面向格密码的可配置基-4 NTT硬件优化与实现

周清雷[1,2]，韩贺茹[1,2]，李斌[1,2]，刘宇航[1,2]
（1.郑州大学计算机与人工智能学院，河南 郑州 450001；2.河南省网络密码技术重点实验室，河南 郑州 450001）

摘　要：针对数论变换（NTT）优化格密码算法中的多项式乘法问题，以及NTT设计趋向多应用场景的需求，提出了一种面向格密码的可配置基-4 NTT硬件结构。通过分析基-4 NTT/INTT（Inverse NTT）算法流程，设计了高效的现场可编程门阵列（FPGA）整体结构。该结构具备参数化的运行时可配置性和满足多样化需求的编译时可配置性，以流水线方式构建基-4 NTT统一化蝶形单元，并对模除、模约减等关键模块进行优化，提高了计算效率和可配置性。此外，提出可配置多RAM存储优化设计方案及数据存储分配算法，以避免内存冲突、提高数据访问效率。与相关方案的对比及分析表明，以Dilithium算法为例，所提方案不仅具有较高的工作频率，还实现了面积上高达54.3%的优化和吞吐量高达2倍的提升，能够充分发挥FPGA的计算优势。

关键词：数论变换；格密码；多项式乘法；现场可编程门阵列；蝶形单元
中图分类号：TP309
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024188

Configurable radix-4 NTT hardware optimization and implementation for lattice-based cryptography

ZHOU Qinglei[1,2], HAN Heru[1,2], LI Bin[1,2], LIU Yuhang[1,2]
1. School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China
2. Henan Province Key Laboratory of Network Cryptography Technology, Zhengzhou 450001, China

Abstract: In response to the complex polynomial multiplication issue in lattice-based cryptography algorithms optimized with number theoretic transform (NTT), as well as the demand for NTT designs catering to multiple application scenarios, a configurable radix-4 NTT hardware architecture for lattice-based cryptography was proposed. By analyzing the radix-4 NTT/INTT (inverse NTT) algorithm process, an efficient FPGA architecture was designed, which parameterized runtime configurability and offered compile-time configurability to meet diverse requirements, a pipeline approach was used to construct the radix-4 NTT unified butterfly unit, key algorithmic modules such as modular division and modular reduction were deeply optimized, thereby enhancing computational efficiency and reconfigurability. Additionally, a configurable multi-RAM storage optimization design scheme and data storage allocation algorithm were proposed to avoid memory conflicts and improve data access efficiency. Comparison and analysis with related approaches show that, using the Dilithium algorithm as an example, the proposed design not only achieves a high operational frequency but also achieves up to 54.3% improvement in area and 2 times optimization in throughput, fully leveraging the computational advantages of FPGA.

Keywords: NTT, lattice-based cryptography, polynomial multiplication, FPGA, butterfly unit

收稿日期：2024–05–15；修回日期：2024–10–08
通信作者：李斌，iebinli@zzu.edu.cn
基金项目：河南省科技攻关基金资助项目（No.232102211055）；河南省网络密码技术重点实验室研究课题基金资助项目（No.LNCT2022-A14）；河南省重大科技专项基金资助项目（No.221100210600）
Foundation Items：Henan Province Science and Technology Key Project (No.232102211055), Research Topic of Henan Province Key Laboratory of Cybersecurity and Cryptology (No.LNCT2022-A14), Major Science and Technology Special Project of Henan Province (No.221100210600)

0 引言

近年来，随着量子计算机方面的稳步推进，传统经典的密码学方案被量子计算机攻破[1]，许多现代加密系统的安全将会受到威胁，尤其是因式分解、离散对数以及椭圆曲线密码学的密钥建立方案和数字签名将受到严重的影响[2]。后量子密码学（PQC, post-quantum cryptography）是在假设攻击者拥有大型量子计算机情况下进行的密码学研究[3]。目前，PQC方案主要包括基于格的密码学、基于编码的密码学、基于多变量的密码学、基于哈希的密码学等[4]。为寻找能够抵御未来量子计算机攻击的安全公钥密码体制，美国国家标准与技术研究院（NIST, National Institute of Standards and Technology）自2016年启动PQC标准化流程。迄今为止，PQC中CRYSTALS-Kyber、CRYSTALS-Dilithium和SPHINCS+这3种加密算法的标准已被NIST确立，第4种算法Falcon的标准化草案也会在2024年发布，其中CRYSTALS-Kyber、CRYSTALS-Dilithium、Falcon均属于格密码方案。

基于格的密码学具有易于集成和芯片化等优点，能够应对不同计算平台（云计算、物联网等）上的部署挑战，并满足各种应用场景的需求[5]。此外，基于格的密码体制在公钥尺寸、计算效率和安全性方面实现了更优的平衡[6]，因此，其广泛应用于解决传统安全问题（密钥交换、数字签名等）以及新兴安全问题（同态加密、基于身份的加密和对称加密等）[7]。

数论变换（NTT, number theoretic transform）作为一种高效的模内多项式乘法算法，能够将计算复杂度降至$O(n\log n)$，其中n为多项式系数。因此被视为加速PQC和全同态加密（FHE, fully homomorphic encryption）中多项式乘法的首选技术。但NTT烦琐的控制逻辑运算结构、较大的存储空间需求和复杂的存取调度模式是限制PQC和FHE进一步发展的主要瓶颈。因此，利用可重构硬件现场可编程门阵列（FPGA, field programmable gate array）设计出能够有效优化运算单元、降低存取复杂度以及提高可配置性的硬件结构，对密码系统乃至硬件资源的发展及优化具有重要意义。

针对运算单元的优化，现有研究主要依据增加算法的并行度和使用高基数的NTT算法，提升计算负荷，加速数据处理。为进一步优化计算性能，研究者们基于统一化设计的思想，深度分解并优化NTT、逆数论变换（INTT, inverse NTT）算法的关键计算路径，以统一化蝶形单元的设计简化其复杂计算，从而实现面积−时间积（ATP, area-time product）和计算性能的优化。在高并行度方面，Li等[8]分别采用16路和32路并行执行方法对蝶形单元进行调度，缩短了整体计算周期。尽管高并行度可以提高吞吐量和计算速度，但会消耗更多的硬件资源。在高基数方面，普遍使用的基-2算法虽然逻辑结构简单、可塑性强，然而随着NTT多项式维度的上升，会导致迭代级数大、时钟周期数量级增加。高基数伴随着高复杂度的逻辑运算和存储结构，而基-4复杂度相对较低，且在处理大规模多项式运算时表现优越，能够加速数据处理而不损失精度，可有效弥补基-2算法的不足。Chen等[9]研究表明，并行度相同的基-4 NTT ATP性能优于基-2，且随着并行度的增加优势更加显著。Garrido等[10]指出，基-4快速傅里叶变换在多项式项数较大时具有明显优势。

针对存储单元的优化，存储单元之间合理的访存设计和数据分配能有效避免内存冲突、减少数据驻留、提高内存利用率，为NTT存储节省时间和空间。目前高效且统一的访存模式相对较少，Chen等[9]提出的内存映射模式随着阶段不同而变化，使得流水线设计更为复杂。文献[11-13]采用乒乓存储策略避免NTT中内存访问冲突，但会导致片内部分RAM资源闲置，内存利用率低。

针对可配置性，NTT硬件结构设计趋向支持多种参数、适应不同计算平台的资源限制、满足多应用场景的需求，即主要集中在运行时可配置性（RTC, run-time configurability）和编译时可配置性（CTC, compile-time configurability）2个方面。RTC允许硬件在不同PQC方案中使用，而无须重新编译。CTC在固定参数的情况下，通过调整BFU的数量进行编译，以实现针对特定应用场景的ATP、功耗和吞吐量优化的设计策略。文献[14-15]提出了多项式乘法器，能够提供RTC，支持固定BFU数量的多种多项式模数，但资源消耗主要集中在存储单元。文献[9,16-18]均支持多个BFU并行化设置，可提供CTC。然而，Chen等[9]提出的内存映射的寻址和调度逻辑较为复杂，Wang等[16]、Beckwith等[17]均以消耗更多的LUT为代

价获取性能优化。文献[19-20]提出的多项式乘法器兼顾RTC和CTC性能，但Hu等[19]在资源利用、ATP等整体性能上表现不佳，Derya等[20]随着多项式系数的增加会降低相应的数据位宽，从而导致设计吞吐量受限。

综上所述，为提高整体性能并满足高性能计算的需求，本文提出了一种面向格密码算法的可配置基-4 NTT FPGA结构。该结构支持多种PQC方案中的多项式系数和模数设置而无须重新编译（即RTC），可配置不同数量的BFU以提高吞吐量或ATP性能（即CTC）。本文方案具有灵活性、可扩展性和高能效性的特点，主要贡献如下。

1) 提出一种基于格密码学的可配置基-4 NTT硬件结构，并针对该结构设计了基于流水线方式实现的基-4 NTT统一化蝶形单元。该单元兼容CT（cooley-tukey）及GS（gentleman-sande）结构，通过控制数据流向能够高效执行NTT、INTT等运算，适用于多种格密码方案，具有运行时可配置性。此外，其编译时可配置性支持多个BFU并行化设置，以实现特定应用场景中ATP、功耗和吞吐量的优化，满足多应用场景的需求。

2) 对Barrett模约减算法进行改进，提出一种运行时可配置、高效恒定时间执行的可重构模约减算法，该算法利用移位和加法替代复杂的乘法操作，减少了DSP的消耗。

3) 提出一种可配置的多RAM存储设计方案及数据访问控制算法。根据不同参数可重组RAM中写入和读取系数，适配不同数量和模数的蝶形单元。合理的流水线时延设计可满足先读后写和连续写入的存储需求，能够有效避免内存冲突，降低调度存储单元的复杂度。

1　预备知识

1.1　符号

令$\mathbb{Z}$为有理整数环，$\mathbb{Z}_q$为整数环模q的商环，q为质数。$R=\dfrac{\mathbb{Z}[x]}{\phi(x)}$表示用$\phi(x)$约减的整系数多项式环，$R_q=\dfrac{\mathbb{Z}_q[x]}{\phi(x)}$表示用$q$约减的整系数多项式环，其中$\phi(x)$为分圆多项式，具有不可约减性，大部分情况下$\phi(x)=x^n+1$，$q\equiv 1(\bmod 2n)$，$n$是2的正整数次幂，则$R_q$中多项式的次数最多为$n-1$。

