

紧凑的 Aigis-sig 数字签名方案软硬件协同实现方法

周朕^{1,2}, 何德彪^{1,2}, 罗敏^{1,2}, 李莉^{1,2}

(1. 空天信息安全与可信计算教育部重点实验室, 湖北 武汉 430072;

2. 武汉大学国家网络安全学院, 湖北 武汉 430072)

摘要: 基于理想格构造的 Aigis-sig 数字签名方案具有实现效率高、签名长度短、抗量子攻击等优势。针对 Aigis-sig 方案, 构造了一种改进的模乘计算元件, 设计了一种基于快速数论变换 (NTT) 算法实现环上多项式运算的紧凑硬件架构; 同时以此架构为基础, 提出了 Aigis-sig 数字签名方案的 FPGA 软硬件协同实现方法。实验表明, 在 Xilinx Zynq-7000 SoC 平台上, CPU 频率和硬件频率分别设置为 666.66 MHz 和 150 MHz 时, 该实现方案相较于纯软件实现, 签名阶段和验签阶段分别取得约 26% 和 17% 的性能提升。

关键词: 后量子密码; 数字签名; Aigis-sig; 现场可编程逻辑门阵列; 快速数论变换

中图分类号: TN918

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2021026

Compact software/hardware co-design and implementation method of Aigis-sig digital signature scheme

ZHOU Zhen^{1,2}, HE Debiao^{1,2}, LUO Min^{1,2}, LI Li^{1,2}

1. Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Wuhan 430072, China

2. China School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China

Abstract: Digital signature scheme Aigis-sig, constructed on ideal lattice, takes the advantages of high efficiency, short signature length and resistant to quantum attack, etc. An optimized modular multiplication arithmetic component was constructed and a compact hardware architecture for polynomial operation over a ring based on number theory transformation (NTT) algorithm for Aigis-sig was designed. Besides, based on this architecture, software/hardware co-design and implementation for Aigis-sig scheme on FPGA platform in cryptography was proposed. Experimental results show that the speed of signature phase and verification phase are increased by about 26% and 17% respectively, compared with the pure software implementation on Xilinx Zynq-7000 SoC platform when CPU clock frequency and hardware clock frequency are set as 666.66MHz and 150 MHz respectively.

Keywords: post-quantum cryptography, digital signature, Aigis-sig, FPGA, number theory transformation

收稿日期: 2020-11-14; 修回日期: 2021-01-09

通信作者: 何德彪, hedebiao@163.com

基金项目: 国家自然科学基金 (61972294, 61932016)

Foundation Item: The National Natural Science Foundation of China (61972294, 61932016)

论文引用格式: 周朕, 何德彪, 罗敏, 等. 紧凑的 Aigis-sig 数字签名方案软硬件协同实现方法[J]. 网络与信息安全学报, 2021, 7(2): 64-76.

ZHOU Z, HE D B, LUO M, et al. Compact software/hardware co-design and implementation method of Aigis-sig digital signature scheme[J]. Chinese Journal of Network and Information Security, 2021, 7(2): 64-76.

1 引言

随着量子计算技术的不断发展与成熟, 量子研究学者对可实用量子计算机的研制不断取得突破性的进展^[1]。量子计算机在带给人们强大算力的同时, 也对基于大整数分解、离散对数问题的传统公钥密码体系构成了一定的威胁。Shor 等^[2]提出了可以在多项式时间内破解大整数分解问题和离散对数问题的量子算法, 一旦足够成熟的量子计算机研制成功并投入使用, 传统的密码体系将不再能提供安全性保障。因此, 学者对新兴的后量子密码产生了浓厚兴趣, 并积极研究如何构造高效安全可实用的后量子密码方案。

作为公认可以抵抗量子计算机攻击的一类密码算法——后量子密码的研发是密码学者应对量子计算威胁的防范措施之一^[3]。主流的后量子密码算法主要包括: 格密码、编码密码、哈希密码、多变量密码。2012 年, 美国国家标准技术研究所 (NIST, National Institute of Standards and Technology) 正式启动了后量子密码的研究工作, 并于 2016 年开展全球范围内的后量子密码算法标准的征集和筛选^[4-6]。截至 2020 年 7 月, NIST 已经完成了 3 轮的筛选工作, 并确定了 7 个最终方案和 8 个候选方案, 预计近几年将最终确认新一代公钥密码算法标准。基于格的密码方案是研究热度较高的后量子密码体系之一, 这主要是由于格密码具有实现效率高、密钥长度较短、允许最坏情况困难假设等优点, 除此之外, 基于格的密码方案在同态加密技术领域极具研究价值, 可以真正意义上支持全同态的加密算法均为格密码方案。在 NIST 后量子密码标准化进程第 3 轮的 7 个最终方案中, 有 5 个方案都属于格密码体系。2019 年, 中国密码学会举办了全国密码算法设计竞赛^[7-8], 公钥密码组的参赛方案中包含大量的后量子密码方案, 其中基于格的参赛密码方案占据了一定的比例, 如 Aigis-sig/enc、LAC.PKE/KEX、AKCN、Fatseal 等^[7]。

在基于格的密码体系中, 错误学习问题 (LWE, learning with error)^[9]和小整数解问题 (SIS, short integer solution)^[10]是密码学者构造方

案时经常涉及和使用的数学困难问题。除此之外, 两类困难问题在环上 (RLWE/RSIS, ring-LWE/ring-SIS) 和模格上 (MLWE/MSIS, module-LWE/module-SIS) 的版本也受到密码学者们的广泛关注, 其主要原因是理想格具备一些标准格所不具备的额外代数结构和代数特性, 更准确地说, 以特定多项式环上的多项式乘法代替大维数的矩阵或向量乘法, 可以使环或模格上所构造的密码方案能够通过一定的算法技巧实现极大程度的性能改进^[11]。在 2019 年全国密码算法设计竞赛的众多方案中, 如 Aigis-sig/enc, LAC.PKE/KEX, AKCN 等获得优秀名次的参赛方案都是在环或模格结构基础之上所提出的格密码方案^[8]。然而, 关于这些优秀方案在不同平台上的高性能实现方面的研究工作非常少。

本文的主要研究目标在于针对 Aigis-sig 方案^[12]提出 FPGA 平台上紧凑高效的软硬件协同实现方法, 该实现方案中的核心工作是构造出一种针对 Aigis-sig 方案参数选择的多项式运算硬件实现架构。实验表明, CPU 频率和硬件频率分别设置为 666.66 MHz 和 150 MHz 时, 本文的实现方法相较于同 FPGA 平台下的纯软件实现在签名和验签方面分别有 26% 和 17% 的提升。

2 研究现状

基于格的密码体系是近几年发展较为迅速、理论较为成熟的后量子密码体系之一, 同时, 针对格密码方案的安全快速实现也是后量子密码研究人员的关注点之一^[13]。对于理想格上密码方案的高性能实现, 研究者的关注点主要集中于多项式乘法的快速实现、特定噪声分布的安全实现、高速哈希函数实现等方面。其中针对多项式乘法, 在特定环上可以采用快速数论变换 (NTT, number theoretic transform) 方法从算法层面上大幅提升性能。

在软件实现方面, 2015 年, De C 等^[14]提出了一种 ARM Cortex-M4F 微处理器平台上基于 RLWE 公钥加密方案的快速软件实现方法; 2016 年, Yuan 等^[15]提出了一种格密码方案的可移植 JavaScript 实现, 该实现方法可以应用于网页浏览器、安卓系统; 同年, Stebila 等^[16]设计了一个名

