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

Duc Tri Nguyen Viet Ba Dang Kris Gaj

Cryptographic Engineering Research Group, George Mason University, Fairfax, VA, U.S.A.

INTRODUCTION

- When **quantum computers** become scalable and reliable, they are likely to **break all public key standards**, such as RSA and Elliptic Curve Cryptography.
- Due to an emerging threat of quantum computing, one of the major challenges facing the cryptographic community is a timely transition from traditional public-key cryptosystems, such as RSA and Elliptic Curve Cryptography, to a new class of algorithms, collectively referred to as **Post-Quantum Cryptography (PQC)**
- 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_2N is even. The latency of our proposed architecture is approximately equal to $(Nlog_2N + 3N)/4$ clock cycles. As a proof of concept, we implemented the NTT operation for several parameter sets used in the PQC algorithms NewHope, FALCON, qTESLA, and CRYSTALS-DILITHIUM.

BACKGROUND

Number Theoretic Transform

- By using NTT, a multiplication in $R_q[x] = Z_q[x]/(x^n + 1)$ can be computed as follows:

$$C = \mathbf{NTT}^{-1}(\bar{C}) = \mathbf{NTT}^{-1}(\bar{A} \quad \bar{B}) = \mathbf{NTT}^{-1}(\mathbf{NTT}(A) \quad \mathbf{NTT}(B))$$

- where $\psi^2 = \omega$, and a, b, c are polynomials in $R_q[x]$

$$A = (a_0, \psi a_1, \psi^2 a_2, \ldots, \psi^{n-1} a_{n-1})$$
$$B = (b_0, \psi b_1, \psi^2 b_2, \ldots, \psi^{n-1} b_{n-1})$$
$$C = (c_0, \psi c_1, \psi^2 c_2, \ldots, \psi^{n-1} c_{n-1})$$

Algorithm 1 Iterative NTT

Require: $F(x)$ $R_q[x]$; $ROM[i] = \omega^i$, $\omega^n = 1 \bmod q$
Ensure: $\bar{F}(x) = NTT(F)$

```
F ← BitReverse(F)
for s = 0 to log2(n) − 1 by 1 do
    m ← 2  s
    ω_m ← n/m
    i ← 0
    for j = 0 to m/2 by 1 do
        for k = 0 to n by m do
            u ← F[k + j]
            t ← F[k + j + m/2]  ROM[i]
            F[k + j] ← u + t
            F[k + j + m/2] ← u − t
        end for
        i ← i + ω_m
    end for
end for
```

- In this paper, we divide the polynomial multiplication into two modes:
  - **NTT**: Forward (NTT) and Inverse NTT ($INTT$) transform
  - **MUL**: Coefficient-wise multiplication by ψ^i ($PSIS\_MUL$), ψ^{-1} ($IPSIS\_MUL$) and two polynomial ($COEF\_MUL$).

KRED Modular Reduction

- In the reference software implementation, Montgomery modular reduction is used. However, this approach requires performing multiplication before reduction. An alternative approach, targeting hardware implementations: KRED Modular Reduction.

Algorithm 2 K-RED

1: **function K-RED**(C)
2: $C_0 \leftarrow C \bmod 2^m$
3: $C_1 \leftarrow C/2^m$
4: $D \leftarrow kC_0 - C_1$
5: return D
6: **end function**

