VHDL-based implementation of CRYSTALS-Kyber components on FPGA

P. Jedlicka, J. Hajny

Department of Telecommunications, Faculty of Electrical Engineering and Communication,
Brno University of Technology, Technicka 12, 616 00 Brno, Czech Republic

E-mail: xjedli23@vut.cz, hajny@vut.cz

Abstract—CRYSTALS-Kyber is one of the finalists of the National Institute of Standards and Technology (NIST) post-quantum cryptography competition. In this paper, we deal with effective hardware-accelerated implementations of components intended for the use in the FPGA (*Field Programmable Gate Array*) implementation of the above-mentioned lattice-based cryptography scheme. The discussed components are NTT (*Number Theoretic Transform*), inverse NTT (NTT^{-1}), CBD (*Centered Binomial Distribution*) and the Parse Algorithm (shortly Parse). The improved implementation of NTT (NTT^{-1}) requires 1189 (1568) Look-Up Tables (LUTs), 1469 (2161) Flip-Flops (FFs), 28 (50) Digital Signal Processing blocks (DSPs) and 1.5 (1.5) Block Memories (BRAMs). The latency of the design is 322 (334) clock cycles at the frequency 637 MHz which makes the presented NTT (NTT^{-1}) implementations to be currently the fastest ones. The implementations of the sampling functions (CBD and Parse) requires less than 100 LUTs and FFs with maximum latency 5 clock cycles at the frequencies over 700 Mhz. All implementations has been synthesized for the Xilinx Virtex UltraScale+ architecture.

Keywords—NTT, CBD, Parse, VHDL, FPGA, Kyber

1. INTRODUCTION

In recent years, there has been a significant progress in the field of quantum computing research. Using special algorithms, quantum computers are able to solve hard problems such as the integer factorization and the discrete logarithm much faster than conventional computers. Many of the widely used protocols with the public key that are considered to be secure are based just on the two problems mentioned above and thus they will become breakable if a quantum computer comprising a sufficient number of qubits is built. As a consequence of these facts, intensive research in the field of so-called post-quantum cryptography (PQC) has started. The PQC represents cryptosystems that are not vulnerable to quantum computer attacks. According to the currently ongoing NIST post-quantum standardization process, one of the most promising PQC cryptosystems is CRYSTALS-Kyber (shortly Kyber), a key-encapsulation mechanism (KEM) and public-key encryption algorithm.

This work presents optimized VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) implementations of some crutial Kyber components such as NTT, inverse NTT and sampling functions including CBD and Parse. All implemented algorithms has been designed according to the reference implementation of Kyber in the C programming language [1].

2. RELATED WORK

Several partial or whole hardware implementations of Kyber are currently available. To the best of the author's knowledge, there is only one complete hardware implementation presented by Dang *et al.*, 2021 [2] who reached the frequency of 450 MHz on high performance devices such as Xilinx Zynq UltraScale+. Ricci *et al.* [3] present the implementation of Kyber components including NTT, inverse NTT and components based on the Keccak algorithm (SHA3-256, SHA3-512, SHAKE128, SHAKE256) using Xilinx Virtex UltraScale+ architecture. Chen *et al.* [4] present the implementation of NTT using lightweight platforms such as Xilinx Artix 7 and Xilinx Spartan 6. On the other hand, Dang *et al.*, 2020 [5] present the results of the implementation of NTT using high performance Zynq UltraScale+. Unfortunately, any of the authors mentioned above does not show the implementation results of inverse NTT. Implementation results of the sampling functions (CBD and Parse) are also not presented.

3. PRELIMINARIES

3.1. Sampling from a binomial distribution

To sample noise, Kyber uses a function CBD (Centered Binomial Distribution) described by the algorithm 1 [1]. In addition to the input byte array, there is also a parameter η at the input of the function. The double of this parameter (2η) defines the number of the input bits needed for the calculation of one coefficient of the output polynomial. Kyber uses $\eta = 2$ and $\eta = 3$.

