工业和信息化部电子科学技术情报研究所
ELECTRONIC TECHNOLOGY INFORMATION RESEARCH INSTITUTE, MIIT.

本文引用格式：王云峰，王静，黄世伟. 一种两路并行调度的点乘算法硬件实现 [J]. 新型工业化，2014，4（7）：20-27.
DIO：10.3969/j.issn.2095-6649.2014.7.02

# 一种两路并行调度的点乘算法硬件实现 *

王云峰，王静，黄世伟
（厦门大学信息科学与技术学院，福建厦门，361005）

**摘要：** 针对 GF（2m）域椭圆曲线点乘算法计算速度慢，硬件实现成本高的问题，提出一种改进的并行调度算法以及专用的硬件调度结构。通过合并点乘算法中底层模乘、模逆及模平方算法，降低底层运算单元硬件成本。同时对点乘算法进行模乘和模平方两路并行调度分解，以提高运算速度。并针对点乘调度算法的特点，设计专用的硬件调度结构，以减少调度复杂性。StratixIII 系列的 FPGA 原型实现结果表明，通过这些方式使点乘运算速度得到提高。

**关键词：** MS 算法；点乘算法；并行调度；二进制有限域；

## Hardware implementation of a point multiplication algorithm based on parallel scheduling

Yunfeng Wang，Jing Wang，Shiwei Huang
（*School of Information Science and Engineering*，*Xiamen University*，*Xiamen*，*361005*，*China*）

**Abstract:** An improved parallel scheduling algorithm and specific hardware scheduling structure is presented in this paper for solving the problem of low speed and high hardware cost of point multiplication algorithm of elliptic curve over GF（2m）.Merging the underlying modular multiplication，modular inversion and modular square algorithm is used to reduce the cost of hardware implementation. Then a parallel scheduling has been made to point multiplication algorithm with modular multiplication and modular square for increasing computing speed. Besides，specific hardware scheduling structure is designed to reduce the complexity according to the feature of point multiplication. The result shows that point multiplication module introduced by this paper has reached an excellent cost performance both on speed and hardware resources on FPGA of StratixIII series.

**Key words:** Inversion-Multiply-Square（IMS）algorithm；Point Multiplication Algorithm；Parallel Scheduling；Binary Finite Fields；

## 0 引言

椭圆曲线密码体制（ECC，Elliptic Curve Cryptography）在信息安全[1-4]中应用广泛，而点乘算法决定了 ECC 的性能。因此点乘算法的硬件设计是目前 ECC 算法研究的重点。

目前已公开 GF（2m）域的点乘算法实现主要采用分别设计底层各域运算单元，然后通过协处理器或其它调用电路的方式来实现对底层各单元的功能调用。文献 [3] 采用最优正规基，提出了 bit-serial 和 bit-parallel 两种方式分别实现模乘算法，由于 bit-serial 方式在每个时钟单元

* 基金项目：国家自然科学基金项目（61274133），福建省自然科学基金项目（2010J05143）
作者简介：王云峰（1977-），男，副教授，博士，主要研究方向：信息安全与集成电路设计

只输出一位，致使模乘运算非常耗时，而 bit-parallel 方式虽在每个时钟周期可以完成一次模乘运算，但其占用面积大，硬件实现成本高。文献 [4] 针对顶层调用复杂的问题，提出了一种对 Montgomery 点乘算法并行调度分解的方法，跳过点加和点倍的调用过程，提高了顶层运算速度，但其底层四路并行调度慢，资源开销大。文献 [5] 针对有限域点乘算法及底层二进制乘法算法进行改进，将元素的相乘和取模多项式过程相结合，虽节省了硬件资源，但运算速度过慢。文献 [6] 基于高效点乘调度策略和 modified Montgomery 模乘算法设计了一种高性能可扩展公钥密码协处理器 RESA，可扩展性和灵活性较高，但是由于采用串行方式处理，运算时钟周期时间多，占用资源多。

