A Better Kyber Butterfly for FPGAs

Jonas Bertels
COSIC
KU Leuven
Leuven, Belgium
jonas.bertels@esat.kuleuven.be

Quinten Norga
COSIC
KU Leuven
Leuven, Belgium
quinten.norga@esat.kuleuven.be

Ingrid Verbauwhede
COSIC
KU Leuven
Leuven, Belgium
ingrid.verbauwhede@kuleuven.be

***Abstract*—Kyber was selected by NIST as a Post-Quantum Cryptography Key Encapsulation Mechanism standard. This means that the industry now needs to transition and adopt these new standards. One of the most demanding operations in Kyber is the modular arithmetic, making it a suitable target for optimization. This work offers a novel modular reduction design with the lowest area on Xilinx FPGA platforms. This novel design, through K-reduction and LUT-based reduction, utilizes 49 LUTs and 1 DSP as opposed to Xing and Li's [XL21] 2021 CHES design requiring 90 LUTs and 1 DSP for one modular multiplication. Our design is the smallest modular multiplier reported as of today.**

***Index Terms*—CRYSTALS-Kyber, Hardware Design, FPGA**

I. Introduction to the Butterflies Used in CRYSTALS-Kyber

Large-scale quantum computers present an ever-increasing threat to currently deployed public key cryptographic algorithms. Quantum attacks will be able to break the security of cryptographic algorithms based on the large integer factorization or discrete logarithm problem [Sho97]. However, cryptographic algorithms resistant to these attacks exist and are commonly referred to as Post-Quantum Cryptography (PQC). The US National Institute of Standards and Technology (NIST) launched its initial PQC standardization effort in 2016, recognizing the need for formulating novel public-key standards [Div16]. The first schemes selected for standardization were announced in 2022 by NIST. Three out of four are lattice-based schemes: Kyber [Div23b], Dilithium [Div23c], and Falcon [P^+20].

Kyber (or 'ML-KEM', see FIPS 203 [Div23a]), and all other lattice-based cryptography, heavily rely on modular polynomial arithmetic. More specifically, the bottleneck is modular polynomial multiplication. Kyber uses the fast Number Theoretic Transform (NTT), performed on multiple large-degree polynomials of size N (e.g., $N = 256$ for Kyber and Dilithium). The open-source `pqm4` library[1] includes implementations and benchmarking of all NIST-selected schemes. Depending on the security level, 10% to 24% of the execution time is spent in the NTT and Inverse NTT (INTT) routines.

A significant component of the NTT and INTT operation is the *butterfly unit*, which is applied recursively. Due to the nature of the NTT operation, optimized (e.g., parallelized) implementations are possible in hardware. More specifically, as illustrated in [BAK21b], duplicating butterfly units and parallelizing the NTT operation reduces the total latency. Area utilization can be exchanged for latency in hardware implementations of NTT. This paper thus focuses on the arithmetic cost of one butterfly as the basic unit, expressed in LUTs and DSPs.

Related Work. Prior work has targeted and optimized the NTT operation for both large AMD and Intel processors, the more resource-constrained Arm Cortex-M4 platform and FPGA [Sei18], [LS19], [BKS19], [BC22], [JGCS21], [GLK22a].

Contribution. The main contributions of this paper are the following:

- An implementation and comparison of the best (i.e., lowest area cost (LUT/DSP)) modular reduction algorithms and a novel 49-LUT reduction design.
- An open source Github repository of said implementations.

The intended audience for this paper is hardware designers looking to implement and/or optimize their Kyber designs on FPGA. Many research papers in this area put significant design effort into innovative modular reduction. However, the design comparison in this paper shows a large fraction does not improve on a state-of-the-art design like **Xing et al.'s [XL21]** which appeared in CHES 2021. To remedy this, a modular reduction design[2] is put in the public domain in our repository which utilizes **49 LUTs** as opposed to the **90 LUTs** in Xing et al.'s state-of-the-art design.

Outline. In this paper, the butterfly operation is explored in detail. First, the architecture of a butterfly is introduced (Section II). Secondly, modular reduction, the most critical operation in a butterfly, is discussed in Section III. Here, techniques relevant to our Kyber modular multiplier implementation in hardware are specifically analyzed. In Section IV, several state-of-the-art modular multiplier designs are presented and compared. Finally, a novel hardware design for the modular multiplication is introduced in Section V and contrasted with prior art in Section VI. The open-source implementation offers a significant improvement over prior work.

[1]https://github.com/mupq/pqm4

[2]https://github.com/axytho/KyberButterflyCollection/blob/main/butterflies/butterfly_Best.v

II. Preliminaries

This section introduces the butterfly operation and its usage as a crucial component in NTTs. Different types of butterflies are introduced, discussed and contrasted.

