An Efficient and Scalable Sparse Polynomial Multiplication Accelerator for LAC on FPGA

Jipeng Zhang[1], Zhe Liu[12(✉)], Hao Yang[1], Junhao Huang[1], Weibin Wu[1]
[1]*Nanjing University of Aeronautics and Astronautics, Jiangsu, China*
Email: jp-zhang@outlook.com, {yang_hao, zhe.liu}@nuaa.edu.cn,
jhhuang_nuaa@126.com, wuweibin@nuaa.edu.cn
[2]*State Key Laboratory of Cryptology, Beijing, China*

***Abstract*—LAC, a Ring-LWE based scheme, has shortlisted for the second round evaluation of the National Institute of Standards and Technology Post-Quantum Cryptography (NIST-PQC) Standardization. FPGAs are widely used to design accelerators for cryptographic schemes, especially in resource-constrained scenarios, such as IoT. Sparse Polynomial Multiplication (SPM) is the most compute-intensive routine in LAC. Designing an accelerator for SPM on FPGA can significantly improve the performance of LAC. However, as far as we know, there are currently no works related to the hardware implementation of SPM for LAC. In this paper, the proposed efficient and scalable SPM accelerator fills this gap. More concretely, we firstly develop the Dual-For-Loop-Parallel (DFLP) technique to optimize the accelerator's parallel design. This technique can achieve 2x performance improvement compared with the previous works. Secondly, we design a hardware-friendly modular reduction algorithm for the modulus 251. Our method not only saves hardware resources but also improves performance. Then, we launch a detailed analysis and optimization of the pipeline design, achieving a frequency improvement of up to 34%. Finally, our design is scalable, and we can achieve various performance-area trade-offs through parameter p. Our results demonstrate that the proposed design can achieve a very considerable performance improvement with moderate hardware area costs. For example, our medium-scale architecture for LAC-128 takes only 783 LUTs, 432 FFs, 5BRAMs, and no DSP on an Artix-7 FPGA and can complete LAC's polynomial multiplication in 8512 cycles at a frequency of 202MHz.**

***Keywords*-lattice-based cryptography; Ring-LWE; sparse polynomial multiplication; hardware accelerator; LAC; FPGA;**

I. Introduction

Public-key cryptography has been widely used to provide encryption, key exchange, and digital signature functions. For a long time, public-key cryptographic schemes such as RSA and ECC have been secure against all known attacks. However, back in 1994, Shor's Algorithm [1] claimed to be able to compute discrete logarithm and integer factorization problems in polynomial time on large quantum computers, thus threatening the security of traditional public-key cryptography such as RSA and ECC. As quantum computing is developing rapidly, quantum computers that are large enough to run Shor's Algorithm seem possible to appear soon. Alternative cryptographic schemes are urgent to be developed to replace the existing public-key cryptography.

Post-quantum cryptography is one kind of cryptography that can resist attacks from both classical computers and quantum computers. There are currently several categories in post-quantum cryptography, including lattice-based cryptography, code-based cryptography, and multivariate cryptography. Among all kinds of post-quantum cryptography, lattice-based cryptography is the most promising one to replace traditional public-key cryptography, due to its quantum-resistant properties, versatility, and efficiency.

Learning with Errors over Ring (Ring-LWE) [2] problem is one of the lattice-based hard problems. The algebraic structure of Ring-LWE based cryptographic schemes is polynomial rings over finite fields. LAC scheme [3] is based on Ring-LWE and shortlisted for NIST-PQC Standardization Round 2 evaluation [4]. One of the unique features of the LAC scheme is its small modulus 251, which can be packed into a single byte.

The most compute-intensive routine in the LAC scheme is polynomial multiplication. The coefficients of secret and noise polynomials in LAC follow the central binomial distribution, and their value range is $\{-1, 0, 1\}$. It is worth noting that more than half of the secret polynomial coefficients are 0 in LAC. Therefore, when multiplied with this polynomial, if the coefficient were 0, then the corresponding multiplication can be eliminated. This technique is the core idea of SPM. The complexity of SPM depends on the number of non-zero coefficients. Intuitively, SPM can halve the number of multiplication instructions in LAC. Besides, the secret polynomial non-zero coefficient in LAC is either -1 or 1, a number multiplied by 1 gets itself, and a number multiplied by -1 gets its opposite. Therefore, polynomial multiplication in LAC can be implemented with simple addition and subtraction instructions, eliminating the use of expensive multiplication instructions. In conclusion, SPM in LAC shows significant performance improvements compared with

✉**Zhe Liu is the corresponding author of this paper.**

Schoolbook multiplication.

In this paper, our efficient and scalable SPM accelerator fills the gap that there is no related work dedicated to hardware design for LAC. Our contributions include several optimizations for more reasonable hardware resource utilization, higher circuit frequency, and scalable design. More specifically, the contributions can be summarized as follows.

