计算机科学
COMPUTER SCIENCE

Dilithium算法的FPGA高效扩展性优化

燕云飞, 李斌, 魏源鑫, 张博林, 马添翼, 周清雷

引用本文

燕云飞, 李斌, 魏源鑫, 张博林, 马添翼, 周清雷. Dilithium算法的FPGA高效扩展性优化[J]. 计算机科学, 2024, 51(6A): 230800138-9.

YAN Yunfei, LI Bin, WEI Yuanxin, ZHANG Bolin, MA Tianyi, ZHOU Qinglei. FPGA Efficient Scalability Optimization of Dilithium [J]. Computer Science, 2024, 51(6A): 230800138-9.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230800138

Dilithium 算法的 FPGA 高效扩展性优化

燕云飞　李　斌　魏源鑫　张博林　马添翼　周清雷
郑州大学计算机与人工智能学院　郑州 450001
(994982837@qq.com)

摘　要　为提高 Dilithium 在实际应用中的运行效率,提出了一种 Dilithium 算法的现场可编程门阵列(Field Programmable Gate Array,FPGA)高效扩展性优化实现。具体在以下几个方面进行优化:将 KOA(Karatsuba-Offman-Algorithm)算法与快速模约减算法相结合,构成快速模乘单元,优化数论转换(Number TheoreticTransform,NTT)实现的大量多项式乘法;采用多 RAM(Random Access Memory)存取参与运算的多项式系数,根据 Dilithium 算法的特点,设计了一种多项式系数读取策略,以快速、正确地读取 RAM 中的多项式系数。针对方案中的采样和散列工作,分析了 SHAKE 算法系列的特点,设计了一种低延迟可扩展的 Keccak 硬件架构,使得其能够根据输入信号的不同执行不同的 SHAKE 算法。实验结果表明,所提方案在频率方面相比其他方案提升了 60.7%～131.9%,兼顾硬件的资源消耗和执行效率。

关键词:Dilithium 算法;现场可编程门阵列;数论变换;硬件实现
中图分类号　TP309.7

FPGA Efficient Scalability Optimization of Dilithium

YAN Yunfei,LI Bin,WEI Yuanxin,ZHANG Bolin,MA Tianyi and ZHOU Qinglei
School of Computer and Artificial Intelligence,Zhengzhou University,Zhengzhou 450001,China

Abstract　To improve the operational efficiency of Dilithium in practical applications,an efficient field programmable gate array (FPGA) implementation of the Dilithium algorithm is proposed. Optimization is carried out in several aspects,including combining the Karatsuba-Offman algorithm(KOA) with the fast modular reduction algorithm to create a fast modular multiplication unit,optimizing the extensive polynomial multiplication achieved through number theoretic transform(NTT) implementation. Multiple RAM accesses are employed for polynomial coefficient operations,and a coefficient reading strategy tailored to the characteristics of the Dilithium algorithm is designed to achieve rapid and accurate reading of polynomial coefficients from RAM. For the sampling and hashing tasks in the scheme,the characteristics of the SHAKE algorithm series are analyzed,leading to the development of a low-latency and scalable Keccak hardware architecture,allowing it to execute different SHAKE algorithms based on the input signal. Experimental results demonstrate that the working frequency of the proposed algorithm is increased by 60.7%～131.9%,while balancing hardware resource consumption and execution efficiency.

Keywords　Dilithium algorithm,FPGA,NTT,Hardware implementation

1　引言

当前公钥密码标准,如 RSA 和椭圆曲线加密 ECC(Elliptic Curve Cryptography),其安全性依赖于整数分解和离散对数问题的难度。然而有了足够大的量子计算机,Shor 算法可以在多项式运行时间内解决这些问题。后量子密码算法是一种针对量子计算的特点,结合传统密码算法的优势,能够抵御量子计算机攻击的密码方案。不同于安全性基于量子力学物理性质的量子密码学,后量子密码与传统密码算法均是基于数学的。

虽然没有已知的量子计算机能运行 Shor 的量子整数分解算法,但设计、标准化和部署新的公钥密码系统的过程可能需要大量的时间。美国国家标准与技术研究院(National Institute of Standards and Technology,NIST)从 2016 年起征集了后量子密码的标准化评选,提出了众多基于格的后量子密码方案[1]。近期 NIST 最终选择了 4 种加密算法,旨在抵御未来量子计算机的攻击。其中 Dilithium 作为一种数字签名算法,通常用于在数字交易期间验证身份、远程签署文档、物联网、加密视频通话等场景。FPGA 因其编程灵活性和良好的性能平衡而成为实现密码算法的重要平台之一。本文选用可重构硬件 FPGA 实现高效的 Dilithium 密码算法,在具有高性能需求的嵌入式场景中,该设计具有显著优势。

在 Dilithium 格密码方案中[2],多项式乘法运算作为算法关键步骤,消耗了绝大部分时间和资源。数论转换(Number Theoretic Transform,NTT)可以快速实现多项式乘法。蝶形运算是 NTT 中的核心运算结构,在 NTT 计算过程中,蝶形

基金项目:河南省科技攻关项目(232102211055);河南省网络密码技术重点实验室研究课题(LNCT2022-A14)
This work was supported by the Key Science and Technology Research Project of Henan Province,China(232102211055) and Research Project of Key Laboratory of Network Cryptography Technology of Henan Province,China(LNCT2022-A14).
通信作者:李斌(iebinli@zzu.edu.cn)

运算被多次执行，且蝶形运算单元包含模加减、模乘、模约简等运算，极大地影响了硬件计算效率。其次，NTT 控制逻辑计算结构复杂，大量的多项式系数的存取和调度复杂。因此，利用可重构硬件 FPGA 实现高效的 NTT 多项式乘法，有助于提升算法整体性能。

目前，国内外学者已对 Dilithium 算法进行了实现、优化。文献[3]针对多项式运算进行优化：使用 Barrett 约减，采用独特的蝶形单元排列以减少写回冲突；在签名阶段将拒绝循环拆分为两级管道。文献[4]在 NTT 组件里加入多个蝴蝶运算单元以实现并行化，并且利用了 FPGA DSPs 的多种操作模式，其 NTT 达到了 311 MHz。文献[5]用硬件描述语言(VHDL)实现了 Dilithium 的所有基本功能和 NTT 组件，因此效率较高。文献[6]通过复用相同的蝴蝶单元提高了其利用率。

Dilithium 算法仍有以下优化空间：1)NTT 算法中计算流程、内存管理、硬件架构还存在优化空间。2)Keccak 算法设计在应用场景方面仍有扩展的可能。基于此，本文采取的优化如下：

1)将格基密码方案中主流使用的数论变换算法结合 KOA(Karatsuba-Offman-Algorithm)进行了实现，并设计流水线结构，避免了额外的乘法运算，减少了乘法次数，实现了在一个时钟之内得到取模结果。

2)在蝶形运算中，采用新的系数访存模式：利用蝶形运算阶段和循环次数对读取索引进行控制；多 RAM 存储交叉计算后的系数；流水线化蝶形运算单元，提高数据吞吐量；采用适用于 Dilithium 算法模值的快速模块化模约减硬件单元，避免了额外的除法运算。

3)为了支持采样器的采样工作，本文分析了 SHAKE 算法系列的特点，设计了一种低延迟可扩展的 Keccak 硬件架构，使其能够根据输入信号的不同执行不同的 SHAKE 算法，进一步降低了资源消耗。

2 Dilithium 算法简介

Dilithium 的 3 个核心算法分别是密钥生成、签名生成和签名验证。算法中需要多次利用 SHAKE128 和 SHAKE256 进行采样，不同阶段所使用的 SHAKE 算法不同，执行的功能也不同。

