MAGIC: Malicious Aging in Circuits/Cores

NAGHMEH KARIMI, New York University
ARUN KARTHIK KANUPARTHI, Security Center of Excellence, Intel Corporation
XUEYANG WANG, New York University
OZGUR SINANOGLU, New York University Abu Dhabi
RAMESH KARRI, New York University

The performance of an IC degrades over its lifetime, ultimately resulting in IC failure. In this article, we present a hardware attack (called MAGIC) to maliciously accelerate NBTI aging effects in cores. In this attack, we identify the input patterns that maliciously age the pipestages of a core. We then craft a program that generates these patterns at the inputs of the targeted pipestage. We demonstrate the MAGIC-based attack on the OpenSPARC processor. Executing this program dramatically accelerates the aging process and degrades the processor's performance by 10.92% in 1 month, bypassing existing aging mitigation and timing-error correction schemes. We also present two low-cost techniques to thwart the proposed attack.

Categories and Subject Descriptors: B.8 [**Performance and Reliability**]; C.1 [**Processor Architecture**]; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Design, Performance

Additional Key Words and Phrases: Hardware security, malicious aging acceleration, NBTI aging

ACM Reference Format:
Naghmeh Karimi, Arun Karthik Kanuparthi, Xueyang Wang, Ozgur Sinanoglu, and Ramesh Karri. 2015. MAGIC: Malicious aging in circuits/cores. ACM Trans. Architec. Code Optim. 12, 1, Article 5 (April 2015), 25 pages.
DOI: http://dx.doi.org/10.1145/2724718

1. INTRODUCTION

With shrinking feature size and increasing complexity of microarchitectures, reliability of integrated circuits (ICs) has become a main concern for IC designers. The circuitry composing an IC degrades over its lifetime, ultimately resulting in IC failure [Sinanoglu et al. 2013]. Performance degradation of an IC due to aging is influenced by

The first and second authors performed this research while being a visiting assistant professor and a Ph.D. student at New York University, respectively.

This work was supported in part by the Center for Interdisciplinary Studies in Security and Privacy (CRISSP) New York, CRISSP-AD, and the National Science Foundation Computing and Communication Foundation (NSFCCF) under grant 1319841. The views expressed in this article solely belong to the authors and do not in any way reflect the views of Intel Corporation. The countermeasures described in this article were implemented in experimental hardware and software environments. The authors of this article have not explored the potential applicability of these countermeasures to commercially available hardware and software.

Authors' addresses: N. Karimi (Current address), Electrical and Computer Engineering Department, Rutgers University, Piscataway, NJ; email: naghmeh.karimi@rutgers.edu. A. K. Kanuparthi, Security Center of Excellence, Intel Corporation, Hillsboro, OR; email: arun.kanuparthi@nyu.edu. X. Wang and R. Karri, Electrical and Computer Engineering Department, New York University, Brooklyn, NY; email: {xueyang.wang,rkarri}@nyu.edu. O. Sinanoglu, Computer Engineering Department, New York University Abu Dhabi, Abu Dhabi, UAE; email: ozgursin@nyu.edu.

the operating conditions of the circuit including temperature, voltage bias, and current density [Feng et al. 2011]. Negative-Bias Temperature-Instability (NBTI) [Kufluoglu and Alam 2007; Chakravarthi et al. 2004; Lu et al. 2009], Hot Carrier Injection (HCI) [Saha et al. 2006], and gate Oxide Breakdown (OB) [Yeoh and Hu 1998; Rodriguez et al. 2003] are a few runtime circuit degradation mechanisms.

NBTI occurs when traps are generated at the Si–SiO_2 interface when a negative voltage is applied to a PMOS device [Bhardwaj et al. 2006]. NBTI increases the magnitude of threshold voltage (V_{th}) of the PMOS transistor under stress and reduces the drain current through it, and hence increases the delay through the distressed PMOS transistor. At the circuit level, this manifests as circuit timing and functional failure [Khan et al. 2011]. As VLSI technology scales, NBTI significantly contributes in degrading circuit reliability [Roy and Pan 2014; Liu and Chen 2014]. In practice, NBTI is becoming an increasingly important reliability issue as the gate oxide gets thinner [Alam et al. 2007; Mahapatra et al. 2006]. High-k gate dielectrics have worsened the effects of NBTI on circuit performance [Wang et al. 2010]. In practice, not only planar MOSFET devices but also FinFETs experience temporal NBTI performance degradation [Kükner et al. 2014; Wang et al. 2012].

Two prevalent theories, Reaction-Diffusion (R-D) and Trapping/Detrapping (T-D), have been proposed in the literature to explain NBTI. The R-D model explains the NBTI phenomenon as the breaking and rebonding of hydrogen–silicon bonds at the silicon–gate dielectric interface of PMOS devices [Schroder 2007; Cha et al. 2014]. The T-D model considers a number of defect states with different energy levels as well as capture and emission time constants. In the T-D model, the threshold voltage increases when a trap captures a charge carrier from the channel of a PMOS device [Sutaria et al. 2015; Grasser et al. 2010]. In this article, to evaluate the NBTI effects, we considered the R-D model presented in Wang et al. [2010].

Guardbanding, gate sizing, voltage tuning (changing V_{dd} or V_{th} at runtime), and body biasing are among the methods used to mitigate performance degradation due to NBTI [Abella et al. 2007; Vattikonda et al. 2006; Bild et al. 2009; Lee and Kim 2011]. Modern processors integrate error detection and recovery circuits to protect against timing errors, parametric variations, aging, and voltage drops [Sarangi et al. 2008; Ernst et al. 2003; Bowman et al. 2011]. Aging-aware scheduling [Tiwari and Torrellas 2008] has been proposed to delay or hide the effect of aging in multicore processors.

1.1. Malicious Aging in Circuits/Cores Attack

While IC designers put tremendous effort into reducing aging effects and enhancing the reliability of electronic chips, adversaries may aim at accelerating the wearout of these chips. In practice, a malicious adversary may accelerate the aging process of an IC and thus shorten the device's lifespan. This type of hardware security threat results in denial of service to the IC user and may cause catastrophic failure of the system.

In this article, we focus on accelerating processor failures. In particular, we focus on NBTI effects and present a framework to wear out processors. The proposed attack is based on the observations that circuit delay is input dependent and a circuit exhibits its worst-case delay for specific input patterns [Wolrich et al. 1984]. This article aims to identify those input patterns in a processor and build instructions that generate such patterns.

Executing a malicious program that consists of these instructions on devices such as mobile phones, tablets, and PCs accelerates IC failure and causes the chip to fail sooner than expected (i.e., shortens the chip's normal lifetime).

Assume that a processor has a service life of Y months (typically 60 months). Initially, the processor has a critical path delay equal to C_0. As the processor is used, the critical path delay gradually increases due to aging. Thereby, after a period of $t = Y$, the critical

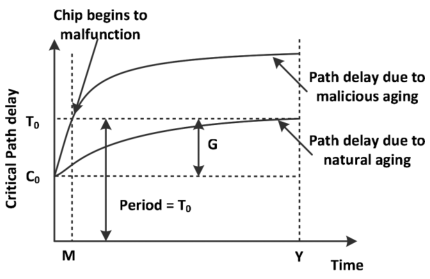


Fig. 1 Impact of accelerated aging on critical path delay and illustration of MAGIC-based attack.

path delay increases to T_0 (as shown in Figure 1). For the processor to be usable during its expected lifetime, it needs to be clocked at a frequency no higher than $f_0 = 1/T_0$. Thus, the designers add a guardband $G = T_0 - C_0$ to prevent any errors due to aging. The guardband is usually within 5% to 10% of the processor bin frequency. However, when processor aging is intentionally accelerated, the critical path delay reaches T_0 at time $t = M$ months, where $M \ll Y$.[1]

1.2. Key Contributions

We present a circuit microarchitecture attack that maliciously accelerates aging in a core (MAGIC). To the best of our knowledge, no prior work has presented a framework to accelerate aging in the open literature. We demonstrate the MAGIC-based attack on the OpenSPARC processor. The key contributions in this article are:

—Adapt VLSI test techniques to obtain the input patterns that accelerate the NBTI aging of a core.
—Construct a MAGIC program that accelerates aging.
—Maliciously age the execute stage (E stage) in the OpenSPARC T1 processor [Oracle 2006a]. When the MAGIC program is executed, the performance of the E stage degrades by 10.92%, 13.25%, and 16.8% after 1, 2, and 6 months, respectively, bypassing protection offered by existing timing error correction techniques, thereby causing the processor to fail. The degradation has been evaluated by using the R-D NBTI model presented in Wang et al. [2010].
—Propose two low-cost approaches to thwart the proposed attack. These schemes are obtained by understanding the operation of the MAGIC attack.

The rest of the article is outlined as follows. Background on NBTI aging and its effect on primitive logic gates is provided in Section 3. Section 4 presents the circuit-level MAGIC-based attack and discusses its impact on IC reliability. The microarchitecture-level MAGIC-based attack, the methodology to maliciously accelerate aging of cores and craft assembly-level instructions from the processor's instruction set, is described in Section 5. Section 6 presents and discusses the experimental results. Section 7 explains how MAGIC can bypass defenses that detect timing errors in digital circuits. Section 8 presents two microarchitecture-level techniques to thwart malicious aging. Section 9 discusses the practicality of the attack. Finally, Section 10 concludes the article.

2. ATTACK SCENARIOS AND THREAT MODEL

2.1. Attack Scenarios

Attack Scenario 1 (*Warranty Attack*): A consumer C purchases a smartphone manufactured by company X. The warranty period for this device is W months. Consumer

[1]As will be shown in Section 6, M can be as small as 1 month.

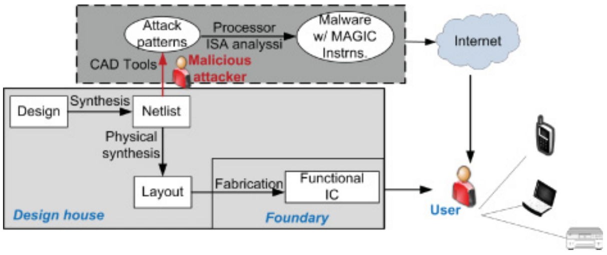


Fig. 2. Processor design flow (solid box) and the MAGIC-based attack (dotted box). Using CAD tools, the attacker identifies the critical path and the input patterns that put the gates in this path under maximum stress. After processor ISA analysis, the attacker identifies assembly-level instructions that generate these input patterns and builds the MAGIC program. Execution of this program accelerates processor aging and causes products to fail sooner than their normal lifetime.

