Security Threats and Countermeasures for Approximate Arithmetic Computing

Pruthvy Yellu, Mezanur Rahman Monjur, Timothy Kammerer, Dongpeng Xu, and Qiaoyan Yu
University of New Hampshire
Durham 03824, USA
Email:qiaoyan.yu@unh.edu

ABSTRACT

Approximate computing (AC) emerges as a promising approach for energy-accuracy trade-off in compute-intensive applications. However, recent work reveals that AC techniques could lead to new security vulnerabilities, which are presented in a format of visionary view. There is a lack of in-depth research on concrete attack models and estimation of the significance of the attacks on approximate arithmetic computing systems. This work presents several practical attack examples and then proposes two attack models with quantitative analysis. Input integrity check and exclusive logic based attack detection methods are proposed to address the attacks on AC systems. The experimental results show that the attack detection failure rate of our method is below $2.2 * 10^{-3}$ and the area and power overhead is less than 6.8% and 1.5%, respectively.

Keywords

Approximate computing (AC), hardware security, Discrete Cosine Transform (DCT), Artificial Neural Network (ANN).

1 INTRODUCTION

Improving energy efficiency has become a critical demand of computation-intensive applications. Unfortunately, today's computing systems are built to perform precise computations at the cost of high energy consumption. Since many applications (e.g. multimedia and signal processing) are inherently error tolerant, exact calculations are not necessary [3, 13]. Approximate computing (AC) leverages the error resilience characteristics in applications to trade accuracy for better energy efficiency [5].

Existing research efforts on AC are distributed at the software [2, 19], architecture [4, 14], storage [10, 12, 17] and circuit [1, 3] levels. At the software level, the majority of approximations are achieved by skipping computations in the algorithm. Approximations in hardware are performed by redesigning the computational logic at the circuit or gate level. In the work [3], the approximation of an adder is obtained by changing the K-map logic such that the sum and carry logic are implemented with minimal logic gates. In the work [10], the authors introduced a scheme that improves energy efficiency by reducing the refresh rate of DRAM blocks. The previous work focuses on ensuring better output quality despite the errors induced by approximate techniques. There is limited literature available to analyze the new security threats and develop specific countermeasures for AC systems. It is imperative to have concrete attack models and numerical analysis for AC system developers to foresee the potential risks of using approximation techniques.

The rest of this work is organized as follows. In Section 2, we survey the security threats available in literature, and highlight our contributions. Three motivation examples are presented in Section 3. In Section 4, we introduce the attacks on interconnect and approximate functional IPs. Two countermeasures are proposed in Section 5. Experimental results are provided in Section 6. We conclude this work in Section 7.

2 BACKGROUND

2.1 Security Threats in AC

Various approximation techniques at different software and hardware levels are summarized in the surveys [9, 16]. In this work, we are particularly interested in the security threats originating from the use of hardware-level approximation techniques. The work [7] indicates that their approximation technique could make chip reverse engineering easier. The over-scaled approximation technique will lead to erroneous chip outputs, which vary from chip to chip due to process variation. Reverse engineering the critical path of the chips using approximate computing will reveal chip identification. In the visionary work [11], the authors analyze the possibility of different hardware security threats in AC. The authors also compare the feasibility of security threats like reverse engineering, cloning and counterfeiting, hardware Trojans and side-channel analysis for the systems with approximate modules and precise modules. The follow-up work [18] presents more security threats with detailed examples, such as authentication failure due to the relaxed refresh rate of DRAM, data corruption in phase change memory because of insufficient writes, and an increased error rate in SRAM due to a change in the supply voltage. The works mentioned above warn us to be aware of the potential security threats on AC systems, but they neither present clear attack models nor provide protection methods for the new security vulnerabilities.

2.2 Our Contributions

The main contributions of this work are as follows:

- We propose detailed attack models for the compromised interconnect and approximate functional modules.
- This is the first work that uses quantitative examples to analyze the impact of potential security attacks on AC systems on the system accuracy, area, power consumption, and critical-path delay.
- We propose two countermeasures against potential attacks on AC systems, one for interconnect and one for approximate arithmetic computing IPs.
- The proposed attack models and countermeasures are evaluated on 64-bit adders, image compression and decompression, and artificial neural networks. Our experimental results show that the proposed method is light weight and has a high attack detection rate.