本文在对上述点乘算法及其硬件实现的分析和比较的基础上，对点乘调度算法进行改进，通过专用硬件调度结构实现模乘和模平方的两路并行计算的调度。实验结果表明，采用这种方式既提高了点乘运算速度，又降低了硬件实现成本。

## 1 点乘算法分析和改进

### 1.1 点乘算法分析

点乘也叫标量乘法，是 ECC 的核心运算，也是最耗时的运算 [7]。常用的点乘算法包括通用点乘算法和固定点乘算法 [8]。通用点乘算法可以计算椭圆曲线上任意点的点乘，比如：Double-and-Add 点乘算法、w-ary 点乘算法、加减点乘算法、符号位 w-ary 点乘算法、固定窗加减点乘算法和 modified Montgomery点乘算法。而固定点乘算法只能计算已知点的点乘。考虑到实际应用，本文将设计通用点乘算法。

六种通用点乘算法中，modified Montgomery 点乘算法不需要预计算，存储点数少，计算复杂度小，且每次迭代点加和点倍的计算次数相同。二进制有限域上的点乘算法中，modified Montgomery 点乘算法是计算速度最快的算法 [3]，如算法 1 所示。

算法 1：modified Montgomery 点乘算法（射影坐标下）[3]

已知：大整数 $k>0$; $P=(x,y)\in E_{2m}(a,b)$;

求：$Q=k\cdot P$;

1: if $(k=0$ or $x=0)$ then

return $Q=(0,0)$;

2: $k=(k_{l-1},k_{l-2},\cdots,k_1,k_0)_2$;

3: $X_1=x$; $Z_1=1$; $X_2=x^4+b$; $Z_2=x^2$;

4: for $i=l-2$ to 0 do

4a: if $(k_i=1)$ then

$$\begin{cases}X_1=x\cdot(X_1Z_2+X_2Z_1)^2+X_1Z_2\cdot X_2Z_1;\\ Z_1=(X_1Z_2+X_2Z_1)^2;\end{cases}$$

$$\begin{cases}X_2=X_2^4+b\cdot Z_2^4;\\ Z_2=X_2^2Z_2^2;\end{cases}$$

else

$$\begin{cases}X_2=x\cdot(X_1Z_2+X_2Z_1)^2+X_1Z_2\cdot X_2Z_1;\\ Z_2=(X_1Z_2+X_2Z_1)^2;\end{cases}$$

$$\begin{cases} X_1 = X_1^4 + b \cdot Z_1^4; \\ Z_1 = X_1^2 Z_1^2; \end{cases}$$

$$5: y_1 = (\frac{X_1}{Z_1} + x) \cdot \left[ (\frac{X_1}{Z_1} + x)(\frac{X_2}{Z_2} + x) + x^2 + y \right] \cdot x^{-1} + y;$$

$$6: \text{return } Q = (x_1, y_1);$$

### 1.2 点乘算法改进

点乘算法主要是由点加和点倍算法构成的，而点加和点倍算法又通过调用底层有限域上的模加、模逆、模乘和模平方等算法来实现。如下图 1 所示。

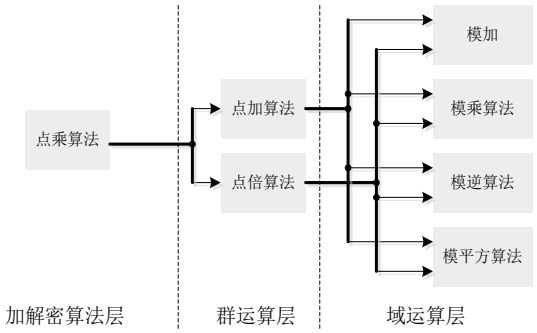


图 1 点乘算法的层次结构

Fig.1 Hierarchy of the point multiplication

由于点乘算法具有明显的层次性[9]，因此点乘运算调度中存在着并行调度的可能，通过使用并行调度可以明显加快点乘运算速度[3]。文献 [3，9，10] 均采用了并行调度的思想。因此本文基于文献 [3] 的设计思想，提出 IMS（Inversion-Multiply-Square）算法，如算法 2 所示。

