Security Threats in Approximate Computing Systems

Pruthvy Yellu
Reliable and Secure Systems Laboratory
Department of Electrical and Computer Engineering
University of New Hampshire
py1007@wildcats.unh.edu

Novak Boskov
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering
Boston University
boskov@bu.edu

Michel A. Kinsy
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering
Boston University
mkinsy@bu.edu

Qiaoyan Yu
Reliable and Secure Systems Laboratory
Department of Electrical and Computer Engineering
University of New Hampshire
qiaoyan.yu@unh.edu

ABSTRACT

Approximate computing systems improve energy efficiency and computation speed at the cost of reduced accuracy on system outputs. Existing efforts mainly explore the feasible approximation mechanisms and their implementation methods. There is limited work that investigates the security threats brought by approximate computing. To fill this gap, we first analyze the approximate mechanisms used in approximate system, software, storage, and arithmetic circuits, and then propose potential attacks that will challenge the integrity and security of approximate systems. Some illustrative examples are provided accordingly to showcase the consequences of the proposed new attacks.

CCS CONCEPTS

• **Security and privacy → Security in hardware**; • **Hardware → Semiconductor memory**; **System-level fault tolerance**.

KEYWORDS

Approximate computing, hardware security, error resilience, image processing, DRAM, SRAM, PCM, machine learning.

ACM Reference Format:
Pruthvy Yellu, Novak Boskov, Michel A. Kinsy, and Qiaoyan Yu. 2019. Security Threats in Approximate Computing Systems. In *Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3299874.3319453

1 INTRODUCTION

Traditionally, computation correctness has been at the forefront of computer systems design concerns - both at the software and hardware levels. However, there are many applications where the final output of the computation does not need to have high accuracy or full precision. For example, the encoding of audio files from an uncompressed pulse code modulation to a lossy format, such as MP3, can tolerate a non-exact translation. The analog signal actuated through the speaker membrane under the two formats sounds nearly identical to most listeners. Similarly, many neural networks already produce probability estimates when performing classification. In the aforementioned applications, minor errors in output probabilities can often be considered negligible. Moreover, many applications can exhibit full functional correctness even when computed in an approximated fashion. Some of these applications are image processing, computer vision, video streaming, machine learning, big data analysis, sensor data processing, and computation on aged devices [16, 19]. Approximate computing has emerged as a promising approach to improve energy efficiency. As a paradigm, it allows the computation to deviate from the reference or deterministic execution behavior.

As illustrated in Figure 1, the concept of approximate computing can be implemented with four different strategies: approximate systems, approximate software, approximate storage, and approximate arithmetic circuits. At the system level, approximate computing is realized using modified architectures where approximate accelerators and programmable processors are typically adopted [13, 24, 26]. With software-level approximation, power consumption is reduced by skipping the execution of the functions in a predicted part of the code [10, 18], by relaxing the constraints on synchronization timing and handshaking [15], by leveraging domain specific knowledge to simplify application algorithms [25], or by altering the implementation of algorithms for a specific function [4]. Techniques for approximate storage include applying different refresh rates on different memory blocks [20], loosening guard band of multi-level memory cells [23], and selective voltage scaling [5]. Circuit-level approximation techniques reduce the computation resolution or

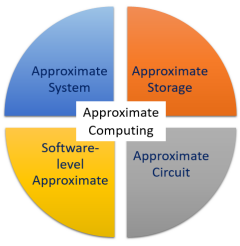

Figure 1: Approximate computing strategies.

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3319453

shorten the critical delay path to improve performance and energy efficiency [7, 12, 17].