978-1-7281-4123-7/20/$31.00 ©2020 IEEE

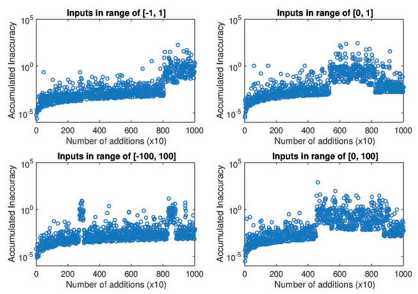

Figure 1: Accumulated inaccuracy of a 64-bit approximate adder.

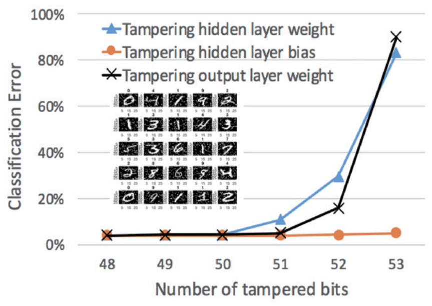

Figure 2: Impact of corrupted weights and bias on classification error of ANN.

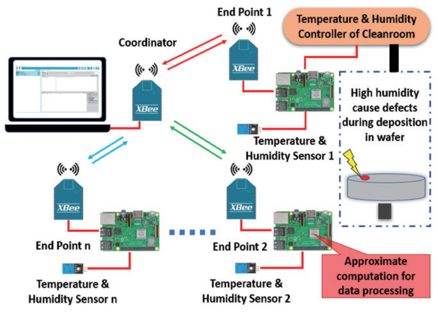
End Point 1
Temperature & Humidity Controller of Cleanroom
Coordinator
XBee
High humidity cause defects during deposition in wafer
Temperature & Humidity Sensor 1
End Point n
End Point 2
Temperature & Humidity Sensor n
Temperature & Humidity Sensor 2
Approximate computation for data processing

Figure 3: Approximation in a network of temperature and humidity sensors.

3 MOTIVATING EXAMPLES

Security vulnerabilities in AC systems are often overlooked because the applications that use AC (e.g. image processing) can tolerate errors. Existing literature has not considered the scenario that the degraded accuracy could be accumulated to trigger a malicious event. Moreover, the approximate techniques could be exploited to implement stealthy attacks since the effect of those attacks could be misinterpreted as the expected approximation effect. With the following examples, we advocate for researchers to be more cognizant about the security threats on AC systems.

3.1 Approximate Floating-Point Adder

We implemented a 64-bit approximate adder for floating-point numbers. One of the adder inputs is a random number and the other input is fed from the adder output (so that the imprecise output will be propagated to the next round of computation). We performed approximate addition 1000 times and plotted the difference between the outputs of the precise and approximate additions in Fig. 1. As shown, the trend of accumulated inaccuracy increases with the number of additions performed. The degraded output precision can go beyond two orders of magnitude. If the adder inputs are in the range of symmetric negative and positive numbers, the accumulated calculation inaccuracy is slightly less than both inputs in a non-negative range. Based on the cases of [-1, 1] and [-100, 100], we can observe that the input data limited in a smaller range will lead to more significant accumulated errors. This example indicates that the inaccuracy due to approximation could result in a non-negligible effect if accumulated.

3.2 Approximate Artificial Neural Network

Tremendous addition and multiplication operations are performed in artificial neural networks (ANN), where AC can be utilized to improve ANN energy efficiency [15]. Two types of attacks could occur in ANN with AC: sabotage the weights and bias parameters for the trained neurons or manipulate approximate arithmetic operations. In this example, we built an ANN that has one input layer, one hidden layer with 100 neural nodes, and one output layer. The ANN we built is used to learn the handwritten numbers provided by the MNIST data set [8]. We assume the mantissa bits for the weights and bias bits in the hidden or output layer are overwritten by an attacker. The number of manipulated bits varies from 48 to 53 (counting from the least significant bit). As shown in Fig. 2, when we tamper the 52^{th} bit (the most significant mantissa bit) of the hidden layer weight, hidden layer bias, and output layer weight, the image classification error rate increases by 7.59×, 1.09×, and 3.94×, respectively, over that of a healthy ANN. If the exponent bit of those parameters is sabotaged, the classification error rate could be over 21× higher than the baseline.

