# Vulnerability-Based Interleaving for Multi-Bit Upset (MBU) Protection in Modern Microprocessors

Michail Maniatakos
EE Department
Yale University
michail.maniatakos@yale.edu

Maria K. Michael
ECE Department
University of Cyprus
mmichael@ucy.ac.cy

Yiorgos Makris
EE Department
University of Texas at Dallas
yiorgos.makris@utdallas.edu

***Abstract*—We present a novel methodology for protecting in-core microprocessor memory arrays from Multiple Bit Upsets (MBUs). Recent radiation studies in modern SRAMs demonstrate that up to 55% of Single Event Upsets (SEUs) due to alpha particle or neutron strikes result in MBUs. Towards suppressing these MBUs, methods such as physical interleaving or periodic scrubbing have been successfully applied to caches. However, these methods are not applicable to in-core, high-performance Content-Addressable Memories (CAM) arrays, due to computational complexity, high delay and area overhead, and lack of information redundancy. To this end, we propose a cost-effective method for enhancing in-core memory array resiliency, called Vulnerability-based Interleaving (VBI). VBI physically disperses bit-lines based on their vulnerability factor and applies selective parity to these lines. Thereby, VBI aims to ensure that an MBU will affect at most one critical bit-field, so that the selective parity will detect the error and a subsequent pipeline flush will remove its effects. Experimental results employing simulation of realistic MBU fault models on the instruction queue of the Alpha 21264 microprocessor in a 65nm process, demonstrate that a 30% selective parity protection of VBI-arranged bit-lines reduces vulnerability by 94%.**

## I. Introduction

In order to support out-of-order, superscalar execution, modern microprocessors employ several in-core memory arrays, such as branch predictors, register allocation tables (RAT), instruction queues, reorder buffers, etc. One of the most common ways to implement such high-performance storage elements, is the use of Content Addressable Memory (CAM) and Random Access Memory (RAM) structures [1]. Indeed, the latest microprocessors incorporate several CAM/RAM-based structures [2] since they achieve power savings of 36%, on average, as compared to latch-based memory structures [3]. These structures are built using Static RAM (SRAM) technology, with a typical CAM cell consisting of two SRAM cells [4]. Hence, similarly to SRAMs, in-core memory arrays become vulnerable to single- and multi- bit errors in modern technology nodes.

Single event upsets (SEUs) due to alpha particle or neutron strikes [5] have been extensively studied over the last decade and various countermeasures have been developed to address the concomitant transient errors [6]. However, as process feature sizes continue to shrink, it has become more likely that adjacent cells may also be affected by a single event [7], thereby causing a multiple-bit upset (MBU). An MBU is defined as an event that causes more than one bit to be upset during a single measurement [8]. During an MBU, multiple bit errors in a single word, as well as single bit errors in multiple adjacent words may be introduced [9]. While, in the past, MBUs were only encountered in space applications [7], advanced memory structures now exhibit a dramatically increasing multi-bit failure rate [10], [11], [12], [13], [14], highlighting the need for incorporating MBUs in accurate vulnerability analyses. Specifically, in [10], experiments show that only 45% of the upsets affect only 1 bit; the rest may affect a greater number of bits, as shown in Figure 1. This failure rate is further accelerated by reduced power supply voltage, increased clock frequency, crosstalk and electromigration effects [15]. Importantly, [12] shows that MBU rates increase as technology shrinks from 180nm to 65nm. Therefore, mechanisms for protecting in-core memory arrays from both SEUs and MBUs become essential.

Advanced Error Correcting Codes (ECC), such as Double Error Correction - Triple Error Detection (DEC-TED) [16] or Error-Locality-Aware Coding [17], can be very effective in terms of MBU protection in SRAMs. However, checkword generation incurs significant area overhead and, more importantly, its computational complexity prohibits the use of ECC for protecting the extremely fast CAM/RAM arrays, which operate at clock speed. Thus, ECC is limited to the protection of the $> 10x$ slower caches and main memory. Physical interleaving [18], [19], as shown in Fig. 2, is the most com-

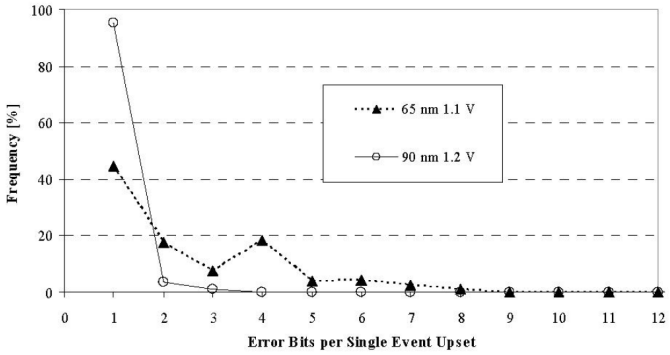



Fig. 1. Frequency distribution of number of faulty bits generated per SEU for different process sizes [10]


