

The Low-Area FPGA Design for the Post-Quantum Cryptography Proposal Round5

Michał Andrzejczak

Military University of Technology in Warsaw
ul. gen. Sylwestra Kaliskiego 2, 00-908 Warszawa, Poland
Email: michal.andrzejczak@wat.edu.pl

Abstract—Post-Quantum Cryptography (PQC) is getting attention recently. The main reason of this situation is the announcement by the U.S. National Institute for Standard and Technology (NIST) about an opening of the standardization process for PQC. Recently NIST published a list of submissions qualified to the second round of this process. One of the selected algorithms is Round5, offering a key encapsulation mechanism (KEM) and public key encryption (PKE). Due to high complexity of post-quantum cryptosystems, only a few FPGA implementations have been reported to date. In this paper, we report results for low-area purely-hardware implementation of Round5 targeting low-cost FPGAs.

I. INTRODUCTION

POST-Quantum Cryptography (PQC) is an answer to a threat coming from a full-scale quantum computer able to execute Shor’s algorithm [1]. With this algorithm executed on a quantum computer, currently used public key schemes, such as RSA [2] and elliptic curve cryptosystems, are no longer secure. The U.S. NIST made a step toward mitigating the risk of quantum attacks, by announcing the PQC standardization process [3]. In March 2019, NIST published a list of candidates qualified to the second round of the PQC process [4]. To date, hardware performance of Round 1 candidates was reported for only a small percentage of all submissions.

In this paper, we present the hardware design for low-area implementation of the PQC Round 2 candidate, called Round5. Our design is able to provide both the Key Encapsulation Mechanism (KEM) and Public-Key Encryption (PKE) functionalities. We provide results for all parameter sets of the ring-versions of the respective schemes. Our main goal was to develop the first full-hardware implementation of this PQC submission, able to operate on low-cost FPGAs.

A. Previous work

Due to complexity of designs, there are only several reports for implementations of PQC candidates in FPGAs. From among lattice-based candidates, Howe et al. [5] reported results for *FrodoKEM*. Kuo et al. [6] and Oder and Güneysu [7] independently reported hardware results for *NewHope*. Aforementioned papers targeted Xilinx Artix-7 FPGA.

In [8], Farahmand et al. proposed a new approach for evaluating PQC candidates by using software/hardware (SW/HW) codesign. They proposed to implement only the most time-consuming functions in the FPGA fabric, while the remaining

parts of the algorithms are implemented in software and run on ARM. Using this SW/HW approach, they reported results for four Round 1 NTRU-based proposals.

For other, non lattice-based candidates, Koziel et al. implemented the isogeny-based SIKE [3], [9]. For multivariate schemes, Ferozपुरi and Gaj reported results for Rainbow [10], implemented using Xilinx Virtex-7 and Kintex-7 FPGAs. From among code-based candidates, Wang et al. reported results for Classic McEliece (a.k.a. classical Niederreiter cryptosystem with binary Goppa codes), implemented using Stratix V FPGAs [11], [12].

B. Contribution

In this paper, we present a novel hardware design for the ring version of the Round5 submission to the NIST PQC standardization process. Our design is oriented to be a low-area implementation, able to run on low-end FPGAs. The area-performance trade-off is obtained by our customizable architecture for polynomial multiplication.

II. ROUND5 DESCRIPTION

In the NIST PQC Round 2, there are 26 proposals, with 12 of them belonging to the family of lattice-based schemes. The lattice-based cryptography is a promising option for secure post-quantum KEMs and PKE schemes. It also offers additional novel functionalities, such as homomorphic encryption [13] and identity-based encryption [14].

Round5 [15] comes from merging two other Round 1 candidates: Round2 [16] and HILA5 [17]. The main underlying problem in *Round5* is Generalized Learning With Rounding (GLWR). In GLWR, the problem randomness comes from rounding, and this feature allows avoiding the necessity of implementing a random bit sampler with any specific distribution, which is a requirement in several other proposals. The submission package contains proposals for indistinguishable under chosen plaintext attack (IND-CPA) KEM and indistinguishable under chosen ciphertext attack (IND-CCA) PKE. Both proposed variants come from the Fujisaki-Okamoto (F-O) transformation [18], by using the main building block of *Round5* — *r5_cpa_pke*, the IND-CPA PKE module. Other required modules to perform F-O transformation are a hash function and authenticated encryption with associated data (AEAD). *Round5* comes also in versions with error correcting codes, but our design does not support this functionality.