为“开源量子安全”的软件平台，该平台不仅内置了许多格密码方案的 C 语言实现，同时还具备将软件实现整合到实际产品中的功能。

在开源库方面，NFLlib 是针对理想格上多项式运算快速实现的函数库之一^[17]，该库基于 C++ 语言开发，并且还在不断地完善；除此之外，微软公司研发了针对格上 BFV 和 CKKS 全同态加密方案的函数库 SEAL^[18]，同样基于 C++ 语言。

在硬件实现方面，Göttert 等^[19]于 2012 年提出了 Virtex-7 平台上 RLWE 加密方案的硬件实现，该硬件架构实现了相当大的数据吞吐量，其运算可以达到数百倍于相应软件实现的运算速度，同时占用了较大的面积资源；2014 年，Pöppelmann 等^[20]在 Spartan-6 和 Virtex-5 系列平台上设计了一种轻量级格密码硬件实现引擎，该架构的面积资源消耗非常低，因此在资源较少的嵌入式平台上有很强应用价值；针对 NTT 算法，Roy 等^[21]在 2014 年设计了一个非常紧凑的硬件协处理器，该工作对后续许多针对 NTT 的硬件实现均具有启发性的指导作用。2019 年，Basu 等^[22]针对 NIST 第二轮候选方案中的大量后量子方案进行了基于高层次综合（HLS）方法的硬件实现测试，并提供了性能对比结果；同年，Dang 等^[23]针对提交到 NIST 的 FrodoKEM、Round5、Saber 3 个密钥封装方案提出了软硬件协同实现方法，与纯软件实现相比，性能大幅提升；Banerjee 等^[24]设计了针对许多格密码方案的可配置协处理器 Sapphire，该协处理器可以高效完成多项式乘法运算与噪声采样任务；2020 年，Xin 等^[25]提供了特定域上的向量化处理器，可以支持并行 NTT 算法和向量式的噪声采样，其性能在 Sapphire 的基础上提高了一个数量级。

3 背景知识

本节将对 Aigis-sig 中涉及的相关数学定义、重要算法以及签名方案流程等进行简单介绍。

3.1 符号定义

文中主要使用的数字符号及其定义如下。

$\parallel$ ：比特拼接操作。

$\gg$ ：右移操作。

$\ll$ ：左移操作。

$\bmod$ ：取模运算。

Z_q ：由 $\{0, 1, 2, \dots, q-1\}$ 以及 $\bmod$ 意义下的加法、乘法所构成的环。

R_q ：多项式环 $Z_q[x]/f(x)$ ，其中 $f(x) = x^n + 1$ 。

R_q^k ： $\{(a_0, a_1, \dots, a_{k-1}) \mid a_m \in R_q, m = (0, 1, 2, \dots, k-1)\}$ 。

$R_q^{k \times l}$ ： $\{(\beta_0, \beta_1, \dots, \beta_{l-1}) \mid \beta_m \in R_q^k, m = (0, 1, 2, \dots, l-1)\}$ 。

a ：无特别说明时，表示环 R_q 中的多项式元素所构成的列向量。

A ：无特别说明时，表示环 R_q 中的多项式元素所构成的矩阵。

$\|\cdot\|$ ：范数运算。

S_η ：由 R_q 中所有系数属于 $\{-\eta, -\eta+1, \dots, -1, 0, 1, \dots, \eta\}$ 的全体多项式元素组成的集合。

3.2 多项式环上的加法与乘法

Aigis-sig 等在环或模格上构造的密码方案中，多项式环 R_q 上元素的加法和乘法在整个方案的总计算量中占有很大的比重。 R_q 上的加法与乘法可以按照如下方式定义。

定义 1 R_q 上的加法

给定 R_q 上的元素 $a(x) = \sum_{i=0}^{n-1} a_i x^i$ ， $b(x) = \sum_{i=0}^{n-1} b_i x^i$ ，

若运算“+”满足 $c(x) = a(x) + b(x) = \sum_{i=0}^{n-1} c_i x^i$ ，其中 $c_i = a_i + b_i$ ， $i = (0, 1, 2, \dots, n-1)$ ，则称运算“+”为 R_q 上的加法。

定义 2 R_q 上的乘法

给定 R_q 上的元素 $a(x) = \sum_{i=0}^{n-1} a_i x^i$ ， $b(x) = \sum_{i=0}^{n-1} b_i x^i$ ，

若运算“ $\times$ ”满足 $d(x) = a(x) \times b(x) = \sum_{i=0}^{n-1} d_i x^i$ ，其

中 $d_i = \sum_{j=0}^i a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}$ ， $i = (0, 1, 2, \dots, n-1)$ ，

则称运算“ $\times$ ”为 R_q 上的乘法。

3.3 快速数论变换

多项式环上的快速乘法实现是格密码实现研

究者关注的热点之一。按照 3.2 节中乘法定义的方式，通过对两个待乘多项式的系数进行卷积计算，其复杂度为 $O(n^2)$ 。然而由于环 R_q 中多项式 $f(x) = x^n + 1$ 在 Z_q 上的根具有优良的数学性质，因而当参数 n 与 q 满足 $2n|(q-1)$ 时，可以采用快速数论变换方法以 $O(n \log n)$ 的复杂度完成 R_q 上的多项式乘法。以下两个算法为两种常用的在线生成旋转因子的正向、反向 NTT 算法，同样也是本文中所使用的算法。

算法 1 正向 NTT 算法

输入 R_q 上一多项式 $g(x)$ 的系数 $(g_0, g_1, \dots, g_{n-1})$ ， Z_q 上的 $2n$ 次单位根 w_{2n}

输出 $g(x)$ 的 NTT 表达形式 $(G_0, G_1, \dots, G_{n-1})$

- 1) $(G_0, G_1, \dots, G_{n-1}) \xleftarrow{\text{比特翻转}} (g_0, g_1, \dots, g_{n-1})$
- 2) for $m=2$ to $n, m=2m$
- 3) $w_m = w_n^{n/m}, w = w_{2n}^{n/m}$
- 4) for $j=0$ to $m/2-1, j=j+1$
- 5) for $k=0$ to $n-1, k=k+m$
- 6) $x = G_{k+j}, y = wG_{k+j+m/2}$
- 7) $G_{k+j} = x + y, G_{k+j+m/2} = x - y$
- 8) end for
- 9) $w = w \times w_m$
- 10) end for
- 11) end for

算法 2 反向 NTT 算法

输入 R_q 上一多项式 $g(x)$ 的 NTT 表达形式 $(G_0, G_1, \dots, G_{n-1})$ ， Z_q 上的 $2n$ 次单位根 w_{2n}^{-1}

输出 $g(x)$ 的系数 $(g_0, g_1, \dots, g_{n-1})$

- 1) for $m=n$ to $2, m=m/2$
- 2) $w_m = w_n^{-n/m}, w = w_{2n}^{-n/m}$
- 3) for $j=0$ to $m/2-1, j=j+1$
- 4) for $k=0$ to $n-1, k=k+m$
- 5) $x = G_{k+j} + G_{k+j+m/2}, y = G_{k+j} - G_{k+j+m/2}$
- 6) $G_{k+j} = x, G_{k+j+m/2} = w * y$
- 7) end for
- 8) $w = w \times w_m$
- 9) end for
- 10) end for
- 11) $(g_0, g_1, \dots, g_{n-1}) \xleftarrow{\text{比特翻转}} (n^{-1}G_0, n^{-1}G_1, \dots,$

