High-speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems

Donald Donglong Chen, Nele Mentens, Frederik Vercauteren, Sujoy Sinha Roy,
Ray C.C. Cheung, Derek Pao, and Ingrid Verbauwhede

Abstract—**Polynomial multiplication is the basic and most computationally intensive operation in ring-Learning With Errors (ring-LWE) encryption and "Somewhat" Homomorphic Encryption (SHE) cryptosystems. In this paper, the Fast Fourier Transform (FFT) with a linearithmic complexity of $O(n \log n)$, is exploited in the design of a high-speed polynomial multiplier. A constant geometry FFT datapath is used in the computation to simplify the control of the architecture. The contribution of this work is three-fold. First, parameter sets which support both an efficient modular reduction design and the security requirements for ring-LWE encryption and SHE are provided. Second, a versatile pipelined architecture accompanied with an improved dataflow are proposed to obtain a high-speed polynomial multiplier. Third, the proposed architecture supports polynomial multiplications for different lengths n and moduli p. The experimental results on a Spartan-6 FPGA show that the proposed design results in a speedup of 3.5 times on average when compared with the state of the art. It performs a polynomial multiplication in the ring-LWE scheme ($n = 256, p = 1049089$) and the SHE scheme ($n = 1024, p = 536903681$) in only 6.3μs and 33.1μs, respectively.**

Index Terms—**Cryptography, Polynomial multiplication, Number theoretic transform (NTT), FFT Polynomial multiplication, Ring-LWE, SHE, Pipelined architecture, Field-programmable gate array (FPGA).**

I. Introduction

Classical cryptosystems like RSA [1], [2] and Elliptic Curve Cryptography (ECC) [3], [4] are based on the hardness of factoring and the Elliptic Curve Discrete Logarithm Problem (ECDLP), respectively. However, by using the algorithms proposed by Shor [5], factoring and ECDLP can be solved in polynomial time by a quantum computer. Though it is unclear whether a sufficient powerful quantum computer can be built within decades, with the fast improvement of cryptanalysis, computation power, and the unpredictable development of the quantum computer, post-quantum secure and yet practical alternatives are needed.

Lattice-based cryptosystems are good candidates for replacing these classical cryptosystems, because of their security proofs and the fact that there is no known quantum algorithm that can solve the lattice problem efficiently. Based on the security of lattice problems, many cryptographic schemes like the identification scheme [6], encryption schemes [7], [8], and digital signature scheme [9] have been proposed. Among them, many encryption schemes are based on the security of the Learning With Error problem (LWE), which Regev [10] proved is at least as hard as solving certain lattice problems in the worst case.

Beside the significant progress in the theory of lattice-based cryptography, practical implementation issues are gaining attention from the research community [11], [12], [13], [14], [15], [16], [17]. One important milestone in bridging the gap between theory and practice is the introduction of ideal lattices [18]. It reduces the size of the parameter set, which makes the efficient construction of lattice-based cryptosystems feasible.

Apart from the practical use in lattice-based cryptography, ideal lattices can be also applied to the Fully Homomorphic Encryption (FHE) [19] and "Somewhat" Homomorphic Encryption (SHE) scheme [20]. Both FHE and SHE provide secure computation on encrypted data in cloud computing, which is a significant breakthrough in cryptography. However, due to the large key size and the complicated computation, there is still lack of sufficient research on the efficient implementation of these schemes.

In order to facilitate researchers' hardware designs of the ring-LWE and SHE cryptosystems, a versatile and efficient polynomial multiplier would be of great help. This is because the most computationally intensive operation in these cryptosystems is the polynomial multiplication, and an efficient design of the polynomial multiplier will have significant benefit to the performance of the system.

The first hardware design for the LWE encryption scheme [8] is proposed by Göttert *et al.* [21] on FPGA. Due to the fully parallel design, the throughputs of the encryption outperform the software implementation by a factor of 316. The tradeoff of the parallel design is the large area consumption.

Pöppelmann *et al.* [22] proposed a fast and yet compact design for the polynomial multiplication in ring-LWE [23] and SHE [20]. In their architecture, one butterfly unit is designed to compute the FFT and IFFT, which reduces the hardware area usage. However, due to the same design decision of one butterfly, the parallel property of the FFT could not be exploited. Thus, a further speedup of this system is possible.

Aysu *et al.* [24] improved the architecture of [22] and targeted for area-efficient design. The used parameter $p = 2^{16}+1$ is a Fermat number, which enables efficient hardware design. Specifically, it can both enable efficient modular reduction [25] and replace multiplication by powers of ω (ω is a power of 2 in this case) in FFT/IFFT by a simple shift operation.

D. Chen, R. Cheung and D. Pao are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong. (E-mail: donald.chen@my.cityu.edu.hk, {r.cheung, d.pao}@cityu.edu.hk)

N. Mentens, F. Vercauteren, S. Roy and I. Verbauwhede are with ESAT/COSIC and iMinds, KU Leuven, Vlaams-Brabant, Leuven, Belgium. (E-mail: {nele.mentens, frederik.vercauteren, sujoy.sinharoy, ingrid.verbauwhede}@esat.kuleuven.be)

The proposed architecture reduces the number of slices and block RAMs by around 67% and 80%, respectively. It can be seen that the selection of parameters that enable efficient implementations has significant impact on the performance of the system.

In this paper, a versatile pipelined hardware architecture is designed for the high-speed polynomial multiplication. The target of our work is to facilitate users' efficient design of cryptosystems like ring-LWE and SHE. In our work, the Fast Fourier Transform (FFT) [26] with linearithmic complexity and parallel inter-stage computation, is exploited in the design of such polynomial multiplier. The main contributions of this paper are as follows:

- Parameter set selection method which supports both efficient modular reduction and the security requirement for ring-LWE encryption and SHE are analyzed and provided;
- A generic high speed pipelined architecture along with an improved dataflow are designed to exploit the parallelism in polynomial multiplication;
- The proposed parameterized architecture supports the computation of polynomial multiplication for different lengths n and moduli p;
- The implementation results show that our high speed design computes a ring-LWE scheme $n = 256$ multiplication in only 6.3μs and a SHE $n = 1024$ multiplication in 33.1μs, which results in a speedup of approximately 3.5 times compared to [22].

The rest of this paper is organized as follows. Section II recaps the mathematical background. In Section III, the parameter sets for the ring-LWE encryption scheme and SHE are analyzed and an efficient parameter set selection method for modular p reduction are proposed. The high speed pipelined architecture is described in detail in Section IV. Section V presents the implementation results and compares our works with the state of the art. Section VI concludes this paper.

II. Mathematical Preliminaries

Ideal lattices, which define ideals in the ring $\mathbb{Z}_p[x]/\langle f(x)\rangle$ for some irreducible polynomial $f(x)$ of degree n, are good tools for the construction of various cryptographic schemes. The polynomial $f(x)$ is represented as $f(x) = f_0 + f_1x + f_2x^2 + \cdots + f_{n-1}x^{n-1}$. Suppose $a(x)$ and $b(x)$ are two polynomials in the ring. The multiplication of $a(x)$ and $b(x)$ by using the school-book algorithm is calculated as

$$a(x) \cdot b(x) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j x^{i+j} \mod f(x),$$

with a quadratic complexity of $O(n^2)$.

In this paper, we focus on the commonly used case [7] in which $f(x) = x^n + 1$, n is a power of 2, and p is a prime number with $p \equiv 1 \mod 2n$. The choice of $f(x) = x^n + 1$ enables the usage of the property $x^n \equiv -1$, which can simplify the polynomial multiplication as

$$a(x) \cdot b(x) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} (-1)^{\lfloor \frac{i+j}{n} \rfloor} a_i b_j x^{(i+j \mod n)}. \qquad (1)$$

Using this property, the multiplication still has quadratic complexity with n^2 multiplications and $(n-1)^2$ additions or subtractions.

