基于2KNTT的多项式乘法单元设计

陈 韬,李慧琴*,吴艾青,李 伟,南龙梅
(中国人民解放军战略支援部队信息工程大学,河南郑州 450000)

摘 要: 在格基抗量子公钥密码算法的基础运算中,多项式乘法在硬件实现上消耗大量的时间. 为提高实际运算性能,本文通过分析多项式乘法运算中数论变换的快速实现算法,提出一种面向CRYSTALS-Kyber算法、适应硬件实现的2n次单位根预处理型快速数论变换算法架构,利用小位宽数论变换的并行处理与复杂度低的计算形式来减少运算时间. 整体运算架构在结合算法特殊性质后,确定了32路并行的设计模型. 在此基础上,设计了一种与该架构匹配的统一化运算单元和数据读写不冲突、地址分配最优的存储单元. 实验结果表明,在65 nm的互补金属氧化物半导体(CMOS)工艺下,97 ns完成一组项数为256、模数为3 329的多项式乘法运算,花费108个周期,最高工作频率可达到1.1 GHz,面积时间积为20.7 (kGE · μs).

关键词: 格基抗量子公钥密码算法;CRYSTALS-Kyber;多项式乘法;2KNTT;硬件实现
基金项目: 国家自然科学基金(No.61404175);国家科技重大专项(No.2018ZX01027101-00)
中图分类号: TN402;TP309 **文献标识码:** A **文章编号:** 0372-2112(2024)02-0455-13
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20220629

A Polynomial Multiplier Design Based on 2KNTT

CHEN Tao, LI Hui-qin*, WU Ai-qing, LI Wei, NAN Long-mei
(*Information Engineering University*, *Zhengzhou*, *Henan* 450000, *China*)

Abstract: Polynomial multiplication consumes a lot of time in hardware implementation in the underlying operations of Lattice-based post-quantum public-key cryptography algorithms. The paper analyzes the fast implementation of number theoretic transform algorithm in polynomial multiplication operations for CRYSTALS-Kyber and proposes a 2n-th unit root preprocessing fast number theoretic transform algorithm architecture that adapts to the hardware implementation. In order to reduce computing time, the architecture uses parallel processing of small bit-width number theoretic transformation and low-complexity computations. Taking into account the characteristics of the algorithm, the overall computing architecture adopts a 32-way parallel design model. Based on this, we design a unified computing unit that matches the architecture and a storage unit with non-conflicting mechanism while reading or writing data and optimal address assignment. Under the CMOS 65 nm process, a set of polynomial multiplication operations with term number 256 and modulus 3 329 can be completed in 108 cycles within 97 ns. The maximum operating frequency can reach 1.1 GHz, and the area time product is 20.7 (kGE · μs).

Key words: Lattice-based post-quantum public-key cryptography; CRYSTALS-Kyber; polynomial multiplication; 2KNTT; hardware design
Foundation Item(s): National Natural Science Foundation of China (No.61404175); National Major Science and Technology Special Project (No.2018ZX01027101-00)

1 引言

网络与信息安全已成为事关国家与军队安全的重大战略问题. 在此背景下,量子计算机的研发和抗量子计算的密码算法[1]设计在网络信息安全领域已成为新兴的研究方向. 2012年,美国国家标准技术研究所(National Institute of Standards and Technology,NIST)开始启动抗量子密码方向的研究. 2017年,谷歌、英特尔和IBM科技公司都已宣布在量子硬件的研发上取得了显著进展. 截至2020年10月27日,NIST公开了三轮入围候选者和候选者提交的调整方法. 其中,基于格加密的

收稿日期:2022-05-31;修回日期:2022-09-13;责任编辑:孙瑶
*通讯作者:李慧琴

算法由于其安全性可证明[2]、公私钥尺寸更小与计算速度更快的优点被认为是最有前途的用于公钥加密和数字签名的通用抗量子算法. 在基于格的抗量子公钥密码算法中,带错误学习(Learning With Errors,LWE)、基于环错误学习(Ring Learning With Errors,RLWE)及基于模错误学习(Module Learning with Error,MLWE)等衍生问题的格基密码学构造发展迅速,被认为是最有希望被标准化的技术路线. 在加密过程中,核心运算为 $u=\boldsymbol{A}^{\mathrm{T}}r+e$. 其中,$\boldsymbol{A}^{\mathrm{T}}r$ 为多项式乘法,硬件设计为格基抗量子算法实现的核心与基础. 传统算法 Schoolbook 需要 n^2 次乘法与 $(n-1)^2$ 次加减法,计算复杂度为 $O(n^2)$. 而快速数论变换(Number Theoretic Transform,NTT)算法的计算复杂度仅为 $O(n\log_2 n)$. 因此,本文将采用 NTT 算法实现多项式乘法并对其设计进行优化与改进.

正如关于格基抗量子密码算法实现的调查[3]中所论述的,此类算法已在数学理论上得到了安全保证,在具体实现时,需从高性能到资源受限等各个层面考虑各种计算平台的多样性. 其中,关于多项式乘法的研究重点在于其具体实现所选用的算法和改进手段,运算阶段中关于模乘器的设计,以及与存储单元之间数据的调度问题. 接下来,就此三方面展开论述.

在算法层面,Lyubashevsky 等人[4]提出将基本型快速数论变换优化为 $2n$ 次单位根型快速数论变换,可以避免传统 NTT 算法中的零填充,使具体运算的长度从 $2n$ 减小为 n,减少了模 (2^n+1) 的运算次数,减少了操作数量以及运算时间. 2015 年,Chen 等人[5]针对 RLWE 加密方案,首次根据文献[4]提出的负包装卷积的方法设计了一种高性能实现多项式乘法的硬件设计方案. 2019 年,中国科学院的王鲲鹏团队提出预处理型 NTT(Preprocess-then-NTT,PtNTT)方法[6],基于 RLWE 的方案由于选择超大模数而导致带宽过大,此方法可削弱对模数 q 的限制,即只需要 $q\equiv 1 \bmod n$ 或 $q\equiv 1 \bmod n/2$,甚至更小. 因此,若要进一步提高性能,需从算法层面进行适应硬件设计的调整,减少由数据冲突带来的多余时间消耗.

运算单元的优化研究也较为广泛,清华大学的白国强等人于 2016 年提出的高速运算方案[7],通过使用两个运算单元进行第 i 级的计算,另外两个运算单元利用流水线执行第 $(i+1)$ 级的计算,最终获得 2.25 倍速度的提高. 除此以外,还有采用统一化设计[8~10]的思想,依据 NTT 运算、INTT 运算以及点乘运算本身的特点,保证最少的运算单元面积,实现更快的多项式乘法. 提高性能最常用的方法还有并行实现. 文献[11]在矢量处理器中实现了多项式乘法,此文献中的设计与软件里的 AVX 矢量指令更加契合,具体由 32 个矢量 NTT 单元组成,为快速实现多项式乘法提供了新思路. 上述研究分别从流水线优化、以面积换时间以及并行实现角度,从运算单元整体进行优化,但是会出现占用面积过大以及控制结构复杂的问题. 具体来看,影响运算单元性能的关键要素在于取模算法的选择. 清华大学的刘雷波团队设计了针对特殊模数 q=12 289 的取模单元[12],比采用增加额外乘法操作的 Barrett 与 Montgomery 算法更加高效,但是只适合单一模数.

此外,大量研究人员通过存储优化节省了时间和空间. 有按需计算通过 on-the-fly 的方式生成旋转因子及其幂值或直接存储的方案:前者可以减少存储单元的使用和访问频率,并且简化地址生成逻辑单元,从而减少了硬件上的资源占用;后者从最大限度节省时间的角度来优化. 文献[13]中的设计存储单元由四端口随机存取存储器(Random Access Memory,RAM)组成. 文献[8]则实现了一种由单端口 RAM 组成的存储 NTT 系数的内存架构,实现 NTT 架构中计算电路的最大利用率,整体上减少读写存储单元的时间. 因此,为尽可能减少 RAM 数量,保证花费时间最少,本身 RAM 设计及其地址逻辑优化方案亟待提出.

针对上述分析,本文采用 $2n$ 次单位根预处理型快速数论变换算法,提出一种快速实现多项式乘法的单元架构. 通过将运算单元统一转换为类蝶形运算结构,保证一路计算只对应一个可重构运算单元,从而减小整体面积和降低运算复杂度;采用多 Bank 组成的多端口 RAM 存储单元,保证了读写时间的精简化以及构建了一种降低控制复杂度的结构. 同时,通过流水线设计和模乘算法的实现减小了关键路径;并结合 32 路并行的整体结构,减少一次多项式乘法的运算时间以及存储单元带来的冗余面积消耗.

2 基础理论

多项式乘法可通过 Cooley 和 Turkey 提出的快速傅里叶变换(Fast Fourier Transform,FFT)将计算复杂度从 $O(n^2)$ 简化到 $O(n\log_2 n)$. 在格基抗量子密码算法中定义的有限域或环的要求下,多项式乘法具体采用 NTT 的方法来实现,不再是浮点数范围上的运算. 与 FFT 相比,NTT 是将单位复根转换为单位整数根,将复数对应于整数上.

