论文题目 椭圆曲线密码算法的FPGA设计与实现

| | |
|---|---|
| 专业学位类别 | 工 程 硕 士 |
| 学　　号 | 201550101050 |
| 作 者 姓 名 | 韩炼冰 |
| 指 导 教 师 | 阎　波　教 授 |

分类号 _____ 密级 _____

UDC [注1] _____

# 学 位 论 文

## 椭圆曲线密码算法的 FPGA 设计与实现

（题名和副题名）


韩炼冰

（作者姓名）

指导教师 阎 波 教 授

电子科技大学 成 都

刘振钧 高工（研究员级）

中国电科第三十研究所 成 都

（姓名、职称、单位名称）


申请学位级别 硕士 专业学位类别 工 程 硕 士

工程领域名称 信息与通信工程

提交论文日期 2018.04 论文答辩日期 2018.05

学位授予单位和日期 电子科技大学 2018 年 06 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

# THE DESIGN AND IMPLEMENTATION OF FPGA OVER ELLIPTIC CURVES CRYPTOGRAPHY

A Master Thesis Submitted to

University of Electronic Science and Technology of China


Discipline: **Master of Engineering**

Author: **Han Lianbing**

Supervisor: **Professor Yan Bo**

School: **School of Information and Communication Engineering**

## 独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：_____ 日期：2018年 6月 6 日

## 论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期：2018 年 6 月 6 日

# 摘　要

随着信息技术和计算机网络的快速发展，信息安全问题越来越引起关注。如何能够保证计算机系统和通信网络中信息的安全、保密、真实和完整，是信息安全技术主要的研究任务。信息安全的核心是密码技术，公钥密码体制作为密码技术的一个分支，能够有效地解决公共信道上的数字签名、身份认证、密钥分发等问题。椭圆曲线密码作为一种公钥密码体制，以其安全性高、计算量小、密钥短、存储空间占用小、处理速度快、带宽要求低等优点，已逐渐成为了下一代公钥密码标准，具有良好的使用前景。

本文对椭圆曲线密码算法的原理进行了深入分析，对椭圆曲线密码算法的FPGA 实现进行了研究设计。按照层次化、模块化的设计思想给出了椭圆曲线密码算法的 FPGA 实现方案。论文的主要内容如下：

1、研究了椭圆曲线密码算法中的数学原理，分析和对比多种实现方法，找出适合于 FPGA 实现的方法并加以优化。分析了椭圆曲线密码算法中的关键运算模块。为了达到面积和性能的平衡，对关键模块采取了速度优先的方法实现，对其他模块采取了资源优先的方法实现。

2、点加运算和倍点运算是椭圆曲线计算的基础，本文结合底层有限域运算的特点和 FPGA 的并行计算特性，选用雅克比射影坐标系，设计了点加运算和倍点运算的计算方法。

3、椭圆曲线密码中最耗时的部分为点乘运算。在分析了椭圆曲线密码算法中的点乘运算后，根据 FPGA 的并行计算特性设计了一种快速方法，大大提高了椭圆曲线密码算法的性能。

4、将有限域运算和椭圆曲线运算封装为大数库，并在此基础上实现了基于素域的 ECC 算法。将设计在 Xilinx 公司的 ISE 中进行编译并在 virtex5 开发板中进行了验证和测试。测试结果表明：在 100MHZ 工作频率下，ECC256 的签名性能为平均 422 次/秒，验证的性能为平均 216 次/秒，产生密钥对的性能为平均 443 次/秒，加密性能为平均 221 次/秒，解密性能为平均 442 次/秒。

**关键词：**椭圆曲线，FPGA，素域，点乘

# ABSTRACT

With the rapid development of information technology and computer network, information security is given moer and more attention. How to ensure the security, secreey, verity and integrity of the information in computer system and communication network is the main research task of information security. Crypto technology is the core of information security technology. Elliptic Curve Cryptography is one of the branch of crypto technology, which can effectively solv Problems about digital signature, identification and key distribution in public channel. Elliptic Curve cryptosystem as a public crypto method has the advantages of higher security, the less calculational complexity, shorter key size, less space, higher speed and limited band width and so on, it has become the public key cryptology gradually and has broad application value in the fields.

This dissertation analyzed the theory of Elliptic Curve Cryptosystem in details and studied the material algorithms of FPGA implementation of Elliptic Cuvre Cyrptosystem. According to the idea of hiberarchy and modularization, the design scheme of Elliptic Cuvre Cyrptosystem base on FPGA has been gived. This dissertation mainly completes these following works:

1、The dissertation researches the mathematic theory of Elliptic Curve Cryptosystem and find out the methods suitable for FPGA implementation after analyse and compair many methods. The key computing modules in elliptic curve cryptography are analyzed. In order to achieve the balance of area and performance，the key modules are implemented by the method of speed first, and the other modules are implemented by the method of resource first.

2、Point addition and double point are the basis of elliptic curve operation. Based on the characteristics of the underlying finite field operation and the parallel computing characteristics of FPGA, the dissertation choose the Jacobian projective coordinate system, and design a computing method of point adding operation and multiple point operation.

3、Point multiplication is a time-consuming issue in Elliptic Curve Cryptography. After analyzing the point multiplication in elliptic curve cryptosystem, this paper designs a fast implementation method according to the parallel computing

characteristics of FPGA, which greatly improves the performance of elliptic curve cryptosystem.

4、The finite field operation and elliptic curve operation are encapsulated into a large number library, and the large number library is used to implement the ECC algorithm based on the prime domain. The design is compiled in the ISE of Xilinx Company and verified and tested in the Virtex5 development board. The test results show that under the 100MHZ working frequency, the signature performance of ECC256 is 422 times per second, the performance of the verification is 216 times per second, the key exchange performance is 217 times per second, the encryption performance is 221 times per second, and the decryption performance is 442 times per second.

**Keywords:** Elliptic Curve, FPGA, Prime domain, Point multiplication

# 目 录

# 缩略词表

| 缩略词 | 英文全称 | 译文 |
|---|---|---|
| ANSI | American National Stardards Institute | 美国国家标准化组织 |
| ASIC | Application Specific Integrated Circuit | 专用集成电路 |
| CPLD | Complex Programmable Logic Device | 复杂可编程逻辑器件 |
| DSP | Digital Signal Processor | 数字信号处理器 |
| ECC | Elliptic Curve Cryptography | 椭圆曲线密码 |
| ECDSA | Elliptic Curve Digital Signature Algorithm | 椭圆曲线数字签名算法 |
| ECES | Elliptic Curve Encryption Scheme | 椭圆曲线加密体制 |
| ECDLP | Elliptic Curve Discrete Logarithm Problem | 椭圆曲线离散对数问题 |
| EDA | Electronics Design Automation | 电子设计自动化 |
| EPROM | Erasable Programmable ROM | 可擦除可编程 ROM |
| FPGA | Filed Programmable Gate Array | 现场可编程门阵列 |
| Fp | Finite Field p | 有限域 p |
| $F_2^m$ | Finite Field $2^m$ | 有限域 $2^m$ |
| FIFO | First Input First Output | 先进先出 |
| GAL | Generic Array Logic | 通用阵列逻辑 |
| HDL | Hardware Description Language | 硬件描述语言 |
| IEEE | Institute of Electrical and Electronics Engineers | 电子与电子工程师协会 |
| ISO | International Organization for Stardardization | 国际标准化组织 |
| IEC | International Electratechnical Commission | 国际电工委员会 |
| LUT | Look Up Table | 查找表 |
| PAL | Programmable Array Logic | 可编程阵列逻辑 |
| PROM | Programmable Read Only Memory | 可编程只读存储器 |
| RAM | Random Access Memory | 随机存取存储器 |
| RSA | Rivest Shamir Adleman | 无 |
| SRAM | Static Random Access Memory | 静态随机存储器 |
| VHDL | Very High Speed Integrated Circuit Hardware Description Language | 超高速集成电路硬件描述语言 |
| WAPI | Wireless LAN Authentication and Privacy Infrastructure | 无线局域网鉴别和保密基础结构 |

# 第一章 绪 论

## 1.1 课题背景及研究意义

随着网络技术的飞速发展以及智能终端的大规模普及，信息安全问题也逐渐暴露出来。如何维护网络社会的正常秩序，确保信息在网络中传输和存储时的秘密性、完整性，避免国家或个人的信息被盗取、破坏、篡改等，将会是信息安全领域一直研究和解决的重要问题。

网络安全主要是信息安全[1]，而密码学是信息安全领域的核心研究问题[2]，是所有信息安全系统建立的基石。公钥密码体制是现代密码学研究的一个重要分支，它能有效地解决公共信道上的数字签名、身份认证等问题，在保密通信、电子政务、电子商务、电子金融[3]等领域得到了广泛应用。RSA 与 ECC 是目前广泛运用的两种公钥密码体制，RSA 的原理较简单，但为了达到安全强度通常需要较长的密钥，从而增加了实现的难度和密钥的存储空间。相比之下，ECC 的原理复杂求解难度更大，达到相同的安全强度，需要的密钥长度比 RSA 要小得多，被认为是当前最安全，也是最有前途的公钥密码体制[4]。

公钥密码算法的实现可以分为软件实现和硬件实现两类。软件实现的优点是设计和开发的成本低，方便维护，缺点是性能不高。所以，只适合用于一般的网络安全[5]；硬件实现可分为 FPGA 和 ASIC 两种，ASIC 实现方式能针对特定的一种或几种算法进行硬件优化实现，达到较高的密码处理性能、较少的硬件资源以及较低的功耗，适用于大量应用且算法固定的场合。但是，它存在的问题是，通用性差、扩展性差、灵活性差、兼容性差、开发成本高且芯片一旦确定，硬件逻辑结构修改极度困难，不便于进行二次开发、升级和修改。FPGA 芯片具有强大的并行计算能力和强大的可编程能力，这使得采用 FPGA 实现算法时，除了能获得较高性能外，还具有升级维护方便的优点。所以，利用 FPGA 实现 ECC 算法是一种较好的方式。

虽然 FPGA 可编程功能非常强大，但椭圆曲线密码本身复杂的数学运算以及应用的多样性，同样给 FPGA 实现的高效性、灵活性、兼容性、适用性提出了严峻的挑战。

## 1.2 研究现状

自从 1985 年 Koblitz[6]和 VMiller[7]提出了椭圆曲线密码体制以来，由于其在密

钥量小、安全强度高、计算速度快等方面的优点，使得国际上许多公司、组织和机构都对其产生了浓厚的研究兴趣。从 1998 年开始，一些国际标准化组织就启动了对椭圆曲线密码的标准化工作，如 ANSI 公布了针对椭圆曲线密码的 ANSI-X9.62 和 ANSI-X9.63 标准， IEEE 公布了关于椭圆曲线的公钥密码标准 IEEE-P1363。与国外相比，我国对椭圆曲线密码体制的研究起步较晚，但随着国家层面的重视和加大研究的投入，使我国在椭圆曲线密码技术上的研究迅速发展，并取得了丰硕的成果。我们国家的 SM2 算法就是由我国自行研发并具有自主知识产权的 ECC 算法[8]，2017 年底 SM2 算法被纳入的 ISO/IEC 国际标准，标志着我国在椭圆曲线密码算法方面的研究达到了国际先进水平。此外，在中国无线局域网国家标准 GB15629.11 中，也将椭圆曲线密码体制算法作为新安全机制 WAPI 中的公钥和签名算法[9]。标志了椭圆曲线密码体制在我国正式成为国家层面的标准性体制。

当前国内外对 ECC 的研究方向主要有三个方面：第一是选择安全的曲线，椭圆曲线密码体制的安全性需要建立在一条安全的椭圆曲线基础之上。随机选取曲线是一种能获得高安全级别的椭圆曲线的方法，这种方法需要不断尝试选取曲线参数 $a$，$b$，并计算椭圆曲线的阶，根据各种安全准则进行分析判断，直到各种安全性指标都符合要求。第二是快速计算椭圆曲线的阶，选择安全的曲线主要依赖于曲线阶的计算，所以快速有效的计算椭圆曲线的阶，有利于加快找到安全的曲线。第三方面是如何快速实现椭圆曲线，椭圆曲线的计算非常复杂，包括了有限域和椭圆曲线上的多种数学运算，以及在计算过程中的坐标系选取、域转换等。近年来，由于对 ECC 计算性能的要求越来越高，对其实现技术的研究多集中在硬件实现方面，如何在 FPGA 中以更少的资源实现更优的性能是一个重要的研究方向。

## 1.3 论文工作与结构安排

本课题的目的是研究椭圆密码算法在 FPGA 中的设计与实现。采用 FPGA 实现必须考虑芯片的资源问题，作者在大量阅读国内外有关椭圆曲线的文献和研究成果的基础上，结合 FPGA 的芯片特性，选择并优化一些适合 FPGA 实现的算法，使性能和面积都能得到较好的兼顾。有限域的大数运算是椭圆曲线密码实现的难点，本课题将有限域的大数运算封装成通用的大数库，以便于椭圆曲线密码算法的研究和实现，并利用该大数库实现了 ECC 算法协议。本课题主要进行了以下几个方面的工作：

(一)理论研究

详细研究了椭圆曲线的基础理论，了解了椭圆曲线的类别及实现椭圆曲线需

要的数学知识。对现有的数学计算方法进行研究，找出适合 FPGA 实现的计算方法，根据 FPGA 的芯片特性，对某些计算方法进行优化改进，使其减少资源消耗或提高计算性能。

(二)椭圆曲线密码算法设计及实现

从椭圆曲线密码算法的分层结构出发，以模块化的方式完成了椭圆曲线密码算法的方案设计。方案中的大数库模块是最复杂的模块，其中包含了有限域的计算和椭圆曲线的计算。有限域计算中的模乘运算模块和椭圆曲线计算中的点运算模块为关键模块，影响着算法的整体性能，实现时采用了多种快速实现手段，尽可能的提高其性能。

(三)功能测试和性能测试

先将编写完成的硬件代码在 modelsim 仿真工具中进行正确性仿真，仿真通过后在编译工具中编译产生相应的执行文件和资源评估报告。最后将执行文件配置到 FPGA 的开发板中进行正确性验证，并通过一些测试用例测试算法的实际运算性能。

本论文主要研究椭圆曲线密码算法在 FPGA 中的设计与实现，论文的主要结构安排如下：

第一章为绪论，介绍了课题的研究背景和意义，以及国内外研究的现状，为本课题的开展做了可行性分析。

第二章 首先介绍了椭圆曲线相关的理论基础，包括素域和二进制域两种有限域，以及在两种有限域内的运算法则。其次，介绍了两种不同有限域下构建的椭圆曲线和椭圆曲线内的运算规则。最后说明了如何用椭圆曲线来构建椭圆曲线密码算法。

第三章 详细介绍了椭圆曲线密码系统的分层结构，对椭圆曲线的各运算层进行了分析，找出了同层的共性部分以及层间的关系，提炼了关键的运算模块，并在此基础上给出了椭圆曲线密码系统的设计框架，介绍了在此框架下系统的工作流程。

第四章 介绍了 FPGA 的设计流程，利用硬件描述语言编程实现了有限域中的加、减、乘、除各种运算，以及椭圆曲线上的各种运算。对关键模块采用速度优先的设计方法，尽可能的提高模块性能，对非关键模块采用面积优先的设计方法，尽可能的节省 FPGA 的资源。

第五章 在 modelsim 仿真工具中验证了上一章实现的各个运算模块，并运用上述运算模块实现了 ECC 算法。将实现的程序代码在 ISE 编译工具中综合编译，得到资源消耗情况和执行文件。将执行文件配置到 FPGA 开发板中进行了正确性

验证和性能测试，得到了 ECC 算法的运算性能。

第六章 为结论，总结了本文所做的工作，介绍了本文的主要贡献，并对下一步工作进行了展望。

# 第二章 椭圆曲线密码的理论基础

## 2.1 密码学基础

密码学是研究信息安全传递和保存的一门学科，本节将介绍密码学的一些基础理论。

### 2.1.1 密码学的基本概念

密码学是计算机科学和数学的一个分支学科，可以分为密码编码和密码分析。前者主要研究如何对信息进行编码从而对其进行保护的一门学科，后者则是研究如何通过攻击和破译的手段将编码保护的信息进行恢复的一门学科。

一个信息加密系统如图 2-1 所示，其包含了：明文、密钥、加密、解密和密文五个部分。加密系统的主要思想是将原始的明文数据经过加密算法进行编码伪装变成密文，然后将密文在公共信道中传输或者保存在存储设备中。授权的接收方在收到密文后，经过解密算法和密钥进行逆变换恢复出明文。而非授权的窃取者在得到密文后，由于没有解密算法或者密钥无法恢复出明文数据，从而对信息进行了保护。

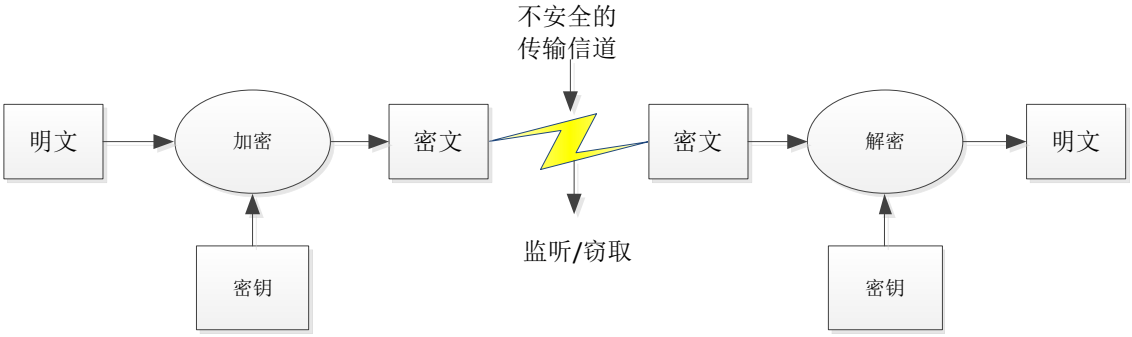


图 2-1 信息加密系统示意图

### 2.1.2 密码体制的分类

密码体制通常是指实现加密和解密的密码方案，现代密码学的研究按照密钥的使用策略来划分，可以分为对称密码体制和非对称密码体制两类，下面分别对两类密码体制进行介绍。

#### 2.1.2.1 对称密码体制

对称密码体制的特点是加密方和解密方使用同一个密钥，也称为单密钥体制，该体制下的密钥是保密的。对称密码体制的安全性主要依赖于两个方面。一是算

法必须足够强大，在实践上无法在仅仅知道密文的情况下推导出明文；二是，加密的安全性不是通过保密加密算法本身来实现，而是通过保密密钥来实现。对称密码体制的优点是加解密速度快，数据处理能力高，易于软件和硬件实现。对称密码体制的缺点是对密钥的分发和管理非常复杂，以及不能实现认证鉴别和不可否认性。

#### 2.1.2.2 非对称密码体制

非对称密码体制也称为公钥密码体制，该体制是基于数学中的单向函数和单向限门函数的困难性问题而设计的。公钥密码体制下的加密方和解密方使用不同的密钥，可以公开的叫做公钥，只能所有者拥有的叫做私钥。私钥和公钥存在着紧密联系，从私钥可以很容易计算得到公钥，但是想由公钥推导私钥，在计算上是不可行的。由公钥加密的数据，只有对应的私钥才能解开，且加密方的密钥是公开的，从而简化了密钥的分配和管理。

## 2.2 有限域

有限域运算是椭圆曲线密码的先决条件，本节主要介绍有限域的重要概念和基本理论[10][11]。

### 2.2.1 群和域

a）任意一个非空集合$G$和其上的运算“ ”，如果满足以下条件[12][13]：

1）封闭特性：对于任意的$a,b \in G$，存在$c \in G$，满足$c = a \cdot b$；

2）结合律：对于任意的$a,b,c \in G$，均满足$(a \cdot b) \cdot c = a \cdot (b \cdot c)$；

3）存在单位元：存在一个单位元$e \in G$，对于任意的$a \in G$，都满足$e \cdot a = a \cdot e = a$；

4）存在逆元：对任意的$a \in G$，存在一个逆元$a^{-1}$使得$a \cdot a^{-1} = e$。

则称$G$关于运算“ ”构成了一个群，记为群$<G, \cdot>$。

在群$<G, \cdot>$内，如果对于任意$a,b \in G$，都满足$a \cdot b = b \cdot a$，则称群$<G, \cdot>$是交换群或者 Abel 群[14]。如果集合$G$中的元素为有限多个，则$<G, \cdot>$为有限群，集合$G$中元素的个数称为群$<G, \cdot>$的阶。

b）在集合$F$上定义了“+”和“×”两个二元运算，$F'$为$F$上所有非 0 元素构成的集合，如果满足下列条件：

1）$<F,+>$是一个交换群，其单位元记为 0；

2）$<F',+>$是一个交换群，其单位元记为 1；

3）对任意 $a, b, c \in F$ 有

$$a \times (b + c) = (a \times b) + (a \times c)$$
$$(a + b) \times c = (a \times c) + (b \times c)$$

则把满足上述性质的<$F$,+,×>称为一个域。

如果域 $F$ 中的元素个数为有限个，则称 $F$ 为有限域或者 Galois 域，域 $F$ 中元素的个数称为有限域 $F$ 的阶。

### 2.2.2 有限域 *Fp*

设一个素数 $p$，整数集合{0，1，2，...，$p$-1}构成了有限域 $Fp$，也称为素数域 $GF(p)$。那么有限域 $Fp$ 中的元素有如下计算法则：

a）加法

如果 $a, b \in Fp$，则 $(a + b) = c$，其中 $c$ 是 $a + b$ 被 $p$ 整除后的余数，且 $0 \le c \le p - 1$，称为 $p$ 的加法模。有限域上的这种运算称为模加运算，可表示为 $c = (a + b) \bmod p$；

b）减法

如果 $a, b \in Fp$，则 $(a - b) = c$，其中 $c$ 是 $a - b$ 被 $p$ 整除后的余数，且 $0 \le c \le p - 1$，称为 $p$ 的减法模。有限域上的这种运算称为模减运算，可表示为 $c = (a - b) \bmod p$；

c）乘法

如果 $a, b \in Fp$，则 $(a \cdot b) = c$，其中 $c$ 是 $a \cdot b$ 被 $p$ 整除后的余数，且 $0 \le c \le p - 1$，称为 $p$ 的乘法模。有限域上的这种运算称为模乘运算，可表示为 $c = (a \cdot b) \bmod p$；

d）逆

如果 $a \in Fp$，且 $a \ne 0$，则存在唯一的元 $c = 1/a$，使 $a \cdot b = 1 \bmod p$ 成立。有限域上这种运算称为模逆运算，可表示为 $c = (1/a) \bmod p$。

有限域 $Fp$ 中的运算，计算开销大的运算是乘法、求逆和取模运算[13]。

### 2.2.3 有限域 $F_2^m$

阶为 $2^m$ 的域称为二进制域，2 是 $F_{2^m}$ 的特征，而 $m$ 是 $F_{2^m}$ 在域 $F_2$ 上的次数。$F_{2^m}$ 通常可用多项式基或者正规基来表示，本文将采用直观的多项式基来表示，如公式（2-1）所示。

$$f(z) = z^m + f_{m-1}z^{m-1} + f_{m-2}z^{m-2} + \ldots + f_1 z + f_0 \qquad (2\text{-}1)$$

