High-Level Synthesis in Implementing and Benchmarking Number Theoretic Transform in Lattice-based Post-Quantum Cryptography using Software/Hardware Codesign

Duc Tri Nguyen, Viet B. Dang, and Kris Gaj

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

Abstract. Compared to traditional hardware development methodologies, High-Level Synthesis (HLS) offers a faster time-to-market and lower design cost at the expense of implementation efficiency. Although Software/Hardware Codesign has been used in many areas, its usability for benchmarking of candidates in cryptographic competitions has been largely unexplored. This paper provides a comparison of the HLS- and RTL-based design methodologies when applied to the hardware design of the Number Theoretic Transform (NTT) – a core arithmetic function of lattice-based Post-Quantum Cryptography (PQC). As a next step, we apply Software/Hardware Codesign approach to the implementation of three PQC schemes based on NTT. Then, we integrate our HLS implementation into the Xilinx SDSoC environment. We demonstrate that an overhead of SDSoC compared to traditional Bare Metal approach is acceptable. This paper also shows that an HLS implementation obtained by modeling a block diagram is typically much better than an implementation obtained by using design space exploration. We conclude that the HLS/SDSoC and RTL/Bare Metal approaches generate comparable results.

1 Introduction

A threat of quantum computers triggered an effort aimed at designing a new class of cryptographic algorithms, collectively referred to as Post-Quantum Cryptography (PQC) [1]. These algorithms have two common features: a) there are no known attacks capable of breaking these cryptosystems, even assuming the availability of full-scale quantum computers, b) all PQC algorithms can be implemented using traditional computing platforms, based on standard semiconductor technology, such as microprocessors and FPGAs. In the standardization process currently run by the National Institute of Standards and Technology (NIST), 26 candidates remain in Round 2 and need to be evaluated from the point of view of their hardware efficiency [1]. A large number of candidates and high complexity of the majority of them make hardware benchmarking extremely challenging. In order to mitigate these difficulties, a new approach based on a)

software/hardware codesign, and b) the development of hardware accelerators using High-Level Synthesis (HLS) has been proposed [2].

In the traditional RTL approach, a path from developing HDL code to running it on a target device is quite long, since the developer has to create an interface between CPU and FPGA. On the other hand, in the Xilinx SDSoC framework, most of these tasks are performed automatically by the tools.

In the remaining 12 Round 2 lattice-based PQC candidates, 5 use Number Theoretic Transform (NTT) for polynomial multiplication. After software profiling, we decided to implement the NTT hardware accelerators for NewHope and Kyber. Since Kyber has been substantially modified between Rounds 1 and 2, we have decided to compare the implementation efficiencies of these two variants, further denoted as Kyber R1 and Kyber R2.

This paper demonstrates: a) Advantages of the HLS approach based on block diagrams vs. the HLS approach based on space exploration. b) Overhead of the SDSoC/HLS methodology over the Bare Metal/RTL approach.

2 Background

2.1 Number Theoretic Transform

Let n be a power of two, and q be a prime modulus. We define a ring $R_q[x] = Z_q[x]/(x^n + 1)$ as the ring of polynomials of degree $n - 1$ with coefficients in Z_q (a field of integers in the range [0, $q - 1$] with addition and multiplication modulo q). Multiplications in $R_q[x]$ can be performed efficiently in software and hardware using the NTT, which has the complexity of $O(n \cdot log(n))$.

If $\psi^2 = \omega \bmod q$ exists, then it is recommended that the input polynomials should be multiplied by ψ^i before Forward NTT instead of supplementing them with n most significant terms equal to zero. As a result, the output of Inverse NTT must be multiplied by ψ^{-i}.