对于两个 n 项多项式,其 n 项系数分别表示为 $\boldsymbol{x}=(x_0,x_1,\cdots,x_{n-1})$,$\boldsymbol{y}=(y_0,y_1,\cdots,y_{n-1})$. 通过快速数论变换将系数表示的多项式转换为点值 $X(k)=\sum_{t=0}^{n-1}x(t)\omega^{tk} \pmod q$,将点值相乘,获得乘积多项式所对应

的点值,再通过快速数论逆变换获得乘积多项式的系数 $x(t)=n^{-1}\sum_{t=0}^{n-1}X(k)\omega^{-tk}\pmod q$. $x(t)$ 表示多项式中的第 t 项系数,$X(k)$ 表示当 x 为 ω^k 时的点值. 其中,n 为 2 的幂次值,ω 为在 Z_q 范围下的单位根,要求 $\omega^n\equiv 1\bmod q$. 根据时间抽取算法,将原序列 $\boldsymbol{x}$ 按照序号的奇偶性拆分成两个序列 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$,则对应的 NTT 表示为

$$\begin{aligned}X(k)&=\sum_{t=0}^{n-1}x(t)\omega^{tk}\\&=\sum_{t=0}^{n-1}x(t)\omega^{tk}\ (t\text{为偶数})+\sum_{t=0}^{n-1}x(t)\omega^{tk}\ (t\text{为奇数})\\&=X_1(k)+\omega^{tk}X_2(k)\end{aligned}\tag{1}$$

$$\begin{aligned}X\left(k+\frac{n}{2}\right)&=\sum_{t=0}^{n-1}x(t)\omega^{tk}\\&=\sum_{t=0}^{n-1}x(t)\omega^{tk}\ (t\text{为偶数})+\sum_{t=0}^{n-1}x(t)\omega^{tk}\ (t\text{为奇数})\\&=X_1(k)-\omega^{tk}X_2(k)\end{aligned}\tag{2}$$

其中,$X(k)$ 和 $X(k+n/2)$ 分别是原多项式中前后 $n/2$ 项系数对应的点值,k 的取值范围是 $0,1,\cdots,n/2-1$. 因此,只要进一步求出偶数项和奇数项系数对应的 $n/2$ 个点值,即 $X_1(k)$ 和 $X_2(k)$,再经过形如 $x_1=a+b\omega$,$x_2=a-b\omega$ 的蝶形运算,采用递归的思想继续划分,直到将 n 减半到 2,即可求出全部 $X(k)$ 的值. 得到两个多项式所有的点值后,进行对应点值的相乘,转换为用系数表示的多项式. 不同格基抗量子密码算法中多项式乘法要求的模多项式不同,具体使用的方法也是不同的. 两个多项式在 $Z_q[x]$ 范围内,计算多项式乘法的公式为

$$\mathrm{INTT}_{2n}\left(\mathrm{NTT}_{2n}\left(\mathrm{zero-pad}(\boldsymbol{a})\right)\odot\mathrm{NTT}_{2n}\left(\mathrm{zero-pad}(\boldsymbol{b})\right)\right)$$

其中,$\odot$ 表示点值对应相乘;zero-pad 表示通过在多项式后 n 项补零将 n 项多项式转换为 $2n$ 项多项式,再进行数论变换的多项式乘法操作. 当多项式所在环的要求为 $Z_q[x]/\langle x^n-1\rangle$ 时,就可以直接用上述数论变换方法进行 n 项多项式乘法的计算:$\mathrm{INTT}_n\left(\mathrm{NTT}_n(\boldsymbol{a})\odot\mathrm{NTT}_n(\boldsymbol{b})\right)$. 当多项式所在环的要求为 $Z_q[x]/\langle x^n+1\rangle$ 时,需采用负包装卷积方法. 此方法除了要求 $\omega^n\equiv 1\bmod q$,还要求 $g^{2n}\equiv 1\bmod q$,$q\equiv 1\bmod 2n$,$g^n\equiv -1\bmod q$,即 g 为 $2n$ 次单位根,$g=\sqrt{\omega}$. 系数转换为点值的表达式为 $\hat{a}_i=\left(\sum_{j=0}^{n-1}a_j\gamma_{2n}^j\omega_n^{ij}\right)\bmod q$,其中 $i=0,1,\cdots,n-1$,$a_j\gamma_{2n}^j$ 对应 $X(k)=\sum_{t=0}^{n-1}x(t)\omega^{tk}\pmod q$ 中的 $x(t)$,$\hat{a}_i$ 就是当 x 为 ω_n^i 时的点值. 最终可化简为

$$\begin{aligned}\hat{a}_i&=\hat{a}_i^{(0)}+\omega_n^i\gamma_{2n}\hat{a}_i^{(1)}\bmod q\\&=\hat{a}_i^{(0)}+\gamma_{2n}^{2i+1}\hat{a}_i^{(1)}\bmod q\\\hat{a}_{i+n/2}&=\hat{a}_i^{(0)}-\omega_n^i\gamma_{2n}\hat{a}_i^{(1)}\bmod q\\&=\hat{a}_i^{(0)}-\gamma_{2n}^{2i+1}\hat{a}_i^{(0)}\bmod q\end{aligned}\tag{3}$$

$$i=0,1,\cdots,\frac{n}{2}-1$$

其中,

$$\hat{a}_i^{(0)}=\left(\sum_{j=0}^{n/2-1}a_{2j}\gamma_n^j\omega_{n/2}^{ij}\right)\bmod q,\ \hat{a}_i^{(1)}=\left(\sum_{j=0}^{n/2-1}a_{2j+1}\gamma_n^j\omega_{n/2}^{ij}\right)\bmod q.$$

最终不断迭代,通过原始多项式的系数以及单位根之间的蝶形运算得到对应点值,如算法 1 所示.

算法 1　NTT

$f=c_0+c_1x+\cdots+c_{n-1}x^{n-1}\in Z_q[x]/\langle x^n+1\rangle$

预计算: $\gamma_k=\gamma^{\mathrm{brv}(k)},0\leqslant k<n$

输入: $c_0,c_1,\cdots,c_{n-1}$

输出: $c_0,c_1,\cdots,c_{n-1}$

```
1.   k ← 1
2.   FOR l ← n/2 ; l > 0; l ← l/2 DO
3.     FOR s ← 0; s < n; s ← j + l DO
4.       FOR j ← s; j < s + l; j ← j + 1 DO
5.         t ← γ_k · c_{j+l}
6.         c_j ← c_j + t
7.         c_{j+l} ← c_j − t
8.       END FOR
9.     k ← k + 1
10.  END FOR
11.  END FOR
```

$a_i=n^{-1}\gamma_{2n}^{-i}\left(\sum_{j=0}^{n-1}\hat{a}_j\omega_n^{-ij}\right)\bmod q$ 为点值转换为系数的表达式,其中 $i=0,1,\cdots,n-1$,$\gamma_{2n}^{-i}\hat{a}_j$ 对应上述表达式 $x(t)=n^{-1}\left(\sum_{t=0}^{n-1}X(k)\omega^{-tk}\right)\bmod q$ 中的 $X(k)$. 最终可化简为

$$\begin{aligned}a_{2i}&=\left(\frac{n}{2}\right)^{-1}\gamma_n^{-i}\left(\sum_{j=0}^{\frac{n}{2}-1}\hat{b}_j^{(0)}\omega_{n/2}^{-ij}\right)\bmod q\\a_{2i+1}&=\left(\frac{n}{2}\right)^{-1}\gamma_n^{-i}\left(\sum_{j=0}^{\frac{n}{2}-1}\hat{b}_j^{(1)}\omega_{n/2}^{-ij}\right)\bmod q\end{aligned}\tag{4}$$

$$i=0,1,\cdots,\frac{n}{2}-1$$

其中,

$$\hat{b}_j^{(0)}=\left(\hat{a}_j+\hat{a}_{j+n/2}\right)/\left(2\bmod q\right),$$

$$\hat{b}_j^{(1)}=\left(\hat{a}_j-\hat{a}_{j+n/2}\right)/\left(2\cdot\omega_n^{-j}\gamma_{2n}^{-1}\bmod q\right).$$

最终不断迭代,通过将 NTT 转换后的点值进行蝶

形运算可得到最终乘积多项式的系数,如算法2所示.

算法2 INTT

$f=c_0+c_1x+\cdots+c_{n-1}x^{n-1}\in Z_q[x]/\langle x^n+1\rangle$

预计算: $\gamma_k=\gamma^{-(\mathrm{brv}(k)+1)},0\leqslant k<n$

输入: $c_0,c_1,\cdots,c_{n-1}$

输出: $c_0,c_1,\cdots,c_{n-1}$

1. $k\leftarrow 0$
2. FOR $l\leftarrow 1;l\leftarrow n;l\leftarrow 2l$ DO
3. FOR $j\leftarrow s;j<s+l;j\leftarrow j+1$ DO
4. FOR $j\leftarrow s;j<s+l;j\leftarrow j+1$ DO
5. $t\leftarrow c_j$
6. $c_j\leftarrow (t+c_{j+l})/2$
7. $c_{j+l}\leftarrow (t-c_{j+l})/2$
8. $c_{j+l}\leftarrow \gamma_k\cdot c_{j+l}$
9. END FOR
10. $k\leftarrow k+1$
11. END FOR
12. END FOR

3 改进型快速数论变换算法

上节方法为基于RLWE加密方案中的大模数多项式乘法提供了高性能的实现途径.但是其应用的前提为模数q应满足$q\equiv 1\bmod 2n$,且大多数加密方案的选择为超大模数,会导致带宽过大从而带来资源上的损耗.文献[9]提出的PtNTT方法削弱了对模数q的限制,即只需要$q\equiv 1\bmod n$或$q\equiv 1\bmod n/2$,甚至更小.在最新一轮NIST候选抗量子密码算法中,基于MLWE问题的Kyber算法中要求$q\equiv 1\bmod n$.本文为进一步对其中使用的PtNTT方法进行具体的实施,需要对多项式转换为小项数多项式的划分组数进行选择与比较,将此划分表示为$\alpha-$Round PtNTT(αPtNTT),运算复杂度表达式为$(n\cdot 2^{\alpha-1}-n/2^{\alpha}-1.5n\alpha+3n+1.5n\log_2 n)$.