Existing research efforts mainly focus on finding approximation mechanisms to trade off accuracy for delay/power/area. The security issues related to approximate computing are often overlooked. The work [21] initiates the investigation on the dark side of approximate computing and provides a brief analysis how a transition to the approximate computing paradigm would affect hardware security. In this work, we study the potential attack surfaces on approximate computing systems from circuit to software stacks. Indeed, approximate computing presents a particularly poignant security challenge, because it runs counter to emerging secure computer system design patterns. For example, attackers could (1) alter the original memory allocation map for precise memory and approximate memory blocks and then maliciously save critical data in the approximate memory block, which often does not have strong integrity check or authentication, (2) exploit the relaxation on design constraints (e.g., guard band) to incorrectly quantize analog input for digital critical data, or (3) tamper the memory controller to lose the original critical data (due to insufficient refresh frequency). More detailed discussions on the potential attack risks are presented in the later sections of this work.

The remainder of this work is organized as follows. In Section 2, we introduce security threats in the system-level framework for approximate computing. In Section 3, we propose the possible attack surfaces in three types of approximate storage, approximate DRAM, approximate phase change memory (PCM), and approximate SRAM. In Section 4, we briefly introduce an approximate arithmetic module and highlight the potential security threats. This work is concluded in Section 5.

2 GENERAL SYSTEM LEVEL APPROXIMATE COMPUTING FRAMEWORK

Broadly speaking, the steps in the computing system design cycle are functional description, functional specification, functional and implementation modeling, and physical implementation. The current secure computer system design patterns aim to eliminate or minimize any semantic gaps between these design steps, and implement strong module and state isolation. For example, both active side-channel attack (e.g., cache side-channel attack) and passive side-channel attack (e.g., thermal analysis) exploit gaps within these steps. Therefore, the push has been towards a greater semantic consistency between the different implementation steps to (a) avoid data leakage, (b) prevent an unauthorized party (users, processes, etc) from accessing services, resources data or discovering the existence of a critical piece of data, (c) check the integrity of computing processes, (d) monitor and control access to system resources, or (e) have predictable computing behaviors. The principle of approximate computing leaves or creates under-defined or under-specified behaviors - "Attack Gadget (AG)" - in the design at all levels. Figure 2 illustrates this problem.

Furthermore, in the light of recent micro-architecture security issues, it has become harder to claim that the undefined or under-specified behaviors of the system, which are being leveraged by approximate computing, could also be used for obfuscation to improve its resiliency. The current momentum is a move away from

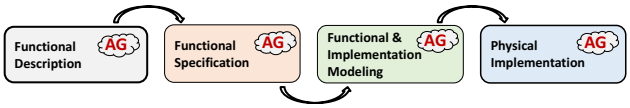

Figure 2: Computing system design cycle steps.

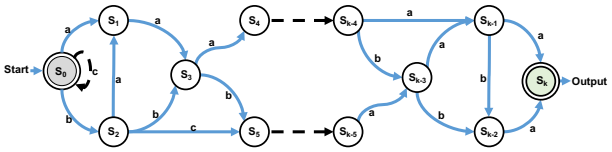

Figure 3: Finite state machine (FSM) representation of the computer system.

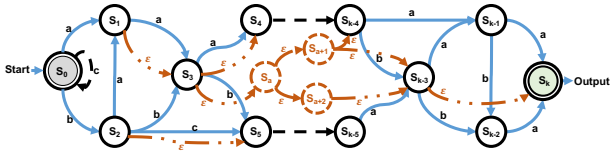

Figure 4: States and transitions illustrating the approximate nature of the computing system either at the software/algorithmic level or hardware/device level.

security by secrecy or obscurity towards a formally secure design paradigm. To illustrate this natural tension between approximation and security in computing, and without loss of generality, let us consider a simple finite state machine.

The canonical *finite state machine (FSM) - deterministic finite automaton (DFA)* definition can be described as a tuple

$$A = (Q, \Sigma, \delta, S_0, F) ,$$

where

- Q is a finite set of *states*,
- Σ is a finite *alphabet*, in this illustrative case $\Sigma = \{a, b, c, \varepsilon, \boxed{d}, \boxed{T}\}$
- $\delta : Q \times \Sigma \rightarrow Q$ is the (total) *transition function*,
- $S_0 \in Q$ is the *initial state*, and
- $F \subseteq Q$ is the set of *final states*.

