Implementing and Benchmarking Three Lattice-based Post-Quantum Cryptography Algorithms Using Software/Hardware Codesign

Viet B. Dang*, Farnoud Farahmand*, Michal Andrzejczak†, Kris Gaj*
* Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, U.S.A.
{vdang6, ffarahma, kgaj}@gmu.edu
† Military University of Technology, Warsaw, Poland
{michal.r.andrzejczak}@gmail.com

***Abstract*—It has been predicted that within the next ten-fifteen years, quantum computers will have computational power sufficient to break current public-key cryptography schemes. When that happens, all traditional methods of dealing with the growing computational capabilities of potential attackers, such as increasing key sizes, will be futile. The only viable solution is to develop new standards based on algorithms that are resistant to quantum computer attacks and capable of being executed on traditional computing platforms, such as microprocessors and FPGAs. Leading candidates for new standards include lattice-based post-quantum cryptography (PQC) algorithms. In this paper, we present the results of implementing and benchmarking three lattice-based key encapsulation mechanisms (KEMs) that have progressed to Round 2 of the NIST standardization process. Our implementations are based on a software/hardware codesign approach, which is particularly applicable to the current stage of the NIST PQC standardization process, where the large number and high complexity of the candidates make traditional hardware benchmarking extremely challenging. We propose and justify the choice of a suitable system-on-chip platform and design methodology. The obtained results indicate the potential for very substantial speed-ups vs. purely software implementations, reaching 28x for encapsulation and 20x for decapsulation.**

***Index Terms*—post-quantum cryptography, lattice-based, software/hardware codesign, system on chip**

I. Introduction

Hardware benchmarking has played a major role in all recent cryptographic standardization efforts, including the AES, eSTREAM, SHA-3 [1]–[4], and CAESAR contests [5], [6]. Traditionally software and hardware benchmarking were conducted separately by different groups of experts equipped with different knowledge and tools. For the post-quantum cryptography (PQC) standardization process [7], this approach is hard to maintain. Algorithms competing to become new PQC standards are too complex and too different from the current state-of-the-art in public-key cryptography to permit the development of optimized purely-hardware implementations for a significant fraction of the remaining candidates by any single group within the time frame imposed by the NIST evaluation process.

Since the publication of Round 1 candidates in December 2017, only a few purely-hardware implementations of these candidates have been reported: [8]–[12], and even fewer are open source. These implementations use different APIs, target different platforms, and are aimed at different optimization targets (e.g., high speed, low area). No conclusions regarding the ranking of these algorithms in terms of their performance in hardware can be reached based on such divergent efforts.

In this paper, we present an alternative approach to evaluating candidates in cryptographic contests, based on software/hardware codesign. This technique has been used for years in the industry and studied extensively in academia, with the goal of reaching performance targets using a shorter development cycle. To the best of our knowledge, no benchmarking of software/hardware designs was reported during any previous cryptographic competitions. As a result, multiple problems specific to cryptographic contests, such as the choice of the most representative platform(s) and the fairness of software/hardware partitioning schemes, have never been addressed. It should be clearly stated that software/hardware benchmarking is not intended as a replacement for purely-hardware benchmarking. On the contrary, applying this approach to the 26 candidates advanced to Round 2, and developing a library of hardware accelerators for major operations of these candidates, will make it much easier to develop hardware-only implementations in subsequent rounds.

Within the proposed framework, the first issue to address is the choice of a representative device. In particular, we need a computing platform allowing fast communication across the software/hardware boundary. We also need reconfigurable hardware, as timing measurements must be performed experimentally, and the platform must be well-suited for attempting various software/hardware partitioning schemes.

In recent years several such platforms have emerged. The most popular in the industry are those based on integrating an ARM-based processor and FPGA fabric on a single chip. These devices support software/hardware codesigns based on a program written in a traditional high-level language and running on an ARM processor, with the most time-critical computations of the algorithm performed on a dedicated hardware accelerator. The advantages of these platforms include: the use of the most popular embedded processor family (ARM) operating at high speed (1 GHz or above), state-of-the-art

commercial tools (available for free, or at a reduced price for academic use), availability of inexpensive prototyping boards, and practical deployment in multiple environments.

The primary alternatives are FPGA-based systems with "soft" processor cores implemented in reconfigurable logic. Examples include Xilinx MicroBlaze, Intel Nios II, and the open-source RISC-V, originally developed at the University of California, Berkeley [13]–[15]. The main advantage of these systems over "hard" processor cores is flexibility in the allocation of resources to processor cores, including the possibility of extending them with special instructions specific to PQC. Additionally, the developed designs are easy to port between different FPGA families, and even between FPGAs and ASICs. A disadvantage compared to the "hard" option is that the "soft" processors operate at much lower clock frequencies (typically 200-450 MHz). During this study, we based our choice of platform primarily on the projected practical importance during the initial period of deployment of new PQC standards and the expected speed-up over purely-software implementations. These priorities led us to choose devices from the "hard" processor class, among them the Zynq UltraScale+ family from Xilinx Inc.

With the preferred platform identified, our second major concern was the fairness of software/hardware benchmarking, especially in terms of deciding which operations within each evaluated scheme should be offloaded to hardware. In this paper, we propose a comprehensive approach to address this issue, aimed at achieving the best possible trade-off between the speed-up (compared to software-only implementation) and the required development time. The proposed methodology was applied to the evaluation of three key encapsulation mechanisms (KEMs) belonging to three different Round 2 PQC submissions (FrodoKEM [16], Round5 [17], and Saber [18]).

II. Basic Features of Compared Algorithms

FrodoKEM, Round5, and Saber are based on the Learning with Errors (LWE), General Learning With Rounding (GLWR), and Module Learning with Rounding (Mod-LWR) problems, respectively. The implemented variant of Round5 relies specifically on the RLWR (Ring Learning With Rounding) variant of GLWR. Major parameters and auxiliary functions of the investigated algorithms are summarized in Table I.

