Systematic Methodology for the Quantitative Analysis of Pipeline-Register Reliability

Reiley Jeyapaul, *Member, IEEE*, Roberto Flores, *Student Member, IEEE*, Alfonso Avila, *Member, IEEE*, and Aviral Shrivastava, *Member, IEEE*

***Abstract*— Decades of rapid aggressive technology scaling have brought the challenge of soft errors to modern computing systems. Sequential elements (registers) in the processor pipeline exposed to charge-carrying particles generate bit flips or soft errors that could translate into system failures. Next to the processor cache, the pipeline registers (PRs)—registers between two pipeline stages—account for more than 50% of soft-error failures in the system. In this paper, for the first time, we apply architectural correct execution models that quantitatively define the vulnerability (or exposure to soft errors) of microarchitectural components, and extend it to define the vulnerability of PRs—PR vulnerability (PRV). We develop gemV-Pipe, a simulation toolset for the systematic, accurate, and quantitative estimation and analysis of PRV. Our detailed ISA-aware analysis in gemV-Pipe reveals interesting facts on the data-access behavior of PRs: 1) the vulnerability of each PR is not proportional to their size; 2) the PR bits used for one instruction may not be used (and are thus not vulnerable) for another, which makes PRV extremely instruction-dependent; and 3) the functionality of stored data on the PR bits can be used to classify them as—instruction, control, and data bits—each of which differ in their instruction-specific behavior and vulnerability. Applying the insight gained, we perform design space exploration on selectively hardening the PR bits, and demonstrate that 75% improved reliability can be achieved for only <15% power overhead.**

***Index Terms*— Fault tolerance, pipeline hardening, pipeline registers (PRs), power efficiency, vulnerability.**

I. Introduction

CONTINUOUS technology scaling provides us with the capability to fabricate complex functionality into smaller processor chips, consuming lower power, and at affordable costs. As a result, the application areas for computing systems have exploded, with them being used in areas not imagined before—medical, automotive, security systems, and in- and out-of-body sensing devices. Researchers indicate that with emerging device technologies in sub-22-nm dimensions, the per-bit soft-error rate (SER) saturates [1], but when a chip is packed with exponentially increasing number of transistors (because of available chip area, we will experience an increase in system-level SER in the not-so-distant future [2]. Soft errors are transient faults that can occur due to one or more of several reasons, such as electrical noise, external interferences, crosstalk, and so on. However, the majority of the soft errors in digital devices happen due to charge-carrying particle strikes on the processor that corrupt its logic value. Such corruption of data used within the processor may lead to system failure. The need for reliable computing, therefore, tends to match (and sometimes supercede) the need for power efficiency and performance [3].

Among the processor components, caches are the most susceptible to soft-error failures, owing to their large area (more than 50% of chip real estate [4]) and highly dense SRAM cell layout. Over the years, several researchers have presented efficient and effective mechanisms (across the design spectrum) to protect the system from soft errors in the caches. The most pressing question then is, if the 50% of chip real estate is protected against soft errors, exactly how reliable is the system? Which of the other processor blocks deserves more attention toward improving system reliability? In our attempt to answer the above questions with certainty, this paper presents for the first time a systematic and quantitative methodology to evaluate pipeline register (PR) reliability. Our experiments show that next to caches, the interface between two pipeline stages—PRs—is the most vulnerable, contributing to more than 57% of soft-error failures in the system with a protected cache.

Mukherjee *et al.* [5] first defined the term vulnerability to quantitatively represent the exposure of architecture components to soft errors. A PR bit is vulnerable (when exposed to soft errors) for the time that it holds data that will be used by the next pipeline stage during its execution. In a system, the sum of all such vulnerable bits defines the total vulnerability of the registers in the system, and therefore, by extension, the sum of all vulnerable bits in the PRs represents PR vulnerability (PRV). In this paper, we develop a two-phase repeatable systematic methodology to quantitatively estimate the PRV of a system. The first phase involves identifying the accurate set of vulnerable PR bits, and the second phase involves computing the time that each of the identified bits is vulnerable during program execution. We integrate our models with gem5 (a mature and very popular cycle-accurate architecture simulator [6]), and develop, gemV-Pipe, a cycle-accurate simulation environment for the quantitative estimation of system PRV.

Since soft errors in PRs significantly affect the reliability of a system, it is essential to analyze the same for possible protection methods. Owing to their high activity factor (read and updated every cycle), the traditional error

Manuscript received September 6, 2015; revised December 29, 2015 and March 7, 2016; accepted April 19, 2016.

R. Jeyapaul is with ARM Research, ARM Ltd., Cambridge GB CB1 9NJ, U.K. (e-mail: reiley.jeyapaul@arm.com).

R. Flores is with Yazaki Services, San Nicolás de los Garza 66470, Mexico (e-mail: rob_flores86@yahoo.com).

A. Avila is with the Tecnologico de Monterrey, Monterrey 64849, Mexico (e-mail: aavila@itesm.mx).

A. Shrivastava is with Arizona State University, Tempe, AZ 85281 USA (e-mail: aviral.shrivastava@asu.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.2016.2574642

detection methods (parity, Error Correction Code, Single Error Correct Double Error Detect, and so on) that involve multicycle operations cannot be applied for pipeline protection. Circuit-level pipeline-hardening techniques have been developed—C-elements [7], [8] and Razor latch [9], [10]—that protect the PRs from latching-on corrupt data (as a result of transient soft errors) or corrupting temporarily stored data. Implementation of such hardening techniques will thus incur significant power overheads—50% for C-elements and 28.5% for RAZOR-II—making them impractical for embedded applications. Power-efficient methods for the protection of the PRs from soft errors are thus the most critical requirement in ensuring system reliability.

Our two-phase systematic methodology to quantitatively estimate PRV and the consequent analysis of the ARM (v7A [11]) processor core revealed interesting insights into the mechanism of soft errors in the processor pipeline: 1) the PRs with varying sizes (number of bits) have varying vulnerabilities, which are not always directly proportional to their size and 2) during program execution, the number of bits (of a PR) used by one instruction differs from that used by another, and therefore, the PRV of each register is instruction-dependent. In this paper, we perform exploratory experiments to analyze the reliability-versus-power tradeoff in the protection afforded by different pipeline protection implementations—C-elements and RAZOR-II. We then propose selective PR protection as a power-efficient methodology to improve system reliability. We demonstrate that by protecting only a subset of the PRs—PC/Fetch, Fetch/Decode, and Decode/Rename—pipeline vulnerability can be reduced by 75%, which incurs only 15% power overhead (over unhardened PRs). And again, through targeted protection of specific bit types (instruction and control bits) in the Fetch/Decode PR, we can achieve 31% improved reliability for less than 3% power overhead.

