分类号：TN402　　密　级：公开
论文编号：200301254

贵 州 大 学

2006 届硕士研究生学位论文

对称算法空间的可重组逻辑研究与 SOC 设计

学科专业：微电子学与固体电子学
研究方向：集成电路设计
导　　师：傅兴华教授
研 究 生：苏海亮

中国 · 贵州 · 贵阳
2006 年 5 月

摘 要

随着信息技术的发展，网络应用、电子政务和电子商务的普及，信息的安全性越来越受人关注。目前密码算法是保证信息安全比较通用的手段。实现密码算法通常有两种方式：一是软件实现，虽然它实现起来灵活、快捷，但它速度比较慢；二就是专用密码芯片，密码芯片速度比较快，但它长期固定一种密码算法成为安全隐患。利用可重组逻辑设计的思想实现的可重组密码算法芯片，不仅使得数据加密保持比较高的速度，还能通过编码更换芯片中运行的密码算法，有效地弥补以上两种实现方式的不足。

目前，SOC(system-on-chip)的集成度已经达到10亿个晶体管的规模，其功能也越来越复杂。大多数SOC设计团队都没时间和知识从头开始开发SOC。在SOC设计中，IP的动态复用和交换已成为关键所在。

算法的可重组性和IP模块的可重组性是密切相关的。本设计首先就密码对称算法进行分析，提炼出重组元素，并以可重组逻辑设计的思想设计重组元素，提供标准的数据接口，使其能在SOC设计中动态复用。然后用Modelsim工具对设计的IP模块进行了仿真验证，用Xilinx ISE工具进行了综合，并对其延时进行了估计。在此基础上，进行了对称算法空间的可重组逻辑分析与设计。采取完全间接相连的体系结构。完成了密码算法3DES的设计，并通过了仿真，为最终完成可重组密码算法的SOC设计打下了基础。

关键词：可重组 IP模块 算法空间 体系结构 SOC设计

Abstract

Along with the development of information technology, the electronic governmental affairs, electronic commerce and the distant application of the network in general have been being used widely. the safety of the information exchanged by the network is payed more and more attention to by users. Currently a cipher algorithm is in common used to protect the information transfered. Usually there are two kinds of methods to implement cipher algorithms. The first is realized by software although its advantage is time-saving and easy to implement, its operation speed is slow. The second is realized in hardware. The speed of the cipher chip is much more higher than that of a software way. but its algorithm is fixed in the chip, this provides opportunities for the hackers' attack. In this paper, we make use the concept of reconfigurable design to the end of the reorganization in a cipher chip. A reconfigurable cipher chip has higher speed in data encryption. and by changing the control code, it may realize different algorithms in one chip. This means that a reconfigurable cipher chip is superior to the above two ways.

Up to date, we can cram as many as one thousand million transistor into one integrated circuit chip. The function of a SOC (System-On-Chip) chip can be very complex. Most SOC design teams have no time and knowledge to design SOC from the very beginning. The reconfigurable efficiency of algorithms is closely-related with that of IP core modules. In SOC design, the dynamic and repeatable use of IP cores has become the key to success of a design.

In this design, we analyse symmetric algorithms, and find reconfigurable elements in them. Then we map these elements into IP cores. We verify these IP cores' function with tool of Modelsim, and synthesize in Xilinx ISE, estimate their delay. Based on these work, We complete the frontend design for 3DES cipher algorithm, including simulation and synthesis. The whole design for reconfigurable cipher chip will be completed in later work.

Key words: reconfigurable; IP cores; space of symmetric algorithms; architecture; SOC design;

第一章 前 言

1.1 可重构系统简介

早在1963年Estrin就在文献[1]中提出了可重构计算RC（Reconfigurable computing）的概念，但是直到20世纪80年代中期RC才随着FPGA技术的成熟而重新成为研究热点[2]。可重构技术的出现使过去传统意义上硬件和软件的界限变得模糊，让硬件系统软件化。可重构技术的本质是利用可编程器件的可多次重复配置逻辑状态的特性，在运行时根据需要动态改变系统的电路结构，从而使系统兼具灵活、简捷、硬件资源可复用、易于升级等多种优良性能。可重构系统(Reconfigurable System)在高速数字滤波器、图像压缩、硬件演化计算、定制计算(Custom Computing)、嵌入式系统等方面，都有着广泛的应用前景。

可重构系统的历史很短，其标准形式还没有形成。现有的各种重构系统，无论是从重构单元的粒度、重构方式、系统结构等等都有极大的不同。

1、按重构的粒度和方式，可重构系统可粗略地分为两种。一种重构单元的粒度较大，为模块级的重构，即重构时改变某一个或若干个子模块的结构。此时不仅电路逻辑改变，连线资源也重新分配。重构所需的电路配置信息事先由编译软件生成。通常重构时系统可能需要暂停工作，待重构完成后再继续。这种重构系统设计简单，但灵活性不足，且有时不能完全发挥出硬件运算的效率。它较适合应用在嵌入式系统中。另一种重构单元的粒度较小，仅为元件级的重构，即重构时仅改变若干元件的逻辑功能。通常情况下重构时连线资源的分配状况不作修改。重构所需的电路配置信息在系统运行过程中动态产生。重构时系统可以不暂停，边重构边工作。这种重构系统设计复杂，但灵活性大，能充分发挥出硬件运算的效率，较适合被应用在高速数字滤波器、演化计算、定制计算等方面。

2、系统结构按可重构系统的结构，常见的有不规则型、流水线型、和处理器集成型等等。（1）不规则型的可重构系统中，部件之间没有统一、严格的组织形式，结构比较随意。随客观要求不同，系统可采用不同的结构，性能也各不相同。较早期的可重构系统一般都采用这样的形式。（2）流水线型，顾名思义，就是系统的主要部件以流水线的形式协同工作。流水线的每一级或其中若干级是可重构的部件，能根据需要改变结构，从而改变流水线的功能。这主要被应用在先进先出的数据流的处理上，比如数字滤波、压缩解压、加密解密等等。（3）处理器集成型，就是把可重构部件集成到微处理器中，以扩展微处理器的功能。在早期的这类系统中，

可重构部件充当处理器中运算单元的角色，也就是用可重构部件作为算术处理单元(ALU)的扩充。但这样的扩展形式产生的处理器性能的提升并不大。另一种形式是将可重构部件独立于处理器的原流水线之外，使其既能在处理器流水线之外进行独立的运算，又能通过扩展的特殊指令实现与处理器内核的通讯。这样的改进使处理器集成型的可重构系统性能大大提高。

当然，这些只是粗略的分类。而实际上可重构系统的形式并没有完全定型，各类型间的分界是非常模糊的，甚至是交叉重合的。此外，可以预见随着可重构技术的发展，还会有新的系统结构出现[3][22]。

1.2 课题研究背景

在信息技术飞速发展的今天，信息技术已经渗透到政治、经济、军事和社会生活的各个方面，为了解决信息的安全性问题，人们采取了许多措施，其中，对信息数据进行加密就是一个行之有效的措施。传统的数据加密方法有两种，一种是软件加密方法；另外一种是硬件加密方法，两种数据加密方法各有特点，软件加密虽然灵活，但加密速度慢，并且容易被人改动；硬件加密虽然速度快，但算法单一，且固定不变，长此以往，总有破解的一天。为了解决上述两种加密方法的缺点，我们运用可重组逻辑技术设计出可以实现多种密码算法的密码芯片，既能灵活、快速的实现多种不同的密码算法，又能避免上述安全上的隐患。

对称算法简单，且系统开销小，适合加密大量数据，它速度极快并且对选择密文攻击不敏感，具有很好的机密性[4]。本课题就是利用可重组逻辑技术实现对称算法空间中的算法重组元素进行可复用的设计，并以此基础对可重组算法逻辑芯片进行分析和设计。