In all three schemes, the elementary operation is multiplication in Z_q, where q is a power of two. In FrodoKEM the most time-consuming operation is a matrix-by-matrix multiplication, where each element of a matrix is an element of Z_q. In the implemented variant of Round5 the most time-consuming operation is a polynomial multiplication, where the n coefficients of one polynomial are elements of Z_q, and the n coefficients of another are in the set {-1, 0, 1}. In Saber, the most time-consuming operations are matrix-by-vector and vector-by-vector multiplications, where each element of a matrix or a vector is a polynomial with n coefficients in Z_q. Additionally, all three algorithms use SHAKE [19] or cSHAKE [20] as an auxiliary cryptographic operation. Saber uses SHA3-256 and SHA3-512 in addition to SHAKE128.

TABLE I
Parameter sets of investigated algorithms

Algorithm	Security Category	Degree n	Modulus q	Auxiliary Functions
FrodoKEM	1	640	2^{15}	SHAKE128
Round5	1	586	2^{13}	cSHAKE128
Saber	1	256	2^{13}	SHAKE128 SHA3-256 SHA3-512
FrodoKEM	3	976	2^{16}	SHAKE256
Round5	3	852	2^{12}	cSHAKE256
Saber	3	256	2^{13}	SHAKE128 SHA3-256 SHA3-512
FrodoKEM	5	1344	2^{16}	SHAKE256
Round5	5	1170	2^{13}	cSHAKE256
Saber	5	256	2^{13}	SHAKE128 SHA3-256 SHA3-512

For a fair comparison, all schemes and parameter sets must have approximately the same resistance against all known attacks. Since FrodoKEM and Saber are by default key encapsulation mechanisms (KEMs) with indistinguishability under chosen ciphertext-attack (IND-CCA) [21], [22], we chose a variant of Round5 with the same IND-CCA security. Additionally, we compared implementation results only for parameter sets belonging to the same security category, as defined by NIST in [23].

III. Previous Work

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 soft core and several parallel acceleration units for the code-based McEliece cryptosystem was implemented on Spartan-3AN FPGAs by Ghosh et al. [24]. Aysu et al. [25] built a high-speed implementation of a lattice-based digital signature scheme using SW/HW codesign techniques. The design targeted the Cyclone IV FPGA family and consisted of the NIOS II soft processor, a hash unit, and a polynomial multiplier. Wang et al. [9] reported a SW/HW implementation of the hash-based digital signature scheme XMSS. The selected platform was an Intel Cyclone V SoC, and the software part of the design was implemented using a RISC-V soft-core processor. All the platforms mentioned above differed substantially from the platform used in this work, and the algorithms and their parameters were also substantially different. As a result, limited information can be inferred regarding the optimal software/hardware partitioning, expected speed-up, or expected communication overhead.

SW/HW implementations of three NIST PQC Round 1 candidates (NTRUEncrypt, NTRU-HRSS, and NTRU Prime), using a similar platform to that used in this paper, were reported in [26]. The implemented algorithms differ substantially from those investigated in this paper.

To the best of our knowledge, no hardware implementations of Round5 or Saber have been reported to date.

For FrodoKEM, a full FPGA implementation was presented in [10]. This design exploits a DSP unit to implement matrix-by-vector and matrix-by-matrix multiplications. The matrices are generated on-the-fly; the next matrix row is generated during the computations involving the current row. In our design, by increasing the number of DSPs used in the matrix operations, we were able to reach the limit when the matrix multiplication speed is nearly equal to the matrix generation speed.

IV. Methodology

A. Software/Hardware Codesign Platform

Our target platform is the Xilinx ZCU102 Evaluation Kit, based on the Xilinx Zynq UltraScale+ MPSoC XCZU9EG-2FFVB1156E device. This device has two major parts located on the same chip: a Processing System (PS) and Programmable Logic (PL). The primary component of the PS is a quad-core ARM Cortex-A53 Application Processing Unit, running at 1.2 GHz. As in the software benchmarking experiments conducted by other groups, we utilize only one core in all our experiments. The PL includes a programmable FPGA fabric similar to that of Virtex UltraScale+ FPGAs. The frequency of operation depends on the particular logic instantiated in the reconfigurable fabric but typically does not exceed 400 MHz. The software used to implement our solution is the Xilinx Vivado Design Suite HLx Edition, the Xilinx Software Development Kit, and Xilinx Vivado HLS, all with version numbers 2018.2.

A high-level block diagram of the experimental software/hardware codesign platform is shown in Fig. 1. The Hardware Accelerator is connected through the dual-clock Input and Output FIFOs to the AXI DMA, supporting high-speed communication with the Processing System. Timing measurements are performed using the popular Xilinx LogiCORE IP unit AXI Timer, which is capable of measuring time in clock cycles of the 200 MHz system clock. The Hardware Accelerator can operate at a variable clock frequency, controlled from software using the Clocking wizard unit.

B. Software/Hardware Partitioning

Our first step in evaluating the suitability of cryptographic algorithms for software/hardware codesign was to profile their software implementations using one core of the ARM Cortex-A53. Profiling produces a list of the most time-consuming functions, including their absolute execution time, percentage of the total execution time, and the number of times they are called. We decided which functions to offload to hardware based on the highest potential for overall speed-up, as well as the fairness of comparison. The total speed-up obtained by offloading an operation to hardware depends on two major factors: the percentage of the execution time taken in software by the operation offloaded to hardware, and the speed-up for the offloaded operation itself.

