后量子密码 CRYSTALS-Kyber 的 FPGA 多路并行优化实现

李斌[1]，陈晓杰[2]，冯峰[1]，周清雷[1]
（1. 郑州大学计算机与人工智能学院，河南 郑州 450001；2. 信息工程大学数学工程与先进计算国家重点实验室，河南 郑州 450001）

摘 要：在基于格的后量子密码中，多项式乘法运算复杂且耗时，为提高格密码在实际应用中的运算效率，提出了一种后量子密码 CRYSTALS-Kyber 的 FPGA 多路并行优化实现。首先，描述了 Kyber 算法的流程，分析了 NTT、INTT 及 CWM 的执行情况。其次，给出了 FPGA 的整体结构，采用流水线技术设计了蝶形运算单元，并以 Barrett 模约简和 CWM 调度优化，提高了计算效率。同时，放置 32 个蝶形运算单元并行执行，缩短了整体计算周期。最后，对多 RAM 通道进行了存储优化，以数据的交替存取控制和 RAM 资源复用，提高了访存效率。此外，采用松耦合架构，以 DMA 通信实现了整体运算的调度。实验结果和分析表明，所提方案可在 44、49、163 个时钟周期内完成 NTT、INTT 及 CWM 运算，优于其他方案，具有较高的能效比。

关键词：后量子密码；CRYSTALS-Kyber；现场可编程门阵列；数论变换；多项式乘法；蝶形运算
中图分类号：TP309
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2022026

FPGA multi-unit parallel optimization and implementation of post-quantum cryptography CRYSTALS-Kyber

LI Bin[1], CHEN Xiaojie[2], FENG Feng[1], ZHOU Qinglei[1]
1. School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China
2. State Key Laboratory of Mathematical Engineering and Advanced Computing, Information Engineering University, Zhengzhou 450001, China

Abstract: In lattice-based post-quantum cryptography, polynomial multiplication is complicated and time-consuming. In order to improve the computational efficiency of lattice cryptography in practical applications, an FPGA multi-unit parallel optimization and implementation of post-quantum cryptography CRYSTALS-Kyber was proposed. Firstly, the flow of Kyber algorithm was described and the execution of NTT, INTT and CWM were analyzed. Secondly, the overall structure of FPGA was given, the butterfly arithmetic unit was designed by pipeline technology, and the Barrett modulus reduction and CWM scheduling optimization were used to improve the calculation efficiency. At the same time, 32 butterfly arithmetic units were executed in parallel, which shortens the overall calculation cycle. Finally, the multi-RAM channel was optimized to improve the memory access efficiency with alternate data access control and RAM resource reuse. In addition, with the loosely coupled architecture, the overall operation scheduling was realized by DMA communication. The experimental results and analysis show that the proposed scheme implemented can complete NTT, INTT and CWM operations within 44, 49, and 163 clock cycles, which is superior to other schemes and has high energy efficiency ratio.

Keywords: post-quantum cryptography, CRYSTALS-Kyber, FPGA, NTT, polynomial multiplication, butterfly arithmetic

0 引言

随着量子计算的发展，传统公钥密码体制面临着严重的安全问题。基于格困难的密码算法具有加密效率高和抗量子攻击等优势，成为研究热点。美国国家标准与技术研究院（NIST, National Institute

收稿日期：2021−10−27；**修回日期**：2022−01−10
基金项目：国家重点研发计划基金资助项目（No.2016YFB0800100, No.2016YFB0800101）；国家自然科学基金资助项目（No.61572444）
Foundation Items: The National Key Research and Development Program of China (No.2016YFB0800100, No.2016YFB0800101), The National Natural Science Foundation of China (No.61572444)

of Standards and Technology）从 2016 年起征集了后量子密码的标准化评选，提出了众多基于格的后量子密码方案[1]。在各种密码方案中，公钥加密体制是非常重要的一个分支，主要被应用于密钥的分配和数字签名。NIST 最近公布了第三轮 4 个基于格的公钥加密算法，CRYSTALS-Kyber 便是其中之一。

在 Kyber 格密码方案中[2]，多项式乘法运算是关键步骤，消耗了绝大部分时间和资源。现有方案中，基于蝶形运算的数论变换[3]（NTT, number theoretic transform）可以快速实现多项式的乘法。但是，在 NTT 计算过程中，蝶形运算会被执行多次，且包含模加减、模乘、模约简等多种运算[4]，极大影响了硬件整体计算效率。其次，NTT 控制逻辑包含多次循环嵌套的运算，计算结构复杂。最后，多项式系数的存取和调度复杂，且对通信带宽具有较高的要求。因此，如何利用可重构硬件现场可编程门阵列（FPGA, field programmable gate array）实现高效的 NTT 多项式乘法，这一问题亟待解决。

当前，对于格密码系统的实现，仍需要兼顾硬件的执行效率和资源消耗，需要尽量多的并行结构，以实现更快的运算速度。由此，围绕后量子密码高效能的应用需求，本文开展了 Kyber 算法的优化设计研究。首先，对 Kyber 算法进行了描述，分析了 NTT、逆向数论变换（INTT, inverse NTT）及系数对位相乘（CWM, coefficient-wise multiplication）等算法流程。然后，详细给出了流水线蝶形运算单元的优化设计，并以多通道 RAM 优化访存，减少计算时延。最后，以 32 个蝶形运算单元并行运算，实现了 FPGA 整体架构，提高了计算效率。

1　相关工作

1.1　Kyber 算法简介

整数环用 $\mathbb{Z}$ 表示，$\mathbb{Z}_q$ 表示整数环模 q 的商环，$R_q=\mathbb{Z}_q[x]/\phi(x)$ 表示模多项式 $\phi(x)$ 的整系数多项式环，其中 $\phi(x)$ 为不可约简多项式。大部分情况下 $\phi(x)=x^n+1$，则 $R_q=\mathbb{Z}_q[x]/(x^n+1)$ 表示次数最多为 $n-1$ 的多项式环。

NTT 是在离散傅里叶变换（DFT, discrete Fourier transform）数论基础上实现的，是定义在环 $\mathbb{Z}_q$ 上的线性正交变换，只在整数环上进行运算。对于多项式 $A(x)=\sum_{i=0}^{n-1}a_ix^i$，$a_i\in\mathbb{Z}_q$，有

$$\overline{A}(x)=\sum_{i=0}^{n-1}\overline{a}_ix^i \tag{1}$$

且 $\overline{a}_i=\sum_{j=0}^{n-1}a_j\omega^{ij}(\bmod q),i=0,1,2,\cdots,n-1$。那么，称 $\overline{A}(x)$ 为 $A(x)$ 的 NTT。ω 为旋转因子，是模 q 的 n 次本原单位根，满足 $\omega^n\equiv1(\bmod q)$ 且 $\omega^i\neq1(\bmod q),\forall i<n$。对于 q，其形式一般为 $q=k_{\text{odd}}\times2^{\text{pe}}+1$，且满足 $q\equiv1(\bmod 2n)$，其中 k_{odd} 为奇数且 $2^{\text{pe}}>k_{\text{odd}}$。

同理，对于 INTT，需要在 $\mathbb{Z}_q$ 上计算模逆 $\omega^{-1}(\bmod q)$ 和 $n^{-1}(\bmod q)$，并按照 $a_i=n^{-1}\sum_{j=0}^{n-1}\overline{a}_j\omega^{-ij}$ 进行计算。这样，对基于 NTT 的多项式乘法有 $C=\text{INTT}(\text{NTT}(A)\odot\text{NTT}(B))$，其中 $\odot$ 表示多项式的 CWM。

