A High-Level Synthesis Approach to the Software/Hardware Codesign of NTT-based Post-Quantum Cryptography Algorithms

Duc Tri Nguyen
ECE Department
George Mason University
Fairfax, VA, USA
dnguye69@gmu.edu

Viet B. Dang
ECE Department
George Mason University
Fairfax, VA, USA
vdang6@gmu.edu

Kris Gaj
ECE Department
George Mason University
Fairfax, VA, USA
kgaj@gmu.edu

***Abstract*—In this paper, we present an improved hardware architecture for NTT, with the hardware-friendly modular reduction, and demonstrate that this architecture can be efficiently implemented in hardware using High-Level Synthesis (HLS). The novel feature of the proposed architecture is an original memory write-back scheme, which assists in preparing coefficients for performing later NTT stages, saving memory storage used for precomputed constants. Our design is the most efficient for the case when $log_2(N)$ is even. The latency of our proposed architecture is approximately equal to $(Nlog_2(N)+3N)/4$ clock cycles. As a proof of concept, we implemented the NTT operation for several parameter sets used in the Round 2 NIST PQC candidates NewHope, FALCON, qTESLA, and CRYSTALS-DILITHIUM.**

***Index Terms*—Post-Quantum cryptography, Number Theoretic Transform, Lattice-based, High-Level Synthesis, programmable logic**

I. Introduction

Several lattice-based candidates in Round 2 of the NIST PQC standardization process use operations in the polynomial ring $R_q = Z_q[x]/(x^N+1)$, with N being a power of 2. Examples include the encryption/KEM scheme NewHope, and the digital signature schemes FALCON, qTESLA, and CRYSTALS-DILITHIUM [1]. The most time-consuming operation of these cryptosystems, a multiplication in the ring R_q, can be sped up by using the Number Theoretic Transform (NTT). The input and output of a transform is a polynomial with N coefficients, which are integers in the range of 0 to q-1. All internal operations of NTT are performed modulo a prime q.

Using High-Level Synthesis can potentially significantly decrease the development time, mostly due to simpler verification and debugging. An additional advantage is the ability to focus during the design process on the main functionality of the algorithm, rather than details of the hardware controller, typically time-consuming to describe and debug using manually written HDL code.

II. Background

A. Number Theoretic Transform

Let N be a power of two, and q be a prime modulus. We define a ring $R_q = Z_q[x]/(x^N + 1)$ 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).

Polynomial multiplication implemented using NTT has the complexity of $O(NlogN)$.

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

$$c = \mathbf{INTT}^{-1}(\mathbf{NTT}(a) * \mathbf{NTT}(b))$$

for $a, b, c \in R_q$, with $q \equiv 1 \bmod 2N$.

In this paper, we divide the polynomial multiplication into 5 steps:

1) **PSIS_MUL**: Coefficient-wise multiplication by ψ^i
2) **NTT**: Forward NTT transform
3) **COEF_MUL**: Coeffcient-wise multiplication of two polynomials
4) **INTT**: Inverse NTT transform
5) **IPSIS_MUL**: Coefficient-wise multiplication by ψ^{-i}.

Furthermore, we treat **PSIS_MUL, COEF_MUL, IPSIS_MUL** as operations performed in the **MUL** mode, and **NTT, INTT** as operations performed in the **NTT** mode.

B. Modular Reduction

Introduced by [2], the method takes advantage of the special form of q: $q = k \cdot 2^m + 1$, where k is odd and $k < 2^m$. Although this function alone does not completely reduce the value of C, it is still referred to as a reduction because it brings D to the close vicinity of the desired range.

For the majority of candidates, KRED and KRED2x can be rewritten, using the shift and add operations, by taking advantage of the special form of k, shown in Table I.

TABLE I
SELECTED NTT-BASED ROUND 2 PQC CANDIDATES INVESTIGATED IN THIS STUDY. PARAMETERS k AND m OF THE LONGA-NAEHRIG MODULAR REDUCTION, AND THE NUMBER OF ADDERS REQUIRED TO PERFORM FUNCTIONS K-RED AND K-RED-2X.

