Evaluating the Potential for Hardware Acceleration of Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Codesign

Farnoud Farahmand, Viet B. Dang, Duc Tri Nguyen, and Kris Gaj(✉)

George Mason University, Fairfax, USA
{ffarahma,vdang6,dnguye69,kgaj}@gmu.edu

Abstract. The speed of NTRU-based Key Encapsulation Mechanisms (KEMs) in software, especially on embedded software platforms, is limited by the long execution time of its primary operation, polynomial multiplication. In this paper, we investigate the potential for speeding up the implementations of four NTRU-based KEMs, using software/hardware codesign, when targeting Xilinx Zynq UltraScale+ multiprocessor system-on-chip (MPSoC). All investigated algorithms compete in Round 1 of the NIST PQC standardization process. They include: ntru-kem from the NTRUEncrypt submission, Streamlined NTRU Prime and NTRU LPRime KEMs of the NTRU Prime candidate, and NTRU-HRSS-KEM from the submission of the same name. The most-time consuming operation, polynomial multiplication, is implemented in the Programmable Logic (PL) of Zynq UltraScale+ (i.e., in hardware) using constant-time hardware architectures most appropriate for a given algorithm. The remaining operations are executed in the Processing System (PS) of Zynq, based on the ARM Cortex-A53 Application Processing Unit. The speed-ups of our software/hardware codesigns vs. purely software implementations, running on the same Zynq platform, are determined experimentally, and analyzed in the paper. Our experiments reveal substantial differences among the investigated candidates in terms of their potential to benefit from hardware accelerators, with the special focus on accelerators aimed at offloading to hardware only the most time-consuming operation of a given cryptosystems. The demonstrated speed-ups vs. functionally equivalent purely software implementations vary between 4.0 and 42.7 for encapsulation, and between 6.4 and 149.7 for decapsulation.

Keywords: Software/hardware implementation · Hardware accelerator · Key Encapsulation Mechanism ·

This paper is partially based upon work supported by the U.S. Department of Commerce/National Institute of Standards and Technology under Grant no. 70NANB18H218, as well as the National Science Foundation under Grant no. CNS-1801512.

J. Ding and R. Steinwandt (Eds.): PQCrypto 2019, LNCS 11505, pp. 23–43, 2019.
https://doi.org/10.1007/978-3-030-25510-7_2

Post-Quantum Cryptography · NTRU · System on Chip · Programmable logic · High-level synthesis · Embedded software platforms

1 Introduction

Hardware benchmarking of Post-Quantum Cryptography (PQC) candidates is extremely challenging due to their high algorithmic complexity, specifications geared more toward mathematicians than toward engineers, and the lack of hardware description language libraries containing code of basic building blocks. As a result, the workload for a single algorithm can easily reach several man-months. Consequently, due to the Round 1 focus on evaluating security and software efficiency [20], only a few candidates in the NIST PQC standardization process have been fully implemented in hardware to date [9,12,15,16,22,24]. To make the matters worse, a substantial number of operations used by PQC algorithms are both complex and sequential in nature. Porting these operations to hardware can take a large number of man-hours, and at the same time bring little benefit in terms of the total execution time.

In this paper, we propose an approach aimed at overcoming these difficulties. This approach is based on the concept of software/hardware codesign. The majority of the algorithm operations are implemented in software. Only a few main operations (optimally just one), taking the majority of the execution time, are offloaded to hardware.

This approach has become very practical in modern embedded systems due to the emergence of special platforms, integrating the software programmability of an ARM-based processor with the hardware programmability of FPGA fabric. Examples include Xilinx Zynq 7000 System on Chip (SoC), Xilinx Zynq UltraScale+ MPSoC, Intel Arria 10 SoC FPGAs, and Intel Stratix 10 SoC FPGAs. These devices support hybrid software/hardware codesigns composed of a traditional C program running on an ARM processor, communicating, using an efficient interface protocol (such as AMBA AXI4), with a hardware accelerator described manually using a hardware description language such as VHDL, or generated automatically, using High-Level Synthesis.

Assuming that an implemented algorithm contains a limited number of operations, suitable for parallelization, and these operations contribute 91% or more to the total execution time, then an order of magnitude (or higher) speed-up is possible, with the amount of development time reduced from months to weeks or even days.

An additional benefit of this approach is the possibility to easily estimate the speed-ups that could be achieved by developing and implementing special instructions of a general-purpose processor (such as ARM) supporting a specific PQC algorithm or a group of related-algorithms.

Based on extensive software profiling experiments, conducted using both ARM and AMD64 platforms, we have determined that all NTRU-based NIST Round 1 KEMs are very suitable for software/hardware codesign. In particular, for all of them, no more than three major operations contribute at least 92% of the total execution time to both encapsulation and decapsulation. Additionally, the most time consuming of these operations, polynomial multiplications in $Z_q[x]/P$ and $Z_3[x]/P$, with P selected as a polynomial of the prime degree n, are very easily parallelizable and straightforward to implement in constant time using moderate amount of hardware resources.

In the rest of this paper, we quantify the influence of a dedicated hardware accelerator on the performance and implementation cost of each of the following four Round 1 KEMs: NTRUEncrypt [6], NTRU-HRSS [13], Streamlined NTRU Prime, and NTRU LPRime [5,21]. The speed-ups of the software/hardware codesigns vs. purely software implementations are measured, and their influence on the ranking of candidates is determined.

Table 1. Features of round 1 NTRU-based KEMs.

Feature	NTRUEncrypt	NTRU-HRSS	Streamlined NTRU Prime	NTRU LPRime
Polynomial P	$x^n - 1$	$\Phi_n = (x^n - 1)/(x - 1)$ irreducible in $Z_q[x]$	$x^n - x - 1$ irreducible in $Z_q[x]$	$x^n - x - 1$ irreducible in $Z_q[x]$
Degree n*	Prime	Prime	Prime	Prime
Modulus q	2^d	$2^{ceil(3.5+log_2(n))}$	Prime	Prime
Weight w	Fixed weight for f and g	N/A	Fixed weight for f and r. $3w \leq 2n$ $16w + 1 \leq q$	Fixed weight for b and a. $3w \leq 2n$ $16w + 2\delta + 3 \leq q$
Quotient rings	R/q: $Z_q[x]/(x^n - 1)$	R/q: $Z_q[x]/(x^n - 1)$ S/3: $Z_3[x]/(\Phi_n)$	R/q: $Z_q[x]/(x^n - x - 1)$ R/3: $Z_3[x]/(x^n - x - 1)$	R/q: $Z_q[x]/(x^n - x - 1)$ R/3: $Z_3[x]/(x^n - x - 1)$
#Poly Mults for encapsulation	1 in R/q	1 in R/q	1 in R/q	2 in R/q
#Poly Mults for decapsulation	2 in R/q	2 in R/q 1 in S/3	2 in R/q 1 in R/3	3 in R/q
Private key f of the form 1+3F	Yes	No	No	No
Invertibility checks in key generation	Yes	No	Yes	No
Decryption failures	Yes	No	No	No