在 Kyber 算法中，NTT、INTT 和 CWM 算法流程如算法 1、算法 2 和算法 3 所示，其中 $\text{br}_{l-1}(k)$ 和 $\text{br}_{l-1}(i)$ 分别表示对 k 和 i 进行 l–1 位的比特逆序操作。

算法 1　NTT 算法

输入　$A(x)\in\mathbb{Z}_q[x]/(x^n+1)$，$\omega$，$l=\text{lb}n$，$q$

输出　$\overline{A}(x)\in\mathbb{Z}_q[x]/(x^n+1)$

1) $k=1$
2) for i = 1 to $(l-1)$ do
3) $m=2^{l-i}$
4) for s = 0 by m to n do
5) for $j=s$ to $(s+m)$ do
6) $U=a[j],V=a[m+j],W=\omega^{\text{br}_{l-1}(k)}\bmod q$
7) $T=(WV)\bmod q$
8) $E=(U+T)\bmod q,O=(U-T)\bmod q$
9) $a[j]=E,a[m+j]=O$
10) end for
11) $k=k+1$
12) end for
13) end for

算法 2　INTT 算法

输入　$\overline{A}(x)\in\mathbb{Z}_q[x]/(x^n+1)$，$\omega$，$l=\text{lb}n$，$q$

输出　$A(x)\in\mathbb{Z}_q[x]/(x^n+1)$

1) $k=0$
2) for $i=(l-1)$ to 1 do

3) $m = 2^{l-i}$

4) for $s = 0$ by m to 2^l do

5) for $j = s$ to $(s + m)$ do

6) $U = \bar{a}[j], V = \bar{a}[m + j],$ $W = \omega^{\mathrm{br}_{l-1}(k)+1} \bmod q$

7) $E = (U + V) \bmod q, O = (U - V)W \bmod q$

8) $\bar{a}[j] = \left(\frac{E}{2}\right) \bmod q, \bar{a}[m+j] = \left(\frac{O}{2}\right) \bmod q$

9) end for

10) $k = k + 1$

11) end for

12) end for

算法 3 CWM 算法

输入 $\bar{A}(x), \bar{B}(x) \in \mathbb{Z}_q[x]/(x^n + 1), \omega, l = \mathrm{lb}n, q$

输出 $\bar{C}(x) \in \mathbb{Z}_q[x]/(x^n + 1)$

1) for $i = 0$ to 2^{l-1} do

2) $W = \omega^{\mathrm{br}_{l-1}(i)+1} \bmod q$

3) $a_0 = \bar{a}[2i], a_1 = \bar{a}[2i + 1]$

4) $b_0 = \bar{b}[2i], b_1 = \bar{b}[2i + 1]$

5) $\bar{c}[2i] = (a_0 b_1 + a_1 b_0) \bmod q$

6) $\bar{c}[2i + 1] = (a_1 b_1 W + a_0 b_0) \bmod q$

7) end for

此外，在 Kyber 算法中，对 NTT/INTT 操作进行了优化处理，将原有的 256 次多项式拆分为 2 个 128 次多项式，并以 2 个 128 次多项式独立进行计算。因此，在 CWM 算法中，需要进行 128 次二次多项式的乘法运算。

Kyber 算法的参数如表 1 所示，其中 k 用于选择多项式矩阵的维度，并调整算法安全等级。

表 1 **Kyber 算法参数**

算法	安全等级	n	k	q
Kyber512	1	256	2	3 329
Kyber768	2	256	3	3 329
Kyber1024	3	256	4	3 329

Kyber 算法的密钥生成、加密和解密过程如下。

1) 密钥生成：生成 NTT 多项式矩阵 $\hat{\boldsymbol{A}} \in R_q^{k \times k}$，生成多项式向量 $\boldsymbol{s}, \boldsymbol{e} \in R_q^k$，则公钥 $\mathbf{pk} = \hat{\boldsymbol{A}} \circ \mathrm{NTT}(\boldsymbol{s}) + \mathrm{NTT}(\boldsymbol{e})$，私钥 $\mathbf{sk} = \mathrm{NTT}(\boldsymbol{s})$，其中 $\circ$ 表示多项式矩阵乘法。

2) 加密：生成 $\hat{\boldsymbol{A}} \in R_q^{k \times k}$，生成向量 $\boldsymbol{r}, \boldsymbol{e}_1 \in R_q^k$，多项式变量 $e_2 \in R_q$，对于信息 msg，有向量 $\boldsymbol{u} = (\mathrm{INTT}(\hat{\boldsymbol{A}}^{\mathrm{T}} \circ \mathrm{NTT}(\boldsymbol{r})) + \boldsymbol{e}_1$，多项式变量 $v = \mathrm{INTT}(\mathbf{pk}^{\mathrm{T}} \circ \mathrm{NTT}(\boldsymbol{r})) + e_2 + \mathrm{msg}$，其中 T 表示转置运算。

3) 解密：$\mathrm{msg} = v - \mathrm{INTT}(\mathbf{sk}^{\mathrm{T}} \circ \mathrm{NTT}(\boldsymbol{u}))$

从 Kyber 算法流程中可以看出，密钥生成过程需要 $2k$ 次 NTT 和 k^2 次 CWM 运算；加密过程需要 k 次 NTT、$k + k^2$ 次 CWM 和 $k + 1$ 次 INTT 运算；解密过程需要 k 次 NTT、k 次 CWM 和一次 INTT 运算。

1.2 硬件相关实现

目前，对于 Kyber 算法的研究主要集中在对 NTT 算法的优化加速，包括蝶形运算、存储访问和指令集的优化。

为此，Yaman 等[5]以 16 个蝶形运算单元加速了 NTT/INTT 的计算，但是其模约简计算较复杂，仍有优化空间。Huang 等[6]在 Xilinx Artix-7 和 Virtex-7 FPGA 上对 Kyber 算法进行了优化实现，并以流水线形式实现了 NTT 模块，但其 NTT 计算周期较长。Mert 等[7]以蒙哥马利模乘设计了蝶形运算单元，并可通过参数进行配置，以多运算单元（PE, processing element）并行完成 NTT 的计算。此外，Mert 等[8]又采用蒙哥马利模乘，以字为单位对 NTT 进行了优化，并通过 PCIe 以直接存储器访问（DMA, direct memory access）方式完成了软硬件协同实现。Xing 等[9]在 Xilinx Artix-7 上实现了完整的 Kyber 算法，包括 SHA3 和 NTT/INTT 的计算，但程序仍以串行执行为主，NTT 的计算需要 512 个时钟周期。Ricci 等[10]在 Xilinx Virtex UltraScale+ XCVU7P 高性能 FPGA 上实现了 Kyber 的 NTT 运算，频率达到了 637 MHz，但其计算周期仍高达 405 个时钟周期。同时，Ricci 等[11]又在该 FPGA 上实现了 CRYSTALS-Dilithium 数字签名算法，以 4 个蝶形运算单元每次输入 2 组数据加速了计算。Chen 等[12]采用双列顺序存储来解决 RAM 读写冲突的问题，并以流水线技术优化了蝶形运算。Seiler 等[13]以 AVX2 指令集优化了 NTT 算法，并在 Skylake 处理器上实现了 6.3 倍的性能提升。Zijlstra 等[14]以高层次综合（HLS, high-level synthesis）对 Kyber 算法进行优化实现，并对各种实现在资源和时间消耗方面