1) Parallel design is a critical factor for improving the performance of SPM. To achieve higher parallelism and better performance, we propose the Dual-For-Loop-Parallel (DFLP) technique. This technique aims to realize the dual-parallel design of the outer and inner loop. In this way, the number of clock cycles is halved with negligible resource costs.
2) The modular reduction is an essential routine in polynomial multiplication. In software implementations, Barrett reduction [5], Montgomery reduction [6], and SAMS2 [7] are the most commonly used algorithms, but the expensive multiplications in these algorithms are unfriendly to hardware. Therefore, we design a new modular reduction for the modulus $q = 251$, which is better than the straightforward method used in [8] and does not require additional multiplications. As a result, this approach presents an efficient and time-constant modular reduction implementation and can also improve the frequency of our accelerator.
3) The maximum frequency of hardware design is limited by pipeline design and the length of the critical path. We launch a detailed analysis and optimization of the pipeline design for the instance, which is instantiated from [8] with LAC's parameters. We shorten several critical paths of this instance, generating a flatter design. A combination of this contribution and the 2nd contribution can achieve an up to 34% improvement in frequency.
4) Our accelerator is scalable because we can achieve various performance-area trade-offs through tuning parameter $p(= 2, 4, 8, 16)$. The parameter p is used to control the degree of parallelism. The design with small $p(= 2, 4)$ is suitable for resource-constrained scenarios, such as IoT. The design with big $p(= 16)$ is capable of performance-first scenarios. Besides, the design with $p = 8$ is medium-scale, which can achieve decent performance with moderate hardware resources costs.

II. Background and Related Work

A. The LAC Scheme

Let $\mathbb{Z}$ be integers, for any integer $q \geq 1$, $\mathbb{Z}_q$ denotes the residue class ring modulo q. $R_q(x) = \mathbb{Z}_q(x)/(x^n + 1)$ defines the quotient polynomial ring $R_q(x)$ where all coefficients of the ring are in $\mathbb{Z}_q$.

The security basis of the LAC scheme is the Ring-LWE problem. LAC scheme consists of four public-key primitives: a public key encryption scheme with IND-CPA security (LAC.CPA.PKE), a key encapsulation scheme with IND-CCA security (LAC.CCA.KEM), a key exchange protocol with passively security (LAC.KE), and an authenticated key exchange protocol (LAC.AKE).

The entire LAC scheme's foundation is the IND-CPA secure public-key encryption (PKE), namely LAC.CPA.PKE. It consists of three algorithms:

KG: As described in Algorithm 1, public key pk and secret key sk are randomly generated. The subroutine aes_ctr takes $seed$ as input and yields three pseudorandom seeds: $seed_a, seed_{sk}, seed_e$. The difference between $Sample_{sk}$ and $Sample_e$ is that the polynomial coefficients generated in the former case are the index of the non-zero coefficients. However, the output of the latter is an original sparse polynomial.

Enc: Algorithm 2, the encryption algorithm, takes message m and public key pk as input and it encrypts the message m using public key. The subroutine ECCEnc deals with the encoding of the error correction codes, and the parameter l_v denotes the length of the encoding.

Dec: The message m can be recovered from ciphertext c using the corresponding sk through Algorithm 3. Decoding with redundant error correction information, which can reduce the failure rate of decryption, is performed within the subroutine ECCDec.

Algorithm 1 KG

```
Ensure: a pair of public key and secret key $(pk, sk)$
1: $seed \xleftarrow{\$} \{0, 1, \ldots, 255\}^{32}$
2: $seed_a, seed_{sk}, seed_e \leftarrow aes\_ctr(3 * 32, seed)$
3: $\vec{a} \leftarrow GenA(seed_a) \in R_q$
4: $\vec{s} \leftarrow Sample_{sk}(seed_{sk}) \in \Psi_n^h$
5: $\vec{e} \leftarrow Sample_e(seed_e) \in \Psi_n^h$
6: $\vec{b} \leftarrow \vec{a}\vec{s} + \vec{e} \in R_q$
7: return $\left(pk := \left(seed_a, \vec{b}\right), sk := \vec{s}\right)$
```

Algorithm 2 Enc $(pk, \vec{m} \in \mathcal{M}, seed')$

```
Require: public key $pk$, message $m$, and a seed $seed'$
Ensure: ciphertext $\vec{c}$
1: $seed_r, seed_{e1}, seed_{e2} \leftarrow aes\_ctr(3 * 32, seed')$
2: $\vec{a} \leftarrow GenA(seed_a) \in R_q$
3: $\vec{r} \leftarrow Sample_{sk}(seed_r) \in \Psi_n^h$
4: $\vec{e_1} \leftarrow Sample_e(seed_{e1}) \in \Psi_n^h$
5: $\vec{e_2} \leftarrow Sample_e(seed_{e2}) \in \Psi_n^h$
6: $\widehat{\vec{m}} \leftarrow \mathrm{ECCEnc}(\vec{m}) \in \{0, 1\}^{l_v}$
7: $\vec{c}_1 \leftarrow \vec{a}\vec{r} + \vec{e}_1 \in R_q$
8: $\vec{c}_2 \leftarrow (\vec{b}\vec{r})_{l_v} + \vec{e}_2 + \left\lfloor \frac{q}{2} \right\rceil \cdot \widehat{\vec{m}} \in \mathbb{Z}_q^{l_v}$
9: return $\vec{c} := (\vec{c}_1, \vec{c}_2) \in R_q \times \mathbb{Z}_q^{l_v}$
```

