Algorithm-Hardware Co-design of Split-Radix Discrete Galois Transformation for KyberKEM

Guangyan Li, Donglong Chen, Gaoyu Mao, Wangchen Dai, Abdurrashid Ibrahim Sanka, Ray C.C. Cheung, *Senior Member, IEEE*

Abstract—KyberKEM is one of the final round key encapsulation mechanisms in the NIST post-quantum cryptography competition. Number theoretic transform (NTT), as the computing bottleneck of KyberKEM, has been widely studied. Discrete Galois Transformation (DGT) is a variant of NTT that reduces transform length into half but requires more multiplication operations than the latest NTT algorithm in theoretical analysis. This paper proposes the split-radix DGT, a novel DGT variant utilizing the split-radix method, to reduce the computing complexity without compromising the transform length. Specifically, for length-128 polynomial, the split-radix DGT algorithm saves at least 10% multiplication operations compared with the latest NTT algorithm in theoretical analysis. Furthermore, we proposed a unified split-radix DGT processor with the dedicated stream permutation network for KyberKEM and implemented it on the Xilinx Artix-7 FPGA. The processor achieves at least 49.4% faster transformation and 65.3% faster component-wise multiplication, with at most 87% and 32% LUT-NTT area-time product and LUT-CWM area-time product, compared with the state-of-the-art polynomial multipliers in KyberKEM with the same BFU setting on similar platforms. Lastly, we designed a highly efficient KyberKEM architecture using the proposed split-radix DGT processor. The implementation results on Artix-7 FPGA show significant performance improvements over the state-of-the-art KyberKEM designs.

Index Terms—Discrete Galois transform, Split-radix, Negative wrapped convolution, Post-Quantum cryptography, Key encapsulation mechanism, Hardware, FPGA

1 Introduction

The rapid development of quantum computers and quantum algorithms such as Shor's algorithm [1] threatens the security basis of conventional public-key cryptosystems such as RSA and ECC. The urgent need to replace the conventional public-key cryptosystem with quantum-resistant cryptography, or the so-called post-quantum cryptography (PQC), drives the attention of researchers and standards organizations. In July 2020, the round 3 candidates of the NIST PQC competition were announced, and four public-key algorithms were disclosed as the finalists. The finalists include three lattice-based cryptography (i.e. CRYSTALS-KyberKEM [2], NTRU [3], SABER [4]), and one code-based cryptography (i.e. Classic McEliece [5]). As stated by the NIST, the PQC competition would evaluate the submissions by different criteria, including security, cost, and algorithm & implementation characteristics [6].

To evaluate the PQC candidates on the criteria of cost, there have been lots of published articles comparing different implementations of candidates. Within the supplementary functions of the proposed PQC schemes, the most time-consuming parts are the polynomial multiplication and the hash functions. The recent hardware and software/hardware co-design works offload the polynomial multiplication and the hash functions to dedicated hardware accelerators [7], [8], [9], [10].

The polynomial multiplication is computationally intensive. Recent solutions to polynomial multiplication could be in quasi-linear time $O(nlogn)$ using the number-theoretic transforms (NTTs) when we treat the polynomial multiplication as a discrete convolution problem. It is remarkable that the polynomial multiplication over different polynomial rings should be treated carefully. For instance, we could treat the polynomial multiplication over $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ as negative wrapped convolution between the vectors of the coefficients of input polynomials. The polynomials ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ are widely used in many lattice-based cryptography algorithms because of the high computing efficiency. For instance, the cryptosystems CRYSTALS-KyberKEM [2], SABER [4], and Luybashevsky's public-key cryptosystem [11], [12] employ the aforementioned polynomial rings in their cryptography algorithm.

1.1 Related Works

There are several works concerning the implementation of Crystals-KyberKEM. Software implementation of KyberKEM has been studied in [13], [14], [15]. Botros *et al.* proposed a memory-efficient high-speed optimization of KyberKEM on ARM Cortex-M4 core in [13]. Furthermore, the side-channel defence of KyberKEM was treated in [14], which is also based on ARM Cortex-M4 core. Likewise, Nguyen *et al.* [15] studied the software optimization of KyberKEM on a high-performance platform.

Pure hardware implementations of KyberKEM have been published in [9], [16], [17], [18], [19]. The work [9]

G. Li, G. Mao, A.I. Sanka, and R.C.C. Cheung are with the Department of Electrical Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong SAR, China. E-mail: guangyali5-c@my.cityu.edu.hk, gaoyumao3-c@my.cityu.edu.hk, iasanka2-c@my.cityu.edu.hk, and ccheung@ieee.org.
D. Chen is with the Faculty of Science and Technology, BNU-HKBU United International College, Zhuhai, China. E-mail: donglongchen@uic.edu.cn.
W. Dai is with the Zhejiang Lab, Hangzhou, China. E-mail: w.dai@my.cityu.edu.hk.
Corresponding author: Donglong Chen.

is a compact hardware implementation of KyberKEM for the third round submission in NIST PQC competition on Xilinx Artix-7 FPGA platform. The work [16] proposed an implementation for both FPGA and ASIC design with an improvement in polynomial sampling cores. Dang *et al.* proposed the high-performance implementation of KyberKEM, NTRU and Saber in [17] with a novel Polynomial Vector Multiplication Unit (PVMU) design. The work of Jati *et al.* [18] concerns the side-channel protection for KyberKEM in pure hardware.

On the other hand, software/hardware co-design implementations of KyberKEM have been published in [20], [21], [22], [23]. Banerjee *et al.* proposed an ASIC crypto-processor based on RISC-V architecture in [20] supporting Crystals-KyberKEM, Crystals-Dilithium, FrodoKEM, NewHope, and qTesla for the second round submission in NIST PQC competition, which was extended to FPGA platform in [21]. The work [22] proposed the integration of instruction sets for finite field arithmetic operations in a RISC-V processor, supporting PQC algorithms including KyberKEM and NewHope. The work [23] integrated the vectorized modular arithmetic operations and NTT computation in a RISC-V processor and presents the ASIC and FPGA implementation result, supporting PQC algorithms including KyberKEM, Saber and NewHope.

We also consider the literature improving the polynomial multiplication in hardware. Several works focusing on the implementation of the NTT computation has been published in [10], [24], [25], [26], [27], [28]. Zhang *et al.* [10] proposed a low-complexity NTT/INTT algorithm, absorbing the pre-process and post-process into NTT and INTT, respectively. In [24], a parallel architecture is proposed for high-speed NTT design. The works [25], [26], [27], [28] proposed NTT-based polynomial multiplication architectures for KyberKEM on FPGA.

1.2 Our Contributions

This paper proposes a highly efficient hardware design of KyberKEM by using the split-radix DGT to replace the NTT in general design. In particular, the main contributions are as follows:

1) We propose a novel negative wrapped convolution (NWC) via split-radix DGT algorithm, including split-radix DGT, split-radix IDGT and component-wise multiplication (CWM) by applying the split radix nature into DGT and IDGT [29]. This method could reduce the transform length into half and requires fewer butterfly operations and transform stages compared to the NWC via NTT. Besides, this method saves 10% modular multiplication operations for length-128 polynomial compared with the NTT algorithm in [10].
2) We propose a dedicated stream permutation network and the corresponding scheduling plan for the split-radix DGT to squeeze out the scheduling bubbles and hence enable a fully pipelined working mode. The proposed stream permutation network could reduce 25% clock cycles compared to the state-of-the-art NTT in KyberKEM scheduling plan in [9].

TABLE 1
Mathematical Notations

Notation	Definition
Regular lower-case letter (a)	Element in $\mathbb{Z}_q$
Lower-case letter with bar ($\bar{a}$)	Element in $GF(q^2)$
Upper-case letter ($A/\hat{A}$)	Polynomial in $\mathbb{Z}_q$ in normal/transform domain
Upper-case letter with bar ($\bar{A}/\hat{\bar{A}}$)	Polynomial in $GF(q^2)$ in normal/transform domain
Bold lower-case letter ($\mathbf{a}/\hat{\mathbf{a}}$)	Polynomial vector in normal/transform domain
Bold upper-case letter ($\mathbf{A}/\hat{\mathbf{A}}$)	Polynomial matrix in normal/transform domain
Sans-serif upper-case letter (NTT)	Help function
A_i	The i-th entity in A
$\mathbf{a}^T/\mathbf{A}^T$	Transpose of vector/matrix
$\hat{A} \circ \hat{B}/\mathbf{a} \circ \mathbf{b}/\mathbf{a} \circ \mathbf{B}$	Component-wise multiplication

3) We propose a compact and unified butterfly unit (BFU) supporting split-radix DGT, IDGT and CWM, along with efficient multiplier over $GF(q^2)$ to achieve a high-throughput NWC. The unified BFU could compute DGT/IDGT/CWM in KyberKEM in 384/384/132 cycles with one BFU setting, respectively. The proposed architecture could achieve at least 49.4% faster transforming, and 65.3% faster CWM with 87% LUT-NTT area-time product (ATP) and 32% LUT-CWM ATP compared with the results of the state-of-the-art designs of NWC in KyberKEM with one BFU setting on similar platforms.
4) We propose a highly efficient KyberKEM - Split Radix DGT architecture. The optimization technique, i.e., bandwidth matching, as well as *just-in-time* principle, is used to design a optimized and constant time architecture for KyberKEM, which significantly improves the ATP and outperforms the state-of-the-arts.

2 Preliminary

Table 1 provides the mathematical notations used in this paper. We use $\mathcal{R}_q$ to denote the polynomial ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ defined over the field $\mathbb{Z}_q$, where q is a prime integer.

2.1 CRYSTALS-KyberKEM

CRYSTALS-KyberKEM [2] is a key-encapsulation mechanism with Adaptive Chosen Ciphertext Attack (IND-CCA2) security. The security of KyberKEM is based on the hardness of the learning-with-errors problem in module lattices (i.e. MLWE problem [30]). To construct a IND-CCA2-secure KEM, CRYSTALS-Kyber uses the slightly tweaked Fujisaki–Okamoto (FO) transform [2], [31] to transfer a Chosen Plaintext Attack (IND-CPA) secure Public-Key Encryption (PKE) scheme, which is called as CRYSTALS-KyberPKE. The parameter sets for CRYSTALS-KyberKEM is shown in Table 2. The key generation, encryption, and decryption of the CRYSTALS-KyberPKE are defined as follows, with following the definition of the help functions CBD, Parse, Compress, NTT, and INTT in [2]:

- KeyGen($\cdot$): Key generation samples $\mathbf{s}$ and $\mathbf{e}$ from centered binomial distribution (CBD), and $\hat{\mathbf{A}}$ from uniform distribution (Parse). The public key $pk = (\rho, \hat{\mathbf{t}})$ and secret key $sk = \hat{\mathbf{s}}$ are returned where ρ is the random seed and $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$.

TABLE 2
Parameter sets for CRYSTALS-KyberKEM in the Third Round Submission [2]

Algorithm	NIST Level	Parameter set				
		n	k	q	(η_1, η_2)	(d_u, d_v)
Kyber-512-CCA	1	256	2	3,329	(3,2)	(10,4)
Kyber-768-CCA	3	256	3	3,329	(2,2)	(10,4)
Kyber-1024-CCA	5	256	4	3,329	(2,2)	(11,5)

- Enc(pk, M): Encryption samples $\mathbf{r}$, $\mathbf{e}_1$ and E_2 from CBD, and $\hat{\mathbf{A}}$ from Parse. The ciphertext $ct = (\mathsf{Compress}(\mathbf{u}), \mathsf{Compress}(V))$ is returned where $\mathbf{u} = \mathsf{INTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ and $V = \mathsf{INTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + E_2 + M$.
- Dec(sk, ct): Decryption returns the recovered message $M = \mathsf{Compress}(V - \mathsf{INTT}(\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}))$ where $\mathbf{u}$ and V are decompressed from ct.

2.2 NTT and Inverse NTT (INTT)

NTT is a variant of DFT by changing the complex number field into finite field $\mathbb{Z}_q$. Given a polynomial of length n, the length-n NTT (noted as NTT_n) is defined as $\hat{A}_j = \mathsf{NTT}_n(A)_j = \sum_{i=0}^{n-1} A_i \omega_n^{ij} \mod q$, where $0 \le j < n$, $\omega_n \pmod q$ denotes the primitive n-th root of unit over $\mathbb{Z}_q$ or twiddle factor of length-n NTT. $\omega_n \pmod q$ exists when $q \equiv 1 \pmod n$.