By using NTT, a multiplication in R_q can be computed as follows:

$$C = \mathbf{NTT}^{-1}(\overline{C}) = \mathbf{NTT}^{-1}(\overline{A} * \overline{B}) = \mathbf{NTT}^{-1}(\mathbf{NTT}(A) * \mathbf{NTT}(B))$$

where $\psi^2 = \omega$, $A = (a_0, \psi a_1, \psi^2 a_2, \ldots, \psi^{n-1} a_{n-1})$, $B = (b_0, \psi b_1, \psi^2 b_2, \ldots, \psi^{n-1} b_{n-1})$, $C = (c_0, \psi c_1, \psi^2 c_2, \ldots, \psi^{n-1} c_{n-1})$, and a, b, c are polynomials in $R_q[x]$, with $q = 1 \bmod 2n$ [3].

The pseudo-code of the iterative version of Forward NTT is shown in Algorithm 1. The Inverse NTT is similar to Forward NTT but instead of multiplying with ω^i, we multiply with ω^{-i}.

In this paper, we divide the polynomial multiplication into two modes:
a) **NTT**: Forward (*NTT*) and Inverse NTT (*INTT*)
b) **MUL**: Multiplication of the respective coefficients by ψ^i (*PSIS_MUL*), multiplication of the respective coefficients by ψ^{-i} (*IPSIS_MUL*), and coefficient-wise multiplication of two polynomials (*COEF_MUL*).

Modular multiplication can be performed using two primary approaches: Montgomery multiplication (REDC) and the method introduced by Longa et al. in [4] (KRED).

Algorithm 1 Iterative NTT

Require: $F(x) \in R_q[x]$; $ROM[i] = \omega^i$, $\omega^n = 1 \bmod q$
Ensure: $\overline{F}(x) = NTT(F)$
1: $F \leftarrow BitReverse(F)$
2: **for** $s = 0$ to $log_2(n) - 1$ by 1 **do**
3: $m \leftarrow 2 \ll s$
4: $\omega_m \leftarrow n/m$
5: $i \leftarrow 0$
6: **for** $j = 0$ to $m/2$ by 1 **do**
7: **for** $k = 0$ to n by m **do**
8: $u \leftarrow F[k + j]$
9: $t \leftarrow F[k + j + m/2] * ROM[i]$
10: $F[k + j] \leftarrow u + t$
11: $F[k + j + m/2] \leftarrow u - t$
12: $i \leftarrow i + \omega_m$

3 Previous Work

The theory of NTT is summarized in [3]. Previous hardware implementations of NTT were reported in [5,6]. The first hardware implementations of NTT targeted the PQC scheme called Ring-LWE [7]. The most recent efforts aimed specifically at the efficient implementations of the Round 1 PQC candidate NewHope, qualified to the second round of the NIST PQC standardization process [8, 9].

Efficient implementations of block ciphers, hash functions, and authenticated ciphers using HLS were reported in [10,11]. In the majority of cases, these implementations closely matched the performance of RTL implementations in terms of throughput and throughput-to-area ratio. The first attempts at the use of HLS for benchmarking of PQC candidates were reported in [2].

4 Block Diagram versus Space Exploration

There are two major approaches to implementing hardware accelerators in HLS:
a) SE/HLS: Identify optimal HLS-ready code using design space exploration based on HLS directives. The final hardware architecture is unknown until the best result is achieved.
b) BD/HLS: Develop block diagram corresponding to the presumed optimal hardware architecture. Write HLS code following this block diagram.

Both SE/HLS and BD/HLS approaches inherit the advantages of HLS: quicker verification and quicker development than in traditional RTL. As shown in Fig. 1, in the SE/HLS approach, a small portion of the total development time is spent on writing HLS-ready code and verifying its functionality. The rest of the time is devoted to design space exploration using *pragma* directives. There are over 20 *pragma* directives in the current version of Vivado HLS; their different combinations lead to different architectures. The impact of a particular *pragma* directive is heavily dependent on the code structure and the algorithm. Some directives