Algorithm 3 $\mathrm{Dec}\,(sk = \vec{s}, \vec{c} = (\vec{c}_1, \vec{c}_2))$

Require: secret polynomial $\vec{s}$, and ciphertext $\vec{c}$
Ensure: plaintext $\vec{m}$

```
1: $\vec{u} \leftarrow \vec{c}_1\vec{s} \in R_q$
2: $\widetilde{\vec{m}} \leftarrow \vec{c}_2 - (\vec{u})_{l_v} \in \mathbb{Z}_q^{l_v}$
3: for $i = 0$ to $l_v - 1$ do
4:     if $\frac{q}{4} \leq \widetilde{\vec{m}}_i < \frac{3q}{4}$ then
5:         $\widehat{\vec{m}}_i \leftarrow 1$
6:     else
7:         $\widehat{\vec{m}}_i \leftarrow 0$
8:     end if
9: end for
10: $\vec{m} \leftarrow \mathrm{ECCDec}(\widehat{\vec{m}})$
11: return $\vec{m}$
```

Table I shows the main parameters of LAC, n denotes the dimensions of polynomials, q denotes the modulus, and h denotes the number of non-zero coefficients of secret polynomials and noise polynomials. For more details about the LAC scheme, we refer to [3].

Categories	n	q	h
LAC-128	512	251	256
LAC-192	1024	251	256
LAC-256	1024	251	384

Table I: Main parameters of LAC

B. Polynomial Multiplication

Polynomial multiplication can be done in many ways. A straightforward method is schoolbook multiplication, whose time complexity is $O(n^2)$. This method multiplies each coefficient of one polynomial with each coefficient of the other polynomial, then merges terms of the same degree to get the result. Another general method is Karatsuba & Toom-Cook multiplication, which splits each polynomial into several lower-degree polynomials and reduces the time complexity down to $O(n^\epsilon)$ with $\epsilon \in (1, 2)$. The fastest method is Number Theory Transform (NTT), but it requires that the modulus q and order n must satisfy $q \equiv 1 \pmod{2n}$. NTT is adopted by most of the lattice-based cryptosystems, such as NewHope [9] and Kyber [10]. Although NTT only has a lower time complexity of $O(n \log n)$, it cannot be used in LAC due to its particular parameters. SPM is adopted by LAC, which is described in Algorithm 4. The advantage of SPM in LAC is that it has no multiplication instructions and can have a high degree of parallelism.

C. Modular reduction

There are three commonly used reduction algorithms for guaranteeing the result in [0,q). A straightforward method

Algorithm 4 Sparse Polynomial Multiplication (SPM)

Require: A dense polynomial: $\vec{a} = a_0 + \cdots + a_{n-1}x^{n-1}, a_i \in \mathbb{Z}_q$. A sparse polynomial: $\vec{r} = r_0 + \cdots + r_{n-1}x^{n-1}, r_i \in \{-1, 0, 1\}$. A position polynomial: $\vec{r'} = r'_0 + \cdots + r'_{h-1}x^{h-1}, r'_i \in \mathbb{Z}_q$ where $\{r'_0, \cdots, r'_{h/2-1}\}/\{r'_{h/2}, \cdots, r'_{h-1}\}$ represent the index of 1/-1 in the sparse polynomial $\vec{r}$.
Ensure: $\vec{b} = \vec{a} \cdot \vec{r}$

```
1: Initialize all the coefficients of $\vec{b}$ to 0
2: for $i = 0$ to $h - 1$ do                                    ▷ outer loop
3:     $pos = \vec{r'}[i]$
4:     for $j = 0$ to $n - 1$ do                                ▷ inner loop
5:         if $(j \geq pos \& i < h/2) \| (j < pos \& i \geq h/2)$ then
6:             $\vec{b}[i] += \vec{a}[(j - pos + n)\ mod\ n]$
7:         else
8:             $\vec{b}[i] -= \vec{a}[(j - pos + n)\ mod\ n]$
9:         end if
10:     end for
11: end for
12: return $\vec{b}$
```

is as follows:

$$s = s - \lfloor \frac{s}{q} \rfloor \cdot q \tag{1}$$

where a division instruction is used. However, division instructions are the most expensive instructions, whether in software or hardware. Hence this method is not suitable for our accelerator. The core idea of Barrett reduction, Montgomery reduction, and SAMS2 [7] is to replace division by other instructions. Barrett reduction uses multiplication and shift instruction to replace division. Montgomery reduction only uses multiplication to replace division, but it needs to transform from/to the Montgomery domain. In SAMS2, the division instruction is replaced by shift, addition, and multiplication.