美国已于 1998 年推出了可编程加密芯片的产品[5]，我国一些公司和高校也开始了对可编程加密芯片的研究和开发。

1.3 课题依据和意义

任何一个密码算法都是由一系列的基本操作按照一定顺序连接而成的，通过对大量的密码算法进行分析和研究，我们发现密码算法具有一个显著的特征：很多不同的密码算法具有相同或相似的基本操作成分，或者说同一基本操作成分在不同算法中出现的频度很高。那么这些基本操作成分所对应的硬件资源就可以被多种不同的密码算法所共用，因此我们就能够以较少的电路规模构造一套逻辑电路来实现多种算法。这就是本课题的设计依据。

表 1.1 列出了我们对 3DES、IDEA、AES 侯选算法等 34 种典型的分组密码算法和 13 种典型

的序列密码算法的统计结果[4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 29, 35]。

表 1.1 对称算法基本操作成分使用频度统计

算法类别	基本操作成分	使用频度
分组密码算法	异或运算	100%
	S 盒变换	50%
	移位运算	58.82%
	置换运算	29.41%
	模加运算	44.12%
	模减运算	8.82%
	模加逆运算	2.94%
	模乘运算	26.47%
	模乘逆运算	2.94%
	逻辑非运算	11.76%
	逻辑与运算	11.76%
	逻辑或运算	11.76%
	指数运算	8.82%
	对数运算	5.88%
序列密码算法	反馈移位寄存器	100.00%

从上表我们看出，在对称算法中，异或单元、S 盒变换、移位单元、置换单元等模块都有很高的使用频率，我们利用可重构的技术对这些模块进行设计，使其实现尽可能多的规格和变换，那么一个模块就可以被多种算法重复利用，这样就可以有效的节省可重组算法芯片的规模。

可重组算法芯片具有以下明显的优点：

（1）可重组算法芯片具有设计和生产上的安全性：密码算法的设计和芯片的设计与生产过程是分离的，芯片在投入使用之前是不含密码算法的白裸片，只有在使用时才由用户将设计好的密码算法装载到芯片上，因此在芯片的设计和生产过程中不会泄露任何密码算法的信息。

（2）可重组算法芯片具有兼容性：可重组算法芯片可由用户编程实现多种不同的密码算法，从这个意义上说，可重组密码芯片对密码算法具有兼容性。该兼容性意味着一块可重组密码芯片可以代替多块针对特定算法的密码芯片，从而大大降低了密码芯片的开发和生产成本。

（3）可重组算法芯片具有扩展性：用户可以根据自己的需求，利用可重组密码芯片中的已

扩展性能够有效地支持密码算法的更新，适应密码算法的发展需求。另外，可重组密码芯片还为密码专家研究设计新的密码算法提供了一个实验平台。

（4）可重组算法芯片可以提高密码系统的安全性：用户可以很方便地随时更换所使用的密码算法，缩短同一密码算法的使用周期，大大降低了算法泄露的可能性，增加了密码分析（攻击）的难度，从而提高了密码系统的安全性。

（5）可重组算法芯片的加/解密速度比软件的加/解密速度快得多，接近专用密码芯片，而其灵活性和安全性则是专用密码芯片所无法比拟的。

由于可重组算法芯片具有上述众多的优点，因此在信息安全领域内具有广泛的应用前景。可重组算法逻辑是可重组算法芯片的关键组成部分，因此对可重组算法逻辑的设计原理和设计方法进行研究具有十分重要的意义[5]。

第二章 可重组算法逻辑的理论基础

2.1 可重组算法逻辑设计思想

我们在前面设计依据中讲过，很多不同的密码算法具有相同或相似的基本操作成分。如果要在一套系统中多种不同的密码算法，我们应该在算法逻辑电路中设置一些可被不同密码算法重复使用的模块，并且重复使用的模块个数和各个重用模块的重用次数越多，整个系统逻辑电路的规模就越小，平均每种算法分担的规模就越小或者说硬件资源的利用率就越高。重用模块在不同的算法中会有不同的功能，那么为了提高重用模块的重复使用次数，我们要使得重用模块实现尽可能多的数据处理功能，从而尽可能地扩大该重用部件的使用范围。例如，很多密码算法都使用了S盒代替变换，但不同的算法要求不同规格的S盒，并且实现的代替函数也不同，为了使同一个S盒能够被尽可能多的算法所使用，我们应该使该S盒能够实现其输入变量的任意的代替函数，并且兼容多种规格，这样不仅使S盒的适应范围达到最大，而且为以后开发新的密码算法奠定了基础。在下面重组元素设计章节将会对重用模块进行详细介绍。

在一个算法系统中，要实现多种不同的密码算法，就是通过把不同的重用模块组合起来实现，那么我们必须在这些重用模块之间建立算法所要求的数据传输路径。因为不同的算法所要求的数据传输路径不同，所以要想实现多种不同的算法，就必须使重用模块之间的连接关系可变。而要实现重用模块之间的连接关系可变，我们就必须在重用模块之间的连接网络中设置一些可控接点，通过对可控接点的控制来实现不同重用模块之间的连接关系。

下面举个简单的例子，用文字和图形来说明模块和数据传输路径的可重组。例如，设C_1、C_2、C_3是三个密码算法，A、B、D、S是算法逻辑中的4个模块，并假设S是单输入的重用模块，设算法C_1需要将A的结果传送到S，即A=>S，设算法C_2需要将B的结果传送到S，即B=>S，设算法C_2需要将D的结果传送到S，即D=>S，因为在一个确定的时刻，模块S只能接受一个输入，，所以我们必须在模块S的输入端设置一个可控接点用于选择A、B或D的输出作为S的输入，通过对该可控接点进行控制，我们就可以在不同的时刻分别实现A=>S、B=>S或D=>S三条不同的数据传输路径。同时模块S也只能满足三种算法中的一种算法所需求的数据处理功能，那么模块S本身也要设置可控接点，用来选择模块S将要实现的功能。如图2.1所示：

在图2.1所示的电路中，MUX表示多路选择器，C_1、C_2、C_3表示3个密码算法，A、B、D是4个模块，S是重组模块。我们在上述电路中设置了2个可控接点，其控制信号分别记为CTRL1

和 CTRL2。通过对 CTRL1 和 CTRL2 赋以不同的值，就可以改变上述电路的逻辑功能，实现数据的不同走向,和重组模块 S 的不同数据处理功能。以此来实现 C_1、C_2、C_3这 3 个不同的密码算法。

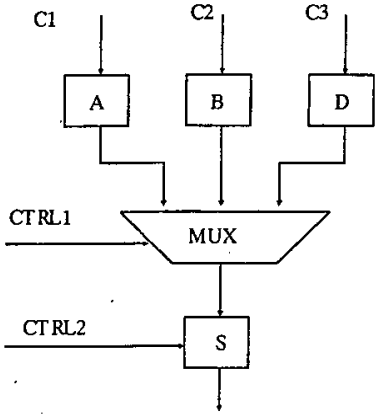

图 2.1：实现不同算法的可重组逻辑举例

根据上述可重组密码逻辑的设计思想，我们知道可重组算法逻辑由三部分组成：重组元素、可控接点、以及它们之间的连线。

重组元素用于实现各种密码算法的基本操作成分，是可重组算法逻辑用于构建各种密码算法的基本元素。重组元素之间的连接网络用于为各种密码算法建立所需的数据传输通路。显然，重组元素及其连接网络对可重组密码逻辑的适应范围、性能和规模起着决定性的作用，但是我们也应该注意到重组元素及其连接网络仅仅为实现不同的密码算法提供了硬件基础，它们必须在可控接点的控制下才能实现不同的密码算法，因此，如何对可控接点进行有效的控制，使可重组算法逻辑能够高效地实现不同的密码算法，也是一个非常重要的问题[5]。

2.2 重组元素的设计原则

任何密码算法实际上都是从明文空间、密钥空间到密文空间的一个变换（或者称为映射），并且这一变换通常都是由一系列的子变换按照一定的时序规则组合而成。利用可重组密码逻辑实现某个算法 A，实际上就是从可重组密码逻辑中选择某些重组元素 E={e1, e2, …, er} (r∈N)，然后让每个重组元素实现一个或几个特定的密码变换，然后再将这些密码变换按照一定的时序规则组合起来就构成了密码算法 A。即：

$$A=(T1,1+T1,2+\cdots+T1,m1)*(T2,1+T2,2+\cdots+T2,m2)*\cdots*(Tn,1+Tn,2+\ldots+Tn,mn) \qquad (2.1)$$

