SN Computer Science (2024) 5:212
https://doi.org/10.1007/s42979-023-02547-w

ORIGINAL RESEARCH

Performance and Efficiency Exploration of Hardware Polynomial Multipliers for Post-Quantum Lattice-Based Cryptosystems

Francesco Antognazza[1] · Alessandro Barenghi[1] · Gerardo Pelosi[1] · Ruggero Susella[2]

Received: 8 June 2023 / Accepted: 7 December 2023

Abstract

The significant effort in the research and design of large-scale quantum computers has spurred a transition to post-quantum cryptographic primitives worldwide. The post-quantum cryptographic primitive standardization effort led by the US NIST has recently selected the asymmetric encryption primitive Kyber as its candidate for standardization and indicated NTRU, as a valid alternative if intellectual property issues are not solved. Finally, a more conservative alternative to NTRU, NTRUPrime was also considered as an alternate candidate, due to its design choices that remove the possibility for a large set of attacks preemptively. All the aforementioned asymmetric primitives provide good performances, and are prime choices to provide IoT devices with post-quantum confidentiality services. In this work, we present a comprehensive exploration of hardware designs for the computation of polynomial multiplications, the workhorse operation in all the aforementioned cryptosystems, with a thorough analysis of performance, compactness and efficiency. The presented designs cope with the differences in the arithmetics of polynomial rings employed by distinct cryptosystems, benefiting from configurations and optimizations that are applicable at synthesis time and/or run time. In this context, we target a use case scenario where long-term key pairs are used, such as the ones for VPNs (e.g., over IPSec), secure shell protocols and instant messaging applications. Our high-performance design variants exhibit figures of latency comparable to the ones needed for the execution of the symmetric cryptographic primitives also included in the Post-Quantum schemes. Notably, the performance figures of the designs proposed for NTRU and NTRU Prime surpass the ones described in the related literature.

Keywords Hardware security · Post-quantum · Lattice-based cryptosystems · Key encapsulation method

Introduction

Public-key cryptography (PKC) plays a fundamental role in today's technology providing the properties of confidentiality, data and origin authentication, and non-repudiability; indeed, its diffusion is witnessed by the number of widely used communication protocols that rely on it, such as the IETF Transport Layer Security (TLS) and IP Security (IPsec) Internet standard protocols. PKC primitives are in wide use to encrypt data between two parties (without a pre-shared secret) over an insecure channel, or to build a Public Key Infrastructure, and to guarantee the integrity and authenticity of data in form of digital signatures.

Currently, the most used algorithms, RSA and Elliptic Curve cryptography, rely on the hardness of integer factoring and the hardness of computing discrete logarithm in finite cyclic groups, respectively. However, in 1994, Peter Shor designed an algorithm for quantum computers that solve both the prime factoring and discrete logarithm

This article is part of the topical collection "Recent Trends on Information Systems Security and Privacy" guest edited by Steven Furnell and Paolo Mori.

✉ Alessandro Barenghi
alessandro.barenghi@polimi.it

Francesco Antognazza
francesco.antognazza@polimi.it

Gerardo Pelosi
gerardo.pelosi@polimi.it

Ruggero Susella
ruggero.susella@st.com

[1] Department of Electronics, Information and Bioengineering - DEIB, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milan, Italy

[2] Department of Electronics Information Technology and Bioengineering, STMicroelectronics S.r.l, Via Paracelso, 20, Agrate Brianza (MB) 20864, Italy

Published online: 20 January 2024

problem with an exponential speedup with respect to classical computers, effectively breaking the corresponding cryptosystems.

Due to the long-term confidentiality and data/origin authentication guarantees required from asymmetric cryptographic primitives, and in sight of the recent advancements in the implementation of quantum computers, a significant effort in standardizing quantum-resistant algorithms for public-key cryptography is required. For that reason, the National Institute of Standards and Technology (NIST) in 2016 started the Post-Quantum Cryptography (PQC) standardization process to assess viable candidates for both Public Key Encryption (PKE) functionalities, in form of Key Encapsulation Mechanisms (KEMs), and digital signatures. The process refined its 69 candidate algorithms, reducing them to a single KEM and three digital signatures for immediate standardization at the end of the third round [1]. Furthermore, NIST provided a list of candidates that are still under investigation as alternates, as they rely on different computationally hard problems. Arguably, the the most successful class of algorithms of this standardization process is the one of lattice-based algorithms, being attractive in terms of computational latency and with practically acceptable key and ciphertext sizes.

Besides the candidate selected for immediate standardization, Kyber [2], three other schemes were deemed particularly interesting in the contest: NTRU [3], NTRU Prime [4] and Saber [5]. NTRU was officially recommended as the fallback alternative in case patent issues cannot be solved by the end of 2023 [6]. As a further testimony of NTRU's security and efficiency, Google LLC adopted it as the key encapsulation method of choice in its internal infrastructure [7, 8]. NTRU Prime is an NTRU variant with conservative choices in the underlying algebraic structure, which prevent a number of attacks preemptively. Thanks to its conservative design choices, it has been adopted, and employed by default in hybrid mode by OpenSSH [9], the most widely diffused implementation of the Secure SHell (SSH) protocol suite. Saber [5] is based on a slightly different algebraic problem with respect to Kyber (i.e., the Module Ring-learning with roundings problem instead of Module Ring-learning with errors problem), which is at least as computationally hard as the one of Kyber.

The four aforementioned lattice-based cryptosystems rely on the arithmetic of polynomials with integer coefficients modulo q, where q is either a power of two, or a small prime number; all considered modulo a polynomial with a low number of terms. Depending on the choices, the polynomial ring obtained in such a way, may be more or less friendly to sub-quadratic multiplication techniques. Among such techniques, the Number Theoretic Transform (NTT) is the most efficient way to perform a multiplication, provided that the maximum degree of the polynomial generating the ring is a power of two and that the ring of coefficients is modulo a prime: given an n degree polynomial, it runs in $\mathcal{O}(n \log_2(n)))$ sequential steps. By contrast, efficient versions of the schoolbook algorithm, which runs in $\mathcal{O}(n^2)$, such as the one by Comba [10], can always be applied, leading to extremely compact designs but also reduced throughput. Software and hardware implementations of the multiplication algorithms also rely on divide-et-impera techniques such as Karatsuba [11] or Toom-Cook decompositions [12]: these techniques trade off an increased design complexity and larger constants hidden in the $\mathcal{O}$ notation for a constant decrease in the complexity exponent.

An emerging hardware design approach is the one known in the literature as x-net or LFSR-based multiplier. Its underlying idea is to perform n coefficient-wise multiplications per clock cycle, resulting in a total computation time that is $\mathcal{O}(n)$. While x-net-based multipliers require a non-negligible amount of resources, their very good performance and flexibility prompted this work, in which we provide results on a unified multiplier design for Kyber, NTRU, NTRU Prime and Saber. We note that the specification of Kyber states that the private and public keys are represented in the NTT-transformed domain, in order to save NTT computations, in the encryption and decryption primitives. Doing so, obtains an advantage in computation speed, at the cost of sacrificing cryptographic agility. Indeed, devising accelerators that are able to speed up the computation of Kyber employing the key pair in the specified transport format, i.e., in the NTT-transformed domain, would result in the design not being compatible with lattice-based cryptosystems where the underlying polynomial ring is not NTT friendly, e.g., NTRU.

To this end, in our work, we consider that our unified multiplier is employed in cryptographically agile components, where the transport format of the Kyber key pairs is first converted back into the canonical domain upon key pair loading. This scenario fits appropriately all the cases where long-term key pairs are used, and cryptographic agility is desired, such as in smartcards, IPSec-based VPNs, instant messaging protocols, and the SSH transport layer protocol [13].

Contributions

Our work aims to show that it is possible to have a unified design for an hardware accelerator computing the polynomial multiplications in all polynomial rings of the four lattice-based cryptosystems: Kyber, NTRU, NTRU Prime, and Saber. The structure of such an accelerator stems from an architecture able to achieve efficiency results beyond the state of the art for NTRU-like cryptosystems. We provide efficiency results of a synthesis-time specialized accelerator for the arithmetic used by the NTRU (both NTRU HPS and NTRU HRSS variants), NTRU Prime, Saber and Kyber

cryptoschemes, namely every round-3 lattice-based KEM proposals at NIST's Post-Quantum standardization contest. Subsequently, we provide a unified design supporting all the polynomial rings, for all security levels of the KEMs, allowing cryptographic agility without the need of replacing the hardware component. We validated the correctness of the results and gathered the performance and resource figures for every parameter set specified by the latest specifications available, for an FPGA design flow. Our design does not depend on FPGA specific resources, allowing for simple portability across different FPGA manufacturers, and retargeting toward ASIC designs. We note that our design uses a sequential memory layout to store polynomial coefficients in memory, and accesses them in a single sweep; therefore, our design is eligible to be used also in a pipelined fashion, a feature not achievable with current NTT-based multipliers. This work is the result of an extension of the one presented in [14]: in particular, we studied further design trade-offs, adding the resource constrained designs, and further exploring the advantages coming from the unified multiplier architecture when a specific NIST security level is specified at design time.

Preliminaries

In this section, we provide a summary of the polynomial arithmetic for the polynomial rings employed in Kyber, NTRU, NTRU Prime and Saber. Subsequently, we provide a summary of linear time hardware modular multipliers obtained with the x-net technique for speed-oriented designs, as well as an outline of generic modular multipliers minimizing the memory accesses obtained with the Comba's strategy for size-oriented designs. In the following, we will denote polynomials of degree n with lowercase letters as $a(x) = \sum_{i=0}^{n-1} a_i x^i$.

The aforementioned cryptosystems consider the arithmetic over two quotient polynomial rings $\mathcal{R}_q$ and $\mathcal{R}_p$, which are defined as $\mathbb{Z}_q[x]/\langle p(x)\rangle$ and $\mathbb{Z}_p[x]/\langle p(x)\rangle$, respectively. The differences in the ring structures arise from the choice of the values p, q, n, and $p(x)$, of which a summary is reported in Table 1. Each cryptosystem specifies multiple parameter sets guaranteeing a security margin equivalent to the one provided by AES-128 (security level 1), AES-192 (security level 3), and AES-256 (security level 5). In particular, p is always chosen to be a small odd number between 3 and 11; q is either a small power of two (between 2^{11} and 2^{13}) or a prime number of the same order of magnitude. The latter choice yields polynomials with coefficient over a field, $\mathbb{Z}_p$, while the former choice allows a trivial modular reduction mod q, via truncation of the most significant bits.