The package submitted to NIST contains 21 parameters sets, supporting three NIST security levels: 1, 3 and 5. The parameters sets considered in this paper are presented in Table I. We provide results only for the ring version of the schemes, without error correcting codes. The parameter n describes the polynomial degree, p , q , and t refer to moduli used in the design for modular reduction and rounding. All moduli are powers of two and must satisfy the requirement $t < p < q$.

TABLE I
PROPOSED PARAMETER SETS FOR THE RING-BASED VERSION OF THE
ROUND5 PQC CANDIDATE

| R5ND Version | n | $\log_2(q)$ | $\log_2(p)$ | $\log_2(t)$ |
|--------------|------|-------------|-------------|-------------|
| 1KEM_0d | 618 | 11 | 8 | 4 |
| 3KEM_0d | 786 | 13 | 9 | 4 |
| 5KEM_0d | 1018 | 14 | 9 | 4 |
| 1PKE_0d | 586 | 13 | 9 | 4 |
| 3PKE_0d | 852 | 12 | 9 | 5 |
| 5PKE_0d | 1170 | 13 | 9 | 5 |

A. Key generation

In the key generation function, a random seed is expanded by cSHAKE [19]. The use of seed for cSHAKE allows decreasing the size of keys at the expense of an additional cost of expanding the key at the beginning of encryption and decryption. To be compliant with the proposed Hardware API for Post-Quantum Public Key Cryptosystems [20], the key generation function is not a part of the reported design. All long-term keys must be provided to the hardware module from outside, before the main functionality starts.

B. Encryption and decryption

Algorithm 1 contains pseudocode for the IND-CPA PKE. The encryption routine starts with expanding a part of the public key and a random input using the cSHAKE function. In the next step, two polynomial multiplications are performed. For polynomial multiplication, one of the polynomials must be lifted to the other's polynomial ring. After the computations, the result is unlifted back. Next, the result is rounded, which is the source of randomness in the GLWR problem.

Algorithm 1 Round5 Encryption

Require: public key pk , message msg , seed ρ

Ensure: ciphertext (U, v)

- 1: $A \leftarrow \text{Create}_A(pk.\text{sigma})$
 - 2: $R \leftarrow \text{Create}_R(\rho)$
 - 3: $U \leftarrow \text{Unlift}(\text{Lift}(A) * R)$
 - 4: $U \leftarrow \text{Round}(U)$
 - 5: $X \leftarrow \text{Unlift}(\text{Lift}(pk.B) * R)$
 - 6: $X \leftarrow \text{Round}(X)$
 - 7: $v \leftarrow msg + X$
 - 8: **return** (U, v)
-

In Algorithm 2, the pseudocode for decryption is shown. In the first step, the secret key is expanded by using cSHAKE.

Next, only one polynomial multiplication is executed with lifting and unlifting. After the polynomial multiplication, a subtraction from a part of the ciphertext is performed. The last operation is rounding.

Algorithm 2 Round5 Decryption

Require: ciphertext (U, v) , private key sk

Ensure: message m

- 1: $S \leftarrow \text{Create}_S(sk)$
 - 2: $X \leftarrow \text{Unlift}(\text{Lift}(U) * S)$
 - 3: $m \leftarrow v - X$
 - 4: $m \leftarrow \text{Round}(m)$
 - 5: **return** m
-

C. Supporting functions

Round5 uses three supporting functions during encryption and decryption. First, to obtain an NTRU-like polynomial in the polynomial ring $\mathbb{Z}_q[x]/(N_{n+1}(x))$ from the key and the random data, the lift function must be applied before multiplication. The lift function is presented in Algorithm 3.

Algorithm 3 Lift function

Require: a polynomial A of length n

Ensure: an NTRU-like polynomial C of length $n + 1$

- 1: $C_0 \leftarrow -A_0 \pmod{n}$
 - 2: **for** $1 \leq i \leq n - 1$ **do**
 - 3: $C_i \leftarrow A_{i-1} - A_i \pmod{n}$
 - 4: **end for**
 - 5: $C_n \leftarrow A_{n-1}$;
-

The second function required for proper polynomial multiplication is unlifting, presented in Algorithm 4. Unlifting is applied after polynomial multiplication, and performs polynomial division, taking back the polynomial to $\mathbb{Z}_q[x]/(\Phi_{n+1}(x))$.

