International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 18 (2015) pp 39570-39574

A Polynomial Multiplication Method Using the Modified Cooley-Tukey FFT Method for Next-Generation Cryptography

Sangook Moon

Mokwon University, Daejeon, Korea smoon@mokwon.ac.kr

Abstract
The emergence of quantum computers equipped with quantum algorithms threatens the existence of RSA and ECCencryption schemes, which are based on the hardness of factorization or discrete logarithm problems. In this paper, we present a polynomial multiplication algorithm for finite fields and VLSI architecture that features an O(nlogn) decrease in time complexity in the number of clock cycles, with trivially increased area. The proposed construction features enhancement in operation speed, modular structure with reconfigurability, and free choice of the finite field. Our contribution can be applied to next-generation quantum-reluctant cryptographic applications such as encryption systems based on ideal lattices defined on the ring $Z_p(x)/(x^n+1)$.

Keywords: polynomial multiplication, FFT, ideal lattice, ring

1. Introduction

With the arrival of quantum computers optimized for quantum algorithmsthat will become common in the near future, 3rd generation encryption methods such as RSA and elliptic curve cryptography (ECC), which guarantee their security based on hard factorization of prime numbers or discrete logarithm problems, are likely [1][2] to be replaced with post-quantum cryptographic algorithms such as NTRU, Learning With Errors (LWE), and Ring-LWE [3]-[5].

Since 2009, when Craig Gentry at Stanford University showed the proof of the hard problem ofa fully homomorphic encryption scheme [6][7], lattice cryptology has been the focusof various academic disciplines including computer science, mathematics, and VLSI implementation.

The hardness of lattice cryptologystarts with the shortest vector problem (SVP) and the closest vector problem (CVP)based on the subset-sum problem, and its application extends to the learning with errors (LWE) [8] problem. Recently, in order to address with efficiency the gigantic key size of $O(n^2)$, Lyebashevsky and Peikert proposed and proved the worst-case to average-case reduction of the Ring-LWE [9][10] in which the key size reduces to O(n) complexity.

The Ring-LWE cryptography uses polynomial multiplication with number theoretic transform (NTT) as itsbasicbuilding block. In the present work, wemodify the Cooley-Tukey FFT algorithm to efficientlyexecute the polynomial multiplication to speed up lattice cryptographic applications. This paper is an extended version of our work presented in [11].

2. Polynomial multiplication using number theoretic transform

Homomorphic encryption schemesare typicallycomprised of 3 procedures, *encrypt*, *decrypt*, and *recrypt*. The most frequently used operation is *polynomial multiplication*. If the bit-widths of the operands are sufficiently large, it is known that multiplication using the fast Fourier transform (FFT) is efficient.

Integer FFT essentially does not require complex floating point arithmetic computations. Given a primitive n-th root of unity w on finite field Z_p ($w^n = 1 \bmod p$), integer FFT (NTT: number theoretic transform) from a vector space $\{a_0, \ldots, a_{n-1}\}$ to $\{A_0, \ldots, A_{n-1}\}$ and the inverse transform are defined as below:

$$NTT_w(a): A_i = \sum_{j=0}^{n-1} a_j w^{ij} \bmod p, \quad i = 0, 1, \ldots, n-1$$

$$NTT_w^{-1}(A): a_i = n^{-1} \sum_{j=0}^{n-1} A_j w^{-ij} \bmod p, \quad i = 0, 1, \ldots, n-1$$

The necessary and sufficient condition for the NTT to exist is that for all prime factors q of p, $w^{n/q} - 1 \neq 0 \bmod p$ for arbitrary prime factor q. Choosing n as an exponential value of 2, NTT can be computed in O(nlogn) time complexity when $n \bmod (p-1) \equiv 1$.

Fig. 1 is an example of applying NTT in the multiplication of large operands, which is known to be mostly efficient when the operands are larger than $2^{2^{15}}$. To help understand, we used decimals as the base 2^W, calculating 1234*5678. Modulo number 337 is taken and w=85 is chosen as the 8th root of unity. As we can see, first zero-padding should be performed on each operand and NTT should be computed as per base. Then we can perform convolution (product-wise multiplication) of the two operands. As the outcome is obtained, we do NTT^{-1} and add the partial sums to yield the final product.

