

面向 Saber 算法的并行乘法器

吕 杰¹, 汪鹏君^{2*}, 张会红¹

(1. 宁波大学 信息科学与工程学院, 浙江 宁波 315211; 2. 温州大学 电气与电子工程学院, 浙江 温州 325035)

摘要: 随着量子计算的发展, 现有密码系统的安全性将受到严重威胁. Saber 算法是抵御量子计算攻击的后量子密码方案之一, 但存在多项式商环上模乘占据运算开销过大的问题. 鉴此, 本文通过对 Karatsuba 算法和 Schoolbook 相乘方式的剖析, 提出一种面向 Saber 算法的并行乘法器设计方案. 该方案首先利用 Karatsuba 算法分解模乘运算的关键路径, 结合乘法复用和加法替换的策略减少硬件开销, 然后采用并行运算电路压缩关键运算路径时长, 最后在 TSMC 65 nm 工艺下, 利用 Modelsim 和 DC 软件仿真验证. 结果表明: 该方案运算时长为 137 个时钟周期, 与传统方式相比速度提升 46.50%, 功耗为 87.83 mW, 面积为 $927.32 \times 10^3 \mu\text{m}^2$.

关键词: 后量子密码; Saber 算法; Karatsuba 算法; Schoolbook 相乘方式; 并行乘法器

中图分类号: TP391

文献标志码: A

文章编号: 1001-5132 (2022) 06-0015-07

公钥密码广泛应用于信息网络安全、国防安全等重要领域, 其密码系统的安全性基于数学难题^[1], 如基于大数分解质因数的 RSA (Rivest Shamir Adleman, RSA) 算法和基于离散对数问题的 ECC (Elliptic Curve Cryptography, ECC) 算法. 然而随着量子计算的发展, Shor 算法和 Grover 搜索算法在理论上可攻破现有密码系统^[2]. Shor 等^[3]引入一种快速量子算法, 其多项式时间内求解任意正整数 N 的质因数分解方法将完全破坏公钥加密体制的安全性. Grover 等^[4]提出解决无结构数据库搜索的算法, 将迫使对称加密增加密钥长度以应对此类攻击. 因此具备抵御量子攻击的新型公钥密码方案, 即后量子密码 (Post-Quantum Cryptography, PQC) 备受密码学者关注.

自 2016 年以来, 美国国家标准与技术研究院已经开展了三轮 PQC 方案的标准化竞赛. 预计在 2024 年, PQC 的国际标准方案最终确立. 基于“Module Learning with Rounding”难题的 Saber 算法作为候选者之一, 以二次幂为模数的规则避免额外模约运算开销与拒绝采样操作^[5], 但是该算

法的核心算子——模多项式间的乘法在算法整体时间开销中占据较大比重^[1], 特别是其模数规则限制数论变换 (Number Theoretic Transform, NTT) 在模多项式相乘的直接应用. 由于 NTT 在素域中才生效^[6], 因此优化 Saber 算法内核心算子能有效提高算法的整体性能. D'Anvers 等^[7]融合 Toom-Cook 和 Karatsuba 算法, 设计不依赖 NTT 的计算流程, 实现软件层面的 Saber 算法方案. Wang 等^[8]使用 ESP32 协处理器设计基于大整数协处理器的 Saber 算法方案, 为 Saber 算法在物联网设备中的应用提供技术支撑. Paksoy 等^[9]采用基于托普利兹矩阵的多项式模乘算法来简化运算, 实现 ARM Cortex-M4 平台上的 Saber 算法方案. Bermudo 等^[10]提出预计算方式的插值技术, 以减少资源开销, 实现基于 Toom-Cook 的乘法架构. 此外, 亦有学者开展硬件层面研究, 如 Maria 等^[11]通过专用硬件协处理单元加速核心算子运算, 以提高 Saber 算法速度. Basso 等^[6]利用 Schoolbook 相乘的便捷性, 结合集中系数相乘的机制以减少面积开销, 实现 3 种不同性能的乘法器. Sinha 等^[12]提出一种多项式乘法的架构,

收稿日期: 2022-05-31.

宁波大学学报(理工版) 网址: <http://journal.g.nbu.edu.cn/>

基金项目: 国家自然科学基金 (62134002, 62174121).

第一作者: 吕杰 (1996-), 男, 浙江绍兴人, 在读硕士研究生, 主要研究方向: 集成电路和信息安全等. E-mail: lv_jie_i_am@163.com

*通信作者: 汪鹏君 (1966-), 男, 浙江宁波人, 教授, 主要研究方向: 集成电路和信息安全等. E-mail: wangpengjun@wzu.edu.cn

以克服内存访问瓶颈, 并完成 Saber 算法的硬件方案.