NTT域的多项式用^表示，例如$\hat{s}$。小写加粗字母表示R_q中的多项式，例如多项式$\boldsymbol{g}$，$\boldsymbol{g}$的幂级数表示形式为$\boldsymbol{g}=\sum_{i=0}^{n-1}\boldsymbol{g}_i x_i,\boldsymbol{g}_i\in\mathbb{Z}_q$。对于整数$s$，$s[a{:}b]$表示整数$s$的第$b$位到第$a$位，其中$a$、$b$为偏移量，$0\leqslant a<q$，$0\leqslant b<q$，$a>b$。符号$\circ$表示多项式系数的点值对应相乘。

1.2　基于NTT的多项式乘法

NTT是有限域$\mathbb{Z}_q$上特殊形式的离散傅里叶变换算法，可以使R_q中的多项式从系数表示转换为点值表示，且没有舍入误差。其次，在多项式环$\dfrac{\mathbb{Z}[x]}{x^n+1}$上使用负包裹卷积算法，可以在不增加输入长度的情况下，实现对x^n+1的模约减。相较于FFT，NTT所有运算均为模运算，且不涉及复数或浮点数运算，不仅简化硬件电路设计，而且避免了精度的损失。与此同时，通过执行NTT可以将多项式乘法的时间复杂度降低至$O(n\log n)$，实现快速多项式乘法。NTT域内$\hat{a}_i=\mathrm{NTT}(a_i)$多项式的计算式为

$$\hat{a}_i=\sum_{j=0}^{n-1}a_j\omega_n^{ij}\bmod q,i=0,1,2,\cdots,n-1 \tag{1}$$

其中，旋转因子ω是$\mathbb{Z}_q$中的n次本原单位根，满足$\omega^n\equiv 1(\bmod q)$，$\omega^i\neq 1(\bmod q),\forall i\in n$且$q\equiv 1(\bmod n)$。将NTT中的旋转因子$\omega$替换成逆元$\omega^{-1}$后乘以$n^{-1}$得到INTT，INTT域内$a_j=\mathrm{INTT}(\hat{a}_j)$多项式的计算式为

$$a_j=n^{-1}\sum_{i=0}^{n-1}\hat{a}_i\omega_n^{-ij}\bmod q,j=0,1,2,\cdots,n-1 \tag{2}$$

因此，有限域上的2个多项式乘法计算式为

$$c=\mathrm{INTT}(\mathrm{NTT}(a)\circ\mathrm{NTT}(b)) \tag{3}$$

由于NTT只在整数环上进行运算，故式(3)中$a,b,c\in R_q$。

为实现低功耗运算需求，在FPGA上采用松耦合结构配置蝶形单元。基-4 NTT操作使用时间抽取（DIT, decimation in time）方式的CT蝶形结构，即$A=a+b\omega_n$，$B=a-b\omega_n$，按顺序输入并以位反转的顺序输出结果。INTT操作采用频率抽取（DIF, decimation in frequency）方式的GS蝶形结构，即$A=a+b$，$B=(a-b)\omega_n$，以位反转的顺序接收输入并以自然顺序输出结果。此外，应用分治策略对基-4 NTT和INTT的关键路径进行划

分和调度，将其拆分为更小的操作单元，从而降低计算复杂度，减少数据冲突，优化NTT算法性能。

基-4 NTT和INTT的详细算法过程分别如算法1和算法2所示。分别使用双层CT和GS蝶形结构执行NTT和INTT蝶形运算，第一层运算生成的中间结果Q_0,Q_1,Q_2,Q_3立即参与到第二层运算，生成最终结果$A_Z,A_{Z+T},A_{Z+2T},A_{Z+3T}$，其中参与运算的旋转因子是预先计算并存储的。

算法1 基-4 NTT算法

输入 ω_4^1, a, q, 第$2n$个单位原根ϕ_{2n}^T

输出 $\hat{A}=\mathrm{NTT}(a)$

1)for $l=0$ to $\log_4 n-1$ do

2) $T=4^l,\omega_m=\phi_{2n}^T,r=0$

3) for $s=0$ to $\dfrac{n}{4T}-1$ do

4) $u=2\mathrm{bit_reversed}(s)+1$

5) $\mathrm{waddr}_1_\mathrm{ROM}=\omega_m^u$,
$\mathrm{waddr}_2_\mathrm{ROM}=\omega_m^{2u}$,
$\mathrm{waddr}_3_\mathrm{ROM}=\omega_m^{3u}$

6) $\omega_1=\mathrm{waddr}_1_\mathrm{ROM}[r]$,
$\omega_2=\mathrm{waddr}_2_\mathrm{ROM}[r]$,
$\omega_3=\mathrm{waddr}_3_\mathrm{ROM}[r]$

7) $r=r+1$

8) for $t=0$ to $T-1$ do

9) $z=4sT+t$

10) $Q_0=\left(a_z+a_{z+2T}\omega_2\right)\bmod q$

11) $Q_1=\left(a_z-a_{z+2T}\omega_2\right)\bmod q$

12) $Q_2=\left(a_{z+T}\omega_1+a_{z+3T}\omega_3\right)\bmod q$

13) $Q_3=\left(a_{z+T}\omega_1-a_{z+3T}\omega_3\right)\bmod q$

14) $A_z=\left(Q_0+Q_2\right)\bmod q$

15) $A_{z+T}=\left(Q_1+Q_3\omega_4^1\right)\bmod q$

16) $A_{z+2T}=\left(Q_0-Q_2\right)\bmod q$

17) $A_{z+3T}=\left(Q_1-Q_3\omega_4^1\right)\bmod q$

18) end for

19) end for

20)end for

算法2 基-4 INTT算法

输入 ω_4^1, $\hat{A}$, q, 算法1生成的3个数组$\mathrm{waddr}_1_\mathrm{ROM}$, $\mathrm{waddr}_2_\mathrm{ROM}$, $\mathrm{waddr}_3_\mathrm{ROM}$

输出 $a=\mathrm{INTT}(\hat{A})$

1)for $l=0$ to $\log_4 n-1$ do

2) $T=4^l,r=\dfrac{n-1}{3}-1$

3) for $s=0$ to $\dfrac{n}{4T}-1$ do

4) $\omega_1=\mathrm{waddr}_1_\mathrm{ROM}[r]$,
$\omega_2=\mathrm{waddr}_2_\mathrm{ROM}[r]$,
$\omega_3=\mathrm{waddr}_3_\mathrm{ROM}[r]$

5) $r=r-1$

6) for $t=0$ to $T-1$ do

7) $z=4sT+t$

8) $Q_0=2^{-1}\left(a_z+a_{z+2T}\right)\bmod q$

9) $Q_1=2^{-1}\left(a_{z+2T}-a_z\right)\omega_4^1\bmod q$

10) $Q_2=2^{-1}\left(a_{z+T}+a_{z+3T}\right)\bmod q$

11) $Q_3=2^{-1}\left(a_{z+3T}-a_{z+T}\right)\bmod q$

12) $A_z=2^{-1}\left(Q_0+Q_2\right)\bmod q$

13) $A_{z+T}=2^{-1}\left(Q_1+Q_3\right)\omega_1\bmod q$

14) $A_{z+2T}=2^{-1}\left(Q_2-Q_0\right)\omega_2\bmod q$

15) $A_{z+3T}=2^{-1}\left(Q_3-Q_1\right)\omega_3\bmod q$

16) end for

17) end for

18)end for

2 算法优化设计

2.1 NTT整体结构设计

如图1所示，本文设计了一种可扩展的FPGA结构，主要包括控制单元、蝶形单元、多路复用器以及ROM、RAM存储模块。

控制单元通过选择信号驱动多路复用器，执行蝶形单元与存储单元RAM之间的数据交互操作。控制单元内的地址生成器根据参数和蝶形单元数量，生成RAM和ROM的读写地址及使能信号，为每个系数生成相对应的内存地址和组数。

蝶形单元由4个可配置的处理元件（PE, processing element）块构建，控制模式控制各阶段的运算状态转换，可支持多种参数执行NTT、INTT运算，在运算过程中依次选取预处理后存储在ROM中的旋转因子ω参与运算。其中，ω由数据总线连接到蝶形单元。由于算法1和算法2中的基- 4 NTT和INTT算法可利用循环结构的展开实现操作的并行化，因此本文设计的蝶形单元数量在编译时可按需

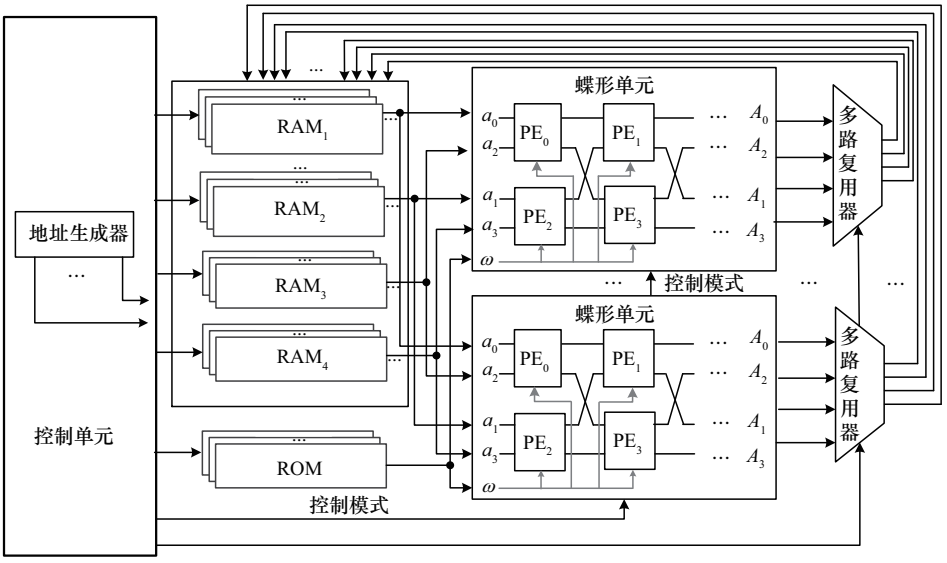


图1　FPGA整体结构

调整，以优化吞吐量和提升并行度。对于n点NTT和INTT、p个蝶形单元，采用每个蝶形单元匹配4个RAM块，其地址大小为$\frac{n}{4p}$的策略，存储输入值、中间结果和输出值。根据不同参数和并行度的存储需求重新配置RAM块，以满足可配置性。

