RPMA:面向 CRYSTALS-Kyber 的高效可重构多项式乘法器

XXXXX

(1.　长沙 410073；2 3. 长沙 410073）

摘　要 后量子密码算法 CRYSTALS-Kyber 的高效硬件实现对于其广泛部署至关重要，而多项式乘法是其核心运算瓶颈。为克服现有设计在灵活性与面积效率上的局限，本文提出了一种面积高效的可重构多项式乘法器架构（RPMA）。首先，提出一种改进的多通道 NTT/INTT/点乘（PWM）算法，通过对循环控制逻辑解耦，实现了对任意并行度的统一支持。其次，引入一种基于交织存储的无冲突内存映射方案，该方案与多项式长度与计算阶段无关，统一了 NTT/INTT/PWM 的访存模式。此外，针对多通道 NTT 流水线中的写后读冲突，给出了形式化建模与冲突检测算法，推导出同时避免空泡与 RAW 冲突的约束条件，从而最大化流水线效率。最后，基于上述方法设计支持 NTT/INTT/PWM 的可重构蝶形计算单元，实现底层算子的最大复用，并构建可按需配置并行度与流水深度的多项式乘法器架构（RPMA），支持面积与性能间的灵活权衡。实验结果表明，与现有可重构工作相比，本文提出的方案在面积-时间积（ATP）上降低了 12.97%~77.43%。

关键词：后量子密码学；多项式乘法器；NTT；可重构；CRYSTALS-Kyber。

1 引言

随着量子计算技术的快速发展，基于Shor算法等量子攻击手段对传统公钥密码体系（如RSA和ECC）构成了实质性威胁[1]。为应对该挑战，美国国家标准与技术研究院于2016年启动了后量子密码学标准化进程[2]，并于2024年将基于模学习错误问题的CRYSTALS-Kyber正式纳入标准[3]。CRYSTALS-Kyber的安全性构建在模学习错误问题之上，其核心运算涉及环上高次多项式乘法。然而，高次多项式乘法计算复杂度高、内存消耗大、处理速度慢，是制约其实际应用的主要瓶颈[4,5]。

为提升多项式乘法效率，数论变换（Number Theoretic Transform，NTT）被广泛采用。然而，将NTT算法高效映射到硬件架构时，仍面临如何在硬件利用率、架构灵活性（可配置性）与内存访问效率三者间取得最佳平衡的关键挑战。在硬件实现中，NTT的循环展开策略直接决定了其结构形态。现有研究主要可以归为三类典型硬件结构：多路延迟换向结构[6,7]（MDC）、脉动阵列结构[8]以及多通道NTT结构[9–13](内存迭代NTT)。其中，多通道NTT通过并发执行同一阶段内的多个蝶形运算，理论上可实现100%硬件利用率，并在面积与性能间实现灵活权衡，已成为当前主流的高性能实现方案。然而，NTT算法的三层循环天然存在耦合，导致难以实现统一的可配置并行化设计。现有研究大多只是针对特定并行度而设计专门的控制逻辑[9,12,14–16]，缺乏架构的灵活性。另一方面，NTT算法中内存访问冲突也是制约其并行化的关键问题。这些冲突主要体现为写后读冲突[17]（Read After Write，RAW）和结构冲突[11]。RAW源于流水线设计中数据依赖导致的冒险，只能通过插入空操作避免。结构冲突则由于NTT各阶段中多项式系数访问顺序动态变化引起，并且随着并行度的提升而变得更加显著。当前许多研究[18–22]往往针对不同并行度或不同NTT阶段专门设计单独的访存控制策略，增加了硬件开销。此外，许多高性能多项式乘法器常采用全

流水线设计以提高系统性能。然而，随着并行度和流水深度的变化，会出现不同程度的流水线冲突，导致执行效率降低。因此，针对上述问题，本文对NTT算法进行深度优化，具体贡献总结如下：

1）提出了一种改进的多通道NTT/INTT/PWM算法，通过对循环控制进行解耦，统一了不同并行度下的控制逻辑设计，支持任意并行度展开。

2）提出了一种基于交织存储策略的无冲突内存映射方案，统一了CRYSTALS-Kyber中NTT/INTT/PWM的访问模式。该方案独立于多项式长度和计算阶段，并能支持任意并行度。

3）提出了一种针对多通道NTT流水线设计的冲突检测算法。通过对多通道NTT中可能存在的RAW冲突进行建模，给出了能同时避免空泡和RAW冲突的约束条件，从而最大化流水线效率。

4）基于上述算法，设计了一种可以支持NTT/INTT/PWM的可重构蝶形计算单元，实现了底层算子最大复用。在此基础上，提出了一种支持并行度和流水级深度灵活配置的多项式乘法器架构（RPMA），在面积和性能之间取得了更好的权衡。

本文其余部分组织如下：第二节介绍基于NTT的多项式乘法并给出了动机；第三节介绍提出的无冲突访存方案；第四节详细介绍提出的循环解耦NTT/INTT/PWM算法；第五节给出了流水线冲突检测算法；第六节详细介绍了所提出RPMA架构；第七节提供了实现结果和对比分析；第八节对全文进行总结。

2 初步、分析和动机

2.1 基于NTT的多项式乘法（给出数据流图）

多项式乘法作为 CRYSTALS-Kyber 算法的主要计算瓶颈，其执行效率和资源消耗将直接影响整个算法部署。在基于格的密码学方案中，多项式乘法是定义在多项式环 $\boldsymbol{R}_q = Z_q[x]/(x^N+1)$ 上的，其中 $N=2^n$。$\boldsymbol{R}_q$ 中多项式的次数为 N-1，系数在整数域 Z_q 上。

目前，通常采用 NTT 对多项式乘法进行加速。NTT 可以看做离散傅里叶变换在整数域上的变形，假设 w 为 Z_q 上的 N 次本原根，满足 $w^N \bmod q = 1$ 且 $\forall i < N, w^i \bmod q \neq 1$，其中 $q \equiv 1 \bmod N$。假设多项式 $\boldsymbol{f}(x)$ 的 NTT 形式为 $\hat{\boldsymbol{f}}(x) = \mathrm{NTT}(\boldsymbol{f}) = \sum_{i=0}^{n-1} \hat{f}_i x^i$，$\boldsymbol{f}(x)$ 称为 $\hat{\boldsymbol{f}}(x)$ 逆 NTT（Inverse Number Theoretic Transform，INTT）形式。NTT 和 INTT 过程如下：

$$\mathrm{NTT}: \hat{f}_i = \sum_{j=0}^{N-1} f_j w^{ij} \bmod q, \quad i = 0,1,\cdots,N-1 \tag{1}$$

$$\mathrm{INTT}: f_i = \frac{1}{N} \sum_{j=0}^{N-1} \hat{f}_j w^{-ij} \bmod q, \quad i = 0,1,\cdots,N-1 \tag{2}$$

上述过程可以将多项式乘法的复杂度 $O(N^2)$ 从降低到 $O(N\log_2 N)$[23]。当满足 $q \equiv 1 \bmod 2N$ 时，可以进一步采用负包裹卷积[24]（NWC）避免系数填充和以 (x^N+1) 为模的约减操作。而在最新的 FIPS 203 标准中，为了缩减 CRYSTALS-Kyber 的公钥和密文长度，将模数 q 设置为更小的 3329[3]。但是 Z_{3329} 中不存在 512 次本位根，即不满足 $q \equiv 1 \bmod 2N$，没法正常执行 NTT 运算。为了解决这个问题，Seiler 等人[25]采用环同构重新定义了 NTT，如公式（3）所示：

$$\begin{aligned}\hat{f}_{2i} &= \sum_{j=0}^{127} f_{2j}\psi^{(2\mathrm{BR}(i)+1)j} \bmod q \\ \hat{f}_{2i+1} &= \sum_{j=0}^{127} f_{2j+1}\psi^{(2\mathrm{BR}(i)+1)j} \bmod q\end{aligned} \tag{3}$$

其中 $\psi=17$ 为 Z_{3329} 中的 256 次本位根，BR(i)的功能为将 7bit 无符号数 i 的 bit 位顺序反转。因此，CRYSTALS-Kyber 中完整 NTT 算法流程如算法 1 所示。

算法 1 原始 NTT 算法

Input: 多项式系数向量 $\boldsymbol{f}=(f_0,f_1,\cdots,f_{N-1})$
N 次本原根 ψ

Output: $\hat{\boldsymbol{f}} \leftarrow \mathrm{NTT}(\boldsymbol{f})$

1:Initial k=1, $\hat{\boldsymbol{f}} \leftarrow \boldsymbol{f}$
2:**for**(i=0; i<n-1; i=i+1) **do** ←**loop-i**
3: $J=2^{n-1-i}$
4: **for**(s=0; s<N; s=s+2J) **do** ←**loop-s**
5: $w=\psi^{BR(k)} \bmod q$
6: k=k+1
7: **for**(j=s; j<s+J; j=j+1) **do** ←**loop-j**
8: /*蝶形运算*/
9: $t=\hat{f}_{j+J}\times w$
10: $\hat{f}_{j+J}=(\hat{f}_j-t) \bmod q$
11: $\hat{f}_j=(\hat{f}_j+t) \bmod q$
12: **endfor**
13: **endfor**
14:**endfor**
15: **return** $\hat{\boldsymbol{f}}$

在 CRYSTAKS-Kyber 中，两个处于 NTT 域表示形式的多项式向量乘法由一系列点乘（PWM）组成。设 $\hat{\mathbf{f}}=(\hat{f}_0,\hat{f}_1,\cdots\hat{f}_{255})$ 和 $\hat{\boldsymbol{g}}=(\hat{g}_0,\ \hat{g}_1,\cdots\hat{g}_{255})$ 分别为 NTT 域多项式点值形式，$\hat{\mathbf{h}}=(\hat{h}_0,\hat{h}_1,\cdots\hat{h}_{255})$ 为向量乘法结果。由于模数的特殊性，CRYSTAKS-Kyber 中的 PWM 不同于其他格基密码方案，并不是 $\hat{\mathbf{f}}=(\hat{f}_0,\hat{f}_1,\cdots\hat{f}_{255})$ 和 $\hat{\boldsymbol{g}}=(\hat{g}_0,\ \hat{g}_1,\cdots\hat{g}_2)$ 对应点的直接乘法，而是 $Z_q[x]/(x^2-\psi^{2BR(i)+1})$ 中两个一次多项式乘法。即：

$$\begin{aligned}\hat{h}_{2i}+\hat{h}_{2i+1}X &= (\hat{f}_{2i}+X\hat{f}_{2i+1})\cdot(\hat{g}_{2i}+X\hat{g}_{2i+1}) \bmod (X^2-\psi^{2\mathrm{BR}(i)+1}) \\ &= (\hat{f}_{2i}\hat{g}_{2i}+X^2\hat{f}_{2i+1}\hat{g}_{2i+1}+X\hat{f}_{2i}\hat{g}_{2i+1}+X\hat{f}_{2i+1}\hat{g}_{2i}) \bmod (X^2-\psi^{2\mathrm{BR}(i)+1}) \\ &= (\hat{f}_{2i}\hat{g}_{2i}+\hat{f}_{2i+1}\hat{g}_{2i+1}\psi^{2\mathrm{BR}(i)+1}+X(\hat{f}_{2i}\hat{g}_{2i+1}+\hat{f}_{2i+1}\hat{g}_{2i})) \bmod (X^2-\psi^{2\mathrm{BR}(i)+1})\end{aligned} \tag{4}$$

