Software/Hardware Codesign of the Post Quantum Cryptography Algorithm NTRUEncrypt Using High-Level Synthesis and Register-Transfer Level Design Methodologies

Farnoud Farahmand, Duc Tri Nguyen, Viet B. Dang, Ahmed Ferozpuri, and Kris Gaj
Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, U.S.A.
{ffarahma, dnguye69, vdang6, aferozpu, kgaj}@gmu.edu

***Abstract*—When quantum computers become scalable and reliable, they are likely to break all public-key cryptography standards, such as RSA and Elliptic Curve Cryptography. The projected threat of quantum computers has led the U.S. National Institute of Standards and Technology (NIST) to an effort aimed at replacing existing public-key cryptography standards with new quantum-resistant alternatives. In December 2017, 69 candidates were accepted by NIST to Round 1 of the NIST Post-Quantum Cryptography (PQC) standardization process. NTRUEncrypt is one of the most well-known PQC algorithms that has withstood cryptanalysis. The speed of NTRUEncrypt 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 speeding up NTRUEncrypt using software/hardware codesign on a Xilinx Zynq UltraScale+ multiprocessor system-on-chip (MPSoC). Polynomial multiplication is implemented in the Programmable Logic (PL) of Zynq using two approaches: traditional Register-Transfer Level (RTL) and High-Level Synthesis (HLS). The remaining operations of NTRUEncrypt are executed in software on the Processing System (PS) of Zynq, using the bare-metal mode. The speed-up of our software/hardware codesigns vs. purely software implementations is determined experimentally and analyzed in the paper. The results are reported for the RTL-based and HLS-based hardware accelerators, and compared to the best available software implementation, included in the NIST submission package. The speed-ups for encryption were 2.4 and 3.9, depending on the selected parameter set. For decryption, the corresponding speed-ups were 4.0 and 6.8. In addition, for the polynomial multiplication operation itself, the speed up was in excess of 75. Our code for the NTRUEncrypt polynomial multiplier accelerator is being made open-source for further evaluation on multiple software/hardware platforms.**

***Index Terms*—Post-Quantum Cryptography, lattice-based, NTRU, hardware/software codesign, High-Level Synthesis**

I. Introduction

Major investment by several big companies, such as Google, IBM, Intel and Microsoft, has led to the record-breaking general-purpose quantum processor with 72 physical qubits, released by Google in March 2018 [1]. It has been predicted that quantum computers will reach the computational power and maturity sufficient to break existing public-key cryptography algorithms by 2031, or earlier [2]. When it happens, all traditional methods of dealing with the growing computational capabilities of potential attackers, such as increasing key sizes, would be futile. The only viable solution is to develop new standards based on algorithms resistant against quantum computer attacks, which at the same time can be executed on traditional computing platforms, such as smartphones, laptops, FPGA-based hardware accelerators, etc. In December 2016, NIST published an official Call for Proposals and Request for Nominations for Public-Key PQC algorithms [3]. A year later, Round 1 of the NIST standardization process commenced [4].

To date, the assessment of candidates has focused primarily on their security, with software efficiency envisioned as the next major evaluation goal. Relatively little progress has been made so far to understand the true potential of these algorithms for efficient hardware and embedded systems implementations. Evaluation challenges include the large number of algorithms, their novelty, complex mathematical descriptions, non-standard fundamental operations, large public/private key sizes, multiple parameter sets, and a multitude of possible optimization targets. A complete, purely hardware implementation of a single candidate may easily take a year-long effort by a qualified graduate student or engineer [5].

In this paper, we present a methodology aimed at substantially reducing the development time for the majority of PQC candidates, with limited influence on the accuracy of the comparison and thus ranking of candidates. This methodology is based on software/hardware (SW/HW) codesign, optionally combined with High-Level Synthesis (HLS). To pave the way for the future implementations of multiple other algorithms (possibly all Round 2 candidates), we demonstrate the viability of this approach using one of the most promising Round 1 PQC candidates, NTRUEncrypt [6]. This candidate is a substantially revised version of the earlier NTRU encryption schemes, such as an early variant proposed in 1996 [7], and its extended version standardized by IEEE in 2008 [8].