Table 1: Proposed attack models.

Attack model	Key features	
No. 1: tamper interconnect	Assumption	(1) AC functional IP is a blackbox; (2) Interconnect between AC and non-AC IPs are accessible;
	Attack method	(1) Swap MSB and LSB bits; (2) Force LSB to stuck-at-0/1;
No. 2: tamper AC function	Assumption	(1) AC functional IP is a whitebox; (2) Non-AC IPs are protected blackbox;
	Attack method	(1) Use hardware Trojan to trigger malicious approximate function; (2) Use external control to alter ambient environment.

3.3 Approximate Sensor Network

A sensor network was implemented for the purpose of monitoring the clean-room environment in the semiconductor manufacturing industry. As shown in Fig. 3, the network consists of microcontrollers (Raspberry Pi 3), XBee radio frequency modules, and sensors that measure the temperature and humidity of the surrounding environment. We assume that the sensed humidity value was approximated to improve the communication throughput between the end points and the coordinator. When we randomly changed 5 of the 52 mantissa bits, 55% of the measured humidity values were altered by the attack. Any malicious modification on the approximated digital representation of the sensor output could mislead the humidity controller in the clean-room and thus result in manufacturing more defective chips [6].

4 PROPOSED ATTACK MODELS

We propose two attack models for a system that employs AC. The key features for each attack model are listed in Table 1. In the following subsections, we explain the attack models and provide the numerical assessment for the case study designed for each model. In the explanation, we use a hybrid (i.e. precise-approximate) adder as a subject. The detailed attacks on the adder are depicted in Fig. 4.

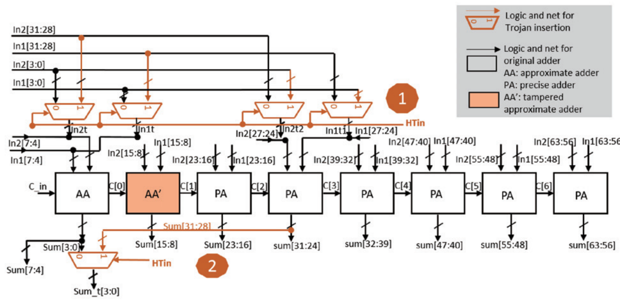

Figure 4: Hybrid adder with hardware Trojan payloads.

Table 2: Overhead induced by hardware Trojan that swaps the outputs of the hybrid 64-bit adder.

Adder design	Area (μm^2)	Dynamic Power(μW)	Leakage Power(μW)	Delay (ns)
Baseline	610.56(100%)	205.54(100%)	2.69(100%)	7.88
Information leak 2b	617.034(101.06%)	206.83(100.63%)	2.71(100.94%)	7.88
Information leak 4b	623.52(102.12%)	208.51(100.45%)	2.74(101.88%)	7.88
Information leak 8b	636.48(104.25%)	211.91(103.10%)	2.79(103.76%)	7.88

4.1 Attack Model 1: Tamper Interconnect

4.1.1 Model Description The attack objective is to exploit the existing AC units to expand the degree of output inaccuracy beyond the system tolerable error range, such that the attack will lead the system to misbehave. Attackers will access the precise and approximate function units as blackbox IPs, but they can swap a portion of inputs or outputs for non-AC IPs (for MSBs) and AC IPs (for LSBs), or shorten the interconnect. This type of attack is easy to implement and is stealthy since no new hardware component is needed to define the wrong functionality.

4.1.2 Example Figure 4 shows two examples of interconnect tampering in a 64-bit hybrid precise-approximate adder. The multiplexers are utilized to swap the inputs for an approximate adder (i.e. AA) and a precise adder (i.e. PA). The selection signal for the multiplexers is a Trojan payload signal (HTIn). The tampering case ① will change the computation accuracy for the critical and non-critical bits, thus yielding the non-tolerable inaccuracy. The tampering case ② will leak critical information from the precise data channel to the approximate one. We implemented the adder shown in Fig. 4 and synthesized it with a TSMC 65nm technology. Based on our synthesis results shown in Table 2, no timing overhead is observed because the tampering logic does not appear in the critical delay path, and the increase on the area and total power is less than 4.25%. In summary, the hardware Trojan aiming to enlarge the computational inaccuracy or leak the MSB in the hybrid computation module is feasible and stealthy.