因此对于 $\hat{\mathbf{h}}$ 中元素 $\hat{h}_{2i},\hat{h}_{2i+1}$，可以表示为：

$$\begin{aligned}\hat{h}_{2i} &= \hat{f}_{2i}\hat{g}_{2i}+\hat{f}_{2i+1}\hat{g}_{2i+1}\psi^{2BR(i)+1} \bmod q \\ \hat{h}_{2i+1} &= \hat{f}_{2i}\hat{g}_{2i+1}+\hat{f}_{2i+1}\hat{g}_{2i} \bmod q\end{aligned} \tag{5}$$

综上，环上多项式乘法的计算公式为：$\boldsymbol{h}(x)=\mathrm{INTT}(\mathrm{PWM}(\mathrm{NTT}(\boldsymbol{f}(x)),NTT(\boldsymbol{g}(x))))$。

2.2 分析（通过多种结构引出动机，多通道设计，内存映射方案，以及流水线效率）

基于算法 1，图 1 给出了一个 16 次多项式的 NTT 信号流图。为了便于描述，作如下定义，循环变量

i 表示迭代阶段数，被称为 stage; 循环变量 s 表示每个 stage 的分块步长，被称为块索引；循环变量 j 负责分块内计数，被称为块内索引。通过对该流图进行不同程度的展开，可形成多种硬件架构。NTT 和 INTT 的核心均由蝶形运算构成，因此最小的展开粒度为蝶形单元（Butterfly unit, BFU）（见算法 1 第 9–11 行）。总的来说，存在以下三种典型展开策略：

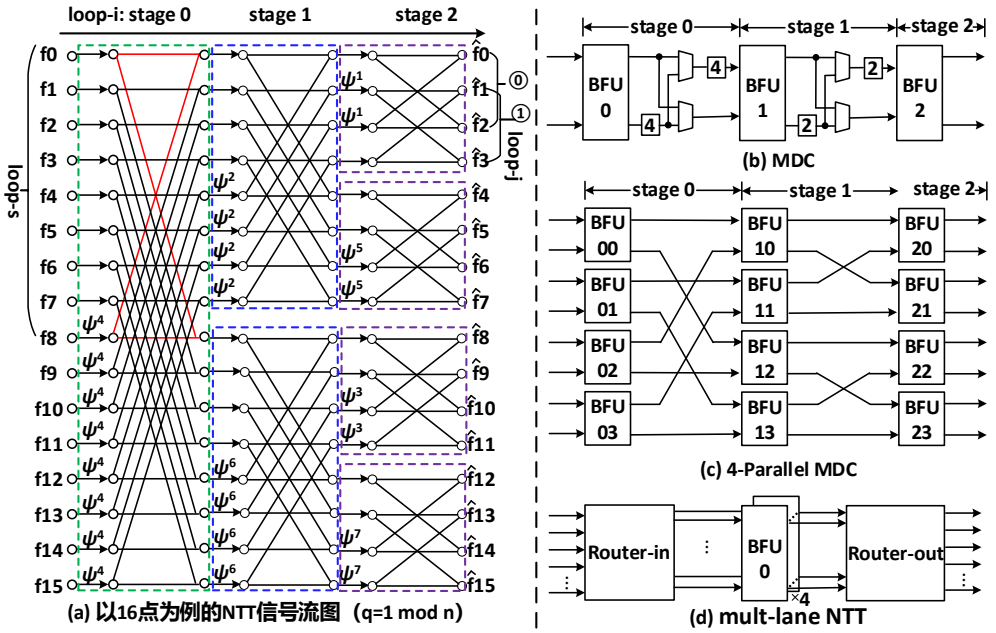

图 1 NTT 信号流图及其硬件结构

（1）仅展开 loop-i：由于各 stage 间存在数据依赖。只能在流水级插入单延迟反馈缓冲或者多延迟换向来避免数据冲突。在这种展开方式中，多路延迟换向结构（MDC）被广泛采用[6]，如图 1（b）所示。

（2）展开 loop-i 的同时对 loop-s 和 loop-j 做部分展开或者完全展开：由此形成多并行度 MDC（Parallel-MDC）或者也称为脉动阵列结构[8]，如图 1（c）所示。

（3）同时展开 loop-s 与 loop-j[9–11]：尽管 loop-s 和 loop-j 之间多项式系数实际上不存在数据依赖关系，loop-s 只控制 loop-j 块大小，但由于不同 stage 的分块步长会逐级减半，导致难以单独展开 loop-j。因此，需同时对 loop-s 和 loop-j 进行展开，其结构如图 1（d）所示。

对上述三种结构进行硬件资源利用率分析（暂不考内存访问冲突）：MDC 结构在蝶形单元间插入延迟寄存器，通过同一蝶形单元迭代执行同一 stage 内的运算，然后在不同蝶形单元间形成流水，其流水线如图 2（a）所示。在 16 点长情况下，蝶形单元的利用率仅为 57.14%。Parallel-MDC 结构如图 2（b）所示，尽管能显著压缩计算时间，但其硬件利用率只有 50%，且需要庞大的硬件资源。最后一种展开方式的流水线如图 2（c）所示，因相同 stage 内的所有蝶形操作不存在数据依赖关系，可以对该结构的并行度进行配置，理论上可达到 100%的硬件利用率，从而在面积和性能之间实现有效权衡。

前两种结构除了蝶形单元硬件资源利用率较低外，还需要消耗双倍的存储空间。并且蝶形单元间形成了互联结构，灵活性差，不利于可重构设计和可配置设计。相较而言，多通道 NTT 结构灵活度高，可重构性强，并能最大化复用底层算子。因此，目前主流的 NTT 硬件加速方案多采用第三种展开方式。

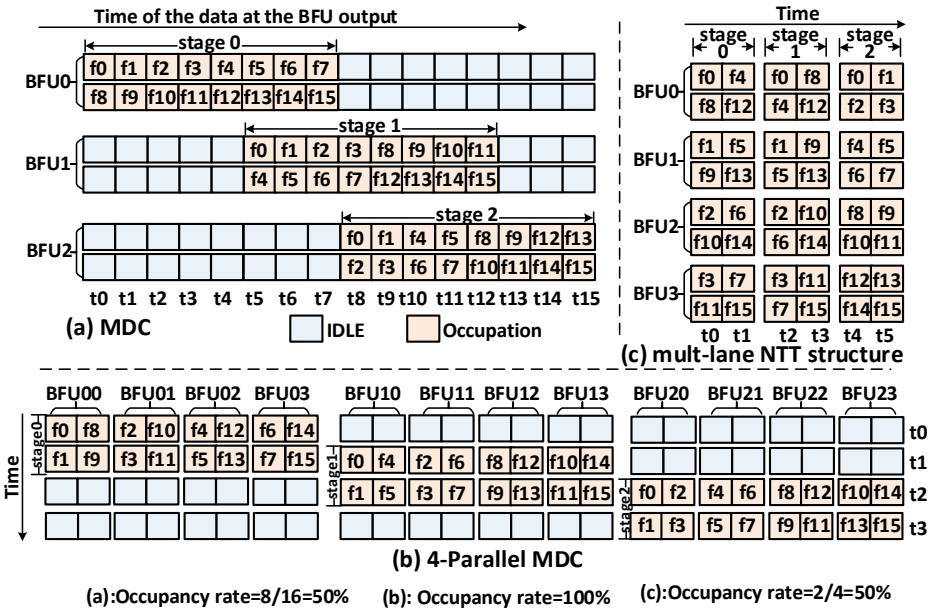

图 2 不同 NTT 硬件结构中蝶形单元利用率分析

然而，多通道 NTT 结构存在严重的访问冲突问题，主要为 RAW 和结构冲突，分别如图 3 和图 4 所示。前者属于真冲突，只能通过插入空操作解决，会造成硬件资源利用率下降[26]。后者则归结于 stage 变化时访问地址步长逐级减半，并行度越高冲突越严重。

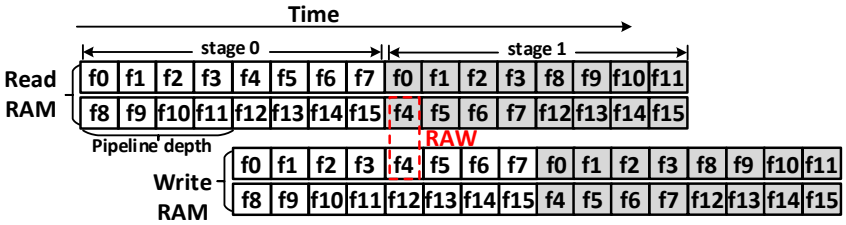

图 3 多通道 NTT 结构存在的写后读冲突

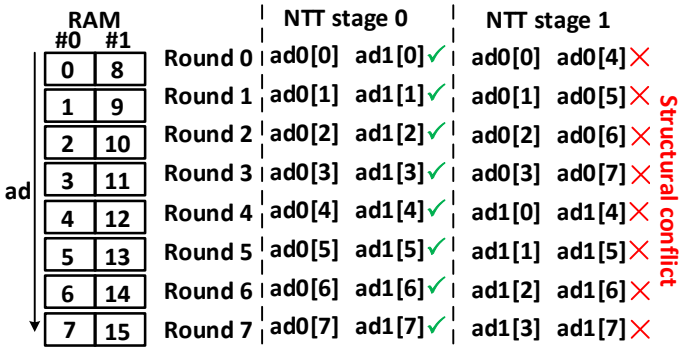

图 4 多通道 NTT 结构存在的结构冲突

2.3 动机

首先，根据 2.2 节分析，NTT 中同一阶段的蝶形运算具备任意并行化的潜力。然而，该算法的三层循环间存在天然耦合，导致现有研究大多局限于针对特定并行度设计专门的控制逻辑[9,12,14–16]，缺乏可灵活配置并行度的多通道 NTT 多项式乘法器方案。其中，Chen 等人[11]提出的可扩展迭代 NTT 算法与支持

任意并行度的内存映射方案，最接近实现通用多通道 NTT 架构的目标。但其方法在并行度大于 loop-s 分块步长时需要引入额外判断，未能彻底解决循环间的耦合问题。实际上，通过分析循环间的依赖关系，可将地址生成相关变量移至最内层循环，并重构循环控制逻辑以实现解耦。因此，本文提出了一种循环解耦的多通道 NTT 算法，支持任意并行度的灵活展开。同时统一了 NTT/INTT/PWM 的控制逻辑，为实现可重构硬件架构奠定了基础。

其次，设计一种与并行度和 NTT/INTT 阶段无关的无冲突内存映射方案至关重要。现有解决方案主要分为三类：一、混合洗牌方案[18,19,22,27]通过分别控制各阶段读写地址来避免冲突，但需引入乒乓内存，控制逻辑复杂，难以灵活适配不同并行度。二、系数置换网络依赖对不同 stage 写回数据进行重排实现无冲突访问[20,21]，对于不同并行度和阶段需要设计不同重排方式，灵活性受限。三、交织存储系统仅通过特定映射规则即可支持任意并行度的无冲突内存访问[11]，灵活性高。然而，这些方案均充分考虑 CRYSTALS-Kyber 逐点乘法的特殊访存需求，例如 Wang 等人[10]和 Chen 等人[11]仅实现朴素点乘，Guo 等人[14]虽统一支持 NTT/INTT/PWM，但仅限于特定并行度，而 Krieger 等人[28]仅实现了 NTT。为此，本文提出了一种无冲突内存映射方案，统一了不同并行度下 NTT/INTT/PWM 的内存结构和访问模式。

