VHDL-BASED IMPLEMENTATION OF NTT ON FPGA

Petr Jedlička
Doctoral Degree Programme (1), FEEC BUT
E-mail: xjedli23@stud.feec.vutbr.cz

Supervised by: Jan Hajný
E-mail: hajny@feec.vutbr.cz

Abstract: This paper is focused on the effective hardware-accelerated implementation of NTT (*Number Theoretic Transform*) and inverse NTT (NTT^{-1}) on FPGA (*Field Programmable Gate Array*). The discussed implementation is intended for the use in the lattice-based cryptography schemes, e.g. CRYSTALS-Dilithium digital signature scheme which is one of the finalists of the third round in the post-quantum standardization process under the auspices of NIST (*The National Institute of Standards and Technology*). The implementation of NTT (NTT^{-1}) requires 1798 (2547) Look-Up Tables (LUTs), 2532 (3889) Flip-Flops (FFs) and 48 (84) Digital Signal Processing blocks (DSPs). The latency of the design is 502 (517) clock cycles at the frequency 637 MHz on Xilinx Virtex UltraScale+ architecture which makes the presented implementation to be currently the fastest one. Regarding the inverse NTT, this is the first implementation at all.

Keywords: NTT, VHDL, FPGA, Dilithium, Montgomery reduction

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 cryptosystems 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, ones of the most promising PQC cryptosystems are the lattice-based ones. The algorithms of these cryptosystems perform mathematical operations over polynomials in modular arithmetic such as additions and multiplications that can be accelerated using the NTT algorithm.

This work presents original and optimized VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) implementations of NTT and inverse NTT according to the reference implementation of the CRYSTALS-Dilithium digital signature scheme (shortly Dilithium) in the C programming language [1].

2 RELATED WORK

To the best of the author's knowledge, there are currently only two hardware implementations of the NTT that are targeted to Dilithium. However, neither of them mention the implementation of the inverse NTT. Nguyen *et al.* [2] compare the efficiency of the HDL (*Hardware Description Language*) and HLS (*High Level Synthesis*) implementations on the UltraScale+ architecture. Their implementation does not fully meet the reference implementation because they replaced the Montgomery reductions by so called K-reductions. As a consequence of this modification, it is not possible to verify the correctness of such design by simple comparison with the reference NTT implementation whose

output is in the Montgomery domain. Unfortunately, they do not mention the way how they verified their design. In the second article, Nejatollahi *et al.* [3] present their NTT implementation on the Zynq UltraScale+ architecture using the HLS method which is usually much less effective in terms of speed and pipelined processing.

3 POLYNOMIAL MULTIPLICATION AND NUMBER THEORETIC TRANSFORM

Dilithium performs operations in the polynomial ring $\mathcal{R}_q = Z[x]_q/(x^n+1)$ where n and q are two integers, i.e. $n = 256$ and $q = 8380417 = 2^{23} - 2^{13} + 1$. 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 [5]:

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

where $\odot$ is the point-wise multiplication. The NTT can be used only for the $q = 1 \bmod (2n)$ [5].

3.1 NTT

The 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 [4]:

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

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

where $r = 1753$ is the $2n$-th primitive root of unity in modulo q arithmetic [5].

3.2 MONTGOMERY REDUCTION

Montgomery reduction is the computation of $xR^{-1} \bmod m$. The computation of $x \bmod m$ is performed in two successive Montgomery reductions and a one intermediate multiplication by $R^2 \bmod m$. The algorithm for one Montgomery reduction $u = xR^{-1} \bmod m$ is following [4]:

$$\tilde{m} = -m^{-1}$$

$$\tilde{q} = x\tilde{m} \bmod R$$

$$u = \frac{x + \tilde{q}m}{R}$$

4 VHDL IMPLEMENTATION

The design has been implemented on a Virtex UltraScale+ FPGA from Xilinx in Vivado 2017.4.1 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. Furthermore, 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. At first, individual functional sub-blocks has been implemented. In next step,

NTT and NTT^{-1} components has been implemented whereas the sub-blocks has been integrated into those two components. All the components follow the reference implementation [1] and are described below.

4.1

The c uction of 64-bit values
to 32- ut per one clock cycle.
The p gure 1.