Table 1 Summary of the features of the polynomial rings $\mathcal{R}_p = \mathbb{Z}_p[x]/\langle p(x)\rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle p(x)\rangle$ for each KEM

Cryptographic Scheme	q	p	n	$p(x)$
NTRU	2^i	3	prime	$x^n - 1$
	Values		values	
NTRU Prime	Prime	3	prime	$x^n - x - 1$
	Values		values	
Kyber	Prime	5, 7	2^i	$x^n + 1$
	Values		256	
Saber	2^i	7, 9, 11	2^i	$x^n + 1$
	Values		256	

The polynomial, $p(x)$, employed to obtain the quotient ring, gives $\mathcal{R}_q$ and $\mathcal{R}_p$ a *nega-cyclic* algebraic structure, as in Kyber and Saber, (i.e., with $p(x) = x^n + 1$) or a *cyclic* algebraic structure, as in NTRU, (i.e., with $p(x) = x^n - 1$). The latter structure name stems from the fact that, given an element $a(x) \in R_p$, computing the result of $x \cdot a(x)$ is equivalent to cyclically shifting its coefficients toward the higher degrees, by one position. Similarly, the nega-cyclic structure implies that the same cyclic shift takes place, but a sign flip of the constant term is also performed after the cyclic shift. The authors of NTRU Prime chose $x^n - x - 1$ as the polynomial modulus $p(x)$, thus obtaining polynomial field algebraic structures for $\mathcal{R}_q$ and $\mathcal{R}_p$: this removes the need to introduce further constraints on the parameters of the structure (which are indeed present in Kyber and Saber), preventing future attacks that may exploit the specific properties of finite fields. To align our notation with the one of the cipher specifications, we will consider the representatives of the residue classes modulo q or modulo p, to which the coefficients of polynomials belong, as the integers balanced around the zero element, for example between $-\lceil (q-1)/2 \rceil$ and $\lfloor (q-1)/2 \rfloor$.

Modular Polynomial Multiplication Algorithms

Modular Polynomial multiplications with large operands are extremely common in cryptographic primitives, and have seen significant efforts in their optimization. A first classification criterion is the strategy that is employed to perform the modular polynomial reduction: indeed, it is sometimes possible to interleave the reduction operation with the intermediate steps of the multiplication algorithm, saving on the memory elements required for the computation. The second classification criterion is the asymptotic complexity of the multiplication method, counted as the number of coefficient-wise multiplications, which in turn is a function of the number of coefficients of the operands, n.

As shown in Algorithm 1, the operand-scanning, schoolbook method for polynomial multiplications involves $\mathcal{O}(n^2)$

coefficient-wise multiplications as it adds together all the results of multiplying the first polynomial factor by each one of the monomials composing the second factor.

Algorithm 1 Schoolbook Multiplication Algorithm

Require: $A = (A_{n-1}, \dots A_0)$, $B = (B_{n-1}, \dots, B_0)$, arrays storing the coefficients of two polynomials $a(x) = a_{n-1}x^{n-1} + \dots + a_0$, $b(x) = b_{n-1}x^{n-1} + \dots + b_0$, with each coefficient belonging to a chosen finite ring.
Ensure: $C = (C_{2n-2}, \dots C_0)$, array storing the coefficients of $c(x) = a(x) \cdot b(x)$, where each coefficient c_i(or C_i, resp.) belongs to the chosen finite ring, $0 \leq i \leq 2n-2$.

```
for j ← 0 to n − 1 do
    for i ← 0 to n − 1 do
        C_{j+i} ← C_{j+i} + A_j · B_i        ▷ single Multiply-and-Accumulate
    end for
end for
```

Finally, it is possible to compute polynomial multiplications in $\mathcal{O}(n \log_2(n))$ exploiting Fourier transformations. The method relies on the fact that multiplying two polynomials yields the same result computed by the convolution of their coefficients, interpreted as integer sequences.

The sub-quadratic methods, pioneered by Karatsuba [11], provide algorithms to compute the polynomial multiplication in $\mathcal{O}(n^{\log_a(2a-1)})$ coefficient-wise multiplications, where $a \geq 2$. In particular, Karatsuba proposed the algorithmic variant for $a = 2$, while Toom and Cook [12] generalized the result for $a > 2$. The reason for avoiding the ubiquitous application of such methods is that, while the number of coefficient-wise application decreases, they require an increasing number of polynomial additions and subtractions to compute the result. While additions and subtractions have a linear cost in n, their overhead offsets the gains coming from saving multiplications for small values of n. Given that the ratio between the absolute values of the computational costs of multiplications and additions/subtractions varies depending on the underlying computational platform, it is commonplace to determine the break-even value for the parameter a through an exhaustive evaluation when designing a specific instance of a cryptographic scheme. In our context, Karatsuba was used in [15] instantiating three parallel Comba multipliers, while the design in [16] involved a 3-way Toom-Cook computing five parallel multiplications recursively with the Karatsuba method.

This allows to perform the multiplication computing the discrete-time Fourier transform of the sequences, performing the element-wise multiplication of the results and computing the inverse Fourier transform of such an outcome. The total cost of the operation depends on the cost of computing the Fourier transform, to which a linear amount of coefficient-wise multiplications must be added. For the special case where n is a power of two, computing the Fourier transform takes $\mathcal{O}(n \log_2(n))$, thus resulting in a $\mathcal{O}(2(n \log_2(n)) + n) = \mathcal{O}(n \log_2(n))$ cost for the entire multiplication. This technique is applied fruitfully to polynomials in a ring $\mathbb{Z}_q[x]/\langle p(x) \rangle$, provided that the degree of $p(x)$ is a power of two, and that $\mathbb{Z}_q$ is a field, (to make use of all the required roots of unity), and goes by the name of Number Theoretic Transform (NTT) [16]. As it is the case for the sub-quadratic multiplication techniques, also the NTT requires some linear time operations to be computed, and thus the break-even point for the value of n is sought experimentally. Of the four cryptosystems we are considering, only Kyber has a parameter choice that may benefit from the use of NTT-based techniques.

Algorithm 2 Parallel Schoolbook Algorithm

Require: $A = (A_{n-1}, \dots A_0)$, $B = (B_{n-1}, \dots, B_0)$, arrays storing the coefficients of two polynomials $a(x) = a_{n-1}x^{n-1} + \dots + a_0$, $b(x) = b_{n-1}x^{n-1} + \dots + b_0$, with each coefficient belonging to a chosen finite ring.
Ensure: $C = (C_{2n-2}, \dots C_0)$, array storing the coefficients of $c(x) = a(x) \cdot b(x)$, where each coefficient c_i(or C_i, resp.) belongs to the chosen finite ring, $0 \leq i \leq 2n-2$.

```
C ← (0, ..., 0) // zeroed array with 2n − 1 components
for i ← 0 to n − 1 do
    C ← C + CoeffWiseMultiplication(A, B_i)
end for
```

Linear-time Modular Multiplication

An orthogonal approach to the redesign of a multiplication algorithm is the one that exploits the inherent parallelism of the schoolbook strategy. Indeed, all the coefficient-wise multiplications involved in a product of a single monomial coefficient by the entire other factor, can be computed independently. This observation leads to the design of a linear time multiplication algorithm that exploits n computation units and n coefficient-wide memories to compute the entire product in $\mathcal{O}(n)$ following the operative pattern shown in Algorithm 2.

The first proposal of a linear time modular multiplication algorithm specialized for the NTRUEncrypt polynomial ring comes from [17]. The work achieves the multiplication in n clock cycles using n parallel multiply-and-accumulate (MAC) units. Furthermore, to reduce the area of each MAC unit, the work replaces the multiplier with a multiplexer, which selects one of the three possible coefficient-wise multiplication outcomes, thanks to the small size of the coefficients of the $\mathcal{R}_p$ operand, with $p=3$. This approach was then separately adapted for the realization of the arithmetic of the different polynomial rings of Saber, NTRU, and NTRU Prime cryptoschemes [16, 18, 19]. The authors of [18] proposed a centralized way to compute the few possible results of a coefficient-wise multiplication, and distribute them to every MAC unit. In [20], the authors proposed to postpone the reduction mod q of the coefficients of the multiplication result to the end of the multiplication. This approach entails larger accumulators to store the coefficients of the resulting polynomial, while allowing to save area as only a single modular reduction unit is required.

Resource Constrained Modular Multiplication

Employing a schoolbook multiplier to perform the polynomial modular multiplication in an operand-scanning fashion results in the smallest amount of computational and memory resources. Indeed, in a software-based implementation, the operand-scanning approach has the minimum code size and register pressure, while in a hardware implementation the compact design of the multiplier and the simple control logic, which in turn reduces the size of the driver Finite State Machine (FSM), yield area savings. A notable alternative to the plain schoolbook multiplier design is represented by the Comba's method [10] reported in Algorithm 3. Such a strategy can be seen an optimization of the schoolbook algorithm, aiming at minimizing the number of memory accesses to compute each coefficient of the result, even if it still involves $\mathcal{O}(n^2)$ coefficient-wise multiplications. This optimization has the additional benefit of requiring a minimal amount of computational resources, in the same fashion as the schoolbook method. The intuition in Comba's method is to reorder the single-coefficient multiplications with respect to the operative pattern of the schoolbook algorithm by performing additions of properly shifted intermediate polynomials (which in turn are obtained multiplying the first polynomial factor by each coefficient of the second one). Comba's method computes and writes the final value of each coefficient of the resulting polynomial into the corresponding memory space only once (proceeding from the least significant coefficient to the most significant one, or vice versa). From an implementation standpoint, such a feature is particularly advantageous because it allows to perform

Algorithm 3 Comba's Multiplication Algorithm

Require: $A = (A_{n-1}, \dots A_0)$, $B = (B_{n-1}, \dots, B_0)$, arrays storing the coefficients of two polynomials $a(x) = a_{n-1}x^{n-1} + \dots + a_0$, $b(x) = b_{n-1}x^{n-1} + \dots + b_0$, with each coefficient belonging to a chosen finite ring.
Ensure: $C = (C_{2n-2}, \dots C_0)$, array storing the coefficients of $c(x) = a(x) \cdot b(x)$, where each coefficient c_i(or C_i, resp.) belongs to the chosen finite ring, $0 \leq i \leq 2n-2$.

$C \leftarrow (0, \dots, 0)$ // zeroed array with $2n-1$ components
for rIdx $\leftarrow 0$ **to** $n-1$ **do**
  for $i \leftarrow 0$ **to** rIdx **do**
    tmp $\leftarrow$ tmp $+ A_{\texttt{rIdx}-i} \cdot B_i$ ▷ No mem write for Multiply-and-Accumulate
  end for
  $C_{\texttt{rIdx}} \leftarrow$ tmp
end for
for rIdx $\leftarrow n$ **to** $2n-1$ **do**
  for $i \leftarrow n$ **to** rIdx **do**
    tmp $\leftarrow$ tmp $+ A_{\texttt{rIdx}-(i-n)} \cdot B_{i-n}$ ▷ No mem write for Multiply-and-Accumulate
  end for
  $C_{\texttt{rIdx}} \leftarrow$ tmp
end for

only $2n-1$ read/write accesses to the memory holding the result, instead of $2n^2$ read/write accesses required by the schoolbook strategy. To achieve a modular multiplication with an operand-scanning Comba multiplier, it is sufficient to accumulate the coefficient of the result in the corresponding appropriate position: in case the computed coefficient is within the maximum degree of the polynomial ring elements, it is added as reported in Algorithm 3, lines 2–7; in case the coefficient belongs to a monomial of higher degree (lines 8–13), it is added or subtracted to a set of monomials of lower degrees, corresponding to the non-null coefficients of the polynomial ring modulus.