* denoted by N in the specification of NTRUEncrypt and by p in the specifications of Streamlined NTRU Prime and NTRU LPRime

2 Background

Basic features of four investigated Round 1 NTRU-based KEMs are summarized in Table 1. NTRUEncrypt is the only candidate that uses a reducible polynomial, which may potentially increase its attack surface. It is also the only candidate with a non-zero probability of decryption failure, and one of the two (together with Streamlined NTRU Prime) requiring invertibility checks in key generation. All polynomials have a prime degree n. Three features that have primary influence on the area of a corresponding hardware accelerator include: (a) prime vs. power-of-two modulus q for operations on polynomial coefficients; (b) requirement for operations in additional rings, such as $Z_3[x]/(\Phi_n)$ in NTRU-HRSS and $Z_3[x]/(x^n - x - 1)$ in Streamlined NTRU Prime; (c) Private key of the form $1 + 3F$ in NTRUEncrypt.

The execution time of Encapsulation and Decapsulation is affected primarily by the required number of polynomial multiplications (Poly Mults), which is the lowest in case of NTRUEncrypt and the highest in case of NTRU LPRime. The fixed weight of polynomials with small coefficients affects the execution time of a polynomial multiplication only in case of using a rotator-based multiplier [4, 8, 14].

In Table 2, the numerical values of parameters in the implemented variants of KEMs are summarized. All investigated KEMs use approximately the same values of the polynomial degree n, which in hardware leads to similar Poly Mult execution times in terms of the number of clock cycles. NTRUEncrypt and NTRU-HRSS have an advantage of using a modulus q being a power of two, which substantially reduces the time of Poly Mult in software, and the area of the Poly Mult accelerator in hardware. Three out of four KEMs are claimed to belong to the security category 5, with the number of pre-quantum security bits estimated to be close to 256. NTRU-HRSS is the only investigated candidate limited to the security category 1, with the number of pre-quantum security bits estimated at 136 (i.e., slightly above 128). It should be stressed that no other sets of parameters, corresponding to any higher security category are provided in the specification of this KEM. Similarly, no other parameter sets, corresponding to any lower security levels, are defined in the specifications of Streamlined NTRU Prime or NTRU LPRime. The public and private key sizes are the smallest for NTRUEncrypt and the largest for Streamlined NTRU Prime.

3 Previous Work

3.1 Hardware Accelerators for NTRUEncrypt

In 2001, Bailey et al. [4] introduced and implemented a Fast Convolution Algorithm for polynomial multiplication, exploiting the sparsity of polynomials. In [14], Kamal et al. analyzed several implementation options for traditional NTRUEncrypt [11] targeting Virtex-E family of FPGAs. In this design, the polynomial multiplier took advantage of the ternary nature of polynomials in NTRUEncrypt and utilized an empirically chosen Barrel shifter (rotator). The results were reported for the parameter set with (n = 251, q = 128). Liu et al. implemented the truncated polynomial ring multiplier using linear feedback shift register (LFSR) in 2015 [17] and an extended LFSR [18] in 2016. Both designs were implemented using Cyclone IV FPGAs. The former paper reported results

Table 2. Numerical values of parameters in the implemented variants of round 1 NTRU-based KEMs.

Feature	NTRUEncrypt	NTRU-HRSS	Streamlined NTRU Prime	NTRU LPRime
Parameter set	NTRU-743	ntruhrss701	sntrup4591761	ntrulpr4591761
Degree n	743	701	761	761
Modulus q	$2048 = 2^{11}$	$8192 = 2^{13}$	$2^{12} < 4591 < 2^{13}$	$2^{12} < 4591 < 2^{13}$
Polynomials with small coefficients	Fixed weight 494 for f and g. Uniform trinary for r and m	Uniform T+ for f and g. Uniform trinary for r and m	Fixed weight 286 for f and r. Uniform trinary for g and m	Fixed weight 250 for b and a.
Expected failure rates	2^{-112}	0	0	0
Security category	5	1	5	5
Pre-quantum security bits	256	136	248	225
Shared key size in bits	384	256	256	256
Public key size*	1023	1140	1218	1047
Secret key size*	1173	1422	1600	1238
Ciphertext size*	1023	1281	1047	1175

* sizes in bytes

for three parameter sets with (n = 251, q = 128), (n = 347, q = 128), and (n = 503, q = 256). The latter paper reported results for 12 parameter sets specified in the IEEE Standard NTRUEncrypt SVES [2]. Out of these parameter sets, the closest one to the cases considered in this paper was the parameter set with (n = 761, q = 2048). None of the aforementioned designs was made open-source. In [8], the first full constant-time implementation of the IEEE Standard NTRUEncrypt SVES [2] was reported. This implementation supported two parameter sets, with (n = 1499, q = 2048) and (n = 1087, q = 2048), and targeted the Xilinx Virtex UltraScale FPGA. As described above, the results reported in these papers concerned different parameter values and/or different (mostly much older) hardware platforms. Additionally, all aforementioned hardware implementations other than [17] and [8] were not constant time. As a result, their comparison with the results presented in this work is neither practical nor fair.

3.2 Software-Hardware Codesign of PQC Algorithms

Only a few attempts to accelerate software implementations of post-quantum cryptosystems have been made through software/hardware (SW/HW) codesign. A coprocessor consisting of the PicoBlaze softcore and several parallel acceleration units for the McEliece cryptosystem was implemented on Spartan-3AN

FPGAs by Ghosh et al. [10]. No speed-up vs. purely software implementation using PicoBlaze was reported.

In 2015, Aysu et al. [3] built a high-speed implementation of a lattice-based digital signature scheme using SW/HW codesign techniques. The paper focused on the acceleration of signature generation. The design targeted the Cyclone IV FPGA family and consisted of the NIOS2 soft processor, a hash unit, and a polynomial multiplier. Compared to the C implementation running on the NIOS2 processor, the most efficient software/hardware codesign reported in the paper achieved the speed-up of 26,250x at the expense of the increase in the number of Logic Elements by a factor of 20.

Migliore et al. [19] presented a hardware/software codesign for the lattice-based Fan-Vercauteren (FV) homomorphic encryption scheme with the majority of the Karatsuba-based multiplication/relinearization operation performed in hardware. The platform used for hardware acceleration was Stratix V GX FPGA. Software ran on a PC, based on Intel i7-4910MQ, with 4 cores operating at 2.9 GHz, connected with the FPGA-based DE5-450 Terasic board using PCI Express (PCIe) 3.0, with eight lines, capable of handling transfers with the throughput up to 250 MB/s per line in full-duplex. The speed-up compared to the purely software implementation was estimated to be 1.4x.