Algorithm 4 Unlift function

Require: a NTRU-like polynomial A of length $n + 1$

Ensure: a polynomial C of length n

- 1: $C_0 \leftarrow -A_0 \pmod{n}$
 - 2: **for** $1 \leq i \leq n - 1$ **do**
 - 3: $C_i \leftarrow C_{i-1} - A_i \pmod{n}$
 - 4: **end for**
-

The last supporting function is rounding, applied to every polynomial coefficient. It is shown in Algorithm 5. It is responsible for rounding elements to smaller values using the exact approach presented in the submission. This function is a source of randomness in the GLWR problem. It is called twice during encryption and once at the end of decryption. Input arguments of rounding are specified in the proposal's documentation.

III. HARDWARE DESIGN OF ROUND5

We present a low-area architecture of *Round5*. The implementation follows the proposed hardware API for Post-Quantum Public Key Cryptosystems [20]. The top-level view,

Algorithm 5 Rounding function

Require: an element x to round, a proper set of $\{rounding_constant, shift_value, mask\}$

Ensure: a rounded element x

- 1: $x \leftarrow x + rounding_constant$
- 2: $x \leftarrow x \ll shift_value$
- 3: $x \leftarrow x \& mask$
- 4: **return** x

compatible with the aforementioned API, is presented in Fig. 1. The API treats various inputs as public, secret, or random. Thus, three different sets of input ports are used. Each port can handle commands and headers required by the API to control the design.

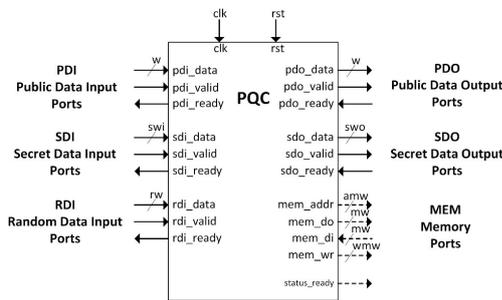


Fig. 1. The PQC Hardware API used in this paper

Going one level down, the architecture of *Round5* is presented in Fig. 2. With the given design, all functionalities of *Round5* are implemented. Each implemented module support all three NIST security levels – 1, 3, and 5. Security level is chosen during compilation and cannot be changed at runtime. The functional modules take input from and write outputs to a shared data bus. The privilege of writing to data bus is granted by the controller’s module.

The main controller is responsible for managing the state of the design and enforcing the proper data flow between modules, depending on a selected operation. It also receives and responds to commands from outside.

The *SHAKE256* module implements the extendable output hash function cSHAKE. It is used for generating pseudo-random polynomials from a given seed. It is also required for the F-O transformation.

The next major component used in our design is the AES-GCM module for authenticated encryption. It is used only in the IND-CCA PKE, as a required part of the F-O transformation. For the IND-CPA KEM, this module can be omitted from the design. The occurrence of AES-GCM is the main difference in the design between the IND-CPA KEM and the IND-CCA PKE versions. In agreement with the F-O transformation, the IND-CCA PKE is build on KEM with additional authenticated encryption.

The most important component is *r5_cpa_enc*, providing all arithmetic operations for IND-CPA PKE, the main building

block of all *Round5* proposals.

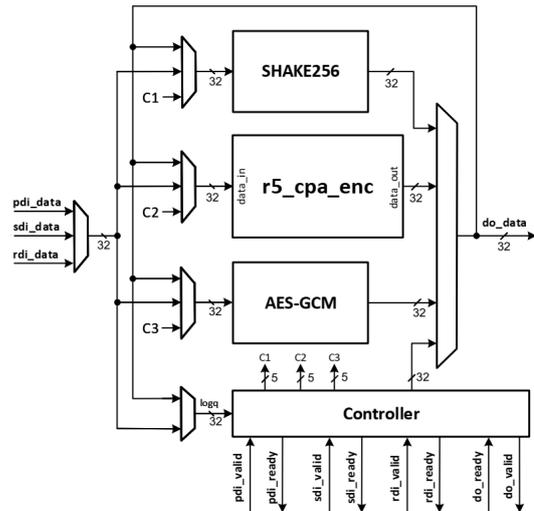


Fig. 2. The proposed top-level architecture of low-area full hardware implementation of *Round5*

In Fig.3 an arithmetic module responsible for public key encryption and decryption is presented. The important part of this module is the controller, located on the left side of the figure. This controller is responsible for receiving commands from outside, managing the state of the module, and providing proper signals for internal sub-modules. Polynomials required for the operation are stored in separate memory banks. Polynomial multiplication is executed twice during encryption. Thus, two memory banks are able to feed the data to the polynomial multiplier for the first argument. There is some additional memory for the ternary polynomial and for the ciphertext. The ternary polynomial is the same for both multiplications executed during encryption. The message is stored in a register and processed at the end of encryption.