Our Unified Multiplier Design

In this section, we provide the description of our digital designs to implement the x-net approach and the Comba approach to polynomial multiplication, for a generic polynomial modulus $p(x)$. In particular, we employ our framework to describe the x-net multiplier design, specialize its structure for each one of the four polynomial rings required in Kyber, Saber, NTRU and NTRU Prime, and describe a unified multiplier architecture. Subsequently, we describe the Comba approach to polynomial multiplication, which can be applied to all polynomial rings needed for Kyber, Saber, NTRU and NTRU Prime, and similarly describe another unified multiplier architecture.

In the following, we consider the case of the multiplication of two polynomials where the first one has coefficients in $\mathbb{Z}_p$, while the second one has coefficients in $\mathbb{Z}_q$, with p q, and the product has coefficients in $\mathbb{Z}_q$, which is the polynomial multiplication taking place in all the cryptosystems at hand. The operation is intended to be computed lifting the coefficients of the first polynomial $\mathbb{Z}_p$ simply reconsidering their values as being in $\mathbb{Z}_q$. We note that NTRU also requires a multiplication between two polynomials with coefficients in $\mathbb{Z}_q$. The described multiplier structure also covers this case, enlarging the width of the signals carrying coefficients in $\mathbb{Z}_p$ to hold the coefficients in $\mathbb{Z}_q$. In the following, we will assume that the polynomial modulus $p(x)$ is monic, as it is always the case in practice.

Algorithm 4 x-net polynomial multiplier

Require: $A = (A_{n-1}, \dots A_0)$, $B = (B_{n-1}, \dots, B_0)$, arrays storing the coefficients of two polynomials $a(x) \in \mathbb{Z}_p[x]/\langle p(x)\rangle$, $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) \in \mathbb{Z}_q[x]/\langle p(x)\rangle$, $b(x) = \sum_{i=0}^{n-1} b_i x^i$, $p(x) \in \mathbb{Z}_q[x]$, monic, with degree n
Ensure: $C = (C_{2n-2}, \dots C_0)$, array storing the coefficients of $c(x) = \text{LIFT}\,(a(x), \mathbb{Z}_q[x]/\langle p(x)\rangle)\, b(x)$
1: $C \leftarrow (0, \dots, 0)$ ▷ All coefficients set to zero
2: **for** $i \leftarrow 0$ **to** $(n-1)$ **do**
3: $\quad C \leftarrow C + B_i \cdot A$ ▷ n parallel Multiply-and-Accumulators
4: $\quad A \leftarrow A \cdot (0, \dots, 0, 1, 0) \bmod p(x)$ ▷ $a(x) \cdot x$ performed via the LFSR structure
5: **end for**
6: **return** C

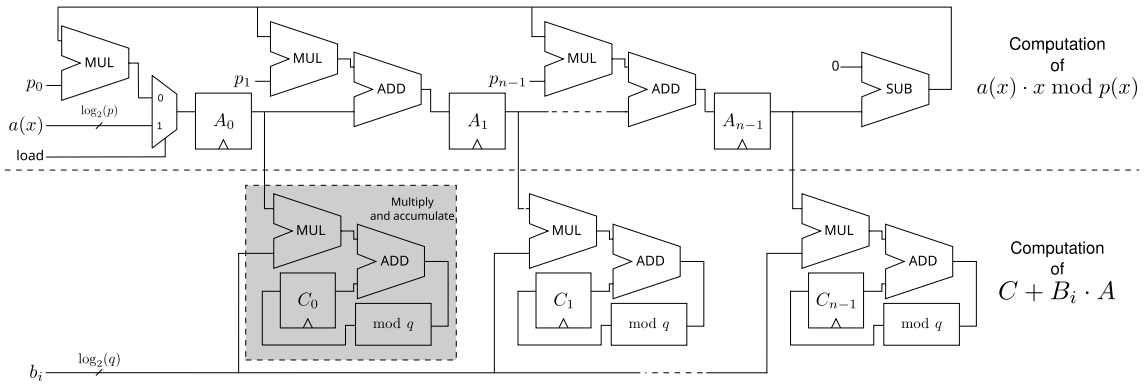


Fig. 1 Structure of an x-net multiplier computing the product $r(x) = (a(x) \cdot b(x)) \bmod p(x)$. The top portion of the modular multiplier takes care of computing $x^i \cdot a(x) \bmod p(x)$ at the i-th clock cycle, while the bottom part performs the coefficient-by-polynomial multiplication

The main idea of the x-net multiplication is to rewrite the computation of the polynomial ring multiplication, i.e., the multiplication of the between $a(x)$ and $b(x)$, and the subsequent modular reduction mod $p(x)$, as described in Algorithm 4. The modular polynomial multiplication, $a(x) \cdot b(x) = c(x) \bmod p(x)$, is decomposed as a sequence of coefficient-by-polynomial multiplications and polynomial additions involving a single coefficient of the second factor and the entire first factor (line 3), and multiplications by x followed by modular reductions of the first factor (line 4). This decomposition of the modular multiplication operation allows an efficient hardware implementation; indeed, the coefficient-wise multiplications of line 3 expose a large amount of data parallelism, while the modular multiplication by x and the subsequent reduction can be implemented by means of an LFSR structure.

The hardware structure of a generic x-net modular multiplier for a monic $p(x)$ is depicted in Fig. 1.

The coefficient-by-polynomial multiplication (line 3 in Algorithm 4) is computed with n independent Multiply and Accumulate (MAC) elements that compute the product of the coefficient b_i by each coefficient of polynomial $a(x)$, and add the result to the corresponding coefficient of $c(x)$. The corresponding portion of the circuit in Fig. 1 is the bottom half, where one MAC element is highlighted in gray. A single MAC element is composed by an integer multiplier, an adder, a modular reducer mod q, and a register containing the value of the coefficient $C_i, 0 \leq i < n$.

The computation of the multiplication of the first factor by x, $a(x) \leftarrow a(x) \cdot x$ is efficiently done by storing the coefficients of $a(x)$ in a shift register, as the multiplication by x acts shifting the coefficients by one position toward higher degree monomials (to the right, in Fig. 1). Since the degree of $a(x)$ is at most $n-1$ before the multiplication by x, the modular reduction $a(x) \leftarrow a(x) \cdot x \bmod p(x)$ can be efficiently computed. Indeed, since $p(x)$ is monic, computing the remainder of $a(x) \cdot x \bmod p(x)$ is equivalent to the subtraction from $a(x) \cdot x$ of the polynomial $A_{n-1} \cdot (p(x) - x^n)$.

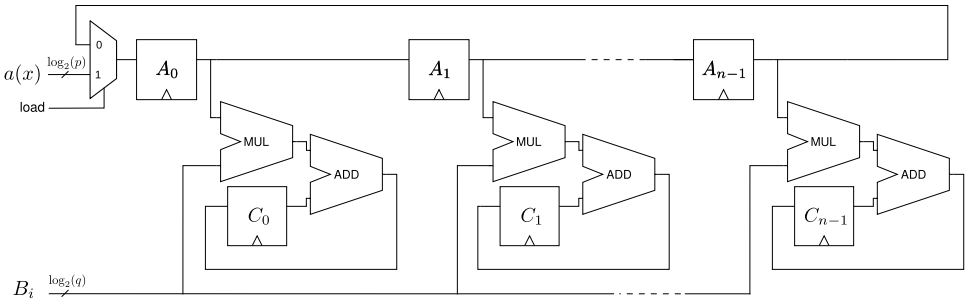
The multiplication and modular reduction are performed in the same clock cycle by the the portion of the x-net multiplier managing the operation (top portion of Fig. 1). This circuit, structured as a shift register with feedback, performs the $a(x) \cdot x$ shifting the contents of the registers containing $(A_0, \ldots A_{n-1})$ toward right. The same circuit also subtracts $A_{n-1} \cdot (p(x) - x^n)$ from $a(x) \cdot x$ by adding the coefficients of $-A_{n-1} \cdot (p(x) - x^n)$ to the ones of $a(x) \cdot x$. This is done inserting the adders on the shift lines between any two elements of the shift register that contains $a(x)$. This feedback network structure will thus need as many multipliers and adders as the number of non-null coefficients in $p(x)$, benefiting from values of $p(x)$ with a very small number of coefficients, as it is the case in the four considered cryptosystems. Finally, we note that the shift register structure also allows to perform the loading of $a(x)$ with minimal additional hardware. Indeed, $a(x)$ in our design is loaded coefficient-wise from A_{n-1} to A_0, inserting a single mux (represented on the left in Fig. 1).

Structural *x*-net Optimizations

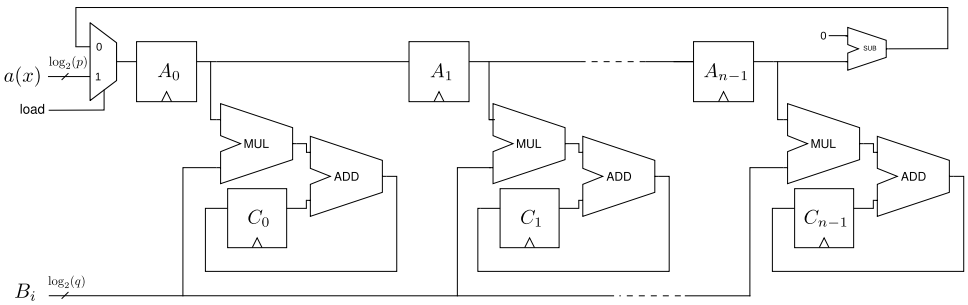
The first observation leading to an optimization is that the topmost portion of the x-net multiplier may operate entirely with values mod p, leading to a significant saving in the resource consumption for the cases where $p \ll q$. The lifting required to multiply coefficient in $\mathbb{Z}_p$ by coefficients in $\mathbb{Z}_q$ is efficiently realized within the multiplier units in the MAC elements by sign-extending the two's complement representation of the $\mathbb{Z}_p$ elements.

The second observation leading to an optimization is that, in case p is very small, as it is the case in our cryptosystems, the multiplier in the MAC can be substituted by a multiplexer that selects among a small set of fixed multiples of B_i, which are in turn computed by a small number of additions. Taking as an example $p = 5$, the multiplier is substituted by a multiplexer selecting among the values $\{-2B_i, -B_i, 0, B_i, 2B_i\}$, depending on the value of the coefficient of the $a(x)$ polynomial. The values can be either precomputed only once, and distributed, or computed within the MAC unit and selected in place. The first approach requires a larger amount of resources for each single MAC unit, while obtaining a reduction in the wiring congestion, which is particularly beneficial for FPGA targeted implementations.