第三、无空泡全流水线设计可以有效提高硬件利用率。如果在多通道 NTT 的蝶形单元中插入了流水线，连续执行蝶形运算，可能会造成 RAW。虽然已有工作给出了判决条件[11]，但并未给出详细的冲突建模。因此本文对 NTT 流水线设计存在的冲突进行了数学建模，并给出了冲突检测算法。

最后，采用可重构且面积高效的硬件架构能够显著提高资源利用率。相较于非可重构设计需为 NTT、PWM 和 INTT 分别配置独立处理单元（Process Element，PE），可重构架构能够在统一的 PE 上动态执行这三种运算，从而将资源消耗降至非可重构设计的约三分之一。此外，非可重构架构在执行单一运算时，其余运算对应的计算单元将处于闲置，导致整体硬件利用率偏低。为此，本文设计了一种可同时支持 NTT、INTT 和 PWM 运算的可重构蝶形单元，并在此基础上构建了一种面积高效的多项式乘法器架构，支持编译时可配置并行度与流水线深度。

3 基于交织存储的无冲突内存映射方案

为了避免复杂的数据路由电路以及本位型 NTT 对乒乓内存的需求，同时兼顾 CRYSTALS-Kyber 中的 PWM。为此，本节首先回顾 Takala 等人[29]提出的线性变换框架，并提出了一种基于交织存储的无冲突内存映射算法，该算法采用简单的控制逻辑即可实现地址映射。

基于交织存储策略的内存映射方案，需要借助线性变换将原数据点索引映射到多 bank 存储系统下的地址索引（BI, BA），其中 BI 是 bank 号，BA 是对应 bank 的地址。Takala 等人[29]给出了满足多项式长度为 $N=2^n$，bank 个数为 $M=R\times p=2^m$ 一种线性变换技术，其中 R 为基数，p 为蝶形单元个数。设多项式某个系数的地址为 a，将地址 a 的二进制位视作向量 $\boldsymbol{a}=(a_{n-1},a_{n-2},\cdots a_0)$，借助行地址变换矩阵 $\boldsymbol{R}$ 和 Bank 映射矩阵 $\boldsymbol{T}_{n,m}$ 实现地址映射，映射规则如下：

$$BA=\boldsymbol{R}\times\boldsymbol{a}^T \tag{6}$$

$$BI_i=\boldsymbol{T}_{n,m}(i)\times\boldsymbol{a}^T=\bigoplus_{k=0}^{h_{n,m}(i)}a_{(k\cdot m+i)\bmod n},i=0,1,\cdots m-1 \tag{7}$$

$$h_{n,m}(i)=\lfloor (n+m-gcd(m,n \bmod m)-i-1)/m \rfloor \tag{8}$$

其中，行地址变换矩阵 $\boldsymbol{R}=(\boldsymbol{I}_{n-m}\boldsymbol{0}_{n-m,m})$，$\boldsymbol{I}_{n-m}$ 为 $n-m$ 阶单位矩阵，$\boldsymbol{0}_{n-m,m}$ 表示 $(n-m)\times m$ 阶零矩阵。该矩阵用于提取原地址的高 $n-m$ 位。Bank 号映射矩阵 $\boldsymbol{T}_{n.m}$，可以通过整理 BI_i 和原地址 $\boldsymbol{a}$ 二进制表达式得出。以 $N=32$，$M=4$ 为例，

$$\begin{aligned} BI_0 &= a_4 \oplus a_2 \oplus a_0 \\ BI_1 &= a_3 \oplus a_1 \oplus a_0 \end{aligned} \tag{9}$$

异或运算等价于有限域下的模 2 运算，因此可以转换为如下矩阵形式：

$$\begin{bmatrix} BI_1 \\ BI_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \Leftrightarrow BI=\boldsymbol{T}_{32,4}\times \boldsymbol{a}^T \tag{10}$$

Chen 等人[11]在此基础上提出了一种适合任意基数 NTT 的交织存储策略内存映射方案，但并未考虑 CRYSTALS-Kyber 中 PWM 的特殊性。Guo 等人[14]将其改进并提出一种 radix-2/4 混合基 NTT 算法，并同时考虑了 CRYSTALS-Kyber 中 PWM 的特殊性，然而其提出的映射方案为 radix-2/4 混合基下固定 PE 数的映射方案，并不具备可配置性。因此，本文在此基础上提出了一种适合任意点长，通道数为 2 的幂次倍数的无冲突内存映射方案，算法如下：

算法 2 无冲突地址映射算法（conflict_free_map）

Input: 原地址 a，并行度 p，点长 N
Input: 模式选择 mode，地址偏移 offset
预先计算：M=2×p, n= $\log_2 N$，ps=$\log_2 p$，m=$\log_2 M$
Output:索引(BI, BA)
1: t= ^a[n-1:m]　/*^按位异或*/
2: S= t<<ps
3: BI=(a[m-1:0]+S) mod M
4: **if** mode==1
5:　BI=(BI+p) mod M
6: **endif**
7: BA=a[n-1, m]+offset×N/M
9: **return** (BI, BA)

根据算法 2，图 5 展示了 32 点基 2 NTT 在 p=2 时的多项式系数映射规则与 NTT 运算调度过程。图中 bank 内数字 0-31 为对应多项式系数的地址索引。可以看出，在不同 NTT 阶段，多项式系数在 bank 中的存储形式始终保持不变。相同颜色标记的四个数据块表示需要同时访问的系数组合，例如在 stage 0 的第零轮蝶形运算所需的四个系数 0、16、1、17 分布在不同 BI 号的 bank 中，可以实现同时读取。遍历整个 NTT 计算过程，该映射方案均满足无冲突访问。

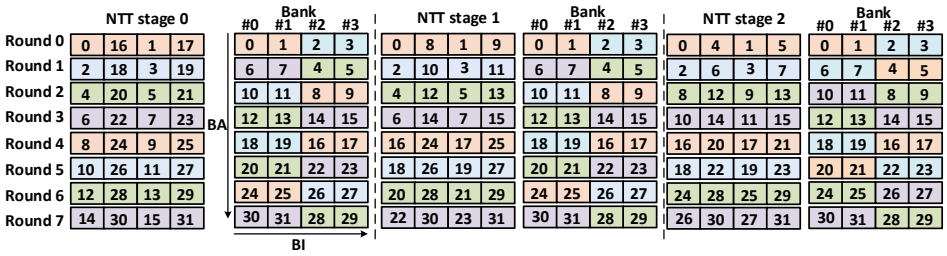

图 5 无冲突映射下的 NTT 运算调度过程（以 32 点基 2 NTT，p=2 为例）

图 6 展示了 PWM 的运算调度过程。由于 PWM 涉及两组多项式系数间的运算，为避免结构冲突，第二组多项式系数的存储位置不能和第一组重合。通过 mode 配置将第二组系数的 BI 号模 M 循环右移 p 位，可使得需要同时访问的两种组多项式系数分布在不同 BI 号内，如图 6 中相同颜色虚线框选的部分。

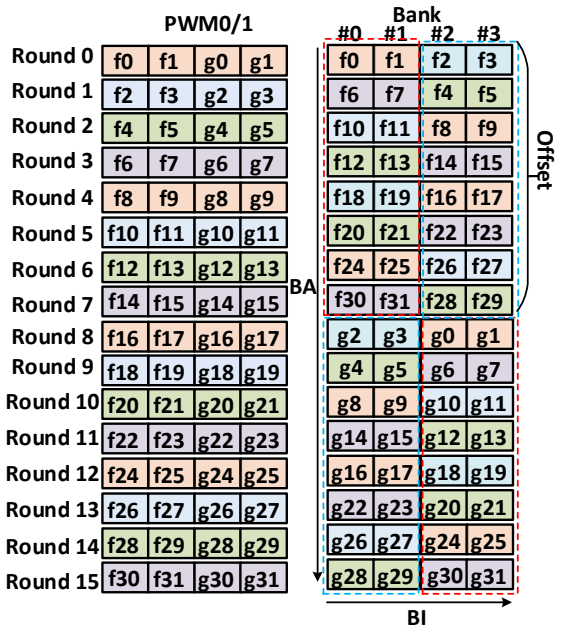

图 6 无冲突映射下的 PWM 运算调度过程

上述算法可以通过配置并行度 p、点长 N、模式选择 mode 以及地址偏移 offset，从完全避免 NTT/INTT/PWM 计算过程的结构冲突。此外，由于并行度 p、点长 N 都是 2 的幂次，因此算法中所有的取模运算，乘法运算以及除法运算都可以通过位运算实现，大幅度降低了硬件资源开销。

4 基于循环解耦的多通道 NTT 算法

4.1 多通道并行 NTT/INTT

深入分析算法 1 的三层循环可以发现，分块步长 J 随 stage 逐级减半，且 loop-s 和 J 之间存在耦合，同时 loop-j 又依赖于 loop-s，导致 loop-s 和 loop-j 的循环次数均不恒定，即不完美循环。这种依赖关系进一步影响了旋转因子和多项式系数的地址生成。因此，在对 loop-s 和 loop-j 进行展开或者部分展开时，必然会出现并行度大于分块步长 J 的情况，导致需要额外的地址生成控制逻辑，难以实现可配置的并行化。

由图 1 可以看出，在同一个 stage 内蝶形运算完全具备任意并行度展开的潜力，只要能够保证所有蝶形运算的系数能够被正确访问。因此，实现任意并行度展开的关键在于将系数地址生成逻辑同循环变量 s、

j 解耦。遗憾的是，上述提到的不完美循环中存在的紧耦合关系，严重制约了这一解耦过程的实现。

一个直观的解决思路是，将不完美循环转换为完美循环，即把与地址以及旋转因子生成有关的逻辑全部移动到最内层。观察到，虽然不同 stage 下 loop-s 和 loop-j 的循环次数均不恒定，但是每个 stage 中蝶形运算次数都为 $N/2$ 次。基于此，重构了 loop-s 和 loop-j 的功能，将 loop-s 从控制分块步长转换为控制并行分组组数，将 loop-j（即解耦后的 loop-k）转换为控制分组内蝶形运算迭代，从而实现完全展开，避免额外控制。最终将问题转为寻求原始循环与新循环间的地址生成映射关系，如图 7（a）所示。

为了直观地说明新循环下的地址生成逻辑，以 16 点长、p=2 为例，不同 stage 需要依次生成以下地址序列组合：对于 stage0,（0,8,1,9）、（2,10,3,11）、（4,12,5,13）、（6,14,7,15）；对于 stage1,（0,4,1,5）、（2,6,3,7）、（8,12,9,13）、（10,14,11,15）；对于 stage2,（0,2,1,3）、（4,6,5,7）、（8,10,9,11）、（12,14,13,15）。其中每个地址序列组合（如（0,8,1,9））对应 loop-j 的完全展开。上述每个 stage 中的 4 个地址序列组合和图 7（c）中用红、蓝、紫、绿色线连接标记蝶形运算一一对应。