Algorithm 1 Fast Fourier transform algorithm [27]

Let ω be a primitive n-th root of unity in $\mathbb{Z}_p$. Let ω^{-1} be the inverse number of ω such that $\omega\omega^{-1} \equiv 1 \mod p$. Let $\mathbf{a} = (a_0, \ldots, a_{n-1})$ be the coefficient vector of degree n for the polynomial $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ where $a_i \in \mathbb{Z}_p, i = 0, 1, \ldots, n-1$.

Input: $\mathbf{a}, \omega, \omega^{-1}, n, p$.
Output: $\mathbf{A} = \mathrm{FFT}_\omega^n(\mathbf{a})$.

```
a ← $Order\_reverse(\mathbf{a})$
for $i = 0$ to $\log_2 n - 1$ do
    for $j = 0$ to $n/2 - 1$ do
        $P_{ij} \leftarrow \lfloor \frac{j}{2^{\log_2 n-1-i}} \rfloor \times 2^{\log_2 n-1-i}$
        $A_j \leftarrow a_{2j} + a_{2j+1}\omega^{P_{ij}} \mod p$
        $A_{j+\frac{n}{2}} \leftarrow a_{2j} - a_{2j+1}\omega^{P_{ij}} \mod p$
    end for
    if $i \neq \log_2 n - 1$ then
        a ← A
    end if
end for
return A
```

A. Number Theoretic Transform and Fast Fourier Transform

The Number Theoretic Transform (NTT) is a discrete Fourier transform defined over a finite ring $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ [27]. Let ω be a primitive n-th root of unity in $\mathbb{Z}_p$, and $a(x)$, $A(X)$ be polynomials of degree less than n. The n-point NTT are defined as:

$$A_i = \mathrm{NTT}_\omega^n(a(x))_i := \sum_{j=0}^{n-1} a_j \omega^{ij} \mod p \qquad (2)$$

where $i = 0, 1, \ldots, n-1$.

Since p is a prime, n has an inverse n^{-1} modulo p where $n \cdot n^{-1} \equiv 1 \mod p$. Similarly ω also has an inverse ω^{-1}. For Inverse NTT (INTT), the computation is similar to NTT after replacing ω with ω^{-1}. INTT also needs an additional final multiplication by n^{-1} on each element of the output. The NTT can be computed by using the Fast Fourier Transform (FFT) datapath [27]. The FFT algorithm is shown in Algorithm 1.

B. Polynomial Multiplication Using FFT

Using FFT, one can compute the cyclic convolution efficiently. However, the polynomial multiplication in $\mathbb{Z}_p$ is not equal to the cyclic convolution as shown above. It is obvious that the computation would become more efficient if we can compute the polynomial multiplication by using FFT. One way to achieve this goal is by zero padding.

Let ω be a primitive $2n$-th root of unity in $\mathbb{Z}_p$. Let $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ be the coefficient vectors of polynomials $a(x)$ and $b(x)$, respectively. By appending n zeros to construct arrays $\hat{\mathbf{a}} =$

Table I: The operation analysis and comparison between FFT multiplication by using zero padding and negative wrapped convolution.

Operation	Zero padding	Negative wrapped convolution
Multiplication	$3n\log_2 2n+4n$	$\frac{3}{2}n\log_2 n+5n$
Add/Sub	$6n\log_2 2n$	$3n\log_2 n$
Mod p	$9n\log_2 2n+4n$	$\frac{9}{2}n\log_2 n+5n$
Mod x^n+1	1	0

$(a_0,\ldots,a_{n-1},0,\ldots,0)$ and $\hat{\mathbf{b}} = (b_0,\ldots,b_{n-1},0,\ldots,0)$, respectively, the polynomial multiplication (linear convolution) of **a** and **b** can be computed by FFT and IFFT as

$$\mathbf{a}\cdot\mathbf{b} = \mathrm{IFFT}_{\omega}^{2n}(\mathrm{FFT}_{\omega}^{2n}(\hat{\mathbf{a}}) \odot \mathrm{FFT}_{\omega}^{2n}(\hat{\mathbf{b}})),$$

where $\odot$ denotes point-wise multiplication of the coefficients.

In this paper, we refer to the polynomial multiplication using FFT as *FFT multiplication*. It can be seen that FFT and IFFT can be computed with complexity $O(n\log n)$ while the point-wise multiplication can be computed with complexity $O(n)$, thus the FFT multiplication has linearithmic complexity $O(n\log n)$.

C. FFT Multiplication Using Negative Wrapped Convolution

Though the polynomial multiplication using FFT and IFFT can be computed with linearithmic complexity, it needs to double the transformation length and the number of point-wise multiplications due to the zero padding. We can use the negative wrapped convolution method [28] to avoid doubling the effort in the polynomial multiplication in $\mathbb{Z}_p[x]/\langle f(x)\rangle$.

The definition of negative wrapped convolution is as follows. Let $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1})$ be the negative wrapped convolution of $\mathbf{a} = (a_0, a_1, \cdots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, \cdots, b_{n-1})$, then the c_i are computed as

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}.$$

One can find that the equation above is equal to the polynomial multiplication over $\mathbb{Z}_p[x]/\langle x^n+1\rangle$ in equation (1). This indicates we can perform the polynomial multiplication over $\mathbb{Z}_p[x]/\langle x^n+1\rangle$ by computing the negative wrapped convolution.

Details of polynomial multiplication using the negative wrapped convolution are shown in Algorithm 2. In order to guarantee the existence of ϕ which satisfies $\phi^2 \equiv \omega \mod p$, when p is prime and n is power of 2, we should have $p \equiv 1 \mod 2n$.

Using the negative wrapped convolution to compute polynomial multiplication over $\mathbb{Z}_p[x]/\langle x^n+1\rangle$, the modular x^n+1 reduction is eliminated. Moreover, compared with the zero padding method, the length of FFT, IFFT and point-wise multiplication reduces from $2n$ to n.

The operation comparison between FFT multiplication by zero padding and FFT multiplication using negative wrapped convolution is shown in Table I. Compared with the zero padding method, the negative wrapped convolution method saves a considerable number of operations.

Algorithm 2 Polynomial multiplication using FFT

Let ω be a primitive n-th root of unity in $\mathbb{Z}_p$ and $\phi^2 \equiv \omega \mod p$. Let $\mathbf{a} = (a_0,\ldots,a_{n-1})$, $\mathbf{b} = (b_0,\ldots,b_{n-1})$ and $\mathbf{c} = (c_0,\ldots,c_{n-1})$ be the coefficient vectors of degree n polynomials $a(x)$, $b(x)$, and $c(x)$, respectively, where $a_i, b_i, c_i \in \mathbb{Z}_p, i = 0,1,\ldots,n-1$.