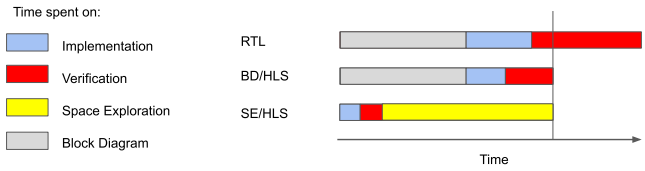

Fig. 1. BD/HLS versus SE/HLS development timeline

Table 1. Results for BD/HLS vs. results for SE/HLS for 1024-point NTT

Work	BRAM 18K	DSP	FF	LUT	Cycles	Cycles Reduc.
[12]	11.5	10	16,402	21,167	7,597	1.59
[12]	21.5	19	30,498	38,984	5,291	1.10
This work	**10**	**4**	**1,342**	**1,110**	**4,776**	**1.0**

may have no impact at all, others may dramatically change the speed vs. cost trade-off. Exploring all possible combinations is often unrealistic. Additionally, in many cases, code refactoring may give better results than an optimal choice and placement of directives. As a result, the HLS design by space exploration may lead to the choice of sub-optimal hardware architecture.

In BD/HLS approach, the large portion of the total development time is spent on developing a block diagram and implementing it in HLS-ready C. The rest of the time is spent on verification. Since the exact hardware architecture is known beforehand, space exploration is not required. With the HLS-ready code based on a block diagram created manually by an experienced designer, the BD/HLS approach can significantly outperform the SE/HLS methodology.

In Table 1, the comparison of the NTT implementations according to two approaches, BD/HLS and SE/HLS, is summarized. The BD/HLS approach uses **2x, 5x, 22x, 35x,** and **1.1x** less BRAMs, DSPs, LUTs, FFs and Clock Cycles, respectively. In [12], the authors experiment with multiple combinations of directives, applied to multiple loops. However, the final design outcome is still not as good as in our BD/HLS design.

5 Hardware Design

5.1 NTT Top Level Design

A top-level block diagram of a hardware accelerator for NewHope and Kyber is shown in in Fig. 2. There are 3 main components: *NTT*, *MUL* and *Reorder*. For NewHope and Kyber R1, the *NTT* unit is responsible for the **NTT** and **MUL** modes of operation, described in Section 2.1. In Kyber R2, the *NTT* unit is only responsible for the **MUL** mode. As a result, a dedicated MUL unit must be added. The role of the Reorder unit is explained at the end of the next section.

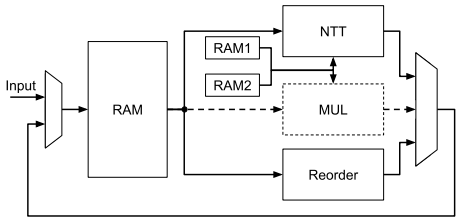

Fig. 2. Top-level block diagram of a hardware accelerator for NewHope and Kyber

Table 2. Selected NTT-based Round 2 PQC candidates investigated in this study. N and q are major parameters of NTT. k and m are used in the Longa-Naehrig modular reduction, and $qinv$ in the Montgomery Reduction.

Candidate	**Cat (#NTT)**	**n**	**q**	2^m	**k**	k^2	**qinv**
NewHope	1,5 (1)	512/1024	12,289	2^{12}	3	2^3+1	$2^{13}+2^{12}-1$
Kyber R1	1,3,5 (2,3,4)	256	7,681	2^9	15	2^8-2^5+1	$2^{13}-2^9-1$
Kyber R2	1,3,5 (2,3,4)	256	3,329	2^8	13	$2^7+2^5+2^3+1$	2^9+2^8+1

5.2 Number Theoretic Transform

A block diagram of our NTT implementation, shown in Fig. 3, is based on the design from [6]. One of the improvements is support for both odd and even numbers of NTT layers. When $log_2(n)$ is odd, the signal X is asserted during the *last* iteration to let coefficients A', B', C', D' pass directly to the SIPO unit instead of going through the 2nd NTT layer. On the other hand, when $log_2(n)$ is even, the multiplexers with the select signal X can be eliminated.