其中，$(f_i \in \{0, 1\}, 0 \le i \le m - 1)$，是 $F_2$ 上 $m$ 次不可约多项式，域 $F_{2^m}$ 的元素

是次数最多为 $m-1$ 次的二进制多项式，如公式（2-2）所示。

$$F_2^m = \left\{ a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_1 z + a_0 : a_i \in \{0,1\} \right\} \quad (2\text{-}2)$$

域中的元素 $\left(a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_1 z + a_0\right)$ 可用长度为 $m$ 的二进制比特串 $\left(a_{m-1}a_{m-2}\ldots a_1 a_0\right)$ 表示，如公式（2-3）所示。

$$F_2^m = \left\{ a_{m-1}a_{m-2}\ldots a_1 a_0 : a_i \in \{0,1\} \right\} \quad (2\text{-}3)$$

所以，$F_2^m$ 中的元素为所有长度为 $m$ 的二进制串的集合。域元素的运算法则如下：

a）加减法

如果 $a = \left(a_{m-1}a_{m-2}\ldots a_1 a_0\right)$，$b = \left(b_{m-1}b_{m-2}\ldots b_1 b_0\right)$ 是 $F_2^m$ 中的元素，其多项式分别为 $a(z) = \left(a_{m-1}z^{m-1} + \ldots + a_1 z + a_0\right)$ 和 $b(z) = \left(b_{m-1}z^{m-1} + \ldots + b_1 z + b_0\right)$，则他们的和为：$c(z) = a(z) + b(z) = \sum \left(a_i \oplus b_i\right) z^i, (0 \le i \le m-1)$。二进制域上的加为模 2 加，即异或运算。由于 $GF(2)$ 上加法的逆元是其本身，所以二进制域上的减法运算与加法运算是等价的。

b）乘法

如果 $a = \left(a_{m-1}a_{m-2}\ldots a_1 a_0\right)$，$b = \left(b_{m-1}b_{m-2}\ldots b_1 b_0\right)$ 是 $F_2^m$ 中的元素，则 $c = a \cdot b$。其中多项式 $c(x) = c_{m-1}z^{m-1} + c_{m-2}z^{m-2} + \ldots + c_1 z + c_0$ 是模多项 $f(z) = \left(a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \ldots + a_1 z + a_0\right)\left(b_{m-1}z^{m-1} + b_{m-2}z^{m-2} + \ldots + b_1 z + b_0\right)$ 的余式。

c）求逆

如果 $a \in F_2^m$，且 $a \neq 0$，则存在唯一的逆元 $c = 1/a$，使 $a \cdot c = 1$。对任意的域元素 $a$ 有等式 $a^{2^m-1} \equiv 1$ 成立，则 $a^{-1} \equiv a^{2^m-2}$。求逆运算即可转化为乘法来运算。

## 2.3 椭圆曲线

### 2.3.1 椭圆曲线定义

一条定义在 $K$ 域上的椭圆曲线 $E$ 是指由一个三次方程所确定的曲线，曲线方程如公式（2-4）所示。

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a^6 \quad (2\text{-}4)$$

公式（2-4）这个方程称为 Weierstrass 方程[11]。其中 $a_1, a_2, a_3, a_4, a_6 \in K$，且 $\Delta \neq 0$，$\Delta$ 为曲线 $E$ 的判别式，各个参数的具体定义如公式（2-5）所示。

$$\begin{cases} d_2 = a_1^2 + 4a_2 \\ d_4 = 2a_4 + a_1a_3 \\ d_6 = a_3^2 + 4a_6 \\ d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^3 \\ \Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \end{cases} \tag{2-5}$$

两条定义在域$K$上的椭圆曲线$E1$，$E2$的 Weierstrass 方程定义如公式（2-6）所示。

$$\begin{cases} E1: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \\ E2: y^2 + \overline{a_1}xy + \overline{a_3}y = x^3 + \overline{a_2}x^2 + \overline{a_4}x + \overline{a_6} \end{cases} \tag{2-6}$$

若存在$u, r, s, t \in K$且$u \neq 0$，通过$(x, y) = (u^2x + r, u^3y + u^2sx + t)$变量的相容性变换把方程$E1$变成$E2$，那么称$E1$和$E2$在域$K$上同构。

通过同构变换，可对 Weierstrass 方程简化，域$K$的特征值不同简化后的方程式或者判别式$\Delta$有所差异。下面分别对域$K$的特征值为 2、3 和其他值的 Weierstrass 方程进行简化[16][17]。

a）域$K$特征值为 2 的情况

域$K$特征值为 2 时将考虑$a_1 = 0$和$a_1 \neq 0$两种情况。

1）当$a_1 = 0$时，通过$(x, y) \to (x + a_2, y)$相容变换将 Weierstrass 方程简化为曲线$y^2 + cy = x^3 + ax + b$，其中$a, b, c \in K$。这种曲线称为超奇异椭圆曲线，判别式$\Delta = c^4$。

2）当$a_1 \neq 0$时，通过$(x, y) \to \left(a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2a_4 + a_3^2}{a_1^3}\right)$相容变换将 Weierstrass 方程简化为曲线$y^2 + xy = x^3 + ax^2 + b$，$a, b \in K$。这种曲线称为非超奇异曲线，判别式为$\Delta = b$。

b）域$K$特征值为 3 的情况

域$K$特征值为 3 时将考虑$a_1^2 = -a_2$和$a_1^2 \neq -a_2$两种情况。

1）当$a_1^2 = -a_2$时，通过$(x, y) \to (x, y + a_1x + a_3)$相容变换将 Weierstrass 方程简化为曲线$y^2 = x^3 + ax + b$，其中$a, b \in K$。这种曲线称为超奇异椭圆曲线，判别式$\Delta = -a^3$。

2）当$a_1^2 \neq -a_2$时，通过$(x, y) \to \left(x + \frac{d_2}{d_1}, y + a_1x + a_1\frac{d_2}{d_1} + a_3\right)$相容变换将 Weierstrass 方程简化为曲线$y^2 = x^3 + ax^2 + b$，其中$d_2 = a_1^2 + a$，$d_2 = a_4 - a_1a_3$，$a, b \in K$。这种曲线称为非超奇异椭圆曲线，判别式为$\Delta = -a^3b$。

c）域$K$特征值不等于 2 或 3 的情况

通过相容变换$(x, y) \to \left( \frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} + \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right)$将 Weierstrass 方程简化为曲线$y^2 = x^3 + ax + b$，其中$a, b \in K$，判别式$\Delta = -16\left(4a^3 + 27b^2\right)$。

密码系统使用的是定义在有限域上的一类受限制椭圆曲线，常用的有限域有二进制域$F_{2^m}$和大于 3 的素数域$Fp$。下面将分别讨论这两种有限域的椭圆曲线。

### 2.3.2 $Fp$ 上的椭圆曲线

正如 2.3.1 节所述，素数域上的椭圆曲线简化后的形式为$y^2 = x^3 + ax + b$，其中$a, b \in Fp$，且$4a^3 + 27b^2 \neq 0$。椭圆曲线中的元素由曲线在素域$Fp$上所有解的集合再加一个无穷远点$O$构成，$O$点通常用坐标$(0, 0)$表示。

若点$P = (x_1, y_1)$和$Q = (x_2, y_2)$是椭圆曲线上的两个点，则素域上的椭圆曲线有如下运算法则：

- $P + O = O + P = P$，即无穷远点和任一点相加等于该点；
- 点$-Q = (x_2, -y_2)$称为$Q$的逆元素，满足$Q + (-Q) = O$；
- 若$P \neq \pm Q$，则$R = (x_3, y_3) = P + Q$，这种运算称为点加运算，其运算如公式（2-7）所示

$$\begin{cases} x_3 = t^2 - x_1 - x_2 \\ y_3 = t(x_1 - x_3) - y_1 \end{cases} \tag{2-7}$$

其中，$t = \frac{y_2 - y_1}{x_2 - x_1}$；

- 若$P = Q$，则$R = (x_3, y_3) = 2P$，这种运算称为倍点运算，其运算公式如公式（2-8）所示

$$\begin{cases} x_3 = t^2 - 2x_1 \\ y_3 = t(x_1 - x_3) - y_1 \end{cases} \tag{2-8}$$

其中，$t = \frac{3x_1^2 + a}{2y_1}$。

### 2.3.3 $F_2{}^m$ 上的椭圆曲线

如 2.3.1 节所述，简化后的$F_{2^m}$上的曲线有超奇异椭圆曲线和非超奇异椭圆曲线两种形式。研究表明超奇异椭圆曲线的安全级别只有亚指数级[18]，所以椭圆曲线密码系统中一般不采用此类曲线。故本节只针对特征 2 域上的非超奇异椭圆曲

线进行研究。

非超奇异椭圆曲线的表达式为 $y^2 + xy = x^3 + ax^2 + b$，若点 $P = (x_1, y_1) \in E(F_{2^m})$，$Q = (x_2, y_2) \in E(F_{2^m})$，则有如下运算法则：

- $P + O = O + P = P$，即无穷远点和任一点相加等于该点；
- 点 $-Q = (x_2, -y_2)$ 称为 $Q$ 的逆元素，满足 $Q + (-Q) = O$；
- 若 $P \neq \pm Q$，则 $R = (x_3, y_3) = P + Q$，这种运算称为点加运算，其运算如公式（2-9）所示

$$\begin{cases} x_3 = t^2 + t + x_1 + x_2 + a \\ y_3 = t(x_1 + x_3) + x_3 + y_1 \end{cases} \tag{2-9}$$

其中，$t = \dfrac{y_2 + y_1}{x_2 + x_1}$；

- 若 $P = Q$，则 $R = (x_3, y_3) = 2P$，这种运算称为倍点运算，其运算公式如公式（2-10）所示

$$\begin{cases} x_3 = t^2 + t + a \\ y_3 = x_1^{\,2} + (t + 1) x_3 \end{cases} \tag{2-10}$$

其中，$t = x_1 + \dfrac{y_1}{x_1}$。

## 2.4 椭圆曲线密码算法

建立在椭圆曲线上的密码算法有很多，常见的有签名算法、密钥对产生算法和加密算法三种。在国外的 ANSI X9.62[19]，ANSI X9.63[20]、IEEE P1363[21]、ISO/IEC 15946 等标准中对上述三种算法都做了详细规定。现就对标准中的签名算法、密钥对产生算法和加密算法分别进行介绍。

### 2.4.1 签名算法

ECDSA 是数字签名算法的椭圆曲线版本，包含了签名运算和验证运算。假设公共通信信道上的通信双方 Alice 和 Bob，他们的椭圆曲线参数 $(Fq, E, n, h, G)$ 是相同的。其中，$E$ 是定义在有限域 $Fq$ 上的椭圆曲线，$G$ 是椭圆曲线的一个基点，$n$ 是基点 $G$ 的阶，$h$ 为 $\#E(Fq)/n$ 的余因子。Alice 选择一个随机数 $dA$ 为私钥，并计算公钥 $PA = dA \cdot G$。那么，$(dA, PA)$ 为 Alice 的密钥对，其中 $PA$ 是公开的，任何人都可以获取到。

假设 Alice 向 Bob 签名，则签名和验证的过程如下：

a）签名运算

输入：私钥$dA$，待签名消息$M$。$M$包含了 Alice 的身份有关信息。

输出：签名值$(r, s)$。

A1：选择一个随机数$k$，　$k \in [1, n-1]$；

A2：计算$(x_1, y_1) = [k]G$；

A3：计算$r = x_1 \bmod n$，如果$r$=0，则返回步骤 A1；

A4：计算$u = H(M)$，$H$为摘要运算；

A5：计算$s = k^{-1} \cdot (u + r.dA) \bmod n$，若$s$=0 则返回步骤 A1；

A6：$(r, s)$即为 Alice 对消息的签名，将其发送给 Bob。

b）验证运算

输入：公钥$PA$，签名值$(r, s)$，消息$M$。

输出：判断签名是否合法。

B1：检验$r \in [1, n-1]$是否成立，若不成立则验证失败；

B2：检验$s \in [1, n-1]$是否成立，若不成立则验证失败；

B3：计算$e = H(M)$，$H$为杂凑运算；

B4：计算$u = s^{-1} \bmod n$；

B5：计算$t_1 = e \cdot u \bmod n$和$t_2 = r \cdot u \bmod n$；

B6：计算椭圆曲线点$(x_1, y_1) = [t_1]G + [t_2]P_A$，若$(x_1, y_1) = O$，则验证失败；

B7：计算$R = x_1 \bmod n$；

B8：检验$R = r$是否成立，若成立则验证通过；否则验证不通过。

### 2.4.2 公钥加密算法

假定公共通信信道上的通信双方 Alice 和 Bob，他们有同样的椭圆曲线参数$(Fq, E, n, h, G)$，$E$是定义在有限域$Fq$上的椭圆曲线，$G$是椭圆曲线的一个基点，$n$是基点$G$的阶，$h$为$\#E(Fq)/n$的余因子。Bob 选择一个随机数$dB$为私钥，并计算公钥$PB = dB \cdot G$。那么，$(dB, PB)$为 Alice 的密钥对，其中$PB$是公开的。

ECES 假设 Alice 要向 Bob 发送一个报文$M$，则加密和解密过程如下：

a）Alice 运行 ECES 加密算法

输入：待加密报文$M$，公钥$PB$。

输出：加密结果$C$。

A1：选择一个随机数$k \in [1, n-1]$；

A2：计算椭圆曲线点$C_1 = [k]G = (x_1, y_1)$；

A3：计算椭圆曲线点$S = (x_2, y_2) = [k]PB$，若$S$是无穷远点，则返回 A1；

A4：计算$C_2 = M \oplus x_2$；

A5：输出密文$C = (C_1, C_2)$。

b）Bob 运行 ECES 解密算法

输入：加密结果$C = (C_1, C_2)$，私钥$dB$。

输出：报文$M$或者解密失败。

B1：从$C$中取出比特串$C_1$；

B2：计算椭圆曲线点$Z = (x_Z, y_Z) = [dB]C_1$，若$Z$是无穷远点，则返回错误并退出；

B3：从$C$中取出比特串$C_2$；

B4：计算$M = C_2 \oplus x_Z$；

B5：输出明文$M$。

### 2.4.3 密钥对生成算法

数字签名验证算法和加解密算法都需要一对公私钥才能完成相应的功能，因此，需要密钥对生成算法来提供产生密钥对的服务。

$E$是定义在有限域$Fq$上的椭圆曲线，$G$是椭圆曲线的一个基点，$n$是基点$G$的阶，$h$为$\#E(Fq)/n$的余因子，$a, b \in Fp$。密钥对生成算法相对简单，选取一个随机数$d$，$d \in [1, n-1]$，计算$Q = d \cdot G$。$(d, Q)$构成了一对密钥对，其中$d$为私钥，$Q$为公钥。密钥对产生算法具体运算如算法 2.1 所示。

算法 2.1 密钥对产生算法

输入：随机数数$d$，$d \in [1, n-1]$，椭圆曲线基点$G$。

输出：$d$，$Q$

1) 计算$Q = d \cdot G$；

2) 输出$d$，$Q$。

## 2.5 本章小结

介绍了密码学基础知识及密码体制的分类，分别对对称密码体制和非对称密码的优缺点进行了描述，说明了两种密码体制各自的用途。

从群和域的基本定义开始，对有限域的定义做了详细的阐述，并分别介绍了有限域$Fp$和有限域$F_{2^m}$上基本运算。

椭圆曲线上的运算是椭圆曲线密码体制的基础，椭圆曲线密码系统大都建立在有限域$Fp$或者有限域$F_{2^m}$上。本章对基于两种域所建立的椭圆曲线运算法则进行了描述，为后续章节的椭圆曲线密码系统的设计与实现提供的理论支撑。

椭圆曲线密码算法是在椭圆曲线理论基础上建立的算法协议，常见的椭圆曲线算法协议包含签名算法、密钥对产生算法和加密算法。本章了描述签名算法、密钥对产生算法和加密算法的实现过程，后续章节将在 FPGA 的既定芯片资源限制的情况下，从少的资源量和更快的时序这两个方面对素数域的椭圆曲线密码系统的设计与实现展开研究。

# 第三章 椭圆曲线密码系统需求分析及硬件框架设计

## 3.1 ECC 系统的应用

ECC 通常有软件、ASIC 和 FPGA 三种实现方式。本文采用 FPGA 来实现 ECC，其主要应用于中等级性能的嵌入式应用系统。本设计最终以模块的形式被集成到系统的 FPGA 中，能为应用系统提供每秒几百次的签名验证运算服务和加解密运算服务。

## 3.2 ECC 系统需求分析

FPGA 实现的 ECC 系统作为一个运算模块，通常需要同其他通信模块配合才能提供 ECC 的各种运算服务。所以，在 FPGA 硬件资源有限的情况下，对 ECC 系统的资源消耗有一定的要求，下面分别从硬件平台、功能和性能三个方面对本文实现的椭圆曲线密码系统的需求进行分析。

### 3.2.1 平台及资源要求

不同型号的 FPGA，其工艺和结构会有很大的差异，同一个 VHDL 程序模块在不同型号的 FPGA 中进行编译，其占用的逻辑资源和编译的时钟频率都有所不同。本文选择的实现硬件平台为 xilinx 公司的 virtex5 系列 FPGA，其型号为 xc5vfx110t。在该型号的 FPGA 中，实现 ECC 系统需要的硬件逻辑资源不能超过芯片总的逻辑资源的 50%。

### 3.2.2 功能需求

ECC 系统提供下列运算功能。

a）签名运算：输入私钥和待签名的数据，返回签名运算的结果。签名运算能够有效的避免发送者发送抵赖的情况。

b）验签运算：输入签名的结果、验签的公钥和签名的数据，返回验签结果，结果为通过或者不通过。验签运算能对签名的结果进行验证和校验，保证签名的唯一性。

c）加密运算：输入公钥和待加密的数据，返回加密运算的结果。加密运算可以用于加密传输一些数据量小数据，如密钥信息等。

d）解密运算：输入加密运算结果、私钥，成功时返回解密运算结果，失败时返回失败标志。解密运算能从加密结果中还原出明文，并对明文数据进行校验，

防止数据被篡改。

e）密钥对产生：输入私钥，返回公钥。提供签名验证和加解密运算中需要的公私钥对。

### 3.2.3 性能需求

ECC 系统在 xc5vfx110t 型号的 FPGA 中提供的运算功能需满足下列指标。

a）签名运算：签名运算的性能不小于 400 次/秒。

b）验证运算：验证运算的性能不小于 200 次/秒。

c）加密运算：加密 256bit 的数据时，运算性能不小于 200 次/秒。

d）解密运算：解密 256bit 的数据时，运算性能不小于 400 次/秒

e）密钥对产生：密钥对产生运算不小于 400 次/秒。

## 3.3 ECC 系统层次结构

ECC 系统包含了签名算法、公钥加密算法和密钥对生成算法三种，ECC 系统按照层次来分，可以分为协议层、群运算层和有限域运算层，层次结构如图 3-1 所示 [22][23[24]]。

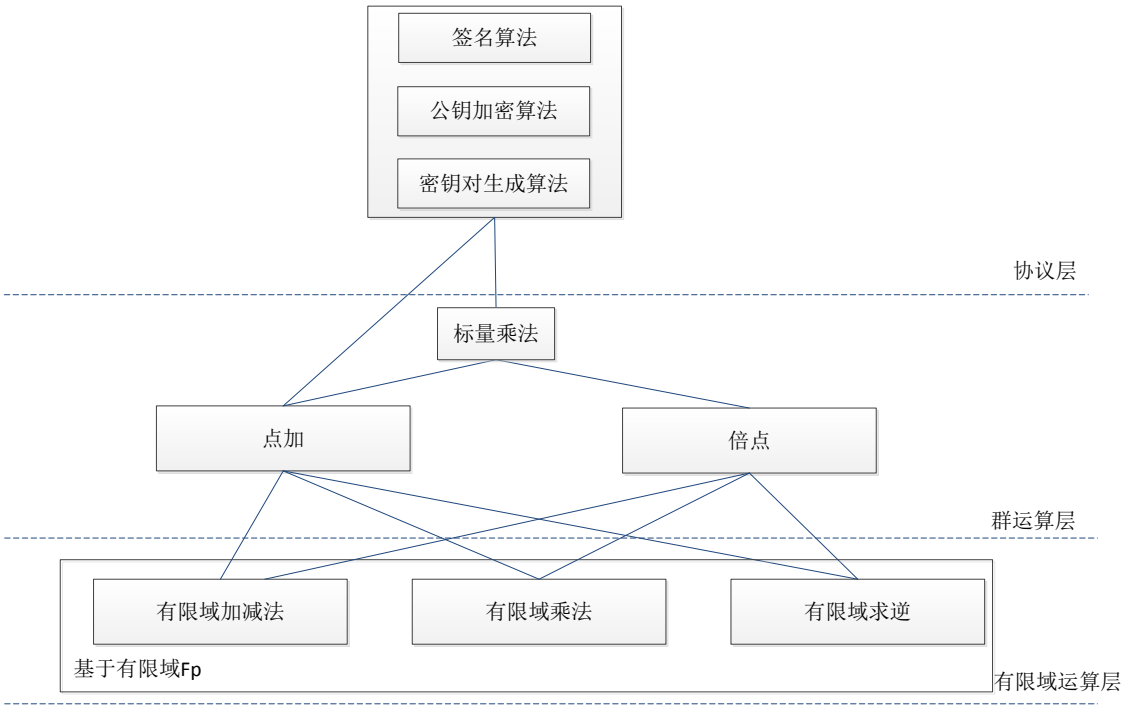


图 3-1 ECC 系统层次图

协议层实现了三个算法：签名算法、公钥加密算法和密钥对生成算法。三个算法间相互独立，但都会调用有限域 $F_n$ 上的运算和椭圆曲线上的运算。在这些算法

协议中，最重要的部分是椭圆曲线上的标量乘法，它影响着整个 ECC 系统的性能。标量乘法的运算过程主要由点加和倍点组成，点加和倍点又由有限域 *Fp* 上的算术运算组成。

群运算层有点加和倍点两种基本运算，点加运算为两个不同点相加，倍点运算为两个相同点相加。点加和倍点运算的结果同样为椭圆曲线上的一个点。

有限域运算层包含了模加减运算、模乘法和模逆运算，这三种运算中模逆运算的耗时最长，模加减法运算的耗时最短。模逆运算在整个 ECC 系统中只需要运算一次，模加减和模乘运算的使用频率相当，所以模乘运算是 ECC 系统中最关键的底层运算模块，它很大程度上影响着 ECC 系统的运算效率。