Algorithm 1 CBD

Input: Byte array $B = b_0, b_1, b_2, ..., b_{64\eta-1}$, parameter η
Output: Polynomial $f \in \mathcal{R}_q$
$(\beta_0, ..., \beta_{512\eta-1}) :=$ BytesToBits(B)
for i from 0 255 **do**
    $a := \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$
    $b := \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$
    $f_i := a - b$
return $f_0 + f_1X + f_2X^2 + ... + f_{255}X^{255}$

3.2. Uniform sampling

To sample elements in $\mathcal{R}_q$ that are statistically close to a uniformly random distribution, Kyber uses a function Parse described by the algorithm 2 [1].

Algorithm 2 Parse

Input: Byte stream $B = b_0, b_1, b_2, ...$
Output: NTT-representation $\hat{a} \in \mathcal{R}_q$ of $a \in \mathcal{R}_q$
$i := 0$, $j := 0$
while $j < n$ **do**
    $d_1 := b_i + 256 \cdot (b_{i+1} mod 16)$
    $d_2 := b_{i+1}/16 + 16 \cdot b_{i+2}$
    if $d_1 < q$ **then**
        $\hat{a}_j := d_1$
        $j := j + 1$
    if $d_2 < q$ and $j < n$ **then**
        $\hat{a}_j := d_2$
        $j := j + 1$
    $i := i + 3$
return $\hat{a}_0 + \hat{a}_1X + ... + \hat{a}_{n-1}X^{n-1}$

3.3. Number theoretic transform

The Number theoretic transform (NTT) is a generalization of the discrete Fourier transform, which is carried out in a finite field instead of complex numbers. Assuming a polynomial $f(x)$ with coefficients $(f_0, f_1, ..., f_{n-1})$ and their representation in the NTT domain $(\hat{f_0}, \hat{f_1}, ..., \hat{f}_{n-1})$, the transformations can be expressed by the following formulas [6]:

$$\hat{f_i} = \sum_{j=0}^{n-1} f_i r^{ij} \mathrm{mod}(q), i = 0, 1, ..., n-1,$$

$$f_i = \frac{1}{n} \sum_{j=0}^{n-1} \hat{f_i} r^{-ij} \mathrm{mod}(q), i = 0, 1, ..., n-1,$$

where $r = 17$ is the $2n$-th primitive root of unity in modulo q arithmetic [1].

Kyber performs operations in the polynomial ring $\mathcal{R}_q = \mathbb{Z}[x]_q/(x^n+1)$ where n and q are two integers, i.e. $n = 256$ and $q = 3329$. Using the textbook definition for the polynomial multiplication, the achieved

time complexity is $O(n^2)$. However, the polynomial multiplication accelerated by the properties of the NTT reduces the time complexity to $O(n \log n)$. The multiplication of polynomials $f(x)$ and $g(x)$) using the NTT can be expressed by the following formula [1]:

$$f(x) \times g(x) = NTT^{-1}(NTT(f(x)) \odot NTT(g(x))),$$

where $\odot$ is the point-wise multiplication. The NTT can be used only for the $q = 1 \bmod (2n)$ [1].

4. VHDL IMPLEMENTATION

The design has been implemented on a Virtex UltraScale+ FPGA from Xilinx in Vivado 2019.1 using the VHDL language. As the selected hardware platform and the series of the mounted FPGA suggests, the main optimization goal of the design is speed rather than area and the chosen implementation strategy corresponds to this fact. To reach as high clock frequency as possible, pipelined processing that reduces the size of combinational logic between two FFs is applied to the individual functional blocks. DSP blocks are used for almost all arithmetical operations, only incrementations or multiplications by a constant power of two are excluded. Due to the speed optimization, DSP blocks are set for the maximum latency whereas the throughput is still one output per one clock cycle. All the components follow the reference implementation [1] and their implementations are described below.