C uses the device for a while, but when the device is still under warranty, a physical damage occurs (e.g., scratch on the LCD). C wants to get a new phone but the warranty does not cover physical damage. C downloads the malicious program and the OS from forums such as Cyanogenmod [Cyanogen], executes the malicious program to intentionally wear out the device, and returns the device to X to get a new one.

Attack Scenario 2 (*Planned Obsolescence*): A malicious device manufacturer M makes previously sold devices very slow in order to encourage (force) its customers to buy an upgraded recently released device. In this scenario, company M sends a patch to its customers shortly before releasing a new device. Installing such patch maliciously makes the hardware slow and the users feel compelled to buy the recently released hardware [Rosoff 2012; Worstall 2013; Rampell 2013; Skipworth 2012; Mims 2013].

Attack Scenario 3 (*State-Sponsored Hardware Backdoor*): A country G purchases some military equipment (e.g., radars, antiaircraft weapons, satellites, communication equipment) from a company Y in another country. The company Y maliciously inserts hardware backdoors on this equipment to be able to control the equipment remotely and execute the malicious program on it. The equipment is installed in country G to take care of boarders even when there is no war. Company Y controls the equipment remotely and maliciously accelerates aging and eventually causes the military equipment to stop working properly.

In the first scenario, the user wants to make the device malfunction, while in the latter scenarios, the manufacturing company aims to wear out the device. These scenarios highlight the proposed attack as a serious concern. In the *warranty attack* scenario, the attack may not impose a considerable financial loss to the victim manufacturer, but even wearing out a few devices may jeopardize the reputation of the manufacturer. In the *planned obsolescence* scenario, the attacker aims at increasing its financial benefit by encouraging its customers to buy the latest device. Finally, in the *hardware backdoor* attack, the attacker intentionally jeopardizes the security of another country. Both the *planned obsolescence* and *hardware backdoor* scenarios aim at making the hardware stop working properly, but in these attacks, the malicious aging is hidden from the user (i.e., pretending that the hardware is normally aged and hiding that the aging is due to an external intervention).

2.2. Threat Model

Figure 2 shows the processor design flow (solid box) and depicts how the MAGIC-based attack is launched during this process (dotted box). As shown, the design is synthesized and the netlist and layout are generated. The layout is sent to the foundry for fabrication. After IC testing, the fault-free ICs are shipped.

The MAGIC-based attack can be implemented by different players. In the *warranty attack* scenario, we have two major players: an *expert attacker* and a *nonexpert malicious user*. An expert attacker is capable of using CAD and VLSI testing tools. He or she also has expertise in processor microarchitecture. His or her goal is to create the malicious program and distribute it to several malicious users in order to create widespread damage. The expert attacker is an individual attacker who either profits by selling the malicious program or is hired by a company that competes with the manufacturer of the targeted device. Even wearing out a few such devices may jeopardize the reputation of the manufacturer. On the other hand, the nonexpert malicious user has no expertise in CAD, VLSI, or processor microarchitecture.[2] The malicious user's goal is to damage his or her device before the warranty period expires and get a new device from the design company. The warranty attack is launched in four steps as follows:

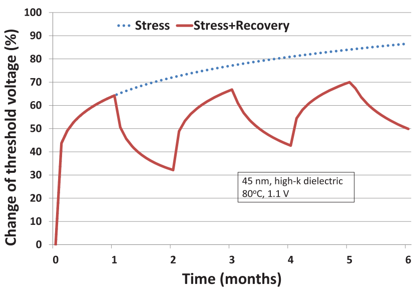
—**Step 1:** An expert attacker obtains the netlist of the processor through a *malicious insider* in the design house or by reverse engineering the manufactured chip[3] [Torrance and James 2011].
—**Step 2:** The expert attacker identifies the critical path of the processor and obtains input patterns that place this path under NBTI stress.
—**Step 3:** The expert attacker analyzes the processor Instruction Set Architecture (ISA), crafts instructions that generate the previously mentioned patterns, and creates a malicious program using them.
—**Step 4:** Finally, the expert attacker uploads the malicious program to the Internet to forums such as Crackberry or Cyanogenmod, from where several nonexpert malicious users download and execute it on the targeted processors.

For the *planned obsolescence* and *hardware backdoor* scenarios, the manufacturing company itself launches the attack for financial benefits or national security benefits, respectively. In these two cases, the manufacturing company has access to the netlist. Therefore, Step 1 is ignored. In addition, in these scenarios, the attack is implemented by only one player, that is, the manufacturing company. Accordingly, the box shown in dotted dark border in Figure 2 should be considered as embedded inside the solid box for these two scenarios. The following steps are taken for these two scenarios:

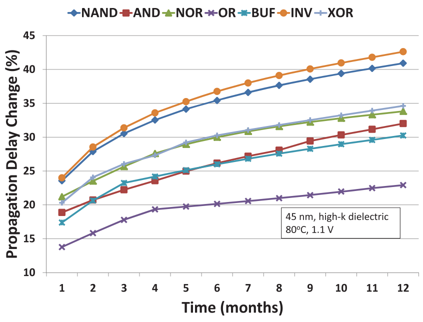
—**Step 1:** The manufacturing company identifies the critical path of the processor and obtains input patterns that place the critical path under NBTI stress.
—**Step 2:** The manufacturing company analyzes the processor ISA, crafts instructions that generate the previously mentioned patterns, and creates a malicious program using them.
—**Step 3:** For the planned obsolescence scenario, the manufacturing company sends the malicious program to its customers shortly before/after an upgraded version of the targeted device is released. For the hardware backdoor scenario, the malicious program is stored on an on-chip ROM. The manufacturing company controls the processor remotely and starts running the program when needed.

Note that the MAGIC-based warranty attack can be applied either in the test or in the functional mode of the processor. In the test mode, logical inputs that age the processor (obtained after Step 2) are applied through a JTAG port. Technical expertise is required to launch the attack in the test mode, and it cannot be performed by nonexpert users. However, in the functional mode, the malicious program (obtained after Step 3) is

[2]We assume that the nonexpert user has a hacked Operating System (OS) installed and has root access to the targeted device.
[3]Reverse engineering is not impractical. Chipworks reverse-engineered Intel chipsets from 1992 onward, including the 22nm Ivy Bridge.



(a)



(b)

Fig. 3. (a) Percentage change in threshold voltage of a PMOS transistor over time. (b) Percentage change in propagation delay of primitive logic gates as a function of stress time.

executed on the processor. This approach is easier and can be performed by anyone who can download this program from the Internet. Hence, the impact of the attack is widespread and dramatic. For the MAGIC-based planned obsolescence and hardware backdoor attacks, the malicious program is executed in the functional mode.

3. PRELIMINARIES

3.1. Background on NBTI

NBTI is one of the leading factors in performance degradation of digital circuits. In practice, a PMOS transistor experiences two phases of NBTI depending on its bias condition. The first phase (i.e., the stress phase) occurs when the transistor is on (i.e., when a negative voltage is applied to its gate). In the stress phase, positive interface traps are generated at the Si–SiO$_2$ interface. As a result, the magnitude of the threshold voltage of the transistor is increased. In the second phase (i.e., the recovery phase), a positive voltage is applied to the gate of the transistor. In this phase, the threshold voltage drift induced by NBTI during the stress phase can partially "recover."

Threshold voltage drifts of a PMOS transistor under stress depend on the physical parameters of the transistor, supply voltage, temperature, and stress time. Figure 3(a) shows the threshold voltage drift of a PMOS transistor (at an operating temperature of 80°C) that is continuously under stress for 6 months as well as a transistor that is under stress and recovery every other month. As shown, the NBTI effect is high in the first couple of months, but the threshold voltage tends to saturate for long stress times.

Throughout this article, to evaluate the impact of NBTI on the performance of a logic circuit under stress, we use the mathematical model presented in Wang et al. [2010]. In this model, the changes in threshold voltage of a PMOS transistor in stress and recovery modes at time t are evaluated by Equation (1) and Equation (2), respectively:

$$\Delta V_{th} = \left(K_v(t-t_0)^{0.5} + \sqrt[2n]{\Delta V_{th0}}\right)^{2n} \tag{1}$$

$$\Delta V_{th} = \Delta V_{th1}\left(1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(t-t_1)}}{2t_{ox} + \sqrt{Ct}}\right), \tag{2}$$

where t_0 and t_1 denote the time at which the stress and recovery phases begin, respectively; t_e denotes the effective oxide thickness; and ξ_1 and ξ_2 are 0.9 and 0.5, respectively. Parameter n is the time exponent parameter, and for H_2 diffusion, it is 1/6. K_v and C are computed by using Equation (3), where E_{ox} is the electrical field, T is the temperature,

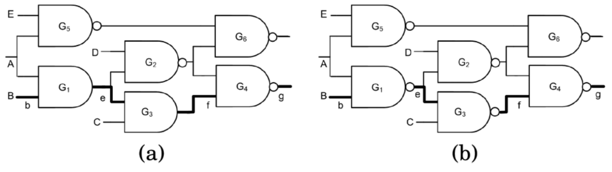


Fig. 4. Sample circuits (longest path P_1 shown in bold).

and E_a, K_1, T_0, and k are constants. As shown in Equation (1), the magnitude of the threshold voltage of a PMOS transistor is increased during stress time:

$$K_v = \left(\frac{qt_{ox}}{\epsilon_{ox}}\right)^3 {K_1}^2 C_{ox}(V_{gs} - V_{th})\sqrt{C} exp\left(\frac{2E_{ox}}{E_{01}}\right), \quad C = exp(-E_a/kT)/T0. \tag{3}$$

Note that in this article, we focus on the malicious NBTI aging of two-dimensional transistors and leave dealing with the malicious NBTI aging of FinFETs for future work [Kükner et al. 2014; Wang et al. 2012].

3.2. Effect of NBTI Aging on Delay of Primitive Logic Gates

To evaluate the effect of NBTI on the propagation delay of primitive gates, we conducted a series of HSpice simulations using 45nm technology with high-k dielectric, at a nominal supply voltage (Vdd) of 1.1V and a nominal temperature of 80°C. We first extracted the nominal propagation delay of each primitive gate. Then, we evaluated the change in threshold voltage using the model discussed in Section 3.1 assuming that the gate is subjected to NBTI stress for different time durations. Finally, using the degraded threshold voltage values, we ran HSpice simulations to extract the propagation delay of each gate under NBTI stress. Figure 3(b) shows the change in propagation delay of primitive logic gates over time when these gates are under NBTI stress. This figure shows the case where the gates are continuously under NBTI stress and they do not experience recovery. As shown, on average, the propagation delay of primitive gates is increased by 19.9% under 1 month of stress. Although the propagation delay of each gate increased over time, the increase is the most in the first couple of months.