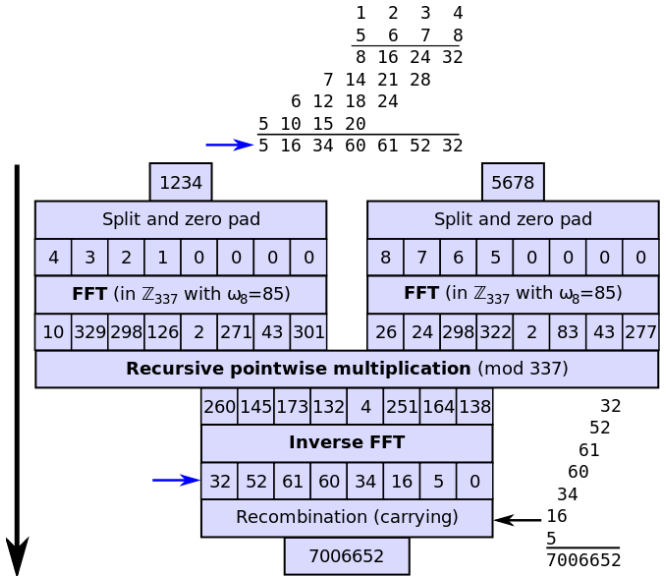

Figure 1. An example of polynomial multiplication using number theoretic transform

3. Finite field multiplier with modified Cooley-Tukey algorithm

We present arithmetic expressions in an extended field of Z_2. The extension order is represented by m, so that the field can be represented by Z_2^m. This field is isomorphic to $Z_2[x]/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$ is an irreducible polynomial of degree m with $p_i \in$ GF(2) and P(□) = 0. The calculation of the product of two arbitrary finite elements in GF(2^m), Z(□) = A(□)·B(□), where $A(\alpha) = \sum_{i=0}^{m-1} a_i \alpha^i$, and $B(\alpha) = \sum_{i=0}^{m-1} b_i \alpha^i$ expands as follows:

$$A(\alpha)\cdot B(\alpha) = A(\alpha)\sum_{i=0}^{m-1} b_i \alpha^i = \sum_{i=0}^{m-1} b_i (\alpha^i A(\alpha)) \bmod P(x)$$
$$= b_0 A(\alpha) + b_1 \alpha A(\alpha) + \ldots + b_{m-2}\alpha^{m-2} A(\alpha) + b_{m-1}\alpha^{m-1} A(\alpha) \bmod P(x) \quad (1)$$

Here we split the lower row of the expression (1) into an even part and an odd part,similarly to this process used in theCooley-Tukey FFT algorithm.

$$Z_{even}(\alpha) = b_0 A(\alpha) + b_2 \alpha^2 A(\alpha) + \ldots + b_{m-3}\alpha^{m-3} A(\alpha) + b_{m-1}\alpha^{m-1} A(\alpha) \bmod P(x)$$
$$Z_{odd}(\alpha) = b_1 \alpha A(\alpha) + b_3 \alpha^3 A(\alpha) + \ldots + b_{m-4}\alpha^{m-4} A(\alpha) + b_{m-2}\alpha^{m-2} A(\alpha) \bmod P(x)$$
$$= \alpha(b_1 A(\alpha) + b_3 \alpha^2 A(\alpha) + \ldots + b_{m-4}\alpha^{m-5} A(\alpha) + b_{m-2}\alpha^{m-3} A(\alpha)) \bmod P(x) \quad (2)$$

Now we perform reduction utilizing the following characteristic:

$$\alpha^m = p_0 + p_1\alpha \ldots + p_{m-1}\alpha^{m-1}$$
$$\alpha^{m+1} = p_0\alpha + p_1\alpha^2 \ldots + p_{m-2}\alpha^{m-1} + p_{m-1}\alpha^m \quad (3)$$

Here, p_{m-1} of the bottom row of expression (3) can be treated asa zero-coefficientfor the purpose ofsuccinctness without losing generality because primitive polynomials such as trinomials or pentanomials, which have p_i=1 in the highest and the lowest order of the polynomial, are given in practical applications. We can describe this as below:

$$\alpha^2 A(\alpha) = a_0\alpha^2 + a_1\alpha^3 + a_2\alpha^4 \ldots + a_{m-2}\alpha^m + a_{m-1}\alpha^{m+1}$$
$$= a_{m-2}p_0 + (a_{m-2}p_1 + a_{m-1}p_0)\alpha + (a_{m-2}p_2 + a_{m-1}p_0 + a_0)\alpha^2 + \ldots$$
$$+ (a_{m-2}p_{m-1} + a_{m-1}p_{m-2} + a_{m-3})\alpha^{m-1} \quad (4)$$