Candidate(s)	Sec. Cat	k	2^m	KRED adder	KRED2x adder
NewHope, Falcon	5	3	2^{12}	2	5
qTESTLA	3	2^9+1	2^{14}	2	5
CRYSTALS-Dilithium	1, 2, 3	$2^{10}-1$	2^{13}	2	5

C. High-Level Synthesis

In this paper, we use the C/C++ code as an input to HLS. However, this code is written taken into account the desired hardware architecture to be inferred from this code by Vivado HLS, and the block diagram of the datapath is used to guide the C/C++ coding process. The HLS-based development effort was still several times shorter than the RTL design, mostly due to the more efficient verification phase and the higher-level of abstraction.

III. PREVIOUS WORK

The hardware implementations of NTT targeted the Encryption/KEM PQC schemes such as Ring-LWE [3], [4], as well as the corresponding lattice-based signatures schemes [5]. 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 [6], [7].

IV. METHODOLOGY

A. Hardware Architecture

We have simplified [8] architecture for the case of even values of $log_2(N)$, used by the selected variants (security categories) of the PQC Round 2 candidates summarized in Table I. The simplified block diagram is shown in Fig. 1, and the corresponding algorithms is shown as Algorithm 2. Similarly to the hardware design from [8], the architecture has the 2x2 butterfly structure, four coefficients are loaded in each clock cycle without stalling. The square box m_1 represents the reduction function KRED, and the square box m_2 the reduction function KRED2x.

When **s** is set to 0, the circuit operates in the **MUL** mode defined in Section II-A. First, four coefficients are loaded from the RAM at the top of the block diagram and placed in the registers located in the diagram immediately below the RAM. We have named four computational lines shown in the block diagram A, B, C, and D, respectively. These lines correspond to four sets of indices: $4i$, $4i+1$, $4i+2$, and $4i+3$ with $i \in [0, 1, 2, \ldots N/4)$.

Coefficients in the lines B and D are multiplied by two 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 using the function KRED2x, and stored in registers B_{save} and D_{save}, respectively. These values are then transferred using a cascade of registers to the Serial-In Parallel-Out (SIPO) units, $SIPO_B$ and $SIPO_D$. Four consecutive $log_2(kq)$ results are written to a single $4log_2(kq)$ appropriate positions in the RAM. The number of cascade registers placed in front of each SIPO differs by one, in order to prevent the case in which two SIPOs are full and attempt to write results back to the RAM in the same clock cycle. Two coefficients from lines A and C are moved to lines B and D, in order to allow their multiplication by coefficients stored in RAM2. After the following reduction function KRED2x, the results are moved back to lines B and D, and from here are transferred to the corresponding SIPOs: $SIPO_A$ and $SIPO_C$. When the following outputs of SIPOs: A_{1st}, C_{2nd},