该算法通过设计一个模块完成模逆、模乘和模平方运算，并能同时计算出模乘和模平方的运算结果。相对于分别设计底层域运算模块，节省了硬件资源，降低了设计复杂度。

其中当 $m/\bar{i}=0$ 时，作模逆运算，完成一次模逆需要 2m-1 次迭代；当 $m/\bar{i}=1$ 时，可同时进行模乘和模平方运算，且一次运算需要 m 次迭代。

算法 2：IMS 算法

已知：$F(x)=\sum_{i=0}^{m} f_i x^i;$

$$A(x)=\sum_{i=0}^{m-1} a_i x^i; \quad B(x)=\sum_{i=0}^{m-1} b_i x^i; \quad Q(x)=\sum_{i=0}^{m-1} q_i x^i;$$

求 $\begin{cases} A(x)^{-1} \bmod F(x); & \text{if } m/\bar{i}=0; \\ A(x) \cdot B(x) \bmod F(x);\ Q(x)^2 \bmod F(x); & \text{if } m/\bar{i}=1; \end{cases}$

$1: (j=0)$

$$S^0(x) = x \cdot A(x);$$

$$R^0(x) = F(x);$$

$$C^0(x) = \begin{cases} x^m; & \text{if } m/\bar{i}=0; \\ 0; & \text{if } m/\bar{i}=1; \end{cases}$$

$$H^0(x) = \begin{cases} 0; & \text{if } m/\bar{i}=0; \\ \{0, B(x)\}; & \text{if } m/\bar{i}=1; \end{cases}$$

$$d^0 = 2 = \{0, \cdots, 0, 1, 0\};$$

$$sign^0 = 1;$$

$$Q_H^0(x) = \{0, Q_H(x)\} = \left\{ 0, \sum_{i=\frac{m+1}{2}}^{m-1} q_i x^{2i-m} \right\};$$

$$Q_L(x)=\sum_{i=0}^{\frac{m-1}{2}} q_i x^{2i};$$

$$k=\begin{cases}2m-1; & \text{if } m/\bar{i}=0;\\ m; & \text{if } m/\bar{i}=1;\end{cases}$$

2 : for j = 1 to k do

$$e=s_m^{j-1}\cdot sign^{j-1};$$

$$sign^j=\begin{cases}\overline{s_m^{j-1}}; & \text{if } sign^{j-1}=1;\\ d_0^{j-1}; & \text{if } sign^{j-1}=0;\end{cases}$$

$$S^j(x)=x\cdot[S^{j-1}(x)+s_m^{j-1}\cdot R^{j-1}(x)];$$

$$C^j(x)=\begin{cases}C^{j-1}(x)+s_m^{j-1}\cdot H^{j-1}(x); & \text{if } m/\bar{i}=0;\\ C^{j-1}(x)+h_0^{j-1}\cdot S^{j-1}(x)/x; & \text{if } m/\bar{i}=1;\end{cases}$$

$$R^j(x)=\begin{cases}S^{j-1}(x); & \text{if } e=1\,\&\,m/\bar{i}=0;\\ R^{j-1}(x); & \text{if } e=0\mid m/\bar{i}=1;\end{cases}$$

$$H^j(x)=\begin{cases}C^{j-1}(x)/x; & \text{if } e=1\,\&\,m/\bar{i}=0;\\ H^{j-1}(x)/x; & \text{if } e=0\mid m/\bar{i}=1;\end{cases}$$

$$d^j=\begin{cases}2\cdot d^{j-1}; & \text{if } sign^j=1;\\ d^{j-1}/2; & \text{if } sign^j=0;\end{cases}$$

$$Q_H^j(x)=x\cdot Q_H^{j-1}(x)+q_{Hm-1}^{j-1}\cdot R(x);$$

$$3:\text{return}\begin{cases}H^k(x); & \text{if } m/\bar{i}=0;\\ C^k(x);\ Q_H^k(x)+Q_L(x); & \text{if } m/\bar{i}=1;\end{cases}$$

