Approximate Computing for Low Power and Security in the Internet of Things

Mingze Gao, Qian Wang, Md Tanvir Arafin, Yongqiang Lyu* and Gang Qu
Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA.
*Research Institute of Information, Tsinghua University, Beijing, P.R. China

***Abstract*— The Internet of Things (IoT) has become ubiquitous in our daily life with billions of devices connected to the Internet infrastructure. This rapid increase of IoT devices brings many non-traditional challenges for the VLSI design and implementation of such "*Things*". Among them, low power is one of the most crucial concerns as most of the IoT devices are mobile and battery operated; security and privacy are another concern because IoT devices cannot afford the resource-demanding cryptographic protocols. In this article, we study a popular low power design methodology known as approximate computing and demonstrate how it can provide low power and security for IoT devices.**

***Keywords*: IoT, Approximate Computing, Low Power, Security**

The Internet of Things (IoT) is an open system consisting of many devices or embedded systems that are connected through the wired or wireless Internet infrastructure. The connectivity in IoT allows its devices or subsystems or Things to collect, process, and exchange data in order to accomplish specific applications more effectively. It was predicted that the total number of IoT devices will reach 30 to 75 billion by 2020. This explosive growth of Things and their diverse applications are inevitably bringing many non-traditional challenges to the circuit and system design community [1]. The first and perhaps one of the most crucial challenges is how to effectively manage power utilization for these tens of billions of Things as a significant portion of them are wireless and battery operated, where power consumption directly affects their lifetime [1, 4]. Fortunately, many of the low power VLSI design techniques developed in the past such as dynamic power and thermal management, dynamic voltage scaling, multiple threshold voltage design, new nano-scale materials like memristors, and energy harvesting can be used to improve the power efficiency of IoT devices. Meanwhile, a lot of new methods have been proposed to take advantage of specific features in IoT applications for power reduction. This article focuses on one of such methods known as approximate computing.

Approximate computing is a computation paradigm that offers resource savings at the cost of reduced computation accuracy. It has found applications in many fields such as multimedia and signal processing, wireless communication, and data mining/analysis. One common feature in these applications is that they can tolerate a certain level of errors in the computation. A similar concept has been proposed as early as in 2003 as the *probabilistic design* [7], where the authors proposed to drop the decoding of some frames to save energy as long as a human cannot tell the difference. The error tolerance in image compression motivated the adaptive pixel and coefficient truncation technique for energy minimization at system level [10, 11].

Generally low power approximate computing is achieved either by using voltage over-scaling (VOS) techniques to reduce the operating voltage of the system below the required threshold level or by redesigning system's functionality at different levels (algorithm, application, architecture or circuit) to save power at the cost of accuracy or correctness of the computation. In a recent survey [5], state-of-the-art approximate computing approaches are classified at the levels of computing, software, compilers, architecture, memory and circuit. Most of the existing research focuses on the arithmetic field, where the power/energy savings can be achieved by well-designed approximate computing function units such as approximate adders [9] and multipliers [8].

To see the potential of these approximate function units, we design circuits in Verilog for adders and multipliers with different bitwidth and synthesize them using Cadence RTL Compiler with FreePDK 45nm library. Table 1 shows that the area and power consumption of adders decrease about linearly with the number of bits, while the savings on multiplier are almost exponential. Most approximate function units take advantage of this by using adders and multipliers with fewer bits whenever possible.

FU / Bitwidth	Adder			Multiplier		
	cell	area	Power(nw)	cell	area	Power(nw)
8-bit	91	212.12	752614.84	377	1037.62	2829745.1
16-bit	230	322.33	2234652.24	1406	4208.68	10815807.06
32-bit	498	1116.93	3818821.94	4916	15126	34033690.3

Table 1. Cost comparison of approximate function units (FUs) with various bit width. Shaded area indicates the value normalized to that for the precise FUs with 32-bit.

