Operations Research and Fuzziology 运筹与模糊学, 2023, 13(6), 7247-7255
Published Online December 2023 in Hans. https://www.hanspub.org/journal/orf
https://doi.org/10.12677/orf.2023.136711

面向物联网安全的素域SM2点乘硬件优化

谭光昭，周　骅*

贵州大学大数据与信息工程学院，贵州　贵阳

收稿日期：2023年10月22日；录用日期：2023年12月18日；发布日期：2023年12月27日

摘　要

为解决现有SM2软硬件实现在物联网应用中由于成本与功耗限制所存在的计算速度较慢问题，采用自下而上的思想对SM2点乘算法分层硬件优化。首先基于DSP乘法器提出单周期并行256位乘法算法KO-5，藉此设计了可流水线操作的2个时钟周期的256位模乘法器；然后基于并行化和流水线技术等设计点运算模块，大大减少了计算时间；最后，通过使用固定窗口点乘算法提升效率。实验结果分析表明，经过优化的点乘模块计算时间为136.68 us，逻辑资源使用仅19.59 kLUTs和144 DSPs，相对同类工作在性能和资源消耗方面均有优势，适用于高性能物联网安全场景。

关键词

并行乘法器，流水线，硬件优化，固定窗口

Hardware Optimization of Prime Field SM2 Point Multiplication for IoT Security

Guangzhao Tan, Hua Zhou*

College of Big Data and Information Engineering, Guizhou University, Guiyang Guizhou

Received: Oct. 22nd, 2023; accepted: Dec. 18th, 2023; published: Dec. 27th, 2023

Abstract

To solve the problems of slow computation speed in the existing SM2 hardware and software implementations under the cost and power constraints of IoT applications, a bottom-up approach was adopted to optimise the SM2 algorithm hardware hierarchically. Firstly, a single-cycle parallel 256-bit multiplie algorithm KO-5 was proposed based on DSP multiplier, and utilized to design a two-clock-cycle pipeline version of the 256-bit modular multiplication; then the point operation

*通讯作者。

文章引用: 谭光昭, 周骅. 面向物联网安全的素域 SM2 点乘硬件优化[J]. 运筹与模糊学, 2023, 13(6): 7247-7255.
DOI: 10.12677/orf.2023.136711

module was designed by through parallelization techniques and pipeline techniques, which greatly reduced the calculation time. Finally, a fixed-window point multiplication algorithm was used to improve the efficiency. The experimental results show that the computation time of the optimized point multiplication module is reduced to 136.68 us only by using 19.59 kLUTs and 144 DSPs logical resources, which has advantages of performance and lower resource utilization compared to similar works, and is suitable for high-performance IoT security scenarios.

Keywords

Parallel Multiplier, Pipeline, Hardware Optimization, Fixed Window

1. 引言

随着人工智能和大数据等技术的发展，人们对于物联网的需求也逐步从“万物互联”向“万物智联”转变[1]。与此同时，物联网和边缘计算设备面临着众多安全挑战[2]。保护物联网安全是一项复杂而持续的任务，需要采取综合性的措施来防御各种威胁。其中，以椭圆曲线密码(Elliptic Curve Cryptography, ECC)为典型的公钥密码，具有经典陷门函数特性，在保障数据安全和隐私、防止数据篡改和伪造、确保身份认证方面发挥着重要的作用，已经成为物联网安全领域的重要基石。为保护我国信息安全，国家密码管理局推出了基于椭圆曲线密码改进的 SM2 算法[3]。