Wang et al. [23] reported a software/hardware implementation of the PQC digital signature scheme XMSS. The selected platform was an Intel Cyclone V SoC, and the software part of the design was implemented using a soft-core processor RISC-V. Hardware accelerators supported a general-purpose SHA-256 hash function, as well as several XMSS specific operations. The design achieved the speed-up of 23x for signing and 18x for verification over a purely software implementation running on RISC-V.

All the aforementioned platforms were substantially different than the platforms used in this work. The algorithms and their parameters were also substantially different. As a result, limited information could be inferred regarding the optimal software/hardware partitioning, expected speed-up, or expected communication overhead.

4 Methodology

4.1 Platform and Software

The platform selected for our experiments is Xilinx Zynq UltraScale+ MPSoC XCZU9EG-2FFVB1156E, which is fabricated using a 16 nm technology and mounted on the ZCU102 Evaluation Kit from Xilinx. This MPSoC is composed of two major parts sharing the same chip, the PS and PL. The PS (Processing System) includes a quad-core ARM Cortex-A53 Application Processing Unit (APU), a dual-core ARM Cortex-R5 Real-Time Processing Unit (RPU), Graphics Processing Unit, 256 kB On-Chip Memory, and more. Each processor of the APU and RPU is equipped with a 32 kB instruction cache and a 32 kB data cache. In our experiments, we use only one processor of the APU (Core 0 of Cortex-A53) running at the frequency of 1.2 GHz. The PL (Programmable Logic) includes a programmable FPGA fabric similar to that of Virtex UltraScale+ FPGAs. The software used is Xilinx Vivado Design Suite HLx Edition, Xilinx Software Development Kit (XSDK), and Xilinx Vivado HLS, all with the versions no. 2017.2.

A high-level block diagram of the experimental software/hardware codesign platform is shown in Fig. 1. The hardware accelerator, implementing the Polynomial Multiplier unit, is denoted as Poly Mult. This accelerator is extended with the Input and Output FIFOs, as well as AXI DMA, for high-speed communication with the Processing System. The details of the Input and Output FIFO interfaces are shown in Fig. 2. Timing measurements are performed using an AXI Timer, capable of measuring time in clock cycles of the 200 MHz system clock. The Poly Mult unit can operate at a variable frequency different than that of DMA. This frequency can be changed at run time using the Clocking Wizard, controlled from software. As a result, the Input and Output FIFOs use different clocks for their read and write operations.

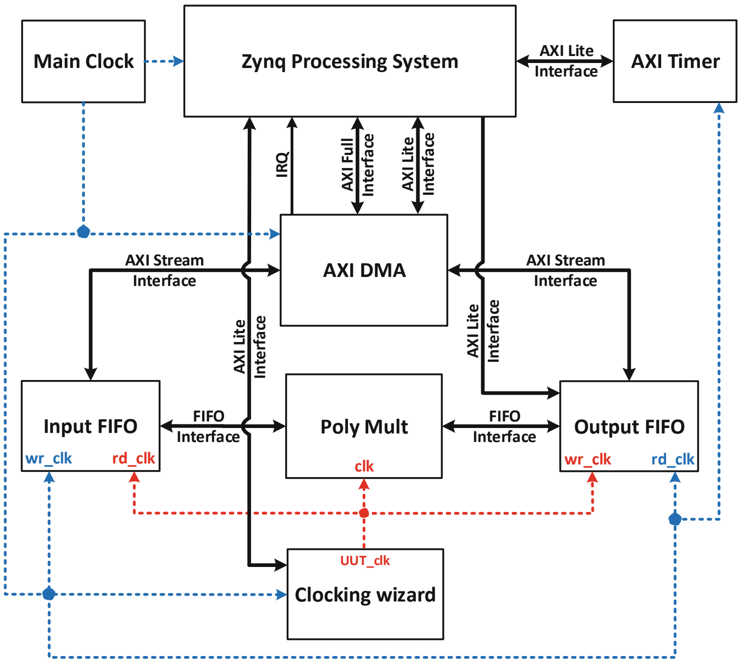


Fig. 1. High-level block diagram of the experimental SW/HW co-design platform.

4.2 Design of Hardware Using the RTL Methodology

The Register-Transfer Level (RTL) designs of hardware accelerators for NTRU-based KEMs follow closely the block diagrams shown in Figs. 3, 4, 5 and 6.

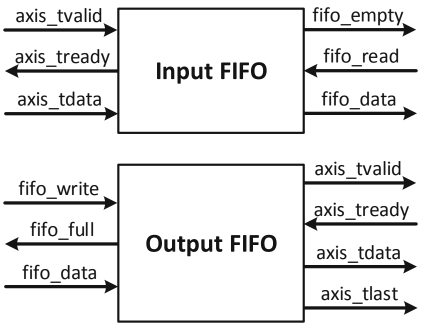
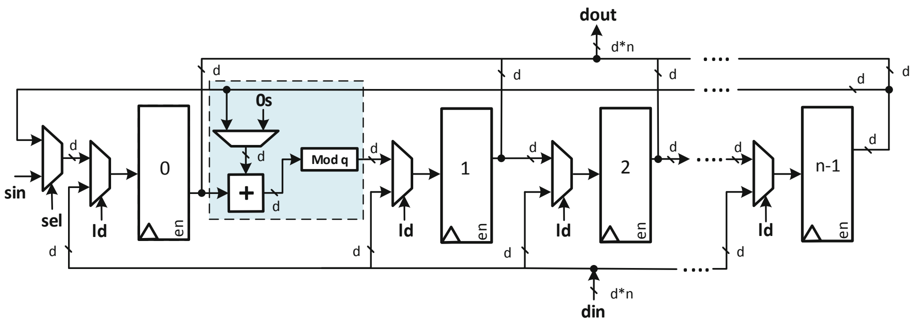
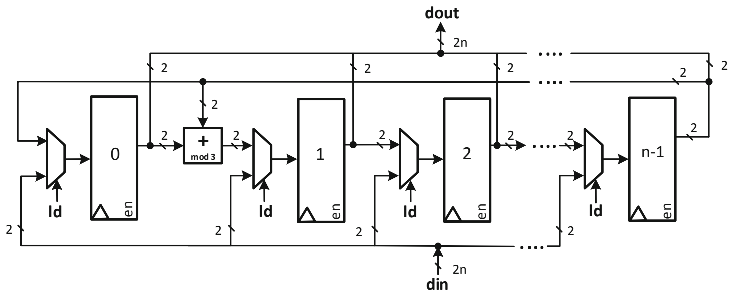


Fig. 2. The input and output FIFO interface.



(a) Zq_LFSR. The blue part is only utilized in Streamlined NTRU Prime and NTRU LPRime



(b) Z3_LFSR

Fig. 3. LFSR block diagrams. (Color figure online)

The Zq_LFSR, used in all KEMs, is initialized with the value of a polynomial $a(x)$ with large coefficients. In each subsequent iteration, the output from LFSR contains the value $a(x) \cdot x^i$ mod P. In a single clock cycle, a simple multiplication by x, namely $a(x) \cdot x^{i+1}$ mod P= $a(x) \cdot x^i \cdot x$ mod P, is performed. For $P = x^n - 1$, this multiplication is equivalent to rotation. For $P = x^n - x - 1$, an extra addition mod q, marked in Fig. 3a with the blue background is required.