综上所述，FPGA整体结构通过各模块协同合作可同时支持RTC和CTC，能够灵活地完成NTT和INTT等运算。

2.2　统一化基-4蝶形单元优化设计

NTT和INTT使用蝶形单元作为基本操作单元，通过多次迭代调用以实现高效运算。而蝶形单元的优化设计主要集中在结构优化和算法优化2个方面。针对结构优化，本文提出可配置的统一化基-4蝶形单元，能够按需调整为CT和GS蝶形配置，可提供RTC。相较于仅支持CT或GS的普通蝶形结构，无须花费额外的时钟周期进行预处理和后处理操作，减少了计算路径，提高了可配置性。针对算法优化，采用基-4算法进行蝶形运算，并对模除、模约减等关键算法进行优化和改进，使其在降低复杂度、实现高效计算的同时，能够适配多种参数，具有较高的灵活性。

统一化基-4蝶形单元由4个PE块组成，PE块包括模加法器、模减法器、2^{-1}模乘法器以及Karatsuba乘法和优化的Barrett模约减单元等，详细介绍如下。

2.2.1　蝶形单元结构设计

大多数硬件方案仅支持CT/GS结构，进行NTT计算时，需要额外的时钟周期调整系数。统一化蝶形单元相较单一的CT/GS结构设计，硬件资源消耗相近，更适合面积优化和高能效加速器的应用[21]。为此，本文设计了一种统一化基-4蝶形单元。通过重复利用CT和GS中的运算模块，使用控制信号调整CT/GS这2种配置，实现无须预处理和后处理的NTT、INTT运算，以达到面积、资源性能优化的目的。其结构设计如图2所示。

PE块根据控制信号以不同顺序执行，实现NTT/INTT运算。在运算过程中，模乘法器具有3个时钟周期的延迟，而模加法器和模减法器在运行时需要一个时钟周期，并引入一个时钟周期的逻辑延迟。CT和GS运算中模乘执行顺序存在差异，导致所需时钟周期不同。为确保时序准确和数据同步，需在蝶形单元中引入额外寄存器以产生额外的延迟，从而平衡执行顺序差异引起的路径延迟。在INTT计算中，需要使用2^{-1}模乘法器，它通过移位、模加法器和多路复用器替代了复杂的除法运算。此外，配备可重置的多路复用器，确保蝶形单元能够以完全流水线的方式工作。而运算所需的旋转因子ω存储在单个ROM中，供所有PE块共享和调度。

在基-4蝶形单元中，选择信号sel控制多路复用器，进而牵制运算状态转换。当sel为0时，蝶形单元执行NTT运算。RAM存储的多项式系数向量从a_0,a_1,a_2,a_3端口输入，ROM存储的旋转因子从$\omega_1,\omega_2,\omega_3$端口输入，在第一层PE内进行第一层CT运算，生成的中间结果Q_0,Q_1,Q_2,Q_3立即参与到第

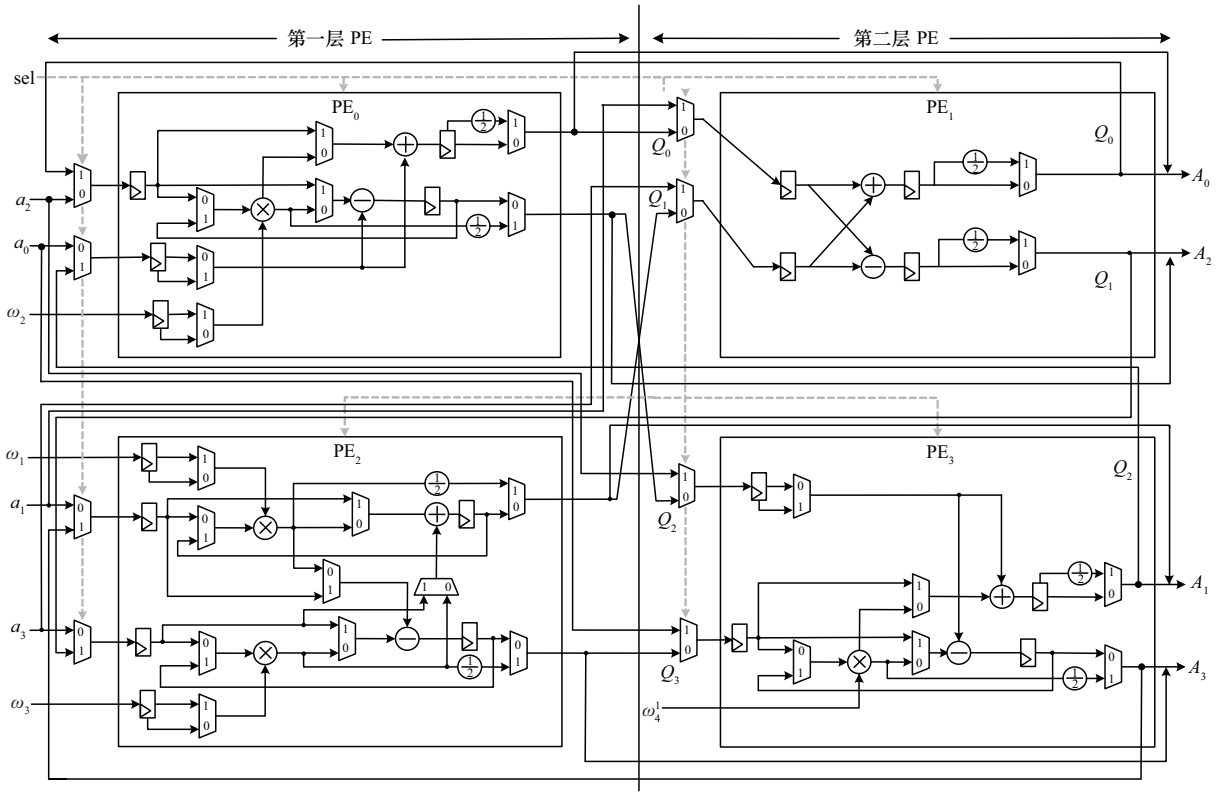


图2　统一化基-4蝶形单元

二层PE执行第二层CT运算。NTT最终计算结果的输出系数向量为$\boldsymbol{A}_0,\boldsymbol{A}_1,\boldsymbol{A}_2,\boldsymbol{A}_3$。当sel为1时，蝶形单元执行INTT运算。与NTT执行顺序相反，第二层PE将接收到RAM存储的多项式系数向量，进行GS运算，运算得到的中间结果输入第一层PE执行第二层GS运算，最终输出运算结果系数向量。

本文提出的统一化蝶形单元设计支持多种格密码方案的系数和模数，具有运行时可配置性。相比之下，Abdulrahman等[22]使用CT结构执行蝶形运算，虽然复杂度相较于本文低，但在执行INTT时，为避免数值溢出，需进行额外的模数约减操作，且缺乏本文提出的可配置性。Mert等[23]蝶形单元采用GS结构，其系数和模数的位宽在编译时确定，因此同样不具备可配置性。文献[16,24-25]均采用具有CT和GS结构的统一化蝶形单元，通过调整控制信号执行NTT、INTT运算。然而Wang等[16]仅适用于Dilithium算法优化，而Wang等[24]在编译时系数和模数固定，均不具备运行时可配置性。

2.2.2　模加减及模除优化

模加法器和模减法器是蝶形单元的基本组件，主要由组合逻辑电路实现，用来判断加减的值是否超出素数q，并进行必要的修正，避免数据溢出。本文参考文献[26]提出的方法，对模加法单元和模减法单元进行硬件结构的设计。如图3和图4所示，a、b为（n + 1）bit的输入，R和R_q为（n + 1）bit的中间结果，c为（n + 1）bit的输出结果，模加和模减的最终运算结果基于R和R_q的最高位判断，并控制多路复用器输出。

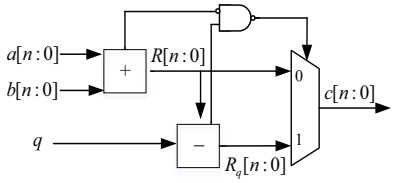


图3　模加法器硬件结构

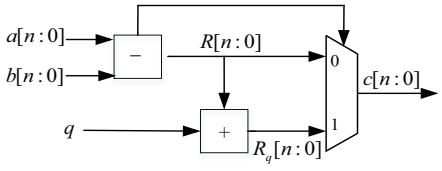


图4　模减法器硬件结构

INTT需要进行类似$\frac{x}{2}\bmod q$的运算，为降低时钟消耗以及硬件复杂度，本文采用文献[27]的方法

避免模乘运算。硬件电路结构如图5所示，2^{-1}模乘运算仅使用移位、模加操作和多路复用器即可实现。当模数q为常数且为奇素数时，在q确定的情况下，$\frac{x}{2}\bmod q$等价于$\frac{q+1}{2}$。当x为偶数时，$\frac{x}{2}\bmod q$等效于$x>>1$；当x为奇数时，$\frac{x}{2}\bmod q$等效于

$$\begin{aligned}\frac{x}{2} &\equiv \left(2\left\lfloor\frac{x}{2}\right\rfloor+1\right)\frac{(q+1)}{2}\\ &\equiv \left\lfloor\frac{x}{2}\right\rfloor+\frac{q+1}{2}\ (\bmod q)\end{aligned}\tag{4}$$

其中，$\left\lfloor\frac{x}{2}\right\rfloor$等价于$x>>1$，$\frac{q+1}{2}$为常数。$\frac{x}{2}\bmod q$的最终结果由$x$的最低位$x[0]$进行判断输出。通过将$q$设为可配置参数，并利用位移、加法以及逻辑与操作，简化了$\frac{x}{2}$的求模运算。

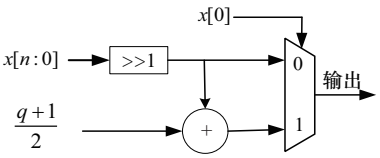


图5　2^{-1}模乘法器硬件结构

2.2.3　Karatsuba乘法

模乘法是NTT/INTT算法中最复杂的算术运算之一，传统模乘算法采用试除法，其效率较低，不易扩展，难以在硬件上实现。若使用传统的乘法运算将2个多项式直接相乘，则复杂度为$O(n^2)$，而使用Karatsuba算法可将复杂度降低到$O\left(n^{\mathrm{lb}3}\right)$，有效降低了计算复杂度。

