A Survey on High-Level Synthesis Approaches for Number Theoretic Transform on FPGAs

Wei He and Sunwoong Kim
Department of Electrical and Microelectronic Engineering
Rochester Institute of Technology, Rochester, NY, USA
Email: {wh9297, sskeme}@rit.edu

***Abstract*—Number theoretic transform (NTT) is a critical operation in several post-quantum cryptography (PQC) schemes, which enables fast polynomial multiplication. However, it constitutes a significant portion of the total computational time in PQC schemes, requiring custom hardware accelerators. Manual register-transfer level (RTL) coding often requires re-designing memory structures and control units when configurations change. In contrast, the high-level synthesis (HLS) approach offers higher design abstraction, reducing development and verification time. This paper provides a comprehensive review of recent HLS-based NTT implementations, presenting their design strategies and comparing hardware resource utilization and performance on field-programmable gate array platforms. Finally, the paper discusses potential future research directions to narrow the performance gap between RTL coding and HLS approaches.**

I. Introduction

Security has become an increasingly critical concern. Specifically, the rapid growth of connected devices over public networks has expanded the potential entry points for cyberattacks. In addition, as real-world applications increasingly integrate artificial intelligence and untrusted third-party servers, which often handles sensitive client data, security risks increase. To strengthen security across these domains, cryptographic techniques have become an essential tool.

Several post-quantum cryptographic (PQC) schemes that involve polynomial multiplications leverage the number theoretic transform (NTT), which is a generalization of the discrete Fourier transform (DFT) to finite fields. NTT enables efficient computation of convolutions on integer sequences, avoiding round-off errors [1]. For example, direct polynomial multiplication, which has a computational complexity of $\mathcal{O}(n^2)$, is converted into a process that involves NTT, pointwise multiplication, and inverse NTT (INTT). This approach achieves a computational complexity of $\mathcal{O}(n\log_2 n)$, which is asymptotically faster than the well-known Karatsuba algorithm with $\mathcal{O}(n^{\log_2 3})$ and the generalized Toom-Cook algorithm with $\mathcal{O}(n^{\log_k(2k-1)})$, where a polynomial is divided into 2 and k parts, respectively [2].

Despite the improved efficiency, performing NTT on a CPU in real time becomes progressively challenging as the length of an integer sequence n increases. For example, fully homomorphic encryption (FHE) that performs operations on encrypted data without decryption [3] often uses large n values, such as 2^{17}, for NTT in contemporary real-world applications [4]. However, these applications become impractically slow on a CPU, with NTT and INTT processes accounting for more than 50% of the total execution time [5]. To address the slow processing speed issue, numerous studies have proposed custom NTT hardware architectures targeting field-programmable gate array (FPGA) platforms [4], [6]–[14].

FPGA-based hardware designs are generally described using two methods: Manual register-transfer level (RTL) coding and high-level synthesis (HLS). Typically, the former is highly effective for creating optimized hardware. However, it is time-consuming and labor-intensive in both development and verification. In contrast, the HLS approach addresses this challenge by automatically converting high-level programming language code, such as C/C++ code, into hardware description language (HDL) code. This method not only boosts productivity but also supports efficient exploration of design options in the early stage by organizing Pareto points by latency and selecting the optimal point that meets design constraints.

Recently, several survey papers on the development of NTT have been published [2], [15]–[18]. However, there is a lack of comprehensive reviews focusing on HLS-based NTT implementations. This review paper consolidates recent HLS-based NTT designs, including their FPGA benchmark results, and offers a thorough comparative evaluation. It serves as a valuable guide for researchers seeking to quickly implement NTT hardware accelerators in PQC schemes.