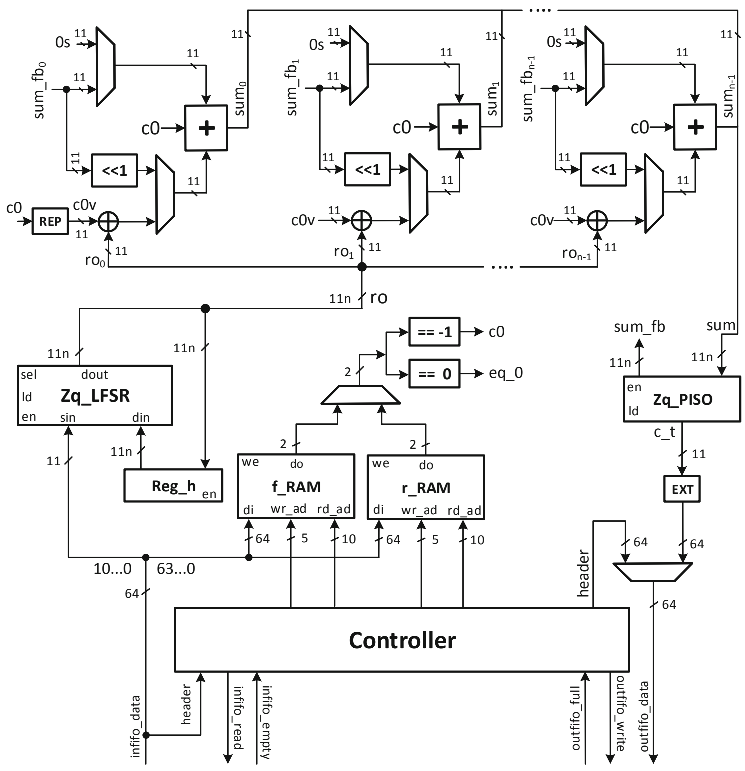


Fig. 4. Block diagram of the Poly Mult unit for NTRUEncrypt.

The multiplication in the ring S/3 for NTRU-HRSS and R/3 for Streamlined NTRU Prime is performed using the Z3_LFSR, shown Fig. 3b. This circuit operates using the same principle as Zq_LFSR, except all polynomial coefficients are reduced mod 3.

The entire Poly Mult unit for NTRUEncrypt is shown in Fig. 4. The multiplication of a polynomial $a(x)$ with large coefficients by a polynomial $b(x)$ with small coefficients (limited to -1, 0, and 1), involves calculating $a(x) \cdot x^i \bmod P$, multiplying it by b_i, and adding it to the partial sum. The multiplication of each coefficient by -1 is accomplished by calculating their one's complement (using an XOR with c0v, obtained by replicating c0 11 times) and the addition of c0 as carry-in to the following adder, represented by a square with +.

Coefficients of the public key h, are preloaded to the NTRUEncrypt Zq_LFSR before an encapsulation starts. All of these coefficients can be stored in Reg_h, and loaded back to Zq_LFSR in a single clock cycle, in case this LFSR is used in-between for any operation not involving h. Similarly, coefficients of the private key f are preloaded to the asymmetric f_RAM, visible at the input as a 32x64 RAM, and at the output as a 1024x2 RAM, before the decryption starts.

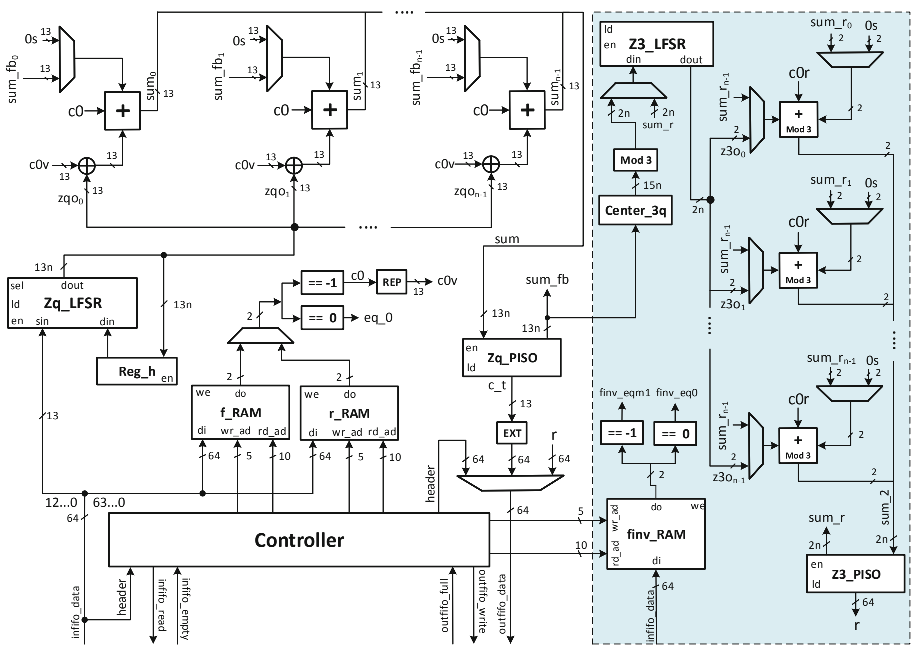


Fig. 5. Block diagram of the Poly Mult unit for NTRU-HRSS. (Color figure online)

The partial and final results are stored in the Zq_PISO (Parallel-In Serial-Out) unit, with the parallel input of the width of $11 \cdot n$ bits, the parallel output of the same width (used to enable the accumulation of intermediate products), and the serial output of the width of 11 bits used to read out the final result to the output FIFO.

The multiplication $t = r * h$, performed during encapsulation and the second part of decapsulation, takes $n = 743$ clock cycles. The multiplication $m' = f * c = (1 + 3 \cdot F) * c$, performed during the first part of decapsulation, requires two additional clock cycles, used respectively for the calculation of $F*c+2 \cdot F*c$ (with the multiplication of each coefficient of $F*c$ by 2 accomplished using a shift to the left by one, denoted in the diagram as $<< 1$) and $c+3 \cdot F*c$. In this paper, $a * b$ denotes polynomial multiplication, and $a \cdot b$ denotes regular multiplication, i.e., a multiplication of a polynomial and a constant, or a multiplication of two polynomial terms.

The Controller is responsible for generating suitable select and enable signals, communication with the Input and Output FIFOs, interpreting the input headers with instructions sent by the respective driver, and generating the output header containing the status and error codes that are sent back to the driver.