We are applying the well-known SW/HW codesign and HLS techniques to the new, mostly unexplored, domain of bench-

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, and by the National Science Foundation under Grant no. CNS-1801512.

marking candidates for new post-quantum cryptography (PQC) standards. Novelty is in comparison to all previous benchmarking efforts undertaken during previous cryptographic competitions, such as AES, SHA-3, CAESAR. During these contests only fully hardware and fully software implementations were compared [5]. Previous work on SW/HW codesign of PQC was not aimed at comparing multiple candidates against each other [9], [10], [11]. What makes our effort different is a focus on developing methodology for efficient and fair evaluation of a large number of candidates, and a close tie of our efforts to the NIST PQC standardization process.

II. Background

A. NTRUEncrypt

NTRUEncrypt has four major parameters (N, p, q, d). We implement two variants of the NIST Round 1 PQC candidate NTRUEncrypt: ntru-pke-443, with N=443, p=3, q=2048, and d=143, and ntru-pke-743, with N=743, p=3, q=2048, and d=247 [6].

The scheme is based on polynomial additions and multiplications in the ring $R = Z_q[X]/X^N - 1$. We use the $*$ to denote a polynomial multiplication in R, which is the cyclic convolution of the coefficients of two polynomials.

During the key generation, the user chooses two random secret polynomials $F \in R$ and $g \in R$, with so called small coefficients. All such polynomials have d coefficients equal to 1, d coefficients equal to -1, and the remaining coefficients equal to 0. The private key f is computed as $f = 1 + p \cdot F$. Please note that in our notation, * denotes the multiplication of two polynomials, and $\cdot$ denotes the multiplication of a polynomial by a number. The public key h is calculated as

$$h = f^{-1} * g \cdot p \ \text{ in } R. \tag{1}$$

The most time consuming operation of encryption is $t = r * h$, where $r \in R$ is a randomly chosen polynomial with small coefficients, and h is a public key (a polynomial with large coefficients). The main operation of the Phase 1 of decryption (denoted as DEC-1) is $m' = f * c = (1+3 \cdot F) * c$, where $F \in R$ is a representation of a private key (a polynomial with small coefficients), and c is a ciphertext (a polynomial with large coefficients). The main operation of the Phase 2 of decryption (denoted as DEC-2) is $t' = r * h$, which is identical to that performed during encryption.

B. Polynomial Multiplication

The algorithm we use for Poly Mult is given below as Algorithm 1. A similar Fast Convolution Algorithm was reported earlier in [12]. The for loop in lines 9-11 corresponds to adding to the temporary polynomial $c = c_{N-1}..c_0$ the input polynomial $a = a_{N-1}..a_0$ rotated by b_i locations to the left, namely, $a <<< b_i = a_{N-b_i-1}..a_0 a_{N-1}..a_{N-b_i}$. Similarly, the for loop in lines 13-15 corresponds to the subtraction of the same value from c.

The operation $c = a * (1 + 3 \cdot b)$ can be performed as

$$a * b + 2 \cdot a * b + a = a * b + a * b << 1 + a. \tag{2}$$

Algorithm 1 Polynomial Multiplication, Poly Mult

1: **Inputs:**
2: Polynomial a(X) with N "big" coefficients in the range $[0, q-1]$.
3: Polynomial b(X) with d coefficients 1 at the locations $b_0, b_1, \ ..., b_{d-1}$ and d coefficients -1 at the locations $b_d, b_{d+1}, \ ..., b_{2d-1}$, where $0 \leq b_i \leq N-1$.
4: **Output:**
5: $c(X) = a(X) * b(X) \, mod \, X^N - 1$
6: **Pseudocode:**
7: **for** $i := 0$ *to* $2d-1$ **do**
8: **if** $i < d$ **then**
9: **for** $j := 0$ *to* $N-1$ **do**
10: $c_j = c_j + a_{j-b_i \, mod \, N}$
11: **end for**
12: **else**
13: **for** $j := 0$ *to* $N-1$ **do**
14: $c_j = c_j - a_{j-b_i \, mod \, N}$
15: **end for**
16: **end if**
17: **end for**

III. Previous Work

A. Hardware Accelerators for NTRUEncrypt

In 2001, Bailey et al. [12] introduced and implemented a Fast Convolution Algorithm for polynomial multiplication, exploiting the sparsity of polynomials, similar to that listed above as Algorithm 1. In [13], Kamal et al. analyzed several implementation options for NTRUEncrypt 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). Liu et al. implemented the truncated polynomial ring multiplier using linear feedback shift register (LFSR) in 2015 [14] and an extended LFSR in 2016 [15]. Their design had a variable execution time and supported all major parameter sets for NTRUEncrypt SVES [8]. The full hardware design of NTRUEncrypt SVES, supporting two major parameter sets, ees1087ep1 and ees1499ep1, has been reported in [16].