The *alphabet* and the transition set can be decomposed in this manner: $\Sigma = \Sigma^s \cup \Sigma^a \cup \Sigma^v$ where

- $\Sigma^s = \{a, b, c\}$ is the set of safe and deterministic *alphabet* and associated transitions, i.e., non-approximate transitions, shown in Figure 3,
- Σ^a is the approximation *alphabet* denoted by ε - in an approximation state could represent a software or hardware level mechanism - Figure 4.
- Σ^v is the vulnerability *alphabet* set - $\Sigma^v = \{\boxed{d}, \boxed{T}\}$ - for a given system, $\boxed{T}$ denotes the set of possible attack triggers that can be used against the system, $\boxed{d}$ denotes the possible ways the attacker can manipulate the compute system when under attack - Figure 5.

From the system behavior analysis standpoint, the challenge is to decouple the approximation states from the vulnerable states. Approximation states provide a certain level of obfuscation to the attack states. Without the awareness of the states' intent, which often cannot be captured by system analysis tools, it is extremely difficult,

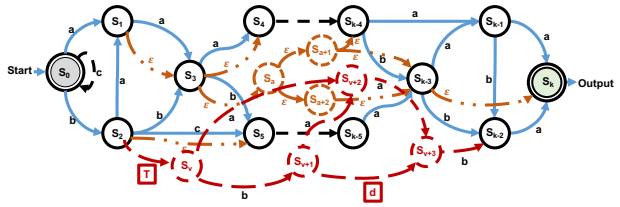

Figure 5: Approximation states and attack states ambiguity.

if not impossible, to isolate malicious states from the under-specified or under-determined or under-provisioned states - Figure 5.

3 SECURITY THREATS IN APPROXIMATE STORAGE

Power consumed by memory elements takes a large portion of system power. Approximate storage is emerging to be an effective way to achieve system-level energy efficiency. Unfortunately, approximate storage may incur new security threats over traditional precise memory.

3.1 Overview of Attacks on Approximate Storage

The use of approximate storage involves software and hardware, both of which could be the attack target. In Figure 6, we highlight some potential attack scenarios that could harm the integrity and reliability of approximate storage. The rest of this section summarizes the attack methods applicable on software and hardware for approximate storage.

To facilitate the use of approximate storage, the authors [9] add a new instruction (e.g. *LDx.a*) to the instruction set so that the compiler will know whether the destination is approximate memory or not. A compromised compiler could replace the instruction *LDx.a* with the regular instruction *LDx*, leading the system to experience unacceptable errors. The compiler and memory allocator need a memory map, which indicates the address range of different memory categories, to assign the registers, variables, and temporal storage space according to the acceptable error tolerance for each data segment. If the memory boundary between the precise memory and approximate memory is altered by an attacker, critical data could be stored in the approximate memory blocks. As a result, adversary may have an easy way to manipulate the data without being captured. One type of approximate storage achieves energy efficiency by relaxing the guard band for multi-level memory cells. The number of writing iterations guarantees the success of write operation. If an attacker maliciously modifies the number of writing iterations, the data stored in the approximate memory will be changed stealthily.

In the hardware component of approximate storage, the precision flag bit could be misinterpreted if the instruction decoder is sabotaged. Once the precision flag is ignored, the critical or approximate data will be sent to a wrong place. Some volatile memory components retains the same value only when the memory is refreshed periodically. A slower refresh rate would lead to the loss of the originally saved data. Any tampering on the refresh counter and refresh enable signal could ruin the memory content. Analog input needs to be converted to digital bits before it is saved in

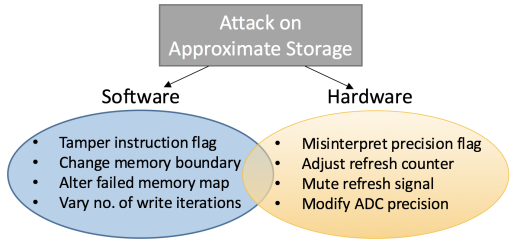

Figure 6: Potential attacks on approximate storage.