进行了详细对比。Basu 等[15]以 HLS 对多个后量子公钥加密算法和数字签名算法进行了实现，并在资源和性能间进行了对比分析。Agrawal 等[16]实现了寄存器传输级（RTL, register transfer level）代码库，以硬件原语的方式提供 n 个点的 NTT 运算。Bisheh-Niasar 等[17]采用 K^2-RED 模约简算法对 NTT 和 INTT 进行了优化，减少了资源占用，并提高了计算频率。但 K^2-RED 算法需要额外对 ω 进行预计算，且不支持 CWM 的运算。Fritzmann 等[18]在 FPGA 上设计了 RISQ-V 指令集，加速了 Kyber 算法的计算。

在国内，陈朝晖等[19]采用乒乓结构存储多项式系数，并以可选层级的流水线结构，减少了蝶形运算单元的逻辑资源占用。刘冬生等[20]在 Xilinx Artix-7 上设计了一种基于 RAM 的可重构多通道数论变换单元，支持不同参数的可重构配置。华斯亮等[21]设计实现了基 32 的数论变换硬件结构，以操作数合并和快速取模，提高了蝶形运算的计算效率。沈诗羽等[22]给出了一种针对我国 Aigis 后量子密码算法的高效 AVX2 和 ARM 优化方案，提升了算法整体性能。

综上可以看出，想要提高 Kyber 算法的 FPGA 实现速度，一是要对关键路径进行分解，深度优化，提高计算效率；二是要增加并行度，包括各模块间的并行和同时处理数据的能力。虽然上述方案在一定程度上加速了 Kyber 算法的计算，但大部分方案的 NTT 计算周期长，并行程度不高，仍无法满足高性能计算的需求。同时，部分方案仅实现了 NTT 和 INTT 运算，并不支持 CWM 运算。为此，本文从 Kyber 的 NTT、INTT 和 CWM 各计算阶段着手分析，针对不同的运算、存储和通信特征选择合适的实现方式，提高计算效率。

2　算法优化设计

2.1　整体结构

为提高 Kyber 算法硬件实现整体灵活性和可扩展性，在 FPGA 内放置 32 个蝶形运算单元，各个蝶形运算单元可独立执行。其次，放置多 RAM 通道对多项式系数和每次迭代的结果进行存取。对于参数 ω，采用预计算的方式提前计算好，并以 ROM 的形式进行存储。最后，由控制逻辑完成对蝶形运算单元的输入输出的控制、RAM 存取地址的控制及参数 ω 的读取控制。FPGA 整体结构如图 1 所示。

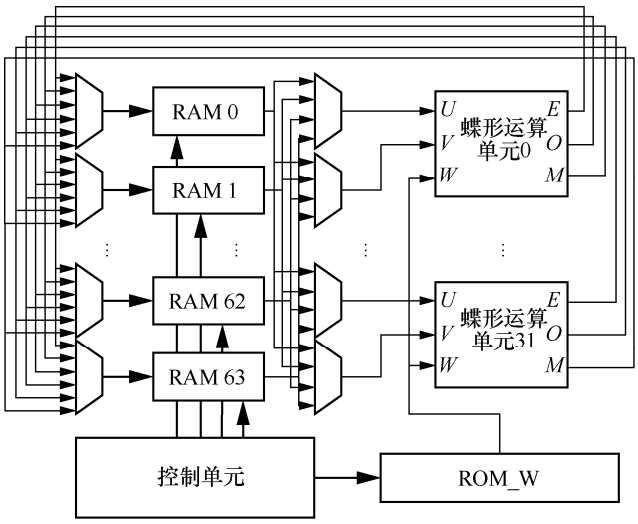

图 1　FPGA 整体结构

从 NTT、INTT 算法流程中可以看出，每次迭代读取和写入的参数是不同的。因此，需要通过仲裁对 RAM 进行读写。同时，在 CWM 算法中，需要多次计算模乘，因此在蝶形运算单元中以 M 输出中间模乘的结果，并再次送入蝶形运算单元进行最终结果的计算。可以看出，FPGA 整个架构以松耦合方式互连，并可根据参数配置，灵活完成 NTT、INTT 和 CWM 这 3 种运算。

2.2　蝶形运算单元

蝶形运算单元是 NTT、INTT 和 CWM 计算的核心。这里，采用流水线的结构进行实现，并根据控制信号，选择 NTT、INTT 和 CWM 不同计算的流程。蝶形运算单元结构如图 2 所示。

由于 NTT 先计算模乘，再计算模加减，而 INTT 先计算模加减再计算模乘，两者计算的先后顺序不同。因此在蝶形运算单元中插入了多个缓存寄存器，用来缓存中间结果，并根据控制信号，进行仲裁输出。同时，通过缓存也降低了蝶形运算的逻辑层级，避免产生较大的路径时延。其次，NTT/INTT 每次存取参数的 RAM 不一致，且输出的 E 和 O 这 2 个结果可能存入同一 RAM 中，因此需要 E 和 O 按次序写入。具体地，在 NTT/INTT 中，输出 E 需要 3 个时钟周期，输出 O 需要 4 个时钟周期，并按先 E 后 O 写入 RAM。再次，对于 CWM 算法，需要多次计算模乘和模加，由此需要对中间模乘结果 M 进行缓存，并以状态机控制传值完成后续计算。最后，使用 DSP 实现乘法器，充分利用 FPGA 芯片资源。蝶形运算单元流水线各阶段如图 3 所示。