B. Software-Hardware Codesign of PQC Algorithms

Some attempts to improve implementations of post-quantum cryptosystems have been made through 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. [9]. Aysu et al. [10] built a high-speed implementation of lattice-based digital signatures using SW/HW codesign techniques. The design targeted the Cyclone IV FPGA family and consisted of a NIOS II processor, a hash unit, and a polynomial multiplier. Migliore et al. [17] presented a SW/HW codesign for the lattice-based Fan-Vercauteren (FV) homomorphic encryption scheme. Taking into account that all aforementioned papers used substantially different platforms, and focused on significantly different algorithms, limited lessons could be learned from these efforts.

C. Use of HLS to Implement Cryptographic Algorithms

The implementation of AES using both RTL and HLS-based approaches was reported in [18]. The HLS/RTL ratios of area and maximum clock frequency were close to 1.00. However, the HLS design required 20% more clock cycles, which caused the reduction in throughput by a factor of 1.19. Additionally, in [19], the RTL and HLS approaches were applied to the implementation and ranking of five final SHA-3 candidates. The relative ranking of candidates was shown to be the same for Altera FPGAs using both approaches, in terms of all three major performance metrics: throughput, area, and throughput to area ratio, and only slightly different for Xilinx FPGAs.

In [20], 16 authenticated ciphers, including the current standard, AES-GCM, and the primary variants of 13 Round 3 CAESAR candidates were evaluated using both approaches. The study has demonstrated a high correlation between the rankings of the evaluated algorithms obtained using the RTL and HLS approach, respectively. In particular, after applying HLS, the algorithm rankings in terms of throughput and throughput to area ratio either remained unchanged or was affected only for algorithms that had a close ranking in RTL.

IV. Methodology

A. Platform and Software

The platform selected for our experiments is Xilinx Zynq UltraScale+ MPSoC XCZU9EG-2FFVB1156E, mounted on the ZCU102 Evaluation Kit from Xilinx. This MPSoC is composed of two major parts sharing the same chip, the Processing System (PS) and the Programmable Logic (PL). The PS includes a quad-core ARM Cortex-A53 Application Processing Unit (APU), out of which, we use only one processor (Core 0 of Cortex-A53), running at the frequency of 1.2 GHz. The PL 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.

B. Partitioning into Software and Hardware

The reference and optimized software implementations of NTRUEncrypt were identical in the submission package for Round 1 of the PQC standardization process [6], and we were not aware of any more optimized software implementation of this algorithm at the time of performing experimental measurements for this paper. The results obtained during experimental profiling of this reference/optimized software implementation, using a single core of ARM Cortex-A53, running with the frequency of 1.2 GHz, are summarized in the left portion of Table I. These results clearly indicate that Poly Mult is the most-time consuming operation for both encryption and decryption. The percentage of the total execution time used by this operation is higher for decryption, and for both encryption and decryption increases with the increase in the security level.

A block diagram of the SW/HW codesign experimental setup is shown in Fig. 1. The hardware portion is composed of the implementation of Poly Mult, extended with Input and Output FIFOs, as well as AXI DMA, for high-speed communication with the PS. The software portion, including all remaining functionality of NTRUEncrypt, is implemented in C and runs in Zynq PS. Timing measurements are performed using an AXI Timer, capable of measuring time in clock cycles of the 200 MHz system clock. Poly Mult can operate at a variable frequency different than that of DMA, which

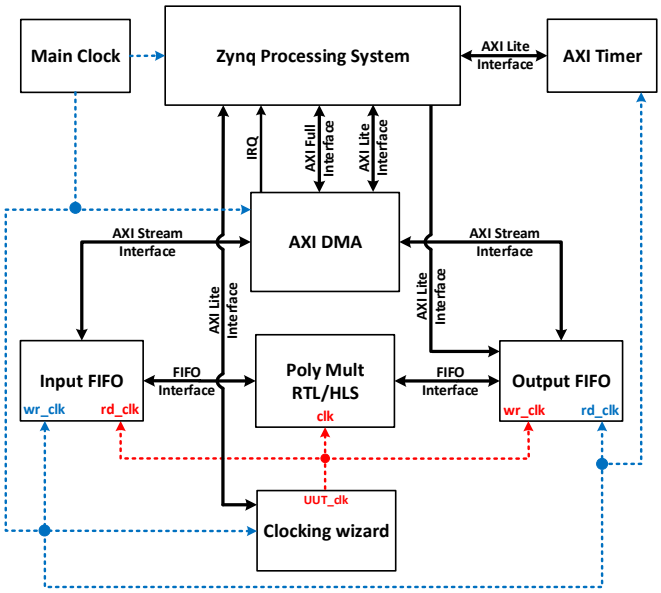

Fig. 1: Block diagram of the SW/HW experimental setup.