II. Background

A. Pipeline Register Vulnerability

Mukherjee *et al.* [12] introduced the concept of architecturally correct execution (ACE) to compute the structure's architectural vulnerability factor (AVF). ACE analysis divides a bit's lifetime into ACE and un-ACE intervals. A bit b is considered to be ACE (or vulnerable) at time t, if the data contained in it will be used by the processor for its execution (at any later time). In this paper, we extend this ACE analysis to define the vulnerability of a PR bit at a more finer granularity when compared with the block-level granularity of ACE analysis in the AVF-based estimations. A PR is nothing but a set of flip-flops (sequential elements) that temporarily hold data between two pipeline stages [Fig. 1(a)]. In a typical execution model, if the data are written into a PR during clock cycle $c1$, it will be read during clock cycle $c2 = c1 + 1$. Fig. 1(b) describes the mechanism of data storage in the PR1. The solid bar under the timeline indicates its vulnerable duration (or the time duration for which the PR bit is recognized as ACE). For data to be stored on a PR1, the data have to arrive from Stage 1 at the register before its hold time, and should remain stable until the end

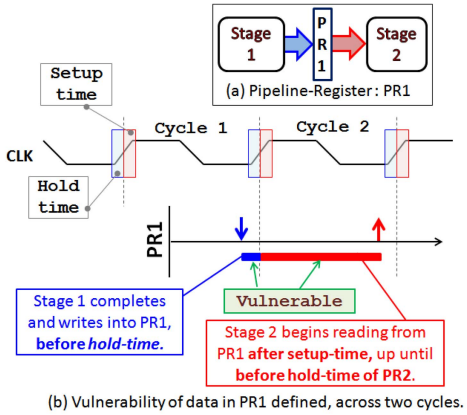

Fig. 1. (a) Data in PR1 is vulnerable from the time the register is written (after the execution of Stage 1) to the time the data on the register can be used for the execution of the instruction on pipeline Stage 2. (b) Vulnerability of the PR is defined over two clock cycles—Cycles 1 and 2—during the execution of an instruction in Stages 1 and 2, respectively.

of its setup time. The data stored in the register (PR1) can be read and used for execution in the succeeding pipeline stage (Stage 2) immediately after the clock edge, up until the time that the succeeding PR2 is updated. Here, the data on PR1 is vulnerable for the entire duration that useful data are stored that can be written into the succeeding PR or used by Stage 2 for execution. In the case of pipeline stalls, this vulnerable duration is extended for the period of consecutive stalls during execution. We implement this vulnerability estimation model in our gemV-Pipe toolset to compute the vulnerability of the PRs in the processor.

B. Related Work

Wang *et al.* [13] develop a fault-injection-based methodology to perform detailed ACE analysis of the processor core, and determine a conservative estimate of processor reliability. Online real-time AVF estimation methods allow for more elaborate ACE analysis of the processor core [14]. This methodology involves fault injections and algorithms to analyze the error propagation within the core to estimate system failure rate. However, the accuracy of this estimate is limited to the number of fault injection runs that can be analyzed in a reasonable amount of time.

Biswas *et al.* [15] in their work perform detailed ACE analysis on the lifetime of bits in address-based structures (e.g., data cache, data/instruction Translation Lookaside/Lookahead Buffer, and store buffer), accurately model vulnerability of data stored on the structures, and then compute their AVF through cycle-accurate simulations. It is evident that the use of analytic models for estimating system reliability at the microarchitecture-level is more efficient. However, the significance and/or accuracy of such results are dependent entirely on the accuracy and validity of the models developed. The first phase in our modeling methodology, to determine vulnerable bits in the PR, is obtained as a result of systematic register transfer level (RTL) analysis and exhaustive fault-injection-based validation, attesting to the accuracy of the models implemented. Mukherjee *et al.* [12] in their work present the methods to use ACE modeling to estimate the AVF (application-dependent evaluation) of the various processor

structures, such as instruction queue, load-store queue, and so on. Like in Mukherjee's work, in our system-level PRV estimation methodology using gemV-Pipe, we develop detailed Instruction Set Architecture (ISA)-aware ACE models for the PR bits, and integrate the same in a cycle-accurate simulation infrastructure to consider program code behavior.

With the help of predictive modeling and using performance metrics (from cycle-accurate simulations), Duan *et al.* [16] develop a method for fast AVF prediction of a processor. In this, the costly exercise of implementing AVF estimation models in an architecture simulator is avoided by utilizing statistical postsimulation estimates to predict system AVF. However, this method attempts to abstract-away many of the intricate data-access patterns on the architecture components (e.g., PRs, caches, buffers, and so on), thus lacking in accuracy and will suffer from a significant error margin. Sridharan and Kaeli [17] develop a mechanism to quantify system vulnerability independent of the hardware architecture, and thus provide a means to estimate software vulnerability. In this, the authors adapt ACE analysis to develop a software-level vulnerability metric—program vulnerability factor (PVF)—which defines the vulnerability of the program as an independent entity. On similar lines, researchers develop thread vulnerability factor to define the vulnerability of multithreaded programs [18]. Though such methods succeed in the fast estimation of system reliability through abstractions at the program and thread level, they are limited by their accuracy and abstracted models used. System-level vulnerability estimation at higher levels of abstraction can become less accurate owing to the lack of details and accuracy in the models. However, the analytic models implemented in gemV-Pipe include the RTL (bit-level) accuracy of PRV estimation, in conjunction with the software interface.

Azarpeyvand *et al.* [19] introduce the metric instruction vulnerability factor to quantify software vulnerability (extending the PVF metric) by also considering the microarchitecture details of the processor. In this, a custom fault-injection framework is used to analyze the impact of transient faults on the instruction set. Since fault-injection methods are used for estimation, this method still is limited by the number and accuracy of faults injected. Rehman *et al.* [20] present instruction vulnerability index (IVI), an instruction-level reliability estimation technique that quantifies the effects of hardware faults at the instruction level. The effect of transient faults on the instructions at different stages of its execution (i.e., erroneous instruction-fetch, load-address, store-address) is analyzed, and the IVI of each instruction is determined. It should be noted here that IVI estimates only consider the vulnerability of the data in the register file, instruction queue, Load Store Queue, and other storage buffers on the pipeline, and do not consider the vulnerability of the PRs. To the best of our knowledge, we present for the first time a systematic repeatable methodology for the accurate quantitative estimation and analysis of PRV.