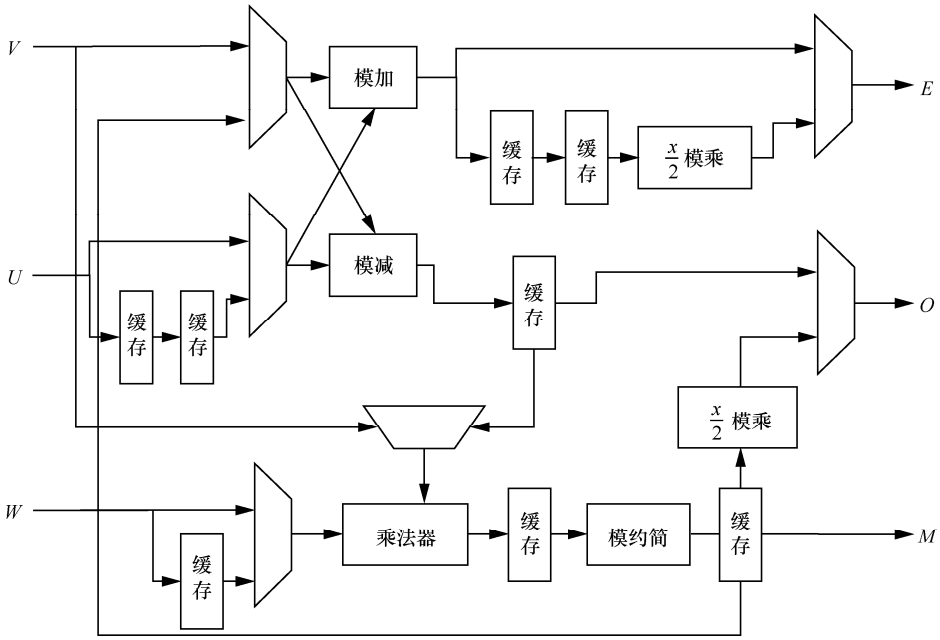

图 2　蝶形运算单元

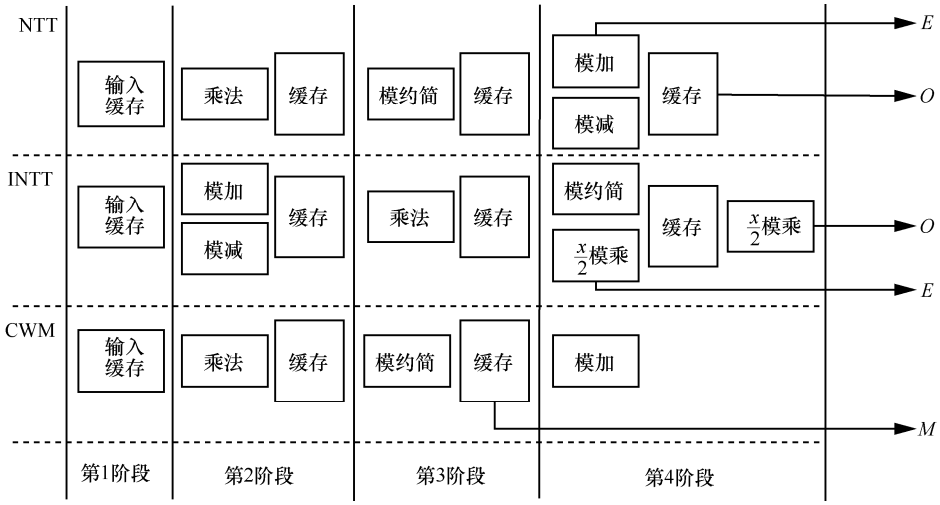

图 3　流水线阶段示意

在蝶形运算单元中，模加减可由组合逻辑进行实现，并根据加减结果是否超出素数 q（q=3 329）的范围，再进行处理判断，其结构如图 4 和图 5 所示。

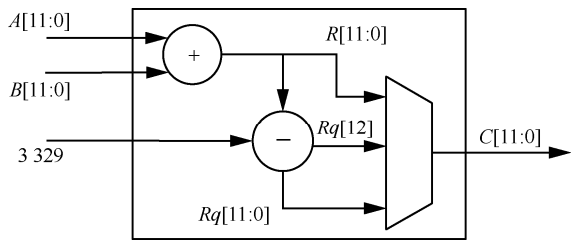

图 4　模加硬件结构

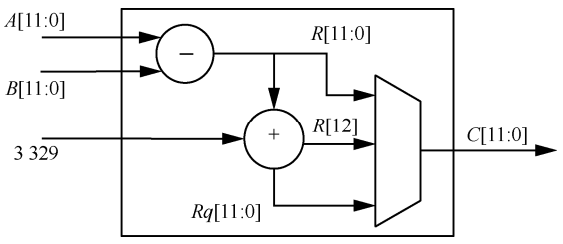

图 5　模减硬件结构

在图 4 和图 5 中，A 和 B 为 12 bit 的输入，C 为 12 bit 的输出，R 和 Rq 为 13 bit 的中间计算结果，并根据 R 或 Rq 的最高位进行判断输出最终的模加

减结果。而模约简和 $\frac{x}{2}$ 模乘需要根据 Kyber 算法参数进行深度优化，以减少资源消耗并提高运算速度。具体地，本文通过 Barrett 算法实现模约简，再以逻辑公式变换优化 $\frac{x}{2}$ 模乘，详细介绍如下。

2.2.1 Barrett 模约简

Barrett 算法利用乘法和模约简运算代替了高成本的除法来实现取模运算，可计算任何参数的模乘。对于 $0 \leqslant x < q$，$0 \leqslant y < q$，则 $xy < q^2$。为计算 $\mathrm{zm} = xy \bmod q$，$0 \leqslant \mathrm{zm} < q$，令 $\mathrm{sp} = xy$，如果存在 wa 使 $\mathrm{sp} = q\mathrm{wa} + \mathrm{zm}$，即可求得 zm。

Barrett 算法首先计算 wa 的近似值 wa′，令 $\mu = \left\lfloor 2^{2n} / q \right\rfloor$，$\mathrm{wa}' = \left\lfloor \mathrm{sp} / q \right\rfloor$，则 $\mathrm{wa}' = \left\lfloor \mathrm{sp} / q \right\rfloor \approx \left\lfloor \mu \mathrm{sp} / 2^{2n} \right\rfloor$，其中 $n = \lceil \mathrm{lb} q \rceil$。在 Kyber 算法中，$q$ 取常数 3 329，则有 $n = 12$，$\mu = \left\lfloor 2^{24} / 3329 \right\rfloor = 5\,039 = 2^{12} + 2^{10} - 2^6 - 2^4 - 1$。另外 $q = 2^{12} - 2^9 - 2^8 + 1$，则有 $2^{12} \equiv (2^9 + 2^8 - 1) \bmod 3\,329$。那么，有 $\mathrm{tp} = 2^{12}\mathrm{sp}[23:12] + \mathrm{sp}[11:0]$，则 $\mathrm{wa}' = \left\lfloor \mu / 2^{24} \mathrm{sp} \right\rfloor \approx (\mathrm{tp}\ \mu) / 2^{24} \approx \mathrm{sp}[23:12] + \mathrm{sp}[23:14] - \mathrm{sp}[23:18] - \mathrm{sp}[23:20]$。经分析 $\left\lfloor \mathrm{sp} / q \right\rfloor - 3 \leqslant \mathrm{wa}' \leqslant \left\lfloor \mathrm{sp} / q \right\rfloor + 1$，wa′的位宽为 13 bit。对于 $\mathrm{sp} - q\mathrm{wa}'$ 的差值，其范围为 $-2^{14} < \mathrm{sp} - q\mathrm{wa}' < 2^{13}$。再者 $q = 2^{11} + 2^{10} + 2^8 + 1$，则有 $q\mathrm{wa}' = (\mathrm{wa}'[12:0] << 11 + \mathrm{wa}'[12:0] << 10 + \mathrm{wa}'[12:0] << 8 + \mathrm{wa}'[12:0])$。那么可以取该结果的前 14 位来简化 $\mathrm{sp} - q\mathrm{wa}'$ 的计算。

具体地，Barrett 模约简如算法 4 所示。

算法 4　Barrett 模约简

输入　sp, q

输出　$\mathrm{zm} = \mathrm{sp} \bmod q$

1) $\mathrm{sum} = \mathrm{sp}[23:12] + \mathrm{sp}[23:14] - \mathrm{sp}[23:18] - \mathrm{sp}[23:20] \approx \left\lfloor \mu / 2^{24} \mathrm{sp} \right\rfloor$
2) $\mathrm{dif} = \mathrm{sp}[13:0] - (\mathrm{sum} + \mathrm{sum}[2:0] << 11 + \mathrm{sum}[3:0] << 10 + \mathrm{sum}[5:0] << 8)$
3) switch(dif[13:12])
4) case 0: $q_{\mathrm{mux}} = 0$
5) case 1: $q_{\mathrm{mux}} = -q$
6) case 2: $q_{\mathrm{mux}} = 3q$
7) case 3: $q_{\mathrm{mux}} = 2q$
8) end switch
9) $\mathrm{zm} = \mathrm{dif} + q_{\mathrm{mux}}$
10) if $\mathrm{zm} > q$ then $\mathrm{zm} = \mathrm{zm} - q$
11) end if
12) return zm

对于结果 dif，其取值范围为$[-3q, 2q)$，因此需要再根据 dif 高 2 位进行判断处理。最后，将 zm 与 q 进行比较判断，并计算输出最终结果。

2.2.2 $\frac{x}{2}$ 模乘优化

在 INTT 计算过程中，需要计算 $\frac{x}{2} \bmod q$，Zhang 等[23]提出的方法可避免模乘运算。具体地，对于奇素数 q，$\frac{x}{2} \bmod q$ 可以采用式(2)进行计算。

$$\frac{x}{2} \bmod q = (x >> 1) + x[0] \,\&\, \left(\frac{q+1}{2} \right) \tag{2}$$