D. Related Work

There are many works related to the hardware implementation for other NIST Round 2 candidates.

Wang et al. [8] designed parameterized hardware accelerators for qTESLA [11]. They also implemented the SPM hardware accelerator, but their SPM is slightly different from ours. Nevertheless, their design trades parameterization and versatility at the cost of performance, so their performance can still be improved.

Zhang et al. [12] proposed a low-complexity NTT and inverse NTT (INTT) hardware architecture for NewHope and an efficient modular reduction for modulus 12289. Their modular reduction cleverly avoids multiplication instruction by utilizing the feature $2^{14} \equiv 2^{12} - 1 \pmod{12289}$. Their method is valuable for our modular reduction design.

Akleylek et al. [13] implemented a low-complexity SPM accelerator on FPGA. The SPM they studied has an extraordinary form because the coefficients of the two polynomials are in the range {-1,0,1}. In other words, both polynomials are sparse in their design. However, there is only one sparse polynomial in LAC. So their work is not compatible with the LAC scheme.

Roy et al. [14] present an instruction set coprocessor architecture for Saber. They proposed a parallel Schoolbook polynomial multiplication architecture that overcomes memory access bottlenecks. They store the entire secret polynomial $s(x)$ in a shift register composing of Flip-Flops, so their architecture can access all the coefficients of $s(x)$ simultaneously.

Mera et al. [15] implemented a better trade-off between time and memory for Toom-Cook. They proposed two novel techniques that can improve the efficiency of Toom-Cook based polynomials. Furthermore, they applied their optimization implementation of Toom-Cook to Saber. Applying Toom-Cook to the LAC scheme for performance discussion is a work we may consider in the future.

Besides, there are many works related to the hardware implementation of NewHope scheme, such as [16–18]. The hardware/software co-design is a research hotspot recently, such as SIKE [19] and NTRU [20]. NTT plays a pivotal role in lattice-based cryptography. Therefore, many works have been done for NTT by [12, 21, 22]. As far as we know, there is currently no hardware accelerator specially designed for LAC.

III. Hardware Accelerator Design

A. Overall Architecture

Overview of the Architecture: The overall architecture of our accelerator is designed and presented in Figure 1, and our design is pipelined and scalable. The proposed architecture consists of three memory blocks, including a position memory ***ram_pos***, p dense polynomial coefficient memories ***ram_poly***, an intermediate value and final result memory ***ram_res***, three data processing units, and a controller unit. Inspired by [8], we also use parameter $p(= 2, 4, 8, 16)$ to determine the number of dense polynomial coefficients computed in parallel. The management of pipeline and control signals is assigned to ***Controller Unit***, which is also responsible for maintaining the inner status of the accelerator and updating the read/write addresses of ***ram_pos*** and ***ram_res***.

Execution Flow: The execution flow of our accelerator is as follows.

1) Once the accelerator is started, ***controller unit*** will prepare the read address of ***ram_pos***, and then issue a read request to ***ram_pos***.
2) The outputs of ***ram_pos***, including the position information of sparse polynomial coefficients, are sent to ***Address Calculating Unit***.
3) The outputs of ***Address Calculating Unit*** contain p read addresses of ***ram_poly***, then p read requests to ***ram_poly*** are issued in parallel.
4) The ***Accumulating and Reduction Unit*** accumulates p outputs from ***ram_poly*** and the intermediate value from ***ram_res***. Then our new modular reduction algorithm is used to correct each coefficients of the result to $[0, q)$, and the corrected result is sent to ***ram_res***.
5) The read addresses of ***ram_poly*** are updated by ***controller unit*** and then return to Step 3. Steps 3 to 5 are equivalent to the inner loop of Algorithm 4 and the loop repeats until all the memory contents of ***ram_res*** are updated.
6) After the inner loop, the ***controller unit*** updates the read addresses of ***ram_pos*** and then return to step 1. Steps 1 and 6 are equal to the *for* statement of the outer loop of Algorithm 4. The counter of the outer loop is $\lceil \frac{h}{p} \rceil$.

