可重构的素域 SM2 算法优化方法

李斌[1]，周清雷[1]，陈晓杰[2]，冯峰[1]
（1. 郑州大学计算机与人工智能学院，河南 郑州 450001；
2. 数学工程与先进计算国家重点实验室，河南 郑州 450001）

摘 要：针对 SM2 算法软件效率低、硬件实现资源利用率低、可扩展性差的问题，提出了一种可重构的素域 SM2 算法优化方法。通过对 SM2 算法的深入分析，从不同计算阶段和计算特点着手，分别采用 KOA 快速乘法、快速模约减和 Barrett 算法实现推荐或任意参数的模乘运算，并优化改进基为 4 的扩展欧几里得算法加速模逆运算。然后，在标准射影坐标系下以蒙哥马利方法提高点乘运算效率，并优化了点加和倍点数据流，将运算周期缩短至 12 个时钟。同时，在 FPGA 内部实现了快速的坐标系转换。最后，设计实现了多 SM2 的并行调度管理，满足日益多样化的应用需求。实验结果分析表明，所优化的 SM2 充分利用了 FPGA 的资源，缩短了点乘周期，每秒计算次数最多较 CPU（Intel i5-8300）高 352.48 倍，提高了计算性能和可扩展性。

关键词：可重构；SM2；FPGA；蒙哥马利点乘；快速模乘
中图分类号：TP309
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2022043

Optimization of reconfigurable SM2 algorithm over prime filed

LI Bin[1], ZHOU Qinglei[1], CHEN Xiaojie[2], FENG Feng[1]
1. School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China
2. State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

Abstract: Aiming at the problems of inefficient of software, low utilization of hardware resources and poor scalability of SM2 algorithm, a reconfigurable optimization method of SM2 algorithm over prime filed was proposed. Through in-depth analysis of the SM2 algorithm, starting from different computation stages and characteristics, respectively using KOA fast multiplication, fast modular reduction and Barrett algorithm to achieve recommended or arbitrary parameters of the modular multiplication operation, and the radix-4 extended Euclidean algorithm was optimized and improved to accelerate the modular inverse operation. Then, in the standard projective coordinate system, the Montgomery method was used to improve the efficiency of point multiplication, and the data flow of point addition and double point was optimized to shorten the operation cycle to 12 clocks. At the same time, fast coordinate system conversion was realized inside the FPGA. Finally, the parallel scheduling management of multi-SM2 was designed and implemented to meet the computational requirements of multiple applications. The experimental results show that the optimized SM2 makes full use of FPGA resources and shortens the cycle of point multiplication. The maximum number of calculations per second is 352.48 times higher than the CPU (Intel i5-8300), which improves the performance and scalability.

Keywords: reconfigurable, SM2, FPGA, Montgomery point multiplication, fast modular multiplication

收稿日期：2021−11−16；**修回日期**：2022−02−08
基金项目：国家重点研发计划基金资助项目（No.2016YFB0800100, No.2016YFB0800101）；国家自然科学基金资助项目（No.61572444）
Foundation Items: The National Key Research and Development Program of China (No.2016YFB0800100, No.2016YFB0800101), The National Natural Science Foundation of China (No.61572444)

0　引言

椭圆曲线密码（ECC, elliptic curve cryptography）算法作为公钥密码学中的一个研究重点，在性能、安全方面都被证明有很大的优势，在许多领域得到了广泛的应用。SM2 算法由国家密码管理局发布，是基于 ECC 的公钥密码算法，旨在替换 RSA 算法。SM2 算法作为一种国密算法，已在电子认证系统、密钥管理系统等多种商用密码产品中得到应用。

SM2 的实现依赖于椭圆曲线群上的点加、倍点和点乘的运算效率，其计算量大、结构复杂，存在算法效率不高、资源消耗较多等问题。为此，国内外众多学者对 ECC 算法进行了大量的研究，包括使用 FPGA（field-programmable gate array）可重部件的优化[1-3]和 ASIC（application specific integrated circuit）专用芯片的设计[4-6]。但仅优化了推荐参数 SM2 的计算，可扩展并行能力差，难以满足日益多样化的应用需求。同时，通过硬件实现的 SM2 在运算过程中由于电磁辐射和功耗，会不可避免地泄露部分侧信道信息，容易受到能量分析攻击的威胁[7]。因此，在保证 SM2 执行效率的同时提高密码系统的安全性具有十分重大的意义。

由此，本文提出了一种可重构的素域 SM2 算法优化方法，通过对 SM2 算法的深入剖析，结合 KOA（Karatsuba-Ofman algorithm）乘法、快速模约减、Barrett 模约减、基 4 求模逆、蒙哥马利点乘、点加和倍点等多种优化方法，利用 FPGA 的可重构特性，实现了高能效和抗攻击的 SM2 算法。同时，自顶向下设计了 SM2 并行架构，可并行执行多个 SM2 算法，满足各种应用场景的计算需求。

1　SM2 椭圆曲线算法

1.1　参数定义

SM2 算法包括素域和二元扩域 2 种基域，对于素域，SM2 算法在素域 F_p 上的椭圆曲线方程为

$$y^2 = x^3 + ax + b \tag{1}$$

其中，$a,b \in F_p$ 且 $(4a^3 + 27b^2) \bmod p \neq 0$，$p$ 为素数。

具体地，国家密码管理局给出的参数为

P_{SM2}=FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFF

a=FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFC

b=28E9FA9E 9D9F5E34 4D5A9E4B CF6509A7 F39789F5 15AB8F92 DDBCBD41 4D940E93

经推导，可知

$$P_{SM2} = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1 \tag{2}$$

$$a = P_{SM2} - 3 \tag{3}$$

1.2　运算规则

在素域中，对于椭圆曲线上的点 P、Q 和无穷远点 O，有以下运算规则。

1) $P + O = P$。

2) 如果 $P = (x_1, y_1)$，则 $-P = (x_1, -y_1)$，且 $P + (-P) = O$。

3) 如果 $Q = (x_2, y_2)$，且 $Q \neq -P$，那么 $P + Q = (x_3, y_3)$，并有

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \tag{4}$$

其中

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & x_1 \neq x_2 \\ \dfrac{3x_1^2 + a}{2y_1}, & x_1 = x_2 \end{cases} \tag{5}$$

当 $P \neq Q$ ($x_1 \neq x_2$)时，$P + Q$ 称为椭圆曲线上的点加运算；当 $P = O$ ($x_1 = x_2$)时，$P + Q = 2P$ 称为椭圆曲线上的倍点运算。

点乘，也称多倍点运算，是决定椭圆曲线密码体制运算速度的关键。椭圆曲线的点乘运算是指曲线上的某一基点 P 与一个整数 k 进行乘法运算，也就是 k 个 P 相加，即

$$kP = \underbrace{P + P + \cdots + P}_{k\text{个}} \tag{6}$$

由以上分析可以看出，对点加、倍点和点乘的优化是提高 SM2 计算效率的重要手段。

1.3　算法分析

从 SM2 的定义和运算描述中可以看出，它以大数模运算为基本运算，运算量大、耗时长，模运算的效率直接决定了 SM2 的计算速度。其次，不同坐标系下椭圆曲线的计算式也会发生变化，坐标系的选择和计算式的优化对 SM2 的计算速度也有很大的影响。最后，如何利用 FPGA 实现高效的、可扩展的和适用的 SM2 也面临严峻的挑战。