978-1-4673-1595-1/12/$31.00 

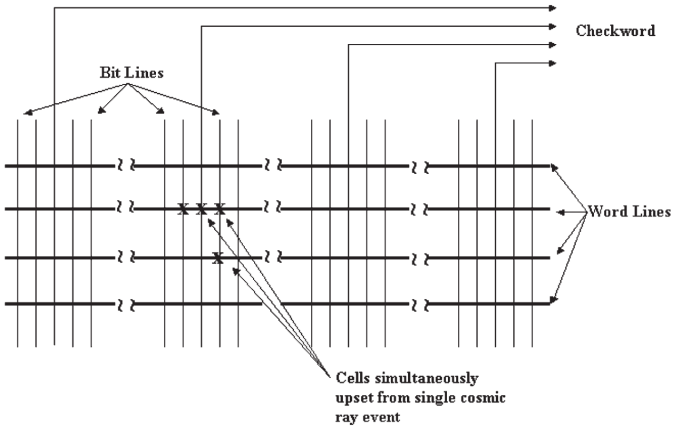



Fig. 2. Physical interleaving [18]

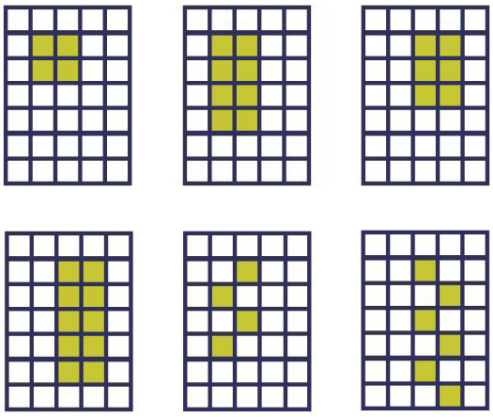


Fig. 3. Typically observed fail bit patterns [10]

monly used technique to protect SRAM cells against MBUs. Interleaving encompasses the creation of logical checkwords from physically dispersed locations in the memory array, forcing MBUs to appear as multiple single-bit errors, instead of a single multi-bit error. However, as physical interleaving implies the use of powerful ECC codes, it can also not be used to protect in-core CAM/RAM arrays. Finally, in [20], the authors suggest periodic scrubbing in order to reduce the multi-bit error rate in caches. However, as scrubbing reuses information that is replicated in higher-levels of memory, it can only be applied to cache hierarchies and not to in-core memory arrays.

In this paper, we present a novel method to increase the resiliency of in-core microprocessor arrays against MBUs, called Vulnerability-based Interleaving (VBI). VBI leverages the different vulnerability factors of individual bits in order to minimize the impact of MBUs. Critical bit columns are placed physically apart, and low-cost parity is used to protect against errors. Note that error detection is sufficient for error-free operation in this case, since we can leverage the speculative execution mechanisms inherent in modern microprocessors and flush the pipeline to prevent architectural state corruption. Evidently, to support VBI, a method for assessing vulnerability of individual bits in in-core microprocessor memory arrays in the presence of MBUs is required. Such vulnerability assessment is discussed in Section II, followed by the presentation of the VBI algorithm for generating MBU-hardened memory arrays in Section III. Our experimental infrastructure for evaluating the effectiveness of VBI is described in Section IV and results are presented in Section V, followed by conclusions and future directions in Section VI.

## II. Vulnerability Assessment

The vulnerability of a microprocessor, expressed as the Soft Error Rate (SER), is defined as the product of the raw Failures In Time (FIT) rate and the probability that a fault results into a visible user error. The FIT rate can be calculated through sophisticated models, usually as a function of elevation, technology node [21], supply voltage, etc. Typical rate numbers vary between 0.001 - 0.01 FIT/bit [5]. However, calculating the probability that a fault results in a visible user error is not trivial and researchers have followed various approaches to provide accurate estimates.

In [22], Architecture Vulnerability Factor (AVF) is defined as the probability of a bit-flip in a microprocessor structure leading to a user visible system error. The authors calculate AVF by tracking the subset of the microprocessor state bits required for architecturally correct execution (ACE). Wang et. al [23] investigated the accuracy of ACE analysis through extensive Statistical Fault Injection (SFI) experiments. Their results corroborated that ACE analysis overestimates microprocessor vulnerability, mostly due to less detailed structures available in the performance model employed in [22]. Furthermore, recent studies with proton and neutron irradiation tests showed that SFI measurements closely match in-field exposure [24]. Finally, [25] introduced a method called Global Signal Vulnerability (GSV) analysis to approximate AVF using single stuck-at fault simulations. GSV provides the same relative ranking of structures, in terms of their vulnerability, as AVF does, yet in much shorter time.

All of the above state-of-the-art methodologies for estimating microprocessor core vulnerability employ a single error model. In terms of microprocessor vulnerability to MBUs, multiple, non-concurrent faults are discussed in [26], in order to evaluate the efficiency of design diversity. The study in [10] triggered the definition of a new probabilistic framework for incorporating vulnerability of memories to different fault multiplicities into AVF [27]. Finally, [15] investigates the effects of multiple faults on the operation of a microprocessor. However, none of the aforementioned studies discusses the vulnerability of in-core memories to MBUs.

