Custard: ASIC Workload-Aware Reliable Design for Multicore IoT Processors

Scott Lerner, *Student Member, IEEE*, Isikcan Yilmaz, *Student Member, IEEE*, and Baris Taskin, *Senior Member, IEEE*

***Abstract*—In the typical application-specified integrated circuit (ASIC) design flow, reliability-driven performance loss is computed, in part, with switching activity files. However, for ASIC designs of multicore processors, the typical switching activity files lack multithreaded software workload information. An accurate switching activity for a multicore design can be generated using a logic simulator. However, the logic simulator process suffers from long runtimes when dealing with real workloads. This paper analyzes the effects of scaling multithreaded workloads and proposes Custard, a hardware methodology for lifetime improvement of multicore processors by obtaining multithreaded switching activity signatures in a short period of time using a performance simulator (gem5), logic simulator (VCS), and thermal simulator (HotSpot). Custard is particularly important for multicore, Internet of Things processors as the runtime feedback-based reliability mechanisms used on current multicore processors incur area and power overhead that could be prohibitive for smaller form factors and power budgets. Experiments are performed with Custard using real workloads on an OpenSPARC T1 design with two, four, and eight cores that are fully synthesized and routed. The default-sized T1 core is improved to have a reliability increase of 4.1×, with 0.08% and 1.57% increase on average in cell area and switching power, respectively.**

***Index Terms*—Custard, Internet of Things (IoT), lifetime, multicore, workload aware.**

I. Introduction

DESIGNS of next-generation multicore processors continue to grow in complexity and require granularity in the analysis. Averaging of transistor switching activity at the block level is typically used for single-core processor reliability [1]. Extra overhead is introduced from averaging, so utilizing switching activity at the transistor level can harden the processors against degradation effects while reducing overhead. However, a gap exists between multithreaded workload information and application-specified integrated circuit (ASIC)-level hardware design. This paper introduces Custard to bridge the gap between multithreaded workloads and ASIC hardware design while focusing on hardening multicore Internet of Things (IoT) processors against degradation.

Manuscript received March 1, 2018; revised June 19, 2018 and September 6, 2018; accepted October 14, 2018. Date of publication November 15, 2018; date of current version February 22, 2019. This work was supported by the National Science Foundation under Grant DGE-1656737. *(Corresponding author: Scott Lerner.)*

The authors are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104 USA (e-mail: spl29@drexel.edu; iy43@drexel.edu; bt62@drexel.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2878664

Nanoscale transistors at advanced technology nodes suffer from temporal device degradation that negatively affects the system reliability [2]–[4]. Negative bias temperature instability (NBTI) is the main contributor to temporal degradation for aging devices. Smaller feature sizes in advanced technology nodes have additional NBTI degradation from an increased electric field over the oxide [5]. Switching activity files are the most common method used to model the degradation caused by NBTI [1]. Using the switching activity as an input, the NBTI model estimates the degradation of a device. However, current techniques for generating switching activity files do not model the threads of multithreaded software workloads. Modeling the threads provides a better estimate of the NBTI degradation compared to averaging switching activity at a block level. This paper explores the effects of multithreaded workloads while scaling the number of cores available on multicore processors.

For multicore processing systems, which are the main target of this paper, the conventionally adopted method to mitigate NBTI-induced reliability degradation is the use of runtime techniques in thread scheduling and reliability feedback mechanisms [6]–[10]. The use of these runtime techniques is particularly common in high-performance processors, where the power/thermal/area overhead of these techniques can be tolerated. Conversely, IoT processors, primarily due to smaller form factors, are subject to tighter power and thermal constraints. Consequently, for specialized processors, the use of runtime techniques such as thread scheduling [11] is not as mandatory. Core leveling is a straightforward thread scheduling solution to balance degradation by shifting threads over time. This paper focuses on hardware solutions to NBTI degradation but could be implemented with runtime techniques to reduce power.

Code optimization [12], [13] is not considered in this paper because of an observation with the master–worker relationship. Even the most sophisticated workload scheduling cannot force the program to use multiple cores evenly. The master thread spawns worker threads, inherently performing more work compared to the worker threads. In addition, multithreading program practices recommend not using the worker threads for certain purposes such as I/O. Instead, this paper proposes the use of a novel ASIC-level solution at design time, without the overhead of conventional runtime solutions for reliability improvement. Theoretically, high-performance processors with conventional runtime solutions can also benefit from the proposed design time, ASIC-level solution, however, at a smaller rate due to exploring the same design space for reliability.

The OpenSPARC T1 core is a multicore processor with a 64-bit reduced instruction set (RISC) architecture and is used in this paper as the model for a representative IoT processor. The IoT processor would be subject to reliability concerns and is partially motivated by the growing demand for high-performance tasks allocated to IoT nodes. For instance, IoT devices have significantly influenced data flow in cloud computing as simple sensors relaying data [14]. This perspective is changing to edge computing where data are analyzed on the edge of a network; meaning more processing is required before sending. In order to facilitate this goal, more specialized IoT devices are required while remaining reliable and low power.

The proposed work, Custard, is a methodology addendum to the traditional ASIC design flow that includes multithreaded workload signatures for temperature-aware lifetime improvement of multicore processors. Within the core of Custard, a performance simulator (gem5), logic simulator (VCS), and a thermal simulator (HotSpot) are combined and allow the ASIC design to include multithreaded workload signatures for multicore processors. Custard can be utilized on any existing multicore design and increase lifetime without any significant redesign effort compared to adding runtime solutions. In addition to the novelty in modeling multithreaded workload signatures as ASIC switching activity files, Custard explores scaling workloads, introduces novelties in encompassing fine-grain impacts of temperature on multicore lifetime, and a reliability estimator based on fine-grain temperature grids, completing the Custard flow. Custard is validated on an OpenSPARC T1 core in experimentation with two, four, and eight cores, leading to a 4.1× lifetime improvement with 0.08% and 1.57% average increase to the area and power overhead, respectively.

Technical background is presented in Section II. Workload-awareness for multicore processors is motivated in Section III. The proposed methodology is introduced in Section IV. Results from the framework with analysis are provided in Section V. This paper is concluded in Section VI.

II. Technical Background

NBTI and lifetime models are presented in Section II-A. Fundamentals of the switching activity file used in the traditional ASIC flow and the software workload signatures of interest in this paper for multicore processors are reviewed in Section II-B.

A. NBTI and Lifetime

The effects of NBTI have been predicted through many various models of temporal degradation [2]–[5], [15]–[22]. There is a growing body of work on NBTI analysis and estimating lifetime, but those that relate to lifetime analysis of multicore processors have not been utilized. NBTI models are presented in [18] that predict the performance degradation of a circuit. A transistor sizing algorithm for improved reliability is presented in [23] that uses a ten-year lifetime goal to size their circuit considering temperature. However, due to the computational framework leveraging Lagrangian relaxation nonlinear optimization over every transistor, the process does not scale. NBTI-aware logic synthesis is presented in [3] that requires recharacterizing of every gate in a standard cell library based on input signal probability. Recharacterizing a standard cell library based on signal probability polynomially increases the amount of time needed for characterization and the degradation model used is overly pessimistic. The architectural aging analysis framework, `ExtraTime` is presented in [1]. A full-system performance simulator is used in conjunction with aging models to determine degradation but lacks multithreaded analysis. Degradation is determined at the block level with an average switching assigned to high-level structural blocks and not individual transistors.