III. Quantitative Estimation of Pipeline Register Vulnerability

At the interface of two pipeline stages, the output data from the execution of one stage are temporarily stored into flip-flops (registers), before being consumed in the subsequent clock cycle. For instance, in an R-type `add` instruction, only the *Rs*, *Rt*, and *Rd* bits are vulnerable. On the other hand, for an I-type `addi` instruction, the *Rd* bits are not vulnerable, while the *Imm* bits are vulnerable. With this valuable insight that the bits in a PR vary in their vulnerability based on the instruction executed, in this paper, we develop a two-phase ISA-aware systematic methodology:

1) ISA-aware analysis identifying the set of vulnerable bits in each PR through: a) RTL-level analysis of the processor core and b) validation using our efficient fault-injection-based analysis methodology;
2) computing PRV of an application through our cycle-accurate simulation environment, gemV-Pipe, where the results of our ISA-aware analysis on the pipeline is integrated into our vulnerability estimation model.

A. Phase 1: ISA-Aware Analysis of Vulnerable Pipeline-Register Bits

By analyzing the RTL of a given processor core, the exact set of bits that are vulnerable (or, may affect the system when corrupted by soft errors) can be identified by careful and detailed ISA-aware analysis of each PR. In this, we also consider the cases when different control-bit patterns of conditional instructions (of the ARM ISA) are also analyzed for their effect on PRV—across different input patterns. The steps followed in our analysis are given in the following.

1) Identify Pipeline Registers: From the processor's RTL code (in verilog or VHDL), the set of registers that will be synthesized into hardware can be identified by applying behavioral analysis on the RTL [21]. In this, the RTL code is analyzed for behavioral patterns, and the latches and/or registers inferred can be identified for each pipeline stage. As a first step, for the given pipeline structure, we isolate the PRs—$\{\mathrm{PR}^{s/s+1}\}$—at the interface between two pipeline stages (s and $s+1$). For example, the PR bits in the previous example are

$$\mathrm{PR}^{D/X} = \{\mathrm{Rs}, \mathrm{Rt}, \mathrm{Rd}, \mathrm{Imm}\}. \tag{1}$$

2) Annotate Active Pipeline Registers: Among the many hardware bits (flip-flops) that constitute a PR between two pipeline stages, only a subset of them is vulnerable when an instruction is executed. The vulnerable PR bits in $\mathrm{PR}^{s/s+1}$ is dependent on the functionality executed in the trailing pipeline stage ($s+1$). In other words, at the interface of two pipeline stages (s and $s+1$), only a subset of the PR bits will be actively used in the execution of the instruction and are therefore considered vulnerable. This analysis is performed at the RTL level through behavioral analysis of the pipeline stage for each instruction ($I \in$ ISA) in the ISA. The result is an annotated set of active PRs ($\{a\mathrm{PR}_I^{s/s+1}\}$), denoting the PR-bits that are involved in the execution of instruction I in the pipeline stage—$s+1$. For instance, in Fig. 2, the PR bits actively involved in the execution of the `add` and `addi` instructions are

$$a\mathrm{PR}_{\mathrm{add}}^{D/X} = \{\mathrm{Rs}, \mathrm{Rt}, \mathrm{Rd}\}$$
$$a\mathrm{PR}_{\mathrm{addi}}^{D/X} = \{\mathrm{Rs}, \mathrm{Rt}, \mathrm{Imm}\}.$$

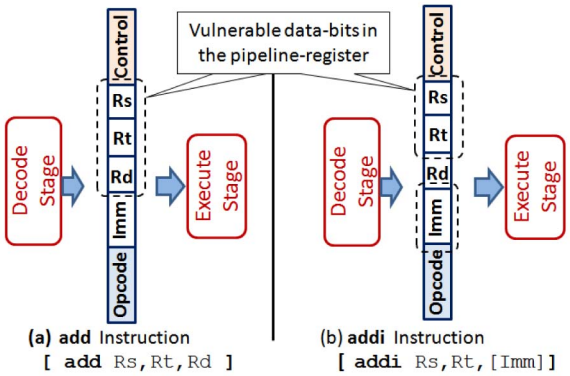

Fig. 2. Difference in the number of vulnerable register bits (in the Decode/ Execute PR), for the two variants of the add instruction.

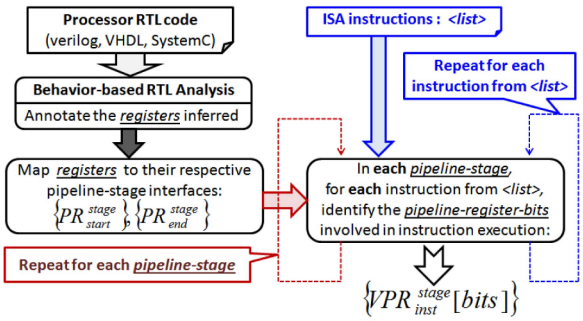

Fig. 3. Flowchart defining our systematic methodology to identify the ISA-specific vulnerable PRs—V $\text{PR}_{\text{inst}}^{\text{stage}}$—through ISA-aware RTL analysis, for each instruction in the ISA across the processor's pipeline stages.

This process (as described in Fig. 3) is repeated for every instruction in the ISA ($I \in$ ISA) for each pipeline stage ($s \in$ Stages, where Stages denotes the set of pipeline stages in the core).

3) Validation Through Fault-Injection Analysis: The result of our ISA-aware RTL analysis is the set $\{a\text{PR}_I^{s/s+1}, \forall s \in \text{Stages}, \forall I \in \text{ISA}\}$. Not all the $a\text{PR}_I^{s/s+1}$[bits] identified through our ISA-aware RTL analysis are actually vulnerable during program execution. A bit $b \in a\text{PR}_I^{s/s+1}$ can be considered vulnerable if and only if a soft error on the bit b will disrupt the execution, and/or cause an aberration in the output of the pipeline stage s, when instruction I is executed. We perform targeted fault-injection analysis on the annotated active PR bits—$b \in a\text{PR}_I^{s/s+1}$—to verify their vulnerability. Our validation process involves the following steps (described in the flowchart Fig. 4).

1) An instruction $I \in$ ISA and a PR at the interface between two pipeline stages—s and $s+1 \in$ Stages—are first considered for experimentation.
2) In an RTL simulation environment (implemented using Modelsim [22] and Verilog VPI [23]), a fault is injected into one bit $b \in a\text{PR}_I^{s/s+1}$ at time-cycle t.
3) At time-cycle $t+1$, if the execution of the stage s is not disrupted, and if the data stored into $\text{PR}_I^{s+1/s+2}$ match with that of the golden copy, then we consider the bit $b \in \text{PR}^{s/s+1}$ not vulnerable when instruction I is executed.