In order to evaluate the vulnerability of in-core memory arrays to MBUs, accurate and realistic modeling of multiple faults is required. Typically observed fail bit patterns, as presented in Figure 3, indicate that multi-bit upsets do not manifest as multiple bit-flips randomly spread across rows and columns; instead, they are clustered in double stripes perpendicular to the word-lines and manifest as ‘force-to-0’

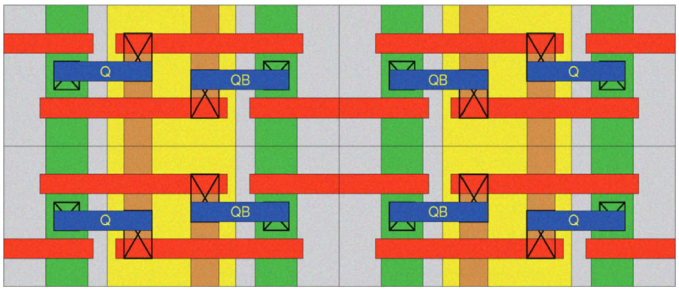
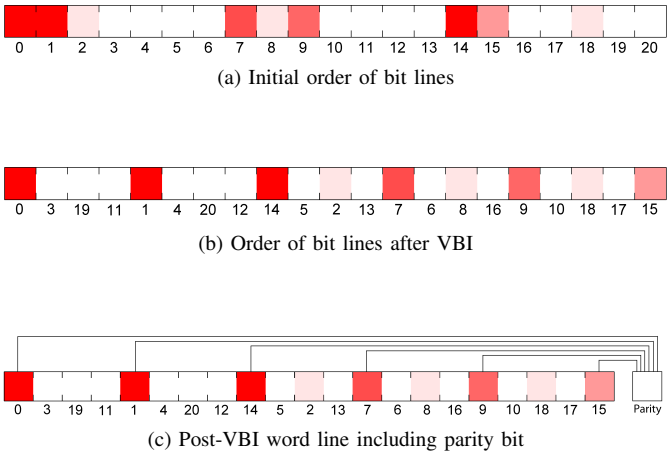



Fig. 4. Compact mirror layout of arrays [27]




(a) Initial order of bit lines

(b) Order of bit lines after VBI

(c) Post-VBI word line including parity bit

Fig. 5. Example of Vulnerability-based Interleaving

or 'force-to-1' effects. This is attributed to the 'battery effect' described by Osada [28]. Figure 4 shows a highly compact layout of bit cells widely used in the design of such arrays. Since the p-well is shared among every pair of columns, in case a particle strikes and causes charge collection, the generated charge raises the potential of the bulk and turns on a parasitic bipolar transistor. Hence, the circuit node is shorted to the bulk and the contents of the cell are flipped. There is also a probability that parasitic bipolar transistors in neighboring cells sharing the same p-well will turn on, effectively generating an MBU. Depending on the node hit by the particle, the value of the cell may or may not change. For example, in case node Q is struck when the bit is holding 0, the bit cell will not be affected; the same applies when node QB is struck when the bit has a value of 1. Consequently, about 50% of the upsets will not result into bit flips.

Towards supporting the VBI method proposed herein, we extend the existing vulnerability analysis methods, such as AVF and GSV, to incorporate the effects of MBUs using realistic multi-bit fault dispersion models, as we describe in Section IV-B. A similar fault model was used to characterize MBUs in memories [27].

## III. Vulnerability-based Interleaving

The main idea presented in this study is a method for interleaving individual bit cells based on their probability of affecting instruction execution. This method, called Vulnerability-based Interleaving (VBI), rearranges the position of certain bit lines in the stored word. Throughout this paper, bit lines refer to the columns of a memory array (vertical lines), and word lines refer to the rows (horizontal lines) (Fig. 2). VBI aims to improve design resiliency by exploiting the fact that important bit lines are usually adjacent, rendering the memory array susceptible to multiple bit errors. Experimental results presented in Section V confirm this observation. Thus, by physically dispersing the critical bit lines and protecting only those with selective parity, VBI greatly improves the resiliency of a given design.

While parity does not offer error correction, microprocessor speculative execution provides the framework to suppress error effects. Thus, when an error is detected by parity, the pipeline is flushed, the instruction queue is cleared and execution restarts from the last committed instruction. Therefore, since we are targeting in-core memory arrays, expensive ECC protection is not required.

An example of VBI appears in Fig. 5. A common layout of the information stored in a typical out-of-order instruction queue consists of a bit for the validity of the instruction, a bit indicating whether the instruction has been issued to the functional units, bits storing the location of the instruction in memory (Program Counter - PC), bits storing the instruction operands, bits storing the predicted branch direction, etc. Evidently, not all fields are equally critical: for example, an error affecting the *issue* and *valid* bits, which are usually the first bits stored in the instruction queue, will certainly result in instruction corruption. However, the PC information is rarely used, thus the likelihood of errors in these bits affecting instruction execution is very low. Fig. 5a shows the initial layout of bit-fields in the stored word, where darker coloring implies higher probability to affect workload output. A possible rearrangement based on VBI appears in Fig. 5b: More critical bit fields are spread throughout the word, thus minimizing the probability that an MBU will affect more than one of them. With the addition of selective parity protection, as shown in Fig. 5c, the vulnerability of the memory array is expected to decrease, as the critical bit lines are protected and the probability of an MBU affecting more than one critical bit-line is very low (and dependent on the interleaving distance).

Horizontal MBUs (i.e. multiple errors in the same word) are now detected by the parity bit of the corresponding word, and vertical MBUs (i.e. multiple errors propagating along the bit lines) are detected by the individual parity bits of each word.