SM2 的核心运算是椭圆曲线标量乘法(Elliptic Curve Scalar Multiplication)，也被称为椭圆曲线点乘法(Elliptic Curve Point Multiplication，简称点乘)。点乘是椭圆曲线密码系统中使用最频繁、耗时最长的运算，点乘运算速度是改进椭圆曲线密码的关键之一；同时，在物联网应用中，FPGA 硬件面临着成本和功耗问题。因此，本文考虑在资源受限情况下[4]，提出分层硬件优化方法：首先从 SM2 底层有限域运算层出发，针对调用最多的模乘部分提出了基于 DSP 乘法器的 256 bit 乘法算法 KO-5，仅需一个时钟周期得到乘法结果的同时大大降低了乘法器模块的逻辑资源使用率，并结合快速模约减算法实现了两个时钟周期的模乘法器，随后在此基础上采用流水线技术、预计算技术和固定窗口算法等优化方法逐层优化。最终设计实现了一个高性能和低资源消耗的 SM2 点乘模块，可应用于具有较高性能需求的物联网数据加解密、设备身份认证和数据源认证等场景。

2. SM2 参数与架构

2.1. SM2 参数

伪梅森素数域 GF(p)上的椭圆曲线表示为：

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

国家密码管理局推荐的 SM2 系统参数为：

p = FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFF

a = FFFFFFFE FFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFC 00000000 FFFFFFFF FFFFFFFC

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

2.2. SM2 架构分析

公钥密码系统一般是由模加、模减、模乘法和模逆等有限域运算单元构建，SM2 算法分层次架构如图 1 所示。

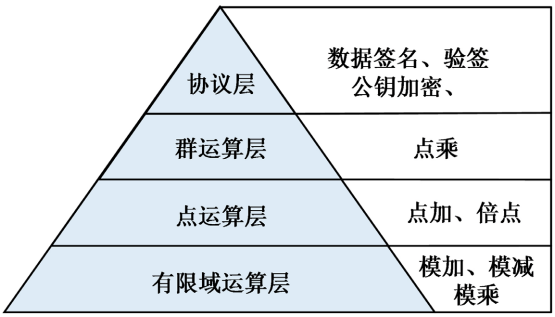


Figure 1. Hierarchy of SM2 algorithm
图 1. SM2 算法分层架构

第一层是有限域运算层，包括模加、模减、模乘法等基本模运算，本层主要是在模乘法器部分对于速度和逻辑资源使用率的优化。

第二层为椭圆曲线上的点运算层，通过对已知的坐标，进行有限域运算得到另一个未知点坐标。

第三层是群运算层，点乘是椭圆曲线密码的核心运算，对椭圆曲线上的一个 P 点求解在曲线上的 k 倍点 kP 坐标。

顶层为协议层，一般有椭圆曲线数字签名算法、验签和公钥加解密算法等。

3. 硬件优化设计

3.1. 模乘优化

模乘由于具有计算密集性和高频次调用[5]，其运算性能和功耗显著影响着 SM2 系统性能。一般将模乘分为大数乘法和模约减两个部分。

3.1.1. 大数乘法优化

两个 N 位整数相加时间复杂度为 $O(n)$，而两个 N 位数乘法时间复杂度为 $O(n^2)$，乘法运算的复杂度对于整数来说太高[6]。一般可以采用 Karatsuba-Ofman 乘法算法，通过用成本低的加减法运算代替部分乘法运算来降低复杂度。

算法 1 Karatsuba-Ofman 乘法

输入 $0 \le x, y \le 2N$ //$2N$ 为位宽

输出 $P = x * y$

1) 划分 $x = x_1 2^N + x_0$，$y = y_1 2^N + y_0$；

2) $p_0 = x_0 * y_0$，$p_1 = x_1 * y_1$；

3) $p_{01} = (x_1 + x_0) * (y_1 + y_0)$；

4) $P = x_1 y_1 2^{2N} + (x_1 y_0 + x_0 y_1) 2^N + x_0 y_0$

$= p_1 2^{2N} + (p_{01} - p_0 - p_1) 2^N + p_0$。

在算法 1 第四步中，KO 算法通过用资源消耗低的减法和加法运算代替部分乘法运算来执行乘法运算。通过转化，原本需要四个 N 位乘法减少到两个 N 位乘法加一个 $N+1$ 位乘法，代价仅是增加了额外