B. Switching Activity and Software Workloads for Multicore

Switching activity interchange format (SAIF) files estimate the average power by recording the switching probability at circuit logic inputs. Due to the sheer number of input combinations, cycle-accurate switching information at the inputs might not be readily available. Therefore, in common ASIC designs that do not utilize workload information, SAIF files include estimated switching activity, e.g., a probability of 0.5 for switching activity [24].

A theoretically possible, but extremely impractical approach to generating workload signatures for multicore processors is through the use of logic simulators. Logic simulators provide cycle-accurate switching activity given an input hardware description language file and a simulation testbench. For small designs, this can be done relatively quickly. However, as the simulation size grows, the difficulty increases exponentially for the following reasons.

1) A large workload simulation takes an unreasonable amount of time.
2) Output files grow to very large sizes, which are unwieldy and hard to parse efficiently.
3) The workload must be written in assembly and loaded into program memory.

Due to these reasons, the use of logic simulation to generate multithreaded workload signatures is considered infeasible.

While switching activity files have been the preferred method of choice in the standard ASIC flow, multicore architectures demand activity analysis at a higher level of abstraction [25]. The multithreaded programs executing on multicore architectures get scheduled to individual processing cores using simple to sophisticated scheduling schemes, which dictate the scheduling of the switching activities on the transistors of the processing core. However, sophisticated workload signatures capable of representing multithreaded applications have not been utilized in ASIC design. The separation is due to the design hierarchy at the transistor level for switching capacitances and the activity of threads at the core architecture level. This paper is the first in the literature to bridge this gap between the standard ASIC flow practice of switching activity files, the workload, and temperature-aware analysis at the computer architecture level for multicore processors.

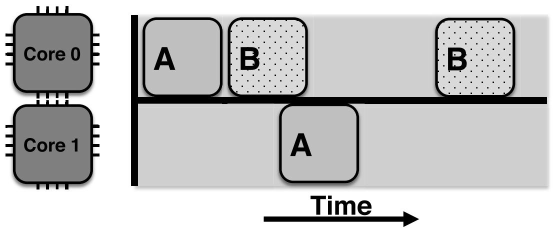


Fig. 1. Workloads that are not optimized for multicore overuses one core. Core 0 is more heavily degraded over time than core 1.

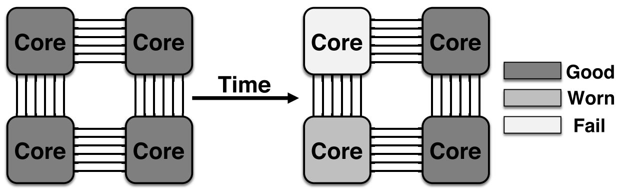


Fig. 2. Overuse of one set of cores causes that set to fail earlier, reducing the lifetime.

III. Motivation for Workload Awareness

Motivation to incorporate application workload signatures with per-core switching activity files in the ASIC design flow is presented in Section III-A. An observation of uneven utilization and scaling issues for instructions per core is described in Section III-B.

A. Workload Switching Activity

Most workloads, including those that are heavily parallelized, inherently have limitations for achieving an even utilization. A side effect of this imbalance of core utilization is the reliability of the cores, investigated in this paper. An example workload is shown in Fig. 1 on a two-core design. Core 0 runs functions A and B multiple times while core 1 only runs function A once. Increased work on core 0 leads to increased degradation of the core. As this workload is allowed to continue running in this order for its lifetime, core 0 will have a shorter lifetime than that of core 1. For this paper, each thread is assumed to be statically scheduled to the same core (e.g., thread 0 to core 0, thread 1 to core 1).

The core that is utilized more has a higher risk of failure and can cause premature failure of the entire chip from just one core. This concept is illustrated in Fig. 2. When the multicore processor is manufactured, it is a safe assumption that all the cores are at the same degradation level. In the case of IoT processors for edge computing, one workload (or a small set of workloads) will be run during its lifetime. Edge computing processors have one workload running and can be compared in concept to accelerators for larger systems. Running the single workload can lead to the failure of the whole processor through the overuse and degradation of a subset of cores, including one core. Lifetime is lost as the other cores can still be utilized and have not been degraded to an unusable state. Custard mitigates this by utilizing homogeneous cores in a heterogeneously sized multicore processor allowing for ease of programming (thanks to the uniformity of computing platform with homogeneous cores) and proactive resistance to the assumed workloads (thanks to the heterogeneous sizing of computing cores). In addition, Custard can be applied to heterogeneous multicore designs using the same framework.

B. Observation on a Scaling Processor

A study is performed on SPLASH2 [26] and PARSEC [27] benchmark suites to investigate how these multithreaded benchmarks use the multiple cores. In order to keep runtime down for initial experiments, the processor chosen to simulate contains two (2) cores.[1]

The percent usage per core for five benchmarks and an average is shown in Fig. 3. These graphs show what percent of the total instructions are run on individual cores, cpu0, or cpu1. The x-axis is a list of instructions that are investigated in this paper. An evenly distributed parallel workload has a percent usage of 50% for the two cores across all instructions, presented as the *Ideal* label in Fig. 3.

The individual instruction *No_OpClass* is a wait performed by the CPU, *IntAlu* is an integer addition or subtraction, *IntMult* is an integer multiply, *IntDiv* is an integer divide, and *FloatAdd* is a floating point addition. In the interest of space, only Fig. 3(c) is analyzed here but similar analyses can be performed for the other benchmarks. Radiosity is a benchmark for rendering based on detailed analysis of light reflections off diffuse surfaces [27]. In Fig. 3(c), there is a large divergence of usage between the cores for the radiosity benchmark. Cpu0 and cpu1 share the *No_OpClass* instructions close to evenly resulting in an even utilization per core for that instruction. The percent of *IntDiv* operations is shared evenly between the two cores, but *IntAlu* and *IntMult* are almost exclusively used by cpu1. The distribution of instructions per cpu implies that cpu1 is performing most of the actual scientific calculation. The resulting *IntDiv* instructions are likely used for a portion of the common computation. The multithreaded version of radiosity is written so that the calculation is executed on the worker thread while the master thread performs some calculation and thread management. Overall, this benchmark inherently separates its type of activity based on what core it is running on; leading to uneven wear. The average shows a 9.2% maximum difference from the ideal value. However, uneven utilization while scaling multicore benchmarks has not been explored in the literature.

The percent usage of a benchmark for a scaling number of cores is shown in Fig. 4. In this case, LU is illustrated scaling from two, four, and eight cores. LU is a benchmark that factors a dense matrix into upper and lower matrix products [28]. This analysis shows how each instruction type is used across cores when there are more cores available. Comparing *No_OpClass* to *Ideal*, it is observed that as the number of cores scale up, the amount of time spent waiting stays close to equal among cores. *IntAlu* shows stratification of the data sent to each core. The two-core case in Fig. 4(a) starts with an 80–20 split of *IntAlu* instructions, but as the number of cores increases, the distribution settles around a 60–40 split between

[1]Additional results are shown in Fig. 4 with four and eight cores, showing even more motivating trends compared to the two-core design.