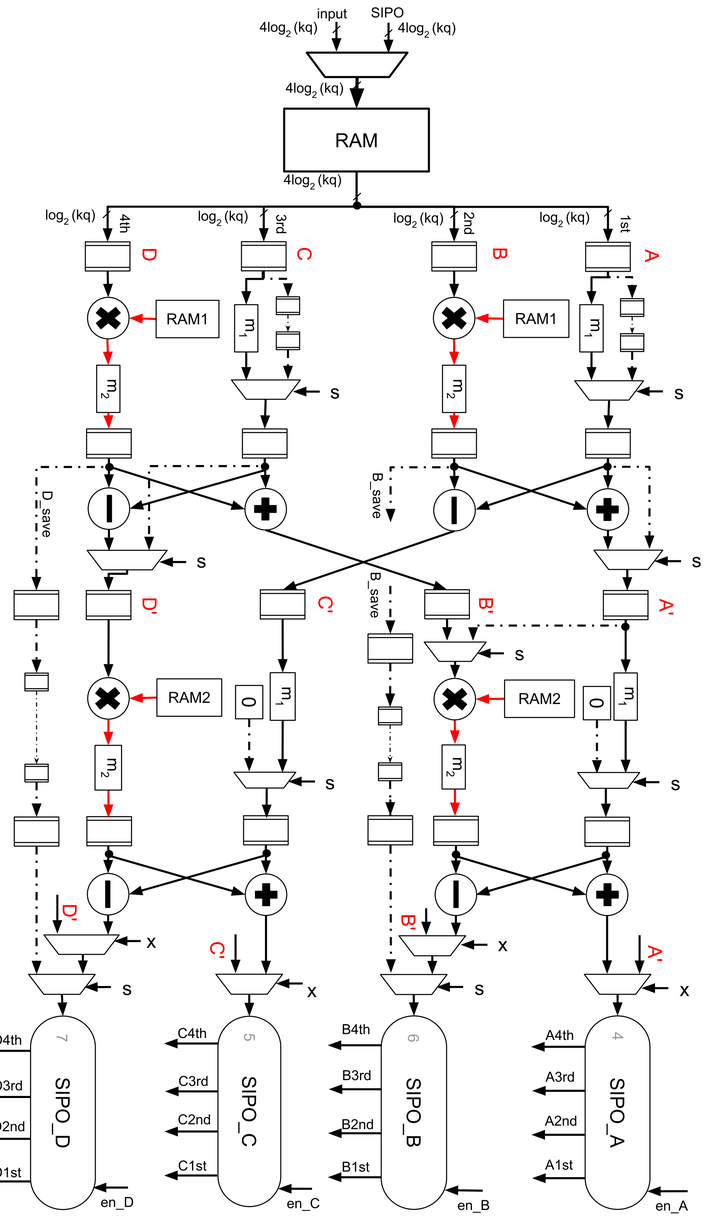
Algorithm 3 K-RED-2x

1: **function K-RED2-x**(C)
2: $C_0 \leftarrow C \bmod 2^m$
3: $C_1 \leftarrow C/2^m \bmod 2^m$
4: $C_2 \leftarrow C/2^{2m}$
5: **return** $k^2C_0 - kC_1 + C_2$
6: **end function**

- K-RED and K-RED-2x can be rewritten, using the shift and add operations, by taking advantage of the special form of $k = 2^{10} - 1$, for example, for CRYSTALS-Dilithium, its modular reduction can be rewritten as,
  K-RED(C) $= kC_0 - C_1 = (2^{10} - 1)C_0 - C_1$
  $= (C_0 \quad 10) - C_0 - C_1$, and
  K-RED-2x(C) $= k^2C_0 - kC_1 + C_2 =$
  $(C_0 \quad 20) - (C_0 \quad 11) + C_0 - (C_1 \quad 10) + C_1 + C_2$.
  Thus, K-RED and K-RED-2x use two and five adders respectively.

HARDWARE DESIGN

- Our NTT hardware architecture has 2x2 butterfly structure, which can process two layers of NTT with two butterfly units per layer. To implement this architecture in High Level Language, we actively avoid stalling, four coefficients are loaded in each clock cycle, without stalling, and placed into registers A, B, C, D. The square boxes m_1 and m_2 are KRED and KRED2x, respectively.
- SIPO (Serial In Parallel Out) is a simple Linear Shift Register, used to convert serial input to parallel output. A different number of registers is placed inside of each SIPO to make sure that only one SIPO can be full at a time. The benefit of the SIPO design is its role in preparing coefficients for the execution of the next NTT stages on the fly.