的加法器。

Xilinx 提供的硬件 DSP 乘法器最高只到 64 bit，而 SM2 需要 256 bit 乘法计算。主流算法采用多级 KO 算法实现，第一级由 64 bit 乘法器与多个加法器构成 128 bit 乘法器，第二级在此基础上构成 256 bit 乘法器，乘积需要多个计算时钟周期。因此，本文基于并行思想提出和设计了将乘数划分成 5 部分的 KO-5 算法，具体算法设计如算法 2 所示。

算法 2 KO-5 乘法

输入 x, y //256 bit

输出 $P = x * y$ //512 bit

1) $x = x_4 2^{4N} + x_3 2^{3N} + x_2 2^{2N} + x_1 2^N + x_0$；

$y = y_4 2^{4N} + y_3 2^{3N} + y_2 2^{2N} + y_1 2^N + y_0$；

2) $p_0 = x_0 * y_0$， $p_1 = x_1 * y_1$，

$p_2 = x_2 * y_2$， $p_3 = x_3 * y_3$， $p_4 = x_4 * y_4$；

3) //部分积计算

$$p_{01} = (x_1 + x_0) * (y_1 + y_0); p_{02} = (x_2 + x_0) * (y_2 + y_0),$$
$$p_{03} = (x_3 + x_0) * (y_3 + y_0); p_{04} = (x_4 + x_0) * (y_4 + y_0),$$
$$p_{12} = (x_2 + x_1) * (y_2 + y_1); p_{13} = (x_3 + x_1) * (y_3 + y_1),$$
$$p_{14} = (x_4 + x_1) * (y_4 + y_1); p_{23} = (x_3 + x_2) * (y_3 + y_2),$$
$$p_{24} = (x_4 + x_2) * (y_4 + y_2); p_{34} = (x_4 + x_3) * (y_4 + y_3);$$

4)
$$\begin{aligned} P = {} & x_4 * y_4 2^{8N} \\ & + (x_4 * y_3 + x_3 * y_4) 2^{7N} \\ & + (x_4 * y_2 + x_3 * y_3 + x_2 * y_4) 2^{6N} \\ & + (x_4 * y_1 + x_3 * y_2 + x_2 * y_3 + x_1 * y_4) 2^{5N} \\ & + (x_4 * y_0 + x_3 * y_1 + x_2 * y_2 + x_1 * y_3 + x_0 * y_4) 2^{4N} \\ & + (x_3 * y_0 + x_2 * y_1 + x_1 * y_2 + x_0 * y_3) 2^{3N} \\ & + (x_2 * y_0 + x_1 * y_1 + x_0 * y_2) 2^{2N} \\ & + (x_1 * y_0 + x_0 * y_1) 2^N \\ & + x_0 * y_0 \\ = {} & p_4 2^{8N} \\ & + (p_{34} - p_3 - p_4) 2^{7N} \\ & + (p_{24} + p_3 - p_2 - p_4) 2^{6N} \\ & + (p_{23} + p_{14} - p_1 - p_2 - p_3 - p_4) 2^{5N} \\ & + (p_{13} + p_{04} + p_2 - p_0 - p_1 - p_3 - p_4) 2^{4N} \\ & + (p_{12} + p_{03} - p_0 - p_1 - p_2 - p_3) 2^{3N} \\ & + (p_{02} + p_1 - p_0 - p_2) 2^{2N} \\ & + (p_{01} - p_0 - p_1) 2^N \\ & + p_0 。 \end{aligned}$$

3.1.2. 快速模约减

乘法运算得到 512 bit 的乘积后，需要约减为 0~P_{256}。为了保证 SM2 运算性能，本文采用针对 SM2 模 P_{256} 的快速模约减方法，如算法 3 所示。

算法 3 快速模约减

输入 a, P_{256} //a 512 bit，P_{256} 256 bit

输出 $r = a \bmod P_{256}$ //r 256 bit

1) ai 均为 32 bit

2) $S_1 = (\text{a07}||\text{a06}||\text{a05}||\text{a04}||\text{a03}||\text{a02}||\text{a01}||\text{a00})$

3) $S_2 = (\text{a15}||\text{a14}||\text{a13}||\text{a12}||\text{a11}||\ 0\ ||\text{a09}||\text{a08})$