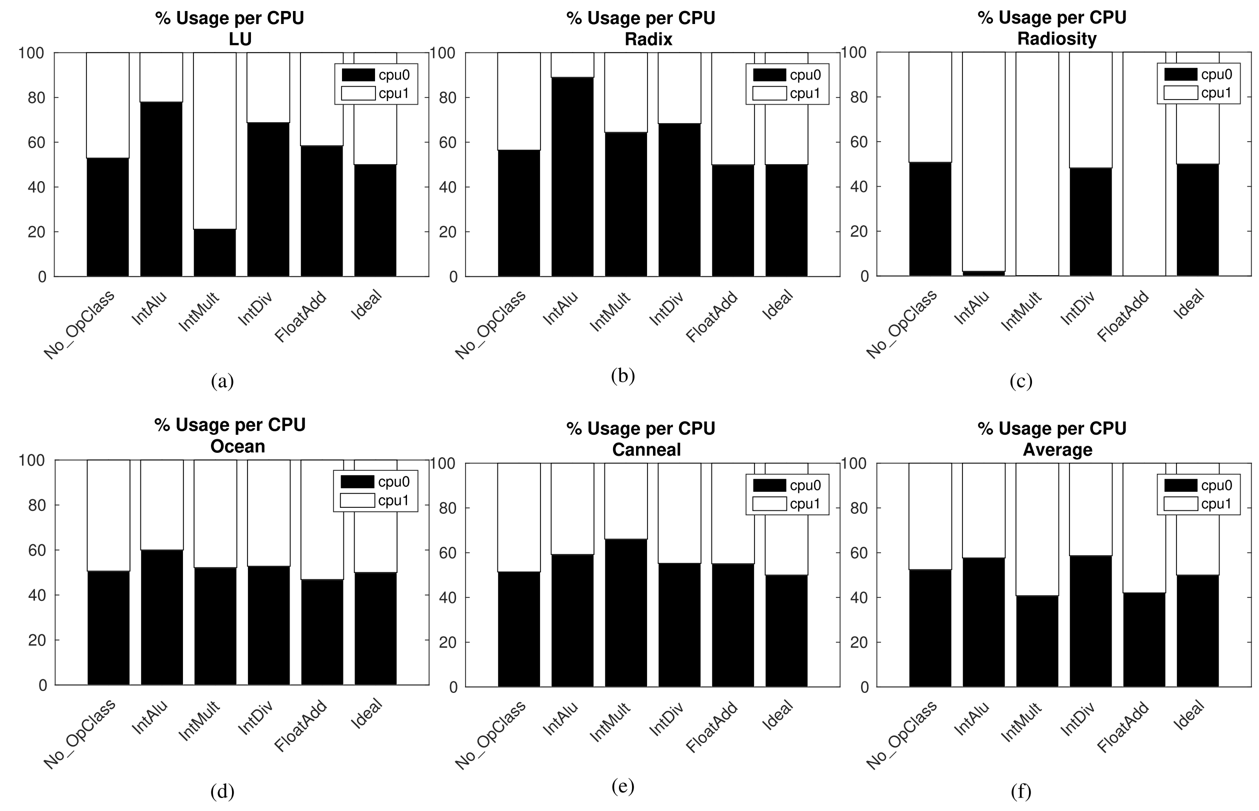


Fig. 3. Percent usage per CPU. If the work was shared evenly per core, the expected result would be 50%. (a) Lu. (b) Radix. (c) Radiosity. (d) Ocean. (e) Canneal. (f) Average.

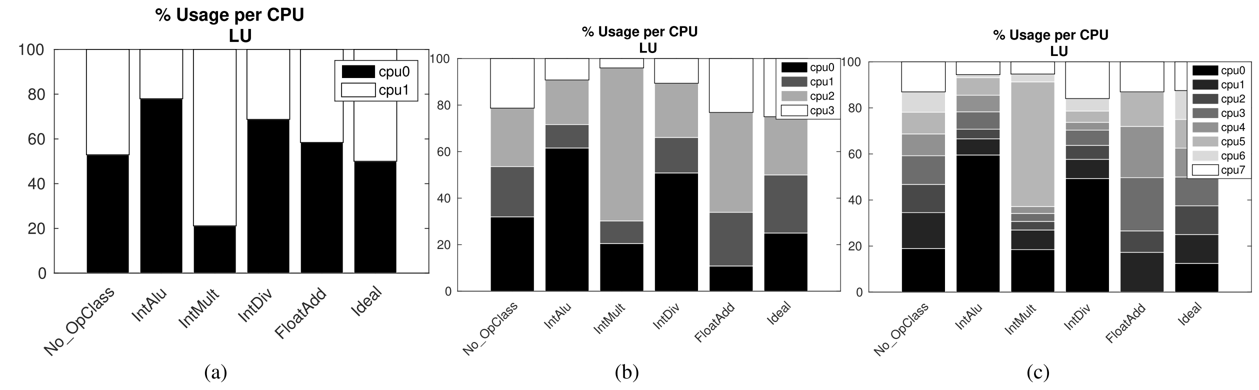


Fig. 4. Percent usage of each type of instruction per CPU for the benchmark LU. The ideal data set is illustrative of how the instructions would be shared evenly. (a) LU 2 Core. (b) LU 4 Core. (c) LU 8 Core.

cpu0 and the remaining cores in Fig. 4(c). This depends on the benchmark and what tasks cpu0 needs to perform aside from the data calculation. The usage stratification demonstrates the magnitude of what has not been shown before, that increasing the core count on a set of data does not always split the task evenly. A similar effect is observed with *IntMult* where the number of cores is increasing, 80%, 65%, and 54% of the *IntMult* instructions are performed on cpu1, cpu2, and cpu5, respectively, as the design scales. Scaling the number of cores available for this benchmark shows that, compared to ideal, there is a 47.0% and 21.7% maximum and average, respectively, amount of uneven usage between the cores. If future workloads do not evenly utilize a scaling number of cores, the degradation problem shown here will become a more pressing issue. By taking a design and sizing individual cores according to the usage numbers shown in this paper, the cores become more resilient without any functional design change.