为此，国内外学者对椭圆曲线的优化加速展开了广泛的研究。文献[8]设计了 4 个模乘模块，可并

行计算点乘的运算。文献[9]在 Jacobian 坐标系下，利用交叉模乘，减少了资源占用和时延。文献[10]在 FPGA 上优化实现了 FourQ 椭圆曲线，并设计了单核和多核结构。文献[11]提出了快速模逆算法，采用基为 8 的扩展欧几里得算法减少了计算周期。文献[12]结合 Karatsuba、快速模约减和蒙哥马利点乘实现了美国国家标准与技术研究院（NIST, National Institute of Standards and Technology）素域的 ECC 算法。文献[13]采用基为 4 的交叉模乘和最大公约数求模除优化了 SM2 素域和二元扩域算法。文献[14]设计了多种素域乘法运算，包括 Booth 乘法、Moore 乘法等，并结合快速模约减实现了模乘运算。文献[15]使用 Toom-Cook 算法减少了椭圆曲线中模乘的运算量。文献[16]结合多种算法和优化技术实现了支持一般参数的蒙哥马利和 Weierstrass 椭圆曲线算法。文献[17]在雅可比坐标系下，使用基为 2 的交叉模乘及蒙哥马利点乘优化了素域 ECC 算法。但是，SM2 的计算复杂度和具体的实现结构、坐标系、点乘算法、模乘算法、模逆算法等紧密相关，一部分方案仅优化了模乘和模逆算法；另一部分方案仍采用基为 2 的低基数实现方式；其他方案未涉及多模块并行的架构设计。另外，大多数方案仅考虑了推荐参数（国家密码管理局和 NIST 推荐的椭圆曲线参数）的椭圆曲线，并未涉及任意参数的实现。同时，大多数方案并未在 FPGA 上实现坐标系的转换。

由以上分析可以看出，要提高 SM2 的 FPGA 实现速度，一是要对关键路径进行分解，深度优化，提高计算效率；二是要增加并行度，包括各模块间的并行和同时处理数据的能力。为此，需要从 SM2 的各个计算阶段着手分析，结合 FPGA 可重构的特性，针对不同的运算、存储和通信特征选择合适的实现方式。本文选择标准射影坐标系，采用蒙哥马利算法实现点乘，并利用 KOA 乘法、快速模约减、模加减实现点加和倍点运算，以极大提高 SM2 算法速度。其中，KOA 乘法采用 DSP 实现，快速模约减、模加减采用加法器和查找表实现。然后，实例化多个点加和倍点模块，两两一组形成点乘运算，从而实现 SM2 的多路并行计算。最后，利用 Barrett 算法实现任意参数的快速模乘运算，并使用基为 4 的扩展欧几里得算法实现模逆，完成坐标系的转换。

2 SM2 的优化设计

2.1 SM2 整体架构

SM2 算法整体架构如图 1 所示。该架构采用 CPU+FPGA 的方式实现，并通过 PCIe 进行数据的传输。其中，CPU 主要完成信息的配置、中间状态的查询和最终结果的显示；FPGA 主要完成 SM2 的加速计算，其核心模块包括点乘、点加、倍点和坐标转换。通过 CPU 重构 FPGA 完成电路逻辑配置，再以软硬件协同方式实现 SM2 的加密、解密、签名、验签和密钥交换等功能。

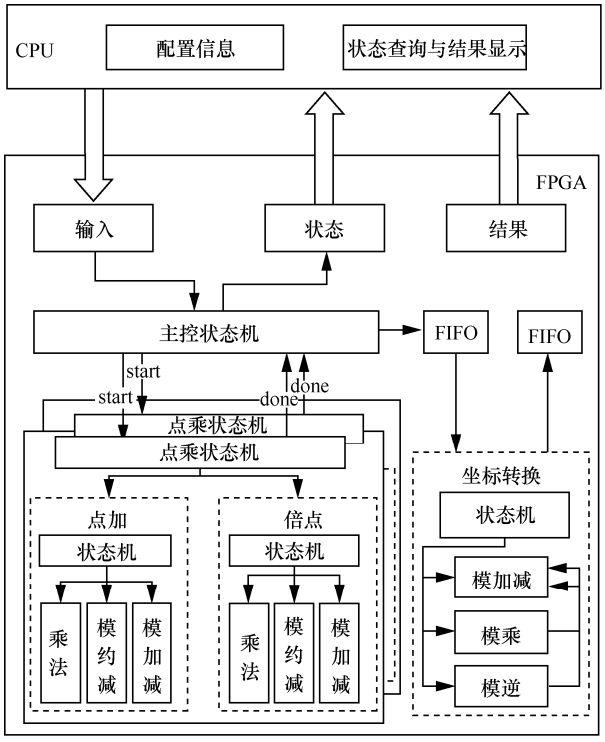

图 1 SM2 算法整体架构

在计算 SM2 点乘时，会多次调用点加和倍点运算，而坐标转换只在最后计算一次，因此本文对点加和倍点以性能最优进行优化，而对坐标转换以资源最优进行优化。其次，由主控状态机对多个点乘模块进行调度管理，实现 SM2 的并行处理，满足不同应用场合的计算需求。最后，各点乘模块共享模加减模块，实现资源的复用，并通过异步 FIFO 完成数据的传输，以减少 FPGA 资源的消耗。

由此可见，本文通过优化底层各种模运算，再以并行和资源共享的方式实现多路点乘，完成了 SM2 的 FPGA 可重构优化设计，支持多种参数的动态配置。此外，本文以 CPU 和 FPGA 搭建了异构架构，通过并行调度和协同计算，并以重构 FPGA 提供高并发和多任务处理阵列，保证了高效能密码运算的业务需求。

2.2　快速模乘

快速模乘采用 KOA 乘法和快速模约减实现，先由 KOA 算法完成 2 个大数的相乘，再由快速模约减算法完成结果的取模运算。其中，KOA 算法和快速模约减算法计算周期分别为一个时钟，则快速模乘可在 2 个时钟内完成整个运算。

2.2.1　KOA 快速乘法

KOA[18]算法的核心思想是“分而治之”，采用递归的方式将一个复杂的乘法运算分解成多个简单的乘法运算，较传统计算速度更快、效率更高。对于 2 个 n 位数，如果直接相乘，其复杂度为 $O(n^2)$；如果使用 KOA 算法，可以将复杂度降低到 $O(n^{\mathrm{lb}3})$。

对于 n 位数 A，可将其表示为 $A=(\underbrace{\alpha_{n-1},\alpha_{n-2},\cdots,\alpha_{\frac{n}{2}}}_{A^{\mathrm{H}}},\underbrace{\alpha_{\frac{n}{2}-1},\cdots,\alpha_0}_{A^{\mathrm{L}}})$，其中 $\alpha_i\in\{0,1\}$，$0\leqslant i<n$。则数 A、B 可等价表示为

$$A=A^{\mathrm{H}}\times 2^{\frac{n}{2}}+A^{\mathrm{L}} \tag{7}$$

$$B=B^{\mathrm{H}}\times 2^{\frac{n}{2}}+B^{\mathrm{L}} \tag{8}$$

那么 $C=A\times B$ 可表示为

$$C=(A^{\mathrm{H}}\times 2^{\frac{n}{2}}+A^{\mathrm{L}})\times(B^{\mathrm{H}}\times 2^{\frac{n}{2}}+B^{\mathrm{L}})=$$

$$C_2\times 2^n+C_1\times 2^{\frac{n}{2}}+C_0 \tag{9}$$

其中，$C_2=A^{\mathrm{H}}\times B^{\mathrm{H}}$，$C_1=A^{\mathrm{H}}\times B^{\mathrm{L}}+A^{\mathrm{L}}\times B^{\mathrm{H}}$，$C_0=A^{\mathrm{L}}\times B^{\mathrm{L}}$。而 C_1 又可表示为 $C_1=(A^{\mathrm{H}}+A^{\mathrm{L}})\times(B^{\mathrm{H}}+B^{\mathrm{L}})-C_2-C_0$，由此，整个算法只需要计算乘法 $A^{\mathrm{H}}\times B^{\mathrm{H}}$、$A^{\mathrm{L}}\times B^{\mathrm{L}}$ 和 $(A^{\mathrm{H}}+A^{\mathrm{L}})\times(B^{\mathrm{H}}+B^{\mathrm{L}})$，从而降低了复杂度。