The inverse NTT (INTT) could be performed by replacing the twiddle factor of length-n NTT $\omega_n \pmod q$ by $\omega_n^{-1} \pmod q$, and multiplying the scalar factor $n^{-1} \pmod q$ after the summation. The length-n INTT (noted as INTT_n) is defined as $A_i = \mathsf{INTT}_n(\hat{A})_i = n^{-1} \sum_{j=0}^{n-1} \hat{A}_j \omega_n^{-ij} \mod q$, where $0 \le i < m$.

2.3 Polynomial Multiplication via NTT

According to [32], polynomial multiplication over $\mathcal{R}_q$ could be solved efficiently by negative wrapped convolution (NWC) when the prime parameter q satisfies $2n|(q-1)$. NWC would introduce pre-processing before NTT and post-processing after INTT. In order to reduce the computing complexity of NWC, Zhang *et al.* [10] proposed low-complexity NTT and INTT algorithms (noted as LC NTT/INTT) by merging the pre and post processing into NTT and INTT without additional modular multiplication. Based on [10], the number of modular multiplications in the LC NTT/INTT algorithms is $\frac{n}{2} log_2 n$.

Starting from the second-round submission of Crystals-KyberKEM, the parameter set (n, q) is selected as $(256, 3329)$ as shown in Table 2. Given that $n \mid (q-1)$ but $2n \nmid (q-1)$, the aforementioned NWC via NTT could not be applied directly. A variant of NTT proposed by [33] is adopted to apply the NWC in Crystals-Kyber KEM. Such a variant is based on the observation that, when polynomial F could be factored into a product $F = GH$ over the finite field $\mathbb{Z}_q$, we have an isomorphism by the Chinese remainder theorem:

$$\mathbb{Z}_q[x]/(F) \cong \mathbb{Z}_q[x]/(G) \times \mathbb{Z}_q[x]/(H). \tag{1}$$

Since $x^{256}+1 = \prod_{i=0}^{127}(x^2 - \omega_{256}^{2i+1})$ given the primitive 256-th roots of unity ω_{256}, the definition of NTT working on $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle$ (noted as $\mathsf{NTT}_{256}^{3329}$) is given by:

$$\begin{aligned} \mathsf{NTT}_{256}^{3329}(A)_i &= A(x) \bmod (x^2 - \omega_{256}^{2i+1}) \\ &= x \sum_{j=0}^{127} A_{2j+1} \omega_{256}^{(2i+1)j} + \sum_{j=0}^{127} A_{2j} \omega_{256}^{(2i+1)j} \\ &= x\hat{A}_{2i+1} + \hat{A}_{2i}, \end{aligned} \tag{2}$$

where $0 \le i < 128$ and $\hat{A}_{2i+1} = \sum_{j=0}^{127} A_{2j+1} \omega_{256}^{(2i+1)j}$, and $\hat{A}_{2i} = \sum_{j=0}^{127} A_{2j} \omega_{256}^{(2i+1)j}$. Now both $\hat{A}_{2i+1}$ and $\hat{A}_{2i}$ could be solved by the length-128 low-complexity NTT. As for the inverse transform, we could use two length-128 low-complexity INTTs to reconstruct $A(x)$ from $\hat{A}_{2i+1}$ and $\hat{A}_{2i}$. The NWC working on $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle$ could also be solved as

$$\mathsf{INTT}_{256}^{3329}(\mathsf{NTT}_{256}^{3329}(\mathbf{a}) \circ \mathsf{NTT}_{256}^{3329}(\mathbf{b})). \tag{3}$$

Such component-wise multiplication is defined as:

$$\begin{aligned} &A(x) \circ B(x) \bmod (x^2 - \omega_{256}^{2i+1}) \\ &= (x\hat{A}_{2i+1} + \hat{A}_{2i})(x\hat{B}_{2i+1} + \hat{B}_{2i}) \\ &\bmod (x^2 - \omega_{256}^{2i+1}), \end{aligned} \tag{4}$$

where $0 \le i < 128$. Thus, polynomial multiplication over $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle$ is performed by four length-128 low-complexity NTTs, one length-128 component-wise multiplication, and two length-128 low-complexity INTTs.

2.4 Negative Wrapped Convolution via DGT

Consider $A(x) \in \mathcal{R}_q$. Let $m = \frac{n}{2}$ and $z = x^m = \sqrt{x^n} \equiv \sqrt{-1} \pmod q$. Then, we rewrite $A(x)$ as:

$$\begin{aligned} &A(x) = A_{n-1}x^{n-1} + A_{n-2}x^{n-2} + \cdots + A_1 x^1 + A_0 \\ =&(A_{n-1}x^m + A_{m-1})x^{m-1} + \cdots + (A_m x^m + A_0) \\ =&(A_{n-1}z + A_{m-1})x^{m-1} + \cdots + (A_m z + A_0) \\ =&\bar{A}_{m-1}x^{m-1} + \cdots + \bar{A}_1 x^1 + \bar{A}_0, \end{aligned} \tag{5}$$

where $\bar{A}_i = (A_{i+m}z + A_i)$, $0 \le i < m$. It is notable that the $\bar{A}_i \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$, which is isomorphic to $GF(q^2)$. Given $0 \le i, j < m$, some arithmetic operations over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ are defined as:

$$\begin{aligned} &\text{Addition: } \bar{A}_i + \bar{A}_j = (A_{i+m} + A_{j+m})z + (A_i + A_j); \\ &\text{Multiplication: } \bar{A}_i \circ \bar{A}_j = (A_i A_j - A_{i+m}A_{j+m})z + \\ &(A_i A_{j+m} + A_{i+m}A_j). \end{aligned} \tag{6}$$

We could define the $\zeta_m \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$, such that $\zeta_m^m = z$. Such ζ_m exists when $4m|(q-1)$ [29]. We observed that $\{\zeta_m^{4i+1}, \forall 0 \le i < m\}$ would be a set of solutions of the equation $x^m = z$ on $\mathbb{Z}_q[z]/\langle z^2+1\rangle$, indicating $\{\zeta_m^{4i+1}, \forall 0 \le i < m\}$ fulfills the following properties:

$$\begin{aligned} &\text{Symmetry: } \zeta_m^{4(i+\frac{m}{2})+1} = \zeta_m^{4i+1}\zeta_m^{2m} = (-1)\zeta_m^{4i+1}; \\ &\text{Periodicity: } \zeta_m^{4(i+m)+1} = \zeta_m^{4i+1}\zeta_m^{4m} = \zeta_m^{4i+1}; \\ &\text{Scalability: } \zeta_{m/k}^{4(i/k)+1} = \zeta_{m/k}^{4(i/k)}\zeta_{m/k}^{1} = \zeta_m^{4i+k}; \\ &\text{Semi-symmetry: } \zeta_m^{4(i+\frac{m}{4})+1} = \zeta_m^{4i+1}\zeta_m^{m} = \zeta_m^{4i+1}z; \end{aligned} \tag{7}$$

where k is a power-of-two integer that is smaller than m. Thus, we define the set of twiddle factors in Discrete Galois Transform (DGT) as $\{\zeta_m^{4i+1}, \forall 0 \le i < m\}$. For a length-$m$

polynomial $\bar{A}$, whose entities $\bar{A}_i \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$, the definition of length-m DGT (noted as DGT_m) would be $\hat{\bar{A}}_j = \mathsf{DGT}_m(\bar{A})_j = \sum_{i=0}^{m-1}(\bar{A}_i\zeta_m^i)\zeta_m^{4ji}$, where $0 \le j < m$. Similarly, the definition of length-m IDGT (noted as IDGT_m) would be $\mathsf{IDGT}_m(\hat{\bar{A}})_i = m^{-1}\zeta_m^{-i}\sum_{j=0}^{m-1}(\hat{\bar{A}}_j\zeta_m^{-4ji})$, where $0 \le i < m$. According to [29], one can perform the DGT_m and IDGT_m algorithms similar to the classic NTT and INTT, by replacing the arithmetic operations in $\mathbb{Z}_q$ with arithmetic operations in $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ defined in Equation 6, which means $\frac{m}{2}log_2m + m$ and $\frac{m}{2}log_2m + 2m$ multiplications in $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ are needed in DGT_m and IDGT_m, respectively [10]. Recall that the addition in $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ involves no modular multiplication while each multiplication in $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ includes three modular multiplications using the Karatsuba method. The number of modular multiplication in DGT_m and IDGT_m would be $3\frac{m}{2}log_2m + 3m$ and $3\frac{m}{2}log_2m + 6m$, respectively.

According to [29], the length-n NWC could also be solved via DGT as

$$\mathsf{IDGT}_m(\mathsf{DGT}_m(\bar{A}) \circ \mathsf{DGT}_m(\bar{B})), \tag{8}$$

where $m = \frac{n}{2}$. Thus, length-n NWC could be performed as two length-m DGTs after pre-processing, one length-m point-wise multiplication, and one length-m IDGT following by post-processing.

3 Proposed Split-Radix DGT and IDGT

In this section, we propose to integrate the split radix and decimation-in-time (DIT) into the low-complexity DGT algorithm, while using split radix and decimation-in-frequency (DIF) to derive IDGT. These novel split-radix DGT/IDGT algorithms inherit the advantages of small multiplication number from split radix nature and the short transformation length from DGT/IDGT, which enable low complexity NWCs.

3.1 Proposed Split Radix DGT

The low-complexity DGT is derived in the split radix [34] and decimation-in-time (DIT) [35] in this subsection. Given a length-m polynomial $\bar{A}$, whose entities $\bar{A}_i \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$. We would start the derivation by splitting the summation of DGT into three groups according to the index of $\hat{\bar{A}}$ as follows:

$$\begin{aligned}\hat{\bar{A}}_j &= \sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+1}(\zeta_m^{4j+1})^{4i+1} \\ &+ \sum_{i=0}^{\frac{m}{2}-1}\bar{A}_{2i}(\zeta_m^{4j+1})^{2i} + \sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+3}(\zeta_m^{4j+1})^{4i+3} \\ &= \zeta_m^{4j+1}\sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+1}(\zeta_{m/4}^{4j+1})^i + \sum_{i=0}^{\frac{m}{2}-1}\bar{A}_{2i}(\zeta_{m/2}^{4j+1})^i \\ &+ \zeta_m^{3(4j+1)}\sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+3}(\zeta_{m/4}^{4j+1})^i,\end{aligned} \tag{9}$$

where $0 \le j < m$. We could decompose the degree-m DGT into two degree-$\frac{m}{4}$ DGTs and one degree-$\frac{m}{2}$ DGT. Namely, we set $\sum_{i=0}^{\frac{m}{2}-1}\bar{A}_{2i}(\zeta_{m/2}^{4j+1})^i$ as $\hat{\bar{W}}_j$, $\sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+1}(\zeta_{m/4}^{4j+1})^i$ as $\hat{\bar{X}}_j$, and $\sum_{i=0}^{\frac{m}{4}-1}\bar{A}_{4i+3}(\zeta_{m/4}^{4j+1})^i$ as $\hat{\bar{Y}}_j$, then

$$\hat{\bar{A}}_j = \hat{\bar{W}}_j + \zeta_m^{4j+1}\hat{\bar{X}}_j + \zeta_m^{3(4j+1)}\hat{\bar{Y}}_j. \tag{10}$$

For $0 \le j < \frac{m}{4}$, we rewrite Equation 10 in terms of $\hat{\bar{W}}_j$, $\hat{\bar{W}}_{j+\frac{m}{4}}$, $\hat{\bar{X}}_j$, and $\hat{\bar{Y}}_j$ as

$$\begin{aligned}\hat{\bar{A}}_j &= \hat{\bar{W}}_j + (\zeta_m^{4j+1}\hat{\bar{X}}_j + \zeta_m^{3(4j+1)}\hat{\bar{Y}}_j), \\ \hat{\bar{A}}_{j+\frac{m}{4}} &= \hat{\bar{W}}_{j+\frac{m}{4}} + z(\zeta_m^{4j+1}\hat{\bar{X}}_j - \zeta_m^{3(4j+1)}\hat{\bar{Y}}_j), \\ \hat{\bar{A}}_{j+\frac{m}{2}} &= \hat{\bar{W}}_j - (\zeta_m^{4j+1}\hat{\bar{X}}_j + \zeta_m^{3(4j+1)}\hat{\bar{Y}}_j), \\ \hat{\bar{A}}_{j+\frac{3m}{4}} &= \hat{\bar{W}}_{j+\frac{m}{4}} - z(\zeta_m^{4j+1}\hat{\bar{X}}_j - \zeta_m^{3(4j+1)}\hat{\bar{Y}}_j).\end{aligned} \tag{11}$$