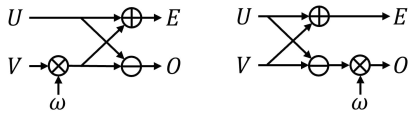
II. Number Theoretic Transform

A generalized version of DFT over finite fields, specifically integers modulo a prime number q, is called as NTT. Let $\mathbf{a}$ be a vector of polynomial coefficients $a_i \in \mathbb{Z}_q$ ($i \in [0, ..., n-1]$) and the degree of a polynomial is $n-1$. NTT is then defined as $\hat{\mathbf{a}}$ = *NTT*($\mathbf{a}$), where:

$$\hat{a}_j = \sum_{i=0}^{n-1} a_i \omega_n^{ij} \pmod q \tag{1}$$

for $j \in [0, ..., n-1]$. ω_n is the primitive n-th root of unity (also known as a twiddle factor), which satisfies $\omega_n^n \equiv 1 \pmod q$ and $\omega_n^k \not\equiv 1 \pmod q$ for $k < n$. The inverse transform that converts from the NTT form back to the regular coefficient form is defined as $\mathbf{a}$ = *INTT*($\hat{\mathbf{a}}$), where:

$$a_j = n^{-1} \sum_{i=0}^{n-1} \hat{a}_i \omega_n^{-ij} \pmod q. \tag{2}$$



(a) (b)

Fig. 1. Radix-2 butterfly configurations for fast NTT/INTT. (a) CT butterfly (b) GS butterfly.

Using NTT and INTT, the (positive-wrapped) convolution of vectors $\mathbf{a}$ and $\mathbf{b}$ is performed as follow:

$$\mathbf{c} = \mathit{INTT}(\mathit{NTT}(\mathbf{a}) \circ \mathit{NTT}(\mathbf{b})), \tag{3}$$

where $\circ$ denotes an element-wise multiplication in $\mathbb{Z}_q$.

Equations (1) and (2) still require a computational complexity of $\mathcal{O}(n^2)$, which limits the reduction in the computational complexity of polynomial multiplications. To address this, the divide-and-conquer approach, where the n-point NTT is recursively divided into k n/k-point NTTs, is commonly used. The value of k determines the radix of a butterfly unit (BU), which is a basic building block of NTT. Using this approach, the overall computational complexity is reduced to $\mathcal{O}(n\log_2 n)$. Two well-known configurations for this approach are Cooley-Tukey (CT) and Gentleman-Sande (GS) algorithms. For a detailed derivation of these algorithms from (1) and (2), refer to [2], [19]. Visual representations of butterfly operations based on the CT and GS algorithms are shown in Fig. 1. Each radix-2 BU contains a modular adder, a modular subtractor, and a modular multiplier to generate two outputs from two inputs.

When performing polynomial multiplication on the polynomial ring $\mathbb{Z}_q[x]/\langle\phi(x)\rangle$, a reduction operation by $\phi(x)$ is required [18]. However, if $\phi(x)$ is of the form x^n+1, which is a common setting in PQC schemes [2], the negative-wrapped convolution technique can be used to eliminate the need for polynomial reduction. This method requires multiplications between polynomials and powers of the $2n$-th root of unity, denoted as ψ_{2n}, which satisfies $\psi_{2n}^{2n} \equiv 1 \pmod q$ and $\psi_{2n}^{i} \not\equiv 1 \pmod q$ for $i < 2n$. To apply this modification, the modulus q must satisfy $q \equiv 1 \pmod{2n}$. With the negative-wrapped convolution technique, (3) is modified as follows:

$$\mathbf{c} = \mathit{INTT}(\mathit{NTT}(\mathbf{a} \circ \boldsymbol{\psi}) \circ \mathit{NTT}(\mathbf{b} \circ \boldsymbol{\psi})) \circ \boldsymbol{\psi}^{-\mathbf{1}}, \tag{4}$$

where $\boldsymbol{\psi} = [\psi_{2n}^0, ..., \psi_{2n}^{n-1}]$ and $\boldsymbol{\psi}^{-1} = [\psi_{2n}^0, ..., \psi_{2n}^{-(n-1)}]$. ψ_{2n}^i and ω_n^{ij} are combined into a single entity. In computation-intensive hardware designs, they are precomputed and stored in memory, whereas in memory-intensive designs, they are calculated on the fly [4].

III. Comprehensive Review and Comparison

This section presents a comprehensive comparison of NTT designs based on FPGA and HLS. Table I summarizes the key features of these designs, including their open-source availability. Note that unrevealed data or data that is difficult to infer is represented by a hyphen.