综上所述, 基于 Schoolbook 相乘方式的计算复杂度过高. 依靠 Toom-Cook 算法可有效降低运算的复杂度, 但其多次划分乘法运算会引入额外寄存器硬件开销. 而 Karatsuba 算法可折中分解多项式乘法的运算, 同时降低计算复杂度^[10]. 鉴此, 本文通过对 Karatsuba 大数相乘分治策略的研究, 分析 Saber 算法中模多项式运算的关键路径, 利用 Karatsuba 算法分解并压缩核心算子关键路径时长. 同时利用 DC(Design Compiler)设计可综合的硬件电路, 提取该乘法器的时序、功耗和面积等的性能参数, 实现并行乘法器电路设计.

1 Saber 算法和 Karatsuba 算法分析

1.1 Saber 算法

Saber 算法是现有公钥密码体系的分支, 由密钥生成、加密和解密三者构成 Saber 公钥加密方案. 核心算子——模多项式间的乘法在三阶段中均涉及密钥生成阶段, 公钥包含多项式矩阵 $A_{L \times L}$ 和多项式向量 $b_{L \times 1}$, 私钥包含 $S_{L \times 1}$. b 由 A 和 S 乘积的缩放舍入操作生成. 加密阶段: 明文 m 与 $v' = b^T s'$ 的运算构成密文 c , 同时密文还包括 b' 的信息, b' 的生成与密钥 b 生成类似; 解密阶段: 明文 m' 从 v 中恢复原有信息, v 的生成亦存在多项式向量间的模乘运算. Saber 方案整体框架和参数细节可参考第三轮原始 Saber 方案, 核心算子如下^[5]:

$$A_{L \times L} = \begin{bmatrix} A_{11} & \cdots & A_{1L} \\ \vdots & A_{ij} & \vdots \\ A_{L1} & \cdots & A_{LL} \end{bmatrix}, \quad (1)$$

$$S_{L \times 1} = \begin{bmatrix} S_{11} \\ \vdots \\ S_{L1} \end{bmatrix}, \quad (2)$$

$$P = x^{256} + 1, \quad (3)$$

$$A_{ij} = a_{255}x^{255} + a_{254}x^{254} + \cdots + a_0x^0, \quad (4)$$

$$a_k \in [0, 8192]$$

$$S_{ij} = s_{255}x^{255} + s_{254}x^{254} + \cdots + s_0x^0, \quad (5)$$

$$s_k \in [-4, 4].$$

上述公式中, A_{ij} 、 S_{ij} 分别以多项式向量 $(a_{255}, a_{254}, \cdots, a_0)$ 、 $(s_{255}, s_{254}, \cdots, s_0)$ 简化表示. P 为有限域内的不可约多项式, 用于进行模运算化简. L 为可