### A. VBI algorithm

The proposed method for rearranging the bit lines appears in Algorithm 1. The inputs of the algorithm consist of the memory array to be protected, the percentage $B$ of bit-lines to be protected with parity (given a specific resiliency budget), as well as a vulnerability figure for the individual cells of the memory array. These vulnerability figures are obtained

through fault simulation, using the MBU fault model defined in Section IV-B. The first step of the algorithm is to calculate the individual vulnerability factor for each vertical bit-line, by summing the vulnerability factors of each bit in this bit-line. Then, the algorithm ranks the list of bit-lines in decreasing order of vulnerability. The next step is to pick the top $B$ percent of this rank-ordered list, and physically disperse them equidistantly along the word line.

The final step is to allocate the remaining bit-lines. In order to minimize the effect of MBUs, the sum of vulnerability factors of the bit-lines placed in between parity-protected bit-lines should be minimized. Thus, the algorithm traverses the ranked list from the end, and allocates the least important bit-lines adjacent to the parity-protected ones, in a circular fashion. For example, if bit-fields a, b and c are protected, then the algorithm assigns the least critical bit-fields to positions in the following order: $a+1$, $b+1$, $c+1$, $a-1$, $b-1$, $c-1$, $a+2$, $b+2$, $c+2$, etc. The algorithm continues until all bit-fields are assigned. While this assignment may slightly increase the routing overhead of the design, experimental results presented in Section V-C indicate that application of VBI incurs minimal power and no area overhead.

**Algorithm 1:** VBI algorithm

```
Assume A_{MxN} is the target memory MxN array;
Assume VF_{MxN} is the vulnerability factor for each bit of the array;
Assume AVBI is the output of the VBI algorithm;
Assume B is the percentage of bit-lines to be protected;
/* Find bit-line vulnerability */
for i = 0; i < M; i++; do
    VFcol_i = 0;
    for j = 0; j < N; j++; do
        VFcol_i += VF_{i,j};
    end
end
/* Sort columns by vulnerability */
Sort(VFcol, key = VFcol_i);
/* Initialize output column order list */
for i = 0; i < M; i++ do
    AVBI_i = -1;
end
/* Distribute the first B columns */
placed = 0;
for i = 0; i < M; i+ = 1/B; do
    index(AVBI_i) = index(VFcol_placed);
    placed += 1;
end
/* Fill in the gaps by placing the least important bits
   around the parity-protected ones */
offset = 1;
for i = M; i > B * M; i - -; do
    for j = 0; j < M; j+ = 1/B; do
        index(AVBI_{j+offset}) = index(VFcol_i);
    end
    if offset > 0 then
        offset = - offset;
    else
        offset = abs(offset) + 1;
    end
end
```

## IV. Experimental Setup

We now discuss the specifics of the large-scale MBU simulation-based study which we performed in order to assess the effectiveness of the proposed VBI method.

### A. Injected modules

The test vehicle used in this study is an RTL model of an Alpha 21264 microprocessor [29]. The Alpha processor incorporates all the features present in current commercial microprocessors, such as aggressive out-of-order scheduling, deep 12-stage pipeline, superscalar execution, etc. The instruction queue, which is part of the instruction scheduler, is the main target of this study. A block diagram of the scheduler appears in Figure 6. The instruction queue features a 32-slot, 219-bit array for storing up to 4 incoming instructions per cycle, which arrive from the rename logic. The embedded scoreboard resolves data hazards and checks operand availability. When an instruction is ready, it is dispatched to one of the 6 functional units. One instruction can be assigned to each functional unit per cycle.

The information stored in the instruction queue is presented in Table I. The first 32-bits contain the instruction word, fetched from the instruction cache. The fetch unit appends information about the location and the target of the instruction and feeds the renaming logic. During the rename stage of the pipeline, several fields are added to the instruction in-flight, i.e., functional unit destination, renamed registers, branch information, etc. When the instruction reaches the scheduler, the ROB id as well as the *issue* and *valid* bits are appended before the instruction is stored in the queue for execution.

### B. MBU fault model

In Section II, we discussed how MBUs propagate as clustered 'force-to-0' or 'force-to-1' effects. Of course, particle effects will not manifest exclusively as 1, 2, 3 or 4 BUs, but rather as a distribution of MBUs of different cardinality. Using the data extracted by radiation experiments performed in [10] (appearing in Figure 1), we define a representative MBU distribution based on the frequency distribution of MBUs for the 65nm process node. This distribution (`65nmRD`) consists of 45% 1 BU, 18% 2 BUs, 10% 3 BUs and 27% 4 BUs. Given the continuous technology scaling, the 65nm model, which includes 45% MBUs, is representative of today's in-field memory corruption profile.