Input: $\mathbf{a}, \mathbf{b}, \omega, \omega^{-1}, \phi, \phi^{-1}, n, n^{-1}, p$.
Output: **c** where $c(x) = a(x)\cdot b(x) \mod \langle x^n+1\rangle$.

```
Precompute: $\omega^i, \omega^{-i}, \phi^i, \phi^{-i}$ where $i = 0,1,\ldots,n-1$
for $i = 0$ to $n-1$ do
    $\bar{a}_i \leftarrow a_i\phi^i \mod p$
    $\bar{b}_i \leftarrow b_i\phi^i \mod p$
end for
$\bar{\mathbf{A}} \leftarrow \mathrm{FFT}_{\omega}^{n}(\bar{\mathbf{a}})$
$\bar{\mathbf{B}} \leftarrow \mathrm{FFT}_{\omega}^{n}(\bar{\mathbf{b}})$
for $i = 0$ to $n-1$ do
    $\bar{C}_i \leftarrow \bar{A}_i\bar{B}_i \mod p$
end for
$\bar{\mathbf{c}} \leftarrow \mathrm{IFFT}_{\omega}^{n}(\bar{\mathbf{C}})$
for $i = 0$ to $n-1$ do
    $c_i \leftarrow \bar{c}_i\phi^{-i} \mod p$
end for
return c
```

III. Parameter Set Selection for Efficient Modular Reduction Design

As shown in Table I modular reduction by p is the most frequently used operation in FFT multiplication, so an efficient design will have significant impact on the whole architecture.

A. Modular Reduction Algorithm Analysis

Inspired by [25], [29], modular reduction can be computed efficiently by using Algorithm 3. Carefully examining Algorithm 3 we conclude that a small value of $p[k-2:0]$ (small l) reduces the number of loops in the computation. Furthermore, if $p[k-2:0]$ is a number with low Hamming weight, the addition of the corresponding segments of $x[m-1:k-1]$ can replace the multiplication $x[m-1:k-1]p[k-2:0]$, which reduces the area usage for modular reduction.

One extreme case which meets the above two conditions is a Fermat number, for which $p[k-2:0] = 1$. However, the currently known Fermat numbers larger than $2^{16}+1$ are not prime [30], thus we cannot use them in the lattice-based cryptosystems. In conclusion, the selection of prime p for which $p[k-2:0]$ is small and with low Hamming weight will have great benefit for the performance of the modular reduction operation, thus for the whole FFT multiplication.

B. Efficient and Secure Parameter Selection Restrictions

In order to demonstrate the performance of our design, we choose parameter sets for the implementation of two popular cryptosystems, namely ring-LWE encryption and SHE. For ring-LWE encryption, the parameter sets which can provide medium and long term security are $(n = 256, p = 1049089)$ and $(n = 512, p = 2941249)$, respectively [31]. In terms of

Algorithm 3 Modular p reduction algorithm

Let k and l be the bit length of p and $p[k-2:0]$, respectively. Let x be the input number with a maximum bit length $m = 2k$.

Input: x, k, l, m, p.
Output: $y = x \mod p$.

```
while $m > k$ do
    $x \leftarrow x[k-2:0] - x[m-1:k-1]p[k-2:0]$
    $m \leftarrow m - k + l + 2$
end while
if $x < 0$ then
    $y \leftarrow x + p$
else
    $y \leftarrow x$
end if
return $y$
```

SHE, the parameter sets ($n = 1024, p = 1061093377$) and ($n = 2048, p = 2^{57} + 25 \cdot 2^{13} + 1$) from [20] are chosen.

Taking into account that $p[k-2:0]$ is neither small nor has low Hamming weight for $p = 2941249$ and $p = 1061093377$, these parameters do not result in an efficient design of the modulo p reduction. Though the Hamming weight of the non-adjacent form (Page 98 in [32]) of $p = 1061093377$ is 5, the value of $p[k-2:0]$ is still too large. In order to improve the performance, searching some nice values of p for these two parameter sets are necessary. In order to find some nice p for efficient computation without affecting the security requirement of ring-LWE encryption and SHE, we have the following restrictions during the selection of p.

1) p should be a prime number;
2) In order to guarantee that the FFT multiplication can use negative wrapped convolution, as n is a power of 2, p should satisfy $p \equiv 1 \mod 2n$;
3) In order to meet the security requirement of LWE encryption and SHE, for parameter sets with the same n, the new p should have the same bit size as the original one;
4) p should be a number with low Hamming weight;
5) $p[k-2:0]$ should be a small number.

Restriction 1 and 2 guarantee the existence of primitive n-th root of unity ω and ϕ such that $\phi^2 \equiv \omega \bmod p$, that enables the negative wrapped convolution method in FFT multiplication. Restriction 1 and 3 are for security consideration while restriction 4 and 5 enable efficient modular reduction by p.

C. Complexity Analysis and Parameter Selection Process

For each length-n parameter set, the selection process could be started from the comparison of the k-bit prime numbers. Among these numbers, the p which has both the lowest Hamming weight and the smallest $p[k-2:0]$ would be the best choice. If no such number exists, the process will switch to the comparison of computation complexity for each modular p reduction.

Note that the number of loops in Algorithm 3 is $\lceil \frac{m-k}{k-l-2} \rceil$. Then, in the i-th loop ($i \in [0, \lceil \frac{m-k}{k-l-2} \rceil - 1]$), one $m - (i+1)k + il + 2i + 1$ by l bits constant multiplication and one $k-1$ by $m - (i+1)k + il + 2i + 1$ bits subtraction are required. Let HW be the Hamming weight of $p[k-2:0]$, then one $m - (i+1)k + il + 2i + 1$ by l bits constant multiplication equivalent to $(HW - 1)$ times $m - (i+1)k + il + 2i + 1$ bits additions. In summary, in each loop of the modular p reduction ($i \in [0, \lceil \frac{m-k}{k-l-2} \rceil - 1]$), there are $(HW - 1)$ times $m - (i-1)k + il + 2i + 1$ bits additions, 1 time $k-1$ by $m - (i+1)k + il + 2i + 1$ bits subtraction, and 1 time k bits addition. Note that the complexity of addition and subtraction are $O(k)$. The p which leads to the smallest computation complexity in modular p reduction will be chosen.

Table II: Secure parameter sets selection and comparison for the ring-LWE encryption scheme and the "somewhat" homomorphic encryption scheme. NAF represents non-adjacent form.

Reference	Length of polynomial (n)	p	Hamming Weight
LWE [31]	256	1049089	3
LWE [31]	512	5941249	8
Our proposed	512	4206593	4
SHE [20]	1024	1061093377	13
SHE [20]	1024	1061093377	5 (NAF [32])
Our proposed	1024	536903681	3
SHE [20]	2048	$2^{57} + 25 \cdot 2^{13} + 1$	5

Following the selection method above, we select two new p for parameter sets $n = 512$ and $n = 1024$ which enable the smallest computation efforts in modular p reduction. These parameter sets for LWE and SHE are listed in Table II for easy comparison. One can find that our newly selected p have low Hamming weight and the values of $p[k-2:0]$ are small, which is suitable for efficient modular p reduction design.