A. *The Cooley-Tukey and Gentleman-Sande Butterfly*

Butterflies and their implementations designed for Kyber-parameter (Inverse) Number Theoretic Transforms ((I)NTTs) come in a range of different forms. The calculations contained in a single butterfly are simple: a modular addition, a modular subtraction, and a single modular multiplication with a twiddle factor. Both types of butterfly, Cooley-Tukey and Gentleman-Sande, require an equal amount of additions, subtractions, and multiplications. For Cooley-Tukey, first multiply, then add and subtract:

$$a_{out} = (a_{in} + b_{in} \cdot w) \bmod q \tag{1}$$

$$b_{out} = (a_{in} - b_{in} \cdot w) \bmod q \tag{2}$$

For Gentleman-Sande, first add and subtract, then multiply:

$$a_{out} = (a_{in} + b_{in}) \cdot w \bmod q \tag{3}$$

$$b_{out} = (a_{in} - b_{in}) \cdot w \bmod q \tag{4}$$

with a_{in} and b_{in}, 12 bit inputs, w a 12 bit constant multiplicand, $w = \omega^i, i \in [0 \dots 127]$ with $\omega^{n/2} = -1 \bmod q$.

B. *Unified Butterflies*

Many Kyber designs choose to support both Gentleman Sande and Cooley Tukey. The reason for this lies in how a polynomial multiplication of two polynomials modulo $X^N + 1$ is done.

It is a very common misconception [BLP$^+$23], [SS23], [DMG23], [BAK21a], [BUC19], [BPC19], [XHY$^+$20], [ZLZ$^+$22], [BAK21b] that a combined CT/GS architecture removes the bit-reversal step. This is not the case: to remove the bit-reversal step, the structure of the forward NTT itself can be changed so that it maps a normal order polynomial to a bit-reversed order polynomial. In this case, the inverse NTT will map a bit-reversed order polynomial to a normal order polynomial, independent of butterfly architecture. Alternatively, the NTTs can be designed to map normal order to normal order (see Chapter 6 of [CCG00], [POG15]). However, as the structure of the NTT is independent of the structure of the butterfly, one can use a Gentleman-Sande butterfly to map normal order to bit-reversed order (see Figure 1) and could do vice versa with a Cooley-Tukey butterfly.
The actual reason for a combined GS/CT architecture is that properly configured (CT for NTT$_{no\to bo}$, GS for INTT$_{bo\to no}$), the architecture efficiently calculates the negatively wrapped convolution (see [POG15]).

III. Modular reduction

Now, various modular reduction techniques relevant to hardware designs are described.

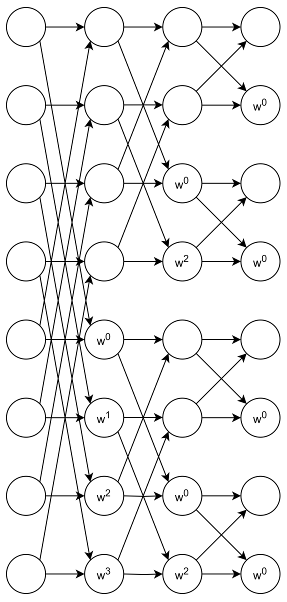

Fig. 1. A Gentleman-Sande (Decimation-In-Frequency) butterfly that maps Normal Order to Bit-reversed Order, proving that this capability is not unique to Cooley-Tukey. Whether a circuit maps normal order to bit-reversed or vice versa depends on the ordering of the butterflies themselves rather than on the internal architecture of the butterfly

A. *K-reduction*

In 2016, Longa and Naehrig [LN16] proposed a new modular reduction algorithm for lattice-based cryptography: K-reduction (Algorithm 1). K-reduction exploits the fact that the Kyber modulus q can be written as a power of two multiplied with a small constant k ($k = 13$) plus one. Under this modulus, reducing numbers that are to be multiplied by k is a low-complexity operation because any number multiplied by $k \cdot 2^8$ is equivalent to the negative of that number. In other words, the upper bits, starting from position 8, can be subtracted from the lower bits, once the lower bits have been multiplied by k (See Algorithm 1). Thus, rather than calculating the desired result:

$$a \cdot b \bmod q \tag{5}$$

The following is calculated (which is off by a constant factor $k = 13$):

$$\textbf{K-red}(a, b) = a \cdot b \cdot k \bmod q \tag{6}$$

with $q = 3329 = k \cdot 2^x + 1$ with $x = 8$ and $k = 13$. The fact that $k \cdot 2^8 \equiv -1 \bmod q$ is then leveraged in a similar fashion to bitwise multiplication. By splitting $k \cdot c$ into $k \cdot c_{[23:8]} \cdot 2^8 + k \cdot c_{[7:0]}$, and replacing the upper part $k \cdot 2^8$ with -1, the end result is a single subtraction $-c_{[23:8]} + k \cdot c_{[7:0]}$ to reduce the size of the product by almost 8 bits. The fact that the calculation is off by a constant factor k is rectified by multiplying one of our inputs (namely the twiddle factor) by a constant factor $k^{-1} \pmod q$.