A block diagram of the hardware accelerator for NTRU-HRSS is shown in Fig. 5. The new part, marked using the blue background, is responsible for operations in the ring S/3. Compared to NTRUEncrypt, the size of all large coefficients increases from 11 to 13 bits. The portion of the circuit responsible for performing

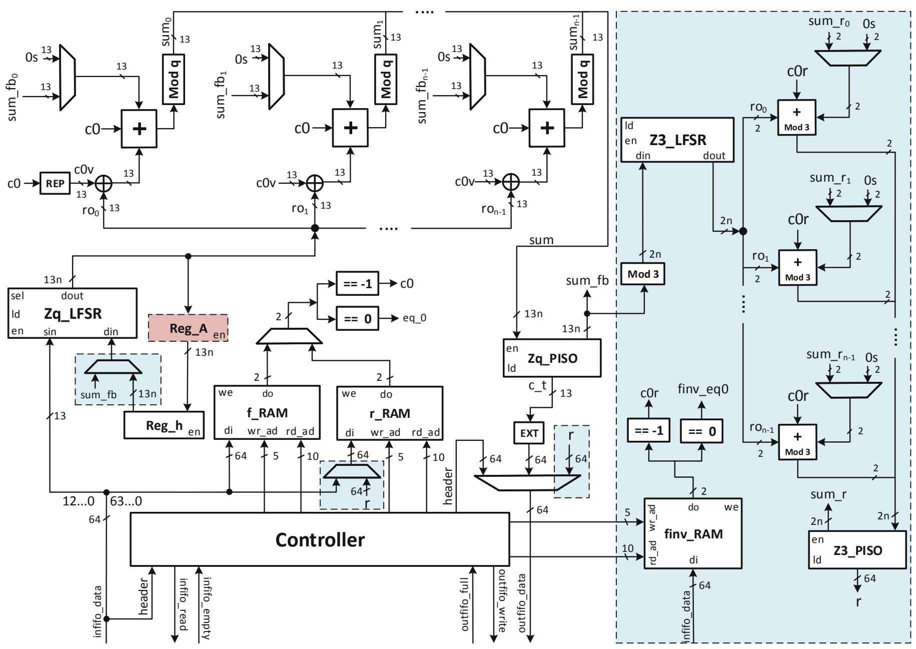


Fig. 6. Block diagram of the Poly Mult units for Streamlined NTRU Prime and NTRU LPRime. The blue parts are used only in the design for Streamlined NTRU Prime and the red part is used only in the design for NTRU LPRime. (Color figure online)

multiplication by $f = (1 + 3 \cdot F)$ is removed. Other than that, the operation of the circuit remains almost identical.

A block diagram of the hardware accelerators for Streamlined NTRU Prime and NTRU LPRime is shown in Fig. 6. The operations in R/3 are necessary only in case of Streamlined NTRU Prime and are similar to operations in S/3 for NTRU-HRSS. Compared to NTRU-HRSS, the main difference is the need for reduction of partial sums, involving large coefficients, mod q. Since now, q is a 13-bit prime, a conditional subtraction is necessary. An additional register A is required for NTRU LPRime only, increasing the number of required flip-flops.

4.3 Design of Hardware Using the HLS Methodology

The reference implementation of NTRUEncrypt in C, for $n = 743$, is based on the grade school algorithm for multiplication (also known as schoolbook, paper-and-pencil, etc.). Only for n equal to a power of 2, the fully recursive Karatsuba multiplication is used. When the grade school implementation of Poly Mult in C was provided at the input of Vivado HLS, the resulting circuit required tens of thousands of clock cycles to complete a single multiplication (even after inserting multiple Vivado HLS directives in the form of pragmas). The similar results were obtained by using an earlier C implementation of Poly Mult, based on the concept of Rotation, developed by OnBoard Security [1].

As a result, the decision was made to treat C like a hardware description language, and implement Poly Mult from scratch, in such a way to infer the circuit from Fig. 4. This attempt appeared to be successful, which was indicated by reaching almost exactly the same number of clock cycles as that required by the RTL implementation. The same approach was then applied to the remaining three candidates.

The HLS-ready C code was first verified using a C testbench, based on the reference software implementation used as a source of test vectors. The resulting HDL code was then verified using exactly the same VHDL testbench which was used to verify the RTL implementation. The implementation phase (logic synthesis, mapping, placing, and routing) was identical for both RTL and HLS approaches. In the HLS flow, the first result estimates, in terms of the number of clock cycles, maximum clock frequency, and resource utilization, were generated in the form of reports by Vivado HLS. However, except for the number of clock cycles (which was accurate), the remaining numbers did not match the final post-place & route results.

5 Results

The results of profiling for the purely software implementations, running on a single core of ARM Cortex-A53, at the frequency of 1.2 GHz, are presented in the left portion of Table 3. For each of the four investigated algorithms and each major operation (Encapsulation and Decapsulation), four most time-consuming functions are identified. In each of the investigated cases, the most time consuming function is poly_mult(), responsible for performing polynomial multiplication in R/q. The contribution of this function varies between 78.2% in case of the NTRUEncrypt encapsulation, up to 99.5% in case of the Streamlined NTRU Prime decapsulation. poly_mult() is the only function listed among the four most time-consuming functions for all 8 investigated operations. It is also a function with a well-known potential for vast parallelization (and thus a very substantial speed-up) in hardware. As a result, poly_mult() was a natural candidate for offloading to hardware, and no other function listed in Table 3 could offer a clear potential for delivering an additional speed-up, especially for multiple algorithms.

The number of clock cycles required by Poly Mult, the maximum clock frequency, and the resource utilization obtained using the RTL and HLS approaches are summarized in Table 4. In both cases, the number of clock cycles is determined using simulation. The maximum clock frequency is obtained by using Vivado in combination with the automated hardware optimization tool called Minerva [7]. The obtained values correspond to the static timing analysis results after placing and routing, and have been confirmed experimentally using our setup shown in Fig. 1. The resource utilization is based on the post-place and route reports of Vivado. Only resources used to implement Poly Mult are listed in

Table 3. Profiling results for the software and software/hardware implementations targeting Zynq UltraScale+ MPSoC. (SW) and (HW) indicate whether poly_mult() is executed in software or in hardware. x2 means that a given function is called twice.

