Comprehensive Circuit Failure Prediction for Logic and SRAM Using Virtual Aging

A comprehensive failure-prediction technique for many-core processors addresses wear out in harsh environments for logic and static RAM using virtual aging. The design has a simple implementation and delivers low complexity, low overhead, and high accuracy. The system ensures no corruptions or missed errors from wear-out failures and predicts failures within 0.4 days for logic and within milliseconds for SRAM.

In the future, especially in harsh environments (such as aerospace, underwater, and military), microprocessors are increasingly likely to fail in the field because of manufacturing test fault escapes and various aging and wear-out phenomena. Circuit failure prediction techniques employ wear-out device physics principles and empirical measurements to predict failures in the field before they occur for logic and static RAM (SRAM). Models of the dominant mechanisms—negative bias temperature instability (NBTI), Hot Carrier Injection (HCI), and time-dependent dielectric breakdown (TDDB)—show logic wear out increases the delay of gates because a degraded $V_{th}$ increases the $(V_{DD} - V_{th})$. However, wear out of SRAM transistors affects the SRAM arrays’ performance parameters (such as read stability, write stability, and read delay) differently. Previous work has shown that read stability is the dominant failure in SRAM arrays because of the wear out. The effect of aging on transistors’ mobility is not considered.

Extensive literature has addressed wear-out-prediction inspired by these observations (in the interest of space, we provide one representative citation). However, as far as we know, no prior work simultaneously addresses both logic and SRAM. Furthermore, they individually suffer from complexity, overhead, and accuracy and generality problems and become particularly ineffective in harsh environments in which wear-out challenges are exacerbated. These prior techniques are discussed further in the “Related Work in Circuit Failure Prediction” sidebar.

Our goal is to develop a unifying yet simple mechanism that covers both logic and SRAM and delivers low complexity, low overhead, and high accuracy. To this end, we developed a comprehensive circuit-prediction technique called the Aged Full-Chip Predictor for both logic and SRAM in many-core systems. Aged Full-Chip Predictor allows safe execution up to 0.4 days before logic failures and extends the typical lifetime by 14 months, over a system with ECC for SRAM.
Related Work in Circuit Failure Prediction

Figure A shows the various alternatives for handling wear out in logic and SRAM. Dimitris Gizopoulos and colleagues provide a good overview of detection techniques for logic.\(^1\) Logic wear-out prediction is based on canaries,\(^2\) in-situ flip-flop techniques,\(^3\) delay measurement,\(^4\) and built-in self-test (BIST).\(^5\) SRAM-based detection and prediction techniques are based on sensors or modifications to the SRAM cell.\(^6,7\) complex error-correcting codes (ECCs), and hybrid ECC and cell sizing.\(^8\) None of these can simultaneously deliver on low complexity, low overheads, and high accuracy because these techniques operate within only a single computing layer. When done at the circuit level, these techniques suffer from complexity and always remain active. On the other hand, an architecture-level-only solution suffers from low accuracy because architecture fault models do not capture most physical effects. (In both logic- and SRAM-based directions, there is a body of work on mitigation and repair, which is complementary and somewhat orthogonal to detection and prediction.)

References

Design
The design of the Aged Full-Chip Predictor leverages three primary mechanisms. We discuss the insight for each and outline their design below. Figure 1 provides an overview of the execution of our comprehensive failure-prediction system.

Virtual aging to manifest faults
Our key insight is to virtually wear out the processor and thus manifest a wear-out fault early. We convert the wear-out degradation into a higher-level and easier-to-detect fault; we then expose and detect the fault, which effectively predicts and detects the wear out.
Figure 1. Two techniques, based on virtual aging, together provide comprehensive failure prediction. Aged-SDMR detects manifested logic errors using sampling and dual-modular redundancy, whereas Aged-AsymChk detects manifested SRAM errors using asymmetric checking.

All device-level wear-out faults eventually must manifest at a higher abstraction level; thus, any detection technique can be repurposed as a prediction technique.