In order to maximize the first factor, we gave priority to operations that take the largest percentage of the execution time, preferably more than 90%. These operations may involve

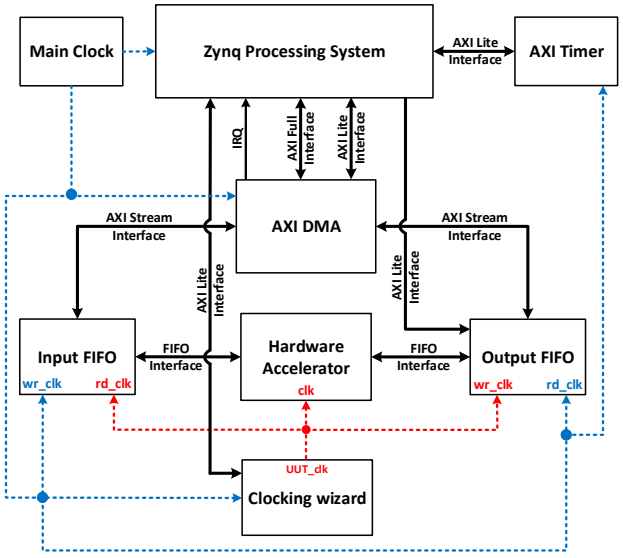

Fig. 1. Block diagram of software/hardware codesign.

a single function call, several adjacent function calls, or a sequence of consecutive instructions in C. It is preferred that a given operation is executed only a few times (ideally only once), as each transfer of control and data between software and hardware involves a certain fixed timing overhead, independent of the size of input and output to the accelerator. In order to maximize the second factor, we gave priority to operations that have a high potential for parallelization in hardware, and a small total size of inputs and outputs (which will need to be transferred to and from the hardware accelerator, respectively).

Most of the data required to make informed decisions regarding software/hardware partitioning can be obtained by profiling the purely-software implementation, possibly extended with some small modifications required to gather all relevant data. However, determining the potential for parallelization requires some knowledge of hardware implementation, or at least of the basic concepts of concurrent computing.

In order to assure fairness in our comparison, we decided to offload to hardware all operations common to or similar across the implemented algorithms (e.g., all polynomial multiplications, all hash-based functions), as well as all operations that contributed significantly to the total execution time. Nevertheless, it should be understood that this heuristic procedure may need to be repeated several times because, after each round of offloading, different software operations may emerge as taking the majority of the total execution time. This iterative process can stop when the development effort required for offloading the next most-critical operation to hardware is disproportionately high compared to the projected speed-up.

C. Interface of Hardware Accelerators

The interface of a hardware accelerator matches the interface of the Input and Output FIFOs. The default width of the data bus is 64 bits. Each operation (e.g., load public

key, start encapsulation) is initiated by sending an appropriate header (in the form of a single 64-bit word) from a program running on the ARM processor to the data input of a hardware accelerator. When an operation requires additional data, this data is transmitted using the subsequent Input FIFO words. After the hardware accelerator produces results or detects an error, a header word is sent in the opposite direction. If additional output data is required, it follows the header and is arranged in 64-bit words. The specific format of the exchanged inputs and outputs is left up to the designer of a hardware accelerator.

D. Verification and Generation of Results

Functional verification of the hardware description language (HDL) code is performed by comparing simulation results with precomputed outputs generated by a reference software implementation. Fully verified and independently optimized VHDL code is then combined with the *optimized* software implementation of a given PQC candidate. Functional verification of the integrated software/hardware design is performed by running the code on the prototyping board and comparing the obtained outputs with outputs generated by a functionally equivalent reference implementation, run on the same ARM Cortex-A53 processor. Experimental timing measurements follow, with the hardware accelerator's clock set (using the Clocking wizard) to the optimal target frequency identified during the synthesis and implementation runs.

V. Hardware Accelerators

A. FrodoKEM

The top-level block diagram of the hardware accelerator for FrodoKEM is shown in Fig. 2. The public key consists of a 128-bit $seed\_A$ and an unpacked public-key matrix B with dimensions n x 8 log_2q-bit words. Both of these elements are assumed to be loaded into the respective memories of the hardware accelerator, `Seed_Asm_Mem` and `Matrix_A_and_B_Dual_Mem` before encapsulation or decapsulation starts.

During encapsulation, the public key is hashed, then the concatenation of the hashed public key and a uniformly random μ (sent from the processor) is hashed to obtain $seed\_SE$. The 256-bit $seed\_SE$ is stored in the asymmetric memory `Seed_Asm_Mem`, with the 8-bit data input and the 64-bit data output. SHAKE128/256 is used to generate a pseudorandom sequence $r^{(0)}..r^{(\overline{m}n-1)}$. This sequence is then fed to `Sampler`, which produces a w-bit output for every 16-bit input word. The first 15 bits represent an unsigned number, and the least significant bit is the sign bit. The noise is sampled by comparing the unsigned number with all values stored in a discrete cumulative distribution table. The smallest index for which the corresponding table entry is larger than the input number is returned as the value of the noise. The Sampler is implemented using a set of comparators and is a purely combinational circuit. The obtained samples, representing the coefficients of the vector S', are stored in the asymmetric memory `Matrix_S'_Asym_Mem`.