1) HLS Tools: AMD-Xilinx Vitis HLS (formerly Vivado HLS) is the most commonly used tool for the HLS-based NTT designs. The design flow in this tool is shown in Fig. 2. The Vitis HLS-based NTT designs use HLS pragmas, such as `ARRAY_PARTITION`, `PIPELINE`, and `UNROLL` in their C/C++ source code. While Vitis HLS has been the dominant tool for HLS-based NTT designs, other HLS tools are also used. Specifically, Mentor/Siemens Catapult HLS is used for NTTSuite [24], and Intel's oneAPI Data Parallel C++ compiler, which provides HLS capabilities through SYCL for Intel/Altera FPGA devices, is used in Intel's HEXL-FPGA project [26].

2) Butterfly Radix: Among the HLS-based NTT designs, the majority use radix-2 BUs. This is because radix-2 BUs can handle input sequences of various lengths and simplify the design of datapath, memory allocation, and control logic. Exceptionally, Nguyen *et al.* use a radix-4 BU, specifically employing a 2×2 butterfly structure aimed at reducing memory usage [6], [21]. To improve efficiency, they implement specialized techniques, such as a serial-in-parallel-out configuration to facilitate grouping of input data.

3) NTT/INTT Algorithm: Most of the works in Table I use the CT and GS algorithms or their variants. In contrast, NTTSuite uses specialized dataflow algorithms, such as the Pease algorithm [24]. While the CT and GS algorithms typically require in-place processing, where the butterfly input pattern changes stage by stage, the Pease algorithm supports out-of-place processing. Specifically, data are stored in an auxiliary array during butterfly computation and later transferred back. It effectively avoids complex array partition conflicts and allows for more efficient pipelining opportunities but requires additional internal memory.

Sample reordering methods are closely tied to the NTT/INTT algorithm. In polynomial multiplication that involves CT butterfly, element-wise multiplication, and GS butterfly in order, the output of CT butterfly and the input of GS butterfly are usually in bit-reversed order [2].

4) Target Applications: The target applications for the HLS-based NTT designs are mainly PQC schemes, aimed at securing communications and data against the future threats posed by quantum computers. Specifically, NewHope is a key-exchange protocol, and CRYSTALS-Kyber is a public-key encryption and key-exchange scheme. CRYSTALS-Dilithium, Falcon, and qTESLA are digital signature schemes. FHE is an encryption form that allows computations on encrypted data without needing to decrypt it. All of these target applications are based on lattice-based cryptography. The number of NTT samples (n) and the sample size ($\lceil\log_2 q\rceil$ bits) are determined based on the specific requirements of the target application. Specifically, FHE demands a larger number of samples, reaching up to 2^{16}.

5) Modular Reduction: Two traditional modular reduction algorithms, Barrett reduction [28] and Montgomery reduction [29], are widely used to replace the expensive division in modular reduction with several integer multiplications [30]. In addition to these algorithms, the modular reduction algorithm