We carry out virtual aging by reducing supply voltage using dynamic voltage scaling. We can tune the prediction’s timeliness by changing the amount of voltage reduction. Virtual aging is instantaneously reversible; resetting to nominal voltage restores the processor’s current age.

Sampled redundancy to expose and detect logic failure

We observed that wear out in logic is first exposed as a logic delay fault, and sampled redundancy with execution on a second core can be effective in handling logic transistors. BIST and stuck-at fault models are insufficient for providing full coverage for these delay-driven failures.

The key idea of the solution, Aged-SDMR, is to couple cores randomly at randomly chosen periods of time, run one core virtually aged, use the second (redundant) core as a checker core, and couple these using a nonintrusive lightweight mechanism. Because logic faults start as delay faults, a comprehensive redundant core is necessary for full coverage. Shuou Nomura and colleagues introduced the concept of Sampling+DMR,7 which solves the overhead problem that historically has plagued redundancy. Our key advancement over their work is to use virtual aging during DMR execution to ensure that faults always occur first in a DMR window, thus ensuring no missed errors.

Asymmetric checkers to expose and detect SRAM failure

Aged-SDMR cannot be used for SRAM because checkpointing the entire SRAM state is infeasible, especially considering today’s megabyte-sized level-2 caches. However, wear out in SRAMs results in read stability problems, and therefore its effect can be captured by a simple stuck-at fault model.

The solution, Aged-AsymChk, leverages this insight and uses established asymmetric checker technology such as BIST to check the SRAMs when they are virtually aged. Specifically, we write known vectors to an SRAM, then read out the values; any mismatch between these indicates an impending failure.
Use of existing techniques
The principles of dynamic voltage scaling, sampling, redundancy, and asymmetric checking using BIST are well known. Our work's implementation and design contribution is a novel use of existing techniques, while avoiding disruptive or intrusive mechanisms and providing comprehensive logic and SRAM wear-out prediction. The implementation requirements are simple or already existent: dynamic voltage scaling capability; separate voltage islands for SRAMs and logic; a reliability manager module added to cores to allow checking of retired instructions; BIST capability in the SRAMs; and a controller (like a cache controller) in the SRAM that allows its contents to be safely evicted prior to being overwritten for BIST.

Implementation
We present the organization of our system and the implementation of virtual aging, fault exposure, and fault detection. Within each, we discuss logic and SRAM. Figure 1 shows the high-level overview and details of each individual approach. We focus on SRAM in this article because our previous work covered the logic.8

Overall organization
Conceptually, we execute the processor in epochs, where at the start of every epoch we have a window where the processor is virtually aged. As Figure 1 shows, we have two types of epochs: logic epochs (L-epochs), in which only the logic is virtually aged, and SRAM-epochs (S-epochs), in which only SRAM is virtually aged. These never overlap and are executed at different rates.

Virtual aging
We virtually age a processor by reducing the supply voltage to both logic and SRAM arrays. Although the enabling mechanism is the same, the failure behavior is different. For SRAM, prior to virtual aging, we must ensure any useful SRAM state is written to some other location. For an SRAM that is part of a cache, the cache controller can be enhanced to evict all dirty lines. Otherwise, it can be done completely in software using instructions like WBINVD (writeback and invalidate cache) in the AMD 64 architecture. SRAMS in speculative structures such as branch predictor tables can simply be overwritten. Precise interrupts that would start an S-epoch ensure that structures such as load queues and the rename table are empty. We can virtually age large memory structures, such as L2 caches with many SRAM blocks, by applying the S-epochs one SRAM array at a time coordinated with the controller to turn off banks.

Effect on logic. The delay of a gate \( t_d \) is inversely proportional to \((V_{DD} - V_{th})^2\). Wear out causes \( V_{th} \) and hence \( t_d \) to increase. Reducing \( V_{DD} \) has the same effect and can be calibrated to mimic weeks or months of aging.