However, existing approximate function units are fully-customized for a specific operation, either addition or multiplication, with a given accuracy. When they are used for other computations, for example, using approximate adders for multiplication, the error may accumulate and become out of control. More importantly, the large data range significantly limits the power saving of these approximate function units from reaching their potentials. Consider a 32-bit adder, when both operands have small values

with many leading zeros, an approximate adder with fewer bits could be used to add only the non-zero bits. On the other hand, if one operand or both are large, less significant bits could be neglected without causing a significant error to allow the use of an approximate adder. Existing approximate adders with fixed bit position for addition, cannot address both scenarios.

In this article, we extend approximate computing from the existing "static" approach to a more data-oriented "dynamic" fashion in order to save more system resource such as power. We achieve this by introducing a segmentation based approximate integer format (AIF) and developing the corresponding basic arithmetic operations that can be carried out by approximate function units for almost all applications [2].

Approximate Addition based on Approximate Integer Format

The proposed approximate integer format (AIF) is based on the segmentation of operands. An n-bit positive integer N is segmented into $\lceil n/k \rceil$ blocks with k bits in each block (except the first and leading block if N is not a multiple of k). A **valid block** is a block that contains at least one bit of value '1' or is after some valid blocks. The **precision control** (pc) value is the number of valid blocks that will be used in the approximate computation. For the i^{th} block of the given positive number, its **sentinel bits** st[i] is defined as:

$$\text{st}[i] = \begin{cases} 1, & block\ i\ is\ a\ valid\ block \\ 0, & block\ i\ is\ not\ a\ valid\ block \end{cases}$$

For example, consider two integers 1500 and 800, we can express them in binary as 1500_{10} = 0000 ***0101 1101 1100***$_2$ and 800_{10} = 0000 ***0011 0010 0000***$_2$, respectively. When we choose the block size k=4, both values will have three valid blocks (shown in bold) and the same sentinel bits 0111. If we set pc = 2 for an 8-bit approximate function unit, the two underlined valid blocks will be used during the approximate computation.

In AIF, data is rounded and stored based on its sentinel bits. We illustrate this with a 4-block operand A= $b_3b_2b_1b_0$, where each b_i is a block of fixed size. There will be five possible values of A's sentinel bits st_a: 0000, 0001, 0011, 0111, 1111. For the first four cases, the value of A will be stored in the following format: A = $st_ab_2b_1b_0$. This will not incur any data loss because the first block b_3 is a block of all 0's in these cases. In the last case when st_a =1111, the data A will be stored as A = $st_ab_3b_2b_1$ after we drop the last block b_0. This will introduce error when the block is not 0. However, since the first and the most significant block b_3 is valid, this error will be very small.

Now with all data represented in AIF, we can perform approximate computing in three phases: use sentinel bits to select the blocks for computation; use accurate computing units of smaller bit width to compute with the selected blocks of data; convert the result to the AIF format (for future computation) or report the result. The following algorithm shows how to add two operands A and B in AIF and an illustrative example can be found in Fig. 1.

1. *Compute the sentinel bits of the result S:* $st_s = st_A \mid st_B$, *where* $\mid$ *is the bit-wise OR;*
2. *Suppose the leading bit '1' in* st_s *is at* $st[i]$ *and pc valid blocks will be used, take the bits from the* i^{th} *to the* $(i\text{-}pc+1)^{th}$ *blocks of A and B and name them as A' and B', respectively;*
3. *Compute S' = A' + B' and the possible carry out Cout;*
4. *Update* st_s *by* $st_s[i+1] = Cout$*;*
5. *Reconstruct S in AIF based on* st_s *and S'. Padding 0's if necessary.*

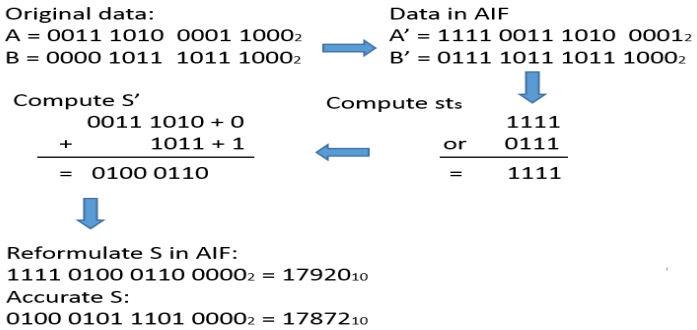

Fig. 1 An illustrative example of approximate addition in AIF. k=4, pc=2, and an 8-bit adder are used to compute S'.