Before polynomial multiplication is executed, one of the polynomials must be lifted to the so called NTRU-ring [21]. This is performed by the *LIFT_ELEMENT* module, which performs Algorithm 3. Due to the low-area optimization goal, the proposed module lifts elements sequentially, one element at a time. Lifted elements are written back to memory.

The next step is the polynomial multiplication performed by the *POLY_MUL* module. It requires new data from memory in each cycle. One of the arguments is a coefficient from the lifted polynomial. The second argument is a set of 16 coefficients from the ternary coefficient memory. The module computes results in a sequential fashion, sending further only one coefficient at a time. The first output is ready after n clock cycles, where n is a degree of the polynomials. Every next coefficient is ready after $\lceil n/16 \rceil$ clock cycles.

The computed coefficient follows directly the remaining data path. At first, the coefficient is unlifted to the primary polynomial ring, as shown in Algorithm 4. The next step depends on the operation type. During decryption, subtraction is applied before rounding. During encryption, rounding is

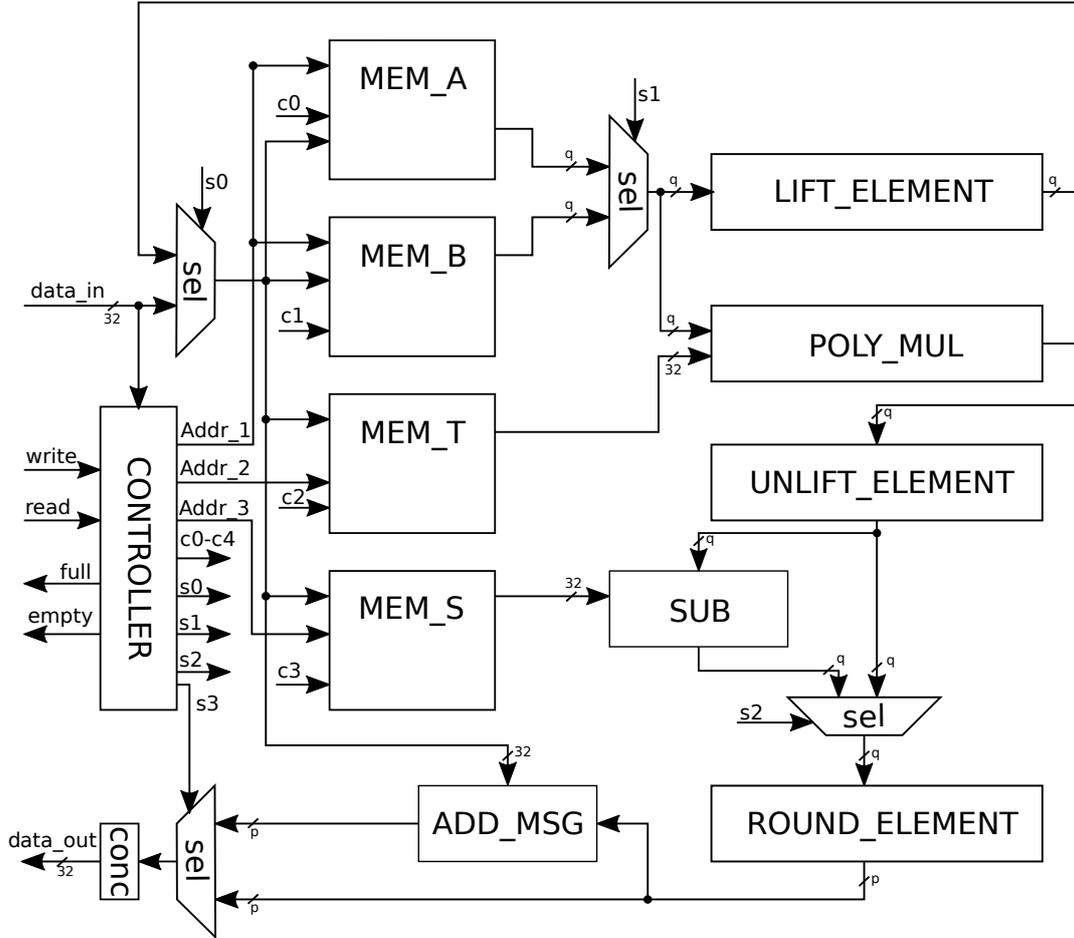


Fig. 3. The arithmetic module for Round5 data encryption and decryption

applied right after unlifting. After rounding, data is stored in the result register directly or with added message bits, depending on the operation type and the state of the encryption process.