4. CIRCUIT-LEVEL MAGIC-BASED ATTACK

4.1. Methodology

In this section, we describe our circuit-level attack to accelerate NBTI-related performance degradation of a digital circuit. Consider the circuit shown in Figure 4(a). Assume that path P_1 (including G_1, G_3, and G_4) is the timing-critical path of this circuit. By holding the primary inputs constant at $V_1(ABCDE)$ = "d0ddd" (where d stands for don't care), the PMOS transistors of each gate in this path are kept "on," and therefore, all these gates experience NBTI aging. However, it is not always possible to stress all the gates in the critical path of a circuit simultaneously. As another example, consider the circuit in Figure 4(b). In this circuit, holding the primary inputs constant at $V_1(ABCDE)$ = "d01dd" accelerates aging of G_1 and G_4, while holding the primary inputs constant at $V_2(ABCDE)$ = "11ddd" accelerates aging of G_3.

In this attack, we look for a minimal set of input patterns that, when applied to the circuit, result in maximal NBTI stress. For the circuit shown in Figure 4(a), this set includes only one input pattern, that is, $ABCDE$ = "d0ddd", while for the circuit in Figure 4(b), this set includes two patterns, that is, $ABCDE$ = "d01dd;11ddd". We call them MAGIC patterns.

To identify MAGIC patterns, we target the critical path of the circuit and find the input patterns that put each individual gate on this path under NBTI stress. We

Table I. MAGIC Patterns That Put Each Gate of the Critical Path of Figure 4(a) and Figure 4(b) Under NBTI Stress

Gates	MAGIC Patterns That Stress Circuit	
	in Figure 4(a)	in Figure 4(b)
G_1	"d0ddd"	"d0ddd"
G_3	"0dddd", "d0ddd"	"11ddd"
G_4	"0dddd", "d0ddd", "dd0dd"	"0d1dd", "d01dd"

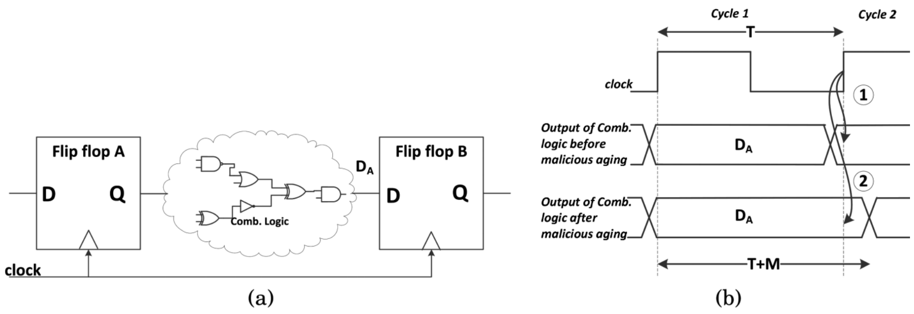


Fig. 5. (a) Digital circuit. (b) Timing error with increased critical path delay due to malicious aging.

then compress these patterns into a small set. To generate the patterns that put each individual gate of the critical path under stress, we leverage automatic test-pattern generation (ATPG) tools and target stuck-at-1 faults on the input of each gate in the critical path (only the input residing on the critical path, and not all inputs of the gates).

The second column of Table I shows the patterns that stress each gate of the circuit shown in Figure 4(a). For example, to find the MAGIC patterns that place the value of 0 on e and to stress gate G_3, we assume that e is observable and then generate the test patterns that detect the stuck-at-1 fault on e. We use a minimum-set-covering algorithm to select a minimal set of vectors that put all the gates in the critical path under NBTI stress. As shown in the second column of Table I, by applying input pattern "d0ddd", all gates of the critical path are stressed. The third column shows the similar results for the circuit depicted in Figure 4(b). As shown, for this circuit, the minimal set of patterns required to target all the gates of the critical path includes "d01dd; 11ddd".

The magnitude of aging is determined by the duration that the MAGIC patterns are continuously applied to the circuit, as well as the type of gates in the targeted path.[4]

Note that the postmanufacturing critical path may differ from the design-time critical path due to process variations. Similarly, the critical path may change during the circuit operation. In these cases, the attacker may have targeted a near-critical path and generated MAGIC patterns for such path instead of the critical path. In these cases, the MAGIC attack is still feasible but may take longer to make the circuit operate improperly. A dynamic change of the critical path is less frequent in older technologies.

4.2. Impact of Circuit-Level Attack

Now consider the circuit in Figure 5(a). The combinational logic block between flip-flops A and B is implemented using primitive logic gates. This digital circuit has a

[4]Due to the large number of paths in real circuits, in this attack, we consider the first 10 longest paths. This is because (1) several paths may have approximately the same delay and (2) two paths with similar delays may age differently; that is, our attack may degrade a near-critical path, making it the bottleneck. In practice, from the set of considered paths, we select the path that ages the most.

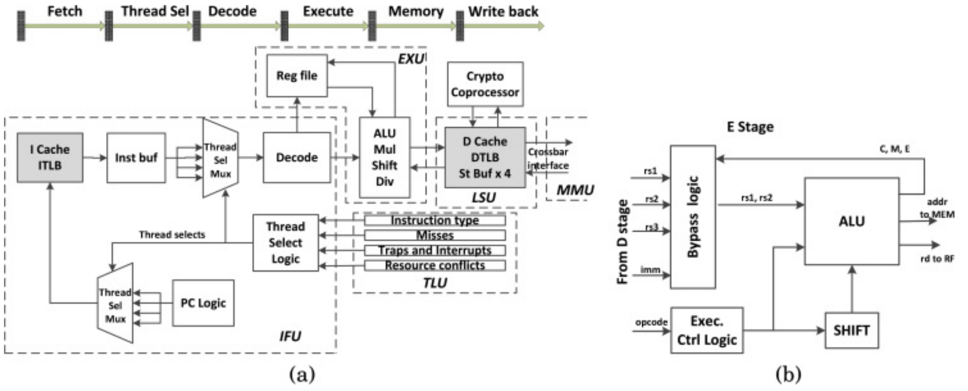


Fig. 6. (a) Microarchitecture of a SPARC processor. (b) Block diagram of the E stage.

critical path delay of T and runs at a clock frequency of $f = 1/T$. The output of the combinational block, D_A, is sampled by flip-flop B at the end of cycle 1, cycle 2, and so forth. This is indicated in Figure 5(b) by curved arrow 1. When the gates in the combinational logic block are maliciously aged, its critical path delay increases to T + M. However, the digital circuit still operates at a frequency of $f = 1/T$, and hence flip-flop B samples incorrect values of D_A, as indicated by curved arrow 2 in Figure 5(b).

4.3. Discussion

The takeaways of the circuit-level attack are as follows:

—MAGIC patterns that stress the gates in the critical path are obtained using CAD and VLSI test tools.
—A circuit-level MAGIC-based attack can be launched in the test mode of the IC by scanning in the logical values through the JTAG port.
—Technical expertise and expensive equipment are required to launch the attack in the test mode. Hence, the impact is not widespread.

5. MICROARCHITECTURE-LEVEL MAGIC-BASED ATTACK

A microarchitecture-level attack does not require a user to have expensive equipment or technical expertise to launch the attack. The generated malicious MAGIC program is executed on the targeted device in its functional mode.

5.1. The OpenSPARC Framework

We demonstrate the attack on the OpenSPARC T1 processor [Oracle 2006b]. We chose this processor since its netlist is open for academic use. Note that the MAGIC-based attack is generic and the MAGIC program can be generated for any processor based on its netlist and instruction set.

The OpenSPARC processor is multithreaded. Each core includes four hardware threads and one full register file per thread. The instruction cache (I-cache), data cache (D-cache), and TLBs (ITLB and DTLB) are shared by the four threads in the core. As Figure 6(a) shows, each core has a single-issue, six-stage pipeline including Fetch (F), Thread Selection (T), Decode (D), Execute (E), Memory (M), and Write Back (W). Maliciously aging each pipestage can jeopardize the correct functionality of the processor.

The maximum frequency of a processor is determined by its slowest pipestage. Aging such stage will result in timing errors. In the OpenSPARC processor, the E stage (Figure 6(b)) is the slowest pipestage and we target it.

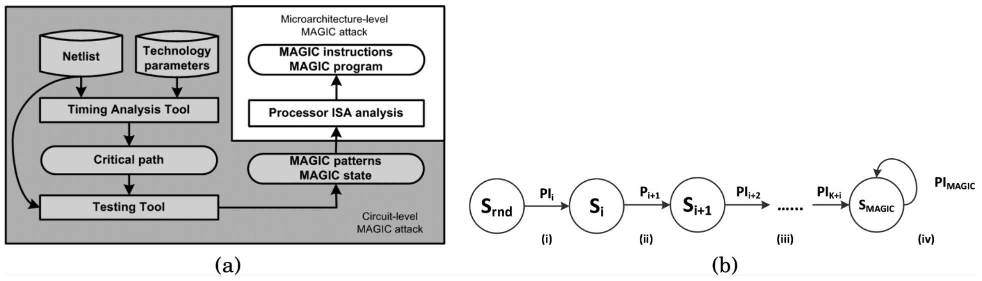


Fig. 7. (a) The steps taken to generate the MAGIC program. The shaded and unshaded areas show the circuit-level and microarchitecture-level MAGIC-based attacks, respectively. (b) State transition from a random state to the MAGIC state by executing assembly-level instructions.

5.2. Methodology

The circuit-level attack is used to identify the patterns required to accelerate NBTI aging in the critical path of the targeted processor. We analyze the processor ISA and generate the MAGIC instructions that produce the MAGIC patterns. These instructions build the MAGIC program. Figure 7(a) shows this procedure. To accelerate NBTI aging, the flip-flops feeding the critical path should hold their values for a certain time; that is, their contents should not be changed. We call this state of flip-flops the MAGIC state. Once the processor is in the MAGIC state, MAGIC instructions are executed repeatedly to hold the appropriate values on the flip-flops that feed the critical path. The MAGIC program has two phases. In Phase 1, certain instructions are executed to bring the processor to the MAGIC state. In Phase 2, the MAGIC instructions are repeatedly executed. In this article, we only focus on the aging of combinational gates. Intuitively, aging of flip-flops will benefit the attacker because propagation, setup, and hold times vary with NBTI stress [Abrishami et al. 2008; Rao and Mahmoodi 2011].