对于 SM2，其参数为 256 位，而 FPGA DSP 支持最大位宽为 64 位的乘法运算，那么可以将 256 位分割成 128 位，再将 128 位分割成 64 位，经过两次递归运算，得到最终的结果。另外，$C_2\times 2^n$ 和 $C_1\times 2^{\frac{n}{2}}$ 可直接通过向左移位（<<）完成，如算法 1 所示。

算法 1　KOA 乘法

输入　A, B, n //n 为位宽

输出　C

1)　if(n == 64) then return $C=A\times B$;

2)　end if

3)　$A=A^{\mathrm{H}}\times 2^{\frac{n}{2}}+A^{\mathrm{L}}$;

4)　$B=B^{\mathrm{H}}\times 2^{\frac{n}{2}}+B^{\mathrm{L}}$;

5)　$C_2=\mathrm{KOA}\left(A^{\mathrm{H}},B^{\mathrm{H}},\frac{n}{2}\right)$;

6)　$C_1'=\mathrm{KOA}\left(A^{\mathrm{H}}+A^{\mathrm{L}},B^{\mathrm{H}}+B^{\mathrm{L}},\frac{n}{2}\right)$;

7)　$C_0=\mathrm{KOA}(A^{\mathrm{L}},B^{\mathrm{L}},\frac{n}{2})$;

8)　$C_1=C_1'-C_2-C_0$;

9)　$C=C_2<<n+C_1<<\left(\frac{n}{2}\right)+C_0$;

为进一步优化 KOA 算法在 FPGA 上的实现，本文将 64 位 DSP 乘法时钟周期设置为 0，并通过 wire 类型变量将各个模块互连。当 256 位数据输入时，可立即计算出结果，并由寄存器缓存输出，整个计算过程为一个时钟。128 位 KOA 乘法的 FPGA 硬件结构如图 2 所示。

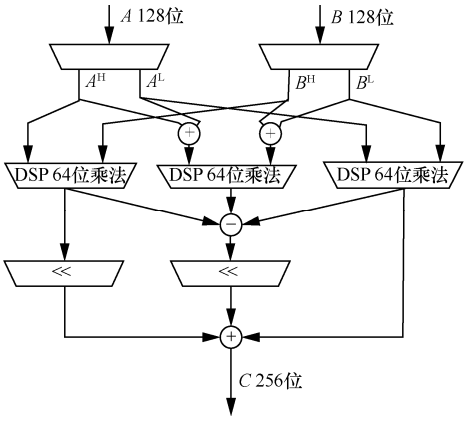

图 2　128 位 KOA 乘法的 FPGA 硬件结构

对于 256 位 KOA 乘法，与图 2 结构相同，只需将 DSP 64 位乘法替换为 128 位 KOA 乘法即可。这样，通过第一层调用 256 位 KOA 算法，第二层计算 128 位 KOA 乘法，第三层调用 DSP 并计算 64 位乘法，逐层完成计算。

2.2.2　快速模约减

对于素数域 SM2 算法，当 P_{SM2} 给定时，可以通过多次模约减操作替代除法操作，从而快速求得取模的结果，称为快速模约减[19]。

设 $c=a\times b$ 且 $0\leqslant a<P_{\mathrm{SM2}}$，$0\leqslant b<P_{\mathrm{SM2}}$，则 $0\leqslant c\leqslant P_{\mathrm{SM2}}^2$，于是 c 可以表示为

$$c=c[15]\times 2^{480}+c[14]\times 2^{448}+c[13]\times 2^{416}+$$
$$c[12]\times 2^{384}+c[11]\times 2^{352}+c[10]\times 2^{320}+$$

$$c[9]\times 2^{288}+c[8]\times 2^{256}+c[7]\times 2^{224}+ c[6]\times 2^{192}+c[5]\times 2^{160}+ c[4]\times 2^{128}+ c[3]\times 2^{96}+c[2]\times 2^{64}+c[1]\times 2^{32}+c[0] \quad (10)$$

其中，$c[i]\in[0,2^{32}),0\leqslant i\leqslant 15$。

定义 256 位数组 s 为

$s[0]=\{c[15], 0, c[15], c[14], c[13], 0, c[15], c[14]\}$

$s[1]=\{c[14], 0, 0, 0, 0, 0, c[14], c[13]\}$

$s[2]=\{c[13], 0, 0, 0, 0, 0, 0, c[15]\}$

$s[3]=\{c[12], 0, 0, 0, 0, 0, 0, 0\}$

$s[4]=\{c[15],c[15], c[14], c[13], c[12], 0, c[11], c[10]\}$

$s[5]=\{c[11], c[14], c[13], c[12], c[11], 0, c[10], c[9]\}$

$s[6]=\{c[10], c[11], c[10], c[9], c[8], 0, c[13], c[12]\}$

$s[7]=\{c[9], 0, 0, c[15], c[14], 0, c[9], c[8]\}$

$s[8]=\{c[8], 0, 0, 0, c[15], 0, c[12], c[11]\}$

$s[9]=\{c[7], c[6], c[5], c[4], c[3], c[2], c[1], c[0]\}$

$s[10]=\{0, 0, 0, 0, 0, c[8], 0, 0\}$

$s[11]=\{0, 0, 0, 0, 0, c[9], 0, 0\}$

$s[12]=\{0, 0, 0, 0, 0, c[13], 0, 0\}$

$s[13]=\{0, 0, 0, 0, 0, c[14], 0, 0\}$

则 $\text{sum}=(s[0]+s[1]+s[2]+s[3])\times 2+s[4]+s[5]+s[6]+s[7]+s[8]+s[9]-s[10]-s[11]-s[12]-s[13]$，那么 $c \bmod P_{SM2}=\text{sum} \bmod P_{SM2}$，显然快速模约减将模除运算变为模加减运算，从而大大降低了算法的复杂度。

进一步，对数组 s 的运算进行观察分析可以看出，$c[8]+c[9]+c[10]+c[11]$被执行了 3 次，$c[12]+c[13]+c[14]+c[15]$被执行了 6 次。因此，可以对频次高的组合数值进行预计算，以此简化后续计算步骤。这里，令 $r_0=c[15]+c[14]$，$r_1=r_0+c[13]$，$r_2=r_1+c[12]$，$r_3=(c[11]+c[10]+c[9])+c[8]$，$r_4=c[13]+c[8]$，$r_5=c[14]+c[9]$，$r_6=r_2+r_3$，则 sum 的计算可由 117 个 8 位加法器在一个时钟内完成。而未优化时，则需要 13×32 = 416 个加法器。

同时，sum 最大不超过 $14P_{SM2}$，可以通过预计算提前给出 kP_{SM2} $(0\leqslant k\leqslant 15)$ 的值，然后根据 sum 的高 4 位选择对应的 kP_{SM2}，并使用 2 次减法完成模约减。然而，在减法过程中可能产生溢出，这里以 carry_1 和 carry_2 为溢出标志位，并根据 carry_1 是否产生溢出而选择 result_1 或 result_2 作为结果，具体如算法 2 所示。

算法 2 快速模约减优化

输入 sum[259:0], P_{SM2}, kP_{SM2}

输出 c_reduce

1) k = sum[259:256];

2) 根据 k 选择对应的 kP_{SM2};

3) $\{\text{carry}_1, \text{result}_1[259:0]\}=\text{sum}-kP_{SM2}$;

4) $\{\text{carry}_2, \text{result}_2[259:0]\}=\text{result}_1-P_{SM2}$;

5) c_reduce = carry_1? result_2[255:0] : result_1[255:0];