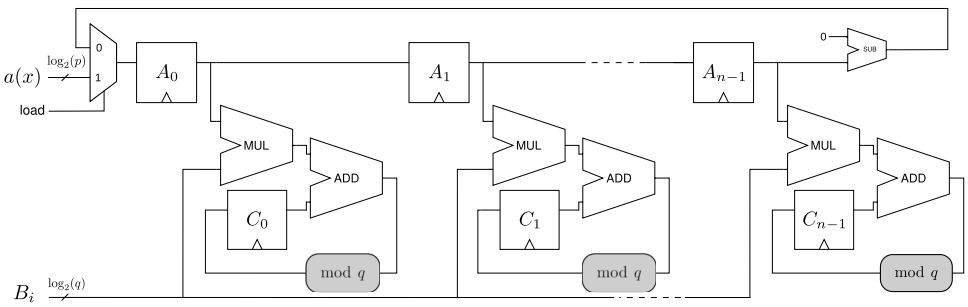
A final point concerning the optimization of the x-net multiplier is the trade-off between performing modular reductions in the MAC complex managing the coefficients of the result, and performing the reductions upon result readout. Choosing to perform the modular reductions at readout requires wider accumulator registers for $C(x)$; in particular, their size grows from $\lceil \log_2(q) \rceil$ to $\lceil \log_2(npq) \rceil$ bits, as n values mod q will be multiplied by a value mod p and added by the x-net multiplier during its operation. This increase in area is however compensated by the removal of n modular reducers mod q from each multiply and accumulate complex, enacting a trade-off that typically gains in area consumption, unless the reduction by q is trivial (e.g., when q is a power of two). We explored both strategies devising modular reducers as follows. In the former case, each accumulator register has $\log_2(q)$ bits size, and we perform the mod q operation by conditionally applying additions and subtractions. Since the distance between each integer multiplication result and a valid $\mathbb{Z}_q$ element is at most $(q-1) \cdot \lceil (p-1)/2 \rceil$, then $\lceil (p-1)/2 \rceil$ additions and subtractions are carried out in parallel with values multiple of q and the only valid result in $\mathbb{Z}_q$ is selected. In case of the reduction operation performed during the readout, a single Barrett reduction module is used.



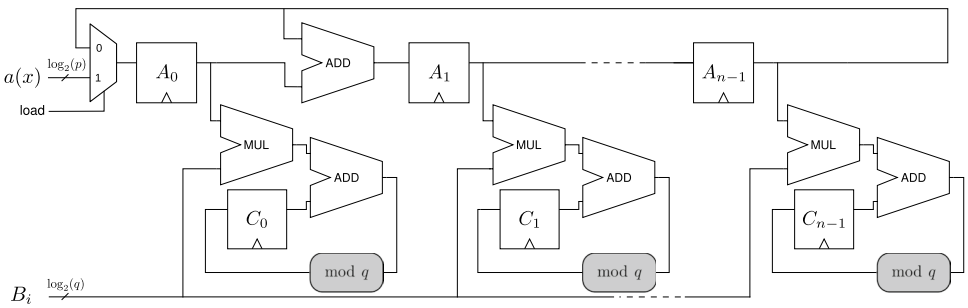
(a) x-net design tailored for NTRU polynomial ring



(b) x-net design tailored for Saber polynomial ring



(c) x-net design tailored for Kyber polynomial ring



(d) x-net design tailored for NTRU Prime polynomial ring

Fig. 2 x-net architectures specifically tailored for each polynomial ring. The readout circuit of the accumulators is omitted for clarity. The mod q reducer is not present whenever the reduction is performed upon result readout

Specialized and Unified *x*-net Designs

We now describe the specializations of the x-net designs that can be performed to optimize its resource consumption according to the specific polynomial ring for NTRU, Saber, NTRU Prime and Kyber. The specialized designs are depicted in Fig. 2. For the case of the NTRU polynomial ring, we have that the large modulus q is a power of two, while the modulus polynomial is $p(x) = x^n - 1$. This in turn allows us to perform the structural optimizations depicted in Fig. 2a: we perform implicitly the modular reduction as it is a simple bitwise truncation; therefore, no explicit reducers modulo q are present; the addition of $-A_{n-1} \cdot (p(x) - x^n)$ becomes the addition of A_{n-1} alone, since $p(x) - x^n = -1$.

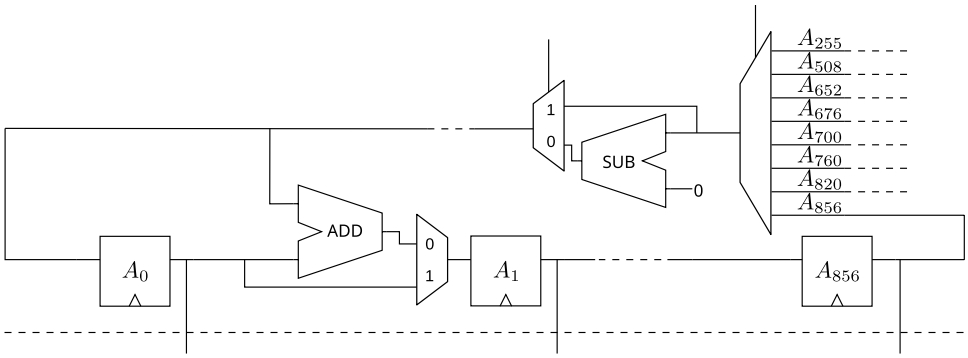


Fig. 3 Multiplexers introduced by the unified x-net multiplier

Given that this element should be added to the A_0 coefficient of the $a(x) \cdot x$ product, no adder is needed, as the product $a(x) \cdot x$ has always $A_0 = 0$. Therefore, the value of A_{n-1} is simply fed back into the first register by the feedback line on top of Fig. 2a.

For the case of Saber, depicted in Fig. 2b, the modulus q is still a power of two, therefore allowing modular reduction mod q with a simple bit truncation as for NTRU, while the polynomial modulus is $p(x) = x^n + 1$. The value of the polynomial modulus implies that adding $-A_{n-1} \cdot (p(x) - x^n)$ is equivalent to adding $-A_{n-1}$ to the null A_0 coefficient of $a(x) \cdot x$. As a consequence, a subtractor is added on the feedback line, fed with 0 as the minuend and A_{n-1} as the subtrahend.

The x-net design for Kyber, depicted in Fig. 2c, manages the fact that the large modulus q is a prime value, therefore requiring modular reduction units between the output of the MAC operation and the input of the register storing $C_i, 0 \leq i < n$. The value of the polynomial modulus for Kyber matches the one of Saber, i.e., $p(x) = x^n + 1$. As a consequence, the computation of $-A_{n-1} \cdot (p(x) - x^n)$ to be added back as a result of the modular reduction $a(x) \cdot x \bmod p(x)$ results again in $-A_{n-1}$ being added to the null A_0 coefficient of $a(x) \cdot x$.

Finally, the design of the x-net multiplier for NTRU Prime, depicted in Fig. 2d, also requires to employ a value of q which is a prime number, in turn requiring a mod q reducer for each MAC complex. The modulus value for NTRU prime is $p(x) = x^n - x - 1$, which in turn implies that adding $-A_{n-1} \cdot (p(x) - x^n)$ is equivalent to adding $-A_{n-1} \cdot (-x - 1) = A_{n-1}x + A_{n-1}$ to $a(x) \cdot x$. While adding A_{n-1} does not require an actual adder, as the A_0 coefficient of $a(x) \cdot x$ is null, adding $A_{n-1}x$ requires an actual coefficient-wise addition $A_{n-1} + A_1$, where A_1 is the coefficient of x in $a(x) \cdot x$. Therefore, the feedback network for the x-net design of NTRU Prime has an adder taking as inputs A_{n-1} and A_0 from $a(x)$, which is indeed A_1 in the $a(x) \cdot x$ product.

Providing a single unified design for all the four cryptosystems was achieved postponing the modulo reduction of resulting coefficients upon readout and considering the largest among all the register sizes required by the four designs, and inserting multiplexers regulating which multiply-add elements are active on the feedback network of the register containing $a(x)$, and whether or not the sign of A_{n-1} should be flipped. We provide a graphical depiction of the selection multiplexer to support the inverted feedback coefficient selection in Fig. 3.

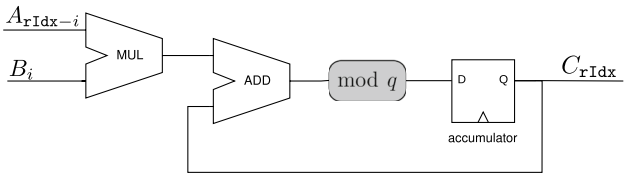


Fig. 4 Datapath of the Comba multiplier

This approach required a Barrett reduction module compatible with multiple modulus, which was achieved through storing the pre-computed constants in a small read-only memory.

Multiplying in Less Than 3*n* Cycles

In our design, we also explored the possibility of reducing the multiplication time under $3n$ cycles. Indeed, the described architecture uses n clock cycles to load the $a(x)$ from memory, n cycles to compute the result of the modular multiplication (potentially without coefficient-wise modular reduction), and n cycles to read out the final polynomial multiplication result and store it into the memory. This process can be sped up devising a memory bus transferring multiple polynomial coefficients at once. Transferring α, β and γ coefficients for respectively the small, large and result polynomials, the overall latency of a polynomial multiplication is $\lceil n/\alpha \rceil + \lceil n/\beta \rceil + \lceil n/\gamma \rceil$. Loading α coefficients of $a(x)$ for each clock cycle is achieved transferring them in parallel from main memory, and having the shift register containing a rotate by α positions at each clock cycle through appropriate connections. The same approach is applied for reading out γ coefficients of the result from the accumulator

registers, possibly instantiating γ parallel Barrett modules when performing the reductions-at-readout approach. To compute the multiplication of β $\mathbb{Z}_q$ coefficients in parallel, we need a total of $\beta \cdot n$ MACs. Indeed, to compute the result of β multiplication steps, β multiplications and sums need to be computed at each clock cycle, to obtain the result which is to be stored $\beta - 1$ cells to the right of each MAC unit. Furthermore, it is to be noted that β steps of the update of $a(x)$ should be computed in a single step. This in turn requires to perform $\beta - 1$ sign flips of the $\mathbb{Z}_p$ coefficient for specific MAC units of Kyber and Saber, and additional $2(\beta - 1)$ multiply and additions for specific MAC units of NTRU Prime.

Comba Multiplication

Realizing a Comba multiplier requires a remarkably small datapath that is only in charge of performing a single MAC operation between polynomial coefficients per clock cycle, and store the result in an accumulator register. The datapath, depicted in Fig. 4 requires, in addition to a multiplier and an adder, an additional reducer modulo q, which can be omitted in case the value of q is a power of two, as the modular reduction amounts to a simple bit truncation of the output of the MAC. This datapath allows to compute one of the iterations of the loops of Algorithm 3, lines 3–5 and lines 9–11 per clock cycle, while retaining the value of the `tmp` variable within the local accumulator. The writeback of the correctly computed value $C_{\texttt{rIdx}}$ onto the memory holding the result is thus done only once at the end of each iteration of the outer loops of Algorithm 3, lines 2–7 and lines 8–13.

Performing a regular multiplication, followed by a polynomial reduction with Comba's approach would produce a result with a maximum degree up to $2n - 1$, which would in turn double the size of the required memory to contain it. We optimized the regular Comba algorithm taking care of saving the result of a single outer loop iteration of lines 8–13 of Algorithm 3, while taking care of the effects of the modular reduction. This entails either adding or subtracting the coefficient of the monomials with degree higher than $n - 1$ in the result of the plain multiplication to the appropriate coefficient of the modular multiplication result. We note that such an approach is non-trivial to realize for the case of the NTRU Prime cryptographic scheme, as coefficients of monomials with degree higher than $n - 1$ have to be added twice, to two subsequent coefficients in the modular multiplication result. We also note that, before depositing the result of the sum/subtraction of the coefficient of the monomial with degree higher than $n - 1$ and the one present in the result accumulator, a modular reduction is required. Since we know that the maximum value admissible as the result of the accumulation is smaller than $2q - 2$, it is possible to perform the modular reduction together with the accumulation through a simple selection of the result of a short chain of adders and subtractors, even when the modulus q is not reduction friendly (i.e., a power of two).