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

- When $S = 0$, **MUL** mode, the coefficients in the lines B and D are multiplied by coefficients from RAM1. The obtained result 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.
- When $S = 1$, **NTT** mode, four coefficients will go through the 2x2 butterfly structure, and results are written to SIPOs, coefficients in lines B and D are multiplied with ω_n^i or ω_n^{-i} depend on whether the circuit compute NTT or $INTT$.
- The red lines represent four likely critical paths in this design, which depends on routing delays, and impossible to predict without running the actual synthesis, placing, and routing.

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 $[0, 1, \ldots, n/2)$ for RAM1 and RAM2, except the gray area of RAM 1, where i $[0, 1, \ldots, n/4)$. For the KRED, the value of k is given in Table 1. If the REDC is used, k is assumed to be 1. $poly\_pk$ and $poly\_sk$ are NTT domain preloaded public and secret polynomials.

- 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.

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. The choice of this platform is consistent with our plan to extend our hardware accelerators for NTT into full software/hardware codesigns of the entire PQC candidates.
- The penalty for using HLS in terms of the maximum clock frequency and latency varies between **2%** and **5%**. The overhead in terms of the resource utilization is below 14% for all investigated candidates, except qTESLA.

Results of the implementations of the NTT unit for selected Round 2 PQC Candidates, using Zynq UltraScale+

	NewHope & Falcon		qTesla		CRYSTALS-DILITHIUM	
N	1024		1024		256	
q	12,289		8,404,993		8,380,417	
	RTL	**HLS/RTL**	RTL	**HLS/RTL**	RTL	**HLS/RTL**
DSP	4	**1.0**	8	**1.0**	8	**1.0**
BRAM	5	**1.0**	8	**1.0**	2	**1.0**
LUT	849	**1.02**	1,286	**1.51**	1,899	**1.04**
FF	802	**1.02**	2,160	**1.58**	2,041	**1.14**
Slice	163	**1.07**	283	**1.60**	392	**1.02**
Freq.	476	**0.96**	467	**0.97**	445	**0.97**
Cycles	1,324	**1.0**	1,363	**1.0**	294	**1.0**
Latency (μs)	2.78	**1.05**	2.92	**1.05**	0.66	**1.02**

- The reported latency of the design by Kuo et al. 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.

Comparison with the results of the previous implementation of the NTT unit Kuo et al.[11], for Zynq-7000

	Kuo et al. [11]	This Work		
N	1024			
q	12289			
	RTL	RTL	HLS	**HLS/RTL**
DSP	8	4	4	**1.0**
BRAM 18K	10	10	10	**1.0**
LUT	2,832	898	1,521	**2.27**
FF	1,381	1117	2,695	**1.68**
Slice	N/A	357	811	**2.41**
Freq.	150	188	180	**0.96**
Cycles	2,616	2,032	2,032	**1.0**
Latency (μs)	17.44	10.80	11.29	**1.04**

Conclusions and Future Work

- We have proposed a modified hardware architecture of one of the major operations of several Round 2 lattice-based PQC candidates - Number Theoretic Transform. We then implemented and benchmarked this architecture using both the novel approach based on High-Level Synthesis and using the traditional RTL-based approach.
- Our results confirm the potential of the HLS-based approach to efficiently implement at least major building blocks of PQC algorithms, if not the entire algorithms themselves. The obtained results are comparable to those obtained using the RTL approach in terms of timing, and leg behind only in terms of the number of LUTs, FFs, and Slices.
- We will also attempt full hardware implementations of all Round 2 PQC candidates that could potentially benefit from NTT, using both the HLS- and RTL-based design methodologies.

Acknowledgment

This material is based upon work supported by the U.S. Department of Commerce / National Institute of Standards and Technology under Grant no. 60NANB15D058 and Grant no. 70NANB18H218, and by the National Science Foundation under Grant no. CNS-1801512.