TABLE I
REVIEW OF FPGA- AND HLS-BASED NTT IMPLEMENTATIONS IN LITERATURE AND OPEN-SOURCE PROJECTS

	Tool	Algorithm	Target Application	No. Samples (n)	Sample size ($\lceil \log_2 q \rceil$ bits)	Reduction Algorithm	Open-source Availability
ISQED'18 [20]	Vivado HLS	GS	FHE	$[2^{10}, 2^{16}]$	10†	-	✘
FPT'19 [21]	Vivado HLS	CT	NewHope, Falcon, qTESLA, Dilithium	$[2^8, 2^{10}]$	[14, 24]	Longa–Naehrig	✘
ESL'20 [16]	Vivado HLS	CT	PQC algorithms, FHE	2^{10}	14	Longa–Naehrig	✘
ARC'20 [6]	Vivado HLS	CT	NewHope, Kyber	$[2^8, 2^{10}]$	[12, 14]	Longa-Naehrig, Montgomery	✘
ISVLSI'21 [22]	Vivado HLS	CT	Dilithium, Kyber	2^8	[12, 23]	Barrett, Montgomery	✘
TC'22 [17]	Vivado HLS	GS	Kyber, NewHope, Dilithium, Falcon, qTESLA, FHE	$[2^8, 2^{12}]$	[13, 60]	Montgomery	✘
VLSI-SoC'22 [1]	Vivado HLS	CT/GS	Dilithium	2^8	23	Montgomery	✘
PAINE'23 [23]	Intel HLS	CT/GS	Kyber	2^8	12	-	✘
NTTSuite [24]	Catapult HLS	Pease	FHE	$[2^{10}, 2^{16}]$	32	Shoup	✔
Vitis Tutorial [25]	Vitis HLS	CT	Kyber	2^7	12	Montgomery	✔
HEXL-FPGA [26]	oneAPI	CT/GS‡	FHE	$[2^5, 2^{15}]$	[20, 62]	Barrett‡	✔

†This value is obtained from [17].
‡These data are obtained from Intel's HEXL library [27].

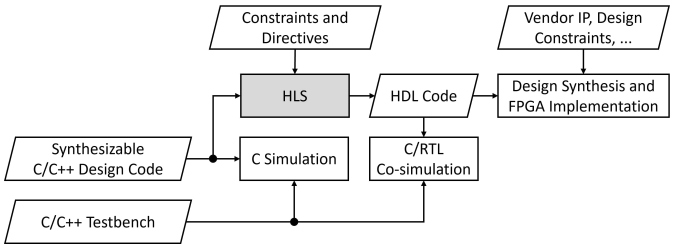


Fig. 2. FPGA design flow using Vitis HLS.

proposed by Longa and Naehrig [31] is used in [6], [16], [21]. This algorithm introduces a bitwise add-shift modular reduction, designed for a special prime of the form $q = k \cdot 2^m + 1$, where k is odd and smaller than 2^m. It offers an advantage in terms of logic size, as it requires fewer multiplications. However, in scenarios involving multiple smaller moduli, which is common in residue number system-based FHE schemes [32], [33], where each modulus may not follow the specific form, the Longa-Naehrig reduction algorithm is not applicable.

6) Twiddle Factor Generation: Although not shown in Table I, all works in this table precompute twiddle factors and pre-load them in internal memory, rather than calculating them on the fly, with a few exceptions that do not explicitly mention how their twiddle factors are handled. However, some manual RTL coding-based studies generate twiddle factors on the fly, particularly in the context of FHE, where various moduli require different twiddle factors and excessive memory space [4], [34]. Specifically, in [34], optimal parameter settings balance pre-calculated storage and on-the-fly generation, either sequentially or in parallel.

IV. FPGA PERFORMANCE BENCHMARKING

1) Comparisons between Different Works: Table II compares the FPGA implementation results of the Vivado/Vitis HLS-based NTT designs. Direct comparisons are challenging due to variations in FPGA devices, sample counts, and sample sizes. However, we focus on comparing designs with the most similar configurations possible here. When comparing designs on the Virtex-7 device with 2^{10}-point sample input, ESL'20 [16] and TC'22 [17] employ 8 BUs and support a 14-bit modulus. Although the frequencies differ, the latency in cycles is roughly identical. In contrast, ISQED'18 [20] achieves a significant reduction in latency, which is approximately one-third of that of TC'22. This is accomplished by using 2.25× BUs, although the reduction algorithm is not specified in this previous work. However, the extensive loop unrolling of the NTT in ISQED'18 fails to achieve an optimal trade-off between latency and resource usage, leaving significant room for optimization.

For $n = 2^{10}$, NTTSuite (Pease_nc version) using four BUs [24], which reduces overhead copying, achieves a low latency of approximately 1,400 cycles at 196 MHz. This results in an 11× improvement in cycle count over TC'22 using eight BUs while consuming fewer digital signal processing (DSP) slices and look-up tables (LUTs). However, NTTSuite significantly increases on-chip block RAM (BRAM) usage, surpassing those of other NTT designs with similar configurations. Therefore, the approach used in NTTSuite may be beneficial for FPGA devices with abundant internal memory.