$n^{-1}G_{n-1})$

3.4 Aigis-sig 数字签名方案

本文中所实现的方案 Aigis-sig 为 Schnorr 签名在格密码上的延伸。本文主要研究 Aigis-sig 方案在推荐参数组 PARAMS-II 下的实现，下面给出 Aigis-sig 方案简略的流程，更详细的方案介绍可参考官方设计文档^[12]。参数组 PARAMS-II 的部分参数如表 1 所示。

表 1 参数组 PARAMS-II 的部分参数
Table 1 Partial parameters of PARAMS-II

(n, q)	(k, l)	(η_1, η_2)	公钥长度 /byte	私钥长度 /byte	签名长度 /byte
256, 3 870 721	5, 4	2, 5	1 312	3 376	2 445

密钥生成算法如下所示。

算法 3 密钥生成算法

输出 公钥 pk, 私钥 sk

- 1) 选取秘密向量 $s_1 \in S_{\eta_1}^l, s_2 \in S_{\eta_2}^k$
- 2) 选取随机矩阵 $A \in R_q^{k \times l}$
- 3) 计算 $t = As_1 + s_2$
- 4) 将 t 分拆成高比特段 t_1 和低比特段 t_0
- 5) $pk \leftarrow (A, t_1), sk \leftarrow (A, s_1, s_2, t_0)$

算法 4 签名算法

输入 私钥 sk, 消息 M

输出 签名 σ

- 1) $(A, s_1, s_2, t_0) \leftarrow sk$
- 2) 选取随机向量 $y \in R_q^l$ ，计算 $w = Ay$
- 3) 获取 w 的高比特段 $w_1, c = \text{Hash}(w_1), c \in R_q$
- 4) 计算 $z = y + cs_1, u = w - cs_2$
- 5) 将 u 分解为高位段 r_1 和低位段 r_0
- 6) 若 $\|z\|, \|r_0\|$ 过大，回到步骤 2)
- 7) 计算 $v = ct_0$ ，根据 u, v 计算提示位多项式 h
- 8) 若 $\|v\|, \|h\|$ 过大，回到步骤 2)
- 9) $\sigma \leftarrow (z, h, c)$

验签算法如下所示。

算法 5 验签算法

输入 公钥 pk, 消息 M , 待验证签名 σ

输出 签名验证结果 {合法, 不合法}

- 1) $(A, t_1) \leftarrow pk, (z, h, c) \leftarrow \sigma$

- 2) 计算 $u' = Az - ct_1$
- 3) 利用 u' 和提示位 h 计算 w_1'
- 4) $c' = \text{Hash}(w_1')$
- 5) 若 $\|z\|, \|h\|$ 过大, 则输出签名不合法
- 6) 若 $c' \neq c$, 则输出签名不合法
- 7) 输出签名合法

4 算法设计

本节将对本文所使用的关键算法以及相应的设计优化进行介绍。

4.1 模乘

在 Aigis-sig 方案中, 多项式乘法在整体计算开销中占据了相当大的比重, 而 Z_q 上的模乘运算是多项式乘法中的重要运算单元, 因此, 根据硬件的实现特性, 对模乘运算进行算法层面上的优化对提高整个签名方案的实现性能极具帮助。模乘运算可以分为标准数乘和模约减两个阶段, 在本文中, 标准数乘所对应的比特长度为 22, 利用模数 $q(q=3\ 870\ 721)$ 本身的比特分布特点和数学性质, 设计了一种针对该模数的改进模乘算法。

对于 FPGA 上的硬件实现而言, 数乘是通过调用平台内置 DSP 资源中的乘法器来完成的, 而乘法器的比特长度是固定的, 对于超过乘法器长度的乘法任务可以通过多个乘法器拼接的方式来实现, 但这意味着额外的乘法器用量以及过长的硬件路径。然而通过采用 Karastuba 乘法算法, 不仅可以从算法层面减少数乘所需要的乘法器数量, 而且通过流水线设计, 可以将硬件路径拆分, 从而增大硬件模块所能承受的最大频率。本文通过采取两段式的 Karastuba 算法来实现硬件上的标准数乘, 具体算法如下。

算法 6 标准数乘算法

输入 $a, b \in \{0, 1, 2, \dots, q-1\}$

输出 $t = ab$

- 1) $a_1 = a \gg 11, a_0 = a \bmod 2^{11}$
- 2) $b_1 = b \gg 11, b_0 = b \bmod 2^{11}$
- 3) $t_0 = a_0 b_0, t_2 = a_1 b_1, t_1 = (a_0 + a_1)(b_0 + b_1) - t_2 - t_0$ $// t_1 = a_0 b_1 + a_1 b_0$
- 4) $t_3 = t_2 \parallel t_0$ $// t_3 = t_2 \times 2^{22} + t_0$
- 5) $t = t_3 + (t_1 \ll 11)$

$$// t = t_3 + t_1 \times 2^{11} = (a_1 \times 2^{11} + a_0)(b_1 \times 2^{11} + b_0) = ab$$

通过除法确定商的方式来完成模约减, 运算复杂度是除数长度的立方级, 因此在实现模约减阶段时, 通常采用 Barret、Montgomery 等快速约减算法。这两种算法会涉及与固定常数相乘的运算操作, 对于形式特殊的常数, 甚至可以不通过乘法运算而仅通过加法和移位等简单操作获得正确结果。本文通过分析模数 q 自身的特性, 采用 Montgomery 算法来实现模约减阶段, 如算法 7 所示。

算法 7 Montgomery 约减算法

输入 $t \in \{0, 1, 2, \dots, (q-1) \times (q-1)\}, R = 2^k,$

$q' \in \{0, 1, 2, \dots, R-1\}$, 这里 k 为使

$R > q$ 的正整数, $qq' = -1 \bmod R$

输出 $r = tR^{-1} \bmod q$ $// RR^{-1} - qq' = 1$

1) $m = tq' \bmod R$

2) $r = (t + mq) \gg k$

3) 执行 $r = r - q$, 直至 $r < q$

选择 Montgomery 约减的原因: 在 Montgomery 约减的过程中涉及两次与固定常数相乘的运算, 当选定 $R=2^{22}$ 时, 两个常数分别是 $q=3\ 870\ 721$ 和 $q'=3\ 870\ 719$ 。

$q-1 = q'+1 = 2^{12} \times (2^4 - 1) \times (2^6 - 1)$, 对于第一个因子 2^{12} , 可以直接通过移位操作来实现, 对于后两个因子, 可以通过移位之后做减法来实现, 在与 $3\ 870\ 720$ 相乘结果的基础上加上原数便可以得到与 q 相乘的结果, 减去原数就得到与 q' 相乘的结果, 同时在 $R=2^{22}$ 的条件下, Montgomery 约减算法的第 3 步最多只执行一次。综上所述, 模数 q 上的 Montgomery 约减过程可以不通过使用乘法来实现。

通过 Montgomery 约减方法, 最终得到的结果和正确的约减结果相比多了一个因子 R^{-1} , 一般情况下, 需要与 $R^2 \bmod q$ 相乘之后再进行一次 Montgomery 约减来进行修正。但在 Aigis-sig 方案中, 主要有两处涉及模乘: 第一处是 NTT 算法蝶形运算中操作数与旋转因子进行模乘, 而旋转因子是由预先存储的若干个起始因子通过模乘运算在线生成的, 因此只要在预存储的因子 w 中额外加入一个因子 R , 根据式(1), 所有生成的因子 w 额外多了一个因子 R , 再根据式(2)、式(3), 经