## 3.4 ECC 系统设计框架

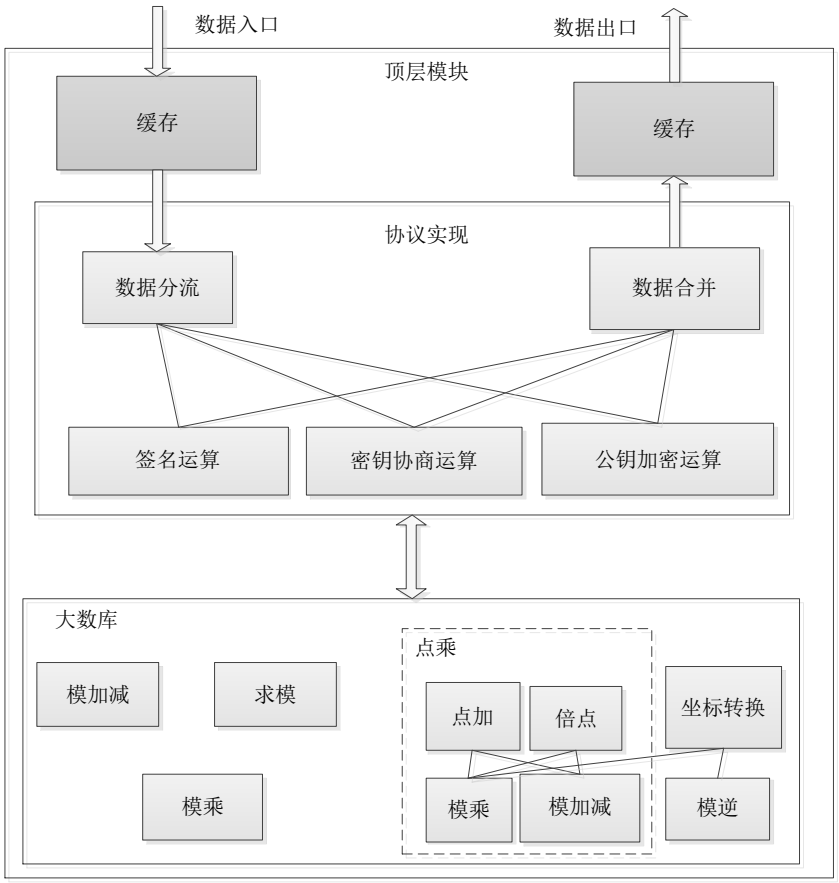


图 3-2 ECC 系统设计框图

ECC 系统的设计框图如图 3-2 所示，系统设计主要包含了协议实现和大数库实现两部分。数据入口和数据出口处的缓存为 FPGA 内部存储块生成的 FIFO。

a）协议实现模块

该模块将输入缓存中的数据取出，经过数据分流模块把数据传递到相应的运

算子模块，待运算模块计算完成后，经过数据合并模块将数据写到输出缓存。签名运算、公钥加密算法和密钥对生成算法三种运算协议需要的有限域计算和椭圆曲线计算都封装在大数库模块中，分时的被三种运算协议调用。为了优化 FPGA 的使用面积，三种运算共享计算过程中的临时变量。

b）大数库模块

在椭圆曲线的域选定为素域后，大多数情况会选择超奇异椭圆曲线来实现椭圆曲线密码系统。这种情况下，有限域的计算和椭圆曲线上的运算方法是通用的，不同的只是协议层的实现方式有差异。所以，实现一个通用、资源占用和计算性能都兼顾的模块是本文研究的重点。

大数库模块提供了五种运算接口：模加减接口、模乘接口、模逆接口、点乘接口和点加接口。为了保证椭圆曲线上点运算的性能，点加和倍点运算将使用独立的模乘和模加减运算模块。此外，考虑到点运算的耗时比有限域运算的耗时要多得多，有限域运算的性能对 ECC 系统的整体性能影响不大，所以点乘外的模乘模块将采用资源占用少、耗时多的串行计算实现方案，点乘内的模乘模块将采用资源占用多、性能快的快速模乘实现方案。

## 3.5 ECC 系统工作流程

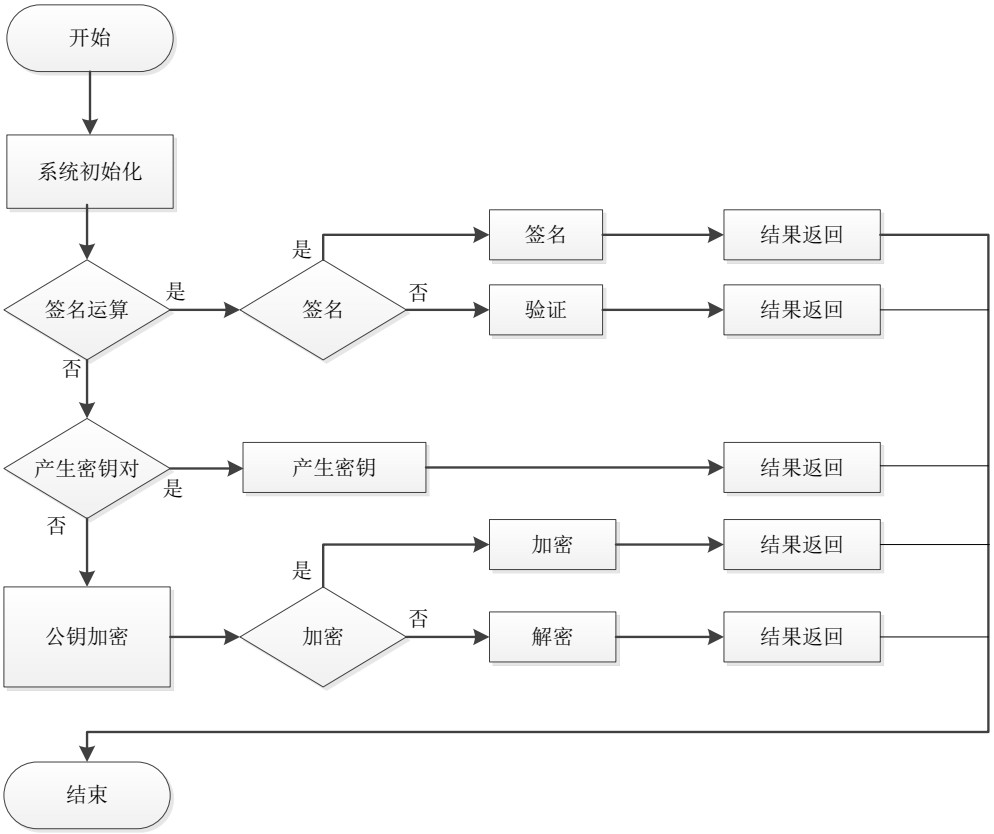


图 3-3 ECC 系统工作流程图

本文设计的椭圆曲线密码系统整体的流程图如图 3-3 所示，该图详细描述了系统的初始化以及各种计算的工作过程。

a）系统初始化

用于初始化系统内部的状态，对椭圆曲线的部分参数进行域的转换，为后续工作态做好准备。

b）签名运算

签名运算有两个接口，一个是签名接口，一个是验证接口，两个接口的运算不能同时进行，需要一个计算完成后才能计算另一个。假定用户 A 为签名者，签名接口需要输入的参数有用户 A 的私钥 *dA*，随机数 *k* 和待签名的消息 *M*，输出结果为 *r*，*s*。验证接口需要输入的参数有用户 A 的公钥 *PA*，签名结果 *r*、*s* 和消息 *M*，返回结果为验证成功或者失败。用户 A 和用户 B 相互签名验证的流程图如图 3-4 所示。

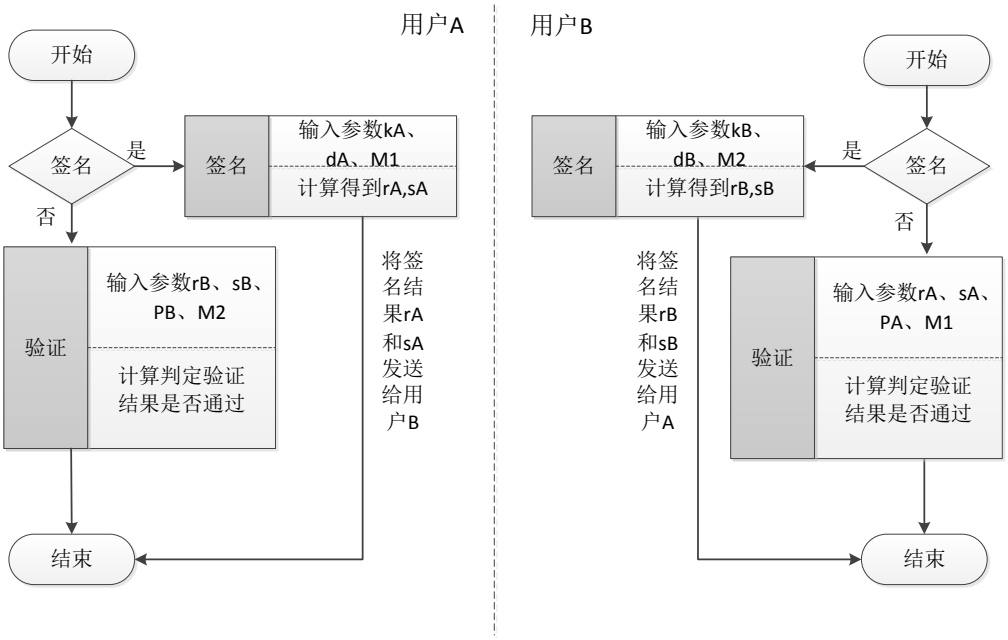


图 3-4 签名验证流程图

c）产生密钥对运算

产生密钥对的接口较简单，输入一个符合要求的随机数，返回生成的公钥。产生密钥对的流程如图 3-5 所示。

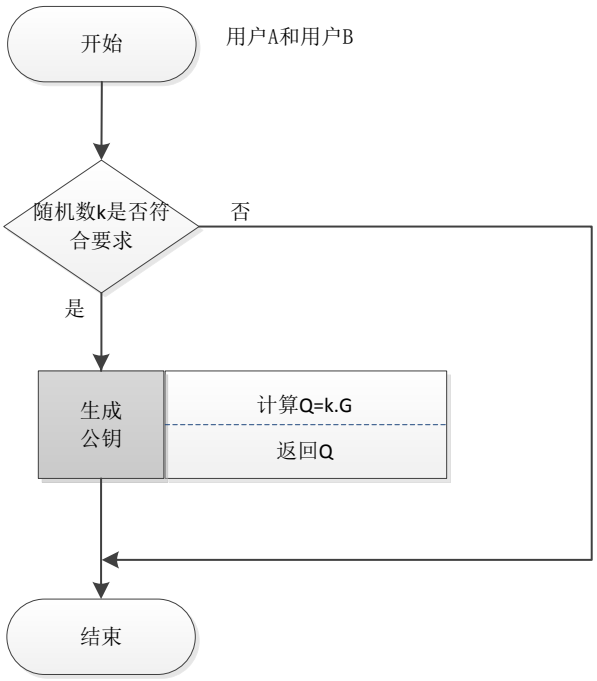


图 3-5 密钥对产生流程图

d）公钥加密运算

公钥加密运算有加密和解密两个接口，两个接口的运算不能同时进行，需要一个计算完成后才能计算另一个。假定用户 A 和用户 B 为加解密的双方，加密接口的输入参数有随机数 $k$、用户 B 的公钥 $P_B$ 以及待加密的消息 $M$，输出结果为 $C1$、$C2$ 、$C3$。解密接口的输入参数有加密的结果 $C1$、$C2$、$C3$、用户 B 的私钥 $d_B$，输出结果为解密后的名为 $M$。用户 A 和用户 B 加解密运算的流程如图 3-6 所示。

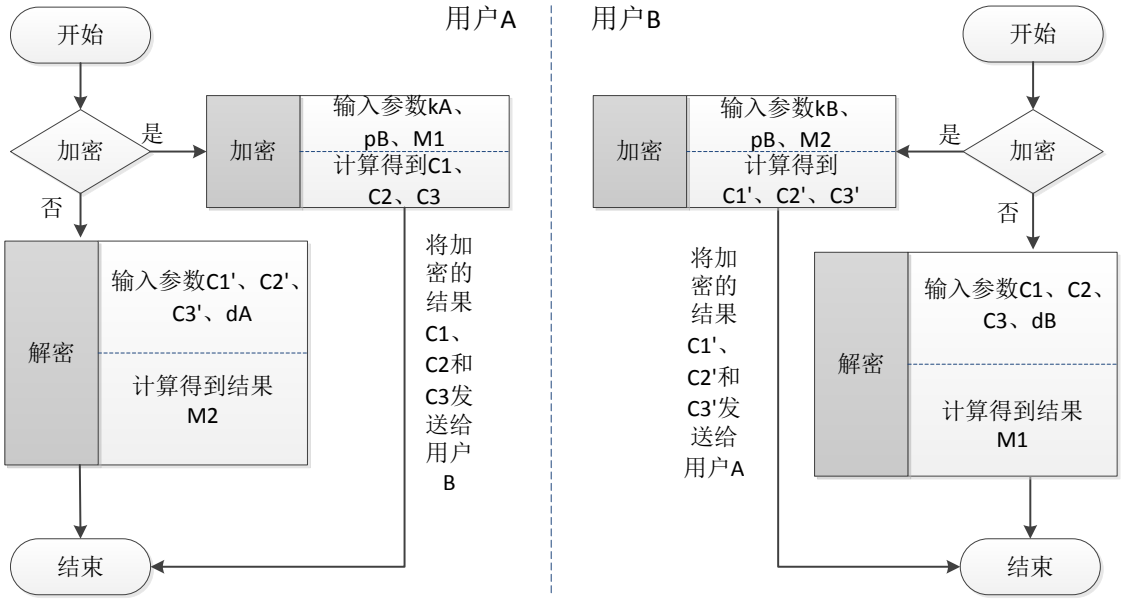


图 3-6 加解密流程图

## 3.6 本章小结

本章描述了设计的应用场景及设计的需求，分别从层次结构、设计框架和工程流程三个方面，对 ECC 系统的原理、实现和使用进行了介绍。

首先，层次结构描述了 ECC 系统每个层次所完成的功能和包含的运算，说明了各层之间的关系，同时给出了各层里最复杂的运算单元以及影响 ECC 系统性能的关键运算单元。

其次，设计框架部分给出了 ECC 系统的详细设计方案，将层次结构部分的运算单元模块化，并对公共的模块进行了封装，以供其他模块的复用，从而节约了 FPGA 的资源。其中，大数库模块是整个 ECC 系统中功能最多也最复杂的一个封装模块，该模块的资源消耗和性能影响着整个 ECC 系统。

最后，介绍了本文所设计的 ECC 系统的工程流程，详细描述了签名验证、密钥对产生和加解密三种算法运算时通信双方的工作流程以及输入的计算参数和返回的结果。

# 第四章 椭圆曲线密码系统的模块设计与 FPGA 实现

## 4.1 FPGA 介绍

### 4.1.1 FPGA 概述

FPGA 是在 PAL、GAL、CPLD 等可编程器件的基础上进一步发展起来的可编程数字集成电路，它可以完全由用户进行编程和配置，进而实现特定的功能。它作为专用集成电路中的一种半定制电路，既解决了定制电路的不足，又克服了原有可编程器件门电路有限的缺点。它采用门阵列的通用结构，具有很高的集成度、较强的逻辑实现能力和好的设计灵活性，因此，一经推出就受到广泛欢迎，并迅速发展。目前，针对不同的使用场景，FPGA 已经形成了各种不同的结构，按基本逻辑功能块的大小分类，FPGA 可分为细粒度 FPGA 和粗粒度 FPGA[18]。细粒度 FPGA 的逻辑功能块较小，可编程的余地更大，资源的利用率高，能够实现更丰富的功能。但细粒度的逻辑功能块，也使连线开关多，一定程度上加大了延时，降低了运行速度；粗粒度 FPGA 的逻辑功能块较大，可编程的余地更小，资源不能充分利用。但其减少了一些连线开关，降低了时延，从而提高了运算的速率。按照连线结构分类，可分为分段互连型和连续互连型两类。分段互连型 FPGA 中具有多种不同长度的金属线，各种金属线间通过开关矩阵来控制通断，这种结构使布线更加灵活方便，能完成一些复杂设计的走线，但无法预测布线延时；连续互连型 FPGA 内部的金属线长度是相同的，能够预测布线的延时，但灵活性不高。按照编程方式分类，可分为单次编程型和多次编程型两类。 单次编程型 FPGA 采用了反熔丝开关元件，具有体积小、互连线阻抗低、集成度高和速度快的特点，不需外接 PROM 或 EPROM 存储器，但缺点是只能编程一次，适合于产品定型后的批量生产。多次编程型 FPGA 采用了 SRAM 开关元件，可实现反复编程。系统上电后，将 FPGA 配置数据加载到内部易失性存储器中，即可实现相应的逻辑功能，需要改变逻辑功能时，只需要重新配置新的配置程序。多次编程型 FPGA 还可以在系统运行中利用局部重配置技术只改变部分模块的功能，而不影响其他模块的正常工作。

FPGA 内部常用的可编程资源有：逻辑单元、嵌入式存储块、乘法单元等基本资源。可编程逻辑单元是由 LUT 和寄存器组成的，LUT 完成组合逻辑功能。寄存器可以配置为触发器，也可以配置为锁存器；嵌入式存储块可以配置为单口 RAM、双口 RAM，也可以配置为 FIFO。RAM 和 FIFO 的深度与宽度都可以根据需求灵

活设置；乘法单元为嵌入式乘法器，运算效率高。

### 4.1.2 FPGA 设计流程

FPGA 的设计流程主要包括了设计输入、功能仿真、综合、实现、时序仿真以及芯片配置等六个步骤[25][26][27]，设计流程如图 4-1 所示。

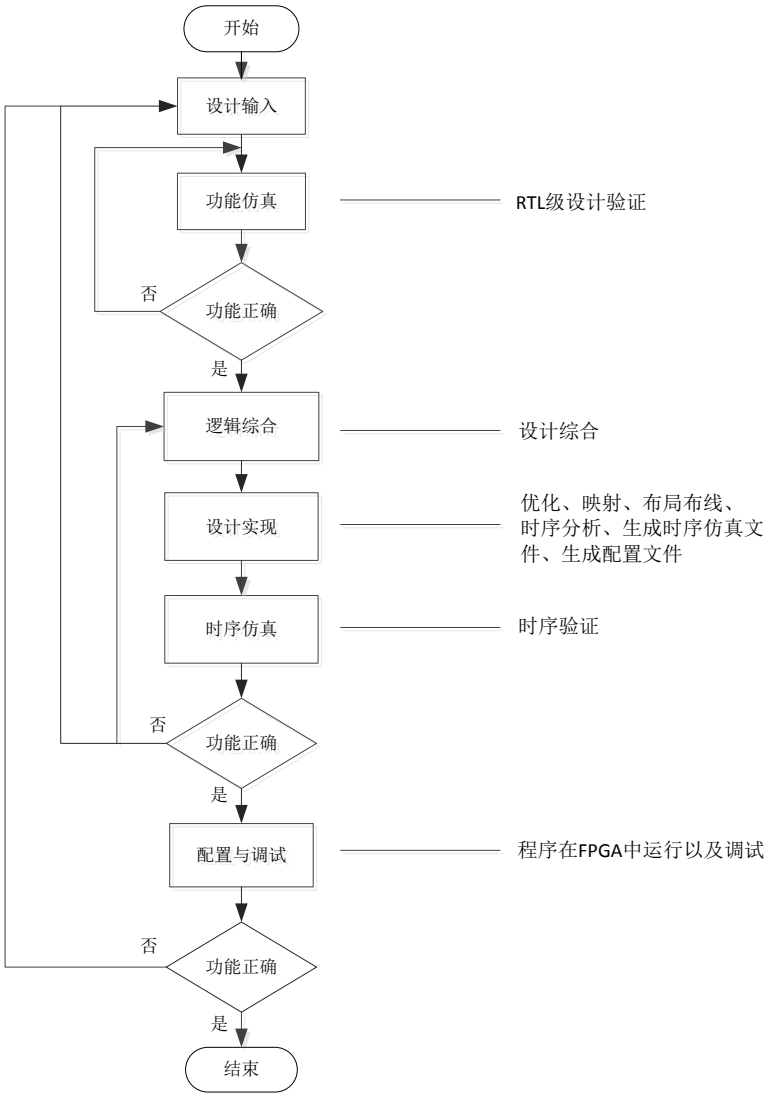


图 4-1 FPGA 设计流程

a）设计描述

设计描述是将需要实现的系统或者电路以开发软件能够识别的方法描述出来，并输入给 EDA 工具的过程。常用的方法有原理图输入和硬件语言描述。原理图输入是一种直观的描述方式，与 PCB 设计的方式类似，先将各功能模块转换为器件，再将各个器件连接在一起画出原理图。这种方法直观，但效率很低，且不易维护，可移植性差，当更换 FPGA 芯片后，原有的原理图都需要作一定的改动。

硬件语言描述广泛应用的是行为 HDL 描述，其主流语言是 Verilog 和 VHDL。采用这两种语言进行描述时，与 FPGA 芯片厂家及采用的工艺没有关系，利用自顶向下或者自底向上的设计，便于功能模块的分割与移植，具有很强的逻辑描述和仿真能力，而且具有很高的输入效率。

b）功能仿真

功能仿真也称为前仿真，不带有 FPGA 的器件特性和延迟信息，仅对用户所设计的功能模块的正确性进行检测。仿真前，先利用波形编辑器或者编写测试文件建立波形文件或测试向量。将需要观察的信号加入到波行窗口中，仿真结束后从波形窗口中可以观察到各个信号随时钟节拍变化的情况。如果发现功能错误，则返回修改模块的逻辑设计。常用的仿真工具有各 FPGA 厂商提供的开发工具，以及 Mentor 公司的 ModelSim 等。

c）逻辑综合

逻辑综合是将设计描述阶段的高层次描述转化成较低层次的描述，根据目标和要求优化逻辑连接。按照层次来看，综合是将设计输入编译成由与门、或门、非门、触发器和存储块等基本逻辑单元组成的逻辑连接网表，而非真实的门级电路。HDL 程序的编写必须符合综合器要求的风格，才能将其转换成标准的门级结构网表。常用的综合工具有各个 FPGA 厂家自己提供的综合开发工具，以及第三方厂家 Synplicity 公司的 Synplify/SynplifyPro 软件。

d）设计实现

设计实现是将综合生成的逻辑网表利用 FPGA 制造商提供的工具将其中的逻辑单元映射到目标器件结构的资源中，包括了布局和布线两个步骤。布局是将逻辑网表中的硬件描述和底层单元逐一合理地映射到芯片内部的硬件资源上，通常为用户提供了性能优先模式、面积优先模式和两者平衡布局模式三种选择方式，以满足不同使用场景的需求。布线与 PCB 设计中的连线类似，是根据布局的拓扑结构，通过控制芯片内部各种连线资源间的矩阵开关的通断，合理地、正确地将各个逻辑单元连接起来。目前，FPGA 的规模都比较大，内部结构非常复杂，尤其是在有时序约束的情况下，需要利用时序驱动引擎来进行布局布线。布局布线完成后，软件工具会自动生成布局布线报告，提供整个设计占用的资源情况和设计内各个子模块所使用的资源情况，以及整个设计在目标器件中的最大运行频率 fmax。在 fmax 不满足设计需求时，可以查看时序分析报告，找出其中的关键路径，对程序设计进行优化。由于 FPGA 芯片生产商对自己芯片的结构最为了解，所以利用芯片厂商提供的 EDA 工具往往能获得最佳的资源优化和最优的时序收敛。

e）时序仿真

时序仿真，也称为后仿真，是指将布局布线的时延信息反标注到网表文件中，再通过仿真工具来检测网表文件是否有不满足时序约束条件或器件固有时序特性的情况。时序仿真包含了精确的延迟信息，能够较好地反映芯片的实际工作情况，是最接近板级工作状态的仿真。不同芯片的固有时延不同，同一款芯片不同的布局布线方案也会造成不同的时延信息。因此，布局布线完成后，对系统和各个子模块进行时序仿真，评估系统性能，分析时序情况，以及检测竞争冒险是非常有必要的。

f）配置与调试

FPGA 设计的最后一步是配置芯片与板级调试。FPGA 配置是指将布局布线阶段产生的配置文件下载到 FPGA 芯片中。目前，FPGA 的配置手段多种多样，可以用仿真器直接配置、可以由外部处理器配置、也可以用配置芯片上电后自动配置，但无论哪种配置方式都需要满足一定的条件，如编程电压、编程时序和编程算法等。FPGA 的调试分为片内调试和片外调试，片内调试指在 FPGA 芯片内部用逻辑资源和存储块资源实现一个逻辑分析仪，监视某些信号的变化状态。目前，大多数的 FPGA 芯片生产商都提供了内嵌逻辑分析仪(如 Xilinx 公司的 ISE 中集成的 ChipScope、Altera 公司 QuartusII 中集成的 SignalTapII)。片外调试指在 FPGA 芯片外部来监视某些信号的状态，这种方法需要借助专门的逻辑分析仪设备，同时还需要引出测试管脚。片外调试比较繁琐，加之逻辑分析仪设备比较昂贵，一般只在 FPGA 与其他芯片间通信异常时使用。