Next, reduction based on □ ^{2}should be performed. The top row of expression (2) indicates that there should be an extra circuit thatperforms a multiplicationof □to the A(□) (which circuit will henceforth be referred to as □-multiplier) so that we can iteratively compute the result of the even part of Z(□) in $\lceil m/2 \rceil$ clock cycles. In the 3rd row of equation (2), iteratively computing in the similar way in $\lceil (m-1)/2 \rceil$ clock cycles, it multiplies□ to the outcome itself, which adds one extra clock cycle, yielding the outcome of the odd part of Z(□). Thus, if we choose an odd-ordered m, we can get the final outcome of Z(□) by simply exclusive-OR-ing Z_{even}(□) and Z_{odd}(□) without wasting even one cycle. Fig. 2 shows the essential block diagram of the□ □-multiplier when implemented with n=193, and Fig. 3 shows the construction of the Z_{even}(□) of expression (2). In the same way, we can construct the Z_{odd}(□) of expression (2) by onlyexclusive-OR-ing two 2-to-1 muxes per bit plus $\alpha_t - 1$ extra muxes for selecting$\alpha -$ or □^{2}-multiplier, where α_t is the hamming weight of the primitive polynomial, which is either zero or one.

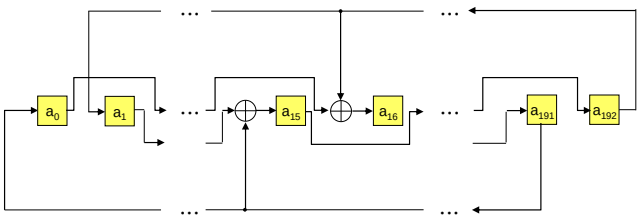

Figure 2. □-multiplier with n=193

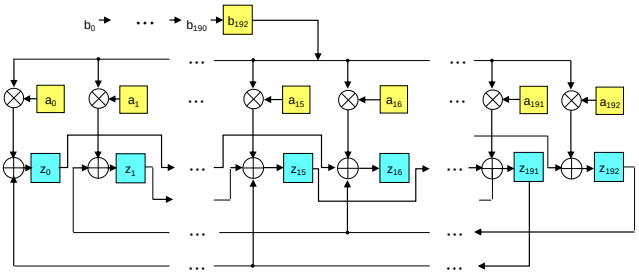

Figure 3. Z_{odd} circuit block diagram on GF(2^{193})

Thus, the number of clock cycles for one GF multiplication was reducedby a factor of two. We can successivelyconstructpolynomialmultiplication circuits that are multiple times as fast as the one using theserial method,employing up to an□ ^{t}-multiplier without spending as many area resources. These resource savingsare possible by sharing the register blocks of A(□).

4. Beyond 2X performance

As we mentioned above in Section 2, the polynomial product can be represented as expression (5) where two arbitrary elements of Z_2^m are $A(x) = \sum a_i x^i$, $B(x) = \sum b_i x^i$. Then the product $Z(x)$ can be obtained as below:

$$Z(x) = A(x)\sum b_i x^i = \sum_{i=0}^{m-1} b_i \left(x^i A(x)\right) \; mod\; P(x)$$
$$= [b_0 A(x) + \cdots + b_{m-1} x^{m-1} A(x)] mod\; P(x) \quad (5)$$

The order of the result expression reduces down to, at most, (m-1)th order using Mastrovito's x multiplier circuit. We propose another method for speed-up beyond 2X speed. First, we divide expression(5) by 3 blocks, which consequently requires an x^3-multiplier circuit as in Fig. 4.