5.3. Set the Processor in the MAGIC State

To place the critical path of the processor under NBTI stress, certain input patterns should feed this path for a certain amount of time. Moreover, the flip-flops that feed the critical path should hold appropriate values for the same time period. Setting all the flip-flops leading to the critical path in the MAGIC state, before the MAGIC input pattern is applied, is a nontrivial task. To feed these flip-flops with required values, one option is to scan in the patterns in the test mode and freeze the test clock for a certain time as discussed in the circuit-level attack. *The same can be achieved in the "functional mode" of the processor by executing certain instructions that can control these flip-flops and eventually make a transition to the MAGIC state*. These instructions are selected from the instruction set of the processor. Repeating execution of the MAGIC instructions for a certain time eliminates the need to freeze the clock signal.

The primary inputs that should be applied to the victim component (pipestage), PI_{MAGIC}, and the state of the flip-flops in the critical path, S_{MAGIC}, before applying these inputs are determined as discussed in Section 4. The next task is to set the flip-flops to the MAGIC state. The MAGIC state can be obtained by applying a sequence of instructions from a known initial state such as the state immediately before (after) the boot sequence is initiated (completed). Since trusted computing initiatives such as Trusted Boot prevent execution of these unknown instructions at boot time, the state has to be changed only after the OS has finished booting.

We bring the victim component to the MAGIC state, S_{MAGIC}, from any state by applying a sequence of instructions that can control the flip-flops leading to the critical path. Applying primary input PI_i to any random state, S_{rnd}, causes a transition to S_i.

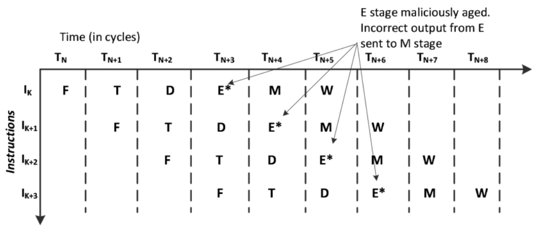


Fig. 8. Maliciously aged E stage produces incorrect results, which are dispatched to the further stages (M, W) in the pipeline, resulting in an incorrect program output.

When primary input PI_{i+1} is applied to state S_i, state S_{i+1} is obtained. This process continues until S_{MAGIC} is reached. Then, the MAGIC patterns, PI_{MAGIC}, are repeatedly applied (Figure 7(b)). By setting constraints on the outputs of the flip-flops, the input patterns that produce these outputs are obtained using an ATPG tool. This approach can be used when only a few flip-flops should get predetermined values and the rest can get arbitrary values (don't-care values).

If all the flip-flops leading to the critical path cannot be set to a particular value by executing any instruction, then the critical path cannot be put under maximum stress. However, if a majority of those flip-flops are controllable, then the critical path can be put under considerable stress as indicated by the results presented in Section 6.

5.4. Construct MAGIC Instructions

The inputs of one pipestage are the outputs of the previous pipestage (or the next pipestages in cases where forwarding is required). For instance, the inputs to the execute stage are outputs of the decode stage and a few inputs from the execute/memory stages. To determine the corresponding input patterns of the decode stage, we map the outputs of the decode stage to its inputs. Then, we analyze the inputs of the decode stage along with the ISA to construct MAGIC instructions.

5.5. Impact of Microarchitecture-Level Attack

A timing error caused by malicious aging in a pipestage has an adverse effect on program output. Execution of a program on an aged processor produces erroneous results and may crash. Consider an instruction execution sequence on a SPARC core shown in Figure 8. The E stage is maliciously aged. Thus, for instruction I_K, at time T_{N+3}, the E stage produces incorrect results, which are sent to the M stage at time T_{N+4} and to the W stage at time T_{N+5}. These incorrect values are forwarded through the W stage, producing incorrect program output.

6. RESULTS AND ANALYSIS

6.1. Experimental Setup

In our experiments, a series of HSpice simulations were conducted to evaluate the effect of aging for each primary logic gate at discrete time units using the 45nm predictive technology model with high-k dielectric [Model 2007]. The source code for the OpenSPARC T1 processor was obtained from Oracle [2006a]. The Synopsys Design Compiler tool was used for logic synthesis. Synopsys PrimeTime was used for extracting timing-critical paths. The scripts to select the final patterns and evaluate the attack outcome were all implemented in Python. Synopsys Tetramax was used to obtain the circuit-level patterns for state and primary inputs. Synopsys VCS was used for functional simulations. In these experiments, the nominal supply voltage (Vdd) is assumed to be 1.1V and the nominal temperature is 80°C. Note that 80°C is considered for the core temperature. Core temperature (so-called Tjunction or operating temperature)

Table II. SPARC Core Configuration Details

Parameter	Specification
Threads	4
L1 I-Cache	16KB, 4-way (Line Size: 32 bytes, Latency: 2 cycles)
L1 D-Cache	8KB, 4-way (Line Size: 16 bytes, Latency: 2 cycles)
L2 Cache	3MB, 12-way (Latency: 12 cycles)
ITLB/DTLB/Load Miss Queue	64/64/4 entries
Store Buffer	32 entries for four threads
Memory	Latency: 100 cycles

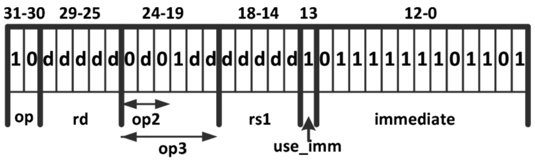

31-30
29-25
24-19
18-14
13
12-0
1 0 d d d d d 0 d 0 1 d d d d d d d 1 0 1 1 1 1 1 1 1 0 1 1 0 1
op
rd
op2
op3
rs1
use_imm
immediate

Fig. 9. Bit fields and values obtained for the 32-bit SPARC MAGIC instruction.

differs from CPU temperature (so-called Tcase) and is also considerably higher than the ambient temperature. As discussed in Inkley [2008], on average, the target operating temperature for 45nm Intel processors is around 90°C. Therefore, given that the MAGIC program is executed continuously on the targeted core, considering 80°C as the nominal temperature for our simulation is justified. The effect of aging over time was estimated by using the mathematical model presented in Wang et al. [2010] and was described in Section 3. We found the threshold-voltage change over time for the transistors in the critical path. Then, we used HSpice to evaluate the delay of each gate in the critical path using the extracted threshold voltage and used it to compute the postaging critical path delay. This methodology for aging was previously used in several reliability studies [Bild et al. 2009; Lee and Kim 2011; Sarangi et al. 2008; Ernst et al. 2003; Bowman et al. 2011].

The gem5 cycle accurate simulator was used to run benchmarks from the SPEC CPU2006 and PARSEC benchmark suites and evaluate the performance degradation due to malicious aging. The configuration of the SPARC core used in these simulations is shown in Table II.

6.2. Experimental Results

6.2.1. MAGIC Assembly Instructions. We used Synopsys PrimeTime to find the critical path of the E stage. This path is located in the bypass logic (Figure 6(b)) and includes 110 gates (46 AND gates, 15 INV gates, 46 OR gates, and three XOR gates). The input pattern that accelerates NBTI aging in the critical path was mapped to the primary inputs of the Execution Unit (EXU). Of the 603 primary inputs to the Estage, 563 were observed to be don't cares. The remaining 40 inputs were:

—*ifu_exu_useimm_d* = 1 (immediate operand must be used),
—*ifu_exu_imm_data_d[31:0]* = 00000*FED* (immediate value),
—*ifu_exu_usecin_d* = 0 (carry-in must not be used),
—*ifu_exu_dbrinst_d* = 0 (instruction is not a branch type),
—*ifu_exu_invert_d* = 0 (do not invert the second operand),
—*ifu_exu_casa_d* = 0, *clk_in* = 0 (clock), *se* = *0* (scan disable), and *se_hold* = *0*.

These 40 inputs to the E stage were the outputs of the decode stage. Inputs to the D stage that produce these outputs were obtained by analyzing the decode logic. The obtained instruction is shown in Figure 9. The primary opcode, *op*, which corresponds to the bits 31-30 of the 32-bit instruction, was “10.” Thereby, the instruction must be an arithmetic instruction. Bit field 29-25 indicates the destination register, *rd*. All these bits were don’t cares. The 6-bit function opcode, *op3*, was observed to be “0d01dd”. Hence, the instruction can either be a SUB, SUBcc, ANDN, ANDNcc, ORN, ORNcc, XNOR, or XNORcc.[5] We arbitrarily chose to use the SUB instruction (while using any of the other instructions would have the same effect). Bit field 18-14 indicates the first source operand, *rs1*. All these bits were don’t cares. Bit 13 indicates whether the instruction is an immediate instruction or not. Since it was “1,” the lower 13 bits indicate the immediate value. The signed immediate value was 0FED. We arbitrarily chose to use r12 as rs1 and r20 as rd. Thus, the first MAGIC instruction was **SUB r12, 0FED, r20**. We used the method discussed in Section 4 to find the minimum number of instructions by applying which NBTI aging is accelerated. As a result, two instructions were extracted. The second MAGIC instruction, **SUB r4, 001A, r27,** was also crafted in a manner similar to the first MAGIC instruction. These two MAGIC instructions collectively put the critical path under maximum NBTI stress.

6.2.2. Possible MAGIC instructions. As Figure 9 shows, 19 bits in the 32-bit instruction are either “0” or “1” and the rest are don’t cares. All five rd bits are don’t cares. Hence, any of the possible $2^5 = 32$ registers can be used as the destination register. Similarly, any of the possible 32 registers can be used as the source register rs1. For the function field, op3, with three don’t cares, eight functions are possible. Thus, a total of $32 \times 32 \times 8 = 8{,}192$ MAGIC instructions are possible. However, a MAGIC instruction cannot have the same source and destination register; execution of such an instruction will modify the MAGIC state. Thus, a total of $8{,}192 - 32 \times 8 = 7{,}936$ MAGIC instructions are possible.