Experimental Evaluation

In this section, we present the results of our synthesis campaign on specialized x-net multipliers for all the four cryptosystems we considered, and compare them with the current state of the art solutions. In addition, we show the results for the design based on the Comba algorithm, optimized for the NTRU, Kyber, and Saber schemes. Furthermore, we report the figures of merit for our unified multiplier designs.

The correctness of the results of our multipliers was tested through test benches obtained with a synthetic computation model written in SageMath, which generated known answer tests according to the reference implementations of the ciphers. We tested the correctness of the polynomial multiplication for every ring defined by the parameter sets of Kyber, Saber, NTRU, and NTRU Prime cryptographic schemes.

We conducted our syntheses for the Xilinx UltraScale+ ZCU106 FPGA (target `xczu7ev-ffvc1156-3-e`) using Vivado 2021.1 with Flow_AlternateRoutability and Performance_NetDelay_high strategies for synthesis and implementation, using out-of-context synthesis mode and fixing the clock source cell in order to produce a realistic timing analysis.

Considering the x-net architecture, we explored four different choices of the amount of coefficients being loaded, namely, loading either one or four $\mathcal{R}_p$ coefficients, and one or two $\mathcal{R}_q$ coefficients. Our systematic exploration led us to

Table 2 Number of $\mathcal{R}_p \times \mathcal{R}_q$ multiplications in key generation, encapsulation and decapsulation primitives of each cryptographic scheme

Cryptographic scheme	Module rank k	$\mathcal{R}_p \times \mathcal{R}_q$ Multiplications		
		Keygen.	Encap.	Decap.
NTRU	1	5	1	$2^\star$
Streamlined NTRU Prime	1	1	1	3
NTRU LPRime	1	1	2	3
Kyber	2,3,4	k^2	$k^2 + k$	$k^2 + 2k$
Saber	2,3,4	k^2	$k^2 + k$	$k^2 + 2k$

One further $\mathcal{R}_q \times \mathcal{R}_q$ multiplication is performed during the decapsulation in NTRU, denoted by $\star$ symbol. Module-based cryptographic schemes Saber and Kyber perform one $k \times k$ matrix–vector and one or two vector-vector multiplications, where the elements are built from the coefficients of polynomials in $\mathcal{R}_q$ or $\mathcal{R}_p$, during key generation, encapsulation and decapsulation, respectively

discover that the x-net configurations loading four $\mathcal{R}_p$ coefficients per clock cycle achieve better performance figures (when transferring two $\mathcal{R}_q$ coefficients) and better area–time product (when transferring one $\mathcal{R}_q$ coefficient) than their alternatives, we therefore report their results alone for the sake of brevity. We thus have that the first operand, i.e., the one on $\mathbb{Z}_p/\langle p(x)\rangle$ is loaded into the registers 4 coefficients at a time, with a data transfer of 8 to 16 bits per clock cycle depending on the cryptographic scheme and parameter set. Moreover, we report the resulting data when the modulo reduction operation is performed every clock cycle or upon readout. By contrast, the design based on Comba algorithm performs a modular reduction at each single coefficient MAC, aiming for extreme compactness and low circuit complexity. Finally, we report the results of our unified multiplier designs, able to support the NIST security levels 1, 3 and 5, which are equivalent to the security margin provided by AES-128, AES-192, and AES-256, respectively. In providing overall latency figures for the computations accelerated by our designs in the four considered cryptosystems, we take into account the number of accelerated multiplications per primitive, as reported in Table 2. We consider, in all cases, the sequential executions of the required multiplication operations to provide the latency figures. We note that the area-time product does not change if multiple parallel multipliers are instantiated whenever data parallelism is available in the scheme.

Net Performance Results

We conducted an exploration for the x-net optimized designs and gathered detailed data of the resource consumption, in terms of Cell Logic Blocks (CLBs), employed by each multiplier, the number of clock cycles taken for an entire modular multiplication, and the maximum target frequency that the design was able to reach. Furthermore, we also computed the total latency taken by all $\mathcal{R}_p \times \mathcal{R}_q$ multiplications in the key generation, key encapsulation (encryption) and key decapsulation (decryption) primitives of the scheme, as some schemes require more than a single multiplication (see Table 2). We evaluated the two coefficient ring reduction strategies described in Sect. Potential function (i.e., the one acting at each clock cycle, and the one acting upon readout) for NTRU Prime and Kyber to determine which solution is to be preferred when targeting an FPGA design. We noted that delaying the reduction to the readout phase yields an important gain in the designs of Kyber and NTRU Prime in terms of utilization of resources and increase of the working frequency, although at the cost of a moderate increase in the number of needed Flip Flops. As a consequence we selected the said reduction strategy to realize Kyber and NTRU Prime multipliers. As far the multipliers employed in Saber and NTRU scheme are concerned, as their parameter set exhibit a value of q that is a power of 2, the most convenient reduction strategy is to reduce the result of the

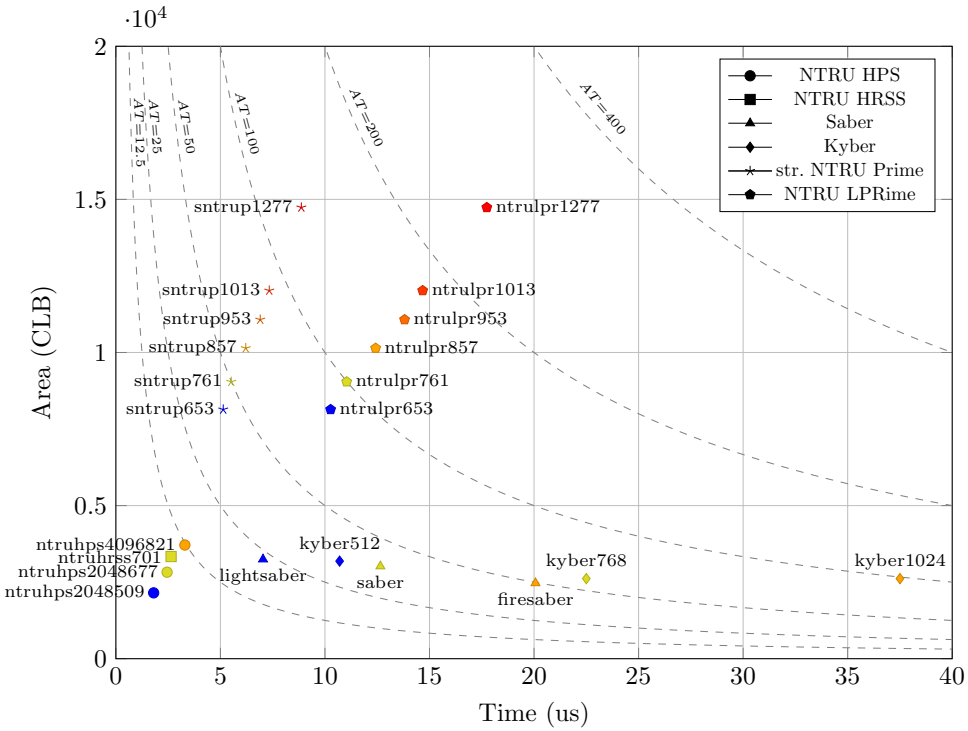


Fig. 5 Efficiency comparison of x-net-based designs. Blue, yellow and orange markers refer to parameters of security level 1, 3, and 5, respectively. Red markers denote parameters above security level 5. Dashed lines exhbit the same area-time (AT) product (lower is better)

multiplication between two polynomial coefficients at each clock cycle (see Sect. Potential function).

In the following, we employ the Area-Time (AT) product as an efficiency indicator, computing it as the number of occupied CLBs times the execution time in milliseconds.

Figure 5 allows to compare the designs of the encapsulation module of Kyber, Saber, NTRU, and NTRU Prime, employing the *x*-net-based multipliers that realize for each of them the more suitable reduction strategy. A blue marker represents a design with a parameter set corresponding to the NIST security level 1, a yellow marker refers to security level 3, an orange marker corresponds to security level 5, while the red ones refer to security levels above level 5. The figure also shows as dashed lines the design space points that exhibit the same area-time (AT) product (lower is better) to the end of easing the evaluation of the efficiency of the encapsulation modules with the parameter sets recommended in the official specification of each cryptographic scheme.

Figures with similar trends were also obtained for decapsulation modules and key generation modules, employing the data in Appendix A, which shows the detailed and complete set of results we obtained from our design space exploration.

In Fig. 5 and in the data referring to decapsulation and key generation modules, by comparing the designs with an equivalent security level, it can be noted that the time spent in polynomial multiplications is larger in Kyber (a module RLWE scheme) and Saber (a module RLWR) than in NTRU-based schemes (right-most values on the *x* axis of Fig. 5). Moreover, such a difference increases with the security level. The only exception to such a trend is the latency of polynomial multiplications for the key generation of NTRU HPS and HRSS for the parameter sets ranked with security level 1 and 3 due to the large number of operations carried out. Nonetheless, this penalty is compensated by the flexibility of Kyber and Saber schemes, which have an almost identical polynomial multiplier usable in every parameter set (almost constant value on the *y* axis of the figure). As it can be clearly seen, given the large amount of parameter sets for NTRU Prime, the latency of the multiplication for our design and this cryptographic scheme is linear in the degree of the polynomials. As a consequence the performance penalty imposed by larger security levels grows more slowly than for Kyber and Saber.