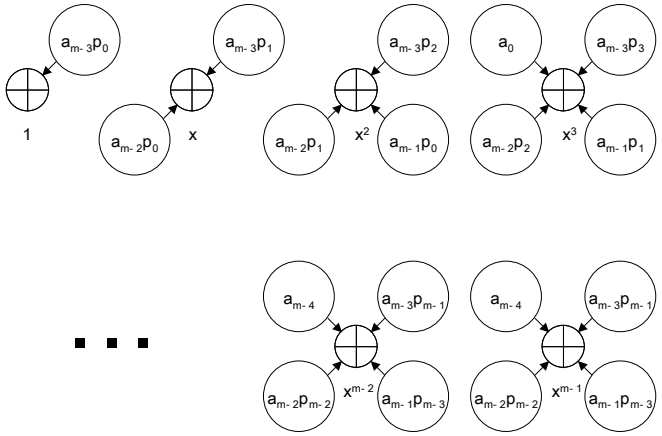

Figure 4. x^3-multiplier circuit

We next present the expansion structure for a 3X GF polynomial multiplier as in Fig. 5.

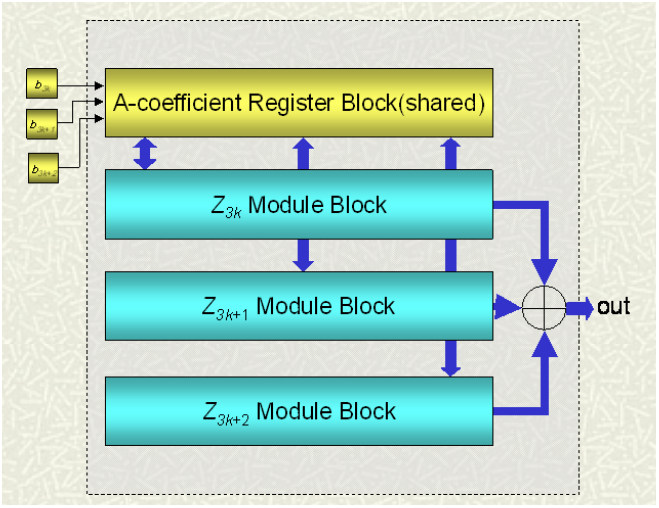

Figure 5. Polynomial multiplication implementation with 3X speed-up

Similarly, we divide expression (5) by t blocks and assign hardware resources respectively, aiming for t-times speed up, as broken down in expression (6):

$$Z_{tk}(x) = [b_0 A(x) + b_t x^t A(x) + \cdots] mod\; P(x)$$
$$Z_{tk+1}(x) = x[b_0 A(x) + b_t x^t A(x) + \cdots] mod\; P(x)$$
$$Z_{tk+2}(x) = x^2[b_0 A(x) + b_t x^t A(x) + \cdots] mod\; P(x)$$
$$\vdots$$
$$Z_{tk+(t-1)}(x) = x^{t-1}[b_0 A(x) + b_t x^t A(x) + \cdots] mod\; P(x) \quad (6)$$

Using this method, we require extra muxes and an x^t-multiplier circuit. We also present a 3X GF polynomial multiplier implementation block diagram in Fig. 5, and the expanded structure in Fig. 6.This construction features almost the same length of critical path delay as the Mastrovito's serial multiplier.

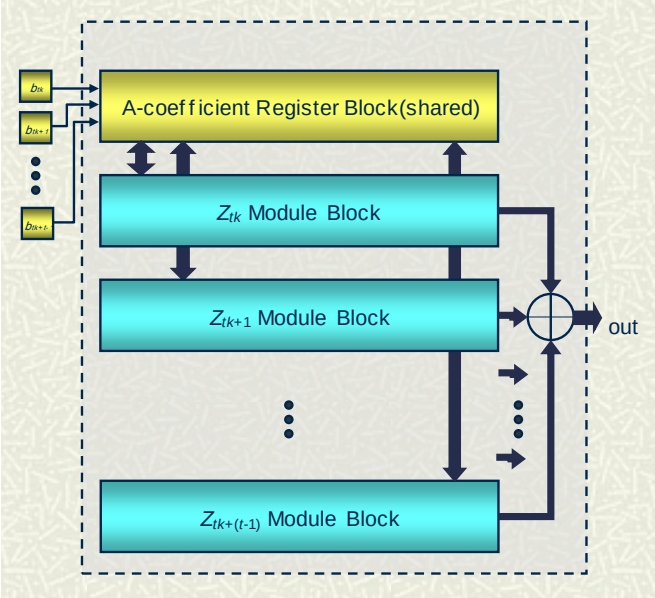

Figure 6. Polynomial multiplier with tX speed-up

5. Implementation with 3X speed-up

In order to implement the 3X serial polynomial multiplier, we modify expression (5) as below.