Effect on SRAM. Consider the basic six-transistor SRAM cell organization. In a newly manufactured cell, the cross-coupled inverters are fairly identical, producing a voltage transfer characteristic as in Figure 2a. The static noise margin (SNM) is the minimum noise or extraneous voltage that can corrupt the stored value. The read failure probability defines this likelihood for a given cell. Owing to wear out, the SRAM’s inverters degrade, reducing the static noise margin as shown in Figures 2b and 2c, which consequently increases the read failure probability. Furthermore, SRAM wear out is asymmetric and depends on the stored value in the SRAM cell. For example, when zero value is stored in the SRAM cell, the \( p \)-channel MOS transistor in one of the inverters is subjected to stress, whereas the \( n \)-MOS transistor in the other one goes into the recovery mode. With extremely high wear out, cells can become stuck at 0 or 1 permanently (see Figure 2d).

Virtual aging’s behavior for SRAM is similar to the logic case. The fundamental source for SNM change is decreased \((V_{DD} - V_{th})\) due to increased \( V_{th} \), which can be achieved equivalently by decreasing \( V_{DD} \) and can be instantaneously reset back to the current age by resetting to nominal \( V_{DD} \). Figure 3 shows an HSpice simulation of virtual aging’s effectiveness. Using MOS reliability analysis (MOSRA) aging models, we ran simulations of the SRAM cell with various amounts of aging—for the technology and the MOSRA parameters that we considered,
failure happened at approximately 12 years (626 weeks) for a worst-case stressed cell (that is, one that constantly stores either one or zero in the SRAM cell for the duration of the aging). The MOSRA parameters are $T_{TH} = 5e - 8; T_{ITT} = 7,5e - 10; T_{ITTD} = 1,45e - 20; TN = 0,5; RelMode = default$ (both HCI and BTI).

At each aging setting, we also ran a simulation with various amounts of voltage reduction. In this case, we first obtained the total amount of stress on transistors during the whole period of the aging with the nominal voltage, which shows itself as shift in the $V_{th}$.

Given the shifted $V_{th}$ values for each transistor, we simulated the SRAM cell with the reduced voltage to observe the aging failure. The dots in the figure indicate the age at which the cell failed for various amounts of voltage reduction. Subtracting this age from 12 years provides the window of advance failure notification. This experiment demonstrates that reducing voltage serves the purpose of virtual aging.

Fault exposure

The fault exposure mechanism is what makes all errors visible to the detection mechanism.

Figure 2. Six-transistor (6T) SRAM cell transfer characteristics and the read failure in the SRAM cell. 6T SRAM transfer characteristics for a (a) new chip, (b) positive read static noise margin (SNM) after wear out, and (c) zero read SNM after wear out. (d) Negative (near-zero) read SNM causes the stored value in the SRAM to flip (initial stored value is zero).
Logic. Exposing permanent faults in the critical path is straightforward. Permanent faults keep producing the fault in the circuit. However, based on the input values, some of the faults might be masked. Therefore, we need a mechanism to do more than one sampling to guarantee the detection mechanism’s completeness. Figure 4a shows how degradation affects a critical path, assuming that guardband is added to accommodate aging. As the chip ages, the delay increases and the guardband slack decreases. When the delay degradation overshoots the guardband (3 years in the figure), soft breakdown occurs. Under virtual aging, the additional delay in gates that fall in near-critical paths show up as faults at the flip-flops they drive. This causes a bit-flip (or metastability) at the output of the flip-flops that can propagate to cause an architectural state corruption. These faults are exposed, with no modifications required to the processor. Figure 1 shows an example circuit block highlighting the fact that the critical path is left unmodified.

Noncritical paths introduce subtle challenges because gates that are exclusively on noncritical paths (fast gates) can degrade directly to hard breakdown without ever manifesting as a delay fault, thus circumventing the prediction mechanism. Simple clock-phase shifting logic can be added to gates on noncritical paths to effectively expose their delays (see Figure 1). Because modifications are only to paths that have much slack, they are not a source of complexity.

SRAM. The goal of fault exposure is to condition a failed cell to produce errors. Our main contribution here is based on a simple observation: the read stability problem in failed cells can be abstracted as a stuck-at-zero or a stuck-at-one fault if we can write known values into the SRAM and then read them. We reuse the pattern generators in memory BIST to produce these values: a simple “March” algorithm that writes all zeros followed by all ones will suffice for Aged-AsymChk.