过 Montgomery 模乘之后, 所得结果刚好是正确的结果, 不需要额外的修正; 第二处是 NTT 表示下两个多项式的全体对应系数之间进行逐点模乘, 这时, 得到的积多项式系数全部会多出一个因子 R^{-1} , 因此可以直接在此基础上进行逆向 NTT 算法, 然后与 $n^{-1}R^2 \bmod q$ 进行 Montgomery 模乘, 一次性将多余的因子 n 和 R^{-1} 同时修正。综上所述, Montgomery 算法中产生的额外因子不会对本文方案的实现产生任何额外的运算开销。

$$w_3 = w_2 \times w_1 \Rightarrow w_3 R = w_2 R \times w_1 R \times R^{-1} \quad (1)$$

$$c = a + wb \Rightarrow c = a + wR \times b \times R^{-1} \quad (2)$$

$$d = a - wb \Rightarrow d = a - wR \times b \times R^{-1}$$

$$d = w \times (a - b) \Rightarrow d = wR \times (a - b) \times R^{-1} \quad (3)$$

4.2 多项式乘法

R_q 上的多项式乘法运算是 Aigis-sig 方案中实现成本非常大的重要组件, 采用传统卷积计算的方式来完成多项式乘法操作需要进行 $O(n^2)$ 次 Z_q 上的模乘运算。然而因为商环 $R_q = Z_q / f(x)$ 中的模多项式 $f(x) = x^n + 1$ 以及参数 n, q 均为特殊选取, 使在多项式乘法具体实现时可以通过 NTT 方法极大程度地减少模乘运算的次数。对于硬件实现而言, 架构中模乘元件的数量是固定且有限的, 因此节约模乘操作的次数等价于减少该部分运算所需的时钟周期数, 从而可以提高方案实现的整体时间性能。以下两个算法为本文中设计的正向、反向 NTT 算法, 其中旋转因子序列在线生成, 而不是预先存储, 其目的是减少硬件面积资源。

算法 8 本文设计的正向 NTT 算法

输入 R_q 上一多项式 $g(x)$ 的系数 $(g_0, g_1, \dots, g_{n-1})$, Z_q 上的单位根 $w_2, w_4, w_8, \dots, w_n, w_{2n}$, 其中 w_i 表示 i 次单位根

输出 $g(x)$ 的 NTT 表达形式 $(G_0, G_1, \dots, G_{n-1})$

1) 对 $(g_0, g_1, \dots, g_{n-1})$ 进行比特翻转重排, 并赋值给 $(G_0, G_1, \dots, G_{n-1})$

2) 将系数对 (G_{2i}, G_{2i+1}) 存放在存储位 M_i , $i=(0, 1, 2, \dots, \frac{n}{2}-1)$

3) for $r=2$ to n , $r=2r$

4) $w = w_{2r}$

5) for $s=0$ to $\frac{r}{2}-1$, $s=s+1$

6) for $t=0$ to $n-1$, $t=t+2r$

7) 从存储位 $M_{t/2+s}$ 中读数据赋值给 (a_1, b_1)

8) 从存储位 $M_{t/2+s+r/2}$ 中读数据赋值给 (a_2, b_2)

9) $c_1 = a_1 + wb_1, d_1 = a_1 - wb_1$

10) $c_2 = a_2 + wb_2, d_2 = a_2 - wb_2$

11) 将 (c_1, c_2) 作为新数据存储回 $M_{t/2+s}$

12) 将 (d_1, d_2) 作为新数据存储回 $M_{t/2+s+r/2}$

13) end for

14) $w = w \times w_r$

15) end for

16) end for

算法 9 本文设计的反向 NTT 算法

输入 R_q 上一多项式 $g(x)$ 的 NTT 表达形式

$(G_0, G_1, \dots, G_{n-1})$, Z_q 上的单位根的逆元 $w_2^{-1}, w_4^{-1}, w_8^{-1}, \dots, w_n^{-1}, w_{2n}^{-1}$, 其中 w_i^{-1} 表示 i 次单位根。

输出 $g(x)$ 的系数 $(g_0, g_1, \dots, g_{n-1})$ (未修正因子 n)

1) for $r=n$ to 2 , $r=r/2$

2) $w = w_{2r}^{-1}$

3) for $s=0$ to $r/2-1$, $s=s+1$

4) for $t=0$ to $n-1$, $t=t+2r$

5) 从存储位 $M_{t/2+s}$ 中读数据赋值给 (c_1, c_2)

6) 从存储位 $M_{t/2+s+r/2}$ 中读数据赋值给 (d_1, d_2)

7) $a_1 = c_1 + d_1, b_1 = w(c_1 - d_1)$

8) $a_2 = c_2 + d_2, b_2 = w(c_2 - d_2)$

9) 将 (c_1, c_2) 作为新数据存储回 $M_{t/2+s}$

10) 将 (d_1, d_2) 作为新数据存储回 $M_{t/2+s+r/2}$

11) end for

12) $w = w \times w_r^{-1}$

13) end for

14) end for

15) 从存储位 M_i 读取 (G_{2i}, G_{2i+1}) , $i=(0, 1, 2, \dots, n/2-1)$

16) 对 $(G_0, G_1, \dots, G_{n-1})$ 进行比特翻转重排, 并赋值给 $(g_0, g_1, \dots, g_{n-1})$

5 硬件架构设计

本节主要介绍本文设计的基于 NTT 算法的多项式乘法硬件架构，其主要包括各个硬件组件的设计、功能、组件之间的连接方式以及该硬件架构的调用方法。

5.1 硬件模块

本文构建的硬件架构的主要功能是完成 R_q 上的多项式乘法运算和多项式向量内积运算，即可实现正向、反向 NTT 以及多项式逐点相乘和累加等。本架构采用双端口随机存取存储器 (RAM) 来存储需要由硬件部分完成相关运算的多项式。所使用 RAM 的端口宽度为 64 bit，深度为 512，每个 RAM 位存储两个系数。每个多项式的 256 个系数均通过填充前置零的方法转化为 32 bit 之后两两拼接成 128 个 64 bit 的数据对，因此该 RAM 可以同时容纳 4 个多项式。采用双端口 RAM 可以保证在同一个时钟周期内同时完成一次数据读操作与一次数据写操作。除了 RAM 模块之外，本架构还包含以下几个核心模块。

(1) 地址生成模块

该模块主要负责生成硬件架构工作时，RAM 的读地址、写地址序列以及相对应的使能信号序列。

(2) 旋转因子生成模块

该模块主要负责生成架构在进行 NTT 算法时所需要的旋转因子序列。

(3) 运算模块

该模块主要负责完成架构工作时待处理数据所需进行的模加、模减、模乘运算以及数据对的调整。

(4) 驱动信号分流模块

该模块主要负责将总线的驱动信号通过连续寄存的方式分流成其余各模块需要的驱动信号，以校正各模块驱动信号的时钟差。

5.2 地址序列生成

本文硬件架构的设计思路之一是在 NTT 算法的每个阶段内将所有对应相同旋转因子的蝶形运算连续化处理，从而最大限度地降低旋转因子序列的变化次数，便于简化硬件电路。具体的地址跳转规律为：阶段 i 的地址序列为

$0, 2^i, 2 \times 2^i, \dots, (L_i - 1) \times 2^i, 1, 2^i + 1, 2 \times 2^i + 1, \dots, (L_i - 1) \times 2^i + 1, 2, \dots, (2^i - 1), 2^i + (2^i - 1), 2 \times 2^i + (2^i - 1), \dots, (L_i - 1) \times 2^i + (2^i - 1)$ ，其中 $L_i = n / 2^{i+1}$ ， $i = (0, 1, 2, \dots, \lg n - 2)$ ；阶段 $\lg n - 1$ 的地址序列与阶段 0 相同。本文的硬件架构仅通过一个逐时钟周期累加的简单计数器加以相应的比特重排即可实现符合上述变化规律的地址变换序列。图 1 是 $n = 256$ 时该过程的简单图解。