The Equation 11 represents the asymmetric DIT butterfly computation for split-radix DGT. We should note there are two boundary cases at $m = 2$ and $m = 4$. When $m = 2$, the DGT problem $\hat{\bar{A}}_j$ would be solved as

$$\begin{aligned}\hat{\bar{A}}_0 &= \bar{A}_0 + \bar{A}_1\zeta_2, \\ \hat{\bar{A}}_1 &= \bar{A}_0 + \bar{A}_1\zeta_2^5 = \bar{A}_0 - \bar{A}_1\zeta_2.\end{aligned} \tag{12}$$

And when $m = 4$, the DGT problem $\hat{\bar{A}}_j$ is solved as

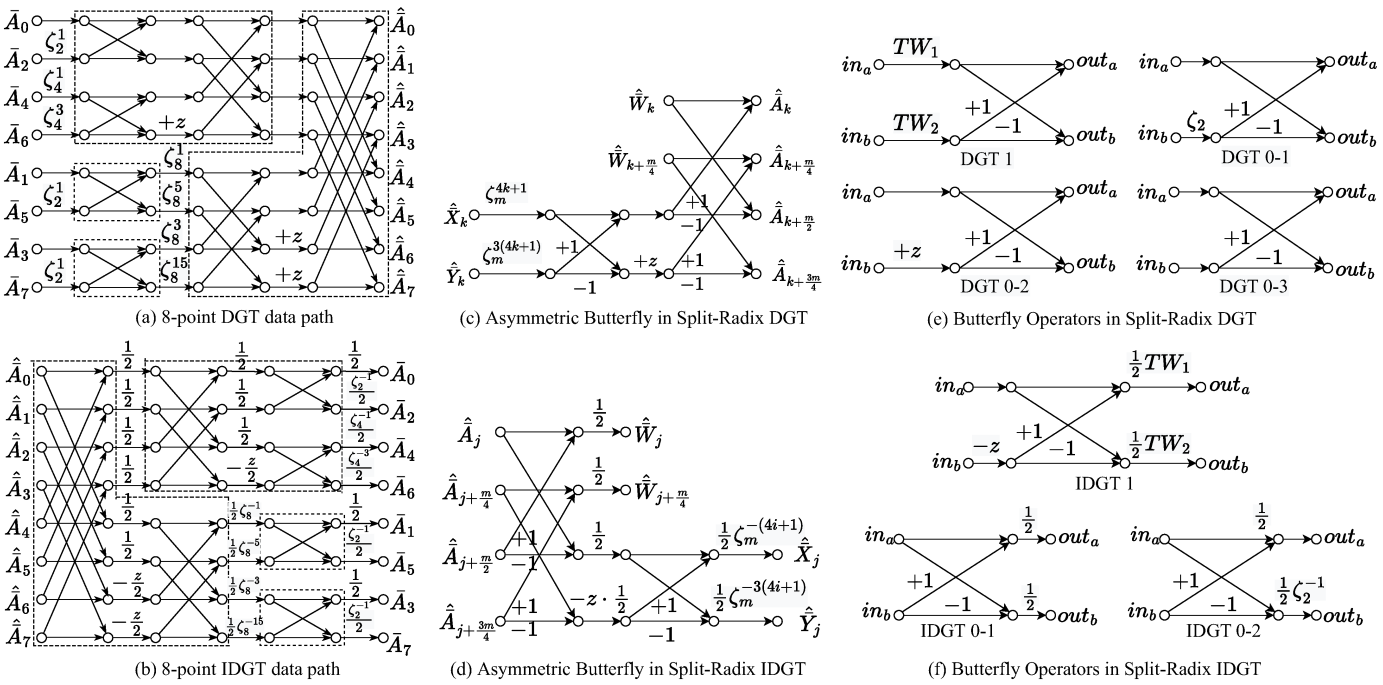
$$\begin{aligned}\hat{\bar{A}}_0 &= (\bar{A}_0 + \bar{A}_2\zeta_4^2) + (\bar{A}_1\zeta_4 + \bar{A}_3\zeta_4^3), \\ \hat{\bar{A}}_1 &= (\bar{A}_0 - \bar{A}_2\zeta_4^2) + z(\bar{A}_1\zeta_4 - \bar{A}_3\zeta_4^3), \\ \hat{\bar{A}}_2 &= (\bar{A}_0 + \bar{A}_2\zeta_4^2) - (\bar{A}_1\zeta_4 + \bar{A}_3\zeta_4^3), \\ \hat{\bar{A}}_3 &= (\bar{A}_0 - \bar{A}_2\zeta_4^2) - z(\bar{A}_1\zeta_4 - \bar{A}_3\zeta_4^3).\end{aligned} \tag{13}$$

Figure 1 shows the data flow and the butterfly of a 8-point split-radix DGT. The details of the proposed split-radix DIT DGT are shown in Algorithm 1. We observed the split-radix DGT butterfly in Equation 11 is asymmetric, and different butterflies would be processed at boundary cases $m = 2, 4$. It is recommended to decompose the asymmetric butterfly operations as well as the butterfly operations in boundary cases into the similar butterfly operators. In the proposed algorithm, four butterfly operators shown in Figure 1 would be applied, namely *DGT 1*, *DGT 0-1*, *DGT 0-2*, and *DGT 0-3*. Additionally, the order sequence of each operators could be pre-computed and stored into an integer SEQ. We also proposed the method to generate the SEQ as well as the corresponding control logic to select the target operator, as shown in Algorithm 1. The help function $\mathsf{br}_l(i)$ can generate the bit reversal of integer i ranging from 0 to (2^l-1). For example, $\mathsf{br}_4(1011_b) = 1101_b$. The help function $\mathsf{scramble}_l(A)$ permutes the length-2^l polynomial A, moving the i-th term to index $\mathsf{br}_l(i)$.

3.2 Proposed Split Radix IDGT

The low complexity IDGT is derived in the split radix [34] and decimation-in-frequency (DIF) nature [35]. Given a length-m polynomial $\hat{\bar{A}}$, one have $\bar{A} = \mathsf{IDGT}_m(\hat{\bar{A}})$. We define $\bar{W}_i = \bar{A}_{2i}$ for $0 \le i < \frac{m}{2}$, $\bar{X}_i = \bar{A}_{4i+1}$ and $\bar{Y}_i = \bar{A}_{4i+3}$ for $0 \le i < \frac{m}{4}$. We would start the derivation of $\bar{W}_i$ by splitting the summation of IDGT into two groups according to the index of $\hat{\bar{A}}$ as follows:

$$\bar{A}_i = m^{-1}\zeta_m^{-i}\sum_{j=0}^{\frac{m}{2}-1}\left(\hat{\bar{A}}_j + \hat{\bar{A}}_{j+\frac{m}{2}}\zeta_m^{-4i\frac{m}{2}}\right)\zeta_m^{-4ji}, \tag{14}$$



(a) 8-point DGT data path (b) 8-point IDGT data path (c) Asymmetric Butterfly in Split-Radix DGT (d) Asymmetric Butterfly in Split-Radix IDGT (e) Butterfly Operators in Split-Radix DGT (f) Butterfly Operators in Split-Radix IDGT

Fig. 1. Dataflow and the proposed butterfly operators of low-complexity split-radix DGT and IDGT. TW_1 and TW_2 are twiddle factors described in Algorithm 1 and 2

For $0 \le i < \frac{m}{2}$, we substitute Equation 14 into $\bar{W}_i = \bar{A}_{2i}$, and apply the scalability property and substituting $(\zeta_m^{4m}) \equiv 1$ on $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ as follows:

$$\begin{aligned}\bar{W}_i &= m^{-1}\zeta_m^{-2i}\sum_{j=0}^{\frac{m}{2}-1}\left(\hat{\bar{A}}_j + \hat{\bar{A}}_{j+\frac{m}{2}}\zeta_m^{-4mi}\right)\zeta_m^{2\times(-4ij)} \\ &= (\frac{m}{2})^{-1}\zeta_{\frac{m}{2}}^{-i}\sum_{j=0}^{\frac{m}{2}-1}\left(\frac{\hat{\bar{A}}_j + \hat{\bar{A}}_{j+\frac{m}{2}}}{2}\right)\zeta_{\frac{m}{2}}^{-4ij} \\ &= (\frac{m}{2})^{-1}\zeta_{\frac{m}{2}}^{-i}\sum_{j=0}^{\frac{m}{2}-1}\hat{\bar{W}}_j\zeta_{\frac{m}{2}}^{-4ij},\end{aligned} \tag{15}$$

with defining $\hat{\bar{W}}_j = \frac{\hat{\bar{A}}_j+\hat{\bar{A}}_{j+\frac{m}{2}}}{2}$. We find that Equation 15 is equivalent to the length-$\frac{m}{2}$ IDGT of $\hat{\bar{w}}_j$. Thus, the subproblem of length $\frac{m}{2}$ is constructed.

To construct the other two subproblems of length-$\frac{m}{4}$, namely $\bar{X}_i$ and $\bar{Y}_i$ for $0 \le i < \frac{m}{4}$, again we start the derivation from the definition of IDGT with post-processing, but splitting the summation to four groups according to the index of $\hat{\bar{A}}$. For $0 \le i < m$,

$$\begin{aligned}\bar{A}_i = m^{-1}\zeta_m^{-i}\sum_{j=0}^{\frac{m}{4}-1}&\Big(\hat{\bar{A}}_j + \hat{\bar{A}}_{j+\frac{m}{4}}\zeta_m^{-4i\frac{m}{4}} \\ &+\hat{\bar{A}}_{j+\frac{m}{2}}\zeta_m^{-4i\frac{m}{2}} + \hat{\bar{A}}_{j+\frac{3m}{4}}\zeta_m^{-4i\frac{3m}{4}}\Big)\zeta_m^{-4ji}.\end{aligned} \tag{16}$$

For $0 \le i < \frac{m}{4}$, we substitute Equation 16 into $\bar{X}_i = \bar{A}_{4i+1}$, and apply the scalability property and substituting $(\zeta_m^{4m}) \equiv 1$ on $\mathbb{Z}_q[z]/\langle z^2+1\rangle$, that is:

$$\begin{aligned}\bar{X}_i = \bar{A}_{4i+1} = (\frac{m}{4})^{-1}\zeta_{\frac{m}{4}}^{-i}\sum_{j=0}^{\frac{m}{4}-1}&\left[\frac{1}{2}(\hat{\bar{A}}_j - \hat{\bar{A}}_{j+\frac{m}{2}})\right. \\ &\left.+\frac{-z}{2}(\hat{\bar{A}}_{j+\frac{m}{4}} - \hat{\bar{A}}_{j+\frac{3m}{4}})\right]\frac{\zeta_m^{-(4j+1)}}{2}\zeta_{\frac{m}{4}}^{-4ji}.\end{aligned} \tag{17}$$

Defining that

$$\hat{\bar{X}}_j = \left[\frac{1}{2}(\hat{\bar{A}}_j - \hat{\bar{A}}_{j+\frac{m}{2}}) + \frac{-z}{2}(\hat{\bar{A}}_{j+\frac{m}{4}} - \hat{\bar{A}}_{j+\frac{3m}{4}})\right]\frac{\zeta_m^{-(4j+1)}}{2}, \tag{18}$$

one could simplify Equation 17 as

$$\bar{X}_i = (\frac{m}{4})^{-1}\zeta_{\frac{m}{4}}^{-i}\sum_{j=0}^{\frac{m}{4}-1}\hat{\bar{X}}_j\zeta_{\frac{m}{4}}^{-4ji}, \tag{19}$$

We find that Equation 19 are equivalent to the length-$\frac{m}{4}$ IDGT of $\hat{\bar{X}}$. Thus, the subproblem of length $\frac{m}{4}$ is constructed. The subproblem $\bar{Y}_i = \bar{A}_{4i+3}$ for $0 \le i < \frac{m}{4}$ could also be constructed similar to Equation 17-19. In summary, the split-radix DIF IDGT butterfly operations are defined as

$$\begin{aligned}\hat{\bar{W}}_j &= \frac{1}{2}(\hat{\bar{A}}_j + \hat{\bar{A}}_{j+\frac{m}{2}}), \\ \hat{\bar{W}}_{j+\frac{m}{4}} &= \frac{1}{2}(\hat{\bar{A}}_{j+\frac{m}{4}} + \hat{\bar{A}}_{j+\frac{3m}{4}}), \\ \hat{\bar{X}}_j &= \left[\frac{1}{2}(\hat{\bar{A}}_j - \hat{\bar{A}}_{j+\frac{m}{2}}) + \frac{-z}{2}(\hat{\bar{A}}_{j+\frac{m}{4}} - \hat{\bar{A}}_{j+\frac{3m}{4}})\right]\frac{\zeta_m^{-(4j+1)}}{2}, \\ \hat{\bar{Y}}_j &= \left[\frac{1}{2}(\hat{\bar{A}}_j - \hat{\bar{A}}_{j+\frac{m}{2}}) - \frac{-z}{2}(\hat{\bar{A}}_{j+\frac{m}{4}} - \hat{\bar{A}}_{j+\frac{3m}{4}})\right]\frac{\zeta_m^{-3(4j+1)}}{2},\end{aligned} \tag{20}$$

Algorithm 1 Iterative Split-Radix DIT DGT Algorithm without Pre-processing

Input: Given the parameter set $n, q, m = n/2$, $\zeta_m \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$. Let $\bar{A}$ be a folded vector of length m composed of the original polynomial (of length n) coefficients. Elements of $\bar{A} = (\bar{A}_0, \bar{A}_1, ..., \bar{A}_{m-1})$.