IV. Custard Methodology

The proposed Custard methodology is shown in Fig. 5 as an addendum to the ASIC design flow. Each part of Custard is vital for proper evaluation of a multicore design. For smaller designs, runtime using a different approach may be acceptable, but as core count increases, Custard can scale to any design size. In addition, Custard can use any technology

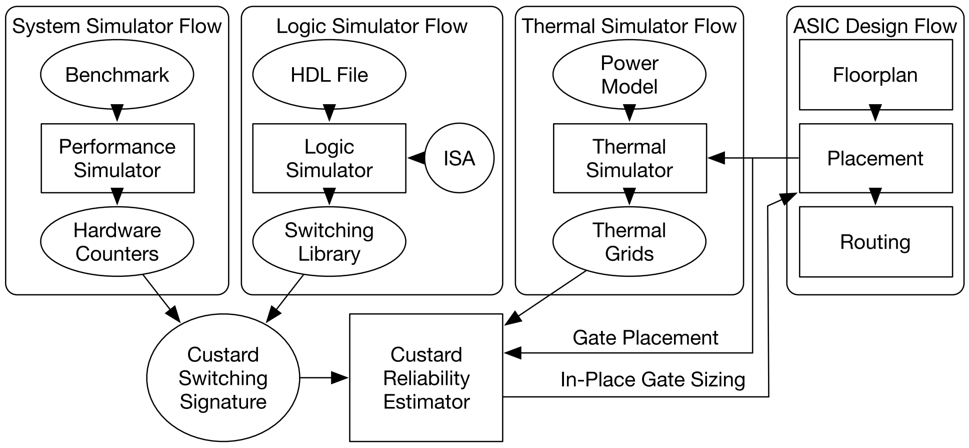


Fig. 5. Custard Methodology.

node compatible with the ASIC design flow. Information from the placement stage is read from the ASIC design flow and in-place gate sizing updates the gate selection; keeping the design flow intact. Multithreaded switching activity signatures are utilized to improve the lifetime of multicore processors in the ASIC design flow. Sections IV-A–IV-E describe, in detail, the system, logic, and thermal simulator flows as well as the switching signature and reliability estimator.

A. System Simulator Flow

The system simulator flow of Custard aims to imprint the system-level operation of a multithreaded workload on the Custard switching signature. A system benchmark workload is an input to the performance simulator. The performance simulator processes the benchmark and collects performance counters. Performance counter information is obtained by feeding a benchmark workload and a hardware model into the performance simulator. The performance simulator then indicates the utilization rates for individual components of a processor, such as the integer execution unit. Utilization rates for hardware components then need to be mapped to transistor activity, using a logic simulator, in order to create a workload switching signature.

B. Logic Simulator Flow

The logic simulator flow simulates functional correctness of a hardware description that can directly map to transistor switching. The hardware description of a core is provided as a Verilog design to the logic simulator along with an instruction testbench. Custom tests are created in Verilog that allow for direct mapping with the performance counters. Each of the custom tests separates individual instructions so that transistor activity can be inferred per instruction.

Each of the main integer operations is run through the logic simulator and a value change dump (VCD) file is created. VCD files contain the detailed time and voltage information for each signal in the design. The files do not scale as the number of instructions in a benchmark and signals simulated in a design grow. To mitigate this, instructions are broken into their basic units (ADD, DIV, MUL) and converted to SAIF files. A custom library of SAIF files is created that is mapped with the hardware counter usage numbers from the system simulator flow to produce a switching signature compatible with standard design tools.

C. Custard Switching Signature

The switching signature (from Fig. 5) is created from the combination of the system simulator and logic simulator flows. The system simulator flow is run once per benchmark and the logic simulator flow is run once per instruction. The switching signature is a new SAIF file that contains the usage from the system simulation with multithreaded information and transistor-level switching from the logic simulator. Let n be the number of times a particular instruction is used. The instruction switching file in the library is multiplied by n and then these scaled instructions are combined together to form the new switching file.

D. Thermal Simulator Flow

The thermal simulator flow provides temperature information about a design. A placement of the hardware design and power models representing the switching activity per core is provided to the thermal simulator. The thermal flow produces steady-state thermal results to be used in the Custard flow. This steady-state temperature gives Custard an advantage over previous work in [29]–[31] by creating temperature grids on a fine-grain level. The fine-grain approach allows for small clusters of gates to have their own temperature instead of worst case analysis. NBTI is significantly impacted by temperature so it is important to have an accurate simulation.

E. Reliability Estimate

After the Custard switching signature is created, the switching signature, gate placement, and the thermal information are inputs to the reliability estimator. Gate locations are obtained postplacement and routing from the ASIC design flow. Gate placement is used to determine temperature information per gate and is assumed as fixed. Legalization after gate sizing is the only time when gate placement is updated. Using a similar procedure to [18], the reliability estimate calculates the temporal degradation of the threshold voltage. Feeding back into the Custard ASIC flow, as shown in Fig. 5, the output of the Custard reliability estimate aids in performing the in-place gate sizing for lifetime improvement.

For any circuit, the reliability estimate determines the lifetime from NBTI degradation based on the threshold voltage. A gate is stated to fail from NBTI degradation once the drain current, I_d, falls 10% from the original value [1]. The Custard reliability estimator, inspired by the estimator from [18], operates by correlating time-dependent variations in V_{th} and I_d to lifetime. Within the proposed Custard ASIC flow, appropriately sized gates are instantiated in order to provide for the lifetime of devices selected in the reliability estimator. By using the trapping–detrapping (T–D) method [32] to predict the V_{th} value, previous results have shown high fidelity to measured silicon. The following equation describes the T–D model and is used to determine the threshold voltage of a transistor after some period of time [18]:

$$\Delta V_{\text{th}}(t) = \phi[A + B \times \log(1 + C \times t)] \tag{1}$$

where ΔV_{th} is the change in threshold voltage, t is time, A, B, and C are the model constants that are technology dependent, and ϕ is described by the following equation:

$$\phi = (k \times T)/q. \tag{2}$$

In (2), k is Boltzmann's constant, T is the temperature, and q is the charge of an electron. The following equation estimates the drain current of a transistor in the saturation region:

$$I_d = \beta(V_g - V_{\text{th}})^{\alpha}, \quad \beta = \frac{\mu C_{\text{ox}} W_{\text{eff}}}{L_{\text{eff}}} \tag{3}$$

where V_g is the gate voltage, V_{th} is the threshold voltage of the transistor, μ is the electron mobility, and W_{eff} and L_{eff} are the effective width and length of the transistor, respectively. The α is a constant whose value ranges from 1 to 2.

Adding (1) into V_{th} of (3) produces a description of how I_d degrades as the V_{th} increases. The reliability estimate determines what gates in a preexisting design need to be resized in order to meet the lifetime. Using the 10% I_d degradation, a gate is determined to fail from lifetime by the following equation:

$$I_{d0} \times 0.9 > \beta(V_g - (V_{\text{th0}} + \Delta V_{\text{th}}(t))^{\alpha} \tag{4}$$

where I_{d0} is the initial drain current and $\Delta V_{\text{th}}(t)$ is change in threshold voltage at time t. Gates are extracted from every path in the design and then run through the reliability estimator. V_{th} and I_d values are used to determine if the path would violate the lifetime, based on (4). The Custard design flow performs in-place sizing on any gates that violate the desired lifetime and performs legalization. A reduction to the overall computation can be achieved by analyzing only the degradation critical paths, but practically, computation time is not seen as a limitation in this paper.

In terms of the NBTI analysis, the temperature and switching activity of each gate are obtained and run through the reliability model. The required reliability of any gate in the design is set to be $t = 10$ years. A ten-year lifetime is selected as a similar scenario to transistor sizing from [23] and is calculated based on voltage for a 32-nm technology [33].