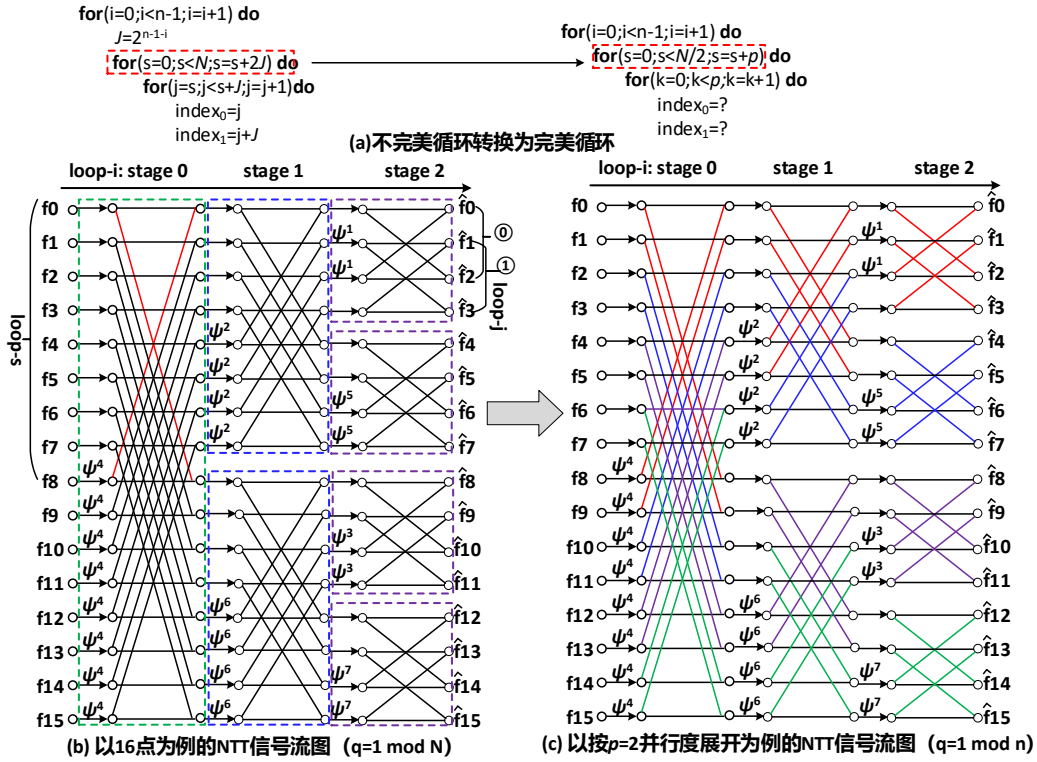

图 7 循环解耦示意图

通过观察，我们得到以下映射过程，可以实现上述系数地址和旋转因子生成逻辑同 s、j 解耦：

$$\begin{cases} h\_\text{part} = (s+k)/2^{n-1-i} \\ l\_\text{part} = (s+k) \bmod 2^{n-1-i} \\ index_0 = h\_\text{part} \times 2^{n-i} + l\_\text{part} \\ index_1 = index_0 + 2^{n-1-i} \\ w = \psi^{BR(h\_part+2^i)} \bmod q \end{cases} \tag{11}$$

上述公式的含义为，先分离出 s+k 的高位 h_part 和低位 l_part，然后重组得到并行分组循环下的系数地址索引，从而实现循环解耦。完整的多通道 NTT/INTT 分别如算法 3 和算法 4 所示。

算法 3 多通道 NTT 算法（mult-lane NTT）

Input: bank 存储下的多项式系数 $\boldsymbol{f}$，并行度 p，mode

Output：bank 存储下的 $\hat{\boldsymbol{f}}$

1:**for**(i=0; i<n-1; i=i+1) **do** ←**loop-i**

2: **for**(s=0; s<N/2; s=s+p) **do** ←**loop-s**

3: **for**(k=0; k<p; k=k+1) **do** ←**loop-k,**可以完全展开

/*Generate address*/

4: $h\_\text{part} = (s+k)/2^{n-1-i}$

5： $l\_\text{part} = (s+k) \bmod 2^{n-1-i}$

6： $index_0 = h\_\text{part} \times 2^{n-i} + l\_\text{part}$

7： $index_1 = index_0 + 2^{n-1-i}$

/* Generate twiddle factor*/

8: $w = \psi^{BR(h\_part+2^i)} \bmod q$

/*Map address*/

9: BI_0, BA_0=conflict_free_map($index_0$, p, N,mode,0)

10: BI_1, BA_1=conflict_free_map($index_1$, p, N,mode,0)

/*Read bank*/

11: u = bank[BI_0][BA_0]

12: v = bank[BI_1][BA_1]

/*Butterfly operation*/

13: $t = w \cdot v$

14: $v = (u-t) \bmod q$

15: $u = (u+t) \bmod q$

/*Write bank*/

16: bank[BI_0][BA_0]=u

17: bank[BI_0][BA_0]=v

18: **endfor**

19: **endfor**

20: **endfor**

由于 NTT 和 INTT 具有对称性，因此，这里以算法 3 为例进行解释。算法 3 依旧由 3 层循环构成，循环体由五个部分组成：第 1 部分为 4-8 行，负责系数地址和旋转因子生成；第 2 部分为 9-10 行，分别调用了算法 2，将系数地址映射到 bank 存储下的地址 BI 和 BA；第 3 部分为 11-12 行，从 bank 中读取对应系数；第 4 部分为 13-15 行，对应蝶形运算；第 5 部分为写 bank。

算法 4 多通道 INTT 算法（mult-lane INTT）

Input: bank 存储下的 $\hat{\boldsymbol{f}}$，并行度 p，以及 mode

Output：bank 存储下的 $\boldsymbol{f}$

1:**for**(i=n-2; i>=0; i=i-1) **do** ←**loop-i**

2: **for**(s=0; s<N/2; s=s+p) **do** ←**loop-s**

3: **for**(k=0; k<p; k=k+1) **do** ←**loop-k,**可以完全展开

/*Generate address*/

4: $h\_\text{part} = (s+k)/2^{n-1-i}$

5： $l\_\text{part} = (s+k) \bmod 2^{n-1-i}$

6： $index_0 = h\_\text{part} \times 2^{n-i} + l\_\text{part}$

7： $index_1 = index_0 + 2^{n-1-i}$

/* Generate twiddle factor*/
8: $w = \psi^{BR(2^{i+1}-1-h\_part)} \bmod q$
/*Map address*/
9: BI_0, BA_0=conflict_free_map($index_0$, p, N,mode,0)
10: BI_1, BA_1=conflict_free_map($index_1$, p, N,mode,0)
/*Read bank*/
11: u = bank[BI_0][BA_0]
12: v = bank[BI_1][BA_1]
13: $t = u$
/*inverse Butterfly operation*/
13: $u = \mathrm{MD}((t+v) \bmod q)$
16: $v = \mathrm{MD}(w \cdot (v-t) \bmod q)$
/*Write bank*/
17: bank[BI_0][BA_0]=u
18: bank[BI_1][BA_1]=v
19: **endfor**
20: **endfor**
21: **endfor**

算法 4 和算法 3 的区别在于，stage 是逆向的，这一点可以从 loop-i 看出。其中，MD(x)的功能为 x/2mod q，其用于消除 IBFU 后的 $n^{-1} \bmod q$ 操作[30]。并且对于奇质数 q，x/2mod q 可等效为式（12）所示过程，无需任何乘法运算。

$$x/2 \bmod q = (x >> 1) + x[0] \cdot \frac{q+1}{2} \tag{12}$$

4.2 多通道并行 PWM

直接计算式（5）需要 5 次模乘运算，Xing 等人[31]通过 karatsuba 算法将其优化为了 4 次，并将其拆分为 PWM0 和 PWM1 两个操作，如下所示：

$$\begin{aligned} &\mathrm{PWM0}: s_0 = \hat{f}_{2i} + \hat{f}_{2i+1} \bmod q, s_1 = \hat{g}_{2i} + \hat{g}_{2i+1} \bmod q, m_0 = \hat{f}_{2i}\hat{g}_{2i} \bmod q, m_1 = \hat{f}_{2i+1}\hat{g}_{2i+1} \bmod q \\ &\mathrm{PWM1}: h_0 = m_0 + m_1\psi^{2BR(i)+1} \bmod q, h_1 = s_0 s_1 - m_0 - m_1 \bmod q \end{aligned} \tag{13}$$

为了实现与 NTT/INTT 硬件资源共享，需要解决以下三个关键问题：

1.计算单元复用：需要将 PWM0 和 PWM1 映射到蝶形运算逻辑。通过分析发现，两个并行的 Radix-2 蝶形单元（等效于 6.2 节中的一个 RBFU）可重构出 PWM0 或者 PWM1 操作，故本文限定通道数 $p \geq 2$。

2.统一数据通路。按照第 3 节设计的数据调度策略已经确保 PWM 与 NTT/INTT 数据流对齐。

3.统一控制逻辑。即保证循环结构大致一致即可。原始 PWM 只有一层循环，因此可以任意进行循环结构的调整，从而实现循环结构统一。

基于上述约束，本文提出的多通道 PWM 如算法 5 所示。算法 5 通过操作码 opcode 选择 PWM0 和 PWM1 操作。

算法 5 多通道 PWM 算法（mult-lane PWM）

Input: bank 存储下的 $\hat{\boldsymbol{f}}$, $\hat{\boldsymbol{g}}$,并行度 p, opcode

Output：bank 存储下的 $\hat{\boldsymbol{h}}$

1:**for**(i=0; i<N/(2p); i=i+1) **do** ←**loop-i**

2: **for**(s=0; s<2; s=s+1) **do** ←**loop-s**

3: **for**(k=0; k<p/2; k=k+1) **do** ←**loop-k,**可以完全展开

/*Generate address*/

4: $index_0$=2k+sp+2ip

5: $index_1$=2k+1+sp+2ip

6: $index_2$= $index_0$

7: $index_3$= $index_1$

/*Map address*/

8: BI_a_0, BA_a_0=conflict_free_map($index_0$, p, N,0,0)

9: BI_a_1, BA_a_1=conflict_free_map($index_1$, p, N,0,0)

10: BI_b_0, BA_b_0=conflict_free_map($index_2$, p, N,1,1)

11: BI_b_1, BA_b_1=conflict_free_map($index_3$, p, N,1,1)

/*Read bank*/

12: a_0= bank[BI_a_0][BA_a_0]

13: a_1= bank[BI_a_1][BA_a_1]

14: b_0= bank[BI_b_0][BA_b_0]

15: b_1= bank[BI_b_1][BA_b_1]

/*PWM0/PWM1*/

16: **if** opcode==PWM1

17: $w=\psi^{2BR(k+sp/2+ip)+1} \bmod q$

18: s_0, s_1, m_0, m_1= PWM1(a_0,a_1,b_0,b_1,w)

19: **else**

20: s_0, s_1, m_0, m_1= PWM0(a_0,a_1,b_0,b_1)

21: **endif**

/*Write bank*/

22: bank[BI_a_0][BA_a_0]= s_0

23: bank[BI_a_1][BA_a_1]= s_1

24: bank[BI_b_0][BA_b_0]= m_0

25: bank[BI_b_1][BA_b_1]= m_1

26: **endfor**

27: **endfor**

28: **endfor**

5 面向多通道 NTT 流水线设计的无冲突检测算法

在多通道 NTT 流水线设计中，避免 RAW 需严格满足数据依赖关系。由于同一 stage 内每个数据地址仅被一次蝶形操作访问，stage 内不会发生冲突。冲突主要出现在不同 stage 之间，其根本原因在于：随着 stage 变化，同一蝶形单元所处理数据点的地址步长是变化的，具体为 $stage_i$ 步长为 2^{n-1-i} 。这就可能导致 $stage_i$ 的运算数据还未完成写入，流水线就已经进行到了 $stage_{i+1}$ 的读取操作，从而访问到旧数据，引发 RAW。若等待 $stage_i$ 阶段所有数据点写入完成后才开始 $stage_{i+1}$ 读取，虽可避免冲突，但会在流水线中引入大量空泡，降低效率。