K-reduction does not fully reduce a 24-bit value to a 12-bit value since $c_{[23:8]}$ is a 16-bit value, but there are multiple approaches to utilize K-reduction for Kyber modular multiplication.

```
Algorithm 1: K-red Algorithm
Data: $a$ and $b$, 12 bit inputs, $q = k \cdot 2^x + 1$ with $x = 8$
      and $k = 13$ so that $q = 3329$ and
      $k \cdot 2^8 \equiv -1 \bmod q$
Result: $r = a \cdot b \cdot k \bmod q$

1 begin
2     $c = a \cdot b$;
3     $c = c_{[23:8]} \cdot 2^8 + c_{[7:0]}$;
4     $r = k \cdot c \bmod q = (c_{[23:8]} \cdot k \cdot 2^8 + k \cdot c_{[7:0]}) \bmod q$;
5     $r = (c_{[23:8]} \cdot (-1) + k \cdot c_{[7:0]}) \bmod q$;
6 end
```

B. *LUT-based reduction*

In LUT-based reduction, the modular reduction is precomputed for a group of bits in the multiplication result $c = a \cdot b$. For instance, precomputing the most significant four bits consists of storing which 12-bit value corresponds to $2^{20} \cdot c_{[23:20]} \bmod q$ for every single possible value of $c_{[23:20]}$. As there are 2^4 possible values for $c_{[23:20]}$, 2^4 values, each of 12 bits, must be stored. If a single 4-to-1 LUT is used per bit, 12 LUTs are used in total. The LUT-based reduction algorithm is described in Algorithm 2.

LUT-based reduction, like K-reduction, does not fully reduce a 24-bit value to a 12-bit value since $c_{[19:0]}$ is a 20-bit value.

```
Algorithm 2: LUT-based Algorithm
Data: $a$ and $b$, 12 bit inputs, LUT a function mapping
      4 bits to 12 bits
Result: $r$ reduced to 21 bits from 24 bits $\pmod q$

1 begin
2     $c = a \cdot b$;
3     $c = c_{[23:20]} \cdot 2^{20} + c_{[19:0]}$;
4     $\mathbf{LUT}(c_{[23:20]}) = (c_{[23:20]} \cdot 2^{20}) \bmod q$;
5     $r \equiv c \equiv \mathbf{LUT}(c_{[23:20]}) + c_{[19:8]}$;
6 end
```

IV. Hardware implementations

Five different K-reduction strategies are discussed here. Based on these five K-reduction strategies, a novel design is introduced in the next section.

K-reduction does not fully reduce the product but rather decreases its size by 7 bits, and thus an extra reduction is required to complete the full reduction for Kyber parameters from 24 bits to 12 bits. Firstly, Bishes-Niasar et al. [BLP$^+$23] created the K^2-RED approach (not to be confused with the K2-RED algorithm from Longa and Naehrig [LN16]). The K^2-RED approach applies the K-RED algorithm twice, requiring six additions in total, followed by a conditional addition. This reduces the result to the desired 12 bits. Li et al. [LQYW24] improve this by utilizing K-reduction with $k = -13$ (as usual) in the first step and $k = -13 \cdot 2^4$ for the second step. The second K-reduction then consists of a series of 4 bit instead of 8 bit additions. We consider this to be K$^{1.5}$-red because it is an 8-bit K-red followed by a 4-bit K-red which has the area usage of doing 1.5 times an 8-bit K-red.

Secondly, Salarifard and Soleimany [SS23] create a trade-off between memory and logic units by pre-calculating $a_{[5:0]} \cdot b \cdot k^{-1} \bmod q$ for all possible b and all possible 6 bits of a. The factor k^{-1} is included to cancel with a later K-reduction. Because b are the twiddle factors in the NTT, there are only 128 possible values for b. This results in:

$$a \cdot b \cdot k^{-1} \bmod q = \Big(a_{[11:6]} \cdot b \cdot 2^6 + a_{[5:0]} \cdot b\Big) \cdot k^{-1} \bmod q \quad (7)$$

In other words, two accesses to memory suffice to reduce the result:

$$r_1 \cdot 2^6 + r_2 \bmod q \quad (8)$$

with r_1 and r_2 12 bit numbers. This result can be reduced using K-reduction. Alternatively, two different memories can be instantiated: one containing $a_{[5:0]} \cdot b \bmod q$ and one containing the calculated values for $a_{[12:6]} \cdot b \cdot 2^6 \bmod q$. These two values can then be combined with a modular addition to gain the final bit result without requiring K-reduction (or pre-calculation with a factor k^{-1}). Each BRAM requires storage for 128×64 elements of 12 bits.