2.2.3 Barrett 模乘

Barrett 算法[20]利用乘法和模约减运算代替了高成本的除法来实现取模运算，可计算任何参数的模乘。对于 $0\leqslant a<P_{SM2}$，$0\leqslant b<P_{SM2}$，则 $a\times b<P_{SM2}^2$。为计算 $c=a\times b \bmod P_{SM2}$，$0\leqslant c<P_{SM2}$，令 $d=a\times b$，如果存在 e 使 $d=e\times P_{SM2}+c$，则可求得 c。

Barrett 算法首先计算 e 的近似值 e'，令 $\mu=\left\lfloor\frac{2^{2n}}{P_{SM2}}\right\rfloor$，$e'=\left\lfloor\frac{d}{P_{SM2}}\right\rfloor$，则

$$e'=\left\lfloor\frac{d}{P_{SM2}}\right\rfloor\approx\left\lfloor d\frac{\mu}{2^{2n}}\right\rfloor\approx\left\lfloor\left\lfloor\frac{d}{2^n}\right\rfloor\frac{\mu}{2^n}\right\rfloor \quad (11)$$

其中，$n=\lceil \text{lb}P_{SM2}\rceil$。对于 e'的计算，由除法转变为了乘法，且 $\frac{d}{2^n}$ 和 $\frac{\mu}{2^n}$ 可以通过向右移位（>>）的方式实现，大大降低了计算量。而对于μ的计算，可以使用软件提前计算出来，例如当模数 P_{SM2} 为 FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409 39D54123 时，对应的 μ 值为 1 00000001 00000001 00000001 00000001 8DFC2096 FA323C01 12AC6361 F15149A0。在大部分应用中，一旦选定模数 P_{SM2}，其在计算过程中就保持不变，因此μ只需要计算一次即可。最后，对于结果 c，其取值范围为$[0, 2P_{SM2})$，因此对 c 与 P_{SM2} 进行比较判断，并计算输出最终结果。Barrett 模约减整体流程如算法 3 所示。

算法 3 Barrett 模约减算法

输入 a, b, P_{SM2}, $\mu=\left\lfloor\frac{2^{2n}}{P_{SM2}}\right\rfloor$

输出 $c=a\times b \bmod P_{SM2}$

1) $d=a\times b$;

2) $e_1=\mu[n-1:0]\times(d>>n)$;

3) $e_2=(\mu[n]==1\text{'b1})?\{(d>>n),256\text{'b0}\}:512\text{'b0}$;

4) $e'=(e_1+e_2)>>n$;

5) $d'=P_{SM2}\times e'$;

6) $c=d-d'$;

7) if($c>P_{SM2}$) then $c=c-P_{SM2}$;

8) end if

9) return c;

在算法 3 中，μ可能为 257 位，因此最高位$\mu[n]$也要参与计算，步骤 2)～步骤 4)为 e'的乘法及向下取整操作。显然对于 Barrett 模约减，必须借助高效的大数乘法才能发挥其优势。因此，算法 3 中的步骤 1)、步骤 2)和步骤 5)均采用 KOA 乘法快速实现，对应的硬件结构如图 3 所示。

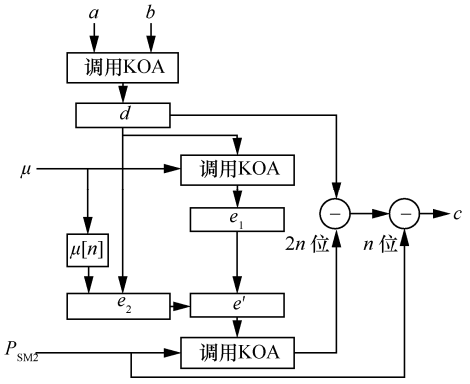

图 3　Barrett 模约减结构

从图 3 中可以看出，Barrett 模约减计算周期为 $T=3T_{\mathrm{KOA}}+2T_{\mathrm{SUB}}+T_{\mathrm{ADD}}$，且关键路径为 KOA 算法。

2.3　扩展欧几里得求模逆

对于素数 $a\in[1,P_{\mathrm{SM2}}-1]$，如果存在 a^{-1}，使 $a\times a^{-1}\equiv 1 \bmod P_{\mathrm{SM2}}$，则$a^{-1}$称为 a 的模逆元。通常在素数域上求模逆的算法有扩展欧几里得、费马小定理和蒙哥马利算法。虽然，扩展欧几里得算法采用辗转相除法，但可以根据公约数的性质，将除法全部改成加减运算，并以二进制移位完成除 2 运算，有利于硬件实现。而费马小定理需要求 a 的次幂，蒙哥马利算法与扩展欧几里得算法原理较相似，但需要将结果从蒙哥马利域转到实数域，计算效率较低。

2.3.1　基 4 的模逆优化

为更具一般性，计算$\frac{b}{a} \bmod P_{\mathrm{SM2}}$，且当 b=1 时，相当于求 a 的模逆元a^{-1}。具体使用 2 个等式

$$u\times b\equiv w_1\times a \bmod P_{\mathrm{SM2}} \tag{12}$$

$$v\times b\equiv w_2\times a \bmod P_{\mathrm{SM2}} \tag{13}$$

其中，u、v、w_1、w_2 分别初始化为 a、P_{SM2}、b、0。然后用扩展欧几里得算法对 u 和 v 进行计算，同时对 w_1 和 w_2 进行相应的线性变换，确保式(12)和式(13)保持成立。最后，当 $v=0$ 时停止迭代，此时有 $u=1$，w_1 即 $\frac{b}{a} \bmod P_{\mathrm{SM2}}$ 所求的结果。

另外，以四进制表示，采用状态机循环控制，对该算法进行优化。在每次迭代中，根据 u 和 v 的最后两位对其进行判断计算。当最后两位为 2'b00 或 2'b10，即 u 和 v 为偶数时，进行除以 4 或 2 的操作。当最后两位相同时，将 u 和 v 相减，并进行除以 4 的操作；否则，将 u 和 v 相减，并进行除以 2 的操作。具体流程如算法 4 所示。

算法 4　扩展欧几里得模逆算法

输入　a, b, P_{SM2}

输出　$c=\frac{b}{a} \bmod P_{\mathrm{SM2}}$

1) $u=a$; $v=P_{\mathrm{SM2}}$; $w_1=b$; $w_2=0$;
2) while($v>0$) do
3)　if (u[1:0]==2'b00) then
4)　　$u=u>>2$;　$w_1=\frac{w_1}{4} \bmod P_{\mathrm{SM2}}$;
5)　else if(v[1:0]==2'b00) then
6)　　$v=v>>2$;　$w_2=\frac{w_2}{4} \bmod P_{\mathrm{SM2}}$;
7)　else if(u[1:0]== v[1:0]) then
8)　　if($u>v$) then
9)　　$u=(u-v)>>2$;　$w_1=\frac{w_1-w_2}{4} \bmod P_{\mathrm{SM2}}$;
10)　　else　$v=(v-u)>>2$;　$w_2=\frac{w_2-w_1}{4} \bmod P_{\mathrm{SM2}}$;
11)　　end if
12)　else if(u[1:0]==2'b10) then
13)　　if ((u>>1) > v) then
14)　　$u=((u>>1)-v)>>1$;　$w_1=\frac{\frac{w_1}{2}-w_2}{2} \bmod P_{\mathrm{SM2}}$
15)　　else　$u=u>>1$;　$w_1=\frac{w_1}{2} \bmod P_{\mathrm{SM2}}$;　$v=(v-(u>>1))>>1$　$w_2=\frac{w_2-\frac{w_1}{2}}{2} \bmod P_{\mathrm{SM2}}$;
16)　　end if
17)　else if(v[1:0]==2'b10) then
18)　　if($u>(v>>1)$) then
19)　　$u=(u-(v>>1))>>1$;　$w_1=\frac{w_1-\frac{w_2}{2}}{2} \bmod P_{\mathrm{SM2}}$　;　$v=v>>1$　;　$w_2=\frac{w_2}{2} \bmod P_{\mathrm{SM2}}$
20)　　else　$v=((v>>1)-u)>>1$;

$$w_2 = \frac{\frac{w_2}{2} - w_1}{2} \bmod P_{\mathrm{SM2}};$$

21)　end if
22)　else if($u \geqslant v$) then
23)　$u = (u - v) >> 1$;　$w_1 = \frac{w_1 - w_2}{2} \bmod P_{\mathrm{SM2}}$;
24)　else if
25)　$v = (v - u) >> 1$; $w_2 = \frac{w_2 - w_1}{2} \bmod P_{\mathrm{SM2}}$;
26)　end if
27) end while
28) return $c = w_1$;