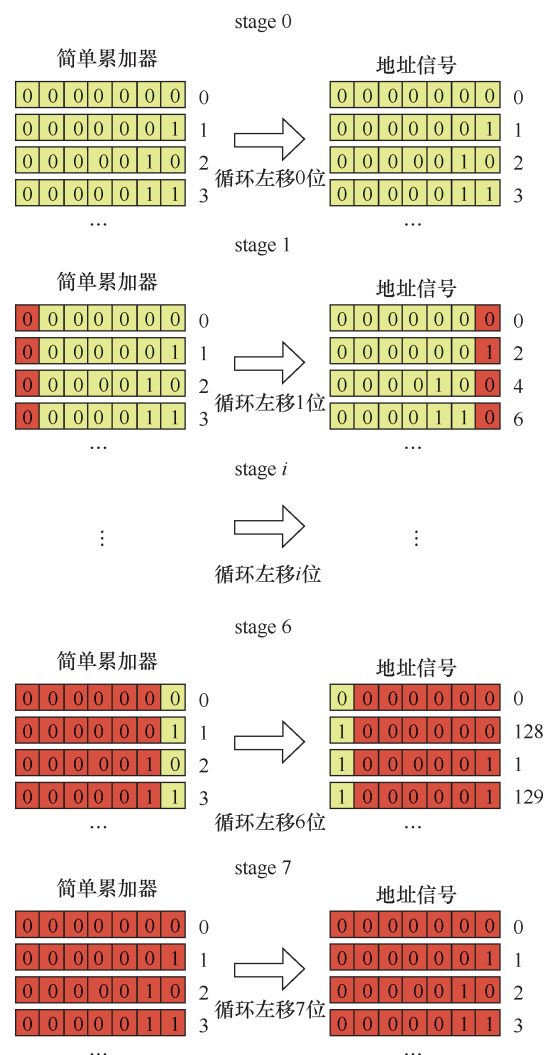


图1 地址生成方式图解
Figure 1 Diagram of address generation

5.3 算数模块

在本文的硬件架构中，算数模块主要负责完成 NTT 算法中的蝶形运算以及多项式的逐点乘法和逐点加法，对于 NTT 算法，该模块还同时承担了数据对重排的作用。架构中的模乘元件是其

中比较重要的元件，如 3.1 节所述，本架构中的模乘是通过标准数乘与特殊模数下的 Montgomery 约减所实现的，但为了进一步提升硬件架构的性能，额外地使用了一些微妙的流水线设计技巧，流水线级数为 6。本架构中流水线模乘算法如下所示。

算法 10 流水线模乘算法

输入 $a, b \in \{0, 1, 2, \dots, (q-1)\}$

$// q = 3870721, R = 2^{22}$

输出 $r = abR^{-1} \bmod q$

$//$ 第一级

1) $a_1 = a \gg 11, a_0 = a \bmod 2^{11}$

2) $b_1 = b \gg 11, b_0 = b \bmod 2^{11}$

3) $t_0 = a_0 b_0, t_2 = a_1 b_1, w_a = a_0 + a_1, w_b =$

$b_0 + b_1$

$//$ 第二级

4) $t_1 = w_a w_b - t_0 - t_2, m_1 = (t_0 \ll 16) - (t_0 \ll 12)$

$//$ 第三级

5) $t = t_2 \parallel t_0 + (t_1 \ll 11), m_2 = (m_1 \ll 6) - m_1$

$//$ 第四级

6) $m = m_2 + t$

7) $s_0 = (m \ll 16) - (m \ll 12)$

8) $t_H = (t \gg 22) + 1$

$//$ 第五级

9) $s_1 = (s_0 \ll 6) - s_0, s = s_1 + m$

$//$ 第六级

10) $r_1 = (s + t_H) \gg 22$

11) if ($r_1 > q$)

12) $r = r_1 - q$

13) else $r = r_1$

如 4.1 节所述，数据成对拼接存放在每一个 RAM 的地址位上，每一次 RAM 的读操作与写操作均同时对应两个系数中间量。地址跳转序列和本模块中的数据分流相互配合，以保证所有前一 NTT 阶段完成蝶形运算后写回 RAM 中的数据对刚好是下一阶段需要在一起进行蝶形运算的数据对，这也是本硬件架构实现 NTT 时的主要思想之一。例如，正向 NTT 算法，阶段 0 中地址位 0 内存储的是 (G_0, G_1) ，地址位 1 内存储的是 (G_2, G_3) 。先后读取 (G_0, G_1) 和 (G_2, G_3) 进入算数模块完成蝶形运算之后，将 (G'_0, G'_2) 和 (G'_1, G'_3) 先后写回地

址位 0 和地址位 1 中，而在阶段 1 中， (G'_0, G'_2) 即在该阶段需要进行蝶形运算的两个数据， (G'_1, G'_3) 也是如此。图 2 为此过程的一个图解说明。

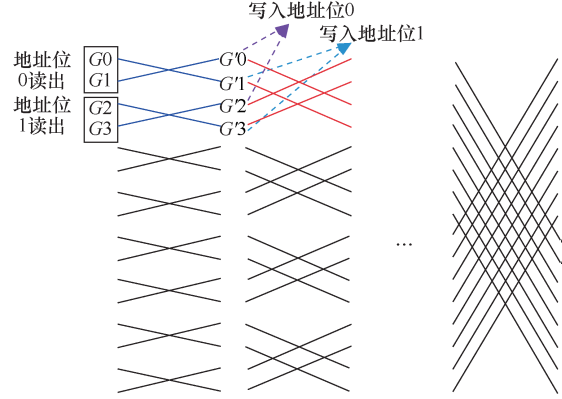


图 2 NTT 数据流向实例
Figure 2 An instance of data flow instance

5.4 旋转因子生成元件

为了尽可能使本文的硬件架构设计面积更小，采用在线生成 NTT 算法所需旋转因子的思路。一次 NTT 运算其中涉及的不同旋转因子共有 256 个，全部预存储会占用较大的硬件空间。以正向 NTT 算法为例，阶段 i 所需要的旋转因子序列为 $w_{2k}^1, w_{2k}^3, w_{2k}^5, \dots, w_{2k}^{k-1}$ ，其中 $k = 2^{i+1}$ ， $i=0, 1, 2, \dots, 7$ 。因此，只需预先存储 $w_2 R, w_4 R, \dots, w_{512} R, w_2^{-1} R, w_4^{-1} R, \dots, w_{512}^{-1} R$ 等 16 个因子就足够了，其余因子可以通过反复将模乘输出回代到输入的循环方式来生成。但在本文的硬件实现中，模乘元件流水线为 6 级，因此计入旋转因子模块中本身的数据缓存之后，每两个相邻的旋转因子之间需要间隔 7 个时钟周期。本架构所用 NTT 算法中旋转因子对应的变化规律为：正向 NTT 算法阶段 i 中旋转因子每 $n / 2^{i+1}$ 个时钟周期变换一次，反向 NTT 算法阶段 i 中旋转因子每 2^i 个时钟周期变换一次。对于正向 NTT 的最后 3 个阶段和反向 NTT 的前 3 个阶段，直接通过模乘元件生成是来不及的，因此针对这些阶段采用多起始因子的方法来保证旋转因子序列的正确性。以正向 NTT 算法的阶段 5 为例，旋转因子需要每 4 个周期更新一次，本文在第 0 个时钟周期向模乘元件输入 w_{128}^1 和 w_{32}^1 ，在第 4 个时钟周期向模乘元件输入 w_{128}^3 和 w_{32}^1 ，在第 8 个时钟周期模乘元件输出 w_{128}^5 ，在第 12 个时钟周期输出 w_{128}^7 ，如此即可得到本阶段所需