Thirdly, Ni et al. [NKLO23] leverage LUT-based reduction to cheaply remove the top 4 bits, utilizing only 12 elements of 4:1 LUTs. It should be noted that LUT-6 and LUT-5:2 elements are standard primitives on 7-series Xilinx FPGAs, including the xc7a200t platform targeted by Ni et al.. As such, Ni et al. do not fully utilize the primitives available.

Finally, Nguyen et al. [NPH23] merge 6:1 LUT-based reduction with Karatsuba [Kar95] and do the final reduction through K-reduction, requiring only three adders. Table I gives the arithmetic cost of the K-reduction algorithms.

Nguyen et al. [NPH23]'s design requires 12 bits of arithmetic cost for the 6:1 LUT reduction, 12 bits for the addition of x with the Karatsuba multiplication, and an addition of 10 bits, a subtraction of 12 bits and a final subtraction of 12 bits. A signed bit is saved from this final subtraction as the result $r \in (-q, q)$. This signed bit is then utilized in the butterfly architecture's modular addition and subtraction to ensure the final output is $[0, q)$. This merging has an arithmetic cost of 12 bits.

Ni et al. [NKLO23] combine the results by adding the result of the Look-Up Table to $-c_{[19:8]} + k \cdot c_{[7:0]}$, then select between adding 0, q or $-q$. This requires a 10-bit, 12 bit and 13-bit adder or subtractor for the K circuit, 12 bits worth of LUTs for calculating x, 14 bits for the addition of x and $-c_{[19:8]} + k \cdot c_{[7:0]}$ and a further 12 bits for the multiplexer and 12 bits for the final subtraction circuit. In total, 85 bits are required for the reduction (see Figure 2).

It should be noted that Ni et al.'s [NKLO23] design does not fully reduce the final result within $[0, q)$. Figure 2 is misleading as it suggests that the sum of the LUT and K-RED method

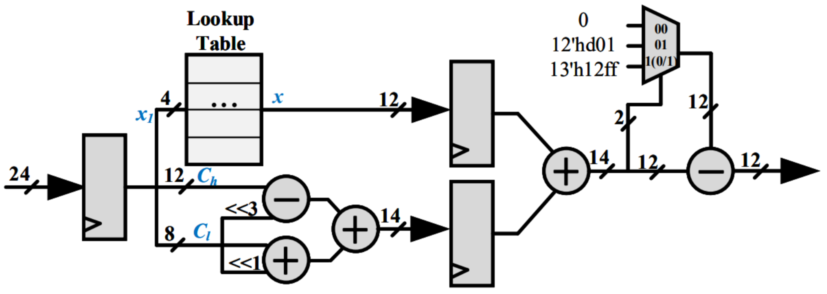

Fig. 2. We based one of our implementations on Ni et al.'s modular reduction, taken from [NKLO23] and used under provisions of fair use with the following corrections: Note that the shift by one at the bottom of the figure should be a shift by two, and the addition of $C_h - (C_l << 3)$ and $(C_l << 2) + C_l$ should be a subtraction. Moreover, note that result $\in [0, 2^{12})$ rather than $\in [0, q)$.

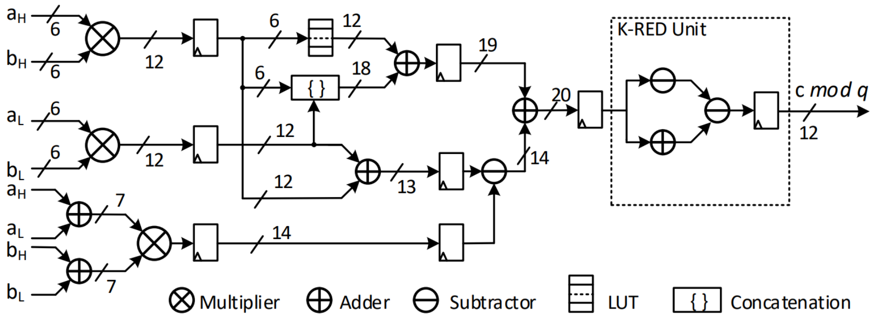

Fig. 3. Nguyen et al.'s modular reduction, taken from [NPH23] and used under provisions of fair use. We modified this design with a conditional subtraction to bring the result within $[0, q)$.