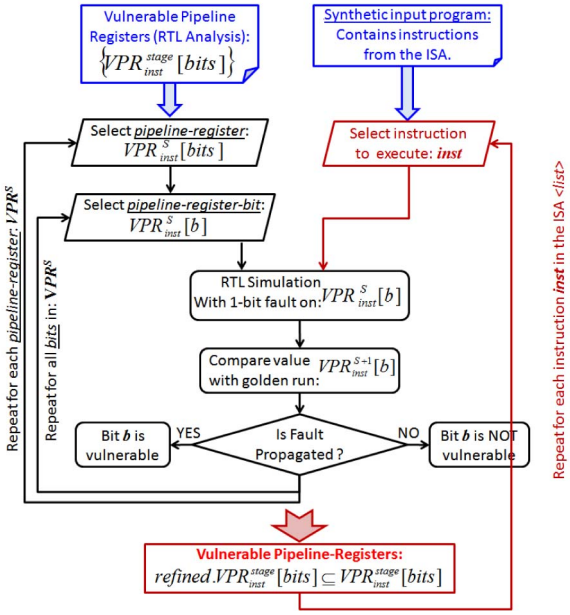

Fig. 4. Flowchart defining our validation process that refines and also validates the ISA-specific vulnerable PRs to obtain—*refined*.$\text{VPR}_{\text{inst}}^{\text{stage}}$—through detailed and ISA-specific fault-injection analysis.

The above steps are repeated for every bit $b \in \{a\text{PR}_I^{s/s+1}, \forall s \in \text{Stages}, \forall I \in \text{ISA}\}$ to finally obtain the validated set of vulnerable PRs—$\{V\text{PR}_I^{s/s+1} \subset a\text{PR}_I^{s/s+1}\}, \forall s \in \text{Stages}, \forall I \in \text{ISA}$.

B. Phase 2: Computing Pipeline Register Vulnerability

To quantitatively estimate the reliability of the processor pipeline, when a program is executed, we implement PRV modeling in a cycle-accurate simulation environment—gem5 [6]. The key component in our vulnerability modeling and analysis is the fact that the vulnerability of each PR is strictly bound to the instruction executed. For this purpose, we implement the access tracker, which functions as a wrapper around each PR to monitor their accesses. This in turn is used to compute their respective vulnerability values (defined as $\text{vul.PR}_{I \in ISA}^{s/s+1}$ in Fig. 5). Another feature of the access tracker is that the computed values are associated with the respective instruction in-flight and stored in a temporary list. When the instructions are committed (at the writeback stage), their PR vulnerabilities are accumulated to add to the total system PRV. For example, in Fig. 5, the PRV_{bne} is the sum of all the $\text{vul.PR}_{\text{bne}}^{s/s+1}$ values computed. When an instruction is `squashed` because of system behavior (exceptions, interrupts, and speculation), their respective vulnerability values computed are discarded. For example, in Fig. 5, the $\text{vul.PR}_{\text{add}}^{F/D}$ and $\text{vul.PR}_{\text{add}}^{D/X}$ values are discarded.

IV. Systematic Analysis of Pipeline Register Vulnerability

A. Experimental Method

To compute the effective pipeline vulnerability (say V_e in bit cycles), the experimental methodology involves the following.

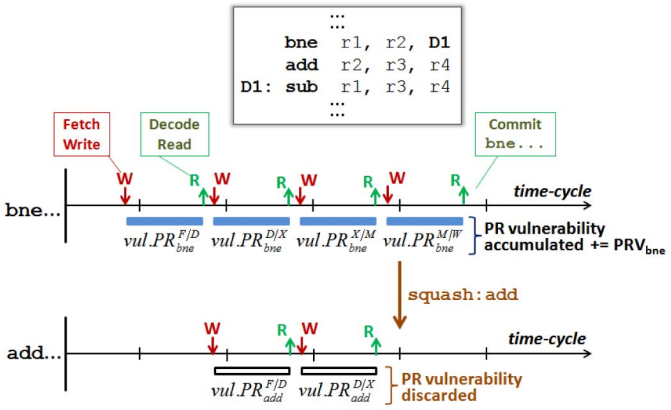

Fig. 5. PRV computation for the committed `bne` instruction and squashed `add` instruction from the code snippet is described. Their vulnerability is computed at every PR for the instruction executed ($PR_I^{s/s+1}$), and is accumulated to account for the system's PRV for the instruction executed. In the case when an instruction is `squash`'d, the computed PRV values are deleted, and will not be included.

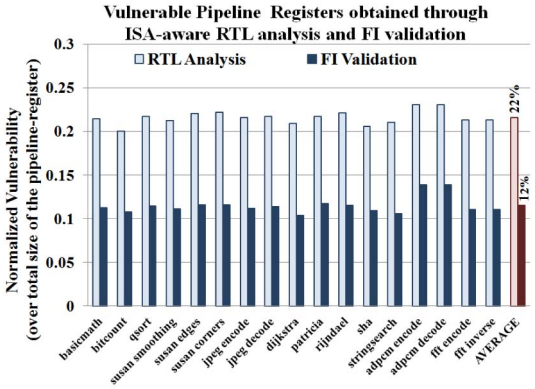

Fig. 6. Vulnerability of the PRs (normalized over the total size of the PR) identified through the two analysis methods (ISA-aware RTL analysis and fault-injection-based validation) is plotted across benchmarks.

1) Analyze the RTL of the AMBER (ARM-v2A) core obtained from OpenCores [24] to obtain detailed bit-level list of the PRs (Fig. 6).
2) Model the PRs in the ARM O3-CPU model of the gem5 [6] cycle-accurate simulator. For this, we utilize the time buffers (in the gem5 infrastructure) to model the data transfer between pipeline stages, and also model the number of hardware bits[1] involved in such data transfers.
3) Incorporate our vulnerability models into the PRs modeled in gem5, modeling an ARM processor with the cache of size 64 kB two-way set associative.

B. PR Vulnerability Is Dependent on the Instructions Executed

From our experiments, we observe that the effective vulnerability (V_e) of the processor pipeline is only 12% that of V_o (without ISA-aware models). Furthermore, we observe that only 30% of the PR bits are actually vulnerable and exposed to soft errors at any given instant during program execution (across MiBench benchmarks). Therefore, as a result of our ISA-aware vulnerability analysis and modeling, we have uncovered abundant potential to achieve highly power-efficient protection of the PR bits.