IV. Pipelined Architecture for FFT Multiplication

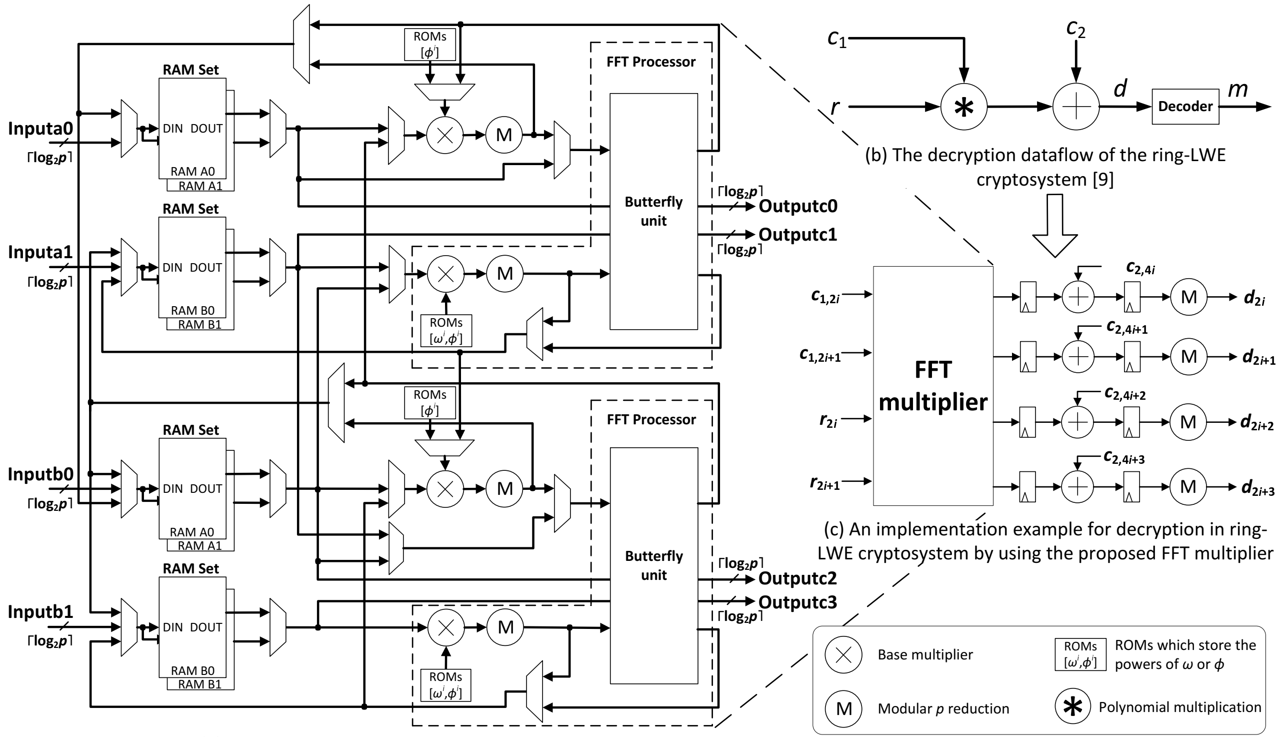
In this section, we will first describe the architecture design of the FFT multiplier. Then the pipelined architecture of the FFT multiplier will be introduced in detail. Finally, the memory control mechanism will be presented.

A. Top Level Architecture Design for FFT Multiplier

The algorithm of polynomial multiplication using FFT is presented in Algorithm 2. Carefully examining the algorithm, one can find that steps 3,4 and steps 6,7 have no data dependency. This enables a high speed design by computing each of these two steps in parallel.

In order to enable two FFT computations in parallel, the smallest architecture consists of two butterflies together with two multipliers. These processing units can compute two coefficients for $\mathrm{FFT}(\bar{\mathbf{a}})$ and $\mathrm{FFT}(\bar{\mathbf{b}})$ concurrently. IFFT is different from FFT in the multiplication by ω^{-i} instead of ω^{i}, and an additional multiplication by n^{-1} at the end of the last stage. Hence, the processing units of the FFT can be fully reused to compute the IFFT. Since only one IFFT instead of two is required, the two butterflies and multipliers are all used to compute $\mathrm{IFFT}(\bar{\mathbf{C}})$ in parallel, which doubles the speed for the IFFT.

With the processing units mentioned above, we found that in the last stage of the FFT, when the two multipliers are



(b) The decryption dataflow of the ring-LWE cryptosystem [9]

(c) An implementation example for decryption in ring-LWE cryptosystem by using the proposed FFT multiplier

(a) The top level architecture for the FFT multiplier

Figure 1: The top level architecture and an implementation example for the proposed FFT multiplier. There are registers between each two operators, we omit them in figure (a) for simplicity.

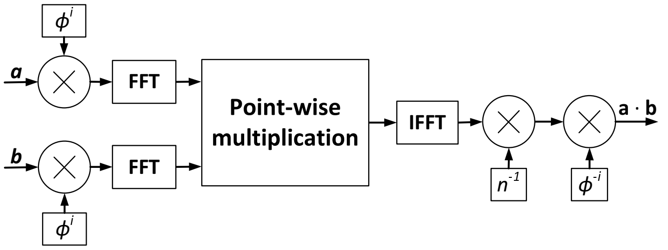


Figure 2: Data flow of the proposed FFT multiplication.

Table III: The cycle analysis of each operation of the proposed FFT multiplier and the improved design. The cycle requirement for the pipeline delay is negligible compared with the total cycle requirement, hence omitted for simplicity.

Operation	Our proposed	Our improved
Multiply by ϕ^i	$\frac{n}{2}$	$\frac{n}{2}$
FFT (stage 0)	$\frac{n}{2}$	
FFT (stage 1 – $\log_2 n - 2$)	$\frac{n}{2}(\log_2 n - 2)$	$\frac{n}{2}(\log_2 n - 2)$
FFT (stage $\log_2 n - 1$) and point-wise multiplication	$\frac{n}{2}$	$\frac{n}{2}$
IFFT	$\frac{n}{4}\log_2 n$	$\frac{n}{4}\log_2 n$
Multiply by n^{-1}	$\frac{n}{4}$	$\frac{n}{4}$
Multiply by ϕ^{-i}	$\frac{n}{4}$	

still working on multiplication by powers of ω, the point-wise multiplication is available for computation. In order to pipeline the last stage of the FFT with point-wise multiplication, two more multipliers are added to compute the point-wise multiplications $\bar{A}_i\bar{B}_i$ and $\bar{A}_{i+\frac{n}{2}}\bar{B}_{i+\frac{n}{2}}$, respectively, between the FFT and IFFT computations.

Following the design ideas mentioned previously, a pipelined architecture for the FFT multiplier is designed and presented in Figure 1 (a). As can be seen from Figure 1 (a), four input buses are designed to enable a pipelined input. After a certain computation cycle, four coefficients of the resulting polynomial appear at the output at each cycle. These coefficient outputs can be fed into other operators of the ring-LWE or SHE cryptosystems for further computation. An implementation example of ring-LWE decryption system by using the proposed FFT multiplier is shown in Figure 1 (c).

1) Proposed Working Dataflow for the FFT Multiplier: The proposed architecture operates according to the dataflow depicted in Figure 2. Two butterflies work in parallel and are responsible for $\mathrm{FFT}(\bar{\mathbf{a}})$ and $\mathrm{FFT}(\bar{\mathbf{b}})$. For the inverse FFT, two butterflies compute one $\mathrm{IFFT}(\bar{\mathbf{C}})$ in parallel. Note that the multiplication by powers of ϕ and point-wise multiplication are absorbed in the first stage and last stage FFT, respectively, thus only $\frac{n}{4}$ more cycles are required for the multiplication by powers of ϕ^{-1}.

The cycle requirement of each operation of the FFT multiplication is shown in Table III. The total cycle requirement for one FFT multiplication is $\frac{3n}{4}\log_2 n + n$.

2) Improved Dataflow for the FFT Multiplier: Carefully examining Algorithm 1, one can find that when $i = 0$ (stage 0), P_{ij} always equal to 0, hence $\omega^{P_{ij}} = 1$. This indicates the multiplication $a_{2j+1} \cdot \omega^{P_{ij}}$ in stage 0 is meaningless because it is always equal to a_{2j+1}. Or putting it in a different way, one can replace the multiplication by $\omega^{P_{ij}}$ in stage 0 by the multiplication by ϕ^i. With this design, the multiplication by ϕ^i can be pipelined with stage 0 of the FFT, which leads to a reduction of $\frac{n}{2}$ cycles compared with the original design.