基于 modified Montgomery 算法[3]，并集合 IMS 算法的运算特点，本文提出了该算法并行调度分解，如算法 3 所示。由于 IMS 算法支持模乘和模平方的并行运算，因此在分解时，将点乘算法拆成能同时执行模乘和模平方的表达形式，形成模乘和模平方两路并行调度算法。对于模加运算，由于实现简单，因此在计算完模逆、模乘和模平方并将计算结果赋给相应的寄存器之前，可同时进行模加计算。从而实现了模加运算与模逆、模乘和模平方运算的并行执行。

算法 3：Modified montgomery 点乘算法的并行调度分解（射影坐标下）

已知：大整数 k>0；P=（x，y）；其中，P 是所选择的椭圆曲线上的点。

求：k · P

0 $x=G_x=x;\ y=G_y=y;\ X_1=G_x=x;\ Z_1=1;\ t=1;\ X_2=0;\ Z_2=G_x=x;\ K=k;$

1 NOP $Z_2=Z_2^2;$

2 NOP $X_2=t+Z_2;$

3 NOP $X_2=X_2^2;$

4 for $i$=$l$−2 to 0 do

if $(k_i=1)$ then

5 $\ddot{e}=X_1\cdot Z_2;$ $t=X_2^2;$

6 $X_1 = X_2 \cdot Z_1; Z_1 = \lambda + X_1;$ $Z_2 = Z_2^2;$

7 $X_1 = X_1 \cdot \lambda; X_2 = t + Z_2;$ $Z_1 = Z_1^2;$

8 $\underline{\underline{\lambda}} = x \cdot Z_1; X_1 = \underline{\underline{\lambda}} + X_1;$ $X_2 = X_2^2;$

9 $Z_2 = t \cdot Z_2;$ NOP

10 else swap($\{X_1, Z_1\}, \{X_2, Z_2\}$);

11 $t = X_1 \cdot Z_2;$

12 $\underline{\underline{\lambda}} = x \cdot Z_1; X_1 = X_1 + \underline{\underline{\lambda}};$ $\underline{\underline{\ddot{e}}}$只是临时变量，不存入$\ddot{e}$中。下同。

13 $X_1 = X_1 \cdot X_2;$

14 $\lambda = x \cdot t; X_1 = \lambda + X_1;$

15 $t = Z_1 \cdot Z_2;$

16 $\underline{\underline{Z_2}} = y \cdot t; X_1 = \underline{\underline{Z_2}} + X_1;$

17 $t = x \cdot t;$

18 $Z_2 = t^{-1};$

19 $\lambda = \lambda \cdot Z_2; x = x + \lambda;$

20 $X_1 = X_1 \cdot Z_2;$

21 $\underline{\underline{t}} = x \cdot X_1; y = y + \underline{\underline{t}};$

22 $x_1 = G_x = \ddot{e};$ $y_1 = G_y = y;$

在算法 3 中，$x$、$y$、$\ddot{e}$、$X_1$、$Z_1$、$t$、$X_2$、$Z_2$ 等为中间寄存器变量，最终运算结果$(x_1, y_1)$分别从 $G_x$ 和 $G_y$ 中取出。该算法总共需要 5 ×（L−1）+14 次模乘时间即可完成，其中一次模逆时间作两次模乘考虑；L 值为整数 k 的有效长度，它决定了该算法需要迭代的次数，而每次迭代所用时间决定了该点乘算法的计算速度。本文提出的两路并行调度法，每次迭代需要 5 次模乘时间，比文献 [3] 中所采用的两路并行调度法的运算时间更少（其为 6 次模乘时间），提高了计算速度，其次由于单元合并，故资源开销只需要 1 个模乘单元和 1 个模加减单元，硬件实现成本更低。与文献 [6] 调度算法相比，本文实现的点乘算法跳过点加和点倍直接调用底层域运算，不仅硬件资源占用少，而且时间周期数减少了 28.81%，中间变量所需的存储空间也减少了 33.33%。