本文使用文献[28]中的Karatsuba算法计算模乘。Karatsuba算法采用分而治之的策略，通过递归的方法将多项式分解为多个较低次数的多项式后进行简单计算，减少逐点乘法的次数，进一步提高效率。对于多项式$f(x)$和$g(x)$，Karasuba算法首先将它们划分为较低次数的多项式$f=f_{\mathrm{H}}+x^{2^{-1}n}f_{\mathrm{L}}$和$g=g_{\mathrm{H}}+x^{2^{-1}n}g_{\mathrm{L}}$，其中，$g_{\mathrm{H}}$、$g_{\mathrm{L}}$、$f_{\mathrm{H}}$、$f_{\mathrm{L}}$分别为

$$f(x)=\left(\underbrace{x_{n-1},x_{n-2},\cdots,x_{2^{-1}n}}_{f_{\mathrm{H}}},\underbrace{x_{2^{-1}n-1},x_{2^{-1}n-2},\cdots,x_0}_{f_{\mathrm{L}}}\right)\tag{5}$$

$$g(x)=\left(\underbrace{x_{n-1},x_{n-2},\cdots,x_{2^{-1}n}}_{g_{\mathrm{H}}},\underbrace{x_{2^{-1}n-1},x_{2^{-1}n-2},\cdots,x_0}_{g_{\mathrm{L}}}\right)\tag{6}$$

因此，$h=f\times g$可被表示为

$$\begin{aligned}h=&\left(f_{\mathrm{H}}+x^{2^{-1}n}f_{\mathrm{L}}\right)\left(g_{\mathrm{H}}+x^{2^{-1}n}g_{\mathrm{L}}\right)=\\ &f_{\mathrm{H}}g_{\mathrm{H}}-f_{\mathrm{L}}g_{\mathrm{L}}+x^{2^{-1}n}\left(f_{\mathrm{L}}g_{\mathrm{H}}+f_{\mathrm{H}}g_{\mathrm{L}}\right)=\\ &f_{\mathrm{H}}g_{\mathrm{H}}-f_{\mathrm{L}}g_{\mathrm{L}}+x^{2^{-1}n}((f_{\mathrm{H}}+f_{\mathrm{L}})(g_{\mathrm{H}}+g_{\mathrm{L}})-\\ &f_{\mathrm{H}}g_{\mathrm{H}}-f_{\mathrm{L}}g_{\mathrm{L}})\end{aligned}\tag{7}$$

从上述分析可知，使用Karatsuba算法计算2个大数乘法只需3次低次多项式的乘法运算和6次加减法运算，即可得到结果。

2.2.4　可重构的高效模约减算法

NTT在整数域$\mathbb{Z}_q$上进行运算，多项式乘法运算中会多次执行蝶形运算，循环调用取模算法，将运算结果进一步约减到有限域。取模算法一般根据模数q选定，常见的模约减算法有快速模约减、Barrett和Montgomery。以计算$t=ab\bmod q$为例，快速模约减算法针对特定的模数q，递归地利用q的特殊属性对ab的值进行比特差分和重组，直至归约结果的每一项位数都不大于k，具有较优的性能，但只针对特定模数，不提供RTC，可扩展性差。Barrett和Montgomery模约减均可支持任意比特的模乘，更适合非特定模数；其不同之处在于，Montgomery算法需要将系数转化为$\mathbb{Z}_q$中的Montgomery域$a_m=a2^n\bmod q$表示，导致中间结果的数据位宽增加，消耗更多硬件资源。

为此，本文对Barrett模约减算法进行优化和改进，提出了一种可为主流格密码方案中的参数提供RTC的模约减算法。该算法实现了恒定时间模约减的高效计算，同时具有灵活性高、复杂度低的特点。

算法3　改进的Barrett模约减算法

输入　$q=h_{q_1}2^{D_1}+h_{q_2}2^{D_2}\cdots+h_{q_i}2^{D_i}$，$z=ab$，$k=\lceil\mathrm{lb}\,q\rceil$，$u=\left\lfloor\frac{r^2}{q}\right\rfloor=h_{u_1}2^{D'_1}+h_{u_2}2^{D'_2}\cdots+h_{u_j}2^{D'_j}$，$r=2^k$

输出　$t=z\bmod q$

1)根据q、u的分解计算D_i,D'_j，$0\leqslant D_i\leqslant k$，$0\leqslant D'_j\leqslant k$

2)根据q、u的二进制展开形式求解每个运算数的符号位$h_u=\{h_{u_1},h_{u_2},\cdots,h_{u_j}\}$,$h_{u_j}\in\{-1,+1\}$，$h_q=\{h_{q_1},h_{q_2},\cdots,h_{q_i}\}$,$h_{q_i}\in\{-1,+1\}$，$0\leqslant q_i\leqslant u_j\leqslant k$

3)$m=\left\lfloor\left\lfloor\frac{z}{r}\right\rfloor\left(\frac{u}{r}\right)\right\rfloor=\left\lfloor\left\lfloor\frac{z}{2^{k-1}}\right\rfloor\left(\frac{u}{2^{k+1}}\right)\right\rfloor=h_{u_1}z\left[2k-1:2k-D'_1\right]+h_{u_2}z\left[2k-1:2k-D'_2\right]+$

$\cdots + h_{u_j} z\left[2k-1:2k-D'_j\right]$

4) $t = z - mq = z - \left(h_{q_1} m \ll D_1 + h_{q_2} m \ll D_2 \cdots + h_{q_i} m \ll D_i\right)$

5) while $t < 0$ do

6) $t = t + q$

7) end while

8) while $t > q$ do

9) $t = t - q$

10) end while

11) return t

如算法 3 所示，Barrett 算法根据预估商值得到结果，所有运算均在 $\mathbb{Z}$ 中进行，取模运算可以表示为 $z = mq + t$。首先，计算 m 的近似值 $m = \left\lfloor \frac{z}{q} \right\rfloor \approx \left\lfloor \frac{zu}{2^{2k}} \right\rfloor \approx \left\lfloor \left\lfloor \frac{z}{2^{k-1}} \right\rfloor \left(\frac{u}{2^{k+1}} \right) \right\rfloor$，可得到 $t = z - mq$。其中，z 为待约减数，最大位宽为 $2k$；a、b 的取值范围为 $[0,q)$；当 q 确定时，u 为确定值。其次，通过对 u 值的分解来确定 m 的处理位宽，进而控制迭代轮数。同时，对 q 值进行分解，得到具体的移位参数。根据 q、u 的二进制展开形式求解每个运算数的符号位 h_q、h_u，利用移位和加法替代乘法运算。在算法的最后阶段，需要判断 t 值以对输出结果进行处理，通过加减 q 值的操作实现输出结果在标准范围 $[0,q)$ 内，确保最终迭代结果无数据溢出。

算法 3 中，具体参数 q 和 u 值分解对应的 D_i、D'_j 及 h_{q_i}、h_{u_j} 如表 1 所示，可依据不同算法的模数 q 和 u 值调整移位参数进行模约减。本文提出改进的模约减方法并不需要额外的乘法运算，因此，对模数 q 的模约减是高效的且运行时间不会随着输入数据的大小而发生改变。相较于传统的 Barrett 算法，优化和改进后的方案可在运行时支持多个参数，而无须重新编译，具有较高的灵活性。

2.3 可配置多 RAM 访存优化设计与控制

在确定运算结构后，为高效地实现多项式乘法运算，需要设计一个精简的控制结构来调度存储单元。多项式系数存储在 RAM 中，蝶形单元运用这些内存块为中心的数据移动进行实例化，这要求每一个多项式系数在 RAM 中必须具有唯一的地址。16 点基-4 NTT/INTT 数据流向及存储访问如图 6 所示。

图 6 使用 4 种不同表现方式表示不同轮数的蝶形操作。例如，黑色实线对应第一阶段和第二阶段的第一轮蝶形操作，其他同理。NTT 的数据流向自上而下，INTT 数据流向与 NTT 相反。数据流向图下方对应蝶形运算的系数存储访问模式，采用 4 个可配置的 PE 执行蝶形操作。为确保在同一时钟周期内从 RAM 中读取到每一组系数对，需要分别存储 (a_0,a_1,a_2,a_3)、(a_4,a_5,a_6,a_7)、(a_8,a_9,a_{10},a_{11}) 和 $(a_{12},a_{13},a_{14},a_{15})$。然而，由于每进行一次 PE 操作，系数对都会发生改变，因此计算后需对 RAM 存储进行调整，以满足后续计算需求。

基-4 NTT 处理大小为 n 的多项式，需要进行 $\log_4 n$ 个阶段的运算，一个蝶形单元每个阶段读取 4 个数据点，执行 $4^{-1}n$ 轮运算。这要求 RAM 存储单元在一个周期内至少能够提供 4 个点的读取和写入操作。同时，由于 NTT/INTT 蝶形结构和调度参数的差异，需确保在 RAM 中多项式参数有唯一地址，并合理存储蝶形运算结果以便后续计算。随着多项式项数的增加，大量的数据交互将导致 RAM 的存取调度复杂化。

基于此，本文提出了一种可配置多 RAM 存储

表 1 PQC 中部分算法的 q 值和 u 值

算法	k	$q = h_{q_1}2^{D_1} + h_{q_2}2^{D_2} + \cdots + h_{q_i}2^{D_i}$	$u = \left\lfloor \frac{r^2}{q} \right\rfloor = h_{u_1}2^{D'_1} + h_{u_2}2^{D'_2} + \cdots + h_{u_j}2^{D'_j}$
Kyber	12	$3\,329 = 2^{11} + 2^{10} + 2^8 + 1$	$5\,040 = 2^{12} - 2^{10} - 2^6 - 2^4$
Aigis	13	$7\,681 = 2^{13} - 2^9 + 1$	$8\,736 = 2^{13} + 2^9 + 2^5$
Falcon	14	$12\,289 = 2^{14} - 2^{12} + 1$	$21\,843 = 2^{14} + 2^{12} + 2^{10} + 2^8 + 2^6 + 2^4 + 2^1 + 2^0$
LIMA	17	$133\,121 = 2^{17} + 2^{11} + 1$	$129\,054 = 2^{17} - 2^{11} + 2^5 + 2^1$
qTESLA v2.0	22	$4\,206\,593 = 2^{22} + 2^{13} + 2^{12} + 1$	$4\,182\,050 = 2^{22} - 2^{13} - 2^{12} + 2^5 + 2^1$
Dilithium	23	$8\,380\,417 = 2^{23} - 2^{13} + 1$	$8\,396\,807 = 2^{23} + 2^{13} + 2^3 - 1$
qTESLA	23	$8\,404\,993 = 2^{23} + 2^{14} + 1$	$8\,372\,254 = 2^{23} - 2^{14} + 2^5 - 2^1$