In the original design, n^{-1} and ϕ^{-i} are precomputed, and

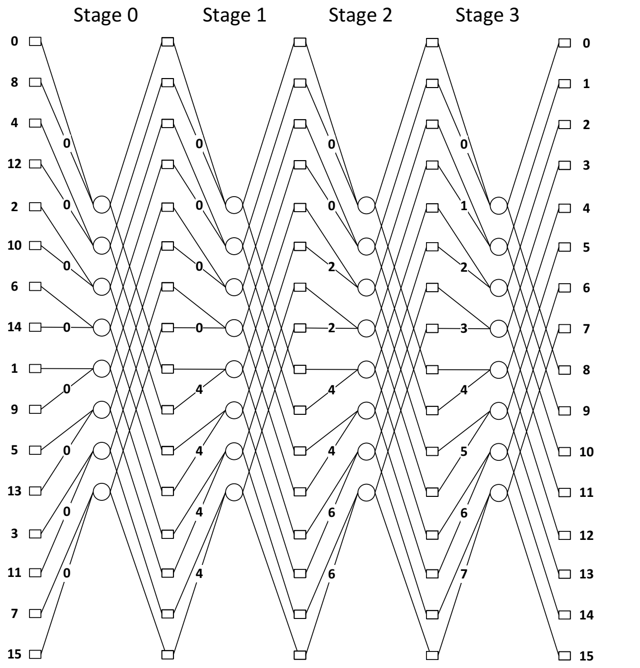


Figure 3: Datapath of the constant geometry FFT. The numbers in the middle of the lines represent the value of P_{ij} in Algorithm 1.

the multiplication of these values is performed separately. Our improved design simplifies the computation by precomputing the products $n^{-1}\phi^{-i}$. With this design the number of cycles for the multiplication by n^{-1} and ϕ^{-i} is reduced from $\frac{n}{2}$ to $\frac{n}{4}$. The cycle requirement of each operation of the FFT multiplication is shown in Table III. The total cycle requirement for one FFT multiplication is reduced to

$$\frac{3n}{4}\log_2 n + \frac{n}{4}.$$

3) Memory Utilization: In order to enable pipelining in our architecture, simple dual-port RAMs (RAMs), which can read and write concurrently, are used to store the coefficient arrays. In our architecture, at least four RAMs are necessary to feed four coefficients into the FFT processors at each cycle; each RAM is responsible for the storage of $n/2$ coefficients. In our design, eight RAMs are employed by using the ping-pong alternative storage mechanism of which the details will be explained in the later subsections.

Instead of generating ω^i, ω^{-i}, ϕ^i, ϕ^{-i} and n^{-1} on the fly, ROMs are used for the storage of these precomputed values. In the FFT computation, one ROM is enough to store the values of ω^i because the two FFT processors always need the same ω^i input. In the IFFT computation, the ROM usage is the same but a dual-port mode is required for the fact that the ω^{-i} inputs are different in the final stage of IFFT.

4) FFT Architecture Selection: Constant geometry FFT [33], which shares the same datapath for all the stages, is a good candidate for our pipelined design. The constant geometry FFT algorithm is shown in Algorithm 1. A 16-point constant geometry FFT datapath is depicted in Figure 3. As can be seen, the datapath for constant geometry FFT is the same both inter- and intra-stage, which makes the read/write control of the sub-stage architecture as simple as of the whole stage.

One may argue that in-place FFT is also a good candidate, in which the storage addresses of input and output are the same. However, the read/write addresses of in-place FFT are different from one stage to another. This makes the read/write control of the in-place FFT more complex than the constant geometry FFT; different control logics are required to manipulate the RAM read/write addresses in different stages. Since high speed is the primary target of our design, the simple control in constant geometry FFT, which could lead to a higher operating frequency of the multiplier, becomes our first choice.

B. Building Blocks and Memory Control Mechanism

1) Modular p Reduction: The architecture for modular reduction by $p = 1048098$ is introduced as an example in Figure 4. Note that the design for other p is similar. The newly selected p are numbers with low Hamming weight, thus the constant multiplication $x[m-1:k-1]p[k-2:0]$ can be performed by adding the corresponding segments of $x[m-1:k-1]$. Using this approach can both reduce the carry chain of the adder and save logic for the shift operation. Moreover, when the bit-width for the shift operation is larger than the bit width of $x[m-1:k-1]$ as shown in the solid line box in Figure 4, the addition can be also eliminated.

It is worth to note that after the first subtraction, the later operations should support a signed representation of the operands. After reducing the bit length to $\lceil \log_2 p \rceil$ (signed number), a detection on the sign bit decides whether a further addition is required to bound the value within $[0, p-1]$.

2) Butterfly Unit for FFT/IFFT: The butterfly unit for constant geometry FFT/IFFT is shown in Figure 5. The butterfly processor and the channel selector constitute the butterfly unit. The butterfly processor performs the addition and subtraction with modular reduction in FFT/IFFT (i.e. Step 5, 6 in Algorithm 1). A channel selector [34] is responsible for wiring the coefficients back to their corresponding RAMs.

3) Pipeline Depth Control for Pipeline read/write: The pipelined dataflow example of a 16-point constant geometry FFT/IFFT is depicted in Figure 6. In order to achieve 100% usage of the FFT processor in the FFT/IFFT computation, two coefficients are required as inputs at each cycle consecutively without pipeline bubbles intra- and inter- stage. Therefore, we should first guarantee that within each stage, the RAM input time $d + \frac{n}{2} - 1 + \frac{n}{2}(k-1)$ (k is the stage number) of the last two coefficients (i.e. $a_{\frac{n}{2}-1}, a_{n-1}$) is smaller than the output time $\frac{3n}{4} - 1 + \frac{n}{2}(k-1)$ of the first output among these two coefficients (i.e. $a_{\frac{n}{2}-1}$). Hence, the pipeline delay should satisfy $d < n/4$ in order to guarantee the values within each stage connect without any bubble.

Figure 6 provides an example for this situation. In this figure, where $d = 3 < n/4$, the RAM input time (time 18) of the last two coefficients (i.e. a_7, a_{15}) of stage 2 is smaller than the RAM output time (time 19) of the first output among these two coefficients (i.e. a_7) within the same stage. Thus the RAM output can follow the RAM input immediately without any delay within the stage.

However, if the condition $d < n/4$ is met, a read/write collision will occur between each stage in the RAMs; the

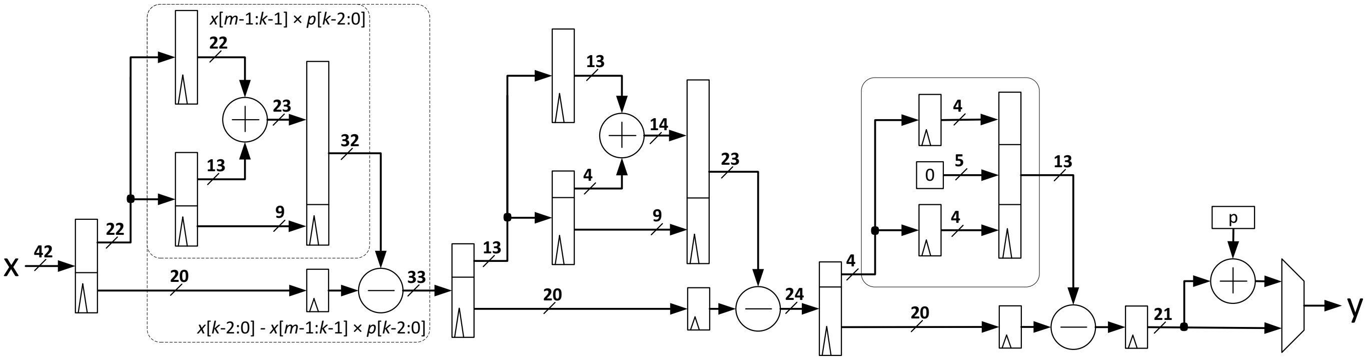


Figure 4: Architecture of the modular reduction by $p = 1048098$. As shown in the solid line box, when the bit width for shift operation (The shift bit width is 9) is larger than the bit width of $x[m-1:k-1]$ (The bit width of $x[m-1:k-1]$ is 4), the addition can be eliminated.