When considering various parameter sets of equivalent security level, it becomes evident that NTRU Prime stands out as having the least efficient implementation among them. Comparing NTRU-based parameters with those of Kyber and Saber, we can see that the former exhibit a considerably lower degree of variability in terms of efficiency when increasing the security level with respect to the latter. The gathered data suggests that the *x*-net architecture is from 4 to 8 times more efficient when employed to compute polynomial multiplications during encapsulations in NTRU rings, as shown by the marks in Fig. 5 which are close to the origin of the chart. This fact however does not hold anymore for the keygen and decapsulation operations, having fewer multiplications to perform than NTRU. Finally, it is worth noting that Kyber lags significantly behind the efficiency achieved

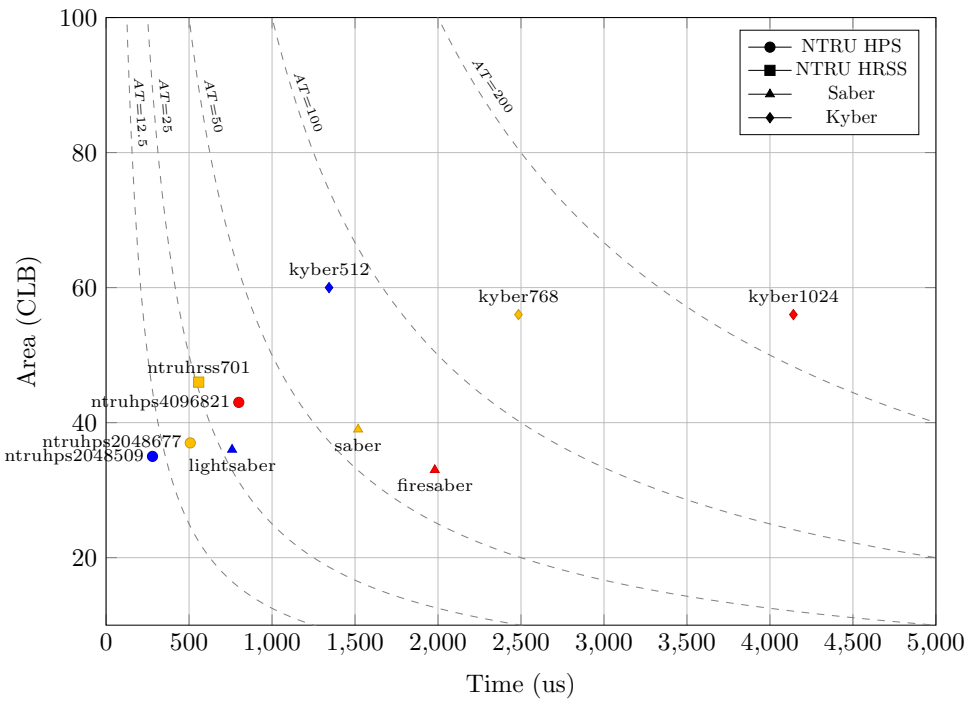


Fig. 6 Efficiency comparison of Comba-based designs. Blue, yellow and orange markers refer to parameters of security level 1, 3, and 5, respectively. Red markers denote parameters above security level 5. Dashed lines exhbit the same area–time (AT) product (lower is better)

Table 3 Comparing results of our x-net and x^2-net with [19], and [21] with multiplier architectures adaptable to multiple cryptographic schemes targeting a UltraScale+ target

Work	Parameter set	CLB	Frequency	CC	LUT	FF	BRAM
x-net	sntrup761	6757	312	762	38798	21768	2
[19]	sntrup761	9699	255	762	65207	32929	6
x-net	ntruhrss701	3088	588	702	18383	10898	2
[19]	ntruhrss701	5476	300	702	33230	32327	6
x^2-net	ntruhps4096821	7766	412	413	46293	12029	2
[21]	ntruhps4096821	8728	187	412	54478	9227	–
x^2-net	firesaber	5602	338	129	30809	4654	2
[21]	firesaber	3427	310	128	22127	7841	–

The count of clock cycles do not consider the time to transfer the operands and the result. No DSP were used

Table 4 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA for the unified designs compatible with the specified parameter sets

Design	Supported ciphers	Security level	CLB	Freq.	LUT	FF	CARRY8	DSP48E2
x-net	NTRU, NTRU Prime, Saber, Kyber	AES-128	12090	272	70704	20184	2630	2
		AES-192	13935	247	83922	24237	3064	2
		AES-256	15273	241	94410	27276	3448	2
x-net	NTRU, NTRU Prime, Kyber	AES-128	8825	272	53479	18750	2624	2
		AES-192	10071	247	63718	22593	3058	2
		AES-256	11435	244	71775	25401	3442	2
Comba	NTRU, Saber Kyber	AES-256	67	328	394	142	30	0

The design based on x-net is configured to transfer 4 $\mathcal{R}_p$ coefficients and 1 $\mathcal{R}_q$ coefficients per clock cycle. Each supported parameter set can be selected at run time

by the Learning With Errors (LWE) scheme Saber. In fact, Kyber's design efficiency is approximately two times worse than the one of Saber (Tables 3, 4).

The full results of our design space exploration of the x-net architecture design are reported in Tables 5, 6, 7, 8 in the Appendix section.

Comba Performance Results

Considering the results of the synthesis campaign of the Comba-based multipliers, the design shows a remarkably small area requiring almost two orders of magnitude less CLBs with respect to the x-net design. This compactness in turn allows to further improve the working frequency up to 36% for NTRU parameter sets.

From the chart depicted in Fig. 6 obtained with analogous conditions of the Fig. 5, see similar trends than the ones obtained with the x-net architecture: Kyber is the slowest cryptoscheme, NTRU has the fastest and most efficient implementation during the encapuslation and decapsulation procedures, and Saber is showing the same results for the key generation.

These results proved that an extremely compact solution requiring almost two order of magnitude less area than the low-latency solution based on x-net still maintains competitive efficiency. In scenarios where the extra incurred cost of a large low-latency design is not feasible, a compact design with competitive efficiency presents an appealing solution when a latency of few milliseconds is admissible.

We report the complete results of the synthesis campaign in Table 9 in the Appendix.

Comparison with the State-of-the-Art Results

Table 3 reports the comparison of our cryptosystem-specialized designs with the existing state of the art on NTRU and NTRU Prime linear time multipliers. We note that our design achieves a 30–40% reduction in the required CLBs for both cryptosystems, when comparing our solution which loads a single $\mathcal{R}_q$ coefficient (x-net) with the one in [19]. Furthermore, we also obtain a 28–96% gain in working frequency with respect to the same design, therefore achieving also a higher area-time efficiency. We compare our solution loading two $\mathcal{R}_q$ coefficients at once, with the only currently available data point in the public technical report [21]. The solution reported in the technical report, where it is denoted as x^2-net, is 10% larger in area a 2.2× slower in the working frequency for the design for NTRU. These results show how

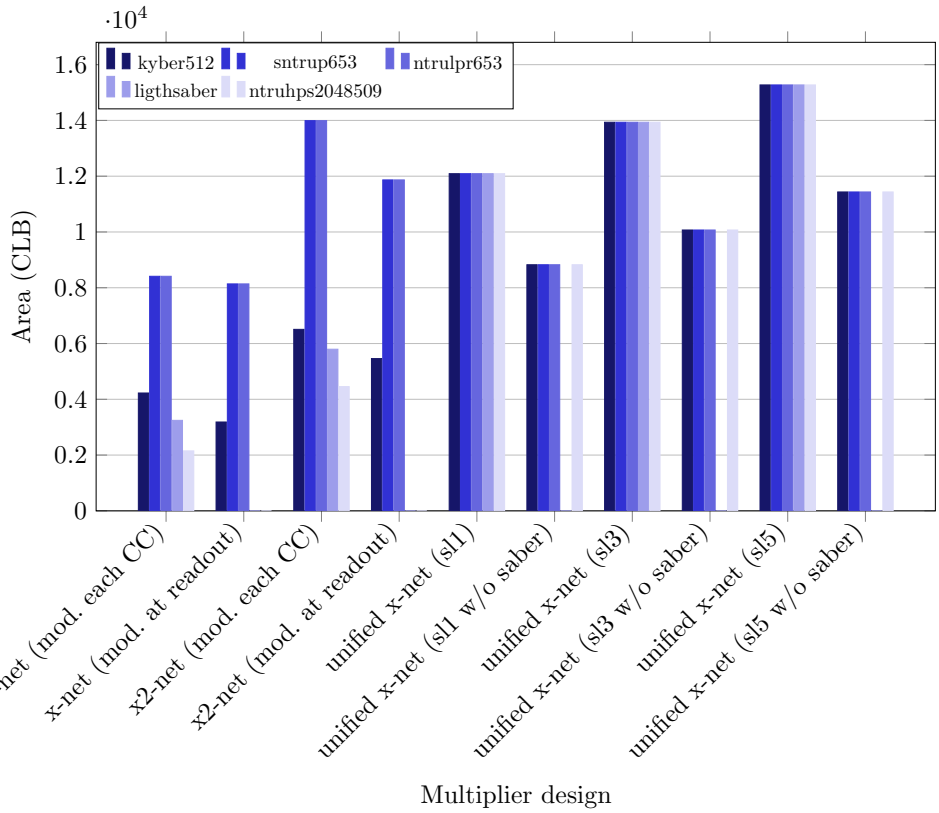


Fig. 7 Comparison of the area of the designs of different cryptoschemes (security level 1)

the x-net design is a remarkable fit for the $\mathcal{R}_p \times \mathcal{R}_q$ multiplications in NTRU and NTRU Prime.

Unified Design Performance Figures

We now analyze the results of the unified architecture supporting polynomial multiplications for all four cryptographic schemes at hand. The results are reported in Table 4.

When considering a unified x-net design capable of adapting at run time to the required polynomial ring, we basically need to instantiate the longest LFSR among the ones for the the supported parameter sets, pre-compute $2p + 1$ integer multiplication outcomes for the largest value of p in use across all schemes, and insert multiplexers to propagate the correct data to the functional units. By contrast, obtaining a unified Comba design is less demanding on FPGA resources, since the only accumulator which is present in the Comba datapath has a size depending uniquely on the value of the large modulus q. As a consequence, we report the synthesis results of the Comba unified design for the highest security level alone, as they match the ones for all other security levels.

Analyzing the results in Table 4, we observe that, while raising the security level from 1 to 3 (5) the unified designs require $\approx 15\%$ ($\approx 30\%$, respectively) more FPGA resources to be implemented. In particular, we also observe that omitting Saber from the supported cipher set allows for a consistent FPGA resource saving, such that the design compatible with parameters up to security level 5 is more compact than the smaller one which only has compatibility with lightsaber parameters. Raising the desired security level for the integrated design also has a negative impact on the maximum working frequency; however, such an impact is quite limited ($\approx 10\%$). Finally, we note that the FPGA resource requirements for the unified Comba design are approximately two orders of magnitude smaller with respect to the x-net design, therefore providing an extremely appealing optimization corner for resource constrained environments.

By looking at the area comparison of the unified design with the ones specialized to a specific parameter set depicted in Fig. 7, the design providing complete run time flexibility takes around 50% more area resources than the largest tailored component (NTRU Prime) required to run the most demanding cipher of the same security level set, and when support to Saber is dropped, the area penalty drops to almost negligible values. Furthermore, the achieved running frequency is only 5 to 22% slower than the slowest component it encompasses, while taking no penalty on the number of clock cycles used to compute any of the multiplications with respect to a dedicated design.