Dilithium 密钥生成算法如算法 1 所示。首先将从 256 位二进制串中均匀采样出的种子传入函数 H 中，以生成种子 ρ，σ，K，其中 H 被实例为 SHAKE256。ρ 传入函数 $ExpandA$ 中，以生成 NTT 域上的矩阵 $\boldsymbol{A}$，以 σ 为参数通过函数 $ExpandS$ 生成(s_1, s_2)。函数 $Powe2Round_q$ 的功能是将输入分解为“高阶”位和“低阶”位，计算公钥和密钥的组件。与传统 M-LWE(Module Learning With Errors)问题描述不同的是，Dilithium 密钥生成算法使用种子 ρ 而不是实际的 $\boldsymbol{A}$ 矩阵，并且只包含 t 的高位。

算法 1 Dilithium Key Generation

输入：$\zeta \in \{0,1\}^{256}$

输出：$pk=(\rho, t_1)$，$sk=(\rho, K, tr, s_1, s_2, t_0)$

1. $(\rho, \sigma, K) \leftarrow H(\zeta)$
2. $\boldsymbol{A} \leftarrow ExpandA(\rho)$
   $(s_1, s_2) \leftarrow ExpandS([(\sigma,0),(\sigma,1)])$
3. $t \leftarrow \boldsymbol{A} \times s_1 + s_2$
4. $(t_0, t_1) \leftarrow Power2Round(t, d)$
5. $tr \leftarrow \{0,1\}^{256} = H(\rho \| t_1)$

Dilithium 签名生成算法如算法 2 所示。函数 $ExpandMask$ 将(ρ', k)映射到 $y \in \widetilde{S}_{r_1}^{l}$，其中，它独立地计算 y 的每一个系数，它们是 $\widetilde{S}_{r_1}^{l}$ 中的多项式。函数 $SampleInBall$ 将 $\hat{c}$ 散列到 B_τ 上。$MakeHint$ 函数生成可以恢复和的高阶位的线索 h。

算法 2 Dilithium Signature Generation

输入：$sk=(\rho, K, tr, s_1, s_2, t_0)$ $M \in \{0,1\}$

输出：$\sigma=(\hat{c}, z, h)$

1. $A \leftarrow ExpandA(\rho)$ $\mu \leftarrow H(tr \| M)$ $\rho' \leftarrow H(K \| \mu)$
2. $k \leftarrow 0$ $abort \leftarrow 1$
3. WHILE abort DO
4. $abort \leftarrow 0$
5. $y \leftarrow ExpandMask(\rho', k)$
6. $w \leftarrow \boldsymbol{A} \times y$
7. $w_1 \leftarrow HighBits_q(w, 2r_2)$
8. $\hat{c} \leftarrow H(\mu \| w_1)$
9. $c \leftarrow SampleInBall(\hat{c})$
10. $z \leftarrow y + c \times s_1$
11. $r_0 = LowBits(w - c \times s_2, 2r_2)$
12. IF $\| z \|_\infty \geqslant r_1 - \beta$ OR $\| r_0 \|_\infty \geqslant r_2 - \beta$ THEN
13. $abort \leftarrow 1$
14. ELSE
15. $h \leftarrow MakeHInt(-c \times t_0, w - c \times s_2 + c \times t_0, 2r_2)$
16. IF $\| c \times t_0 \|_\infty \geqslant r_2$ OR $\sum h_i > \omega$ THEN
17. $abort \leftarrow 1$
18. $k = k + 1$

Dilithium 验签算法如算法 3 所示。在算法 3 中，首先将种子 ρ 映射到 NTT 域表示的矩阵 $\boldsymbol{A}$，种子 ρ 与 t_1 拼接后输入散列函数中，并将结果与消息 M 拼接后传输到散列函数中得到 μ，将质询种子 $\hat{c}$ 散列到 B_τ 上得到 c，计算 $\boldsymbol{A} \times z - c \times t_1 \times 2^d$ 并将结果分为高位 w_1 和低位 w_0，最后使用 μ 对 w_1 进行散列，并与质询种子 $\hat{c}$ 进行比较，以确定签名是否有效。

算法 3 Dilithium Signature Verification

输入：$pk=(\rho, t_1)$，$M \in \{0,1\}^*$，$\sigma=(\hat{c}, z, h)$

输出：Valid or Invalid

1. $\boldsymbol{A} \leftarrow ExpandA(\rho)$
2. $\mu \leftarrow H(H(\rho \| t_1) \| M)$
3. $c \leftarrow SampleInBall(\hat{c})$
4. $(w_1, w_0) \leftarrow UseHInt(h, \boldsymbol{A} \times z - c \times t_1 \times 2^d)$
5. IF $\| z \|_\infty < r_1 - \beta$ && $c = H(\mu \| w_1)$ && $h_i \leqslant \omega$ THEN
6. RETURN Valid
7. RETURN Invalid

3 核心运算优化

在 Dilithium 中需要做大量多项式乘法运算，利用 NTT，INTT(Inverse Number TheoreticTransform)可以大大加快多项式的硬件计算速度[7]，但 NTT，INTT 中存在的大量模加、模减、大数乘法、求模运算等操作严重消耗了硬件资源，对这些操作进行化简、替代从而提高算法运行效率。

3.1 多项式运算模块

为提高 Dilithium 算法的灵活性和可扩展性，在多项式运算单元内放置 4 个蝶形运算单元，各个蝶形运算单元都能够执行基本的算术运算以及蝶形运算。NTT，INTT 每次迭代

时所读取和存储的参数不同,因此放置多 RAM 通道对多项式系数和每次迭代的结果进行存取。对于参数 ω,采用预计算的方式提前计算好,并在 ROM(Read-Only Memory)中存储。最后,由控制逻辑完成对蝶形运算单元的输入输出的控制、RAM 存取地址的控制及参数 ω 的读取控制。此外,PWM(Point-wise Multiplication)算法中进行的多次模乘的中间运算结果以 mul 输出,并返回 RAM 中参与运算。多项式运算整体结构如图 1 所示。可通过配置旋转因子 ω、数据位宽 *data_width* 等参数,设置相应数量的 RAM 和蝶形运算单元,支持其他格密码算法。整个结构以松耦合方式互连,控制单元参与调度,并根据参数配置灵活完成 NTT,INTT,PWM 运算。