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
NTRUEncrypt - Encaps					
1. poly_mult (SW)	743.510	78.177	1. generate_r	91.665	38.286
2. generate_r	91.665	9.638	2. mask_m	40.960	17.108
3. mask_m	40.960	4.307	3. poly_mult (HW)	32.115	13.414
4. crypto_hash_sha512 x2	17.650	1.856	4. crypto_hash_sha512 x2	17.650	7.372
NTRUEncrypt - Decaps					
1. poly_mult (SW) x2	1492.870	87.800	1. generate_r	79.890	29.999
2. generate_r	79.890	4.699	2. poly_mult (HW) x2	55.966	21.015
3. unmask_m	40.865	2.403	3. unmask_m	40.865	15.345
4. unpack_secret_key_CCA	17.975	1.057	4. unpack_secret_key_CCA	17.975	6.750
NTRU-HRSS - Encaps					
1. poly_mult (SW)	3091.550	97.585	1. poly_Rq_frommsg	31.570	28.138
2. poly_Rq_frommsg	31.570	0.997	2. poly_mult (HW)	31.521	28.094
3. owcpa_samplemsg	11.445	0.361	3. owcpa_samplemsg	11.445	10.201
4. poly_Rq_getnoise	10.595	0.334	4. poly_Rq_getnoise	10.595	9.443
NTRU-HRSS - Decaps					
1. poly_mult (SW) x2	9302.780	99.211	1. poly_mult (HW) x2	51.333	39.678
2. poly_Rq_frommsg	30.460	0.325	2. poly_Rq_frommsg	30.460	23.544
3. unpack_sk	10.315	0.110	3. unpack_sk	10.315	7.973
4. poly_Rq_getnoise	9.975	0.106	4. poly_Rq_getnoise	9.975	7.710
Streamlined NTRU Prime - Encaps					
1. poly_mult (SW)	11,846.950	92.702	1. small_random_weightw	766.025	77.933
2. small_random_weightw	766.025	5.994	2. FIPS202_SHA3_512	155.080	15.777
3. FIPS202_SHA3_512	155.080	1.214	3. poly_mult (HW)	34.003	3.459
4. rq_decode	10.165	0.080	4. rq_decode	10.165	1.034
Streamlined NTRU Prime - Decaps					
1. poly_mult (SW) x2	35,546.140	99.489	1. FIPS202_SHA3_512	154.535	64.734
2. FIPS202_SHA3_512	154.535	0.433	2. poly_mult (HW) x2	52.428	21.962
3. rq_decode	10.145	0.028	3. rq_decode	10.145	4.250
4. rq_round3	9.045	0.025	4. rq_round3	9.045	3.789
NTRU LPRime - Encaps					
1. poly_mult (SW) x2	23,693.840	97.908	1. small_seeded_weightw	327.195	57.686
2. small_seeded_weightw	327.195	1.352	2. FIPS202_SHA3_512 x2	106.355	18.751
3. FIPS202_SHA3_512 x2	106.355	0.439	3. poly_mult (HW) x2	53.663	9.461
4. rq_fromseed	28.995	0.120	4. rq_fromseed	28.995	5.112
NTRU LPRime - Decaps					
1. poly_mult (SW) x2	35,540.750	98.598	1. small_seeded_weightw	339.285	58.920
2. small_seeded_weightw	339.285	0.941	2. FIPS202_SHA3_512 x2	102.960	17.880
3. FIPS202_SHA3_512 x2	102.960	0.286	3. poly_mult (HW) x2	68.484	11.893
4. rq_fromseed	29.000	0.080	4. rq_fromseed	29.000	5.036

Table 4. Differences in results obtained using the RTL and HLS approaches.

Metric	RTL	HLS	HLS/RTL
NTRUEncrypt			
#cycles for Poly Mult in Encaps	744	743	0.999
#cycles for Poly Mult in Decaps	1, 491	1, 488	0.971
Maximum Clk Freq [MHz]	330	251	0.761
#LUTs	27, 912	42, 667	1.529
#Slices	4, 431	6, 268	1.415
#FFs	24, 697	24, 756	1.002
#BRAMs	4	3	0.750
NTRU-HRSS			
#cycles for Poly Mult in Encaps	702	703	1.001
#cycles for Poly Mult in Decaps	2, 111	2, 110	0.999
Maximum Clk Freq [MHz]	300	295	0.983
#LUTs	33, 230	32, 196	0.969
#Slices	5, 476	6, 622	1.209
#FFs	32, 327	48, 792	1.609
#BRAMs	6	4	0.667
Streamlined NTRU prime			
#cycles for Poly Mult in Encaps	762	761	0.998
#cycles for Poly Mult in Decaps	2, 291	2, 291	1.000
Maximum Clk Freq [MHz]	255	155	0.608
#LUTs	65, 207	88, 678	1.360
#Slices	9, 699	13, 690	1.411
#FFs	32, 929	31, 764	0.965
#BRAMs	6	4	0.667
NTRU LPRime			
#cycles for Poly Mult in Encaps	1, 524	1, 522	0.998
#cycles for Poly Mult in Decaps	2, 287	2, 283	0.998
Maximum Clk Freq [MHz]	255	158	0.620
#LUTs	52, 297	77, 385	1.480
#Slices	8, 483	12, 215	1.440
#FFs	39, 730	39, 832	1.002
#BRAMs	4	3	0.750

Table 4. Additional logic implemented in hardware, shown in Fig. 1, such as AXI DMA, Input FIFO, Output FIFO, Clocking Wizard, and AXI Timer, requires additional 7,858 LUTs, 1,593 Slices, 8,794 flip-flops, and 11 BRAMs.

Overall, the HLS-based implementations match very well (or even outperform) manually developed RTL implementations in terms of the number of clock cycles and the number of storage elements (flip-flops and BRAMs). The only exception is NTRU-HRSS, where the number of flip-flops is about 61% larger in case of using HLS. However, the HLS-based implementations require between 36% and 53% of more LUTs, and between 41% and 44% of more Slices. Once again the only exception is NTRU-HRSS, where the number of LUTs is comparable, at the expense of the substantial increase in the number of flip-flops. Additionally, the maximum clock frequency of the HLS-generated designs reached between 61% and 98% of the frequency of the manually-generated RTL designs. The development time was comparable because of the additional learning curve and more frequent trial-and-error tests necessary to develop an optimal HLS-ready C code.

Overall, the RTL approach was demonstrated to be superior, although not by a high margin. This approach is also more mature and more trusted by the cryptographic engineering community. As a result, in the rest of this paper, only results obtained using the RTL approach are reported and analyzed.

In Table 5, area overhead caused by special operations specific to particular KEMs is listed. Overall, Streamlined NTRUPrime and NTRU LPRime pay quite substantial price in terms of both maximum clock frequency and area compared to NTRUEncrypt and NTRU-HRSS. For example, replacing $q = 2^{13}$ by the 13-bit prime $q = 4591$ between NTRU-HRSS and Streamlined NTRU Prime, results in the 15% decrease in the maximum clock frequency, and increase in the number of LUTs by approximately a factor of two. The number of storage elements, flip-flops and BRAMs, remains approximately the same. Supporting operations in S/3 for NTRU-HRSS and R/3 for Streamlined NTRU Prime requires 26.4%

Table 5. Area overhead of special operations of NTRU-based KEMs.