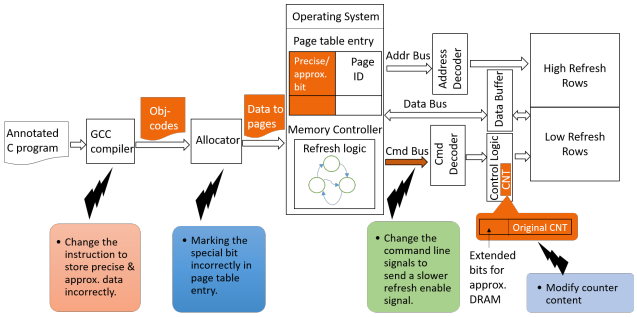

Figure 7: Process flow of approximate DRAM storage.

memory. Some types of approximate storage based on emerging memory vary the precision of Analog-to-Digital Converter (ADC) to reduce energy consumption. If an attacker takes charge of the ADC precision adjuster, the critical data will be saved with a lower precision than it should be, thus harming the system.

3.2 Attack on Approximate DRAM

3.2.1 Approximation Mechanism. A basic DRAM cell consists of an access transistor and a capacitor. When the DRAM cell is in the idle state, the capacitor starts to discharge. If the idle DRAM cell is not refreshed periodically on time, the logic value stored in that DRAM cell will be lost. As reported in [1], the power consumption for memory refresh is almost 50% of the total power consumed by DRAM. Moreover, write and read operations are prohibited during the period of memory refreshing. This fact limits the throughput of DRAM. To improve energy efficiency and throughput, approximate DRAM selectively reduces the refresh rate. The DRAM segment for precise storage is refreshed at every standard interval of time. In contrast, the DRAM blocks for approximate storage is refreshed with a slower rate, incurring storage error on some memory cells.

3.2.2 Potential Attack Surfaces. Despite of energy savings, approximate DRAM opens new exploration space for an adversary to implement new unique attacks. In Figure 7, we point out the vulnerable spots, where adversary could execute attacks in the process of storing and maintaining data in a memory with mixed precise and approximate DRAMs. Here, we assume that the application developer is trusted but the software and hardware that approximate DRAM involves are not trusted. Users annotate approximate and precise data in the high level description of the application. Main attacks on the software and hardware are summarized in Figure 7.

The compiler determines the specific instructions to save precise and approximate data. If attackers interchange the instruction for precise data (e.g. LDx) to the one for approximate data (e.g. LDx.a), the critical data for precise storage will be transferred to the approximate memory blocks. The memory allocator decides which specific memory pages to store the precise and approximate data, respectively, and marks the pages containing approximate data with a special bit in the page table entry. If an attacker can manipulate the allocator, the memory pages will be purposely marked with a different indication bit. For instance, loading precise data to the approximate memory will affect the accuracy of the final computational output. The operating system converts the logical address into physical address and guides the memory controller to issue appropriate control signals over the command bus. Once the refresh logic is compromised, the refresh interval for the approximate DRAM could be increased or reduced. The increased refresh rate results in unnecessary power consumption. The decreased refresh rate causes the memory block experience more error bits than the upper limit that the error correction circuit can handle. Interfering the command bus could cause timing violations. The counter *CNT* inside the control logic unit provides the next memory address to be refreshed in the self refresh mode [22]. If the counter hardware is tampered by an attacker, the DRAM blocks for precise and approximate storage will both suffer from the reduced memory refresh. Although precise DRAM typically has error correction circuit, the frequent usage of error correction will slow down the memory access speed and dramatic power increase. Due to the inappropriate refresh frequency, the approximate DRAM may not be able to achieve the acceptable precision.

3.2.3 Attack Example. A memory composed of precise and approximate DRAM cells will be refreshed in two modes: auto refresh and self refresh [22]. In the auto refresh mode, the external memory controller issues refresh commands. In the self refresh mode, the refresh commands are generated inside the DRAM module, and the counter in the control logic unit keeps track of the next address to be refreshed. The work [22] proposes to extend the counter CNT shown in Figure7 by adding $log_2(n)$ bits, where n represents the ratio of standard refresh frequency to the lower refresh frequency for approximate DRAM. Once the MSB bits meet a pre-defined condition (e.g. all zeros), the approximate DRAM will be refreshed.