4) $S_3 = (\text{a14}||\ 0\ ||\text{a15}||\text{a14}||\text{a13}||\ 0\ ||\text{a14}||\text{a13})$

5) $S_4 = (\text{a13}||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a15}||\text{a14})$

6) $S_5 = (\text{a12}||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a15})$

7) $S_6 = (\text{a11}||\text{a11}||\text{a10}||\text{a15}||\text{a14}||\ 0\ ||\text{a13}||\text{a12})$

8) $S_7 = (\text{a10}||\text{a15}||\text{a14}||\text{a13}||\text{a12}||\ 0\ ||\text{a11}||\text{a10})$

9) $S_8 = (\text{a09}||\ 0\ ||\ 0\ ||\text{a09}||\text{a08}||\ 0\ ||\text{a10}||\text{a09})$

10) $S_9 = (\text{a08}||\ 0\ ||\ 0\ ||\ 0\ ||\text{a15}||\ 0\ ||\text{a12}||\text{a11})$

11) $S_{10} = (\text{a15}||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0)$

12) $S_{11} = (0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a14}||\ 0\ ||\ 0)$

13) $S_{12} = (0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a13}||\ 0\ ||\ 0)$

14) $S_{13} = (0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a09}||\ 0\ ||\ 0)$

15) $S_{14} = (0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\ 0\ ||\text{a08}||\ 0\ ||\ 0)$

16) $r = (S_1 + S_2 + S_3 + S_4 + S_5 + S_6) + 2(S_7 + S_8 + S_9 + S_{10}) - (S_{11} + S_{12} + S_{13} + S_{14})$

对特定的伪梅森素数简化，只需要一个时钟周期的组合逻辑运算即可完成 512 bit 数据的快速模约减运算。

为了提高模乘计算性能，在第 1 个时钟周期计算乘法后将 512 位乘积寄存，第 2 个时钟周期直接进行模约减，所以模乘可以采用流水线技术提高计算效率。

3.2. 点运算优化

对于点运算，本文采用并行化和流水线等技术进行优化提高性能，并行表现为 1M + 1AS 架构，即一路 M (模乘法器)和一路 AS (模加、减法器)，其中模乘法器需要两个时钟周期，采取流水线方式计算，预计算 $T_0 = 4b$，则倍点、点加计算流程[7]如表 1、表 2 所示。

Table 1. The process of point doubling
表 1. 倍点运算流程

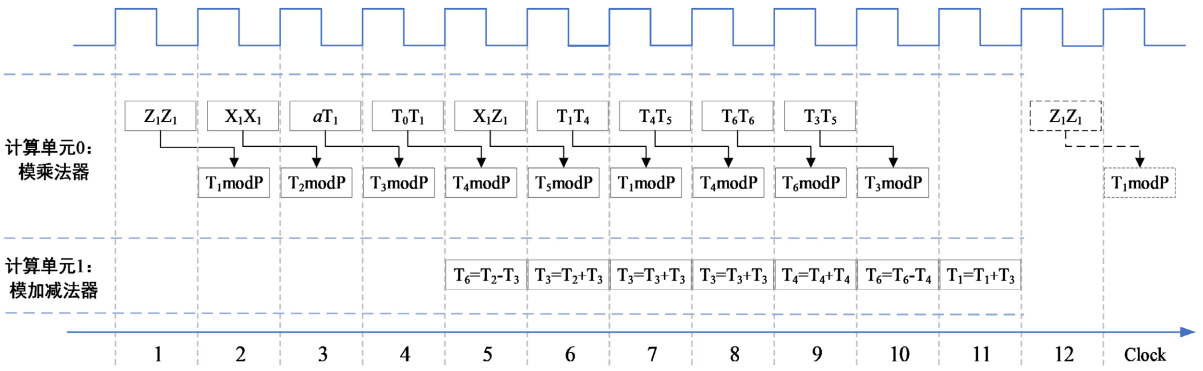
步骤	模乘(M)	模加减(AS)	计算结果
1	$T_1 = Z_1 Z_1$	--	--
2	$T_2 = X_1 X_1$	--	--
3	$T_3 = aT_1$	--	Z_1^2
4	$T_4 = T_1 T_0$	--	X_1^2
5	$T_5 = X_1 Z_1$	$T_6 = T_2 - T_3$	aZ_1^2