TABLE I: Results of profiling for NTRU

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			**Software/Hardware**		
ntru-pke-443 - Encryption					
1.Poly Mult	279.7	62.3%	1.generate_r	68.1	35.9%
2.generate_r	68.1	15.1%	2.pad_msg	44.4	23.4%
3.pad_msg	44.4	9.9%	3.mask_m	25.9	13.7%
4.mask_m	25.9	5.8%	**4.Poly Mult**	20.1	10.6%
Other	31.2	6.9%	Other	31.2	16.4%
Total	449.3	100.0%	Total	189.7	100.0%
ntru-pke-443 - Decryption					
1.Poly Mult	561.1	80.8%	1.generate_r	67.5	38.8%
2.generate_r	67.5	9.7%	**2.Poly Mult**	41.0	23.5%
3.unmask_m	26.2	3.8%	3.unmask_m	26.2	15.0%
4.lift_msg	10.2	1.5%	4.lift_msg	10.2	5.8%
Other	29.2	4.2%	Other	29.2	16.8%
Total	694.2	100.0%	Total	174.1	100.0%
ntru-pke-743 - Encryption					
1.Poly Mult	744.2	77.7%	1.generate_r	82.5	34.0%
2.generate_r	82.5	8.6%	2.pad_msg	43.4	17.9%
3.pad_msg	43.4	4.5%	3.mask_m	39.7	16.3%
4.mask_m	39.7	4.1%	**4.Poly Mult**	29.6	12.2%
Other	47.5	5.0%	Other	47.5	19.6%
Total	957.2	100.0%	Total	242.6	100.0%
ntru-pke-743 - Decryption					
1.Poly Mult	1,488.7	88.7%	1.generate_r	81.8	33.3%
2.generate_r	81.8	4.9%	**2.Poly Mult**	55.8	22.7%
3.unmask_m	39.8	2.4%	3.unmask_m	39.8	16.2%
4.lift_msg	17.2	1.0%	4.lift_msg	17.2	7.0%
Other	51.1	3.0%	Other	51.1	20.8%
Total	1,678.5	100.0%	Total	245.5	100.0%

can be set at run time. Input and Output FIFOs use different clocks for writing and reading data. This SW/HW codesign platform can be easily adapted for other PQC candidates for hardware acceleration by replacing the Poly Mult unit with a new hardware accelerator. In addition, the aforementioned design has the capability to leverage partial reconfiguration to change the hardware accelerator during the run time.

C. Design of Hardware Using the RTL Methodology

The RTL design of Poly Mult follows closely the block diagram shown in Fig. 2. Positions of 1s and -1s in F, constituting a private key, are assumed to be preloaded to the 512x9 F_RAM, before any decryption begins. Similarly, the coefficients of h, constituting a public key, are assumed to be preloaded to the SIPO with parallel input (SIPO w/PI) before any encryption or decryption starts. All of these coefficients can be stored in Reg_h, and loaded back to SIPO w/PI in a single clock cycle in case SIPO w/PI is used in-between for any operation not involving h.

Each multiplication involves rotating the output of the SIPO w/PI by the number of positions read from r_RAM (for ENC and DEC-2) or F_RAM for (DEC-1). The result of rotation is then either added to or subtracted from the previous contents of t_m_PISO (denoted by sum_fb). The conditional subtraction is accomplished by one's complementing using an XOR with c0v (c0 replicated 11 times) and the addition of c0 as the carry in of the adders represented by squares with +. The multiplication $m' = f * c = (1+3 \cdot F) * c$, performed during DEC-1, requires on top of that adding sum_fb shifted by 1 to the left and adding the contents of SIPO w/PI rotated by 0 positions (in agreement with Eq. (2)). The multiplication $t = r * h$, performed during ENC and DEC-2, takes $1 + 2 \cdot d + \#pipeline\_stages - 1$ clock cycles. The multiplication $m' = f * c = (1 + 3 \cdot F) * c$, performed during DEC-1, requires two additional clock cycles.