We applied the hardware Trojan described in attack model 1 to an image compression and decompression application, discrete cosine transform (DCT)-Inverse DCT. The inserted Trojan will exchange a few exponent bits with the same number of mantissa bits of the pixels on an image row. As shown in Fig. 5, the proposed attack indeed causes a noticeable impact on the image. If the tampered interconnect is located before the DCT inputs, the impact on the image is minor. In contrast, if the interconnect tampering happens after DCT, the decompressed image will be significantly affected.

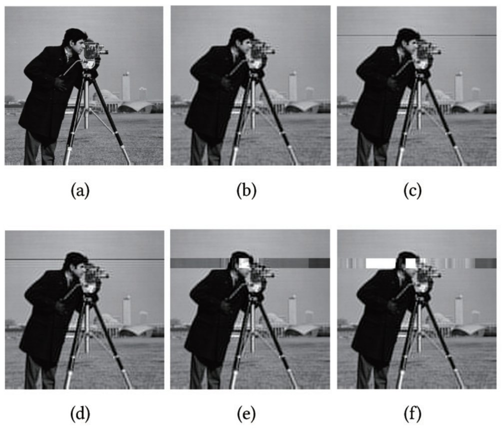

Figure 5: Images in DCT-IDCT. (a) Original image before normal DCT, (b) image after normal IDCT, (c) image sabotaged before DCT, (d) image sabotaged after DCT (two bits swapped), (e) image sabotaged after DCT (four bits swapped).

4.2 Attack Model 2: Tamper Approximate Computing Function

4.2.1 Model Description Approximation leaves attackers exploration space to either alter the original approximation function or add extra functions, which could be misinterpreted as a predefined approximation. Due to the limited testing/verification typically performed for AC systems, the tampered approximation function is likely to be undetected. As the output of AC IPs will be inexact, those IPs are often not protected and thus we assume they are whiteboxes. The malicious changes made on AC IPs could be triggered by functional hardware Trojans or environmental factor (e.g. temperate or supply voltage fluctuation).

4.2.2 Example In a 1-bit full adder, the logic for the correct carry is expressed in Eq. 1. The approximation function proposed in [3] is shown in Eq. 2. We also created over-propagation (Eq. 3) and under-propagation (Eq. 4) cases for the malicious modification on the carry logic.

$$Cout_{orig} = A\&B + B\&Cin + A\&Cin; \tag{1}$$

$$Cout_{approx} = B + (A\&Cin); \tag{2}$$

$$Cout_{op} = B + A + Cin; \tag{3}$$

$$Cout_{up} = B\&A\&Cin; \tag{4}$$

If not tampered, the error yielded by the approximation logic is acceptable. As shown in Table 3, the probability of logic error due to the approximation is 12.50%, if thorough input patterns are applied in the testing stage. Even if the inputs are biased, the error rate only increases by about 3%. In contrast, malicious logic modification will cause a higher error rate, 37.50%. Once the inputs are biased, the carry bit will lean towards a higher probability of logic 1 or logic 0.

Next, we examine the impact of over-propagation and under-propagation of the carry on the output accuracy in the adder. As

Table 3: Error rate of the carry logic for 1-bit full adders with precise and approximated addition operations or tampered approximate functions.

Carry logic	Thorough Test	Biased Inputs prob0=0.7 prob1=0.3	Biased Inputs prob0=0.3 prob1=0.7	Uniform Inputs prob0=0.5 prob0=0.5
$Cout_{orig}$	0%	0%	0%	0%
$Cout_{approx}$	12.50%	15.48%	6.29%	12.82%
$Cout_{op}$	37.50%	44.12%	18.80%	37.59%
$Cout_{up}$	37.50%	19.15%	44.01%	37.81%

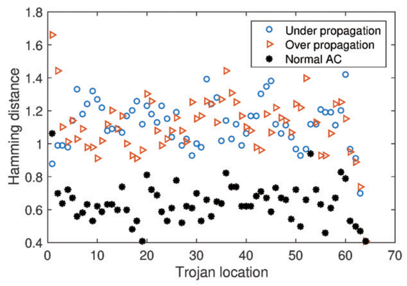

Figure 6: Hamming distance (with precise) due to Trojan tampering the logic for approximate carry bit.

shown in Fig. 6, in general, the malicious modification on the approximate logic increases the difference (i.e. Hamming distance) between the outputs of the precise adder and logic-tampered adder.