2) Comparisons in Same Works: ISQED'18, TC'22, and NTTSuite present implementation results across different configurations. Specifically, in ISQED'18, the number of BUs and the number of samples change together. Note that ISQED'18 does not explicitly specify the number of BUs deployed in their designs. Therefore, the values are inferred by multiplying the total number of iterations in the outer loop by the number of BUs operating in parallel in the inner loop. Although not clearly detailed in the previous work, the reported number of DSP slices implies that the number of parallel BUs in the inner loop doubles when n reaches 2^{13}. In contrast to the trends observed in DSP, flip-flop (FF), and LUT resources, BRAM consumption and latency increase exponentially as the value of n changes. This indicates that BRAM is likely the primary performance bottleneck.

TABLE II
BENCHMARKING HLS-BASED NTT HARDWARE DESIGNS ON AMD-XILINX FPGAS

	Device	No. BUs	No. Samples (n)	Sample size ($\lceil \log_2 q \rceil$ bits)	Freq. (MHz)	BRAM (36K)	DSP	FF	LUT	Latency (μs)
ISQED'18 [20]	Virtex 7	18	2^{10}	10	100	21.5	19	30,498	38,984	53
		20	2^{11}			24.5	21	38,224	46,738	107
		22	2^{12}			41.5	22	44,767	58,082	221
		44	2^{13}			75.5	44	52,552	65,143	457
		48	2^{14}			188.5	48	60,592	76,667	949
		52	2^{15}			402.0	53	69,476	87,477	1,974
		56	2^{16}			885.0	56	80,275	102,584	4,105
FPT'19 [21]	Zynq US+	4†	2^{8}	23	434	2.0	8	2,329	1,977	1
			2^{10}	14	455	5.0	4	822	865	3
			2^{10}	24	455	8.0	8	3,423	1,939	3
ESL'20 [16]	Virtex 7	8	2^{10}	14	200	2.0	8	3,243	4,737	83
ISVLSI'21 [22]	Artix 7	-	2^{8}	23	58	0.0	248	2,827	7,849	10
TC'22 [17]	Virtex 7	1	2^{10}	14	100	4.0	3	0	1,045	307
		8	2^{10}	14		16.0	24		11,305	154
		8	2^{11}	30		64.0	96		13,886	338
		8	2^{12}	60		128.0	360		17,768	737
VLSI-SoC'22 [1]	Zynq US+	2	2^{8}	23	150-400	5.0	12	1,371	2,015	5‡
NTTSuite [24]*	Virtex 7	4	2^{10}	32	196	40.0	4	8,592	6,005	7
		4	2^{12}			40.0	4	6,198	6,079	32
		4	2^{14}			40.0	4	5,334	6,073	147
		4	2^{16}			40.0	64	5,409	6,145	669
		16	2^{10}			160.0	16	32,866	23,474	2
		16	2^{12}			160.0	16	32,976	23,696	9
		16	2^{14}			160.0	16	27,902	23,737	38
Vitis Tutorial [25]§	Versal Premium	7	2^{7}	12	100	19.0	21	13,720	17,940	999

†This value is derived based on the use of radix-2 butterfly units.
*The results are for the Pease_nc designs.
‡The frequency is assumed to be 300 MHz.
§Among the four implementations, version 3 is the one selected for use.

In TC'22, increasing the number of BUs leads to a decrease in latency, but this reduction is not proportional. Specifically, when the number of BUs increases eightfold, the latency is reduced by approximately half. The authors of this previous work attribute this observation to inefficiencies in the HLS tool. When the number of BUs is fixed at 8, increasing the number of samples results in a latency growing roughly proportionally. However, the number of DSPs increases at a much higher rate, primarily driven by the sample size. DSP slices are used for multipliers in each BU and a Montgomery modular reduction operator. The multiplier block in a DSP slice has a limited bit width. For example, in the FPGA device used in TC'22, the multiplier of the DSP slice (DSP48E1) supports operand bit widths of 25 and 18 (signed), respectively. When the bit width exceeds these limits, additional DSP slices are required.