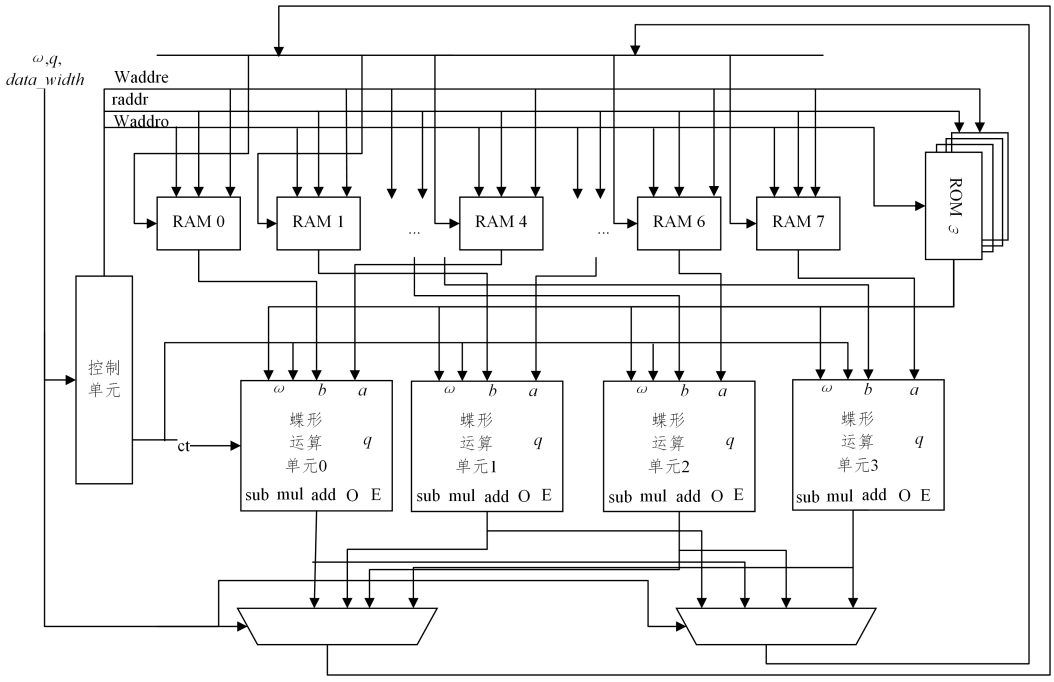

图 1　多项式运算整体结构

Fig. 1　Overall structure of polynomial operation

3.2　蝶形运算流水线优化

NTT,INTT,PWM 计算的关键在于蝶形运算单元,为提高吞吐量,将蝶形运算单元进行流水线并行加速优化。由于 NTT,INTT 两者计算的先后顺序不同,因此在蝶形运算单元中插入多个缓存寄存器,用来缓存中间结果,使得 *even* 和 *odd* 在输出处同步。同时,根据控制输入信号 ct 将缓存中间结果进行仲裁输出,通过缓存达到降低路径时延的效果。蝶形运算单元结构如图 2 所示。

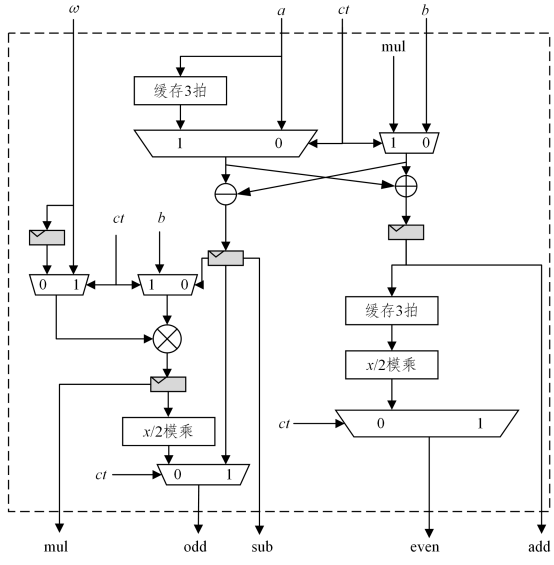

图 2　蝶形运算单元结构

Fig. 2　Structure of butterfly operation unit

3.3　KOA 乘法优化

Dilithium 中需要进行大量的多项式乘法运算。KOA 算法起初应用于大数乘法,也可以应用在多项式乘法。KOA 算法是一种基于分治策略的算法,它的核心思想是通过增加额外的移位和加法操作来减少乘法操作。例如两个 n 位整数 X 和 Y 相乘,已有 $X=A\parallel B, Y=C\parallel D, XY$ 乘法如式(1)所示。

$$\begin{aligned} X\times Y &= \left(\left(A\ll\frac{n}{2}\right)+B\right)\times\left(\left(C\ll\frac{n}{2}\right)+D\right) \\ &= ((AC)\ll n+\left((AD+BC)\ll\frac{n}{2}\right)+BD \end{aligned} \tag{1}$$

对于$(AD+BC)$部分,可重复利用乘积结果 AC 和 BD,将原式转化为$(A+B)(C+D)-AC-BD$,从而减少了一次乘法运算。对于两个 n 位数相乘,使用 KOA 算法可以将复杂度从 $O(n_2)$降低到 $O(n\log 23)$,消耗门电路从 $O(n_2)$降低为 $O(n\mu)$,其中 $1<\mu<2$。使用 *KOA* 乘法器相比传统乘法性能更优且资源消耗低。

算法 4　KOA 乘法

输入:X,Y,n

输出:P

1. IF n=1 THEN P=X×Y
2. ELSE
3. SumX=High(X)+Low(X)
4. SumY=High(Y)+Low(Y)
5. $P_0=\mathrm{KOA}\left(\mathrm{High(X)},\mathrm{High(Y)},\frac{n}{2}\right)$
6. $P_1=\mathrm{KOA}\left(\mathrm{Low(X)},\mathrm{Low(Y)},\frac{n}{2}\right)$
7. $P_2=\mathrm{KOA}\left(\mathrm{Sum(X)},\mathrm{Sum(Y)},\frac{n}{2}\right)$

8. $P_3=P_2-P_1-P_0$

9. $P=(P_0\ll n)+\left(P_3\ll\frac{n}{2}\right)+P_1$

KOA 乘法器通过调用 FPGA 底层 DSP 实现。同时，为了提高 KOA 工作频率，以寄存器对中间结果缓存，并形成 2 级流水线。本文中 23 bit 数据先补全为 24 bit，再通过 KOA 拆分到 6 bit 进行运算。KOA 乘法器对应的电路如图 3 所示。

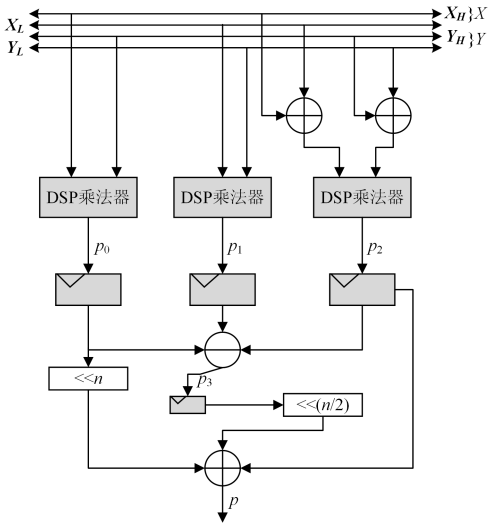

图 3 KOA 乘法器电路图

Fig. 3 Circuit diagram of KOA multiplier

3.4 模约减优化

NTT，INTT 存在着大量的模加、模减和模乘。在硬件设计时，大量的取模运算必然会出现繁杂的除法操作，不仅会加剧硬件资源消耗，而且会导致算法整体运算时间的增加。Dilithium 的参考实现[8]使用 Montgomery 算法进行乘法后的模约减。该算法适用于硬件，但该算法需要额外的乘法，并且没有利用模数 q 的特殊性质。另一种常用的算法是 Barrett 约减，Banerjee 等[9]发现，Barrett 约减对 Dilithium 更有效，但算法包含更长的进位链，限制了最大频率。

针对 Dilithium 中的模数 q，使用类似文献[9]的算法对取模运算进行快速模约减。为了便于描述，首先介绍 Verilog 硬件描述语句的语法规则，$t[a,b]$表示一个$(b-a+1)$位宽的数据，λ 表示 μ 和 v 的乘法结果，其中 $0\leqslant\mu\leqslant q$，$0\leqslant v\leqslant q$。所以 λ 的最大值为 46'h3D96A7772C21，表示的最大位宽为 46 bit 位。为了能够有效而快速地求出 $\lambda \bmod q$，我们根据 $2^{23}\equiv 2^{13}-1 \bmod q$ 这一关系，递归地调用设计好的模约减算法进行求模运算。模约减算法的具体流程如算法 5 所示。