The fault effect radius of the defined representative distributions is limited to 4 bits. Additional experiments showed that the vulnerability does not increase significantly for upset radius of more than 4 bits. This can be attributed to the timing of the strike and the possible inactivity of the affected

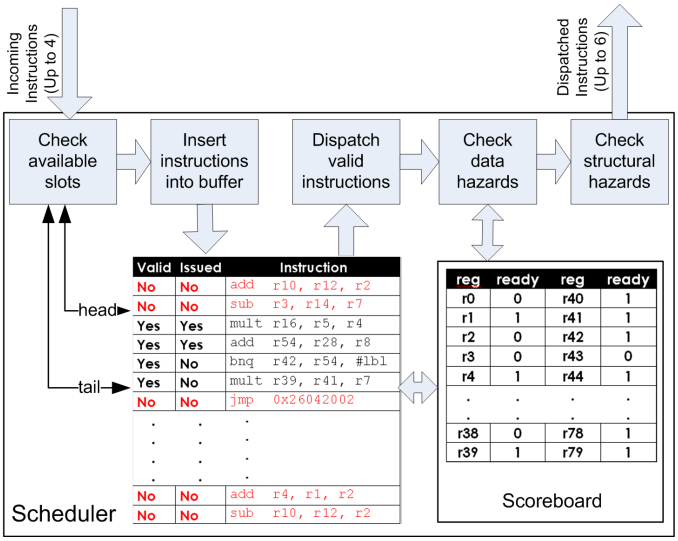



Fig. 6. Block diagram of Alpha 21264 instruction scheduler

TABLE I
BIT FIELDS OF THE INSTRUCTION STORED IN THE INSTRUCTION QUEUE

| Instruction after fetch | | Instruction after rename | | Instruction after issue | |
|---|---|---|---|---|---|
| Name | Bits | Name | Bits | Name | Bits |
| ARCH_DEST | 4:0 | NEEDS_DEST_REG | 161 | ROBID | 216:203 |
| OP_FUNC | 11:5 | BRANCH | 162 | ISSUED | 217 |
| OPLIT | 12 | SIMPLE | 163 | VALID | 218 |
| LIT | 15:13 | COMPLEX | 164 | | |
| SRCB | 20:16 | MULTIPLY | 165 | | |
| SRCA | 26:21 | MEMORY | 166 | | |
| OPCODE | 31:26 | WRITES_TO_DEST | 167 | | |
| FETCH_PC_IN | 95:32 | LOAD | 168 | | |
| FETCH_PCTAG_IN | 159:96 | USES_SRCA | 169 | | |
| VALID_FETCH | 160 | USES_SRCB | 170 | | |
| | | ARCH_DEST | 175:171 | | |
| | | CONDITIONAL_BR | 176 | | |
| | | UNCONDITIONAL_BR | 177 | | |
| | | INDIRECT_BR | 178 | | |
| | | MEM_OP_SIZE | 180:179 | | |
| | | MEM_UN_OR_CMOV | 181 | | |
| | | DEST_PHYS | 188:182 | | |
| | | OLD_DEST_PHYS | 195:189 | | |
| | | SRCA_PHYS | 202:196 | | |
| | | SRCB_PHYS | 209:203 | | |

structure. Specifically, 3+ BUs will most probably affect the outcome of the simulation, as long as the structure is in use. Thus, 5+ BUs can be approximated by 4 BUs for saving simulation time.

### C. Fault injection

Hierarchical statistical fault injection [30] was employed for the large-scale fault injection campaigns performed. Specifically, a 3-level hierarchical design is used: Scheduler, Out-Of-Order cluster, Full chip.

Using the MBU fault model discussed in IV-B, up to 4 MBUs were injected using a uniform distribution across location and time. For each fault model used, a minimum of 250 injections per bit were performed. In total, the results were acquired after approximately 10M fault simulations. The simulations were performed using two servers featuring two Quad-Core Xeon processors with 16GB of RAM.

TABLE II
SPEC BENCHMARK STATISTICS

| Workload | Masked | SDC | Stall | TLB miss |
|---|---|---|---|---|
| **bzip2** | 84.57% | 0.11% | 14.93% | 0.39% |
| **cc** | 90.20% | 0.10% | 9.22% | 0.48% |
| **gzip** | 84.77% | 0.25% | 14.72% | 0.26% |
| **parser** | 89.19% | 0.03% | 10.44% | 0.34% |
| **vortex** | 93.64% | 0.00% | 6.10% | 0.26% |
| **Average** | 88.47% | 0.10% | 11.08% | 0.35% |

### D. Benchmarks and fault outcomes

In order to meet in-field reliability expectations, vulnerability analysis should rely on employing real life applications. For the purpose of this study, we utilize 5 SPEC benchmarks, namely `bzip2`, `cc`, `gzip`, `parser` and `vortex`.

In the presence of an injected SEU/MBU fault, workload execution can be affected in four different ways:

- *Fault masked*: The output of the SPEC benchmark is correct, thus the fault was masked by the architecture or the application.
- *Stall*: The fault caused the microprocessor to stall, either due to an invalid state, instruction commitment halt or unhandled instruction exception. Typically, a stall appears to the user as a system error or crash.
- *Silent Data Corruption (SDC)*: The application finished successfully, but the output is erroneous.
- *TLB miss*: The fault changed the memory access location to an invalid one, leading to a system crash. The TLB miss includes both data and instruction TLB misses.

In the results presented in the next section, an erroneous execution encompasses the superset of Stall, SDC and TLB miss outcomes. As presented in Table II, the architecture and application successfully mask, on average, 88.47% of the injected faults. This number is consistent with the figures appearing in recent studies [29]. The majority of the faults affecting execution stall the microprocessor. Very few errors escape as TLB miss or SDC, but the latter is the most important possible outcome, requiring elaborate protection mechanisms. Summarizing, the instruction scheduler of the Alpha 21264 has an average AVF of 0.12.