在推导公式的过程中,为降低时间复杂度,在系数相乘时尽可能减少点乘的次数.借鉴文献[14]中采用Karatsuba算法进行优化的思想,点乘所需的数据都在两组多项式相对应的分组中,具体称为KNTT.具体运算复杂度还可根据最初划分组数进行进一步的降低,本文将就其划分表示为$\alpha-$Round KNTT(αKNTT),接下来将就α的选择进行具体分析.

整体时间复杂度随着α发生变化.其中,$n/2^{\alpha}$维NTT需要$2^{\alpha+1}$次,进行2^{α}次$n/2^{\alpha}$维INTT,需要$(2^{2\alpha-2}+3\cdot 2^{\alpha-1}-1)$次$n/2^{\alpha}$维的点乘运算,具体包括:$2^{\alpha}$次对应分组点乘运算$\widehat{\mathrm{NTT}}(\overline{f}_i(z))\circ\widehat{\mathrm{NTT}}(\overline{g}_i(z))$,$C_{2^{\alpha}}^{2}$次对应NTT分组结果先相加后点乘运算$(\widehat{\mathrm{NTT}}(\overline{f}_i(z))+\widehat{\mathrm{NTT}}(\overline{g}_i(z)))\circ(\widehat{\mathrm{NTT}}(\overline{f}_j(z))+\widehat{\mathrm{NTT}}(\overline{g}_j(z)))$,以及$(2^{\alpha}-1)$次关于$\widehat{\mathrm{NTT}}(z)$的点乘运算.最终时间表达式如下:

$$\begin{aligned}T(\alpha)&=2^{\alpha+1}\cdot\frac{n}{2^{\alpha+1}}\log_2\frac{n}{2^{\alpha}}+2^{\alpha}\cdot\frac{n}{2^{\alpha+1}}\log_2\frac{n}{2^{\alpha}}\\&\quad+\left(2^{2\alpha-1}+3\cdot 2^{\alpha-1}-1\right)\frac{n}{2^{\alpha}}\\&=n\cdot 2^{\alpha-1}-\frac{n}{2^{\alpha}}-\frac{3}{2}n\alpha+\frac{3n}{2}+\frac{3}{2}n\log_2 n\end{aligned}\tag{5}$$

$(\alpha\geqslant 1\text{且}\alpha\in Z)$

通过一次求导$T'(\alpha)=n(\mathrm{In}2(2^{\alpha-1}+2^{-\alpha})-1.5)$和二次求导$T''(\alpha)=n(\mathrm{In}2)^2(2^{\alpha-1}-2^{-\alpha})$分析,当$\alpha=2$时,$T(\alpha)$取得最小值.因此,本文选择2KNTT的方法进行多项式乘法的计算.最终具体算法如算法3所示,其中$\overline{h_0}(z)$,$\overline{h_1}(z)$,$\overline{h_2}(z)$与$\overline{h_3}(z)$共同组成多项式对应的点值,在此过程中需要8次$n/4$位的NTT,8次$n/4$位的INTT与13次$n/4$位的模乘运算,满足上述分析.

算法3 2KNTT

输入: $f(x),g(x)\in R_q=Z_q[x]/(x^n+1)$,模数$q$,维数$n$.

输出: $h(x)\in R_q$

1. $\left(\widehat{\mathrm{NTT}}(f_0),\widehat{\mathrm{NTT}}(f_1),\widehat{\mathrm{NTT}}(f_2),\widehat{\mathrm{NTT}}(f_3)\right)\leftarrow(f_0,f_1,f_2,f_3)\leftarrow f$
2. $\left(\widehat{\mathrm{NTT}}(g_0),\widehat{\mathrm{NTT}}(g_1),\widehat{\mathrm{NTT}}(g_2),\widehat{\mathrm{NTT}}(g_3)\right)\leftarrow(g_0,g_1,g_2,g_3)\leftarrow g$
3. $r_i=\widehat{\mathrm{NTT}}(f_i(z))\circ\widehat{\mathrm{NTT}}(g_i(z))$
4. $r_{i,j}=\left(\widehat{\mathrm{NTT}}(f_i(z))+\widehat{\mathrm{NTT}}(f_j(z))\right)\circ\left(\widehat{\mathrm{NTT}}(g_i(z))+\widehat{\mathrm{NTT}}(g_j(z))\right)$
5. $\widehat{\mathrm{NTT}}(\bar{h}_0(z))=r_0+\widehat{\mathrm{NTT}}(z)\circ(r_{1,3}-r_1-r_3+r_2)$

   $\widehat{\mathrm{NTT}}(\bar{h}_1(z))=r_{0,1}-r_0-r_1+\widehat{\mathrm{NTT}}(z)\circ(r_{2,3}-r_2-r_3)$

   $\widehat{\mathrm{NTT}}(\bar{h}_2(z))=r_{0,2}-r_0-r_2+r_1+\widehat{\mathrm{NTT}}(z)\circ r_3$

   $\widehat{\mathrm{NTT}}(\bar{h}_3(z))=r_{0,3}-r_0-r_3+r_{1,2}-r_1-r_2$
6. $h\leftarrow(\bar{h}_0,\bar{h}_1,\bar{h}_2,\bar{h}_3)\leftarrow\widehat{\mathrm{INTT}}\left(\widehat{\mathrm{NTT}}(\bar{h}_0),\widehat{\mathrm{NTT}}(\bar{h}_1),\widehat{\mathrm{NTT}}(\bar{h}_2),\widehat{\mathrm{NTT}}(\bar{h}_3)\right)$
7. END

注:这里使用的$\widehat{\mathrm{NTT}}$和$\widehat{\mathrm{INTT}}$代入的维数是$n/4$.其中,$\widehat{\mathrm{NTT}}(z)$的计算,对应点值$\widehat{z_i}=\gamma^{2i+1}$.以$n$=8为例,$\widehat{z_0}=\gamma^1$,$\widehat{z_1}=\gamma^3$,$\widehat{z_2}=\gamma^5$,$\widehat{z_3}=\gamma^7$,$\widehat{z_4}=\gamma^9=\gamma^8\cdot\gamma^1=-\gamma^1$,同理,$\widehat{z_5}=-\gamma^3$,$\widehat{z_6}=-\gamma^5$,$\widehat{z_7}=-\gamma^7$.

4 以提高单位面积性能为目标的结构优化

通过以上分析了解了具体的运算顺序,确定了此单元设计的流程,在此基础上需明确实际的整体电路结构设计,以确保快速实现多项式乘法.以提高单位面积性能为目标,一方面需在确定后的优化算法基础上,进行整体运算框架的优化,保证整体运算顺序需在满足数据不冲突的前提下,尽可能以最小的硬件资源消耗实现最高的性能;另一方面,增加运算单元的并行度进行性能整体上的提升,在考虑存储单元粒度和地址转换情况后,找到一种最优的并行设计机制.因此,本节围绕整体结构优化与并行设计机制两个方面展开研究.

4.1　整体运算框架优化

确定了此单元设计的整体流程,一方面需要保证在数据不冲突的前提下,通过改进整体运算结构来缩短整体运算周期,不仅减少了整体运行时间,也精简了占用面积资源;另一方面,根据具体算法需要调整统一计算公式,为之后的可重构设计打下基础. 因此,整体运算框架从以上两方面进行优化.

通过分析KNTT算法的具体流程,可观察出其每一步具体运算包括由基本蝶形运算组成的NTT算法、点乘运算、取和后乘运算、乘加运算、乘减运算、连加减运算以及由变形蝶形运算组成的INTT算法,包含运算种类过多. 针对这一问题,需要通过改变运算架构使公式中的加减乘除运算尽量统一化,从而控制结构的输入输出调度会随着公式的统一化进行精简.

起初将包括一个模乘运算的模块作为基本的调度块,由基本蝶形运算组成的NTT算法与由变形蝶形运算组成的INTT算法都包含一次模乘和一次模加/减,后者会增加一次移位运算;r_i的计算只包含一次点乘,无须再进行精简;$r_{i,j}$的计算在一次模乘的基础上还需要2次模加运算;其余运算则至多1次模乘的基础上至少有5次数据的模加减,有些甚至是作为模乘的输入. 整体来看,若是完全按照此运算架构,能囊括所有运算形式的运算单元需要11个输入数据,在保证数据不发生冲突的最大限制下,需要5轮运算,且控制结构会随着输入输出数目增多导致调度复杂化.