The rotation, and thus, the entire multiplication, can be significantly sped-up by introducing pipeline registers inside of the rotator, as shown in Fig. 3. Rectangles with dotted lines represent all possible positions of pipeline registers. Rectangles with solid lines represent four positions (0, 3, 6, and 8), which were determined optimal for the ntru-pke-443 RTL design. Thus, together with the combinational logic following the rotator, the design contains 5 pipeline stages. 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, with the format shown in Fig. 4.

The RTL code was first verified using a comprehensive testbench written in VHDL. This testbench applied not only a correct sequence of inputs, but also an incorrect sequence, generating error codes, such as missing public key, missing private key, ENC after DEC-1 (DEC-2 expected), etc. The RTL code of Poly Mult was then integrated with the remaining components of the SW/HW codesign shown in Fig. 1, and tested experimentally again for correct functionality.

D. Design of Hardware Using the HLS Methodology

The reference implementation of NTRUEncrypt in C, for N=443 and N=743, is based on the Grade School algorithm for multiplication (also known as Schoolbook, Paper-and-Pencil, etc.) [6]. 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 [21].

As a result, the decision was made to treat C like a hardware description language, and implement Algorithm 1 from scratch, in such a way to infer the circuit from Fig. 2. This attempt appeared to be successful, which was indicated by reaching exactly the same number of clock cycles as that required by the RTL implementation. Furthermore, introducing pipeline stages inside of the rotator was easily controlled by static variables. This way, multiple placements of pipeline registers could be easily attempted, and led to an optimal choice from the point of view of the total latency expressed in ns. This choice appeared to be the same as in RTL.

Although heavy refactoring of the HLS code is not normally considered acceptable for a widespread use of HLS, our use scenario is much narrower. Our target audience are members of other cryptographic engineering groups, including software programmers already familiar with concurrent computing concepts, e.g., through the use of special instructions, such as AVX2, and special platforms, such as GPUs. The reference code enhanced with HLS directives gave an execution time in the range of N^2 cycles. The VHDL RTL code gave an execution time equal to approximately N cycles. The only way to remove this inefficiency was to refactor the C code, by using coding techniques aimed at inferring a circuit with a specific structure. These techniques cannot be described in the paper in full detail due to space constraints, but they should become evident to all readers after the review of the full optimized code [22]. The refactoring effort was still several times shorter than the RTL design, mostly due to more efficient verification phase, with testbenches written in C.

The C code used as an input to Vivado HLS was first verified using a C testbench, based on the Grade School multiplication method. 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, the remaining numbers did not match the final post-place & route results. In this paper, only results obtained after placing and routing are reported.