[1]Since the ARM architecture modeled in gem5 is that of the ARM-v7A architecture, while that of the RTL (freely available) is ARM-v2, we extrapolate (with appropriate architectural correction) the bit-level PR details obtained from our RTL analysis.

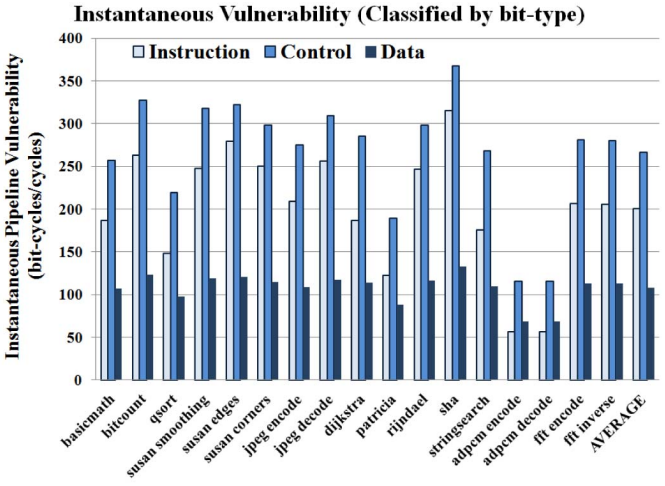

Fig. 7. Instantaneous vulnerability (average number of bits vulnerable at any instant/cycle of the program execution) of the processor pipeline classified based on the type of bits (instruction, control, and data bits) in each PR is plotted across benchmarks.

Fig. 7 shows the vulnerability statistics of the processor pipeline based on bit type (instruction, control, and data bits). In the pipeline, a breakdown of PR bits into type shows that 56% is data, 19% instruction, and 25% control bits. However, our ISA-aware RTL analysis (average across all instructions in the ISA), we observe a significant difference in their relative contributions to pipeline vulnerability—30% data bits, 48% control bits, and 22% instruction bits. This can be attributed to the code behavior, data-access pattern, and instruction composition of the applications involved.

Another interesting finding from this experiment is that the data bits are less vulnerable than control and instruction bits. Though the number of data bits (in hardware) is large, only a subset of the bits will be used for computation for any particular instruction, which depends on the type of instruction executed. For instance, a 4-byte data word when accessed by an arithmetic instruction (like `add`) will only have one byte as vulnerable for an instruction that only operates on one byte (byte-operations in ARM ISA). This instruction-specific vulnerability behavior of the bit types in the PRs motivates for the case that certain specific bits (based on type) in a PR can be protected for power-efficient pipeline protection.

C. Vulnerability Increase of Multi-Issue Pipelines

The ARM-v7-A processor has a total of 1890 PR bits, and among them only around 532 bits are vulnerable on average across all instruction types. Fig. 8 shows the instantaneous vulnerability (number of vulnerable PRs per cycle) of the benchmarks, and compares the default pipeline with that for

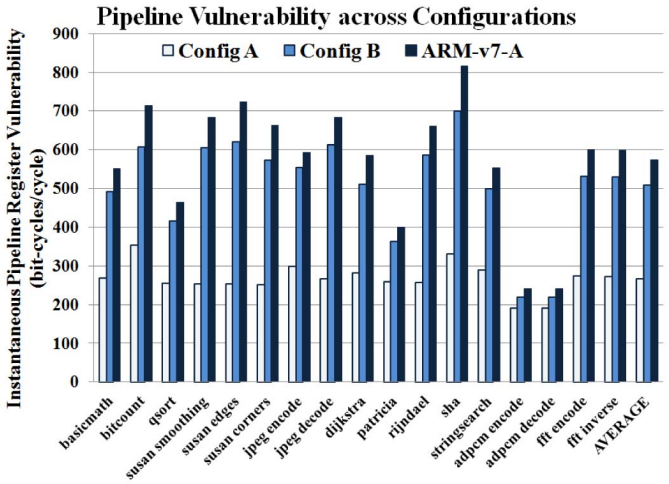

Fig. 8. Average instantaneous vulnerability (in bits) of the processor pipeline, obtained by the ratio of total vulnerability and program runtime, is plotted across benchmarks for three varying pipeline configurations. Based on the pattern and type of instructions that form a program and together with the nature of their execution on the pipeline, their respective instantaneous vulnerabilities vary. (Config A: Each pipeline stage has the width = 1, simulating an in-order pipeline. Config B: Each pipeline stages has half the width of the default ARM-v7-A configuration.)

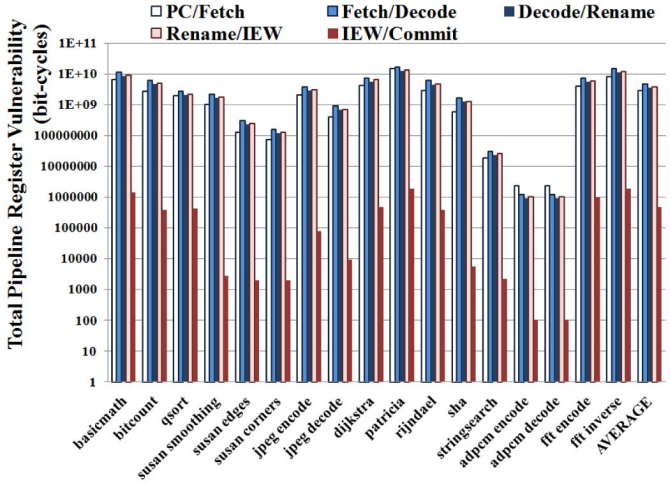

Fig. 9. Total vulnerability of the PRs, at the interface of each of the pipeline stages, is plotted for the ARM-v7-A processor architecture (in logarithmic scale). Across benchmarks, the *IEW/Commit* register has the least vulnerability, while the other four registers have almost the same vulnerability contribution.

two variant pipeline configurations of the ARM processor core. During program execution, on a multi-issue out-of-order processor architecture, during pipeline stalls the data temporarily stored on all the PRs remain vulnerable throughout the stalled period, which accumulates to form the total pipeline vulnerability (in bit cycles) of the system. In the case of Config A where each of the PRs have the widths of 1, the configuration resembles that of an in-order pipeline. In this, the pipeline is stalled in the case of data hazards, control hazards, and memory latencies, and therefore, contribute to total PRV and also the runtime of the application. On the other hand, in the case of Config B and ARM-v7-A (which are out-of-order multi-issue pipeline configurations), the program suffers lesser stalls with increased pipeline widths and, therefore, lower instantaneous pipeline vulnerability. The need for protection techniques for the PRs is therefore evident for modern high-end computing systems.