针对上述问题,需调整统一计算公式,并在此基础上重新搭建运算架构. 算法3中主要问题聚焦在第5步中模乘的输入数据为第4步和第3步输出结果的排列组合,且发现第5步中$r_{i,j}$的后续运算步骤中一定包含$(-r_i-r_j)$,因此将$(r_{i,j}-r_i-r_j)$整体作为第4步运算的输出结果$R_{i,j}$.

与此相对应的是,第5步的各个公式都得到了相应的化简. 最终改进算法如算法4所示.

本设计通过运算式的统一提取,为硬件上的设计提供了可简化的思路. 如图1所示,将原先的5层运算框架通过扩展6层运算框架分析转换为整体时间和面积资源复杂度更低的5层运算框架,具体体现为将其中的两层简化为关键路径更短且更易可重构运算的2层. 改进前,关键路径延时体现在算法3的第5步,模乘前后至多包括3层模加减法. 基于传统的5层运算顺序,可通过分割运算进程将关键路径缩减到模乘与一次模加,但带来的缺点为增加一层运算,根据评估,减少的时延不足以弥补周期数增多带来的时间复杂度影响. 因此,在不增加运算阶段的前提下调整运算架构,将关键路径调整为算法4的第4步,只包括模加-模乘-模加运算. 上述分析从时延角度证明了改进算法的优越性,

算法4　改进后的2KNTT

输入: $f(x),g(x)\in R_q=Z_q[x]/(x^n+1)$,模数$q$,维数$n$.

输出: $h(x)\in R_q$

1. $\left(\widehat{\mathrm{NTT}}(f_0),\widehat{\mathrm{NTT}}(f_1),\widehat{\mathrm{NTT}}(f_2),\widehat{\mathrm{NTT}}(f_3)\right)\leftarrow(f_0,f_1,f_2,f_3)\leftarrow f$
2. $\left(\widehat{\mathrm{NTT}}(g_0),\widehat{\mathrm{NTT}}(g_1),\widehat{\mathrm{NTT}}(g_2),\widehat{\mathrm{NTT}}(g_3)\right)\leftarrow(g_0,g_1,g_2,g_3)\leftarrow g$
3. $r_i=\widehat{\mathrm{NTT}}(f_i(z))\circ\widehat{\mathrm{NTT}}(g_i(z))$
4. $R_{i,j}=\left(\widehat{\mathrm{NTT}}(f_i(z))+\widehat{\mathrm{NTT}}(f_j(z))\right)\circ\left(\widehat{\mathrm{NTT}}(g_i(z))+\widehat{\mathrm{NTT}}(g_j(z))\right)-r_i-r_j$
5. $\widehat{\mathrm{NTT}}(\bar{h}_0(z))=r_0+\widehat{\mathrm{NTT}}(z)\circ(R_{1,3}+r_2)$
   $\widehat{\mathrm{NTT}}(\bar{h}_1(z))=R_{0,1}+\widehat{\mathrm{NTT}}(z)\circ R_{2,3}$
   $\widehat{\mathrm{NTT}}(\bar{h}_2(z))=R_{0,2}+r_1+\widehat{\mathrm{NTT}}(z)\circ r_3$
   $\widehat{\mathrm{NTT}}(\bar{h}_3(z))=R_{0,3}+R_{1,2}$
6. $h\leftarrow(\bar{h}_0,\bar{h}_1,\bar{h}_2,\bar{h}_3)\leftarrow\widehat{\mathrm{INTT}}\left(\widehat{\mathrm{NTT}}(\bar{h}_0),\widehat{\mathrm{NTT}}(\bar{h}_1),\widehat{\mathrm{NTT}}(\bar{h}_2),\widehat{\mathrm{NTT}}(\bar{h}_3)\right)$
7. END

另一方面从面积资源占用角度来考虑. 关键路径最短的6层运算框架需要不同的运算单元分开执行先乘后加和先加后乘运算,模乘单元的使用冗余率增加,使单位面积性能降低. 即使复用模乘单元也会使关键路径增长,与原先的优点产生矛盾. 改进前的5层运算框架经过此小节最开始的分析也可得到运算单元难于可重构的结论. 所以综合时延与占用面积资源来看,改进后的运算框架尽最大可能复用面积资源,从而在未增加运算面积的前提下减少了运算时间. 且整体运算架构的优化精简了控制结构,只需要6个输入数据,简化了输入输出调度. 后续还需在保证数据不发生冲突的前提下,进一步优化具体并行实现方案.

4.2　并行设计优化

提高运算性能是本文的目标,并行设计以及尽可能减少关键路径长度为实现这一目标的主要方法. 从并行设计角度来看,不是以数倍的面积带来数倍时间上的缩减,而是在并行化的同时以较少的面积增量带来较大的性能提高. 诚然,并行度越高,需要的时钟周期越短,然而,这并不是无限制的. 考虑到单位面积性能的提高,并行度P的选择需从减少时延与占用面积两个角度进行分析.

实际上,关于2KNTT算法的并行度设计与关于适应NTT和INTT运算性质的存储单元设计息息相关. 若整体NTT运算的实现方式为串行,需不断更新存储单元的读写地址信号,则最初两个多项式系数存入所有RAM中,每轮具体蝶形运算从RAM中读写数据,具体地址信号变换如图2所示. 从图中可看出,每次运算需要从RAM中同时读写两组数据,双端口RAM无法实现此功能. 若通过复制相同两份RAM进行同时更新与实

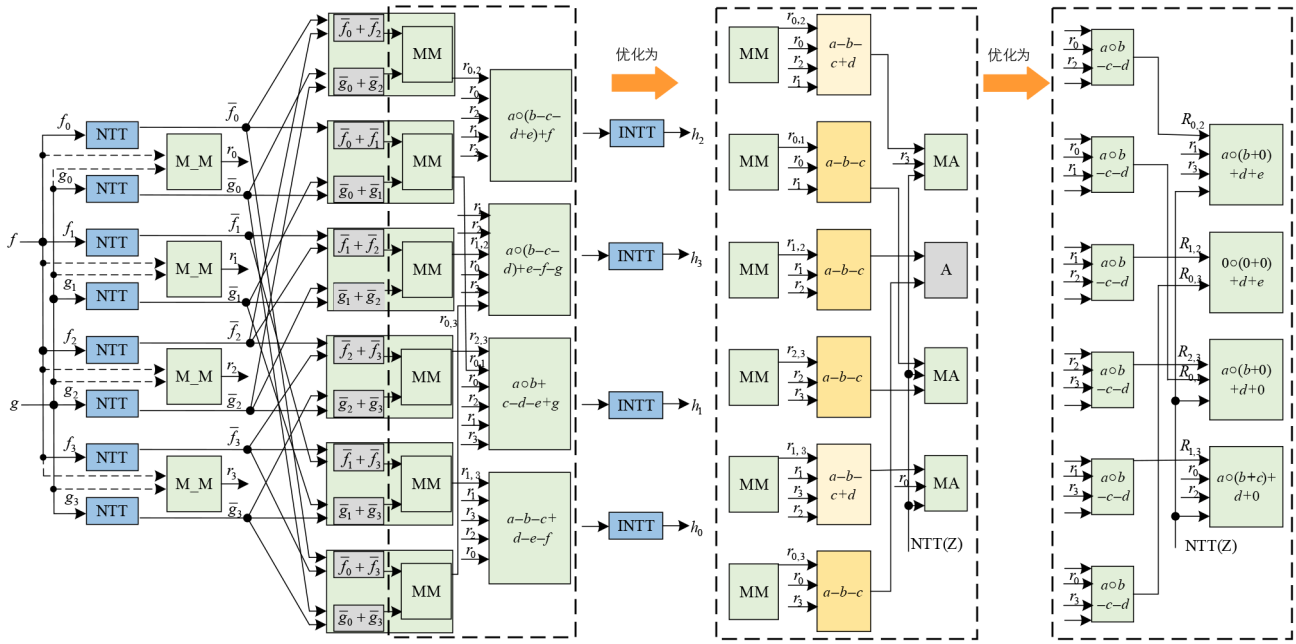


图1 运算式统一提取前后的运算顺序框架对比

注：M_M代表点乘，MM代表$(a+b)\circ(c+d)$，Ma代表$(a+b)\circ c$.

时交互数据值，会大大增加占用面积和存储控制单元设计的复杂度. 故本设计希望在不增加存储单元RAM总面积的前提下，将一组RAM划分为几个BANK，地址之间不产生冲突，做到能实时独立更新数据. 同时，若RAM分组数量越多（即存储粒度越小），高并行度设计优化的可行性越大.