6.2.3. Driving to the MAGIC State. Seventy-seven flip-flops had to be set to appropriate values (“0” or “1”) to reach the MAGIC state. All 77 flip-flops were controllable. The primary inputs to the E stage were analyzed and the information was used to trace back to the decode stage and construct the 32 bits of the instruction. The opcodes and operands were obtained and the instructions were crafted. Table III shows the sequence of instructions (1 through 12) that need to be applied to drive the processor to the MAGIC state from any random state. As shown in this table, after placing the processor to the MAGIC state, each MAGIC instruction (instruction 14 and instruction 21) is executed for an extended period of time. Note that the number of instructions to reach the MAGIC state is specific to the ISA and the critical path. For the OpenSPARC execute stage, it is 12. It could be longer for other ISAs and paths. In the table, the contents in bold indicate the alternate options that are possible. For instance, for instruction 1, the secondary opcode is “0000dd”. Thus, the possible instructions are “000000” (ADD), “000001” (AND), “000010” (OR), and “000011” (XOR). The alternate options are presented as comments in Table III. Instructions 13, 19, 20, 26, and 27 have been selected such that they do not modify the MAGIC state. These instructions implement a loop and repeatedly execute MAGIC instructions 1 and 2. By repeated execution of MAGIC instruction 1 (14 to 18) and MAGIC instruction 2 (21 to 25), we effectively increase the number of times the MAGIC instructions are executed. To overcome the limitation on the upper bound of a branch counter, an attacker can put several instances of MAGIC instruction 1 and MAGIC instruction 2 to effectively increase the number of times the MAGIC instructions are executed (Table III shows five occurrences of MAGIC instruction 1 and MAGIC instruction 2). Note that the MAGIC instructions are also selected among alternatives such that they do not change the MAGIC state.

[5]If the instruction has *cc* as postfix, it sets the condition flags. Otherwise, it does not.

Table III. MAGIC Program

1:	**ADD** r0, r7, r8	; AND/OR/XOR
2:	**SUB** r7, RIS, **r13**	; ANDN/ORN/XNOR, r15
3:	**ORCC r6**, r17, r15	; ORCC, r5/r7
4:	ADDcc r11, **r23, r23**	; r22, r22
5:	XOR **r23**, 16741, r15	; r22
6:	**ADD** r27, r18, **r26**	; AND, r8
7:	ORcc **r12, r18**, r11	; r8, r22
8:	AND **r8**, r12, r19	; r24
9:	SETHI 0AAAAA, **r10**	; r11/r14/r15
10:	OR r10, 23A, r16	;
11:	**SETHI** 0231B, r4	; ILLTRAP
12:	OR r10, 23A, r16	;
13:	ADD r0, r0, r30	; Clearing r30
		; MAGIC state reached
14:	**SUB r12**, 0FED, **r20**	; MAGIC instruction 1
15:	**SUB r12**, 0FED, **r20**	; MAGIC instruction 1
16:	**SUB r12**, 0FED, **r20**	; MAGIC instruction 1
17:	**SUB r12**, 0FED, **r20**	; MAGIC instruction 1
18:	**SUB r12**, 0FED, **r20**	; MAGIC instruction 1
19:	ADDcc r30, 1, r30	; Counter
20:	BVS 14:	; Branch to 14 if no overflow
21:	SUB r4, 001A, r27	; MAGIC instruction 2
22:	SUB r4, 001A, r27	; MAGIC instruction 2
23:	SUB r4, 001A, r27	; MAGIC instruction 2
24:	SUB r4, 001A, r27	; MAGIC instruction 2
25:	SUB r4, 001A, r27	; MAGIC instruction 2
26:	SUBcc r30, 1, r30	; Counter
27:	BNE 21:	; Branch to 21 if zero flag not set

Increasing the switching activity in a core increases the core power dissipation and temperature and in turn the NBTI effects (as discussed in Section 3). Accordingly, to increase the switching activity of the targeted core, instead of repeatedly executing each MAGIC instruction (e.g., instructions 14 to 18 in Table III), we can extract a number of alternatives as discussed earlier and interleave those instructions in the loop.

6.2.4. Performance Degradation of SPARC. This set of results quantifies the effect of MAGIC in degrading the performance of the SPARC core. Figure 10(a) shows the magnitude of performance degradation of the E stage when MAGIC instructions are applied to this microprocessor for 1 to 6 months. The results show the maximum performance degradation that the E stage experiences when MAGIC instructions are applied to the processor. On average, the performance of the execution unit is degraded by 10.92% after 1 month, by 13.25% after 2 months, and by 16.8% after 6 months. Note that in the results, we didn't consider the NBTI effects that resulted from the burn-in test process of the processor. We assume that the state-of-the-art NBTI controlling schemes [Chakraborty and Pan 2011] are applied during the burn-in test process to decrease the NBTI effects.

Effectiveness of MAGIC instructions vis-a-vis random instructions: To compare the effectiveness of MAGIC instructions with randomly generated instructions in accelerating NBTI aging, we generated a pair of random instructions and continuously executed each instruction on the SPARC processor. We repeated this experiment

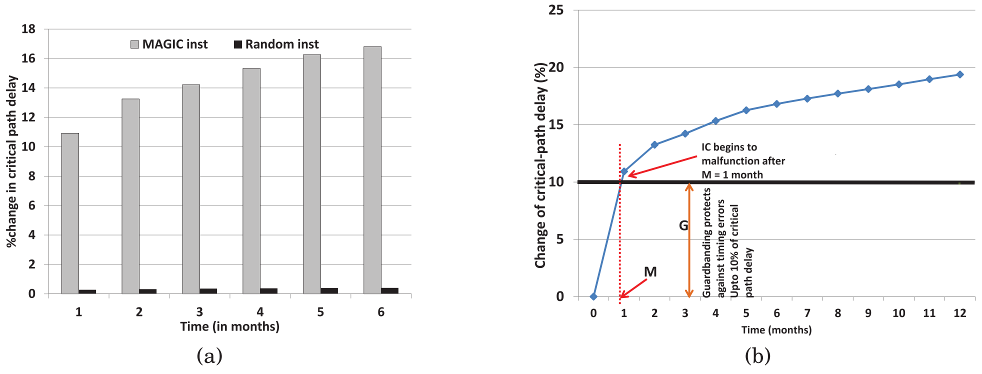


Fig. 10. (a) Comparing the effect of MAGIC versus random instructions in change of the critical path delay of the OpenSPARC processor. (b) Bypassing a 10% guardband in 1 month by using MAGIC to attack the OpenSPARC processor.

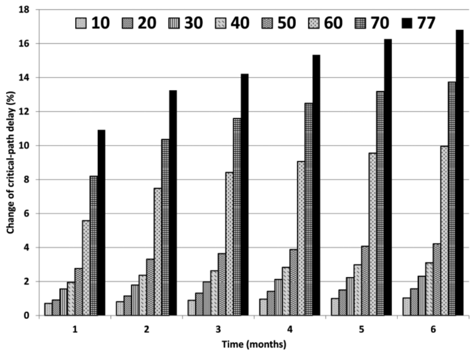


Fig. 11. Percentage change in critical path delay of SPARC E stage due to partial control of MAGIC state.

10 times (i.e., for 10 pairs of randomly generated instructions) and showed the average result in Figure 10(a) while applying random instructions for 1 to 6 months. Applying random instructions does not have a significant effect on the performance. On average, the performance of the E stage degrades by 0.28% after 1 month and by 0.41% after 6 months. This observation reinforces the effectiveness of MAGIC instructions in intentional performance degradation of processors.

Assume that in OpenSPARC, the guardband protects against timing errors that result in an up to 10% change of the critical path delay ($G = 0.1 \times C_0$, where C_0 is the critical path delay). When the MAGIC program is executed, the critical path delay increases by 10.92% in 1 month ($M = 1$ month), bypassing the protection offered by the guardband. The MAGIC attack on the SPARC processor is shown in Figure 10(b).

6.2.5. Partial Control of the MAGIC State. If all the flip-flops leading to the critical path are not controllable, the gates in the critical path will not be maximally stressed. However, the critical path can still be stressed if the number of controllable flip-flops is close to the total number. As shown in Figure 11, if only 10 arbitrarily selected flip-flops out of a total 77 flip-flops are controllable, the critical path delay is only changed 0.71% after 1 month and 1.03% after 6 months. If 20 flip-flops are controllable, the change in critical path delay is 0.91% after 1 month and 1.57% after 6 months. However, if the number of controllable flip-flops increases to 60 (70), the change in critical path delay is 9.95% (13.73%) after 6 months.

7. EFFECTIVENESS OF MAGIC AGAINST RELATED WORK

In this section, we will discuss how MAGIC bypasses current protection mechanisms.

7.1. Circuit-Level NBTI Effect Mitigation Techniques

Razor [Ernst et al. 2003] dynamically detects and corrects circuit timing errors. Razor samples the pipestages twice, one with a normal clock and the other with a delayed clock, and compares the samples to detect any error. If an error is detected, error recovery is performed by using clock gating or counterflow pipelining. In clock gating, when an error is detected, the entire pipeline is stalled for one cycle. In counterflow pipelining, when an error is detected, the effect of the erroneous computation is nullified, the pipeline is flushed, and the instruction is re-executed from the failing stage. An alternate approach toward dynamic variation tolerance has been proposed in Bowman et al. [2011], where error-differential sequentials (EDSs) and tunable replica circuits (TRCs) are embedded in the processor core to detect delay faults. Both Razor and EDS-TRC were designed to detect timing errors in the margin range of 2% to 10%.

Assume that the methods discussed previously are used against MAGIC-based attacks. The clock is delayed enough to detect the timing errors caused by MAGIC. Then, the error recovery techniques are used to correct these errors. One recovery method flushes the pipeline and re-executes the faulty instruction several times (10 times in Bowman et al. [2011]) at the same clock frequency, after which normal execution continues. This method cannot correct the timing errors caused by MAGIC because the same error will reoccur due to increased critical path delay. A second recovery method replays errant instructions at half the clock frequency to guarantee correct execution even if dynamic variations persist. This technique corrects the timing errors, but the processor is forced to run at half the bin frequency to prevent erroneous results.

An internal node control technique to mitigate NBTI effects has been proposed in Bild et al. [2012]. In this scheme, internal node controls are inserted at the outputs of individual gates to force them to specific values when a core and accordingly its critical path are inactive. This method cannot be used against MAGIC-based attacks since in such attacks, instructions are executed continuously.

Using a variable supply voltage to characterize the physical-level properties of an IC by means of a set of nonlinear equations is proposed in Wei and Potkonjak [2011]. However, this method cannot be applied to processors that include millions of gates.

7.2. Architecture-Level NBTI Effect Mitigation Techniques

Penelope [Abella et al. 2007] applies certain input patterns that reverse the effect of aging to idle components in the processor. Penelope cannot be used to protect against MAGIC since the MAGIC instructions are executed continuously. Thus, the critical path is never idle and cannot recover.

Facelift [Tiwari and Torrellas 2008] hides the effect of aging in multicore processors through aging-driven application scheduling. Jobs that speed up aging, such as high-temperature jobs, are scheduled to run on the faster cores and the low-temperature jobs are issued to the slower cores. Thereby, the slowest core ages the slowest and the fastest core ages the quickest. Thus, the effect of aging is hidden. Facelift cannot protect against MAGIC, since once the program is scheduled, MAGIC accelerates aging on the core.