在算法 4 中，步骤 1)为初始化状态；步骤 2)为循环判断状态；步骤 3)～步骤 26)分别为各种判断和计算；步骤 28)为输出。其次，对于 u 和 v 的计算始终保证其结果值小于 P_{SM2}，不需要额外处理。而 w_1 和 w_2 这 2 个数的加减可能大于 P_{SM2} 或者溢出，且除以 2 和 4 需要额外进行判断并调用模加减来处理。具体的计算式为

$$\frac{x}{2} \bmod P_{\mathrm{SM2}} = \begin{cases} x >> 1 & , x[0] == 1'\mathrm{b0} \\ x >> 1 + P_{\mathrm{SM2}} >> 1 + 1 & , x[0] == 1'\mathrm{b1} \end{cases} \tag{14}$$

$$\frac{x}{4} \bmod P_{\mathrm{SM2}} = \begin{cases} x >> 2 & , x[1:0] == 2'\mathrm{b00} \\ (x >> 1 + P_{\mathrm{SM2}} >> 1 + 1) >> 1, & x[1:0] == 2'\mathrm{b01} \\ x >> 2 + P_{\mathrm{SM2}} >> 1 + 1 & , x[1:0] == 2'\mathrm{b10} \\ (x >> 1 + P_{\mathrm{SM2}} >> 1 + 1) >> 1 + P_{\mathrm{SM2}} >> 1 + 1, & \\ x[1:0] == 2'\mathrm{b11} & \end{cases} \tag{15}$$

2.3.2　模加减优化

为求模逆元，需要用到模加减操作，这里将其合并在一个模块中实现，以减少资源占用。该模块根据模加减模式选择不同的初值进行计算，其中模加的结果可能超过 P_{SM2}，因此结果需要再减去 P_{SM2}；而模减的结果可能为负数，因此可先加 P_{SM2} 再做减法，并根据溢出标志位判断最终的输出结果，如算法 5 所示。

算法 5　模加减
输入　a, b, P_{SM2}, SEL
输出　$c = a \pm b \bmod P_{\mathrm{SM2}}$
1) 若 SEL 为模加，则 $a_1 = a$; $b_1 = b$; $m = P_{\mathrm{SM2}}$;
2) 若 SEL 为模减，则 $a_1 = a$; $b_1 = P_{\mathrm{SM2}}$; $a_2 = a$; $b_2 = b$; $m = b$;
3) $\{c_0, s_0\} = a_1 + b_1$;
4) $\{c_1, s_1\} = s_0 - m$;
5) $\{c_2, s_2\} = a_2 - b_2$;
6) 若 SEL 为模加，if(c_1==1'b1) then $c = s_0$; else $c = s_1$;end if
7) 若 SEL 为模减，if(c_2==1'b1) then $c = s_1$; else $c = s_2$; end if

算法 5 中，SEL 为模式选择，根据模加和模减对 a_1、b_1、a_2、b_2 和 m 赋不同的初值，并通过步骤 3)～步骤 5)完成计算，其中 c_0、c_1、c_2 为溢出标志位，可根据 c_1、c_2 的值确定最终的计算结果。

2.4　点乘优化

2.4.1　蒙哥马利点乘

点乘算法有二进制展开扫描法、NAF（none adjacent form）扫描法、NAF 滑动窗口法及蒙哥马利点乘算法[21]。目前，蒙哥马利点乘算法是实现效率最高和应用范围最广的算法。蒙哥马利点乘算法首先根据基点 G，计算初始值 R_0 和 R_1，$R_0 = G$，$R_1 = 2G$；然后根据整数 k 的二进制编码的每一位对 R_0 和 R_1 进行点加和倍点运算；最后根据 k 的二进制位数进行循环计算，并求得最终点乘结果。其具体运算过程如算法 6 所示。

算法 6　蒙哥马利点乘
输入　$k = (k_{l-1}, \cdots, k_0)$，点 G
输出　$Q = kG$
1) 初始化 $R_0 = G$; $R_1 = 2G$; $i = l - 2$;
2) while($i \geqslant 0$) do
3)　if(k_i==0) then $R_1 = R_0 + R_1$; $R_0 = 2R_0$;
4)　else if(k_i==1) then $R_0 = R_0 + R_1$; $R_1 = 2R_1$;
5)　$i = i - 1$;
6)　end if
7) end while
8) $Q = R_0$;

从算法 6 中可以看出，无论 k_i 的取值如何，每一次的循环过程中都会计算点加和倍点，且两者相互独立，可并行执行。同时，由于点加和倍点同时计算，使点乘运算过程中泄露的功耗信息无律可循，可有效抵抗简单功耗攻击（SPA, simple power analysis）。

2.4.2　点加和倍点优化

F_p 上的椭圆曲线常用的坐标系有仿射坐标系、标准射影坐标系、Jacobian 加重射影坐标系和 LD（Lopez & Dahab）射影坐标系。不同的坐标系表示在计算点加和倍点时的运算效率不同。这里，选择标准射影坐标系完成计算，并在最后一步进行坐标系转换。同时，为提高点加和倍点的计算效率，蒙

哥马利点乘只需 x 坐标参与计算，并在最后一步计算 y 坐标，大大简化了计算过程。

在标准射影坐标系下，有点 $P(X_1, Y_1, Z_1)$和点 $Q(X_2, Y_2, Z_2)$，则点加和倍点的计算式[22-23]分别为

$$\begin{cases} X(P+Q)= \\ (X_1X_2 - aZ_1Z_2)^2 - 4bZ_1Z_2(X_1Z_2 + X_2Z_1) \\ Z(P+Q) = x_G(X_1Z_2 - X_2Z_1)^2 \end{cases} \tag{16}$$

$$\begin{cases} X(2P) = (X_1^2 - aZ_1^2)^2 - 8bX_1Z_1^3 \\ Z(2P) = 4Z_1(X_1^3 + aX_1Z_1^2 + bZ_1^3) \end{cases} \tag{17}$$

最后，将结果转换为仿射坐标系，有

$$x_1 = \frac{X_1}{Z_1} \tag{18}$$

$$x_2 = \frac{X_2}{Z_2} \tag{19}$$

$$y_1 = \frac{2b + (a + x_Gx_1)(x_G + x_1) - x_2(x_G - x_1)^2}{2y_G} \tag{20}$$

则点(x_1, y_1)即所求，其中(x_G, y_G)为基点 G 的坐标。

通过对比可以看出，在仿射坐标系下，每次计算点加和倍点时都需要求模逆。而模逆计算周期长，计算复杂度高，难于优化。如果采用标准射影坐标，就可以消除点乘迭代过程中的模逆运算，从而提高运算效率。同时，在标准射影坐标下，会将投影点和仿射点进行一一映射，运算开始时将仿射坐标变换到射影坐标中表示，运算结束时再映射回仿射坐标。所以在整个运算过程中，仅在最后一次使用模逆运算，中间迭代过程没有模逆参与。