Continued

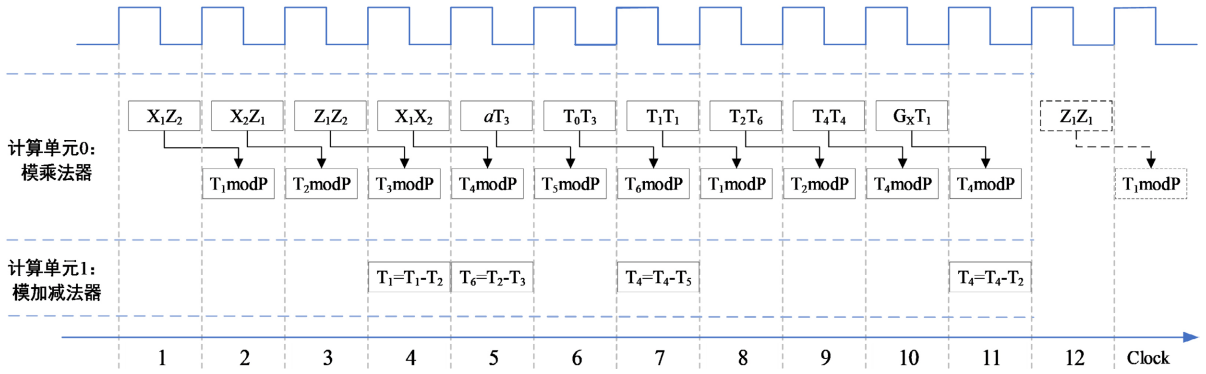
6	$T_1 = T_1T_4$	$T_3 = T_2 + T_3$	$4bZ_1^2, X_1^2 - aZ_1^2$
7	$T_4 = T_5T_4$	$T_3 = T_3 + T_3$	$X_1Z_1, X_1^2 + aZ_1^2$
8	$T_6 = T_6T_6$	$T_3 = T_3 + T_3$	$4bZ_1^4,\ 2\left(X_1^2 + aZ_1^2\right)$
9	$T_3 = T_5T_3$	$T_4 = T_4 + T_4$	$4bX_1Z_1^3, 4\left(X_1^2 + aZ_1^2\right)$
10	--	$T_6 = T_6 - T_4$	$\left(X_1^2 - aZ_1^2\right)^2, 8bX_1Z_1^3$
11	--	$T_1 = T_1 + T_3$	$4X_1Z_1\left(X_1^2 + aZ_1^2\right), \left(X_1^2 - aZ_1^2\right)^2 - 8bX_1Z_1^3$
12	$T_1 = Z_1Z_1$ ①	--	$4X_1Z_1\left(X_1^2 + aZ_1^2\right) + 4bZ_1^4$

Table 2. The process of point addition
表 2. 点加运算流程

步骤	模乘	模加减	
1	$T_1 = X_1Z_2$	--	--
2	$T_2 = X_2Z_1$	--	--
3	$T_3 = Z_1Z_2$	--	X_1Z_2
4	$T_4 = X_1X_2$	$T_1 = T_1 - T_2$	X_2Z_1
5	$T_5 = aT_3$	$T_2 = T_1 + T_2$	$Z_1Z_2, X_1Z_2 - X_2Z_1$
6	$T_6 = T_0T_3$	$T_2 = T_1 + T_2$	$X_1X_2, X_1Z_2 + X_2Z_1$
7	$T_1 = T_1^2$	$T_4 = T_4 - T_5$	aZ_1Z_2
8	$T_2 = T_6T_2$	--	$4bZ_1Z_2, X_1X_2 - aZ_1Z_2$
9	$T_4 = T_4^2$	--	$\left(X_1Z_2 - X_2Z_1\right)^2$
10	$T_1 = G_xT_1$	--	$4bZ_1Z_2\left(X_1Z_2 + X_2Z_1\right)$
11	--	$T_4 = T_4 - T_2$	$\left(X_1X_2 - aZ_1Z_2\right)^2$
12	$T_1 = Z_1Z_1$ ②	--	$G_x\left(X_1Z_2 - X_2Z_1\right)^2, \left(X_1X_2 - aZ_1Z_2\right)^2 - bZ_1Z_2\left(X_1Z_2 + X_2Z_1\right)$