Our NTT hardware architecture has a 2x2 butterfly structure, which can process two layers of NTT with two butterfly units per layer. Four coefficients are loaded in each clock cycle and placed into registers A, B, C, D. If KRED is selected as a modular reduction method, the square boxes m_1 and m_2 are KRED and KRED2x, respectively. If REDC Montgomery reduction is chosen, m_1 can be removed and m_2 represent REDC.

When $S = 0$, the circuit operates in the **MUL** mode, used to perform operations *PSIS_MUL*, *COEF_MUL*, and *IPSIS_MUL*. The coefficients in the lines B and D are multiplied by coefficients from RAM1, which are $(\psi^{4i+1}, \psi^{4i+3})$ or (r_{4i+1}, r_{4i+3}) or $(\psi^{-(4i+1)}, \psi^{-(4i+3)})$, depending on the performed operation. The obtained results are reduced by the function m_2 and stored in B_{save} and D_{save}, which go to SIPO_B and SIPO_D later on. After that, coefficients from lines A and C are switched to lines B and D, allowing them to be multiplied with coefficients from RAM2, reduced by m_2, and directed to SIPO_A and SIPO_C, respectively. When the following outputs of SIPOs: $A1st, B2nd, C3rd$ and $D4th$ become available, they are concatenated and written back to RAM at the index where $A1st$ was loaded from.

When $S = 1$, the circuit operates in the **NTT** mode use to perform operations *INTT* and *NTT*. Four coefficients go through the 2x2 butterfly structure, and

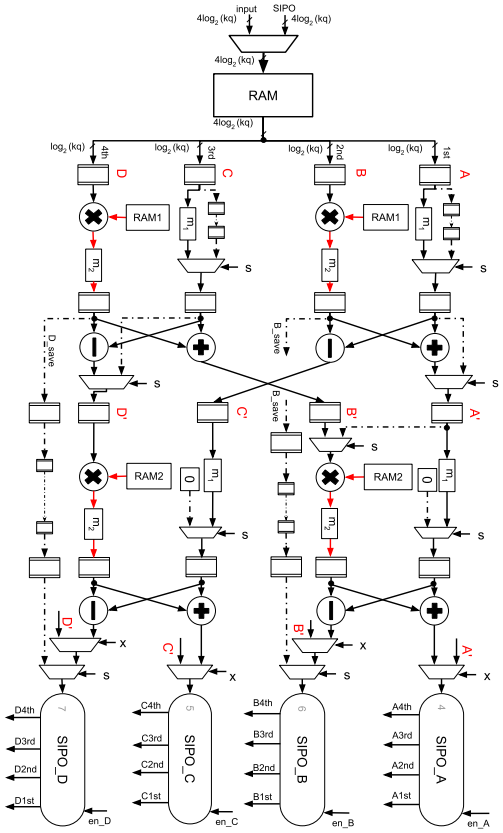

Fig. 3. Block diagram of the proposed hardware architecture for fast polynomial multiplication using NTT. The red lines represent four likely critical paths.

results are written to SIPOs. Coefficients in lines B and D are multiplied with ω_n^i or ω_n^{-i}, depending on whether the circuit computes NTT or $INTT$. When $SIPO_A$ is full, four coefficients available at the outputs $A1st$, $A2nd$, $A3rd$, $A4th$ are concatenated, and stored back to the RAM at the position where $A1st$ was loaded from. After one clock cycle, the same happens with results accumulated in $SIPO_C$, and then $SIPO_B$ and $SIPO_D$.

Shuffle and Reordering The order of coefficients is changed in the **NTT** mode. Thus, after each NTT operation, one must shuffle and reorder the obtained coefficients. We apply the in-place matrix transposition proposed in [13]. The number of clock cycles for 128-, 256-, and 1024-point NTT is 64, 80, 318 clock cycles, respectively. In particular, for the 1024-point NTT, we use 318 clock cycles vs. 1024 clock cycles in [9].