配置参数, 图 1 中 L 取 3. 矩阵-向量相乘运算结构如图 1 所示, 模乘运算的关键路径为

$$(A_{ij} \cdot S_{ij}) \bmod P (A_{ij} \cdot S_{ij}) \bmod P.$$

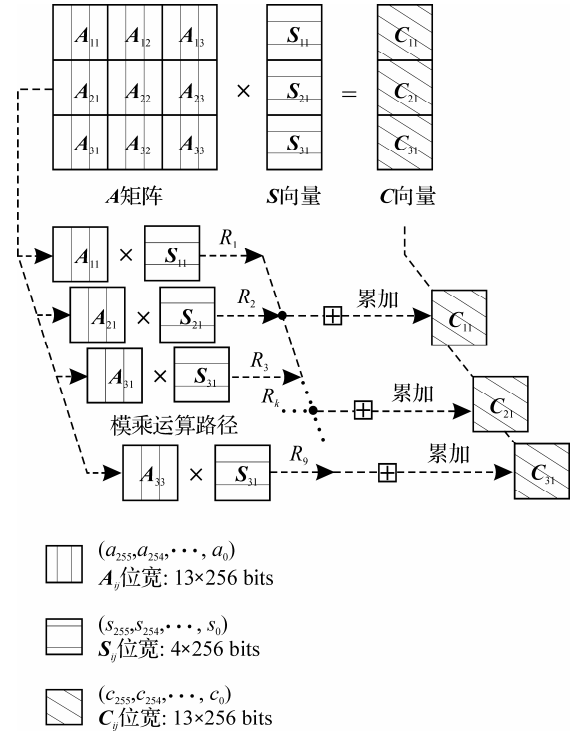


图 1 矩阵-向量相乘

根据矩阵-向量相乘规则, 矩阵 A 的行向量乘以列向量 S 得向量 C . 如 R_1 为 $(A_{11} \cdot S_{11}) \bmod P$; R_2 为 $(A_{12} \cdot S_{21}) \bmod P$; R_3 为 $(A_{13} \cdot S_{31}) \bmod P$; R_4 为 $(A_{21} \cdot S_{11}) \bmod P$; R_5 为 $(A_{22} \cdot S_{21}) \bmod P$; R_6 为 $(A_{23} \cdot S_{31}) \bmod P$; R_7 为 $(A_{31} \cdot S_{11}) \bmod P$; R_8 为 $(A_{32} \cdot S_{21}) \bmod P$; R_9 为 $(A_{33} \cdot S_{31}) \bmod P$; R_1, R_2, R_3 三者累加得数值 C_{11} ; 同理计算数值 C_{21} 和 C_{31} . A_{ij} 、 S_{ij} 和 C_{ij} 多项式向量均为高位宽.

1.2 Karatsuba 算法

Karatsuba 算法是以分治策略为思想的快速乘法. 该算法在折中分解计算时, 用加法代替原有乘法, 降低运算复杂度, 以额外寄存器和加法器替换乘法的开销. 因此 Karatsuba 算法可降低直接相乘 $O(n^2)$ 的时间复杂度至 $O(n^{1+\epsilon})$, $0 < \epsilon < 1$, 其原理见式(6)~式(9)^[10-11]. $n-1$ 阶多项式 $f(x)$ 、 $g(x)$ 以式(6)和式(7)折中分解, 二者乘积见式(8). f_H 、 f_L 、 g_H 和 g_L 均为多项式向量的 $n/2$ 项高位系数和 $n/2$ 项低位系数. 在式(9)中, $(f_H + f_L) \cdot (g_H + g_L) - f_H \cdot g_H - f_L \cdot g_L$ 替换式(8)的 $(f_H \cdot g_L + f_L \cdot g_H)$. 与式(8)

相比, 式(9)复用 $f_H \cdot g_H$ 和 $f_L \cdot g_L$ 乘法单元, 减少重复运算.

$$f(x) = f_H \cdot x^{n/2} + f_L, \quad (6)$$

$$g(x) = g_H \cdot x^{n/2} + g_L, \quad (7)$$

$$f(x) \cdot g(x) = f_H \cdot g_H \cdot x^n + (f_H \cdot g_L + f_L \cdot g_H) \cdot x^{n/2} + f_L \cdot g_L, \quad (8)$$

$$f(x) \cdot g(x) = f_H \cdot g_H \cdot x^n + f_L \cdot g_L + ((f_H + f_L) \cdot (g_H + g_L) - f_L \cdot g_L - f_H \cdot g_H) \cdot x^{n/2}. \quad (9)$$

将多项式向量 A_{ij} 、 S_{ij} 分别以式(6)、式(7)的形式代入式(9), 结合 Saber 算法模约规则, 得式(10). P_0 、 P_1 和 P_2 均包含多项式的 256 位系数, 其包含的乘加运算参见式(11)~式(13), 此处的乘加运算均指向多项式间的运算. A_H 、 A_L 、 S_H 和 S_L 均为各向量的 128 项高、低位系数, 参见式(14)、式(15). 至此, 核心算子关键路径被 Karatsuba 算法压缩为并行预加 ($A_H + A_L, S_H + S_L$)、并行乘法 (P_0, P_1, P_2) 和后加.

$$R_K = (A_{ij} \cdot S_{ij}) \bmod P = (A_H \cdot x^{128} + A_L) \cdot (S_H \cdot x^{128} + S_L) \bmod P = (P_0 x^{256} + (P_2 - P_0 - P_1) x^{128} + P_1) \bmod P, \quad (10)$$

$$P_0 = (A_H \cdot S_H) \bmod P, \quad (11)$$

$$P_1 = (A_L \cdot S_L) \bmod P, \quad (12)$$

$$P_2 = (A_H + A_L) \cdot (S_H + S_L) \bmod P, \quad (13)$$

$$A_H = (a_{255}, \dots, a_{128}), A_L = (a_{127}, \dots, a_0), \quad (14)$$

$$S_H = (s_{255}, \dots, s_{128}), S_L = (s_{127}, \dots, s_0). \quad (15)$$

上述公式的取模运算可参见 Saber 算法的模数规则^[5]. mod8192 和 mod9 通过直接限定数据位宽实现模约电路, 同时 mod P 利用 Saber 算法的模约规则, 通过更改系数正负号实现. 促成 Karatsuba 算法与 Schoolbook 相乘方式的融合. R_k 计算如图 2 所示, 具体可分为 3 个阶段: 预加(Pre-add)、乘法(Multiplication)和后加(Post-add), 分别对应图中算法的(1)~(3).