D. Vulnerability Is Not Always Directly Proportional to PR Size

Analysis of PRV using our systematic methodology reveals that the vulnerable bits in each of the five registers are not the same, and neither are their respective vulnerability values (Fig. 9). For instance, the *Rename/IEW* PR has around 15× more vulnerable bits than the *IEW/Commit* PR, which accounts for the extremely low vulnerability on the *IEW/Commit* PR across benchmarks. The average vulnerable bits in the *Fetch/Decode* (278) and *Decode/Rename* (297) PRs though are approximately equal, their respective vulnerability values computed through simulations differ by around 30%. During program execution, based on the instruction executed, the data storage and access pattern on the registers differ and thus reveal that the *Fetch/Decode* PR is actually the most vulnerable register. The dominant reason for the relatively higher vulnerability of the *Fetch/Decode* PR over the other PRs is because the size of the *Fetch/Decode* PR is relatively small, while most of the bits occupying the *Fetch/Decode* PRs are useful and consumed by the subsequent stages in the pipeline and therefore vulnerable to soft error. In addition, it should be noted here that the number of vulnerable bits in each PR differs for the type of instruction executed. Since the number of vulnerable bits in the *Fetch/Decode* register for ALU and memory instructions is far greater than that for control instructions, the *MiBench* benchmark applications being predominantly either data intensive or compute-intensive demonstrate an increased vulnerability on this PR. When an instruction is processed and data bits in the pipeline populated, an error in any one bit at a particular PR could alter the execution of the instruction in the succeeding stages based on how the data bits are interpreted. Therefore, such ISA-aware analysis is of paramount importance in system reliability estimation. Through instruction-specific analysis of the PRs, we can determine that power-efficient protection could be achieved by efficiently protecting only specific PRs.

V. Power-Efficient Protection of the Pipeline Registers

To protect the vulnerable data as it is stored on the PRs, researchers have developed techniques at the device-level [25], circuit level (C-elements [7], [8]) and microarchitecture level (RAZOR II [9], [10]). The most effective and power-efficient techniques for pipeline hardening among them are C-elements and RAZOR II. In their implementations at the hardware level, every single PR bit is considered vulnerable, and therefore is protected with pipeline hardening. In doing this, all the vulnerability in the processor pipeline is protected, but at the cost of significant pipeline-power overheads around 28.5%∼50%. The power overhead numbers are obtained from the respective publications, which is used in our experimental setup.

Such overheads, and also the additional hardware area, and design cost render their use in modern embedded systems impractical. Here, we use the results from our quantitative analysis of the processor pipeline (Fig. 9), and we explore the power-reliability tradeoff when protecting only a subset of the PR bits.

Applying the results of our detailed analysis experiments on the vulnerability behavior of the processor pipeline, we perform experiments to protect the processor pipeline with two of the most popular circuit-level PR protection techniques: 1) SIL (C-element-based protection) [7] and 2) RazorII [10]. From the quantitative analysis experiments, we take the two key learnings.

1) All the PRs do not have uniform vulnerability behavior (Fig. 9). Each PR has a different size, and the number of bits that are vulnerable depends on the type of instruction executed on the pipeline stage interfacing with the particular PR.
2) In each PR, the type of bits (instruction, data, or control) varies in their vulnerability. Their vulnerability behavior varies based on their functionality and significance in the execution of instructions. The control bits have been observed to contribute the most to pipeline vulnerability, followed by the instruction bits. The data bits, which are larger in size, contribute the least to pipeline vulnerability (Fig. 7).

A. Selective Protection of Pipeline Registers

Incorporating our first learning from our quantitative analysis, we perform a wide range of experiments that explore the possibility of achieving power-efficient pipeline protection through the selective protection of PRs. In our experiments, while we consider configurations with partially protected PRs, we record the power overheads incurred by the implementation of C-elements (50% power overhead [7]) and RazorII (28.5% power overhead [10]) pipeline protection techniques.

Fig. 10 shows the average (across benchmarks) effective vulnerability achieved after protection of each of the PRs individually. We observe two interesting phenomenon.

1) By protecting the *Fetch/Decode* PR, we can achieve 31% reduction in pipeline vulnerability for a low cost of only 5% power overhead. This behavior is due to the fact that of the 346 bits in the *Fetch/Decode* PR, around 30% of the bits are vulnerable for all type of instructions in the ISA. In addition, vulnerability analysis across benchmarks reveals that owing to the access pattern of data stored on the *Fetch/Decode* PR, it is the highest contributor to total pipeline vulnerability (see Fig. 9).
2) By protecting either the *Rename/IEW* or *Decode/Rename* PRs, we achieve around 25% reduction in pipeline vulnerability, while they differ greatly around two times in power overhead. The power overhead can be attributed to the fact that the *Rename/IEW* register has a size of 822 bits, while the *Decode/Rename* register is only around half of that with similar difference in the number of vulnerable bits across ISA instructions.

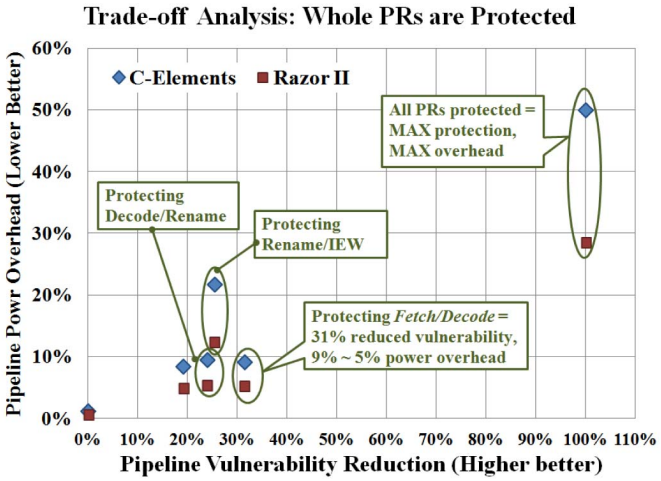

Fig. 10. Tradeoff graph exploring the impact of vulnerability protection achieved through the protection of the PRs individually. When protecting the *IEW/Commit* PR, ∼0% vulnerability reduction is achieved for negligible power overhead, while protecting the *Fetch/Decode* PR achieves 31% vulnerability savings for only ∼5% power overhead.

