第32卷 第23期 计 算 机 工 程 2006年12月
Vol.32 No.23 Computer Engineering December 2006
·安全技术· 文章编号：1000—3428(2006)23—0156—03 文献标识码：A 中图分类号：TP311

基于FPGA的有限域求逆算法的改进及实现

鲍可进，宋永刚
（江苏大学计算机学院，镇江 212013）

摘 要：介绍了椭圆曲线密码和超椭圆曲线密码算法中一个重要的模块——求逆模块。分析并比较了现有的3种求逆算法，提出了针对FPGA快速实现的改进算法。根据改进的算法设计了求逆的硬件框图，并用VHDL实现了该设计。该设计使用Altera公司的Quartus II软件在EP1S10F780C6上进行编译、综合、布局布线。实验结果证明，该改进的算法无论在速度上还是在芯片面积上都具有比以往的算法更优秀的性能。

关键词：FPGA；椭圆曲线密码；超椭圆曲线密码；有限域；逆

Optimizing and Realization of the Finite Field Inversion Algorithm Based on FPGA

BAO Kejin, SONG Yonggang
(College of Computer Science, Jiangsu University, Zhenjiang 212013)

【Abstract】 Finite field inversion, an important module of elliptic curve cryptosystems and hyper-elliptic curve cryptosystems, is introduced. Through analyzing and comparing three inversion algorithms that have been invented, an optimized fast algorithm based on FPGA is put forward. According to the optimized algorithm, an inversion module is designed and realized with VHDL. This design is compiled, synthesized, and fitted into Altera's EP1S10F780C6 FPGA, using its QuartusII software. The result indicates that this optimized algorithm has advanced performance in both speed and area than other past algorithms.

【Key words】 FPGA; ECC; HECC; Finite field; Inversion

有限域求逆运算是实现椭圆曲线密码（ECC）和超椭圆曲线密码（HECC）的基础。快速实现有限域求逆是实现ECC和HECC的关键。由于HECC具有比ECC更高的加密强度，可以使用更少的带宽，在更小的域上运算等一系列优点，近年来已有大量的研究集中于HECC的实现上。而要实现HECC则对有限域求逆提出了更高的要求。此外，由于对硬件加密系统可重配置的要求，基于FPGA实现HECC已经成为目前研究的热点之一。

以往已有一些论文专门论述了HECC中有限域求逆的快速算法。实现的算法主要有3种：(1)利用费尔马定理的反复平方相乘算法；(2)扩展的欧几里德算法（EEA）；(3)Modified Almost求逆算法（MAIA）。第1种算法由于计算量太大已基本不被采用。后两种算法是目前常用的算法，但是这两种算法仍不能满足HECC对求逆速度的更高要求。本文分析了两种算法的特点，通过改进MAIA算法提出了适合FPGA实现的快速算法。并针对GF(2^{83})和GF(2^{113})域给出了有限域快速求逆的FPGA实现。

1 相关知识及研究进展

系统讨论伽罗瓦域$GF(2^n)$属于近似代数的内容，可以参考的书很多。这里仅简要介绍论文中用到的$GF(2^n)$域和与求逆运算有关的背景知识。$GF(2^n)$域包含2^n个元素，它是$GF(2)$（只含有0，1两个元素的有限域）的扩域。所有的$GF(2^n)$都有一个零元，一个单位元，和至少一个本原元素$f(x)=x^n+p_{n-1}x^{n-1}+p_{n-2}x^{n-2}+\cdots+p_1x+p_0$。这个本原元素是一个不可约的多项式，域中的其它元素都可以由本原元素生成。所以$GF(2^n)$又可以表示成$GF[2, f(x)]$。因此，在实现$GF(2^n)$的运算时可以把它理解成多项式的集合：

$$GF(2^n)=\{A=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_1x+a_0, a_j\in GF(2), 0\le j\le n-1\} \quad (1)$$

这种多项式表示法非常适合有限域的运算，加(+)减(-)即按位异或，乘法（*）则通过多项式相乘和模$f(x)$来实现。除法（/）和求逆（inverse）可由加法和乘法运算得到。