5 PROPOSED COUNTERMEASURES

In Section 4, we identified two security attacks on AC systems. Now, we propose countermeasures to protect approximate arithmetic computing systems from those attacks. Figure 7 depicts the overview of our proposed methods. To avoid the increase in the critical path delay, our countermeasure always runs in parallel with the arithmetic unit. The *Input Integrity Check* (IIC) module examines whether the interconnect between the inputs from other IPs to the precise and approximate arithmetic modules are compromised. The *Output Integrity Check* module serves the same purpose as the IIC module does. More details on *IIC* are provided in Section 5.1. In the exclusive logic based attack detection module (*ELA Detection*), the outputs of the approximate function is selectively examined to generate an alert signal to indicate hardware attacks on AC IPs. An example of alert logic is presented in Section 5.2. Note that the countermeasure against the attack on the global operation environment is beyond the scope for this work.

5.1 Securing Interconnect between Precise and Approximate Computing IPs

We propose an IIC algorithm to detect the interconnect tampering attack defined in attack model 1. The inputs for the arithmetic computation IPs are divided into subgroups, followed by key-controlled interleaving. The user key will determine how to interleave the inputs. Even parity check code is adopted to generate a check bit for each group of interleaved inputs. Next, the check bits will be fed to the arithmetic computation IPs. In the IIC unit, the check bits will be calculated again after the inputs from each

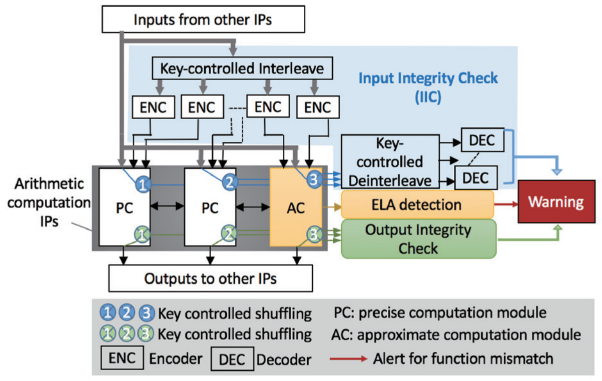
Inputs from other IPs
Key-controlled Interleave
Input Integrity Check (IIC)
ENC
ENC
ENC
ENC
Key-controlled Deinterleave
DEC
DEC
Arithmetic computation IPs
PC
PC
AC
ELA detection
Warning
Output Integrity Check
Outputs to other IPs
Key controlled shuffling
Key controlled shuffling
ENC Encoder
DEC Decoder
PC: precise computation module
AC: approximate computation module
Alert for function mismatch

Figure 7: Overview diagram of proposed countermeasure against interconnect and approximate function tampering.

Algorithm 1: Proposed Input Integrity Check (IIC).

```
Data: Inputs for PC and AC IPs, user key (Ukey)
Result: Alert logic for interconnection tampering
1  Interleave inputs from other IPs with Ukey ;
2  Generate parity check bits, $\Theta_{in,j}$, for a group of inputs:
     $\Phi_j, j \in (1, N)$, where N is the number of IPs ;
3  Shuffle a group of inputs in each submodule:
     $\{\Phi_j(Ukey), \Theta_j\} \rightarrow \{\Phi'_j(Ukey), \Theta'_j\}$ ;
4  Deshuffle inputs $\{\Phi', \Theta'\} \rightarrow \Psi$ ;
5  while $j \leq N$ do
6      Recalcuate parity check bits $\Theta(\Psi_j)$;
7      if $\Theta_{p,j} != \Theta_{in,j}$ then
8          Discard current output and alert warning system;
9      else
10         No alert;
11     end
12     j++;
13 end
```

module are deshuffled. If the check bits transferred from the PC and AC IPs do not match the newly calculated ones, the interconnect tampering attack is detected and an alert signal turns on the warning system. We use even parity check code as a test case. In practical applications, the error detection code for ENC and DEC units in Fig. 7 can be other type of codes. The detailed IIC method is further presented in Algorithm 1.