为了避免 RAW 对流水线性能的影响，本文对多通道 NTT 算法中的 RAW 行为建立数学模型，并给出了能同时保证蝶形运算单元不出现空泡又不发生 RAW 的约束条件。

(1)关键参数定义：

蝶形单元的数量为 p （即第 4 节所述通道数），满足 1≤p≤7。NTT 点长为 $N=2^n$，这里暂时取 CRYSTALS-Kyber 中的 n=8。每个蝶形单元的流水线级数为 L，如图 8 所示。

(2)分析

在 NTT 开始前，数据按照地址划分为 *ie* 和 *io* 两个序列，其中 *ie*=0,1,2…127，*io*=128,129,…255。设 $G_{i,s,k}^{ie}$ 和 $G_{i,s,k}^{io}$ 分别表示第 i 阶段、第 s 组、第 k 个蝶形单元输入的数据点对地址。硬件执行流程如图 8 所示：首先从由多个简单双端口 RAM 构成的多存储体（mult-bank）中并行读取多对数据点；随后各蝶形单元并发执行流水线操作；最后，将结果写回相应的 bank。需要注意的是，RAM 读操作需 1 个时钟周期，因此从首个数据读操作到写操作共需 L+1 个周期。

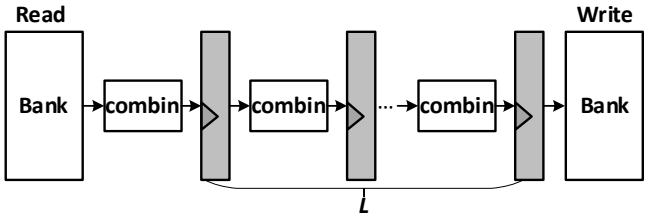

图 8 蝶形单元读写 RAM 示意图

为了保证流水线连续执行不同 stage 的读操作不出现停顿，每个 stage 所需周期数 C 应等于读取所有点数的周期数，即数据点的总组数。每个 stage 包含 128 对数据点，由 p 个蝶形单元并行处理，因此总组数为 $C=\lfloor 128/p \rfloor$。观察发现，在任何阶段，无论是读操作还是写操作，第 *ie* 列第 j 个地址出现时，地址区间[0，j−1]内的所有元素（包含 *ie* 和 *io* 序列）均已完成读取或写入。基于此规律，只需比较 stage_i 写操作地址序列和 stage_{i+1} 读操作序列的重合起始点，即可判别是否发生冲突。

根据算法，stage_{i+1} 读取起始点地址为 $G_{i+1,0,0}^{ie}=0$ 和 $G_{i+1,0,0}^{io}=0+2^{(n-2-i)}$。考虑到流水线延迟为 L+1 个周期，在 stage_{i+1} 开始读取时，stage_i 的写回操作已进行至第 C−(L+1)组。因此冲突判决条件为：

$$\begin{aligned} &G_{i,C-(L+1),0}^{ie} > G_{i+1,0,0}^{ie} \\ &G_{i,C-(L+1),0}^{ie} > G_{i+1,0,0}^{io} \\ &C > L+1 \end{aligned} \tag{14}$$

将上述多通道 NTT 算法中的实际地址带入，可得如下完整判决条件：

(3)判决条件：

将判决条件推广至任意点长。对于所有阶段 $i=0,1,\cdots,n-2$，共 $n-1$ 个 stage，必须满足：

$$\begin{aligned} &((C-L-1)\cdot p//2^{n-1-i})\cdot 2^{n-i}+((C-L-1)\cdot p)\bmod 2^{n-1-i} > 2^{n-1-i} \\ &((C-L-1)\cdot p//2^{n-1-i})\cdot 2^{n-i}+((C-L-1)\cdot p)\bmod 2^{n-1-i} > 0 \\ &C > L+1 \end{aligned} \tag{15}$$

如果该条件成立，则一定不会发生 RAW。算法 6 完整描述了式（15）的检测流程。

算法 6 冲突检测算法

Input: 并行度 p，点长 N

Input: 蝶形单元流水级深度 L

Output:冲突计数 flag

1: $n=\log_2 N$

```
2: flag=0
3: for(i=0;i<n-1;i=i+1) do
4:    temp1=(((N/(2p)-L-1)×p)>>(n-1-i))<<(n-i)
5:    temp2= temp1+((N/(2p)-L-1)×p) & ((1<<(n-1-i))-1)
6:    if temp1>(1<<(n-2-i)) && N/(2p)>L+1
7:          flag+=0
8:    else
9:          flag+=1
10: endfor
11: return flag
```

表 1 展示点长 N=256 情况下，不同并行级和流水深度的冲突检查结果。从表中可以知道，并行度大于等于 32 的情况下是不可能设计出无空泡的多通道 NTT 流水线，只能插入空操作避免。

表 1 *N*=256 下不同并行级和流水深度的冲突检查结果

L*/*p	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1	0	0	0	0	1	7	7
2	0	0	0	0	2	7	7
3	0	0	0	1	7	7	7
4	0	0	0	2	7	7	7
5	0	0	0	3	7	7	7
6	0	0	0	7	7	7	7
7	0	0	1	7	7	7	7
8	0	0	1	7	7	7	7

6 可重构多项式乘法器设计

6.1 整体硬件架构

图 9 为提出的可重构多项式乘法器（RPMA），其由调度器、旋转因子存储体（Twiddle Factor Bank）、bank 地址路由（BA Router）、可重构蝶形单元阵列（Reconfigurable Butterfly unit Array）以及 bank 阵列 5 个部分组成。

调度器主要由 5 个部分组成，分别是配置单元、控制器、地址生成单元，地址映射单元以及仲裁器。其中配置单元用于配置并行度 p 和流水深度 L，控制器用于控制 bank 阵列的读写使能，以及对前两次循环进行计数。地址生成单元则用于生成相应的系数地址。地址映射单元将地址映射成 BI 和 BA。地址生成单元和地址映射单元的数量均为 p。仲裁器根据所有的 BI 号生成 bank 地址路由模块中的选通信号 sel。

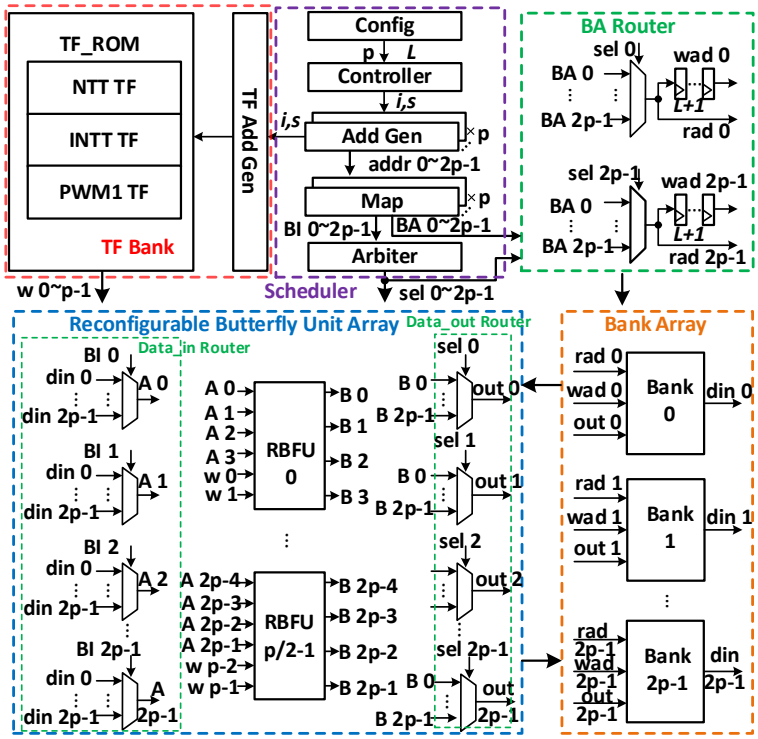

图 9 可重构多项式乘法器架构

旋转因子存储体由 3 个部分组成，分别是 NTT 旋转因子、INTT 旋转因子以及 PWM1 旋转因子。由于旋转因子是预计算的，因此旋转因子存储体为只读型。图 10 给出了 p=2 情况下旋转因子排列情况。NTT 和 INTT 的宽度为 $2p\times12$，PWM1 的宽度为 $p\times12$，因此存储体的宽度统一设置为 $2p\times12$。NTT 和 INTT TF 的深度为 $(n-1)\frac{N}{2p}$，PWM1 TF 的深度 $\frac{N}{p}$，因此 ROM 的深度为 $\frac{nN}{p}$。

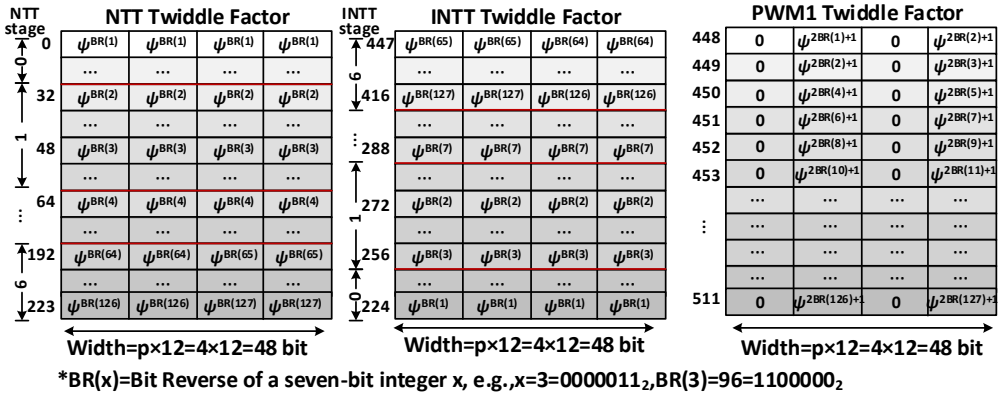

图 10 旋转因子详细排列情况

地址路由模块，由 $2p$ 个多路选择器和以及一些寄存器构成，用于将读写 BA 选通到对应 BI 号的 bank 中，完成运算数据的读写。Bank 阵列由 $2p$ 个读写地址分离的简单双端口 RAM 组成，如果点长为 N，则每个 RAM 存储深度为 $\frac{N}{p}$（存储 2 组多项式系数的深度）。

最后，可重构蝶形阵列作为整个架构的计算核心，其包含 $2p$ 个多路选择器构成的输入数据路由网络，$p/2$ 个可重构蝶形单元（为了便于 PWM 运算，每个蝶形单元可以并行执行两组蝶形操作），以及 $2p$ 个多路选择器构成的输出数据路由网络。其中可重构蝶形单元可以实现 4 种计算模式，分别是 NTT，INTT，

PWM0 和 PWM1。其中，为了将 NTT 和 INTT 映射到可重构蝶形阵列上，只需要把算法 3 和算法 4 的 loop-k 的循环边界设置为 $p/2$，即两个循环体映射到一个可重构蝶形单元。具体配置见 6.2 小节。