如有限域$GF(2^4)$的本原多项式$f(x)=x^4+x+1$，若a的逆是a^{-1}，则有$aa^{-1}=1 \bmod f(x)$。设$a=x^2+x$，因为

$$\begin{aligned}(x^2+x)(x^2+x+1)&=x^4+x^3+x^2+x^3+x^2+x\\&=x^4+x\\&=x+1+x\\&=1\end{aligned}$$

所以，x^2+x的逆就是x^2+x+1。

反复平方相乘求逆算法的原理是费尔马（Fermat）定理。根据费尔马定理：对于$GF(2^m)$有$a^{2^m-1}=1 \bmod (f(x))$成立；可得$a^{-1}=a^{2^m-2} \bmod (f(x))$。

这样，求逆共需$2^{m-1}-1$次域的平方运算。所以这种算法运算量非常大，效率很低，只适合于基域很小的简单应用，如RS码译码器等。

EEA和MAIA算法都是通过反复迭代来计算有限域的逆。基本思想都是：a和$f(x)$反复各乘以或除以x并相加，

作者简介：鲍可进(1958—)，男，副教授，主研方向：嵌入式网络控制；宋永刚，硕士生

收稿日期：2006-01-10 **E-mail：**bkj@ujs.edu.cn

同时将 1 和 0 作同样的变换。这样，当 a 变成 1，1 就变成 a 的逆。其理论依据是多项式和有限域原理。具体算法如下：

算法 1：EEA Field Inversion

Input: $a \in GF(2^n)$

Output: $b = a^{-1} \in GF(2^n)$

1 $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f$ ；

2 While $\deg(u) \neq 0$

2.1 $j \leftarrow \deg(u) - \deg(v)$ ；

2.2 if ($j < 0$) then $u \leftrightarrow v, b \leftrightarrow c, j \leftarrow -j$ ；

2.3 $u \leftarrow u + (v << j), b \leftarrow (c << j)$ ；

3 return b ；

算法 2：MAIA Field Inversion

Input: $a \in GF(2^n)$

Output: $b = a^{-1} \in GF(2^n)$

1 $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f$ ；

2 While x divides u do:

2.1 $u \leftarrow u / x$ ；

2.2 if x divides b then $b \leftarrow b / x$ ；

else $b \leftarrow (b + f) / x$ ；

3 If $u = 1$ then return(b)；

4 If $\deg(u) < \deg(v)$ then: $u \leftrightarrow v, b \leftrightarrow c$ ；

5 $u \leftarrow u + v, b \leftarrow b + c$ ；

6 Goto step(2)；

研究进展：有限域求逆作为 ECC 和 HECC 的一个基本模块，无论是在国内还是在国外都有大量的研究。Cantor 在他的一篇硕士论文“Analysis of FPGA-based hyperelliptic curve cryptosystems”中讨论了有限域的求逆，他使用的是第 1 种算法。Wollinger 在研究 HECC 的课题中也多次研究了有限域求逆的实现，并得出了基于 FPGA 的有效数据。国内也有很多论文专门讨论有限域的求逆，然而通过改进算法针对 FPGA 实现的讨论并不多。

2 EEA 和 MAIA 的分析及硬件结构

两种算法的主要区别是在迭代时分别采用了高位优先消去和低位优先消去的两种方案。从算法实现上来看，EEA 通过每一次左移对齐最高位，从而相加消去最高位，最终使 u 寄存器达到 1。MAIA 则每次判断 u 最低位是否为 0，若为 0 则右移，否则与 v 相加后使最低位为 0 再进行右移。经过若干次右移最终使 u 寄存器达到 1。

(1)EEA 算法分析

对于两个多项式：

$$u = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{2}$$

$$v = x^m + b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0 \tag{3}$$

若 $m > n$，即 $\deg(u) < \deg(v)$，则：$u \leftrightarrow v$，$u \leftarrow v << (m-n)$；

$$u = b'_{m-1}x^{m-1} + b'_{m-2}x^{m-2} + \cdots + b'_1 x + b'_0 \tag{4}$$

$$v = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{5}$$

如果 $b_{m-1}\cdots b_{m-j+1} = 0,\ b_{m-j} \neq 0$，则：若 $(m-j) > n$，则 $u \leftarrow v << (m-j-n)$。