## 4.2 有限域 *Fp* 运算模块设计与实现

### 4.2.1 模加减运算

#### 4.2.1.1 多精度加法

有两个 $n$ 位的非负整数$(a_{n-1}a_{n-2}...a_1a_0)_m$和$(b_{n-1}b_{n-2}...b_1b_0)_m$相加，多精度加法得到了 $m$ 进制的和$(s_{n-1}s_{n-2}...s_1s_0)_m$。这里的$s_n$为进位位，总是等于 0 或者 1。假定$b$的值为$2^{32}$，$c$为每次$a_i$和$b_i$相加的进位，对$a_i$，$b_i$，$c$，$b$及$s_i$的关系进行如下分析：$a_i$和$b_i$的值在$[0,(2^{32}-1)]$之间，$c$为 0 或者 1，$a_i$和$b_i$分别有两种情况：$a_i < (2^{32}/2)$或$a_i \geq (2^{32}/2)$，$b_i < (2^{32}/2)$或$b_i \geq (2^{32}/2)$，$a_i$，$b_i$，$c$及$s_i$有如下三种情况：

a）当$a_i < (2^{32}/2)$，$b_i < (2^{32}/2)$时

$a_i + b_i + c < (2^{32}-1)$，这种情况下$s_i$不会有溢出进位，即$c$等于 0。

b）当$a_i \geq (2^{32}/2)$，$b_i \geq (2^{32}/2)$时

$a_i + b_i + c \geq (2^{32} + 1)$，这种情况下$s_i$会有产生进位，即 $c$ 等于 1。

c）当$a_i \geq (2^{32}/2)$，$b_i < (2^{32}/2)$或者$b_i \geq (2^{32}/2)$，$a_i < (2^{32}/2)$时

$a_i + b_i + c$的结果$s_i$有可能有进位，有可能没有进位。

算法 4.1 给出了多精度加法的计算方法。

算法 4.1 多精度加法

输入：整数$a = (A_{d-1}A_{d-2}...A_1A_0)_w$，$b = (B_{d-1}B_{d-2}...B_1B_0)_w$。

输出：$(c, s) = a + b$，其中$c$为进位，$s$为结果。

1) $(c, S[0]) \leftarrow A[0] + B[0]$；

2) 对 $j$ 从 1 到 $d$-1，循环执行

2.1）$(c, C[j]) \leftarrow A[0] + B[0] + c$；

3) 返回$(c, s)$。

#### 4.2.1.2 多精度减法

有两个 $n$ 位的非负整数$(a_{n-1}a_{n-2}...a_1a_0)_m$和$(b_{n-1}b_{n-2}...b_1b_0)_m$相减，多精度减法得到了 $m$ 进制的和$(s_{n-1}s_{n-2}...s_1s_0)_m$。这里的$s_n$为借位，总是为 0 或者 1，等于 0 时表示$a$大于$b$，等于 1 表示$a$小于$b$。假定$b$的值为$2^{32}$，$c$为每次$a_i$和$b_i$相减的借位，对$a_i$，$b_i$，$c$及$s_i$的关系进行如下分析：

a）当$(a_i - c) < 0$时，

$c = 1$，$s_i = 2^{32} - 1 - b_i$。

b）当$(a_i - c) \geq 0$，且$(a_i - c) < b_i$时

$c = 1$，$s_i = 2^{32} - b_i + a_i - c$。

c）当$(a_i - c) \geq 0$，且$(a_i - c) \geq b_i$时

$c = 0$，$s_i = a_i - b_i - c$。

算法 4.2 给出了多精度减法的计算方法。

算法 4.2 多精度减法

输入：整数$a = (A_{d-1}A_{d-2}...A_1A_0)_w$，$b = (B_{d-1}B_{d-2}...B_1B_0)_w$。

输出：$(c, s) = a - b$，其中$c$为借位，$s$为结果。

1) $(c, S[0]) \leftarrow A[0] - B[0]$；

2) 对 $j$ 从 1 到 $d$-1，循环执行

2.1）$(c, C[j]) \leftarrow A[0] - B[0] - c$；

3) 返回$(c, s)$。

#### 4.2.1.3 模加减法

有限域 *Fp* 上的模加减法与常规的加减法基本相同，差异的地方在于计算结果必须保证是有限域 *Fp* 中的元素，所以对加减法的结果要增加一步“求模 $p$ 的剩余”的运算，即通常所说的取模运算。

a）模加减计算原理

传统的有限域 *Fp* 的模加运算和模减运算，是将加减运算后的结果约化到$[0, p-1]$的范围内。传统的有限域 *Fp* 模加和模减算法分别如算法 4.3 和算法 4.4 所示。

有限域上两个 *n* 比特的整数 *a,b* 的模加运算，先将两个数相加的和 *c* 求出来，再判断 *c* 是否大于 *p*，如果 *c*>*p*，将 *c* 减去 *p* 后返回，否则直接返回 *c*。

算法 4.3 有限域 *Fp* 上的模加法

输入：$a$，$b \in [0, p-1]$。

输出：$c = (a+b) \bmod p$。

1) $c = a + b$；

2) 若 $c \geq p$，则 $c = c - p$；

3) 返回$(c)$。

有限域上两个 *n* 比特的整数 *a,b* 的模减运算，先将两个数相减的差 *c* 求出来，再判断 *c* 是否小于 0，如果小于 0，将 *c* 加上 *p* 后返回，否则直接返回 *c*。

算法 4.4 有限域 *Fp* 上的模减法

输入：$a$，$b \in [0, p-1]$。

输出：$c = (a-b) \bmod p$。

1) $c = a - b$；

2) 若$c < 0$，则$c = c + p$；

3) 返回$(c)$。

算法 4.3 和 4.4 都是要先进行加减运算，再进行取模，这样不仅速度慢，而且还需要一个长度为 n+1 比特的中间临时变量 *c*。本文将利用多精度加减法来计算模加减法，计算方法如算法 4.5 和算法 4.6 所示。

算法 4.5 有限域 *Fp* 上的多精度模加法

输入：$a$，$b \in [0, p-1]$。

输出：$s = (a+b) \bmod p$。

1) 通过算法 4.1 计算得到$(c, s)$，其中$c$为进位；

2) 如果$c = 1$，则$s = (c, s) - (0, p)$，否则，若$s \geq p$，则 $s = s - p$；

3) 返回$(s)$。

算法 4.6 有限域 *Fp* 上的多精度模减法

输入：$a$，$b \in [0, p-1]$。

输出：$s = (a - b) \bmod p$。

1) 通过算法 4.2 计算得到$(c, s)$，其中$c$为借位；

2) 如果$c = 1$，则$s = s + p$；

3) 返回$(s)$。

b）模加减的 FPGA 实现方法

模加减模块的接口示意图如图 4-2 所示，接口信号定义如表 4-1 所示。

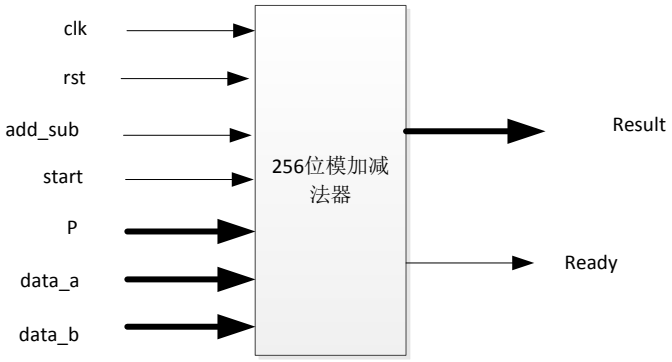


图 4-2 模加减模块接口示意图

表 4-1 模加减模块接口信号列表

| 信号名称 | 位宽 | IN/OUT | 功能描述 |
|---|---|---|---|
| clk | 1 | IN | 全局时钟 |
| rst | 1 | IN | 全局复位（高电平有效） |
| add_sub | 1 | IN | 模加模减选择信号,1 表示模加运算，0 表示模减运算，在 start 信号有效时有效。 |
| start | 1 | IN | 运算启动信号，脉冲信号。 |
| P | 256 | IN | 运算的模值 |
| data_a | 256 | IN | 运算的被加数或者被减数 |
| data_b | 256 | IN | 加数或者减数 |
| ready | 1 | OUT | 运算完成信号。为电平信号，即运算完成后一直保持高电平，直到新的一次运算开始。 |
| result | 256 | OUT | 运算结果。 |

为节省资源和方便调度，本文将模加减模块实现为同时支持加法和减法的模块[28]。实现过程中，为了提高算法 4.5 和算法 4.6 的运算效率，将数据分成宽度为

$w$ 的数据块进行计算，$w$ 越大总的循环次数越少，但加减法的位宽就越大，资源消耗越多。表 4-2 给出了数据宽度为 256 比特时，$w$ 与总循环次数和加减法位宽的关系。本文选取 $w$ 的宽度为 64，先实现一个带进位和借位的 64 位多精度加减法器，再用两级加减法循环移位进行运算。图 4-3 给出了模加减运算的实现原理，实现中包含了两级多精度加减法，第一级加减法用于 $a_i$ 与 $b_i$ 的加减计算，通过 add_sub 信号控制计算模式，模加时用于计算 $a_i+b_i$，模减时用于计算 $a_i-b_i$。第二级加减法用于取模计算，模加时计算 $c_i-p_i$，模减时计算 $c_i+p_i$。此外，需要两个寄存器存储加减法的结果 $c$ 和 $r$。$c$ 用于存储第一级加减法的结果，$r$ 用于存储第二级加减法的结果，$r$ 和 $c$ 的位宽比 $a,b$ 多 1 比特，多出的 1 比特用于存放加法的最高进位或者减法的最高借位。然后，通过判断 add_sub、$r$ 和 $c$ 的最高比特即可得到运算结果为 $c$ 或者 $r$，最后通过一个选择开关输出运算结果。

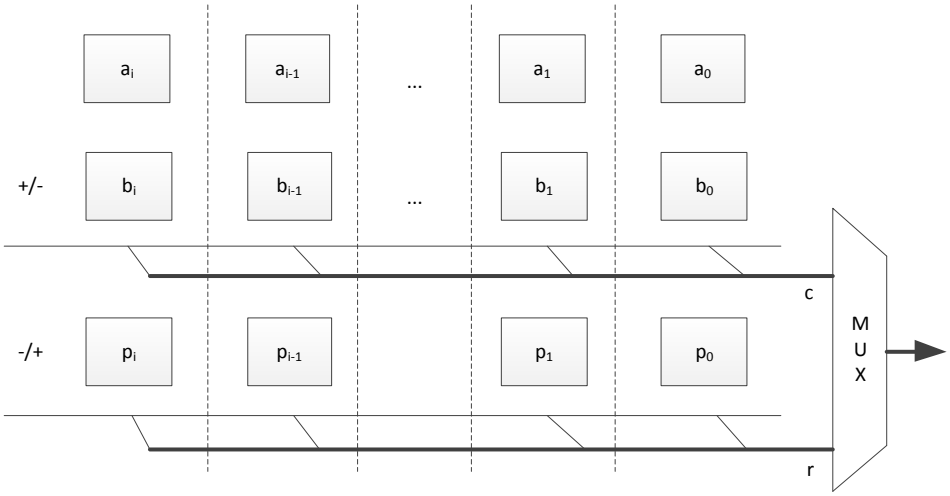


图 4-3 模加减实现原理

表 4-2 $w$ 与总循环次数和位宽的关系

| $w$ 值（比特） | 需要的循环次数（次） | 加减法的位宽（比特） |
|---|---|---|
| 2 | 128 | 2 |
| 4 | 64 | 4 |
| 8 | 32 | 8 |
| 16 | 16 | 16 |
| 32 | 8 | 32 |
| 64 | 4 | 64 |
| 128 | 2 | 128 |
| 256 | 1 | 256 |

实现过程需要将 256 比特的数据转换成 64 比特宽的数据块，转换的 VHDL 代

码如下所示：

```
FOR i IN 0 to 3 GENERATE
    num_pi(i)<=P((64*(i+1) -1) downto 64*i);          --转换模数
    data_ai(i)<=data_a ((64*(i+1) -1) downto 64*i);   --转换被加数或者被减数
    data_bi(i)<=data_b ((64*(i+1) -1) downto 64*i);   --转换加数或者减数
END GENERATE;
```

### 4.2.2 串行模乘运算

a）计算原理

域元素 $a$，$b$ 用多项式表示为 $A(x) = a_{n-1}x^{n-1} + \ldots + a_1x + a_0$ 和 $B(x) = b_{n-1}x^{n-1} + \ldots + b_1x + b_0$，则元素 $a$ 和 $b$ 的乘积公式如公式（4-1）所示：

$$c(x) = A(x).B(x) \bmod P(x) = \sum_{i=0}^{n-1}\sum_{i=0}^{n-1} a_i b_j x^{i+j} \bmod P(x) \quad (4\text{-}1)$$

这里 $P(x)$为有限域 $p$ 的多项式表示，将多项式 $A(x)$展开分别与 $B(x)$相乘，则有：

$$c(x) = a_{m-1}x^{m-1}B(x) + \ldots + a_1xB(x) + b_0B(x) \bmod P(x) \quad (4\text{-}2)$$

提取公因式 x，得到：

$$c(x) = (\ldots(((a_{m-1}B(x))x + a_{m-2}B(x))x + \ldots + a_1B(x)) + b_0B(x) \bmod P(x) \quad (4\text{-}3)$$

这种运算方式即为串行乘法。

串行乘法的计算步骤为：

1) 令 $C(x) = 0$；

2) 对 $i$ 从 $n$-1 以 1 为单位递减至 0，计算 $C(x) = (C(x) + C(x)) \bmod P(x)$；

3) 如果 $a_i$ 为 1，计算 $C(x) = C(x) + B(x)$；否则，回到步骤 2)，继续进行计算直到运算完成。

b）FPGA 实现方法

串行模乘模块的接口示意图如图 4-4 所示，其中输入参数 data_a 为被乘数，data_b 为乘数，P 为模数，start 为运算启动脉冲信号。输出参数 Ready 为计算完成脉冲信号，Result 为运算结果。

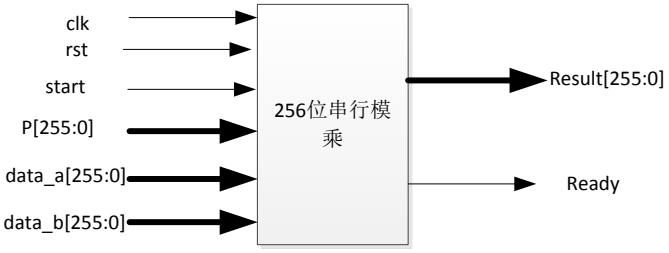


图 4-4 串行模乘模块的接口示意图

串行模乘可以用一个移位寄存器和一个模加法模块来实现。实现时定义一个临时变量 $c$，初始值为 0，将 data_a 置于移位寄存器内，由高至低逐比特移位，如果比特位为 1 则做一次模加运算 $c=(c + \text{data\_b}) \bmod p$，否则直接进行下一比特移位判断，直到 data_a 的所有比特位都判断完毕。

串行模乘模块内部状态机跳转的部分 HDL 代码如下：

```
if clk'event and clk='1' then
    case state is
    when "00"=>    --该状态控制启动运算并复位中间临时变量
        if start='1' then
            state<="01";
        end if;
    when "01"=>--该状态在 a_i 为 1 是计算 C(x)+B(x)，为 0 计算 C(x)+C(x)
        if ready_add='1' then     --模加运算完成
            if data_reg(255)='1' then
                state<="10";
            else
                if cnt=256 then     --判断计算是否结束
                    state<="11";
                end if;
             end if;
        end if;
    when "10"=>    --该状态完成 C(x)+B(x)的计算
        if ready_add='1' then
                if cnt=257 then  --判断是否为最后一次运算
                    state<="11";
                else
                    state<="01";
                end if;
          end if;
    when "11"=>
          state<="00";
    end case;
```

### 4.2.3 快速模乘运算

当前，国内外研究人员提出了很多提高大数模乘运算速度的算法，比较流行的方法有 Barrett 模乘算法及 Montgomery 模乘算法。

#### 4.2.3.1 Barrett 模乘

假设 $A = \sum_{i=0}^{n-1} A_i r^i$，$B = \sum_{i=0}^{n-1} B_i r^i$，$P = \sum_{i=0}^{n-1} P_i r^i$，其中 $r = 2^w$、$w$ 为字长。这里我们先采用先乘后模的方法计算模乘，即先通过常规方法计算 $A \times B$，再利用 $Q = \lfloor (A \times B)/P \rfloor$ 以及 $C = A \times B - Q \times P$ 来求模。算法 4.7 给出一个模乘与估商相结合的算法 CMM。

算法 4.7 CMM 算法

输入：$A$，$B$，$P$。

输出：$C = A \times B \bmod P$。

1) $C = 0$；

2) For $j = n - 1$ downto 0

2.1) $C = C \times w + A_j \times B$；

2.2) $Q = \left\lfloor C / P \right\rfloor$；

2.3) $C = C - Q \times P$；

3) 返回 $C$。

但是算法 4.7 有一个缺点，在求模过程中需要进行除法运算，当 $n$ 值很大时，除法运算的运算量将非常耗时。所以，为了避免除法运算，Barrett 于 1987 年提出了一种不带除法的估商型模乘算法。

Barrett 提出的一种新的方案是先计算出 $A \times B$，然后用一个新的变量

$$\hat{Q} = \left\lfloor \left( \left\lfloor Q \times P / 2^{k-1} \right\rfloor \times \left\lfloor 2^{2k} / P \right\rfloor \right) / 2^{k+1} \right\rfloor$$

代替原来的 $Q$。新变量 $\hat{Q}$ 里涉及的除法运算，由于是 2 的幂次，可以通过移位来实现，$\left\lfloor 2^{2k} / P \right\rfloor$ 可以进行预计算。算法 4.8 给出了 Barrett 算法的计算过程。

算法 4.8 Barrett 算法

输入：$a$，$b$，$p$，$\lambda = \left\lfloor 2^{2L+3} / p \right\rfloor$。

输出：$c = a \cdot b \bmod p$。

1) 计算 $z = a \cdot b$；

2) 计算 $\hat{q} = \left\lfloor \left( \left\lfloor z / 2^{L-2} \right\rfloor . \lambda \right) / 2^{L+5} \right\rfloor$；

3) 计算 $c = (z - \hat{q}p) \bmod 2^{L+1}$；

4) 如果 $c \geq p$，则

$c = c - p$；

5) 返回（$c$）。

#### 4.2.3.2 Montgomery 乘法

Montgomery 模乘算法把数用模 $p$ 的余数表示，把对 $p$ 的取余转化为对 2 的除法运算，在计算机系统中除 2 运算只是一个简单的移位操作，从而大大简化了计算的难度。Montgomery 模乘运算的描述如公式（4-4）所示。

$$z = MM(x, y) = x.y.R^{-1}(\mathrm{mod}\ p) = xy2^{-n}(\mathrm{mod}\ p) \qquad (4\text{-}4)$$

其中 $a,b<p$，$R$ 和 $p$ 要满足互质。具体的 montgomery 模乘算法如算法 4.9 所示。

算法 4.9 Montgomery 模乘 $MM(x,y)$

输入：$x$，$y$，$R$，$p$。

输出：$xyR^{-1}\ \mathrm{mod}\ p$。

1) $c = x \cdot y$；

2) $t = (c + (c \cdot p' \bmod R) \cdot p)/R$；

3) 若$t > p$，则$t = t - p$；

4) 返回$t$。

其中 $R^{-1}$ 是 $R$ 的逆，满足 $R \times R^{-1} = 1 \bmod p$，$p'$ 满足 $(R \times R^{-1} - p' \times p) = R^{-1} \times R - 1$，输出的结果$t = x \cdot y \cdot R^{-1} \bmod p$。

Montgomery 模乘在计算前，需要把计算的数据都转换到 Montgomery 域。比如普通数据 $x, y$ 转换到 Montgomery 域后分别为 $x' = x \cdot R(\mathrm{mod}\ p)$ 和 $y' = y \cdot R(\mathrm{mod}\ p)$，Montgomery 域的数据也属于$[0, p-1]$的范围。

两个 Montgomery 域的数据 $x'$和 $y'$进行 Montgomery 模乘运算得到的结果：

$$c' = cR(\mathrm{mod}\ p) = MM(x', y') = xyR(\mathrm{mod}\ p)$$

$c'$恰好是$c = ab(\mathrm{mod}\ p)$结果转换到 Montgomery 域的数据。图 4-5 给出了普通域到 Montgomery 域转换的关系图，从图中可以看出，一个数从普通域转换到 Montgomery 域只需要这个数与 $R^2$ 做一次 Montgomery 运算，一个数从 Montgomery 域转换到普通域只需要这个数和 1 做一次 Montgomery 运算。

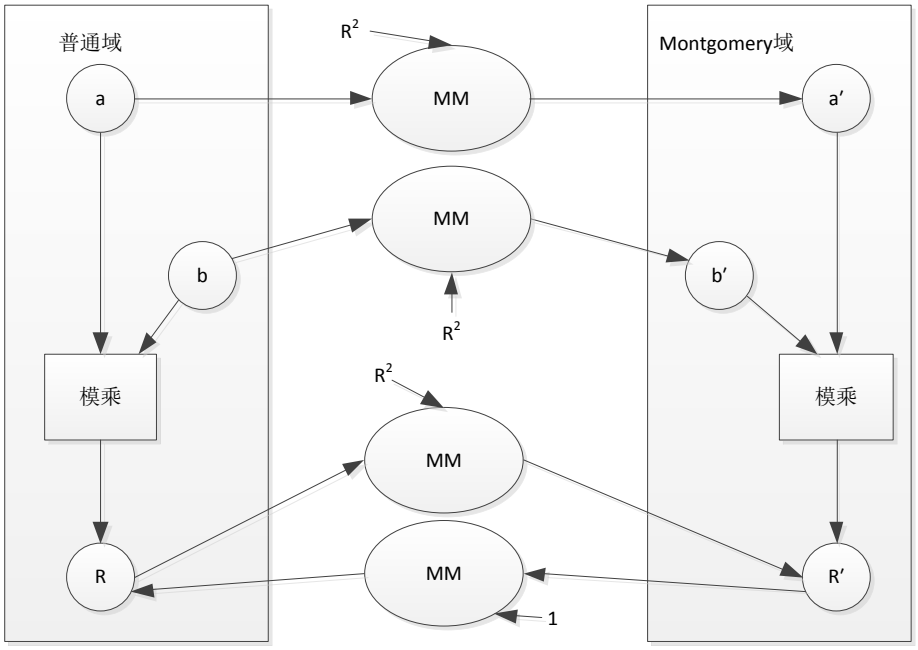


图 4-5 Montgomery 模乘域的转换关系图

如果直接利用算法 4.6 进行计算，2 个 $n$ 比特长的数据相乘需要一个 $n \times n$ 比特的乘法器和一个 $2n$ 比特的加法器，中间临时结果最大达到 $2n$ 比特，当 $n$ 较大时，实现需要的资源开销将非常大，且对于硬件实现来说编译的时钟频率也无法得到保证。所以 S.E.Eldridge 和 C.D.walter 在文献[26]中提出了利用整数的基表示($r=2^n$)将乘法和加法运算分成多次执行的方法，以解决中间临时结果太大的问题。基表示的 Montgomery 模乘算法如算法 4.10 所示。