由于 q 为常数，这样，对于 INTT 算法中 $\frac{E}{2} \bmod q$ 和 $\frac{O}{2} \bmod q$ 的计算，可通过向右移位、加法和逻辑与进行实现，简化了 x 乘 $\frac{1}{2}$ 的求模运算。

2.2.3 CWM 调度优化

对于 CWM，需要在 $\mathbb{Z}_q[x] / (x^2 - \omega)$ 上计算二次多项式乘法 $(a_0 + a_1 x)(b_0 + b_1 x)$。在算法 3 中，可以看到需要分别计算 a_0b_1、a_1b_0、a_1b_1W 和 a_0b_0，共 5 次模乘和 2 次模加。对于模乘需要 3 个时钟进行计算，而模加只要需要一个时钟周期。由此，在第 1 个时钟周期先计算 a_1b_0，并在第 4 个时钟周期将 a_1b_0 缓存的结果与 a_0b_1 共同输入蝶形运算单元，在 a_0b_1 做完模乘运算后，直接完成 $a_1b_0 + a_0b_1$ 的模加运算。同理，先计算 a_1b_1，再计算 a_0b_0。如此，不仅减少了 CWM 中间等待的过程，并可在 8 个时钟周期内完成二次多项式乘法，还提高了 CWM 的计算效率。具体地，CWM 中二次多项式调用流程如图 6 所示。

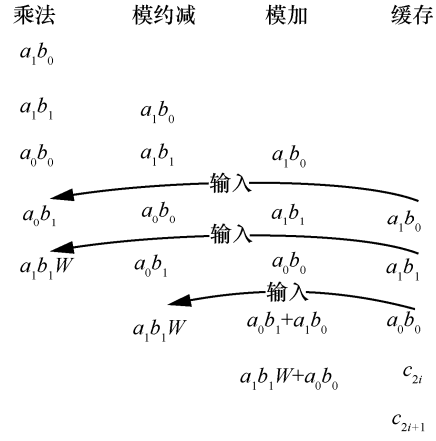

图 6　CWM 中二次多项式乘法调用流程

2.3 多 RAM 访存优化

对于 N 个点的 NTT，共需要执行 lb$N$ 轮蝶形运算，且每轮对 RAM 的访问需要唯一的地址来存取参数。8 个点的蝶形运算和 RAM 访问如图 7 所示，灰色节点表示蝶形运算。

从图 7(a)中可以看出，在第 1 阶段，4 组系数对(0,4)、(1,5)、(2,6)和(3,7)会分别进行蝶形运算。因此需要在一个时钟周期内，从 RAM 中读取各组系数对。那么，可以采用 2 个 RAM（RAM0 和 RAM1）分别存储第 0、1、2、3 和 4、5、6、7 个系数进行实现。此外，由于每个阶段的系数对都会发生变化。在第 2 个阶段，参与蝶形运算的 4 组系数对为(0,2)、(1,3)、(4,6)和(5,7)。因此需要将第 1 阶段(0, 4)和(1, 5)的输出结果存入 RAM0 中，(2, 6)和(3, 7)的结果存入 RAM1 中。如此，可以确保在第 2 阶段，仍在一个时钟周期内从 RAM0 和 RAM1 中读取到相应的系数对。同理，对于第 3 阶段的计算，也需要调整第 2 阶段 RAM 的存储。

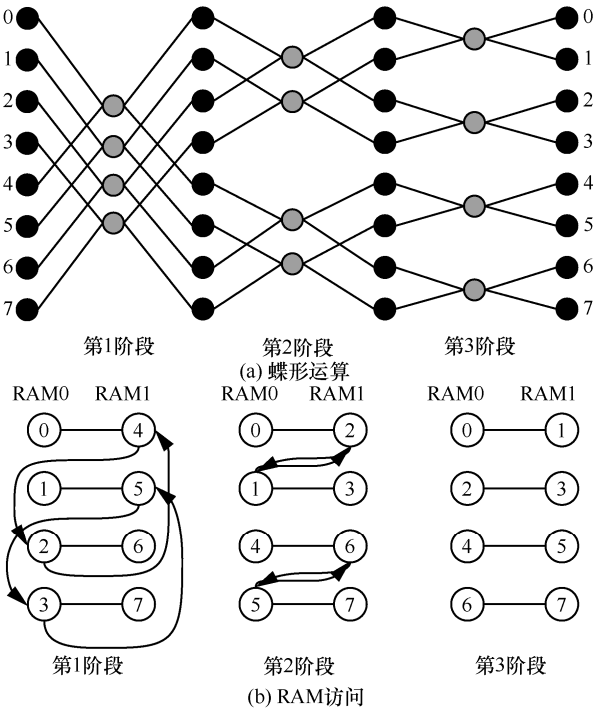

图 7 8 个点的蝶形运算和 RAM 访问

对于 Kyber 算法，多项式有 256 个系数，由于将其拆分为 2 个 128 次多项式，则共需要执行 7 轮蝶形运算，对 RAM 的存取更为复杂。以下给出 Kyber 算法的访存优化方法。

2.3.1 多 RAM 存储

首先，本文方案共有 32 个蝶形运算单元，则需要采用 64 个 RAM，每个 RAM 对应有 4 个地址，共有 64×4 = 256 个参数。RAM 初始化参数存储如图 8 所示。

	RAM0	RAM1	RAM2	RAM3	…	RAM62	RAM63
地址0	0	128	1	129		31	159
地址1	32	160	33	161		63	192
地址2	64	192	65	193		95	223
地址3	96	224	97	225		127	255

图 8 RAM 初始化参数存储

图 8 中，每 2 个 RAM 对应一个蝶形运算单元，且第 $2i$ 个和第 $2i+1$ 个（$0 \leqslant i < 32, i = i + 2$）RAM 分别存取各自的系数对。如此，32 个蝶形运算单元可独立并行读取参数并执行。

其次，由于采用 64 个 RAM 对中间结果进行存取，需要确保 RAM 地址的读写顺序，防止读写冲突。为此，对 RAM 进行了扩容，将 RAM 分为高低 2 个地址段。以 NTT 为例，在首轮参数存入 RAM 后，从地址 0～3 开始读取数据，并将蝶形运算的结果写入地址 4～7。而后再从地址 4～7 读取数据，并将蝶形运算的结果写入地址 0～3。如此循环，以低地址空间和高地址空间的交替迭代操作，实现了每轮参数的有序存取。RAM 存取模式如图 9 所示。

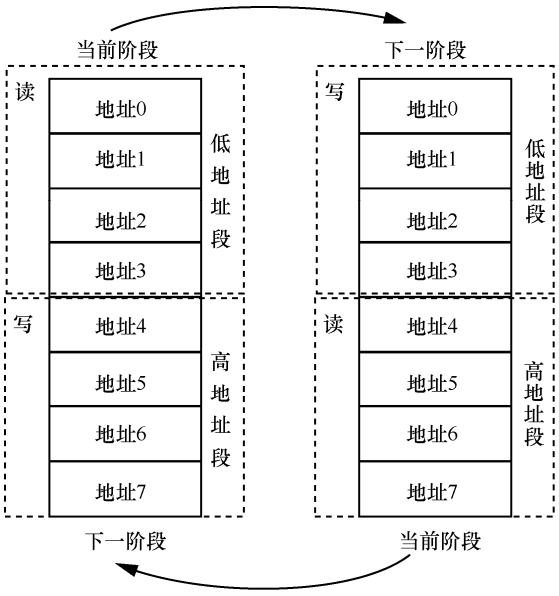

图 9 RAM 存取模式

最后，对于 ω 的读取，由于每轮蝶形运算输入的参数不同，需要预处理后，按一定的规则进行存储。具体地，对于 NTT，从地址 0 到 22；对于 INTT，从

地址 23 到 45；对于 CWM，从地址 46 到 49，每个地址分别有 32 个数据，存入 ROM 中。以 NTT 运算为例，其 ROM 存储内容如图 10 所示。