## 2 点乘算法的硬件实现

### 2.1 IMS 模块硬件实现

由于本文提出的 IMS 算法可同时支持模逆、模乘和模平方运算，为了实现该功能，设计了如图 2 所示的两种 AND-XOR 结构。基于这两种结构，本文最终实现的 IMS 硬件结构单元如图 3 所示，其中 msel_bits 信号用来控制阈值选择；该结构中，AND-XOR Unit-1 与 AND-XOR Unit-3 为普通 AND-XOR 电路，改进的复合电路（AND-XOR Unit-2）与 AND-XOR Unit-1 电路组合，可以实现模逆和模乘运算的复用。而 AND-XOR Unit-3 电路用以实现模平方运算，这样在不增加寄存器组的条件下，添加了模平方功能，最终在一个硬件单元中实现了模逆、模乘和模平方运算，且模乘和模平方运算可同时执行。这种方式比分别设计各域运算单元更节省硬件资源，也降低了控制电路设计的复杂性，其中控制电路只需对迭代次数计数器设置不同的初始值即可，且不需要为每种域运算设计不同的控制状态机。为快速实现 IMS 算法，本文将 4 个 IMS 基本单元进行串接，形成一个局部并行全局串行的硬件结构，从而使计算速度提高了 4 倍。

### 2.2 点乘调度算法的硬件实现

点乘算法一般采用协处理器的方式来实现，文献[11]提出的协处理器方式具有一定的通用性，但当域值较大时，会占用大量硬件资源。若采用 32/64 位协处理器来实现，则在点乘及其底层各

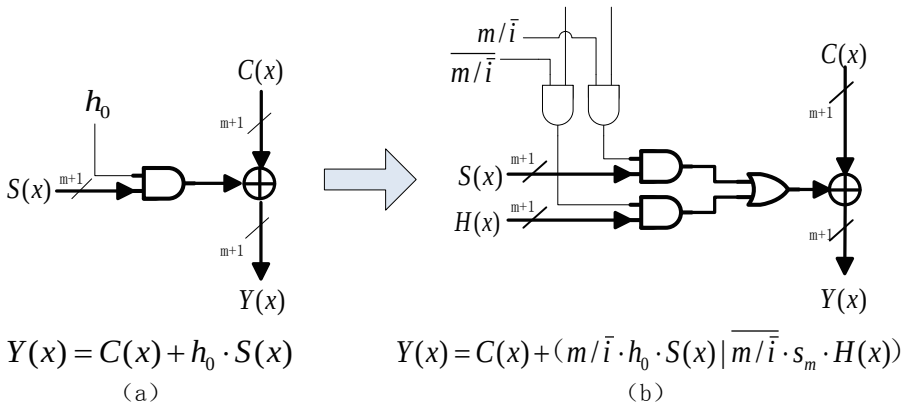


图 2　普通 AND–XOR 电路（a）及改进的复合 AND–XOR 电路（b）

Fig.2 General AND–XOR circuit (a) and modified AND–XOR circuit (b)