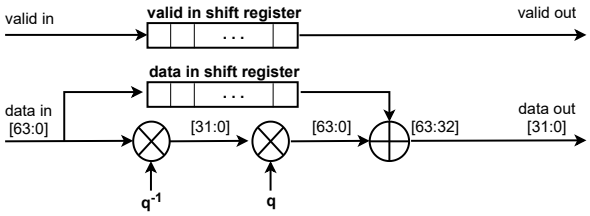

Figure 1: The block scheme of the Montgomery reduce implementation

There are two input signals, `valid_in` and `data_in`, and two corresponding output signals, `valid_out` and `data_out`. The `valid` signals indicates the valid data on the input or output data bus. This signal propagates through a shift register whose length has to be equal to the latency of the component. Whereas there is the maximum possible throughput at data bus there is no need for a backpressure signal. All the arithmetic operations run in parallel, are pipelined and use DSP blocks. There is another shift register used for the input 64-bit value and its length corresponds to the overall latency of the multipliers. Thanks to the appropriately chosen value of the parameter R, the modulo operations and divisions can be done effectively by simple dropping unnecessary bits from the buses.

4.2 BUTTERFLIES

The NTT and NTT^{-1} butterflies has been implemented by applying the same methodology as in the case of Montgomery reduce. The latency is in both cases 24 clock cycles and the throughput is one output per one clock cycle. The algorithm exactly follows the reference implementation [1]. The blo

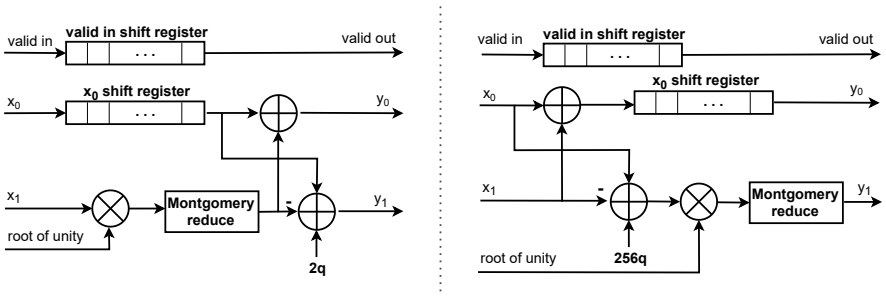

Figure 2: The block schemes of the implementation of the butterflies (left: NTT, right: NTT^{-1})

4.3 NTT AND INVERSE NTT

The described implementation is based on the FFT (*Fast Fourier Transform*) algorithm with decimation in the frequency domain as it is handled in the reference implementation [1]. The transformation takes place over 256 samples, which corresponds to the calculation performed in 8 iterations ($256 = 2^8$). Within each iteration, 128 partial transformations are calculated using butterflies. The difference in the order of the coefficients entering this partial transformation is $\frac{128}{2^{n-1}}$, where n denotes the order of the iteration with indexing starting from 1. To compromise between the speed of calculation and hardware resources, the parallelization of the calculation using 4 butterflies in a 2x2

arrangement is deployed. This means a calculation in two iterations at the same time with two butterflies in each of them. The butterflies in the first iteration have to be shifted by a value corresponding to the required difference in the coefficients of the following iteration. The reason for spreading the calculation between two iterations is to reduce the number of output coefficients (for one iteration) and thus minimize the number of Block Random Access Memory (BRAM) for storing intermediate results too. Each block memory has only two ports which can be accessed at the same time and therefore it is needed N block memories to store $2N$ values in parallel regardless the memory size actually needed. Another reason of the 2x2 arrangement is to reduce the number of interludes between iterations. The order of reading the input intermediate results differs depending on the current iteration and therefore each iteration can start after storing all intermediate results from the previous one which is delayed du

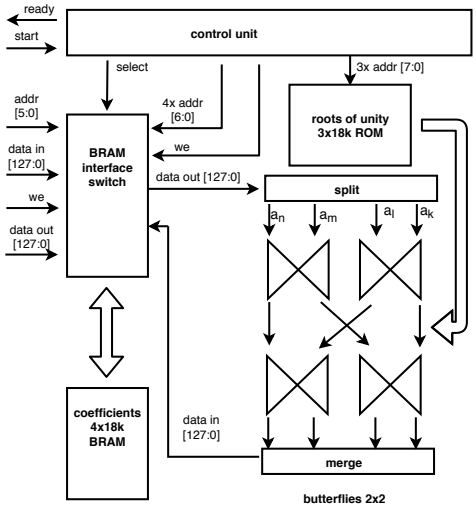

Figure 3: The block scheme of the NTT implementation

The block diagram of the NTT component is shown in Figure 3. Four BRAMs are used to store input values, intermediate results and output values. In the inactive mode, the interface of these memories is switched by the control unit to the component interface, then inputs and outputs can be written or read. During calculating the transformation, the memory interface is made available to the control unit, which sets the addresses, and to the computational structure consisting of butterflies in a 2x2 layout connected to the data buses. Four coefficients are transmitted in parallel over the data bus, which are divided between the inputs of the butterflies and merged again at the output. The last block is the ROM memory with the roots of the unit equation (so-called roots of unity). [5]