算法 5 模约减

输入：λ[45:0] 模数 q=8380417

输出：λ[45:0] mod q

1. x=λ[45:43]+λ[42:33]+λ[32:23]
2. y=λ[45:43]+λ[45:33]+λ[45:23]
3. z=λ[22:0]
4. R1=((x[11:10]+x[9:0]≪13)−(y+x[11:10])+z
5. Diff1=R1−q，Diff2=R1+q，Diff3=R1−(q≪1)
6. IF R1[24]==1 THEN
7. R2=Diff3
8. ELIF Diff3[24]==0 THEN
9. R2=Diff3
10. ELIF Diff1[23]==0 THEN
11. R2=Diff1
12. ELIF Diff2[23]==0 THEN
13. R2=Diff2
14. ELSE
15. R2=R1
16. R=(R1[24:16]==0)? R1:R2
17. RETURN R

3.5 模加减优化

在蝶形运算单元中，模加减由组合逻辑实现，并且可依据加减结果是否超出素数 q(8380417)的范围，再进行处理，从而降低硬件设计的复杂度，其结构如图 4、图 5 所示。

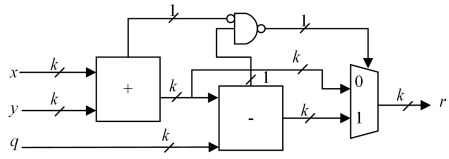

图 4 模加硬件结构

Fig. 4 Modular addition hardware structure

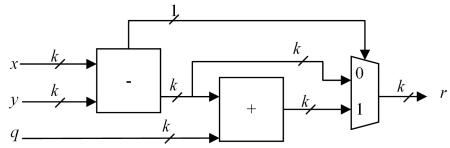

图 5 模减硬件结构

Fig. 5 Modular subtraction hardware structure

利用图图 4、图 5 所示的组合逻辑方式实现模加减，在硬件中不仅实现简单，而且易于操作，节省了资源开销。

3.6 多 RAM 存取控制优化

图 6 给出了 NTT 的系数访问模式，这种访问模式称为蝶形运算。

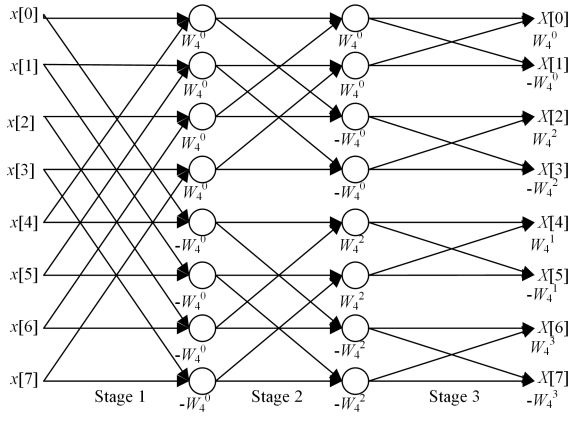

图 6 8 个点的蝶形运算

Fig. 6 Butterfly operation with 8 points

NTT 的不规则访问模式要求每个系数存储在一个唯一的地址，且属于同一蝴蝶操作的系数对需要分别存储在两个不同的内存块中。在 NTT 循环期间，新生成的系数被写回 RAMs 中，这样在 NTT 循环的下一次迭代期间，每个蝴蝶所需的系数可以成对地从内存中读取。图 7 展示了 8 个系数的内存访问例子，绿色和蓝色格子分别代表读和写操作，红色数字代表已写入内存的系数。第一阶段 4 个系数对(0,4)，(1,5)，(2,6)，(3,7)要进行蝴蝶操作，这些系数对需要在相同的时钟周期内读取，0，1，2，3 和 4，5，6，7 应该存储在不同的内存块中并行读取。这些系数对在每个阶段都会发生变化，因此

在一个阶段输出时需要将新生产的系数根据下一阶段的系数对写回 RAM 中。如图 7 所示，在第二阶段需要将(0，4)(1，5)写回 RAM0 中，同样地，将(2，6)，(3，7)写回 RAM1 中。

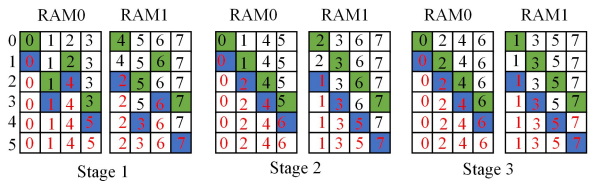

图 7　8 个点的内存访问

Fig. 7　Memory access at 8 points

对于 Dilithium 算法，多项式有 256 个系数，也就是 $N=256$，共需要执行 8 轮蝶形运算，RAM 的存取更为复杂。以下介绍 Dilithium 算法的访存优化方法。

在读取参数时，为方便操作，对参数进行重排，统一读取 8 个 RAM 的某一地址，获取所有 256 个参数并组成系数对。因此，在写入中间结果时，需要对输出的 *even*(NTT 的偶数输出)和 *odd*(NTT 的奇数输出)分别按不同的地址进行存储，以保证在下一轮可以从同一个地址再次读取到对应的系数对。具体地，采用 1 个地址 *raddr*(读 RAM 地址)对所有 RAM 进行读取，2 个地址 *waddre*(写入 RAM 偶地址)和 *waddro*(写入 RAM 奇地址)分别完成 *even* 和 *odd* 的 RAM 写入。在对这 3 个地址的参数进行设置时，由于 NTT，INTT，PWM 共进行 8 轮蝶形运算，且 RAM 块中存储空间最多有 64 个，因此这 3 个地址的后 5 bit 由所做运算的轮数以及循环次数决定。

以 NTT，INTT 运算为例，具体的 *raddr*，*waddre* 和 *waddro* 计算如算法 6－算法 9 所示。

算法 6　NTT 阶段 RAM 读地址的变换

输入：蝶形运算阶段 c_stage[3:0]，循环次数 c_loop[5:0]

输出：RAM 读地址 raddr[4:0]

1. 初始化 raddr[4:0]=0
2. IF(c_stage<5)
3.　IF(~c_loop[0])
　　raddr=(c_loop≫1)+((c_loop≫(5−c_stage))≪(4−c_stage))
4.　ELSE
　　raddr=(1≪(4−c_stage))+(c_loop≫1)+((c_loop≫(5−c_stage))≪(4−c_stage))
5. ELSE
6. RETURN raddr

算法 7　NTT 阶段 RAM 写地址的变换

输入：蝶形运算阶段 c_stage[3:0]，循环次数 c_loop[5:0]

输出：RAM 写地址 waddre[4:0]，waddro[4:0]

1. 初始化 waddre[4:0]=0；waddro[4:0]=0
2. IF(c_stage<5)
3.　waddre=(c_loop≫1)+((c_loop≫(5−c_stage))≪(4−c_stage))
4.　waddro=(c_loop≫1)+((c_loop≫(5−c_stage))≪(4−c_stage))+(1≪(4−c_stage))
5. ELSE waddre=c_loop
6.　waddro=c_loop
7. RETURN waddre，waddro

算法 8　INTT 阶段 RAM 读地址的变换

输入：蝶形运算阶段，c_stage[3:0]，循环次数 c_loop[5:0]

输出：RAM 读地址 raddr[4:0]

1. 初始化 raddr[4:0]=0
2. IF(c_stage≥2)
3.　IF(~c_loop[0])
　　raddr=(c_loop≫1)+((c_loop≫(c_stage−2))≪(c_stage−3))
4. ELSE raddr=(1≪(c_stage−3))+(c_loop≫1)+((c_loop≫(c_stage−2))≪(c_stage−3))
5. ELSE
6. RETURN raddr