Unlike existing approximate arithmetic units designed for specific operation, AIF based approach is suitable for all arithmetic operations: addition, subtraction, multiplication, and division, without redesigning the arithmetic units. More importantly, AIF can be conveniently extended to fixed point data and incorporated into a high level programming language, making it applicable to a large range of real-life applications (see Fig. 2 for five popular applications and their power saving) and letting designer control the accuracy-power tradeoff at a high level. More detailed technical discussion can be found in [2].

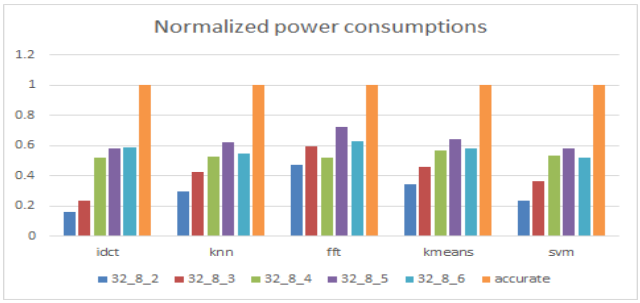

Fig. 2. Power consumption comparison for different configurations for approximate computing. 32_8_2 means 32-bit operands, 8 blocks (with k=4), and pc=2.

Besides the effectiveness in power saving, AIF based approach guarantees the computation quality. Table 2 lists the error of the first 40 terms in the Fibonacci sequence computed by approximation addition with different parameters (data is partitioned into 8 blocks and pc value varies between 2 and 4). From the table, we see that the errors are small, relatively unbiased, and decreases as pc value increases. When pc=4, we correctly generate the first 24 terms with no errors. This is better than any of the known approximated methods.

#	pc=2	pc=3	pc=4	#	pc=2	pc=3	pc=4	#	pc=2	pc=3	pc=4
1~13	0	0	0	22	-0.02732	3.49E-05	0	31	-0.02291	-0.00035	2.30E-06
14	-0.00984	0	0	23	-0.02692	0	0	32	-0.02267	-0.00059	-8.51E-06
15	-0.01317	0	0	24	-0.02707	1.33E-05	0	33	-0.02276	-0.0005	-4.38E-06
16	-0.01691	0	0	25	-0.02912	0.000404	8.24E-06	34	-0.02273	-0.00053	-5.96E-06
17	-0.01858	0	0	26	-0.03225	0.000336	1.02E-05	35	-0.02274	-0.00052	-5.36E-06
18	-0.01985	0	0	27	-0.0375	0.000362	9.44E-06	36	-0.02274	-0.00052	-5.59E-06
19	-0.02173	-0.00044	0	28	-0.03947	0.000352	9.72E-06	37	-0.0328	-0.0001	1.05E-06
20	-0.02905	0.000183	0	29	-0.04118	0.000356	9.61E-06	38	-0.03621	-0.00046	-5.53E-06
21	-0.02625	-5.65E-05	0	30	-0.04205	0.000354	9.66E-06	39	-0.04003	-0.00064	-3.02E-06
								40	-0.04174	-0.00077	-3.98E-06

Table 2. Fibonacci sequence error of first 40 terms using approximate addition under different configuration

Other Low Power VLSI Techniques in IoT

Besides approximate computing, there are many effective techniques to meet the low power demand in IoT systems. We briefly mention a couple of them.

Circuit Level Low Power Techniques: The power dissipation model in circuit level mainly contains three parts: dynamic power, static/leakage power, and short-circuit power. In today's VLSI technology, dynamic and leakage power are the dominant sources. Researchers have proposed various methods at circuit level for power and energy minimization. These methods include clock gating, transistor sizing, glitch and path balancing, technology mapping, temperature/thermal-aware, and dual threshold voltage and input vector control, don't care condition optimization, and more.