可以看出，u 每次至少可以消去 1 个最高次项，每次消去 2 个最高次项的条件是 $a_{n-1} = b_{m-1}$，每次消去 3 个最高次项的条件是 $a_{n-1} = b_{m-1}$ 且 $a_{n-2} = b_{m-2}$。因为 $a_{n-1,}, a_{n-2}, b_{m-1}, b_{m-2} \in \{0,1\}$，所以概率 $p(a_{n-1} = b_{m-1}) = 1/2$，即每次消去 2 个最高次项的概率为 $\frac{1}{2}$。又因为 $p((a_{n-1} = b_{m-1}) \cap (a_{n-2} = b_{m-2})) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$，所以每次消去 3 个最高次项的概率为 $\frac{1}{4}$。以此类推。以 $GF(2^{83})$ 为例，设每次迭代可消去最高次项的个数为 c，概率为 $p(c)$，其分布率如表 1 所示。

表 1 $p(c)$ 概率分布表

c	1	2	3	4	...	81	82	83
$p(c)$	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	…	$\frac{1}{2^{80}}$	$\frac{1}{2^{81}}$	$\frac{1}{2^{82}}$

c 的均值为

$$Ec = 1\times 1 + 2\times\frac{1}{2} + 3\times\frac{1}{4} + \cdots + 81\times\frac{1}{2^{80}} + 82\times\frac{1}{2^{81}} + 83\times\frac{1}{2^{82}} \tag{6}$$

$$\frac{1}{2}\times Ec = 1\times\frac{1}{2} + 2\times\frac{1}{4} + 3\times\frac{1}{8} + \cdots + 81\times\frac{1}{2^{81}} + 82\times\frac{1}{2^{82}} + 83\times\frac{1}{2^{83}} \tag{7}$$

式(6)-式(7)

$$\begin{aligned}\frac{1}{2}\times Ec &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{81}} + \frac{1}{2^{82}} - 83\times\frac{1}{2^{83}} \\ &= 2 - \frac{1}{2^{82}} - 83\times\frac{1}{2^{83}} \\ &\approx 2\end{aligned}$$

所以 $Ec \approx 4$。

由于在迭代过程中 u 和 v 要进行交换，也就是说 u 和 v 的次数都要降低，因此每次迭代平均可消去的最高次项数为 Ec 的 $1/2$，即 2，完成求逆运算需要的平均迭代次数为 $83/2 = 42$ 次。实现 EEA 的硬件结构见图 1。

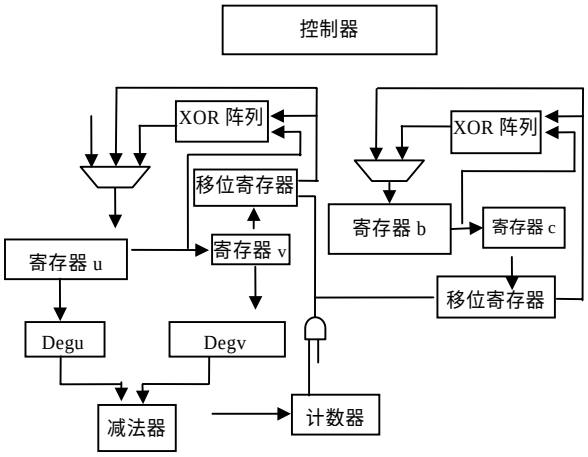

图 1 EEA 算法实现的硬件框图

(2)MAIA 算法分析

对于两个多项式：

$$u = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{8}$$

$$v = x^m + b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0 \tag{9}$$

u 每次右移（While 语句）可消去最高次的项数至少为 1，同时消去 2 项的概率为 $\frac{1}{2}$。但考虑到 $u \leftrightarrow v$ 和 $u \leftarrow u + v$ 操作，每次平均可消去的项数为 2。分析方法同 EEA。但是，MAIA 算法每次右移时只能移一位，下一次能否继续右移需要判断最低位是否为 0。如果不能右移，就要判断 $\deg(u)$ 和 $\deg(v)$ 的大小，并且 $u = u + v$。因为 v 的最低位一定是 1，所以 $u = u + v$ 的值至少可以右移一位。MAIA 对应的硬件结构见图 2。