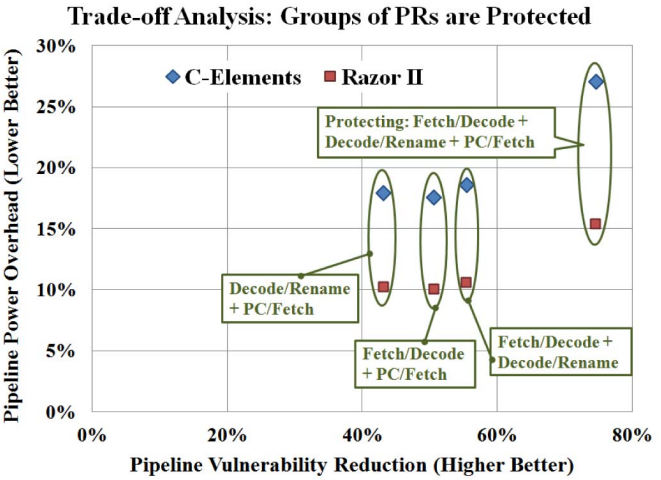

Fig. 11. Reliability-power tradeoff analysis when the combinations of PRs are protected together. By protecting the three top individual PRs, around 55% vulnerability reduction is achieved for around 11% power overhead, while groups of two also perform with comparable power efficiency.

The behavior of data stored on the two PRs indicates that the *Decode/Rename* holds the data for longer periods of time on the PR than the *Rename/IEW*. This attributes to their almost similar vulnerability behavior during application execution (see Fig. 9), which in-turn results in similar vulnerability savings upon protection.

To further explore power-efficient protection opportunities, we perform experiments across benchmarks, protecting multiple number of PRs. Fig. 11 shows the average power overheads and vulnerability reduction achieved through such protection configurations. We observe here that by combining the three top contenders for individual PR protection (*Fetch/Decode*, *Decode/Rename*, and *PC/Fetch*), we can achieve significant vulnerability reduction (75%) with only 15% cost in power overhead. On the other hand, by combining these registers in groups of two, vulnerability reduction of around 50% is

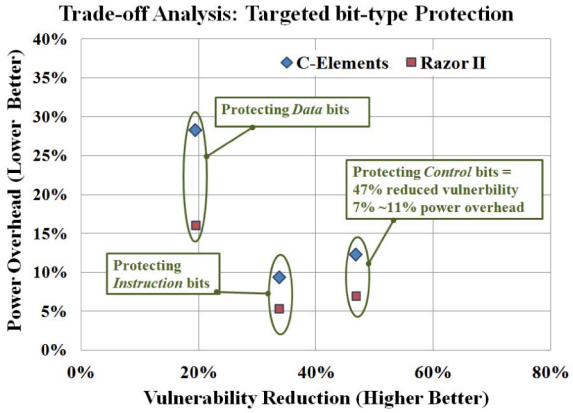

Fig. 12. Reliability-power tradeoff analysis when only a subset of bits in all the PRs is protected based on the bit type (instruction, control, or data). Though large in size, protecting the data bits is inefficient, while protecting the control bits proves more power-efficient.

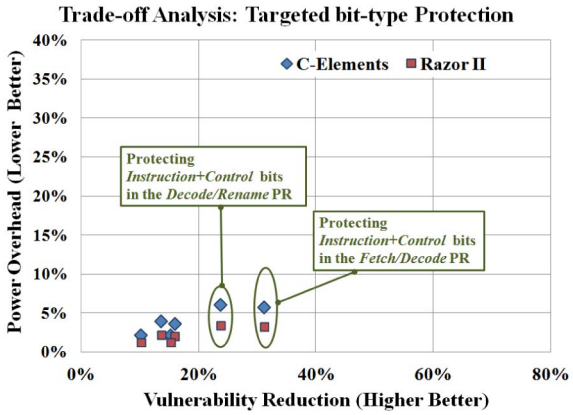

Fig. 13. Reliability-power tradeoff analysis when only a subset of bits in specific PRs is protected based on the bit type (instruction, control, or data). Protecting the instruction and control bits of the *Fetch/Decode* and *Decode/Rename* PRs proves power-efficient.

achieved for only around 10% power overhead. This only proves to show that, based on the system requirements for power and reliability, selective protection of PRs can achieve significant vulnerability reduction for small cost in power overhead.

B. Targeted Type-Specific Protection of Pipeline Registers

Incorporating our second learning from our quantitative analysis, we perform a wide range of experiments to analyze the reliability-power tradeoff achieved through targeted type-specific protection of PR bits. Our experiments (Fig. 12) show that by protecting only the control bits in all the PRs, we achieve 47% reduction in vulnerability with a cost of only around 7% in power overhead. Similarly, by protecting the next major vulnerability contributor (instruction bits), we achieve around 35% improved reliability, for only around 5% power cost. This experiment reveals that in a PR by selectively protecting the most vulnerable bit type, power savings can be improved for comparable improvement in reliability. Fig. 13 shows that through selectively protecting only the instruction and control bits of the *Fetch/Decode* and *Decode/Rename* PRs, significant vulnerability reduction (around 35%) can be achieved for negligible cost in power consumption (around 3%).

VI. Cost-Effective ISA-Aware RTL Analysis

In the design of soft-error protection for the processor pipeline, the key questions that require answers are as follows.

1) What is the quantitative contribution of a PR bit to the total pipeline reliability, and thereby system reliability?
2) Is there a systematic methodology for reliability-versus-power tradeoff analysis?

In this paper, we demonstrate that through our ISA-aware RTL analysis and vulnerability modeling on a cycle-accurate simulator—gemV-Pipeline—benchmarking and domain specific analysis can be performed to determine quantitatively the vulnerability contribution by each PR bit. In attempting reliability-power tradeoff analysis, selected PRs or specific PR bits can be modeled as protected (in our gemV-pipeline toolset) and thereby compute the effective pipeline vulnerability.