Dynamic Voltage and Frequency Scheduling: DVFS varies the supply voltage and clock frequency based on the computation load and deadline requirements to provide the required performance while minimizing the total amount of energy consumption. State-of-the-art DVFS designs have been proposed for the functionalities which are conventionally used in IoT devices. This can be naturally integrated with approximate function units.

Probabilistic Design: Probabilistic design methodology [7] aims at not over-designing systems for their worst case scenario. It utilizes the fact that much real time IoT applications do not require high performance and can tolerate a certain degree of faults. It uses prior or posterior execution information and takes advantages of the unique features of IoT devices' functionalities in order to relax the rigid hardware or software overdesigned implementation.

Security Challenges for IoT

As the second example, we show how approximate computing can help to address the security challenges in IoT design. IoT devices will collect, process and exchange massive data that could be confidential or privacy-sensitive. Therefore, the design and manufacture processes of IoT devices need to consider security and privacy in order to avoid potential malicious attacks and design flaws, such as the malware and Trojan embedding in software and hardware, unsophisticated EDA design tools [1], and untrusted supply chains. Although there exist many well-designed security protocols based on modern cryptography, the low power design requirement of IoT devices makes them unsuitable because they are computationally expensive and power hungry. As a result,

many IoT devices such as implantable medical devices do not have any data-protection. On the other hand, major security threats to IoT device itself exist, as we will survey later in this section.

There are active research on lightweight cryptography which aim to deliver affordable but weak security (for example, with short cryptographic keys) to IoT devices [3,6] as well as on hardware security primitives such as silicon physical unclonable functions (PUFs) and hardware-based random number generator and authentication. These approaches are much more energy efficient than applying the classic cryptographic solutions. However, they add security as a non-functional feature into the system and will need hardware or software support. In the next section, we will show how approximate computing can be used for information hiding to address some of these challenges.

Hardware is the fundamental element in IoT, but it is now becoming a new attacking surface through various physical attacks such as the hardware Trojan injection, side-channel analysis attacks, reverse engineering (RE) and intellectual property (IP) infringements. These problems have been addressed in the hardware design community for more than two decades, but they are unique and more dangerous for IoT devices, which are not fabricated with the latest semiconductor technology and make the above attacks much easier. Next, we briefly describe these attacks and introduce available countermeasures.

Hardware Trojan is a piece of a circuit that is implanted to the design or modified from the original design for malicious purposes. It can be as simple as several logic gates, but it can cause severe damage such as altering or disabling certain function units, leaking sensitive information or shortening the lifetime of IoT devices.

Side-channel attack is a classic non-invasive and passive attack by monitoring, measuring and analyzing the system's physical characteristics leaking from side-channels when the system is running. These characteristics include timing, current, voltage, electromagnetic radiation, power consumption, optical or acoustic information, *etc*. Side-channel attacks are effective and easy to implement but hard to detect and prevent. Moreover, they target vulnerabilities in the hardware or software implementation instead of the algorithms or protocols. Therefore, theoretically proved secure algorithms or protocols may become vulnerable if they are not implemented properly.

Reverse engineering (RE) is the invasive process of extracting IP from an IoT device and reproducing it based on the achieved information with little or no investment in research and development. These low-cost illegitimate products can be sold at a much lower price, giving them an unfair competitive edge against the authentic products. Moreover, when the high-level functionality of the IoT device is extracted, the attacker can redesign the device to avoid the infringement of copyright, or insert hardware Trojan into the system for malicious purposes. RE attacks can be very effective against IoT devices because they are simple and not using the latest semiconductor technology (to raise the cost of RE tools).