Output: $\hat{\bar{A}} = \mathsf{DGT}_m(\bar{A}) = (\hat{\bar{A}}_0, \hat{\bar{A}}_1, ..., \hat{\bar{A}}_{m-1})$.

$\hat{\bar{A}} := \bar{A}$
$SEQ := 0$ ▷ SEQ could be pre-computed
for $i = 0$ to $log_2 m - 1$ **do**
  $SEQ := SEQ + (-1)^i (2^{2^{log m - i - 2}} - 1)$
end for
for $s = 0$ to $log_2 m - 1$ **do**
  $l := 2^s$
  $d := m/(2l)$
  for $i = 0$ to $l - 1$ **do**
    $TW_1 := \zeta_m^{[\mathsf{4br}_{log_2 m}(i 2^{log_2 m - s}) + 1]\frac{d}{2}}$
    $TW_2 := \zeta_m^{[\mathsf{4br}_{log_2 m}(i 2^{log_2 m - s}) + 1]\frac{3d}{2}}$
    for $j = 0$ to $d - 1$ **do**
      if $SEQ_{[(d-j) \ll s]} = 1$ **then**
        ▷ Check the $((d-j) \ll s)$-th bit of SEQ
        $u := (TW_1)\hat{\bar{A}}_{2id+j}$
        $t := (TW_2)\hat{\bar{A}}_{2id+j+d}$ ▷ *DGT 1*
      else if $s = 0$ **then**
        $u := \hat{\bar{A}}_{2id+j}$
        $t := (\zeta_m^{m/2})\hat{\bar{A}}_{2id+j+d}$ ▷ *DGT 0-1*
      else if $i \bmod 2 = 1$ **then**
        $u := \hat{\bar{A}}_{2id+j}$
        $t := (+z)\hat{\bar{A}}_{2id+j+d}$ ▷ *DGT 0-2*
      else
        $u := \hat{\bar{A}}_{2id+j}$
        $t := \hat{\bar{A}}_{2id+j+d}$ ▷ *DGT 0-3*
      end if
      $\hat{\bar{A}}_{2id+j} := u + t; \hat{\bar{A}}_{2id+j+d} := u - t$
    end for
  end for
end for
$\hat{\bar{A}} := \mathsf{scramble}_{log_2 m}(\hat{\bar{A}})$
return $\hat{\bar{A}}$

We should also note the two boundary cases at $m = 2, 4$. When $m = 2$, the IDGT problem $\bar{A}_i$ would be solved as

$$\begin{aligned} \bar{A}_0 &= \frac{1}{2}(\hat{\bar{A}}_0 + \hat{\bar{A}}_1) \\ \bar{A}_1 &= \frac{1}{2}(\hat{\bar{A}}_0 + \hat{\bar{A}}_1 \zeta_2^{-4})\zeta_2^{-1} = \frac{1}{2}(\hat{\bar{A}}_0 - \hat{\bar{A}}_1)\zeta_2^{-1}. \end{aligned} \tag{21}$$

And when $m = 4$, the IDGT problem is solved as

$$\begin{aligned} \bar{A}_0 &= \frac{1}{2}(\frac{\hat{\bar{A}}_0 + \hat{\bar{A}}_2}{2} + \frac{\hat{\bar{A}}_1 + \hat{\bar{A}}_3}{2}), \\ \bar{A}_1 &= \frac{\zeta_4^{-1}}{2}(\frac{\hat{\bar{A}}_0 - \hat{\bar{A}}_2}{2} - z\frac{\hat{\bar{A}}_1 - \hat{\bar{A}}_3}{2}), \\ \bar{A}_2 &= \frac{\zeta_4^{-2}}{2}(\frac{\hat{\bar{A}}_0 + \hat{\bar{A}}_2}{2} - \frac{\hat{\bar{A}}_1 + \hat{\bar{A}}_3}{2}), \\ \bar{A}_3 &= \frac{\zeta_4^{-3}}{2}(\frac{\hat{\bar{A}}_0 - \hat{\bar{A}}_2}{2} + z\frac{\hat{\bar{A}}_1 - \hat{\bar{A}}_3}{2}). \end{aligned} \tag{22}$$

Algorithm 2 Iterative Split-Radix DIF IDGT Algorithm without Post-processing

Input: Given the parameter set $n, q, m = n/2$, $\zeta_m \in \mathbb{Z}_q[z]/\langle z^2+1\rangle$. Let $\hat{\bar{A}}$ be a folded vector of length m composed of the original polynomial (of length n) coefficients. Elements of $\hat{\bar{A}} = (\hat{\bar{A}}_0, \hat{\bar{A}}_1, ..., \hat{\bar{A}}_{m-1})$.

Output: $\bar{A} = \mathsf{IDGT}_m(\hat{\bar{A}}) = (\bar{A}_0, \bar{A}_1, ..., \bar{A}_{m-1})$.

$\bar{A} := \mathsf{scramble}_{log_2 m}(\hat{\bar{A}})$
$SEQ := 0$ ▷ SEQ could be pre-computed
for $i = 0$ to $log_2 m - 1$ **do**
  $SEQ := SEQ + (-1)^i (2^{2^{log m - i - 2}} - 1)$
end for
for $s = log_2 m - 1$ to 0 **do**
  $l := 2^s$
  $d := m/(2l)$
  for $i = 0$ to $d - 1$ **do**
    for $j = i$ to $m - 1$ step $2d$ **do**
      $TW_1 := \zeta_m^{(-1)[\mathsf{4br}_{log_2 m}(j 2^{log_2 m - 1 - s}/d) + 1]\frac{d}{2}}$
      $TW_2 := \zeta_m^{(-1)[\mathsf{4br}_{log_2 m}(j 2^{log_2 m - 1 - s}/d) + 1]\frac{3d}{2}}$
      if $SEQ_{[(d-i) \ll s]} = 0$ **then**
        ▷ Check the $((d-j) \ll s)$-th bit of SEQ
        $u := \bar{A}_j + \bar{A}_{j+d}$
        $t := \bar{A}_j - \bar{A}_{j+d}$
        if $s = 0$ **then** ▷ *IDGT 0-2*
          $\bar{A}_j := \frac{u}{2}; \bar{A}_{j+d} := \zeta_m^{m/2} \cdot \frac{t}{2}$
        else ▷ *IDGT 0-1*
          $\bar{A}_j := \frac{u}{2}; \bar{A}_{j+d} := \frac{t}{2}$
        end if
      else ▷ *IDGT 1*
        $u := \bar{A}_j + (-z) \cdot \bar{A}_{j+d}$
        $t := \bar{A}_j - (-z) \cdot \bar{A}_{j+d}$
        $\bar{A}_j := TW_1 \cdot \frac{u}{2}; \bar{A}_{j+d} := TW_2 \cdot \frac{t}{2}$
      end if
    end for
  end for
end for
return $\bar{A}$

Figure 1 shows the data flow and the butterfly of a 8-point IDGT. The details of the split-radix DIF IDGT are shown in Algorithm 2. Help functions $\mathsf{scramble}_l$ and br_l are defined in Section 3.1. Similar to DGT, we also need to decompose the butterfly operations as well as the boundary cases $m = 2, 4$ into multiple computing operators. In the proposed IDGT algorithm, three butterfly operators shown in Figure 1 would be applied, namely *DIF 1*, *DIF 0-1*, *DIF 0-2*. We observed that the proposed DGT and IDGT could share the pre-computed integer(*SEQ*), thus the memory overhead for operator selection would be reduced.

3.3 Complexity Analysis on Split radix DGT/IDGT

To analyze the computation cost of split-radix DIT DGT, one could set up the recurrent equations based on the asymmetric split-radix butterflies and the two boundary cases. The number of modular multiplication and modular addition in a length-m DGT is defined as $M(m)$ and $A(m)$, respectively. Given the size of each sub-problems in Equation 11 is $m/4$, one could find that $m/2$ additions over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ and

$m/2$ multiplications over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ are needed in the first stage of split-radix DIT DGT butterfly computation. The second stage of the DGT butterfly computation involved m additions over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ but no multiplication. In summary, $3m/2$ additions over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ and $m/2$ multiplications over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ are required to compute the length-$m/4$ sub-problem of split-radix DIT DGT. Recall that each addition over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ is separated into 2 modular additions, and each multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ involves 5 modular additions and 3 modular multiplications when using Karatsuba algorithm. Accordingly, $11m/2$ modular additions and $3m/2$ modular multiplications in $GF(q)$ are required for the length-$m/4$ sub-problem of split-radix DIT DGT. Recall that when $m=2$, the DGT problem consists of 2 additions over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ and 1 multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ as shown in Equation 12. Accordingly, 9 modular additions and 3 modular multiplications in $GF(q)$ are required when $m=2$. Similarly, 31 modular additions and 9 modular multiplications are required when $m=4$.

Similar to the split-radix DIT DGT, we would also set up the recurrence equations based on the asymmetric split-radix DIF IDGT butterfly and the two boundary cases (i.e. Equation 20, 21, and 22) to analyze the computation cost. Observing that the split-radix DIF IDGT butterfly and the two boundary cases requiring the same number of multiplication and addition over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ as in DGT, the cost of the split-radix DIT DGT and the split-radix DIF IDGT can be represented in terms of modular multiplications $M(m)$ and modular additions $A(m)$ by the following recurrences:

$$M(m)=\begin{cases} M(\frac{m}{2})+2M(\frac{m}{4})+3m/2, & \text{if } m>4,\\ 9, & \text{if } m=4,\\ 3, & \text{if } m=2. \end{cases} \quad (23)$$

$$A(m)=\begin{cases} A(\frac{m}{2})+2A(\frac{m}{4})+11m/2, & \text{if } m>4,\\ 31, & \text{if } m=4,\\ 9, & \text{if } m=2, \end{cases} \quad (24)$$

such that,

$$\begin{aligned} M(m) &= mlog_2 m+\frac{m}{3}-\frac{(-1)^{log_2 m}}{3},\\ A(m) &= \frac{11m}{3}log_2 m+\frac{5m}{9}-\frac{5(-1)^{log_2 m}}{9}. \end{aligned} \quad (25)$$

Having the above analysis, Table 3 compares the modular multiplication and modular addition of low complexity NTT/INTT [10], the classic DGT/IDGT [29] and the proposed split-radix DGT/IDGT for given problem sizes n. In terms of modular multiplication, the proposed split-radix DGT/IDGT has the smallest number of modular multiplications among the three algorithms. Comparing with the classic DGT and IDGT, split-radix DGT and IDGT reduce 47.3% and 57.8% of modular multiplications, respectively, when the polynomial size $n=128$ (i.e. DGT/IDGT size of $m=\frac{n}{2}=64$).

The split-radix DGT and IDGT could also save 9.6% of modular multiplications compared to the low-complexity NTT and INTT. Similarly, the split-radix DGT/IDGT needs one less stage than the low-complexity NTT/INTT. The reason is that a length-n NTT/INTT is equivalent to a length-$\frac{n}{2}$ DGT/IDGT (which means the transform size is halved in DGT/IDGT compared to NTT/INTT). Additionally, as DIT is applied in DGT and DIF is used in IDGT, no bit-reordering on the coefficients are required.

The split-radix DGT and IDGT could be applied to solve the polynomial multiplication on $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ when $2n|(q-1)$ and n is a power of 2, and it is a more efficient variant comparing with the classic DGT/IDGT. The split-radix DGT and IDGT could also provide a shorter transform length and need one less stage comparing with the other NTT/INTT algorithms. Thus, the proposed split-radix DGT/IDGT is competitive in the design of high-performance NWC architecture.