其中，每个 Ti, j(j=1, 2, …, mi, i=1, 2, …, n)表示由 E 中的某个重组元素实现的密码变换，

“+”表示并行关系、“*”表示串行关系。由此可见，可重组密码逻辑所能实现的任何一个密码算法都是由其重组元素所能实现的密码变换组合而成，因此，每个重组元素实现的密码变换越多，则可重组密码逻辑能够实现的算法也就越多。所以，为了以有限的规模实现尽可能多的密码算法，我们在设计重组元素的时候，应该使每种类型的重组元素实现该类型中的所有的密码变换，这样就使每个重组元素能够适应的算法最多，或者说重组元素的适应性达到最大。这就是我们设计重组元素的总体原则，我们称之为重组元素的最大适应性设计原则。另外，提出重组元素的最大适应性设计原则还出于扩展性的需要。我们希望我们设计的可重组密码芯片能够具有一定的扩展性，即在芯片制造完毕之后，它不仅能够实现设计时确定的多种密码算法，而且能够利用可重组密码逻辑中现有的重组元素开发新的密码算法，而未来所开发的新的密码算法需要每个重组元素实现什么样的密码变换，我们在设计时并不知道，因此，我们在设计重组元素的时候，应该使每种类型的重组元素实现该类型中的所有的密码变换。例如，假设我们在可重组密码逻辑中设置了一个 n*m 置换单元，置换单元是分组密码经常使用的一个密码组件，但不同的密码算法要求置换单元所实现的选择变换不同，未来所开发的新的算法也有可能使用选择变换，但具体是什么样的变换我们无法知道，因此为了使该 n*m 置换单元能够适应未来算法的需要，我们应该使该 n*m 置换单元能够实现所有的 n 输入、m 输出的选择变换[5]。

2.3 连接网络的设计原则

重组元素之间的连接网络是可重组算法逻辑的数据传输网络，对可重组算法逻辑的算法适应性、可扩展性、加/解密速度和规模具有决定性的影响，我们利用连通性、平均路径长度、网络宽度和网络规模等概念来说明这个问题。

连通性说明了可重组算法逻辑的连接网络的适应性和可扩展性，反映了连接网络对于利用重组元素重组密码算法所能够提供的支持的程度。我们知道，可重组算法逻辑所实现的任何一个密码算法，都是由重组元素以某种组合方式和连接关系构成的。如果一个连接网络是连通的，则说明我们可以以任意的组合方式、任意的连接关系使用现有的重组元素，从而使可重组算法逻辑在现有重组元素的基础上实现最多的密码算法，或者说使可重组算法逻辑达到最大的适应性和扩展性。反之，如果一个连接网络不是连通的，则因为连接网络对重组元素的某些组合方式和连接关系不提供支持，使得我们无法使用重组元素的这些组合方式和连接关系构造算法，从而使得可重组算法逻辑所能够适应的算法的数量减少，或者说使可重组算法逻辑的适应性和扩展性降低。因此，可重组算法逻辑的连接网络的第一个设计原则是满足连通性。

平均路径长度说明了可重组算法逻辑的任何两个重组元素之间的平均数据传输速度，网络宽度则说明了连接网络的并行传输数据的能力、反映了连接网络对并行操作的支持程度，所以平均路径长度和网络宽度结合起来就能够较好地刻画连接网络的数据传输速度。显然，平均路径长度越小、网络宽度越大，连接网络的数据传输速度就越快。所以，在设计连接网络的时候，我们应该使连接网络的平均路径长度尽可能地小、使网络宽度尽可能地大，这是我们设计连接网络的第二个原则。

网络规模包括了 3 个特征参数：引脚数（pins）、线网数（nets）以及多路开关的数量。引脚数（pins）和线网数（nets）越多，则连线所占的硅片的面积越大、线延迟越大、布局布线的难度也越大；开关数量越多，则连接网络所占的规模越大、数据传输路径的延时就越大、需控制的可控接点就越多。因此，在满足其他约束条件的前提下，我们应该尽量减小连接网络的引脚数（pins）、线网数（nets）和开关数量，这是我们设计连接网络的第三个原则[6]。

上面几个因素既是相互矛盾的，又是相互制约的。平均路径长度越小、网络宽度越大，则网络规模越大；反之，网络规模越小，则平均路径长度越大、网络宽度越小。因此，在设计可重组算法逻辑的连接网络时，我们需要根据特定的设计目标和约束条件，综合考虑上述各个因素。下面我们给出可重组算法逻辑连接网络的三种典型结构，并针对其连通性、平均路径长度、网络宽度和网络规模等各个方面进行分析。

2.3.1 全部直接相连型连接网络

所谓全部直接相连型连接网络，是指连接网络中的任意两个重组元素都直接相连，即任一重组元素的输出直接连接到所有重组元素的所有输入端口，或者说任一重组元素的任一输入端口直接与所有的重组元素的输出相连[22]。下面给出了 4 个重组元素的全部直接相连网络的结构图：

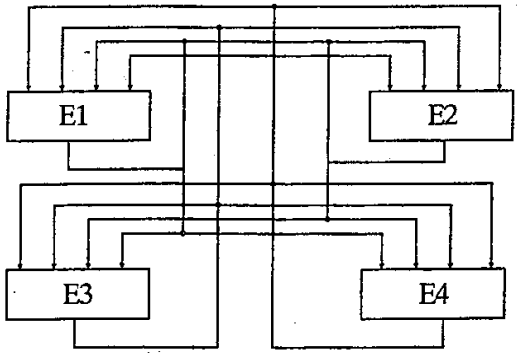

图 2.2　4 个重组元素的全部直接相连网络的结构图

全部直接相连型连接网络特点如下：

1、全部直接相连型连接网络是连通的。

2、在全部直接相连型连接网络中，任何两个重组元素之间的数据传输路径的长度为 1，连接网络的平均路径长度也为 1。

3、设某一全部直接相连型连接网络中共有 n 个重组元素，分别记为 e_1、e_2、…e_n，其中 $e_i(1 \le i \le n)$ 的输入数据的宽度为 iw_i 位、输出数据的宽度为 ow_i 位，则该连接网络的网络宽度为 $\min\{\sum_{i=1}^{n} ow_i, \sum_{i=1}^{n} iw_i\}$。

4、设某一全部直接相连型连接网络中共有 n 个重组元素，分别记为 e_1、e_2、…e_n，其中 $e_i(1 \le i \le n)$ 的输入端口的数据宽度为 iw_i 位（在这里我们假设所有的重组元素都只有一个输入端口）、输出数据的宽度为 ow_i 位，则该连接网络共有 $(n+1)\sum_{i=1}^{n} ow_i$ 个引脚（pins）、$\sum_{i=1}^{n} ow_i$ 个线网（nets）、$\sum_{i=1}^{n} \left(\left\lceil \frac{\sum_{i=1}^{n} ow_i}{iw_i} \right\rceil - 1 \right) iw_i$ 个 2 选 1 的开关（选通器）。

例如，假设某全部直接相连型连接网络中共有 64 个重组元素，每个重组元素只有一个输入端口，且输入数据和输出数据的宽度均假定为 16 位，则特点 4 该连接网络共有 66560 个引脚（pins）、1024 个线网、64512 个 2 选 1 的选通器。另外，由特点 3 知，该连接网络的网络宽度为 1024 位。