输入: A_{ij}, S_{ij}

输出: R_k

$$(1) R_A \leftarrow (A_H + A_L) \bmod 8192, R_S \leftarrow (S_H + S_L) \bmod 9;$$

$$(2) P_0 \leftarrow (R_A \times S_H) \bmod P, P_1 \leftarrow (A_L \times S_L) \bmod P,$$

$$P_2 \leftarrow (R_A \times R_S) \bmod P;$$

$$(3) R_k \leftarrow (P_0 x^{256} + (P_2 - P_0 - P_1) x^{128} + P_1) \bmod P.$$

图 2 基于 Karatsuba 的多项式并行模乘

2 多项式乘法器设计

2.1 多项式乘法器的状态机

由式(9)的分解过程可知, 应用于 Saber 算法乘法器的硬件电路有 6 种状态(图 3), 空闲(Idle)状态、加载数据(Load)状态、预加状态(Pre-add)、乘法状态(Multiplication)、后加状态(Post-add)、完成状态(done). 预加状态、乘法状态和后加状态分别对应图 2 中的 3 个步骤. 其中, T_{clk} 为单个时钟周期. 在 $131T_{clk}$ 乘法状态的关键时序中, $128T_{clk}$ 用于并行乘法计算, $3T_{clk}$ 用于满足数字后端的建立时间的时序要求.

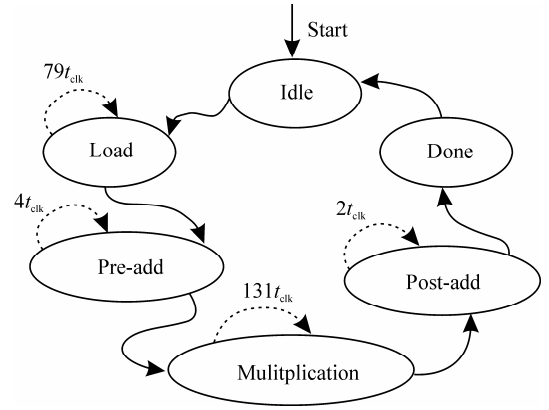


图 3 多项式乘法器的状态机转换图

2.2 基于选择器的累加器

乘积累加器(Multiply Accumulator, MAC)根据文献[6]提出的预先相乘方式设计, 部分乘法单元替换为数据选择器, 复用 128 次相同的乘法, 从而减少面积开销. 基于 Karatsuba 的多项式并行模乘电路中包含 2 种类型的 MAC, 记图 2 内 P_0 和 P_1 的运算电路为 MAC_I , P_2 的运算电路为 MAC_{II} . 在计算式(12)过程中, A_L 和 S_L 中的系数分别存储在 2 组系数寄存器中. 由于 s_j 的系数通过两随机变量的汉明之差获得, 且取值为 $[-4, 4]$, 因此 $a_i \times s_j$ 共有 4 种不同数值(正负用符号位表示). 由图 4 可见, 该电路将预先相乘的输出 $(a_i, 2a_i, 3a_i, 4a_i)$ 作为数据选择器的输入; 同时, s_j 为并行数据选择器的选择信号, 选择对应的预乘结果; 然后通过移位操作累加至寄存器组, 此移位操作与 a_i 系数项 x 的幂次相关. 完成上述累加共需计算 128 次, 输出 P_1 . 图 4 中, MAC 处理的操作为多项向量 $(a_{127}, \dots, a_0)$ 和 $(s_{127}, \dots, s_0)$ 间的乘累加运算. a_i 依次与 $(s_{127}, \dots, s_0)$

的 128 项系数相乘, 以 s_j 信号选择预乘结果 ($a_i, 2a_i, 3a_i, 4a_i$) 中的数值代替直接相乘的结果. 因此每次 $a_i \cdot (s_{127}, \dots, s_0)$ 计算均能复用 128 轮的预乘结果. 预乘的方式复用 128 轮乘法运算, 节省并行硬件电路的开销. MAC_{II} 的设计类似, 但 $a_i \times s_j$ 有 8 种不同的数值.