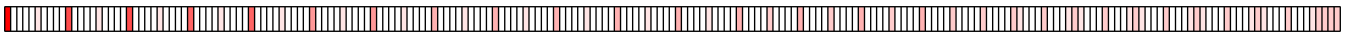
## V. RESULTS AND DISCUSSION

Figures of merit for the proposed VBI methodology are provided in this section, including vulnerability improvement and the corresponding overhead for the instruction queue of the Alpha 21264 instruction scheduler.

### A. VBI performance

We first demonstrate the effectiveness of the VBI method in protecting the Alpha 21264 instruction queue. Figure 7a shows the initial order of bit-lines, before application of VBI. Darker color indicates higher vulnerability factors. Figure 7b presents the new order of bit-fields, after dispersing the bit-lines using the VBI algorithm, assuming a parity protection

(a) Initial order of bit-lines in the instruction queue



(b) Order of bit-lines after VBI

Fig. 7. Pre- and Post-VBI order of bit-lines in the instruction queue (10% parity-protected bit-lines)

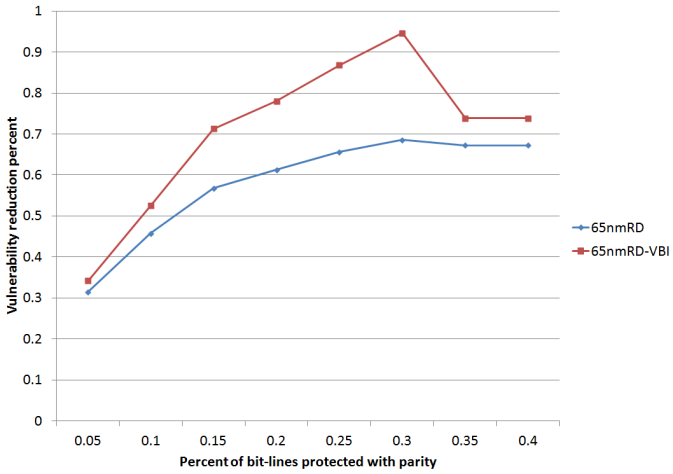



Fig. 8. Vulnerability reduction estimates for different parity schemes

budget of $B = 10\%$ (i.e. 22 bits). As expected, the most critical bit-lines are placed at a maximum distance, based on the given percentage of parity protected bits.

The vulnerability reduction achieved by VBI deployment appears in Fig. 8. For example, using VBI placement with 30% selective parity offers a vulnerability factor reduction of 94%, which exceeds by 26% the vulnerability reduction offered by the original placement and selective parity (68%). The results corroborate that VBI and parity deployment provide better results in terms of vulnerability reduction as compared to a parity-only configuration, for any number of protected bit-lines.

Another interesting observation in Fig. 8 is that the fault coverage figure drops when more than 30% of the bit-fields are used to form the selective parity. This result is attributed to the small interleaving distance between the protected bit-lines in such configuration. When including more than 30% of the parity bits, the interleaving distance is less than 4. Thus, MBUs with a radius of 3 and 4 bits (horizontally) will corrupt two parity protected bit-fields, effectively masking the error effect on the parity bit. Evidently, given the model of expected in-field failures, which includes such 3 and 4 bit horizontal MBUs, exceeding 30% of bits included in the formation of selective parity is not advisable.

As discussed in Section III, after an error is detected by parity, error correction is performed by removing the pending instructions from the pipeline and resuming execution from the last error-free instruction. This leads to a performance penalty, as errors are not corrected on the fly and execution needs to be restarted. However, since the expected error rate is very small, this penalty is negligible.

### *B. Selective parity overhead*

The trade-off between the number of bit-fields included in the selective parity and the corresponding logic and delay overhead is summarized in Table III. The overhead figures are relative to the size of the instruction scheduler (which occupies 11% of the Alpha 21264 core). The parity trees were synthesized using Synopsys Design Compiler, with one parity tree added per instruction word (for a total of 32 trees).

As expected, the logic overhead is proportional to the number of parity-protected bit-fields, yet it is overall very low, amounting to 6.79% of the instruction scheduler when the optimal value for $B = 30\%$ is used. The delay overhead incurred for evaluating the parity bit is slightly higher, amounting to 9.02% for the same configuration. These relatively low overhead figures highlight the benefits of applying VBI to modern microprocessor arrays, especially when one points out that what one gets in return is a 94% vulnerability reduction for in-core memory arrays, wherein ECC is not applicable due to its excessive delay.

### *C. VBI overhead*

Lastly, we estimate the overhead of applying VBI to the instruction scheduler of the Alpha 21264 microprocessor. As described in Section IV-A, the scheduler features a 32-slot, 219-bit wide instruction queue for storing the incoming instructions (up to 4) from the renaming logic, and can dispatch up to 6 instructions to the corresponding functional units. We synthesized the Alpha 21264 scheduler using Synopsys Design Compiler and we generated a floorplan and layout using Synopsys Integrated Circuits Compiler. The target library was the Synopsys Generic Library which includes 9 metal layers. The floorplanning for the instruction scheduler was initialized to match the actual layout used for the commercial Alpha 21264, as shown in Fig. 9 (INT IBOX). The instruction queue SRAM array is placed on the top of the module, and the scoreboard on the bottom.