算法 9　INTT 阶段 RAM 写地址的变换

输入：蝶形运算阶段 c_stage[3:0]，循环次数 c_loop[5:0]

输出：RAM 写地址 waddre[4:0]，waddro[4:0]

1. 初始化 waddre[4:0]=0；waddro[4:0]=0
2. IF(c_stage≥2)
3.　waddre=(c_loop≫1)+((c_loop≫(c_stage−2))≪(c_stage−3))
4.　waddro=(c_loop≫1)+((c_loop≫(c_stage−2))≪(c_stage−3))+(1≪(c_stage−3))
5. ELSE waddre=c_loop
　　waddro=c_loop
6. RETURN waddre，waddro

在算法 6 中，当 c_stage 大于 5 时，直接根据 c_loop 即可获取到 RAM 的读地址。对于其他情况，则需要根据当前 c_stage 和 c_loop 进行额外的计算，从而获得 RAM 的读地址。

同样地，在算法 7 中，当 c_stage 大于 5 时，可直接根据 c_loop 获取到 RAM 的写地址，而其他情况要根据当前 c_stage 和 c_loop 进行计算，从而获得 RAM 的写地址。INTT 阶段 RAM 读写索引类似。算法 8、算法 9 类似。为解决蝶形运算单元的地址冲突问题，每两个 RAM 对应一个蝶形运算，8 个 RAM 分配给对应的 4 个蝶形运算单元。由于读写 RAM 需要先预计算出地址，再进行读写运算，而且由于蝶形运算单元采用流水线结构设计，因此需要对预计算结构进行打拍缓存。NTT 运算的 RAM 存取索引如图 8 所示。

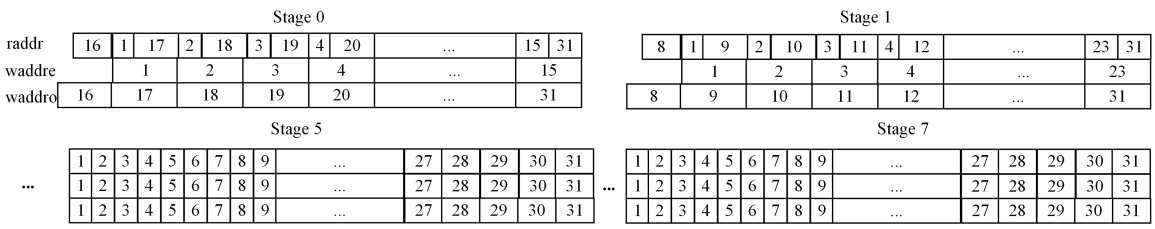

图 8　RAM 读写索引

Fig. 8　RAM read and write index

3.7　SHAKE128 和 SHAKE256 优化

SHAKE128 和 SHAKE256 都是 NIST 标准选定的 SHA-3 系列的可扩展输出的加密哈希函数[10]，其后缀“128”和“256”表示这两个函数通常支持的安全强度。这类函数是

各种信息安全应用的重要组成部分，通常被使用在数字签名生成与验证、密钥推导、伪随机位生成等领域。Keccak 算法中有两个重要参数，分别为比特率 r 和容量 c，Keccak-p 置换中内部状态的长度称为 b，且 $b=r+c$。

在 Dilithium 算法的密钥生成、签名生成、签名验证 3 个阶段中都使用到了 SHAKE256 和 SHAKE128 算法，但在这 3 个不同阶段内，所使用的 SHAKE 算法不同，且 SHAKE 算法所要执行的功能也不同。在设计时，使用 Keccak 核来实现 SHAKE 算法需要进行适当的填充和截断操作。SHAKE 算法的具体运算步骤如表 1 所列。

表 1 SHAKE 算法计算流程

Table 1 Calculation flow of SHAKE algorithm

步骤	操作
步骤 1	对输入信息用添加一个二进制 1，接着添加若干个二进制 0，使得填充后的信息长度是比特率(bitrate) r 的整数倍
步骤 2	SHAKE 算法使用 keccak 核的海绵结构进行哈希计算。该结构定义了一个固定大小的状态和固定输入长度 r。将要进行迭代运算的信息分为大小相等的块，块的大小等于 r。每一块输入后，会执行以下两个步骤： 1)将当前块与状态进行异或操作； 2)对状态进行一定的置换操作。 重复执行以上两个步骤直到所有块都输入完成，每块被处理的次数等于轮数 24。在所有块被处理之后，每块的输出作为下一块的输入。得到最后的状态值
步骤 3	将最后一轮块运算的结果拼接，形成输出哈希

SHAKE 算法的核心运算模块是迭代哈希运算操作，其整体结构是由、、、χ、五大运算组成。在这五大模块中，输入是被称为状态的三维数组(5×5×64)，它依次经过、、、、运算完成对输入数的填充和减缩，经过 24 轮循环之后得到最终的结果。其步骤如下所示。

$$a[x][y][z]=a[x][y][z]\oplus\sum_{y=0}^{4}a[x-1][y][z]\oplus\sum_{y=0}^{4}a[x+1][y][z-1] \tag{2}$$

五大运算构成了图 9 的 Keccak_round 模块，SHAKE128 和 SHAKE256 算法只是在初始填充和输出 bit 上有所区别，在这五大运算上却是一致的。

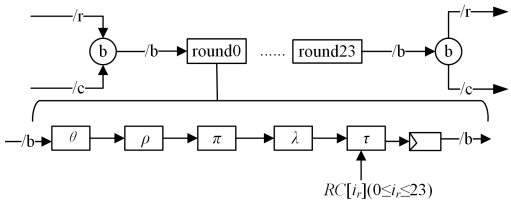

图 9 Keccak 整体结构

Fig. 9 Overall structure of Keccak

运算电路图如图 10 所示。运算主要是在三维矩阵上进行线性的异或运算，其作用主要是使得整个算法具有很好的扩展性。如果没有运算，压缩函数将不会得到良好的扩散效果[11]。运算的主要作用就是在每轮运算的最后加上一个轮常数来破坏原有三维数组结构的对称性。相比其他 4 个预算，其实现比较简单。

SHAKE 算法在 Dilithium 算法中用于获得可变长度的随机字节流，从而创建该算法所需的随机性，在该内部架构中，总体采用流水线技术实现。初始化输入使用上一阶段的输出的 64bit 数据作为种子输入进行 Keccak 运算。在本文设计的 Keccak 结构中，首先对输入数据进行 bit 填充，在设计该硬件架构时，对 Keccak 的 24 轮循环全部展开进行设计。对输入数的 bit 填充需要在特定 mode 下才能进行，mode 由输入数的第 62 位和 61 决定。0 表示 SHAKE128 填充，2 表示 SHAKE256 填充。填充数为 0×1f 和 0×06。填充完毕后，为了增加随机性，对 dout_output 进行逆位置变换后进行与运算，得到 64 bit 结果。在 Keccak_padded 模块中将其扩展到 SHAKE128 对应的 1 444 bit 填充位，SHAKE256 对应的 1 084 bit 填充位。在后续操作中，将 1 600 bit 的 to_round 输入图 10 中的 Keccak_round 模块中进行运算。循环往复24 轮，迭代结束。其中 rd_ctr 表示该 24 轮循环次数，此过程中，将 Keccak 运算得到的 to_round 的 0 到 1 084 bit 和到 1 343 bit 位拆分为对应的 64bit 向量，根据上述流程依次按照指定的 mode_ctrl 输出 64 bit 的随机数种子。当 mode_ctrl 为 0 时输出 SHAKE128 结果，为 2 时输出 SHAKE256 结果。