$$Z_{3k}(x) = [b_0 A(x) + b_3 x^3 A(x) + \cdots] mod\; P(x)$$
$$Z_{3k+1}(x) = x[b_1 A(x) + b_4 x^3 A(x) + \cdots] mod\; P(x)$$
$$Z_{3k+2}(x) = x^2[b_2 A(x) + b_5 x^3 A(x) + \cdots] mod\; P(x) \quad (7)$$

As we analyze the rows in expression (7), we can see that the order in the multiplication increases by a factor of 3, and we should handle the x term and x^2 term in the 2nd and the 3rd rows, respectively. In order to obtain the most essential part of $x^3A(x)$, we can express this as in equation (8):

$$x^3 A(x) = a_0 x^3 + a_1 x^4 + \cdots + a_{m-4} x^{m-1} + a_{m-3} x^m + a_{m-2} x^{m+1} + a_{m-1} x^{m+2} \quad (8)$$

We modify the 3 right-most terms as in expression (9).

$$x^m = p_0 + p_1 x + \cdots + p_{m-2} x^{m-2} + p_{m-1} x^{m-1}$$
$$x^{m+1} = p_0 x + p_1 x^2 + \cdots + p_{m-2} x^{m-1} + p_{m-1} x^m$$
$$x^{m+2} = p_0 x^2 + p_1 x^3 + \cdots + p_{m-2} x^m + p_{m-1} x^{m+1} \quad (9)$$

Here, we utilize the characteristics of the primitive polynomial given. NIST recommends mostly trinomials or pentanomials. These polynomials feature zero-value p_{n-1} and p_{n-2}, respectively. Therefore, using this feature, expression (7) can be simplified as (10):

$$x^{m+1} = p_0 x + p_1 x^2 + \cdots + p_{m-2} x^{m-1}$$
$$x^{m+2} = p_0 x^2 + p_1 x^3 + \cdots + p_{m-3} x^{m-1} \quad (10)$$

We put (9) and (10) to (8) and obtain the expression (11):

$$\begin{aligned} x^3 A(x) &= a_{m-3} p_0 \\ &+ (a_{m-3} p_1 + a_{m-2} p_0) x \\ &+ (a_{m-3} p_2 + a_{m-2} p_1 + a_{m-1} p_0) x^2 \\ &+ (a_0 + a_{m-3} p_3 + a_{m-2} p_2 + a_{m-1} p_1) x^3 \\ &+ (a_1 + a_{m-3} p_4 + a_{m-2} p_3 + a_{m-1} p_2) x^4 \\ &+ \ldots \\ &+ (a_{m-4} + a_{m-3} p_{m-1} + a_{m-2} p_{m-2} + a_{m-1} p_{m-3}) x^m \end{aligned} \quad (11)$$

Expression (11) accounts for Fig. 3 in detail.

6. Performance evaluation

We implemented a 3X GF polynomial multiplication circuit using the x^3 multiplier represented in Fig. 3.The multiplier cell in Fig. 3 is shown in detail in Fig. 7.

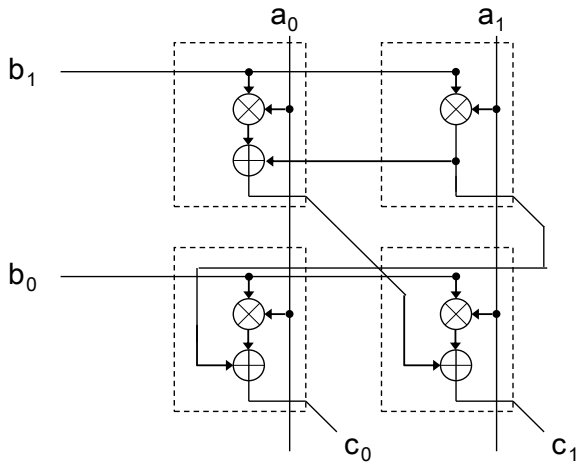

Figure 7. GF(2^2) multiplication cell structure

In order to validate the operation and reliability of the implemented 3X polynomial multiplier, we also implemented both the polynomial multiplication version and the serial multiplication version of test bench programs in C programming language and injected arbitrary 193-bit-width random numbers to verify the results.