2.3.2 寄存器堆间接相连型连接网络

所谓寄存器堆间接相连型连接网络，是指在连接网络中设置一个寄存器堆，使寄存器堆与其它重组元素都直接相连，而除寄存器堆以外的任意两个重组元素都通过寄存器堆间接相连。即除寄存器堆以外的任何一个重组元素的输出都与寄存器堆的所有输入端口相连、而不与除寄存器堆以外的任何其它重组元素相连，并且，除寄存器堆以外的任何一个重组元素的输入端口也只与寄存器堆的某个输出端口相连、而不与除寄存器堆以外的任何其它重组元素相连。显然，在寄存器堆间接相连型连接网络中，寄存器堆是整个网络的数据交换的中心。图 2.3 给出了四个重组元素通过一个寄存器堆间接相连的例子[22]：

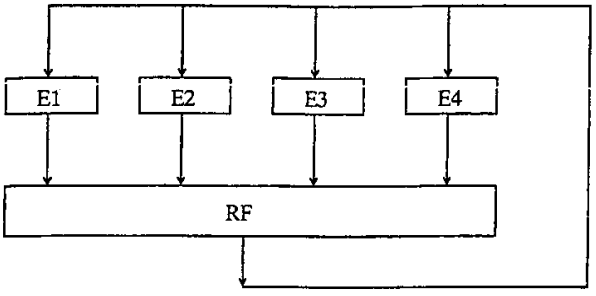

图 2.3 四个重组元素通过一个寄存器堆间接相连的连接网络图

寄存器堆间接相连型连接网络特点如下：

1、寄存器堆间接相连型连接网络是连通的。

2、在寄存器堆间接相连型连接网络中，寄存器堆和其它重组元素之间的数据传输路径的长度为 1，而除寄存器堆之外的任何两个重组元素之间的数据传输路径的长度为 2。

3、设某一寄存器堆间接相连型连接网络中，除寄存器堆外共有 n 个重组元素，则该连接网络的数据传输路径的平均长度为 $1+\left(\frac{n}{n+1}\right)^2$。

4、设某一寄存器堆间接相连型连接网络中的寄存器堆有 k 个输入端口、m 个输出端口，且其输入、输出数据的数据宽度为 w 位，则该连接网络的网络宽度为(k+m)w 位。

设某一寄存器堆间接相连型连接网络中的寄存器堆有 k 个输入端口、m 个输出端口，且其输入、输出数据的宽度为 w 位，另外假设除寄存器堆外共有 n 个重组元素，分别记为 e_1、e_2、…e_n，其中 $e_i(1\le i\le n)$的输入数据宽度为 iw_i 位(假设只有一个输入端口)、输出数据的宽度为 ow_i 位，则该连接网络共有 $\sum_{i=1}^{n}(iw_i+ow_i)+k(w+\sum_{i=1}^{n}ow_i)+mw$ 个引脚(pins)、$\sum_{i=1}^{n}ow_i+mw$ 个线网(nets)和 $kw\left(\left\lceil\frac{w+\sum_{i=1}^{n}ow_i}{w}\right\rceil-1\right)$个 2 选 1 的选通器。

例如，假设某寄存器堆间接相连型连接网络中共有 1 个寄存器堆和其它 63 个重组元素，并假定寄存器堆有 8 个写端口、8 个读端口，而除寄存器堆之外的任一重组元素都只有 1 个输入端口、1 个输出端口，并且所有的输入、输出端口的数据宽度均假定为 16 位，则根据特点 5，该连接网络共有 10336 个引脚（pins)、1136 个线网（nes)、8064 个 2 选 1 的选通器。另外，根据特点知，该连接网络的网络宽度为 256 位。

2.3.3 部分直接部分间接相连型连接网络

所谓部分直接部分间接相连型连接网络，是在寄存器堆间接相连型连接网络的基础上增加某些直接相连路径而构成的。即首先保证任何两个重组元素都可以经过寄存器堆间接相连，然后在此基础上增加一些连线，使某些重组元素能够直接相连。显然，在部分直接部分间接相连型连接网络中，部分重组元素是直接相连的，而另外一部分重组元素则是通过寄存器堆间接相连的。图2.4给出了四个重组元素部分直接部分间接相连的例子[22]：

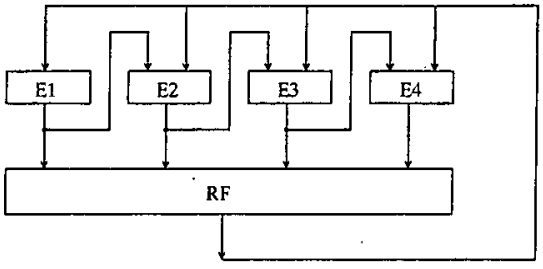

图2.4：四个重组元素的部分直接部分间接相连网络

部分直接部分间接相连型连接网络特点如下：

1、部分直接部分间接相连型连接网络是连通的。

2、在部分直接部分间接相连型连接网络中，任何两个重组元素之间的数据传输路径的长度或者为1或者为2。

3、部分直接部分间接相连型连接网络的数据传输路径的平均长度、网络宽度以及网络规模都介于全部直接相连型连接网络和寄存器堆间接相连型连接网络之间。

2.3.4 三种类型连接网络的比较

我们从连通性、数据传输路径的平均长度、网络宽度、网络规模等各个方面对上述三种类型的连接网络进行比较：

（1）连通性：三种类型的连接网络都是连通的。

（2）数据传输路径的平均长度：全部直接相连型连接网络的数据传输路径的平均长度最短、为1，寄存器堆间接相连型连接网络的数据传输路径的平均长度最长、接近于2，部分直接部分间接相连型连接网络的数据传输路径的平均长度介于上述二者之间。

（3）网络宽度：全部直接相连型连接网络的网络宽度最大，在同一时刻，能够将所有重组

元素的输出结果传输到任何需要的地方，因此能够最大限度地支持并行操作；寄存器堆间接相连型连接网络的网络宽度最小，等于在同一时刻能够读出或写入寄存器堆的不同数据的宽度之和，实际上就是寄存器堆的输入、输出端口的个数及其数据宽度的乘积；部分直接部分间接相连型连接网络的网络宽度介于上述二者之间。我们知道网络宽度刻画了连接网络支持并行操作的能力，网络宽度越大说明支持并行操作的能力越大，但由于数据依赖性等因素的影响，密码算法中的并行操作的个数是有限度的，因此并非网络宽度越大越好，只要网络宽度能够适应密码算法中并行操作的需求即可，过大的网络宽度将造成规模的增加和资源利用率的降低。例如，目前绝大多数分组密码算法的分组长度都不超过 128 位，因此在密码算法的执行过程中需要同时传输的数据的宽度一般不超过 256 位，所以我们可以将连接网络的网络宽度设计为 256 位。

（4）网络规模：全部直接相连型连接网络中的引脚数（pins）、线网数（nets）和所需的开关数最多，因此网络规模最大，而且，随着重组元素个数的增加，全部直接相连型连接网络的网络规模增加的速度非常快。我们知道，引脚、线网和开关的数量越多，连线所占的面积就越大、布局布线的难度就越大、连线的延迟就越大，因此在重组元素个数较多的时候，全部直接相连型连接网络难以实现。寄存器堆间接相连型连接网络的网络规模最小。部分直接部分间接相连型连接网络的网络规模介于上述两种结构之间。

2.4 可控接点的控制方法

可重组算法逻辑实现逻辑重组的基础是重组元素的功能可变、重组元素之间的数据传输路径可变。重组元素的功能可变，是为了适应不同算法的不同的操作功能需求；重组元素之间的数据传输路径可变，是为了适应不同算法的数据传输需求。为了实现重组元素的可变，我们必须在重组元素的内部电路结构中设置某些可控接点，以控制重组元素的功能的改变，这类可控接点我们称之为功能控制接点；为了实现重组元素之间的数据传输路径的可变，我们必须在重组元素的数据传输网络中设置某些可控接点，以控制重组元素之间的数据传输路径的改变，这类可控接点我们称之为通路控制接点。综上所述，可重组算法逻辑的可控接点主要有两类：功能控制接点和通路控制接点。