The x^2-net designs specialized for the NTRU HPS/HRSS parameters are more compact than the ones tailored for Saber parameter sets, suggesting that the increasing of the

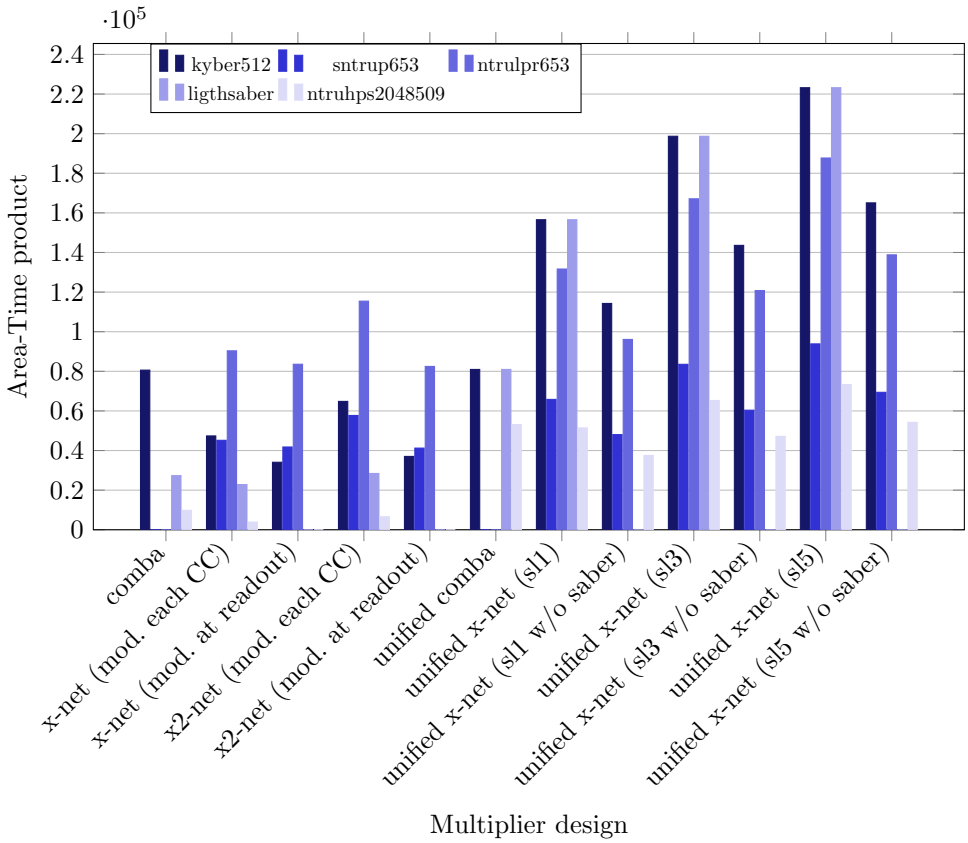


Fig. 8 Comparison of the efficiency of the multipliers among the encapsulation algorithms (security level 1)

size of $\mathcal{R}_p$ coefficients is more impactful than the increase of the length of polynomials.

Unsurprisingly, those high-performance designs require 2× the area than the x-net designs processing just one $\mathcal{R}_q$ coefficient. However, this is not the case for the designs tailored for the parameters of NTRU Prime and Kyber which, delaying the modulo of the coefficient at readout phase, use only 1.45× (1.71×, respectively) more resources.

The unified design implemented by means of a Comba-based multiplier needs to drop support for NTRU Prime ciphers, but shows an even greater flexibility supporting all security levels with a single design with no area penalties incurred. This peculiarity is due to the fact that supporting higher security levels for the investigated cryptographic schemes translates into smaller values of the parameter p (decreasing the complexity of the MAC unit) and larger values of n or k (increasing the latency of the operation for the Comba algorithm).

Ultimately this multiplier requires only 10% more area with respect to the largest tailored component (kyber512), and does not manifest a decrease in maximum working frequency.

When comparing the area–time (AT) product of all our designs in Fig. 8, we can see that for NTRU parameters both the x-net and Comba designs have higher efficiency than the other proposed schemes, suggesting a remarkable fit for the task. In particular, for the x-net design performing the coefficient modulo operation each clock cycle shows an order of magnitude difference. During the keygen and decapsulation, this no longer holds, and we recall that one of the three multiplications during decapsulation specified in round 3 submission of NTRU does not have one operand with $\mathcal{R}_p$ coefficients, thus requiring an additional cost.

Moreover, considering the Saber parameters, we see an almost equivalent AT product of the Comba and x-net designs.

For the unified x-net design supporting parameters up to security level 1, its efficiency is matching the one produced by the specialized design supporting the NTRU Prime parameter, suggesting that its support is the limiting factor leading to a 9.6× decrease of efficiency when compared with the specialized design supporting only the NTRU HPS parameter.

Similarly, when considering the unified Comba design and the specialized one for Kyber parameters we can determine a matching area-time product, signaling that the coefficient modulo operation is limiting the maximum frequency and increasing the area requirements.

The x^2-net designs performing the coefficient modulo each clock cycle shows a drastic reduction of efficiency with respect to the design processing a single $\mathcal{R}_q$

Table 5 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA specialized for each supported parameter sets

x-net algorithm – 4 $\mathcal{R}_p$ and 1 $\mathcal{R}_q$ coeffs. transfer – modulo each CC										
Security level	Parameter set	CLB	CC	Freq. MHz	Latency (µs)			AT product		
					Keyg.	Enc.	Dec.	Keyg.	Enc.	Dec.
AES-128	kyber512	4226	583	312	7.47	11.21	14.95	31.58	47.37	63.17
	ntruhps2048509	2150	1153	638	9.04	1.81	3.61	19.42	3.88	⋆7.77
	sntrup653	8411	1477	275	5.37	5.37	16.11	45.17	45.17	135.52
	ntrulpr653	8411	1477	275	5.37	10.74	16.11	45.17	90.34	135.52
	lightsaber	3245	583	497	4.69	7.04	9.38	15.22	22.83	30.45
AES-192	kyber768	3202	583	328	16.00	21.33	26.66	51.22	68.29	85.37
	ntruhps2048677	2825	1531	625	12.25	2.45	4.90	34.60	6.92	⋆13.84
	ntruhrss701	3336	1585	600	13.21	2.64	5.28	44.06	8.81	⋆17.62
	sntrup761	9691	1720	325	5.29	5.29	15.88	51.28	51.28	153.86
	ntrulpr761	9691	1720	325	5.29	10.58	15.88	51.28	102.57	153.86
	saber	3019	583	553	9.49	12.65	15.81	28.64	38.19	47.74
AES-256	kyber1024	3202	583	328	28.44	35.55	42.66	91.06	113.82	136.59
	ntruhps4096821	3712	1855	562	16.50	3.30	6.60	61.26	12.25	⋆24.50
	sntrup857	11142	1936	312	6.21	6.21	18.62	69.13	69.13	207.41
	ntrulpr857	11142	1936	312	6.21	12.41	18.62	69.13	138.27	207.41
	firesaber	2468	583	581	16.06	20.07	24.08	39.62	49.52	59.43
Above AES-256	sntrup953	12770	2152	312	6.90	6.90	20.69	88.08	88.08	264.24
	ntrulpr953	12770	2152	312	6.90	13.79	20.69	88.08	176.16	264.24
	sntrup1013	13017	2287	275	8.32	8.32	24.95	108.25	108.25	324.76
	ntrulpr1013	13017	2287	275	8.32	16.63	24.95	108.25	216.50	324.76
	sntrup1277	16686	2881	262	11.00	11.00	32.99	183.48	183.48	550.44
	ntrulpr1277	16686	2881	262	11.00	21.99	32.99	183.48	366.96	550.44

The design is based on the *x*-net algorithm, when 4 $\mathcal{R}_p$ and 1 $\mathcal{R}_q$ coefficients are loaded/read per clock cycle, and the modulo reduction operation is performed each clock cycle. One further $\mathcal{R}_q \times \mathcal{R}_q$ multiplication is performed during the decapsulation in NTRU, denoted by ⋆ symbol. Area–Time product computed as latency (ms) × CLB

coefficient. This fact suggests that the design is too large for our FPGA target design, probably congesting the routing resources and sparsely occupying the CLBs. This is not the case when the modulo operation is performed during the readout: we see an identical area–time product, suggesting that this technique is a viable solution for increasing the performance.

An important detail is that our unified design based on the Comba algorithm achieves substantial better efficiency than the security level 5 unified solutions based on the *x*-net architecture. When Kyber's parameter sets are in use, the gain is between 2× and 5.5×.

Finally, we want to highlight the fact that our *x*-net-based unified solutions would not perfectly suit a power-efficient scenario, particularly when considering the results of the design specialized for the NTRU parameter sets.

The Comba-based unified solution mitigates this drawback, and the efficiency improvement has profound implications for applications where power efficiency and cryptographic agility is paramount, such in case of low-power edge devices in a cloud computing architecture deployed in remote locations. Although, this solution exhibits a significant latency of few milliseconds, which could pose challenges for certain class of applications.

Concluding Remarks

In this work, we analyzed a flexible design for linear time polynomial multiplications, applicable to accelerate four post-quantum cryptographic primitives: Kyber, Saber, NTRU, and NTRU Prime. We reported quantitative results of the efficiency of primitive-tailored designs, obtaining area savings (10–40%) and significant frequency gains (96–120%) with respect to the state of the art of NTRU and NTRU Prime multipliers. Our unified design provides the first hardware implementation of a polynomial multiplier able to accelerate the computation of Kyber, Saber, NTRU, and NTRU Prime at all security levels in a single component with a frequency reduction in the range of 5–22%, and only a 50% multiplier area increase.

Table 6 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA specialized for each supported parameter sets

x-net algorithm – 4 $\mathcal{R}_p$ and 1 $\mathcal{R}_q$ coeffs. transfer – modulo at readout										
Security level	Parameter set	CLB	CC	Freq. MHz	Latency (µs)			AT product		
					Keyg.	Enc.	Dec.	Keyg.	Enc.	Dec.
AES-128	kyber512	3186	585	328	7.13	10.70	14.27	22.72	34.09	45.45
	sntrup653	8138	1479	288	5.14	5.14	15.41	41.79	41.79	125.37
	ntrulpr653	8138	1479	288	5.14	10.27	15.41	41.79	83.58	125.37
AES-192	kyber768	2615	585	312	16.88	22.50	28.12	44.12	58.83	73.54
	sntrup761	9043	1722	312	5.52	5.52	16.56	49.91	49.91	149.73
	ntrulpr761	9043	1722	312	5.52	11.04	16.56	49.91	99.82	149.73
AES-256	kyber1024	2615	585	312	30.00	37.50	45.00	78.45	98.06	117.67
	sntrup857	10141	1938	312	6.21	6.21	18.63	62.99	62.99	188.97
	ntrulpr857	10141	1938	312	6.21	12.42	18.63	62.99	125.98	188.97
above AES-256	sntrup953	11073	2154	312	6.90	6.90	20.71	76.44	76.44	229.33
	ntrulpr953	11073	2154	312	6.90	13.81	20.71	76.44	152.89	229.33
	sntrup1013	12022	2289	312	7.34	7.34	22.01	88.19	88.19	264.59
	ntrulpr1013	12022	2289	312	7.34	14.67	22.01	88.19	176.39	264.59
	sntrup1277	14735	2883	325	8.87	8.87	26.61	130.71	130.71	392.13
	ntrulpr1277	14735	2883	325	8.87	17.74	26.61	130.71	261.42	392.13

The design is based on the *x*-net algorithm, when 4 $\mathcal{R}_p$ and 1 $\mathcal{R}_q$ coefficients are loaded/read per clock cycle, and the modulo reduction operation is performed at readout. Area-Time product computed as latency (ms) × CLB