6.2 可重构蝶形单元

在基于 NTT 的多项式乘法中，NTT 和 INTT 以及 PWM 操作通常不会并行执行，且许多工作采用了独立硬件分别实现 NTT 和 INTT 的蝶形操作，导致硬件利用率只有 50%。为了提高硬件利用率，本文设计了一种可重构蝶形单元（RBFU），如图 11 所示。每个 RBFU 可同时执行两组蝶形计算，并通过将 PWM 拆分为 PWM0 和 PWM1 两个子操作，以兼容 CRYSTALS-Kyber 中 PWM 的运算特性。RBFU 由 4 个模加器（MA），4 个模减器(MS)，两个模乘器（MM）以及 4 个 1/2 模除器(MD)构成，其中模乘法器采用[12]提出的按位取模约减器 MR 和一个 DSP 实现。MA、MS、MM 和 MD 的详细结构分别如图 11（b）至(e)所示。

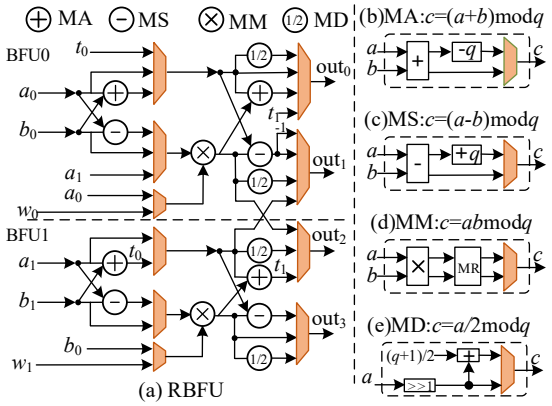

图 11 可重构蝶形单元

图 12 给出了可重构蝶形单元的 4 种配置模式。黑线表示启用该路径，灰线表示弃用该路径。该结构除了支持上述 4 种计算模式外，还能扩展为向量加和向量减法等运算。

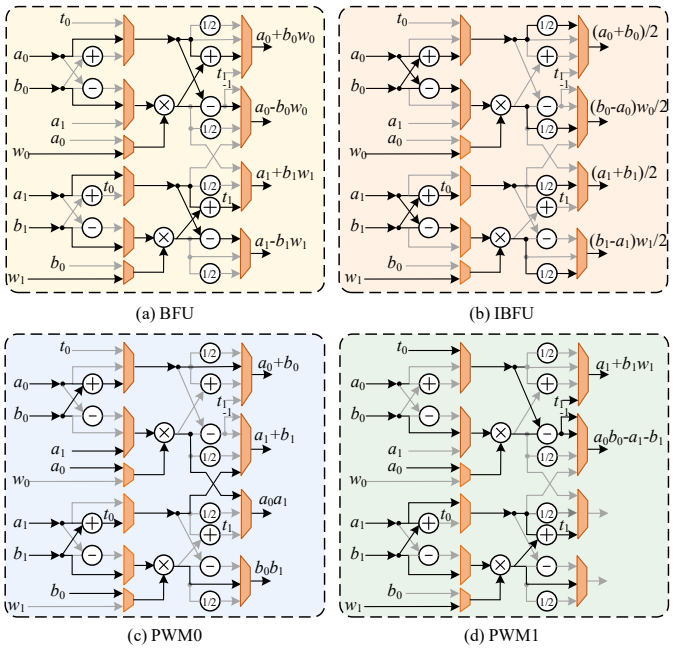

图 12 可重构蝶形单元 4 种模式配置

6.3 算法映射及执行流程

算法 3 至 5 所描述计算过程均为 3 层完美循环结构。在硬件映射层面，其循环体可以系统性地解构为如下几个关键过程：地址生成，地址映射，旋转因子生成，读 bank，BFU/IBFU/PWM0/PWM1 操作，以及写 bank。RPMA 在地址生成、地址映射以及 BFU/IBFU/PWM0/PWM1 操作可以和算法一一对应。然而，硬件实现的核心差异在于对多 bank 并行读写机制的支持，这需要通过仲裁器、地址路由以及数据路由模块协同实现，如图 9 所示。仲裁和路由的原理分别如算法 7 和算法 8 所示。具体而言，算法 8 实现一个可配置的路由功能，当为地址路由时，X=sel，Y=BA，输出 Z 为读地址 rad；当为数据路由输入算法时 X=BI，Y=din，输出 Z 为 A；当为数据路由输出算法时 X=sel，Y=B，输出 Z=out；

算法 7 仲裁算法

Input: 并行度 p，BI 0~$2p$-1
Output: sel 0~$2p$-1
1: **for**(k=0;k<$2p$; k=k+1) **do**
2: **for**(j=0;j<$2p$;j=j+1) **do**
3: **if** BI j==k
4: sel k=j
5: endif
6: **endfor**
7: **return** sel 0~$2p$-1

算法 8 BA router/Data_in Router/Data_out Router

Input: 并行度 p，X 0~$2p$-1, Y 0~$2p$-1
Output: Z 0~$2p$-1
1: **for**(k=0;k<$2p$; k=k+1) **do**
2: **for**(j=0;j<$2p$;j=j+1) **do**
3: **if** X k==j
4: Z k=Y j
5: endif
6: **endfor**
7: **return** Z 0~$2p$-1

为了更清晰地展示 RPMA 的全流水线工作原理，图 13 展示了其抽象化的数据流图。首先由 controller 产生循环变量 i 和 s，发送至地址生成单元生成地址索引，然后由 Map 映射成 BI 和 BA。随后，仲裁器根据 BI_ff_1（BI 打一拍后的信号）生成选通信号 sel。BA Router 在 sel 的控制下将 BA_ff_1 选通到对应的 bank，完成数据读取操作。数据路由算法根据 BI_ff_2 将数据选通到对应的 RBFU 中。数据流经 RBFU 的 L 级流水线处理后，需借助延迟对齐的选通信号 sel_ff_{L+1} 将结果写回 bank。由于在存储体数据输出端引入了一级寄存器以改善时序，加之 RBFU 本身的 L 级流水，存储体的读操作与写操作之间间隔 L+1 个时钟周期。值得注意的是，写回操作采用原位更新策略，即写地址为读地址经过 L+1 个周期延迟后的信号。

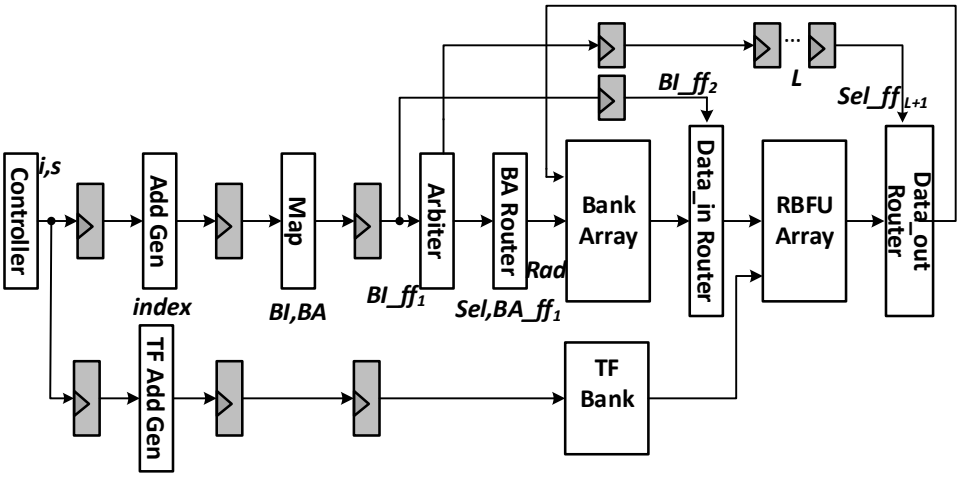

图 13 RPMA 的流水线工作流

7 实现结果与比较

为了评估 RPMA 的性能，我们在 Zynq UltraScale+、Artix-7 以及 Virtex-7 等平台对 RPMA 进行了硬件实现。资源开销和最高频率是在 Vivado2024.1 的默认综合和实现策略下获得的。在 7.1 节中我们将对不同并行度和流水线深度进行性能评估。在 7.2 节中，我们将 RPMA 的实现结果与现有工作进行比较。

7.1 并行度和流水级对性能的影响

在 RPMA 中，较高的并行度 P 能够显著减少循环体迭代次数，而一定深度的流水级 L 则有助于改善时序特性，从而提升系统运行频率。为充分发挥 RPMA 的性能，需对其核心计算单元 RBFU 进行合理的流水线划分。考虑到 RBFU 由两个功能相同的 BFU 构成，其内部结构具备对称性，因此本文仅以 BFU0 为例展示流水线寄存器的具体插入位置，如图 14 所示。流水线划分主要依据各运算模块的逻辑级数进行设置。在 BFU0 中，MA、MS 以及 MD 都是组合逻辑单元，而 MM 的所有关键操作都均被完全展开，详细结构可以参考[12]。表 2 列出了在不同流水线深度下，寄存器插入位置的具体配置组合。

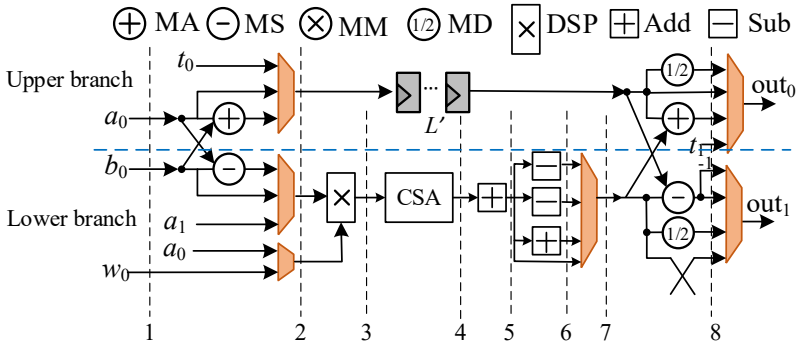

图 14 RBFU 中流水线寄存器插入位置标记

表 2 流水线寄存器插入位置的配置组合

流水级数	Upper branch	Lower branch
L=1	L'=1	4
L=2	L'=2	2,5

L=3	L'=3	2,4,6
L=4	L'=4	2,3,5,7
L=5	8,L'=4	2,3,5,7,8
L=6	1,8,L'=4	1,2,3,5,7,8
L=7	1,8,L'=5	1,2,3,5,6,7,8
L=8	1,8,L'=6	1,2,3,4,5,6,7,8

本研究针对表 1 中所有满足无冲突条件的 L/p 参数组合进行了实验评估，如表 3 所示。在周期数方面，基于图 13 的流水级划分以及算法循环结构能够计算出理论周期数，即 RPMA 中执行单次 NTT 和 INTT 各需要消耗 $(n-1)N/(2p)+L+3$ 个周期，执行 PWM 需要消耗 $2(N/p+L+3)$ 个周期，表 3 中的实验结果和理论完全吻合。在资源使用方面，DSP 和 BRAM 的数量仅和 p 成正比，与 L 无关。为了综合评估不同参数组合下的硬件性能，本文采用面积时间积(Area-Time-Product，ATP)作为评价指标，其计算方式见表 3 注释，ATP 越小，代表硬件性能越高。