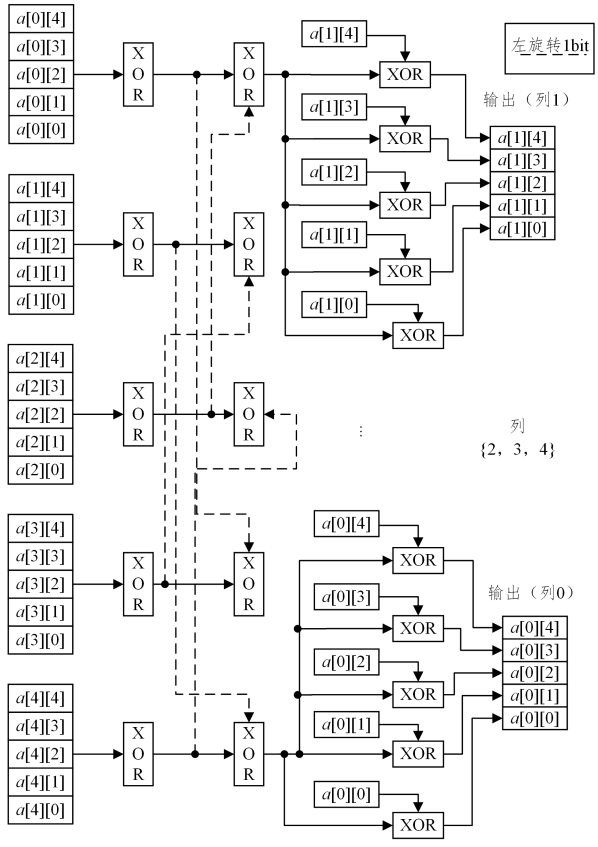

图 10 运算电路图

Fig. 10 Operation circuit diagram

无论是 SHAKE256 算法还是 SHAKE128 算法，其核心架构都是由 Keccak 算法组成。设计了一种低延迟可扩展的 Keccak 架构，使得该架构支持 SHAKE128 和 SHAKE256 算法，在密钥生成、签名生成、签名验证这 3 个阶段内，根据输入信号的不同对该架构进行控制，使得其执行不同的 Keccak 算法。

本文通过设计图 11 所示的 Kecca 计算流程，将 SHAKE128 和 SHAKE256 的运算模块全部在一个架构中实现，免除了 Dilithlium 算法 3 个阶段需设置 3 个不同的结构来匹配，从而降低了时间延迟。同时该架构易于扩展，使得其可

以单独执行 Keccak 系列算法。

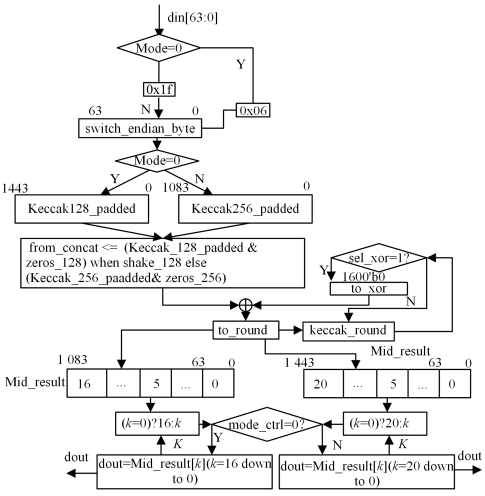

图 11　可扩展 Keccak 计算流程

Fig. 11　Scalable Keccak calculation flow

4　结构设计

4.1　主要架构设计

密钥生成、签名生成、验签模块结构相似，其中主要包括初始化模块、控制模块和多项式运算单元。图 12 给出了这三大模块的逻辑整体结构。其中多项式运算模块封装了 NTT、INTT、PWM、系数相乘、系数相加和系数相减等功能。首先，初始化模块由 FIFO(First In First Out)接收数据，并写入 RAM。然后，根据相应的算法流程，执行运算，最后将计算结果经 FIFO 返回。

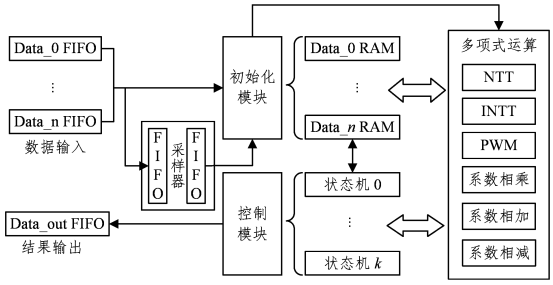

图 12　三大运算逻辑结构

Fig. 12　Logical structure of three operations

签名生成模块被拆分成了两部分并行计算，如图 13 所示，一部分只需计算一次的初始内容通过预计算产生；其余为拒绝循环 $\hat{c}$，h，z 等部分的计算，这两部分整体结构如图 14 所示。

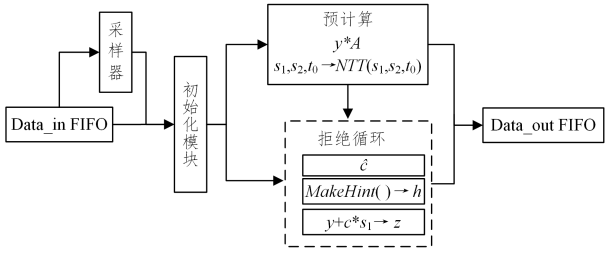

图 13　签名并行计算

Fig. 13　Signature parallel computation

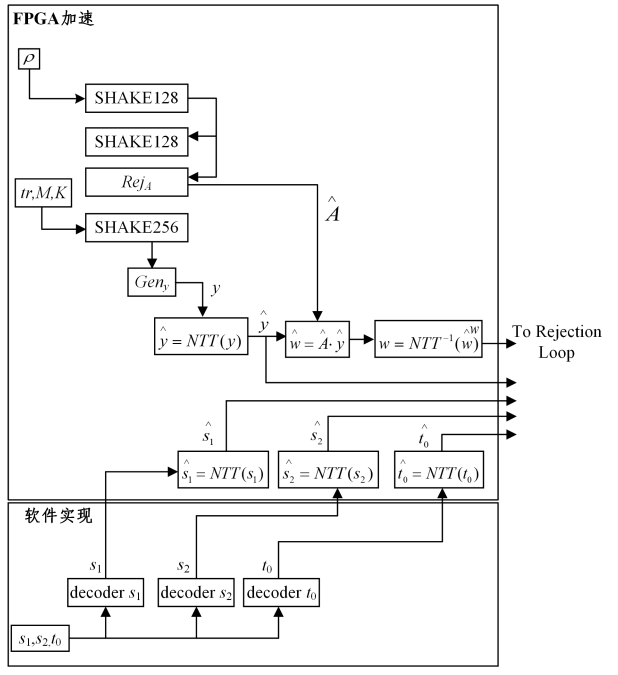

图 14　Dilithium 预计算签名

Fig. 14　Dilithium pre-computational signature

4.2　软硬件结合

为了平衡模块之间的执行速度，降低模块之间的路径延迟，匹配不同模块间的计算速度，我们对密钥生成、签名、验签这 3 个模块采用软硬件结合的方式进行实现。以签名模块的预计算为例，具体流程如下。

