacrificial cache line is often not in the same cache set as the data line, to allow both cache line accesses to complete around the same time, the number of banks in the cache has to be doubled from what is needed to meet the fetch and issue width [25]. This not only increases cache area and latency, but also requires modifying the access scheduling for the different cache banks to improve synchronization of accesses between different banks. Cache access scheduling is further complicated because sacrificial line store chunks from different data cache lines. Therefore, on a write to a data cache line, Archipelago requires updating the appropriate chunks within the corresponding sacrificial line using an expensive read-modify-write operation. EPT only reorders bits within a word, not across words, and, therefore, avoids all of the above complexities. Finally, EPT can also be applied on top of Archipelago to improve its effectiveness by aggregating faults in unused chunks, as illustrated in Fig. 14. In this way, EPT is orthogonal to Archipelago.

IPatch [26] protects caches operating at low voltages by reusing unused spaces in other memory structures, such as the store queue, micro-op cache, MSHR buffers, etc., to store redundant copies of words in the cache. IPatch requires modifying a large number of processor components to make them aware of faulty cache words; this may significantly complicate processor design and verification. In addition, these structures can only protect a small fraction of the cache due to the smaller sizes of these memory structures relative to a cache; as such, IPatch disables unprotected faulty cache words in data lines [26]. Disabling a cache word in a data line leads to uncachable logical words, which can significantly impact performance. EPT, on the other hand, restricts the required modifications to within a cache and does not require disabling cache words in data lines. In addition, EPT can also be applied to a cache along with IPatch and is, therefore, orthogonal to IPatch.

EPT bears some resemblance to bit-interleaving, which statically interleaves adjacent physical bits across different segments of a segmented ECC. For *physically adjacent faults*, bit-interleaving converts multiple *adjacent* bits of errors in one segment into single-bit errors in different segments. Causes of physically adjacent bit faults include large alpha particle strikes [27] and complete DRAM chip failures [28]. Faulty SRAM cells at low voltages, however, are randomly distributed across a cache[4], not typically physically adjacent to one another. As such, bit-interleaving does not improve the coverage of faulty SRAM cells during low voltage operation, which EPT does by adaptively modifying the bit ordering in each cache word according to the identified fault pattern of the cache word.

EPT also bears some resemblance to *fault dispersion*, which seeks to transfer faults from a line with too many faults to lines with fewer faults, in off-chip main memories [29], [30], [31], [32], [33], [34], [35]. In off-chip main memories, a line is typically striped across multiple memory chips/cards called a *rank*, such that all chips/cards in a rank receive the same memory address input and operate in lockstep to satisfy a single memory request. Exploiting this architecture, prior works perform fault dispersion by physically swapping the memory chips/cards or by reconfiguring the memory chips/cards in a rank to access different intra-chip/card locations for the same address presented to all the chips/cards in the rank. As such, these prior works differ vastly in implementation from EPT, which targets on-chip memories. Fault dispersion in the context of L1 caches equates to transferring logical bits between words in different sets, which requires accessing multiple cache words per access to a faulty cache word; this can incur high latency overheads for low voltage SRAM caches, where latencies are low and fault rates are high, unlike off-chip main memories, where latencies are high and the fault rates are low. Finally, unlike fault dispersion, EPT does not reduce the number of errors in a cache word, but simply transforms the error pattern generated by the cache word. As such, EPT can also be applied on

top of fault dispersion, and is, therefore, also orthogonal to fault dispersion.

VII. Conclusion

In this paper, we presented error pattern transformation, a general technique for low cost error correction which enables memory voltages to be scaled further than prior works on error correction. We observed that although many ECCs only guarantee correction of a small number of errors, they can actually correct a large number of erroneous bits if these bits are in the particular error patterns. Since the same physical fault pattern in a cache word can manifest as different error patterns depending on the ordering of the logical bits stored in a physical cache word, EPT adaptively rearranges the logical bit to physical bit mapping per word according to the known BIST-detectable fault pattern in the physical word. The adaptive logical bit to physical bit mapping transforms many uncorrectable error patterns in the logical words into correctable error patterns and, therefore, improves ECC error coverage and reduces the minimum required voltage of operation. This reduces the minimum voltage at which memory can run by 70mV over the best low-latency ECC baseline leading to a 25.7% core-wide power reduction for an ARM Cortex-A7-like core. Energy per instruction is reduced by 15.7% compared to the best baseline.