的因子序列, 即 $w_{128}^1, w_{128}^3, w_{128}^5, \dots, w_{128}^{63}$ 。图 3 给出了此流程更加直观的图解说明。

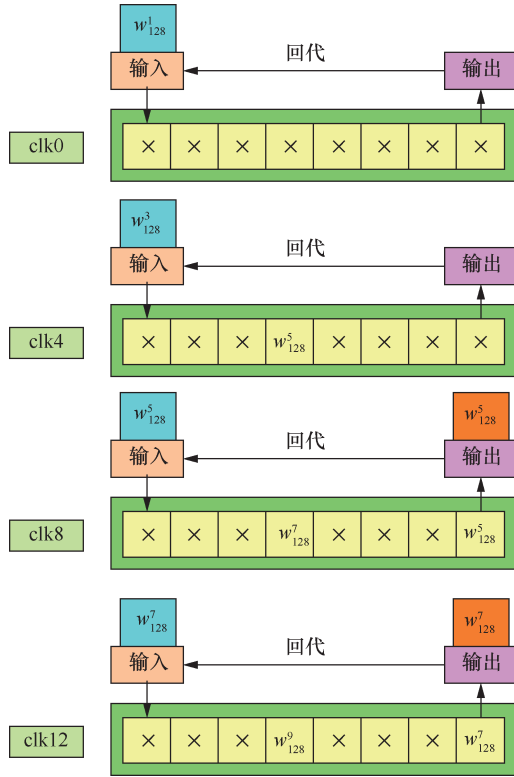


图 3 旋转因子生成部分过程 (正向阶段 5)
Figure 3 Partial process of twiddle factor generation (forward stage 5)

对于正向 NTT 阶段 6 和阶段 7, 方法是相同的, 只是增加的起始因子个数更多而已, 反向 NTT 的情况同样是类似的。该方法可以使用的前提是模乘元件流水线的总级数 (包含输出回代到输入

的一级寄存器缓冲) 需具备 $2^k - 1$ 的形式, 同时级数越长, 需要额外预存储的起始旋转因子就会越多。为了达到预存储的空间大小和硬件模块的最大频率 (即寄存器间逻辑电路最大长度) 之间的平衡和折中, 本文选择 6 级流水线结构。综上所述, 本架构中需要预先存储的旋转因子一共有 $40(20 \times 2)$ 个, 如表 2 所示。相比较于将 512 (256×2) 个旋转因子全部预先存储而言, 使用调整后的在线生成方法仍然可以节约大量存储空间。

表 2 正向与反向 NTT 各阶段所需的旋转因子
Table 2 Twiddle factors required for each stage of forward and backward NTT

正向与反向 NTT 阶段	所需旋转因子次数 (以为 w_{512} 基准)
0 / 7	128
1 / 6	64
2 / 5	32
3 / 4	16
4 / 3	8
5 / 2	4, 12
6 / 1	2, 6, 10, 14
7 / 0	1, 3, 5, 7, 9, 11, 13, 15

5.5 整体架构

在本文中, 各个硬件模块组合成完整的硬件架构之后通过总线协议与平台上内置的 CPU 软核相连接, 与之共同完成 Aigis-sig 方案的软硬件协同实现。本文的整体硬件架构如图 4 所示。

CPU 与硬件部分通过总线通信, 传递内容主

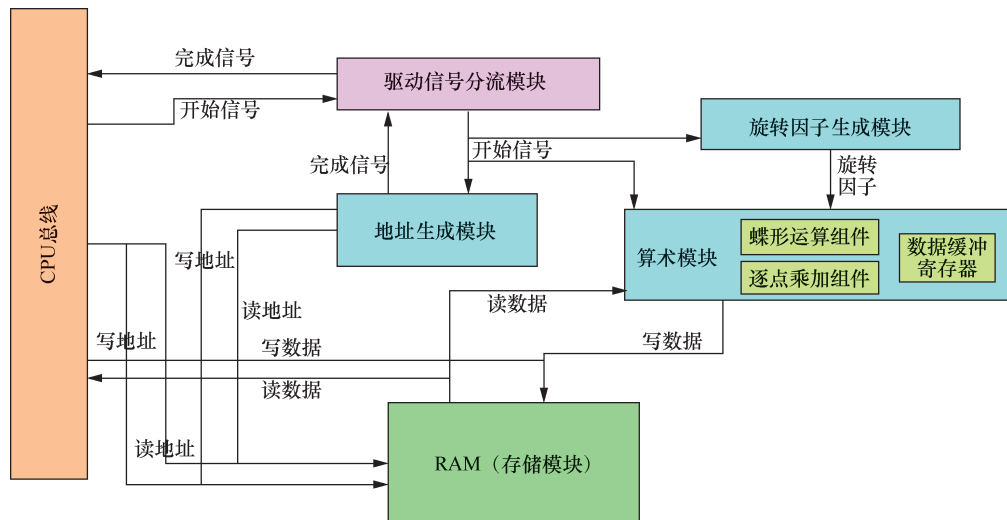


图 4 整体硬件架构
Figure 4 Overall hardware architecture

要有两种：一是通过总线将待处理多项式系数数据写入硬件的 RAM 中以及将 RAM 中经过处理的系数数据读出；二是通过总线向硬件发送相应的驱动信号以及监听硬件部分抛出的完成信号。架构中的地址生成模块、旋转因子生成模块以及算术模块均包含有限状态机(FSM)，但均仅包含“空闲”和“工作”两种状态，当接收到相应的开始驱动信号时，模块从“空闲”进入“工作”状态，完成相应任务，完成之后将自动切换回“空闲”状态，并由地址生成模块负责产生完成信号。由于 3 个 FSM 之间开始工作的时钟周期存在一定偏差，因此需要驱动信号分流模块将总开始信号按照校正值分流成 3 个开始信号来保证各模块间的正常运行。

5.6 架构功能与调用方式

本文的硬件架构中的 RAM 可以同时存储 4 个多项式，分别记为 0 号、1 号、2 号、3 号位，架构可以完成的所有操作指令主要有以下 6 种。

- (1) 写多项式：向硬件模块的 0 号、1 号、2 号、3 号写入多项式。
- (2) 读多项式：从硬件模块的 0 号、1 号、2 号、3 号读出多项式。
- (3) 正向 NTT：0 号、1 号多项式进行正向 NTT 操作。
- (4) 反向 NTT：2 号、3 号多项式进行反向 NTT 操作。
- (5) 逐点相乘：0 号多项式与 1 号多项式逐点相乘之后存入 2 号多项式位。
- (6) 逐点相加：2 号多项式与 3 号多项式逐点相加之后存入 3 号多项式位。

以上 6 种指令可以保证 Aigis-sig 方案中大部分的多项式运算操作由硬件架构来代替 CPU 完成，以下给出一个示例来说明如何通过本硬件架构实现 R_q 上 4 个分量的多项式向量内积运算，如算法 11 所示。