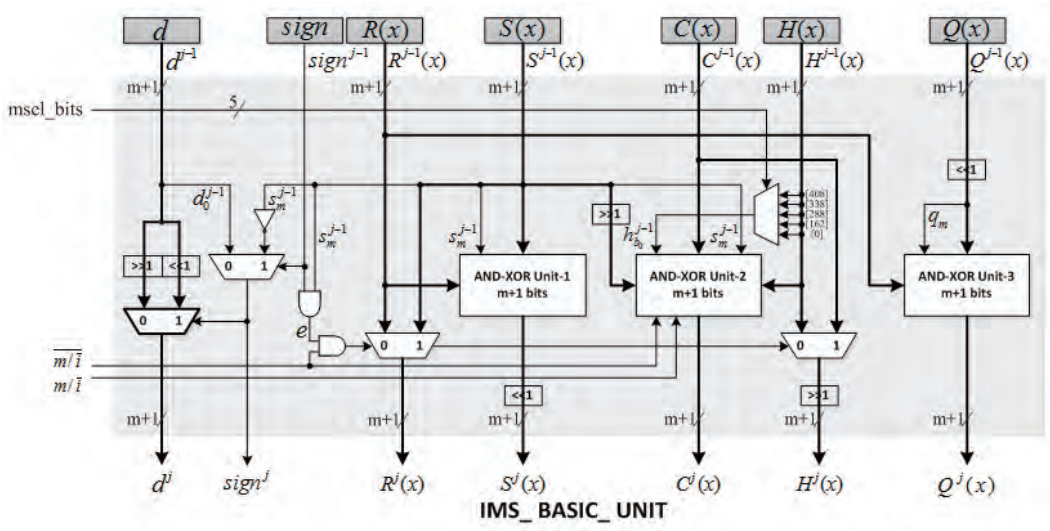


图 3 IMS 基本单元结构图

Fig.3 The basic architecture of IMS

域运算过程中，每次迭代后都要进行大量的数据搬移，严重限制了点乘运算速度。因此本文根据点乘调度算法的运作过程，通过下述“寻迹”的方式来实现点乘算法。

首先，为点乘调度算法各步骤的计算表达式设计专门的硬件通路。例如，点乘调度算法步骤 6 中有 $X_1=X_2\cdot Z_1$，$Z_1=\ddot{e}+X_1$ 和 $Z_2=Z_2^2$ 两个并行计算表达式，其中，$X_1=X_2\cdot Z_1$，$Z_1=\ddot{e}+X_1$ 需要先设计输入多路器使 $X_2$ 和 $Z_1$ 进入 IMS 模块中进行模乘运算，再设计输出多路器将运算结果存储在 $X_1$ 中，同时还要设计异或电路使运算结果与 $\ddot{e}$ 模加后存入 $Z_1$ 中。对于 $Z_1$=$Z_1^2$ 模平方运算也同样需要设计相应的硬件通路来满足运算需求。然后，根据设计的各通路的相似程度，尽可能采用复用的方式对通路进行重组合并，减少通路个数，降低通路硬件面积，以达到最好的优化效果。

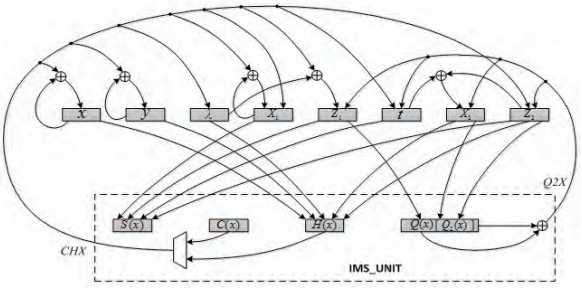


图 4　点乘硬件通路

Fig.4 Hardware path of the point multiplication

最终点乘硬件通路如图 4 所示。图中，$x$、$y$、$\ddot{e}$、$X_1$、$Z_1$、$t$、$X_2$、$Z_2$ 为调度算法的

中间变量。$S(x)$、$C(x)$、$H(x)$、$Q(x)$ 为 IMS 模块的寄存器变量。

## 3 性能分析与比较

本文设计用 verilog 语言编码实现。为了与以往研究相比较，我们在 Altera 公司 StratixIII–EP3SL340H1152 开发平台上实现了点乘算法的 FPGA 原型，并通过参数化的模式配置方式来满足不同域值长度的需求，以实现 163bit、233bit、283bit、409bit 和 571bit 等域值的 ECC 加解密系统。表 1 给出了本文设计的点乘运算单元在最坏情况下（即 k=163）工作的最高时钟频率、吞吐率、运算时间、所占资源以及每秒可计算的次数等相关数据。表 2 给出本文与其他文献的比较结果。