(3)两种算法的比较

在每次迭代中，EEA 算法和 MAIA 算法都可以至少消去一个最高次项，且都需要移动若干位。二者都不需要进行复

杂的乘法或除法，容易用硬件来实现。从上面的分析还可以看出，EEA 算法移动 j 位平均可以消去的项数是 2 项。虽然在 MAIA 算法中的每一次右移操作都能消去一项，但是可以连续进行的概率也是满足 EEA 的概率分布关系。所以两种算法每次迭代平均可以消去的项数是相等的，都是 2。

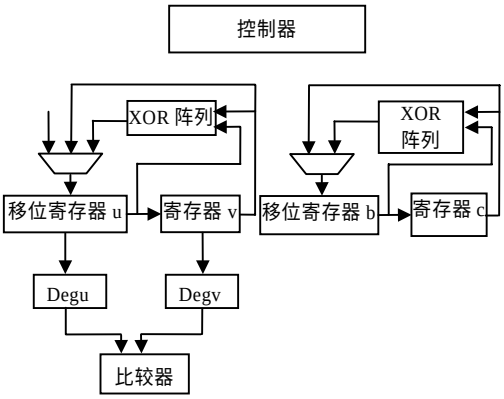

图 2 MAIA 算法实现的硬件框图

理论上虽然 EEA 算法可以利用桶式移位器在一个周期内移动 j 位，但是 j 的平均值只有 2，而超过 4 的概率就只有 $\frac{1}{16}$ 了。对于 $GF(2^{83})$ 以上的求逆，由于输入的数据宽度很大，用 FPGA 实现桶式移位器的代价极高，这是由 FPGA 以查找表来实现组合逻辑功能的结构决定的。实验表明，在 stratix EP101s 器件上实现 32 位的桶式移位器需要 200 个功能单元，且主频只能达到 50MHz。在 FPGA 中实现 83 位的桶式移位器是不能忍受的。所以，现有的实现都是采用图 1 所示的用计数器控制移位次数的方法来实现的。这样，运算速度就难以提高。MAIA 算法的 while 循环与 EEA 的左移 j 位情况相同，效率也相当。但是，MAIA 不需要保留移位前的值，并且不需要用计数器，因此结构简单，容易实现。

3 算法的改进及实验结果

由于实现 HECC 需要各个部件有更高的运算速度，因此如何改进算法一直是人们研究的热点问题之一。应该指出的是，运算速度的提高都是以芯片面积的增加作为代价的。实现快速的求逆运算就是要使各个运算部件尽量地并行起来，最有效的方法就是优化算法的循环部分。根据上面的分析可知，对于 MAIA 算法，由于其采用的是右移消去最高项的方法，因此可以根据寄存器最低位的不同情况将循环展开。本文算法的第 1 个改进就是根据 u 的最低两位将运算分为 3 种情况。这样，如果 u 能够连续 2 次除以 x 则使其在一个周期内完成。另外，由于在 $u=u+v$ 操作之前已使 $\deg(u)\geq\deg(v)$，因此根据移位的情况预测 $\deg(u)$ 和 $\deg(v)$ 的变化情况。本文的第 2 个改进是在初始化时得到 $\deg(v)$，在后面的运算中通过 $\deg(u)$ 和 $\deg(v)$ 的交换获得更新 $\deg(v)$ 的值。因为获取 $\deg(v)$ 的组合电路也需要占用很多的逻辑单元，所以此项改进能够明显减小占用的芯片面积。改进后的算法如下：