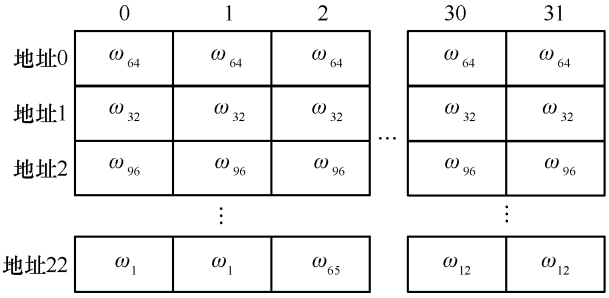

图 10　NTT 中 ω 的 ROM 存储

2.3.2　数据存取控制

Kyber 每轮蝶形运算读写的参数不同，且地址唯一。在读取参数时，为方便操作，对参数进行重排，统一读取 64 个 RAM 的某一地址，获取所有 256 个参数并组成系数对。因此，在写入中间结果时，需要对输出的 E 和 O 分别按不同的地址进行存储，以保证在下一轮可以从同一个地址再次读取到对应的系数对。具体地，这里采用一个地址 raddr 对所有 RAM 进行读取，2 个地址 waddre 和 waddro 分别完成 E 和 O 的 RAM 写入。

另外，由于 RAM 空间被分为低地址空间和高地址空间，每轮交替使用高地址空间或低地址空间。因此需要对 raddr、waddre 和 waddro 的高位进行交替翻转，低位则按照每轮读写的地址正常进行变化。以 NTT 运算为例，具体的 raddr、waddre 和 waddro 计算如算法 5 和算法 6 所示。

算法 5　RAM 读地址的变换

输入　蝶形运算阶段 bau_stage[2:0], 循环次数 bau_loop[1:0]

输出　RAM 读地址 raddr[2:0]

初始化　raddr[2:0]=0

1) if(bau_stage > 1) then raddr[1:0] = bau_loop
2) else raddr[1:0] = ((bau_loop>>1) & ((1<<(1−bau_stage))−1)) + (bau_loop [0] << (1−bau_stage)) + ((bau_loop>>(2−bau_stage))<<(2− bau_stage))
3) end if
4) if(raddr[1:0]==3) then raddr[2] = ～raddr[2]
5) else raddr[2] = raddr[2]
6) end if
7) return raddr[2:0]

在算法 5 的步骤 1)中，当 bau_stage 大于 1 时，直接根据 bau_loop 即可获取 RAM 的读地址。对于步骤 2)，则需要根据当前 bau_stage 和 bau_loop 进行额外的计算，从而获得 RAM 的读地址。最后，在步骤 4)～步骤 6)，当 raddr[1:0]由 0 累加到 3 时，raddr[2]会进行翻转，从而使 raddr[2:0]由 4 到 7 进行地址累加。

算法 6　RAM 写地址的变换

输入　蝶形运算阶段 bau_stage[2:0], 循环次数 bau_loop[1:0]

输出　RAM 写地址 waddre[2:0], waddro[2:0]

初始化　waddre[2:0] = 0, waddro[2:0] = 0

1) if(bau_stage > 1) then waddre[1:0] = bau_loop; waddro[1:0] = bau_loop
2) else waddre[1:0] = ((bau_loop>>1) + ((bau_loop>>(2−bau_stage)) << (1−bau_stage))
3) waddro[1:0] = ((bau_loop>>1) + ((bau_loop>>(2−bau_stage)) << (1−bau_stage)) + (1 << (1−bau_stage))
4) end if
5) if(waddro[1:0]==3) then waddre[2] = ～waddre[2]; waddro[2] = ～waddro[2]
6) else waddre[2] = waddre[2]; waddro[2] = waddro[2]
7) end if
8) return raddr[2:0]

同理，在算法 6 中，当 bau_stage 大于 1 时，可直接根据 bau_loop 获取 RAM 的写地址，而其他情况要根据当前 bau_stage 和 bau_loop 进行计算，从而获得 RAM 的写地址。最后，在步骤 5)～步骤 7)，对 waddre[2]和 waddro[2]高位进行翻转。

由于 RAM 的读写需要先获取地址，然后才能读写数据，即 RAM 的操作需要一定的时间间隔。而蝶形运算是以流水线方式执行的，可连续对数据进行处理。因此需要处理好流水线和 RAM 访问之间的关系，防止数据中断出错。

如图 11 所示，将数据送入流水线时，预先计算出 RAM 读写地址，并对地址进行打拍缓存。具体地，在蝶形运算的前 2 轮，中间需要等待 2～3 个时钟周期。这是由于前 2 轮，地址不是按序读写的，需要额外的等待。在第 3～7 轮，中间只需要等待一个时钟周期，即在当前 RAM 地址写入后，下一个时钟可立即读取。

2.3.3　RAM 资源复用

Kyber 算法中需要输入 2 个多项式 $A(x)$和

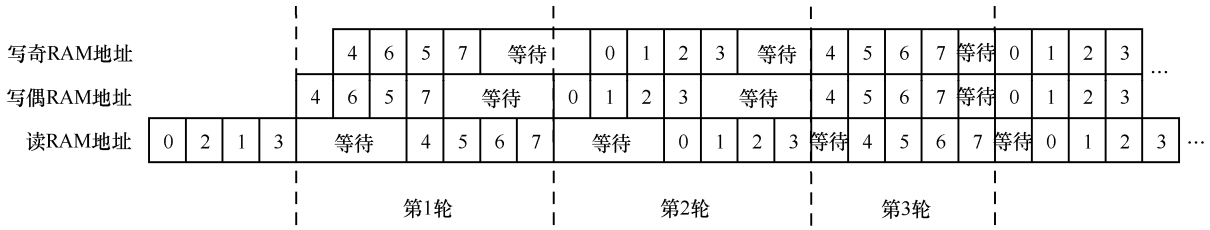

图 11　RAM 读写等待示意

$B(x)$，为此通过 2 组 RAM 分别存储，如图 12 所示。这样可一次将 $A(x)$和 $B(x)$的所需参数写入 RAM。然后在 CWM 计算过程中，可直接从 2 组 RAM 中读取数据，并将多项式乘法结果再次存入其中一组 RAM 中。随后，再由控制信号完成 INNT 运算，并写入其中一组 RAM 中。通过 2 组 RAM 的复用，不仅减少了外部通信次数，同时减少了 FPGA 片上 RAM 资源的消耗。

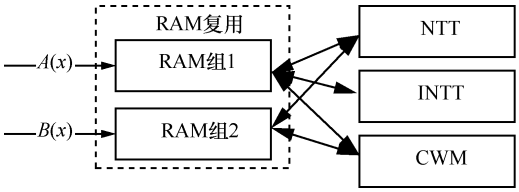

图 12　RAM 资源复用

2.4　通信调度

通过 DMA 的方式，以 AXI FIFO 实现数据的传输，如图 13 所示。由于 RAM 组入口数据位宽为 384 bit，而 FIFO 传输数据位宽为 64 bit，因此需要对位宽进行转换。同时，通过增加的 FIFO 和位宽转换模块，使算法的实现与数据之间进行解耦合，各模块操作的端口相互独立，最大化地减少各模块之间的相关性，从而降低布线路由的复杂度。然后，以 AXI BRAM 实现控制信号的传输，并在解析后完成对 Kyber 蝶形运算单元阵列的调度控制。