5.2 Securing Approximate Function

To address the security threats defined in attack model 2, we propose an exclusive logic based attack (ELA) detection method. Recall that the AC IP is a whitebox in this attack model. Thus, our countermeasure will NOT be located in the AC IP. We scan the input patterns that will result in different outputs from the PC and AC IPs. Next, we choose one input pattern (if there are multiple input patterns leading to different outputs) and the expected precise output to generate the alert logic. If attackers modify the logic defined in the AC IP, the alert logic is set to notify the occurrence

Algorithm 2: Proposed Exclusive Logic based Attack (ELA) Detection.

```
   Data: Precise logic function $\Gamma$, approximate logic function $\gamma$
   Result: Find the optimal tampering alert logic
 1 Exhaustive search for the list of input patterns $\Xi$ that lead to
     $\Gamma(\xi_i)! = \gamma(\xi_i), \xi_i \in \Xi$ ;
 2 Use randomly modified logic $\delta(\,)$ to replace the approximate
     logic $\gamma(\,)$ ;
 3 Initialize detection rank $det_i (i \in [1, size(\Xi)]$ as 0;
 4 while $i \leq size(\Xi)$ do
 5     if $\delta(\xi_i)$ !=$\gamma(\xi_i)$ then
 6         Increase $det_i$ ;
 7     else
 8         $det_i$ remains same ;
 9     end
10     Form the alert logic using $\xi_i$ and expected output $out_p$;
11     Estimate hardware cost for the alert logic $HW_i$ ;
12     i++ ;
13 end
14 Choose the optimal $\xi_i$ for alert logic generation;
```

of the attack. To trade off the hardware cost and detection success rate, we search for the optimal combination of input pattern and the outputs. More details of our method are shown in Algorithm 2.

We resume the example analyzed in Section 4.2.2 and use Algorithm 2 to develop the alert logic for the malicious carry. After an exhaustive search, we find that the input pattern of '0 1 0' (for A, B, and Cin, respectively) will lead to different carry bits for the precise and approximate adders. Thus, we form the Boolean logic expressed in Eq. 5 to generate the alert signal.

$$Alert = A + \overline{B} + Cin + Cout; \tag{5}$$

The proposed algorithm is able to detect the over-propagation and under-propagation attacks mentioned in Section 4.2.2 with success rates of 66.67% and 100%, respectively. To improve the detection success rate, we can bring in *Sum* to the alert logic in Eq. 6.

$$Alert = A + \overline{B} + Cin + Cout + \overline{Sum}; \tag{6}$$

6 EXPERIMENTAL RESULTS

6.1 Assessment on Proposed IIC

We implemented four 64-bit ripple carry adders, precise addition with or without IIC and approximate addition with or without IIC, in a TSMC 65nm technology. We used Synopsys Design Compiler to report the area, power and critical-path delay shown in Table 4. The approximate adder consumes less area and power and runs faster than the precise adder due to simplified logic. The proposed IIC does not increase the delay on the critical path since the integrity detection runs in parallel with the normal addition logic. Compared to the baseline adders, the area and power consumption for the adders with IIC is almost doubled. This is because the IPs (i.e. 1-bit adders) under protection use only a few logic gates. When the logic complexity for PC and AC IPs increases, the overhead percentage is expected to drop.

Table 4: Comparison of hardware cost for precise and approximate 64-bit adder with and without proposed IIC.

Adder design	Area (μm^2)	Dynamic Power(μW)	Leakage Power(μW)	Delay (ns)
Precise Adder w/o IIC	622.08	224.62	2.76	8.52
Approx. Adder w/o IIC	610.56	205.61	2.69	7.88
Precise Adder w/ IIC	1354.68	346.24	6.28	8.52
Approx. Adder w/ IIC	1343.16	327.54	6.21	7.88

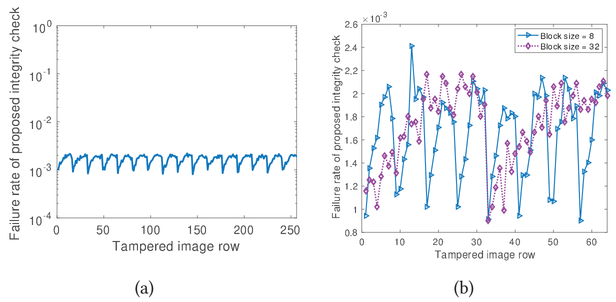

Figure 8: Attack detection failure rate of proposed IIC algorithm. (a) Failure rate for all possible attack locations, and (b) impact of DCT block size on the detection failure rate.