The choice of Block RAM and Distributed RAM: There are two different kinds of RAM in FPGA, Block RAM (BRAM) and Distributed RAM. The former is dedicated, but the configuration of width and depth is limited. For example, the total capacity of BRAM36/BRAM18 is 36/18 Kbit, and the maximum width of them is 36/18 bits in dual-port mode. The latter consists of Look Up Tables (LUTs), the most basic logic element in FPGA. Ideally, as long as there are enough LUTs, we can implement a Distributed RAM of any width and depth. As shown in Table II, in order not to overuse BRAM while getting p positions from ***ram_pos*** per cycle, we implement ***ram_pos*** as Distributed RAM. In this way, the width limitations of BRAM are avoided cleverly. In contrast, ***ram_poly*** and ***ram_res*** are implemented as BRAM with 16-bit width, for taking full advantage of BRAM's bandwidth.

RAM	Type	Width (Bit)	Depth	Mode
ram_pos	DRAM	$p \cdot \log_2(n)$	h/p	Single-Port
ram_poly	BRAM16	16	n	Dual-Port
ram_res	BRAM16	16	$n/2$	Dual-Port

Table II: The choice of Block RAM and Distributed RAM

B. Parallel Design

We proposed the Dual-For-Loop-Parallel (DFLP) technique to optimize our parallel design, which elegantly combines the small modulus advantage of LAC and the bandwidth characteristic of BRAM on our hardware platform.

Outer Loop Parallel: The 2th and 3th lines of Algorithm 4 determine the read request of ***ram_pos***, and only one position can be read at a time without parallel design. As shown in Figure 2a, we can get p positions once at a time in our parallel design. Accordingly, the outer loop corresponding to the 2nd line of Algorithm 4 is accelerated by p times, and the loop control variable i is

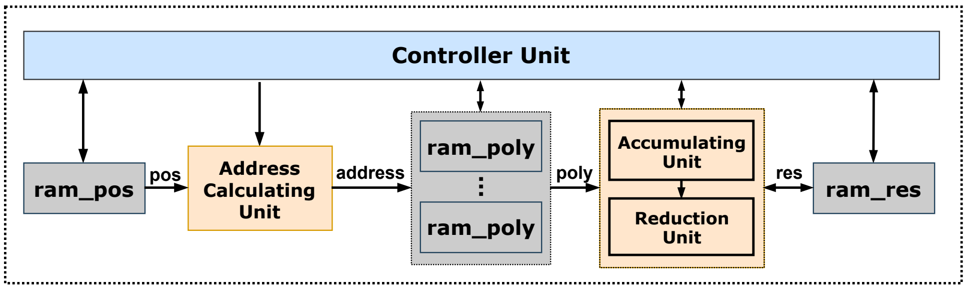
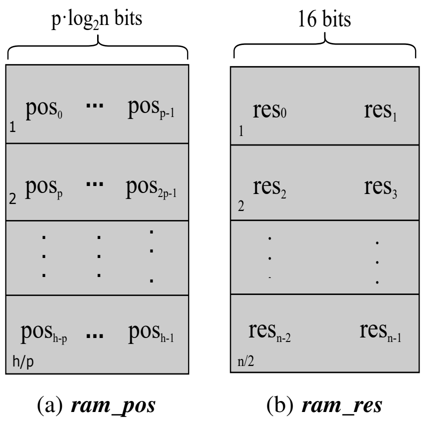


Figure 1: SPM Accelerator Architecture

incremented by p instead of 1 each cycle. In order to meet the aforementioned parallelism requirements, we implement ***ram_pos*** as Distributed RAM with $(p \cdot \log_2 n)$-bit width, which is much larger than the maximum width of Block RAM.



(a) ***ram_pos*** (b) ***ram_res***

Figure 2: The Structure of ***ram_pos*** and ***ram_res***

The Structure of **ram_poly***:* As described in Figure 3, the structure of ***ram_poly*** is special, and there are p/2 identical copies. For each ***ram_poly***, each coefficient appears twice, for example, a_1 appears in the first and second rows. The reason is that, in order to meet the aforementioned parallelism, we hope to get two coefficients from each port of ***ram_poly*** at a time, but the read address is random. If the structure of ***ram_poly*** is similar to that of ***ram_res***, then the index at the beginning of each row of ***ram_poly*** is an even number. When the read address is an odd number, then we can only get one coefficient from one port at a time. We have to admit that there is 50% data redundancy here, but it meets our requirements for the parallel design of the inner loop.

Inner Loop Parallel: The modulus of LAC is 251, so a polynomial coefficient can be stored in 8 bits. As described in Section III-A, the maximum width of BRAM16 is 16 bits in dual-port mode. This means that 32 bits can be read from BRAM16 at a time, which can accommodate four polynomial coefficients of LAC. Similarly, as depicted in Figure 2b, ***ram_res*** is also instantiated as a dual-port BRAM16 with a 16-bit width. However, one port is used for reading, while another port is dedicated to writing. Both of them can operate two polynomial coefficients at the same time. Considering that ***ram_poly*** has $p/2$ identical copies, as shown in Figure 3, $2p$ coefficients can be read from dual ports at a time. But only p coefficients can be obtained at a time without inner loop parallel design, so the inner loop is accelerated by 2 times.