算法 3：Modified MAIA Field Inversion

Input: $a\in GF(2^n)$

Output: $b=a^{-1}\in GF(2^n)$

1 $b\leftarrow 1, c\leftarrow 0, u\leftarrow a, v\leftarrow f, \deg\_v\leftarrow\deg(f),$
$b\leftarrow b>>2, f_1\leftarrow f>>1, f_2=f>>2$；

2 case $u[1..0]$ is

2.1 when “00” =>

$u\leftarrow u>>2$;

$b\leftarrow b>>2+f_2\bullet b[0]$
$+f_1\bullet(\overline{b[1]}f[1]+b[1]\overline{b[0]}+\overline{b[1]}b[0]f[1])$；

goto 2;

2.2 when “10” =>

$u\leftarrow u>>1$;

$b\leftarrow b>>1+f_1\bullet b[0]$;

2.3 when others =>;

end case;

3 if $u=1$ then return b;

4 if $\deg(u)<\deg\_v$ then $u\leftrightarrow v; b\leftrightarrow c; \deg\_u\leftrightarrow\deg\_v$;

5 $u\leftarrow u+v; b\leftarrow b+c$;

6 Goto step(2)；

上述算法中移位寄存器只需考虑移 2 位和移 1 位两种情况。使用 FPGA 实现此算法时可以设计最简单（2 种情况）的桶式移位器。桶式移位器用 $u[1],u[0]$ 译码选择移位的位数，移位均在一个周期内完成。这样，对于两位同时为 0 这种出现概率较大的情况可以节省一个周期。当然，更快速的算法可以考虑同时移 3 位以上的情况，但是实现起来比较复杂，而且连续 3 位以上全为 0 的概率会成倍减小。本文在设计中采用只判断低两位的情况。因为需要在一个周期内得出 $\deg(u)$ 的值，所以求取 $\deg(u)$ 的组合电路不可缺少。那么，在算法的开始求取 $\deg(u)$ 的初值而在过程中通过减法更新 $\deg(u)$ 的方法并不可取。因此，此算法把 $u\leftrightarrow v; b\leftrightarrow c;\ \deg\_u\leftrightarrow\deg\_v$ 的条件表示为 $\deg(u)<\deg\_v$，目的是为了表明 $\deg\_u$ 的值是通过 deg() 计算出的，而 $\deg\_v$ 的值不需要使用 deg() 来计算。没有采用减法更新 $\deg\_u$ 的另一个原因是考虑可能存在 $\deg\_u=\deg\_v$ 的情况。若 $\deg\_u=\deg\_v$ 成立，由于 $u\leftarrow u+v$，则还需要考虑 $\deg\_u$ 减小的情况，而判断这种情况很难。实现此算法的硬件结构如图 3 所示。

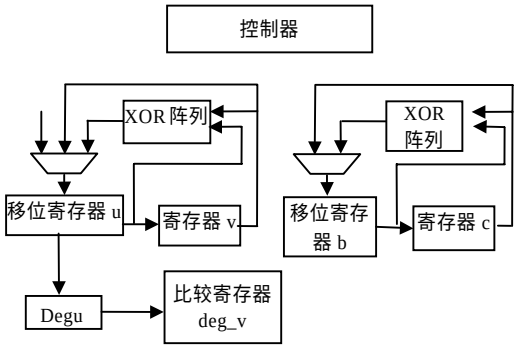

图 3 改进算法实现的硬件框图

在图 3 中，移位寄存器 u 和移位寄存器 b 都比前面两种结构的复杂。这便是所谓的“面积换取时间”。在设计中采用了列状态表的方法来分析控制器的时序，根据状态表来设计状态机。程序使用 VHDL 语言编写。整个 $GF(2^{83})$ 求逆器设计在 Altera 公司 Quartus II-4.1 上进行编译，并针对 EP1S10F780C6 FPGA 芯片进行综合、布局布线后，最大工作频率可以达到 100MHz，占用的总逻辑单元为 683 个。以 0xabcd667 求逆(逆为 0x1618675e10277a77b18f1)为例，根据 3 种算法分别设计的求逆器性能对照见表 2。

（下转第 170 页）