4 Architecture Design of Split Radix DGT/IDGT

As mentioned in Section 2.1, the CRYSTALS-KyberKEM adopted the parameter set (n,q) as (256,3329), which would divide the length-256 NTT into two length-128 NTTs of odd-index terms and the even-index terms, respectively. Considering using DGT to replace the length-128 NTT in CRYSTALS-KyberKEM, the pack operation (as shown in Equation 5) is required to pack the odd-index terms and the even-index terms from $\mathbb{Z}_q$ into $\mathbb{Z}_q[z]/\langle z^2+1\rangle$. Therefore the DGT in CRYSTALS-KyberKEM consists of two length-64 DGTs for the odd-index terms and the even-index terms (noted as odd polynomial and even polynomial in this paper, respectively). Additionally, we found the available twiddle factor ζ_m in KyberKEM, as $\zeta_{64}=1+737*z$, based on the method proposed in [29].

The overall architecture of the split-radix DGT/IDGT is shown in Figure 2. In order to cut down the hardware overhead on implementing the NWC via split-radix DGT/IDGT, we decided to integrate the operation of the split-radix DGT, the split-radix IDGT and the component-wise multiplication into a unified Split Radix DGT/IDGT (SRDGT) module. In the proposed module, a unified DGT butterfly unit (BFU), a twiddle factor memory *ZETA_ROM*, a stream permutation network (SPN), and a control unit are involved.

4.1 Unified SRDGT Butterfly Unit

The unified SRDGT BFU is designed to compute DGT and IDGT in iterative nature [36]. As shown in Figure 3, a pipelined architecture is designed to increase the throughput of the unified DGT BFU module. When the pipeline is fulfilled, the proposed unified DGT BFU could read and write two data points if working under DGT/IDGT mode. When the BFU is switched to compute CWM, it could support read and write of four data points simultaneously.

The proposed unified SRDGT BFU is designed to support nine working modes to implement the SRDGT butterfly shown in Figure 1 in a compact way. The 6-bits control signal *sel* and its corresponding mode is shown in Table 4 and Figure 3. Figure 4 illustrates the details of the active data path and the operators for each mode. Among the nine working modes of the unified DGT BFU, four are for the iterative DGT (*DGT 0-1*, *DGT 0-2*, *DGT 0-3*, and *DGT 1* shown in Figure 1), three are for the iterative IDGT (*IDGT 0-1*, *IDGT 0-2*, and *IDGT 1* shown in Figure 1), and two are for

TABLE 3
The comparison on the number of modular operations between low complexity NTT/INTT [10], classic DGT/IDGT [29] and the split-radix DGT/IDGT for given problem size n.

n	Relative # of Modular Addition			Relative # of Modular Multiplication		
	LC NTT/INTT[1*]	DGT/IDGT[2]	SRDGT/SRIDGT[3*]	LC NTT/INTT[1]	DGT/IDGT[2]	SRDGT/SRIDGT[3]
128	0.62/0.74	1.42/1.64	1.00/1.02	1.11/1.11	1.90/2.37	1.00/1.00
256	0.61/0.72	1.39/1.58	1.00/1.02	1.09/1.09	1.84/2.25	1.00/1.00
512	0.60/0.72	1.37/1.54	1.00/1.02	1.08/1.08	1.80/2.16	1.00/1.00
1024	0.59/0.71	1.36/1.50	1.00/1.02	1.07/1.07	1.77/2.09	1.00/1.00

[1] LC NTT/INTT represents the low complexity NTT/INTT [10].
[2] DGT/IDGT represents the classic DGT/IDGT [29].
[3] SRDGT/SRIDGT represents the split-radix DGT/IDGT proposed in this work.
[*] In the LC INTT [10] and the SRIDGT, the half operations would be introduced to avoid post processing. Such half operation could be preformed by a modular addition and an bit shift, which would be expected to increase 1.25 or 0.44 modular addition to each LC INTT butterfly operation or SRIDGT butterfly operation, respectively. The modular addition caused by half operations are included in the Table above.

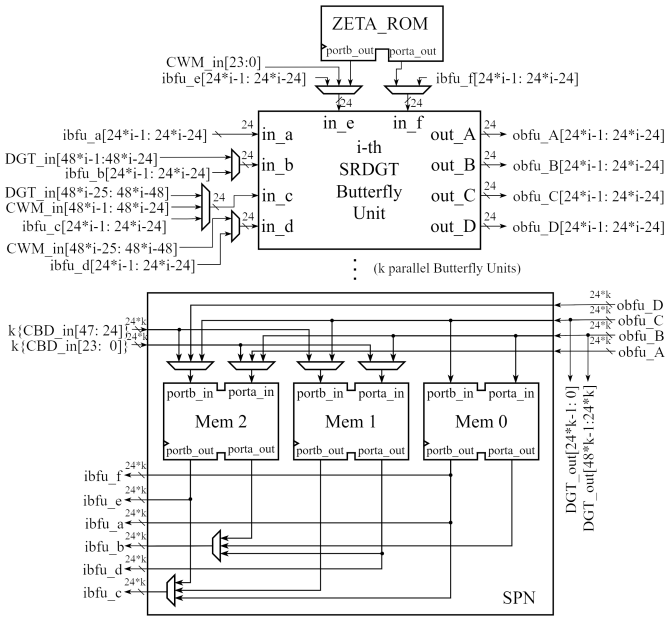


Fig. 2. Overall architecture of Split Radix DGT/IDGT (control unit is not shown). Extensions to k parallel BFUs are demonstrated, where k is noted as scalability coefficient. The index of the BFU determines the route, ranging from 0 to $k-1$. Mem 0, Mem 1 and Mem 2 are instantiated by dual-port RAM. ZETA_ROM is instantiated by dual-port ROM.

CWM (*CWM 0*, and *CWM 1*). The modes for iterative DGT and IDGT need to be switched during the computation, as described in Algorithms 1 and 2. The SEQ would also be used to control the mode switch. Since the computation of CRYSTALS-KyberKEM only involves length-64 DGT/IDGT, the SEQ would be a 32-bit constant integer `0X0000FF0D`.

The CWM is defined as

$$\begin{aligned}&(x\hat{\bar{r}}_{2i+1} + \hat{\bar{r}}_{2i}) \bmod (x^2 - \zeta_{64}^{4\text{br}(i)+1}) \\ \equiv&(x\hat{\bar{a}}_{2i+1} + \hat{\bar{a}}_{2i})(x\hat{\bar{b}}_{2i+1} + \hat{\bar{b}}_{2i}),\end{aligned} \tag{26}$$

By using the Karatsuba-based CWM based on [9], the number of multiplications over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ could be four:

$$\begin{aligned}\hat{\bar{r}}_{2i} &= \hat{\bar{a}}_{2i}\hat{\bar{b}}_{2i} + \hat{\bar{a}}_{2i+1}\hat{\bar{b}}_{2i+1}\cdot\zeta_{64}^{4\text{br}(i)+1}, \\ \hat{\bar{r}}_{2i+1} &= (\hat{\bar{a}}_{2i} + \hat{\bar{a}}_{2i+1})(\hat{\bar{b}}_{2i} + \hat{\bar{b}}_{2i+1}) \\ &\quad - (\hat{\bar{a}}_{2i}\hat{\bar{b}}_{2i} + \hat{\bar{a}}_{2i+1}\hat{\bar{b}}_{2i+1}).\end{aligned} \tag{27}$$

TABLE 4
Nine modes for our proposed unified BFU with the setup signal 'sel'. Symbol '$+$' demotes coresponding operator is in use, while '$-$' denotes it is idle at the moment.

Mode	sel[5:0]	Mul 0	Mul 1	Rotator
DGT 0-1	101010	$-$	$+$	$-$
DGT 0-2	100010	$-$	$-$	$-$
DGT 0-3	100000	$-$	$-$	$+$
DGT 1	101110	$+$	$+$	$-$
IDGT 0-1	001011	$+$	$+$	$-$
IDGT 0-2	000011	$-$	$-$	$-$
IDGT 1	001101	$+$	$+$	$+$
PWM 0	001110	$+$	$+$	$-$
PWM 1	011110	$+$	$+$	$-$

The Karatsuba-based CWM would be computed by using two working modes in the proposed unified SRDGT BFU, namely *CWM 0* and *CWM 1*. We map the computation of Equation 27 to the data flow of BFU as:

$$\begin{aligned}\text{CWM 0: } &s_0 = \hat{\bar{a}}_{2i} + \hat{\bar{a}}_{2i+1}, s_1 = \hat{\bar{b}}_{2i} + \hat{\bar{b}}_{2i+1}, \\ &m_0 = \hat{\bar{a}}_{2i}\hat{\bar{b}}_{2i}, m_1 = \hat{\bar{a}}_{2i+1}\hat{\bar{b}}_{2i+1} \\ \text{CWM 1: } &\hat{\bar{r}}_{2i} = m_0 + m_1\cdot\zeta_{64}^{4\text{br}(i)+1}, \\ &\hat{\bar{r}}_{2i+1} = s_0\cdot s_1 - (m_0+m_1).\end{aligned} \tag{28}$$

The detailed dataflow and working mechanism of the unified SRDGT BFU are shown in Figure 2 and Table 4.

4.2 Multiplier over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$

Notice that each multiplication in Equation 27 is multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$. Therefore, a multiplier over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ is required to compute this operation.

The DSP48E1 slice in Xilinx FPGA consists of one multiplier and two adders. Since all the operators in DSP48E1 is programmable by fully utilizing these high performance hardware resources, one could design a high throughput multiplier over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$. In this work, the Karatsuba algorithm is adopt, given $\bar{s} = (a)z + b, \bar{t} = (c)z + d$, the multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ is rearranged and shown as:

$$\begin{aligned}\bar{s}\circ\bar{t} = &[a(d-c) + c(a+b) \mod q]z \\ &+ [b(c+d) - c(a+b) \mod q].\end{aligned} \tag{29}$$

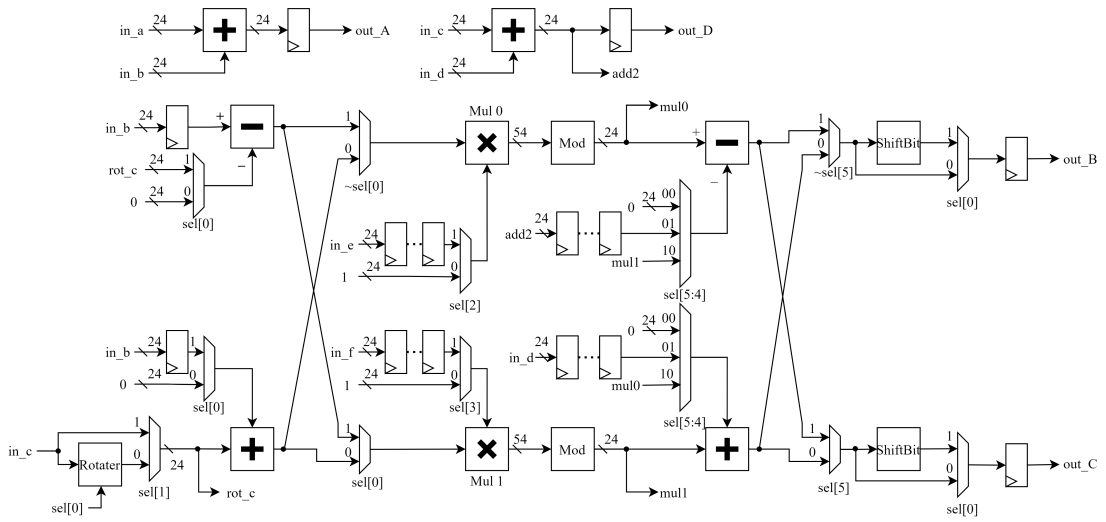


Fig. 3. The detailed block diagram of the proposed unified SRDGT BFU. Each output ports (out_A, out_B, out_C, out_D) are illustrated separately, with input ports (in_a, in_b, in_c, in_d, in_e, in_f). Control signal 'sel' in different operations are presented in Table 4.