2.4.3 数据流优化

为进一步优化点加和倍点的计算效率，本文对数据流进行了深度优化，使其在最短的时间内完成计算。由于快速模乘由 KOA 乘法和快速模约减 2 个模块组成，且都可在一个时钟内计算出结果，因此本文调整点加和倍点的部分计算流程，交替调用 KOA 乘法和快速模约减模块，充分发挥计算效率。

点加和倍点计算过程和数据流分别如表 1 和表 2 所示。

从表 1 和表 2 中可以看出，经过 12 个时钟，即可完成点加和倍点的计算。

2.5 并行调度管理

为了进一步加速 SM2 的计算，在 FPGA 上实例化多个模块，各模块相互独立，由上层状态机调度管理，从而实现多任务的并行处理，提高可扩展性。这里采用贪心策略对任务进行分发，当计算完成时，进行仲裁汇总，并以地址为标识将对应的结果上报给 CPU。假设 SM2 并行模块数为 n，具体流程如下。

表 1　　点加计算过程和数据流

时钟	KOA 乘法	快速模约减	中间变量	计算结果
1	$T_1=X_1Z_2$	—	$T_6=2b$	—
2	$T_2=X_2Z_1$	T_1 Mod P_{SM2}	$T_6=4b$	—
3	$T_3=Z_1Z_2$	T_2 Mod P_{SM2}	—	X_1Z_2
4	$T_4=X_1X_2$	T_3 Mod P_{SM2}	$T_1=T_1-T_2$	X_2Z_1
5	$T_5=aT_3$	T_4 Mod P_{SM2}	$T_2=T_1+T_2$	Z_1Z_2, $X_1Z_2-X_2Z_1$
6	$T_6=T_6T_3$	T_5 Mod P_{SM2}	—	X_1X_2, $X_1Z_2+X_2Z_1$
7	$T_1=T_1^2$	T_6 Mod P_{SM2}	$T_4=T_4-T_5$	aZ_1Z_2
8	$T_2=T_6T_2$	T_1 Mod P_{SM2}	—	$4bZ_1Z_2$, $X_1X_2-aZ_1Z_2$
9	$T_4=T_4^2$	T_2 Mod P_{SM2}	—	$(X_1Z_2-X_2Z_1)^2$
10	$T_1=x_GT_1$	T_4 Mod P_{SM2}	—	$4bZ_1Z_2(X_1Z_2+X_2Z_1)$
11	—	T_1 Mod P_{SM2}	$T_4=T_4-T_2$	$(X_1X_2-aZ_1Z_2)^2$
12	—	—	—	$x_G(X_1Z_2-X_2Z_1)^2$, $(X_1X_2-aZ_1Z_2)^2-4bZ_1Z_2(X_1Z_2+X_2Z_1)$

表 2　　倍点计算过程和数据流

时钟	KOA 乘法	快速模约减	中间变量	计算结果
1	$T_1=Z_1^2$	—	$T_4=2b$	—
2	$T_2=X_1^2$	T_1 Mod P_{SM2}	$T_4=4b$	—
3	$T_3=aT_1$	T_2 Mod P_{SM2}	—	Z_1^2
4	$T_4=T_4T_1$	T_3 Mod P_{SM2}	—	X_1^2
5	$T_5=X_1Z_1$	T_4 Mod P_{SM2}	$T_2=T_2-T_3$	aZ_1^2
6	$T_1=T_1T_4$	T_5 Mod P_{SM2}	$T_3=T_2+T_3$	$4bZ_1^2$, $X_1^2-aZ_1^2$
7	$T_4=T_5T_4$	T_1 Mod P_{SM2}	$T_3=2T_3$	X_1Z_1, $X_1^2+aZ_1^2$
8	$T_2=T_2^2$	T_4 Mod P_{SM2}	$T_3=2T_3$	$4bZ_1^4$, $2(X_1^2+aZ_1^2)$
9	$T_3=T_5T_3$	T_2 Mod P_{SM2}	$T_4=2T_4$	$4bX_1Z_1^3$, $4(X_1^2+aZ_1^2)$
10	—	T_3 Mod P_{SM2}	$T_2=T_2-T_4$	$(X_1^2-aZ_1^2)^2$, $8bX_1Z_1^3$
11	—	—	$T_1=T_1+T_3$	$4X_1Z_1(X_1^2+aZ_1^2)$, $(X_1^2-aZ_1^2)^2-8bX_1Z_1^3$
12	—	—	—	$4X_1Z_1(X_1^2+aZ_1^2)+4bZ_1^4$

1) 任务分发

① 从 0 到 n 轮询 SM2 是否空闲，如果存在空闲模块 i，跳转②；否则继续执行①。

② 根据 i 对应的地址，将任务和控制命令载入第 i 个 SM2 模块对应的 RAM。

2) 执行计算

① 第 i 个模块从 RAM 中读取自身的任务和控制命令。

② 如果控制命令为开始，则说明任务已配置好，开始计算；否则跳转①继续等待。

3) 结果汇总

① 当第 i 个模块计算完成时，将地址、完成标

志位和结果写入第 i 个 FIFO。

② 从 0 到 n 轮询所有 FIFO，如果完成标志位为 1，将该结果汇总到最终的一个 FIFO，并上报。

其中，步骤 1)～步骤 3)并行执行，并通过异步 RAM 或 FIFO 进行数据传输。任务分发可依次写入多个任务，各 SM2 模块收到任务后，开始并行计算。最后将结果汇总到一个 FIFO 中，并由 CPU 依次读取结果。

3　实验结果与分析

本文实验的硬件平台是由 4 块大规模 FPGA 构成的加速卡，芯片型号为 Xcku-060-ffva1156-2-i，软件为 Vivado 2019.2。

3.1　各模块实现

在 FPGA 上以 8 路并行实现 SM2 整体算法，其各模块时钟频率、资源占用和计算周期的具体情况如表 3 所示。

表 3　　SM2 各模块实现情况

模块	频率/MHz	LUT	REG	DSP	计算周期
模加减	27	3 185	0	0	1
KOA 快速乘法	27	1 993	0	144	1
快速模约减	27	2 609	0	0	1
Barrett 模乘	27	5 131	2 323	144	6
基 4 模逆	27	8 625	1 524	0	212
点加	27	8 583	4 380	144	12
倍点	27	9 627	3 620	144	12
点乘	27	19 087	12 906	288	3 064
坐标转换	27	25 486	5 925	144	225
主控状态机	50	9 999	22 943	0	—

从表 3 中可以看出，单个 SM2 中各模块占用资源适中，充分利用了 LUT、REG 和 DSP 等资源。其中，点乘频率为 27 MHz，经过 3 064 个时钟即可完成计算。当 8 路 SM2 并行时，整体占用 FPGA 总 LUT 资源的比例为 60.23%，REG 为 24.48%，DSP 为 88.70%，整体功耗仅为 3.375 W。由于 FPGA 总的 DSP 个数为 2 760，当前实现共占用 2 448 个，其使用接近饱和，具有较高的资源利用率。显然，如果拥有更多的 DSP 资源，SM2 的性会进一步提高。

3.2　性能分析对比

表 4 给出了不同坐标系下点加和倍点运算次数对比。其中，I、M 和 S 分别表示模逆、模乘和模平方运算。显然，蒙哥马利方法优于其他方法，对于点加，其经过 8 次模乘、2 次模平方即可完成计算；对于倍点，其经过 6 次模乘、3 次模平方即可完成计算。

表 4　不同坐标系下点加和倍点运算次数对比

运算	坐标系			
	仿射坐标	标准射影坐标	Jacobian 加重射影坐标	蒙哥马利(标准射影坐标)
点加	I+2M+S	13M+2S	12M+4S	8M+2S
倍点	I+2M+2S	8M+5S	4M+6S	6M+3S