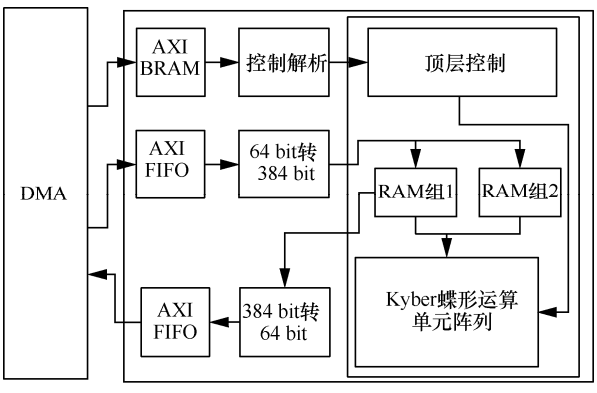

图 13　DMA 通信调度

具体地，Kyber 算法的通信调度流程如下。

1) 通过 DMA 对 RAM 组 1 和 RAM 组 2 进行参数配置；

2) 由 NTT 使能信号，按先 $A(x)$再 $B(x)$进行 NTT 计算，并将结果保存在对应的 RAM 组 1 和 RAM 组 2 中；

3) 由 CWM 使能信号，从 RAM 组 1 和 RAM 组 2 中读取数据，完成 CWM 计算，并将结果保存在 RAM 组 1 中；

4) 再由控制信号完成 INNT 运算，并写入 RAM 组 1；

5) 最后，将结果由 DMA 传出。

3　实验结果与分析

3.1　实验环境

本文实验的硬件平台为 FPGA 加速卡，芯片型号为 Xilinx 公司的 xc7z035ffg676-2，其查找表（LUT, look up table）资源是 171 900，触发器（FF, flip flop）资源是 343 800，块 RAM（BRAM, block RAM）存储器资源是 500，数字信号处理器（DSP, digital signal processing）资源是 900。软件平台为集设计、仿真、综合、布线、生成于一体的 Vivado 2019.2 软件。

首先，实验通过对算法的各模块进行深度优化，给出了综合布线后的资源占用情况。其次，通过仿真分析了各模块计算周期，并在可运行的最高频率下，给出了硬件的具体性能和能效比。最后，与其他 Kyber 算法设计方案进行了综合对比，并对结果进行了分析说明。

3.2　各模块实现

Kyber 算法中各模块资源占用如表 2 所示。其中蝶形运算单元以流水线方式实现；控制单元以串行方式实现；多 RAM 通道以并行方式实现；ROM 存储以串行方式实现。Kyber 算法通过串并混合设计，提高整体计算性能。

表 2　Kyber 算法各模块资源占用

模块	实现方式	LUT/个	FF/个	BRAM/个	DSP/个
蝶形运算单元	流水线	235	108	0	1
控制单元	串行	5 642	159	0	0
多 RAM 通道	并行	0	0	64	0
ROM 存储	串行	0	0	5	0
Kyber 算法	串并混合	14 634	5710	69	32

从表 2 中可以看出，蝶形运算单元占用资源较少，且仅消耗了一个 DSP 即可完成乘法运算。而控制单元消耗了更多的 LUT 资源，这是因为需要计算各种计数、地址及控制信号使能，包括蝶形运算轮次、本轮迭代次数、RAM 和 ROM 的读写地址、读写使能信号及等待控制等。其次，对于多 RAM 通道和 ROM 存储，需要存储 Kyber 算法计算过程中的全部参数，仅消耗了 FPGA 部分 BRAM 资源。最后，对于 Kyber 整体实现，通过 Vivado 软件综合布线后，LUT 资源占用比例为 8.51%，FF 占用为 1.66%，BRAM 占用为 13.80%，DSP 占用为 3.56%，资源消耗适中。

3.3　性能分析

描述 Kyber 算法硬件平台的指标有性能、功率、能效比。其中性能指硬件平台单位时间内的密码计算速率，简记为 speed。功率用于计算硬件设备工作时的能耗，简记为 power。能效比表示平台性能与设备功率之比，简记为 eer，计算式如下

$$\mathrm{eer}=\frac{\mathrm{speed}}{\mathrm{power}} \tag{3}$$

Kyber 算法实现的各项硬件性能指标如表 3 所示。其中，能效比计算对应的芯片功耗为 1.088 W。从表 3 中可以看出，利用 FPGA 实现的 Kyber 算法不仅性能达到了每秒百万级别的计算速度，且功耗较低，具有很高的能效比，适合在物联网、云计算、高性能计算等多种场景中使用。

表 3　Kyber 算法的各项硬件性能指标

模块	频率/MHz	计算周期	性能/(p·s^{-1})	能效比/(p·J^{-1})
NTT	175	44	3 977 273	3 655 582
INTT	175	49	3 571 429	3 282 563
CWM	175	163	1 073 620	986 783

3.4　与其他方案对比

电路面积与运算时间的乘积 AT[17,19]客观反映了资源消耗和算法性能之间的关系。AT 值越低，表明在有限的资源条件下，与性能之间取得了更好的平衡。为了方便与其他方案对比，所有硬件实现均用 AT 值进行归一化处理。

具体地，本文采用的 AT 值计算式如下

$$\mathrm{AT}=\mathrm{LUT}\times\text{时间} \tag{4}$$

其中，时间为执行一次 NTT 运算所消耗的时间，即时间 ＝NTT 周期 / 频率。

本文方案与其他 Kyber 算法的 FPGA 实现方案在 NTT 周期、频率、资源消耗和 AT 值等方面综合对比如表 4 所示。

在表 4 中，本文方案具有最短的计算周期。这是由于本文采用 32 个蝶形运算单元并行执行，且工作在较高的频率，缩短了整体计算周期。同时，合理利用了芯片的 DSP 和 RAM 资源。在 AT 值方面本文方案优于大部分方案，但差于文献[17]文案和文献[10]文案。文献[5]文案采用的是 16 个蝶形运算单元，缩短了计算周期，但其模约简计算稍复杂，且消耗了更多的 LUT 资源，AT 值稍高。文献[9]方案在 FPGA 内放置了 2 个蝶形运算单元，并支持 CWM 运算，但程序仍以串行执行为主，导致 NTT

表 4　与其他 FPGA 实现方案的综合对比

方案	器件	NTT 周期	频率/MHz	LUT/个	FF/个	DSP/个	BRAM/个	时间/μs	AT
文献[5]	Artix-7	69	172	9 508	2 684	16	35	0.40	3 803
文献[9]	Artix-7	512	161	1 737	1 167	2	3	3.18	5 524
文献[10]	UltraScale+ XCVU7P	405	637	1 107	1 407	28	3.5	0.64	708
文献[12]	Artix-7	2 055	136	442	237	1	1.5	15.11	6 679
文献[17]	Artix-7	324	222	801	717	4	2	1.46	1 169
本文方案	Zynq-7035	44	175	8 428	3 979	32	11	0.25	2 107