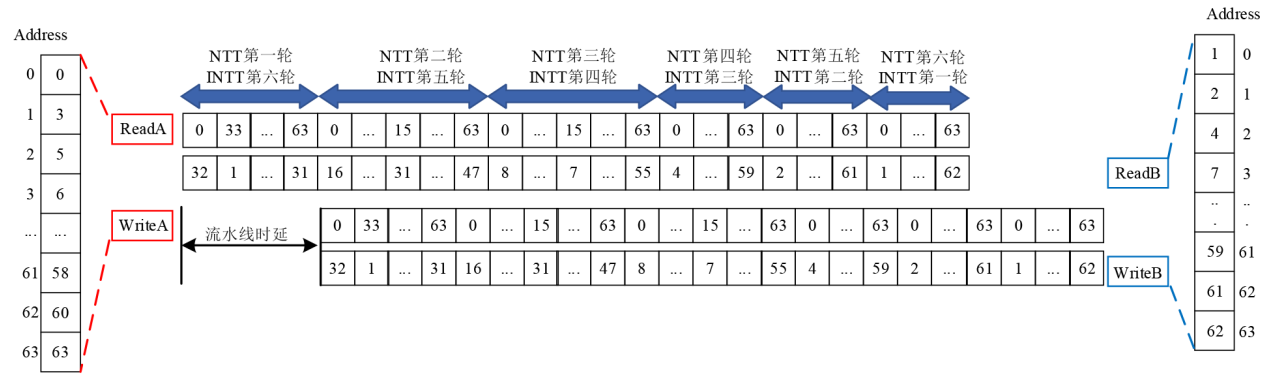


图2 NTT/INTT具体地址信号变换图

首先，需要证明适应NTT和INTT运算性质的存储单元分成的BANK可满足地址之间互不产生冲突，可实时独立更新数据的可行性，这样才能保证存储单元资源占用不会随着存储粒度的减小而增加.

此问题实质上为m-染色优化问题. 给定无向连通图和m种不同的颜色. 若一个图最少需要m种颜色才能使图中任意相邻的2个顶点着不同颜色，则称m为该图的色数. 求一个图的最小色数m的问题称为m-染色优化问题. 现已知本设计中的地址冲突情况，即图中任意相邻点的情况，地址冲突对应图中的点相邻，通过算法得到的m种颜色对应最终每个RAM内的具体Bank数量. 已知图中各点的具体相连情况，通过回溯的方式不断地为每一个节点着色，通过枚举给定可用的颜色种类，比较此节点相邻节点颜色的种类，来判断此着色是否合理. 具体算法如算法5所示.

通过算法模拟，当串行实现NTT时（代表地址之间的矛盾情况为最小程度且图的形式最简单），最小色数为2，代表最终每组RAM划分为两个BANK. 随着并行组数增加，同时刻需要读写的数据成倍增加. 根据算法模拟的结果可知，当并行组数为2的幂次时，最小色数也会随着并行组数成倍增加，保证了并行度与存储粒度的互相制约，说明了最佳并行度的选取需考虑存储

算法5　m-染色优化算法

给定无向图G,其中,顶点个数为n,可用染色种类数量为e

1. FOR i from 0 to $n-1$　　//初始化G内所有顶点为未染色状态
   $v[i]=0$;
2. $G[0]=1$;$v[0]=1$;　　//初始点染色为1,并将初始点置为染色状态
3. color=0;　　//初置颜色种类为0
4. FOR each v in G　　//将所有点的颜色初始化为种类0
   $G[n]=0$;
5. 循环至所有顶点着色
   color++;　　//取下一种颜色,直至e取完
   FOR each v in G
   IF $v[i]==1$ //若第i个点已染色,转至下一顶点
   ELSE
   若该点用color着色与其他相邻点不冲突:$G[i]$=color;
   否则不染色
6. END

粒度的限制. 同样,最终存储粒度的选定需在最佳并行度的基础上进行设计. 以上模拟结果和分析保证了存储单元不会随着粒度的减小而面积增加产生冗余,并为后续最佳并行度的选取增加了一项限制因素.

起初从低并行度来分析,影响因素包括并行度设置导致的最大存储粒度以及较少运算单元数量产生的数据冲突对存储单元数量的影响. 由于本设计采用的为2KNTT算法,最初将多项式分为4组,并行度基量设为4. 为对并行度进行更加全面的分析,最终将低并行度的选择范围设置为2~8. 但奇数类型的并行度不能保证在进行单元分配时,在进行相同运算时,每个单元实现相同数据大小的运算,所以具体是并行度为2,4,6与8之间的比较. 然而在对并行度为2和6的设计模型进行评估时发现,整体设计极不规整. 前者由于运算单元过少导致数据冲突带来的存储单元明显增多,冗余面积增加,所以单位面积性能远远不如其他并行度模型. 后者虽然存储单元数量与并行度为8的设计模型相同,但是由于此并行度未能让其中的NTT运算达到最佳并行化,所以最终实现时间为$(3n/8)\cdot(\log_2 n/4)+(3n/8)$,表明增加的两个运算单元数量并未使运行时间有质的提升,单位面积所实现的性能远不如并行度为4的设计模型. 因此,需要对4路与8路模型进行进一步分析.

不同并行路数设计模型中运算单元各个阶段所需的时钟周期数如图3所示.

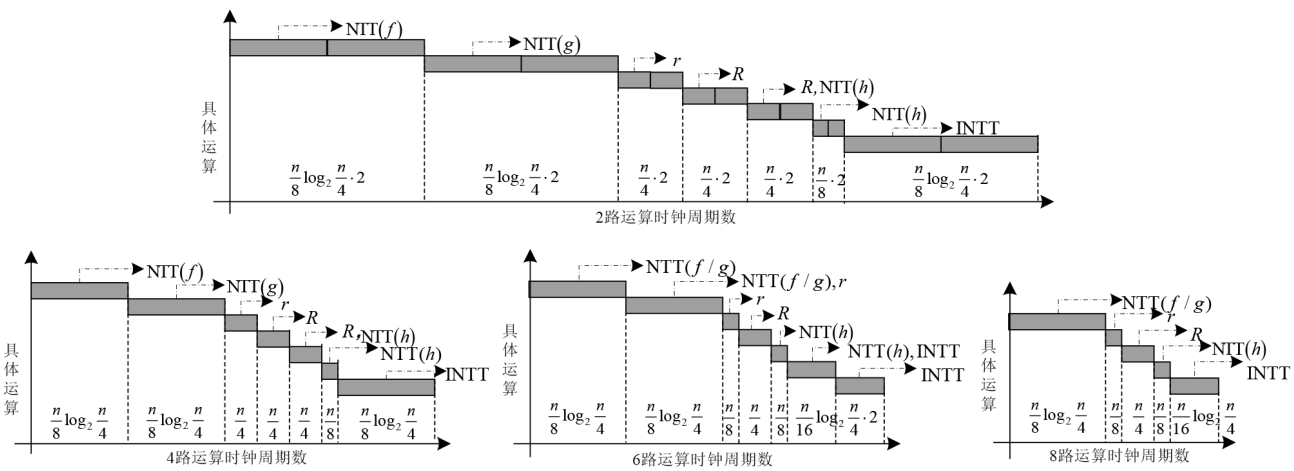


图3　4路与8路运算单元并行时钟周期数对比

在对具体运算单元划分实现阶段时,4路并行设计模型可以较好地对应不同运算,使得在每一轮运算阶段内不会发生运算单元闲置的情况,其具体运算顺序以及运算单元分配如表1所示. 但是在第5运算阶段,会出现不同运算单元实际所花时间不同的情况. 另外,由于要满足读写地址不冲突以及按照运算顺序更新RAM的合理性,虽然此设计模型符合设计算法中关于α的设置,但运算单元较少会导致数据冲突的状况,产生一些冗余的存储单元在较多的运算阶段为闲置状态. 最终,4路并行需要4个运算单元、16个RAM单元以及2个ROM单元.

若设置8个运算单元并行,每个模块进行的具体运算如表2所示. 虽然在第三运算阶段会出现运算单元闲置的情况,但是所用存储单元的数量能够精简到12个,这是由算法与并行模型的适配性决定的. 除此以外,在最后两阶段,设计了保证并行优化条件下的交错存储设计. 这是由于RAM划分组数为2时,在不同运算单元处理的数据必须存储在不同RAM中,确保不会出现从一个RAM中同时读4个数据的冲突. 在保证存储单元不增加的前提下,对输出数据进行交错存储.

分析相对低并行度的单位面积性能后,进行相对高并行度的选择与分析. 上一部分得到了非2幂次并

表1 4路并行运算单元中的具体运算

PE0	PE1	PE2	PE3
$\widehat{\mathrm{NTT}}(f_0)$	$\widehat{\mathrm{NTT}}(f_1)$	$\widehat{\mathrm{NTT}}(f_2)$	$\widehat{\mathrm{NTT}}(f_3)$
$\widehat{\mathrm{NTT}}(g_0)$	$\widehat{\mathrm{NTT}}(g_1)$	$\widehat{\mathrm{NTT}}(g_2)$	$\widehat{\mathrm{NTT}}(g_3)$
r_0	r_1	r_2	r_3
$R_{1,3}$	$R_{0,1}$	$R_{2,3}$	$R_{0,2}$
$R_{0,3}$	$R_{1,2}$	$\widehat{\mathrm{NTT}}(\bar{h}_0(z))$	$\widehat{\mathrm{NTT}}(\bar{h}_1(z))$
$\widehat{\mathrm{NTT}}(\bar{h}_2(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_2(z))$ [63:32]	$\widehat{\mathrm{NTT}}(\bar{h}_3(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_3(z))$ [63:32]
$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_0))$	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_1))$	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_2))$	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_3))$