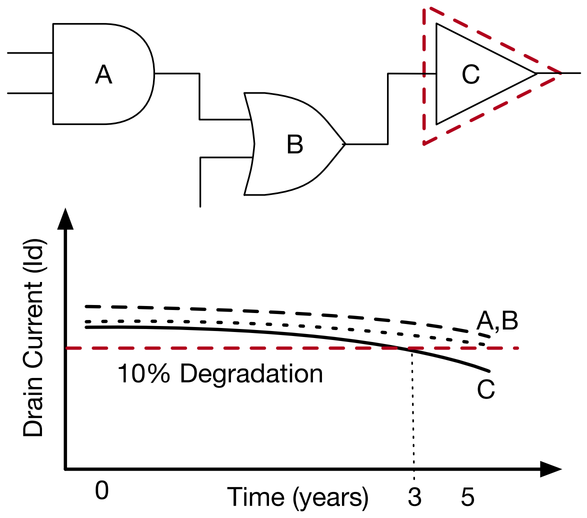


Fig. 6. Gates in a design with drain current shown degrading over time. Gate C reaches degradation threshold in three years.

Degradation of the drain current for gates is shown in Fig. 6 with an illustrative example. The degradation is applied to every gate based on their on-time and temperature. At the desired lifetime, if that V_{th} and I_d of the gate results in a lifetime failure, then the gate is upsized. The 10% degradation of drain current is shown to intersect with the curve for gate C at the three-year mark. At that intersection, gate C has degraded to the time of failure. The size of the gate is iteratively sized until the time of failure is after the required lifetime. Upsizing a gate is determined by the minimum gate size required to satisfy the degradation calculated from (4). A legalization step is performed to verify that the design meets specifications and then metrics are reported.

F. Experimental Setup

In experimental validation of the Custard methodology, the following selection of tools are used.

1) Logic simulator `VCS` [34] (utilized through OpenPiton [35]).
2) System simulator `gem5`-driven [36] architecture simulation flow.
3) Thermal simulator, `Hotspot` [37] is provided with power information to represent realistic temperature of the design.
4) The Custard ASIC flow is implemented with Synopsys tools [34].

An OpenSparc T1 core is chosen as the hardware description for this paper [38]. The logic simulator, VCS, is a commercial tool from Synopsys [34] that simulates hardware design functionality. Testbenches for VCS are created and run through a simulation wrapper, OpenPiton [35], for ease of programming. Gem5 is an open-source full-system performance simulator developed to provide event-based performance counters for a computer architecture [36]. HotSpot is an open-source, fast thermal model generation tool suitable for architectural studies [37]. Each of these tools is commonly used in the design flow.

The gem5 output of benchmarks contains performance counters that are mapped to instruction usage per core. Running concurrently with gem5, custom Verilog testbenches are provided to VCS for the OpenSPARC architecture. Accurate switching profiles are obtained by loading instructions such as ADD, MULT, DIV, and SUB into the program memory of the T1 core and running these operations through VCS. Integer operations are analyzed in this paper as the other instructions are in an unsimulated part of the core. The switching profiles for the OpenSPARC architecture are stored in a library. This library of switching files is used in conjunction with the workload signature from gem5 to form the novel switching activity file that is still compatible with the Synopsys tool flow. An initial synthesis of the design is performed using the switching files. This switching information allows the design flow to provide more accurate power information to the thermal simulator. The fine-grain temperature grid is run to a steady state and analyzed to provide thermal information for every gate in the design. Thermal information is required to be run for each core of the design, but the model represented in the thermal flow does not take a significant amount of time to arrive at a steady state. It is possible for this process to be run within a few minutes. Layout, switching, and temperature information are provided to the Custard reliability estimator to calculate gate failures and sizing. Resizing information is then passed to the design flow to update gate sizes, producing a reliable design.

V. Simulation Results

Following the procedure from Section IV-F, an OpenSPARC T1 core is synthesized using multiple benchmarks. Multiple SPLASH2 and PARSEC benchmarks are simulated through this framework to estimate and improve their lifetime. Each core of the OpenSPARC design is sized accordingly in order to obtain a lifetime of ten years. This lifetime analysis using Custard can be performed for any time constraint. Power, area, and timing results are obtained from Synopsys PrimeTime using the SAED32nm library [34].

A. Lifetime Improvement and Overhead

The baseline lifetime is calculated with the reliability estimator for the baseline two-core OpenSPARC T1 processor design. A ≈2.44-year lifetime is reported from the reliability estimator. Next, the reliability estimator is used to validate the design lifetime after resizing from Custard. Overall, the proposed workload-aware ASIC design methodology achieves a 4.1× improvement in lifetime compared with the unsized core with minimal overhead.

The area and power overhead of the proposed Custard flow are shown in Table I. Switching power is obtained at the worst corner. The *baseline* benchmark is the typical assumption of 50% input switching at the primary inputs. The notation for each design is the benchmark name with a core number followed by the individual cpu of that design, i.e., radiosity2_cpu0 is the first core of a two-core design. It is observed that the area of the design changes minimally (≈0.4%) when the proposed workload-aware ASIC flow sizes for reliability. The OpenSPARC core is a large design so the area is not expected to change and is similar to the results obtained from [15]. The minimal overhead of switching power (≈1.5%) is also not unexpected due to the limited upsizing that affects only <3% of gates.

Table II illustrates the overhead associated with cell sizing for radiosity over the synthesis of two-, four-, and eight-core designs. The switching power is obtained at the same corner from Table I and again the switching power for these larger designs changes only ≈1.5% after the sizing. Cell area for cpu0 after the cell sizing is lower than the other cpus and the same is seen for switching power. The amount of work being done on the master core is less than that in the worker cores, but the type of work matters when determining the lifetime. If the work being sent to the cores is similar, the same blocks

TABLE I
Custard Overhead of Cell Area and Switching Power on a Two-Core Design With Local Temperature Data

Benchmark	% Change from Presizing	
	Cell Area	Switching Power
baseline	0.432%	1.917%
blackscholes2_cpu0	0.051%	1.555%
blackscholes2_cpu1	0.038%	1.502%
fft2_cpu0	0.049%	1.539%
fft2_cpu1	0.045%	1.555%
lu2_cpu0	0.127%	1.641%
lu2_cpu1	0.038%	1.502%
cholesky2_cpu0	0.051%	1.555%
cholesky2_cpu1	0.044%	1.502%
radix2_cpu0	0.038%	1.502%
radix2_cpu1	0.038%	1.502%
Average	0.086%	1.570%

TABLE II
Custard Area and Power Overhead Per Core for Increasing Number of Cores on One Benchmark, Radiosity

Benchmark	% Change from Presizing	
	Cell Area	Switching Power
baseline	0.432%	1.917%
radiosity2_cpu0	0.038%	1.502%
radiosity2_cpu1	0.049%	1.539%
radiosity4_cpu0	0.044%	1.502%
radiosity4_cpu1	0.051%	1.555%
radiosity4_cpu2	0.049%	1.539%
radiosity4_cpu3	0.051%	1.555%
radiosity8_cpu0	0.038%	1.502%
radiosity8_cpu1	0.051%	1.555%
radiosity8_cpu2	0.049%	1.539%
radiosity8_cpu3	0.051%	1.555%
radiosity8_cpu4	0.049%	1.539%
radiosity8_cpu5	0.049%	1.539%
radiosity8_cpu6	0.051%	1.555%
radiosity8_cpu7	0.051%	1.555%
Average	0.074%	1.563%