lies within $[-q, 2q)$ and requires a single adder and 3:1 MUX to reduce the circuit. In reality, the final result lies within $[-q, 2^{12}+q)$, and even if this were not the case, the design in Figure 2 would only reduce the result to $[0, 2^{12})$ as the circuit has no way to detect that the result lies in $[q, 2^{12})$. Modifying Ni et al.'s design to reduce the final result within $[0, q)$ by moving two conditional subtractions with q to the very end of the circuit ensures a fair comparison between the different butterfly units.

TABLE I
ARITHMETIC COMPLEXITY OF DIFFERENT K-REDUCTION DESIGNS.

Paper	Type	Add bits	# Mults
Bisheh-Niasar et al. [BAK21a]	K^2-red	96	1
Li et al. [LQYW24]	$K^{1.5}$-red	72	1
Salarifard et al. [SS23]	BRAM	36	2 BRAM
Ni et al. [NKLO23]	LUT/K-red	85	1
Nguyen et al. [NPH23]	LUT/K-red	269	0

There is a trade-off between the LUT-6 strategy used by Nguyen et al. and the LUT-4 approach, favored by Ni et al. A LUT-6 reduction, *followed* by K-reduction, requires only a conditional addition to ensure the final result lies in $[0, q)$. A LUT-4 addition requires conditional subtraction and a conditional addition to reduce the final result. The LUT-6 approach should be favored when LUT-6 elements are primitives in the architecture, whereas the LUT-4 approach should be favored when LUT primitives are unavailable, with the LUT-4 implemented in logic gates.

V. A Novel Design

A novel design that combines the best features of Ni et al. and Nguyen et al. for FPGA use is proposed here. It utilizes LUT-6s and DSPs to minimize area usage. The design is described in Figure 4 and can be found in our repository [3]. It is also described by Algorithm 3. This novel design

Algorithm 3: Our Kyber Modular Reduction

Data: a and b, 12 bit inputs, $q = k \cdot 2^x + 1$ with $x = 8$ and $k = 13$ so that $q = 3329$ and $k \cdot 2^8 \equiv -1 \bmod q$, **LUT** a function mapping 6 bits to 12 bits
Result: $r = a \cdot b \cdot k \bmod q$

```
1 begin
2     $c = a \cdot b$;
3     $y = \textbf{LUT}(c_{[23:18]}) = (c_{[23:18]} \cdot 2^{18} \cdot k) \bmod q$;
4     $\text{LUT-RED} = c_{[17:0]} + y$;
5     $\text{K-RED} = -\text{LUT-RED}_{[18:8]} + k \cdot \text{LUT-RED}_{[7:0]}$;
6     $r = \textbf{if } (\text{K-RED} < 0) \textbf{ then } \text{K-RED} + q \textbf{ else } \text{K-RED}$;
7 end
```

[3]https://github.com/axytho/KyberButterflyCollection

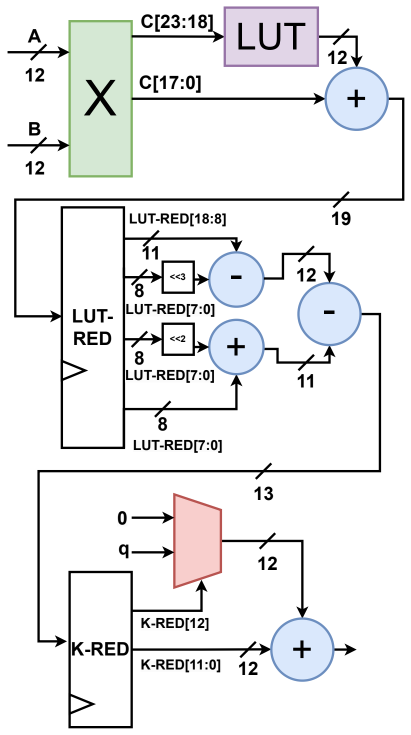

Fig. 4. Our design consists of a LUT-6-based reduction, followed by K-reduction, followed by a conditional addition.

consists of a sequential LUT-based reduction, followed by K-reduction, followed by a conditional addition. We conclude this composition of steps is most efficient, based on the following observations:

1) LUT-6 elements are available on Xilinx FPGAs as primitives and LUT-based reduction is the cheapest form of reduction when viewed as area consumed
2) K-reduction is a slightly more expensive reduction method which returns a value between $-q$ and q
3) Conditional addition or subtraction is a relatively expensive step (requiring 24 LUTs per element)

In order to minimize area utilization, it is critical to minimize the number of conditional additions or subtractions (3). Our design takes advantage of K-reduction as it reduces inputs to a narrower range than LUT-reduction and thus requires only a single conditional addition at the end of the circuit (2). The initial stage is completed by LUT-reduction because it is cheaper than K-reduction and there is no need for reducing the result to a small range in the first stage (1).