算法 11 多项式向量内积算法

输入 $(a_0, a_1, a_2, a_3) \in R_q^4, (b_0, b_1, b_2, b_3) \in R_q^4$,

$$E = \sum_{i=0}^{n-1} ex^i, \quad e = 2\,566\,823 = n^{-1}R^2 \bmod q$$

输出 $c = \sum_{i=0}^3 a_i b_i$

1) 3 号多项式位清零

- 2) for $i=0$ to 3, $i=i+1$:
- 3) 将多项式 a_i 写入 0 号位
- 4) 将多项式 b_i 写入 1 号位
- 5) 对 0 号多项式进行正向 NTT
- 6) 对 1 号多项式进行正向 NTT
- 7) 执行逐点相乘
- 8) 执行逐点相加
- 9) end for
- 10) 对 3 号多项式进行反向 NTT
- 11) 将 3 号多项式读出赋值给 d
- 12) 将多项式 d 写入 0 号位
- 13) 将多项式 e 写入 1 号位
- 14) 执行逐点相乘
- 15) 将 2 号多项式读出赋值给 c

然而，在具体实现 Aigis-sig 方案中的多项式运算时，并不需要严格按照最完整的过程执行所有步骤。例如，矩阵 A 的行向量作为多项式向量内积运算输入之一，依照方案，该矩阵为随机矩阵，因而可以直接视为 NTT 表达形式，不需要额外的正向 NTT 变换；又如矩阵 A 与向量 y 进行乘法运算时，需要计算 k 次内积，但向量 y 是公共的，因此不需要对其重复执行正向 NTT 变换。

6 软硬协同实现

6.1 软件实现

在本文所描述的实现方法中，软件部分所使用的 C 语言程序代码是 Aigis-sig 方案设计团队在全国密码算法设计竞赛中所提交的参考实现代码的基础上修改而来的^[12]，代码的修改主要是将一些本文的实验平台所不支持的数据结构、函数指令以及外源依赖库等全部替换为平台支持且逻辑完全相同的功能实现。修改之后的 C 程序代码即本文在实验阶段用于性能对比的纯软件实现代码。

6.2 协同实现

本文所提出的软硬件协同实现方法本质上是纯软件实现中涉及多项式运算的部分由软件 CPU 实现转化为硬件架构实现。平台内置的 CPU 为主设备，硬件架构为从设备，软件部分由 CPU 独自实现，硬件部分则是 CPU 通过相应的总线协议，将待处理数据以及处理指令传送给硬件架构，从而调用其完成相应的逻辑功能，再将处理完的

数据返还给 CPU。图 5 为纯软件实现与软硬件协同实现的简单说明。

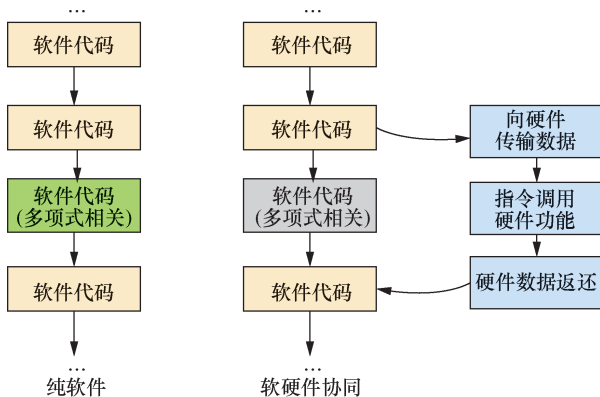


图 5 纯软件实现与软硬件协同实现
Figure 5 Software-only implementation and software/hardware cooperative implementation

7 实验

7.1 平台及测试方法

为了更客观地评估本文所提出的实现方案的性能,采用与同一平台上的纯软件实现进行对比的方式展示相关数据。本文所使用的 FPGA 平台为 Xilinx Zynq 7000 Soc XC7Z020-2CLG400I,内置 PS 端 Cortex-A9 ARM 处理器与 PL 端 Artix-7 架构可编程逻辑资源,软件与硬件(PS 端与 PL 端)之间采用 AXI 总线协议传递相关数据。本文的对比实验包括 NTT 算法和签名、验签两个层面,针对 NTT 算法的性能评估,本文提供一些已存在且性能较优的硬件设计的相关数据,并从硬件面积和时钟周期消耗数两个方面进行比较。

7.2 多项式运算性能

对 Aigis-sig 方案的实现而言,环上多项式乘

法的实现效率很大程度上影响整个方案的性能。本文利用 NTT 算法实现多项式相乘是硬件架构主要实现的功能之一。表 3 给出了本架构的硬件部分在本文实验平台的逻辑资源消耗、时钟周期、最大频率等数据,其中资源消耗指标包括查找表(LUT)、数字信号处理器(DSP)、块存储器(Bram),同时给出一些具有对比价值的设计成果用于参照。

表 3 的数据表明,与 Roy 等在文献[21]中的设计相比,本文的架构消耗了更多的 DSP 和 LUT 资源,最大频率则大约是文献[21]的 50%~60%,这主要是模数 7 681 和 12 289 的比特长度较短,模乘本身复杂度就低,然而本文的架构可以在更少的时钟周期内完成 256 次多项式的 NTT 算法。Feng 等在文献[26]中给出的硬件设计其所需的时钟周期仅为工作的 20%,然而本文的架构所消耗的 LUT 资源仅相当于文献[26]的 10%左右,对于资源受限的应用设备,本文的硬件架构设备更具优势。模数 q 仅在 Aigis-sig 方案中出现,因此目前还没有模数相同,更具有公平参照价值的现有研究成果用作对比。

在密码算法方案的软硬件协同实现方法中,硬件模块执行相关运算的时间消耗即便是在低时钟频率驱动的前提下一般也是非常低的,远小于将软核内的待处理数据通信传输到硬件模块,或将硬件模块中的结果数据返还给 CPU 软核所消耗的时间,表 4 给出了本文的实验测试结果,可以看出 NTT 算法硬件执行时间仅占总消耗时间的 6.2%左右。正是出于这个原因,本文的实验架构中才选择加入了逐点乘和逐点累加的功能,这种方法使在计算多项式向量内积操作时可以节约通信次数,从而提高实现的时间性能。

表 3 NTT 性能对比(多项式长度 $n=256$)
Table 3 Performance comparison of NTT(length $n=256$)

设计	模数 q	平台	LUT	DSP	Bram	频率/MHz	时钟数
本文	3 870 721	Artix-7	1 623	7	1	168	1 041
文献[21]	7 681	Virtex-7	1 349	1	2	313	$\approx 1\,210$
文献[21]	12 289	Virtex-7	1 536	1	3	278	$\approx 1\,210$
文献[26]	1 049 089	Spantan-6	$\approx 14\,000$	128	1	—	220
文献[27]	—	Spantan-6	533	1	2	233	4 066
文献[24]	7 681	TSMC	—	—	—	72	1 289

注:“—”表示无法获得。

表4 总线通信性能
Table 4 Performance of bus communication

测试项目	所用时间/s
5×10^5 万次 RAM 读 + 5×10^5 次 RAM 写 + 5×10^5 次 NTT	75.42
1 次 RAM 读 + 1 次 RAM 写 + 5×10^5 次 NTT	4.68

7.3 数字签名性能

本文所实现的 Aigis-sig 方案的参数组为 PARAMS II, 该组参数为推荐参数。表 5、表 6 分别给出了在本文的 FPGA 平台上, 不同的实现方法中签名阶段和验签阶段在本架构上实现的时间消耗。除此之外, 为了说明本文的硬件架构对方案性能提升的程度, 本文也给出了在 FPGA 平台上纯 ARM 核实现的时间消耗结果。实验测试中, 软硬件协同实现和纯软件实现的 CPU 软核频率设置均为 666.66 MHz, 该设置为本文所使用的开发板推荐的默认设置, 同时也是保证程序实现正常运行的最大频率。在表 3 中已说明本文的硬件模块正常运行的最大时钟频率为 168MHz, 因此将硬件架构的时钟频率设置为 150MHz, 即不超过最大频率前提下, 便于平台晶振准确生成的最大频率。