To sabotage the data saved in the precise DRAM, the attacker may only need to manipulate the configuration signal. As shown in Figure 8, once the configuration signal goes to high, the extended MSB bits of the refresh counter starts to increase. As a result, the memory refresh enable signal drops down earlier. Since the precise DRAM is not refreshed with the regular refresh rate, the data stored in the precise DRAM may lose charge and result in an uncorrectable error. We conducted a set of experiments to show the impact of attack induced by the reduced refresh rate on approximate DRAM storage. The image *Lena* shown in Figure 9(a) stored in the precise DRAM blocks was refreshed with an interval of 64ms before attack. When the configuration signal shown in Figure 8 is pulled up due to attack, the refresh happens every 60 seconds. We adopted the error rate reported in [11] and accordingly injected bit errors to the RGB values of the image pixels. Bit-flip and stuck-at-0 error models were applied in our error injection. The targeted image pixels could

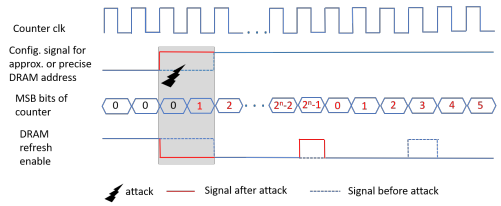

Figure 8: Timing diagram for the DRAM with incorrect refresh frequency due to tampered memory counter.

(a) (b) (c) (d)

Figure 9: Impact of incorrect refresh rates on the output of approximate DRAM. (a) original, (b) random attack, (c) regional attack(bit-flip model), and (d) regional attack (stuck-at-0 model).

be random or around in a region. Figures 9(b) and (c) show the impact of the random and regional errors on the recovered image, respectively. In this example, the attacker blurs the important characteristics of the critical image by manipulating the configuration signal for DRAM refreshing. If such an image is used for authentication, attacks on the approximate DRAM will significantly reduce the effort to pass authentication.

3.3 Attack on Approximate Phase Change Memory

3.3.1 Approximation Mechanism. Phase Change Memory (PCM) is a non-volatile memory. The voltage applied on the memory cell will turn the PCM material into either amorphous or crystalline. As the analog resistance presented by the PCM material can be quantized into multiple levels, the PCM is often observed as multi-level memory cells. PCM has been used as a media to implement approximate storage [23]. The main principle of approximation here is to reduce the size of guard band. As shown in Figure 10, the analog resistance is quantized into four levels to represent 2-bit information stored in a single PCM memory cell, and the approximate PCM relaxes the noise margin from a small $T_{precise}$ to a large T_{approx} (i.e. the guard band of approximate PCM is less than that of precise PCM). To save power consumption, the number of writing iterations for approximate PCM is less than that for precise PCM [23]. Although the approximation technique [23] improves the speed of writing by 1.7× over the precise PCM, the reduced guard band and the less number of writing iterations make the approximate PCM vulnerable to security threats.

3.3.2 Potential Attack Surfaces. Based on the PCM writing procedure introduced in [23], we highlight the key steps of writing operation for approximate PCM in Figure 11. The possible attacks along the writing flow are proposed on the right side of Figure 11. The parameter T_{approx} defines the noise margin. If the initialization of T_{approx} is compromised, a precise PCM block will be turned

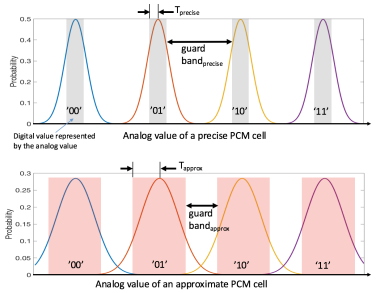

Figure 10: Comparison of precise and approximate multi-level PCM cells.