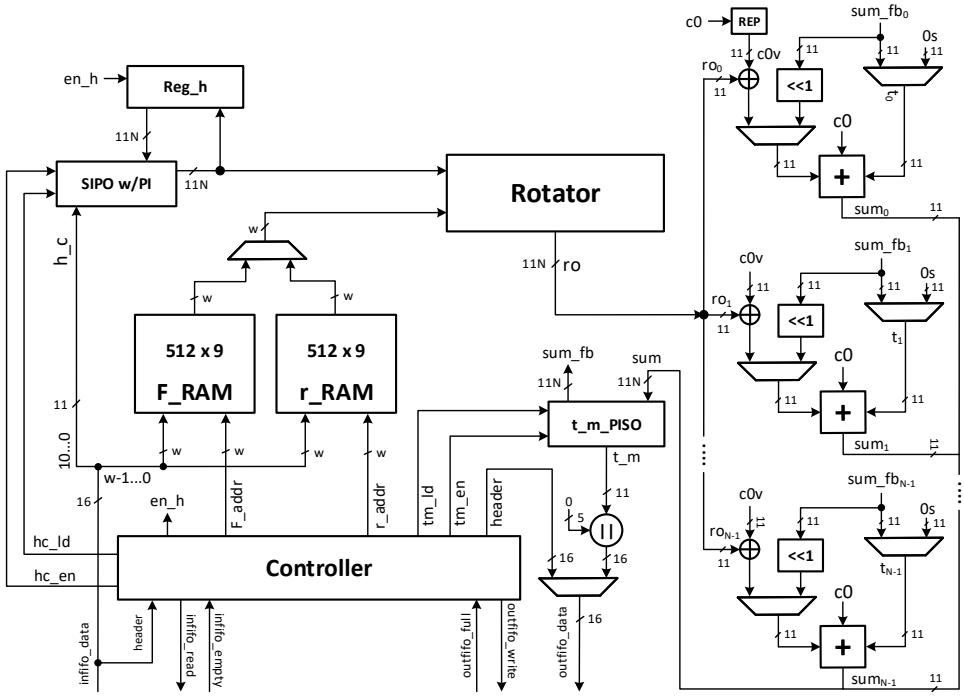

Fig. 2: Block diagram of the hardware accelerator, implemented in RTL and inferred in HLS. REP denotes a unit replicating a single bit 11 times. w=9 for ntru-pke-443 and w=10 for ntru-pke-743.

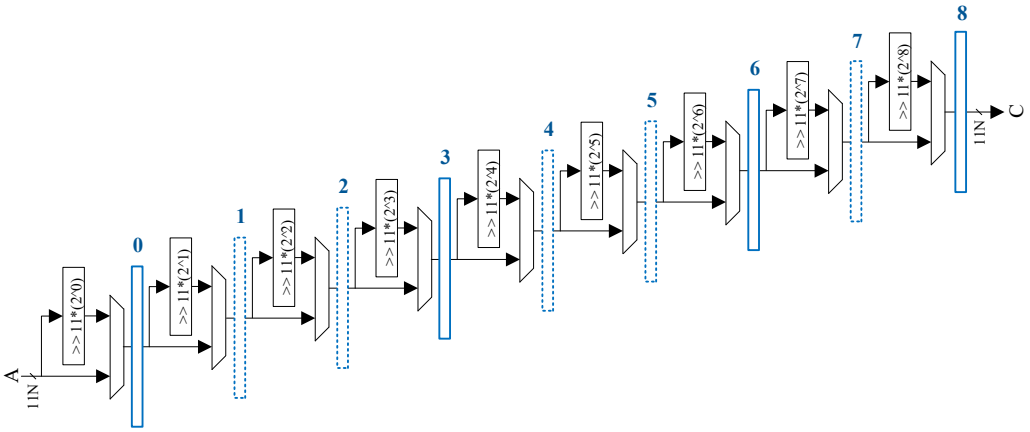

Fig. 3: Possible locations of pipeline registers inside of the Rotator. Rectangles with solid lines represent positions of registers optimal from the point of view of the minimal latency of the implementation of ntru-pke-443 Poly Mult, expressed in ns.

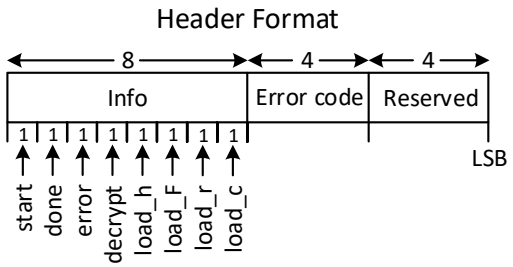

Fig. 4: Format of a header used to pass instructions from software to the hardware accelerator, as well as status and error codes for transmission in the opposite direction.

V. Results

Timing results for the best software and two SW/HW implementations of Poly Mult are summarized in Table II for the ntru-pke-443 and ntru-pke-743 parameter sets. In the SW/HW implementations, software is responsible for converting an array of polynomial coefficients to the corresponding positions of ones and minus ones, passing instructions and positions to the hardware accelerator, and receiving the obtained results and/or status/error codes. All calculations described by Algorithm 1 are performed in hardware. For each parameter set, one of the hardware implementations is developed using RTL, and the other using HLS. The results were generated for the case

TABLE II: Timing analysis of software and SW/HW implementations of Poly Mult operating at maximum supported frequency reported in Table III. AXI DMA and remaining IPs in the system operating at 200 MHz.

Poly Mult Implementation	Total Exe. Time [us]	DMA Transfer Overhead [us]	Poly mult latency [us]	Speed up over Ref.
ntru-pke-443 - Encryption				
SW Opt Ref	279.7	0.0	279.7	1.0
SW/HW RTL	20.1	10.2	3.1	89.1
SW/HW HLS	24.4	10.3	3.4	81.9
ntru-pke-443 - Decryption				
SW Opt Ref	561.1	0.0	561.1	1.0
SW/HW RTL	41.0	16.7	6.8	82.8
SW/HW HLS	41.0	16.8	7.4	76.1
ntru-pke-743 - Encryption				
SW Opt Ref	744.2	0.0	744.2	1.0
SW/HW RTL	29.6	14.0	5.8	128.5
SW/HW HLS	30.9	13.2	7.0	106.8
ntru-pke-743 - Decryption				
SW Opt Ref	1488.7	0.0	1488.7	1.0
SW/HW RTL	55.8	21.5	12.4	119.8
SW/HW HLS	56.7	20.1	15.0	99.6

TABLE III: Comparison of the best achieved RTL and HLS designs for Poly Mult in terms of the maximum supported clock frequency and resource utilization determined using Minerva [23].