Polynomial multiplication has the biggest computational complexity in the *Round5* design. The result is computed by the following formula

$$c_k = \sum_{i+j=k \bmod n} a_i * b_j \quad (1)$$

Due to the form of the polynomial ring and Equation 1, the multiplication can be easily parallelized to speed-up the computations. This is a classical problem of area-performance trade-off, where better performance is achieved by implementing more parallel multipliers, increasing logic usage. Using only one multiplier results in a very large clock cycle latency and slows execution time. On the opposite side of the spectrum, with as many as possible multipliers, the design size is too large to fit in many FPGAs.

We propose a small, in terms of logic utilization, polynomial multiplication module, offering results comparable to those reported for other PQC submissions. Our module executes the

standard schoolbook multiplication with parallel operations. Polynomial multiplication in *Round5* always requires a ternary polynomial and a polynomial with coefficients reduced modulo q , where q is a power of 2. Each coefficient in ternary polynomial is from the set $\{-1, 0, 1\}$, so only two bits are required to store each coefficient value. All required polynomials are stored separately in internal memory. For polynomials from the ring $\mathbb{Z}_q[x]$, each coefficient is accessible directly under different address. Ternary polynomial is stored differently, where one memory cell stores 16 concatenated ternary coefficients. This allows to reduce memory requirements by avoiding only two bits per memory cell utilization. The last memory cell is padded with zeros, if needed.

A new set of coefficients to multiply is loaded from the memory in every clock cycle. The memory address pointers start from the opposite sides, and move in the opposite directions. The ternary polynomial is loaded from the beginning to the end, but the second polynomial is loaded from the last to the first coefficient. The memory pointer for the ternary coefficients is increased by one after each load operation. The second pointer is decreased by the number of parallel multipliers, then the address number is reduced modulo the

polynomial degree. In this scenario, one specific multiplier is computing the same result during one loop over memory. The implemented polynomial multiplication is constant-time.

The proposed polynomial multiplier uses 16 parallel multipliers and is shown in Fig. 4. The number of multipliers is directly linked to the number of coefficients stored in one memory cell. On every memory load, each ternary coefficient is sent to a different multiplier. The second argument is the same for every multiplication unit. The proposed design utilizes the special form of input arguments. The multiplication is done only by addition or subtraction. The value of the accumulator is moved after specific number of multiplications to the next multiplicand. First 15 results are stored in a shift register. This operation is required for computing a proper value. These results are pushed back to multipliers at the end of the computations, to be updated with the remaining multiplication values.

IV. RESULTS

We report results for the low-cost DE1-SOC board, manufactured by Terasic. This board is equipped with Intel Cyclone V 5CSEMA5F31C6N FPGA. The chip contains 32,070 adaptive logic modules (ALMs), 128,300 registers, 87 DSP blocks and 3,970 Kb of memory. It contains also the dual-core ARM Cortex-A9 processor. However, in this paper, we focus only on an FPGA part. The post-place and route results were obtained from Intel Quartus Prime v18.1. There is no license requirement for the selected device to perform compilation and deployment.

In Table II, we report results for all security levels and all proposed parameter sets of the IND-CPA KEM and the IND-CCA PKE. We report logic usage for the full design and also for *r5_cpa_enc* module separately. This allows us to distinguish the cost of the post-quantum arithmetic module from the remaining costs of the hash-based function and the AEAD module required for the F-O transformation. For presented architecture, the lattice-based arithmetic takes only a small portion of the entire design, several times less than the standard cryptographic elements, such as a hash function or a block cipher. Thus, the implementation goal is achieved. Our design also does not use DSP modules for multiplication, so it can be deployed also with older FPGAs.

The main difference for the logic usage between the IND-CPA KEM and the IND-CCA PKE comes from the additional implementation cost of the AEAD module, not used in the IND-CPA KEM. The logic usage across all security levels is almost the same, as a result of using exactly the same design. The number of multipliers and other arithmetic modules remains always the same.

The memory requirements vary the most among different security levels, due to the necessity of storing significantly larger polynomials. Memory is also used as an input and output buffer to modules in the FIFO queues and in the SHAKE256 implementation. Thus, the overall memory requirements are larger than the sum of all *Round5* elements, such as keys, ciphertext, plaintext, and random data.