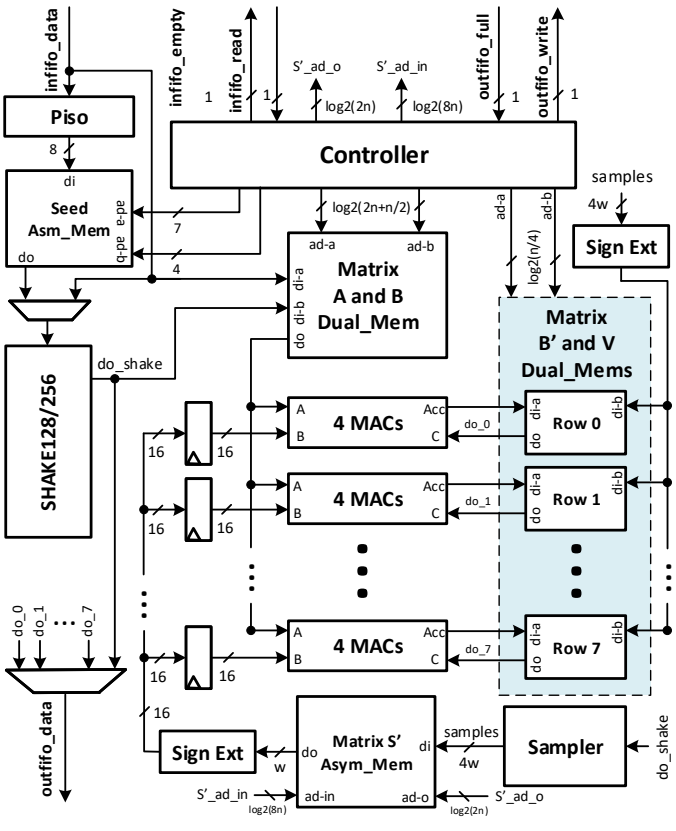

Fig. 2. Block diagram of the hardware accelerator for FrodoKEM. All bus widths are 64-bit unless specified.

The sequence of words generated by SHAKE128/256 is fed to `Sampler`. The output of `Sampler` is stored as subsequent coefficients of E', in the memory `Matrix_B'_and_V_Dual_Mems`.

Subsequently, SHAKE128/256 is used to generate elements of the n x n matrix A, with each element expressed using log_2q bits. In order to reduce execution time and the size of the `Matrix_A_and_B_Dual_Mem` memory, only one row of the A matrix is generated at a time. This row is used for the computations of $B' = S'A + E'$, in parallel with calculating the subsequent row of A. The elements of A are multiplied by the corresponding 8 elements from a column of S', read from `Matrix_S'_Asym_Mem`, sign-extended to 16 bits, and stored in one of the eight registers preceding the 4MAC units. The temporary results are stored back in the `Matrix_B'_and_V_Dual_Mem`. B' is then transferred back to the processor using the `outfifo_data` bus. After the subsequent computation $V = S'B + E''$, V is transferred to the processor for further computations in software. The final results received from software are sent to the hardware SHAKE128/256 unit to compute the shared secret.

The operations performed by the hardware accelerator during decapsulation are identical to those executed during encapsulation (with B' replaced by B'').

B. Round5

The main computations of Round5 are performed in the polynomial ring $\mathbb{Z}_q[x]/(\Phi_{n+1}(x))$. The most time consuming operation is multiplication in the aforementioned ring, described by the equation

$$c_k = \sum_{i+j \equiv k \mod n} a_i \cdot b_j \mod q \quad (1)$$

This operation is executed twice during encapsulation and three times during decapsulation. Moreover, a polynomial multiplication in this scheme can be implemented more efficiently than in the general case, due to a special form of one of the polynomials. In each Round5 multiplication, one of the polynomials is always a ternary polynomial, i.e., all of its coefficients belong to the set $\{-1, 0, 1\}$. In this case, the multiplication is reduced to the addition or subtraction of coefficients of the second polynomial.

The entire encryption and decryption functionality of KEM, as well as calls to cSHAKE for the secret key and public key expansion, are implemented in hardware. This approach allows the generation of the majority of polynomials used in multiplication directly in hardware, removing the need for generating them in software and passing through the relatively slow communication channel. The inputs for encryption and decryption are passed to the FPGA fabric directly, without unpacking by CPU. The (un-)packing functions, based on bit-shifting operations, are implemented in hardware. These operations are very inexpensive in hardware. Thus, the speed-up comes from the faster execution of cSHAKE in hardware, as well as the lower communication overhead, achieved by sending only the seed for cSHAKE, instead of the expanded data. The remaining operations, such as rounding, addition, and subtraction, are also fast and cost-efficient in hardware, providing additional speed-up. Thus, with the little additional area, the design is able to execute encryption and decryption on the input data and return results in the packed format.

The top-level block diagram of *r5_cpa_pke* is shown in Fig. 3. The required data is read from the input FIFO using the port `infifo_data`. If the incoming data is a seed for expansion, it is passed directly to the `cSHAKE` unit. Otherwise, it is passed to an input port of one of the two arithmetic modules. The main controller is responsible for managing the state of the accelerator. After all required data is received, including the expanded data generated by `cSHAKE`, the controller initializes the arithmetic modules and waits till the end of computations. The last step is to send the result back to the software.

Encryption and decryption are performed by the arithmetic modules `Rounding` and `Poly_Mul`, shaded with colors in Fig. 3. Provided with the necessary data and operation type, the aforementioned modules execute specific instructions. At first, a polynomial multiplication is performed. Based on the operation type, the temporary result may be rounded. During encryption, the message is added to the end of the data flow, before the results are prepared to be sent back to the software.

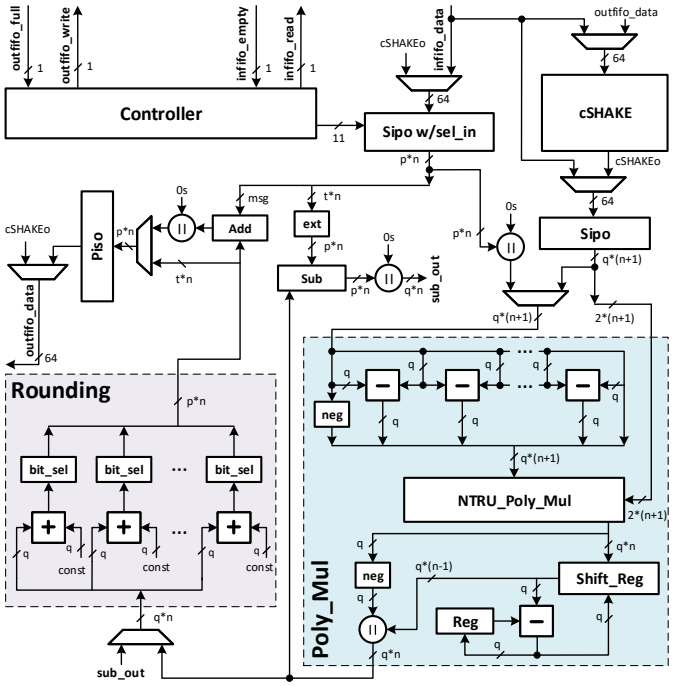

Fig. 3. Block diagram of the hardware accelerator of Round5.