	Max Freq	LUTs	FFs	Slices	36kb BRAMs
ntru-pke-443					
RTL	325	44,257	29,655	7,802	1
HLS	300	51,953	49,293	9,413	1
HLS/RTL	0.92	1.17	1.66	1.21	1.00
ntru-pke-743					
RTL	300	76,972	49,674	11,425	1
HLS	250	95,329	82,221	16,686	1
HLS/RTL	0.83	1.24	1.66	1.46	1.00

of all hardware units operating at their respective maximum clock frequencies after placing and routing. These frequencies are summarized in Table III.

For the RTL-based SW/HW implementation, the speed-up over the best software implementation were 89.1x for Encryption and 82.8x for Decryption in case of ntru-pke-443, and 128.5x for Encryption and 119.8x for Decryption in case of ntru-pke-743. For the HLS-based SW/HW implementation, the speed-ups were smaller by 8% for ntru-pke-443 and by 16% for ntru-pke-743. This drop was caused by a lower frequency of the HLS Poly Mult Core. However the difference was minimized by the predominant contribution of the Data Transfer Overhead to the Total Execution Time (in the range of 35-51%). This overhead was the same for RTL and HLS.

HLS implementations are bigger than RTL implementations in terms of resource utilization, as shown in Table III. In particular, the HLS implementation of ntru-pke-443 requires 1.17x more LUTs and 1.21x more slices. In case of ntru-pke-743, the HLS design requires 1.24x and 1.46x more LUTs and slices respectively. In terms of storage elements, the penalty is bigger, 66%, for registers, but none in case of BRAMs.

The results for the entire encryption and decryption operations are summarized in Table IV. For encryption and decryption, for both parameter sets, the speed-up over the software implementation is practically independent of using HLS vs. RTL, and reaches 2.4x and 3.9x for the ntru-pke-443 and ntru-pke-743 encryption, respectively. Decryption speed-up is higher compared to encryption for both parameter sets, and reaches 4.0x and 6.8x for ntru-pke-443 and ntru-pke-743, respectively. The higher speed-up for decryption is related to the higher percentage of the total execution time in software taken by operations offloaded to hardware, as shown in the left portion of Table I. The right portion of Table I indicates that accomplishing even higher speed-ups might be possible by offloading to hardware the next most time-consuming operations, such as generate_r, pad_msg, mask_m, lift_msg.

TABLE IV: Timing analysis for the entire encryption and decryption operations. Public and private keys are assumed to be precalculated, and preloaded to the appropriate arrays in software and appropriate memories and/or registers in hardware.

Poly Mult Implementation	**Total Exe. time [us]**	**Speed up over Opt. SW**
ntru-pke-443 - Encryption		
SW Opt Ref	449.3	1.0
SW/HW RTL	189.7	2.4
SW/HW HLS	193.9	2.3
ntru-pke-443 - Decryption		
SW Opt Ref	694.2	1.0
SW/HW RTL	174.1	4.0
SW/HW HLS	175.7	4.0
ntru-pke-743 - Encryption		
SW Opt Ref	957.2	1.0
SW/HW RTL	242.6	3.9
SW/HW HLS	244.0	3.9
ntru-pke-743 - Decryption		
SW Opt Ref	1678.5	1.0
SW/HW RTL	245.5	6.8
SW/HW HLS	246.5	6.8

VI. 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 operations that are both most time-consuming and most suitable for parallelization.

In this study, we have clearly demonstrated the viability of this approach in case of the Round 1 NIST PQC candidate NTRUEncrypt, and its major operation, Poly Mult. We have also determined that the use of the HLS vs. RTL implementation approach had a negligible influence on the obtained speed-ups vs. the best software implementation, while at the same time providing quite substantial productivity gains.