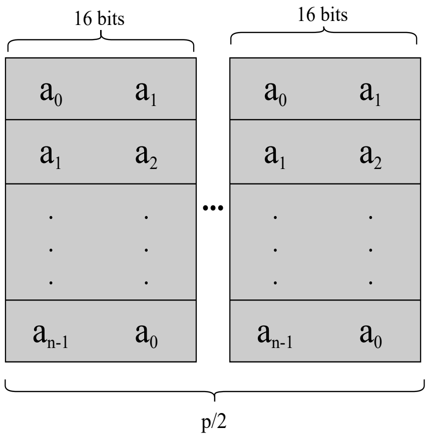


Figure 3: The Structure of ***ram_poly***

In summary, we parallelized the outer loop by using the bandwidth advantage of Distributed RAM and designed the parallel strategy for the inner loop by thoroughly analyzing and utilizing the bandwidth characteristics of BRAM. This is the core idea of the DFLP technique.

C. Modular Reduction for LAC

Barrett, Montgomery, and SAMS2[7] are modular reduction algorithms commonly used in software implementations. Nevertheless, all three require multiplication instruction, which is expensive for hardware. A modular reduction without multiplication was proposed in [12] for $q = 12289$, which is not compatible with LAC.

For the convenience of presentation, similar to Verilog hardware description language syntax, we use $c[MSB : LSB]$ to represent $(MSB - LSB + 1)$ bits data. In our architecture, p 8-bit coefficients are added and stored in $8+\log_2 p$ bits. Take $p=16$ as an example, the length of c is 12 bits, which is used to store the accumulation result of p coefficients. In order to avoid expensive multiplication instruction, the feature $2^8 \equiv 2^2+1 \pmod{251}$ is used, so c can be expressed as:

$$\begin{aligned} c &\equiv 2^8 c[11:8] + c[7:0] \\ &\equiv 2^2 c[11:8] + c[11:8] + c[7:0] \end{aligned} \tag{2}$$

where the largest value of $c[11:8]$ is 15, the largest value of $c[7:0]$ is 255, so the largest value of c is $(15 << 2 + 15 + 255) = 330$ which can be expressed with 9 bits. Finally, an additional comparison and subtraction are needed for correcting the result to $[0, q)$.

Similarly, for $p=2$ only a comparison and subtraction are needed, for $p=4,8$ we can get the following results respectively:

$$\begin{aligned} c &\equiv 2^8 c[9:8] + c[7:0] \\ &\equiv 2^2 c[9:8] + c[9:8] + c[7:0] \end{aligned} \tag{3}$$

$$\begin{aligned} c &\equiv 2^8 c[10:8] + c[7:0] \\ &\equiv 2^2 c[10:8] + c[10:8] + c[7:0] \end{aligned} \tag{4}$$

And an additional comparison and subtraction for correcting the result to [0,q) is still needed.

Figure 4 shows the hardware architecture of this modular reduction algorithm, which is designed with a two-stage pipeline. At the first stage, three additions are calculated, and the output is sent to the second stage. Thereafter, comparison and subtraction are executed to obtain the result in the second stage.

A straightforward reduction method is used in [8], and its design is excellent in terms of parameterization and versatility. For example, 4 polynomial coefficients are accumulated into c ranging from 0 to $4q$, their reduction process is as follows:

1) $if(c \geq 2q)\ c = c - 2q$
2) $if(c \geq q)\ c = c - q$

This modular reduction method is completed by $\log_2 p$ comparison and subtraction, which is obviously sub-optimal from the view of efficiency. Compared with this straightforward method, our design can save hardware resources, improve performance, and achieve a higher frequency.

D. Optimization of Pipeline Design

The maximum frequency of hardware design depends on the pipeline design and length of critical paths. The shorter the critical paths are, the higher frequency it can achieve. We analyze three critical paths of SPM hardware design in [8] and optimize their pipeline design.

Firstly, their straightforward modular reduction method is sub-optimal. For $p=8$, three cycles are required to complete the reduction. More specifically, during the first/second/third cycle, they correct the result from $[0, 8q)$/$[0, 4q)$/$[0, 2q)$ to $[0, 4q)$/$[0, 2q)$/$[0, q)$. This process requires a 11/10/9-bit width comparison and subtraction. Our reduction algorithm has some minor differences when $q=2,4,8,16$, and only two cycles are needed. In the first cycle, a 9-bit addition is performed. In the second cycle, a 9-bit comparison and 9-bit subtraction are performed. Therefore, our design's critical path length is shorter, and our design is more efficient than the straightforward method on account of fewer cycle counts.