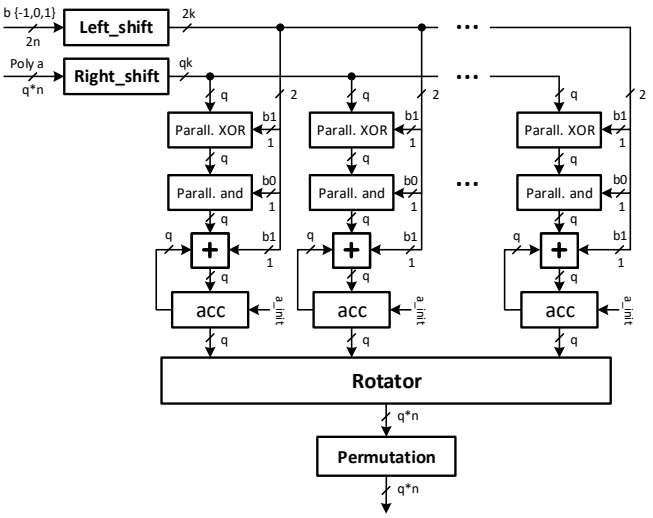

Fig. 4. Block diagram of the Round5 NTRU Poly Mul.

The majority of the area taken by the arithmetic modules is due to Poly Mul, shown in Fig. 4. We utilize the fact that one of the arguments is from the set $\{-1, 0, 1\}$. Thus, the second argument is XOR-ed bit-by-bit, in parallel, with the bit b1, describing the sign of the first argument. Next, the parallel AND operation is performed, with the bit b0, denoting a zero value of the ternary coefficient. The result is passed to an adder and then to an accumulator.

The `NTRU_Poly_Mul` is surrounded with additional logic performing necessary operations to prepare polynomials for multiplication. One of the polynomials is lifted from the ring $\mathbb{Z}_q[x]/(\Phi_{n+1}(x))$ to the ring $\mathbb{Z}_q[x]/(N_{n+1}(x))$. The lifted

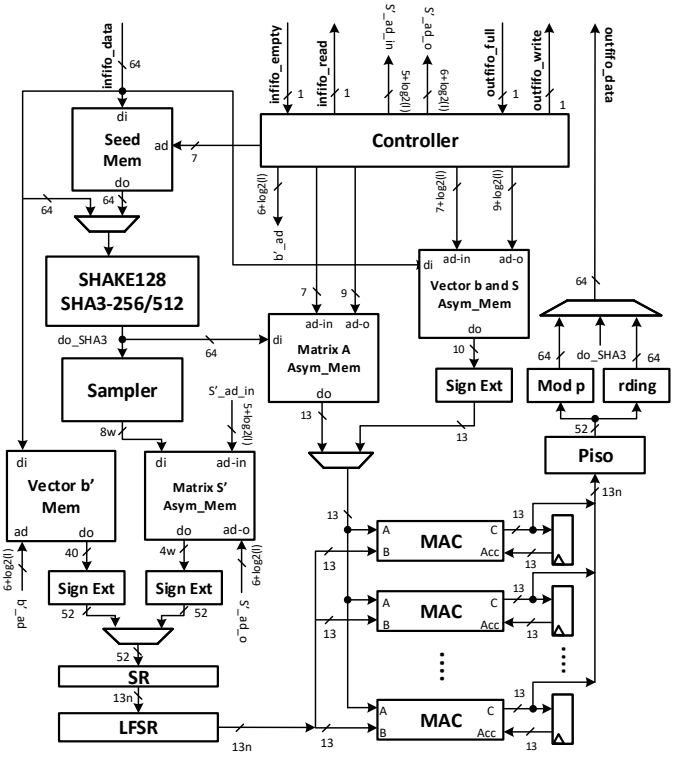

Fig. 5. Block diagram of the hardware accelerator of Saber.

polynomial is equal to

$$-a_0 + (a_0 - a_1)x + (a_1 - a_2)x^2 + \cdots + a_{n-1}x^n \quad (2)$$

There are no dependencies between the operands, so each coefficient is lifted in parallel, and the operation takes exactly one clock cycle. After multiplication, a polynomial is unlifted to the previous ring. Unlifting is computed recursively, as $a_i = a_{i-1} - a_i^L$. Unfortunately, this operation must be performed sequentially and has almost the same latency as multiplication. The second arithmetic module named `Rounding` is responsible for properly shrinking the bit size of coefficients by adding a rounding constant (specific to a given computational step) and applying a proper mask.

C. Saber

The top-level block diagram of the hardware accelerator of Saber is shown in Fig. 5. The public key consists of a 256-bit $seed\_A$ and a vector b, including l polynomials with n=256 coefficients each (where l=2, 3, 4 for the security levels 1, 3, 5, respectively). The polynomial coefficients have $log_2 q$=13 bits for each security level. Both $seed\_A$ and b are assumed to be loaded into the respective memories of the hardware accelerator, `Seed_Mem` and `Vector_b_and_S_Asym_Mem`, using the 64-bit input bus `infifo_data`, before encapsulation or decapsulation starts.

During encapsulation, vector s' is generated first. The random value m and the public key is hashed by SHA3-256. The result is then hashed again using SHA3-512 to obtain the seed r and $\hat{K}'$, which is used to compute the shared secret at the end of the encapsulation. The 256-bit seed r is loaded into `Seed_Mem`. The generation of s' involves `SHAKE128`, followed by `Sampler`, generating w-bit integers using a centered binomial distribution (CBD). The obtained samples, representing the coefficients of the vector s', are stored in the asymmetric memory `Matrix_S'_Asym_Mem`.