Fault detection
The fault-detection mechanism compares measured (read) values against known (written) values to determine when a fault has occurred.

SRAM. The detection phase is trivial for Aged-AsymChk, because the BIST controller knows what values to expect—any differences are flagged as impending failures.

Discussion
An important question to consider is, compared to prior works, what do we lose or what assumptions are broken or ignored? We make one judicious cross-layer (circuit to
architecture layer) assumption: the state or values in the SRAM can be drained using an architectural mechanism, allowing the SRAM’s contents to be overwritten to allow BIST-based stuck-at-fault testing periodically. In the context of a microprocessor execution, this is a reasonable and easy-to-implement assumption. However, the circuit-based techniques attempt to address wear out in isolation and hence avoid such assumptions.

**Evaluation**

Our goal of understanding wear out and the Aged Full-Chip Predictor’s effectiveness is organized around eight questions, of which questions 5 through 8 address overhead and accuracy.

- Q1: Are wear out and its effects measurably observable?
- Q2: Can voltage reduction virtually manifest wear-out faults?
- Q3: Are the manifested faults exposed to a higher level?
- Q4: Are the faults exposed to the higher level detected?
- Q5: What are the overheads?
- Q6: What is the delay to predict the wear out?
- Q7: When does this technique probably fail to predict wear out?
- Q8: How does this technique compare to the current state-of-the-art methods?

We examine each question for logic and SRAM. By design, we achieve low complexity, which was our other key goal.

**Methodology**

Our evaluation of the Aged Full-Chip Predictor uses a prototype system we built on the basis of the OpenRISC processor (see Figure 5). For logic and Aged-SDMR, our general philosophy is as follows:

- Use Spice and MOSRA with the 32-nm silicon-on-insulator library to evaluate any gate-level effects.
- Use gate-level delay-aware simulations to check for timing faults.
- Use full-system emulation on the field-programmable gate array when actual runtime data is required.

For Aged-AsymChk, our evaluation is similar:
- Use Spice and MOSRA to evaluate any gate-level effects, including the noise margin.
- Use the noise-margin results to determine failures in SRAM reads.
- Use analytical models and workload measurements to determine the effect of applications on wear out.

One difference is that we run more benchmarks using larger input sets, totaling 35 and spanning SPEC2K, SPEC2006, MediaBench, and Parboil, to capture cache and SRAM effects more representatively.

**Aged-SDMR results**

Table 1 summarizes the key results for Aged-SDMR, and Table 2 compares Aged-SDMR to three state-of-the-art techniques. 9-11

**Aged-AsymChk results**

We address the evaluation questions for Aged-AsymChk in detail below.

**Understanding degradation (Q1).** Degradation in SRAM devices is measurably observable and cannot be statically determined because it depends on the switching activity. Figure 3 previously showed this aging behavior at the cell level. Figure 6a shows the wear out at the application level for every cell in a 64-Kbyte data cache (a two-way set associative, level-1 cache with 64-byte blocks). Here, we quantify and visualize wear-out intensity using a simple model: we count the number of cycles that a cell is 1 as a unit of wear out, and we assume every transition to 0 is \(-1/100\)th of one unit (modeling NBTI recovery). For all applications, we consider a 200-million-cycle window, and pixel values are normalized to maximum wear out. Two banks form the cache ways, shown side by side.

We also determined the average and standard deviation of wear out across all the

---

**Table 1**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average Wear Out</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aged-SDMR</td>
<td>0.05</td>
<td>0.02</td>
</tr>
<tr>
<td>Aged-AsymChk</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>Aged-Full-Chip Predictor</td>
<td>0.03</td>
<td>0.01</td>
</tr>
</tbody>
</table>

---

**Figure 5.** Evaluation setup. We built a prototype system based on the OpenRISC processor to evaluate the Aged Full-Chip Predictor.
bits with all 35 applications and computed it to be 0.278 and 0.2895. Even simply looking at distributions of wear out among the bits, we observe they sometimes follow a normal distribution but with large differences in standard deviation and variance across benchmarks (see Figure 6b). These data measurements demonstrate the diversity and