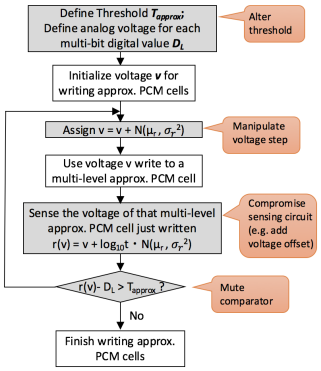

Figure 11: Flowchart of PCM writing procedure with potential security attacks.

into an approximate memory block. Consequently, the integrity of the critical data saved in the precise PCM will be affected. In the iterative writing procedure, the parameters μ_r and σ_r are the mean and standard deviation of the error effect (due to insufficient writing voltage). The writing voltage v increases incrementally in each writing iteration. When the noise function $N(\mu_r, \sigma_r^2)$ is underestimated or overestimated, the voltage step will be manipulated by adversary and thus writing operation will fail. The determination of the number of writing iterations relies on the voltage difference sensed by the reading operation. If a voltage offset is stealthily introduced to the sensing logic, the content stored in the PCM can be modified arbitrarily. Or, the voltage comparator $(r(v) - D_L)$ is muted by the attacker so that he/she could overwrite confidential information, which is supposed to be saved in a precise PCM block.

3.3.3 Attack Example. We model the resistance distribution probability for the multi-level PCM cells shown in Figure 10, and set the threshold T in a range of 60% to 90%. We consider the 90% case as precise PCM, and the other cases are approximate PCM. The attack implemented in this example consists of adding an offset to the output voltage from the PCM cells. Due to the voltage offset Δnoise, the logic content saved in the PCM cell will be misinterpreted. When the resistance cannot be converted to the correct multi-bit state, we consider this to be a quantization error. As shown in Figure 12, the approximation mechanism incurs a higher quantization error rate than the precise case and the error rate further increases with the increasing noise. As presented in Figures 12(a) and (b), the imprecise writing makes the PCM more sensitive to a small noise. If the

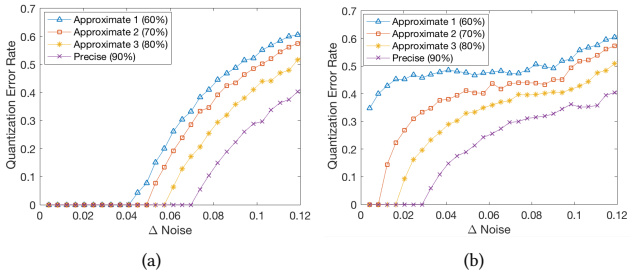

Figure 12: Quantization error rate of a precise/approximate PCM cell written with (a) sufficient and (b) insufficient number of writing iterations.

write and read operations in the approximate PCM are executed, the approximate PCM may suffer from more quantization errors.

3.4 Attack on Approximate SRAM

3.4.1 Approximation Mechanism. Approximate SRAM trades off accuracy and energy efficiency by scaling supply voltage. In the work [2], 8T SRAM cells are used to store the sign and exponent bits of floating-point numbers as 8T SRAM cells have better resilience than 6T SRAM cells against the noise. In contrast, 6T SRAM cells are utilized to save mantissa bits. Thus, even if the 6T SRAM cells experience read or write failures due to the lower supply voltage, the effect of scaled voltage on the SRAM quality is acceptable by some applications (e.g., image processing). The work [14] follows the similar approximate mechanism by replacing 8T SRAM cells with oversized 6T SRAM cells. A dual supply voltage technique is proposed in [3], where high and low voltages are applied to the precise and approximate SRAM cells, respectively. Alternatively, error correction code is implemented in the precise SRAM blocks to compensate for the errors induced by the overscaled supply voltage [6].