Operations	LUTs	FFs
NTRUEncrypt		
Logic supporting multiplication by 1+3F	12.0%	0%
NTRU-HRSS		
Logic supporting operations in S/3	26.4%	16.4%
Streamlined NTRU Prime		
Logic supporting operations in R/3	8.4%	9.3%
Logic supporting mod q	38.0%	0%
NTRU LPRime		
Logic supporting mod q	53.2%	0%
Logic for register A	0%	24.9%

and 8.4% of the total number of accelerator LUTs, respectively. The percentage is larger in NTRU-HRSS primarily because of the smaller total area required by this KEM. The resource utilization in absolute area units (LUTs, FFs) is comparable. Supporting special multiplication by 1+3F in NTRUEncrypt costs about 12% of the total number of LUTs, and the special register A in NTRU LPRime requires about 25% more flip-flops.

Table 6. Timing results.

Algorithm	Total SW [ms]	Total SW/ HW [ms]	Total speed-up	Poly Mul SW [ms]	Poly Mul HW [ms]	Poly Mul speed-up	SW part Sped up by HW [%]
Encapsulation							
NTRUEncrypt	0.951	0.239	**4.0**	0.744	0.032	23.2	**78.18**
NTRU-HRSS	3.168	0.112	**28.2**	3.092	0.032	98.1	97.58
Strl NTRU Prime	12.780	0.983	**13.0**	11.847	0.034	348.4	92.70
NTRU LPRime	24.200	0.567	**42.7**	23.694	0.054	441.5	97.91
Decapsulation							
NTRUEncrypt	1.700	0.266	**6.4**	1.493	0.056	26.7	87.80
NTRU-HRSS	9.377	0.129	**72.5**	9.303	0.051	181.2	98.95
Strl NTRU Prime	35.729	0.239	**149.7**	35.546	0.052	678.0	**99.49**
NTRU LPRime	36.046	0.576	**62.6**	35.541	0.068	519.0	98.60

Timing results are summarized in Table 6. For each investigated KEM and each major operation (Encapsulation and Decapsulation), we list the total execution time in software (for the reference software implementations in C running on ARM Cortex-A53 of Zynq UltraScale+ MPSoC), the total execution time in software and hardware (after offloading polynomial multiplications to hardware), and the obtained speed-up. The ARM processor runs at 1.2 GHz, DMA for the communication between the processor and the hardware accelerator at 200 MHz, and the hardware accelerators at the maximum frequencies, specific for the RTL implementations of each algorithm, listed in Table 4. All execution times were obtained through experimental measurements using the setup shown in Fig. 1.

The total speed-up varies from 4.0 for encapsulation in NTRUEncrypt to 149.7 for decapsulation in the Streamlined NTRU Prime. The main reason for such big differences is the percentage of time spent by the respective software implementation for operations offloaded to hardware. For the aforementioned two operations, this percentage varies from 78.18% to 99.49%.

The time required for the polynomial multiplication in hardware is similar for all algorithms, to the large extant because a significant percentage of that time is spent for the DMA initialization and data transfer, and only a small percentage on actual computations. The software/hardware communication overhead is quantified in Table 7. It is defined as the percentage of the total number of clock cycles used for the DMA initialization and the input/output data transfer vs. the total number of clock cycles used by the hardware accelerator. As shown in the respective rows of Table 7, this overhead varies between 78% and 89%.

Table 7. Software/hardware communication overhead.

Feature	NTRU Encrypt	NTRU-HRSS	Streamlined NTRU Prime	NTRU LPRime
Encapsulation				
#cycles for transfer (input + output)	25 + 744	23 + 702	25 + 762	25 + 1523
#cycles for Poly Muls	746	702	765	1,531
#cycles for DMA init	4,908	4,877	5,249	7,654
Total #cycles	6,423	6,304	6,801	10,733
Transfer overhead %	88.39	88.86	88.75	85.74
Decapsulation				
#cycles for transfer (input + output)	769 + 1488	725 + 725	763 + 787	787 + 2285
#cycles for Poly Muls	1,494	2,111	2,296	2,295
#cycles for DMA init	7,442	6,706	6,640	8,330
Total #cycles	11,193	10,267	10,486	13,697
Transfer overhead %	86.65	79.44	78.10	83.24

Table 8. Actual speed-up for Zynq UltraScale+ MPSoC (with Proc. Clk =1.2 GHz, Comm. Clk = 200 MHz, Accel. Clk = Max. Clk Freq from Table 3) vs. estimated speed-up for the case of Special Instructions (SI) of ARM Cortex A53 (Proc. Clk = Comm. Clk = Accel. Clk = 1.2 GHz).

Feature	NTRU Encrypt	NTRU-HRSS	Streamlined NTRU Prime	NTRU LPRime
Encapsulation				
Poly Mul speed-up act	24.56	101.92	349.98	444.03
Poly Mul speed-up SI	471.14	2,079.81	7,328.01	7,387.49
Ratio SI/Actual	19.19	20.41	20.94	16.64
Total speed-up act	3.97	28.24	13.00	42.67
Total speed-up SI	4.51	38.01	13.44	46.80
Ratio SI/A	1.14	1.35	1.03	1.10
Decapsulation				
Poly Mul speed-up act	29.03	192.35	682.91	522.98
Poly Mul speed-up SI	382.07	2,507.91	8,872.67	6,357.21
Ratio SI/Actual	13.16	13.04	12.99	12.16
Total speed-up actual	6.38	72.48	149.67	62.60
Total speed-up SI	7.77	110.68	187.38	70.20
Ratio SI/A	1.22	1.53	1.25	1.12

In spite of this communication penalty, the speed-up for the polynomial multiplication itself is very high. For all KEMs other than NTRUEncrypt, this speed-up exceeds 98. For NTRUEncrypt, it is about 23 for encapsulation and 27 for decapsulation. This lower speed-up can be attributed primarily to the faster software implementation (due to the use of $q = 2^{11}$).

Overall, offloading polynomial multiplication to hardware has substantially changed the ranking of investigated KEMs. In pure software, NTRUEncrypt was by far the most efficient, followed by NTRU-HRSS, and trailed by Streamlined NTRU Prime and NTRU LPRime. In the software/hardware implementation, NTRU-HRSS was the fastest for both basic operations. For encapsulation it was followed by NTRUEncrypt, NTRU LPRime, and Streamlined NTRU Prime, and for decapsulation, by Streamlined NTRU Prime, NTRUEncrypt, and NTRU LPRime. However, when analyzing these results, one needs to keep in mind that NTRU-HRSS provides much lower security level compared to all remaining KEMs (the security strength category 1 vs. 5), and the specifications of these KEMs do not support comparing all of them at the same security level.

Using the actual results for the existing modern embedded systems platform, Zynq UltraScale+ MPSoC, we can also estimate the results for a hypothetical future platform, an ARM processor, equipped with special instructions capable of executing polynomial multiplication. We assume that in such platform, the number of clock cycles required for computations and input/output transfer will remain the same. However, both the Poly Mult and the transfer of data will be performed at the same frequency as the frequency of the processor itself (e.g., 1.2 GHz). We also assume that the DMA initialization is not any longer required.