算法 4.10 基表示的 Montgomery 模乘算法

输入：$X = \sum_{i=0}^{m-1} x_i (2^k)^i, x_i \in (0, 1, \ldots, 2^k - 1)$,

$Y = \sum_{i=0}^{m-1} y_i (2^k)^i, y_i \in (0, 1, \ldots, 2^k - 1)$,

$P = \sum_{i=0}^{m-1} y_i (2^k)^i, y_i \in (0, 1, \ldots, 2^k - 1)$,

$P' = -P^{-1} \bmod 2^k$。其中 $0 < X, Y < P$， $P < R = 2^{km}$， $\gcd(P, 2^k) = 1$。

输出：$Z = XYR^{-1} \bmod P$。

1) $Z_0 = 0$；

2) For $i$=0 to $m$-1

$q_i = (((Z_0 + x_i Y) \bmod 2^k) P') \bmod 2^k$；

$Z_{i+1} = (Z_i + q_i P + x_i Y) / 2^k$；

3) 如果 $Z_m \geq P$，则 $Z_m = Z_m - P$；

4) 返回$Z_m$。

在算法 4.10 中，引入基表示后，步骤 2 中计算$q_i$只需要进行$k \times m$比特的乘法运算和$k$比特的加法运算，计算$Z_{i+1}$时需要$k \times k.m$比特的乘法运算和$k \times k.m$比特的加法运算，中间临时数据的长度小于$k.m + k + 2$，与算法 4.9 相比，大大的降低了计算的复杂度，同时也减少了资源开销。

根据不同的使用场景，算法 4.10 可以选择低基设计和高基设计。基选择越小计算越简单，缺点是循环体执行次数将越多，从而导致整体执行速度越慢。基选择越大循环体执行的次数将越少，一方面带来了提高速度的可能，另一方面每次循环中乘法和加法运算的开销变大，关键路径时延增加，需要的资源也会增多。本文根据 FPGA 的特点，从资源开销、编译的最高频率 fmax 以及性能需求等多方面进行了评估，选择了基为 $2^{32}$ 来实现 Montgomery 模乘。基为 $2^{32}$ 的 Montgomery 模乘算法如算法 4.11 所示。

算法 4.11 基为 $2^{32}$ 的 Montgomery 模乘算法

输入：$X = \sum_{i=0}^{m-1} x_i (2^{32})^i, x_i \in (0, 1, \ldots, 2^{32} - 1)$,

$Y = \sum_{i=0}^{m-1} y_i (2^{32})^i, y_i \in (0, 1, \ldots, 2^{32} - 1)$,

$P = \sum_{i=0}^{m-1} y_i (2^{32})^i, y_i \in (0, 1, \ldots, 2^{32} - 1)$,

$P' = -P^{-1} \bmod 2^{32}$,其中$0 < X, Y < P, P < R = 2^{32m}, \gcd(P, 2^{32}) = 1$。

输出：$Z = XYR^{-1} \bmod P$。

1）$Z_0 = 0$；

2）For $i$=0 to $m$-1

$q_i = (((Z_0 + x_i Y) \bmod 2^{32}) P') \bmod 2^{32}$；

$Z_{i+1} = (Z_i + q_i P + x_i Y) / 2^{32}$；

3）如果$Z_m \geq P$，则$Z_m = Z_m - P$；

4）返回$Z_m$。

#### 4.2.3.3 快速模乘的实现

通过分析和比较 Barrett 和 Montgomery 两个算法的运算复杂度和计算特点，Montgomery 模乘算法的运算效率较高，所以本文选取 Montgomery 模乘算法来实现模乘运算。

快速模乘模块的接口示意图如图 4-6 所示，其中输入参数 data_a 为被乘数，

data_b 为乘数，P 为模数，start 为运算启动脉冲信号。输出参数 Ready 为计算完成脉冲信号，Result 为运算结果。

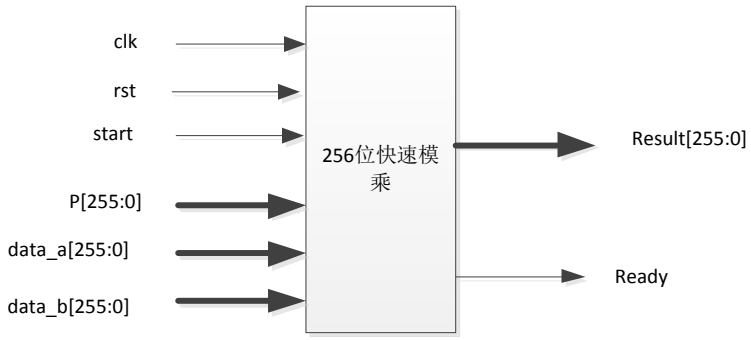


图 4-6 快速模乘模块接口示意图

在算法 4.11 的步骤 2 中，假如模乘的数据长度为 256 比特，那么需要进行 32x256 的乘法运算，如果在 FPGA 中直接进行 32x256 的运算，需要消耗很多的硬件乘法器，同时也使编译频率非常低。所以，为了解决这一问题，对算法 4.11 的步骤 2 进行了改进，改进后的算法如下所示：

算法 4.12 改进后的基为 $2^{32}$ 的 Montgomery 模乘算法

输入：$X = \sum_{i=0}^{7} x_i (2^{32})^i, x_i \in (0,1,\ldots,2^{32}-1)$,

$Y = \sum_{i=0}^{7} y_i (2^{32})^i, y_i \in (0,1,\ldots,2^{32}-1)$,

$P = \sum_{i=0}^{7} y_i (2^{32})^i, y_i \in (0,1,\ldots,2^{32}-1)$,

$P_0 = -P_0^{-1} \bmod 2^{32}$,其中$0 < X, Y < P, P < R = 2^{256}, \gcd(P, 2^{32}) = 1$。

输出：$Z = XYR^{-1} \bmod P$

1）$Z_0 = 0$；

2）for($i$=0；$i$<=7；$i$++)

$temp = Z_0 + X_i \times Y_0$；

$t_i = temp + t_i \times P_i$；

$(spill, Z_0) = temp + t_i \times P_i$；

for($j$=1；$j$<=7；$j$++)

$(spill, Z_j) = (Z_{j-1} + spill + X_i \times Y_j + t_i \times P_j) / 2^{32}$；

$Z_7 = spill$；

3）if $Z > P$ then

$Z = Z - P$；

4）返回$Z$。

改进后的算法 4.12 只需要进行 32×32 乘法运算和最长 66 比特的加法运算，从而降低了资源消耗，提高了硬件编译频率。算法 4.12 虽然优化了资源和编译频率，但增加了循环次数，采取了时间换面积的设计方法。模块的设计框图如图 4-7 所示。

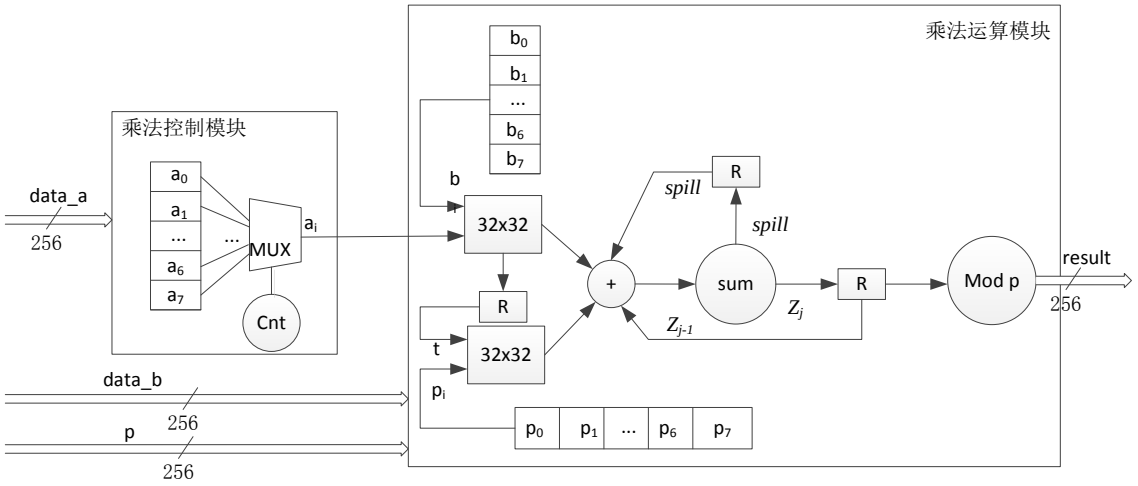


图 4-7 Montgomery 模乘结构框图

Montgomery 模乘模块由乘法控制模块和乘法处理模块组成，现对各模块实现的功能分别进行描述。

乘法控制模块为乘法处理模块准备计算的数据，实现时将其中一个 256 比特乘数 $X$ 按照 32 比特长度进行分段，共分为 $X_7$，$X_6$，$X_5$，$X_4$，$X_3$，$X_2$，$X_1$，$X_0$ 八段，每循环一次送 1 段数据到乘法处理模块。乘法控制模块共有八次循环，第一次循环将 $X_0$ 送给乘法处理模块，第二次循环将 $X_1$ 送给乘法处理模块，以此类推，第八次循环将 $X_7$ 送给乘法处理模块。每次循环等到乘法处理模块运算完成后，才进行下一次循环。

乘法处理模块实现了算法 4.12 步骤 2 中第一级循环内的所有运算，该模块内部实现了两个 32×32 的乘法器，一个乘法器用于计算 $X_i \times Y_j$，另一个乘法器用于计算 $t_i$ 以及 $t_i \times P_j$。模块内部将 256 比特的乘数 $Y$ 和模数 $P$ 按照 32 比特长度进行分段，分别分为 $Y_7$，$Y_6$，$Y_5$，$Y_4$，$Y_3$，$Y_2$，$Y_1$，$Y_0$ 和 $P_7$，$P_6$，$P_5$，$P_4$，$P_3$，$P_2$，$P_1$，$P_0$ 八段。乘法处理模块共有八次循环，第一次循环完成 $X_i \times Y_0$ 和 $t_i \times P_0$ 的运算，以及两个乘积的和运算；第二次循环完成 $X_i \times Y_1$ 和 $t_i \times P_1$ 的运算，以及两个乘法的结果与第一次循环结果的和运算；以此类推，第八次循环完成 $X_i \times Y_7$ 和 $t_i \times P_7$ 的运算，以及两个乘法的结果与第七次循环结果的和运算。八次循环完成后，得到的结果 Z 最多可能为 66 比特。

### 4.2.4 模逆运算

有限域的运算中，求逆运算是最复杂，运算最耗时的一种运算。素数域 $Fp$ 上的非零元素 $a$ 的逆记为 $a^{-1} \bmod p$，计算 $a^{-1}$ 的常见方法有扩展的 Euclidean 算法、Montgomery 求逆算法。

#### 4.2.4.1 Montgomery 求逆

Montgomery 方法已在前面的 4.2.3.2 节做了介绍，Montgomery 求逆的思路是选择一个特定的 $R$，用低成本的 $aR^{-1} \bmod p$ 运算代替复杂的 $a \bmod p$ 运算。Montgomery 求逆首先通过算法 4.13 计算出 $a^{-1}2^k \bmod p$。

算法 4.13　$Fp$ 上的 Montgomery 部分求逆算法

输入：$a \in [1, p-1]$，奇整数 $p > 2$，$n=\lceil \log_2 P \rceil + 1$。

输出：$(x, k)$，其中 $a^{-1}2^k \bmod p$，$n \le k \le 2n$。

1）令 $u = a$，$v = p$；

2）$x = 1$，$y = 0$，$k = 0$；

3）当 $v > 0$ 时，循环执行

3.1）如果 $v$ 是偶数，则 $v = v/2$，$x = 2x$；

否则，如果 $u$ 是偶数，则 $u = u/2$，$y = 2y$；

否则，如果 $v \ge u$，则 $v = (v-u)/2$，$y = y + x$，$x = 2x$；

否则，$u = (u-v)/2$，$x = x + y$，$y = 2y$；

3.2）$k = k + 1$；

4）如果 $x > p$，则 $x = x - p$；

5）返回 $(x, k)$。

在求得了 $(x, k)$ 的基础上，通过算法 4.14 来计算 $a^{-1} \bmod p$ 或者 $aR^{-1} \bmod p$，这里 $R = 2^{wt}$。

算法 4.14　$Fp$ 上的 Montgomery 求逆算法

输入：整数 $a'$ $(a' = aR \bmod p)$，奇整数 $p > 2$，$R^2 \bmod p$，且 $\gcd(a, p) = 1$。

输出：$aR^{-1} \bmod p$。

1）通过算法 4.13 求出 $(x, k)$，其中 $x = x'^{-1}2^k \bmod p$；

2）如果 $k < wt$，则

2.1）$x = MM(x, R^2) = (aR)^{-1}2^k R^2 R^{-1} = a^{-1}2^k \bmod p$，$MM$ 为蒙哥马利模乘；

2.2）$k = k + wt$；

否则 $x = MM(x, R^2) = (aR)^{-1}2^k R^2 R^{-1} = a^{-1}2^k \bmod p$；

3）$x = MM(x, 2^{2wt-k}) = a^{-1}2^k 2^{2wt-k} R^{-1} = a^{-1}2^{2wt} R^{-1} = a^{-1}R \bmod p$ (因 $R = 2^{wt}$)；

4）返回($x$)。

算法 4.14 求得的是$aR^{-1} \bmod p$，如果要求$a^{-1} \bmod p$，只需要再进行一次$MM(a^{-1}R, 1)$运算。

#### 4.2.4.2 Euclidean 求逆

$a$和$b$是两个非零整数，$a$和$b$的最大公因式表示为$\gcd(a, b)$，传统的Euclidean算法则用于求取$a$和$b$的最大公因子$d = \gcd(a, b)$。经扩展后，Euclidean 可用于求两个整数$x$和$y$，使得$ax + by = d\ (d = \gcd(a, b))$。算法 4.15 给出了扩展的整数 Euclidean 算法的运算过程。

算法 4.15 扩展的整数 Euclidean 算法[10]

输入：整数$a$和$b$，且$a \le b$。

输出：$d = \gcd(a, b)$，且满足$ax + by = d$的整数$x$和$y$。

1）令$u = a$，$v = b$；

2）令$x_1 = 1$，$x_2 = 0$，$y_1 = 0$，$y_2 = 1$；

3）当$u \neq 0$时，循环执行

3.1）$q = [v/u]$，$w = v - qu$，$x = x_2 - qx_1$，$y = y_2 - qy_1$；

3.2）$v = u$，$u = w$，$x_2 = x_1$，$x_1 = x$，$y_2 = y_1$，$y_1 = y$；

4）$d = v$，$x = x_2$，$y = y_2$；

5）返回$d$，$x$，$y$。

假设$p$是一个素数，且 $a \in [1, p-1]$， $\gcd(a, p) = 1$。将算法 4.15 中的参数$b$换成$p$，步骤 3.1 最后的非零余数$w = 1$。步骤 3.2 中的$u$，$x_1$，$y_1$更新后，使满足$ax_1 + py_1 = u$，$u = 1$。因此，$ax_1 = 1 \bmod p$，则$a^{-1} = x_1 \bmod p$。值得注意的是，确定$x_1$不需要$y_1$和$y_2$，所以可推导出素数域$Fp$上的求逆算法 4.16。

算法 4.16 利用扩展 Euclidean 求$Fq$上的逆

输入：整数$a$和素数$p$，其中$a \in [1, p-1]$。

输出：$a^{-1} \bmod p$。

1）令$u = a$，$v = p$ ；

2）令$x_1 = 1$，$x_2 = 0$；

3）当$u \neq 0$时，循环执行

3.1）$q = [v/u]$，$w = v - qu$，$x = x_2 - qx_1$ ；

3.2）$v = u$，$u = w$，$x_2 = x_1$，$x_1 = x$；

4）返回 $x_1 \bmod p$。

算法 4.16 需要进行复杂的除法运算，从而使算法的运算效率不高，也不适合

于硬件实现，为了解决这个问题，通常将除法变换成简单的移位和减法运算。算法 4.17 为改进后的求逆算法[29]。

算法 4.17 改进后的 *Fp* 求逆算法

输入：整数 $a$ 和素数 $p$，其中 $a \in [1, p-1]$。

输出：$a^{-1} \bmod p$。

1）令 $u = a$，$v = p$；

2）令 $x = 1$，$y = 0$；

3）当 $u \neq 1$ 且 $v \neq 1$ 时，循环执行

3.1）当 $u$ 为偶数，循环执行 $u = u/2$。

当 $x$ 为偶数，则 $x = x/2$，否则 $x = (x + p)/2$；

3.2）当 $v$ 为偶数，循环执行 $v = v/2$。

当 $y$ 为偶数，则 $y = y/2$，否则 $y = (y + p)/2$；

3.3）若 $u \geq v$，则 $u = u - v$，$x = x - y$ 。

否则，$v = v - u$，$y = y - x$；

4）若 $u = 1$，则返回（$x \bmod p$），否则，

返回（$y \bmod p$）。

算法 4.17 对操作数进行除 2，即右移位运算。在步骤 3 每次运算前，$u$ 和 $v$ 最多有一个为奇数，因此步骤 3.1 和步骤 3.2 的除 2 运算不改变 $\gcd(u, v)$ 的值。每次循环步骤 3.1 和步骤 3.2 后，$u$ 和 $v$ 都变成奇数，但经过步骤 3.3 后其中一个则会变成偶数。所以，步骤 3 的每一次循环都会缩减 $u$ 或 $v$ 的二进制长度，最多循环 $2n$ 次即可求得 $a$ 的逆，这里 $n$ 为 $a$ 或 $p$ 的最大二进制长度。

#### 4.2.4.3 模逆的 FPGA 实现

Montgomery 求逆算法在计算前和计算后都需要进行域的转换，单次求逆时在效率上没有优势。而扩展的 Euclidean 算法则主要采用移位和加减法单元来实现，计算效率不高，但其计算方法简单，非常适合硬件实现。本文设计的 ECC 系统只需要进行一次求逆运算，其运算效率对整个 ECC 系统的影响很小，所以本文选择扩展的 Euclidean 算法进行求逆运算。

模逆模块的接口示意图如图 4-8 所示，其中输入参数 data_a 为被乘数，data_b 为乘数，P 为模数，start 为运算启动脉冲信号。输出参数 Ready 为计算完成脉冲信号，Result 为运算结果。

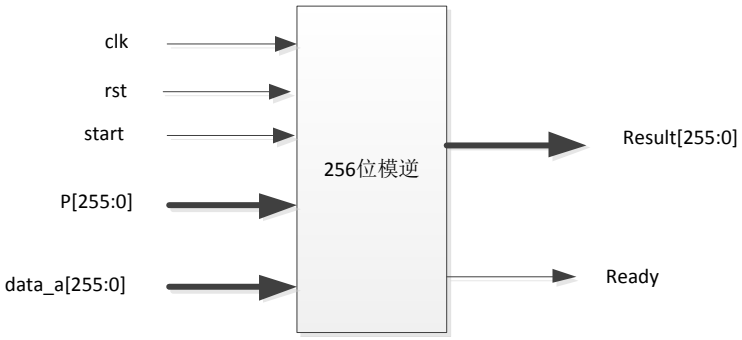


图 4-8 模逆模块接口示意图

本文的模逆运算模块采用了两个模加减法模块、一个比较模块和一个主控模块来实现，实现的框图如图 4-9 所示。

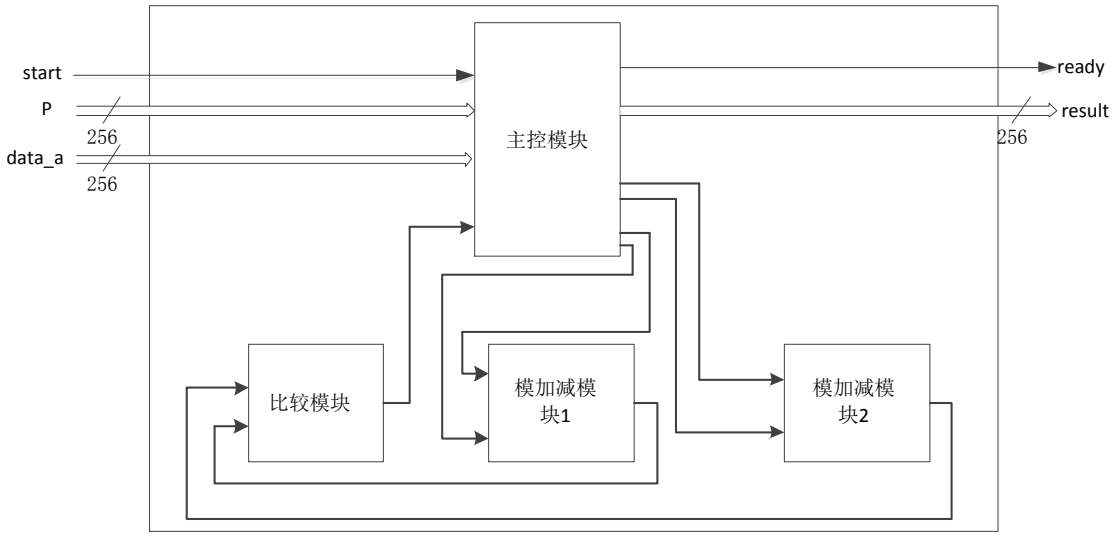


图 4-9 模逆内部子模块框图

模加减模块 1 用于计算 $x + p$，以及 $u \geq v$ 时计算 $u - v$，$u < v$ 计算 $v - u$。

模加减模块 2 用于计算 $y + p$，以及 $u \geq v$ 时计算 $x - y$，$u < v$ 计算 $y - x$。

比较模块用于比较 $u$ 和 $v$ 的大小，将比较结果反馈到主控模块，从而控制两个模加减模块的输入数据。$u$ 和 $v$ 通常是很大的数据，直接比较将影响编译时序，因此本模块采用分段比较的方法进行实现，每段数据为 64 比特。

主控模块内包含了一个主状态机和两个子状态机，主状态机用于实现算法 4.17 的运算过程，并根据比较模块的结果调度两个模加减模块的运算数据。两个子状态机分别用于实现算法 4.17 中的步骤 3.1 和步骤 3.2 的中的运算。主状态机的状态转移图如图 4.10 所示。