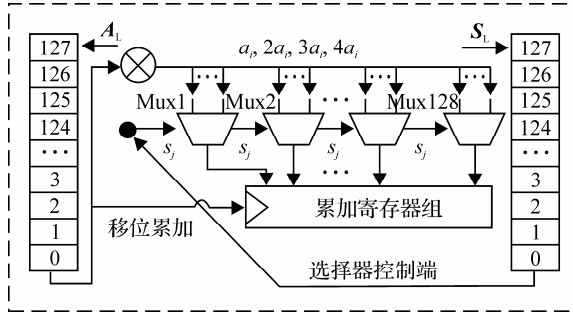


图 4 基于选择器的 MAC 电路

2.3 基于 Karatsuba 的多项式并行模乘

基于 Karatsuba 的多项式并行模乘电路框架如图 5 所示, 包括 3 个计算电路: 并行预加电路、并行乘法电路和后加电路, 分别对应图 2 中(1)~(3). 通过预加电路并行计算 R_A 和 R_S , 如 $R_A = (A_H + A_L) \cdot \text{mod} 8192$ 表示 $(a_{255}, \dots, a_{128})$ 与 $(a_{127}, \dots, a_0)$ 对应位置

的系数相加, $(a_{255} + a_{127}) \text{mod} 8192$ 为其中 1 组预加电路, 共 128 组电路构成计算 A_j 的预加电路. 同理, R_S 的计算方式类似, 每个对应系数 s 相加, 但其结果的数值范围从 $[-4, 4]$ 变为 $[-8, 8]$, 对应图 5 中右边 S_j 的预加电路阵列. 预加电路输出的信号 R_A 和 R_S 将被传输至并行乘法电路 (MAC_{II}) 的输入端.

在并行乘法电路中, MAC_I 的输入从存储 A_H 、 A_L 、 S_H 和 S_L 的寄存器获取. 该单元用于计算 P_0 和 P_1 , 而 MAC_{II} 单元用于处理 R_A 和 R_S 的乘积, 所产生的信号 P_0 、 P_1 和 P_2 将被装载至后加电路的寄存器队列. 电路结构如图 5 所示, 包括 2 组 MAC_I 和 1 组 MAC_{II} . 该电路以压缩关键路径时长为目的, 利用各模块电路的并行执行策略来缩短运算的执行时间.

后加电路的功能为调整各自系数正负再累加. 该步骤对应于 Saber 算法运算内部的模约规则, 实现幂次的降幂化简, 如 ax^{256} 化简后为 $-ax^0$, ax^{257} 化简后为 $-ax^1$, 逐一降幂化简. 在并行乘法电路中, P_0 、 P_1 和 P_2 均为 255×13 bits, 其最高次幂为 254, 因此无需降幂化简. 但后加电路涉及降幂化简操作, 如图 2 中(3)的 P_0x^{256} 运算, 即 P_0x^{256} 中每一项的幂