Increasing the idle ratio of the functional units of a processor to decrease NBTI effects is also considered in Corbetta and Fornaciari [2012] and Oboril et al. [2012]. The former presents an instruction allocation strategy and the latter proposes a scheduling scheme to increase the idle ratio of timing-critical components. However, these methods cannot be used to protect against MAGIC since the MAGIC instructions are executed continuously and the critical path is never idle.

7.3. Guardbanding

Guardbanding is the current industrial practice to cope with transistor aging and voltage droops [Agarwal et al. 2007]. It entails slowing down the clock frequency (i.e., adding timing margin during design) based on the worst degradation the transistors might experience during their lifetime. The guardbands ensure that enough current passes through the processor to keep it above the threshold voltage and in turn ensure that the processor functionality is intact for an average period of 5 to 7 years [Tiwari and Torrellas 2008]. However, inserting wide guardbands degrades performance and increases energy consumption. Hence, processor design companies usually have small guardbands, typically 10% [Agarwal et al. 2007]. However, the MAGIC-based attack can deteriorate the critical path by 11% and cause erroneous results in 1 month.

7.4. OS Context Switching and Timer Interrupts

Context switching is an essential feature of a multitasking operating system. A context switch is the process of storing/restoring the state (context) of a process so that execution can be resumed from the same point at a later time. This enables multiple processes to share a single CPU. Every time a context switch occurs, the MAGIC state is lost. Hence, it makes the proposed attack ineffective. Similarly, the OS's task scheduler periodically interrupts execution to reschedule the priorities of running processes. During every OS timer interrupt, the MAGIC state is lost. However, as mentioned in Section 2.2, in the MAGIC-based attack, the attacker has total control over the system and the OS such that the OS does not perform a context switch or timer interrupt.

7.5. Core Frequency Scaling

In order to conserve power, the core operates at different frequencies depending on the amount of activity. If the core is active and busy doing something, it will run at a higher frequency (turbo). When idle, it will run at a lower frequency (unturbo). When running at lower frequencies, such as $f_{max}/2$, the increase in critical path delay due to execution of the MAGIC program will not cause timing errors. This is because of the longer clock period at lower frequencies. These core frequency turbo/unturbo decisions are made by the OS based on the configuration made during the BIOS setting. Since the OS is under the attacker's control, for the warranty attack and hardware backdoor attack, the attacker can turn off this feature and successfully launch the proposed attack. For the planned obsolescence attack, the attacker benefits from frequency scaling and making the device slower. Therefore, for such attack, the attacker turns on this feature.

7.6. Monitoring CPU Utilization

A straightforward solution to detect the MAGIC-based attack is to monitor the CPU utilization. Since the malicious program needs to run for several weeks, it is very easy to detect the MAGIC attack. However, in the MAGIC-based warranty attack, the user is the attacker and simply ignores the monitors or disables them. For the other two attack scenarios, the manufacturing company launches the attack and hence disables the CPU monitoring feature.

7.7. Dynamic Integrity Checking Techniques

Dynamic integrity checking algorithms investigate the correct operation of a processor during its runtime [Austin 1999; Rotenberg 1999; Kasbekar and Das 2001]. DIVA [Austin 1999] compares the output of the execution unit with the output of a robust on-chip functional unit during the normal operation of the processor. In case of a discrepancy, a roll-back is performed and the processor state is changed to the state before the error occurs and the faulty instruction is re-executed. Although DIVA can detect the MAGIC attack, it can't prevent it since the effect of the MAGIC attack is not transient. Kasbekar and Das [2001] present a selective check-pointing and roll-back algorithm to recover from the transient faults. Similar to DIVA, this method cannot prevent MAGIC since MAGIC effects are not transient.

AR-SMT [Rotenberg 1999] is another dynamic integrity checking scheme that employs redundant multithreading without lockstepping. It combines program-level time redundancy and instruction re-execution to provide fault tolerance. However, AR-SMT can only be used to recover from transient faults. Indeed, it cannot prevent MAGIC-based attacks since both the primary and redundant threads produce the same incorrect output.

7.8. Pipeline Flush

Flushing the pipeline will modify the architectural and physical state of the processors. Therefore, at first glance, one may consider periodically flushing the pipeline as a useful method to prevent malicious aging. In this case, when program execution resumes, the component being aged would not be in the MAGIC state. Thus, although the MAGIC instruction is repeatedly executed, the critical path would not be placed under stress. However, this technique cannot prevent MAGIC attack. In practice, by applying a small modification to the MAGIC program, the attacker can prevent such countermeasures. In fact, if the MAGIC program is located in a larger loop, then the flip-flops will periodically place in the MAGIC state and therefore the MAGIC-based attack cannot be prevented. As an example, consider the MAGIC program shown in Table III. By placing all 27 instructions shown in that table in a larger loop, pipeline flushing will not be helpful in preventing the MAGIC-based attacks. Accordingly, the MAGIC program crafted to maliciously attack the SPARC processor (Table III) should include such loop, but for the sake of space, we avoid representing the MAGIC program with such loop in another table.

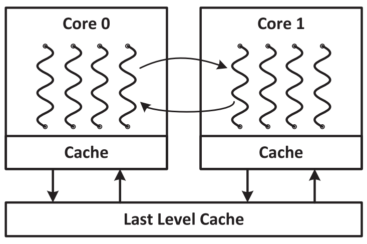
8. PREVENTING MALICIOUS AGING

Protection mechanisms such as those described in Section 7 can only detect an erroneous computation after the processor has been maliciously aged but cannot prevent the malicious aging process. For a MAGIC-based attack to be successful, it is crucial for the processor to be in the MAGIC state while the MAGIC instructions are executed repeatedly for prolonged periods. If a defense mechanism can prevent any of these from happening, the proposed attack can be thwarted.

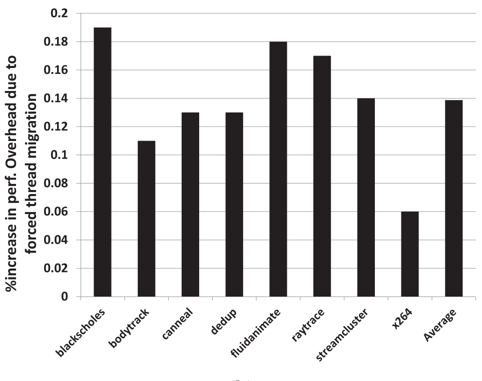
A straightforward solution to thwart the MAGIC attack is to abort execution when a particular instruction is continuously executed for a certain number of times. However, this solution results in false-positives for certain types of benchmarks that repeatedly apply certain instructions. Also, it is not appropriate for SIMD (Single Instruction Multiple Data) processors, where the same instruction is executed several times. Another simple solution is to modify the decoder such that it generates semantic equivalents for MAGIC instructions. For instance, a subtract instruction is decoded as an add instruction, but one of the operands is the two's complement of the original operand. This approach fails when the attacker attempts to age the near-critical paths instead of the critical path. In this section, we propose two techniques that void the requirements of the MACIC-based attack and prevent malicious aging in processor cores.

8.1. Thread Migration

Thread migration is usually performed to get the best performance/watt or reduce the energy delay product in heterogeneous processors [Saripalli et al. 2011]. However,



(a)



(b)

Fig. 12. (a) Illustration of thread migration. (b) Performance overhead of periodic thread migration.

thread migration can also prevent malicious aging. Periodic migration of the threads executing on one core to another core causes the physical state to change from the MAGIC state. Thus, instructions that are executed after thread migration will not accelerate the aging of the transistors.

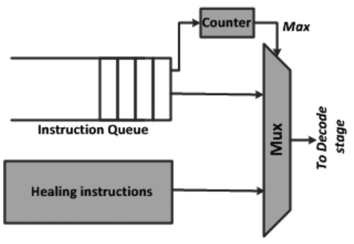
Thread migration is expensive. It involves transferring the dirty cache lines from one private cache to the other through the shared cache as shown in Figure 12(a). The architectural state of the registers must be transferred too. If performed too often, it increases the performance overhead.

We modified the gem5 cycle accurate simulator [Binkert et al. 2011] to implement the previous thread mitigation methodology. We implemented periodic thread migration in a two-core SPARC processor, with each core containing four threads, and evaluated the performance overhead using benchmarks from the SPEC CPU2006 suite. Each core is configured as shown in Table II. Periodically, all the threads from core 0 are migrated to core 1 and vice versa. In this article, we assumed a core-to-core migration latency of 400 cycles. We assume that the threads from one core are forced to migrate to the other core after 2 billion instructions have been executed. To get the highest performance overhead possible, we assume that all the lines in the cache are dirty; that is, all the lines must be transferred to the other core. The performance impact was evaluated using workloads from the multithreaded PARSEC benchmark suite. The average performance overhead was 0.14%. Figure 12(b) illustrates the performance overhead of forced thread migration.

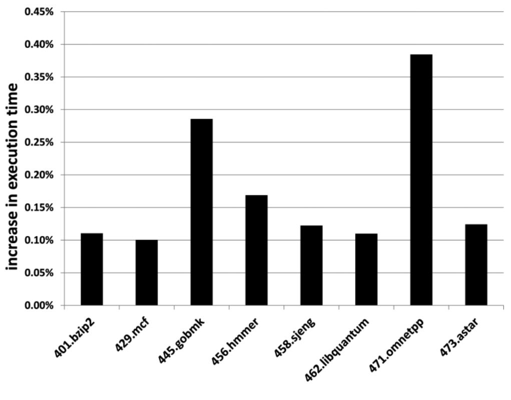
8.2. Application of Inverse Patterns

One approach to thwart the MAGIC attack is to periodically allow the transistors in the critical path to recover. This can be achieved by issuing *healing instructions*. Healing instructions generate input patterns at the critical path that cause the transistors to recover from NBTI stress. Periodic execution of these instructions has two benefits: (1) it results in a loss of the MAGIC state, and (2) it causes the transistors to recover.

The healing patterns can be obtained in a manner similar to that used for generating the patterns that stress the transistors. To generate the healing patterns, we first target each gate in the critical path and find a set of patterns that apply a “1” to their input. Using a minimum-set-covering algorithm (Section 4.1), we select a minimal set of vectors that put all the gates in the critical path under NBTI recovery. The instructions that generate these patterns are then generated in a manner similar to that used for crafting MAGIC instructions. The healing instructions must be crafted such that their execution does not affect the architectural state of the executing program.



(a)



(b)

Fig. 13. (a) Microarchitecture support to apply inverse patterns to the gates of the critical path. Shaded blocks indicate the new components. (b) Performance overhead due to periodic application of inverse MAGIC instructions.