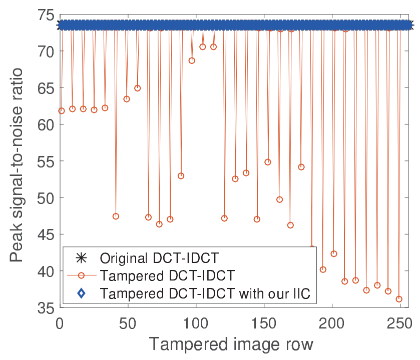

Figure 9: PSNR between original and decompressed images.

We used DCT-IDCT to process the *cameraman* picture shown in Fig. 5 as a case study. A triggered hardware Trojan (attack model 1) will swap the LSB of the exponent and the MSB on the mantissa of the floating-point number representing the pixel content. The image row under attack was swept from 1 to 256 (last row) to examine whether our IIC is data-dependent. As shown in Fig. 8(a), our attack detection failure rate is in the range of $2.2 * 10^{-3}$ and $8.087 * 10^{-4}$ and it is data independent. We further changed the block size in the DCT algorithm. Figure 8(b) shows that the range of our attack detection failure rate remains nearly consistent for the block size of 8 and 32. Next, we analyzed the peak signal-to-noise ratio (PSNR) between the original and decompressed images. As shown in Fig. 9, our approach successfully maintains the same PSNR as the DCT-IDCT without interconnect tampering. Our worst PSNR is 2.03× higher than the PSNR of the tampered DCT-IDCT without protection.

6.2 Assessment on Proposed ELA Detection

We continue to use the same adder in Section 6.1 to evaluate our ELA detection method. Hamming distance was adopted to

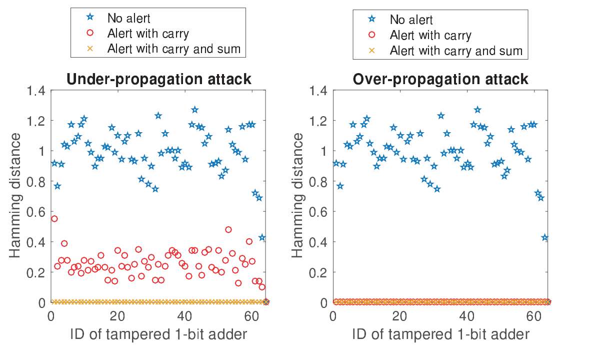

Figure 10: Hamming distance improved by proposed ELA detection method in a 64-bit adder.

Table 5: Hardware cost of the 64-bit approximate adder with and without proposed ELA detection method.

Alert logic and location	Area (μm^2)	Dynamic Power(μW)	Leakage Power(μW)	Delay (ns)
Baseline w/o any alert logic	610.56	205.61	2.69	7.88
Submodule w/ alert in Eq. 5	627.84	205.61	2.71	7.88
Submodule w/ alert in Eq. 6	640.08	206.66	2.80	7.88
Top module w/ alert in Eq. 5	641.16	207.52	2.77	7.93
Top module w/ alert in Eq. 6	652.32	208.69	2.80	7.95

compare the number of different output bits between the precise and approximate adders (w/o our protection method) experiencing under-propagation (up) and over-propagation (op) tampering attacks. Hamming distance for all test cases are shown in Fig. 10. For the under-propagation case, the proposed alert logic will reduce the Hamming distance by 74.6%. If we use both sum and carry (Eq. 6) to detect attacks, our method successfully reduces Hamming distance to zero (i.e. 100% attack detection rate).

Table 5 shows the hardware cost for different alert logic designs. The baseline is the 64-bit approximate adder implemented with a TSMC 65nm technology. The ELA detection unit was applied at the top module or each 1-bit full adder (i.e. submodule). The alert logic applied in the submodule does not incur delay overhead since the attack detection logic is not on the critical path. If applied in the submodules, our detection logic expressed in Eq. 5 consumes 2.8% and 0.55% more area and power, respectively, than the baseline. More complicated alert logic in Eq. 6 increases the area and power overhead to 4.8% and 0.95%, respectively. When we moved the attack detection logic to the top module of the 64-bit adder, the delay, area, and power overhead goes up by 0.88%, 6.8%, and 1.5%, respectively. In summary, our ELA method is a light-weight solution against approximate function tampering.