Secondly, we notice that the Step 4 of *Execution Flow* in Section III-A can also be optimized. Specifically, the addition and reduction of intermediate result and accumulation result are calculated in one cycle, as described below:

$$\begin{aligned} assign\ sum = &((accu\_res + inter\_res) \geq q)? \\ &(accu\_res + inter\_res - q) : \\ &(accu\_res + inter\_res); \end{aligned} \tag{5}$$

where $accu\_res$ is the result of p polynomial coefficients accumulation and reduction, and $inter\_res$ is read from ***ram_res*** as an intermediate result of the whole calculation process. This method generates a relatively long path, resulting in a lower frequency. Our optimization strategy is to split the above process into two cycles. The addition is calculated in the first cycle, and reduction is performed in the second cycle. The path length can be roughly halved in this way.

Finally, the synthesis tool prompted us to insert registers at the output of BRAM, which can improve the frequency of the design. The reason is that BRAM is distributed in a specific area in FPGA, so the connection between BRAM and logic circuit is relatively long. Inserting register can cut off the long path and achieve a higher frequency.

In summary, our optimization can improve the frequency by up to 34%. For example, for $p=2/4$ and the LAC-128 scheme, the frequency increases from 196MHz to 263MHz.

IV. Performance Evaluation and Comparison

In this section, we present the performance and comparison of the SPM accelerator we designed in Table III. This table only shows the results of LAC128, because there are similar performance improvements for LAC192 and LAC256. Our design is described by Verilog hardware description language, synthesized on the Zynq-7000 platform (xc7z020clg400-2) using Vivado 2019.2 with "PerfOptimized" option. To make a fair comparison, we instantiated the hardware design of SPM in [8] with the parameters of LAC and synthesized it on our platform.

The design of Wang et al. [8] is parametric, which gives up some performance optimizations of specific schemes for the sake of compatibility. For example, their straightforward modular reduction is sub-optimal in terms of performance but is compatible with most cryptosystems. The reduction from $[0, 2q)$ to $[0, q)$ is achieved by subtracting q directly if the value is greater than q, and the bottleneck is comparison

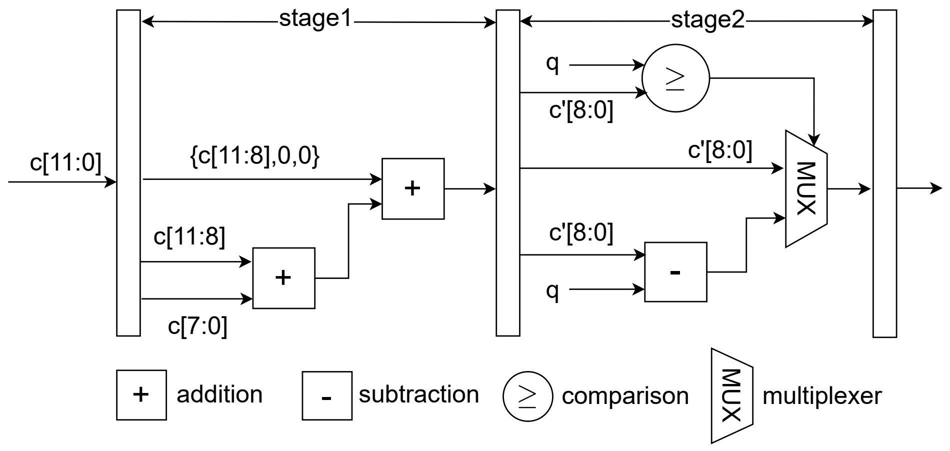


Figure 4: Modular Reduction Architecture

Design	p	Devices	LUTs/FFs/ BRAM18	Freq MHz	Cycles
Our work	2	xc7z020	328/230/2	263	34048
Our work	4	xc7z020	462/297/3	263	17024
Our work	8	xc7z020	783/432/5	202	8512
Our work	16	xc7z020	1407/704/9	157	4256
Wang el al.[8]	2	xc7z020	364/120/2	196	66432
Wang el al.[8]	4	xc7z020	476/163/3	196	33280
Wang el al.[8]	8	xc7z020	699/241/7	196	16672
Wang el al.[8]	16	xc7z020	1114/384/13	155	8352

Table III: Performance and Comparison of LAC128

operation. In this way, it takes $\log_2 8 = 3$ cycles to correct a value from $[0, 8q)$ to $[0, q)$. Besides, their pipeline design remains much scope for implementation.