For NewHope and Kyber R1, all five operations from Section 2.1 are supported by the circuit from Fig. 3. In the case of Kyber R2, *PSIS_MUL* and *IPSIS_MUL* do not apply. Only *NTT* and *INTT* are performed by the NTT

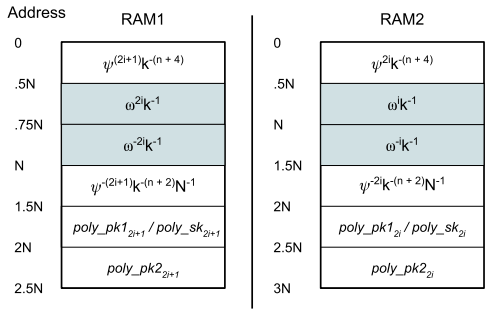

Fig. 4. The memory maps of RAM1 and RAM2, including formulas for values of constants stored in specific memory ranges. $n{=}log_2 n$, $\omega = \omega_n$, $i \in [0, 1, \ldots, n/2)$ for RAM1 and RAM2, except the gray area of RAM 1, where $i \in [0, 1, \ldots, n/4)$. For the KRED, the value of k is given in Table 2. If the REDC is used, k is assumed to be 1. *poly_pk* and *poly_sk* are NTT domain preloaded public and secret polynomials.

Table 3. Results of the HLS implementations of KRED and REDC

Candidate	Modular Reduction	DSP	LUT	FF	Slice	Max. Freq
NewHope	KRED	1	118	100	28	530
KyberR1		1	125	93	32	507
KyberR2		1	150	112	35	502
NewHope	REDC	1	370	357	85	515
KyberR1		1	387	333	69	512
KyberR2		1	391	382	91	476

unit. The *COEF_MUL* is performed by a separate unit. Therefore, the NTT mode of Kyber R2 can be simplified by stripping the dot line and removing multiplexers to save resources and improve maximum clock frequency.

The precomputed values of all constants are stored in the dual-port memories RAM1 and RAM2, of the size $2.5n$ and $3n$ memory locations, respectively. The number of bits stored at each memory location is equal to $log_2 q$. The memory map and formulas for the values of constants stored within each specific address range are shown in Fig. 4.

6 Results

The target device is Zynq UltraScale+ MPSoC ZCU104, with CPU Cortex-A53 running at 1.2 GHz. All results presented in this section are obtained after placing and routing.

Results of the HLS implementation of two alternative reduction methods, KRED and REDC, for the value of q corresponding to investigated candidates, are shown in Table 3. These results demonstrate that compared to REDC, the implementation of KRED uses less resources and is comparable in term of performance. Therefore, KRED is selected as a modular reduction method.

Table 4. Resources Utilization for HLS and RTL

Algorithm	#NTT	DSP	BRAM 36K	LUT	FF	Slice	Freq. (Mhz)
RTL							
NewHope 1	1	4	3	1,040	940	190	476
NewHope 5	1	4	5	842	803	170	476
Kyber R1-1	2	8	2	2,185	2,625	411	500
Kyber R1-3	3	12	3	3,318	3,937	605	500
Kyber R1-5	4	16	4	4,363	5,237	795	500
Kyber R2-1	2	24	5	2,040	3,223	433	500
Kyber R2-3	3	36	8	3,054	5,098	637	500
Kyber R2-5	4	48	10	4,055	6,803	960	500
HLS/RTL							
NewHope 1	1.00	1.00	1.00	1.14	1.49	1.26	0.95
NewHope 5	1.00	1.00	1.00	1.32	1.67	1.29	0.96
Kyber R1-1	1.00	1.00	1.00	1.28	1.03	1.45	0.91
Kyber R1-3	1.00	1.00	1.00	1.27	1.03	1.45	0.91
Kyber R1-5	1.00	1.00	1.00	1.27	1.06	1.54	0.91
Kyber R2-1	1.00	1.00	1.40	1.35	1.43	1.57	0.91
Kyber R2-3	1.00	1.00	1.40	1.40	1.51	1.65	0.89
Kyber R2-5	1.00	1.00	1.40	1.47	1.53	1.67	0.89