签名预计算阶段，Decoder 由软件完成，向量 $\boldsymbol{y}$ 与矩阵 $\boldsymbol{A}$ 的采样生成以及 $\boldsymbol{y}*\boldsymbol{A}$ 由 FPGA 加速完成。硬件实现中由采样器生成矩阵 $\hat{\boldsymbol{A}}$，同时采样生成向量 $\boldsymbol{y}$，之后 $\boldsymbol{y}$ 做 NTT 运算使向量 $\boldsymbol{y}$ 转为 NTT 域表示。当 $\boldsymbol{y}$ 做完 NTT 运算后，直接与 $\hat{\boldsymbol{A}}$ 相乘得到 $\hat{w}$，最后再做 INTT 变换得到 w。另一个模块并行计算 s_1，s_2 和 t_0 的 NTT 变换，并完成后续计算。预计算签名的软硬件结合实现如图 14 所示。

4.3　采样器

采样模块如图 15 所示，主要包括控制模块、SHAKE128/256 Keccak 运算模块。其中 Keccak 运算模块共有 3 个，前 2 个用于计算矩阵 $\boldsymbol{A}$，最后 1 个用于计算向量 $\boldsymbol{s}_1$，$\boldsymbol{s}_2$，$\boldsymbol{y}$ 和 $\boldsymbol{c}$。Keccak 模块单个时钟计算 1 轮运算，共 24 轮，并配合外围控制进行输入数据填充、结果输出，最后，将计算结果经 FIFO 返回。

如果使用多个 Keccak 核，则允许对多项式进行并行采样，Dilithium 需要大量伪随机数据来执行多项式采样。为提高离散高斯采样和二项式分布采样效率，实现高性能设计，使用 3 个 Keccak 核进行计算，对多项式矩阵 $\boldsymbol{A}$ 的生成，使用 2 个 Keccak 核并行计算，并对采样的随机系数进行并行判断；对于剩余的散列和采样工作，使用 1 个 Keccak 核进行计算。此外，我们还为向量和矩阵添加了多条拒绝通道，以并行处理伪随机数据。拒绝通道的数量为最大限度地利用相应的 Keccak 核最小值。例如，对矩阵 $\boldsymbol{A}$ 的一个系数进行采样需要 24 位，而 Keccak 每周期能够产生 64 位。因此，使用了 3 个拒绝通道，它们的总吞吐量(72 bits/cc)高于 Keccak 核心的

吞吐量。这可以防止 Keccak 核在采样过程中失速。每个时钟周期处理 2 或 3 个系数，允许 Keccak 内核以最大吞吐量运行。$\boldsymbol{y}$ 向量也采用了类似的方法。这种方法的唯一例外是多项式 c，必须使用 Fisher-Y 随机抽样。

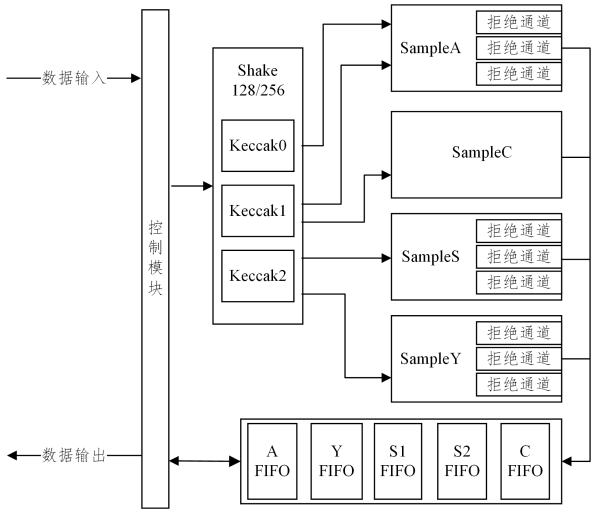

图 15 采样器整体结构

Fig. 15 Overall structure of sampler

4.4 松耦合架构

本文所述的 Dilithium 优化技术中采用了松耦合的架构，将整个程序优化为四大模块：密钥生成、签名模块、验签模块、采样器模块。各模块间数据经 FIFO 传输，并利用 FIFO 使各模块计算频率相匹配，最终结果由控制模块仲裁输出。各模块功能职责清晰，对任一模块的优化不会影响其他模块的运行，该结构便于将来进一步的优化。松耦合架构如图 16 所示。

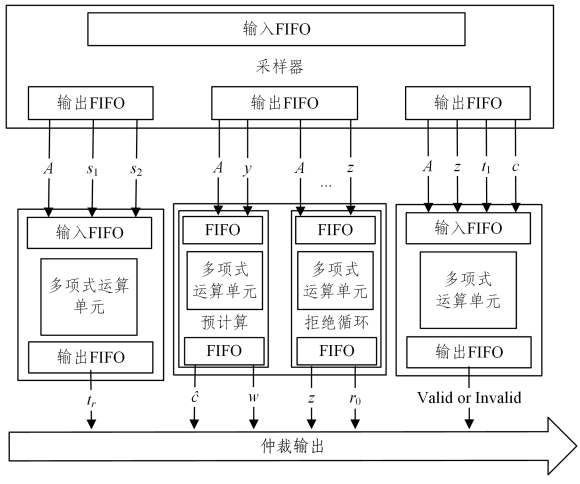

图 16 松耦合架构

Fig. 16 Loosely coupled architecture

5 实验结果与分析

本文实验的硬件平台为 FPGA 加速卡，芯片型号为 Xilinx 公司的 xcku060，其查找表 LUT 资源是 331680，FlipFlops 寄存器资源是 663360。软件平台为集设计、仿真、综合、布线、生成于一体的 Vivado2019.2 软件。

5.1 各模块资源

Dilithium 密钥生成模块、签名模块、采样模块、验签模块以及 Dilithium 的总体消耗资源如表 2 所列。

表 2 模块资源占用

Table 2 Module resource usage

模块	LUTs	FFs	DSPs	BRAMs	频率/MHz
SHAKE	5942	4458	—	—	275
NTT	4398	2827	12	6.0	275
密钥生成模块	5318	4635	12	19.5	250
签名模块	13474	11438	24	51.0	250
采样模块	20918	15868	—	—	—
验签模块	7245	5605	12	27.0	250
Dilithium	46979	36868	48	97.5	225

表 2 列出了 Dilithium 各个模块的资源消耗，签名模块由预计算签名和拒绝循环签名两个阶段组成，它是 Dilithium 硬件实现中最复杂、消耗最大的操作。本文将预计算签名与拒绝循环签名两阶段并行计算，减少了该模块所需的运行时间和硬件消耗。在采样模块中，主要是对 Keccak 模块执行结果进行多重采样，涉及了多个 Keccak 计算。导致该模块消耗的资源在 Dilithium 总体中占比最多。程序整体实现所占芯片 LUT 资源为 14.2%，FF 所占资源为 5.7%。本文所实现的 Dilithium 总体频率为 225MHz，同时本文的密钥生成、签名和验签阶段的时钟分别为 5.3k，18k，8k。

5.2 Keccak 核对比

为展现方案中设计的 Keccak 核优势，在表 3 中，与其他文献中实现的 Keccak 算法进行对比。

表 3 Keccak 模块性能对比

Table 3 Performance comparison of Keccak modules

模块	设备	LUTs	FF	频率/MHz	适用情况
文献[12]	Virtex-5	5216	3912	312.98	SHAKE256
文献[13]	Artix-7	6322	6993	229.00	SHAKE512
文献[14]	Artix-7	6841	4279	250.00	SHAKE128/256
本文 Keccak	xcku060	4660	4356	290.00	SHAKE128/256
本文 SHAKE	xcku060	5942	4458	275.00	—