Subsequently, `SHAKE128` is used to generate elements of the l x l matrix A, with each element representing a polynomial. In order to reduce the execution time and the size of `Matrix_A_Asym_Mem` memory, only one row of the A matrix is generated at a time. This row is used for the computation of $b' = (As' + h)\ mod\ q$ (where h is a constant), in parallel with the calculation of the subsequent row of A. The elements of A are multiplied by the corresponding elements of s', read from `Matrix_S'_Asym_Mem`, sign-extended to 13-bits, and stored in the n-stage LFSR. The temporary results are stored in the registers shown to the right of `MACs` in Fig. 5. Each coefficient of b' is then shifted right by 3 positions (corresponding to the division by q/p = $2^{13}/2^{10}$ = 8) and transferred back to the processor. In the subsequent calculation, $v' = b^T(s' \bmod p)$, the reduction mod p is performed on the fly, and the result is transferred to the processor for further computations in software. The final results computed in the software are sent to the hardware SHA3-256 unit to calculate the shared secret.

The secret key s is assumed to be loaded before decapsulation starts. In the first phase of decapsulation, a new calculation $v = b'^T s \bmod p$, specific to decapsulation, is performed by the hardware accelerator. b' is a part of the ciphertext, and thus must be loaded prior to the start of decapsulation. In the second phase of decapsulation, the function Saber.PKE.Enc is called, and as a result, the hardware accelerator performs exactly the same operations as during the encapsulation.

VI. Results

The results of profiling for the purely-software implementations, running on a single core of ARM Cortex-A53, with the clock frequency of 1.2 GHz, are presented in the left portions of Table II. For each of the investigated algorithms and each major operation (Encapsulation and Decapsulation), three to six most time-consuming functions are identified. For each of these functions, we provide the execution time in microseconds, and the percentage of the total execution time. In the right portions of the same tables, we list in bold the functions offloaded to hardware. Where functions are combined, they are listed in the same field of the table, with sub-indices, such as 1.1, 1.2, 1.3, etc. Single execution time and a single percentage of the software/hardware execution time is given for such a combined function.

For each investigated KEM and each major operation (Encapsulation and Decapsulation), we list in Table III the total execution time in software (for the optimized software implementations in C running on ARM Cortex-A53 of Zynq UltraScale+ MPSoC), the total execution time in software and hardware (after offloading the most time-consuming operations to hardware), and the obtained total speed-up. The ARM

TABLE II
RESULTS OF PROFILING FOR FRODOKEM, ROUND5 AND SABER

Function	Time [μs]	Time [%]	Function	Time [μs]	Time [%]
Software			**Software/Hardware**		
FrodoKEM : FrodoKEM-1344 : Encapsulation					
1.sa_plus_e	58,577.48	94.36	**1.1 sa_plus_e**	1,328.39	60.76
2. Shake128	1,416.27	2.28	**1.2 Shake128**		
3. sb_plus_e	654.64	1.05	**1.3 sb_plus_e**		
4. hash	569.60	0.92	**1.4 hash**		
5. pack	386.22	0.62	2. pack	386.22	17.67
6. unpack	276.00	0.44	3. unpack	276.00	12.62
Other	195.62	0.32	Other	195.62	8.95
Total	62,075.83	100.00	Total	2,186.23	100.00
FrodoKEM : FrodoKEM-1344 : Decapsulation					
1. sa_plus_e	58,754.02	94.19	**1.1 sa_plus_e**	1,316.52	42.20
2. Shake128	883.14	1.42	**1.2 Shake128**		
3. unpack	765.56	1.23	**1.3 sb_plus_e**		
4. sb_plus_e	649.68	1.04	**1.4 hash**		
5. mul_bs	507.08	0.81	2. unpack	765.56	24.54
6. hash	286.64	0.46	3. mul_bs	507.08	16.25
Other	530.74	0.85	Other	530.74	17.01
Total	62,376.86	100%	Total	3,119.9	100.00
Round5 : R5ND-5PKE_0d : Encapsulation					
1. pke_encrypt	290.81	86.41	**1.1 pke_encrypt**	30.01	94.66
2. hash	44.10	13.10	**1.2 hash**		
3. randombytes	1.52	0.45	3. randombytes	1.67	5.27
Total	336.52	100.00	Total	31.7	100.0
Round5 : R5ND-5PKE_0d : Decapsulation					
1. pke_encrypt	287.10	69.05	**1.1 pke_encrypt**	42.56	100.00
2. pke_decrypt	83.61	20.11	**1.2 pke_decrypt**		
3. hash	45.09	10.8	**1.3 hash**		
Total	415.80	100.00	Total	42.56	100.00
Saber : FireSaber-KEM : Encapsulation					
1. MatrixVecMul	815.40	68.22	**2.1 MatrixVecMul**	50.09	67.33
2. InnerProduct	204.60	17.12	**2.2 InnerProduct**		
3. GenMatrix	92.58	7.75	**2.3 GenMatrix**		
4. Hash	45.82	3.83	**2.4 GenSecret**		
5. GenSecret	12.46	1.04	**2.5 Hash**		
Other	24.31	2.03	Other	24.31	32.67
Total	1,195.17	100.00	Total	74.40	100.00
Saber : FireSaber-KEM : Decapsulation					
1. MatrixVecMul	815.98	59.17	**1.1 MatrixVecMul**	55.14	69.06
2. InnerProduct	408.96	29.65	**1.2 InnerProduct**		
3. GenMatrix	92.60	6.71	**1.3 GenMatrix**		
4. Hash	24.50	1.78	**1.4 GenSecret**		
5. GenSecret	12.44	0.90	**1.5 Hash**		
Other	24.62	1.79	Other	24.62	30.93
Total	1,379.14	100.00	Total	79.84	100.00

processor runs at 1.2 GHz, the 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 IV. All execution times were obtained through experimental measurements using the setup shown in Fig. 1. The speed-up for each component offloaded to hardware is given in the column Accel. Speed-up. This speed-up is the ratio of the execution time of the original component in software (column Accel. SW [ms]) and the execution time of the accelerated component in hardware, including all overheads (column Accel. HW [ms]). The last column indicates the percentage of the software-only execution time that is taken by an accelerated component.