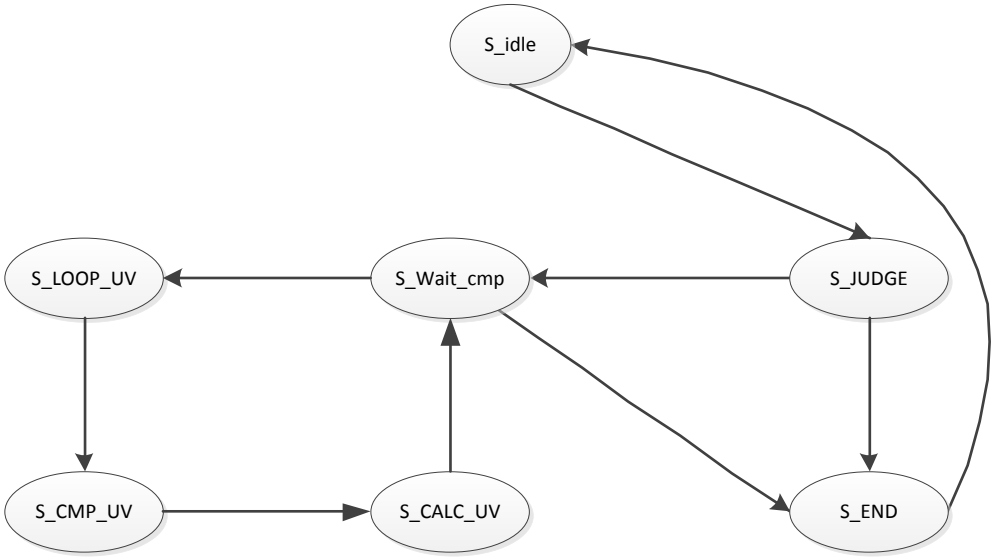


图 4.10 主状态机状态转移图

状态介绍：

S_idle：为初始状态，复位所有中间临时寄存器变量。运算启动信号 Start 有效时跳转。

S_JUDGE：判断状态，判断输入的 data_a 是否为 0，为 0 则结束运算。

S_Wait_cmp：判断 $u$ 和 $v$ 是否有一个为 1，是则结束运算。

S_LOOP_UV：等待两个子状态机运算结束。

S_CMP_UV：比较 $u$ 和 $v$ 的大小。

S_CALC_UV：计算 $u$ 和 $v$ 的差，以及 $x$ 和 $y$ 的差。

S_END：计算完成，输出求逆结果。

两个子状态机的状态转移图如图 4.11 所示

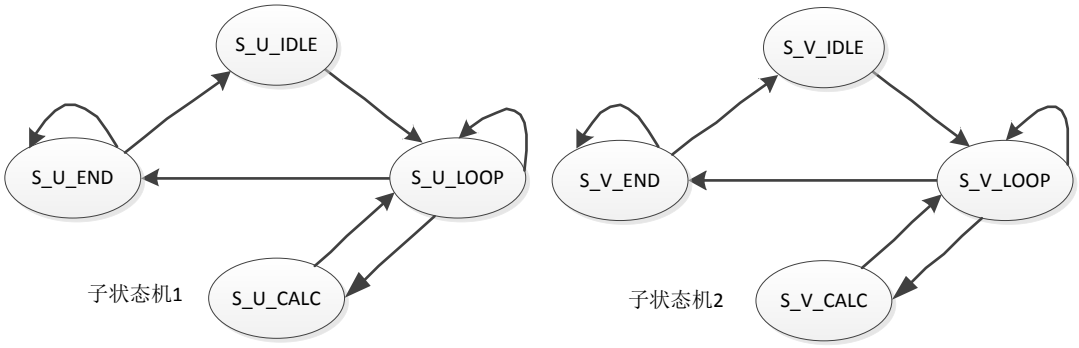


图 4.11 子状态机状态转移图

由于子状态机 1 和子状态机的各状态机定义和跳转条件都是相同的，下面仅对子状态机 1 的状态进行介绍。

S_U_IDLE：为空闲等待状态，在主状态机为 S_LOOP_UV 状态时跳转到 S_U_LOOP 状态。

S_U_LOOP：先判断 $u$ 的奇偶性，为奇数时跳转到 S_END，为偶数时再判断 $x$ 的奇偶性，$x$ 为奇数则跳转到 S_U_CACL，$x$ 为偶数则维持本状态。

S_U_CACL：用于计算 $x + p$。

S_END：在主状态机处于 S_CMP_UV 状态时，跳转至空闲状态，否则维持本状态。

## 4.3 椭圆曲线运算模块设计与实现

### 4.3.1 射影坐标

通过 2.3.2 节可知，仿射坐标下的点加运算和倍点运算各需要一次有限域的模逆运算，而模逆运算非常耗时。为了解决这个问题，需引入了射影坐标来消除模逆运算。素域 $Fp$ 上的射影坐标常用的有标准射影坐标和雅克比射影坐标。

#### 4.3.1.1 标准射影坐标

在标准射影坐标系中，素域 $Fp$ 上的椭圆曲线方程为

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

椭圆曲线上的无穷点表示为$(0:1:0)$。$Z \neq 0$ 时，射影坐标上的点$(X:Y:Z)$与仿射坐标中的点$(X/Z, Y/Z)$对应，点$(X:Y:Z)$的负点是$(X:-Y:Z)$。

令 $P = (X_1:Y_1:Z_1)$，$Q = (X_2:Y_2:Z_2)$分别是椭圆曲线上的两个点，且 $P \neq Q$。设 $P + Q = (X_3:Y_3:Z_3)$，则

$$\begin{aligned} X_3 &= AC \\ Y_3 &= B(A^2X_1Z_2 - C) - A^3Y_1Z_2 \\ Z_3 &= A^3Z_1Z_2 \end{aligned} \tag{4-5}$$

其中，$A = X_2Z_1 - X_1Z_2$，$B = Y_2Z_1 - Y_1Z_2$，$C = B^2Z_1Z_2 - A^2 - 2A^2X_1Z_2$。

设 $2P = (X_3:Y_3:Z_3)$，则

$$\begin{aligned} X_3 &= 2AD \\ Y_3 &= B(4C - D) - 8Y_1^2A^2 \\ Z_3 &= 8A^3 \end{aligned} \tag{4-6}$$

其中，$A = Y_1Z_1$，$B = aZ_1^2 + 3X_1^2$，$C = X_1Y_1A$，$D = B^2 - 8C$。

#### 4.3.1.2 雅克比射影坐标

在雅克比射影坐标系中，素域 $Fp$ 上的椭圆曲线方程为：

$$Y^2 = X^3 + aXZ^4 + bZ^6 \tag{4-7}$$

椭圆曲线上的无穷点表示为$(0:1:0)$。$Z \neq 0$时，射影坐标上的点$(X:Y:Z)$与仿射坐标中的点$(X/Z^2, Y/Z^3)$对应，点$(X:Y:Z)$的负点是$(X:-Y:Z)$。

令$P=(X_1:Y_1:Z_1)$，$Q=(X_2:Y_2:Z_2)$分别是椭圆曲线上的两个点，且$P \neq Q$。设$P+Q=(X_3:Y_3:Z_3)$，则

$$\begin{aligned} X_3 &= F^2 - E^3 - 2BE^2 \\ Y_3 &= F(BE^2 - X_3) - DE^3 \\ Z_3 &= Z_1Z_2E \end{aligned} \tag{4-8}$$

其中，$A=X_2Z_1^2$，$B=X_1Z_2^2$，$C=Y_2Z_1^3$，$D=Y_1Z_2^3$，$E=A-B$，$F=C-D$。

设$2P=(X_3:Y_3:Z_3)$，则

$$\begin{aligned} X_3 &= A^2 - 2B \\ Y_3 &= A(B - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned} \tag{4-9}$$

其中，$A=3X_1^2+aZ_1^4$，$B=4X_1Y_1^2$。

令$P=(X_1:Y_1:Z_1)$，$Q=(X_2:Y_2:1)$其中 $P$ 是在雅克比射影坐标系下表示的点，$Q$ 是在仿射坐标系下表示的点，且$P \neq \pm Q$。设$P+Q=(X_3:Y_3:Z_3)$的雅可比坐标算法为：

$$\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1)^2 - ((X_2Z_1^2 - X_1)^3 + 2X_1(X_2Z_1^2 - X_1)) \\ Y_3 &= (X_1(X_2Z_1^2 - X_1)^2 - X_3)(Y_2Z_1^3 - Y_1) - Y_1(X_2Z_1^2 - X_1)^3 \\ Z_3 &= (X_2Z_1^2 - X_1)Z_1 \end{aligned} \tag{4-10}$$

通过存储一些中间变量，可以将上述算法变换为：

$$\begin{aligned} X_3 &= E^2 - (I + 2H) \\ Y_3 &= E(H - X_3) - Y_1I \\ Z_3 &= Z_1F \end{aligned} \tag{4-11}$$

其中，$A=Z_1^2$，$B=Z_1A$，$C=Y_2B$，$D=X_2A$，$E=C-Y_1$，$F=D-X_1$，$G=F^2$，$I=GF$，$H=X_1G$。

#### 4.3.1.3 坐标系选取

假设用 M、S、AS 分别表示模乘、模平方、模加减运算，那么不同坐标系的点加和倍点运算次数如表 4-3 所示。

表 4-3 坐标系的运算比较

| 有限域 | 坐标系 | 点加 | 倍点 |
|---|---|---|---|
| GF(p) | 标准坐标系 | 12M+2S+6AS | 7M+5S+10AS |
| | 雅克比坐标系 | 12M+4S+7AS | 4M+6S+12AS |
| | 雅克比混合坐标系 | 8M+3S+6AS | —— |

从表 4-3 可以看出，标准坐标系下的点加运算次数比雅克比坐标系下的点加运算次数少，倍点的运算次数比雅克比坐标系下的倍点运算次数多，雅克比混合坐标系下的点加运算次数最少。所以，本文选择雅克比混合坐标系来实现点加运算，雅克比坐标系来实现倍点运算。

### 4.3.2 点加倍点运算

为了尽可能的节省资源开销，本文将点加运算和倍点运算封装在一起。点加倍点运算模块的接口示意图如图 4-12 所示。输入参数 P1X，P1Y，P1Z 为点 P1 的坐标，P2X，P2Y，P2Z 为点 P2 的坐标，start 为运算启动脉冲信号，mode 为点加或倍点运算选择信号。输出参数 Ready 为计算完成脉冲信号，P3X，P3Y，P3Z 为运算结果点 P3 的坐标。点加运算时 P1 点和 P2 点的坐标值均有效，倍点运算是只有 P1 点的坐标值有效。

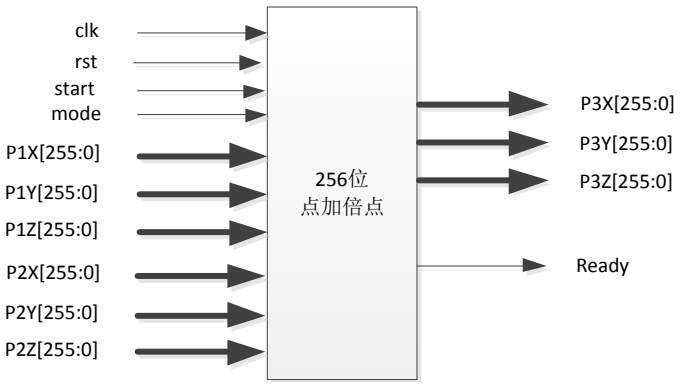


图 4-12 点加倍点模块接口示意图

在点加和倍点运算中，模平方可以通过模乘实现，模加和模减通过一个统一的模加减模块实现。本文采用了 1 个模乘模块，一个模加减法模块，并引入一些临时变量，对其进行处理[14]。实现的点加算法如算法 4.18 所示，倍点算法如算法 4.19 所示，其中 $t_x$ 为中间临时变量，*null* 为空操作。

算法 4.18 点加调度算法

输入：$P=(X_1,Y_1,Z_1)$，$Q=(x_2,y_2)$。

输出：$R=P+Q=(X_3,Y_3,Z_3)$。

1) $t_1=Z_1Z_1$ *null*;

2) $t_2 = t_1 Z_1$ *null*;

3) $t_3 = t_1 x_2$ *null*;

4) $t_4 = t_2 y_2$ $t_5 = t_3 - X_1$;

5) $t_6 = t_5 t_5$ $t_7 = t_4 - Y_1$;

6) 如果$t_5 = 0$，则

6.1）若$t_7 = 0$，则$(X_3, Y_3, Z_3) = 2(X_1, Y_1, Z_1)$，返回利用倍点公式计算；

6.2）否则，返回$R = O$；

7) $t_8 = t_7 t_7$ $t_9 = t_3 + X_1$;

8) $t_9 = t_6 t_9$ *null*;

9) $t_9 = t_5 t_6$ $X_3 = t_8 - t_9$;

10) $t_9 = t_9 Y_1$ *null*;

11) $t_6 = X_1 t_6$ *null*;

12) $Z_3 = t_5 Z_1$ $t_6 = t_6 - X_3$;

13) $t_6 = t_7 t_6$ *null*;

14) *null* $Y_3 = t_6 - t_9$。

算法 4.18 共有 13 个运算步骤。第 1 步是模乘运算，模加减模块则处于空闲状态。第 4 步是模乘模块和模减模块同时进行运算。第 13 步只进行模减运算，模乘模块则处于空闲状态。为了存取数据方便，将所有的中间变量都存储在 FPGA 的寄存器里。由于模乘运算与模加减运算的计算时间不同，为了避免计算混乱，规定每一个步骤都必须等到所有运算都完成后才进入下一个步骤。算法 4.18 中的乘法运算次比表 4-1 中混合坐标系下点加的乘法运算次数要多 1 次，那是因为实现时为了少用临时变量，多计算了 1 次乘法。此外，混合坐标系下点加的两个点无法直接判断是否为同一个点，在算法 4.18 计算过程中，步骤 6）对点进行判断，如果 $t_5$ 和 $t_7$ 同时为 0，则说明此时点加运算的两个点为同一个点，需要调用倍点的算法来计算。如果 $t_5$=0 且 $t_7$/=0，则返回点加的结果为无穷远点。

算法 4.19 倍点调度算法

输入：$P = (X_1, Y_1, Z_1)$。

输出：$R = 2P = (X_3, Y_3, Z_3)$。

1) $t_1 = Z_1 Z_1$ $t_2 = X_1 + X_1$;

2) $t_1 = t_1 t_1$ $t_2 = t_2 + t_2$;

3) $t_1 = a t_1$ $t_3 = t_2 + t_2$;

4) $t_4 = X_1 X_1$ *null*;

5) $t_5 = Y_1 Y_1$ $t_6 = t_4 + t_4$;

6) $t_4 = t_2 t_5$ $t_6 = t_4 + t_6$；

7) $t_7 = t_5 t_5$ $t_6 = t_6 + t_1$；

8) $t_1 = t_6 t_6$ $t_3 = t_4 + t_4$；

9) $t_3 = Y_1 Z_1$ $X_3 = t_1 - t_3$；

10) $t_7 = 8t_7$ $t_4 = t_4 - X_3$；

11) $t_4 = t_6 t_4$ $Z_3 = t_3 + t_3$；

12) *null* $Y_3 = t_4 - t_7$。

为了减小硬件实现模乘与模加减单元的复杂度，算法 4.19 把有些倍乘运算转化为多次模加运算实现，如步骤 5)和步骤 6)的两次模加运算完成的是$3X_1^2$运算。但是，为了尽可能让每个运算步骤内的模乘运算模块和模加运算模块不出现空闲状态，有些倍乘运算则采用乘法模块来实现，如步骤 10)中的$8t_7$完成的则是$8Y_1^4$运算。

### 4.3.3 点乘运算

椭圆曲线密码算法中的协议层都会用到点乘运算，如公钥加密算法里的$[k]P_B$以及签名算法里的$[k]G$。

点乘运算内包含了点加运算和倍点运算，这里将点乘运算、点加运算和倍点运算都称为点运算。通过分析 2.4 节里的三种算法协议可知，协议层只会使用到点乘运算和点加运算。为了减少资源，尽可能的复用点乘运算和点加运算过程中的临时变量，本文使用一套接口与其他模块对接。256 位点运算模块的接口如图 4-13 所示，接口信号定义如表 4-4 所示。下面将详细介绍点乘运算的计算原理和实现方法。

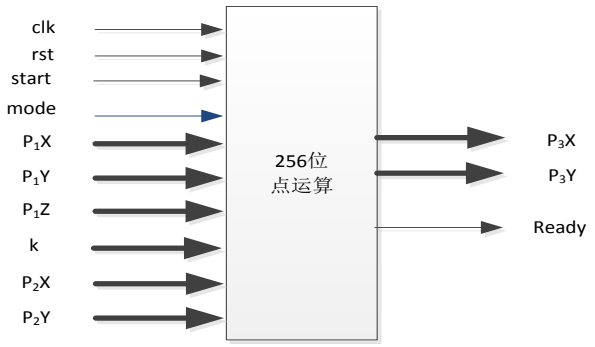


图 4-13 点乘模块接口示意图

表 4-4 点乘模块接口信号列表

| 信号名称 | 位宽 | IN/OUT | 功能描述 |
|---|---|---|---|
| clk | 1 | IN | 全局时钟。 |
| rst | 1 | IN | 全局复位（高电平有效）。 |
| start | 1 | IN | 运算启动信号，脉冲信号，高有效。 |
| mode | 1 | IN | 运算模式 1 为点乘，0 为点加。 |
| $P_1X$ | 256 | IN | 点加运算的第一个点的 X 坐标，或者点乘运算的 X 坐标。 |
| $P_1Y$ | 256 | IN | 点加运算的第一个点的 Y 坐标，或者点乘运算的 Y 坐标。 |
| P1Z | 256 | IN | 点加运算的第一个点的 Z 坐标，点乘运算时该信号无意义。 |
| k | 256 | IN | 点乘运算的整数，点加运算时无意义。 |
| $P_2X$ | 256 | IN | 点加运算的第二个点的 X 坐标，点乘运算时该信号无意义。 |
| $P_2Y$ | 256 | IN | 点加运算的第二个点的 Y 坐标，点乘运算时该信号无意义。 |
| ready | 1 | OUT | 运算完成信号，脉冲信号，高电平有效。 |
| $P_3X$ | 256 | OUT | 运算结果的 x 坐标。 |
| $P_3Y$ | 256 | OUT | 运算结果的 y 坐标。 |

#### 4.3.3.1 点乘运算设计

椭圆曲线上的点乘运算是通过调用点加和倍点来实现的，最简单的是基于二进制的方法。点乘运算 $kP$ 中的 $k$ 用二进制表示为 $k = \sum_{i=0}^{l-1} k_i 2^i$，其中 $k_i = (0,1)$。算法 4.20 是从右至左的二进制计算方法，算法 4.21 是从左至右的二进制计算方法。

算法 4.20 点乘从右至左的二进制计算方法[31]

输入：$k = (k_{l-1}, \ldots k_1, k_0)_2$，$Q \in E(GF(p))$。

输出：$kQ$。

1) $P \leftarrow O$；

2) 对 $i$ 从 0 到 $l-1$，循环执行

2.1) 如 $k_i = 1$，则 $P \leftarrow P + Q$；

2.2) $Q \leftarrow 2Q$；

3) 返回 $Q$。

算法 4.21 点乘从左至右的二进制计算方法

输入：$k = (k_{l-1}, \ldots k_1, k_0)_2$，$Q \in E(GF(p))$。

输出：$kQ$。

1) $P \leftarrow O$；

2) 对 $i$ 从 0 到 $l-1$，循环执行

2.1) $P \leftarrow 2P$；

2.2) 如 $k_i = 1$，则 $P \leftarrow P + Q$；

3) 返回 $Q$。

在算法 4.20 中，如果整数 $k$ 的二进制长度为 256 比特，步骤 2）的循环次数将为 256 次。如果 $k$ 用二进制表示后，0 和 1 出现的概率相等，即 0 和 1 的个数各占 128 个。步骤 2.2）将运算 256 次，步骤 2.1）将运算 128 次。由于初始的 $P = O$，$P + Q$ 直接等于 $Q$ 不需要计算，步骤 2.1）的实际运算次数为 127 次。所以，在 $k$ 为 256 比特时，算法 4.21 的平均计算量为点加运算 127 次，倍点计算 256 次。

在算法 4.21 中，如果整数 $k$ 的二进制长度为 256 比特，步骤 2）的循环次数将为 256 次。如果 $k$ 用二进制表示后，0 和 1 出现的概率相等，各占 128 个。步骤 2.1）理论计算 256 次，由于初始的 $P = O$，首次 $2P$ 不需要计算，所以实际运算次数为 255 次。步骤 2.2)的计算次数为 128 次。在 $k$ 为 256 比特时，算法 4.21 的平均计算量为点加运算 128 次，倍点计算 255 次。

通过上述对算法 4.20 和 4.21 的分析可以看出，两种算法的运算量基本相同，都要执行 $l$ 次倍点运算以及 $l/2$ 次点加运算，$l$ 为 $k$ 的二进制表示的长度。算法 4.20 和算法 4.21 的差异在于点加和倍点的计算顺序不同，算法 4.20 是先计算点加，再计算倍点，倍点的计算与点加的结果没有关系。算法 4.21 是先计算倍点，再计算点加，点加的其中一个输入为倍点运算的结果。所以，算法 4.20 的点加运算和倍点运算可以同时进行，算法 4.21 只能先计算倍点，再计算点加。算法 4.20 计算时，由于点加和倍点同时运算，点加运算和倍点运算内部的资源必须独立实现，不能共用，所以在 FPGA 资源丰富的情况下可以采用算法 4.20 来提高点乘的运算性能。本文考虑到 FPGA 的资源问题，选择了算法 4.21 的计算方法进行研究。

#### 4.3.3.2 点乘运算实现

点乘运算模块由一个控制进程来循环调用点加运算和点乘运算。控制进程用于实现算法 4.21 中的步骤 2），完成整个运算需要 256 次循环，每次循环先进行倍点运算，再判断 $k_i$ 值是否为 0，为 0 则结束本次循环，不为 0 则将 $Q$ 值与前面倍点的结果做点加运算。判断 $k_i$ 时，将 $k$ 值存放在一个 256 比特的移位寄存器 REG 中，

每循环一次将移位寄存器向高位进行一次移位，对 $k_i$ 的判断即可转化为对比特 REG(255)的判断。与直接判断 $k_i$ 相比，减少了判断前的选择逻辑，从而节省了 FPGA 的逻辑资源。

## 4.4 椭圆曲线算法协议实现

本文的椭圆曲线算法协议包含了三个：签名算法协议、公钥加密算法协议和密钥对生成算法协议。协议层模块的接口如图 4-14 所示，其中 load 为数据加载脉冲信号，intype 指示加载的是数据类别，datain 为输入的数据总线，start 为运算启动脉冲信号，mode 为协议指示信号，dval 为输出结果有效脉冲，intype 指示输出的什么数据，dout 为输出数据总线。