The design of the Inverse NTT (NTT^{-1}) component is almost identical to the NTT scheme. Only the iterations are performed in the reverse order, and thus the distribution of the butterflies in the first and second parallel iterations is inverse. In contrast to NTT, the Montgomery reduction of the output coefficients is performed at the end of the whole calculation. [5]

5 IMPLEMENTATION RESULTS AND COMPARISON

Table 1 depicts the implementation results for the presented design 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}. The proposed HDL-based design needs on average 20 times less hardware resources and is 6 times faster than the implementation of Nejatollahi *et al.* [3]. In comparison to the implementations (HLS-based and HDL-based) of Nguyen *et al.* [2], their designs have comparable results of the hardware utilization but they are not so much optimized in terms of the clock frequency and the speed as the presented ones. The presented components are able to run

at the clock frequency 637 MHz while the value reached by Nguyen *et al.* [3] is only 445 MHz. It is remarkable that Nguyen *et al.* [2] reached exactly the same latency for the both implementation strategies.

Table 1: The implementation results and the comparison with the other implementations

	LUT	**FF**	**DSP**	**BRAM**	**LUTRAM**	**Frequency** [MHz]	**Latency** [cycles]
NTT							
This work	1798	2532	48	3.5	438	637	502
Nejatollahi *et al.* [3]	47332	38108	1282	2	-	199	1058
Nguyen *et al.* [2], HDL	1899	2041	8	2	-	445	294
Nguyen *et al.* [2], HLS	1977	2329	8	2	-	434	294
Inverse NTT							
This work	2547	3889	84	3.5	762	637	517

6 CONCLUSION

To the best of the author's knowledge, this work introduced the fastest and most effective hardware implementation of the NTT and the first implementation of the inverse NTT which are the essential and most complex functional blocks of the lattice-based PQC schemes whose implementation on FPGA will be the next step. Using pipelined processing, the components are able to perform over 1,200,000 transformations per second with the clock frequency 637 MHz which is the highest reached value in comparison with the previous implementations whose clock frequencies are in the range between 199 MHz and 445 MHz.

ACKNOWLEDGEMENT

The presented research was financed by the Ministry of Interior under grant no. VJ01010008.

REFERENCES

[1] DUCAS, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 238-268 (2018). Available from: doi:10.13154/tches.v2018.i1.238-268

[2] NGUYEN, Duc Tri, Viet B. DANG, Kris GAJ. A High-Level Synthesis Approach to the Software/Hardware Codesign of NTT-Based Post-Quantum Cryptography Algorithms. In: 2019 International Conference on Field-Programmable Technology (ICFPT) [online]. IEEE, 2019, 2019, s. 371-374 [cit. 2021-03-10]. ISBN 978-1-7281-2943-3. Available from: doi:10.1109/ICFPT47387.2019.00070

[3] NEJATOLLAHI, Hamid, Sina SHAHHOSSEINI, Rosario CAMMAROTA, Nikil DUTT. Exploring Energy Efficient Quantum-resistant Signal Processing Using Array Processors. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) [online]. IEEE, 2020, 2020, s. 1539-1543 [cit. 2021-03-10]. ISBN 978-1-5090-6631-5. Available from: doi:10.1109/ICASSP40776.2020.9053653

[4] NEJATOLLAHI, Hamid, Nikil DUTT, Sandip RAY, Francesco REGAZZONI, Indranil BANERJEE, Rosario CAMMAROTA. Post-Quantum Lattice-Based Cryptography Implementations. In: ACM Computing Surveys [online]. 2019, s. 1-41 [cit. 2021-03-10]. ISSN 0360-0300. Available from: doi:10.1145/3292548

[5] RICCI, Sara, Lukas MALINA, Petr JEDLICKA, David SMEKAL, Jan HAJNY, Petr CIBIK, Patrik DOBIAS. Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs [online]. 2021, [cit. 2021-03-10]. Available from: https://eprint.iacr.org/2021/108