计算周期长，AT 值较高。文献[10]方案使用的是 Xilinx 高端芯片，采用全新架构，资源密度是本文 Zynq-7035 的 6.37 倍，其工作频率也是所有方案中最高的。该方案放置了 4 个蝶形运算单元，并消耗了 28 个 DSP 完成模乘运算。由于其 DSP 工作频率高，降低了模乘运算时间，AT 值也是最低的。但该方案的 NTT 和 INTT 各自独立实现，其 INTT 还需要额外消耗 60 个 DSP，逻辑复用率低，占用了更多的资源。文献[12]方案以运算逻辑复用实现各步操作，占用资源最少，但导致并行程度低，NTT 计算周期长，AT 值最高。文献[17]方案虽然具有较好的 AT 值，占用资源少，但它采用了 K^2-RED 模约简算法，通过预计算的方式仅能实现 NTT 和 INTT 运算，无法直接实现 CWM 运算。

总体而言，本文提出的 Kyber 算法的 FPGA 多路并行优化实现，至少缩短了 36.23%的 NTT 计算周期，并缩短了 37.50%计算时间，在保证较高的工作频率下，充分利用了芯片逻辑资源，具有较优的 AT 值。

4 结束语

基于格密码方案对于未来后量子密码标准，乃至对未来信息系统的安全都有至关重要的作用。本文通过对 Kyber 格密码的描述，首先，深入剖析了 NTT、INTT 及 CWM 算法；然后，采用流水线技术实现了蝶形运算单元的优化，并以多通道 RAM 优化参数存取，降低整体计算时延；最后，以 32 个蝶形运算单元并行计算，实现了 FPGA 整体架构，提高了计算效率。显然，本文实现方案具有很高的研究和实际应用价值。

在实际应用中，侧信道分析攻击仍是密码攻击的主要手段。因此，在硬件层面如何提供 Kyber 格密码的安全防护策略，显得至关重要。这不仅增加了额外的资源消耗，且提高了设计难度，仍需要探索解决，也是未来的研究工作。

参考文献：

[1] DANG V, FARAHMAND F, ANDRZEJCZAK M, et al. Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches[J]. IACR Cryptol EPrint Arch, 2020, 2020: 795.

[2] AVANZI R, BOS J, DUCAS L, et al. CRYSTALS-Kyber[R]. 2017.

[3] LYUBASHEVSKY V, SEILER G. NTTRU: truly fast NTRU using NTT[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019, 2019(3): 180-201.

[4] ZHANG N, QIN Q, YUAN H, et al. NTTU: an area-efficient low-power NTT-uncoupled architecture for NTT-based multiplication[J]. IEEE Transactions on Computers, 2020, 69(4): 520-533.

[5] YAMAN F, MERT A C, ÖZTÜRK E, et al. A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQC scheme[C]//Proceedings of 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). Piscataway: IEEE Press, 2021: 1020-1025.

[6] HUANG Y M, HUANG M Q, LEI Z K, et al. A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse[J]. IEICE Electronics Express, 2020, 17(17): 1-6.

[7] MERT A C, KARABULUT E, OZTURK E, et al. An extensive study of flexible design methods for the number theoretic transform[J]. IEEE Transactions on Computers, 2020: doi.org/10.1109/TC.2020.3017930.

[8] MERT A C, ÖZTÜRK E, SAVAŞ E. Design and implementation of a fast and scalable NTT-based polynomial multiplier architecture[C]//Proceedings of 2019 22nd Euromicro Conference on Digital System Design (DSD). Piscataway: IEEE Press, 2019: 253-260.

[9] XING Y F, LI S G. A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-Kyber on FPGA[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021, 2021(2): 328-356.

[10] RICCI S, JEDLICKA P, CIBIK P, et al. Towards CRYSTALS-Kyber VHDL implementation[C]//Proceedings of the 18th International Conference on Security and Cryptography. [S.l.]: Science and Technology Publications, 2021: 760-765.

[11] RICCI S, MALINA L, JEDLICKA P, et al. Implementing CRYSTALS-dilithium signature scheme on FPGAs[C]//Proceedings of 16th International Conference on Availability, Reliability and Security. New York: ACM Press, 2021: 1-11.

[12] CHEN Z H, MA Y, CHEN T Y, et al. Towards efficient kyber on FPGAs: a processor for vector of polynomials[C]//Proceedings of 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). Piscataway: IEEE Press, 2020: 247-252.

[13] SEILER G. Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography[J]. IACR Cryptology EPrint Archive, 2018, 2018: 39.

[14] ZIJLSTRA T, BIGOU K, TISSERAND A. Lattice-based cryptosystems on FPGA: parallelization and comparison using HLS[J]. IEEE Transactions on Computers, 2021: doi.org/10.1109/TC.2021.3112052.

[15] BASU K, SONI D, NABEEL M, et al. NIST post-quantum cryptography-a hardware evaluation study[R]. 2019.

[16] AGRAWAL R, BU L K, EHRET A, et al. Open-source FPGA implementation of post-quantum cryptographic hardware primitives[C]//Proceedings of 2019 29th International Conference on Field Programmable Logic and Applications (FPL). Piscataway: IEEE Press, 2019: 211-217.

[17] BISHEH-NIASAR M, AZARDERAKHSH R, MOZAFFARI-KERMANI M. High-speed NTT-based polynomial multiplication

accelerator for post-quantum cryptography[C]//Proceedings of 2021 IEEE 28th Symposium on Computer Arithmetic (ARITH). Piscataway: IEEE Press, 2021: 94-101.

[18] FRITZMANN T, SIGL G, SEPÚLVEDA J. RISQ-V: tightly coupled RISC-V accelerators for post-quantum cryptography[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 239-280.

[19] 陈朝晖, 马原, 荆继武. 格密码关键运算模块的硬件实现优化与评估[J]. 北京大学学报(自然科学版), 2021, 57(4): 595-604.
CHEN Z H, MA Y, JING J W. Hardware optimization and evaluation for crucial modules of lattice-based cryptography[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2021, 57(4): 595-604.

[20] 刘冬生, 赵文定, 刘子龙, 等. 应用于格密码的可重构多通道数论变换硬件设计[J]. 电子与信息学报, 2021: doi.org/10.11999/ JEIT210114.
LIU D S, ZHAO W D, LIU Z L, et al. Reconfigurable hardware design of multi-lanes number theoretic transform for lattice-based cryptography[J]. Journal of Electronics & Information Technology, 2021: doi.org/10.11999/JEIT210114.

[21] 华斯亮, 张惠国, 王书昶. 用于全同态加密的数论变换乘法蝶形运算优化及实现[J]. 电子与信息学报, 2021, 43(5): 1381-1388.
HUA S L, ZHANG H G, WANG S C. Optimization and implementation of number theoretical transform multiplier butterfly operation for fully homomorphic encryption[J]. Journal of Electronics & Information Technology, 2021, 43(5): 1381-1388.

[22] 沈诗羽, 何峰, 赵运磊. Aigis 密钥封装算法多平台高效实现与优化[J]. 计算机研究与发展, 2021, 58(10): 2238-2252.
SHEN S Y, HE F, ZHAO Y L. Multi-platform efficient implementation and optimization of aigis-enc algorithm[J]. Journal of Computer Research and Development, 2021, 58(10): 2238-2252.

[23] ZHANG N, YANG B H, CHEN C, et al. Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2020(2): 49-72.

[作者简介]

李斌（1986− ），男，河南郑州人，博士，郑州大学讲师，主要研究方向为信息安全、可重构计算。

陈晓杰（1993− ），男，河南武陟人，信息工程大学博士生，主要研究方向为信息安全、可重构计算。

冯峰（1990− ），男，河南新乡人，郑州大学博士生，主要研究方向为信息安全。

周清雷（1962− ），男，河南新乡人，博士，郑州大学教授，主要研究方向为信息安全、自动机理论和计算复杂性理论。