(a)



(b)

Figure 2. Point Doubling and Point Addition timing sequence. (a) Point doublingtiming sequence; (b) point addition timing sequence

图 2. 点加和倍点计算时序。(a) 倍点计算时序；(b) 点加计算时序

由于采用并行化和流水线技术，倍点和点加均只需要 11 个步骤完成点计算，即 12 个时钟周期得到计算结果。由于倍点计算次数相比较多，因此考虑在倍点和点加计算的第 12 个时钟周期进行预计算①、②，在连续倍点计算时可减少到 11 个时钟周期，具体计算时序如图 2 所示。

3.3. 群运算(点乘)优化

对于一个二进制序列，汉明重量(Hamming Weight)是指非零位的个数。在椭圆曲线点乘运算中，倍点运算次数难以减少，对点乘运算的速度优化主要是降低二进制随机数 k 的汉明重量，即减少点加的运算次数。对于 256 位二进制随机数 k，传统的 Double-and-Add 算法(倍点－点加算法)汉明密度为 128，即需要 128 次点加运算。

为了提升点乘计算效率，本文采用窗口 $w=4$ 的固定窗口算法。固定窗口算法利用窗口表预先计算出窗口内的点乘结果，在计算过程中能够将汉明密度降低至 64，点加运算次数降低了 50%，提高了计算速度。固定窗口点乘算法如下所示：

算法 4 固定窗口点乘

输入 $k,n,G(x,y)$ //k, n256bit, n 为曲线的阶

输出 $Q=kG$

1) $l=\lceil 256/4 \rceil$；

2) k 按窗口分组 $k_4[l],k_4[l-1],\cdots,k_4[0]$；

3) 预计算 $2G,3G,4G,\cdots,15G$；

4) $Q=(k_4[l])G$；

5) for i from l-1 to 0: //$k_4[i]\in(0\sim15)$

5.1 $Q=2^4Q$;//四次倍点计算

5.2 if ($k_4[i]\neq 0$):

$Q=Q+(k_4[i])G$;//点加计算

5.3 if ($Q=\infty$): //出现无穷远点

5.3.1 $Q=(k_4[i-1])G$;

5.3.2 $i=i-1$；

6) 返回 $Q=kG$。

算法 4 中，为了避免无穷远点出现，步骤 5.3 在计算结束进行点的验证。

4. 实现结果与对比

为了验证前文所述的SM2点乘结构，本文采用Xilinx Vivado 18.3开发平台进行设计和仿真，并在高性价比的 ZYNQ7020 平台对 256 位素域 SM2 点乘进行了功能和性能测试验证。

4.1. 大数乘法器优化对比

表 3 显示了本文所提出设计的乘法器优化前后性能对比结果。256 bit 大数乘法器优化前单周期内计算相对较少，乘法器最大时钟频率较高。但由于需要进行两次 KO 乘法，且需要分别用 64 位和 128 位乘法器分别修正得到 65 位和 129 位乘法器结果，因此计算时间较长，LUT 资源消耗较高。

优化后 LUT 资源消耗有所减少，DSP 资源不变，计算时间减少，且由于只需要 1 个时钟周期，更方便上层点运算采用流水线调度。

Table 3. The multiplier performance comparison before and after optimization
表 3. 乘法器优化前后性能对比