We used Verilog-HDL to save the time for hardware construction and error correction in register transfer level and performed automatic synthesis and layout. We used a Samsung 0.35um technology standard cell library (std90) and the Synopsys Design Compiler in the Synosys Design Analyzer tool package. CubicDelay from CubicWare was selected as a tool standard delay format (SDF) file extraction.

Table 1 shows the final synthesis result of the 3X speed, 193-bit polynomial multiplication circuit. The critical delay, operating frequency, and number of gates indicate that this implantation is well-suited for applications such as lattice cryptography.

Table 1. Synthesis result of 3X 193-bit polynomial multiplier

	Proposed 193-bit 3X Polynomial Multiplier
Critical path delay	0.7 ns
Operating frequency	1.42 GHz
Unit gates	6,563
Technology	Samsung Semiconductor 0.35um st90

7. Conclusions

The emergence of quantum computers equipped with quantum algorithms is about to threaten the existence of RSA and ECC technologies, which are based on the hardness of factorization or discrete logarithm problems. In this paper, we present a polynomial multiplication algorithm for finite fields and VLSI architecture that features an O(nlogn) decrease in time complexity in the number of clock cycles, with trivially increased area. The proposed construction features enhancement in operation speed, modular structure with reconfigurability, and free choice of the finite field. Our contribution can be applied to next-generation quantum-reluctant cryptographic applications such as encryption systems based on ideal lattices defined on the ring $Z_p(x)/(x^n+1)$.

ACKNOWLEDGMENTS

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A1A2A16053925).

References

[1] C. Lu, D. E. Browne, T.Yang, and J. Pan, Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits.Physical Review Letters, **(2007)**,Vol. 99, No. 25.

[2] E.Lucero, R.Barends, Y. Chen, J.Kelly, M.Mariantoni, A.Megrant, P.O'Malley, D.Sank, A.Vainsencher, J.Wenner, T.White, Y.Yin, A. N. Cleland and J. M.Martinis, Computing prime factors with a Josephson phase qubit quantum processor.Nature Physics, **(2012)**,Vol. 8, pp. 719-723.

[3] J. Hoffstein, J. Pipher and J. H. Silverman, Lecture Notes in Computer Science.1423 **(2006).**

[4] O. Regev, Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, **(2005)** May 22, CO, United States.
[5] Z. Brakerski, V. Vaikuntanathan, Lecture Notes in Computer Sciences. 6841**(2011).**
[6] C.Gentry, A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, **(2009)**September;CA, United States.
[7] C.Gentry, Fully homomorphic encryption using ideal lattices. Symposium on the Theory of Computing, **(2009)**, pp. 169-178.
[8] V.Lyubashevsky, C.Peikert and O.Regev, On ideal lattices and learning with errors over rings. Eurocrypt 2010, Lecture Notes in Computer Science, **(2010)**, Vol. 6110, pp. 1-23.
[9] V.Lyubashevsky, C.Peikert, and O.Regev, A Toolkit for Ring-LWE Cryptography. Eurocrypt 2013, Lecture Notes in Computer Science,**(2013)**, Vol. 7881, pp. 35-54.
[10] Z. Chen, W. Jian, C. Liqun and X. Song, Review of How to Construct a Fully Homomorphic Encryption Scheme. International Journal of Security & Its Applications, **(2014)**, Vol. 8, Issue 2, p. 221.
[11] S. Moon, Proceedings of 2015 Workshop on Security, Reliability and Safety. Modified Cooley-Tukey FFT Method for Polynomial Multiplication in Lattice Cryptography, **(2015)** April 14-17, Jeju, Korea.

Authors

Sangook Moon
He received his Bachelor's degree, the Master's degree, and the Ph.D. degree in electronic engineering from Yonsei University, Seoul, Korea in 1995, 1997, and 2002, respectively. From 2002 to 2004, he was with Hynix Semiconductor, Seoul, Korea, where he developed Bluetooth baseband SoCs. Since 2004 he has been with the Department of Electronic Engineering at Mokwon University, Daejeon, Korea, where he currently serves as an Associate Professor. His research interests include computer architecture, embedded systems, SoCs, data encryption, and computer arithmetic.