表 1　点乘运算单元的 FPGA：StratixIII–EP3SL340H1152 原型实现结果

Tab.1 Implementation results of the point multiplication on the StratixIII–EP3SL340H1152 devices

| 位长 | 时钟频率 /MHz | 吞吐率 /Kbps | 运算时间 /ms | 逻辑单元 /ALUTs | 寄存器 /bits | 每秒计算次数 |
|---|---|---|---|---|---|---|
| 163 | 177.71 | 810.054 | 0.201 | 8500 | 4516 | 4969 |
| 233 | 169.98 | 549.456 | 0.424 | 10830 | 5912 | 2358 |
| 283 | 167.9 | 454.613 | 0.623 | 12622 | 6913 | 1606 |
| 409 | 163.72 | 309.306 | 1.322 | 17136 | 9432 | 756 |
| 571 | 154.39 | 211.699 | 2.697 | 22958 | 12677 | 370 |

表 2　点乘运算单元的 FPGA 原型实现比较

Tab.2 Comparison related works for FPGA implementations of point multiplication

| 文献 | 时间 | 域阶 | 器件 | 吞吐率 /Kbps | 最大频率 /MHz | 点乘时间 /ms | 资源 | |
|---|---|---|---|---|---|---|---|---|
| [5] | 2009 | 191 | Cyclone II–EP2C8Q208C8N | 18.19 | 82.56 | 10.5 | 7365 ALUTs | |
| [4] | 2010 | 163 | StratixIII–EP3SL340H1152 | 168.563 | 102 | 0.967 | 28336 ALUTs | 33278 regs |
| | | 163 | | 303.538 | 100.2 | 0.537 | 54187 ALUTs | 36572 regs |
| | | 163 | | 567.944 | 99.4 | 0.287 | 94293 ALUTs | 68753 regs |
| [6] | 2011 | 163 | | 812.157 | 250 | 0.2007 | 157.3 kgates | |
| [3] | 2013 | 233 bit–serial | StratixIII–EP3SL150F1153C | 143.83 | 276.24 | 1.62 | 8799 ALUTs | 7143 regs |
| | | 233 bit–parallel | | 9320 | 108.96 | 0.025 | 61023 ALUTs | 5133 regs |
| 本文 | 2014 | 163 | StratixIII–EP3SL340H1152 | 810.054 | 177.71 | 0.201 | 8500 ALUTs | 4516 regs |

表 2 中，文献 [3] 提出 bit–serial 和 bit–parallel 两种方式实现模乘，bit–serial 虽然占用资源较少，但是点乘速度慢，而 bit–parallel 方式虽然很大程度上缩短点乘运算时间，但使用了大量硬件资源。文献 [4] 方案对点乘算法进行两路和四路的并行调度分解，但未提出底层运算单元的优化方法，而且仍采用并行调用底层单元的方式来实现点乘运算，虽然点乘速度较快，但硬件面积大，实现成本高。由数据结果可以看出，本文设计的 ECC 模块运算速度较快，资源占用少，在已公开的文献中，性价比最高。

## 4 总结

本文针对传统点乘运算速度慢，硬件实现成本高的缺点，首先对底层算法进行改进合并，提出一种能够实现模逆、模乘和模平方运算的 IMS 合并算法以及相应的改进电路，该算法的关键点在于能够同时进行模乘和模平方运算，这为点乘算法的并行调度分解提供了基础和依据，并降低了硬件实现成本。然后通过对点乘算法的并行调度分解，并根据算法的运作过程设计专用硬件调度结构，降低了点乘的运算时间，提高了运算效率。数据结果表明，本文的设计显著提高了点乘运算的整体性能，使速度和面积达到了较好的平衡，能够适用于设计不同域值的高速 ECC 加解密系统及其信息安全的应用。