All implemented versions of *Round5* run with a similar clock frequency. The reported design is able to perform encapsulation and decapsulation for the highest security level under 1 ms. The encryption and decryption is significantly longer as a result of additional computations required by the F-O transformation and can be performed in around 2 ms also for the highest security level. Only for the lowest security level, the IND-CCA encryption is performed faster than the IND-CPA encapsulation for the parameter set proposed by the submission's authors. These operations are very similar, but for the selected parameter set, the IND-CCA version has smaller polynomial degree. The polynomial degree has the biggest impact on the computational complexity. Thus, faster execution is obtained for the IND-CCA encryption than for the IND-CPA encapsulation.

TABLE II
OBTAINED RESULTS FOR THE ROUND5 IND-CPA KEM AND THE ROUND5 IND-CCA PKE. * – RESULTS IN BYTES; ** – RESULTS IN CLOCK CYCLES.

| <i>Parameter</i> | <i>IND-CPA KEM</i> | <i>IND-CCA PKE</i> | <i>Ratio</i> |
|--------------------------|--------------------|--------------------|--------------|
| Security level: 1 | | | |
| Parameter set | R5ND_1KEM_0d | R5ND_1PKE_0d | — |
| PK size* | 634 | 676 | 1.066 |
| SK size* | 16 | 708 | 44.25 |
| CT size* | 682 | 754 | 1.106 |
| Enc latency** | 49,714 | 44,808 | 0.90 |
| Dec latency** | 25,556 | 67,504 | 2.64 |
| Total ALMs | 4,084 | 6,305 | 1.54 |
| Arithm ALMs | 448 | 487 | 1.08 |
| Memory* | 8,445 | 9,134 | 1.08 |
| Max freq. | 142 MHz | 136 MHz | 0.95 |
| Enc time | 0.35 ms | 0.33 ms | 0.94 |
| Dec time | 0.18 ms | 0.50 ms | 2.75 |
| Security level: 3 | | | |
| Parameter set | R5ND_3KEM_0d | R5ND_3PKE_0d | — |
| PK size* | 909 | 983 | 1.081 |
| SK size* | 24 | 1,031 | 42.958 |
| CT size* | 981 | 1,119 | 1.140 |
| Enc latency** | 80,565 | 94,083 | 1.17 |
| Dec latency** | 41,162 | 141,546 | 3.44 |
| Total ALMs | 4,098 | 6,312 | 1.54 |
| Arithm ALMs | 494 | 467 | 0.95 |
| Memory* | 9,466 | 10,112 | 1.06 |
| Max freq. | 135 MHz | 132 MHz | 0.98 |
| Enc time | 0.60 ms | 0.71 ms | 1.19 |
| Dec time | 0.30 ms | 1.07 ms | 3.52 |
| Security level: 5 | | | |
| Parameter set | R5ND_5KEM_0d | R5ND_5PKE_0d | — |
| PK size* | 1,176 | 1,349 | 1.145 |
| SK size* | 32 | 1,413 | 44.156 |
| CT size* | 1,274 | 1,525 | 1.197 |
| Enc latency** | 132,877 | 175,965 | 1.32 |
| Dec latency** | 67,556 | 264,534 | 3.92 |
| Total ALMs | 4,116 | 6,337 | 1.54 |
| Arithm ALMs | 522 | 502 | 0.96 |
| Memory* | 10,753 | 11,765 | 1.09 |
| Max freq. | 133 MHz | 130 MHz | 0.98 |
| Enc time | 1.00 ms | 1.35 ms | 1.35 |
| Dec time | 0.50 ms | 2.03 ms | 4.01 |