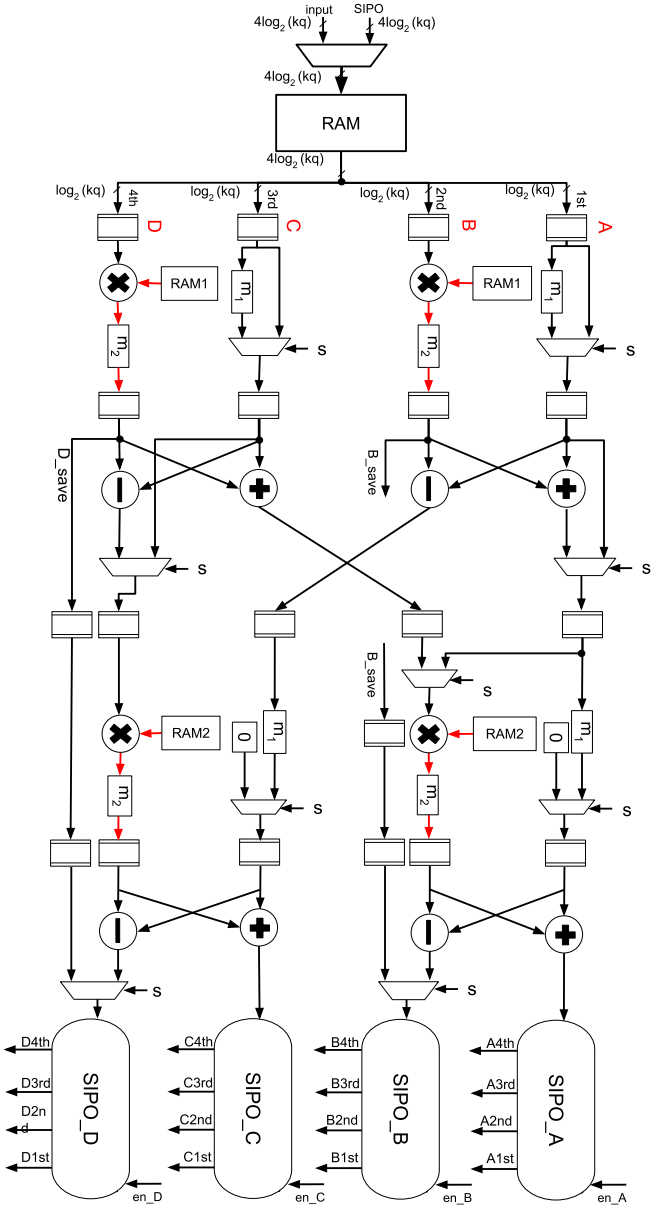


Fig. 1. Block diagram of the proposed hardware architecture to perform fast polynomial multiplication using NTT.

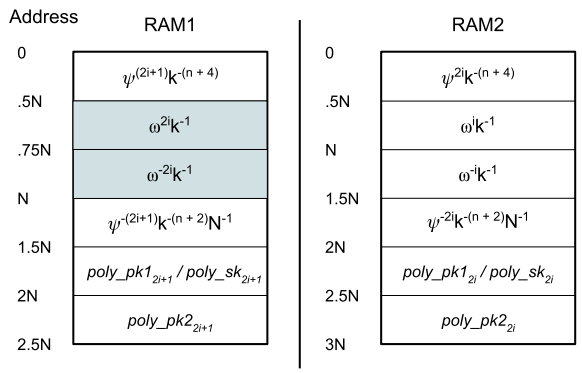


Fig. 2. The memory maps of RAM1 and RAM2, including formulas for values of constants stored in specific memory ranges. $n{=}log_2 N$, $\omega = \omega_N$, $i \in [0, 1, \ldots, N/2)$ for RAM1 and RAM2, except the gray area $i \in [0, 1, \ldots, N/4)$, the meaning of k explained in Section II-B, poly_pk and poly_sk are preloaded public and secret polynomials.

B_{3rd}, and D_{4th} are available, they are concatenated and written back to the RAM. The same happens every clock cycle afterwards until the computations are completed.

When **s** is set to 1, the circuit operates in the **NTT** mode. Four coefficient go through the 2x2 butterfly structure, and the results are written to the respective SIPOs, and then to the RAM, in the way similar as for the case of s=0. The reduction function KRED is applied after each addition and subtraction performed in layer 1, not followed by a multiplication. Coefficients in lines B and D are multiplied by ω_N^i or ω_N^{-i}, depending on whether the circuit computes NTT or INTT.

Algorithm 1 SIPO writeback scheme

if $s = 0$ **then**
  $RAM \leftarrow A_{1st}||C_{2nd}||B_{3rd}||D_{4th}$
else
  if $SIPO_A$ is full **then**
    $RAM \leftarrow A_{1st}||A_{2nd}||A_{3rd}||A_{4th}$;
  else if $SIPO_B$ is full **then**
    $RAM \leftarrow B_{1st}||B_{2nd}||B_{3rd}||B_{4th}$;
  else if $SIPO_C$ is full **then**
    $RAM \leftarrow C_{1st}||C_{2nd}||C_{3rd}||C_{4th}$;
  else if $SIPO_D$ is full **then**
    $RAM \leftarrow D_{1st}||D_{2nd}||D_{3rd}||D_{4th}$;
  end if
end if

Afterward, the coefficients in lines B and C are swapped, in order to perform computations of the next NTT layer. When $SIPO_A$ is full, four coefficients available at the outputs A_{1st}, C_{2nd}, B_{3rd}, D_{4th} are concatenated, and stored back to the RAM. After one clock cycle, the same happens with results accumulated in $SIPO_C$, and then $SIPO_B$ and $SIPO_D$. The precomputed values of all constants are stored in the dual-port memories RAM1 and RAM2. The total amount of memory required is $2.5N$ in RAM1 and $3N$ in RAM2, the bitwidth is equal to $log_2(q)$.