To demonstrate that K-reduction reduces to a narrower range than LUT-based reduction, consider the middle (K-reduction) stage of Figure 4. The maximum output of the K-reduction circuit `K-RED` will be $2^{11} - 1$ and the minimum output $-13 \cdot 2^8$. In contrast, a LUT6-reduction circuit that is fed a 19-bit input will result in the sum of a value smaller than $q = 3329$ and a value of 13 bits. The sum will be at most $2^{13} - 1 + 3329$, and would thus require at least two conditional subtractions (one with $3329 \cdot 2$, one with 3329) to fully reduce the value.

To fully demonstrate that a combination of a LUT-based reduction and a K-reduction method is the optimal design for FPGAs, we have optimized four different cases: (i) A design with two LUT-based reduction (implemented in parallel, as this is slightly more optimal), (ii) A design with two K-reductions (utilizing the techniques from Li et al. [LQYW24], but with a better implementation of the multiplication with 13 and other optimizations), (iii) a design with a K-reduction step followed by a LUT-reduction step and (iv) our design, which as previously mentioned consists of a LUT-based reduction followed by K-reduction. The results can be found in Table II. A design with double LUT-based reduction (i) suffers from the area consumption of two conditional subtraction blocks and as such requires 72 LUTs. Two conditional subtractions are necessary as LUT6 will remove 6 bits but add an addition of a value $< q$. A design with an optimized double K-reduction (ii) requires a single conditional addition but utilizes slightly more expensive K-reduction as opposed to LUT-reduction. In total, it requires 56 LUTs. A design (iii) implementing K-reduction first, followed by LUT reduction still requires a conditional subtraction and requires 60 LUTs. Finally, our best optimized design (iv) takes only 49 LUTs as it utilizes LUT based reduction for the first stage (1) and K-reduction for the second (2), requiring only a single conditional addition for the final stage (3). It can be argued that lazy reduction obviates the

TABLE II
K-RED + LUT-RED COMBINATIONS AREA COST

	LUT first	K-red first
LUT second	72 LUTs	60 LUTs
K-red second	49 LUTs	56 LUTs

need for a full reduction to a value between 0 and 3329. One could for instance allow the output to be all values of 12 bits, including values between 0 and $2^{12} = 4096$. However, this merely shifts the problem of modular reduction to the modular addition and subtraction of the butterfly. When adding two 12 bit numbers modulo q, it is not enough to simply subtract the modulus $q = 3329$, as the sum can have a maximum value of $2^{13} - 2$, requiring a subtraction by $2q$ to reduce below 12 bit. Therefore an extra conditional subtraction unit would be required in the addition and subtraction if lazy reduction is utilized for the modular multiplication. In other words, one conditional subtraction is saved at the cost of adding two extra conditional subtractions, which is not a worthwhile trade-off.

The following section provides the synthesis results of all state-of-the-art design techniques for modular multiplication on a Xilinx 7-Series FPGA platform. The state-of-the-art designs are open sourced [4], so that the reader can run his own synthesis on his desired hardware platform.

[4]https://github.com/axytho/KyberButterflyCollection

VI. Results

In this section, the best algorithms from each category of modular multiplication are implemented to verify their resource usage. Moreover, this provides an opportunity to verify that the algorithms correctly implement a Kyber butterfly. Xing et al. [XL21] and Yaman et al. [YMÖS21] provided open-source implementations of their designs, whereas Nguyen et al.'s [NPH23] strategy is implemented from scratch. Ni et al.'s [NKLO23] design was also implemented to provide a K-reduction implementation that does not rely on Karatsuba.

Some implementations target flexibility, whereas others try to minimize area or maximize throughput, which poses challenges for comparison. To keep the comparison fair and to increase the usability of the open-sourced implementations, the butterfly unit of Yaman et al. [YMÖS21] is taken for all implementations. Yaman et al.'s design has been placed in the public domain by its authors and is a hardware implementation of the full Kyber algorithm. To compare the various modular multiplication techniques, only the modular multiplier inside the design is replaced. Yaman et al.'s implementation supports NTT with different numbers of butterflies, guaranteeing that the implementation of the butterfly is independent of the targeted specifications.