We then repeated the process, this time using the new bit-line arrangement for the instruction queue, as dictated by the VBI algorithm, while keeping the floorplan and the I/O port location the same. The modified order of incoming bits resulted in an increase in total wire length within the instruction

TABLE III
OVERHEAD FOR DIFFERENT PARITY SCHEMES

| Percentage of bit-fields protected | Number of bits protected | Logic overhead | Delay overhead | Parity-only vulnerability reduction | Parity+VBI vulnerability reduction |
|---|---|---|---|---|---|
| **5%** | 11 | 0.02% | 2.27% | 31.47% | 34.16% |
| **10%** | 22 | 0.20% | 6.70% | 45.77% | 52.54% |
| **15%** | 33 | 1.85% | 8.15% | 56.76% | 71.32% |
| **20%** | 44 | 3.99% | 8.40% | 61.28% | 78.07% |
| **25%** | 55 | 5.53% | 8.85% | 65.63% | 86.83% |
| **30%** | 66 | 6.79% | 9.02% | 68.60% | 94.26% |
| **35%** | 77 | 8.01% | 9.05% | 67.22% | 73.82% |
| **40%** | 88 | 8.76% | 9.08% | 67.22% | 73.82% |

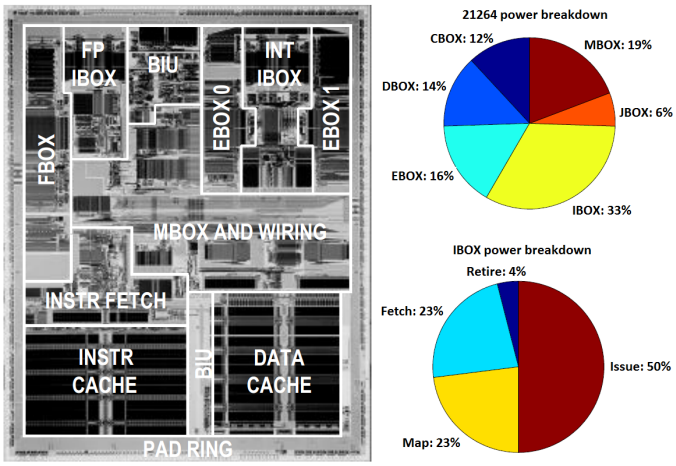



Fig. 9. Alpha IBOX [31], [32]

scheduler, as well as extra buffers which the synthesis and layout tools added in order to meet the timing constraints set forth (in our experiments the clock frequency was set to 1GHz). However, the results show that no area overhead was incurred by applying VBI. This can be explained by observing that the instruction scheduler has a large number of I/O ports, due to which the utilization of the control logic area is rather low. This, in turn, allows for efficient routing and buffer addition even in the case of a completely random rewiring of the incoming bits. The added wires and buffers, however, do cause an increase in the power consumed by the design. Specifically, after applying the VBI algorithm, the total dynamic power of the control logic increased by 0.09%. Given that the IBOX consumes 33% of the total power of the microprocessor (Fig. 9), this overhead is negligible. Overall, the area and power overhead incurred by the VBI algorithm is minimal and is well justified given the achieved vulnerability reduction.

## VI. CONCLUSIONS - FUTURE DIRECTIONS

Recent irradiation studies in contemporary process nodes reveal a significant increase in multiple bit upsets, highlighting the need for revisiting vulnerability analysis and developing novel methods for protecting modern microprocessor core memory arrays against MBUs. To this end, we introduced a new method called Vulnerability-based Interleaving (VBI), which rearranges the bit-lines of the words stored in such arrays based on their relative vulnerability, in order to maximize the effectiveness of a selective parity approach in suppressing MBUs. In order to evaluate the effectiveness of VBI, we employed a fault model that includes MBUs based on the findings of recent irradiation studies. Experimentation with realistic multi-bit faults in the instruction queue of the the Alpha 21264 scheduler elucidates that, when combined with low-cost selective parity protection, VBI can lead to vulnerability reduction of 94% with minimal overhead. The benefits of employing VBI are expected to increase further, as smaller process nodes exhibit greater vulnerability to MBUs.

Towards further exploring and improving the effectiveness of the VBI method, our future research plans include various directions. First, we believe that VBI can be refined by using back-annotated information from the physical layer during bit-line rearrangement, in order to further contain logic and power overhead. Second, we see an opportunity for extending VBI beyond detection of horizontal MBUs, which is its current focus. Indeed, vertical MBUs are currently detected by separate parity bits, yet one can envision word-line interleaving or even efficient creation of parity trees from the entire memory array. Third, formation of more than one parity bits per word may allow further exploration of the effectiveness vs. overhead trade-off, especially with respect to delay. Finally, we intend to apply VBI to various other in-core memory arrays, such as the Reorder Buffer, in order to further evaluate its potential.

## ACKNOWLEDGEMENT

This work is co-funded by the European Regional Development Fund and the Republic of Cyprus through the Research Promotion Foundation (DESMI/New Infrastructure Project/Strategic/0308/26).

## REFERENCES

[1] J. Abella, R. Canal, and A. Gonzalez, "Power- and complexity-aware issue queue designs," *IEEE Micro*, vol. 23, no. 5, pp. 50–58, 2003.

[2] S. Chaudhry, R. Cypher, M. Ekman, M. Karlsson, A. Landin, S. Yip, H. Zeffer, and M. Tremblay, "Rock: A high-performance SPARC CMT processor," *IEEE Micro*, vol. 29, no. 2, pp. 6–16, 2009.