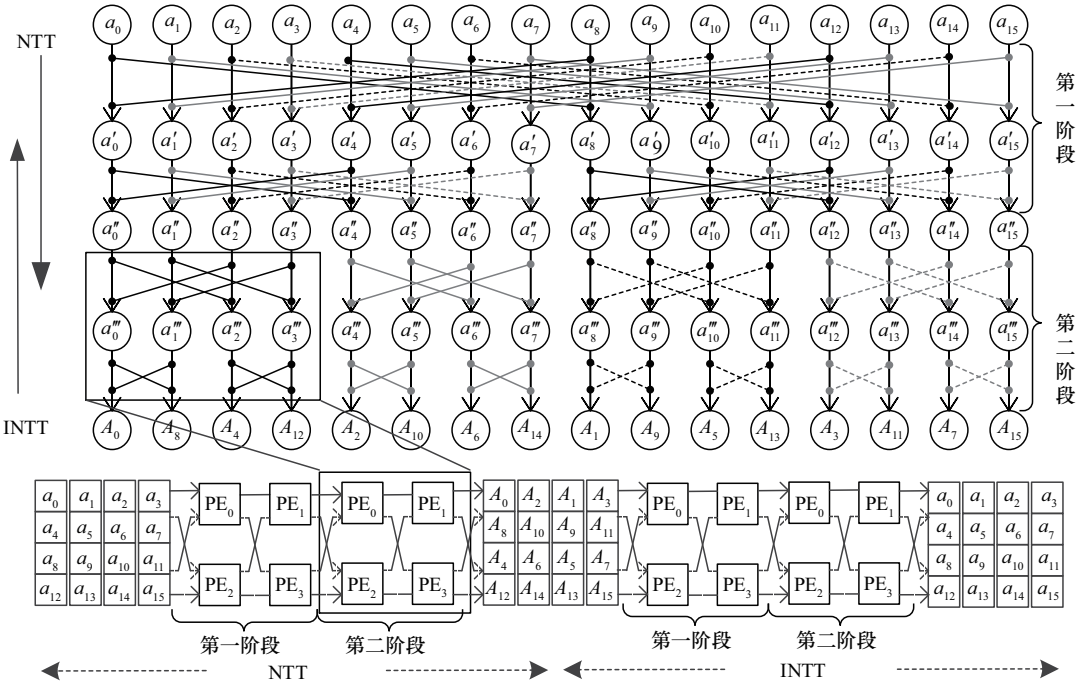


图6　16点基-4 NTT/INTT数据流向及存储访问

设计方案及数据存储分配算法，根据不同参数可重组RAM中写入和读取系数，能够在避免内存冲突和实现精简控制内存的前提下，适配不同数量和模数的蝶形单元。

2.3.1　可配置多RAM存储优化设计方案

可配置多RAM存储优化的设计，一方面需要在避免内存冲突的前提下提高内存利用率，以精简ATP；另一方面需要提供RTC和CTC，以满足蝶形单元可重构性读取数据的需求。因此，整体设计方案应从以上2个方面进行优化。

对多RAM存储进行设计时，采用每个蝶形单元匹配4个RAM块，系数数组存储在不同的内存单元中，各蝶形单元独立执行策略避免并行计算产生地址冲突。对于p并行的NTT运算，共需设计4p个RAM块，每个RAM块内的索引地址分成MEM_1、MEM_2，用于存储多项式系数以及NTT、INTT蝶形运算的结果。

以单并行的一个RAM组为例，RAM分割和加载方案如图7所示。在运算过程中，首先，将n个多项式系数分别存储在内存MEM_1中。然后，在进行同一时钟周期的蝶形运算时，分别从不同RAM块的第i个、第$i+\frac{n}{4}$个、第$i+\frac{n}{2}$个和第$i+\frac{3n}{4}$个（$0 \leqslant i < \frac{n}{4}$）MEM地址读取系数对至蝶形单元，执行NTT运算。最后，将NTT运算存入MEM_2中并读取至蝶形单元，利用其内部的乘法单元进行点对位相乘操作，将计算结果暂存在MEM_1，等待

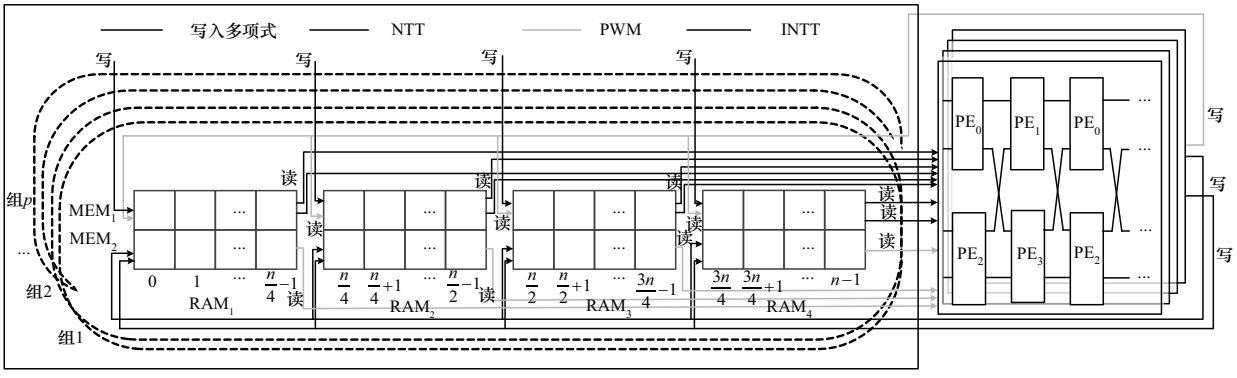


图7　RAM存储流程

INTT阶段的执行。在p并行的情况下，每个RAM块匹配$\frac{n}{4p}$个地址，存取方式与单并行同理。

上述加载模式可以参数化以支持CTC，并允许在不重新编译的情况下，将不同参数加载到RTC设计中。由于每一阶段的蝶形运算只需统一读取一次得到n个参数组成的系数对，蝶形运算执行后写入中间结果。因此，通过合理分配、存储中间结果到不同RAM地址中，以确保下一次运算仍能从同一RAM地址读取相对应的系数对，满足下一阶段的计算需求。这种复用有限的RAM存储单元，完成了多项式乘法单路多路并行存储的结构，显著提高了内存利用率。

2.3.2 *可配置的数据访问控制算法*

为实现精简的数据存储控制，对不同参数和并行度的NTT/INTT运算数据流向及存储访问过程深入剖析，进而得到多项式系数及其运算结果在读写地址上的变换规律。具体而言，不同阶段的蝶形运算之间读写地址变换仅与阶段数有关，同一阶段内部地址变换依照循环次数的值对RAM地址读写模式固定。综上分析，可从循环次数和蝶形运算阶段2个方向简化复杂的数据流控制。

NTT存储单元数据分配的读写地址变换过程分别如算法4和算法5所示。循环次数cloop更新每个阶段的蝶形运算次数，初始值为cloop[1:0]==2`b00，每个时钟加1，临界值为$(n \gg (\mathrm{lb}\,p+1))-1$，达到临界值后重置归零重新计数，等待下一阶段蝶形运算开始。蝶形运算阶段数cstg的值在$[0,\log_4 n-\lceil \mathrm{lb}\,p\rceil-1)$时，当前RAM的读写地址需根据cstg和cloop进行额外计算得到；当cstg的值在$[\log_4 n-\lceil \mathrm{lb}\,p\rceil-1,\log_4 n)$时，当前RAM的读写地址由cloop直接决定。

算法4 基-4 NTT RAM读地址变换

输入 多项式项数n，BFU个数p，循环次数cloop，蝶形运算阶段cstg

输出 RAM的读地址raddr

1)初始化:raddr = 0

2)clim = $\mathrm{lb}\,n-\lceil \mathrm{lb}\,p\rceil-1$

3)cstp = 2(cstg + 1),csum = cstp − clim

4)raddr_1 = ((cloop >> (csum + 1)) << csum),
raddr_2 = raddr1 >> 1,raddr_ntt = raddr_1 + raddr_2

5)if cstg < $\log_4 n-\lceil \mathrm{lb}\,p\rceil-1$then

6) if cloop[1:0] == 2`b00 then

7) raddr = (cloop >> 2) + raddr_ntt

8) else if cloop[1:0] == 2`b01 then

9) raddr = (1 << (csum − 1) + (cloop >> 2) + raddr_ntt

10) else if cloop[1:0] == 2`b10 then

11) raddr = (1 << csum) + (cloop >> 2) + raddr_ntt

12) else if cloop[1:0] == 2`b11 then

13) raddr = (1 << (csum − 1) + (1 << csum) + (cloop >> 2) + raddr_ntt

14) end if

15) else raddr = cloop

16) end if

17) return raddr

算法5 基-4 NTT RAM写地址变换

输入 多项式项数n，BFU个数p，循环次数cloop，蝶形运算阶段cstg

输出 RAM的写地址waddr_0,waddr_1,waddr_2,waddr_3

1)初始化:$\mathrm{waddr}_0=\mathrm{waddr}_1=\mathrm{waddr}_2=\mathrm{waddr}_3=0$

2)clim = $\mathrm{lb}\,n-\lceil \mathrm{lb}\,p\rceil-1$

3)cstp = 2(cstg + 1),csum = cstp − clim

4)waddr_1 = ((cloop >> (cum + 1)) << cum),
waddr_2 = waddr1 >> 1,waddr_ntt = waddr_1 + waddr_2

5)if cstg < $\log_4 n-\lceil \mathrm{lb}\,p\rceil-1$ then

6) if cloop[1:0] == 2`b00 then

7) waddr_0 = (cloop >> 2) + waddr_ntt

8) else if cloop[1:0] == 2`b01 then

9) waddr_1 = (1 << (csum − 1) +(cloop >> 2) + waddr_ntt

10) else if cloop[1:0] == 2`b10 then

11) waddr_2 = (1 << csum) + (cloop >> 2) + waddr_ntt

12) else if cloop[1:0] == 2`b11 then

13) waddr_3 = (1 << (csum − 1) + (1 << csum) +(cloop >> 2) + waddr_ntt

14) end if

15) else

16) waddr_0 = cloop

17) waddr_1 = cloop

18) $waddr_2$ = cloop

19) $waddr_3$ = cloop

20) end if

21) return $waddr_0$,$waddr_1$,$waddr_2$,$waddr_3$

如算法4 NTT RAM读地址变换所示，RAM读地址raddr的初始值为0，根据当前多项式的个数、循环次数、蝶形单元数和执行阶段数变换得到基础读取地址raddr_ntt = raddr_1 + raddr_2。以raddr_ntt为基础，依据cloop和cstg的循环变换得到最终读地址raddr。算法5的4个写地址$waddr_0$、$waddr_1$、$waddr_2$、$waddr_3$同理。INTT与NTT的地址变换规律类似，此处不再赘述。