3.4.2 Potential Attack Surfaces. Attackers could exploit the overscaled supply voltage technique in the work [6] to introduce more errors than what can be tolerated by the error correction code (ECC) available in the memory. The SRAM cells for the storage of ECC check bits may be the potential attack surface. Assume a hybrid precise and approximate SRAM is used to implement a cache memory, which is managed by a cache controller with the pre-knowledge of the hybrid SRAM map. If an attacker sabotages the SRAM map in the cache controller, critical data from the main memory could be loaded to the approximate SRAM cells, which have limited (or no) error correction capability. Or, data from the compromised main memory will be successfully transferred to the CPU via the cache.

4 SECURITY THREATS IN APPROXIMATE CIRCUIT

Circuit-level approximation techniques facilitate the system to reduce the computational power at the cost of decreased precision on the system output. In this section, we use an approximate adder as an illustrative example to present the potential security threats in approximate circuits.

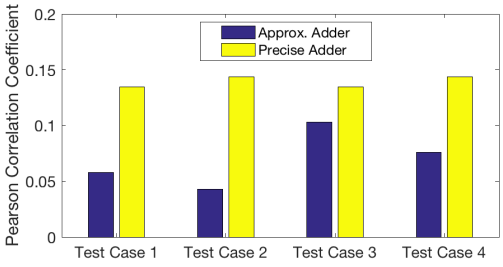

Figure 13: Impact of approximation in adder circuit on Pearson correlation coefficient (PCC) between output and power.

The power consumption and critical-path delay due to the carry-bit calculation is typically prominent in an adder. To minimize delay and also achieve energy efficiency, researchers redesign adder architecture or alter the truth table for addition operations to reduce the logic complexity and shorten the long path for carry propagation. The work [17] proposes to replace the original carry-in bit with an estimated one, which is based on the few input bits close to the current bit position. Depending on the number of bits used for carry estimation, the probability of getting the correct carry-in bit is in the range of 75% to 99.98% [17]. In the work [8], the truth table is modified in a way to achieve a better logic optimization. This type of approximation is feasible for floating-point additions in image processing applications, where the fractional part carried by the LSBs often time does not need precise calculation.

Adversaries could leverage the natural inaccuracy of the approximate adder to implement several attacks. If an attacker manipulates the adder inputs to persistently make the adder yield wrong outputs, the accumulated errors could trigger the system to utilize the fault-tolerance mechanism more often than usual. The use of the approximate adder may make verification process more challenging than the case of precise adders. If the verification is not exhaustive, the adversary could tamper the hardware implementation of approximate adders. The malicious modification on the approximate adder may not be detected by the side-channel analysis method. As a case study, we implemented two approximate adders and one precise adder, 8 bits per input, with an IBM 180nm CMOS technology, and compared the power correlation between the measured power and output-based power estimation. Hamming distance power model was used to estimate the power. As shown in Figure 13, the power correlation coefficient for the approximate adder is less than that for the precise adder. The test cases 1 and 2 were applied to an approximate adder with 25% output errors and the rest of the test cases were for the adder with 50% errors.

5 CONCLUSION

Approximate computing systems are emerging as an effective way to achieve energy efficiency without significantly sacrificing the accuracy. Despite of the promising benefits, the approximate mechanisms adopted in storage, arithmetic circuit, system architecture, and software stacks make approximate storage vulnerable to new security attacks. This work is one of the early efforts that comprehensively foresee the potential security threats on approximate computing systems.

ACKNOWLEDGMENTS

This work is partially supported by National Science Foundation CAREER Award No. 1652474.