As expected, the total speed-up increases as the security level increases. This dependency exists because, typically, for larger parameter values, a higher level of parallelization can be achieved for the operations offloaded to hardware. Additionally, the operations offloaded to hardware tend to account for a larger percentage of the total execution time in software, as illustrated by the column SW part Sped up by HW [%] in Table III.

The ranking of the algorithms (1. Round5, 2. Saber, 3. FrodoKEM) is identical for a) encapsulation and decapsulation, b) software-only and software/hardware implementations, and c) all three security categories. Only relative ratios among the execution times of the investigated algorithms are affected. For the same security category, the speed-ups are generally the highest for FrodoKEM, followed by Saber, and Round5.

The maximum clock frequencies and the corresponding resource utilization obtained using Minerva [27] are summarized in Table IV. DSP units are utilized in Saber, and to a lower extent in FrodoKEM. Round5 does not involve any integer multiplications in hardware, because the coefficients of one of the multiplied polynomials always belong to the set {-1, 0, 1} and thus require only additions and/or subtractions.

Due to the timing dependencies, and in particular the bottleneck caused by SHAKE, our implementation of FrodoKEM cannot be easily sped up by trading additional resources for speed. This example clearly illustrates the limits imposed by specific algorithms on the amount of potential parallelization (and thus the maximum speed-up), which is independent of the amount of hardware resources available to the designer. FrodoKEM is also the algorithm with the highest utilization of BRAMs, which reaches 17.5. Round5 and Saber use only 2 and 3.5 36kb BRAMs, respectively. Round5 uses the highest number of LUTs, Slices, and flip-flops (FFs), followed by Saber and FrodoKEM. The amount of resources used (except the number of BRAMs and DSPs) increases noticeably with the increase in security level. For FrodoKEM and Saber, the increase in security level does not significantly affect the resource utilization (except for the small increase in the number of BRAMs in FrodoKEM).

FrodoKEM is able to achieve the highest clock frequency, above 400 MHz, for all parameter sets. This is possible because of pipelining inside of the FrodoKEM 4MACs unit, taking advantage of internal registers of DSP units. The same optimization is not possible for Saber, because immediate feedback from the output registers is necessary for the operation performed in the next clock cycle.

In Table V, we provide an example of hardware calls for a specific algorithm, FrodoKEM-1344. main_hw represents the main functionality of the hardware accelerator. shake256 represents calls to the specific operation within this accelerator. The ratio between the transfer time and the total HW execution time depends primarily on the operation executed in hardware, and, to a lower extent, on the amount of data transferred.

VII. CONCLUSIONS

In this paper, we have demonstrated the feasibility of a new benchmarking approach, based on software/hardware codesign, with application to 3 PQC schemes representing 3 submissions qualified to Round 2 of the NIST PQC standardization process. For all analyzed schemes, and both major operations (encapsulation and decapsulation) in each, the total speed-up always exceeded a factor of 7. The highest speed-ups were accomplished for FrodoKEM, reaching 28.4 for

TABLE III
TIMING RESULTS

Algorithm	Security Category: Parameter Set	Total SW [ms]	Total SW/HW [ms]	Total Speed-up	Accel. SW [ms]	Accel. HW [ms]	Accel. Speed-up	SW part Sped up by HW [%]
Encapsulation								
FrodoKEM	1:Frodo-640	16.192	1.223	**13.2**	15.32190	0.352	43.5	94.62
FrodoKEM	3:Frodo-976	34.609	1.642	**21.1**	33.72683	0.760	44.3	97.45
FrodoKEM	5:Frodo-1344	62.076	2.186	**28.4**	61.21799	1.328	46.1	98.62
Round5	1:R5ND-1PKE_0d	0.154	0.019	**8.3**	0.15350	0.018	8.7	99.35
Round5	3:R5ND-3PKE_0d	0.245	0.024	**10.2**	0.24385	0.023	10.6	99.54
Round5	5:R5ND-5PKE_0d	0.337	0.202	**10.6**	0.33502	0.030	11.1	99.56
Saber	1:LightSaber-KEM	0.379	0.053	**7.1**	0.36904	0.043	8.5	97.46
Saber	3:Saber-KEM	0.725	0.060	**12.1**	0.71390	0.049	14.5	98.48
Saber	5:FireSaber-KEM	1.195	0.074	**16.1**	1.12595	0.050	23.4	97.98
Decapsulation								
FrodoKEM	1:Frodo-640	16.192	1.319	**12.3**	15.21528	0.342	44.4	93.97
FrodoKEM	3:Frodo-976	34.649	1.866	**18.6**	33.53212	0.750	44.7	96.78
FrodoKEM	5:Frodo-1344	62.377	3.120	**20.0**	60.57344	1.317	46.0	97.11
Round5	1:R5ND-1PKE_0d	0.193	0.024	**8.1**	0.19273	0.024	8.1	100.00
Round5	3:R5ND-3PKE_0d	0.309	0.033	**9.4**	0.30946	0.033	9.4	100.00
Round5	5:R5ND-5PKE_0d	0.416	0.042	**9.8**	0.41580	0.042	9.8	100.00
Saber	1:LightSaber-KEM	0.474	0.056	**8.4**	0.45872	0.041	11.1	93.97
Saber	3:Saber-KEM	0.867	0.065	**13.2**	0.84965	0.048	17.6	98.01
Saber	5:FireSaber-KEM	1.379	0.080	**17.3**	1.35440	0.055	24.6	98.21