The speed-ups calculated under such assumptions are referred to as speed-ups for the case of Special Instructions (SI). These speed-ups are summarized and compared with the actual speed-ups (obtained for Zynq UltraScale+ MPSoC) in Table 8. The SI speed-ups for Poly Mult itself exceed the actual speed ups by a factor varying between 16.64 and 20.94 for encapsulation, and between 12.16 and 13.16 for decapsulation. At the same time, the total speed-ups improve for the case of special instructions by much smaller factor, varying between 1.03 and 1.35 for encapsulation, and between 1.12 and 1.53 for decapsulation. As a result, our study can be used as a relatively accurate predictor of the improvements possible by extending a modern ARM processor with special instructions capable of performing the respective variants of Poly Mult.

On the other hand, our current study cannot be used to predict the performance and ranking of the investigated candidates when implemented entirely in hardware. Such implementations can benefit from elimination of the communication overhead between a processor and a hardware accelerator. They may also take advantage of an ability to parallelize some additional operations, other than Poly Mult. At the same time for many auxiliary operations, which are sequential in nature, moving from a processor to reconfigurable fabric, operating at much lower clock frequency, may have either negative or at least negligible effect on the overall performance. As a result, the actual full hardware implementations are required to properly rank candidates in terms of their performance in FPGAs and ASICs.

When it comes to alternative software/hardware implementations, the right side of Table 3, may serve as a starting point for future work. This side, presents the results of profiling for our software/hardware implementations. Only for the NTRU-HRSS decapsulation, Poly Mult remains the most time-consuming operation. For all remaining algorithms it moves to the second or third position in the ranking. The new most time consuming functions, such as generate_r for NTRUEncrypt, small_random_weightw for the Streamlined NTRU Prime - Encapsulation, and small_seeded_weightw for NTRU LPRime are likely to be parallalizable and thus suitable for offloading to hardware. On the other hand, FIPS202_SHA3_512 is mostly sequential, and thus it is likely to offer a lower performance gain when implemented in hardware. Additional factors, such as the development effort, the total size of inputs and outputs of a given function, as well as the area/memory requirements may need to be taken into account when investigating any alternative software/hardware partitioning schemes.

6 Conclusions

Using SW/HW codesign allows the implementers of candidates for new cryptographic standards (such as NIST PQC standards) to substantially reduce the development time compared to the use of purely hardware implementations. The implementers avoid reproducing in hardware the cumbersome and mostly sequential operations required for input/output, as well as multiple auxiliary operations that have a negligible influence on the total execution time. Instead, they can focus on major and most time consuming operations, which can easily contribute about 90% to the total execution time, and are suitable for parallelization. In this study, we have clearly demonstrated the viability of this approach in case of four Round 1 NIST PQC candidates and their major operation, Poly Mult. The obtained results shed a light on the correct ranking of the investigated four NTRU-based KEMs when offloading the most time consuming operations to hardware is a design option.

References

1. NTRU Open Source Project. https://github.com/NTRUOpenSourceProject
2. IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices, P1363.1-2008, March 2009
3. Aysu, A., Yuce, B., Schaumont, P.: The future of real-time security: Latency-optimized lattice-based digital signatures. ACM Transact. Embed. Comput. Syst. (TECS) **14**(3), 43 (2015)
4. Bailey, D.V., Coffin, D., Elbirt, A., Silverman, J.H., Woodbury, A.D.: NTRU in constrained devices. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 262–272. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_22

5. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU Prime, August 2017. https://ntruprime.cr.yp.to/ntruprime-20160511.pdf
6. Chen, C., Hostein, J., Whyte, W., Zhang, Z.: NIST PQ Submission: NTRUEncrypt A lattice based encryption algorithm, May 2018. https://www.onboardsecurity.com/nist-post-quantum-crypto-submission
7. Farahmand, F., Ferozpuri, A., Diehl, W., Gaj, K.: Minerva: automated hardware optimization tool. In: 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–8. IEEE, December 2017
8. Farahmand, F., Sharif, M.U., Briggs, K., Gaj, K.: A high-speed constant-time hardware implementation of NTRUEncrypt SVES. In: 2018 International Conference on Field Programmable Technology (ICFPT) (2018)
9. Ferozpuri, A., Gaj, K.: High-speed FPGA implementation of the NIST round 1 rainbow signature scheme. In: International Conference on ReConFigurable Computing and FPGAs (ReConFig 2018), pp. 1–6. IEEE, December 2018
10. Ghosh, S., Delvaux, J., Uhsadel, L., Verbauwhede, I.: A speed area optimized embedded co-processor for McEliece cryptosystem. In: 23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Delft, Netherlands, 9–11 July 2012, pp. 102–108. IEEE (2012). https://doi.org/10.1109/ASAP.2012.16
11. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) Algorithmic Number Theory, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
12. Howe, J., Oder, T., Krausz, M., Güneysu, T.: Standard lattice-based key encapsulation on embedded devices. IACR Transact. Cryptogr. Hardw. Embed. Syst. **2018**(3), 372–393 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7279
13. Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: NTRU-HRSS-KEM: algorithm specifications and supporting documentation, November 2017. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/NTRU_HRSS_KEM.zip
14. Kamal, A.A., Youssef, A.M.: An FPGA implementation of the NTRUEncrypt cryptosystem. In: 2009 International Conference on Microelectronics - ICM, pp. 209–212, December 2009
15. Koziel, B., Azarderakhsh, R.: SIKE - supersingular isogeny key encapsulation: VHDL implementation, November 2017. https://sike.org
16. Kuo, P.C., et al.: High performance post-quantum key exchange on FPGAs. Cryptology ePrint Archive, Report 2017/690 (2017). https://eprint.iacr.org/2017/690
17. Liu, B., Wu, H.: Efficient architecture and implementation for NTRUEncrypt system. In: 2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1–4, August 2015
18. Liu, B., Wu, H.: Efficient multiplication architecture over truncated polynomial ring for NTRUEncrypt system. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1174–1177, May 2016
19. Migliore, V., Real, M.M., Lapotre, V., Tisserand, A., Fontaine, C., Gogniat, G.: Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using Karatsuba algorithm. IEEE Transact. Comput. **67**(3), 335–347 (2018)
20. National Institute of Standards and Technology: Post-Quantum Cryptography, December 2017. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

21. National Institute of Standards and Technology: Post-Quantum Cryptography: Round 1 Submissions, December 2017. https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
22. Oder, T., Güneysu, T.: Implementing the NewHope-simple key exchange on low-cost FPGAs. In: Fifth International Conference on Cryptology and Information Security, Latin America, La Habana, Cuba, 20–22 September 2017 (2017)
23. Wang, W., et al.: XMSS and embedded systems - XMSS hardware accelerators for RISC-V. Cryptology ePrint Archive, Report 2018/1225 (2017). https://eprint.iacr.org/2017/138.pdf
24. Wang, W., Szefer, J., Niederhagen, R.: FPGA-based Niederreiter cryptosystem using binary Goppa codes. In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 77–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_4. http://caslab.csl.yale.edu/code/niederreiter/