4.1. CBD

The CBD algorithm has been implemented using only Look-Up Tables and Flip-Flops. No DSP block has been utilized although there are mathematical operations. The reason of the absence of DSP blocks is a low overall number of input bits used for the calculation of one output coefficient. For $\eta = 2$, respectively $\eta = 3$, there are only 4, respectively 6, input bits. The mathematical operation based on such a small input operands can be ensured by single 6-input Look-Up Tables placed in parallel. As can be seen from the figure 1, the component uses standard AXIStream interfaces. The input data is split into η-bit fields which serves as the inputs for the parallel calculation of output coefficients. For $\eta = 2$, respectively $\eta = 3$, the input data bus has 32, respectively 24, bits and the component performs parallel computation of 8, respectively 4 output coefficients. The latency is 3 clock cycles and the throughput is one output coefficient set per one clock cycle.

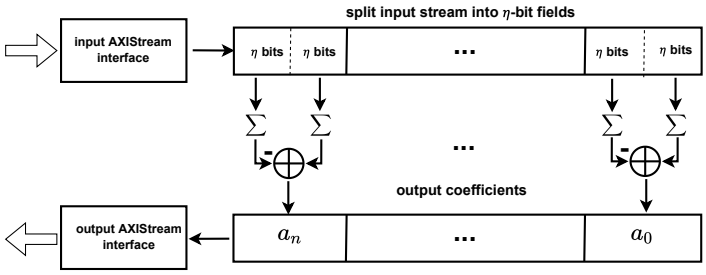

Figure 1: The block scheme of the CBD implementation

4.2. Parse

The Parse function has been implemented by applying the same meyhodology as in the case of the CBD algorithm including no DSP utilization and using AXIStream interfaces. The latency is 5 clock cycles and the throughput is two output coefficients set one clock cycle. The block scheme is shown in Figure 2. Invalid coefficients which do not correspond to the required range are indicated by AXIStream TKEEP signal and discarded.

4.3. NTT and Inverse NTT

The presented implementation of NTT and NTT^{-1} is based on our previous implementation described in Ricci *et. al* [3]. The NTT implementation whose block scheme is shown in Figure 3 uses 4 butterflies running in a 2x2 arrangement which leads to the computation of two iterations in parallel with 2 butterflies in each of them. Butterflies utilize Montgomery and Barrett reduction algorithm for modular reduction after multiplications. Except for input coefficients, the butterflies are supplied with the roots of unity [1]

[1] In relation to the Fast Fourier Transform also known as twiddle factors.

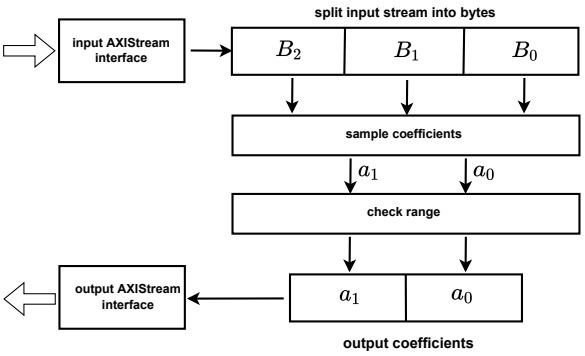

Figure 2: The block scheme of the Parse implementation

which are saved in the Read-Only memory.

The presented implementation has an improvement in added possibility for a parallel computation of two input polynomials while keeping almost the same hardware utilization. In the original version, there were delays between sets of the parallel iterations because of the latency of the butterfly structure (26 clock cycles). These delays has been eliminated by an immediate computation of the same set of parallel iterations of the second polynomial which is not dependent on the iteration results of the first polynomial. After finishing the iteration set of the second polynomial, the computation is switched back on the following iterations of the first polynomial. These feature saved approximately 80 clock cycles of the latency per each input polynomial.