## 参考文献

[1] 王小亮，刘彬，王春露 . 云计算可信机制的有效性评估方法研究 [J]. 新型工业化，2012，2（12）：47-55
WANG Xiaoliang，LIU Bin，WANG Chunlu.Quantization and Assessment the Effectiveness of Cloud Trusted Mechanism[J].The Journal of New Industrialization，2012，2（12）：47-55. WANG Xiaoliang，LIU Bin，WANG Chunlu.Quantization and Assessment the Effectiveness of Cloud Trusted Mechanism[J].The Journal of New Industrialization，2012，2（12）：47-55.

[2] 沈志东 . 基于可信链的移动代理安全增强方法 [J]. 新型工业化，2012，2（9）：37-42.
SHEN Zhidong. The improved securtiy for mobile agent based on trusted link[J].The Journal of New Industrialization，2012，2（9）：37-42.

[3] Urbano-Molano F A，Trujillo-Olaya V，Velasco-Medina J. Design of an elliptic curve cryptoprocessor using optimal normal basis over GF（2233）[C]// 2013 IEEE Fourth Latin American Symposium on Circuits and Systems（LASCAS）. Cusco：IEEE，2013：1-4.

[4] 赵前进 . 椭圆曲线密码点乘算法的并行调度研究 [D]. 解放军信息工程大学，2010.
Q J Zhao. Research on Parallel Schedule of Scalar Multiplication of Elliptic Curve Cryptography[D]. PLA Information Engineering University，2010.

[5] 杨自恒，周平，刘佳，丁群 . 基于 FPGA 的椭圆曲线点乘算法设计与实现 [J]. 仪器仪表学报，2009，30（7）：1546-1551.
Z H Yang，P Zhou，J Liu，Q Ding. Improvement and implementation of the algorithm design of elliptic curve dot product based on GF（2n）[J]. Chinese Joumal of Scientific Instrument，2009，30（7）：1546-1551.t

[6] 黎明，吴丹，戴葵，邹雪城 . 高性能可扩展公钥密码协处理器研究与设计 [J]. 电子学报，2011，39（3）：665-670.
M Li，D Wu，K Dai，X C Zou. Research and Design of a High-Performance Scalable Public-Key Cipher Coprocessor[J]. Acta Electronica Sinica，2011，39（3）：665-670.

[7] P G Shah，X Huang，D Sharma. Sliding window method with flexible window size for scalar multiplication on wireless sensor network nodes[C] // International Conference on Wireless Communication and Sensor Computing（ICWCSC）. Chennai：IEEE，2010：1-6.

[8] 卫学陶 . GF（2~ m）域上可配置椭圆曲线密码系统的设计与硬件实现 [D]. 解放军信息工程大学，2007.
X T Wei. [D]. Design and Hardware Implementation of Reconfigurable Elliptic Curve Cryptography Over GF（2m）.PLA Information Engineering University，2007.

[9] 王庆先 . 有限域运算和椭圆曲线数乘运算研究 [D]. 电子科技大学，2006.
Q X Wang. Research on Finite Field Arithmetic and Scalar Multiplication of Elliptic Curve Cryptographys[D]. University of Electronic Science and Technology of China，2006.

[10] 赵前进，李西萍，戴紫彬，张永福 . NAF 点乘算法的并行计算研究 [J]. 电子技术应用，2010（7）：160-162.
Q J Zhao，X P Li，Z B Dai，Y F Zhang. Research for parallel computation on NAF scalar multiplication[J]. Application of Electronic Technique，2010（7）：160-162.

[11] Jarvinen K，Skytta J. On parallelization of high-speed processors for elliptic curve cryptography[J]. Very Large Scale Integration（VLSI）Systems，IEEE Transactions on. 2008，16（9）：1162-1175.