RAM进行读写操作时，必须先获取读写地址以执行读写操作，需要一定的时间间隔。为提高系统的稳定性和高效性，需要协调好流水线和RAM访存之间的关系，确保蝶形运算在连续流水线中无数据中断和错误。

如图8所示，以n=256单并行NTT蝶形运算的RAM读写为例。首先预计算RAM的读写地址，接着发送RAM读地址到流水线中，经过16个时钟周期的等待，最后发送RAM写地址，这使得每个阶段的等待时间为20个时钟周期。此流水线时延设计即可实现先读取后写入的操作，又可满足蝶形运算完成后将计算结果连续写入RAM存储单元的合理性，有效避免了数据中断和内存冲突。

3　结果对比分析

本文硬件模块设计使用Verilog HDL语言进行描述，在FPGA目标开发平台实现设计，通过Vivado 2019.2进行综合、布局、布线，芯片型号选用Xilinx Virtex-7 FPGA (xc7vx690tffg1761-2)和Kintex UltraScale FPGA (xcku060-ffva1156-2)。对Kyber（n=256，q=7 681）、Dilithium（n=256，q=8 380 417）、Falcon（n=1 024，q=12 289）等多种格密码方案进行不同并行度的实验。

3.1　NTT各模块实现

以Dilithium算法为例，使用xcku060-ffva1156-2芯片型号，对基-4 NTT的整体实现进行综合仿真。本文方案1BFU的物理设计布局如图9所示，图中突出显示了NTT中各个模块的区域。从图9可以看到，设计中的大部分面积被蝶形单元、地址控制单元、多RAM存储单元和ROM存储单元占据。由于在NTT模块的设计中，一个蝶形单元配置4个PE模块，因此占比最多。剩余的空间则由连接各个模块的多路复用器和基于FSM的控制逻辑占用。

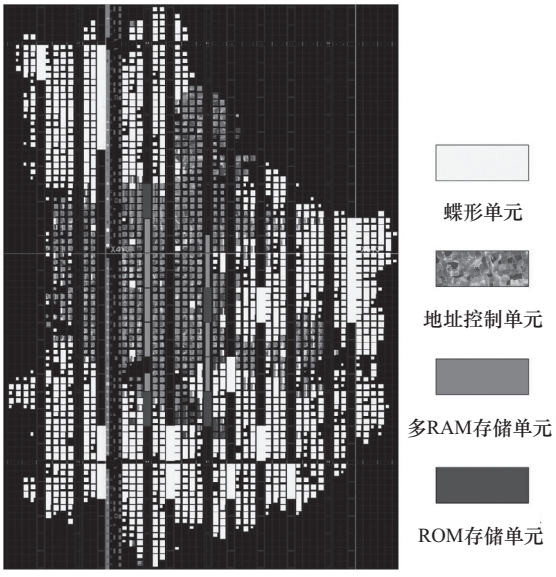


图9　物理设计布局

基-4 NTT子模块的资源消耗如表2所示，NTT总体采用串并行混合策略以提高整体计算性能。地址控制单元承担了多项任务，如控制使能信号、生成系数索引、读写使能信号等，同时还需要计算蝶形运算阶段数和迭代轮次数，导致消耗更多的硬件资源。蝶形单元占用较多的LUT资源，一个4BFU基-4 NTT需要消耗32个DSP完成256项多项式乘法运算。RAM用于多项式参数存储，ROM存储旋

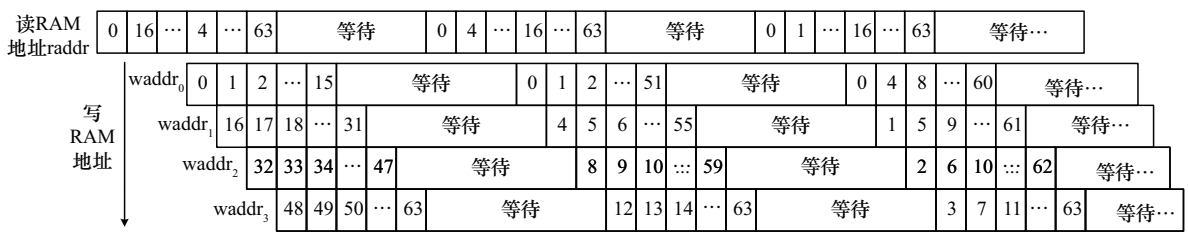


图8　RAM读写等待示意

表 2 基-4 NTT 子模块的资源消耗

模块	实现方式	1BFU 资源消耗				4BFU 资源消耗			
		LUT/个	FF/个	BRAM/个	DSP/个	LUT/个	FF/个	BRAM/个	DSP/个
蝶形单元	流水线	1 573	1 329	0	8	4 891	3 728	0	32
地址控制单元	串行	185	162	0	0	977	201	0	0
多 RAM 存储单元	并行	97	0	2	0	143	0	8	0
ROM 存储单元	串行	36	0	1	0	42	0	4	0
NTT 总体	串并混合	1 948	1 675	3	8	6 678	4 391	12	32

转因子 ω，二者仅需消耗少量的 LUT、BRAM，这是由于本文设计对内存控制进行了优化，将更多的资源部署到计算机上。

本文方案在 vivado 上的仿真波形如图 10 所示。图中显示了 1BFU 基-4 NTT 运算中部分随机生成的多项式系数、旋转因子、多项式系数输入数据和旋转因子输入数据，以及经过运算后 NTT 运算结果的输出数据。

3.2 模约减单元硬件评估

目前常用的模约减算法有 Montgomery、Barrett 以及快速模约减等。表 3 给出了 $h = z \bmod q$ 的 4 种模约减算法实现，其中模 q=8 380 417，$z = ab$，以对本文提出改进的 Barrett 模约减算法进行评估。4 种算法均以流水线方式实现，且将 $z = ab$ 的值作为输入，并通过移位操作实现除以或乘以 2^n 的函数计算。Montgomery 算法[18]需要执行 3 次乘法运算，消耗的资源最多；Barrett 算法[26]消耗的资源其次；快速模约减算法[29]消耗的资源最少，且消耗了最少的加法器 CARRY8，这得益于其对模数 q 的特殊性质进行了优化，只需简单的移位和加法操作后，再进行一次多路复用选择即可实现，简化了部分运算过程。

表 3 4 种模约减算法性能对比

算法	实现方式	执行操作	扩展性	资源消耗/个		
				LUT	FF	CARRY8
Montgomery[18]	3 级流水线	3 次乘法 3 次移位 10 次加法	好	198	156	13
Barrett[26]	3 级流水线	2 次乘法 2 次移位 2 次加法	差	135	92	8
快速模约减[29]	3 级流水线	7 次加法 1 次移位 1 次 MUX	差	56	0	4
改进的 Barrett	2 级流水线	10 次加法 2 次移位 1 次 MUX	好	84	5	7

本文提出改进的 Barrett 算法采用提前预先计算的加法和移位代替 Barret 算法[26]的乘法操作，并以预计算的方式存储中间参数，需要进行一次多路复用选择，资源消耗仅高于快速模约减算法[29]，可适用于多种格密码方案的参数。对于可扩展性，快速模约减算法通过递归地利用 q 的特殊性质进行优化，不同的 q 值导致重组形式和位宽存在差异，优化方法也有所不同，因此该算法的扩展性较差。Barrett 算法同样需要针对特定模数进行优化，一旦

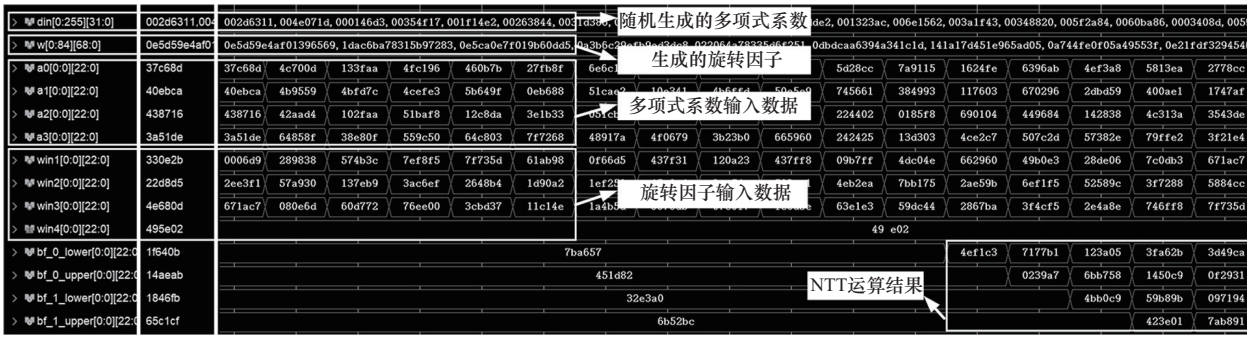
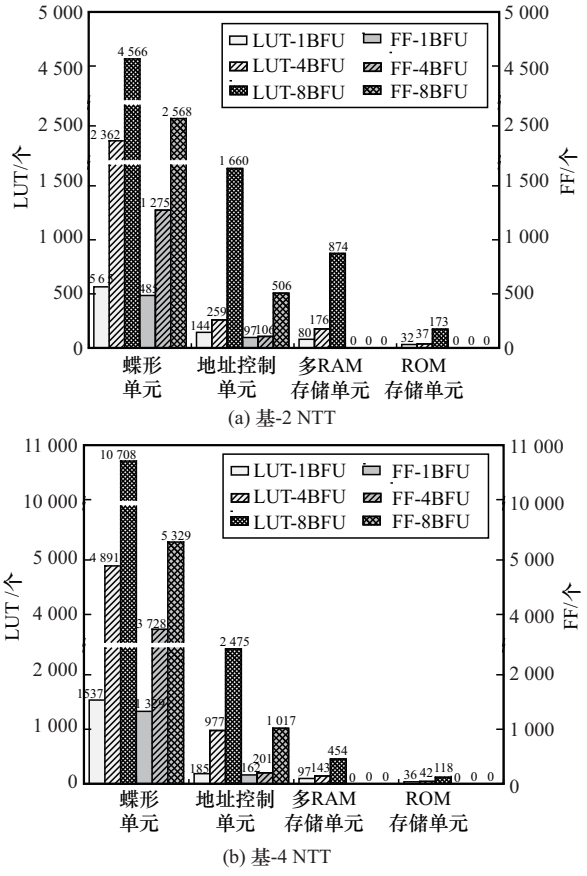