7 CONCLUSION

Approximate computing has emerged as a promising strategy to improve systems' energy efficiency at the cost of reduced precision or accuracy. The majority of approximate computing research focuses on the algorithms for approximation and the applications of approximate arithmetic units and storage, rather than the potential security threats brought by approximate computing. Following up the most recent visionary literature on security threats in approximate computing systems, this work numerically demonstrates the impact of two security threats on a floating-point adder and the image classification error of an ANN. We propose two attack models to characterize the attacks that affect interconnect and function modules in approximate arithmetic units. Furthermore, we suggest a general framework to strengthen the attack resilience of approximate computing systems. An input integrity check algorithm is proposed to secure the interconnect between precise and approximate computational IPs. The approximate arithmetic function is protected with an exclusive logic based detection method. Our experimental results show that the attack detection failure rate is below $2.2 * 10^{-3}$. The application of our input integrity check algorithm improves the peak signal-to-noise of image processed in a DCT-IDCT by over 2× that of the baseline. Our case study shows that the proposed exclusive logic alert method reduces the Hamming distance between precision and approximate addition output by up to 74.6%, while the delay, area and power overhead is less than 0.88%, 6.8% and 1.5%, respectively.

Acknowledgments

This research is partially supported by the NSF CAREER grant (No. CNS-1652474).

REFERENCES

[1] V. Camus, J. Schlachter, C. Enz, M. Gautschi, and F. K. Gurkaynak. 2016. Approximate 32-bit floating-point unit design with 53% power-area product reduction. In *Proc. ESSCIRC'16*. 465–468.

[2] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu. 2014. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *Proc. ASP-DAC'14*. 201–206.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. 2013. Low-Power Digital Signal Processing Using Approximate Adders. *TCAD'13* 32, 1 (Jan 2013), 124–137.

[4] L. Ceze D. Burger H. Esmaeilzadeh, A. Sampson. 2012. Architecture Support for Disciplined Approximate Programming. In *Proc. ASPLOS'12*. 301–312.

[5] J. Han. 2016. Introduction to approximate computing. In *Proc. VTS'16*. 1–1.

[6] Q. Huang, D. Zeng, S. Tian, and C. S. Xie. 2012. Synthesis of defect graphene and its application for room temperature humidity sensing. *Materials Letters'12*, 76–79.

[7] S. Keshavarz and D. Holcomb. 2017. Privacy leakages in approximate adders. In *Proc. ISCAS'17*. 1–4.

[8] Y. LeCun and C. Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/

[9] S. Mittal. 2016. A Survey of Techniques for Approximate Computing. *Proc. CSUR'16* 48 (2016), 62:1–62:33.

[10] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan. 2017. Quality Configurable Approximate DRAM. *TC'17* 7 (July 2017), 1172–1187.

[11] F. Regazzoni, C. Alippi, and I. Polian. 2018. Security: The Dark Side of Approximate Computing?. In *Proc. ICCAD'18*. 1–6.

[12] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. 2013. Approximate storage in solid-state memories. In *Proc. MICRO'13*. 25–36.

[13] D. Shin and S. K. Gupta. 2008. A Re-design Technique for Datapath Modules in Error Tolerant Applications. In *Proc. ATS'08*. 431–437.

[14] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *Proc. MICRO'13*. 1–12.

[15] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Proc. ISLPED'14*. 27–32.

[16] Q. Xu, T. Mytkowicz, and N. S. Kim. 2016. Approximate Computing: A Survey. *D&T'16* 33, 1 (Feb 2016), 8–22.

[17] L. Yang and B. Murmann. 2017. SRAM voltage scaling for energy-efficient convolutional neural networks. In *Proc. ISQED'17*. 7–12.

[18] P. Yellu, N. Boskov, M. Kinsy, and Q. Yu. 2019. Security Threats on Approximate Computing Systems. In *Proc. GLSVLSI'19*. 387–392.

[19] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. 2015. ApproxANN: An approximate computing framework for artificial neural network. In *Proc. DATE'15*. 701–706.