### Table 1. Aged-SDMR results

<table>
<thead>
<tr>
<th>Evaluation questions</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding degradation (Q1)</td>
<td>Delay degradation in CMOS logic is measurably observable.</td>
</tr>
<tr>
<td></td>
<td>Dependent on factors including switching activity (cannot be statically determined).</td>
</tr>
<tr>
<td>Manifesting faults (Q2)</td>
<td>Reducing VDD mimics aging. For example, a 50-mV (4.1%) reduction corresponds to predicting up to nine months in advance.</td>
</tr>
<tr>
<td>Exposing faults (Q3)</td>
<td>While in Aged-SDMR mode, timing faults indicate impending hard or soft breakdowns. Virtual aging induces timing faults at the rate of between 0 to 9.8%.</td>
</tr>
<tr>
<td>Detecting faults (Q4)</td>
<td>Faults introduced in Aged-SDMR mode translate to architectural errors and can be caught without escapes. Empirically, errors were seen in at least 0.02% of cycles and were caught within a few samples.</td>
</tr>
<tr>
<td>Estimating overheads (Q5)</td>
<td>Aged-SDMR has small area (8.9%), power (2.54%), and energy (0.7%) overheads.</td>
</tr>
<tr>
<td>Delay to predict (Q6)</td>
<td>We can guarantee an upper bound on Aged-SDMR’s prediction latency mathematically, based on defect and sampling rates. The longest latency to predict is 0.4 days.</td>
</tr>
<tr>
<td>When the technique does not work (Q7)</td>
<td>Aged-SDMR cannot predict faults that do not start as delay faults. For delay-based faults, missed sites are those that have high switching activity but do not affect the architectural trace (integer benchmarks might do this to the floating-point pipeline). If more than 0.4 days of life remain, Aged-SDMR will still predict correctly. Masking scenario is rare in commercial designs because power/value gating avoids unnecessary switching.</td>
</tr>
<tr>
<td>Comparison to state-of-the-art methods (Q8)</td>
<td>Aged-SDMR is comparable, if not better, on other metrics and also provides generality. Previous techniques do not provide generality and accuracy, leaving fast gates (30 to 40% of gates) uncovered.</td>
</tr>
</tbody>
</table>

### Table 2. A comparison of Aged-SDMR and three state-of-the-art techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Overheads</th>
<th>Time to predict</th>
<th>Prediction horizon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (%)</td>
<td>Power (%)</td>
<td></td>
</tr>
<tr>
<td>Online wear-out prediction⁹</td>
<td>4.6†</td>
<td>8.6†</td>
<td>4 days</td>
</tr>
<tr>
<td>WearMon¹¹</td>
<td>~14‡</td>
<td>Not reported</td>
<td>Varies</td>
</tr>
<tr>
<td>FIRST¹⁰</td>
<td>Not reported</td>
<td>0</td>
<td>1 day</td>
</tr>
<tr>
<td>Aged-SDMR</td>
<td>8.94</td>
<td>3.2</td>
<td>0.4 days</td>
</tr>
</tbody>
</table>

†For every eight signals monitored.
‡Rough estimates from field-programmable gate array use numbers reported by the authors.
*Assuming a virtual aging mechanism similar to this work.
substantiate two points—that the degradation is highly application dependent, and that degradation within the different cells of an SRAM block can vary significantly.

Manifesting faults (Q2). As we demonstrated earlier, reducing $V_{DD}$ mimics aging (see Figure 3). Empirically, for example, a 45-mV reduction emulated 28 weeks of aging.

Exposing faults (Q3) and detecting faults at a higher level (Q4). Figure 2d showed that the end effect of SRAM cell aging is read failure stability. By design, writing 1s and then reading them exposes the wear-out fault under virtual aging.