表2 8路并行运算单元中的具体运算

PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7
$\widehat{\mathrm{NTT}}(f_0)$	$\widehat{\mathrm{NTT}}(f_1)$	$\widehat{\mathrm{NTT}}(f_2)$	$\widehat{\mathrm{NTT}}(f_3)$	$\widehat{\mathrm{NTT}}(g_0)$	$\widehat{\mathrm{NTT}}(g_1)$	$\widehat{\mathrm{NTT}}(g_2)$	$\widehat{\mathrm{NTT}}(g_3)$
r_0[31:0]	r_0[63:32]	r_1[31:0]	r_1[63:32]	r_2[31:0]	r_2[63:32]	r_3[31:0]	r_3[63:32]
$R_{1,3}$	$R_{0,1}$	$R_{2,3}$	$R_{0,2}$	$R_{0,3}$	$R_{1,2}$		
$\widehat{\mathrm{NTT}}(\bar{h}_0(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_0(z))$ [63:32]	$\widehat{\mathrm{NTT}}(\bar{h}_1(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_1(z))$ [63:32]	$\widehat{\mathrm{NTT}}(\bar{h}_2(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_2(z))$ [63:32]	$\widehat{\mathrm{NTT}}(\bar{h}_3(z))$ [31:0]	$\widehat{\mathrm{NTT}}(\bar{h}_3(z))$ [63:32]
$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_0))$ [31:0]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_0))$ [63:32]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_1))$ [31:0]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_1))$ [63:32]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_2))$ [31:0]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_2))$ [63:32]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_3))$ [31:0]	$\widehat{\mathrm{NTT}}^{-1}(\widehat{\mathrm{NTT}}(\bar{h}_3))$ [63:32]

行度的不规整设计带来的面积冗余,所以接下来从并行度为8开始,进行2幂次并行度的选择. 只有当并行组数为2的幂次时,最小色数才会随着并行组数成倍增加.

时延上,首先从关键路径角度分析,设运算单元关键路径为t_1,控制单元关键路径为t_2,一级数据选择器关键路径为t_3. 考虑到并行度P的增加带来的关键路径延长,具体体现为在控制单元组合路径上增加$(\log_2 P-3)$级数据选择器,所以最终关键路径为$\max\left(t_1, t_2+\left(\log_2 P-3\right)t_3\right)$. 接下来,从周期数上进行分析,NTT运算所需周期数为$(n/P)\cdot\left(\log_2 n-2\right)$,INTT运算所需周期数为$(n/2P)\cdot\left(\log_2 n-2\right)$,$r_i$与$\widehat{\mathrm{NTT}}(\bar{h}_i(z))$运算所需周期数为$n/P$,$R_{i,j}$运算所需周期数则为$n/\left(4\lfloor P/6\rfloor\right)$,则最终周期数为$(3n/2P)\cdot\left(\log_2 n-2\right)+(2n/P)+\left\lceil n/\left(4\lfloor P/6\rfloor\right)\right\rceil$. 从占用面积资源角度分析,运算单元面积随着$P$的变化成倍增加;由于并行度的增加会导致存储单元粒度的减小,因此控制单元会有一些数据选择器资源的增加,在地址生成部分需要5P/4级8选1数据选择器;在运算单元和存储单元之间的数据和地址选择需要P个P/4选1数据选择器,可换算为$P\cdot\sum_{i=0}^{\log(P/4)-1}2^i$个2选1数据选择器.

为了更加合理地评价此并行设计的优越性,本文构建单位面积实现时间的评价体系. 面积时间积(Area Time product,AT积)越小,代表越小的面积实现更高的性能,计算关系如式(6)所示. 其中,x_1代表运算单元的面积,x_2代表存储单元的面积,x_3代表控制单元的面积,TIME代表运行一次多项式乘法所需的时间.

$$\mathrm{AT}=\mathrm{TIME}\cdot S(x_1,x_2,x_3) \tag{6}$$

等式右边的参数根据并行度不同得到的分析结果进行评估,通过将实验仿真得到的面积消耗(以LUT数量代替)以及实际花费时间结果进行提取,且根据并行度P和项数n参数的构建,得到不同并行度设计模型在次数不同的情况下的具体花费时间以及AT积的比较图(图4). 柱状图表示不同设计时延参数的仿真结果,折线图表示不同设计AT积参数的仿真结果. 从图4可看出,随着并行度的增加,花费时间越来越短. 从单位面积性能角度来看,低并行度设计由于运算单元数量较少而无法解决数据冲突带来的存储单元面积冗余的问题,AT积随着并行度的增加而减小,且非2次幂并行度的设计明显不如其余设计的降幅明显. 高并行度由于关键路径的影响,当P=64时,关键路径开始增长,增加的面积消耗无法承载更优的时延性能. 所以无论多项式次数多少,当并行度为32时,AT积最小,单位面积实现性能最高.

综上所述,当并行度为32时,满足读写地址不冲突以及按照运算顺序更新RAM的合理性,且能实现单位面积下的最佳性能,最终需要32个运算单元,12个大小

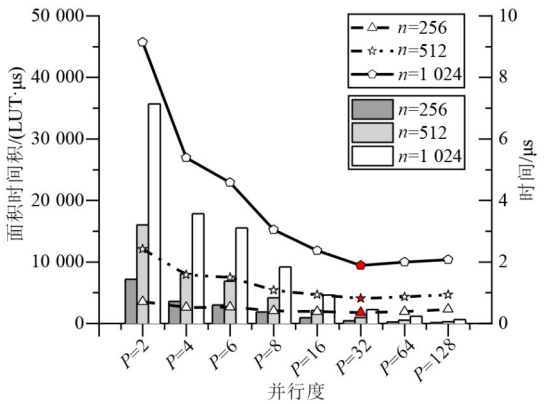


图4　不同并行度设计模型性能相关参数图

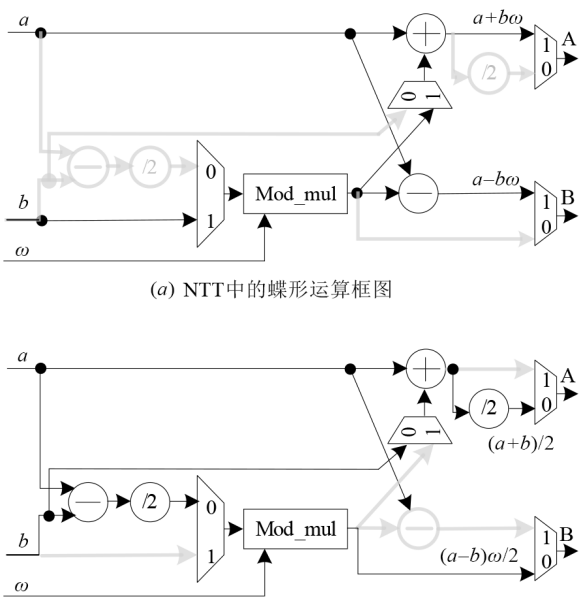
为 $(n/4)\log_2 q$ 的 RAM 单元及 2 个大小为 $(n/4)\log_2 q$ 的 ROM 单元.

5　多项式乘法单元设计研究

5.1　统一化蝶形运算单元

运算单元设计的最大前提为尽最大可能减少关键路径以及实际消耗的面积. 基于此,本文关于运算单元的设计采用可统一化的单元及5级流水线的设计,在模乘内部再加上3级流水,使整体关键路径的延迟与模加减的延迟相近.

整体运算顺序框架确定后,需确定具体运算单元的设计. 关于NTT算法,基本运算形式为蝶形运算 $x_1=a+b\omega, x_2=a-b\omega$;关于INTT算法,基本运算形式为 $x_1=(a+b)/2, x_2=(a-b)\omega/2$,实际是将前者运算形式中的蝶形运算因子进行了调换,在最后添加了 $(\times 2^{-1})$ 的操作,也都包括一次模乘运算. 故二者可进行运算上的重构,将模乘模块进行复用. 具体设计如图5所示.



(a) NTT中的蝶形运算框图

(b) INTT中的蝶形运算框图

图5　基本蝶形运算单元硬件设计框图

图5中的设计既可满足NTT的基本蝶形运算,也可实现INTT算法以及点值的模乘运算. 在此基础上,为实现改进后的2KNTT,需添加相应的运算模块,以可重构的形式完成运算. 根据时间逻辑顺序,算法4中第3步 r_i 的运算实际上是模乘;第4步 $R_{i,j}$ 的运算可表示为 $(b+m)(\omega+n)-a-p$,不再是典型的先加或减后乘的结构,而是先进行乘法操作,这意味着模乘之后需增加减法的操作;第5步的运算是典型的乘加结构,可以较容易地与上述结构进行重构. 最终具体设计如图6所示.

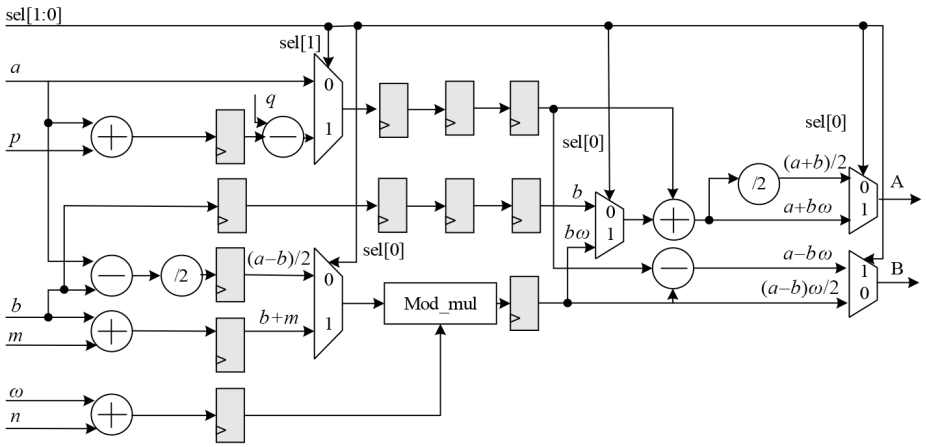


图6　改进后的蝶形运算单元硬件设计框图

统一化的类蝶形运算单元可以实现改进后的2KNTT算法中的任一操作. 除此以外,可通过流水线的设计来减小关键路径. 最初以模加减运算为最小延迟单位,每一阶段后设置一组寄存器,最终整体的关键路径消耗在模乘运算单元上. 为进一步减少关键路径,将模乘中的具体运算进一步地分割,划分为3个周期. 故

最终呈现出3级流水设计，使关键路径的延迟减少为模加减的延迟.