The butterfly units were exhaustively tested for all possible twiddle factors (256 possibilities) and all possible inputs ($q = 3329$). Wherever designs did not fully reduce the result and thus offered a misleading area utilization for the modular reduction circuit (see Figure 2 & Figure 3), the necessary reduction logic was added.

A. Area Results

TABLE III
Butterfly Area Usage

Paper	Technique	# LUT	# DSP
Yaman et al. [YMÖS21]	Bit-wise	269	1
Xing and Li [XL21]	Barrett	192	1
Nguyen et al. [GLK22b]	LUT/K-red	342	0
Li et al. (their impl.) [LQYW24]	$K^{1.5}$-red	140	2
[NKLO23] (our improv.)	**LUT4/K-red**	**185**	**1**
[LQYW24] (our improv.)	**$K^{1.5}$-red**	**152**	**1**
Figure 4	**LUT6/K-red**	**151**	**1**

The modular multiplication unit is also synthesized separately in Table IV for a better comparison between the different designs.

TABLE IV
Modular Multiplication Area Usage

Paper	# LUT	# FF	# DSP
Yaman et al. [YMÖS21]	167	1	1
Xing and Li [XL21]	90	1	1
Nguyen et al. [GLK22b]	248	111	0
Li et al. (their impl.) [LQYW24]	38	0	2
Ni et al. (our improv.) [NKLO23]	**83**	**38**	**1**
Li et al. (our improv.) [LQYW24]	**56**	**0**	**1**
Figure 4	**49**	**32**	**1**

VII. The System Level

To measure the area improvement resulting from our butterfly circuit, we replace Yaman et al.'s [YMÖS21] highly optimized modular reduction circuit with our own. Our preliminary findings show that the latency of our design is smaller than Yaman et al.'s. We thus will not consider the critical path in our analysis. The designs were implemented for the same Artix-7 FPGA (xc7a200tffg1156-3) that Yaman et al. utilized for 1, 4 and 16 butterflies. The results, generated by Vivado 2020.2, are in Table V. The area reduction predictably improves as butterfly units account for a greater percentage of the area on the circuit. For a Kyber implementation utilizing one butterfly, ours takes 14.7% less LUTs whereas a full Kyber implementation utilizing 16 requires 22.9% less LUTs. This area improvement can be achieved with very little effort by the reader: simply replacing the modular multiplication in the butterfly suffices.

TABLE V
Kyber Area usage

# PE	Mod Mult.	# LUT	# FF	# BRAM	# DSP
1	[YMÖS21]	809	388	2.5	1
1	**Ours**	**690**	**419**	**2.5**	**1**
4	[YMÖS21]	2238	903	9	4
4	**Ours**	**1782**	**1027**	**9**	**4**
16	[YMÖS21]	8293	3106	33	16
16	**Ours**	**6393**	**3604**	**33**	**16**

VIII. Conclusion

This paper provides a highly optimized modular multiplication circuit for Kyber modular multiplication. The techniques discussed here are partially applicable to other PQC schemes, but their relative costs will change depending on the structure of the modulus. The discussion on modular arithmetic and the implementation results make it clear that future hardware acceleration of Kyber should maximally exploit the fixed nature of the modulus for the modular multiplication arithmetic. On 7-Series FPGAs, by far the most commonly studied hardware architecture, a merging of K-reduction with LUT-reduction has the lowest area.
As for future work, it should be noted that further improvements on modular reduction in FPGAs can only reduce its cost by up to 49 LUTs as this is the current cost of the modular multiplication circuit without the multiplier. In contrast, a Karatsuba multiplier has a logic cost of ≈ 211 LUTs. Short of a significant decrease in the arithmetic cost of 12 bit multiplication, Amdahl's law prohibits serious gains from optimizing the modular reduction circuit. In conclusion, future Kyber hardware optimizations must be made outside the butterfly unit.

Acknowledgements

This work was partly supported by CyberSecurity Research Flanders (VR20192203), by the European Commission through the Horizon 2020 Belfort ERC Advanced Grant 101020005, and by DARPA under contract No. HR0011-21-C-0034 DARPA DPRIVE BASALISC.

References

[BAK21a] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari Kermani. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*, pages 94–101. IEEE, 2021.

[BAK21b] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari Kermani. A monolithic hardware implementation of Kyber: Comparing apples to apples in PQC candidates. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings*, volume 12912 of *Lecture Notes in Computer Science*, pages 108–126. Springer, 2021.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based kems. Cryptology ePrint Archive, Paper 2022/158, 2022. https://eprint.iacr.org/2022/158.

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of Kyber on cortex-M4. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 209–228. Springer, Heidelberg, July 2019.