方法	最大频率/MHz	时钟	LUTs	DSPs	计算时间/us
优化前	40	2	4161	144	5
优化后	33.3	1	3680	144	3.3

4.2. 点乘设计对比

表 4 显示了本文提出的点乘设计与其他现有素域 P-256 的点乘设计结果对比。

Table 4. The result comparison of different point multiplication designs
表 4. 不同点乘设计结果对比

文献	平台	域	资源消耗	频率/MHz	计算周期/k	时间/us
[7]	Xcku060	P-256	19.09 kLUTs + 288 DSPs	27.0	3.064	110
[8]	Xc7vx330t	P-256	29.88 kLUTs + 144 DSPs	73.60	16.3	221
本文	Xc7z020	P-256	19.59 kLUTs + 144 DSPs	29.4	4.02	136.68

文献[7]采用 Xcku060 平台，在单周期内基于 64 位 DSP 乘法器完成两级 KO 分治乘法，采用 Montgomery 点乘算法并行实现点加和倍点运算，但最大频率较低，且 DSP 资源消耗过高，不适用于物联网应用中的低成本 FPGA 平台。

文献[8]采用 Xc7vx330t 平台，基于 64 位 DSP 乘法器，应用两级 KO 分治算法实现大数乘法器，并通过三路 MAD (复用的模乘、加减法器)并行加速。

以上工作中均使用了高性能高成本的 FPGA 平台，本文由于应用于物联网安全场景，在有限域、点和群运算层均对性能和逻辑资源使用率上的权衡和优化，DSP 资源使用率占 ZYNQ7020 的 65.5%，点乘计算需要 4020 个时钟周期，计算时间为 136.68 us，比仅文献[7]增加 24.3%的计算时间，但降低了 50%的 DSP 资源消耗，相比则减少了 34.4%的 LUT 资源，计算时间也减少了 38.2%。综合对比各研究，本文在性能和资源消耗均有较大优势，但优化时序仍有提升空间，以获得更高的系统时钟频率。

5. 结语

本文提出了一种对素域 SM2 点乘算法自下而上分层硬件优化方法，首先针对 SM2 中调用最多的高性能大数乘法器需求提出了基于 DSP 乘法器的 256 bit 乘法算法 KO-5，大幅减少乘法计算和乘法器模块的逻辑资源使用率，随后在此基础上采用流水线技术、预计算技术和快速模约减算法，固定窗口算法等优化方法逐层优化。设计实现了一个高性能和低资源消耗的素域 SM2 点乘模块。经过测试比较，本文相对于同类工作具有更高的性能和较低的资源消耗，可应用于具有较高性能需求的物联网数据加解密、设备身份认证和数据源认证等场景。

参考文献

[1] 于全, 梁丹丹, 张伟. 面向万物智联的云原生网络[J]. 物联网学报, 2021, 5(2): 1-6.

[2] 张妍, 黎家通, 宋小祎, 等. 物联网设备安全检测综述[J]. 计算机研究与发展, 2023, 60(10): 2271-2290.

[3] 国家密码管理局. GM/T 0003-2012 SM2 椭圆曲线公钥密码算法[S]. 北京: 国家密码管理局, 2012.

[4] Pravin, Z. and Raghavendra, D. (2022) Optimization of Elliptic Curve Scalar Multiplication Using Constraint Based Scheduling. *Journal of Parallel and Distributed Computing*, **167**, 232-239. https://doi.org/10.1016/j.jpdc.2022.05.006

[5] Wang, X.J. (2016) Speed and Area Optimized Parallel Higher-Radix Modular Multipliers. *Cryptology ePrint Archive*, **2016**, 53.

[6] Eyupoglu, C. (2015) Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths. *Procedia—Social and Behavioral Sciences*, **195**, 1860-1864. https://doi.org/10.1016/j.sbspro.2015.06.420

[7] 李斌, 周清雷, 陈晓杰, 等. 可重构的素域 SM2 算法优化方法[J]. 通信学报, 2022, 43(3): 30-41.

[8] 李凡, 李云峰, 翁天恒, 等. 基于 FPGA 的 SM2 点运算快速并行实现[J]. 电子测量技术, 2020, 43(15): 105-111.