IP watermarking embeds the signature of the designer to claim the authorship and IP rights. It is used to detect and catch IP piracy. The carefully designed watermark can provide high confidence of IP's authorship, incur low design overhead, and are resilient against various attempts of watermark removal and modification. However, since it is designed to prove IP's authorship, the watermark remains the same for all copies of a given IP and one cannot trace the source of illegally distributed IPs. Digital **fingerprinting** addresses this concern by embedding IP user's information together with the IP author's signature. Thus, fingerprinted IPs can be viewed as multiple distinct watermarked IPs. Its goal is to identify each copy of the IP and protect honest IP users.

While digital watermarking and fingerprinting techniques can deter IP piracy, they do not prevent IP misuse and reverse engineering. **Circuit obfuscation** takes a step towards this direction by making RE harder. It seeks to modify the design and implementation of a circuit in order to make it difficult to interpret the layout and hence increase the cost and complexity of RE attacks. Currently, there are two types of obfuscation techniques: logic encryption and circuit camouflage. Logic encryption is based on the insertion of additional key gates that have the secret key values as part of their input signals. The key gates and thus the entire circuit will malfunction on incorrect key values. Circuit camouflage is based on replacing original logic gates with configurable logic cells. These configurable cells can be configured to perform as different logic gates, but the difference between configurations is too little to be observed by existing reverse engineering tools.

Physical Unclonable Function (PUF) is a small piece of circuitry embedded in the design that extracts silicon chip's fabrication variation and uses such intrinsic physical feature for security applications. PUF has been successfully used for secret key storage, random number generation, chip authentication, intellectual property protection, and anti-counterfeiting. It can be a promising hardware security primitive for IoT devices. However, its usability is limited due to its reliability concerns under different operating environments such as supply voltage, temperature, and humidity as well as circuit aging. Therefore, plenty of work has been proposed to enhance its robustness and stability.

Approximate Arithmetic Computing based Information Hiding

To minimize the power cost while still providing a practical solution for the IoT security problems, we introduce a novel approximate computing based approach to embedding information for authentication and other security related applications. The idea is inspired by data segmentation where the operands are divided into most significant bits (MSB) and least significant bits

(LSB). In approximate computing, the MSB part is used for precise operation and should be preserved, but the LSB part is either ignored or replaced by simple operations such as logical OR. In our approach, we hide information into LSB such that it does not affect approximate computation but can be recovered for security purpose. The implementation of our approach requires slight modification to the arithmetic unit, *e.g.,* adder or multiplier, without building any extra hardware primitives and corresponding systems.

A. Floating Point Format with Security Embedding

In IEEE754 single precision standard (double precision is very expensive and seldom used in IoT devices), the 32-bit data consists of 1 sign bit, 8 exponent bits, and 23 mantissa bits, as shown in the figure below. The value of an IEEE 754 number can be computed by $sign \times mantissa \times 2^{exponent}$, where the sign can be 1 and -1 if the leading bit is 0 and 1, respectively; the mantissa is a real value between 1.0 and 2.0 with fractional part represented in binary; the exponent equals to the 8 bits in the middle subtracting 127. For example, 0,10000000,10010010000111111010000 is 1x1.570795…x2, which is roughly 3.14159 in decimal.

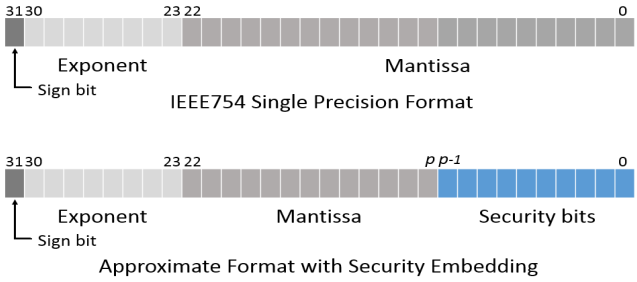

Fig. 3. Approximate single precision floating point format

.

Obviously, the LSB in mantissa will have little impact on the value. Therefore, we propose to use the last p bits, which we call security bits as shown in Fig. 3, to embed information without changing the rest 32-p bits. In the above example, if we set the last 10 bits to 0's, the value will become to 3.1413574, only a 0.0074% decrease from the original value. The error introduced by any changes to the last p bits will be less than 2^{p-24} relatively to the original value.