Table 7 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA specialized for each supported parameter sets

x-net algorithm – 4 $\mathcal{R}_p$ and 2 $\mathcal{R}_q$ coeffs. transfer – modulo each CC										
Security level	Parameter set	CLB	CC	Freq. MHz	Latency (µs)			AT product		
					Keyg.	Enc.	Dec.	Keyg.	Enc.	Dec.
AES-128	kyber512	6508	327	197	6.64	9.96	13.28	43.21	64.81	86.42
	ntruhps2048509	4455	645	438	7.36	1.47	2.95	32.80	6.56	*13.12
	sntrup653	13992	825	200	4.12	4.12	12.38	57.71	57.71	173.15
	ntrulpr653	13992	825	200	4.12	8.25	12.38	57.71	115.43	173.15
	lightsaber	5796	327	400	3.27	4.91	6.54	18.95	28.42	37.90
AES-192	kyber768	5579	327	206	14.29	19.05	23.81	79.70	106.27	132.83
	ntruhps2048677	6208	855	475	9.00	1.80	3.60	55.87	11.17	*22.34
	ntruhrss701	7428	885	425	10.41	2.08	4.16	77.33	15.46	*30.93
	sntrup761	16429	960	188	5.11	5.11	15.32	83.89	83.89	251.67
	ntrulpr761	16429	960	188	5.11	10.21	15.32	83.89	167.78	251.67
	saber	4993	327	425	6.92	9.23	11.54	34.57	46.10	57.62
AES-256	kyber1024	5579	327	206	25.40	31.75	38.10	141.69	177.11	212.54
	ntruhps4096821	8052	1035	438	11.82	2.36	4.73	95.13	19.02	*38.05
	sntrup857	19034	1080	188	5.74	5.74	17.23	109.34	109.34	328.03
	ntrulpr857	19034	1080	188	5.74	11.49	17.23	109.34	218.68	328.03
	firesaber	4200	327	375	13.95	17.44	20.93	58.59	73.24	87.89
above AES-256	sntrup953	21165	1200	188	6.38	6.38	19.15	135.09	135.09	405.28
	ntrulpr953	21165	1200	188	6.38	12.77	19.15	135.09	270.19	405.28
	sntrup1013	22219	1275	188	6.78	6.78	20.35	150.68	150.68	452.06
	ntrulpr1013	22219	1275	188	6.78	13.56	20.35	150.68	301.37	452.06
	sntrup1277	27283	1605	200	8.03	8.03	24.07	218.94	218.94	656.83
	ntrulpr1277	27283	1605	200	8.03	16.05	24.07	218.94	437.89	656.83

The design is based on the *x*-net algorithm, 4 $\mathcal{R}_p$ and 2 $\mathcal{R}_q$ coefficients are loaded/read per clock cycle, and the modulo reduction operation is performed each clock cycle. One further $\mathcal{R}_q \times \mathcal{R}_q$ multiplication is performed during the decapsulation in NTRU, denoted by ⋆ symbol. Area–Time product computed as latency (ms) × CLB

Table 8 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA specialized for each supported parameter sets

x-net algorithm – 4 $\mathcal{R}_p$ and 2 $\mathcal{R}_q$ coeffs. transfer – modulo at readout										
Security level	Parameter set	CLB	CC	Freq. MHz	Latency (µs)			AT product		
					Keyg.	Enc.	Dec.	Keyg.	Enc.	Dec.
AES-128	kyber512	5460	329	291	4.52	6.78	9.04	24.69	37.03	49.38
	sntrup653	11867	827	238	3.47	3.47	10.42	41.23	41.23	123.70
	ntrulpr653	11867	827	238	3.47	6.95	10.42	41.23	82.47	123.70
AES-192	kyber768	4733	329	291	10.18	13.57	16.96	48.15	64.21	80.26
	sntrup761	13762	962	238	4.04	4.04	12.13	55.62	55.62	166.87
	ntrulpr761	13762	962	238	4.04	8.08	12.13	55.62	111.25	166.87
AES-256	kyber1024	4733	329	291	18.09	22.61	27.13	85.61	107.02	12.842
	sntrup857	15404	1082	238	4.55	4.55	13.64	70.02	70.02	21.008
	ntrulpr857	15404	1082	238	4.55	9.09	13.64	70.02	140.05	21.008
above AES-256	sntrup953	18111	1202	238	5.05	5.05	15.15	91.46	91.46	274.40
	ntrulpr953	18111	1202	238	5.05	10.10	15.15	91.46	182.93	274.40
	sntrup1013	19201	1277	238	5.37	5.37	16.10	103.02	103.02	309.07
	ntrulpr1013	19201	1277	238	5.37	10.73	16.10	103.02	206.04	309.07
	sntrup1277	23332	1607	238	6.75	6.75	20.26	157.54	157.54	472.62
	ntrulpr1277	23332	1607	238	6.75	13.50	20.26	157.54	315.08	472.62

The design is based on the *x*-net algorithm, 4 $\mathcal{R}_p$ and 2 $\mathcal{R}_q$ coefficients are loaded/read per clock cycle, and the modulo reduction operation is performed at readout. Area-Time product computed as latency (ms) × CLB

Furthermore, we presented a simpler and compact multiplier based on the Comba algorithm specialized for NTRU, Saber and Kyber schemes, in some cases able to achieve a comparable design efficiency than the *x*-net-based solutions, and proposing a more efficient unified design able to adapt at run time to the different polynomial rings using only 10% more resources and without a frequency drop compared to its specialized solution.

Appendix A

Design Space Exploration Results

This section includes the results of our comprehensive design space exploration.

Table 9 Results of the synthesis targeting an Xilinx UltraScale+ ZCU106 FPGA specialized for each supported parameter sets

Comba algorithm – modulo each CC										
Security level	Parameter set	CLB	kCC	Freq. MHz	Latency (µs)			AT product		
					Keyg.	Enc.	Dec.	Keyg.	Enc.	Dec.
AES-128	ntruhps2048509	35	260.1	930	1398.4	279.6	559.3	48.9	9.7	⋆19.5
	lightsaber	36	66.1	522	506.1	759.2	1012.3	18.2	27.3	36.4
	kyber512	60	66.1	295	895.6	1343.4	1791.2	53.7	80.6	107.4
AES-192	ntruhps2048677	37	459.7	906	2536.9	507.3	1014.7	93.8	18.7	⋆37.5
	ntruhrss701	46	492.8	883	2790.5	558.1	1116.2	128.3	25.6	⋆51.3
	saber	39	66.1	522	1138.8	1518.4	1898.1	44.4	59.2	74.0
	kyber768	56	66.1	319	1863.5	2484.7	3105.9	104.3	139.1	173.9
AES-256	ntruhps4096821	43	675.7	845	3998.1	799.6	1599.2	171.9	34.3	⋆68.7
	firesaber	33	66.1	667	1584.5	1980.6	2376.7	52.2	65.3	78.4
	kyber1024	56	66.1	319	3313.0	4141.3	4969.5	185.5	231.9	278.2

The design is based on the Comba algorithm, and the modulo reduction operation is performed each clock cycle. Area–Time product computed as latency (ms) × CLB

These tables are meant to strengthen the reproducibility of our designs and trade-offs.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

Data availability The complete dataset is also available in CSV format at https://doi.org/10.5281/zenodo.8337625.

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Human and animal rights No research involving humans and animals was performed, nor personal data being collected.

Informed consent There was no need to ask for informed consent.

References

1. NIST PQC Team: PQC standardization process: announcing four candidates to be standardized, plus fourth round candidates. 2022. https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4.
2. The CRYSTALS-Kyber Team: CRYSTALS-cryptographic suite for algebraic lattices-Kyber 2020. https://pq-crystals.org/kyber/.
3. The NTRU Team: NTRU–a submission to the NIST post-quantum standardization effort 2020. https://www.ntru.org/.
4. The NTRU Prime Team: NTRU Prime. 2022. https://ntruprime.cr.yp.to/.
5. The SABER Team: SABER–MLWR-Based KEM 2019. https://www.esat.kuleuven.be/cosic/pqcrypto/saber/.
6. Alagic G, Apon D, Cooper D, Dang Q, Dang T, Kelsey J, Lichtinger J, Miller C, Moody D, Peralta R, Perlner R, Robinson A, Smith-Tone D, Liu Y-K. Status report on the third round of the NIST post-quantum cryptography standardization process. 2022. https://doi.org/10.6028/NIST.IR.8413-upd1.
7. ISE Crypto PQC working group: Securing tomorrow today: Why Google now protects its internal communications from quantum threats. https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms.
8. Schmieg S. PQC at Google. Invited talk at The 14th International Conference on Post-Quantum Cryptography, PQCrypto 2023. https://pqcrypto2023.umiacs.io/slides/Invited.3.pdf.
9. The OpenSSH Team: OpenSSH Changelog for version 9.0 2022. https://www.openssh.com/txt/release-9.0.
10. Comba PG. Exponentiation cryptosystems on the IBM PC. IBM Syst J. 1990;29(4):526–38. https://doi.org/10.1147/sj.294.0526.
11. Karatsuba A. Multiplication of multidigit numbers on automata. Soviet Phys Doklady. 1963;7:595–6.
12. Bodrato M. Towards optimal toom-cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In: Carlet C, Sunar B (eds) Arithmetic of finite fields, first international workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. In: Proceedings. Lecture Notes in Computer Science, vol. 4547, pp. 116–133. Springer 2007. https://doi.org/10.1007/978-3-540-73074-3_10.
13. Ylonen T. IETF RFC 4252—The secure shell (SSH) Authentication protocol. 2006. https://www.rfc-editor.org/rfc/rfc4252.
14. Antognazza F, Barenghi A, Pelosi G, Susella R. An efficient unified architecture for polynomial multiplications in lattice-based cryptoschemes. In: Mori P, Lenzini G, Furnell S (eds) Proceedings of the 9th International Conference on Information Systems Security and Privacy, ICISSP 2023, Lisbon, Portugal, February 22-24, 2023, pp. 81–88. SciTePress 2023. https://doi.org/10.5220/0011654200003405.
15. Marotzke A. A constant time full hardware implementation of streamlined NTRU prime. In: Liardet P, Mentens N (eds) Smart card research and advanced applications-19th international conference, CARDIS 2020, virtual event, november 18–19, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12609, pp. 3–17. Springer 2020. https://doi.org/10.1007/978-3-030-68487-7_1.
16. Dang VB, Mohajerani K, Gaj K. High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber 2021. https://eprint.iacr.org/2021/1508.
17. Liu B, Wu H. Efficient architecture and implementation for NTRUEncrypt system. In: IEEE 58th International Midwest Symposium on Circuits and Systems, MWSCAS 2015, Fort Collins, CO, USA, August 2-5, 2015, pp. 1–4. IEEE 2015. https://doi.org/10.1109/MWSCAS.2015.7282143.
18. Basso A, Roy SS. Optimized polynomial multiplier architectures for post-quantum KEM Saber. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021, pp. 1285–1290. IEEE 2021. https://doi.org/10.1109/DAC18074.2021.9586219.
19. Farahmand F, Dang VB, Nguyen DT, Gaj K. Evaluating the Potential for Hardware Acceleration of four NTRU-based key encapsulation mechanisms using software/hardware codesign. In: Ding J, Steinwandt R (eds) Post-quantum cryptography-10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers. Lecture Notes in Computer Science, vol. 11505, pp. 23–43. Springer 2019. https://doi.org/10.1007/978-3-030-25510-7_2.
20. Peng B, Marotzke A, Tsai M, Yang B, Chen H. Streamlined NTRU prime on FPGA 2021. https://eprint.iacr.org/2021/1444.
21. Carter E, He P, Xie J. High-performance polynomial multiplication hardware accelerators for KEM Saber and NTRU 2022. https://eprint.iacr.org/2022/628.