可重组算法逻辑的电路结构和功能的改变是通过对可控接点赋以不同的编码值来实现的。可控接点的数量及其编码宽度与可重组算法逻辑的灵活性密切相关。一般来说，可控接点的数量越多、编码宽度越大，可重组算法逻辑的灵活性也越大。当可控接点的数量较多、编码宽度较大时，如何对可控接点进行控制就成为一个非常重要的问题。

密码算法的每一个操作步骤都是通过对多个可控接点赋予一定的控制编码来完成的，如果

每一个操作步骤所需的所有控制编码都来自当前的指令，由于某些可控接点的控制编码的宽度较大(例如置换控制接点，S 盒控制接点等)，而指令编码的容量又受到指令长度的限制，因此每条指令所包含的可控接点的个数较少，往往难以满足并行操作的需求，从而降低了算法的执行速度。

通过对大量的算法进行分析，我们发现，有很多可控接点的控制编码在算法执行过程中保持不变（例如置换控制接点和 S 盒控制接点），而另一部分可控接点的控制编码在算法执行过程中需要频繁的改变（例如读/写寄存器堆控制接点、通路控制接点）。根据这一特点，我们将可控接点分成两类，将在算法执行过程中编码保持不变的可控接点称为静态可控接点，将在算法执行过程中需要频繁改变编码的可控接点称为动态可控接点。对于静态可控接点，我们用专门的装载指令在算法执行之前赋予需要的控制编码，这些编码在算法的执行过程中保持不变，因此在算法的执行过程中，我们不需要再对静态可控接点加以控制，而只需要控制那些动态可控接点即可，这样大大地节省了指令字的空间，使得每条指令字能够包含更多的动态控制编码，从而提高了操作的并行性，加快了算法的执行速度[5]。

通常情况下，静态可控接点都是功能控制接点，而通路控制接点都是动态可控接点。

第三章 加解密算法分析

3.1 密码算法原理及密码算法分类

在信息领域中，密码技术是保护信息安全的关键技术，它是集数学、通信技术和计算机科学等学科于一身的交叉学科。密码技术从根本上来说是密码算法的使用，是安全的基础和保障[23]。密码算法的目的是为了保护信息的保密性、完整性和安全性，简单地说就是信息的防伪造与防窃取[22]。

3.1.1 密码算法原理简述[4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 29, 30, 35]

密码系统由密码算法以及所有可能的明文、密文和密钥组成。伪装前的信息被成为明文，用某种方法伪装信息以隐藏它的内容的过程称为加密，被加密的消息称为密文，而把密文转变为明文的过程称为解密。用于加密和解密的数学函数称为密码算法。密钥是密码算法的加解密参数，所有算法的安全性都基于密钥的安全性。

整个密码系统可以用数学符合描述：S＝（P，C，K，E，D），其中 P 是整个明文空间，C 是密文空间，K 是密钥空间，E 是加密算法，D 是解密算法。当给定的密钥为 k∈K，这时加密/解密算法表示为 Ek /Dk，这样，加密/解密函数表示为（如图 3.1）：

Ek（P）=C （表示用加密算法对明文 P 加密得到密文 C）

Dk（C）=P （表示用解密算法对密文 C 解密得到密文 P）

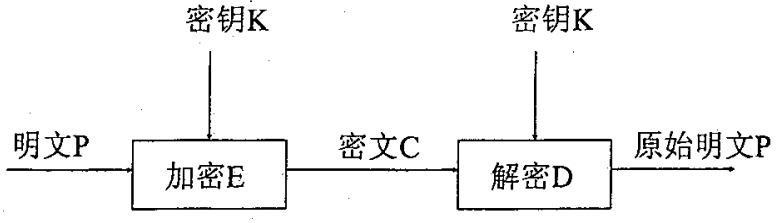

图 3.1 数据加密/ 解密的基本过程

这些函数还具有下面的特性：

$$Dk(Ek(P)) = P \tag{3.1}$$

这表明加密变换 Ek 和解密 Dk 变换是可逆的。

3.1.2 密码算法分类[4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 29, 30, 35]

根据密码算法所使用的加密密钥和解密密钥是否相同，能否由加密过程推导出解密过程，或由解密过程推导出加密过程，可将密码算法分为对称密码算法(也称单钥密码算法、秘密密钥

密码算法、对称密钥密码算法)和公开密钥密码算法(非对称密码算法、也称双钥密码算法、非对称密钥密码算法)，除此之外，密码学中还较多使用Hash函数作为辅助的加密算法。

1、对称密码算法

如果一个密码算法的加密密钥和解密密钥相同，或由其中一个很容易推导出另一个，该算法就是对称密码算法。对称密码的特点是速度快、安全强度高，主要用作数据加密算法。

对称密码根据加密模式又可分为分组密码和序列密码。分组密码的典型算法有：DES，IDEA和AES等，分组密码是目前在商业领域比较重要使用较多的密码，广泛用于信息的保密传输和加密存储；序列密码的典型算法有：RC4，SEAL，A5等，序列密码多用于流式数据的加密，特别是对实时性要求比较高的语音和视频流的加密传输。

2、公开密钥密码算法

如果一个密码算法的加密密钥和解密密钥不同，或者由其中的一个推导不出另一个，则该算法就是公开密钥密码算法，简称公钥密码算法。使用公钥密码的每一个用户都拥有基于特定公钥算法的一个密钥对(e，d)，公钥e公开，公布于用户所在系统认证中心(CA)的目录服务器上，任何人都可以访问，私钥d为所有者保管并严格保密，两者不相同且互为对方的解密密钥。

公钥密码的典型算法有：RSA，ECC，DSA，ElGamal，Diffie-Hellman(DH)密钥交换算法等。公钥密码能够用于数据加密、密钥分发、数字签名、身份认证、信息的完整性认证、信息的非否认性认证等。其中可以用于加密的算法有：RSA，ECC，ElGamal等；可以用于密钥分发的算法有：RSA，ECC，DH等；可以用于数字签名、身份认证、信息的完整性认证、信息的非否认性认证的有RSA，ECC，DSA，ElGamal等。

3、单向密码算法

单向密码体制使用单向的散列(Hash)函数，它是从明文到密文的不可逆函数，也就是说只能加密不能还原。单向散列函数H作用于任意长度的信息M，返回一固定长度的散列值(也称摘要信息)h=H(M)。

典型的散列函数有：MD5，SHA-1，HMAC等。单向散列函数主要用在一些只需加密不需解密的场合：如验证数据的完整性、口令表的加密、数字签名、身份认证等。

3.2 加解密算法分析

本课题我们选择对称算法中的五种算法作为提炼重组元素的样本空间，这五种算法是IDEA、GOST、RC6、RIJNDAEL、3DES。下面我们对这五种算法进行详细的介绍和分析。

3.2.1 3DES 算法简介与资源分析

1、3DES 算法介绍

3DES 就是三重 DES，下面我们先来介绍 DES 算法。

DES 算法在文献[4][14][15][30][36]中都有介绍，现总结如下：

1977 年 1 月，美国政府颁布：采纳 IBM 公司设计的方案作为非机密数据的正式数据加密标准（Data Encryption Standard）。数据加密标准作为 ANSI 的数据加密算法（Data Encryption Algorithm, DEA）和 ISO 的 DEA-1 成为一个世界范围内的标准已经 20 多年了。尽管它带有过去时代的特征，但它很好地抗住了多年的密码分析，除可能的最强有力的敌手外，对其他的攻击仍是安全的。

DES 是一个分组加密算法，它以 64 位为分组对数据加密。64 位一组的明文从算法的一端输入，64 位的密文从另一端输出。DES 有 16 轮，这意味着要在明文分组上 16 次实施相同的组合技术，DES 流程如图 3.2。