Estimating overheads (Q5). In terms of area, there is practically no additional overhead—we simply reuse the existing BIST circuitry. In terms of performance slowdown, Aged-AsymChk can be run quite infrequently. Because it predicts wear out without memory corruption and is 100 percent accurate, the only requirement is to run at periods less than the age mimicked by virtual aging, which is on the order of weeks. On the basis of our empirical data, the overhead of checking is pessimistically on the order of 1 million cycles. Even assuming that S-epochs are activated as often as every 100 context switches, which at a 5-ms OS scheduling quantum would be half a second, a 1-GHz processor at one instruction per cycle would have negligible overhead (0.2 percent). Therefore, Aged-AsymChk introduces no significant performance, power, or area overhead to the system.

Delay to predict (Q6). Compared to logic, the delay to predict for SRAM is on the order of milliseconds, because the prediction happens in a single S-epoch and is application independent. The delay guarantees for logic are probabilistic and are for the worst case, because some sampling windows are required to guarantee overlap of the DMR window with a fault occurrence by the application.

When the technique does not work (Q7). Failures in SRAM that do not start as read failures cannot be detected. Although these exist and include electromigration, for example, there is evidence that NBTI, which we cover, is dominant. Unlike the logic case, for device faults that adhere to the model, Aged-AsymChk is 100 percent correct because it is based on the formal BIST model that can generate vectors with 100 percent coverage.
Comparison to state-of-the-art methods (Q8)
As we mentioned earlier, prior work does not provide low overhead, high accuracy, and low complexity. Quantitatively, Aged-AsymChk either eliminates silent data corruptions for baselines without ECC or it increases the array’s lifetime.

We developed an SRAM array defect-rate model to show how we can extend the average proficient lifetime by 14 months, considering common wear-out patterns. We first used a fixed cell-failure model (excluding dynamic sources of wear out such as the application and temperature) and then extended those results, considering time-varying failure rates.

Failure model preliminaries. Using basic probability, we built a simple analytical model for how wear out affects SRAM array failure. The key input was a cell’s read failure probability at a given time \( f_c(t) \). (The read failure probability indicates the probability that a six-transistor SRAM cell has a read failure at a given time. For example, the read failure probability \( 10^{-7} \) indicates that one SRAM cell out of 10^7 cells has read failure.)

We considered an SRAM made of \( n \) blocks and used cache-block granularity single-error correction and double-error detection ECC.

We used two cache block sizes with \( k \) data bits and \( e \) ECC bits: (16, 6) and (256, 10). Also, we define the defect rate as the defective parts per million. Furthermore, the single-failure defect rate considers one bit failure to be a defect, whereas the double-failure defect rate considers two failures (in a single block) to be a defect. ECC-only arrays are proficient only until the first error, at which point they must be decommissioned to prevent uncorrectable errors. Arrays with prediction capability are proficient until just before the second error, extending their lifetime.

SRAM array model for fixed defect rates. We can build a defect rate model, based on the binomial probability model, for an SRAM array by calculating the failure probability of bits in a cache block \( f_c(t) \), then the failure probability of blocks in the array. We consider both single-failure (Equation 1) and double-failure (Equation 2) cases below.

\[
\begin{align*}
&f_{\text{block, 1}}(t) = 1 - (1 - f_c(t))^{k+e} \\
&f_{\text{block, 2}}(t) = 1 - \left((1 - f_c(t))^{k+e} + (k + e)/1 \ast f_c(t) \ast (1 - f_c(t))^{k+e-1}\right) \\
&f_{\text{array}}(t) = 1 - \left(1 - f_{\text{block, 2}}(t)\right)^n
\end{align*}
\]

Equations 1 and 2 calculate the probability that one or two bits, respectively, in a given \((k + e)\)-bit block are erroneous at a given time. Equation 3 finds the probability that one block in a given SRAM array made of \( n \) blocks is faulty at a given time.

Table 3 shows the single- and double-failure defect rates for various cell failure probabilities \( f_c(t) \) and two extreme granularities of ECC.