表 3 RPMA 在不同 L/p 参数组合下的实现结果

p	***L***	**Freq(MHz)**	**Resource**	**cycles**	**ATP***
			LUT/FF/DSP/BRAM	NTT/INTT/PWM	
2	1	222.22	830/271/2/3	452/452/264	5473.52
	2	322.58	799/313/2/3	453/453/266	3774.45
	3	344.83	1006/428/2/3	454/454/268	3768.47
	4	370.37	890/382/2/3	455/455/270	3409.82
	5	416.67	863/529/2/3	456/456/272	3074.23
	6	588.24	903/578/2/3	457/457/274	2217.5
	7	588.24	948/684/2/3	458/458/276	2274.62
	8	588.24	964/753/2/3	459/459/278	2307.92
4	1	217.39	1985/509/4/4.5	228/228/128	4996.4
	2	303.03	1897/599/4/4.5	229/229/130	3588.05
	3	322.58	2230/850/4/4.5	230/230/132	3603.88
	4	344.83	2088/707/4/4.5	231/231/134	3301.87
	5	357.14	2007/1010/4/4.5	232/232/136	3239.07
	6	526.32	2055/1171/4/4.5	233/233/138	2249.42
	7	555.56	2152/1368/4/4.5	234/234/140	2198.63
	8	555.56	2265/1493/4/4.5	235/235/142	2261.43
8	1	196.08	5072/1012/8/9	116/116/72	6193.02
	2	277.78	4991/1206/8/9	117/117/74	4433.5
	3	294.12	5567/1729/8/9	116/116/76	4406.45
	4	312.5	5490/1423/8/9	118/118/78	4170.3
	5	322.58	5279/2004/8/9	116/116/80	4033.47
	6	416.67	5352/2173/8/9	119/119/82	3232.97
16	1	169.49	16385/2069/16/18	60/60/40	9019.88
	2	238.1	16346/2547/16/18	61/61/42	6615.71

SEC=#BRAM×200+#DSP×100+#LUT/4+#FF/8[14]　Cycle=#NTT+#INTT+#PWM　* ATP=SEC×T=SEC×Cycle/Freq

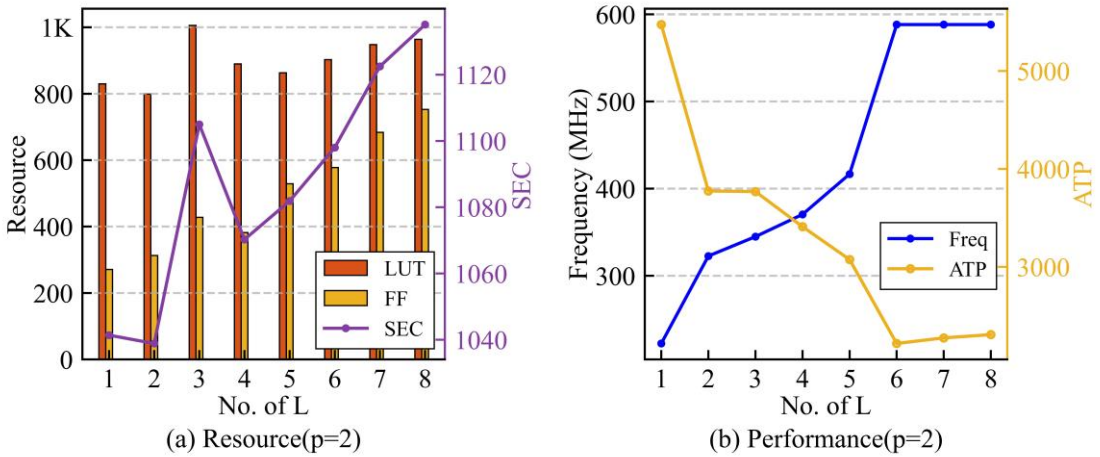

图 15 资源和性能与流水深度 L 的关系（p=2 时）

图 15 展示了 p=2 时，RPMA 的资源消耗与性能随流水深度 L 的变化情况。根据图 15（a），随着 L 的增加，LUT 数量仅在小范围内波动，而 FF 数量和等效面积（SEC）总体呈线性增长趋势。从图 15（b）可以明显看出，在 L 较小时，系统频率与 L 成正比；当 L>6 后，频率增加趋于饱和。尽管随着 L 增大，等效面积在增加，但其增幅远低于频率提升的速度，因此 ATP 总体呈下降趋势，表明在一定范围内增加流水深度可以有效提升 RPMA 的性能。

图 16 展示了在固定流水深度下，RPMA 的资源与性能随 p 的变化关系。在 RPMA 中，RBFU、MAP、Add Gen、Bank Array 的数量都和 p 成正比，而 Controller、TF Add Gen 以及 TF Bank 都是常数和并行度无关，然而随着并行度增加，消耗 LUT 和 FF 却呈现指数级增长，如图 16(a)所示。造成该现象主要原因为，路由网络消耗的资源随 p 呈指数增长趋势，并且在 p=16 时路由网络消耗的资源占 RPMA 整体的 67.95%，具体如图 17 所示。等效面积由于折算的原因，大致和并行度呈线性关系。从理论上讲，随着 p 增加，等效面积线性增长，而周期数反比减少，并不会导致 ATP 变差。但从图 16(b)中可以观察到，随 p 增加，ATP 总体呈增长趋势，即系统性能变差。造成这一趋势的原因在于，路由网络中 MUX 的扇入数呈指数级增加，导致路由部分组合逻辑级数增加，进而导致系统频率恶化。综上，ATP 总体上随 p 增大而上升，换言之，RPMA 的性能随并行度增大而有所降低。

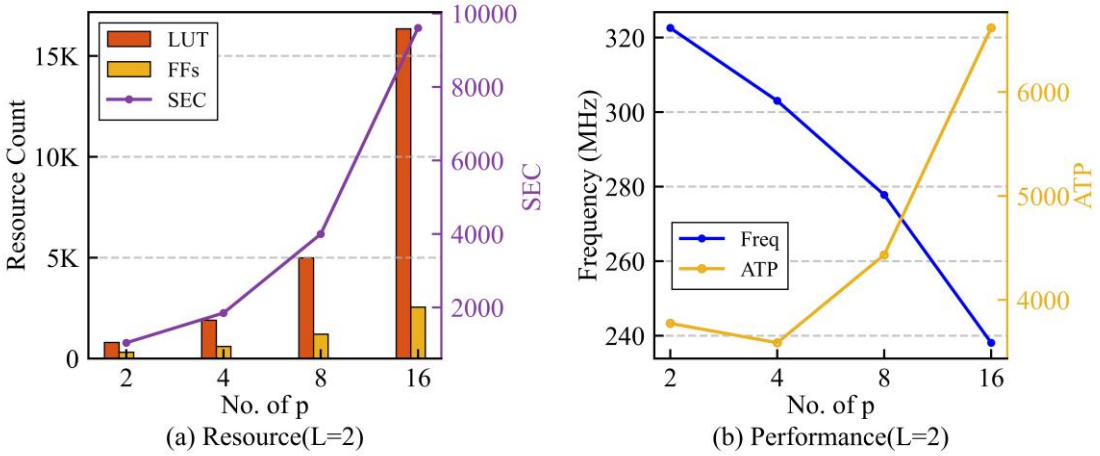

图 16 资源和性能与 p 的关系（L=2 时）

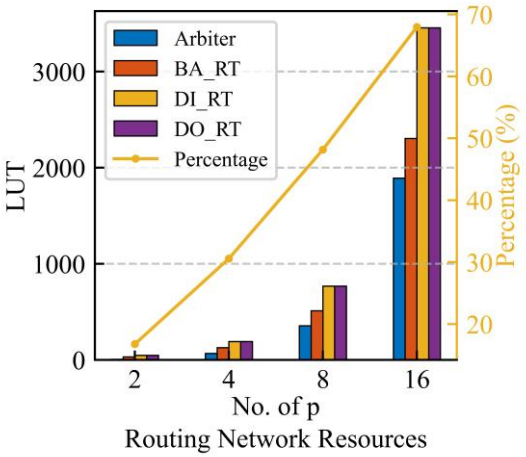

图 17 路由网络资源随 p 的变化情况（L=2 时）

（BA_RT:BA_Router，DI_RT:Data_in Router，DO_RT:Data_out Router）

7.2 与相关工作比较

表 3 针对与 CRYSTALS-Kyber 中多项式乘法器相关的各类研究进行了比较。该表格列出了所使用的 FPGA 平台、不同并行度配置的实现结果以及 ATP。注意，所有对比测试均基于相同的 FPGA 设备。

表 3 与相关工作的比较

Ref	Platform	p	Freq (MH)	Cycles	Resource	Total Cycles	SEC	ATP (us×SEC)
				NTT/INTT/PWM	LUT/FF/DSP/BRAM			
TCAS I'24[7]	Artix-7	2	227	448/448/256	1005/599/2/1.5	1152	826.13	4188.45
TCAS'22[8]	Artix-7	32	540	32/32/128	25674/3137/64/6	192	14410.63	5187.83
DATE'21[12]	Artix-7	1	190	904/904/3359	948/352/1/2.5	5167	881	23963.2
		4	182	232/232/864	2543/792/4/9	1328	2934.75	21394.33
		16	172	69/71/256	9508/2684/16/35	396	11312.5	26018.75
TCAD'23[13]	Virtex-7	1	286	1031/1031/261	449/271/3/3	2323	1046.13	8484.07
		4	278	270/270/69	1288/888/12/4.5	609	2533	5547.27
		8	256	135/135/37	6245/1864/24/12	307	6594.25	7913.1
TCAS I'23[14]	Artix-7	4	200	224/224/64	1740/643/4/-	512	-	-
TC'23[15]	Artix-7	2	229	448/448/256	880/999/2/1.5	1152	844.88	4258.17
TC'24[16]	Artix-7	2	200	313/313/270	784/441/2/3	896	1051.13	4719.55
TCAS I'21[22]	Airtx-7	2	115	474/602/1289	737/290/6/4	2365	1620.5	33317.48
TCAS I'24[27]	Artix-7	1	350	906/906/649	407/411/0/7.5	2461	1653.13	11621.47
		2	342	458/458/328	590/671/0/7.5	1244	1731.38	6302.21
		4	334	234/235/169	1052/1058/0/13.5	638	3095.25	5911.93
TCAD'25[28]	Virtex-7	1	300	1056/-/-	577/723/6/2	-	1234.63	-
TCHES'21[31]	Artix-7	2	161	512/512/1856	1737/1167/2/3	2880	1380.13	24690.44
Ours	Artix-7	2	294	457/457/274	912/562/2/3	1188	1098.25	4425.95
		4	244	233/233/138	2103/1031/4/4.5	604	1954.63	4827.92
		8	204	119/119/82	5447/2044/8/9	320	4217.25	6578.91
	Virtex-7	4	286	233/233/138	2104/1041/4/4.5	604	1956.13	4107.86
		8	244	119/119/82	5432/2021/8/9	320	4210.63	5558.03

我们首先与[8]进行比较，该工作使用高层次语言 Chisel 设计了包含 32 个 BFU 的脉动阵列以实现 NTT 多项式乘法。尽管其运算周期数极少，但资源代价巨大。等效面积高达 14410.63，且 ATP 比我们提供的所有并行度都差。（由于其只提供了 DC 综合频率，ATP 是在假定的 300MHz 下计算的，注意这是 Airtex-7 较高的极限频率）。

与同样使用 2 个 BFU 的[7]和[15]相比，RPMA 的 ATP 略微偏高。具体地，[7]和[15]在周期数、LUT、FF 以及 DSP 资源消耗方面与本文相近，频率则低与本文。然而，[7]和[15]采用了多路转接结构设计重排序单元，降低了存储带宽需求，所用 BRAM 仅为 RPMA 的一半。因此，RPMA 的等效面积分别是[7]和[15]的 1.33×和 1.30×，并且 ATP 也分别 1.06×和 1.04×。但 RPMA 支持可配置并行度和流水级，提供了更佳的灵活性和可扩展性。