对于点乘，在计算 k 倍点时，设 k 的比特数为 l，如果采用二进制展开法，需要 l 次倍点和 $\frac{l}{2}$ 次点加运算；NAF 扫描法需要 l 次倍点和 $\frac{l}{3}$ 次点加运算；NAF 滑动窗口法需要约 l 次倍点和 $2^{\omega-2}+\frac{l}{\omega+1}$ 次点加运算，其中 ω 为窗口宽度；蒙哥马利法需要 l 次倍点和 l 次点加运算。但是对于 FPGA，蒙哥马利倍点和点加可并行计算，相当于整体只需要 l 次运算，而其他方法无法并行且整体运算次数明显大于 l。显然，蒙哥马利法具有更好的计算性能。

其次，表 5 给出了 FPGA 与 CPU SM2 的速度对比。其中，FPGA 的速度为 4 块 FPGA 的速度总和。

表 5　　FPGA 与 CPU SM2 的速度对比

器件	速度/（次·秒$^{-1}$）	提升倍数
CPU（Intel i5-8300）	800	—
FPGA（任意参数）	31 936	39.92
FPGA（推荐参数）	281 984	352.48

最后，将本文方案与其他硬件方案的点乘进行性能对比，如表 6 所示。

表 6　本文方案与其他硬件方案点乘性能对比

方案	器件	参数类型	时钟频率/MHz	点乘性能（次·秒$^{-1}$）	运算时钟周期数
文献[2]	Virtex-7	任意	225.0	671	335 360
文献[6]	ASIC 65 nm	任意	546.5	1 375	397 300
文献[9]	Virtex-7	任意	90.7	1 378	65 783
文献[17]	Virtex-7	任意	177.7	676	262 650
文献[4]	ASIC 0.13 μm	推荐	163.7	49 105	3 333
文献[5]	ASIC 0.13 μm	推荐	214.0	45 147	4 740
文献[12]	Virtex-6	推荐	166.0	10 807	15 360
文献[16]	XC7Z020	推荐	268.1	2 173	123 187
本文方案	Xcku-060	任意 推荐	27.0 27.0	998 8 812	27 028 3 064

从表 6 可以看出，无论是任意参数还是推荐参数，相比于其他 FPGA 实现，本文方案的点乘周期要远低于其他方案。在点乘性能方面，由于时钟频率较低，单个模块的计算性能不如其他部分方案。但是本文方案在 FPGA 中实例化了 8 个模块，对于任意参数，其总速度为 7 984 次/秒，高于其他方案；对于推荐参数，其总速度为 70 496 次/秒，远高于文献[4-5]的 ASIC 实现，且当加速卡 4 块 FPGA 同时工作时，其最高总速度更是达到了 281 984 次/秒，具有十分可观的效率。

3.3　安全性与资源对比

SM2 算法除了对运算速度的要求外，还必须具备一定的抗攻击能力。而 SPA 是最常见的攻击方式。本文 SM2 算法每次都执行点加和倍点，且点加和倍点的功耗波形不存在差别，攻击者无法得到运算规律，可有效抵抗 SPA。

此外，面积与时间的乘积 AT 客观反映了资源消耗和算法性能之间的关系。AT 值越低，表明在有限的资源条件下，与性能之间取得的平衡越好。这里，将本文方案与其他 FPGA 实现方案在安全性和 AT 值方面进行了对比，如表 7 所示。

从表 7 中可以看出，本文 SM2 算法在具有抗 SPA 的同时，也具有较优的 AT 值。对于任意参数的椭圆曲线密码算法实现，虽然文献[2]占用资源少，AT 值低，但它不具有抗攻击性。此外，由于文献[2]采用蒙哥马利模乘，需要额外进行预处理并对结果进行转换。

最后，将本文方案与其他 FPGA 实现的 SM2 算法进行了综合对比，包括抗攻击性、性能和 AT 值等方面，结果如表 8 所示。

从表 8 中可以看出，本文方案在性能和 AT 值方面均高于其他方案，性能至少提高了 3.26 倍，AT 值至少提高了 4.84 倍，且具有抗攻击性。这是由于本文方案深度优化了模乘运算，并优化了点加和倍点计算流程，缩短了点乘周期，提高了计算性能。同时，本文合理利用了 FPGA 逻辑资源，减少了资源消耗，具有较优的 AT 值。

3.4　应用与可扩展性分析

SM2 签名/验签、加解密和密钥交换过程中会多次调用点乘。以签名/验签应用为例，在签名过程中会调用一次点乘和多次 SM3 哈希值计算，在验签过程中会调用两次点乘和多次 SM3。为节省资源，对于 SM3 采用将两轮压缩为一轮的串行方式实现；中间大数的模乘、模逆及坐标转换采用共享模块的方式实现，具体资源占用如表 9 所示。

表 7　本文方案与其他 FPGA 方案在安全性和 AT 值方面的对比

方案	器件	参数类型	抵抗 SPA	面积/kslice	时间/ms	AT
文献[2]	Virtex-7	任意	否	1.70	1.49	2.53
文献[9]	Virtex-7	任意	否	19.33	0.73	14.11
文献[17]	Virtex-7	任意	是	8.90	1.48	13.17
文献[12]	Virtex-6	推荐	是	6.78	0.09	0.61
文献[16]	XC7Z020	推荐	是	2.02	0.46	0.93
本文方案	Xcku-060	任意	是	4.68	1.00	4.68
		推荐	是	4.80	0.11	0.53

表 8　本文方案与其他 FPGA 实现的 SM2 算法综合对比

方案	器件	抵抗 SPA	时钟频率/MHz	性能/（次·秒$^{-1}$）	面积/ kslice	时间/ms	AT	本文性能提高倍数	本文 AT 提高倍数
文献[19]	StratixII	是	62.3	1 298	4.74	0.77	3.65	6.79	6.91
文献[24]	xc6vlx760	否	38.0	2 703	6.91	0.37	2.56	3.26	4.84
文献[25]	XC7V585-3	是	244.0	1 645	5.35	0.61	3.26	5.36	6.18
文献[26]	ZYNQ ZC706	是	110.0	444	7.15	2.25	16.09	19.85	30.47
文献[27]	xc6vlx760	否	38.4	2 703	10.06	0.37	3.72	3.26	7.05
本文方案	Xcku-060	是	27.0	8 812	4.80	0.11	0.53	—	—

表 9 SM2 签名/验签的具体资源占用

功能	LUT	REG	DSP	计算周期
SM3	2 677	1 712	0	182
共享模块	29 531	13 001	144	519
签名	30 823	22 981	288	3 768
验签	32 543	22 586	288	7 591

在 27 MHz 频率下，由表 9 中的计算周期计算可得，单模块签名速度为 7 165 次/秒，验签速度为 3 556 次/秒。文献[28]签名速度为 26 063 次/秒，但其未提及 SM3 算法和坐标系转换的耗时。本文方案 FPGA 可用 8 个模块并行执行，当以一个 SM3 模块+8 个点乘模块+一个共享模块的架构实现时，总签名速度即点乘速度为 70 496 次/秒，具有更高的计算效率。

在可扩展性方面，本文 SM2 的并行架构在工程实现上方便移植。根据不同 FPGA 芯片资源实例化不同数量的算法模块，仅需修改主控状态机，即可完成多模块的通信与配置，以此实现多任务并行计算。

其次，SM2 各模块间通过寄存器或 FIFO 互连，具有松耦合架构，仅需替换快速模约减模块，即可实现 NIST ECC 256、区块链中 Secp256k1 等其他椭圆曲线算法。同时，本文实现了任意参数的 SM2 计算，适用范围更广。

最后，本文所有运算均在 FPGA 内部实现，可配合哈希算法实现签名/验签、加解密和密钥交换等应用，且具有高效能计算的特点，可即插即用，灵活地应用在边缘设备的认证、云端服务、隐私保护等多种场合。

4 结束语

本文提出的可重构 SM2 素域算法优化通过对 SM2 各计算阶段的深入剖析，以软硬件协同的方式实现，在标准射影坐标系下优化并实现了蒙哥马利点乘、快速模乘、快速模逆和坐标系转换。同时，通过 SM2 多模块并行，加速了数据的并行处理，可满足不同应用的计算需求。实验结果表明，该方法可重构出的 SM2 算法在性能、资源等方面具有明显优势，较 CPU 有 352.48 倍以上的提升，并可扩展到签名/验签、加解密和密钥交换等功能，兼顾计算的高效性和灵活性。

下一步，将实现其他位宽的椭圆曲线的并行计算，通过对 FPGA 进行合理布局划分，完成相应算法的切换和共享模块的使用，以获取更高的计算性能和适用范围。

参考文献：

[1] JAVEED K, WANG X J. Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp[C]//Proceedings of 2014 24th International Conference on Field Programmable Logic and Applications (FPL). Piscataway: IEEE Press, 2014: 1-6.