Algorithm 2 When s = 1 NTT operation, else coefficient-wise multiplication

for $(l = 1; l < log_2(N); l = l + 2)$ **do**
  $m = 1 \ll (l - 1)$;
  $\omega_{idx1} = N \gg l; \omega_{idx2} = N \gg (l + 1)$;
  $NTT_{idx} = NTT_{idx3} = 0; NTT_{idx4} = N/4$;
  for $(j = 0; j < m; j = j + 1)$ **do**
    for $(k = 0; k < N/4; k = k + m)$ **do**
      $b = k + j$;
      $A = RAM[b]_{63\ldots48}; B = RAM[b]_{47\ldots32}$;
      $C = RAM[b]_{31\ldots16}; D = RAM[b]_{15\ldots0}$;
      if $s = 0$ **then**
        $idx1 = (b \ll 2) + 1; idx2 = (b \ll 2) + 3$;
      else
        $idx1 = idx2 = NTT_{idx}$;
      end if
      if $s \neq 0$ **then**
        $A =$ **K-RED**$(A); C =$ **K-RED**(C);
      end if
      $B =$ **K-RED-2x**$(B * RAM1[idx1])$;
      $D =$ **K-RED-2x**$(D * RAM1[idx2])$;
      $B_{save} = B; D_{save} = D$;
      $A' = A + B; B' = C + D$;
      $C' = A - B; D' = C - D$;
      $A = A'; B = B'; C = C'; D = D'$;
      if $s = 0$ **then**
        $B = A; D = C$;
      end if
      if $s = 0$ **then**
        $idx3 = (b \ll 2); idx4 = (b \ll 2) + 2$;
      else
        $idx3 = NTT_{idx3}; idx4 = NTT_{idx4}$;
      end if
      $B =$ **K-RED-2x**$(B * RAM2[idx3])$;
      $D =$ **K-RED-2x**$(D * RAM2[idx4])$;
      if $s = 0$ **then**
        $A = 0; C = 0$;
      else
        $A =$ **K-RED**$(A); C =$ **K-RED**(C);
      end if
      $A = A + B; B = A - B$;
      $C = C + D; D = C - D$;
      if $s = 0$ **then**
        $B = B_{save}; D = D_{save}$;
      end if
      $SIPO_A \leftarrow A; SIPO_B \leftarrow B$;
      $SIPO_C \leftarrow C; SIPO_D \leftarrow D$;
    end for
    $NTT_{idx} = NTT_{idx} + \omega_{idx1}$;
    $NTT_{idx3} = NTT_{idx3} + \omega_{idx2}$;
    $NTT_{idx4} = NTT_{idx4} + \omega_{idx2}$;
  end for
end for

The memory map and formulas for the values of constants stored within each specific address range are shown

TABLE II
RESULTS OF THE IMPLEMENTATIONS OF THE NTT UNIT FOR SELECTED ROUND 2 PQC CANDIDATES, USING ZYNQ ULTRASCALE+.