Compared to TC'22, NTTSuite shows a more ideal relationship between latency and the number of samples (when the number of BUs is fixed). For example, when four BUs are used and the value of n increases from 2^{14} to 2^{16}, the approximate increase ratio in latency is calculated as $(\frac{2^{16}}{2} \cdot \log_2 2^{16})/(\frac{2^{14}}{2} \cdot \log_2 2^{14}) = 4.57$, and the actual observed ratio is 4.55. This indicates that NTTSuite minimizes the execution time outside of the BU operations by using additional BRAMs.

V. CONCLUSIONS AND FUTURE DIRECTIONS

This paper offers a comprehensive review and comparison of HLS-based NTT designs. The FPGA implementation results serve as benchmarks for assessing the feasibility and efficiency of these designs. This paper could be a useful resource for the development of future NTT-based PQC hardware accelerators.

HLS-based NTT implementation provides a convincing trade-off between development/verification time and final performance. According to [17], [23], manual RTL coding for NTT designs requires approximately 450 man-hours, while HLS-based design reduces this to around 60 man-hours. However, HLS does come with performance limitations. For example, the hardware utilization results for NTT-based PQC accelerators in [6] show that the HLS-to-RTL ratio ranges from 1.00 to 1.53. This gap may widen further when dealing with a large number of BUs or samples, potentially requiring manual adjustments [17].

To reduce this gap, the following research directions can be explored: Although HLS enables efficient design space exploration to identify the optimal parameter combination, its adoption has been limited. Developing specific design space exploration frameworks for NTT-based PQC hardware accelerators could be a promising area for further research. In addition, the final NTT hardware designs are heavily influenced by the HLS pragmas and directives applied. To address this challenge, supplementary compiler-level tools that refine C/C++ code to be more friendly to HLS tools, such as ScaleHLS [35], could be used.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2347253.

REFERENCES

[1] A. El-Kady, A. P. Fournaris, E. Haleplidis, and V. Paliouras, "High-level synthesis design approach for number-theoretic multiplier," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2022, pp. 1–6.

[2] A. Satriawan, I. Syafalni, R. Mareta, I. Anshori, W. Shalannanda, and A. Barra, "Conceptual review on number theoretic transform and comprehensive review on its implementations," *IEEE Access*, vol. 11, pp. 70 288–70 316, 2023.

[3] J. H. Cheon, A. Costache, R. C. Moreno, W. Dai, N. Gama, M. Georgieva, S. Halevi, M. Kim, S. Kim, K. Laine *et al.*, "Introduction to homomorphic encryption and schemes," *Protecting Privacy through Homomorphic Encryption*, pp. 3–28, 2021.

[4] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware architecture of a number theoretic transform for a bootstrappable rns-based homomorphic encryption scheme," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 56–64.

[5] A. Kim, A. C. Mert, A. Mukherjee, A. Aikata, M. Deryabin, S. Kwon, H. C. Kang, and S. S. Roy, "Exploring the advantages and challenges of fermat ntt in fhe acceleration," in *Annual International Cryptology Conference*. Springer, 2024, pp. 76–106.

[6] D. T. Nguyen, V. B. Dang, and K. Gaj, "High-level synthesis in implementing and benchmarking number theoretic transform in lattice-based post-quantum cryptography using software/hardware codesign," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th International Symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16*. Springer, 2020, pp. 247–257.

[7] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of newhope-nist on fpga using low-complexity ntt/intt," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 49–72, 2020.

[8] Y. Su, B.-L. Yang, C. Yang, Z.-P. Yang, and Y.-W. Liu, "A highly unified reconfigurable multicore architecture to speed up ntt/intt for homomorphic polynomial multiplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 8, pp. 993–1006, 2022.

[9] P. Duong-Ngoc, S. Kwon, D. Yoo, and H. Lee, "Area-efficient number theoretic transform architecture for homomorphic encryption," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 3, pp. 1270–1283, 2022.