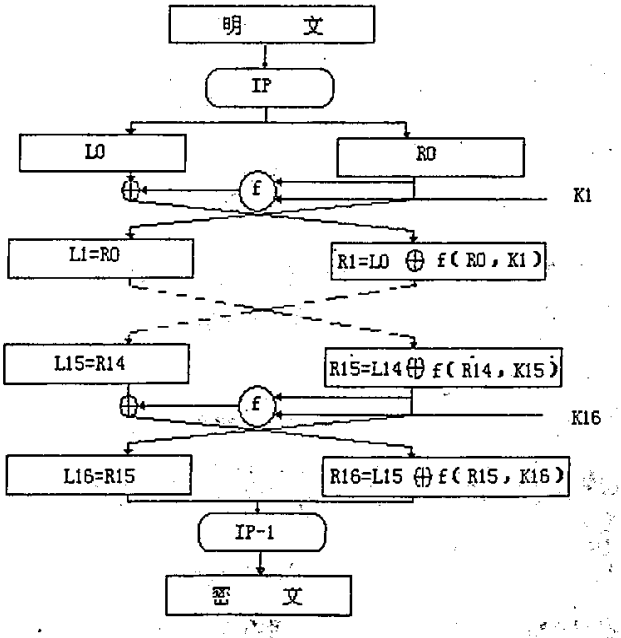

图 3.2 DES 流程

DES 对 64 位的明文分组进行操作。通过一个初始置换，将明文分组分成左半部分和右半部分，各 32 位长。然后进行 16 轮完全相同的运算，这些运算被称为函数 f，在运算过程中数据与密钥结合。经过 16 轮后，左、右半部分合在一起经过一个末置换（初始置换的逆置换），这样该算法就完成了。

在每一轮（见图 3.3）中，密钥位移位，然后再从密钥的 56 位中选出 48 位。通过一个扩展置换将数据的右半部分扩展成 48 位，并通过一个异或操作与 48 位密钥结合，通过 8 个 S 盒将这 48 位替代成新的 32 位数据，再将其置换一次。这四步运算构成了函数 f。然后，通过另一个异或运算，函数 f 的输出与左半部分结合，其结果即成为新的右半部分，原来的右半部分成为新的左半部分。将该操作重复 16 次，便实现了 DES 的 16 轮运算。

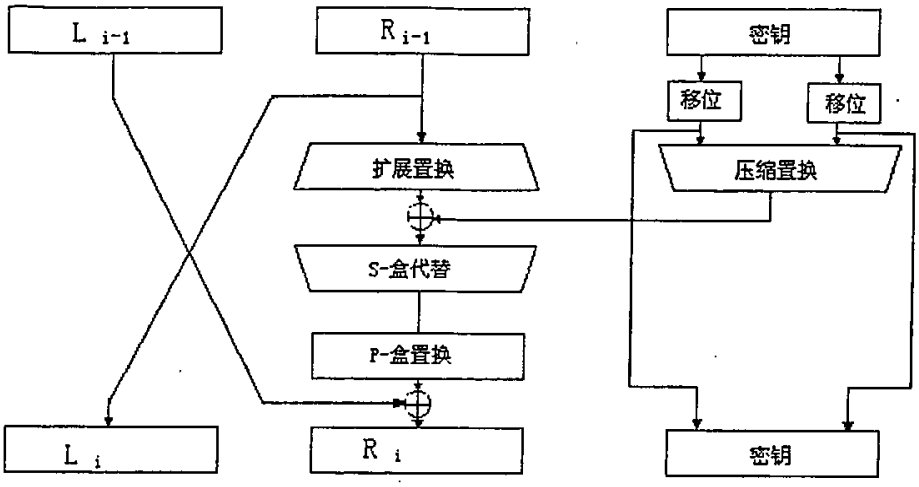

图 3.3　一轮 DES

3DES 就是三重 DES，如图 3.4。

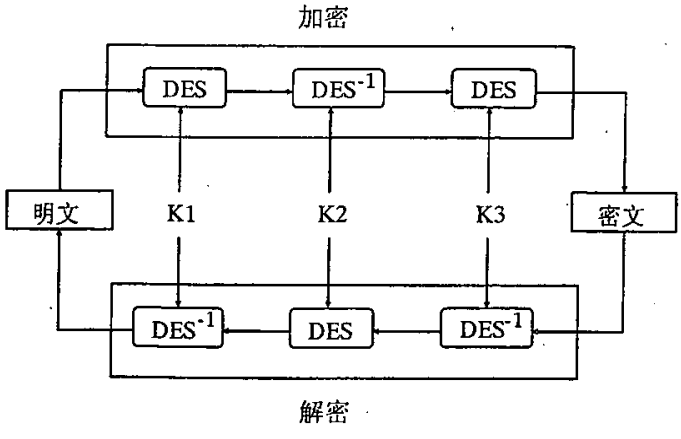

图 3.4 3DES 示意图

其中 K1, K2, K3 是密钥，且 K2 与 K1、K3 不能相同，K1、K3 可以相同，为了设计简单，我们就取 K1、K3 一样。

2、3DES 资源分析

由上可以看出，3DES 算法操作过程如下：

子密钥产生过程：输入 64 位用户密钥、28 位循环移位 2 个、压缩置换 56*48 一个、子密钥结果保存。

加密/解密过程：输入 64 位明文、初始置换 64*64 一个、扩展置换 32*48 一个、48 位异或一个、6*4 S-盒 8 个、P-置换 32*32 一个、32 位异或一个、循环、数据传输、末置换 64*64 一个、结果保存。

在下表中我们给出了用到的器件名称、规格、功能及器件数量。

表 3.1：3DES 算法资源表

IP 名称	规格	功能	数量
SHFT_28	28 位循环左移单元	实现 28 位数据的左移位操作	2 个
PMT_56*48	56*48 压缩置换	实现 1-56 输入到 1-48 输出的任意压缩置换	1 个
PMT_64	64*64 置换	实现 1-64 输入到 1-64 输出的任意置换	2 个
PMT_32*48	32*48 扩展置换	实现 1-32 输入到 1-48 输出的任意扩展置换	1 个
XOR_48	48 位逐位异或	实现 48 位数据的逐位异或操作	1 个
SBOX	6*4S 盒	实现 6 输入到 4 输出的非线性代替变换	8 个
PMT_32	32*32 置换	实现 1-32 输入到 1-32 输出的任意置换	1 个
XOR_32	32 位逐位异或	实现 32 位数据的逐位异或操作	1 个

3.2.2 GOST 算法简介与资源分析

1、GOST 算法介绍

GOST 算法在文献[4]中有介绍，现总结如下：

GOST 是前苏联设计的分组密码算法，GOST 是 Gosudarstvennyi Standard 或 Government Standard 的缩写。GOST 算法明文为 64 比特，密钥 256 比特，对加密算法执行 32 趟，即迭代 32 次。加密时，首先将明文分成左半部分 L 和右半部分 R，各 32 比特，第 i 轮的子密钥为 Ki，其中左为 Li，右为 Ri，GOST 的第 i 轮为：

$$Li = Ri-1 \tag{3.2}$$

$$Ri = Li-1 \oplus f(Ri-1, Ki) \tag{3.3}$$

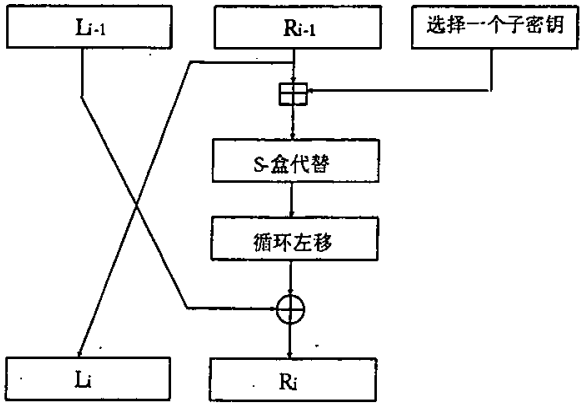

图 3.5 GOST 算法

图 3.5 给出了 GOST 算法的框图，函数为 f：首先右半部分 Ri-1 与子密钥 Ki 作 $\mathrm{mod}\,2^{32}$ 的加法，即

$$S \leftarrow (Ri\text{-}1+Ki) \mod 2^{32} \tag{3.4}$$

该结果分成 8 个 4-位分组，每个分组被作为不同的 S-盒的输入，GOST 有 8 个不同的 S 盒。最前面一组的 4 比特输入到第一个 S 盒，第二组的 4 比特输入到第二个 S 盒，依次类推。每个 S 盒都是 0，1，2，3，…，15 的一个排列，8 个 S 盒各不相同这些被认为是附加的密钥。8 个 S-盒的输出重组为 32 位字，然后整个字循环左移 11 位。最后，该结果与左半部分异或变为新的右半部分，原右半部分为新的左半部分，将此操作重复进行 32 次。