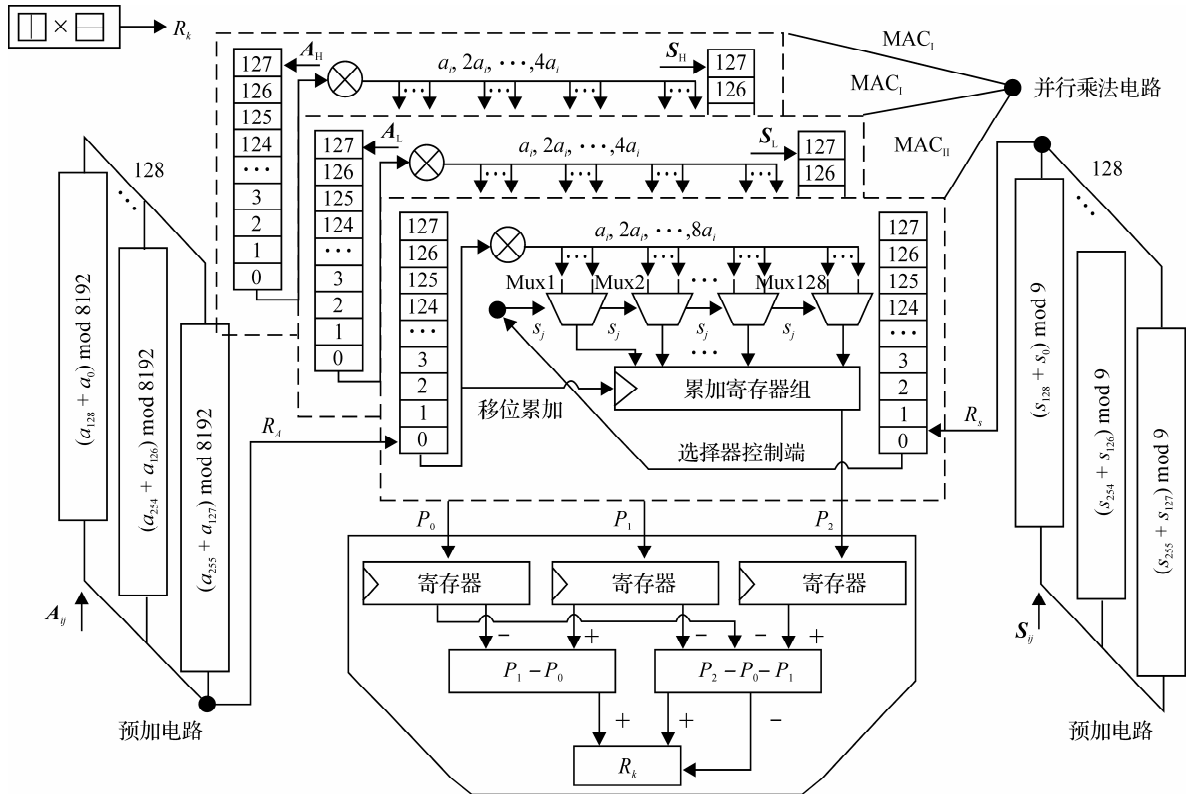


图 5 基于 Karatsuba 的多项式并行模乘电路

次均超过 255. 根据 Saber 算法以 P 为不可约多项式的模约规则, P_0 中所有系数均变为负号, 因此 $P_0x^{256} + P_1$ 经后加电路计算变为 $P_1 - P_0$. 此处的运算对应于同幂次系数间的计算, 共 256 项的并行运算; 同理 $(P_2 - P_0 - P)x^{128}$ 的降幂化简类似. 最终运算得结果 R_k , 其对应于图 1 算法中的输出.

3 实验结果与分析

使用 Modelsim 和 MATLAB 软件对设计的硬件电路进行仿真, 并在 TSMC 65 nm 工艺下, 利用 Synopsys 公司的 DC 软件进行电路的可综合性验证. 通过仿真平台提取电路的性能参数, 采用时序、功耗和面积作为评估指标.

3.1 并行乘法器的时序

时序信息反映电路的工作状态. 图 6 为并行乘

法器时序仿真波形, 分为初始化($80t_{clk}$)和相乘($137t_{clk}$)两个工作状态. 由于本文融合 Karatsuba 算法与压缩关键路径的策略可弥补核心算子时序路径过长的计算缺陷, 因此相乘的状态可分为 3 个计算阶段. 阶段 1, 预加状态(Pre-add)占用 $4t_{clk}$ 计算时长, 该并行预加状态(Parallelpre-Add)的波形分别对应于图 5 中 R_4 、 R_5 的计算电路; 阶段 2, 乘法状态占用 $131t_{clk}$ 计算时长; 阶段 3, 后加状态(Post add)以 $2t_{clk}$ 时长结束 R_K 的计算. 依据流水线的计算特点, 下一轮数据(Next data)的加载可在 $137t_{clk}$ 的相乘阶段同时进行. 因此可不考虑初始化的数据加载, 单次乘法所需计算周期为 $137t_{clk}$, 该并行运算方式将加速 Saber 算法中向量间的内积计算和矩阵-向量相乘.