Table 5. Comparison of the transfer time & overhead between SDSoC and Bare Metal

Algorithm	Total Transfer Size (bytes)		Times	Total Transfer Time (μs)		Transfer Ratio	Transfer Overhead	
	In	Out		BM	SDSoC	SDSoC/ BM	BM	SDSoC
ENCAPSULATION								
NewHope 1	2,048	2,048	1	7.91	12.64	1.60	4.51%	7.01%
NewHope 5	4,096	4,096		11.90	19.50	1.64	3.67%	5.87%
Kyber R1-1	1,024	1,536		7.85	9.86	1.26	4.94%	6.12%
Kyber R1-3	1,536	2,048		8.05	11.71	1.46	3.58%	5.12%
Kyber R1-5	2,048	2,560		9.42	13.49	1.43	2.86%	4.04%
Kyber R2-1	1,024	1,536		7.85	9.86	1.26	7.77%	9.54%
Kyber R2-3	1,536	2,048		8.05	11.71	1.46	3.99%	5.69%
Kyber R2-5	2,048	2,560		9.42	13.49	1.43	3.12%	4.40%
DECAPSULATION								
NewHope 1	3,072	3,072	2	15.22	21.57	1.42	8.56%	11.69%
NewHope 5	6,144	6,144		19.81	32.13	1.62	5.93%	9.26%
Kyber R1-1	2,048	2,048		15.15	17.99	1.19	9.35%	10.89%
Kyber R1-3	3,072	2,560		15.90	20.76	1.31	7.02%	8.96%
Kyber R1-5	4,096	3,072		17.91	23.47	1.31	5.39%	6.94%
Kyber R2-1	2,048	2,048		15.15	17.99	1.19	11.36%	13.17%
Kyber R2-3	3,072	2,560		15.90	20.76	1.31	7.53%	9.58%
Kyber R2-5	4,096	3,072		17.91	23.47	1.31	5.78%	7.42%

Table 6. Speed up of the Software/Hardware Codesign vs. Pure Software

Algorithm	Total SW (μs)	Total SW NTT (μs)	%SW NTT	Total SW/HW (μs)		Total Speed-up @Max Freq	
				BM	SDSoC	BM	SDSoC
ENCAPSULATION							
NewHope 1	360.3	199.8	55%	175.2	180.3	2.06	2.00
NewHope 5	737.0	438.1	59%	324.0	332.2	2.27	2.22
Kyber R1-1	389.2	240.9	62%	158.9	161.1	2.45	2.42
Kyber R1-3	582.3	368.3	63%	224.8	228.7	2.59	2.55
Kyber R1-5	826.9	509.4	62%	329.6	334.0	2.51	2.48
Kyber R2-1	328.5	237.8	72%	101.1	103.4	3.25	3.18
Kyber R2-3	533.9	343.0	64%	201.5	205.7	2.65	2.60
Kyber R2-5	785.2	495.4	63%	301.8	306.4	2.60	2.56
DECAPSULATION							
NewHope 1	427.5	273.5	64%	177.8	184.6	2.40	2.32
NewHope 5	895.7	598.0	67%	334.0	347.1	2.68	2.58
Kyber R1-1	483.2	340.8	71%	161.9	165.2	2.98	2.92
Kyber R1-3	710.4	504.2	71%	226.5	231.8	3.14	3.06
Kyber R1-5	992.1	682.4	69%	332.0	338.0	2.99	2.94
Kyber R2-1	429.5	315.5	73%	133.3	136.6	3.22	3.14
Kyber R2-3	667.8	476.8	71%	211.1	216.8	3.16	3.08
Kyber R2-5	950.8	662.9	70%	310.0	316.4	3.07	3.00

The comparison between HLS and RTL is shown in Table 4. The PQC candidates are compared at the multiple security levels: 1, 3, and 5. The number of BRAMs in HLS is higher than in RTL due to a higher abstraction level description of HLS. In particular, the tool duplicates RAM1 and RAM2 for each MUL component. Thus, the number of BRAMs for Kyber R2 is higher than in RTL. There are two pairs of RAM1 and RAM2 in a single HLS NTT module, instead of just one. The number of LUTs, FFs, and Slices is consistently greater in HLS.