[12]采用混合洗牌内存调度方案实现了 NTT/INTT/PWM 操作。所提出的结构采用统一蝶形单元用于 NTT/INTT/PWM 操作。但其采用乒乓内存，调度过程中天然存在空泡。在 p=4 时，其 BRAM 消耗是 RPMA 的两倍。由于其蝶形结构中没有专门设计 PWM 通路，而是通过分解操作来实现，导致 p=4 时 PWM 周期数为 RPMA 的 6.26×，ATP 也比我们差了 4.43×。此外，在其他并行度下，RPMA 的 ATP 都明显优于[12]。论文[13]提出一种可配置 NTT 加速器的生成方法，并设计了无流水线停顿的无冲突访存方式，适应不同 PE 阵列配置参数。但其采用了蒙哥马利约减算法实现模乘单元，DSP 消耗较高。在 p 为 4 和 8 时，消耗的 DSP 均为 RPMA 的 3 倍。因此，在 BFU 为 4 和 8 时，RPMA 分别实现 25.9%和 29.7%的 ATP 改善。

[14]首次提出基于混合基数-2/4 的 NTT/INTT 算法，无需预处理和后处理，提升了执行效率。并通过“lazy-last-layer”技巧提高了平均内存带宽利用率。在 p=4 时，其执行一次多项式乘法的周期数比 RPMA 少 15.6%。LUT 和 FF 均低于 RPMA。由于其并未提供 BRAM 数，无法比较 ATP。尽管其具备较低面积和周期数，但缺乏可扩展性。[16]提出了一种基于分裂基 NTT 算法的内存组织和无冲突内存映射方案，使得 split-radix NTT/INTT/PWM 可以无冲突高效执行。其分裂蝶形结构能有效减少内存访问次数，周期数比 RPMA 少 24.6%。此外，其互联结构更少，降低了面积开销。然而，RPMA 的频率比[16]高了 47%，RPMA 的 ATP 改善了 6.2%。

[22]同样设计了可重构蝶形单元，采用两个并行 BFU 加速多项式乘法。其采用 Barrett 算法实现模约减，每个 BFU 消耗 3 个 DSP(两个用于约减，一个用于整数乘法)，两个 BFU 共消耗 6 个 DSP，而本文 RBFU 采用优化的 bitwise 约减，仅消耗 2 个 DSP，显著降低了资源开销，等效面积减少了 32.2%。此外，本文通过细致优化各模块逻辑深度，实现了较高频率。与 RPMA 相比，[22]频率表现差了 2.55×，ATP 差了 7.53×。[28]提出了通用的无冲突内存访问算法，但不支持 PWM，且和[13]类似采用蒙哥马利约减算法，单个 BFU 的 DSP 消耗为 RBFU 的 3×，等效面积是 RPMA 的 1.15×，NTT 周期数为本文的 2 倍以上。[31]提出了统一的蝶形单元，支持 NTT 相关操作，包括特殊点乘，并优化了 Barrett 约减算法以加速了模乘。尽管其同样采用了 karatsuba 算法优化 PWM，但调度流程复杂，PWM 周期数为 RPMA 的 6.77×，等效面积和 ATP 分别比 RPMA 差了 1.26×和 5.58×。

[27]提出了一种基于查找表的 RNS 方法执行模乘运算，将操作分解为子模运算并使用查找表来加速计算，避免了 DSP 消耗，LUT 消耗也低于 RPMA。但 BRAM 消耗远高于 RPMA。在 p=2 和 4 时，与 RPMA

相比，[27]的 ATP 分别差了 1.42×和 1.22×。

8 总结

本文提出了一种可重构且面积高效的多通道多项式乘法器架构。提出了一种循环解耦的多通道 NTT/INTT/PWM 算法，支持任意并行度展开，并实现了统一的控制逻辑。借助所提出冲突检测算法，显著提升了硬件资源利用率。此外，还提出了一种基于交织存储策略的无冲突内存映射方案，该方案与多项式长度和计算阶段无关，统一了 NTT/INTT/PWM 的访问模式。FPGA 的实现结果表明，与之前的研究相比，所提出的 RPMA 在兼顾灵活性同时实现了较优的面积效率，为后量子密码硬件加速提供了一种高效、可扩展的解决方案。

参考文献

[1] SHOR P W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer[J/OL]. SIAM J. Comput., 1997, 26(5): 1484-1509. DOI:10.1137/S0097539795293172.

[2] CHEN L, JORDAN S, LIU Y K, 等. Report on Post-Quantum Cryptography: NIST IR 8105[R/OL]. National Institute of Standards and Technology, 2016: NIST IR 8105[2025-04-09]. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf. DOI:10.6028/NIST.IR.8105.

[3] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (US). Module-lattice-based key-encapsulation mechanism standard: NIST FIPS 203[R/OL]. Washington, D.C.: National Institute of Standards and Technology (U.S.), 2024: NIST FIPS 203[2025-03-05]. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf. DOI:10.6028/NIST.FIPS.203.

[4] YE Z, CHEUNG R C C, HUANG K. PipeNTT: A Pipelined Number Theoretic Transform Architecture[J/OL]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2022, 69(10): 4068-4072. DOI:10.1109/TCSII.2022.3184703.

[5] BISHEH-NIASAR M, AZARDERAKHSH R, MOZAFFARI-KERMANI M. A Monolithic Hardware Implementation of Kyber: Comparing Apples to Apples in PQC Candidates[J]. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2021: 108-126.

[6] NGUYEN T H, KIEU-DO-NGUYEN B, PHAM C K, 等. High-Speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium[J/OL]. IEEE Access, 2024, 12: 34918-34930. DOI:10.1109/ACCESS.2024.3371581.

[7] NGUYEN T H, DAM D T, DUONG P P, 等. Efficient Hardware Implementation of the Lightweight CRYSTALS-Kyber[J/OL]. 2024: 1-13. DOI:10.1109/TCSI.2024.3443238.

[8] ZHAO Y, XIE R, XIN G, 等. A High-Performance Domain-Specific Processor With Matrix Extension of RISC-V for Module-LWE Applications[J]. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, 2022, 69(7).

[9] ZHANG J, LU J, LI A, 等. Super-K: A Superscalar CRYSTALS-KYBER Processor Based on Efficient Arithmetic Array[J]. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, 2024, 71(9).

[10] WANG J, YANG C, MENG Y, 等. A Reconfigurable and Area-Efficient Polynomial Multiplier Using a Novel In-Place Constant-Geometry NTT/INTT and Conflict-Free Memory Mapping Scheme[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2025, 72(3): 1358-1371. DOI:10.1109/TCSI.2024.3483229.

[11] CHEN X, YANG B, YIN S, 等. CFNTT: Scalable Radix-2/4 NTT Multiplication Architecture with an Efficient Conflict-free Memory Mapping Scheme[J/OL]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021: 94-126. DOI:10.46586/tches.v2022.i1.94-126.

[12] YAMAN F, MERT A C, OZTURK E, 等. A Hardware Accelerator for Polynomial Multiplication Operation of

CRYSTALS-KYBER PQC Scheme[C/OL]//2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). Grenoble, France: IEEE, 2021: 1020-1025[2024-10-17]. https://ieeexplore.ieee.org/document/9474139/. DOI:10.23919/DATE51398.2021.9474139.

[13] MU J, REN Y, WANG W, 等. Scalable and Conflict-Free NTT Hardware Accelerator Design: Methodology, Proof, and Implementation[J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(5): 1504-1517. DOI:10.1109/TCAD.2022.3205552.

[14] GUO W, LI S. Highly-Efficient Hardware Architecture for CRYSTALS-Kyber With a Novel Conflict-Free Memory Access Pattern[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(11): 4505-4515. DOI:10.1109/TCSI.2023.3306347.

[15] DANG V B, MOHAJERANI K, GAJ K. High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber[J/OL]. IEEE Transactions on Computers, 2023, 72(2): 306-320. DOI:10.1109/TC.2022.3222954.

[16] GUO W, LI S. Split-Radix Based Compact Hardware Architecture for CRYSTALS-Kyber[J]. IEEE TRANSACTIONS ON COMPUTERS, 2024, 73(1).

[17] CHEN D D, MENTENS N, VERCAUTEREN F, 等. High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2015, 62(1): 157-166. DOI:10.1109/TCSI.2014.2350431.

[18] MERT A C, KARABULUT E, OZTURK E, 等. A Flexible and Scalable NTT Hardware : Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography[C/OL]//2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). Grenoble, France: IEEE, 2020: 346-351[2024-01-26]. https://ieeexplore.ieee.org/document/9116470/. DOI:10.23919/DATE48585.2020.9116470.

[19] LI M, TIAN J, HU X, 等. Reconfigurable and High-Efficiency Polynomial Multiplication Accelerator for CRYSTALS-Kyber[J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(8): 2540-2551. DOI:10.1109/TCAD.2022.3230359.

[20] HU X, TIAN J, LI M, 等. AC-PM: An Area-Efficient and Configurable Polynomial Multiplier for Lattice Based Cryptography[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2023, 70(2): 719-732. DOI:10.1109/TCSI.2022.3218192.

[21] XIN G, HAN J, YIN T, 等. VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(8): 2672-2684. DOI:10.1109/TCSI.2020.2983185.

[22] BISHEH-NIASAR M, AZARDERAKHSH R, MOZAFFARI-KERMANI M. Instruction-Set Accelerated Implementation of CRYSTALS-Kyber[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2021, 68(11): 4648-4659. DOI:10.1109/TCSI.2021.3106639.

[23] XING Y, LI S. An Efficient Implementation of the NewHope Key Exchange on FPGAs[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(3): 866-878. DOI:10.1109/TCSI.2019.2956651.

[24] PÖPPELMANN T, GÜNEYSU T. Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware[M/OL]//HEVIA A, NEVEN G. Progress in Cryptology – LATINCRYPT 2012: 卷 7533. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 139-158[2024-03-02]. http://link.springer.com/10.1007/978-3-642-33481-8_8. DOI:10.1007/978-3-642-33481-8_8.

[25] LYUBASHEVSKY V, SEILER G. NTTRU: Truly Fast NTRU Using NTT[J/OL]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019: 180-201. DOI:10.46586/tches.v2019.i3.180-201.

[26] HENNESSY J L. Computer Architecture: A Quantitative Approach[J].

[27] ALHASSANI A, BENAISSA M. High-Speed Polynomials Multiplication HW Accelerator for

CRYSTALS-Kyber[J/OL]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2024: 1-9. DOI:10.1109/TCSI.2024.3427011.

[28] KRIEGER F, HIRNER F, MERT A C, 等. A Flexible Hardware Design Tool for Fast Fourier and Number-Theoretic Transformation Architectures[J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2025: 1-1. DOI:10.1109/TCAD.2025.3595834.

[29] TAKALA J H, JARVINEN T S, SOROKIN H T. Conflict-free parallel memory access scheme for FFT processors[C/OL]//Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.: 卷 4. Bangkok, Thailand: IEEE, 2003: IV-524-IV-527[2025-06-09]. http://ieeexplore.ieee.org/document/1205957/. DOI:10.1109/ISCAS.2003.1205957.

[30] ZHANG N, YANG B, CHEN C, 等. Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT[J/OL]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020: 49-72. DOI:10.46586/tches.v2020.i2.49-72.

[31] XING Y, LI S. A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA[J/OL]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021: 328-356. DOI:10.46586/tches.v2021.i2.328-356.