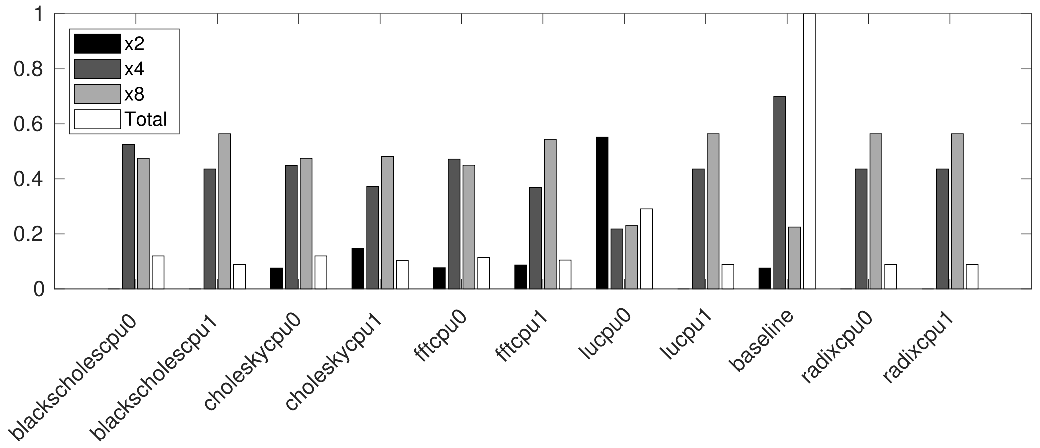


Fig. 7. Normalized cell sizing using local temperature information for all benchmarks split out by core in a two-core processor.

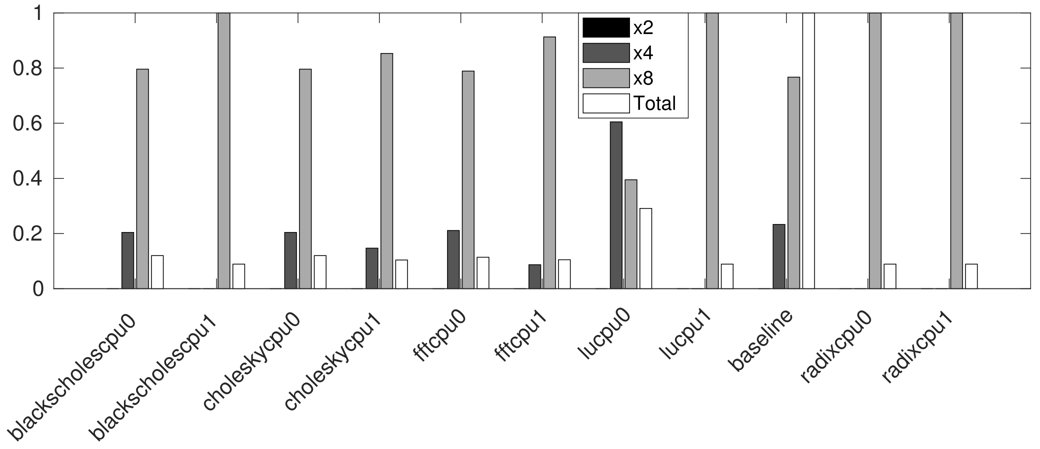


Fig. 8. Normalized cell sizing using worst case temperature for all benchmarks split out by core in a two-core processor.

get selected for resizing. For the radiosity benchmark, the cells in each core that are selected for resizing have a small overlap but are not the same. Each of the benchmark threads stresses different parts of the design. The difference in the number of resized cells between cores for radiosity is up to 43.8% between cpu0 and cpu7.

Comparison of overhead to runtime techniques is limited as scaling technology and design implementation have a large impact on overhead. The circuit-level timing error detection and correction flip-flop presented in [9] report a total power overhead of $\approx$3.1% for an Alpha processor in 180-nm technology. Area overhead for each flip-flop in terms of transistors contain 4.3$\times$ more compared to a traditional flip-flop.

TABLE III
CUSTARD TIMING OVERHEAD POSTGATE SIZING

Benchmark	% Change in Arrival Time
baseline	1.16%
blackscholescpu0	8.71%
blackscholescpu1	8.71%
fftcpu0	8.13%
fftcpu1	8.13%
lucpu0	8.13%
lucpu1	8.13%
choleskycpu0	8.13%
choleskycpu1	8.13%
radixcpu0	8.13%
radixcpu1	8.71%

B. Cell Sizing Overhead

The normalized cell sizing for various benchmarks are shown in Fig. 7 for a two-core processor. Each core is denoted by *cpu0* or *cpu1*. The percentage of gates that are sized to a particular drive strength is shown normalized for each core. Due to the industrial size of the OpenSPARC core, the changes, as shown in Fig. 7, with respect to the area and power are minimal; the number of gates that need upsizing only constitutes $\approx$3% of the total number of gates. In addition to this low number of gates being sized, $\approx$44.1% on average of those that are being sized are sized by X4, as shown with normalized values in Fig. 7. *Total* in Fig. 7 is a normalized bar to the total number of gates sized in *baseline*. It is shown that including workload information in the ASIC design flow reduces the overestimation of baseline switching. The largest amount of difference between cores in a benchmark is LU. LU has differences of 55%, 22%, and 33% for X2, X4, and X8 sizing respectively, between cpu0 and cpu1. This indicates that LU has a strong difference in usage between the cores and is a likely candidate for a single core to fail early without this sizing.

A timing analysis is performed to determine the effect on timing from the Custard approach. The change in arrival time of the critical path is shown in Table III. Compared to the

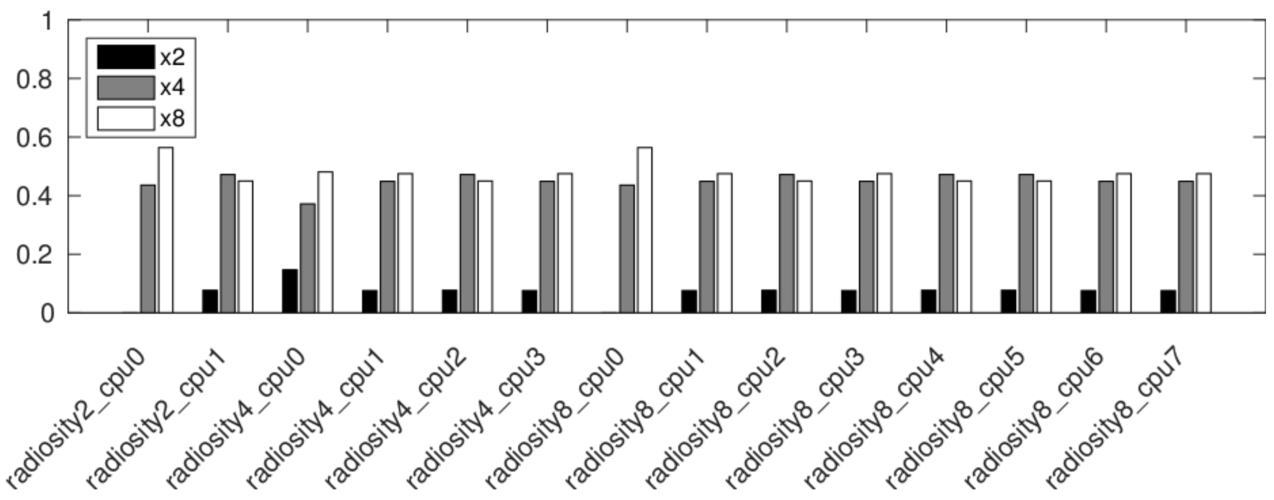


Fig. 9. Normalized cell sizing with local temperature for radiosity with a two-core, four-core, and eight-core configurations.

presized core, the cores show between 2%–8% degradation for the critical path. The timing change is less than 1 ps, however, this can still be detrimental in high-speed applications. For the designs presented here, the change in arrival time does not affect the operating frequency and only reduces the slack margin. Timing critical paths with no available slack margin can be removed from consideration. Path selection techniques presented in [15] and [39] identify critical paths due to aging effects.