REFERENCES

[1] I. Bhati, Z. Chishti, S. Lu, and B. Jacob. 2015. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *Proc. 2015 ISCA*. 235–246.
[2] I. J. Chang, D. Mohapatra, and K. Roy. 2011. A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications. *IEEE Trans. on Circuits and Syst. for Video Technology* 21, 2 (Feb 2011), 101–112.
[3] M. Cho, J. Schlessman, W. Wolf, and S. Mukhopadhyay. 2011. Reconfigurable SRAM Architecture With Spatial Voltage Scaling for Low Power Mobile Multimedia Applications. *TVLSI* 19, 1 (Jan 2011), 161–165.
[4] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. 2013. Neural Acceleration for General-Purpose Approximate Programs. *IEEE Micro* 33, 3 (May 2013), 16–27.
[5] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto. 2015. Better-than-voltage scaling energy reduction in approximate SRAMs via bit dropping and bit reuse. In *Proc. 2015 PATMOS*. 132–139.
[6] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto. 2015. Better-than-voltage scaling energy reduction in approximate SRAMs via bit dropping and bit reuse. In *Proc. 2015 PATMOS*. 132–139.
[7] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. 2011. IMPACT: IMPrecise adders for low-power approximate computing. In *Proc. 2011 ISLPED*. 409–414.
[8] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. 2013. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Trans. on Computer-Aided Design of Integr. Circuits and Syst.* 32, 1 (Jan 2013), 124–137.
[9] L. Ceze H. Esmaeilzadeh, A. Sampson and D. Burger. 2012. Architecture Support for Disciplined Approximate Programming. In *Proc. 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Syst.* 301–312.
[10] H. Hoffmann, Sasa Misailovic, S. Sidiroglou, A. Agarwal, and Martin C. Rinard. 2009. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures.
[11] M. Jung, D. M. Mathew, C. Weis, and N. Wehn. 2016. Invited: Approximate computing with partially unreliable dynamic random access memory âĂŤ Approximate DRAM. In *Proc. 2016 DAC*. 1–4.
[12] S. Keshavarz and D. Holcomb. 2017. Privacy leakages in approximate adders. In *2017 ISCAS*. 1–4.
[13] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke. 2016. Quality Control for Approximate Accelerators by Error Prediction. *IEEE Design Test* 33, 1 (Feb 2016), 43–50.
[14] J. Kwon, I. J. Chang, I. Lee, H. Park, and J. Park. 2012. Heterogeneous SRAM Cell Sizing for Low-Power H.264 Applications. *TCAS-I* 59, 10 (Oct 2012), 2275–2284.
[15] J. Mengte, A. Raghunathan, S. Chakradhar, and S. Byna. 2010. Exploiting the forgiving nature of applications for scalable parallel execution. In *Proc. 2010 IPDPS*. 1–12.
[16] D. T. Nguyen, H. Kim, H. Lee, and I. Chang. 2018. An Approximate Memory Architecture for a Reduction of Refresh Power Consumption in Deep Learning Applications. In *Proc. 2018 ISCAS*. 1–5.
[17] N.Zhu, W. L. Goh, and K. S. Yeo. 2009. An enhanced low-power high-speed Adder For Error-Tolerant application. In *Proc. 2009 Intl. Symp. on Integr. Circuits*. 69–72.
[18] D. Palomino, M. Shafique, A. Susin, and J. Henkel. 2016. Thermal optimization using adaptive approximate computing for video coding. In *Proc. 2016 DATE*. 1207–1212.
[19] F. Qiao, N. Zhou, Y. Chen, and H. Yang. 2015. Approximate Computing in Chrominance Cache for Image/Video Processing. In *Proc. 2015 IEEE Intl. Conf. on Multimedia Big Data*. 180–183.
[20] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan. 2017. Quality Configurable Approximate DRAM. *IEEE Trans. on Computers* 66, 7 (July 2017), 1172–1187.
[21] F. Regazzoni, C. Alippi, and I. Polian. 2018. Security: The Dark Side of Approximate Computing?. In *Proc. 2018 ICCAD*. 1–6.
[22] M. Thomas Benjamin G. Zorn S. Liu, P. Karthik. 2011. Flikker: saving DRAM refresh-power through critical data partitioning.
[23] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. 2013. Approximate storage in solid-state memories. In *Proc. 2013 MICRO*. 25–36.
[24] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *Proc. 2013 MICRO*. 1–12.
[25] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Proc. 2014 ISLPED*. 27–32.
[26] S. Xu and B. C. Schafer. 2017. Approximate Reconfigurable Hardware Accelerator: Adapting the Micro-Architecture to Dynamic Workloads. In *Proc. 2017 ICCD*. 113–120.