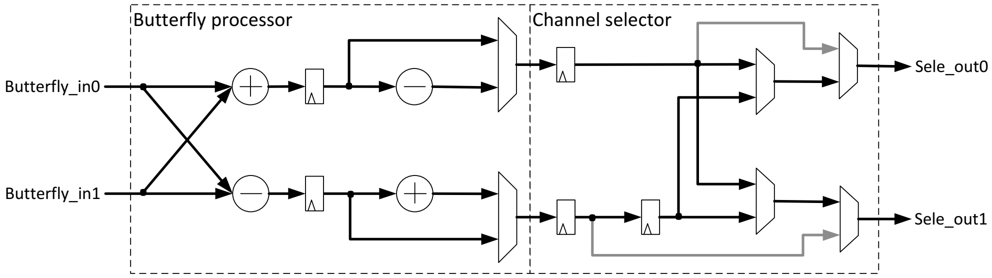


Figure 5: Architecture of the butterfly unit. Selecting the black-colored channels in the channel selector enables output schedule as shown in Figure 8 while selecting the gray-colored channels enables the in-sequence output schedule.

processed new data for the next stage FFT/IFFT will overwrite the unread data of the current stage. As described as an example in Figure 6, coefficient a_0, a_1 of stage 1 will be at the output of the RAM at time 8, after $d = 3$ cycles, the processed new coefficient a_8 for stage 2 will be written back to RAM at time 11. However, following the schedule of FFT/IFFT, the coefficient a_8 of stage 1 will not be computed until time 12. Since the values of a_8 for stage 1 and stage 2 should be stored in the same RAM address, a direct write back of the stage 2 a_8 will overwrite the unread stage 1 a_8 value. This will create a wrong stage 1 FFT input of a_8 at time 13.

Therefore, a read-before-write behavior could only provide a correct read for the coefficients with indices smaller than $d/2$. In order to solve this collision problem without adding pipelining bubbles, one could double the number of RAM blocks and use the ping-pong alternative storage method. Manipulated by the control signals, multiplexers are used to select the RAM blocks as shown in Figure 1.

4) Coefficient RAM control mechanism: Carefully examining the coefficient input schedules for FFT/IFFT (output schedules for the RAMs) of each stage of Algorithm 1, we can find that there are two input schedules for the butterfly of the constant geometry FFT. The first input schedule is for stage 0, where the coefficients with small indices $(a_0, \ldots, a_{\frac{n}{2}-1})$ are led to one input, while the coefficients with large indices $(a_{\frac{n}{2}}, \ldots, a_{n-1})$ are led to the other input. The second type of input schedule is for the rest of the stages, where the coefficients with even indices $(a_0, a_2, \ldots, a_{n-2})$ are led to the first input, while the coefficients with odd indices $(a_1, a_3, \ldots, a_{n-1})$ are led to the other input. Therefore, we can arrange the coefficient storage in the RAMs as shown in Figure 7. With this design, a reverse order read operation in stage 0 and a sequence read in other stages provides the correct coefficient schedule for FFT/IFFT.

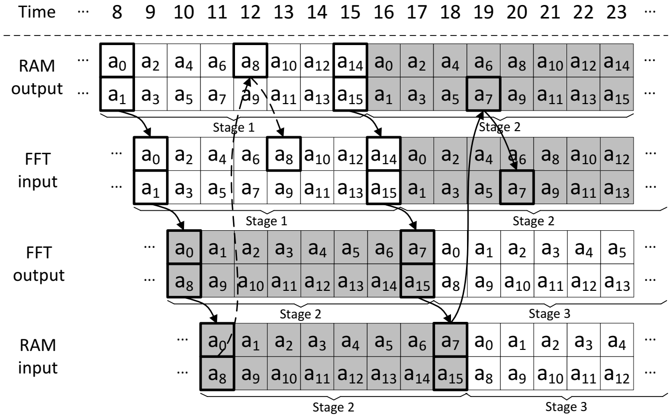
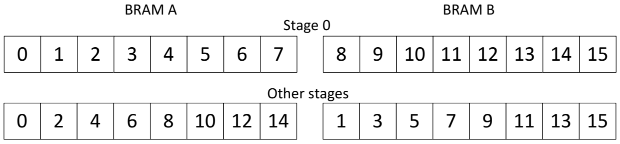

Time ··· 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ···
RAM output
··· a0 a2 a4 a6 a8 a10 a12 a14 a0 a2 a4 a6 a8 a10 a12 a14 ···
··· a1 a3 a5 a7 a9 a11 a13 a15 a1 a3 a5 a7 a9 a11 a13 a15 ···
Stage 1
Stage 2
FFT input
··· a0 a2 a4 a6 a8 a10 a12 a14 a0 a2 a4 a6 a8 a10 a12 ···
··· a1 a3 a5 a7 a9 a11 a13 a15 a1 a3 a5 a7 a9 a11 a13 ···
Stage 1
Stage 2
FFT output
··· a0 a1 a2 a3 a4 a5 a6 a7 a0 a1 a2 a3 a4 a5 ···
··· a8 a9 a10 a11 a12 a13 a14 a15 a8 a9 a10 a11 a12 a13 ···
Stage 2
Stage 3
RAM input
··· a0 a1 a2 a3 a4 a5 a6 a7 a0 a1 a2 a3 a4 ···
··· a8 a9 a10 a11 a12 a13 a14 a15 a8 a9 a10 a11 a12 ···
Stage 2
Stage 3

Figure 6: The pipelined dataflow and the read/write collision example of a 16-point constant geometry FFT. The solid arrows guide the coefficient dependency without read/write collision while the dashed arrows lead the coefficient dependency which will cause read/write collision.

When it comes to the input schedule of RAMs, we should first consider the output schedule of stage 0 to stage $\log_2 n-2$ in the FFT. From Algorithm 1, we can find that the index gap between the outputs of each butterfly is $n/2$, and these two indices have the same parity. Since simple dual-port RAMs are used, with the storage arrangement shown in Figure 7, these two outputs cannot be written back to the same RAM concurrently. Notice that the output indices between two cycles have different parity. The channel selector can be used to tackle this problem. The input/output schedule of the butterfly



BRAM A | BRAM B

Stage 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Other stages

0	2	4	6	8	10	12	14	1	3	5	7	9	11	13	15

Figure 7: Coefficient storage arrangement of 16-point FFT/IFFT in RAMs.

processor and the channel selector is depicted in Figure 8. With this design, the two outputs from the butterfly can be written back to the corresponding RAMs at each cycle with only 1 cycle delay.

The output schedule of stage $\log_2 n - 1$ in the FFT determines the input of stage 0 of the IFFT. In order to make the output of the IFFT in sequence order, the input of the IFFT should be in reverse order as compared to the FFT. Hence, the coefficient storage in stage 0 for the IFFT in RAMs should be in sequence, which is the same as stage 0 in the FFT shown in Figure 7. Note that the indices of two coefficients generated by the butterfly processor have the same parity, which makes a direct write back to the RAMs possible. By selecting the gray-colored channels from the channel selector, which are shown in Figure 5, the in-sequence coefficient storage can be implemented.