VII. Conclusion

As device dimensions shrink rapidly, embedded processors become more vulnerable to failures due to soft errors. Among the architecture components, PRs are the most vulnerable next only to the cache blocks. Researchers have developed many power-efficient methods to protect the caches from soft errors, but very little work has been done to protect the PRs. Circuit-level hardening techniques (C-elements and RAZOR latches) have been developed to protect the PRs, but do so with significant power overheads. We develop here, for the first time, a two-phase systematic methodology for the quantitative analysis of PR reliability (in the presence of soft errors). We implement the same and develop—gemV-Pipeline toolset—for simulation-based quantitative estimation of PRV. As part of our two-phase systematic methodology, we validate the set of vulnerable PR bits as identified through ISA-aware RTL analysis. Our gemV-Pipeline toolset can thus be used for cost-effective analysis of PRV: 1) by protecting only a subset of the PRs, reliability can be improved by 75% for less than 15% power overhead and 2) by protecting only the instruction and control bits on specific PRs, for only around 3% pipeline-power overhead reliability of the pipeline can be reduced by 31%.

References

[1] H. Liu, M. Cotter, S. Datta, and V. Narayanan, "Soft-error performance evaluation on emerging low power devices," *IEEE Trans. Device Mater. Rel.*, vol. 14, no. 2, pp. 732–741, Jun. 2014.

[2] G. Hubert, L. Artola, and D. Regis, "Impact of scaling on the soft error sensitivity of bulk, FDSOI and FinFET technologies due to atmospheric radiation," *Integr., VLSI J.*, vol. 50, pp. 39–47, Jun. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926015000048

[3] T. Granlund, B. Granbom, and N. Olsson, "Soft error rate increase for new generations of SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 6, pp. 2065–2068, Dec. 2003.

[4] N. Muralimanohar, "Wire aware cache architecture," Ph.D. dissertation, School Comput., Univ. Utah, Salt Lake City, UT, USA, 2009.

[5] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "Measuring architectural vulnerability factors," *IEEE Micro*, vol. 23, no. 6, pp. 70–75, Nov. 2003.

[6] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[7] B. Vaidyanathan, Y. Xie, N. Vijaykrishnan, and H. Zheng, "Soft error analysis and optimizations of C-elements in asynchronous circuits," in *Proc. 2nd Workshop Syst. Effects Logic Soft Errors*, 2006, pp. 1–4.

[8] M. Zhang *et al.*, "Sequential element design with built-in soft error resilience," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 12, pp. 1368–1378, Dec. 2006.

[9] D. Blaauw *et al.*, "Razor II: *In situ* error detection and correction for PVT and SER tolerance," in *IEEE Int. Solid-State Circuits Conf. (ISSCC), Dig. Tech. Papers*, Feb. 2008, pp. 400–622.

[10] S. Das *et al.*, "Razor II: *In situ* error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.

[11] ARM Inc. (2008). *Arm Architecture Reference Manual ARMv7-a*. [Online]. Available: https://silver.arm.com/download/download.tm?pv=1603196

[12] 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 *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Washington, DC, USA, Dec. 2003, pp. 29–40. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956570

[13] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *Proc. 34th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2007, pp. 460–469. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250719

[14] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online estimation of architectural vulnerability factor for soft errors," in *Proc. 35th Int. Symp. Comput. Archit. (ISCA)*, Jun. 2008, pp. 341–352.

[15] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proc. 32nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2005, pp. 532–543.

[16] L. Duan, B. Li, and L. Peng, "Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2009, pp. 129–140.

[17] V. Sridharan and D. R. Kaeli, "Quantifying software vulnerability," in *Proc. Workshop Radiat. Effects Fault Tolerance Nanometer Technol. (WREFT)*, New York, NY, USA, 2008, pp. 323–328. [Online]. Available: http://doi.acm.org/10.1145/1366224.1366225

[18] I. Oz, H. R. Topcuoglu, M. Kandemir, and O. Tosun, "Thread vulnerability in parallel applications," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1171–1185, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2012.05.002

[19] A. Azarpeyvand, M. E. Salehi, S. M. Fakhraie, and S. Safari, "Fast and accurate architectural vulnerability analysis for embedded processors using instruction vulnerability factor," *Microprocessors Microsyst.*, vol. 42, pp. 113–126, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933116000247

[20] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, New York, NY, USA, Oct. 2011, pp. 237–246. [Online]. Available: http://doi.acm.org/10.1145/2039370.2039408

[21] Y.-C. Hsu, B. Tabbara, Y.-A. Chen, and F. Tsai, "Advanced techniques for RTL debugging," in *Proc. 40th Annu. Design Autom. Conf. (DAC)*, New York, NY, USA, 2003, pp. 362–367. [Online]. Available: http://doi.acm.org/10.1145/775832.775927

[22] Mentor Graphics. (2013). *ModelSim—Leading Simulation and Debugging*. [Online]. Available: http://www.mentor.com/products/fpga/model

[23] S. Williams. (2013). *Icarus Verilog*. [Online]. Available: http://iverilog.icarus.com/

[24] C. Santifort. (2013). *Amber ARM-Compatible Core*. [Online]. Available: http://opencores.org/project,amber

[25] A. Chavan, E. MacDonald, J. Neff, and E. Bozeman, "Radiation hardened Flip-Flop design for super and sub threshold voltage operation," in *Proc. IEEE Aerosp. Conf.*, Mar. 2011, pp. 1–6.

Reiley Jeyapaul (M'07) received the bachelor's degree from the University of Madras, Chennai, India, and the master's degree in electrical engineering and the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA.

He is currently a Senior Research Engineer with ARM Research, Cambridge, U.K. His current research interests include the compiler microarchitecture interface for embedded and multicore systems, with the goal of improving its reliability, robustness, and power efficiency.

Roberto Flores received the B.S. degree in electronic technologies engineering and the M.S. degree in electronic systems from the Tecnológico de Monterrey, Monterrey, Mexico, in 2010 and 2013, respectively.

He has been with Yazaki Services as an HMI Engineer since 2014. His current research interests include computer architecture, embedded systems, and automotive.

Alfonso Avila (M'06) received the B.S. degree in electrical engineering from the Tecnológico de Monterrey, Monterrey, Mexico, in 1989, and the M.S. and Ph.D. degrees in computer engineering from the University of Arkansas, Fayetteville, AR, USA, in 1994 and 1997, respectively.

He is currently a Professor with the Electrical and Computer Engineering Department, Tecnológico de Monterrey. His current research interests include computer architecture, embedded systems, and tele-health.

Aviral Shrivastava (M'02) received the bachelor's degree in computer science and engineering from IIT Delhi, New Delhi, India, and the master's and Ph.D. degrees in information and computer science from the University of California at Irvine, Irvine, CA, USA.

He is currently an Associate Professor with the School of Computing Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA, where he has established and is the Head of Compiler and Microarchitecture Laboratories. His current research interests include the intersection of compilers and architectures of embedded and multicore systems, with the goal of improving power, performance, temperature, energy, reliability, and robustness.