Traditional RTL SW/HW Codesign often uses Bare Metal (BM) to handle transfer between CPU and FPGA. The DMA in BM is often implemented manually. Contrary to that, SDSoC creates an abstraction layer of the interface handler. As a result, switching from software to hardware is very easy. To demonstrate the overhead of abstraction in using SDSoC, the best selected transfer interface in SDSoC is compared with Bare Metal in Table 5. Additionally, the **Transfer Overhead** column is the percentage of **Total Transfer Time** over the **Total SW/HW** in Table 6.

In Table 6, timing results are summarized. The HLS/SDSoC approach generates comparable accelerator speed up for all investigated algorithms. The number of clock cycles of NTT HW acclerator for polynomial multiplication (excluding the transfer time) in Encapsulation and Decapsulation phase for NewHope 1, NewHope 5, Kyber R1, and Kyber R2 are 3300, 6300, 1400, 1300 and 4100, 7900, 2200, 2100, respectively.

The **Total SW** is the software only execution time, the **Total SW NTT** column is the time spent on NTT operations in SW, **%SW NTT** is the percentage of the total execution time in software devoted to NTT, the **Total SW/HW** is the total time after offloading the critical function (NTT) to hardware. The **Total Speed-up @Max Freq** is the ratio between **Total SW** and **Total SW/HW**. This speed-up is roughly equal between the SDSoC and Bare Metal approaches.

7 Conclusions

Using HLS and SDSoC are two promising approaches to benchmarking SW/HW implementations of PQC. With the help of these approaches, the development time is substantially reduced, with the relatively small penalty in terms of the total execution time, HW-SW transfer time, and the total speed-up vs. purely SW implementation. Overhead in terms of resource utilization is more substantial, especially in terms of the number of LUTs, FFs, and Slices. The BD/HLS approach, based on the use of block diagrams, was shown to be substantially more efficient than the approach, SE/HLS, based on applying various pragmas to existing code and letting the tool to infer the best possible architecture.

References

1. "NIST Post-Quantum Cryptography Standardization."
2. 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 *PQCrypto*, 2019, pp. 23–43.
3. E. C.-h. Chu and A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, ser. Computational Mathematics Series.
4. P. Longa, M. Naehrig, P. Longa, and M. Naehrig, "Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography," in *Cryptology and Network Security - CANS 2016*, vol. 10052.
5. T. Pöppelmann and T. Güneysu, "Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware," in *LATINCRYPT 2012*.
6. C. Du, G. Bai, and X. Wu, "High-Speed Polynomial Multiplier Architecture for Ring-LWE Based Public Key Cryptosystems," in *GLSVLSI'16*.
7. C. P. Renteria-Mejia and J. Velasco-Medina, "High-Throughput Ring-LWE Cryptoprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.
8. T. Oder and T. Guneysu, "Implementing the NewHope-Simple Key Exchange on Low-Cost FPGAs," in *LATINCRYPT 2017*, Havana, Cuba, Sep. 2017.
9. P.-C. Kuo *et al.*, "High Performance Post-Quantum Key Exchange on FPGAs," Cryptology ePrint Archive 2017/690, Feb. 2018.
10. E. Homsirikamol and K. Gaj, "Hardware Benchmarking of Cryptographic Algorithms Using HLS Tools: The SHA-3 Contest Case Study," in *ARC 2015*.
11. ——, "A new HLS-based methodology for FPGA benchmarking of candidates in cryptographic competitions: The CAESAR contest case study," in *FPT 2017*.
12. K. Kawamura, M. Yanagisawa, and N. Togawa, "A loop structure optimization targeting high-level synthesis of fast number theoretic transform," in *ISQED 2018*.
13. D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, 1997.