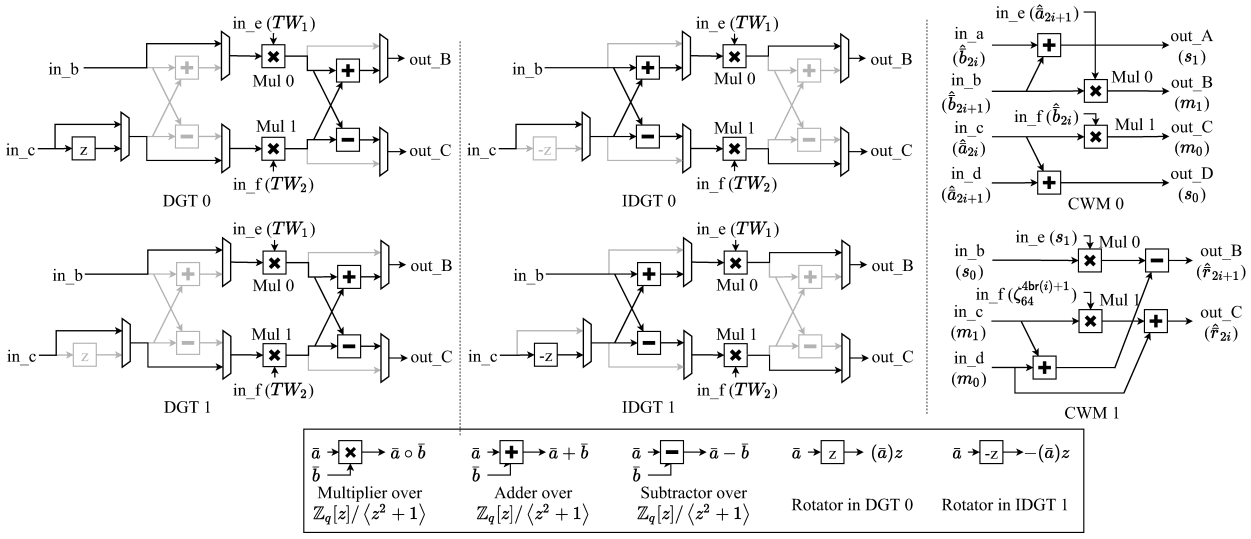


Fig. 4. Active data path diagram for DGT, IDGT and CWM of the proposed unified BFU. All the registers in the proposed architecture are waived in this diagram. The mode *DGT 0-1*, *DGT 0-2*, and *DGT 0-3* in Table 4 are working on 'DGT 0' data path. The mode *IDGT 0-1* and *IDGT 0-2* are working on 'IDGT 0' data path. And the mode *DGT 1*, *IDGT 1* , *CWM 0*, and *CWM 1* are working on 'DGT 1', 'IDGT 1', 'CWM 0', and 'CWM 1' data path, respectively. In 'DGT 0', 'DGT 1', 'IDGT 0', and 'IDGT 1', we map the twiddle factor data points (TW_1 and TW_2) in Algorithm 1 and 2 to the port in_e and in_f, respectively. And in 'CWM 0' and 'CWM 1', we map the data points in Equation 28 to the corresponding ports. The data points are noted in brackets. It is notable there are only 4 unique data points reading required in 'CWM 0'.

As can be seen from Equation 29, there are three multiplications, four additions, two subtractions , and two modular reductions in each multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$. We propose mapping the whole computations in Equation 29 into three DSP48E1, as shown in Figure 5, to reduce the wiring delay between the logics and the DSP cores. The proposed architecture of $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ multiplier achieves a working frequency of 299MHz on Xilinx Artix-7 platform.

4.3 Stream Permutation Network and Fully Pipelined Scheduling

The proposed stream permutation network (SPN) and the data scheduling plan are designed to support two main goals for single SRDGT BFU: 1) SPN should satisfy the bandwidth requirement of the SRDGT BFU. 2) the schedule of SPN should ensure a fully pipelined working mode of DGT/IDGT.

The above goals could be achieved based on three important observations from Figure 4 as follows:

1) The SRDGT BFU provides 4 active input ports in 'DGT' and 'IDGT' modes, 6 active input ports in 'CWM 0' mode and 5 active input ports in 'CWM 1' mode.
2) The 2 input data points from *ZETA_ROM* (i.e., the twiddle factors TW_1 and TW_2) use specific datapath and would not depend on SPN.
3) The pair of input ports that have the same data input (i.e., the port pairs (in_a, in_f) and (in_d, in_e) in 'CWM 0') can share one Reading operation from SPN.

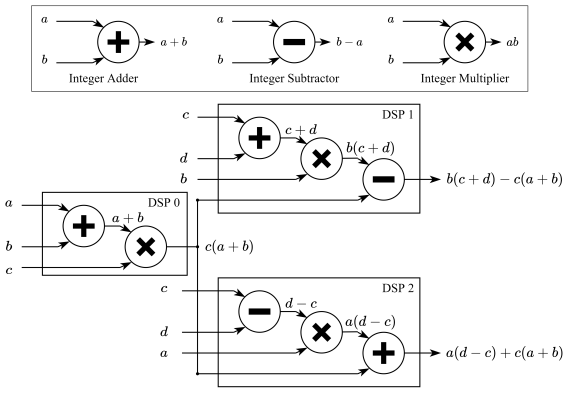


Fig. 5. Proposed architecture of multiplier over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ with three DSP48E1 slices. The rearranged multiplication over $\mathbb{Z}_q[z]/\langle z^2+1\rangle$ are described in Equation 29.

Based on the above observations, SPN should support 2/2/4 data points reading per cycle in DGT/IDGT/CWM mode, respectively. Similarly, SPN should also support 2/2/4 data points writing per cycle in DGT/IDGT/CWM mode, respectively.

In order to satisfy the SPN data width requirement, three true dual-port BRAM (namely *MEM 0*, *MEM 1*, and *MEM 2*) as shown in Figure 2 are placed to work in parallel. This design enables a maximum 6 data points read/write simultaneously.

When SRDGT BFU works in DGT/IDGT mode, the two read ports of one BRAM and the two write ports of another BRAM are enabled. As shown in Figure 6, this design enables 2 data points read and 2 data points write simultaneously. Note that the coefficients of the intermediate polynomial are stored in the memory in order, and each coefficient occupies one address in the BRAM. Such configuration allows us to use the memory address generating method stated in Algorithm 1 and 2 straightforwardly.

When SRDGT BFU works in CWM mode, there are 4 data points send to BFU and 4 data points output from BFU in each cycle. We enable the read ports of *MEM 0 port b* and *MEM 1 port a*, and the write ports of *MEM 0 port a*, *MEM 1 port b*, *MEM 2 port a*, and *MEM 2 port b*, as shown in Figure 7. As the data input port *CWM_in* also supports 2 data points input, the bandwidth requirement of SPN is fulfilled.

The main challenge of implementing a fully pipelined iterative DGT/IDGT lies in the data dependency between adjacent transform stages. The proposed fully pipelined scheduling plan is specific for the DGT/IDGT in KyberKEM, consisting of two length-64 DGTs. We intersperse the two length-64 DGTs and process them alternately to eliminate the data dependency between adjacent transform stages. Figure 6 provides a detailed example of memory scheduling for DGT in KyberKEM in the first two stages. This fully pipelined scheduling plan is also extended to compute IDGTs in KyberKEM.

We analyze and compare the cycle count of the fully pipelined SRDGT BFU and the state-of-the-art LC NTT in [9]. The SRDGT BFU requires $2\times 64/2\times log_2 64 = 384$ cycles for the length-64 DGTs of odd and even polynomials, and no pipelined bubble exists. Calculating the same length-128 NTT, LC NTT [9] requires $128/2 \times log_2 128 = 448$ cycles of odd and even polynomials, with additional 64 cycles of pipelined bubbles to write the results back to BRAMs. The above comparison demonstrates the advantages of the proposed halved transform length DGT, data scheduling plan, and fully pipelined architecture.

4.4 Extensions to multiple BFUs

In KyberKEM, a higher security level requires more DGT computation tasks. In order to support multiple tasks simultaneously, we designed the extension to multiple BFUs, as shown in Figure 2. If we note the scalability coefficient as k, each memory block in SPN will be expanded to $k \times 24$ bit wide, corresponding to k BFUs operating simultaneously for k independent DGT/IDGT/CWM tasks. Meanwhile, since the bit width of the DGT data points is a multiple of 8, our extended Split Radix DGT architecture can use the byte write function of the Xilinx BRAM instances to specify a storage location for the inputting data. Thus, the extended Split Radix DGT architecture accepts a single polynomial or k polynomials that need to be operated simultaneously as inputs, improving the flexibility of the schedule when applied in the upper-level modules.

5 Hardware Architecture of KyberKEM

KyberKEM involves key generation, encapsulation, and decapsulation. Interested readers could refer to [2] for the details of these algorithms. In this work, hardware architecture is designed and shown in Figure 8 to support these KyberKEM algorithms. The Centered Binomial Distribution (CBD) module and Reject Sampling module perform sampling in the functions CBD_η and Parse, respectively. The Compress and Decompress modules are responsible for the Compress and Decompress of ciphertext, respectively. The Encode module could transfer the data format from the byte array to the coefficients of a polynomial, and the Decode module transfers the coefficients of a polynomial back to the byte array. The Encode and Decode modules are modified from the open-source code of [9]. The Keccak module computes the functions of SHAKE128, SHAKE256, SHA3-256, and SHA3-512. The functionality of the Keccak module is expanded from the open-source code [37], and it would take 24 clock cycles to execute 24 rounds in the function KECCAK-f.

We used the bandwidth matching carrying through the architecture to increase the area time efficiency as [9] did. In addition, we divide the entire structure into three parts, with different data bit widths for different parts. The advantage of setting bandwidth matching in different parts is the overall hardware latency, and the consumed resources can trade off based on the security level. The data bandwidth is 64 bits, 48 bits, and $48 \times k$ bits in the I/O part, the sample/serialization part, and the DGT part, respectively, where k is the security level parameter of KyberKEM and equals to the scalability parameter in the Split Radix DGT architecture defined in Section 4.4. The I/O part includes the input and output FIFOs, working as the input/output buffer of the architecture. In the sample/serialization part, the byte array from input FIFO would be sent to the Keccak module to sample and the Decode module to de-serialize into 48-bit

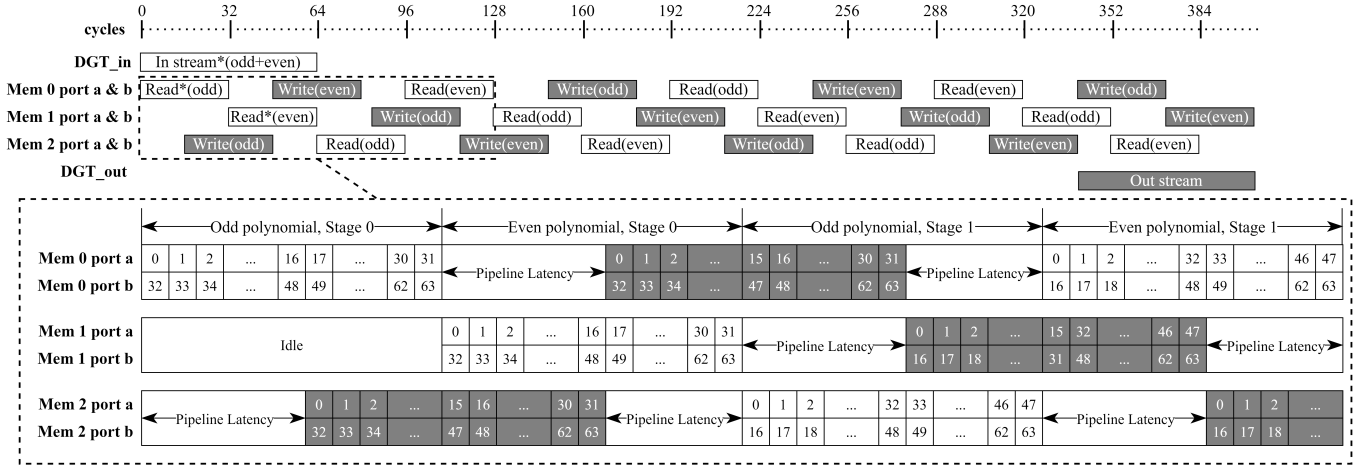


Fig. 6. Scheduling of memory operations for SRDGT in KyberKEM. Two sources are acceptable: *DGT_in* or pre-stored in *Mem 0* and *Mem 1*. When the input polynomial coming from *DGT_in*, the operation box **In stream**$*$ are enabled and the **Read**$*$ are disabled. When the input polynomial stored in *Mem 0* and *Mem 1*, the operation boxes **Read**$*$ are enabled and the **In stream**$*$ are disabled. Detailed scheduling of memory operations in the first two stages are also shown below. The white boxes represents Read operations, and the black boxes represents Write operations. The address of data is presented inside the boxes if applicable.

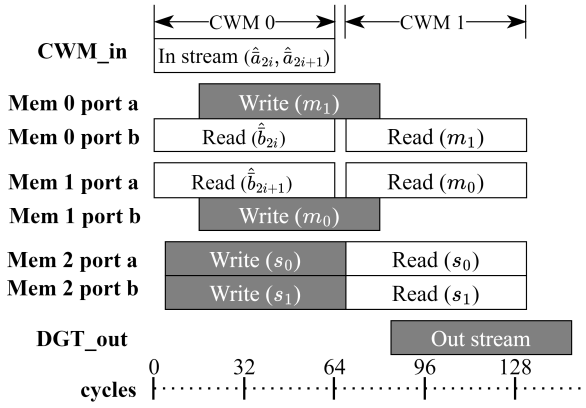


Fig. 7. Scheduling of memory operations for the proposed CWM. The data points in Equation 28 are noted on the BRAM ports in the working modes 'CWM 0' and 'CWM 1'.