[3] A. Buyuktosunoglu, D.H. Albonesi, P. Bose, P.W. Cook, and S.E. Schuster, "Tradeoffs in power-efficient issue queue design," in *International Symposium on Low Power Electronics and Design*. ACM, 2002, pp. 184–189.

[4] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[5] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[6] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[7] T.L. Criswell, P.R. Measel, and K.L. Wahlin, "Single event upset testing with relativistic heavy ions," *IEEE Transactions on Nuclear Science*, vol. 31, no. 6, pp. 1559–1561, 1984.

[8] R.A. Reed, M.A. Carts, P.W. Marshall, C.J. Marshall, O. Musseau, P.J. McNulty, D.R. Roth, S. Buchner, J. Melinger, and T. Corbiere, "Heavy ion and proton-induced single event multiple upset," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2224–2229, 1997.

[9] R. Koga, S.D. Pinkerton, T.J. Lie, and K.B. Crawford, "Single-word multiple-bit upsets in static random access devices," *IEEE Transactions on Nuclear Science*, vol. 40, no. 6, pp. 1941–1946, 1993.

[10] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, and F. Ruckerbauer, "Investigation of increased multi-bit failure rate due to neutron induced SEU in advanced embedded SRAMs," in *IEEE Symposium on VLSI Circuits*, 2007, pp. 80–81.

[11] A.D. Tipton, J.A. Pellish, R.A. Reed, R.D. Schrimpf, R.A. Weller, M.H. Mendenhall, B. Sierawski, A.K. Sutton, R.M. Diestelhorst, G. Espinel, et al., "Multiple-bit upset in 130 nm CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3259–3264, 2006.

[12] G. Gasiot, D. Giot, and P. Roche, "Multiple cell upsets as the key contribution to the total SER of 65 nm CMOS SRAMs and its dependence on well engineering," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2468–2473, 2007.

[13] Y. Tosaka, H. Ehara, M. Igeta, T. Uemura, H. Oka, N. Matsuoka, and K. Hatanaka, "Comprehensive study of soft errors in advanced CMOS circuits with 90/130 nm technology," in *IEEE International Electron Devices Meeting*, 2004, pp. 941–944.

[14] Y. Yahagi, H. Yamaguchi, E. Ibe, H. Kameyama, M. Sato, T. Akioka, and S. Yamamoto, "A novel feature of neutron-induced multi-cell upsets in 130 and 180 nm SRAMs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 1030–1036, 2007.

[15] E. Touloupis, J.A. Flint, V.A. Chouliaras, and D.D. Ward, "Study of the effects of SEU-induced faults on a pipeline-protected microprocessor," *IEEE Transactions on Computers*, pp. 1585–1596, 2007.

[16] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs," in *IEEE European Solid-State Circuits Conference*, 2008, pp. 222–225.

[17] S. Shamshiri and K.T. Cheng, "Error-locality-aware linear coding to correct multi-bit upsets in SRAMs," in *IEEE International Test Conference*, 2010, pp. 1–10.

[18] C.W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, 2005.

[19] S. Baeg, S.J. Wen, and R. Wong, "SRAM interleaving distance selection with a soft error failure model," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, 2009.

[20] S.S. Mukherjee, J. Emer, T. Fossum, and S.K. Reinhardt, "Cache scrubbing in microprocessors: Myth or necessity?," in *IEEE Pacific Rim International Symposium on Dependable Computing*, 2004, pp. 37–42.

[21] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[22] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 29–40.

[23] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," *SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 460–469, 2007.

[24] P.N. Sanda, J.W. Kellington, P. Kudva, R. Kalla, R.B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C.R. Jones, "Soft-error resilience of the IBM POWER6 processor," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 275–284, 2008.

[25] M. Maniatakos, C. Tirumurti, R. Galivanche, and Y. Makris, "Global signal vulnerability (GSV) analysis for selective state element hardening in modern microprocessors," *IEEE Transactions on Computers*, 2012 (to appear).

[26] S. Mitra, N.R. Saxena, and E.J. McCluskey, "A design diversity metric and analysis of redundant systems," *IEEE Transactions on Computers*, pp. 498–510, 2002.

[27] N.J. George, C.R. Elks, B.W. Johnson, and J. Lach, "Transient fault models and AVF estimation revisited," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010, pp. 477–486.

[28] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, "SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 5, pp. 827–833, 2004.

[29] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *International Conference on Dependable Systems and Networks*, 2004, pp. 61–70.

[30] M. Maniatakos, C. Tirumurti, A. Jas, and Y. Makris, "AVF analysis acceleration via hierarchical fault pruning," in *IEEE European Test Symposium*, 2011, pp. 87–92.

[31] M. Matson, D. Bailey, S. Bell, L. Biro, S. Butler, J. Clouser, J. Farrell, M. Gowan, D. Priore, and K. Wilcox, "Circuit implementation of a 600 MHz superscalar RISC microprocessor," in *IEEE International Conference on Computer Design*, 1998, pp. 104–110.

[32] A. Buyuktosunoglu, A. El-Moursy, and D.H. Albonesi, "An oldest-first selection logic implementation for non-compacting issue queues," in *IEEE International ASIC/SOC Conference*, 2002, pp. 31–35.