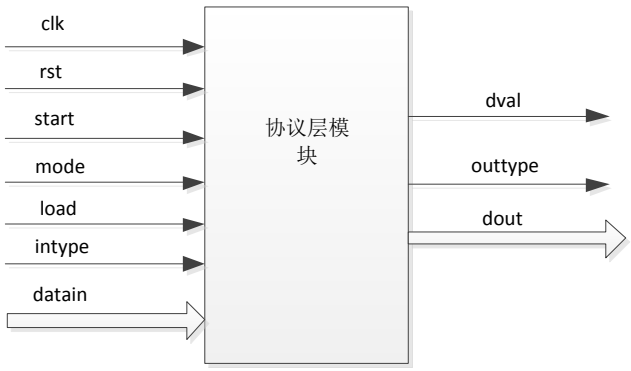


图 4-14 协议层模块接口示意图

从 2.4 节中各个算法协议的描述可知，每种协议的运算各不相同，但都可以归纳为椭圆曲线的运算和有限域的运算。为了节省资源，本文未将椭圆曲线的每个运算协议实现为单独的模块，而是将所有协议合并在一起实现。协议层模块实现的框图如图 4-15 所示。

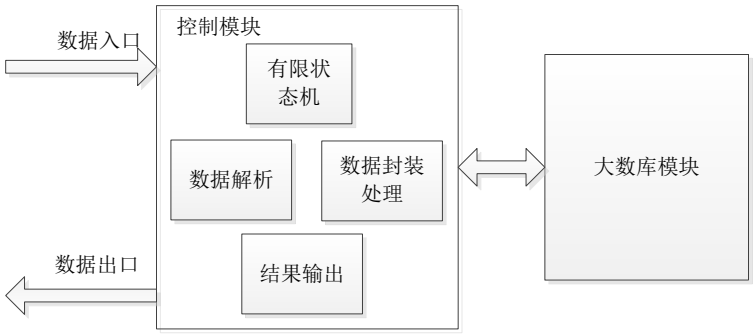


图 4-15 协议层模块内部框图

控制模块主要包含了数据解析、数据封装处理、结果输出和有限状态机等进程。数据解析进程用于处理外部输入的数据，根据数据的类型将其存放入在内部

临时寄存器中；数据封装处理进程在有限状态机的控制下，将内部临时寄存器中的数据封装后送给大数库模块，运算完成后将结果存入内部临时寄存器中；结果输出进程根据结果的类型，返回运算结果。有限状态机进程根据各运算条件控制状态机的跳转，实现三种不同的算法协议，协议层模块的状态转移图如图 4-16 所示。

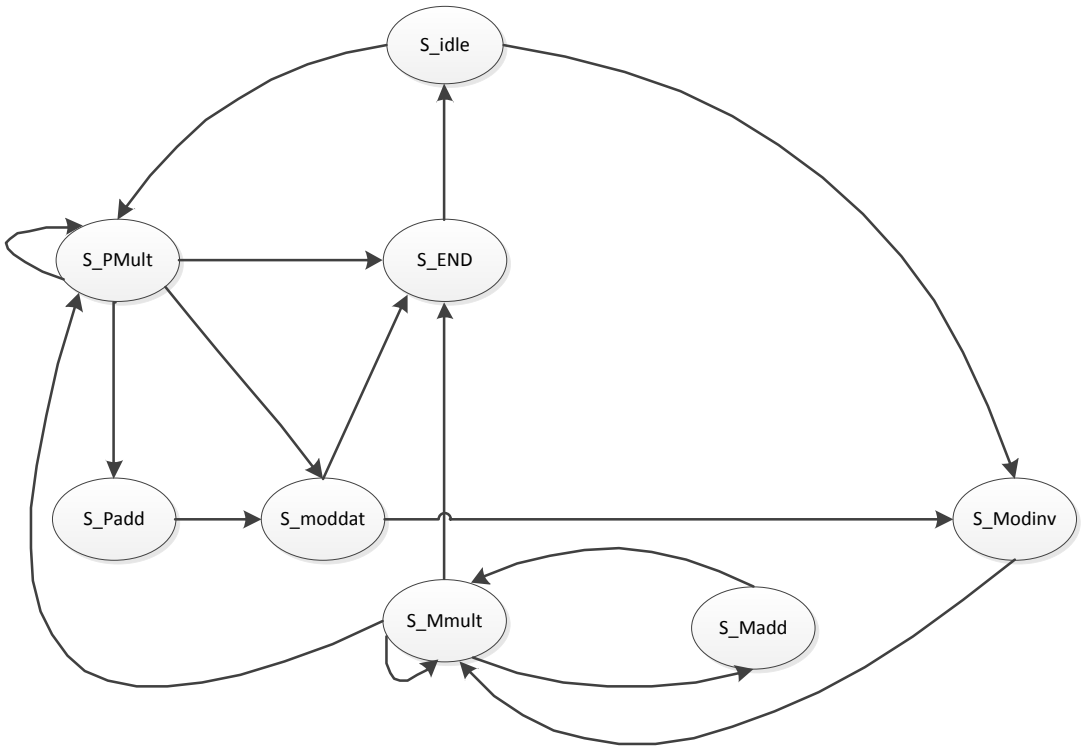


图 4-16 状态机转移图

状态介绍：

S_idle：初始状态，复位所有中间临时寄存器。根据 Start 信号和 mode 信号的值进行对应的状态跳转。

S_PMult：点乘运算状态。完成签名验证、加解密和生成密钥对的点乘运算，运算完成后根据 mode 信号进行对应的状态跳转。

S_moddat：取模运算状态。完成签名验证算法内的取模运算。

S_Padd：点加运算状态。完成验证算法内的点加运算。

S_Mmult：模乘运算状态。完成签名验证算法内模乘运算。

S_Madd：模加减运算状态。完成签名验证算法内模加减运算。

S_Modinv：模逆运算状态。完成签名验证算法内模逆运算。

S_end：结束状态。该状态用于对运算结果的封装输出。

大数库模块将有限域上的各运算模块和椭圆曲线上的各运算模块封装在一起，其接口信号列表如表 4-5 所示。

表 4-5 大数库模块信号列表

| 信号名称 | 位宽 | IN/OUT | 功能描述 |
|---|---|---|---|
| clk | 1 | IN | 全局时钟 |
| rst | 1 | IN | 全局复位（高电平有效） |
| point_start | 1 | IN | 点运算启动信号 |
| point_done | 1 | OUT | 点运算完成信号 |
| point_load | 1 | IN | 点运算数据输入有效信号 |
| point_intype | 3 | IN | 点运算数据输入类型 |
| point_din | 256 | IN | 点运算数据输入 |
| point_dval | 1 | OUT | 运算结果输出有效信号 |
| point_outype | 2 | OUT | 运算结果输出类型 |
| point_dout | 256 | OUT | 运算结果输出 |
| modinv_Start | 256 | IN | 模逆运算启动信号 |
| modinv_num | 1 | IN | 模逆运算的模数 |
| modinv_Ready | 1 | OUT | 模逆运算完成信号 |
| modinv_ain | 256 | IN | 待求逆的数据 |
| modinv_cout | 256 | OUT | 模逆运算的结果 |
| modmul_start | 1 | IN | 模乘运算启动信号 |
| modmul_done | 1 | OUT | 模乘运算完成信号 |
| modmul_p | 256 | IN | 模乘运算的模数 |
| modmul_dina | 256 | IN | 模乘运算的被乘数 |
| modmul_dinb | 256 | IN | 模乘运算的乘数 |
| modmul_dout | 256 | OUT | 模乘运算的结果 |
| modadd_start | 1 | IN | 模加减运算启动信号 |
| modadd_mode | 1 | IN | 模加运算或模减运算指示。 |
| modadd_done | 1 | OUT | 模加减运算完成信号 |
| modadd_p | 256 | IN | 模加减运算的模数 |
| modadd_dina | 256 | IN | 模加运算的被加数或模减运算的被减数 |
| modadd_dinb | 256 | IN | 模加运算的加数或模减运算的减数 |
| modadd_dout | 256 | OUT | 模加运算或模减运算的结果 |

## 4.5 本章小结

本章介绍了 FPGA 的发展及工作原理，从代码开发到最后的芯片配置调试详细介绍了 FPGA 的设计开发流程，详述了每一步开发流程所完成的工作。

有限域运算是 ECC 系统实现的基础，ECC 的数学运算最终都会调用有限域的运算来实现。本章从理论出发介绍了各种有限域运算的计算原理，并结合 FPGA 的硬件特性，从资源和性能两方面综合考虑，给出了适合 FPGA 的实现方法。此外，通过分析 ECC 系统的计算特性，根据使用频率对运算模块进行了差异化的设计实现。对频繁调用的关键运算采用了性能优先的实现策略，运用了多种快速实现手段来提高运算效率。对于使用频率低的运算模块，若采用性能优先的方式实现需要消耗更多的资源，然而对 ECC 的整体性能提高不明显，往往会得不偿失。所以，这种使用频率低的模块采用了资源优先的实现策略。

椭圆曲线上有点加、倍点和点乘三种运算，点加和倍点是计算点乘的基础。本文分析了几种射影坐标系下点加和倍点的计算量，再结合 FPGA 的并行计算特性，提出了适合 FPGA 实现的计算公式，并给出了具体的实现方法。点乘作为 ECC 系统的核心运算，它的运算效率基本上就是 ECC 系统的运行效率。本章根据点乘运算的特点和 FPGA 的芯片特性，设计和实现了适合于 FPGA 的点乘算法模块。

将有限域上的运算模块和椭圆曲线上的运算模块封装成大数库模，并在此基础上实现了椭圆曲线算法协议。

# 第五章 仿真和验证

## 5.1 仿真与分析

### 5.1.1 仿真环境搭建

Modelsim 是 Mentor Graphics 公司开发的一款 HDL 硬件描述语言的仿真软件，该软件可以实现对 VHDL 语言、Verilog HDL 语言或者两种语言混合的程序进行仿真，同时也支持 IEEE 常见的各种硬件描述语言标准，是 FPGA 和 ASIC 硬件设计开发中的一款常用的软件。具有单内核 VHDL 和 Verilog 混合仿真、systemC 和 HDL 的混合仿真、前仿真和后仿真、C 和 Tcl/Tk 脚本接口、C 调试等功能，集成了性能分析，波形比较，存储块内容查看窗口，手动改变信号激励等多种调试手段。

Modelsim 的仿真主要有以下几步：建立仿真库；编写仿真激励文件；编译源代码和 Testbench 执行仿真。

a）建立仿真库

Modelsim 的库可以分成两类：一类是通用库，通用库是将已经编译好的文件所对应的目录映射到 Modelsim 中，只需创建一次，后续在创建仿真工程时 Modelsim 工具会自动将这类库添加到工程中；另一类是工作库，其默认的名字为 work。工作库用于存放自己开发设计的程序模块以及仿真用的激励文件。

Modelsim 在安装过程中已经创建了一些通用库，如 ieee、std 等，这些通用库是 VHDL 程序开发所需要的库文件，只能满足基本的 VHDL 模块仿真。本文中的部分 VHDL 程序模块使用了 FPGA 厂商独有的一些模块（如 RAM、FIFO、LPM 等），则需要将 FPGA 厂商提供的仿真库编译到 Modelsim 的通用库中，Xilinx 公司的仿真库可以通过开始菜单->Xilinx 安装目录->ISE design tools->simulation library compilation wizard 工具自动编译。编译仿真库时，首先在 modelsim 安装目录下新建一个 Xilinx_lib 的文件夹；然后打开 Modelsim 软件，将路径指到 Xilinx_lib 这个文件夹下，将 Xilinx 的仿真库编译到 Xilinx_lib 文件夹中；最后在 Modelsim 安装目录下，修改配置文件 modelsim.ini 里所编译生成的 Xilinx 仿真库的路径。

工作库 work 在新建 Modelsim 工程时自动建立，将自己开发的程序模块和仿真文件手动添加到 work 库中即可进行编译仿真。

b）编写仿真激励文件

仿真激励文件对仿真的模块提供时钟和仿真的数据，本文实现时为每个仿真模块的输入接口定义的信号变量，并给信号变量赋相应的测试值。

c）执行仿真

建立好通用库和工作库后，对工作库内的所有模块进行完全编译，编译通过后，将每个模块的输入输出信号添加到 wave 界面中，执行 run 命令，即可查看每个模块的输入输出信号值进行正确性比对。

### 5.1.2 仿真结果

本章以椭圆曲线方程：$y^2= x^3+ ax + b$ 和 NIST 推荐的曲线参数[31]为例对有限域 *Fp* 运算模块和椭圆曲线运算模块进行验证，其中椭圆曲线参数的参数（*p*，*a*，*b*，*gx*，*gy*，*n*，*h*）分别为：

*p*=FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF，

*a*=FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFC,

*b*=5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B，

*gx*=6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296，

*gy*=4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5，

*gy*=4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5，

*n*=FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551。

a）模加运算模块

输入被加数 modadd_a，加数 modadd_b 和模数 modadd_p：

modadd_a=85832B3916D247EF1E218CB5D4E9D01AFAAC6CAEC146AA8E0DA56A4BC69B069A，

modadd_b=B524F552CD82B8B028476E005C377FB19A87E6FC682D48BB5D42E3D9B9EFFE76，

modadd_p=FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551。

仿真输出：

modadd_dout=3AA8208CE455009E4668FAB631214FCCD84D58FD825C54C4772E83628427DFBF。

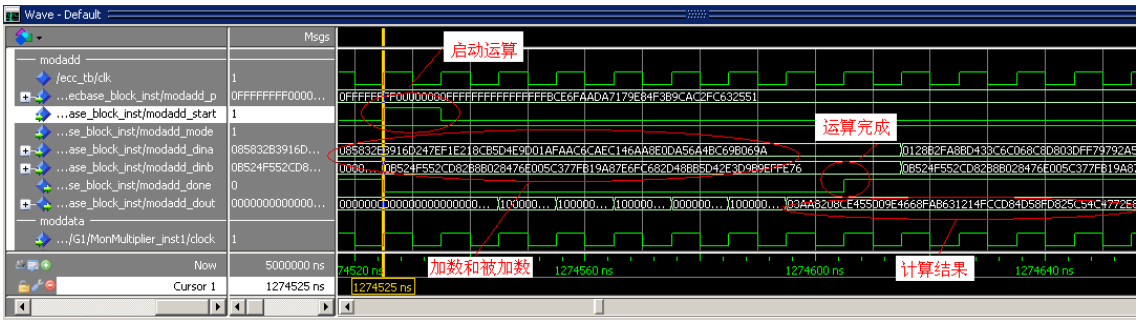


图 5-1 模加运算

验证模加模块时，输入的 modadd_a 和 modadd_b 需小于 modadd_p 且大于等于零。设置运算模式 modadd_mode 信号为 1，modadd_start 信号为高脉冲时，模块开始运算，结束时 modadd_done 信号置高，结果输出到端口 modadd_dout。图 5-1 为模加运算的仿真图，模块运算一次需要 7 个时钟周期。

b）模减运算模块

输入被减数 modadd_a，减数 modadd_b 和模数 modadd_p：

modadd_a=6CB28D99385C175C94F94E934817663FC176D925DD72B727260D BAAE1FB2F96F，

modadd_b=2C26F6D58F88CCBD2A461F3CE0EDBABDAB659667CB07D2346 862327759920893，

modadd_p=FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B 9CAC2FC632551。

仿真输出：

modadd_dout=408B96C3A8D34A9F6AB32F566729AB82161142BE126AE4F2B DAB8836C620F0DC。

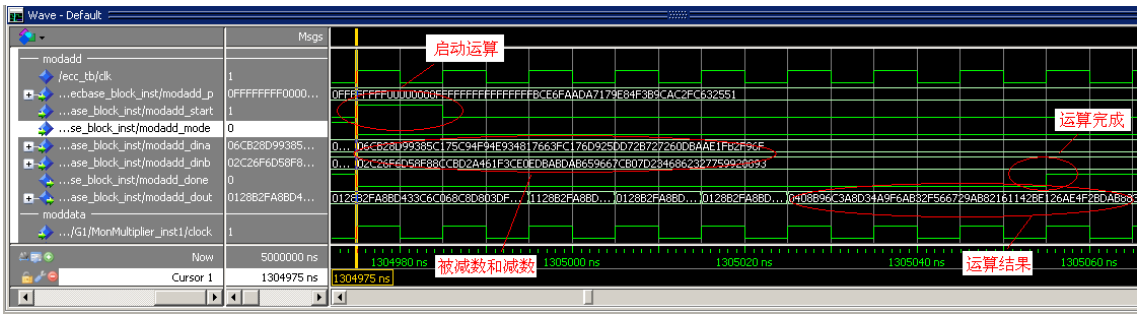


图 5-2 模减运算

验证模减模块时，输入的 modadd_a 与 modadd_b 需小于 modadd_p 且大于等于零。设置运算模式 modadd_mode 信号为 0，modadd_start 信号为高脉冲时，模块开始运算，结束时 modadd_done 信号置高，结果输出到端口 modadd_dout。图 5-2

为模减运算的仿真图，模块运算一次需要 7 个时钟周期。

c）快速模乘模块

输入被乘数 mul_dina，乘数 mul_dinb 和模数 num_p：

mul_dina=89220559D42FC94983FE594C3C80FCC02016B1ADD9E65F208C4C F37FFCF12AD5，

mul_dinb=AAF8B26050E61DF8B690D73DA83568AA6730CCD3ABFB6918835 117E6A2FC8C4C，

num_p=FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFF FFFFFFF。

仿真输出：

mul_dout=486DB35A380E68C4597E43E9585CD45CB15D2590733B3CC2E3F4 F7A570575744。

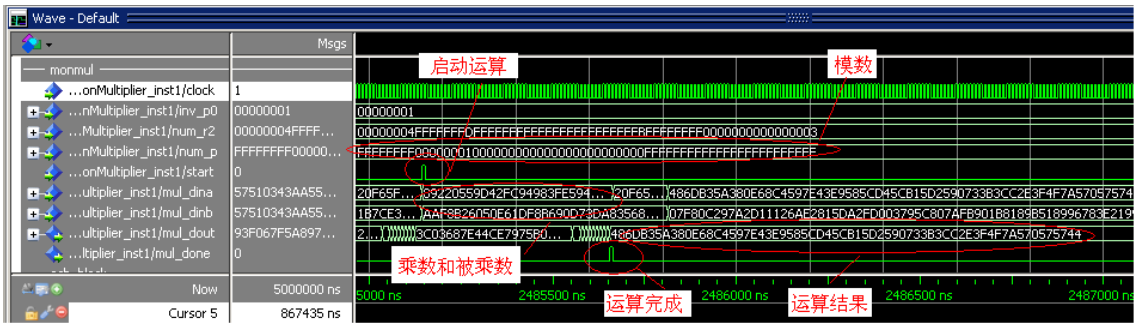


图 5-3 快速模乘运算

验证快速模乘模块时，需预先计算得到 *num_r*2 和 *inv_p*0，计算公式为：

$$num\_r2 = (2^{256})^2 \bmod p,$$

$$inv\_p0 = (-p)^{-1} \bmod 2^{32},$$

输入两个 256 比特的数据 mul_dina 和 mul_dinb，mul_dina 与 mul_dinb 需要先转换到蒙哥马利域。start 为高脉冲时启动运算，运算结束后，mul_done 信号置高，结果输出到端口 mul_dout 上，计算的结果在蒙哥马利域内。图 5-3 为快速模乘运算的仿真图。蒙哥马利域与普通域间的转换会带来额外开销，但实际运用中，只需要在点乘运算前将数据从普通域转换到蒙哥马利域，点乘计算完成后，再从蒙哥马利域转换到普通域。两次域的转换对点乘运算的时间开销而言完全可忽略不计。

d）模逆运算

输入待求逆的数据 Modinv_ain 和模数 modinv_num：

Modinv_ain=128B2FA8BD433C6C068C8D803DFF79792A519A55171B1B650C 23661D15897264，

modadd_p=FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551。

仿真结果：

Modinv_out=69E8765BE6FA15BC91A31D14E9A53675654E7743BD74149D3735259D48E2A723。

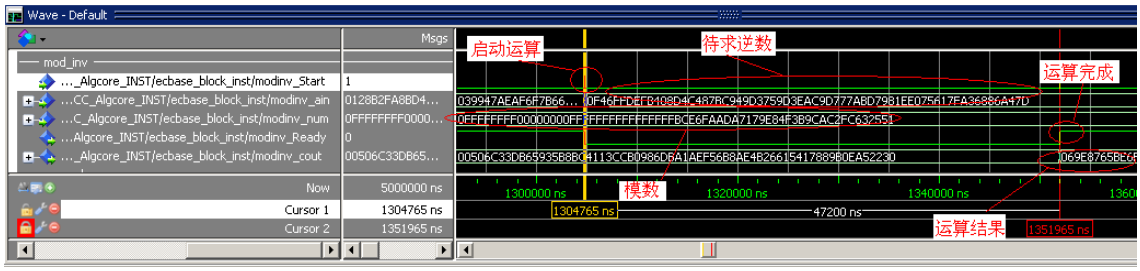


图 5-4 模逆运算

验证模逆运算时，输入计算参数，modinv_start 信号为高脉冲时启动模块运算，modinv_ready 信号为高时运算结束，运算结果输出到端口 modinv_dout。图 5-4 为模逆运算的仿真图，该模块运算一次需要 4720 个时钟周期。

e）点乘运算

输入点的横坐标 Point1_x，纵坐标 Point1_y 以及整数 k：

Point1_x=54EB18091B3FF2C84E35D9ABE9A154AD27877FD6EDFE07A670E7784F4755A288，

Point1_y=3DE3E0D8D8B01858AA0BC28024496281BF82CA40A6FC421B670E717BA844D5FC，

k=4C62EEFD6ECFC2B95B92FD6C3D9575148AFA17425546D49018E5388D49DD7B4F。

仿真结果：

Presult_x=48959EF6C48BC34097EE88F8163B3B5FCD2CB6729CB25D3F732AADAF3AEFF6B6，

Presult_y=48D545F41EEF309DE8C07A6718E796278059A6A89DECB5C827A0146A83B04911。

验证点乘运算时，先指定椭圆曲线的参数，再输入点的坐标和整数，点的坐标值和整数为普通域下的数据。将运算模式 point_mode 信号设置为 0，point_start 信号为高脉冲时启动模块运算，point_ready 信号为高时运算结束，运算的结果输出到端口 presult_x、presult_y，且已转换到了普通域。图 5-5 为点乘运算的仿真图，点乘运算一次需要 226029 个时钟周期。