V. FPGA Optimization results and Comparisons

The proposed architecture is implemented on a Spartan-6 (xc6slx100-3) FPGA using Verilog. The post place-and-route (PAR) implementation results are generated using Xilinx ISE 14.7 with the default synthesis option.

In order to facilitate future comparisons, the results which the FFT multipliers are built by pure LUTs are provided (**Design 1**). In order to guarantee the speed and have a relatively fair comparison of the area-latency product with [22], we tried to construct the base multiplier by using as less DSPs as in [22]. For this reason the 58×58-bit base multipliers for $n = 2048$ FFT multiplier are constructed by using Karatsuba's algorithm [35]. We also tried to reduce the BRAM number by using block memory to realize RAMs and using distributed memory to build ROMs (**Design 2**). In order to achieve the high-speed target, we fully use the DSPs and block RAMs (BRAMs) in the device to realize the base multiplier and RAM/ROM, respectively (**Design 3**). The implementation results of the above designs are shown in Table IV.

In our design, LUT based components are combined with registers to split into balanced delay paths, which make the design operate under high frequency. Note that the critical path is the route between the input and output of the base multiplier or the RAM. Therefore, the speed performance of the base multiplier or the RAM determines the operating frequency of the whole system. The cycle delay of the base multiplier is set to the optimum number in order to achieve the highest operating frequency, thus the total cycle count may be different for the same length n FFT multipliers.

The comparisons between the latest work of Pöppelmann *et al*. [22] and our Design 2 are shown in Table V. We compare the area-latency product for the FFT multipliers which have the same number of DSPs and less BRAM usage than [22] ($n = 256$, 512, and 1024). The best achieved improvement on area-latency product is 68% for the $n = 512$ multiplier. The performance gain is mainly thanks to the newly selected p, which enables a more efficient modular p reduction design.

Table IV: Implementation results of the proposed high-speed polynomial multiplier on a Spartan-6 (xc6slx100-3) FPGA. The superscript number (1), (2), and (3) for length n represent **Design 1, 2, and 3**, respectively. A 18Kbit BRAM is denoted as 1 BRAM while 0.5 BRAM represents a 9Kbit one.

Length n	LUT	Slice	DSP	BRAM	Period (ns)	Cycles	Latency (μs)
$256^{(1)}$	4407	1471	0	0	4.050	1618	6.552
$256^{(2)}$	2829	886	4	4	3.877	1618	6.272
$256^{(3)}$	1754	580	16	8.5	3.802	1630	6.197
$512^{(1)}$	6119	2062	0	0	4.454	3618	16.114
$512^{(2)}$	3750	1348	4	4	3.938	3622	14.263
$512^{(3)}$	2502	870	16	8.5	3.946	3630	14.323
$1024^{(1)}$	10801	3176	0	0	5.150	7959	40.988
$1024^{(2)}$	6689	2112	4	8	4.154	7967	33.094
$1024^{(3)}$	2464	915	16	14	4.050	7971	32.282
$2048^{(1)}$	37552	10120	0	0	9.471	17382	164.62
$2048^{(2)}$	20762	6154	12	28	5.423	17402	94.371
$2048^{(2)}$	14105	4406	12	50	4.805	17402	83.616
$2048^{(3)}$	6295	2374	64	50	4.751	17454	82.923

The improvement is 36.5% for the $n = 1024$ FFT multipliers. This is because the usage of distributed memory and the construction of the base multipliers (using both DSP and LUTs) consume more LUTs in this case, which increase the number of slices and reduce the operating frequency. It can be seen from the latency comparison of the two Design 2 results for $n = 2048$ multiplier that for the large data size FFT multiplier, block memory instead of distributed memory is preferred to achieve high speed in FPGA.

In our design, the cycle requirement of a length-n polynomial multiplication is

$$\frac{3n}{4}\log_2 n + \frac{n}{4}.$$

In the work of [22] the cycle requirement is

$$\frac{3n}{2}\log_2 n + \frac{11n}{2}.$$

Compared with [22], our design will save at least 50% of the cycles in the computation, theoretically. It can be seen that the cycle reduction is higher than 60% for the four FFT multipliers in our design.

The latency comparison between [22] and our design is depicted in Figure 9. Compared with [22], our high-speed pipelined architecture has approximately 3.5 times speedup on average. The speedup is mainly achieved by the reduction in cycles and the increase of operating frequency. The cycle reduction is achieved thanks to the pipelined design and the improved dataflow while the increase in frequency is achieved by the simple read/write control of the constant geometry FFT. When compared with the software implementation from [20], which takes $11ms$ for an $n = 2048$ FFT multiplication by using 2.1GHz Intel Core 2 Duo, our design realizes a speedup of approximately 130 times.

VI. Conclusions and Future Works

A high-speed pipelined design for FFT multiplication is presented for efficient implementation of ring-LWE and "some-

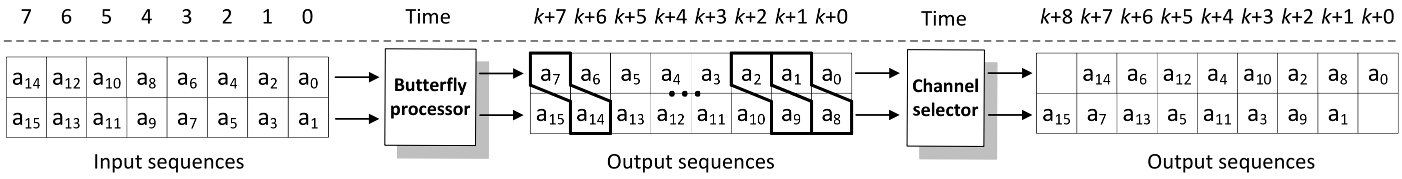

7 6 5 4 3 2 1 0
Time
k+7 k+6 k+5 k+4 k+3 k+2 k+1 k+0
Time
k+8 k+7 k+6 k+5 k+4 k+3 k+2 k+1 k+0
a14 a12 a10 a8 a6 a4 a2 a0
a15 a13 a11 a9 a7 a5 a3 a1
Butterfly processor
a7 a6 a5 a4 a3 a2 a1 a0
a15 a14 a13 a12 a11 a10 a9 a8
Channel selector
a14 a6 a12 a4 a10 a2 a8 a0
a15 a7 a13 a5 a11 a3 a9 a1
Input sequences
Output sequences
Output sequences

Figure 8: Input and output schedule of the butterfly processor and the channel selector of a 16-point FFT (one stage example from stage 1 to stage $\log_2 n - 2$). k is the pipelined delay of the butterfly processor.

Table V: Comparisons between the design of Pöppelmann *et al.* [22] and our high-speed FFT multiplier on a Spartan-6 (xc6slx100-3) FPGA.