子密钥的产生过程异常的简单，将 256 比特的密钥分成 8 个 32 位分组：K1，K2，…，K8。每一轮使用表 3.1 给出的不同的子密钥，解密除了密钥 Ki 逆序外与加密相同。

表 3.2：GOST 算法中不同轮中所用的密钥表：

轮次	1, 2, 3, 4, 5, 6, 7, 8	9, 10, 11, 12, 13, 14, 15, 16
I	1, 2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 6, 7, 8
轮次	17, 18, 19, 20, 21, 22, 23, 24	25, 26, 27, 28, 29, 30, 31, 32
j	1, 2, 3, 4, 5, 6, 7, 8	8, 7, 6, 5, 4. 3. 2. 1

2、GOST 算法资源分析

由上可以看出，GOST 算法操作过程如下：

子密钥产生过程：输入 256 位用户密钥、子密钥结果保存。

加密/解密过程：输入 64 位明文、模 2 的 32 次方加一个、4*4 S 盒变换 8 个、32 位循环左移位一个、32 位异或一个、循环、数据传输、结果保存。

在下表中我们给出了用到的器件名称、规格、功能及器件数量。

表 3.3：GOST 算法资源表

IP 名称	规格	功能	数量
Mod2n32add	模 2 的 32 次方加	实现 2-32 输入到 1-32 输出的模 2 的 32 次方加	1 个
SBOX	4*4S 盒	实现 8 输入到 8 输出的非线性代替变换	8 个
SHFT_32	32 位移位单元	实现 32 位数据的移位操作	1 个
XOR_32	32 位逐位异或	实现 32 位数据的逐位异或操作	1 个

3.2.3 RIJNDAEL 算法简介与资源分析

1、RIJNDAEL 算法介绍

RIJNDAEL 算法在文献[7][14][35]中都有介绍，现总结如下：

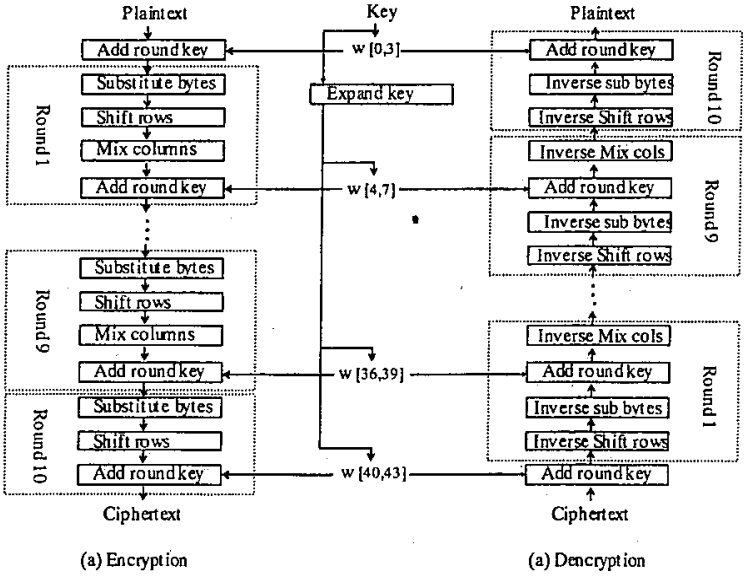

图 3.6 Rijndael 的一般流程图

Rijndael 是由比利时的 Joan Daemen 和 Vincent Rijmen 设计的一迭代分组密码，它是在 Square 密码的基础上发展起来的。Rijndael 在 2000 年 10 月 2 日被推荐为高级加密标准（AES-Advanced Encryption Standard）。Rijndael 分组长度和密钥长度都可变，各自可以独立地指定为 128/192/256bits。在分组长度和密钥长度取不同的值，密码算法运算的流程稍有区别，

下面我们就以分组长度和密钥长度都是 128 位进行介绍。

Rijndael 的一般流程如图 3.6。在图中，(a)是加密流程，(b)是解密流程。由于采用了模块化设计，算法包含 4 个步骤：1.字节替换；2.行位移(循环移位)；3.列混淆；4.密钥加法，这些步骤循环 10 轮。第 10 轮的又不一样，没有列混淆。

图 3.7 中 Key 代表密钥，密钥 bit 的总数＝分组长度×（轮数 Round＋1）例如当分组长度为 128bits 和轮数 Round 为 10 时，轮密钥长度为 128×(10＋1)＝1408bits。将 128 位密码密钥扩展成一个扩展密钥。从扩展密钥中取出轮密钥：第一个轮密钥由扩展密钥的第一个 Nb 个 4 字节字，第二个圈密钥由接下来的 Nb 个 4 字节字组成，以此类推。

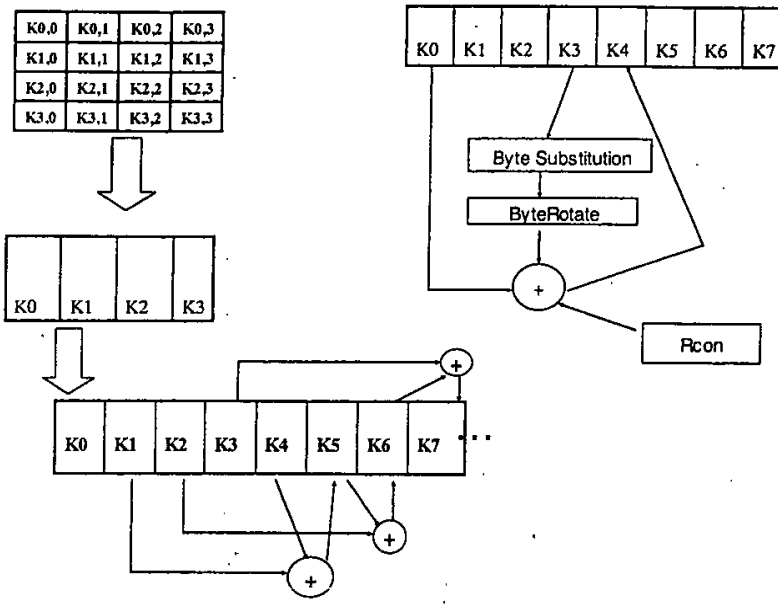

图 3.7 Rijndael 的密钥产生流程图

2、Rijndae 算法资源分析

由上可以看出，RIJNDAEL 算法用到了 32 位异或运算，32 位环移位单元，8*8S 盒运算，模 8 次多项式乘法运算；

子密钥产生过程：输入 128 用户密钥、8*8 S 盒变换、32 位环移位、32 位异或、数据传输、结果保存。

加密过程：输入 128 位明文、8*8 S 盒变换、32 位移位、32 位异或、模 8 次多项式乘法、循环、数据传输、结果保存。

解密过程：输入 128 位密文、8*8 S 盒变换、32 位移位、32 位异或、模 8 次多项式乘法、循环、数据传输、结果保存。

在下表中我们给出了用到的器件名称、规格、功能及器件数量。

表 3.4：RIJNDAEL 算法资源表

IP 名称	规格	功能	数量
SHFT_32	32 位循环移位单元	实现 32 位数据的循环移位操作	4 个
XOR_32	32 位逐位异或	实现 32 位数据的逐位异或操作	4 个
SBOX	8*8S 盒	实现 8 输入到 8 输出的非线性代替变换	16 个
MPOLMUL_8	模 8 次多项式乘法单元	实现模 8 次多项式乘法运算	4 个