表5 签名阶段性能
Table 5 Performance of signature

实现方法	FF	LUT	DSP	Bram	速度/ms·次 ⁻¹
软硬件协同	2 057	1 997	7	1	49.82
纯软件	498	374	0	0	68.18

表6 验证阶段性能
Table 6 Performance of verification

实现方法	FF	LUT	DSP	Bram	速度/ms·次 ⁻¹
软硬件协同	2 057	1 997	7	1	16.26
纯软件	498	374	0	0	19.61

由表 5 和表 6 可知, CPU 频率和硬件频率分别设置为 666.66 MHz 和 150 MHz 时, 本文设计的硬件架构设计的引入最终使 Aigis-sig 方案的性能在签名阶段和验签阶段, 相较于同样平台下的纯 ARM 核软件实现, 分别取得约 26.93% 和 17.08% 的提升。在本文之前尚未有 Aigis-sig 方案在嵌入式平台上实现的相关工作发表和展开, 因此没有对比数据, 然而单从 FF、LUT、DSP、Bram 消耗量上来看, 本文实现的资源消耗是很少的。

8 结束语

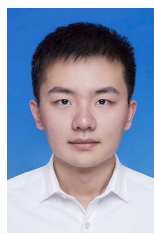
基于理想格构造的密码方案的快速实现在量子计算机不断发展的时代背景下具有强大的研究价值和应用前景。本文基于 Aigis-sig 方案设计了一种多项式运算的紧凑硬件架构, 并以此为基础, 提出了 FPGA 平台下 Aigis-sig 方案的软硬件协同快速实现方法。实验结果表明, 在 Xilinx Zynq-7000 SoC 平台上, CPU 频率和硬件频率分别设置为 666.66 MHz 和 150 MHz 时, 本实现方法使 Aigis-sig 的签名阶段和验签阶段分别取得约 26.93% 和 17.08% 的性能提升。

参考文献:

- [1] KELLY, J, BARENDT R, FOWLER A G, et al. State preservation by repetitive error detection in a superconducting quantum circuit[J]. Nature, 2015, 519(7541): 66-69.
- [2] SHOR P W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer[J]. SIAM Review, 1999, 41(2): 303-332.
- [3] 杨妍玲. 量子密码在信息安全中的应用与分析[J]. 信息与电脑 (理论版), 2020, 32(8): 177-181.
YANG Y L. Application and analysis of post-quantum cryptography in information security[J]. Information and Computers (Theoretical), 2020, 32(8): 177-181.
- [4] CHEN L, JORDAN S, LIU Y K, et al. Report on post-quantum cryptography[M]. Maryland: National Institute of Standards and Technology, 2016.
- [5] ALAGIC G, ALPERIN-SHERIFF J, APON D, et al. Status report on the first round of the NIST post-quantum cryptography standardization process[M]. Maryland: National Institute of Standards and Technology, 2019.
- [6] ROMA C A, TAI C A, HASAN M A. Energy consumption of round 2 submissions for NIST PQC standards[R]. Maryland: National Institute of Standards and Technology, 2019.
- [7] 中国密码学会. 全国密码算法设计竞赛进入第 2 轮公钥算法[EB]. Chinese Association for Cryptologic Research. Public key scheme selected to the second-round competition of national cryptographic algorithm competition[EB].
- [8] 中国密码学会. 关于全国密码算法设计竞赛算法评选结果的公示[EB]. Chinese Association for Cryptologic Research. Announcement of the selection results of the national cryptographic algorithm competition[EB].
- [9] REGEV O. On lattices, learning with errors, random linear codes, and Cryptography[J]. Journal of the ACM (JACM), 2009, 56(6): 1-40.
- [10] AJTAI M. Generating hard instances of lattice problems[C]//Proceedings of the twenty-eighth annual ACM symposium on Theory

- of Computing. 1996: 99-108.
- [11] LYUBASHEVSKY V, PEIKERT C, REGEV O. On ideal lattices and learning with errors over rings[J]. Journal of the ACM (JACM) 2013, 60(6): 1-35.
- [12] 中国密码学会. 数字签名方案 Aigis-sig 算法提交文档[EB]. Chinese Association for Cryptologic Research. Submitted document of the digital signature scheme Aigis-sig [EB].
- [13] NEJATOLLAHI H, DUTT N, RAY S, et al. Software and hardware implementation of lattice-based cryptography schemes[J]. University of California Irvine, CECSTR 17 4 (2017).
- [14] DE C R, ROY S S, VERCAUTEREN F, et al. Efficient software implementation of Ring-LWE encryption[C]//Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'15). 2015.
- [15] YUAN Y J, CHENG C M, KIYOMOTO S, et al. Portable implementation of lattice-based cryptography using JavaScript[J]. International Journal of on Computing and Networking. 2016, 6(3): 309-327.
- [16] STEBILA D, MOSCA M. Post-quantum key exchange for the internet and the open quantum safe project[R]. Cryptology ePrint Archive, 2016.
- [17] AGUILAR-MELCHOR C, BARRIER J, GUELTON S, et al. NTLlib: NTT-based fast lattice library[C]//RSA Conference. 2016: 341-356.
- [18] CHEN H, HAN K, HUANG Z, et al. Simple encrypted arithmetic library - seal (v2.3)[R]. 2017.
- [19] GÖTTERT N, FELLER T, SCHNEIDER M, et al. On the design of hardware building blocks for modern lattice-based encryption schemes[C]//Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES'12). 2012.
- [20] PÖPPELMANN T, GÜNEYSU T. Area optimization of lightweight Lattice-based encryption on reconfigurable hardware[C]// Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'14). 2014.
- [21] ROY S S, VERCAUTEREN F, MENTENS N, et al. Compact ring-LWE cryptoprocessor[C]//Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES'14). 2014.
- [22] BASU K, SONI D, NABEEL M, et al. NIST post-quantum cryptography-a hardware evaluation study[C]//IACR Cryptology ePrint Archive. 2019: 47.
- [23] DANG V B, FARAHMAND F, ANDRZEJCZAK M, et al. Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign[C]//2019 International Conference on Field-Programmable Technology (ICFPT). 2019: 206-214.
- [24] BANERJEE U, UKYAB T S, CHANDRAKASAN A P. Sapphire: a configurable crypto-processor for post-quantum Lattice-based protocols (Extended Version)[C]//IACR Crypto. ePrint Arch. 2019: 1140.
- [25] 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.
- [26] FENG X, LI S G, XU S F. RLWE-oriented high-speed polynomial multiplier utilizing multi-lane stockham NTT algorithm[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2019, 99: 1.
- [27] DU C H, BAI G Q. Towards efficient polynomial multiplication for Lattice-based cryptography[C]//2016 IEEE International Symposium on Circuits and Systems (ISCAS). 2016: 1178-1181.

[作者简介]



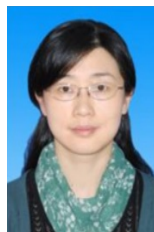
周朕（1996—），男，安徽亳州人，武汉大学硕士生，主要研究方向为后量子密码、密码算法设计实现。



何德彪（1980—），男，山东阳谷人，武汉大学教授、博士生导师，主要研究方向为密码学、区块链安全等。



罗敏（1974—），男，湖北武汉人，武汉大学副教授，主要研究方向为密码学、区块链安全等。



李莉（1976—），女，湖北武汉人，武汉大学副教授，主要研究方向为密码学、区块链安全等。