[BLP$^+$23] Mojtaba Bisheh-Niasar, Daniel Lo, Anjana Parthasarathy, Blake Pelton, Bharat Pillilli, and Bryan Kelly. PQC cloudization: Rapid prototyping of scalable NTT/INTT architecture to accelerate Kyber. *IACR Cryptol. ePrint Arch.*, page 1038, 2023.

[BPC19] Utsav Banerjee, Abhishek Pathak, and Anantha P. Chandrakasan. An energy-efficient configurable lattice cryptography processor for the quantum-secure internet of things. In *IEEE International Solid- State Circuits Conference, ISSCC 2019, San Francisco, CA, USA, February 17-21, 2019*, pages 46–48. IEEE, 2019.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):17–61, 2019.

[CCG00] E. Chu, E.W. Chu, and A. George. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. Computational Mathematics Series. CRC-Press, 2000.

[Div16] NIST Computer Security Division. Post-quantum cryptography standardization. https://csrc.nist.gov/projects/post-quantum-cryptography, 2016. [Online; accessed 20-December-2023].

[Div23a] NIST Computer Security Division. Comments requested on three draft FIPS for post-quantum cryptography. https://csrc.nist.gov/news/2023/three-draft-fips-for-post-quantum-cryptography, 2023. [Online; accessed 30-October-2023].

[Div23b] NIST Computer Security Division. FIPS 203 (draft): Module-lattice-based key-encapsulation mechanism standard. https://csrc.nist.gov/pubs/fips/203/ipd, 2023. [Online; accessed 30-October-2023].

[Div23c] NIST Computer Security Division. FIPS 204 (draft): Module-lattice-based digital signature standard. https://csrc.nist.gov/pubs/fips/204/ipd, 2023. [Online; accessed 30-October-2023].

[DMG23] Viet Ba Dang, Kamyar Mohajerani, and Kris Gaj. High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber. *IEEE Trans. Computers*, 72(2):306–320, 2023.

[GLK22a] Wenbo Guo, Shuguo Li, and Liang Kong. An efficient implementation of KYBER. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1562–1566, 2022.

[GLK22b] Wenbo Guo, Shuguo Li, and Liang Kong. An efficient implementation of KYBER. *IEEE Trans. Circuits Syst. II Express Briefs*, 69(3):1562–1566, 2022.

[JGCS21] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, and Somitra Kumar Sanadhya. A configurable CRYSTALS-Kyber hardware implementation with side-channel protection. *ACM Transactions on Embedded Computing Systems*, 2021.

[Kar95] Anatolii Alexeevich Karatsuba. The complexity of computations. *Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation*, 211:169–183, 1995.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the Number Theoretic Transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139, 2016.

[LQYW24] Lu Li, Guofeng Qin, Yang Yu, and Weijia Wang. Compact instruction set extensions for kyber. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(3):756–760, 2024.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, May 2019.

[NKLO23] Ziying Ni, Ayesha Khalid, Weiqiang Liu, and Maire O'Neill. Towards a lightweight CRYSTALS-Kyber in fpgas: an ultra-lightweight BRAM-free NTT core. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, July 2023.

[NPH23] Trong-Hung Nguyen, Cong-Kha Pham, and Trong-Thuc Hoang. A high-efficiency modular multiplication digital signal processing for lattice-based post-quantum cryptography. *Cryptography*, 7(4), 2023.

[P$^+$20] T. Prest et al. FALCON: Technical report, national institute of standards and technology. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions, 2020. [Online; accessed 20-December-2023].

[POG15] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2015.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography. *IACR Cryptol. ePrint Arch.*, page 39, 2018.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[SS23] Raziyeh Salarifard and Hadi Soleimany. Efficient accelerator for NTT-based polynomial multiplication. *IACR Cryptol. ePrint Arch.*, page 686, 2023.

[XHY$^+$20] Guozhu Xin, Jun Han, Tianyu Yin, Yuchao Zhou, Jianwei Yang, Xu Cheng, and Xiaoyang Zeng. VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 67-I(8):2672–2684, 2020.

[XL21] Yufei Xing and Shuguo Li. A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):328–356, 2021.

[YMÖS21] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savas. A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 1020–1025. IEEE, 2021.

[ZLZ$^+$22] Qingru Zeng, Quanxin Li, Baoze Zhao, Han Jiao, and Yihua Huang. Hardware design and implementation of post-quantum cryptography Kyber. In *IEEE High Performance Extreme Computing Conference, HPEC 2022, Waltham, MA, USA, September 19-23, 2022*, pages 1–6. IEEE, 2022.