其中，模乘、模加与模减的具体设计为关键路径的决定因素. 模加与模减的设计较为常规，模乘的设计包括大数乘法与取模算法. 为减少常规乘法运算带来面积和时间上的损耗，先采用移位运算得到部分积；再利用三二变换，使进位与加法和的数据路径分开，此前步骤都采用并行运算，大大减少时钟周期数；最后一步进行常规的有进位运算的加法运算. 取模采用Barrett算法[15]，原理为通过此算法得到取模后商的估计值，再与模值相乘，最后通过与大数乘法积相减，得到模乘值. 具体来看，取模商值的估计以及乘法可通过将q值转换为2的幂值相加，采用移位操作和少量的加法操作来实现. 具体设计如图7所示.

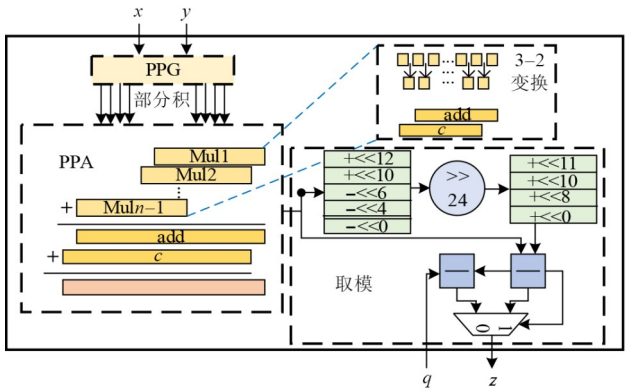


图7 大数乘法与Barrett取模单元硬件设计框图

最终，统一化的类蝶形运算单元可满足2KNTT算法不同阶段的运算，以较小的关键路径实现较高的工作频率.

5.2 具有运算友好性质的存储单元

在运算架构确定的基础上，关于其与存储单元之间的调度也需要为快速实现多项式乘法这一目标服务. 需要根据具体算法的实际需求设计出具有精简式控制结构的存储单元，控制完成多项式系数的调度、计数使能信号的产生和转换以及其他关键控制信号的产生.

接下来，在此基础上精简存储单元的复杂度. 由于最佳并行度为32，RAM分组组数则为8，可从图2看出，每一轮运算地址的转换需要计数器进行辅助设计. 整体来看，$\log_2(n/4)$轮运算之间需要一个计数器Cnt0进行轮数k的更新；每一轮运算内部需要一个计数器Cnt1进行2^{k-3}次系数的调度；具体进行系数调度时，计数器Cnt2进制与第一个轮数计数值Cnt0有关，每当加到$(n/8)/2^k$重新以全新的数字进行计数. 若按照该设计方法至少需要3个计数器，随着并行运算单元的增多，不仅实际控制结构的复杂性会增加，而且关键路径会过于冗长. 最终，8个读地址由3个计数器的值加减组合得到. 针对这一问题，需要通过布尔函数、移位等简单逻辑运算将地址转换进行优化，从而减少计数器的数目.

将计数器Cnt1(Cnt_Addr)进数值设置为最基本的$\log_2(n/4)$，实际系数是在此基础上通过移位或其他简单运算产生的；计数器Cnt0(Cnt_RAM)将Cnt0的进位信号作为计数使能信号；Cnt2则由计数器Cnt0值和Cnt2值共同决定，不需要再设置冗余的计数器. 最终读写地址则由计数器Cnt1值硬连线形成. 根据Cnt_RAM值的不同，在读数时，决定在不同的位置插入“0”或“1”. 表示为$\mathrm{Addr}=\{3'bZ_0,\mathrm{Cnt_Addr}[\log(n/32)-1,0]\}$.

填充的数据取决于不同的Bank选择，填充位置则与实现NTT的轮数Cnt1有关. 根据上述分析，本文设计可采用8个伪双端口RAM(BankA, BankB, …, BankH)组成一个8入8出的RAM单元，具体设计如图8所示. 根据地址汉明重量的奇偶性以及最高两位的不同将其分为8组. 不同Bank地址的选择生成逻辑具体为

```
Addr_BankX
  =(^Addr_X == 1'bZ1
    &&Addr_X[log(n/4) - 1, log(n/4) - 2]== 2'bZ2)
    ? Addr_X: other_Addr.
```

单个Bank的地址生成逻辑为

$$\mathrm{Addr\_Bank\_wt}=\mathrm{Addr\_wt}>>3,$$

$$\mathrm{Addr\_Bank\_rd}=\mathrm{Addr\_rd}>>3.$$

图8中的sel_x信号对应上述地址选择生成公式括号内的判断条件.

6 结果与比较

本设计在基于65 nm的CMOS标准单元库工艺下进行综合，电路的最高工作频率可达到1.1 GHz. 本文还通过Vivado 2019.1软件进行仿真综合，芯片型号为Artix-7中的xc7a35tfgg484-3，最高工作频率可达到

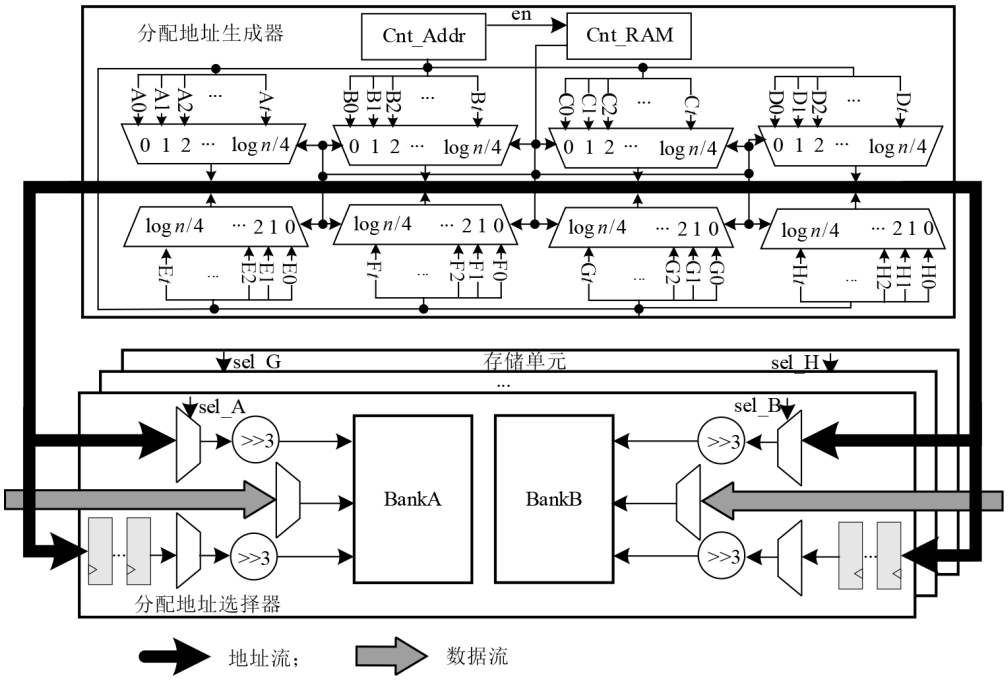


图8　多端口RAM单元硬件设计框图

152 MHz,共占用21 000个LUT硬件资源.

为了进一步验证本文设计的性能,基于不同文献中的硬件架构对同种参数的乘法实现,表3与表4分别列出了本文提出的2KNTT架构在FPGA和ASIC条件下的时延、AT积性能参数与近几年相关研究成果的比较.