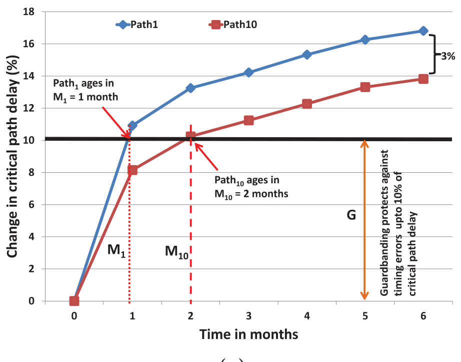
Figure 13(a) shows the required microarchitectural modifications to support periodic recovery. A counter is added between the instruction queue (IQ) and the instruction decode stage to keep track of the number of issued instructions. The *Max* signal is set when the counter reaches its maximum value. When set, *Max* chooses the healing instructions instead of instructions from the IQ. Healing instructions are precalculated, stored on-chip, and periodically executed. Figure 13(b) shows the performance overhead incurred by SPEC CPU2006 benchmarks on a SPARC processor if a period of 2 billion instructions is chosen and the healing instructions are applied for 1,000 cycles. The average performance overhead is 0.18%.

Note that in a circuit, several paths may have approximately the same delay. In such case, if a near-critical path is targeted for aging instead of the critical path, the MAGIC attack is still feasible. However, since the healing patterns have been generated to target the critical path, applying such patterns may not be useful in thwarting the attack. One may think of monitoring all paths during the operation of the circuit to detect the degraded path and to apply healing patterns related to the detected path. However, due to the large number of paths in a circuit, it is not feasible to monitor all paths. An alternative solution would be a hybrid scheme including applying healing instructions and periodic thread migration. In this case, if the critical path has been targeted by MAGIC, periodically applying healing patterns and thread migration both thwart the attack. Otherwise, if a near-critical path has been targeted by MAGIC, then the thread migration would help in recovery.

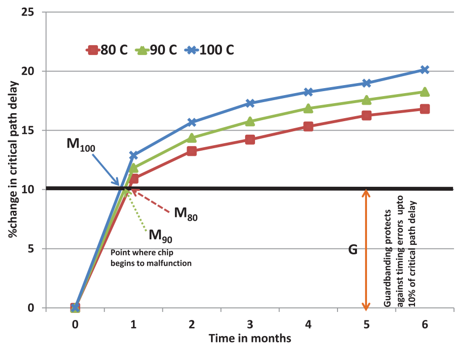
9. DISCUSSION

Can the device be used during the attack period? In a MAGIC-based attack, only one core is targeted and executes the MAGIC program. Hence, the device is functional during the attack period and can be used normally. However, it is slower since the targeted core is busy.

Do process variations affect MAGIC? The postmanufacturing critical path may differ from the design-time critical path due to process variations. After manufacturing, when the MAGIC program is executed, the design-time critical path will age and become longer than the manufacture-time critical path. We observed that the top 10 longest paths in the E stage of OpenSPARC were very close to each other (2% difference). Figure 14(a) shows the change in critical path delay for the longest path



(a)



(b)

Fig. 14. (a) Effect of process variations on changing the delay of the longest (Path1) and 10th longest path (Path10). Path1 produces erroneous results after 1 month and Path10 after 2 months. (b) Effect of temperature on changing the delay of the longest path.

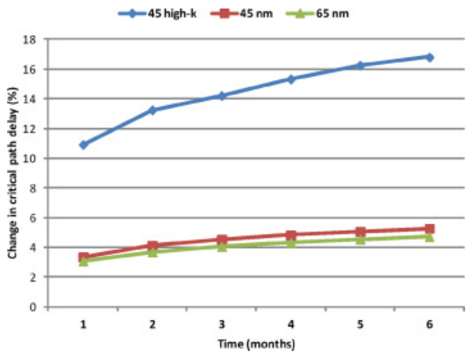


Fig. 15. Comparing the effect of MAGIC-based attack on SPARC E stage using different fabrication technologies.

(Path1) and the 10th longest path (Path10). The delay of Path1 increases by 10% in less than 1 month, and the delay of Path10 increases by 10% in 2 months. The difference between the delay change of these paths is just 3% after 6 months. After manufacturing, if the critical path changes, we still age the design-time critical path. It might take slightly longer for this path to bypass the protection offered by guardbanding (an extra month in the case of Path1 and Path10), but it will eventually produce erroneous results.

How does the technology impact MAGIC? As discussed in Section 1, even though tremendous efforts have been spent to improve the fabrication process, the impact of NBTI on circuit performance has become severe, especially after the introduction of high-k gate dielectrics since the 45nm technology node [Wang et al. 2010]. Figure 15 shows the magnitude of performance degradation of the E stage of the SPARC processor fabricated using different technologies when MAGIC instructions are applied to this microprocessor for 1 to 6 months at the operating temperature of 80°C. The results confirm the severity of NBTI degradation in high-k technologies.

Is layout needed to create the MAGIC program? The expert attacker may use the netlist or layout of the processor to identify the critical path and the input patterns that place such path under NBTI stress. However, having access to the netlist will suffice. In practice, layout synthesis tools may insert a number of buffers in the netlist, but inserting buffers does not change the aging patterns. On the other hand, by inserting buffers, the postlayout critical path may differ from the postnetlist critical path. But, as discussed earlier, we can still age the design-time critical path and wear out the device.

Is it possible to generate MAGIC patterns for other pipestages? MAGIC patterns can be generated for any pipeline stage. For OpenSPARC, we chose the E stage since it includes the critical path of the entire chip.

How does temperature impact MAGIC? As discussed in Section 3, the change in threshold voltage, and in turn the critical path delay, is directly affected by the temperature. When operated at higher temperatures, the devices age faster. Thus, temperature is another knob for the attacker. Figure 14(b) shows the impact of operating temperature on the increase of critical path delay. When the MAGIC program is executed at an operating temperature of 100°C, the critical path delay increases by 12.9% after 1 month, 15.68% after 2 months, and 20.13% after 6 months. The difference between the change in critical path delay when the temperature is 80°C and 100°C is 2.02% after 1 month and 3.32% after 6 months. Note that, in the MAGIC-based attack, the malicious program is executed on the processor continuously and so the *"Operating Temperature"* increases rapidly.

Do pipeline bubbles from non-MAGIC instructions reduce the aging effects? Non-MAGIC instructions are selected such that they do not modify the MAGIC state. The flip-flops lead the critical path to hold their values, and the inputs feeding the gates in the critical path do not change. Thus, bubbles due to non-MAGIC instructions do not allow the gates in the critical path to recover. The recovery due to bubbles caused by loop-ending branches is negligible.

10. CONCLUSION AND FUTURE WORK

This article presented MAGIC, a technique to maliciously accelerate the NBTI-based aging of cores. By analyzing the structural information of the processor, a sequence of assembly instructions that accelerate the aging process was developed. A program consisting of these instructions was crafted. By executing this application, the core is maliciously aged and the chip fails sooner than expected. We demonstrated the MAGIC attack on the OpenSPARC processor and showed that the critical path delay can be increased by 10.92% in just 1 month, bypassing the protection offered by traditional aging tolerance techniques such as guardbanding or resilient microprocessors. We proposed two microarchitectural modifications to prevent malicious aging of processors.

While MAGIC was demonstrated on the E stage of a SPARC processor, it can be customized to other processor architectures such as ARM and MIPS, whose source code is publicly available. While SPARC is an in-order processor, we believe that MAGIC can also be applied to out-of-order processors. We are currently investigating the feasibility of MAGIC on out-of-order processors. We are also working on generating patterns to accelerate aging due to other mechanisms such as hot carrier injection and gate oxide breakdown. In this article, we showed the feasibility of the MAGIC-based attack using simulation tools. Demonstrating the impact of the proposed attack on fabricated chips is our next trivial direction in this research. Finally, we are working on hiding the MAGIC program within another seemingly innocuous application, which, when executed by even nonmalicious users, causes chip failure.

REFERENCES

Jaume Abella, Xavier Vera, and Antonio Gonzalez. 2007. Penelope: The NBTI-aware processor. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 85–96. DOI:http://dx.doi.org/10.1109/MICRO.2007.11

Hamed Abrishami, Safar Hatami, Behnam Amelifard, and Massoud Pedram. 2008. NBTI-aware flip-flop characterization and design. In *Proceedings of the ACM Great Lakes symposium on VLSI (GLSVLSI'08)*. 29–34.

Mridul Agarwal, Bipul C. Paul, Ming Zhang, and Subhasish Mitra. 2007. Circuit failure prediction and its application to transistor aging. In *Proceedings of the IEEE VLSI Test Symposium*. 277–286. DOI:http://dx.doi.org/10.1109/VTS.2007.22

Muhammad Ashraful Alam, Haldun Kufluoglu, Dhanoop Varghese, and Souvik Mahapatra. 2007. A comprehensive model for PMOS NBTI degradation: Recent progress. *Microelectronics Reliability* 47, 6 (2007), 853–862.

Todd Austin. 1999. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 196–207. DOI:http://dx.doi.org/10.1109/MICRO.1999.809458

Sarvesh Bhardwaj, Wenping Wang, Rakesh Vattikonda, Yu Cao, and Sarma Vrudhula. 2006. Predictive modeling of the NBTI effect for reliable design. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 189–192. DOI:http://dx.doi.org/10.1109/CICC.2006.320885

David Bild, Gregory Bok, and Robert Dick. 2009. Minimization of NBTI performance degradation using internal node control. In *Proceedings of IEEE Design Automation and Test in Europe*. 148–153.

David R. Bild, Robert P. Dick, and Gregory E. Bok. 2012. Static NBTI reduction using internal node control. *ACM IEEE Transactions on Design Automation Electronic Systems* 17, 4 (2012), 45.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Computer Architecture News* 39, 2 (2011), 1–7. DOI:http://dx.doi.org/10.1145/2024716.2024718

Keith A. Bowman, James W. Tschanz, Shih-Lien Lu, Paolo A. Aseron, Muhammad M. Khellah, Arijit Raychowdhury, Bibiche M. Geuskens, Carlos Tokunaga, Chris Wilkerson, Tanay Karnik, and Vivek K. De. 2011. A 45 nm resilient microprocessor core for dynamic variation tolerance. *Journal of Solid-State Circuits* 46, 1 (2011), 194–208.

Soonyoung Cha, Chang-Chih Chen, Taizhi Liu, and Linda S. Milor. 2014. Extraction of threshold voltage degradation modeling due to Negative Bias Temperature Instability in circuits with I/O measurements. In *Proceedings of VLSI Test Symposium (VTS'14)*. 1–6.