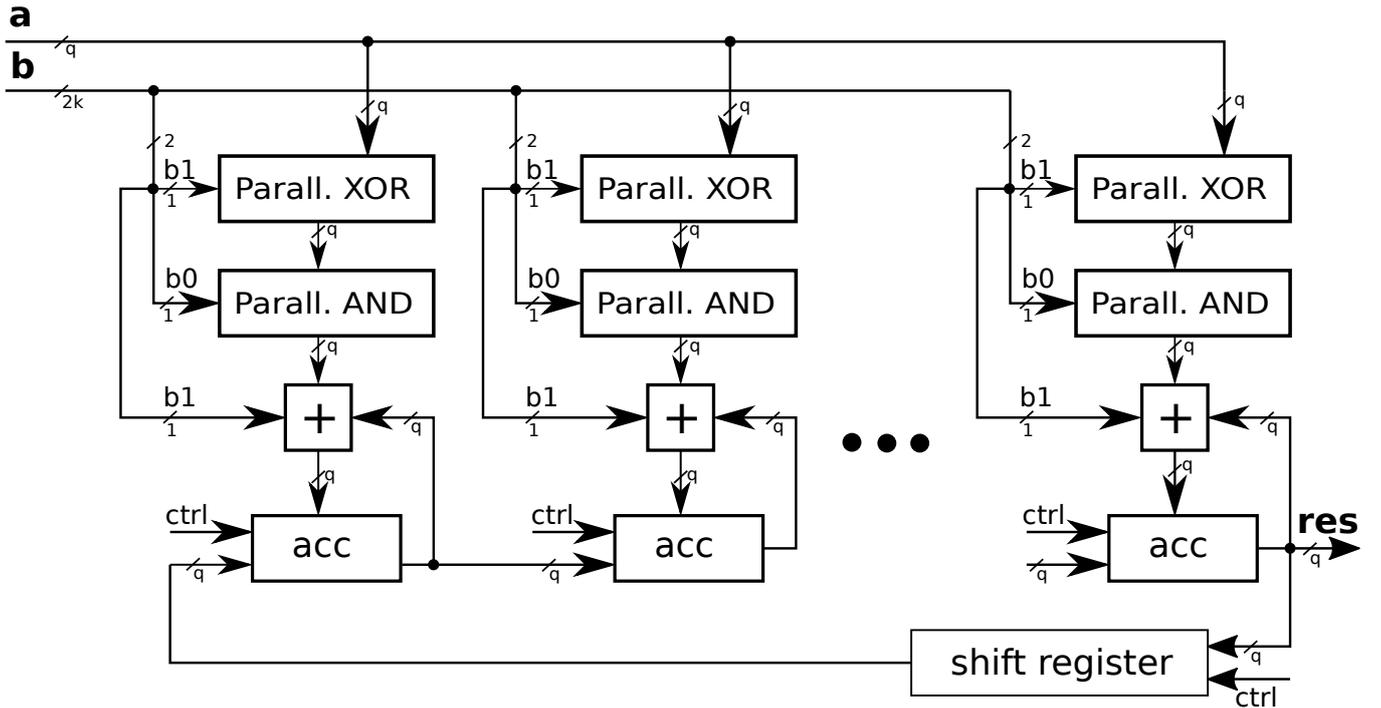


Fig. 4. The parallel polynomial multiplier for Round5

A. Comparison to other results

A fair comparison to other results reported to date is hard and complex due to multiple factors directly affecting the obtained results. Moreover, there are no specific guidelines from NIST about proper evaluation of candidates, regarding an FPGA device, implementation goal, API, or compliance criteria. In terms of API and the compliance criteria our design follows the proposal by Ferozpur et al. [20]. As for the FPGA board, we selected one of the least expensive boards, with the free license for the compiler. Any conclusions from comparing logic usage for different FPGA vendors and for different PQC submissions, we are leaving up to the reader.

Howe et al. [5] reported results for full hardware implementation of another lattice-based candidate *FrodoKEM*. They report results for Xilinx Atrix-7 FPGA, and their design balances between area consumption and performance. Their maximum frequency is 167 MHz and is higher than reported in this paper for *Round5*. However, the time required to perform decapsulation is at least an order of magnitude higher, requiring around 20 ms for the execution. Logic requirements are reported for separate modules, not for the entire design able to perform all operations. These modules require around 2,000 slices each.

Most of the other papers reporting results for non-lattice-based PQC candidates, also provide results for Xilinx FPGAs. However, Wang et al. reported results for the Classic McEliece [11], [12] implementation on other high-end Intel FPGA, Stratix V. Their time-optimized implementation uses 121,806 ALM and run at 250 MHz clock, being able to encrypt

and decrypt in less than 0.1 ms.

V. CONCLUSIONS AND FUTURE WORK

This paper presented a complete low-area FPGA design for ring version of *Round5*, a lattice-based submission to NIST PQC Standardization process. We reported the post-place and route results for main parameters sets covering all security levels for KEMs and PKEs.

For future work, we consider exploring the area-performance trade-off offered by the proposed polynomial multiplier. A similar polynomial ring is also used by other NTRU-based proposals. Thus, our multiplier can be used for the performance evaluation of other candidates. As for *Round5*, an extension with error correcting codes and the non-ring versions of all schemes is the next big step to provide coverage of all possible parameter sets and versions.