表3　不同多项式乘法架构在FPGA下的性能对比

对比文献	年份	频率/MHz	时钟周期数	时延/μs	工艺	参数	资源/LUT	时延性能对比	AT积对比
文献[9]	2020	155	3668	23.665	Artix-7	n=256, q=3 329	1 416*	33.28	2.24
文献[10]	2021	161	1856	11.528	Artix-7	n=256, q=3 329	2 173*	16.21	1.68
文献[16](均衡版)	2021	182	1329	7.302	Artix-7	n=256, q=3 329	3 201*	10.27	1.57
文献[16](高性能版)	2021	172	396	2.302	Artix-7	n=256, q=3 329	10 166*	3.24	1.57
文献[17]	2022	175	256	1.463	Zynq-7035	n=256, q=3 329	14 070	2.82	1.86
本文	—	152 208	108	**0.711** **0.519**	Artix-7 Zynq-7035	n=256, q=3 329	21 000 21 294	1	1

表4　不同多项式乘法架构在ASIC下的性能对比

对比文献	年份	频率/MHz	时钟周期数	时延/μs	工艺	参数	资源/kGE	时延性能对比	AT积对比	面积周期数积对比#
LEIA[13]	2018	300	392	1.307	40 nm	n=256, q=7 681	88*	13.47	5.47	1.47
Sapphire[8]	2019	72	1289	17.903	40 nm	n=256, q=7 681	19	184.57	16.19	1.05
VPQC[11]	2020	300	41	0.137	28 nm	n=256, q=3 329	521	1.41	3.38	0.93
本文	—	1 111	108	**0.097**	65 nm	n=256, q=7 681	216.6	1	1	1
						n=256, q=3 329	213.8			

表3和表4中,“*”代表其具体资源情况来源于与本文实际所占资源的换算,其中运算单元位数(模数)相同的情况下,运算单元面积取决于数量和其中具体模乘器、模加减器的个数,因此存储单元面积设为存储数据所需占用的最小面积,控制资源理想化为本文资源的并行度比分之一. 所以最终资源换算情况如表3

与表4所示.#列“面积周期数对比”是由于ASIC设计的选择文献大多完成的是整体算法,频率受算法中其他运算模块的关键路径所影响,但由于文献中具体介绍了完成一次多项式乘法所需的周期数和NTT运算模块占用的资源,所以单独增加一列用来对比本设计在单位面积性能方面的优越性.

经对比可得,采用改进后的2KNTT算法的多项式乘法单元架构,其AT积、时延性能均优于其他文献中的数据.这是由于选用算法本身带来的时间复杂度低,整体运算架构调整和存储架构优化决定的时间和面积上的平衡,以及整体并行结构和统一化蝶形运算单元保证在性能上的提升.

从整体并行结构来看,文献[11,16,17]与本文相同,多项式乘法都是由32个运算单元并行实现.其中,文献[11]与软件里的AVX矢量指令更加契合,为快速实现多项式乘法提供了新思路,但本文设计通过降低控制复杂度得到的整体架构使AT积提升了2.38倍;文献[16,17]则是将执行多项式乘法具体划分为NTT,INTT与系数对位相乘(Coefficient-Wise Multiplication,CWM),实质上第三部分运算对应于本文采用算法中关于r_i和$R_{i,j}$的运算,且由于将其进行了相同分组之间计算的转换,同种参数在单位面积下得到了不同的性能.相较于这两种设计,本文设计采用此算法将AT积分别提升了2.24倍和1.82倍.

从并行设计角度来看,文献[9]为一路串行实行,由两个蝶形单元和一个乘法单元分别进行NTT运算,INTT运算和点乘运算来实现多项式乘法.从同款FPGA上验证结果得知,与本文设计关键路径相近,都采用了高效并行的流水线设计.本文则是由于根据3种运算不能并行的特点,将其统一到一个单元里进行运算,减小了单位运算单元的面积,使得在32倍运算单元的基础上将周期数目减少为此文献设计的1/33,实现了并行实现下的低面积成本.文献[13]中LEIA芯片则为8路并行,每个运算单元由2个蒙哥马利乘法器和4个模加减器组成.本文具体采用统一化类蝶形运算单元,一个运算单元中只有一个乘法器,减少了运算单元中乘法单元的数目;硬件使用率更高.所以从功耗上来看,本文完成一次多项式乘法仅需26.5 nJ,而文献[13]完成一次NTT却需要31 nJ.从实现时间上来看,本文流水线设计的不同减小了关键路径延时.同时,由于本文设计采用并行度为32的2KNTT算法进行实现,所以花费时钟周期数目较少,性能更高.此设计关于不同项数的可重构设计在将来可以进行研究和探讨.

由于采用算法的不同,清华大学设计的紧凑型设计[10]将256次的多项式运算转换为2个128次的多项式运算,由两个蝶形运算单元并行实现,分别处理多项式的偶数部分和奇数部分.实现算法的思想与本文1 pt划分的思想相近,其与本文相差n周期的时间复杂度符合前文第3节的比较与评估,花费时间为本文设计的16倍多.可见本文设计在性能上更具优势.

文献[8]中的运算单元与本文设计比较来看,都是将具体NTT运算与INTT运算进行了统一化设计,但由于未将INTT最终的除n运算统一在每一轮的蝶形运算中,在此部分花费的时钟周期数增多.从存储来看,文献[8]中的设计实现了一种由单端口RAM组成的存储NTT系数的内存架构.但由于整体运算的串行设计以及未有流水线设计,其存储单元的读写对应同一时间上的同一地址,所以在NTT前后需要相同大小的存储单元保存多项式对应的系数和点值.本文设计利用并行流水线设计,使存储单元没有被复制,使其利用率的最大化,可避免复制存储单元带来的面积资源损耗.

最终从实现功能来看,文献[8]与文献[13]基于n=256,q=7 681的多项式乘法,其参数不满足$q\equiv 1 \bmod n$的要求,所以具体架构与本文提出的架构不同.本文设计更适应不同的应用场景.

整体来看,本文提出的基于2KNTT算法多项式乘法架构在ASIC和FPGA上仿真实现时,总运算时间结果明显优于表中列出的设计,97 ns完成一次256位的多项式乘法,AT积都有明显提升.与高并行度设计相比,本文提出的设计很好地平衡了高并行度下运算单元统一化和存储单元调度的复杂问题,能利用更少资源实现更高的性能,减少了单位性能需要的资源占用量.

7 结束语

本文提出了一种面向格基抗量子密码中核心运算多项式乘法的$2n$次单位根预处理型快速数论变换硬件架构,可高性能实现n和q满足$q\equiv 1 \bmod n$限制要求的多项式乘法.在此过程中,对参数满足$q\equiv 1 \bmod 2n$要求多项式乘法的高性能实现具有一定的研究意义.通过与其他研究比较,在实现算法和地址转换上对实现多项式乘法提供了新思路.同时,实现该架构基于存储单元粒度对并行设计有所限制,虽然本文在粒度可变的情况下找到了最佳并行度,但下一步可在并行调度控制设计上进行探究,以适应在实现格基抗量子密码算法过程中更高的性能要求.

参考文献

[1] UMANA V G, BERNSTEIN D J, BUCHMANN J, et al. Post-Quantum Cryptography[M]. Berlin: Springer, 2009.

[2] AJTAI M. Generating hard instances of lattice problems (extended abstract) [C]//Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing. New

York: ACM, 1996: 99-108.
[3] NEJATOLLAHI H, DUTT N, RAY S, et al. Post-quantum lattice-based cryptography implementations: A survey[J]. ACM Computing Surveys, 2019, 51(6): 1-41.
[4] LYUBASHEVSKY V, MICCIANCIO D, PEIKERT C, et al. SWIFFT: A modest proposal for FFT hashing[C]//International Workshop on Fast Software Encryption. Berlin: Springer, 2008: 54-72.
[5] CHEN D D, MENTENS N, VERCAUTEREN F, et al. High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2015, 62(1): 157-166.
[6] ZHOU S, XUE H Y, ZHANG D D, et al. Preprocess-then-NTT technique and its applications to kyber and NewHope [C]//International Conference on Information Security and Cryptology. Cham: Springer, 2019: 117-137.
[7] DU C H, BAI G Q. Efficient polynomial multiplier architecture for Ring-LWE based public key cryptosystems[C]// 2016 IEEE International Symposium on Circuits and Systems (ISCAS). Piscataway: IEEE, 2016: 1162-1165.
[8] BANERJEE U, UKYAB T S, CHANDRAKASAN A P. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(4): 17-61.
[9] 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): 20200234.
[10] 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(2): 328-356.
[11] XIN G Z, HAN J, YIN T Y, et al. VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(8): 2672-2684.
[12] 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(2): 49-72.
[13] SONG S M, TANG W, CHEN T, et al. LEIA: A 2.05mm2 140mW lattice encryption instruction accelerator in 40nm CMOS[C]//2018 IEEE Custom Integrated Circuits Conference (CICC). Piscataway: IEEE, 2018: 1-4.
[14] ZHU Y M, LIU Z, PAN Y B. When NTT meets karatsuba: Preprocess-then-NTT technique revisited[C]//International Conference on Information and Communications Security. Cham: Springer, 2021: 249-264.
[15] BARRETT P. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor[M]//Advances in Cryptology - CRYPTO' 86. Berlin: Springer, 2007: 311-323.
[16] YAMAN F, MERT A C, ÖZTÜRK E, et al. A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme[C]//2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). Piscataway: IEEE, 2021: 1020-1025.
[17] 李斌, 陈晓杰, 冯峰, 等. 后量子密码CRYSTALS-Kyber的FPGA多路并行优化实现[J]. 通信学报, 2022, 43(2): 196-207.
LI B, CHEN X J, FENG F, et al. FPGA multi-unit parallel optimization and implementation of post-quantum cryptography CRYSTALS-Kyber[J]. Journal on Communications, 2022, 43(2): 196-207. (in Chinese)

作者简介

陈　韬　男,1979年生,湖北咸宁人. 现为中国人民解放军战略支援部队信息工程大学副教授. 主要研究方向为安全专用芯片设计、抗量子公钥算法芯片设计.
E-mail: 349080310@qq.com

李慧琴　女,1998年生,山西临汾人. 现为中国人民解放军战略支援部队信息工程大学电子信息专业硕士研究生. 主要研究方向为抗量子公钥算法硬件实现.
E-mail: lihuiqinjiayou@163.com