We can draw three implications from Table 3. First, as expected, fine-grained ECC has a lower defect rate. Second, at low cell-failure probabilities, the number of failures with only a single defect is orders of magnitude more than when allowing prediction. And third, schemes decommissioning arrays and cache blocks at first failure incur wasted lifetime: nearly 100 and 36 percent of coarse- and fine-grained ECC, with \( f_c(t) = 10^{-5} \).
Extending results for dynamic wear out. To quantify the wasted lifetime for SRAM arrays, we extend the model to include dynamic SRAM wear out, the primary effect of which is to cause $f_1(t)$ to become time dependent (increasing over time). Our extended model must incorporate several issues. First, the wear out of different bits will vary, implying that a single $f_1(t)$ no longer models the entire array. Second, depending on the SRAM’s usage, the $f_1(t)$ changes to some value by the end of the SRAM array’s lifetime. Third, $f_1(t)$ changes at some rate with time to reach this final value. Finally, we must determine when the array is single-failure defective or double-failure defective. These phenomena are highly application dependent, and we make some simplifying assumptions to capture first-order effects.

First, we assume the highest $f_1(t)$ of the bits in a block, thus providing a lower-bound estimate on wasted life. Second, we assume $f_1(t)$ changes by one order of magnitude due to wear out—this has strong empirical evidence from circuit literature. Finally, to model the rate of change of $f_1(t)$, we consider reciprocal, linear change and exponential as in Figure 7a. Linear change is likely the common case. Exponential and reciprocal represent the worst (pessimistic) case and best (optimistic) case for the benefits of our technique, respectively. We considered a 36-month period discretized at monthly granularity, and we assumed the second error occurs at the end of this period. We used $f_1(t)$ at each month to calculate the defect rates, which determine how many arrays are wasted due to early decommissioning based on the first failure.

Figure 7b shows the dynamic wear-out model’s results in terms of months of added life for a percent of the SRAM arrays, which suggests two things. First, the lifetime can be extended significantly to 17, 14, and 7 months on average for the three scenarios. Second, significant fractions of SRAM arrays are improved by 95, 87, and 46 percent, respectively.

By providing a unified technique for error prediction in both logic and SRAM settings, which is low overhead and has high fault coverage, the Aged Full-Chip Predictor could serve as an important component for future fault-dominated technologies. The mechanisms behind the concepts of virtual aging and sampling are well understood and easy to implement, making the idea attractive and practical to deploy. One primary implication is that future designs can more aggressively provision the resources for recovering from soft errors (such as ECC in SRAMs), while relying on the Aged Full-Chip Predictor for the prediction and detection of hard errors. Looking forward, understanding the relationship between delay degradation and failure modes in far-out semiconductor technologies will be the key to using virtual aging to address future reliability challenges.

References

Amir Yazdanbakhsh is a PhD student in the School of Computer Science at the Georgia Institute of Technology and a research assistant in the Alternative Computing Technologies (ACT) Lab. His research interests include computer architecture, approximate general-purpose computing, mixed-signal accelerator design, machine learning, and programming languages for hardware design. Yazdanbakhsh has an MS in computer engineering from the University of Wisconsin–Madison and an MS in electrical and computer engineering from the University of Tehran. He is a student member of IEEE. Contact him at a.yazdanbakhsh@gatech.edu.

Raghuraman Balasubramanian is a digital design engineer at Google. His research interests include microprocessor architecture and circuit design. Balasubramanian has an MS in computer science from the University of Wisconsin–Madison, where he completed the work for this article. Contact him at raguehuran.b@gmail.com.

Tony Nowatzki is a PhD student in the Department of Computer Sciences at the University of Wisconsin–Madison and a member of the Vertical Research Group. His research interests include architecture and compiler co-design and mathematical modeling. Nowatzki has an MS in computer science from the University of Wisconsin–Madison, where he is a student member of IEEE. Contact him at tjn@cs.wisc.edu.

Karthiskeyan Sankaralingam is an associate professor in the Department of Computer Sciences and the Department of Electrical and Computer Engineering at the University of Wisconsin–Madison, where he also leads the Vertical Research Group. His research interests include microarchitecture, architecture, and very large-scale integration. Sankaralingam has a PhD in computer science from the University of Texas at Austin. He is a senior member of IEEE. Contact him at karu@cs.wisc.edu.