表 1 为本文与不同文献在电路性能上的数据比

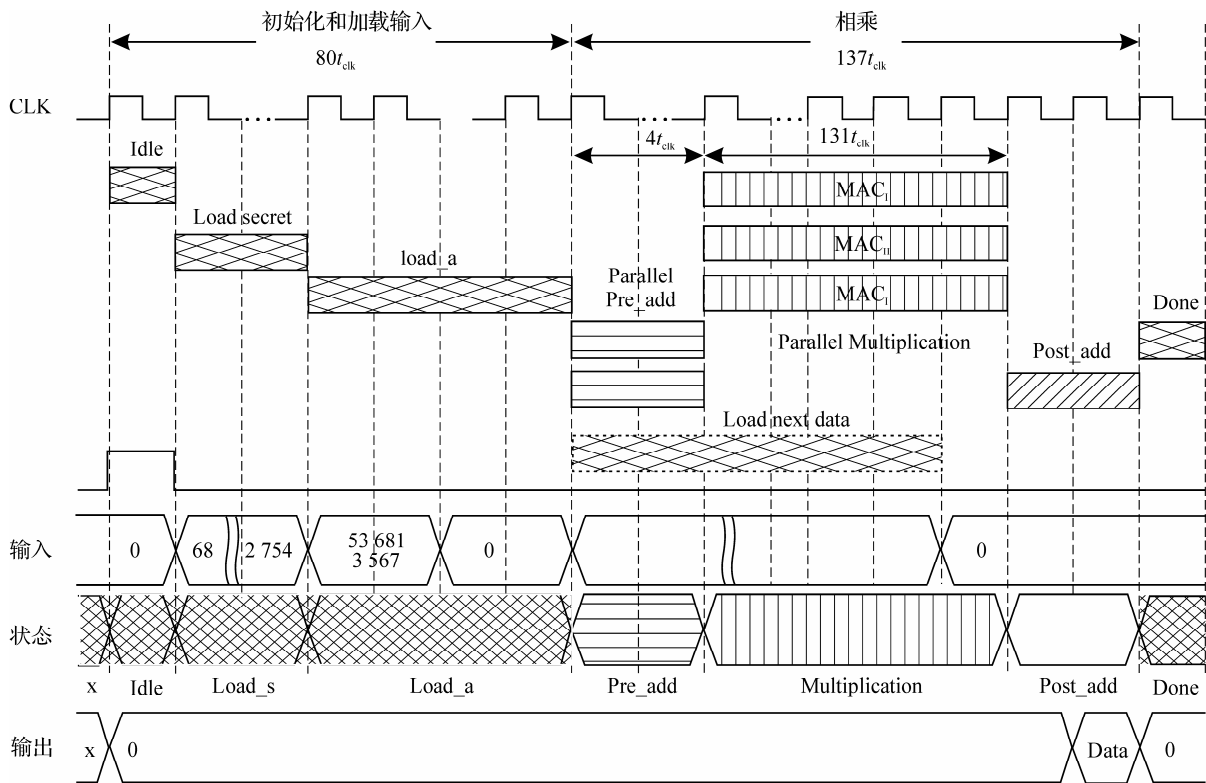


图 6 并行乘法器时序仿真波形

表 1 本文与相关文献的电路性能数据比较

文献	周期(t_{clk})	频率/MHz	面积/ μm^2	功耗/mW
文献[6]-High Speed I 256	256	250	137.47×10^3	13.78
文献[12]	256	250	178.38×10^3	14.41
本文	137	250	927.32×10^3	87.83

较. 在时钟频率为 250 MHz 条件下, R_K 运行所需的计算周期为 $137t_{clk}$, 较文献[6]-High Speed I 256、文献[12]均提速 46.50%. 式(16)表明该并行乘法器具有较高的速度性能.

$$\eta = \frac{t_{clk1} - t_{clk2}}{t_{clk1}} \times 100\% = \frac{256t_{clk} - 137t_{clk}}{256t_{clk}} \times 100\% \approx 46.50\%, \quad (16)$$

式中, t_{clk1} 和 t_{clk2} 分别代表对比文献的计算周期和本文乘法器的计算周期; t_{clk} 为单个时钟周期.

3.2 并行乘法器的面积功耗

由表 1 可知, 本文乘法器的面积开销为 $927.32 \mu m^2$, 较文献[6]-High Speed I 256、文献[12]大 5~7 倍; 功耗开销为 87.83 mW, 较文献[6]-High Speed I 256、文献[12]大 6~7 倍. 由此可知, Karatsuba 算法提升乘法器运算速度的同时, 也增加面积与功耗的开销.

图 7 为并行乘法器各模块面积占比和功耗占比. 由图可知, 预加电路面积和功耗占比均最低, 分别为 2.35% 和 1.90%; 在硬件电路中, 并行乘法电路面积和功耗占比均最高, 分别占 54.48% 和 41.48%. 另外由于 Karatsuba 算法的引入, 控制电路也占据 28.78% 的面积开销和 42.15% 的功耗开销. 控制部分的开销多数源自寄存器的消耗, 后加电路的面积和功耗分别占据 14.39% 和 14.48%. 综合分析, 面向 Saber 算法并行乘法器的硬件资源开销主要分布于控制电路和并行相乘电路.