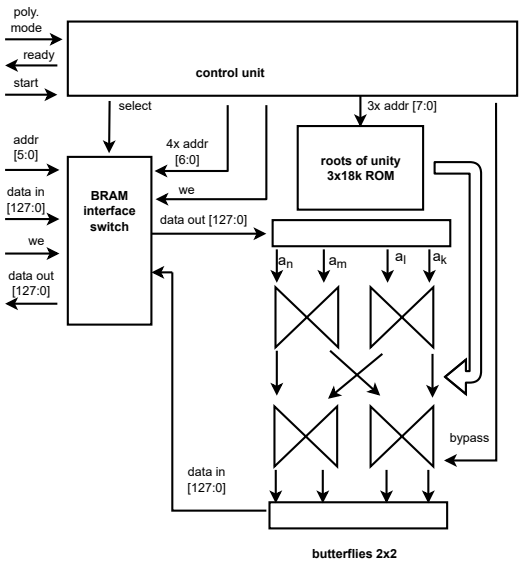

Figure 3: The block scheme of the NTT implementation

5. IMPLEMENTATION RESULTS AND COMPARISON

Table I depicts the implementation results for the presented designs and the aforementioned previous works whereas the comparison can be made only for the NTT because the other works do not mention the implementation of the NTT^{-1} and the sampling functions. The proposed HDL-based design needs on average 2 times less hardware resources and is 4 times faster than the implementation of Dang *et al.*, 2020 [5]. In comparison to the implementation of Chen *et al.* [4], our design needs on average 5 times more hardware resources but is much better optimized in terms of the clock frequency and the latency which makes our design almost 30 times faster. In comparison with our previous design (Ricci *et al.*, 2021 [3]), we have almost the same hardware utilization but, thanks to multiple polynomial calculation, we have decreased the latency by 83 clock cycles per one polynomial. Sampling functions are much less complex and thus their utilization is minimal when compared to NTT implementations.

Table I: The implementation results and the comparison with the other implementations

	LUT	FF	DSP	BRAM	LUTRAM	Frequency [MHz]	Latency [cycles]
NTT							
This work	1189	1469	28	1.5	302	637	322
Ricci *et al.*, 2021 [3]	1107	1407	28	3.5	-	637	405
Dang *et al.*, 2020 [5]	2040	3223	24	5	-	500	1271
Chen *et al.*, 2020 [4]	442	237	1	1.5	-	136	2055
Inverse NTT							
This work	1568	2161	50	1.5	368	637	334
Parse							
This work	67	94	0	0	0	702	5
CBD							
This work	56	89	0	0	0	709	3

6. CONCLUSION

To the best of the author's knowledge, we proposed in this work the fastest and most effective hardware implementations of NTT and inverse NTT which are the essential and most complex functional blocks of the lattice-based PQC schemes. The NTT and inverse NTT components are able to perform over 1,900,000 transformations per second with the clock frequency 637 MHz which is the highest reached value in comparison with the implementations of other authors whose clock frequencies are in the range between 136 MHz and 500 MHz. We also introduced highly optimized implementations of the sampling functions CBD and Parse with the clock frequency exceeding 700 MHz. These implementation results of the crucial Kyber components are great prerequisite for a highly effective and the fastest implementation of the whole cryptosystem with the clock frequency exceeding 600 MHz.

ACKNOWLEDGMENT

The presented research was financed by the Ministry of Interior under grant no. VJ02010010.

REFERENCES

[1] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler and D. Stehlé, "CRYSTALS-Kyber algorithm specifications and supporting documentation," *NIST PQC Round*, vol. 3, 2021.

[2] V. B. Dang, K. Mohajerani and K. Gaj, "High-Speed Hardware Architectures and Fair FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *NIST PQC Round*, vol. 3, 2021.

[3] S. Ricci, P. Jedlicka, P. Cibik, P. Dzurenda, L. Malina and J. Hajny, "Towards CRYSTALS-Kyber VHDL Implementation," In Proceedings of the 18th International Conference on Security and Cryptography - SECRYPT, 2021, pages 760-765.

[4] Z. Chen, Y. Ma, T. Chen, J. Lin and J. Jing, "Towards efficient kyber on fpgas: A processor for vector of polynomials," In 25th Asia and South Pacific Design Automation Conference (ASP-DAC), 2020, pages 247-252.

[5] V. B. Dang, F. Farahmad, M. Andrzejczak, K. Mohajerani, D. T. Nguyen and K. Gaj, "Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," Cryptology ePrint Archive: Report 2020/795, 2020.

[6] H. N. Nejatollahi, S. Dutt, F. Ray, I. Regazzoni, R. Banerjee, "Post-Quantum Lattice-Based Cryptography Implementations," In ACM Computing Surveys, 2019, pages 1-41.