文献[12]使用 Virtex-5 实现 SHA-3，提出了一种新型 Keccak 架构，如添加预处理单元、不使用软件辅助或接口等，并评估了它们对吞吐量和加速比的影响；优化后的 SHA-3 在 Virtex-5 上可以达到 7.511Gbps 的吞吐量。文献[13]使用高级综合实现并优化了 post-quantum cryptography(PQC)原语。通过设计安全原语以达到面积优化、速度优化 Keccak，并且可以抵御侧信道攻击。文献[14]介绍了一种新的面积-时间高效的硬件架构，在 Artix-7 基于格的 CRYSTALS-Kyber，该架构针对 Kyber-768 进行了实现，并实现了算法中使用的所有 SHA-3 原语。该架构可以在 FPGA 上运行 Keccak 算法，并具有较低的资源占用和较高的吞吐量。本文通过优化整体结构、增加算法扩展性等方法，占用资源消耗低于文献[12-14]，时钟周期更短，且适用于 SHAKE128/256，扩展性更好。因此可以看出，本文优化后的 Keccak 模块在资源消耗、扩展性上具有优势。

5.3 方案对比

为了进一步反映方案性能，在表 4 中，与其他 Dilithium 算法经过 FPGA 优化后的资源消耗、频率进行了综合对比。其中，RE 代表资源占用；PR(Performance-to-Resource Ratio)代表性能资源比；PR 客观反映了资源消耗与性能之间的关系，PR 值越大，说明方案在资源消耗和性能之间平衡得越好。资源占用和性能资源比的计算式如下。

$$RE = LUTs \times 2 + FF \tag{3}$$

$$PR = \frac{\text{频率} \times 10^4}{RE} \tag{4}$$

表 4　FPGA 优化 Dilithium 方案综合对比

Table 4　Comprehensive comparison of FPGA optimization Dilithium schemes

方案	设备	LUTs	FF	DSPs	BRAM	频率/MHz	时钟周期	PR
文献[3]	Artix-7	53187	28318	16	29.0	116.0	53037	8.6
文献[4]	Artix-7	44653	13814	45	31.0	140.0	50982	13.6
文献[5]	Virtex 7 U+	68461	86295	965	145.0	300.0	54258	14.0
文献[6]	Artix-7	29998	10366	11	10.0	96.9	66800	13.7
本文	xcku060	46979	36868	48	97.5	225.0	35034	17.1

在表 4 中，文献[3]的设计各项资源占用较少。其模约减算法未利用模数特性，因此包含了更长进位链，相较于本文，运算频率更低。文献[4]中 NTT 具有很高的频率，但也会导致非常高的 DSPs 使用率，另外其高频 NTT 核心因为只能运行在一个较低的频率，所以并不能提升整体性能。相较于本文，NTT 核心中 DPSs 数量减少了 30%，整体频率提高了 60%。文献[5]增加了过多的资源消耗，资源消耗远高于本文的方案。文献[6]由于仅支持 NTT/INTT，因此需要额外的 DSPs 对应点乘操作。其硬件资源消耗较少，但关键路径过长会导致最大时钟频率偏低。与之相比，本文方案性能资源比提升了 30%。与其他 FPGA 优化后的 Dilithium 方案在资源消耗和频率方面进行了综合对比，可以看出，本文利用 KOA 算法、快速模约减算法、流水线并行加速蝶形运算单元和多 RAM 访存优化算法降低了硬件复杂度，提高了计算效率，优化后的 Dilithium 在资源消耗和频率上有着更好的表现。同时，本文方案中的多项式运算模块可通过参数配置将应用场景扩展到其他格密码算法，例如 Kyber，Saber。

结束语　后量子密码得到了各方的重点关注，特别是格基密码算法。格基密码算法是未来后量子密码标准的主流技术路线，关键是多项式环上的元素乘法，NTT 有效加速多项式计算。蝶形运算是 NTT 中的核心操作，针对蝶形运算，设计一种高吞吐、低延迟的流水线结构。采用新型系数访存模式，减少资源消耗。针对性采用适用于 Dilithium 算法模值的模约减硬件单元，减少乘法器消耗。结合流水线型 KOA 进一步优化乘法运算，提高运算效率。设计了一种低延迟可扩展的 Keccak 硬件架构支持采样工作，提高了资源利用率。本文实现了 Dilithium 在资源消耗、算法效率、执行时间上的平衡，在实现了较高计算效率的同时兼顾了低延时和低功耗的实际应用需求。

最后，实验结果分析表明，本文设计实现的密钥生成、签名、验证签名 3 个模块单独执行的频率达到了 250 MHz，合并执行的工作频率为 225 MHz，在指定的时钟下实现了良好的实验结果，具有良好的性能。

未来工作将对模乘、模加减、NTT 进行扩展性设计，使其能够满足不同 bit 运算。对于后量子密码的侧信道攻击，可采用掩码技术进行防御。

参 考 文 献

[1] CHEN L，CHEN L，JORDAN S，et al. Report on post-quantum cryptography[M]. Gaithersburg，MD，USA：US Department of Commerce，National Institute of Standards and Technology，2016.

[2] DANG V B，FARAHMAND F，ANDRZEJCZAK M，et al. Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches[J]. IACR Cryptol EPrint Arch，2020，2020(795)：1-86.

[3] LAND G，SASDRICH P，GÜNEYSU T. A hard crystal-implementing dilithium on reconfigurable hardware[C]// Smart Card Research and Advanced Applications. 2022：210-230.

[4] MERT A C，JACQUEMIN D，DAS A，et al. A Unified Cryptoprocessor for Lattice-based Signature and Key-exchange[J]. IEEE Transactions on Computers，2022，14(8)：1-13.

[5] RICCI S，MALINA L，JEDLICKA P，et al. Implementing crystals-dilithium signature scheme on fpgas[C]// The 16th International Conference on Availability，Reliability and Security. 2021：1-11.

[6] ZHAO C，ZHANG N，WANG H，et al. A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium[J]. IACR Trans. Cryptogr. Hardw. Embed. Syst.，2022，2022(1)：270-295.

[7] BECKER H，HWANG V，KANNWISCHER M J，et al. Neon ntt：Faster dilithium，kyber，and saber on cortex-a72 and apple m1[J/OL]. Cryptology ePrint Archive，2021. https://eprint.iacr.org/2021/986.

[8] SONI D，BASU K，NABEEL M，et al. CRYSTALS－dilithium[M]// Hardware Architectures for Post-Quantum Digital Signature Schemes. 2021：13-30.

[9] 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，2019，2019(4)：17-61.

[10] DWORKIN M J. SHA-3 standard：Permutation-based hash and extendable-output functions[S]. National Institute of Standards and Technology，Gaithersburg. 2015.

[11] ASSAD F，ELOTMANI F，FETTACH M，et al. An optimal hardware implementation of the KECCAK hash function on virtex-5 FPGA[C]// 2019 International Conference on Systems of Collaboration Big Data，Internet of Things & Security(SysCoBIoTS). IEEE，2019：1-5.

[12] SONI D，KARRI R. Efficient hardware implementation of PQC primitives and pqc algorithms using high-level synthesis[C]// 2021 IEEE Computer Society Annual Symposium on VLSI(ISVLSI). IEEE，2021：296-301.

[13] DOLMETA A. Hardware architecture for CRYSTALS-Kyber cryptographic primitives[D]. Politecnico di Torino，2022.

[14] BECKWITH L，NGUYEN D T，GAJ K. High-Performance Hardware Implementation of CRYSTALS-Dilithium[C]// 2021 International Conference on Field-Programmable Technology (ICFPT). IEEE，2021：1-10.

YAN Yunfei，born in 1999，postgraduate. His main research interests include post-quantum cryptography and high-performance computing.

LI Bin，born in 1986，Ph.D，lecturer. His main research interests include high-performance computing and information security.