Acknowledgment

This work was partially supported by NSF, Cisco, and CFAR, within STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] J. Kulkarni, K. Kim, and K. Roy, "A 160 mv robust schmitt trigger based subthreshold sram," *Solid-State Circuits, IEEE Journal of*, vol. 42, pp. 2303–2313, Oct 2007.

[2] J. Kulkarni and K. Roy, "Ultralow-voltage process-variation-tolerant schmitt-trigger-based sram design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, pp. 319–332, Feb 2012.

[3] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[4] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, (Washington, DC, USA), pp. 203–214, IEEE Computer Society, 2008.

[5] S. M. Khan, A. R. Alameldeen, C. Wilkerson, J. Kulkarni, and D. A. Jimenez, "Improving multi-core performance using mixed-cell cache architecture," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, (Washington, DC, USA), pp. 119–130, IEEE Computer Society, 2013.

[6] S. Vangal and S. Jain, "Claremont: A solar-powered near-threshold voltage ia-32 processor," in *Design Technologies for Green and Sustainable Computing Systems* (P. P. Pande, A. Ganguly, and K. Chakrabarty, eds.), pp. 229–239, Springer New York, 2013.

[7] Z. Chishti, A. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 89–99, 2009.

[8] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, (New York, NY, USA), pp. 461–472, ACM, 2011.

[9] T. N. Miller, R. Thomas, J. Dinan, B. Adcock, and R. Teodorescu, "Parichute: Generalized turbocode-based error correction for near-threshold caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 351–362, IEEE Computer Society, 2010.

[10] M. K. Qureshi and Z. Chishti, "Operating secded-based caches at ultra-low voltage with flair," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pp. 1–11, 2013.

[11] D. Strukov, "The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, pp. 1183–1187, 2006.

[12] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, "Sram for error-tolerant applications with dynamic energy-quality management in 28 nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 50, pp. 1310–1323, May 2015.

[13] Synopsys, *Synopsys Design Compiler User's Manual*.

[14] Candence, *Cadence SoC Encounter User's Manual*.

[15] ARM, "Cortex-a7 technical reference manual, rev r0p5."

[16] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," *SIGARCH Comput. Archit. News*, vol. 38, pp. 83–93, June 2010.

[17] H. Duwe, X. Jian, and R. Kumar, "Correction prediction: Reducing error correction latency for on-chip memories," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 463–475, Feb 2015.

[18] M. Hempstead, G. yeon Wei, and D. Brooks, "Architecture and circuit techniques for low-throughput, energy-constrained systems across technology generations," in *Proceedings of CASES*, pp. 368–378, ACM Press, 2006.

[19] Standard Performance Evaluation Corporation, "Spec cpu2000."

[20] Standard Performance Evaluation Corporation, "Spec cpu2006."

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 469–480, ACM, 2009.

[23] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, ""cacti 5"," 2008.

[24] S. Morioka and Y. Katayama, "Design methodology for a one-shot reed-solomon encoder and decoder," in *Computer Design, 1999. (ICCD '99) International Conference on*, pp. 60–67, 1999.

[25] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation.," in *HPCA*, pp. 539–550, IEEE Computer Society, 2011.

[26] D. Palframan, N. S. Kim, and M. Lipasti, "ipatch: Intelligent fault patching to improve energy efficiency," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 428–438, Feb 2015.

[27] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pp. 197–209, Dec 2007.

[28] Intel, "Intel e7500 chipset mch intel x4 single device data correction (x4 sddc) implementation and validation," Aug 2002.

[29] G. Bond, "Fault alignment control system and circuits," Dec. 18 1984. US Patent 4,489,403.

[30] G. Bond, F. Cartman, and P. Ryan, "Multi-bit error scattering arrangement to provide fault tolerant semiconductor static memories," Dec. 11 1984. US Patent 4,488,298.

[31] D. Bossen and M. Hsiao, "Deterministic permutation algorithm," July 17 1984. US Patent 4,461,001.

[32] W. Beausoleil, "Method of manufacturing a full capacity monolithic memory utilizing defective storage cells," Aug. 5 1975. US Patent 3,897,626.

[33] H. M. Bossen D, Haugh C, "Dynamic address translation scheme using orthogonal squares," May 21 1974. US Patent 3,812,336.

[34] W. Beausoleil, "Monolithic memory utilizing defective storage cells," Dec. 25 1973. US Patent 3,781,826.

[35] B. W. F, "Memory with reconfiguration to avoid uncorrectable errors," Feb. 22 1972. US Patent 3,644,902.