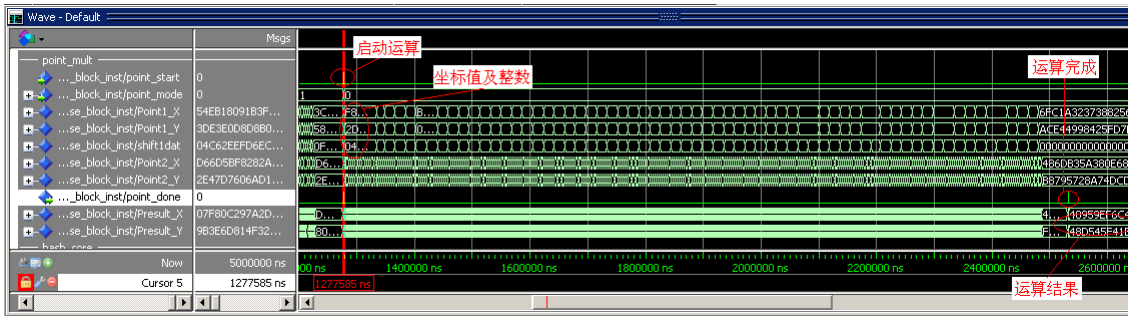


图 5-5 点乘运算

f）签名运算

签名运算需要输入的参数有随机数 k，私钥 d 以及待签名的数据 M：

k=6CB28D99385C175C94F94E934817663FC176D925DD72B727260DBAAE1FB2F96F，

d=128B2FA8BD433C6C068C8D803DFF79792A519A55171B1B650C23661D15897263，

M=B524F552CD82B8B028476E005C377FB19A87E6FC682D48BB5D42E3D9B9EFFE76。

签名结果输出：

r=85832B3916D247EF1E218CB5D4E9D01AFAAC6CAEC146AA8E0DA56A4BC69B069A，

s=C47E9A646EF89BAA740E24BF759CB8F34F8D4DD863D2368CCC4A2BB15BBB0EF4。

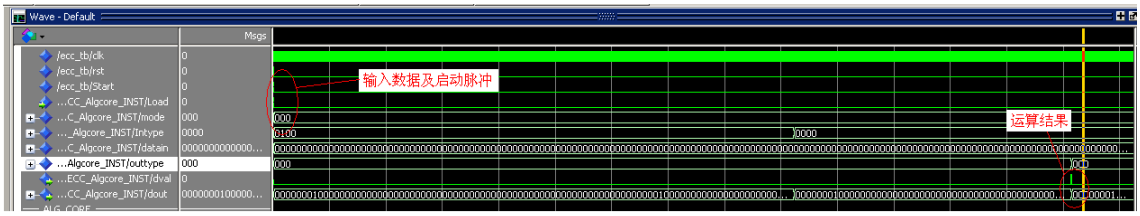


图 5-6 签名运算

仿真签名运算时，输入运算数据后，启动运算脉冲 start，运算完成后数据通过总线 dout 输出。签名运算的仿真图如图 5-6 所示，完成一次签名运算的需要 240905 个时钟周期。

## 5.2 FPGA 的板级验证

### 5.2.1 验证环境

本文选择 xilinx 公司 virtex5 系列的 FPGA 来验证本设计，virtex5 采用

了 65 nm 铜工艺技术，是定制 ASIC 技术的可编程替代方案。Virtex-5 FPGA 为逻辑、DSP、软硬微处理器和连接功能提供了好的解决方案，可满足高性能逻辑设计人员、高性能 DSP 设计人员和高性能嵌入式系统设计人员的需求。本文使用的 FPGA 型号为 xc5vlx110t，其包含的资源如表 5-1 所示。

表 5-1 FPGA 资源

| 逻辑块 CLB | | DS P48 | RAM 块 | | CMT | PCIE | 以太网 mac | I/Obank | 最大用户 I/O |
|---|---|---|---|---|---|---|---|---|---|
| Slice | 最大分布式 RAM(kb) | | 18Kb | 36Kb | | | | | |
| 17280 | 1120 | 64 | 296 | 148 | 6 | 1 | 4 | 20 | 680 |

图 5-7 Virtex5 开发板

本文验证所用的开发板如图 5-7 所示，开发板提供了网口、USB 口、VGA 接口、PCIE、串口等多种外设接口。本文通过 xilinx 下载线将 FPGA 的配置文件下载到 FPGA 中，然后通过串口与计算机连接进行数据交互，进而进行数据的验证。

## 5.2.2 验证结果

### 5.2.2.1 功能验证

a）布局与布线

本文选用的开发工具版本为 ISE14.6，在 xc5vlx110t 芯片中对本设计编译后得到的整体布局布线图如图 5-8 所示。

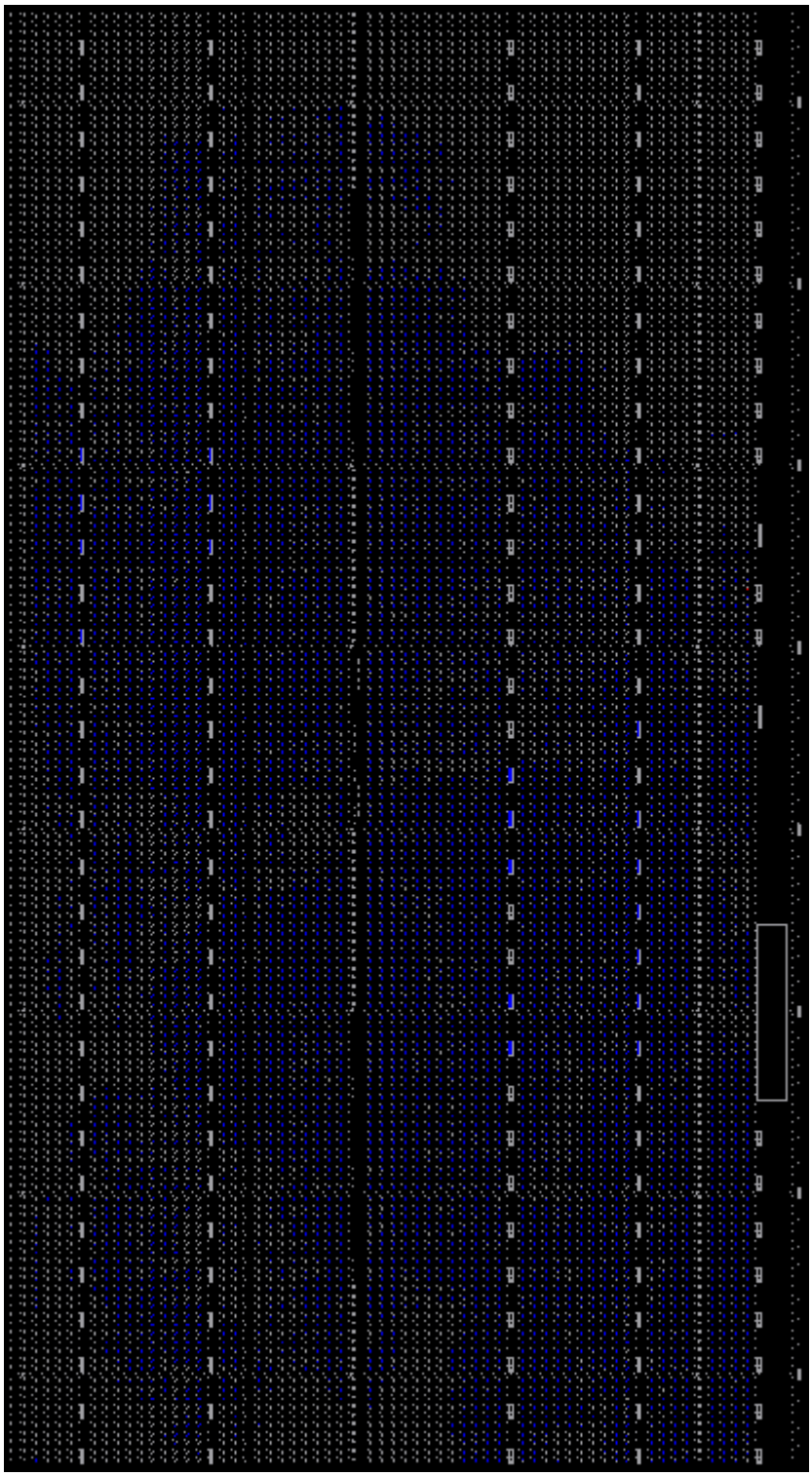

图 5-8 布局布线图

b）资源使用情况

在 xc5vfx110t 芯片中编译后的资源占用情况如表 5-2 所示。编译的最高时钟频率为 113Mhz。

表 5-2 资源使用情况

| Slice Logic Utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of Slice Registers | 25,675 | 69,120 | 37% |
| Number used as Flip Flops | 25,667 | | |
| Number used as Latch-thrus | 8 | | |
| Number of Slice LUTs | 25,021 | 69,120 | 36% |
| Number used as logic | 24,878 | 69,120 | 35% |
| Number using O6 output only | 23,413 | | |
| Number using O5 output only | 209 | | |
| Number using O5 and O6 | 1,256 | | |
| Number used as Memory | 130 | 17,920 | 1% |
| Number used as Shift Register | 130 | | |
| Number using O6 output only | 130 | | |
| Number used as exclusive route-thru | 13 | | |
| Number of route-thrus | 888 | | |
| Number using O6 output only | 221 | | |
| Number using O5 output only | 666 | | |
| Number using O5 and O6 | 1 | | |
| Number of occupied Slices | 9,419 | 17,280 | 54% |
| Number of LUT Flip Flop pairs used | 34,810 | | |
| Number with an unused Flip Flop | 9,135 | 34,810 | 26% |
| Number with an unused LUT | 9,789 | 34,810 | 28% |
| Number of fully used LUT-FF pairs | 15,886 | 34,810 | 45% |
| Number of unique control sets | 222 | | |
| Number of slice register sites lost to control set restrictions | 223 | 69,120 | 1% |
| Number of BlockRAM/FIFO | 19 | 148 | 12% |
| Number using BlockRAM only | 19 | | |
| Number of 36k BlockRAM used | 18 | | |
| Number of 18k BlockRAM used | 1 | | |
| Total Memory used (KB) | 666 | 5,328 | 12% |
| Number of DSP48Es | 32 | 64 | 50% |

c）板级验证结果

将 virtex5 的开发板通过串口与计算机连接，将签名的结果通过串口调试工具在计算机中显示出来，签名的结果 r，s 如 5-9 所示。

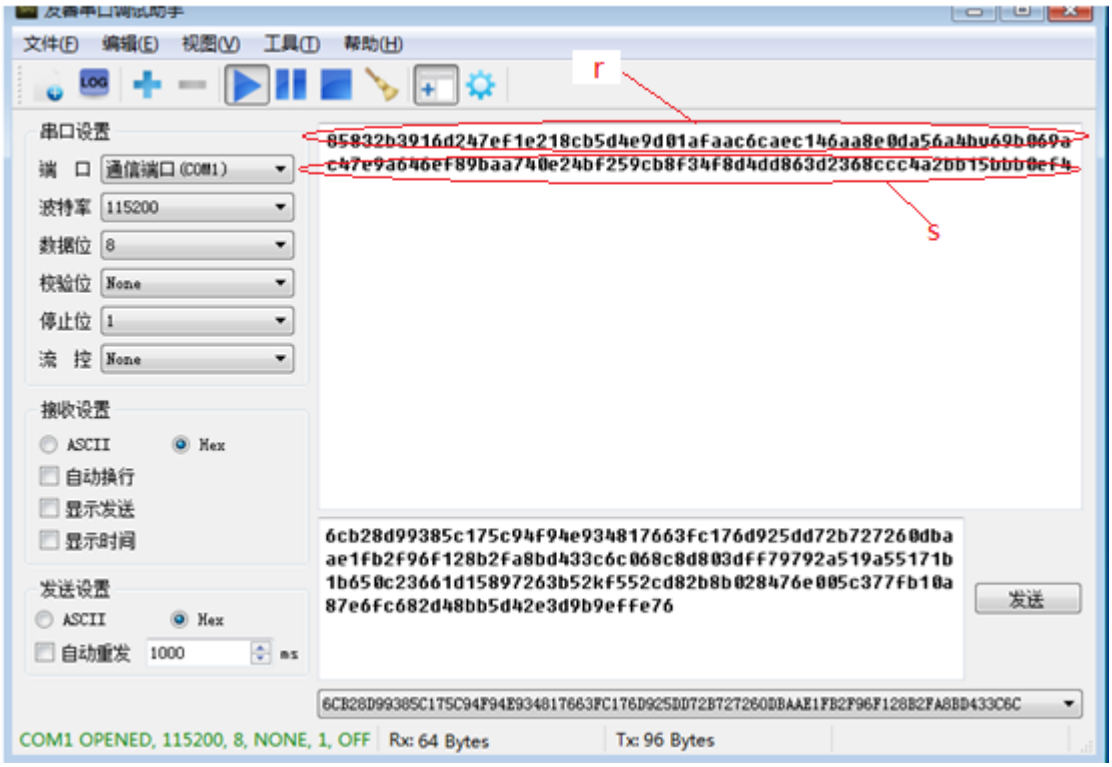


图 5-9 签名运算结果

#### 5.2.2.2 性能验证

性能测试用于测试签名验证、公钥加密、产生密钥对三个算法各自的计算性能，测试方法为统计一分钟内每种算法运算的次数。下面分别介绍三个算法五种运算的测试方法及测试结果。

1) 签名运算

签名运算需要输入的参数有签名的消息 $M$，随机数 $k$，私钥 $d$。测试性能时，消息 $M$ 和私钥 $d$ 为样本数据且固定不变，随机数 $k$ 则每次随机产生。启动运算后一分钟统计的签名次数平均为 25327 次，所以签名运算的性能约为 422 次/秒。

2) 验证运算

验证运算需要输入的参数有签名的消息 $M$，签名的结果 $R$,$S$，公钥 $P$。测试性能时，消息 $M$，签名的结果 $R$、$S$，公钥 $P$ 均使用样本数据且固定不变。启动运算后一分钟统计的验证次数为 12961 次，所以验证运算的性能约为 216 次/秒。

3) 加密运算

加密运算需要输入的参数有加密的消息 $M$，加密的随机数 $k$，加密的公钥 $P$，加密消息的长度 $klen$。性能测试时，除随机数 $k$ 每次随机生成外，其他参数都使用样本数据且固定不变。启动运算后一分钟统计的加密次数平均为 13269 次，所以加密运算的性能约为 221 次/秒。

4) 解密运算

解密运算需要输入的参数有加密运算的结果 *C*，解密的私钥 *d*，解密的消息长度 *klen*。性能测试时，以上参数都使用样本数据且固定不变。启动运算后一分钟统计的解密次数为 26521 次，所以解密运算的性能约为 442 次/秒。

5) 密钥对产生算法

产生密钥对运算只需要输入随机数 *k*，启动运算后一分钟统计的加密次数平均为 26602 次，所以加密运算的性能约为 443 次/秒。

# 第六章 结 论

## 6.1 本文的主要贡献

随着信息技术和网络技术的快速发展，网络安全问题逐渐进入公众视野，我们国家也三番五次的强调了网络安全的重要性。网络安全是为了保护网络中信息的安全存储和传递，而保护信息的安全离不开密码技术。椭圆曲线密码体制作为一种公钥密码体制，自问世以来因其具有短密钥、高安全性的特点而备受关注。椭圆曲线密码的实现分为硬件实现和软件实现，硬件实现比软件实现具有速度快、性能好的优势。因此，本文基于 FPGA 的椭圆曲线密码算法的研究具有重要的现实意义。

本文在深入学习和研究密码学理论，尤其是椭圆曲线密码理论和技术的基础上，完成了椭圆曲线密码系统的硬件设计及其 FPGA 实现。现将论文完成的工作及技术创新总结如下：

a）在深入研究椭圆曲线密码理论的基础上，按照层次结构对 ECC 系统的运算进行了划分，并在此基础上设计了椭圆曲线密码算法的 FPGA 实现方案。此外，本文把椭圆曲线中的共性运算提炼出来，封装成专门的大数运算库，这样既减少了 FPGA 的资源开销，也提高了后续研究椭圆曲线密码的效率。

b）按照模块化的设计思想对椭圆曲线中的运算进行了设计实现，使得模块的功能相对独立，便于后期的调试和升级维护。

c）设计和实现过程中充分考虑资源和性能两方面的因素。首先从顶层设计开始尽可能的采用分时复用的方法来减少硬件资源消耗；其次将椭圆曲线中不影响性能或者对性能影响较小的运算提取出来，采用一种运算速度慢但资源消耗少的方法来实现；最后，对影响性能的关键模块采用了快速实现方法尽可能的减少其运算时间。

d）对椭圆曲线中点乘运算进行了针对性优化，根据计算特点将点运算封装在一个模块内，使其尽可能的复用中间临时寄存器，减少资源消耗。最后，以较少的额外资源消耗为代价，提升了 ECC 系统的整体性能。

e）用硬件描述语言对椭圆曲线所有的运算模块进行了编程实现，封装了适用于椭圆曲线计算的大数库，在此基础上实现了签名算法、公钥加密算法和密钥对生成算法，并在 ModelSim SE 10.1c 中进行了仿真验证。最后，通过 Xilinx 公司的 ISE14.6 开发工具对代码进行综合编译，并在该公司的 virtex5 FPGA 进行了应用测试，验证了设计的正确性。

## 6.2 下一步工作的展望

本文主要对椭圆曲线密码系统的 FPGA 实现做了初步的研究，虽然用硬件描述语言编程实现了椭圆曲线密码算法，并在 FPGA 芯片中进行了测试验证，但仍然还有值得改进和进一步研究的地方，主要有以下几个点：

a）本文在设计实现时，只考虑到减少硬件资源的消耗，对一些计算模块的通用性考虑较少，缺乏灵活性。如本文的运算模块仅支持 256bit 位宽的数据计算，计算其他位宽的数据则需要修改程序模块。

b）本文设计实现的程序模块，在中高端的 FPGA 中能轻松的编译，并获得较高的编译频率，但在低端 FGPA 中编译频率则较低。所以程序模块的进一步优化和提高是今后努力的方向。

c）本文主要对 $GF(p)$域上的椭圆曲线密码算法进行了研究，对 $GF(2^m)$上的椭圆曲线密码算法没作太多研究。由于 $GF(2^m)$域和 $GF(p)$域的计算有差异，本文实现的大数库模块不能直接运用在 $GF(2^m)$域中，所以今后有必要研究适用于 $GF(2^m)$域的大数运算库以及同时支持两种有限域的大数库模块。

# 致 谢

首先，我要衷心感谢电子科技大学阎波教授和中国电子科技集团公司第三十研究所刘振钧高级工程师(研究员级)，在攻读硕士期间得到了他们的悉心指导和关怀，论文期间，从论文的选题、开题、资料阅读调研、方案设计直到论文的修改和完成,都得到了导师们的指点和帮助,使我的论文能够有序的、顺利的进行。在攻读工程硕士期间，各个科目的任课老师尽职尽责，他们认真负责的精神和严谨的工作作风给我留下了深刻的印象，使我受益良多，指引着我不断进步。在此，向他们表示最衷心的感谢。

在论文完成过程中，还得到了诸多同事和同学的帮助和支持，能与他们一起工作和学习是一件非常高兴和幸运的事情，他们各自的研究方向和专长使我受益匪浅。在此特向他们表示我最诚挚的谢意。

感谢父母在读研期间对我的帮助和关心，感谢他们生活上对我的照顾，为我在读研期间提供了良好的环境。同时也感谢妻子对我的理解和支持，三年来占用了太多本该陪伴她和孩子的时间。

最后，衷心感谢对本文进行评阅并提出宝贵意见和建议的各位专家和老师。

# 参考文献

[1] 周敏.基于椭圆曲线密码体制的数字签名研究[D].青海:青海师范大学,2013,1-1.

[2] 姜量超.椭圆曲线密码算法及应用[D].沈阳:东北大学,2006,1-2.

[3] 李海平,凌广明,裴宸平.基于椭圆曲线数字签名系统的设计与实现[J].计算机时代,2015,05:44-49.

[4] 英海燕，王友波，韩月秋.基于 FPGA 椭圆曲线密码体制的研究[J].计算机工程与设计,2006,27(5):752-755.

[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(70):1546-1551.

[6] Koblitz N. Elliptic curve cryptosystems[J].Mathematics of Computation, 1987,48:203-209.

[7] Miller V. Use of elliptic curves in cryptography[J]. Advances in CRYPTO'85,Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986:417-426.

[8] 李扬扬.椭圆曲线密码体制及其在电子商务中的应用研究[D].淮南:安徽理工大学,2015,3-5.

[9] 李德庆.椭圆曲线密码体制的研究与实现[D].西安:西安电子科技大学,2008,3.

[10] Alfred Menezes 等著.椭圆曲线密码学导论[M].张焕国等译.北京:电子工业出版社,2005.

[11] R.Lidl and H.Niederreiter. Introduction to finite fields and their application[M]. Cambridege University Press: New York, 1994.

[12] 李艳梅.椭圆曲线标量乘算法的快速实现[D].江苏:扬州大学.2017,5-5.

[13] 陈梦婷.椭圆曲线密码体制标量乘快速算法研究[D].成都:西南交通大学.2016,9-9.

[14] 袁勇.基于椭圆曲线的签名方案设计及快速实现[D].西安:西安电子科技大学,2015.6-6.

[15] 胡予濮,张玉清,肖国镇.对称密码学[M].北京:机械工业出版社，2002.

[16] 仲先海.并行可配置 ECC 协处理器关键技术研究[D].郑州:解放军信息工程大学,2008,9-10.

[17] 孙万忠.椭圆曲线密码快速硬件实现算法研究与设计[D].郑州:解放军信息工程大学,2009,11-12.

[18] N.Takagi. Modular Inversion Hardware with a Redundant Binary Representation[J]. IEICE Trans on Ingormation and Syatem, E76-D(8):863-869,Aug.1993.

[19] US-ANSI. ANSI X9.62-2005: Public Key Cryptography for Financial Service Industry: The Elliptic Curve Didital Signature Algorithm[S]. Nov 16, 2005.

[20] US-ANSI. ANSI X9.63-2011: Public Key Cryptography for Financial Service Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography[S]. 2011.

[21] US-IEEE. IEEE P1363-2000: Standard Specifications for Public-Key Cryptography[S], Jan 30,

2000.

[22] 李曼,徐和根.基于 FPGA 的椭圆曲线密码(ECC)算法硬件设计[J].机电一体化.2013,01:85-88.

[23] 刘帅.椭圆曲线密码算法的硬件加速研究[D].济南:山东大学.2015,9-10.

[24] 胡诗玮,徐和根.有限域 GF($2^{256}$)上椭圆曲线密码算法的硬件设计[J].机电一体化,2015,02:71-74.

[25] 蔡振国.基于GF($2^n$)的椭圆曲线加密算法在FPGA上的设计及实现[D]. 郑州:解放军信息工程大学.2006,11-12.

[26] 陈文宇.基于椭圆曲线加密系统的 FPGA 实现[D].上海:中国科学院上海系统与信息技术研究所.2002,47-51.

[27] 许伟.基于 FPGA 的 RSA 密码算法的模幂模乘的快速实现[D].济南:山东大学.2010,22-24.

[28] 何德彪,陈建华,胡进. 高速椭圆曲线密码协处理器的设计与实现[J].华南理工大学学报(自然科学版),2009,38(5):90-94.

[29] 王健.椭圆曲线加密体制的双有限域算法及其硬件实现[D].北京:北京大学.2008,74-75.

[30] 王凡.素域上椭圆曲线密码算法的硬件设计[D].郑州:东南大学.2016,53-54.

[31] 白羽.椭圆曲线密码体制的研究及其应用[D].成都:西南交通大学.2011,23.

# 在学期间取得的与学位论文相关的研究成果

[1] 韩炼冰,段俊红,王松. 基于 FPGA 的 Edwards 曲线标量乘法实现方法[J].通信技术,2015,48(10):1179-1182.

[2] 韩炼冰,段俊红,王松. FPGA 局部重配置技术的实现及应用[J].通信技术,2016,49(12):1728-1732.

[3] 韩炼冰,段俊红,房利国.一种基于 FPGA 的随机数检验实现方法[J].通信技术,2017,50(11):2584-2588.