[2] AMIET D, CURIGER A, ZBINDEN P. Flexible FPGA-based architectures for curve point multiplication over GF(p)[C]//Proceedings of 2016 Euromicro Conference on Digital System Design (DSD). Piscataway: IEEE Press, 2016: 107-114.

[3] FENG X, LI S G. A high-speed and SPA-resistant implementation of ECC point multiplication over GF(p)[C]//Proceedings of 2017 IEEE Trustcom/BigDataSE/ICESS. Piscataway: IEEE Press, 2017: 255-260.

[4] ZHAO Z W, BAI G Q. Ultra high-speed SM2 ASIC implementation[C]//Proceedings of 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. Piscataway: IEEE Press, 2014: 182-188.

[5] ZHANG D, BAI G Q. Ultra high-performance ASIC implementation of SM2 with power-analysis resistance[C]//Proceedings of 2015 IEEE International Conference on Electron Devices and Solid-State Circuits. Piscataway: IEEE Press, 2015: 523-526.

[6] HOSSAIN M S, KONG Y N, SAEEDI E, et al. High-performance elliptic curve cryptography processor over NIST prime fields[J]. IET Computers & Digital Techniques, 2017, 11(1): 33-42.

[7] 韩晓薇, 乌力吉, 王蓓蓓, 等. 抗简单功耗攻击的 SM2 原子算法[J]. 计算机研究与发展, 2016, 53(8): 1850-1856.
HAN X W, WU L J, WANG B B, et al. Atomic algorithm against simple power attack of SM2[J]. Journal of Computer Research and Development, 2016, 53(8): 1850-1856.

[8] JAVEED K, WANG X J. Low latency flexible FPGA implementation of point multiplication on elliptic curves over GF(p)[J]. International Journal of Circuit Theory and Applications, 2017, 45(2): 214-228.

[9] RAHMAN M S, HOSSAIN M S, RAHAT E H, et al. Efficient hardware implementation of 256-bit ECC processor over prime field[C]//Proceedings of 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE). Piscataway: IEEE Press, 2019: 1-6.

[10] JÄRVINEN K, MIELE A, AZARDERAKHSH R, et al. FourQ on FPGA: new hardware speed records for elliptic curve cryptography over large prime characteristic fields[C]//International Conference on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2016: 517-537.

[11] LI W, LIU J H, BAI G Q. High-speed implementation of SM2 based on fast modulus inverse algorithm[C]//Proceedings of 2018 China Semiconductor Technology International Conference (CSTIC). Piscataway: IEEE Press, 2018: 1-3.

[12] DING J N, LI S G. A reconfigurable high-speed ECC processor over NIST primes[C]//Proceedings of 2017 IEEE Trustcom/ BigDa-

taSE/ICESS. Piscataway: IEEE Press, 2017: 1064-1069.

[13] YANG D Y, DAI Z B, LI W, et al. An efficient ASIC implementation of public key cryptography algorithm SM2 based on module arithmetic logic unit[C]//Proceedings of 2019 IEEE 13th International Conference on ASIC. Piscataway: IEEE Press, 2019: 1-4.

[14] HOSSAIN M R, HOSSAIN M S. Efficient FPGA implementation of modular arithmetic for elliptic curve cryptography[C]//Proceedings of 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE). Piscataway: IEEE Press, 2019: 1-6.

[15] DING J N, LI S G, GU Z. High-speed ECC processor over NIST prime fields applied with toom-cook multiplication[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2019, 66(3): 1003-1016.

[16] ROY D B, MUKHOPADHYAY D. High-speed implementation of ECC scalar multiplication in GF(p) for generic Montgomery curves[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(7): 1587-1600.

[17] ISLAM M M, HOSSAIN M S, HASAN M K, et al. FPGA implementation of high-speed area-efficient processor for elliptic curve point multiplication over prime field[J]. IEEE Access, 2019, 7: 178811-178826.

[18] KHAN S, JAVEED K, SHAH Y A. High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications[J]. Microprocessors and Microsystems, 2018, 62: 91-101.

[19] ZHANG D, BAI G Q. High-performance implementation of SM2 based on FPGA[C]//Proceedings of 2016 8th IEEE International Conference on Communication Software and Networks. Piscataway: IEEE Press, 2016: 718-722.

[20] GARG H K, XIAO H S. New residue arithmetic based barrett algorithms: modular integer computations[J]. IEEE Access, 2016, 4: 4882-4890.

[21] BRIER E, JOYE M. Weierstraß elliptic curves and side-channel attacks[C]//International Workshop on Public Key Cryptography. Berlin: Springer, 2002: 335-345.

[22] JAVEED K, WANG X J. FPGA based high speed SPA resistant elliptic curve scalar multiplier architecture[J]. International Journal of Reconfigurable Computing, 2016, 2016: 6371403.

[23] YU W, WANG K P, LI B, et al. Montgomery algorithm over a prime field[J]. Chinese Journal of Electronics, 2019, 28(1): 39-44.

[24] HU X H, ZHENG X, ZHANG S S, et al. A high-performance elliptic curve cryptographic processor of SM2 over GF(p)[J]. Electronics, 2019, 8(4): 431.

[25] WU T, YE J H, LU J. Hardware implementation of SM2 ECC protocols on FPGAs[C]//Proceedings of 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference. Piscataway: IEEE Press, 2021: 33-37.

[26] 王腾飞, 张海峰, 许森. SM2 专用指令协处理器设计与实现[J]. 计算机工程与应用, 2022, 58(2): 102-109.
WANG T F, ZHANG H F, XU S. Design and implementation of SM2 co-processor with specific instructions[J]. Computer Engineering and Applications, 2022, 58(2): 102-109.

[27] XIAO Y, LIN W B, ZHAO Y, et al. A high-speed elliptic curve cryptography processor for teleoperated systems security[J]. Mathematical Problems in Engineering, 2021, 2021: 6633925.

[28] 杨国强, 丁杭超, 邹静, 等. 基于高性能密码实现的大数据安全方案[J]. 计算机研究与发展, 2019, 56(10): 2207-2215.
YANG G Q, DING H C, ZOU J, et al. A big data security scheme based on high-performance cryptography implementation[J]. Journal of Computer Research and Development, 2019, 56(10): 2207-2215.

[作者简介]

李斌（1986− ），男，河南郑州人，博士，郑州大学讲师，主要研究方向为信息安全、可重构计算。

周清雷（1962− ），男，河南新乡人，博士，郑州大学教授，主要研究方向为信息安全、自动机理论和计算复杂性理论。

陈晓杰（1993− ），男，河南武陟人，数学工程与先进计算国家重点实验室博士生，主要研究方向为信息安全、可重构计算。

冯峰（1990− ），男，河南新乡人，郑州大学博士生，主要研究方向为信息安全。