图 10 基-4 NTT 运算仿真波形

模数发生变化，就需要再次优化，仍不具备可扩展性。而 Montgomery 算法和改进的 Barrett 算法仅需修改部分可配置的参数即可兼容不同模数，具有较好的可扩展性。

3.3　基-2 与基-4 NTT 性能对比分析

对基-2 和基-4 NTT 不同并行度的子模块实现进行比较，如图 11 所示。在实现 Dilithium 算法中的多项式乘法计算时，针对蝶形单元和地址控制单元，基-4 蝶形单元相比基-2 涉及更多的逻辑计算和蝶形级数，且逻辑结构更为复杂，需要更缜密的蝶形控制和数据流向设计，导致消耗更多的硬件资源。此外，随着并行度的增加，基-2 和基-4 蝶形单元占用的 LUT 和 FF 资源大幅上升，占据 NTT 总体消耗资源的主要部分。存储方面，RAM 和 ROM 用于存储执行 Dilithium 算法时 NTT 计算所需的全部数据，仅消耗 FPGA 小部分的 LUT 和 FF 资源。



(a) 基-2 NTT

(b) 基-4 NTT

图 11　基-2 和基-4 子模块 LUT/FF 资源消耗

电路实际性能不仅取决于硬件资源利用率，还与时钟周期和频率相关。基于完成一次 NTT 计算所需的时间和加速器的面积，引入 ATP 作为一项评价指标，ATP 计算式为

$$\text{ATP} = \text{时间} \times \text{LUT} \tag{8}$$

此外，引入吞吐量指标以比较 1 s 内通过 NTT 加速器的比特数，计算式为

$$\text{吞吐量} = \frac{\text{总比特数}}{\text{时延}} \tag{9}$$

在相同数量的蝶形单元下，基-4 NTT 的 ATP 和吞吐量优于基-2，且随着并行度的增加，优势显著放大，如图 12 所示。但是，基-4 NTT 相较于基-2 每次存取、计算的数据量较大，导致其 BRAM 和 DSP 资源消耗更高。尽管如此，由于不同并行度的基-2 和基-4 NTT 执行同等数量的模数乘法操作，因此，从 1BFU 到 8BFU 在 DSP 和 BRAM 的资源消耗上呈现相同的增长趋势。

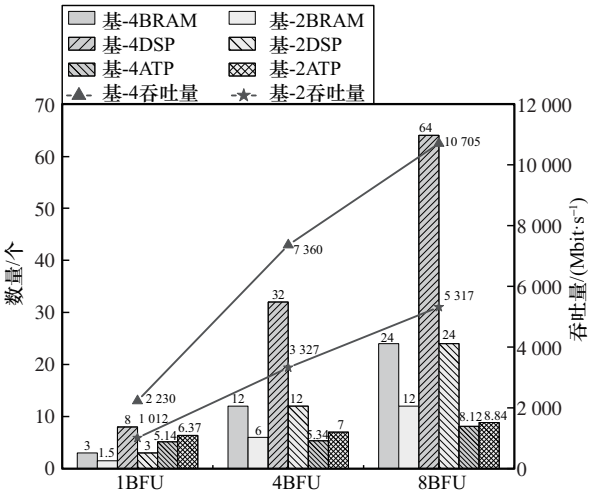


图 12　基-2 和基-4 中 ATP 及吞吐量变化

在不同算法参数（其中 k=[lb$q$]）下，由表 4 仿真综合的详细数据可知，基-4 NTT 的时延优于基-2 NTT 设计，这主要得益于基-4 算法在计算效率上的优势。具体而言，基-4 NTT 的迭代次数仅为基-2 NTT 的一半，且一个基-4 蝶形单元由 4 个 PE 组成，仅需 3 个乘法器，而基-2 NTT 的蝶形单元仅有一个 PE，在完成相同计算量时，后者需要 4 个乘法器[30]。

此外，在表 4 中，对于 Kyber、Dilithium、Falcon 这 3 种算法，16BFU 和 32BFU 的基-2NTT 和基-4 NTT 完成相同计算任务的速度相近，但基-2 NTT 的资源消耗却显著高于基-4 NTT，且相较于基-2 NTT，基-4 NTT 在吞吐量和 ATP 性能平均优化了 22.2% 和 60.6%。对于 Kyber 算法，从 4BFU 到 8BFU，

基-4 NTT的LUT和ATP分别增加2.21倍和1.51倍，而基-2 NTT分别增加3.87倍和2.34倍。相比基-2 NTT完全复用蝶形单元模块，基-4 NTT统一化蝶形单元设计在资源消耗及ATP性能上改进显著。这一差异主要归因于本文所设计的统一化基-4 NTT蝶形单元及其对内存地址访问的简化控制，显著降低了整体资源消耗。

本文采用流水线方法设计的统一化蝶形单元，通过改变BFU数量，可在编译时调整吞吐量和并行度。为提升多项式运算速度和系统吞吐量，本文采取复用蝶形单元的方法，对基-4 NTT运算实现不同参数和并行度的设计。在每层级内部复用4个PE，每个PE可依次处理2对数据，同时执行逻辑运算。整体而言，完成相同计算量时，基-4 NTT在吞吐量、ATP、计算速度等方面均优于基-2 NTT，且随着并行度的增加优势更加显著。

显然，本文提出的基-4 NTT方案具有低时延和高吞吐量的特点，能够满足云计算中大规模并发处理的需求。此外，还可通过减少并行BFU数量、优化资源利用，适应边缘计算、物联网等对低功耗和高效计算要求较高的平台。因此，该设计能够为不同应用场景提供更加灵活且高效的解决方案。

3.4　其他方案性能对比

本文对基-4 NTT 4BFU和8BFU蝶形运算进行实验，并与相同(n,q,k)参数的方案进行比较。由于对比方案实验参数多样，为公平起见，本文保留其他方案可配置性的同时，仅对比相同实验参数。如表5所示，本文方案相较于大多数其他方案来说具有较高的工作频率和计算速度。随着并行度的增加，8BFU的NTT蝶形单元相较于4BFU处理虽占用更多的硬件资源，但具有更高的速度和较优的时钟周期。

对于只支持CTC的方案，文献[9]提出了一种可扩展的基-2/基-4 NTT乘法架构，其支持多个并行蝶形单元和任意基数配置的无冲突内存映射方

表4　　基-2NTT和基-4NTT对比

参数	基数	BFU	周期	频率/MHz	时延/μs	资源消耗/个				吞吐量/(Mbit·s^{-1})	ATP
						LUT	FF	BRAM	DSP		
n=256 *q*=7 681 *k*=13	基-2	4	365	226	1.62	1 692	1 032	6	12	2 061	2.73
		8	192	194	0.99	6 454	2 637	12	16	3 363	6.399
		16	144	175	0.82	12 768	4 736	24	32	4 044	10.51
		32	95	147	0.65	45 821	10 828	48	96	5 150	29.61
	基-4	4	155	227	0.68	6 678	4 391	12	32	4 894	4.54
		8	91	195	0.47	14 758	8 745	24	64	7 081	6.94
n=256 *q*=8 380 417 *k*=23	基-2	4	327	185	1.77	3 959	2 528	6	12	3 327	7
		8	196	177	1.11	7 984	4 968	12	24	5 317	8.84
		16	142	145	0.99	16 872	6 937	24	64	6 012	16.52
		32	88	119	0.74	54 286	18 336	48	96	7 962	40.14
	基-4	4	182	227	0.8	6 678	4 391	12	32	7 360	5.34
		8	107	195	0.55	14 758	8 745	24	64	10 705	8.12
n=1 024 *q*=12 289 *k*=14	基-2	4	1 316	240	5.48	1 772	918	6	12	2 614	9.72
		8	685	193	3.55	3 858	1 786	12	24	4 039	13.69
		16	432	167	2.59	10 843	5 104	24	48	5 542	28.05
		32	273	149	1.83	16 188	9 035	48	96	7 824	29.66
	基-4	4	664	250	2.66	5 416	3 728	14	32	5 389	14.41
		8	343	225	1.52	12 836	7 213	28	64	9 432	19.51

案。该基-4方案由DSP实现计算，节省了LUT、FF开销，但内存映射的寻址和调度逻辑较为复杂，且缺乏一定的灵活性。文献[17]中的蝶形单元DSP利用率较低，同时以消耗更多的LUT和FF为代价获得时钟周期的优化，导致硬件资源消耗过多、工作频率低、整体性能较差、不具有运行时可配置性。文献[16]采用基-4 NTT架构和无冲突内存映射方案，以加速Dilithium算法中的多项式乘法，与文献[17]类似，同样以消耗更多的LUT为代价获取较优的性能。与本文方案相比，文献[18]工作频率较低，吞吐量和计算速度远不及本文方案。文献[31]提出一种高效的NTT加速器，适用于不同长度和模数的多项式，以及单个或多个PE，但在执行不同参数的方案时仍需重新编译，不支持RTC。与之相比，本文方案虽消耗资源较多，但除了提供CTC外，还支持RTC，且性能较优，能够在不同平台上进行快速部署。本文计算速度较文献[16]优化34.4%，时钟周期较文献[9]优化49.2%和48.7%，工作频率和ATP较文献[17]分别优化95.7%和54.3%，吞吐量是文献[18] 4BFU的2.42倍，能够充分发挥FPGA的计算优势。

对于只支持RTC的方案，文献[15]设计串并行可转换的运算单元，可满足实现Kyber、Saber、Dilithium和NTRU等4类算法中不同位宽基-k NTT的可扩展设计，具有高灵活性的特点。但大量的资源消耗在存储上，导致吞吐量和ATP等整体性能较差。