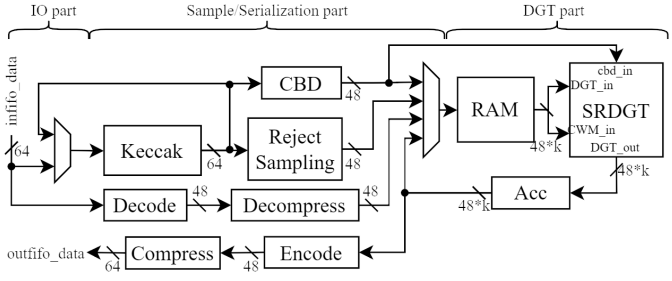


Fig. 8. The architecture of KyberKEM - Split Radix DGT hardware. (control units, the input and output FIFOs are not shown)

width. Compress would accept the 48-bit-width data from Encode and serialize it to 64-bit width data for the output FIFO. RAM stores the sampling polynomials from CBD and Reject Sampling and the decompressed polynomials from Decompress. The byte write function of the Xilinx BRAM instance would be used in the RAM module to facilitate the flexibility of the write bandwidth. When k polynomials for DGT/IDGT/CWM are ready, the SRDGT module will load these k polynomials and process them simultaneously.

In this work, we also applied the *just-in-time* strategy as [38] to minimize the memory footprint. The *just-in-time* strategy means that the sampling polynomials are generated based on the requirement of the succeeding computation. For example, we applied the strategy for the data generated by Reject Sampling. The Reject Sampling module samples the output from the Keccak module under the uniform distribution. The output of the Reject Sampling is stored in the RAM module, including $\hat{\mathbf{A}}$ in key generation and $\hat{\mathbf{A}}^T$ in encryption, and would be passed to the SRDGT module until k polynomials are ready. Each of these polynomials in the cases would be used only once. Thus, the memory space could be overwritten by the following k polynomials based on the *just-in-time* strategy, and the memory space reserved would be reduced from k^2 polynomials to k polynomials.

6 Implementation Results and Comparisons

The proposed hardware design of KyberKEM-SRDGT has been synthesized and implemented using Vivado 2019.2 design suite on Xilinx XC7A200 (Artix-7) FPGA device, with all the building blocks implemented in hardware.

6.1 Split-Radix DGT Module Results and Comparisons

The hardware resource utilization and the latency specification of the proposed SRDGT module are shown in Table 5. The detailed cycle counts of NTT (DGT), INTT (IDGT), and component-wise multiplication (CWM) are also compared with state-of-the-art implementations. We only enclosed the $k = 1$ case of our design for a fair comparison since the proposed design with larger k is designed to process k independent polynomials simultaneously. The measurement of the efficiency of the hardware implementations is based on the area-time product (ATP), which is computed by the product of LUT, BRAM, and DSP resources and the computing time. We analyzed the ATP for NTT and CWM for a detailed

TABLE 5
FPGA Implementation Result for the Proposed Split Radix DGT Core and Comparison with State-of-the-Art (n=256)

Work	Method	BFU	Area				Freq[MHz]	Cycles[CCs]			NTT/CWM ATP ratio[1]		
			#LUTs	#FFs	#DSPs	#BRAMs		NTT	INTT	CWM	LUT	BRAM	DSP
[13]^{C}	SW	-	-	-	-	-	100	7725	9347	27873	-/-	-/-	-/-
[22]^{A}	HW/SW	-	-	-	-	-	59	6868	6367	2395	-/-	-/-	-/-
[21]^{A}	HW/SW	-	2983	0	0	11	25	1289	-	-	59.7/-	78.4/-	0.0/-
[23]^{Z}	HW/SW	2	2908	170	9	0	-	1935	1930	-	-/-	-/-	-/-
[18]$^{\dagger, A}$	HW	2	230	175	0	1	312	570	570	-	0.2/-	0.3/-	0.0/-
[28]$^{\dagger, A}$	HW	2×2	801	717	4	2	222	324	324	-	0.5/-	0.4/-	0.6/-
[27]$^{\dagger, A}$	HW	1	479	472	1	2	246	4108	-	-	3.1/-	4.6/-	1.7/-
[25]$^{\dagger, A}$	HW	2	609	640	2	4	257	490	490	-	0.5/-	1.1/-	0.4/-
[16]^{A}	HW	1	360	145	3	2	115	940	1203	1289	1.1/4.6	2.3/9.0	2.5/10.1
[16]^{A}	HW	2	737	290	6	4	115	474	602	1289	1.2/9.3	2.3/18.0	2.6/20.3
[9]^{A}	HW	2	1737	1167	2	3	161	512	576	256	2.1/3.1	1.3/1.9	0.7/1.0
[26]^{A}	HW	1	948	325	1	2.5	190	904	904	647	1.8/3.6	1.6/3.4	0.5/1.0
[26]^{A}	HW	4	2543	792	4	9	182	232	233	167	1.3/2.6	1.6/3.3	0.5/1.1
[26]^{A}	HW	16	9508	2684	16	35	172	69	71	47	1.5/2.9	1.9/3.8	0.7/1.3
This work(k=1)^{A}	**HW**	**1**	**1603**	**2004**	**6**	**4.5**	**239**	**384**	**384**	**132**	**1.0/1.0**	**1.0/1.0**	**1.0/1.0**

† Do not support component-wise multiplication (CWM).
^{C} Implemented on Cortex-M4 platform.
^{Z} Implemented on Zynq-7000 FPGA platform.
^{A} Implemented on Artix-7 FPGA platform.
[1] NTT ATP ratio is the normalized product of FPGA resources and the NTT total time,
and CWM ATP ratio is the normalized product of FPGA resources and the CWM total time, by setting this work as baseline.

comparison since only the hardware architectures of [9], [16], [26], and the proposed design support NTT/INTT and CWM. Notably, we do not include the comparison of the total latency and ATP for the polynomial multiplication in Table 5, since we noted the KyberKEM would not use the complete polynomial multiplication (including 2 NTTs, 1 CWM, and 1 INTT) during the key generation, encapsulation and decapsulation [2]. For simplicity, the ATP ratios are provided instead of the original ATP indices.

Due to the careful placement of registers and the usage of high-speed DSP48E1 slides in Artix-7 FPGA, the proposed SRDGT module could operate at a frequency of 239MHz. Another merit of the SRDGT algorithm and the proposed architecture is the relatively small cycle count. Specifically, the DGT, IDGT and CWM computations require 384, 384, and 132 cycles, respectively, for length-256 polynomial multiplication. And the latency of DGT, IDGT and CWM are 1.6 μs, 1.6 μs, and 0.55 μs, respectively.

In comparison to the SW implementation in work [13], the cycle count of the SRDGT architecture achieves a speedup of 20.1×, 24.3×, 211.2× for NTT(DGT), INTT(IDGT), and CWM, respectively. In comparison to the HW/SW implementations in [21], [22], [23], the SRDGT hardware achieves more than 32.4× speedup for NTT (DGT) computation. Besides, [21] and [23] use 1.86× and 1.81× more LUTs than our design in a similar FPGA platform, respectively.

The state-of-the-art HW implementations are divided into two groups depending on whether the CWM is supported. Among these works, the hardware in [9], [16] and [26] support CWM. [16] has a higher NTT ATP ratio in LUT, BRAM, and DSP compared to our work, indicating the high efficiency of our architecture. The proposed hardware still has lower cycle counts because the transform size is halved, and only six stages are required in our split radix DGT and IDGT, with the full-pipelined working nature provided by our SPN. Xing *et al.* [9] also presented a unified butterfly unit for NTT, INTT, and CWM. However, taking advantage of the novel split-radix DGT algorithm, the cycle count of this work is only $384/512 = 75\%$ of the counts in [9] for NTT (DGT), and only $132/256 = 51.6\%$ of the counts in [9] for CWM. Our work outperforms the NTT ATP and CWM ATP compared to [9] except for the NTT-DSP ATP because of the compact design in the unified BFU of [9]. The designs in [26] propose three different configurations to trade off the hardware resources and speed. Our architecture outperforms all these configurations concerning the LUT-NTT and BRAM-NTT ATP, while their work could have a better DSP-NTT ATP. Besides, our architecture outperforms the CWM ATP ratios for LUT, BRAM, and DSP compared to [26].

We use the clock cycle counts of NTT (DGT) and INTT (IDGT), unlike directly using the ATP of the NTT and CWM when comparing our work with [18], [25], [27], [28] for fairness since these works do not support CWM while our work uses additional hardware resources for CWM. Only [28] among the previous works listed above consumes fewer clock cycles than the proposed architecture. The authors apply the KRED reduction algorithm into a 2-layer merged NTT, which relies on the special form of the prime. We would explore the 2-layer merged DGT butterfly unit in our future work.

6.2 KyberKEM Results and Comparisons

Table 6 shows the hardware resource utilization and the latency of the proposed KyberKEM hardware system. Different security level parameter sets, including Kyber-512-CCA, Kyber-768-CCA, and Kyber-1024-CCA, were implemented, and the results were compared with the state-of-the-art implementations concerning speed and hardware resource utilization. The speed of the hardware is obtained by taking the cycle counts and total time, including the key generation, encapsulation, and decapsulation. For simplicity, the total cycle ratio is provided in Table 6 using our results as the baseline. The overall efficiency of the hardware

TABLE 6
FPGA Implementation Result for the Proposed KyberKEM-Split Radix DGT and Comparison with State-of-the-Art.

Work	Method	Platform	Area				Freq	Cycles[kCCs]			Total	Total	ATP ratio[1]		
			#LUTs	#FFs	#DSPs	#BRAMs	[MHz]	KG	Enc	Dec	time [μs]	cycle ratio	LUT	BRAM	DSP
Kyber-512-CCA															
[13]	SW	Cortex-M4	-	-	-	-	24	499	634	597	72083.3	227.6	-	-	-
[22]	HW/SW	Artix-7	2K	2K	5	34	59	710	971	870	43237.3	335.7	201.0	4577.8	504.9
[21]	HW/SW	Artix-7	15K	3K	11	14	25	75	132	142	13960.0	45.9	485.9	608.6	358.6
[23]	HW/SW	Zynq-7000	24K	11K	21	32	-	150	193	205	-	72.1	-	-	-
[16]	HW	Artix-7	18K	5K	6	15	115	4	7	10	182.6	2.8	7.6	8.5	2.6
[17]	HW	Artix-7	9K	9K	4	4.5	220	2.2	3.2	4.5	45.2	1.3	1.0	0.6	0.4
[9] - server	HW	Artix-7	7K	5K	2	3	161	3.8	5	6.7	96.9	2.1	1.7	0.9	0.5
This work	**HW**	**Artix-7**	**12K**	**14K**	**12**	**9**	**213**	**1.7**	**2.4**	**3.5**	**35.7**	**1.0**	**1.0**	**1.0**	**1.0**
Kyber-768-CCA															
[13]	SW	Cortex-M4	-	-	-	-	24	947	1113	1059	129958.3	311.9	-	-	-
[21]	HW/SW	Artix-7	15K	3K	11	14	25	112	178	191	19240.0	48.1	429.4	435.1	246.9
[23]	HW/SW	Zynq-7000	24K	11K	21	32	-	273	326	340	-	93.9	-	-	-
[16]	HW	Artix-7	16K	6K	9	16	115	7	10	14	269.6	3.1	6.4	7.0	2.8
[17]	HW	Artix-7	11K	10K	6	6.5	220	2.6	3.7	4.9	51.3	1.1	0.8	0.5	0.4
[9] - server	HW	Artix-7	7K	5K	2	3	161	6.3	7.9	10	150.4	2.4	1.7	0.7	0.4
This work	**HW**	**Artix-7**	**14K**	**17K**	**18**	**13**	**210**	**2.1**	**3.4**	**4.5**	**47.6**	**1.0**	**1.0**	**1.0**	**1.0**
Kyber-1024-CCA															
[13]	SW	Cortex-M4	-	-	-	-	24	1525	1732	1653	204583.3	350.7	-	-	-
[22]	HW/SW	Artix-7	2K	2K	5	34	59	2203	2619	2429	122898.3	517.9	227.9	4059.2	373.1
[21]	HW/SW	Artix-7	15K	3K	11	14	25	149	223	241	24520.0	43.8	340.5	333.5	163.8
[23]	HW/SW	Zynq-7000	24K	11K	21	32	NA	350	405	425	-	84.3	-	-	-
[16]	HW	Artix-7	16K	6K	12	17	112	10	14	18	375.0	3.0	5.6	6.2	2.7
[17]	HW	Artix-7	12K	11K	8	8.5	220	3.6	4.8	5.8	64.6	1.0	0.7	0.5	0.3
[18] - RB	HW	Artix-7	7K	4K	2	5.5	258	43.8	48.8	57.2	580.6	10.7	3.8	3.1	0.7
[18] - CB	HW	Artix-7	5K	2K	2	6.5	250	1148	1236	1172	14224.0	254.0	69.5	89.8	17.3
[9] - server	HW	Artix-7	7K	5K	2	3	161	9.4	11.3	13.9	214.9	2.5	1.5	0.6	0.3
This work	**HW**	**Artix-7**	**16K**	**19K**	**24**	**15**	**204**	**3.9**	**4.5**	**5.6**	**68.6**	**1.0**	**1.0**	**1.0**	**1.0**

[1] ATP ratio is the normalized product of FPGA resources and the total time by setting this work as baseline.

architecture is mainly measured by the ATP ratio, obtained by the product of LUT, BRAM, and DSP resources and the total time (noted as LUT-Time ATP, BRAM-Time ATP, and DSP-Time ATP, respectively), and normalized using our results as the baseline.