B. Approximate Computing with Information Embedding

Given two real numbers A and B, we rewrite them in the approximate data format: $A = A' \oplus K_A$ and $B = B' \oplus K_B$, where A' and B' are identical to A and B except that the last p mantissa bits are replaced by 0's; K_A and K_B are the last p bits of A and B; $\oplus$ is the bitwise XOR for the last p bits. For any binary arithmetic operation A⊗B, we propose the following method to perform approximate computing and information embedding simultaneously:

1. *rewrite A and B in the approximate data format*
2. compute *A'⊗B' and rewrite it as O' ⊕K_O,*
3. *generate a p-bit secret K_S to be embedded*
4. *return O' ⊕K_S as the result of A⊗B*

For example, with A=3.14159 and B=12.31, the previous result of A x B will be 38.6729729. Following the above scheme with *p=10*, we have

K_A
A = 0,10000000,10010010000111**1110100000**
A' = 0,10000000,10010010000111*0000000000*
K_B
B = 0,10000010,10001001111101**0111000011**
B' = 0,10000010,10001001111101*0000000000*
$K_{O'}$
A' x B' = 0,10000100,00110101010110**0011001111**
O' = 0,10000100,00110101010110*0000000000*
$K_S = K_A \oplus K_B \oplus K_{O'} \oplus Key = 1110001001$
Output: 0,10000100,00110101010111110001001

Fig. 4. A motivational example of information hiding approximate computing

where we generate the secret K_S using the simple bitwise XOR of K_A,K_B,$K_{O'}$ and a random Key= 01010101. The output value is 38.67124, 0.00448% less than the accurate result 38.6729729.

C. *Information Hiding for Security Applications*

The secret K_S can be as simple as a constant or some function of K_A, K_B, $K_{O'}$ and Key. In general, we can write this as $K_S = F(K_A, K_B, K_{O'}, \text{Key})$. We now give several examples where K_S can be generated and used for different purposes.

IP Watermarking: during the design and implementation of an IP, we can either use IP owner's digital signature as the Key or pick selective operand values to enable the above-proposed steps. By enabling, we mean a mechanism such as returning A'⊗B' directly (and skipping steps 3 and 4) unless the operands match the given values when the watermark is embedded or a watermark verification signal is activated. Then by checking the output error, we can reveal the watermark.

Device authentication/fingerprinting: similar to watermarking, each device's unique fingerprint can be embedded as the error value in the LSB. For the same verification inputs or operands, we can select different Key values and different F()'s so we will be able to distinguish all the individual devices.

Lightweight encryption: Since the result will be encrypted, we can use the entire 32-bit space for information hiding. For example, an encryption key up to 32 bits can be generated from function F() based on the values of operands and some given Key values. This encryption key will be used to encrypt the (approximated) computational result, for example by the efficient bitwise XOR operation. This is a symmetric encryption and the result can be decrypted easily once the key is available.

There are two potential threats to this information hiding protocol. First, an attacker can study the values of the operands and the results in order to reveal the Key value, the secret K_S, and the function that generates K_S. Second, authorized parties should be able to reveal the hidden information to verify the watermark and fingerprint or decrypt the encrypted results. An insider attacker or someone who has managed to gain this permission can further alter or forge the hidden information. These attackers can be easily prevented by enhancing the proposed method with additional cost. For example, the simple XOR operation can be replaced by a one-way hash function; information can be hidden on multiple operands and at different computation stage during a complex operation; intrinsic hardware features such PUF and secure memory can be utilized to secure Key and K_S.

The main advantage of our proposed approximate computing based information hiding method is its potential for low-cost implementation with a guarantee on the result. This is a result of the fact that we have utilized the energy efficiency of approximate computing and the tolerable error it introduces. The hidden information can be verified easily by disabling the approximate computing. As shown above, this security primitive can provide lightweight security for multiple applications such as authentication and IP protection.