C. Temperature for Reliability

Normalized cell sizing with the added local temperature from the thermal simulator flow is shown in Fig. 7. Local temperature uses fine-grain temperature grids spaced equally around the hardware design. Having a larger number of temperature grids increases the complexity of the analysis, but provides a more accurate temperature. The two-core cpu synthesized using local temperature shows the usage of all cell sizes for most benchmarks. Cholesky, fast Fourier transform, and LU all use various cell sizes to achieve the required lifetime. Between the cores of each benchmark, there is a separation of work based on the sizing. LU cpu0 has a high number of cells that get sized, 55.2% of cells by X2. Core 1 for LU shows a reduced number of total cells sized and none of the cells are sized by X2, relying more on the larger X4 and X8 sizes. Benchmarks like blackscholes and radix do not have any cells resized by X2. For the given time, the paths on the design are degraded more, leading to larger cells.

The impact of adding worst case temperature is presented in Fig. 8. The worst case temperature shows up to a 2.4× increase in X8 cell sizing when compared to the baseline with local temperature. A significant increase of the X8 cell sizing is seen for the worst case temperature across all benchmarks. Over the degradation period, both cores for radix are seen to use X8 for all of the resized cells. Smaller cells for LU are resized to larger gates accounting for additional degradation. An increase in temperature is seen to shift toward the use of larger cell sizes.

Comparing between the local and worst temperature provides insight into the effects of temperature on a benchmark. Cholesky has various sizing levels of X2, X4, and X8 with local temperature. When the worst case temperature is used Cholesky no longer uses X2 cell sizing. An average of 85.4% of the cells used in the worst case are the largest size, leading to power and area overhead so by including temperature, the number of the largest size gates is reduced by 36.3% in the synthesized design.

Increasing the size of gates introduces more power dissipation and, therefore, more heat. The design is simulated in HotSpot postsizing and compared against the design before sizing. However, the increase in temperature is less than 1 °C. The power increases 1.563% on average, which does not affect the gate sizing calculation for lifetime.

D. Scaling the Number of Cores

Scaling the number of cores shows how cell sizing is affected with more options for the distribution of work. Fig. 9 shows the cell sizing for one benchmark, radiosity when scaling from two to eight cores with local temperature. The master core, cpu0, of each design is seen to have more usage than the other worker cores as cpu0 has larger sized cells to meet reliability. Work is distributed evenly to the worker cores for this benchmark and is sized similar to one another. Considering the original motivation, in Fig. 3(c), there is a significant differentiation between instructions executed on master and worker cores. The key reason for this is that Fig. 3 demonstrates the percentage usage of instructions and not the total number of instructions. If the total number of instructions is low enough for certain instruction types, the impact on cell sizing is shown to be small. The impact for radiosity8 is a 7.7%, 2.3%, and 2.5% difference between cores for X2, X4, and X8 sizing, respectively.

E. Selectable Lifetime

Custard is capable of analyzing a design for any selected lifetime constraint. A lifetime sweep is performed on the baseline benchmark and shown in Fig. 10. The baseline benchmark is analyzed from one to ten years for gate sizing. As expected, the sizing to achieve a short lifetime does not

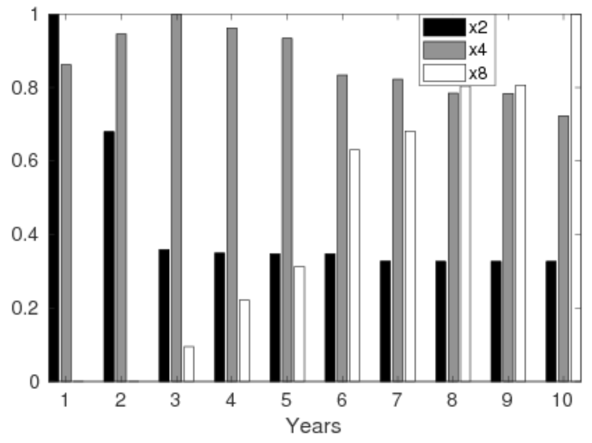


Fig. 10. Normalized number of gates selected over different years for the baseline benchmark.

require large increases in gate size. After the original estimated lifetime of $\approx$2.4 years, X8 gates are starting to be utilized. Larger gates continue to be used as lifetime is increased at the cost of area and power. Given a lifetime constraint to meet ten years, the calculated improvement from the original lifetime is 4.1$\times$.

VI. Conclusion

This paper proposed a framework, Custard, to increase the lifetime of multicore processors by using workload and temperature information in the ASIC design flow. By sizing cores in a processor based on the workload signature, the cores that get used more frequently are sized so that they do not fail (i.e., due to NBTI) as early. Each of the gates has their switching activity evaluated independently, allowing fine-grain analysis of degradation including temperature. Designing the chip to be correct by design allows the reduction of other runtime feedback mechanisms that induce too much overhead in IoT processors. Using this method produces a 4.1$\times$ increase in lifetime compared with the unsized core, and only at a cost of a $<1\%$ increase in area and a $<2\%$ increase in power.

Acknowledgment

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Science Foundation.

References

[1] F. Oboril and M. B. Tahoori, “ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level,” in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2012, pp. 1–12.

[2] M. A. Alam, “A critical examination of the mechanics of dynamic NBTI for PMOSFETs,” in *IEDM Tech. Dig.*, Dec. 2003, pp. 14.4.1–14.4.4.

[3] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “NBTI-aware synthesis of digital circuits,” in *Proc. 44th ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 370–375.

[4] B. C. Paul, K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy, “Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits,” in *Proc. Design, Autom. Test Eur. (DATE)*, vol. 1, Mar. 2006, pp. 1–6.

[5] M. A. Alam, H. Kufluoglu, D. Varghese, and S. Mahapatra, “A comprehensive model for PMOS NBTI degradation: Recent progress,” *Microelectron. Rel.*, vol. 47, no. 6, pp. 853–862, Jun. 2007.

[6] H. Aydin, P. Mejia-Alvarez, D. Mossé, and R. Melhem, “Dynamic and aggressive scheduling techniques for power-aware real-time systems,” in *Proc. 22nd IEEE Real-Time Syst. Symp.,* Washington, DC, USA, Dec. 2001, pp. 95–105.