REFERENCES

[1] (2019, Mar) Timeline of quantum computing. [Online]. Available: https://en.wikipedia.org/wiki/Timeline_of_quantum_computing

[2] L. Vandersypen. (2017, June) A "Spins-inside" Quantum Processor. Invited talk at PQCrypto 2017, June 26-28, 2017, Utrecht, Netherlands. [Online]. Available: https://2017.pqcrypto.org/conference/slides/vandersypen.pdf

[3] "Post-Quantum Cryptography: Call for Proposals," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals, 2019.

[4] National Institute of Standards and Technology. (2019, March) Post-Quantum Cryptography: Project Overview. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[5] K. Gaj, "Challenges and Rewards of Implementing and Benchmarking Post-Quantum Cryptography in Hardware," in *2018 Great Lakes Symposium on VLSI, GLSVLSI 2018*. Chicago, IL, USA: ACM Press, 2018, pp. 359–364.

[6] NTRUEncrypt Submission Team, "Round 1 Submissions - NTRUEncrypt candidate submission package," https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions, 2017.

[7] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory*. Springer, 1998, pp. 267–288.

[8] "IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices, P1363.1-2008," March 2009.

[9] S. Ghosh, J. Delvaux, L. Uhsadel, and I. Verbauwhede, "A Speed Area Optimized Embedded Co-processor for McEliece Cryptosystem," in *2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2012*. Delft, Netherlands: IEEE, Jul. 2012, pp. 102–108.

[10] A. Aysu, B. Yuce, and P. Schaumont, "The Future of Real-Time Security: Latency-Optimized Lattice-Based Digital Signatures," *ACM Transactions on Embedded Computing Systems*, vol. 14, no. 3, pp. 1–18, Apr. 2015.

[11] W. Wang, J. Szefer, and R. Niederhagen, "FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes," in *9th International Conference on Post-Quantum Cryptography, PQCrypto 2018*, ser. LNCS, T. Lange and R. Steinwandt, Eds., vol. 10786. Fort Lauderdale, Florida: Springer International Publishing, Apr. 2018, pp. 77–98.

[12] D. V. Bailey, D. Coffin, A. Elbirt, J. H. Silverman, and A. D. Woodbury, "NTRU in Constrained Devices," in *Cryptographic Hardware and Embedded Systems — CHES*. Springer, 2001, pp. 262–272.

[13] A. A. Kamal and A. M. Youssef, "An FPGA implementation of the NTRUEncrypt cryptosystem," in *2009 International Conference on Microelectronics - ICM*, Dec 2009, pp. 209–212.

[14] B. Liu and H. Wu, "Efficient architecture and implementation for ntru-encrypt system," in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2015, pp. 1–4.

[15] ——, "Efficient multiplication architecture over truncated polynomial ring for NTRUEncrypt system," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 1174–1177.

[16] F. Farahmand, M. U. Sharif, K. Briggs, and K. Gaj, "A High-Speed Constant-Time Hardware Implementation of NTRUEncrypt SVES," in *2018 International Conference on Field-Programmable Technology, FPT 2018*. Naha, Okinawa, Japan: IEEE, Dec. 2018, pp. 190–197.

[17] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/Software Co-Design of an Accelerator for FV Homomorphic Encryption Scheme Using Karatsuba Algorithm," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 335–347, Mar. 2018.

[18] E. Homsirikamol and K. Gaj, "Can High-Level Synthesis Compete Against a Hand-Written Code in the Cryptographic Domain? A Case Study," in *Reconfigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, Dec 2014.

[19] ——, "Hardware Benchmarking of Cryptographic Algorithms Using High-Level Synthesis Tools: The SHA-3 Contest Case Study," in *Applied Reconfigurable Computing*. Springer, 2015, pp. 217–228.

[20] ——, "Toward a New HLS-based Methodology for FPGA Benchmarking of Candidates in Cryptographic Competitions: The CAESAR Contest Case Study," in *2017 International Conference on Field Programmable Technology (FPT)*, Dec 2017, pp. 120–127.

[21] (2019, March) NTRU Open Source Project. [Online]. Available: https://github.com/NTRUOpenSourceProject

[22] F. Farahmand, V. B. Dang, and D. T. Nguyen, "GMU Source Code of PQC Algorithms: Round 1 NIST PQC Candidate NTRUEncrypt," https://cryptography.gmu.edu/athena/index.php?id=PQC, Aug. 2019.

[23] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated Hardware Optimization Tool," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Dec 2017, pp. 1–8.