Conclusion

In this article, we focus on how approximate computing can address the low power and security challenges of IoT device design. We summarize the existing low power techniques and hardware security threats and countermeasures. We introduce the Approximate Integer Format (AIF) and demonstrate its advantages to meet the low power demand of IoT devices. In addition, we propose an approximate floating point based security primitive that enables us to embed information during the process of approximate computing. The hidden information can be generated, embedded, and retrieved for several security applications.

Acknowledgement. This work was supported in part by AFOSR MURI under award number FA9550-14-1-0351. Yongqiang Lyu was supported in part by Tsinghua University Initiative Scientific Research Program and Chinese National Key Research and Development Program 2016YFB1000102.

REFERENCES

[1] Gang Qu and Lin Yuan. 2014. "Design things for the internet of things: an EDA perspective". In Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '14). IEEE Press, Piscataway, NJ, USA, 411-416.

[2] M. Gao, Q. Wang, A. Sharma K. Nagendra, G. Qu, "A Novel Data Format for Approximate Arithmetic Computing", 22nd Asia and South Pacific Design Automation Conference (ASP-DAC 2017)

[3] M. Katagi and S. Moriai. "Lightweight cryptography for the internet of things." Sony Corporation (2008): 7-10.

[4] H. Jayakumar, K. Lee, W.S. Lee, A. Raha, Y. Kim, and V. Raghunathan. "Powering the internet of things". In Proceedings of the 2014 international symposium on Low power electronics and design (ISLPED '14).

[5] Q. Xu, T. Mytkowicz, N. S. Kim, "Approximate Computing: A Survey." IEEE Design and Test, pp. 8-22, Jan./Feb. 2016.

[6] Md T. Arafin, M. Gao, G. Qu, "VOLtA: Voltage Over-scaling Based Lightweight Authentication for IoT Applications", 22nd Asia and South Pacific Design Automation Conference (ASP-DAC 2017)

[7] S. Hua, G. Qu, and S.S. Bhattacharyya, "Energy Reduction Techniques for Multimedia Applications with Tolerance to Deadline Misses", DAC, 2003.

[8] S. Hashemi, R. Bahar, and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications," in Proceedings of ICCAD, 2015, pp. 418–425.

[9] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in Proceedings of ICCAD, 2013, pp. 48–54.

[10] Se Hun Kim, Saibal Mukhopodhyay, and Marilyn Wolf, "System-level energy optimization for error-tolerant image compression," IEEE Embedded System Letters, 2(3), September 2010, pp. 81-84.

[11] Se Hun Kim, Saibal Mukhopadhyay, and Marilyn Wolf "Modeling and analysis of image dependence and its implications for energy savings in error tolerant image processing," IEEE Transactions on CAD/ICAS, 30(8), August 2011, pp. 1163-1172.

Mingze Gao is a Ph.D. student at the University of Maryland, College Park. His research interests include hardware security, low power design, and approximate computing. Email: mgao1@umd.edu

Qian Wang received the M.S. degree in Electrical Engineering from Tsinghua University, China in 2014. She is now a Ph.D. student at the Maryland Embedded Systems and Hardware Security Lab, University of Maryland, College Park. Her research interests include Embedded System and Hardware Security such as side channel attacks and PUF-related applications.
Email: qwang126@umd.edu

Md Tanvir Arafin is pursuing his graduate studies at University of Maryland, College Park. His research interests include semiconductor physics, integrated circuits and embedded security of microelectronic devices. Email: marafin@umd.edu

Yongqiang Lyu is an associate professor of Tsinghua University. His research interest focuses on the hardware-software fusion architecture in advanced computing systems, such as secure computing systems, high-performance networks, and smart human-computer interactions. Email: luyq@tsinghua.edu.cn

Dr. Gang Qu is a professor at the University of Maryland at College Park. He studies optimization and combinatorial problems and applies his theoretical discovery to applications in VLSI CAD, wireless sensor network, bioinformatics, and cybersecurity. His recent research interests are on hardware security and its impact to IoT and embedded systems. Email: gangqu@umd.edu