[7] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, “Power-aware scheduling for periodic real-time tasks,” *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.

[8] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, “A dynamic voltage scaled microprocessor system,” *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[9] D. Ernst *et al.*, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2003, pp. 7–18.

[10] A. Tiwari and J. Torrellas, “Facelift: Hiding and slowing down aging in multicores,” in *Proc. 41st IEEE/ACM Int. Symp. Microarchitecture*, Nov. 2008, pp. 129–140.

[11] Y.-K. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors,” *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.

[12] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, “SWIFT: Software implemented fault tolerance,” in *Proc. Int. Symp. Code Gener. Optim.*, Mar. 2005, pp. 243–254.

[13] P. P. Shirvani, N. R. Saxena, and E. J. McCluskey, “Software-implemented EDAC protection against SEUs,” *IEEE Trans. Rel.*, vol. 49, no. 3, pp. 273–284, Sep. 2000.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[15] W. Wenping, Z. Wei, S. Yang, and Y. Cao, “An efficient method to identify critical gates under circuit aging,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2007, pp. 735–740.

[16] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, “Predictive modeling of the NBTI effect for reliable design,” in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2006, pp. 189–192.

[17] S. P. Park, K. Kang, and K. Roy, “Reliability implications of bias-temperature instability in digital ICs,” *IEEE Design Test Comput.*, vol. 26, no. 6, pp. 8–17, Nov. 2009.

[18] J. Velamala *et al.*, “Logarithmic modeling of BTI under dynamic circuit operation: Static, dynamic and long-term prediction,” in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Apr. 2013, pp. CM.3.1–CM.3.5.

[19] W. Wang, V. Reddy, A. T. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, “Compact modeling and simulation of circuit reliability for 65-nm CMOS technology,” *IEEE Trans. Device Mater. Rel.*, vol. 7, no. 4, pp. 509–517, Dec. 2007.

[20] B. Velamala *et al.*, “Compact modeling of statistical BTI under trapping/detrapping,” *IEEE Trans. Electron Devices*, vol. 60, no. 11, pp. 3645–3654, Nov. 2013.

[21] R. Vattikonda, W. Wang, and Y. Cao, “Modeling and minimization of PMOS NBTI effect for robust nanometer design,” in *Proc. 43rd ACM/IEEE Design Autom. Conf.*, Jun. 2006, pp. 1047–1052.

[22] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao, “The impact of NBTI effect on combinational circuit: Modeling, simulation, and analysis,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 173–183, Feb. 2010.

[23] B. C. Paul, K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy, “Negative bias temperature instability: Estimation and design for improved reliability of nanoscale circuits,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 4, pp. 743–751, Apr. 2007.

[24] *Synopsys IC Compiler User Guide*. Accessed: Nov. 2017. [Online]. Available: http://www.synopsys.com

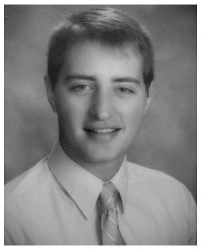
[25] K. Sutaria, A. Ramkumar, R. Zhu, R. Rajveev, Y. Ma, and Y. Cao, “BTI-induced aging under random stress waveforms: Modeling, simulation and silicon validation,” in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[26] S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *Proc. ACM Int. Symp. Comput. Archit. (ISCA)*, 1995, pp. 24–36.

[27] C. Bienia, S. Kumar, J. Singh, and K. Li, “The PARSEC benchmark suite: Characterization and architectural implications,” in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2008, pp. 72–81.

[28] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 24–36, 1995.

[29] S. Lerner and B. Taskin, "Workload-aware ASIC flow for lifetime improvement of multi-core IoT processors," in *Proc. IEEE Int. Symp. Quality Electron. Design*, Mar. 2017, pp. 379–384.
[30] S. Lerner, V. Pano, and B. Taskin, "NoC router lifetime improvement using per-port router utilization," in *Proc. Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.
[31] V. Pano, S. Lerner, I. Yilmaz, M. Lui, and B. Taskin, "Workload-aware routing (WAR) for Network-on-Chip Lifetime Improvement," in *Proc. Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.
[32] V. Huard and M. Denais, "Hole trapping effect on methodology for DC and AC negative bias temperature instability measurements in PMOS transistors," in *Proc. 42nd Annu. IEEE Int. Rel. Phys. Symp.*, Apr. 2004, pp. 40–45.
[33] G. Chen, M. F. Li, C. H. Ang, J. Z. Zheng, and D. L. Kwong, "Dynamic NBTI of p-MOS transistors and its impact on MOSFET scaling," *IEEE Electron Device Lett.*, vol. 23, no. 12, pp. 734–736, Dec. 2002.
[34] Synopsys. (2016). *Synopsys Software*. [Online]. Available: https://solvnet.synopsys.com
[35] J. Balkind *et al.*, "OpenPiton: An open source manycore research framework," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2016, pp. 217–232.
[36] N. Binkert *et al.*, "The Gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.
[37] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.
[38] *OpenSPARC T1*, Oracle, Redwood City, CA, USA, 2007.
[39] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. Tahoori, "Aging-aware logic synthesis," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 61–68.



Scott Lerner (S'09) received the B.S. degrees in electrical and computer engineering from Drexel University, Philadelphia, PA, USA, in 2015, where he is currently working toward the Ph.D. degree in the Electrical Engineering Department.

He was with Advanced Micro Devices, Austin, TX, USA, where he was involved in low-power clock tree synthesis. He was a DRAM Product Engineer at Micron Technologies, Boise, ID, USA. He was an RF Spectrum Denial at Lockheed Martin, Cherry Hill, NJ, USA. His current research interests include low-power design, clock tree synthesis, and design for reliability.

Mr. Lerner received the National Science Foundation Graduate Research Fellowship Program and National Defense Science and Engineering Graduate Fellowship in 2015.

Isikcan Yilmaz (S'16) received the B.S. degree in computer engineering and the M.S. degree in electrical engineering from Drexel University, Philadelphia, PA, USA, in 2016 and 2018, respectively.

He was with Biznet Informatics, Istanbul, Turkey, with a focus on penetration testing, and Beats Electronics, Culver City, CA, USA, as an Embedded Firmware Engineer. His current research interests include embedded systems, thread scheduling, and cache technologies.

Baris Taskin (S'01–M'05–SM'12) received the B.S. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 2000, and the M.S. and Ph.D. degrees in electrical engineering from the University of Pittsburgh, Pittsburgh, PA, USA, in 2003 and 2005, respectively.

In 2005, he joined the Electrical and Computer Engineering Department, Drexel University, Philadelphia, PA, USA, where he is currently a Professor. His current research interests include electronic design automation for VLSI, low-power circuits, resonant clocking, clock network synthesis, hardware/software design space exploration and network-on-chip for chip multiprocessors.

Dr. Taskin was a recipient of a number of awards for his research and professional contributions, including the Association for Computing Machinery Special Interest Group on Design Automation, A. Richard Newton Award in 2007, the National Science Foundation Faculty Early Career Development Award in 2009, an ACM SIGDA Distinguished Service Award in 2012, and the Delaware Valley Young Electrical Engineer of the Year Award from the IEEE Philadelphia Section in 2013.