As shown in Table III, our implementation is nearly two times faster than the design proposed in [8] with the same parameters and platform. The cycle counts in the table do not include the time of loading data to RAM. The cycle counts of us and Wang et al. [8] can be calculated by the following formulas respectively.

$$h/p * (N/2 + 10) \tag{6}$$

$$h/p * (N + 6 + \log_2 p) \tag{7}$$

h/p is used to represent the count of outer loop which corresponds to the 2th line in the Algorithm 4, and obviously, both of us design p-way parallel strategy for the outer loop. Our implementation also designs 2-way parallel strategy for the inner loop using the DFLP technique, so the count of the inner loop is $N/2$ instead of N. 10 and $6 + \log_2 p$ are the cycles consumed by the pipeline initialization, where 2 out of 10 cycles are used to perform our new modular reduction, and $\log_2 p$ cycles are consumed by their straightforward modular reduction. Our pipeline initialization cycles are longer than their design because we designed a deeper pipeline for higher frequency. A combination of the proposed DFLP technique and the optimization of pipeline design doubles the number of Flip-Flops (FFs) consumed by our design. Specifically, our design consumes 230 FFs, while the design of Wang et al. consumes only 120 when $p = 2$. The new modular reduction we proposed reduces 36, 50, 84, and 293 LUTs for $p = 2, 4, 8, 16$ respectively. Our design frequency increases by 34% for $p = 2, 4$ with the optimization of pipeline design and the utilization of new modular reduction. We thoroughly considered the width of BRAM18 and the requirements of our design and matched them correctly, so our design reduced 2, 4 BRAM18 for $p = 8, 16$, respectively.

V. Conclusion

This paper proposed several optimizations for the design of accelerator for SPM. First of all, the DFLP technique utilizes the bandwidth advantage of BRAM18 to improve the efficiency of our design at a small cost of resources. Secondly, our new modular reduction improves efficiency and reduces the resource overhead of the LUTs. A flatter pipeline was designed by careful analysis and optimization of the critical paths, achieving a higher frequency. A combination of these optimizations yields an efficient and scalable SPM accelerator that is twice as fast as the previous best implementation in the literature. Also, our design can be scaled according to the parameter $p(= 2, 4, 8, 16)$, the design with a small p is suitable for resource-constrained equipment, and the design with a large p can be applied to performance-first scenarios. Although this paper focuses on the implementation of SPM for LAC, we finally remark that the proposed new modular reduction is applicable to other lattice-based cryptosystems with similar mathematical derivations. Our optimizations of pipeline design are also useful for other works.

Acknowledgment

Zhe Liu is supported by the National Natural Science Foundation of China (Grant No.61802180), the

Natural Science Foundation of Jiangsu Province (Grant No.BK20180421), the National Cryptography Development Fund (Grant No.MMJJ20180105) and the Fundamental Research Funds for the Central Universities (Grant No.NE2018106).

References

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

[2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *EUROCRYPT 2010*, p. 1.

[3] X. Lu, Y. Liu, Z. Zhang, D. Jia, H. Xue, J. He, B. Li, K. Wang, Z. Liu, and H. Yang, "Lac: Practical ring-lwe based public-key encryption with byte-level modulus." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 1009, 2018.

[4] D. Moody, "The 2nd round of the nist pqc standardization process," National Institute of Standards and Technology, Tech. Rep., 2019.

[5] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 311–323.

[6] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[7] Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede, "Efficient ring-lwe encryption on 8-bit avr processors," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 663–682.

[8] W. Wang, S. Tian, B. Jungk, N. Bindel, P. Longa, and J. Szefer, "Parameterized hardware accelerators for lattice-based cryptography and their application to the HW/SW co-design of qtesla," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 269–306, 2020.

[9] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange - A new hope," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 327–343.

[10] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, "CRYSTALS - kyber: a cca-secure module-lattice-based KEM," *IACR Cryptology ePrint Archive*, vol. 2017, p. 634, 2017.

[11] F. Gérard and M. Rossi, "An efficient and provable masked implementation of qtesla," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 606, 2019.

[12] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of newhope-nist on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 49–72, 2020.

[13] S. Akleylek, E. Alkim, and Z. Y. Tok, "Sparse polynomial multiplication for lattice-based cryptography with small complexity," *J. Supercomput.*, vol. 72, no. 2, pp. 438–450, 2016.

[14] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020.

[15] J. M. B. Mera, A. Karmakar, and I. Verbauwhede, "Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 222–244, 2020.

[16] T. Oder and T. Güneysu, "Implementing the newhope-simple key exchange on low-cost fpgas," vol. 11368, pp. 128–142, 2017.

[17] Y. Xing and S. Li, "An efficient implementation of the newhope key exchange on fpgas," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 67-I, no. 3, pp. 866–878, 2020.

[18] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 4, pp. 17–61, 2019.

[19] P. M. C. Massolino, P. Longa, J. Renes, and L. Batina, "A compact and scalable hardware/software co-design of SIKE," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 245–271, 2020.

[20] T. Fritzmann, T. Schamberger, C. Frisch, K. Braun, G. Maringer, and J. Sepúlveda, "Efficient hardware/software co-design for NTRU," in *IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2018*, vol. 561. Springer, 2018, pp. 257–280.

[21] H. Nejatollahi, S. Gupta, M. Imani, T. S. Rosing, R. Cammarota, and N. D. Dutt, "Cryptopim: In-memory acceleration for lattice-based cryptographic hardware," *Design Automation Conference*, 2020.

[22] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2012, pp. 139–158.