[10] S. Di Matteo, M. L. Gerfo, and S. Saponara, "Vlsi design and fpga implementation of an ntt hardware accelerator for homomorphic seal-embedded library," *IEEE Access*, vol. 11, pp. 72 498–72 508, 2023.

[11] S.-H. Liu, C.-Y. Kuo, Y.-N. Mo, and T. Su, "An area-efficient, conflict-free, and configurable architecture for accelerating ntt/intt," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 3, pp. 519–529, 2023.

[12] Z. Cheng, B. Zhang, and M. Pedram, "A high-performance, conflict-free memory-access architecture for modular polynomial multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 2, pp. 492–505, 2023.

[13] F. Hirner, A. C. Mert, and S. S. Roy, "Proteus: A pipelined ntt architecture generator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[14] R. Shrivastava, C. P. Ratnala, D. M. Puli, and U. Banerjee, "A unified hardware accelerator for fast fourier transform and number theoretic transform," in *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2025, pp. 1–5.

[15] F. Valencia, A. Khalid, E. O'Sullivan, and F. Regazzoni, "The design space of the number theoretic transform: A survey," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2017, pp. 273–277.

[16] E. Ozcan and A. Aysu, "High-level synthesis of number-theoretic transform: A case study for future cryptosystems," *IEEE Embedded Systems Letters*, vol. 12, no. 4, pp. 133–136, 2019.

[17] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2829–2843, 2020.

[18] A. C. Mert, F. Yaman, E. Karabulut, E. Öztürk, E. Savaş, and A. Aysu, "A survey of software implementations for the number theoretic transform," in *International Conference on Embedded Computer Systems*. Springer, 2023, pp. 328–344.

[19] Z. Liang and Y. Zhao, "Number theoretic transform and its applications in lattice-based cryptosystems: A survey," *arXiv preprint arXiv:2211.13546*, 2022.

[20] K. Kawamura, M. Yanagisawa, and N. Togawa, "A loop structure optimization targeting high-level synthesis of fast number theoretic transform," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2018, pp. 106–111.

[21] D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of ntt-based post-quantum cryptography algorithms," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 371–374.

[22] D. Soni and R. Karri, "Efficient hardware implementation of pqc primitives and pqc algorithms using high-level synthesis," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 296–301.

[23] M. Bisheh-Niasar, D. Lo, A. Parthasarathy, B. Pelton, B. Pillilli, and B. Kelly, "Pqc cloudization: Rapid prototyping of scalable ntt/intt architecture to accelerate kyber," in *2023 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. IEEE, 2023, pp. 1–7.

[24] J. Ding, Y. Liu, L. Sun, and B. Reagen, "Nttsuite: Number theoretic transform benchmarks for accelerating encrypted computation," *arXiv preprint arXiv:2405.11353*, 2024.

[25] "Polynomial vectorization using number theoretic transforms: A real world case study using vitis hls code analyzer," https://github.com/Xilinx/Vitis-Tutorials, 2025.

[26] Y. Meng, S. Butt, Y. Wang, Y. Zhou, S. Simoni *et al.*, "Intel Homomorphic Encryption Acceleration Library for FPGAs (version 2.0)," https://github.com/intel/hexl-fpga, 2022.

[27] F. Boemer, S. Kim, G. Seifu, F. DM de Souza, and V. Gopal, "Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52," in *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2021, pp. 57–62.

[28] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[29] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[30] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "Fpga-based accelerators of fully pipelined modular multipliers for homomorphic encryption," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2019, pp. 1–8.

[31] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *International Conference on Cryptology and Network Security*. Springer, 2016, pp. 124–139.

[32] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

[33] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," in *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer, 2019, pp. 83–105.

[34] N. Im, H. Yang, Y. Eom, S.-C. Park, and H. Yoo, "Efficient twiddle factor generators for ntt," *Electronics*, vol. 13, no. 16, p. 3128, 2024.

[35] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, and D. Chen, "Scalehls: A new scalable high-level synthesis framework on multi-level intermediate representation," in *2022 IEEE international symposium on high-performance computer architecture (HPCA)*. IEEE, 2022, pp. 741–755.