TABLE IV
MAXIMUM FREQUENCY AND RESOURCE UTILIZATION

Algorithm	Security Category: Parameter Set	Clock Freq. [MHz]	LUTs	Slices	FFs	36kb BRAMs	DSPs
FrodoKEM	1:Frodo-640	402	7,213	1,186	6,647	13.5	32
FrodoKEM	3:Frodo-976	402	7,087	1,190	6,693	17	32
FrodoKEM	5:Frodo-1344	417	7,015	1,215	6,610	17.5	32
Round5	1:R5ND-1PKE_0d	260	55,442	10,627	82,341	2	0
Round5	3:R5ND-3PKE_0d	249	73,881	14,307	109,211	2	0
Round5	5:R5ND-5PKE_0d	212	91,166	18,733	151,019	2	0
Saber	1:LightSaber-KEM	322	12,343	1,989	11,288	3.5	256
Saber	3:Saber-KEM	322	12,566	1,993	11,619	3.5	256
Saber	5:FireSaber-KEM	322	12,555	2,341	11,881	3.5	256

TABLE V
HARDWARE CALLS IN DETAILS FOR FRODOKEM-1344

Function	Data In (bytes)	Data Out (bytes)	Transfer (μs)	Total HW (μs)	% in SW/HW
Encapsulation					
shake256	21,520	32	5.50	11.96	0.55
shake256	64	64	5.61	5.66	0.26
shake256	21,164	32	5.55	11.91	0.54
main_hw	32	43,008	24.94	1,298.85	59.41
Decapsulation					
shake256	64	64	5.61	5.66	0.18
shake256	21,664	32	5.50	12.01	0.39
main_hw	32	43,008	24.94	1,298.85	41.63

encapsulation and 20.0 for decapsulation. Due to very large differences among the investigated algorithms in software-only implementations, the moderate differences in total speed-ups did not affect the ranking of candidates after offloading the most time-consuming operations to hardware.

The hardware accelerator of Round5 has the highest resource utilization in terms of LUTs, Slices, and FFs, and the lowest in terms of DSP units. Saber requires by far the largest number of DSP units, 8 times more than FrodoKEM. Its number of Slices is also larger by a factor varying between 1.67 and 1.93, depending on the security category. Due to the substantial differences in functionality among Slices, DSPs, and BRAMs, the overall resource utilizations are very difficult to rank. For the specific MPSoC they are mostly insignificant, especially if the PL part of this device is dedicated to hardware accelerators. If the hardware accelerators were to be ported to ASICs, and shared with other functional units, then the differences might be significant, but this is difficult to predict based only on the obtained FPGA results.

Future work will include extending this analysis to the remaining NIST Round 2 PQC candidates, as well as the exploration of other software/hardware codesign platforms and development tools.

REFERENCES

[1] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates," in *2010 International Conference on Field Programmable Logic and Applications, FPL 2010*, Milan, Italy, Aug. 2010, pp. 400–407.

[2] K. Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, and M. U. Sharif, "Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs," Cryptology ePrint Archive 2012/368, 2012.

[3] J.-P. Kaps, K. K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, and J. Pham, "Lightweight Implementations of SHA-3 Candidates on FPGAs," in *12th International Conference on Cryptology in India, Indocrypt 2011*, ser. LNCS, vol. 7107, Chennai, India, Dec. 2011, pp. 270–289.

[4] M. Knezevic, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, Ü. Kocabas, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 5, pp. 827–840, May 2012.

[5] "CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness - web page," https://competitions.cr.yp.to/caesar.html, 2019.

[6] Cryptographic Engineering Research Group (CERG) at George Mason University, "Hardware Benchmarking of CAESAR Candidates," https://cryptography.gmu.edu/athena/index.php?id=CAESAR, 2019.

[7] "Post-Quantum Cryptography Standardization," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, 2019.

[8] T. Oder and T. Guneysu, "Implementing the NewHope-Simple Key Exchange on Low-Cost FPGAs," in *LATINCRYPT 2017*, Havana, Cuba, Sep. 2017.

[9] 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.

[10] J. Howe, T. Oder, M. Krausz, and T. Güneysu, "Standard Lattice-Based Key Encapsulation on Embedded Devices," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 3, pp. 372–393, Aug. 2018.

[11] B. Koziel, R. Azarderakhsh, M. Mozaffari Kermani, and D. Jao, "Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 86–99, Jan. 2017.

[12] A. Ferozpuri 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)*. Cancun, Mexico: IEEE, Dec. 2018, pp. 1–8.

[13] D. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas*, book version: 0.0.1 ed. Strawberry Canyon LLC, Oct. 2017.

[14] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual . Volume I: Unprivileged ISA v2.2," Tech. Rep. 20190608-Base-Ratified, Jun. 2019.

[15] ——, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, v1.12," p. 113, Jun. 2019.

[16] FrodoKEM Submission Team, "Round 2 Submissions - FrodoKEM candidate submission package," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions, Apr. 2019.

[17] Round5 Submission Team, "Round 2 Submissions - Round5 candidate submission package," Apr. 2019.

[18] Saber Submission Team, "Round 2 Submissions - Saber candidate submission package," Apr. 2019.

[19] National Institute of Standards and Technology, *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, Aug. 2015.

[20] J. Kelsey, S.-j. Chang, and R. Perlner, "NIST Special Publication 800-185: SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., Dec. 2016.

[21] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*, May 2005.

[22] S. Goldwasser and M. Bellare, *Lecture Notes on Cryptography*, Jul. 2008.

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

[24] 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.

[25] 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.

[26] F. Farahmand, V. B. Dang, D. T. Nguyen, and K. Gaj, "Evaluating the Potential for Hardware Acceleration of Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Codesign," in *10th International Conference on Post-Quantum Cryptography, PQCrypto 2019*, ser. LNCS. Chongqing, China: Springer, May 2019.

[27] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *2017 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2017*. Cancun: IEEE, Dec. 2017, pp. 1–8.