In the proposed KyberKEM architecture with SRDGT module, all the dimensions k defined in KyberKEM specification are supported. The data bandwidth of our implementation is set to 64 bits. Our design achieves more than 227.6× speedup when compared with [13], which is software implementation on ARM Cortex-M4. Compared with the HW/SW co-design in [21], our architecture achieves at least 43.8× speedup and 340.5/333.5/163.8× smaller LUT-Time ATP, BRAM-Time ATP, and DSP-Time ATP, respectively, among all the security level of KyberKEM.

We also compare our work with the related pure hardware implementations. For all the security levels of KyberKEM, the proposed hardware obtains at least 1.0×, 2.1×, 2.8×, and 10.7× speedup compared with [9], [16], [17], [18], respectively. Compared with [17], our architecture utilizes 1.0/0.6/0.4× ATP in Kyber-512-CCA, but only 0.7/0.5/0.3× ATP in Kyber-1024-CCA, in terms of LUT-Time ATP, BRAM-Time ATP, and DSP-Time ATP, respectively. The reason could be that our KyberKEM architecture could benefit more from the Split Radix DGT module at a lower security level. Nevertheless, at a higher security level, the schedule bottleneck would be Keccak and Reject Sample, but not the Split Radix DGT. This fact will cause the total cycle gap between our design and [17] to decrease gradually, namely from 1.3× to 1.0× total cycles from Kyber-512-CCA to Kyber-1024-CCA. More efficient Keccak and Reject Sample architectures are necessary to further reduce the cycle count, which would be part of our future works. In terms of ATP, the design in [9] accomplished at least 1.5× larger LUT-Time ATP, but at most 33.4% smaller BRAM-Time ATP and 83.9% smaller DSP-Time ATP than our work. The small ATP index in BRAM and DSP is due to their highly compact KyberKEM architecture and scheduling design. [16] reported at least 5.6/6.2/2.7× larger ATP compared to us, in terms of LUT-Time ATP, BRAM-Time ATP, and DSP-Time ATP, respectively. Furthermore, [18] reported 3.8/3.1/0.7× ATP compared to us, in terms of LUT-Time ATP, BRAM-Time ATP, and DSP-Time ATP, respectively.

7 Conclusion

The development of quantum computers threatens the security of the conventional public-key cryptography algorithms. CRYSTALS-KyberKEM is one of the leading algorithms in the ongoing NIST Post-Quantum Cryptography (PQC) competition. As a lattice-based cryptographic scheme, the efficiency of CRYSTALS-KyberKEM is dependent on the polynomial multiplication over $\mathcal{R}_q$ or equivalently NWC. In this paper, we explored the implementation of DGT with the split-radix method, providing a higher level of parallelism compared to the LC NTT [10] and less computational complexity compared to classic DGT [29]. The proposed architecture of split-radix DGT module could support DGT, IDGT and CWM specific for KyberKEM, and outperforms the state-of-the-arts on NWC modules. In the meantime, KyberKEM architecture with split-radix DGT module is proposed supporting all the security levels of KyberKEM. The proposed architecture could increase performance and hardware efficiency than the state-of-the-arts, specifically only 35.7μs, 47.6μs, and 68.6μs are required for Kyber-512-CCA, Kyber-768-CCA and Kyber-1024-CCA, respectively.

ACKNOWLEDGMENTS

This work is supported by Hong Kong Innovation and Technology Commission (ITF Seed Fund ITS/216/19), City University of Hong Kong (Project 9440242), and National Natural Science Foundation of China (No. 62002023 and No. 62002239)

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. IEEE, 1994, pp. 124–134.

[2] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367.

[3] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International Algorithmic Number Theory Symposium*. Springer, 1998, pp. 267–288.

[4] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *International Conference on Cryptology in Africa*. Springer, 2018, pp. 282–305.

[5] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang, "Classic McEliece: conservative code-based cryptography," *NIST submissions*, 2017.

[6] D. Moody, G. Alagic, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and J. Alperin-Sheriff, "Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process," 2020.

[7] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," *Cryptology ePrint Archive: Report 2020/795*, 2020.

[8] H. Nejatollahi, R. Cammarota, and N. Dutt, "Flexible NTT Accelerators for RLWE Lattice-Based Cryptography," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 329–332.

[9] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 328–356, 2021.

[10] 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.

[11] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2010, pp. 1–23.

[12] ——, "A Toolkit for Ring-LWE Cryptography," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 35–54.

[13] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-M4," in *International Conference on Cryptology in Africa*. Springer, 2019, pp. 209–228.

[14] H. M. Steffen, L. J. Kogelheide, and T. Bartkewitz, "In-depth Analysis of Side-Channel Countermeasures for CRYSTALS-Kyber Message Encoding on ARM Cortex-M4," *Cryptology ePrint Archive*, 2021.

[15] D. T. Nguyen and K. Gaj, "Optimized software implementations of CRYSTALS-Kyber, NTRU, and Saber using NEON-based special instructions of ARMv8," in *Proceedings of the NIST 3rd PQC Standardization Conference (NIST PQC 2021)*, 2021.

[16] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-Set Accelerated Implementation of CRYSTALS-Kyber," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4648–4659, 2021.

[17] V. B. Dang, K. Mohajerani, and K. Gaj, "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *Cryptology ePrint Archive*, 2021.

[18] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, "A Configurable CRYSTALS-Kyber Hardware Implementation with Side-Channel Protection," *Cryptology ePrint Archive*, 2021.

[19] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST Post-Quantum Cryptography-A Hardware Evaluation Study," *Cryptology ePrint Archive*, 2019.

[20] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, no. 4, p. 17–61, Aug. 2019.

[21] ——, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (extended version)," *Cryptology ePrint Archive*, 2019.

[22] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic: Accelerating Kyber and NewHope on RISC-V," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 219–242, 2020.

[23] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 239–280, 2020.

[24] D.-e.-S. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra High-Speed Polynomial Multiplications for Lattice-based Cryptography on FPGAs," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2022.

[25] C. Zhang, D. Liu, X. Liu, X. Zou, G. Niu, B. Liu, and Q. Jiang, "Towards Efficient Hardware Implementation of NTT for kyber on FPGAs," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[26] F. Yarman, A. C. Mert, E. Öztürk, and E. Savaş, "A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-Kyber PQC Scheme," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1020–1025.

[27] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "High-Performance Area-Efficient Polynomial Ring Processor for CRYSTALS-Kyber on FPGAs," *Integration*, vol. 78, pp. 25–35, 2021.

[28] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography," in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2021, pp. 94–101.

[29] A. Al Badawi, B. Veeravalli, and K. M. M. Aung, "Efficient Polynomial Multiplication via Modified Discrete Galois Transform and Negacyclic Convolution," in *Future of Information and Communication Conference*. Springer, 2018, pp. 666–682.

[30] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Designs, Codes and Cryptography*, vol. 75, no. 3, pp. 565–599, 2015.

[31] E. Fujisaki and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes," in *Annual international cryptology conference*. Springer, 1999, pp. 537–554.

[32] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*, ser. Springer Series in Information Sciences, K.-s. Fu, T. S. Huang, and M. R. Schroeder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, vol. 2. [Online]. Available: http://link.springer.com/10.1007/978-3-642-81897-4

[33] V. Lyubashevsky and G. Seiler, "NTTRU: truly fast NTRU using NTT," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 180–201, 2019.

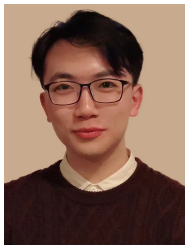
[34] E. Chu and A. George, *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press, 1999.

[35] W. M. Gentleman and G. Sande, "Fast Fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*, 1966, pp. 563–578.

[36] M. Garrido, F. Qureshi, J. Takala, and O. Gustafsson, "Hardware architectures for the fast Fourier transform," in *Handbook of signal processing systems*. Springer, 2019, pp. 613–647.

[37] "Sha3 (keccak)." [Online]. Available: https://opencores.org/projects/sha3

[38] A. Karmakar, J. M. B. Mera, S. S. Roy, and I. Verbauwhede, "Saber on ARM CCA-secure module lattice-based key encapsulation on ARM," *Cryptology ePrint Archive*, 2018.

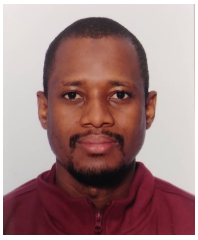


Guangyan Li received the B.Eng degree in 2020 from the Department of Electrical Engineering, City University of Hong Kong. He joined the overseas internship scheme to the LIRMM at Montpellier, France from June to August of 2019. He is now pursuing the PhD degree in the Department of Electrical Engineering from City University of Hong Kong. His research interests include reconfigurable computing with FPGA, and post-quantum cryptography algorithm design.

Donglong Chen received the PhD degree from the Department of Electronic Engineering, City University of Hong Kong, in 2015. He was a visiting research scholar of COSIC, KU Leuven, Belgium, in 2013. After completing his PhD degree study, he spent four years in the industry including Huawei Technology Co., Ltd. and Tencent Technology Co., Ltd. He is currently an Associate Professor in the Falcuty of Science and Technology, BNU-HKBU United International College (UIC), China. His research interests include algorithm-hardware co-optimization for cryptosystems, software/hardware co-design for AI algorithms, and secure multi-party computation.

Gaoyu Mao received the B.Eng. degree from School of Microelectronics, Shandong University in 2020. He is now pursuing the PhD degree in the Department of Electrical Engineering from City University of Hong Kong. His research interests include software/hardware co-design architecture and post-quantum cryptography algorithm design.

Wangchen Dai received the B.Eng. degree in electrical engineering and automation from Beijing Institute of Technology, China, in 2010, the M.A.Sc. degree in electrical and computer engineering from the University of Windsor, Canada, in 2013, and the Ph.D. degree in electronic engineering from the City University of Hong Kong in 2018. After completing the Ph.D. study, he had appointments at Hardware Security Lab, Huawei Technologies Company Ltd., in 2018, and the Department of CSSE, Shenzhen University in 2020, respectively. He is currently working as a Senior Researcher with Zhejiang Lab, Hangzhou, China. His research interests include cryptographic hardware and embedded systems, fully homomorphic encryption, and reconfigurable computing.



Abdurrashid Ibrahim Sanka received his B.Eng Electrical (first class) from the Department of Electrical Engineering, Bayero University, Kano in 2011. He then got his master degree in Embedded Microelectronics and Wireless Systems from Coventry University, United Kingdom (UK) in 2014. He works as a lecturer at Bayero University, Kano since 2012. Dr. Sanka joined the CALAS group in 2017 for his Ph.D. in the Department of Electrical Engineering, City University of Hong Kong, and he received his Ph.D. degree in 2022. His research interests include blockchain technology, embedded systems, high performance reconfigurable computing with FPGA, and hardware and information security.

Ray C.C. Cheung (Senior Member, IEEE) received the B.Eng. (Hons.) and M.Phil. degrees in computer engineering and computer science and engineering from The Chinese University of Hong Kong (CUHK) in 1999 and 2001, respectively, and the D.I.C. and Ph.D. degrees in computing from Imperial College London (IC) in 2007. After completing his Ph.D. study, he received the Hong Kong Croucher Foundation Fellowship and moved to Los Angeles, at the Electrical Engineering Department, UCLA, where he spent two years with the Image Communication Lab for continuing his research work. He is currently an Associate Professor with the Department of Electrical Engineering, City University of Hong Kong, and the Digital Systems Lab. His current research interests include cryptographic hardware and embedded system designs.