ACKNOWLEDGEMENTS

Special thanks to Kris Gaj for his help and valuable comments.

REFERENCES

- [1] "Algorithms for quantum computation: Discrete logarithms and factoring."
- [2] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," vol. 21, no. 2, pp. 120–126.
- [3] Post-Quantum Cryptography: Call for Proposals. [Online]. Available: <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals>
- [4] Post-Quantum Cryptography: Round 2 Submissions. [Online]. Available: <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions>

- [5] “On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography,” vol. 67.
- [6] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, “High Performance Post-Quantum Key Exchange on FPGAs,” p. 17. [Online]. Available: <https://eprint.iacr.org/2017/690.pdf>
- [7] T. Oder and T. Guneyasu, “Implementing the NewHope-Simple Key Exchange on Low-Cost FPGAs,” in *LATINCRYPT 2017*. [Online]. Available: https://www.ei.ruhr-uni-bochum.de/media/seceng/veroeffentlichungen/2018/04/16/newhope_fpga.pdf
- [8] F. Farahmand, V. Dang, D. T. Nguyen, and K. Gaj, “Evaluating the Potential for Hardware Acceleration of Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Codesign.”
- [9] B. Koziel, R. Azarderakhsh, M. Mozaffari Kermani, and D. Jao, “Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves,” vol. 64, no. 1, pp. 86–99. [Online]. Available: <http://ieeexplore.ieee.org/document/7725935/>
- [10] A. Ferozपुरi and K. Gaj, “High-speed FPGA Implementation of the NIST Round 1 Rainbow Signature Scheme,” in *2018 International Conference on ReConfigurable Computing and FPGAs (ReConFig)*. IEEE, pp. 1–8. [Online]. Available: <https://doi.org/10.1109/reconfig.2018.8641734>
- [11] W. Wang, J. Szefer, and R. Niederhagen, “FPGA-based Key Generator for the Niederreiter Cryptosystem Using Binary Goppa Codes,” in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma, Eds. Springer International Publishing, vol. 10529, pp. 253–274. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4_13
- [12] —, “FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes,” in *PQCrypto 2018*, ser. LNCS, T. Lange and R. Steinwandt, Eds., vol. 10786. Springer International Publishing, pp. 77–98. [Online]. Available: https://doi.org/10.1007/978-3-319-79063-3_4
- [13] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the 41st Annual ACM Symposium on Symposium on Theory of Computing - STOC '09*. ACM Press, p. 169. [Online]. Available: <https://doi.org/10.1145/1536414.1536440>
- [14] T. Guneyasu and T. Oder, “Towards lightweight Identity-Based Encryption for the post-quantum-secure Internet of Things,” in *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE, pp. 319–324. [Online]. Available: <https://doi.org/10.1109/ISQED.2017.7918335>
- [15] I. T. L. Round5 Submission Team. Round 2 Submissions -Round5 candidate submission package. [Online]. Available: <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/Round5-Round2.zip>
- [16] H. Baan, S. Bhattacharya, O. Garcia-Morchon, R. Rietman, L. Tolhuizen, J.-L. Torre-Arce, and Z. Zhang, “Round2: KEM and PKE based on GLWR,” p. 72.
- [17] M.-J. O. Saarinen, “HILA5: On Reliability, Reconciliation, and Error Correction for Ring-LWE Encryption,” pp. 192–212. [Online]. Available: https://doi.org/10.1007/978-3-319-72565-9_10
- [18] E. Fujisaki and T. Okamoto, “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” vol. 26, no. 1, pp. 80–101. [Online]. Available: <https://doi.org/10.1007/s00145-011-9114-1>
- [19] J. Kelsey, S.-j. Chang, and R. Perlner, “SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash.” [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>
- [20] A. Ferozपुरi, F. Farahmand, V. B. Dang, M. U. Sharif, J.-P. Kaps, and K. Gaj, “Hardware API for Post-Quantum Public Key Cryptosystems.” [Online]. Available: https://cryptography.gmu.edu/athena/PQC/PQC_HW_API.pdf
- [21] J. Hoffstein, J. Pipher, and J. H. Silverman, “NTRU: A ring-based public key cryptosystem,” in *Algorithmic Number Theory*, J. P. Buhler, Ed. Springer Berlin Heidelberg, vol. 1423, pp. 267–288. [Online]. Available: <https://doi.org/10.1007/BFb0054868>