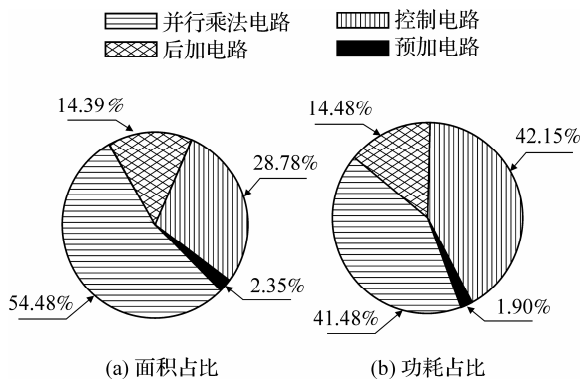


图 7 各模块面积占比和功耗占比

4 结论

通过对 Saber 算法内模多项式运算瓶颈和 Karatsuba 算法的分析, 在分解核心算子的运算机

制基础上结合并行运算电路, 实现面向 Saber 算法的并行乘法器设计. 本方案使用压缩关键路径的方式弥补核心算子时序路径过长的缺陷. 实验结果表明, 在工作频率 250 MHz 下, 并行乘法器的计算周期为 $137t_{clk}$, 功耗为 87.83 mW, 面积为 $927.32 \mu m^2$. 与常规计算方式相比, 该乘法器可减少 46.50% 的运算时间. 由于核心算子分布于 Saber 算法内部的密钥生成、加密和解密各个阶段, 所以采用并行压缩的流水线设计思想, 对提高 Saber 算法的硬件 IP 核性能具有理论研究意义和实际应用价值.

参考文献:

- [1] 刘冬生, 赵文定, 刘子龙, 等. 应用于格密码的可重构多通道数论变换硬件设计[J]. 电子与信息学报, 2022, 44(2):566-572.
- [2] Bernstein D J, Lange T. Post-quantum cryptography[J]. Nature, 2017, 549(7671):188-194.
- [3] Shor P W. Algorithms for quantum computation: Discrete logarithms and factoring[C]//Proceedings 35th Annual Symposium on Foundations of Computer Science. Santa Fe, NM, USA. IEEE, 1994:124-134.
- [4] Grover L K. A fast quantum mechanical algorithm for database search[C]//STOC '96: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing. 1996:212-219.
- [5] Basso A, Mera J M B, D'Anvers J P, et al. Saber: Mod-LWR based Kem (Round 3 Submission to NIST PQC)[R]. 2020.
- [6] Basso A, Roy S S. Optimized polynomial multiplier architectures for post-quantum KEM Saber[C]//2021 58th ACM/IEEE Design Automation Conference (DAC), 2021:1285-1290.
- [7] D'Anvers J P, Karmakar A, Vercauteren F, et al. Saber: module-LWR based key exchange, CPA-secure encryption and CCA-secure kem[C]//2018 International Conference on Cryptology in Africa (AFRICACRYPT), 2018:282-305.
- [8] Wang B, Gu X Z, Yang Y S. Saber on ESP32[C]//2020 18th International Conference on Applied Cryptography and Network Security (ACNS), 2020:421-440.
- [9] Paksoy I K, Murat C. TMVP-based multiplication for polynomial quotient rings and application to Saber on arm cortex-m4[J]. IACR Cryptology Eprint Archive, 2020, 2020:1302-1323.

- [10] Bermudo M J M, Karmakar A, Verbaudhede I. Time-memory trade-off in Toom-Cook multiplication: An application to module-lattice based cryptography[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2020(2):222-244.
- [11] Maria B M J, Turan F, Karmakar A, Roy S S, et al. Compact domain-specific co-processor for accelerating module lattice-based kem[C]//2020 57th ACM/IEEE Design Automation Conference. San Francisco, CA, USA. IEEE, 2020:1-6.
- [12] Sinha R S, Basso A. High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2020(4):443-466.

Design of parallel multiplier for Saber

LÜ Jie¹, WANG Pengjun^{2*}, ZHANG Huihong¹

(1.Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China;

2.College of Electrical and Electronic Engineering, Wenzhou University, Wenzhou 325035, China)

Abstract: With the development of the quantum computer, the security of modern cryptosystems will be placed under threat. In defending against quantum computing attacks, Saber algorithm becomes one of the schemes in the ongoing post-quantum cryptography standardization project. The bottleneck of Saber is multiplication in polynomial quotient rings, where the multiplication occupies high costs in computation operation. In this paper, a parallel multiplier for Saber is proposed, which is based on analysis of Karatsuba and Schoolbook. Firstly, disassembling the calculated path of polynomial modulo multiplication is carried out by the Karatsuba algorithm, whereby a strategy of combining multiplication reused with addition superseding multiplication is utilized to reduce hardware overhead. Secondly, the critical path is compressed by a parallel structure. Finally, in the TSMC 65 nm process, analysis is conducted using the tools of Modelsim and DC. The results indicate that the designed scheme accomplishes 137 clock cycles' critical path of an operation, improving 46.5% in speed performance as compared with the conventional method, the power consumption is 87.83 mW and the area overhead is $927.32 \times 10^3 \mu\text{m}^2$.

Key words: post-quantum cryptography; Saber algorithm; Karatsuba algorithm; Schoolbook multiplication method; parallel multiplier

(责任编辑 章践立)