Design	Application	Length n	Slice	DSP	BRAM	Period (ns)	Cycles	Latency (μs)	Area-latency product (slice × μs)	Improvement (%)	Cycle red. (%)	Speedup (times)
[22]	LWE	256	640	4	5.5	4.587	4806	22.045	14108	-	-	-
Design 2	LWE	256	886	4	4	3.877	1618	6.272	5556	60.6	66.3	×3.51
[22]	LWE	512	1145	4	7	5.181	10174	52.711	60354	-	-	-
Design 2	LWE	512	1348	4	4	3.938	3622	14.263	19226	68.1	64.4	×3.70
[22]	SHE	1024	997	4	11.5	5.154	21405	110.32	109989	-	-	-
Design 2	SHE	1024	2112	4	8	4.154	7967	33.094	69894	36.5	62.8	×3.33
[22]	SHE	2048	1310	16	22.5	6.211	45453	282.31	369826	-	-	-
Design 2	SHE	2048	6154	12	28	5.423	17402	94.371	580759	-	61.7	×2.99
Design 2	SHE	2048	4406	12	50	4.805	17402	83.616	368412	-	61.7	×3.38

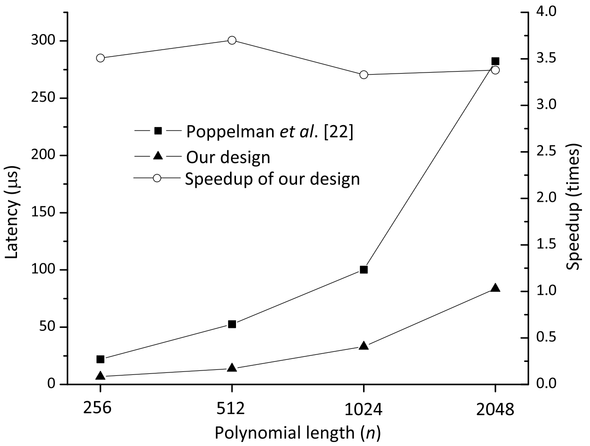


Figure 9: The comparison of the latency between the design of Pöppelmann $et\ al.$ [22] and our design.

what" homomorphic encryption cryptosystems. Parameter selection restrictions and an efficient selection method are analyzed and provided. Considering both security and efficiency of the design, new parameter sets for the ring-LWE encryption scheme and the SHE scheme are selected. Four parameter sets with n ranging from 256 to 2048 are implemented to demonstrate the performance of our FFT multipliers. The implementation results on a Spartan-6 FPGA show that our architecture achieves a 3.5 times speedup on average when compared with the state of the art. The results also show that the selected new parameters p support efficient modular p reduction design, which improves the performance of the whole FFT multiplier.

Future works will exploit the full usage of the processing units of the proposed pipelined architecture, and will simplify the control logic of our design. We will consider using the ATHENa framework to improve the synthesis results [36]. We will also investigate how to incorporate the proposed FFT multiplier in the ring-LWE and SHE cryptosystems.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] M.-D. Shieh, J.-H. Chen, W.-C. Lin, and H.-H. Wu, "A new algorithm for high-speed modular multiplication design," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 9, pp. 2009–2019, Sept 2009.

[3] N. Koblitz, A. Menezes, , and S. Vanstone, "The state of elliptic curve cryptography," *Designs, Codes and Cryptography*, vol. 19, no. 2-3, pp. 173–193, 2000.

[4] R. Azarderakhsh, K. Jarvinen, and M. Mozaffari-Kermani, "Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 4, pp. 1144–1155, April 2014.

[5] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, 1994, pp. 124–134.

[6] V. Lyubashevsky, "Lattice-based identification schemes secure under active attacks," in *Public Key Cryptography PKC 2008*, ser. Lecture Notes in Computer Science, R. Cramer, Ed. Springer Berlin Heidelberg, 2008, vol. 4939, pp. 162–179.

[7] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology EUROCRYPT 2010*, ser. Lecture Notes in Computer Science, H. Gilbert, Ed. Springer Berlin Heidelberg, 2010, vol. 6110, pp. 1–23.

[8] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Topics in Cryptology CT-RSA 2011*, ser. Lecture Notes in Computer Science, A. Kiayias, Ed. Springer Berlin Heidelberg, 2011, vol. 6558, pp. 319–339.

[9] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Advances in Cryptology EUROCRYPT 2012*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds. Springer Berlin Heidelberg, 2012, vol. 7237, pp. 738–755.

[10] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. New York, NY, USA: ACM, 2005, pp. 84–93.

[11] J. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden, "Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers," in *Selected Areas in Cryptography – SAC 2013*, ser. Lecture Notes in Computer Science, T. Lange, K. Lauter, and P. Lisonk, Eds. Springer Berlin Heidelberg, 2014, pp. 402–417.

[12] S. Sinha Roy, F. Vercauteren, and I. Verbauwhede, "High precision discrete gaussian sampling on fpgas," in *Selected Areas in Cryptography – SAC 2013*, ser. Lecture Notes in Computer Science, T. Lange, K. Lauter, and P. Lisonk, Eds. Springer Berlin Heidelberg, 2014, pp. 383–401.

[13] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-lwe based cryptoprocessor," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, ser. LNCS. Springer, 2014, vol. PP, pp. 1–1.

[14] T. Pöppelmann, L. Ducas, and T. Güneysu, "Enhanced lattice-based signatures on reconfigurable hardware," Cryptology ePrint Archive, Report 2014/254, 2014, http://eprint.iacr.org/.

[15] A. Boorghany and R. Jalili, "Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers," Cryptology ePrint Archive, Report 2014/078, 2014, http://eprint.iacr.org/.

[16] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Compact and side channel secure discrete gaussian sampling," Cryptology ePrint Archive, Report 2014/591, 2014, http://eprint.iacr.org/.

[17] T. Güneysu, V. Lyubashevsky, and T. Pöeppelmann, "Lattice-based signatures: Optimization and implementation on reconfigurable hardware," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[18] V. Lyubashevsky and D. Micciancio, "Generalized compact knapsacks are collision resistant," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, vol. 4052. Springer Berlin Heidelberg, 2006, pp. 144–155.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178.

[20] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, ser. CCSW '11. ACM, 2011, pp. 113–124.

[21] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in *Cryptographic Hardware and Embedded Systems CHES 2012*, ser. Lecture Notes in Computer Science, E. Prouff and P. Schaumont, Eds. Springer Berlin Heidelberg, 2012, vol. 7428, pp. 512–529.

[22] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology LATINCRYPT 2012*, ser. Lecture Notes in Computer Science, A. Hevia and G. Neven, Eds. Springer Berlin Heidelberg, 2012, vol. 7533, pp. 139–158.

[23] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 197–206.

[24] A. Aysu, C. Patterson, and P. Schaumont, "Low-cost and area-efficient fpga implementations of lattice-based cryptography," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, 2013, pp. 81–86.

[25] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Computer Arithmetic – ARITH 1999*, 1999, pp. 158 – 167.

[26] J. Cooley and J. Turkey, "An algorithm for the machine computation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[27] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of Computation*, vol. 25, pp. 365–374, 1971.

[28] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *Fast Software Encryption*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed. Springer Berlin Heidelberg, 2008, vol. 5086, pp. 54–72.

[29] J. A. Solinas, "Generalized Mersenne Numbers," Faculty of Mathematics, University of Waterloo, Tech. Rep., 1999.

[30] H. J. Naussbaumer, *Fast Fourier transform and convolution algorithms*. Springer, Berlin, Germany, 1982.

[31] M. Rückert and M. Schneider, "Estimating the security of lattice-based cryptosystems," in *Cryptology ePrint Archive, Report 2010/137*, 2010.

[32] D. Hankerson, A. Menezes, and S. Vanstone, in *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[33] M. Pease, "An adaptation of the fast Fourier transform for parallel processing," *Journal of the Association for Computing Machinery*, vol. 15, pp. 252–264, 1968.

[34] D. Chen, G. Yao, C. Koc, and R. C. C. Cheung, "Low complexity and hardware-friendly spectral modular multiplication," in *Field-Programmable Technology (FPT), 2012 International Conference on*, 2012, pp. 368–375.

[35] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, no. 7, pp. 595–596, 1963.

[36] "ATHENa: Automated Tool for Hardware EvaluatioN." [Online]. Available: http://cryptography.gmu.edu/athena/