文献[19-20,32]均设计了具有RTC和CTC性能的乘法器。文献[19]可配置不同数量的蝶形单元，同时支持多种多项式系数而无须重新编译。该方案

表5　　NTT性能对比

方案	平台	可配置性	参数			BFU	周期	频率/MHz	时延/μs	资源消耗/个				吞吐量/(Mbit·s^{-1})	ATP
			n	q	k					LUT	FF	BRAM	DSP		
文献[9]	Virtex-7	CTC	1 024	12 289	14	4	1 308	270	4.8	1 196	969	3	12	2 987	5.74
						8	668	250	2.7	2 953	1 875	5.5	27	5 310	7.97
文献[15]	Artix-7	RTC	256	8 380 417	23	4	896	152	5.90	13 794	—	3×36K	48	998	81.38
			256	—	13		243		1.60					2 080	22.07
文献[16]	Zynq7000	CTC	256	8 380 417	23	4	264	217	1.22	2 386	932	1	8	4 826	2.91
文献[17]	Artix-7	CTC	256	8 380 417	23	4	300	116	2.59	4 509	3 146	—	8	2 273	11.68
文献[18]	Virtex-7	CTC	1 024	—	14	8	680	125	5.4	2 584	—	16	24	2 655	13.95
文献[19]	Artix-7	CTC&RTC	256	—	24	8	163	140	1.16	11.1k	8.7k	4	32	5 297	12.90
文献[20]	Virtex-7	CTC&RTC	256	—	13	8	156	186	0.84	11k	5 422	12	64	3 692	9.24
	Artix-7		256	—	23		156	140	1.11	11k	5 182	12	64	5 305	12.21
文献[31]	Virtex-7	CTC	256	—	13	4	270	278	1.0	1 288	888	4.5	12	3 328	1.29
						8	135	256	0.5	6 245	1 864	12	24	6 656	3.12
文献[32]	Artix-7	CTC&RTC	256	8 380 417	23	4	263	201.9	1.3	1 688	672	4	4	4 529	2.19
			256	7 681	13									2 560	
本文	Virtex-7	CTC&RTC	1 024	12 289	14	4	664	250	2.66	5 416	3 728	14	32	5 389	14.41
						8	343	225	1.52	12 836	7 213	28	64	9 432	19.51
	UltraScale		256	8 380 417	23	4	182	227	0.80	6 678	4 391	12	32	7 360	5.34
			256	3 329	13		155		0.68					4 894	4.54
			256	8 380 417	23	8	107	195	0.55	14 758	8 745	24	64	10 705	8.12
			256	3 329	13		91		0.47					7 081	6.94

通过重用旋转因子，显著降低了BRAM和DSP的消耗，然而为实现ATP的高效性和整体的可配置性，在多项式乘法架构及可重构处理元件上消耗了大量的硬件资源。文献[20]通过串行处理系数可以跳过不必要的操作，但随着多项式系数的增加会降低相应的数据位宽，导致吞吐量受限，在Dilithium算法实现中，本文的吞吐量是文献[20]的2倍。文献[32]深入分析了主流格基密码算法中NTT技术的应用，并针对不同并行度进行可重构优化实现。相较于文献[32]的4BFU计算，本文在计算速度和吞吐量方面实现了近2倍的增长。从表5可以看出，在同时提供RTC和CTC的情况下，本文方案在时钟周期、计算速度、工作频率以及吞吐量方面均显著优于其他方案中的NTT设计，充分证明了本文在基-4蝶形单元及内存优化存储等方面设计的可行性。

总体而言，本文提出的高效可配置基-4NTT多项式乘法设计方案在FPGA硬件实现上具有较高的工作频率，且相较于大多数对比方案，吞吐量和计算速度性能较优。这是由于本文对基-4算法的核心、模约减及多RAM存储结构进行优化，提高了整体性能，同时支持多模值可并行的整体设计并提供CTC及RTC性能。

4 结束语

本文提出了一种基于格密码学的可配置基-4 NTT硬件结构，该结构适用于多种格密码方案的参数，具有运行时可配置性，同时其编译时可配置性支持多个蝶形单元并行化设置，能够满足多种应用场景中的性能和功耗需求，具有广泛的适用性。实验分析及结果表明，本文方案有较高的工作频率和计算速度，且兼顾CTC和RTC，能够充分发挥FPGA的计算优势，整体性能优于现有方案，具有一定的借鉴意义和应用价值。

在未来的工作中，笔者将进一步探索可兼容多种基数的紧凑型统一化蝶形架构，以及重点研究如何设计一个安全的侧信道保护机制，将其集成到后量子密码算法的硬件架构中。

参考文献：

[1] SHOR P W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer[J]. SIAM Review, 1999, 41(2): 303-332.

[2] ALAGIC G, APON D, COOPER D, et al. Status report on the third round of the NIST post-quantum cryptography standardization process[R]. 2022.

[3] BERNSTEIN D J, LANGE T. Post-quantum cryptography[J]. Nature, 2017, 549(7671): 188-194.

[4] KUMAR A, OTTAVIANI C, GILL S S, et al. Securing the future Internet of things with post-quantum cryptography[J]. Security and Privacy, 2022, 5(2): 1-11.

[5] NEJATOLLAHI H, DUTT N, RAY S, et al. Post-quantum lattice-based cryptography implementations[J]. ACM Computing Surveys, 2019, 51(6): 1-41.

[6] 李延斌, 朱嘉杰, 唐明, 等. 面向格密码的能耗分析攻击技术[J]. 计算机学报, 2023, 46(2): 331-352.
LI Y B, ZHU J J, TANG M, et al. Power analysis attacks for lattice-based cryptography[J]. Chinese Journal of Computers, 2023, 46(2): 331-352.

[7] ZENG C K, HE D B, FENG Q, et al. The implementation of polynomial multiplication for lattice-based cryptography: a survey[J]. Journal of Information Security and Applications, 2024, 83: 103782.

[8] LI B, YAN Y F, WEI Y X, et al. Scalable and parallel optimization of the number theoretic transform based on FPGA[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2024, 32(2): 291-304.

[9] CHEN X R, YANG B H, YIN S Y, et al. CFNTT: scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021: 94-126.

[10] GARRIDO M. A survey on pipelined FFT hardware architectures[J]. Journal of Signal Processing Systems, 2022, 94(11): 1345-1364.

[11] ZHANG C, LIU D S, LIU X J, et al. Towards efficient hardware implementation of NTT for kyber on FPGAs[C]//Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS). Piscataway: IEEE Press, 2021: 1-5.

[12] MA L J, WU X J, BAI G Q. Parallel polynomial multiplication optimized scheme for CRYSTALS-KYBER post-quantum cryptosystem based on FPGA[C]//Proceedings of the 2021 International Conference on Communications, Information System and Computer Engineering (CISCE). Piscataway: IEEE Press, 2021: 361-365.

[13] XING Y F, LI S G. An efficient implementation of the NewHope key exchange on FPGAs[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(3): 866-878.

[14] MERT A C, ÖZTÜRK E, SAVAŞ E. FPGA implementation of a run-time configurable NTT-based polynomial multiplication hardware[J]. Microprocessors and Microsystems, 2020, 78: 103219.

[15] 陈韬, 李慧琴, 李伟, 等. 面向格基后量子密码算法的可重构多项式乘法架构[J]. 电子与信息学报, 2023, 45(9): 3380-3392.
CHEN T, LI H Q, LI W, et al. Reconfigurable polynomial multiplication architecture for lattice-based post-quantum cryptography algorithms[J]. Journal of Electronics & Information Technology, 2023, 45 (9): 3380-3392.

[16] WANG T F, ZHANG C, CAO P, et al. Efficient implementation of dilithium signature scheme on FPGA SoC platform[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022, 30(9): 1158-1171.

[17] BECKWITH L, NGUYEN D T, GAJ K. High-performance hardware implementation of CRYSTALS-dilithium[C]//Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT). Piscataway: IEEE Press, 2021: 1-10.

[18] MERT A C, KARABULUT E, ÖZTÜRK E, et al. An extensive study of flexible design methods for the number theoretic transform[J]. IEEE Transactions on Computers, 2022, 71(11): 2829-2843.

[19] HU X, TIAN J, LI M H, et al. AC-PM: an area-efficient and configurable polynomial multiplier for lattice based cryptography[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(2): 719-732.

[20] DERYA K, MERT A C, ÖZTÜRK E, et al. CoHA-NTT: a configurable hardware accelerator for NTT-based polynomial multiplication[J]. Microprocessors and Microsystems, 2022, 89: 104451.

[21] IMRAN M, KHAN S, KHALID A, et al. Evaluating NTT/INTT implementation styles for post-quantum cryptography[J]. IEEE Embedded Systems Letters, 2024, PP(99): 1.

[22] ABDULRAHMAN A, HWANG V, KANNWISCHER M J, et al. Faster kyber and dilithium on the cortex-M4[C]//International Conference on Applied Cryptography and Network Security. Berlin: Springer, 2022: 853-871.

[23] MERT A C, KARABULUT E, ÖZTÜRK E, et al. A flexible and scalable NTT hardware: applications from homomorphically encrypted deep learning to post-quantum cryptography[C]//Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). Piscataway: IEEE Press, 2020: 346-351.

[24] WANG W, TIAN S Q, JUNGK B, et al. Parameterized hardware accelerators for lattice-based cryptography and their application to the HW/SW co-design of qTESLA[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 269-306.

[25] MAO G Y, CHEN D L, LI G Y, et al. High-performance and configurable SW/HW co-design of post-quantum signature CRYSTALS-dilithium[J]. ACM Transactions on Reconfigurable Technology and Systems, 2023, 16(3): 1-28.

[26] SATRIAWAN A, SYAFALNI I, MARETA R, et al. Conceptual review on number theoretic transform and comprehensive review on its implementations[J]. IEEE Access, 2023, 11: 70288-70316.

[27] 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: 49-72.

[28] WONG Z Y, WONG D C K, LEE W K, et al. KaratSaber: new speed records for saber polynomial multiplication using efficient karatsuba FPGA architecture[J]. IEEE Transactions on Computers, 2023, 72(7): 1830-1842.

[29] GUPTA N, JATI A, CHATTOPADHYAY A, et al. Lightweight hardware accelerator for post-quantum digital signature CRYSTALS-dilithium[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(8): 3234-3243.

[30] TAKALA J, PUNKKA K. Butterfly unit supporting radix-4 and radix-2 FFT[C]//Proceedings of The 2005 International TICSP Workshop on Spectral Methods and Multirate Signal Processing. Piscataway: IEEE Press, 2005: 47-54.

[31] MU J N, REN Y, WANG W, et al. Scalable and conflict-free NTT hardware accelerator design: methodology, proof, and implementation[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(5): 1504-1517.

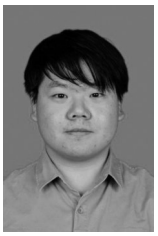
[32] 赵旭阳, 梁志闯, 胡跃, 等. NTT架构研究及其FPGA硬件优化实现[J]. 计算机学报, 2023, 46(12): 2670-2686.
ZHAO X Y, LIANG Z C, HU Y, et al. NTT architecture research and its FPGA hardware optimization implementation[J]. Chinese Journal of Computers, 2023, 46(12): 2670-2686.

[作者简介]

周清雷（1962– ），男，河南郑州人，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。

韩贺茹（1999– ），女，河南驻马店人，郑州大学硕士生，主要研究方向为后量子密码、高性能可重构计算。

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。



刘宇航（1999– ），男，河南商丘人，郑州大学硕士生，主要研究方向为隐私保护计算、可重构硬件架构设计。