Algorithms	N	q		DSPs	BRAM 36K	LUT	FF	Slices	Max Freq (MHz)	Clock Cycles	Latency (μs)
NewHope & FALCON	1024	12,289	RTL	4	5	849	802	163	476	1,324	2.78
			HLS	4	5	865	822	175	455	1,324	2.91
			HLS/RTL	1.0	1.0	1.02	1.02	1.07	0.96	1.0	1.05
qTESLA	1024	8,404,993	RTL	8	8	1,286	2,160	283	467	1,363	2.92
			HLS	8	8	1,939	3,423	453	455	1,363	2.99
			HLS/RTL	1.0	1.0	1.51	1.58	1.60	0.97	1.0	1.03
CRYSTALS-DILITHIUM	256	8,380,417	RTL	8	2	1,899	2,041	392	445	294	0.66
			HLS	8	2	1,977	2,329	401	434	294	0.67
			HLS/RTL	1.0	1.0	1.04	1.14	1.02	0.97	1.0	1.02

TABLE III
COMPARISON WITH THE RESULTS OF THE PREVIOUS IMPLEMENTATION OF THE NTT UNIT [7], FOR ZYNQ-7000. $N = 1024, q = 12,289$.

	DSPs	BRAM 18K	Slices	LUTs	FFs	Max Freq (MHz)	Clock cycles	Latency (μs)
Kuo et al. [7]	8	10	N/A	2,832	1,381	150	2,616	17.44
This work RTL	4	10	357	898	1,117	188	2,032	10.80
This work HLS	4	10	811	1,521	2,695	180	2,032	11.29
HLS/RTL	1.00	1.00	2.27	1.69	2.41	0.96	1.00	1.04

in Fig. 2.

The approximate total number of clock cycles, for the case when $log_2(N)$ is even, is: **PSIS_MUL**: $N/4 + $ **NTT**: $N/4 * log_2(N)/2 + $ **COEF_MUL**: $N/4 + $ **INTT**: $N/4 * log_2(N)/2 + $ **IPSIS_MUL**: $N/4 = (Nlog_2(N) + 3N)/4$. The red lines in Fig. 1 represent four likely critical paths in this design.

V. RESULTS

The maximum clock frequency and resource utilization have been generated by performing logic synthesis, placing, and routing using Vivado 2018.3. Our target platform is Zynq UltraScale+ MPSoC. Our results indicate that the latency in clock cycles, the number of DSPs, and the number of BRAMs is almost identical for the RTL- and HLS-based implementations. The penalty for using HLS in terms of the maximum clock frequency varies between 3% and 4%. The overhead in terms of the resource utilization varies in the following ranges: 7%-60% in terms of Slices, 2%-52% in terms of LUTs, and 2%-58% in terms of FFs.

The latency of the entire polynomial multiplication, assuming that one operand is already in the NTT domain as a result of precomputations, was reported by Du et al. in [8] as $(Nlog_2(N)/4 + N/2)$ clock cycles. The design by Chen et al. [3] does not make the same assumption, and reports the latency equal to $(3Nlog_2(N) + N)/4$ clock cycles. The latency of our design, making the same assumption as [8] is $(Nlog_2(N) + 3N)/4$, and thus is very similar to that reported in [8], and significantly smaller than that reported in [3].

All results in Table III were generated using Zynq-7000. The reported latency of the design by Kuo et al. [7] does not include the Order-Reverse step, which typically contributes 40% of latency, while the designs presented in this paper include the latency of the Reordering step. Thus, it is clear that both the RTL and HLS designs presented in this paper outperform the design by Kuo et al. in terms of all performance metrics (except the number of FFs for the HLS design).

REFERENCES

[1] "Post-Quantum Cryptography: Round 2 Submissions," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions, Apr. 2019.

[2] 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. Cham: Springer International Publishing, 2016, pp. 124–139.

[3] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede, "High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[4] C. P. Renteria-Mejia and J. Velasco-Medina, "High-Throughput Ring-LWE Cryptoprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2332–2345, Aug. 2017.

[5] T. Guneysu, V. Lyubashevsky, and T. Poppelmann, "Lattice-Based Signatures: Optimization and Implementation on Reconfigurable Hardware," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1954–1967, Jul. 2015.

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

[7] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, "High Performance Post-Quantum Key Exchange on FPGAs," Cryptology ePrint Archive 2017/690, Feb. 2018.

[8] C. Du, G. Bai, and X. Wu, "High-Speed Polynomial Multiplier Architecture for Ring-LWE Based Public Key Cryptosystems," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI - GLSVLSI '16*. Boston, Massachusetts, USA: ACM Press, 2016, pp. 9–14.