Ashutosh Chakraborty and David Z. Pan. 2011. Controlling NBTI degradation during static burn-in testing. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 597–602.

Srini Chakravarthi, Anand Krishnan, Vijay Reddy, Chuck Machala, and Srikanth Krishnan. 2004. A comprehensive framework for predictive modeling of negative bias temperature instability. In *Proceedings of the Reliability Physics Symposium*. 273–282. DOI:http://dx.doi.org/10.1109/RELPHY.2004.1315337

Simone Corbetta and William Fornaciari. 2012. NBTI mitigation in microprocessor designs. In *Proceedings of ACM Great Lakes Symposium on VLSI*. 33–38. Retrieved from http://dblp.uni-trier.de/db/conf/glvlsi/glvlsi2012.html#CorbettaF12.

Cyanogen. Cyanogenmod Forum. Retrieved from http://forum.cyanogenmod.com/.

Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 7–18. DOI:http://dx.doi.org/10.1109/MICRO.2003.1253179

Kai Feng, Thomas Fleischman, Ping-Chuan Wang, Xiaojin Wei, and Zhijian Yang. May 5, 2011. On-chip accelerated failure indicator. U.S. Patent 3247503. (May 5, 2011).

Tibor Grasser, Hans Reisinger, Paul-Jürgen Wagner, Franz Schanovsky, Wolfgang Gös, and Ben Kaczer. 2010. The time dependent defect spectroscopy (TDDS) for the characterization of the bias temperature instability. In *Proceedings of the International Reliability Physics Symposium (IRPS'10)*. 16–25.

Benson Inkley. 2008. Digital thermal sensors and the DTS based thermal specification for the Intel Core i7 Processor. Intel Developer Forum (IDF). Retrieved from http://lenry.atw.hu/tjmax.pdf.

Mangesh Kasbekar and Chita R. Das. 2001. Selective checkpointing and rollbacks in multithreaded distributed systems. In *Proceedings of the International Conference on Distributed Computing Systems*. 39–46. DOI:http://dx.doi.org/10.1109/ICDSC.2001.918931

Seyab Khan, Nor Zaidi Haron, Said Hamdioui, and Francky Catthoor. 2011. NBTI monitoring and design for reliability in nanoscale circuits. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. 68–76. DOI:http://dx.doi.org/10.1109/RELPHY.2004.1315337

Haldun Kufluoglu and Muhammad Ashraful Alam. 2007. A generalized reaction-diffusion model with explicit H-H2 dynamics for Negative-Bias Temperature-Instability (NBTI) degradation. *IEEE Transactions on Electron Devices* 54, 5 (2007), 1101–1107. DOI:http://dx.doi.org/10.1109/TED.2007.893809

Halil Kükner, Moustafa Khatib, Sebastien Morrison, Pieter Weckx, Praveen Raghavan, Ben Kaczer, Francky Catthoor, Liesbet Van Der Perre, Rudy Lauwereins, and Guido Groeseneken. 2014. Degradation analysis of datapath logic subblocks under NBTI aging in FinFET technology. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED'14)*. 473–479.

Yongho Lee and Taewhan Kim. 2011. A fine-grained technique of NBTI-aware voltage scaling and body biasing for standard cell based designs. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 603–608. DOI:http://dx.doi.org/10.1109/ASPDAC.2011.5722260

Bao Liu and Chiung-Hung Chen. 2014. Testing, diagnosis and repair methods for NBTI-induced SRAM faults. In *Proceedings of the International Conference on IC Design and Technology*. 1–4.

Yinghai Lu, Li Shang, Hai Zhou, Hengliang Zhu, Fan Yang, and Xuan Zeng. 2009. Statistical reliability analysis under process variation and aging effects. In *Proceedings of the IEEE/ACM Design Automation Conference*. 514–519.

Souvik Mahapatra, Dipankar Saha, Dhanoop Varghese, and Bharat. Kumar. 2006. On the generation and recovery of interface traps in MOSFETs subjected to NBTI, FN, and HCI stress. *IEEE Transactions on Electron Devices* 53, 7 (2006), 1583–1592. DOI:http://dx.doi.org/10.1109/TED.2006.876041

Christopher Mims. 2013. If It Aint Broke, Of Course Apple Is Engaging in Planned Obsolescence. Retrieved from http://qz.com/141297/of-courseapple-is-engaging-in-planned-obsolescence.

Predictive Technology Model. 2007. Introduction. Retrieved from http://ptm.asu.edu/.

Fabian Oboril, Farshad Firouzi, Saman Kiamehr, and Mehdi Baradaran Tahoori. 2012. Reducing NBTI-induced processor wearout by exploiting the timing slack of instructions. In *Proceedings of Hardware/Software Codesign and System Synthesis*. 443–452. Retrieved from http://dblp.uni-trier.de/db/conf/codes/codes2012.html#OborilFKT12.

Oracle. 2006a. OpenSPARC T1. Retrieved from http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html.

Oracle. 2006b. OpenSPARC T1 Microarchitecture Specification.

Catherine Rampell. 2013. Cracking the Apple Trap. Retrieved from http://www.nytimes.com/2013/11/03/magazine/why-apple-wants-to-bust-your-iphone.html.

Vikram G. Rao and Hamid Mahmoodi. 2011. Analysis of reliability of flip-flops under transistor aging effects in nano-scale CMOS technology. In *Proceedings of IEEE International Conference on Computer Design*. 439–440.

Rosana Rodriguez, James Stathis, and Barry Linder. 2003. Modeling and experimental verification of the effect of gate oxide breakdown on CMOS inverters. In *Proceedings of IEEE International Reliability Physics Symposium*. 11–16. DOI:http://dx.doi.org/10.1109/RELPHY.2003.1197713

Matt Rosoff. 2012. Microsoft: Apple Makes Old iPhones 'Unusably Slow' on Purpose. Retrieved from http://www.businessinsider.com/microsoft-apple-makes-old-iphones-unusably-slow-on-purpose-2012-3.

Eric Rotenberg. 1999. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of the International Symposium on Fault-Tolerant Computing*. 84–91. DOI:http://dx.doi.org/10.1109/FTCS.1999.781037

Subhendu Roy and David Z. Pan. 2014. Reliability aware gate sizing combating NBTI and oxide breakdown. In *Proceedings of the International Conference on VLSI Design and Embedded Systems*. 38–43.

Dipankar Saha, Dhanoop Varghese, and Souvik Mahapatra. 2006. Role of anode hole injection and valence band hole tunneling on interface trap generation during hot carrier injection stress. *IEEE Electron Device Letters* 27, 7 (2006), 585–587. DOI:http://dx.doi.org/10.1109/LED.2006.876310

Smruti Sarangi, Brian Greskamp, Radu Teodorescu, Jun Nakano, Abhishek Tiwari, and Josep Torrellas. 2008. VARIUS: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing* 21, 1 (2008), 3–13. DOI:http://dx.doi.org/10.1109/TSM.2007.913186

Vinay Saripalli, Guangyu Sun, Asit Mishra, Yuan Xie, Suman Datta, and Vijaykrishnan Narayanan. 2011. Exploiting heterogeneity for energy efficiency in chip multiprocessors. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 1, 2 (2011), 109–119. DOI:http://dx.doi.org/10.1109/JETCAS.2011.2158343

Dieter K. Schroder. 2007. Negative bias temperature instability: What do we understand? *Microelectron Reliability* 47, 6 (2007), 841–852.

Ozgur Sinanoglu, Naghmeh Karimi, Jeyavijayan Rajendran, Ramesh Karri, Yier Jin, Ke Huang, and Yiorgos Makris. 2013. Reconciling the IC test and security dichotomy. In *Proceedings of European Test Symposium (ETS'13)*. 1–6.

Hunter Skipworth. 2012. The Myth of the Sony Kill Switch. Retrieved from http://www.telegraph.co.uk/technology/news/7054587/Themyth-of-the-Sony-kill-switch.html.

Ketul B. Sutaria, Jyothi B. Velamala, Athul Ramkumar, and Yu Cao. 2015. Compact modeling of BTI for circuit reliability analysis. In *Circuit Design for Reliability*. Springer, 93–119.

Abhishek Tiwari and Josep Torrellas. 2008. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 129–140. DOI:http://dx.doi.org/10.1109/MICRO.2008.4771785

Randy Torrance and Dick James. 2011. The state-of-the-art in semiconductor reverse engineering. In *Proceedings of the IEEE/ACM Design Automation Conference*. 333–338.

Rakesh Vattikonda, Wenping Wang, and Yu Cao. 2006. Modeling and minimization of PMOS NBTI effect for robust nanometer design. In *Proceedings of the IEEE/ACM Design Automation Conference*. 1047–1052. DOI:http://dx.doi.org/10.1109/DAC.2006.229436

Wenping Wang, Shengqi Yang, Sarvesh Bhardwaj, Sarma Vrudhula, Frank Liu, and Yu Cao. 2010. The impact of NBTI effect on combinational circuit: Modeling, simulation, and analysis. *IEEE Transactions on Very Large Scale Integration Systems* 18, 2 (2010), 173–183. DOI:http://dx.doi.org/10.1109/TVLSI.2008.2008810

Yao Wang, Sorin D. Cotofana, and Liang Fang. 2012. Statistical reliability analysis of NBTI impact on FinFET SRAMs and mitigation technique using independent-gate devices. In *Proceedings of the International Symposium on Nanoscale Architectures (NANOARCH'12)*. 109–115.

Sheng Wei and Miodrag Potkonjak. 2011. Integrated circuit security techniques using variable supply voltage. In *Proceedings of IEEE/ACM Design Automation Conference*. 248–253.

Gil Wolrich, Edward McLellan, Larry Harada, James Montanaro, and Robert Yodlowski. 1984. A high performance floating point coprocessor. *IEEE Journal of Solid-State Circuits* 19, 5 (1984), 690–696. DOI:http://dx.doi.org/10.1109/JSSC.1984.1052209

Tim Worstall. 2013. Certainly theres planned obsolescence in Apples iKit its just not planned by Apple. Retrieved from http://www.forbes.com/sites/timworstall/2013/10/31/certainly-theres-planned-obsolescence-in-apples-ikit-its-just-not-planned-by-apple.

Teong-San Yeoh and Shze-Jer Hu. 1998. Influence of MOS transistor gate oxide breakdown on circuit performance. In *Proceedings of IEEE International Conference on Semiconductor Electronics*. 59–63. DOI:http://dx.doi.org/10.1109/SMELEC.1998.781150

Received June 2014; revised January 2015; accepted January 2015