3.2.4 IDEA 算法简介与资源分析

1、IDEA 算法介绍

IDEA 算法在文献[4][14][15][29][35]中都有介绍，现总结如下：

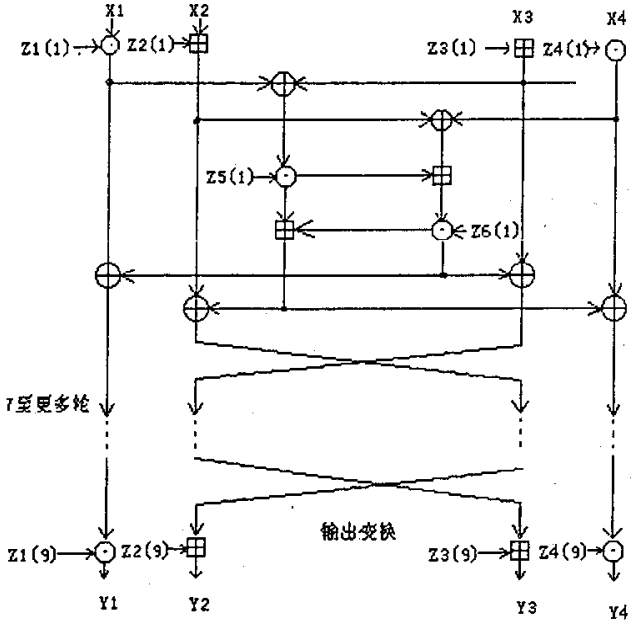

图 3.8 IDEA 算法的流程图

图 3.8 是 IDEA 的一个纵览。64-位数据分组被分成 4 个 16-位子分组：X1，X2，X3，和 X4。这四个子分组成为算法的第一轮输入，总共有 8 轮。在每一轮中，这四个子分组相互间相异或，

相加，相乘，且与6个16-位子密钥相异或，相加，相乘。在轮与轮间，第二和第三个子分组交换。最后在输出变换中四个子分组与四个子密钥进行运算。

在每一轮中，执行的顺序如下：

1. X1和第一个子密钥相乘。
2. X2和第二个子密钥相乘。
3. X3和第三个子密钥相乘。
4. X4和第四个子密钥相乘。
5. 将第1步和第3步的结果相异或。
6. 将第2步和第4步的结果相异或。
7. 将第5步的结果与第五个子密钥相乘。
8. 将第6步和第7步的结果相加。
9. 将第8步的结果与第六个子密钥相乘。
10. 将第7步和第9步的结果相加。
11. 将第1步和第9步的结果相异或。
12. 将第3步和第9步的结果相异或。
13. 将第2步和第10步的结果相异或。
14. 将第4步和第10步的结果相异或。

每一轮的输出是第11、12、13和14步的结果形成的4个子分组。将中间两个分组交换（最后一轮除外）后，即为下一轮的输入。

经过8轮运算之后，有一个最终的输出变换：

Xi：16-位明文子分组

Yi：16-位密文子分组

Zi：（r）：16-位子密钥

$\oplus$：16-位子密钥的相异或

$\odot$：16-位整数的模2^{16}加

$\boxplus$：16-位整数与2^{16}对应0子分组的模$2^{16}+1$乘

1．X1和第一个子密钥相乘。

2．X2和第二个子密钥相加。

3．X3和第三个子密钥相加。

4．X4 和第四个子密钥相乘。

最后，这 4 个子分组重新连接到一起产生密文。

产生子密钥也很容易。这个算法用了 52 个子密钥（8 轮中的每一轮需要 6 个，其他 4 个用于输出变换）。首先，将 128-位密钥分成 8 个 16-位子密钥。这些是算法的第一批 8 个子密钥（第一轮 6 个，第二轮的头 2 个）。然后，密钥向左环移动 25 位产生另外 8 个子密钥，如此进行直到算法结束。

2、IDEA 算法资源分析

由上面流程分析可以看出，IDEA 算法操作过程如下：

子密钥产生过程：输入 128 位用户密钥、128 位循环移位一个、子密钥结果保存。

加密/解密过程：输入 64 位明文、模 $2^{16}+1$ 乘 2 个、模 $2^{16}+1$ 乘逆 2 个、模 2^{16} 加 2 个、模 2^{16} 加逆 2 个、16 位异或 4 个、循环、数据传输、结果保存。

在下表中我们给出了用到的器件名称、规格、功能及器件数量。

表 3.5：GOST 算法资源表

IP 名称	规格	功能	数量
SHFT_128	128 位循环移位单元	实现 128 位数据的循环移位操作	1 个
Mod2n161mul	模 $2^{16}+1$ 乘	实现 2-16 输入到 1-16 输出的模 $2^{16}+1$ 乘	2 个
Mod2n161mulag	模 $2^{16}+1$ 乘逆	实现 1-16 输入到 1-16 输出的模 $2^{16}+1$ 乘逆	2 个
Mod2n16add	模 2^{16} 加	实现 2-16 输入到 1-16 输出的模 2^{16} 加	2 个
Mod2n16addag	模 2^{16} 加逆	实现 1-16 输入到 1-16 输出的模 2^{16} 加逆	2 个
XOR_16	16 位逐位异或	实现 16 位数据的逐位异或操作	4 个

3.2.5 RC6 算法简介与资源分析

1、RC6 算法介绍

RC6 算法在文献[4]中有介绍，现总结如下：

RC6 是参数变量的分组算法，实际上是由三个参数确定的一个加密算法族。一个特定的 RC6 可以表示为 RC6-w/r/b。其中这三个参数 w、f 和 b 分别按照表 3.2 所列定义。

表 3.6 RC6 算法参数定义

参　数	定　义	常　用
w	以比特表示的字的尺寸	16, 32, 64
r	加密轮数	0～255
b	密钥的字节长度	0～255

我们在本次设计中取 w=32，r=16，b=32。即此时的 RC6 算法是 128.位明文（密文）分组，128 位密钥，16 轮操作运算。

RC6 算法加密流程如图 3.9。

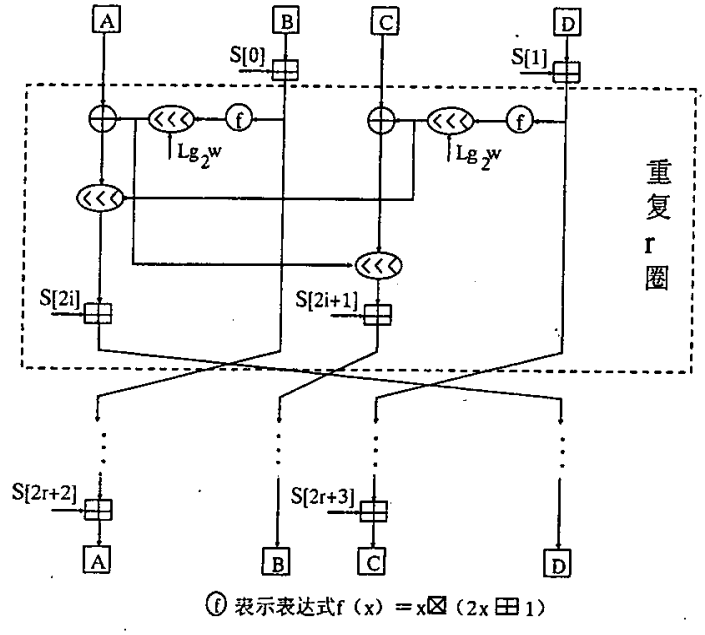

图 3.9　RC6 算法加密流程图

其中“⊞”表示模 2^W 加法运算，“⊕”表示逐位异或运算，⊠模 2^W 乘法，⋘表示循环左移。

RC6 由三部分组成，分别为混合密钥生成过程、加密过程和解密过程。在这两种算法中，共使用了六种基本运算：

①模 2^W 加法运算，表示为“＋”；

②模 2^W 减法运算，表示为“－”；

③逐位异或运算，表示为“⊕”；

④循环左移，字 a 循环左移 b 比特表示为“a<<<b”；

⑤循环右移，字 a 循环右移 b 比特表示为“a>>>b”；

⑥模 2^W 乘法，表示为“×”。

RC6 算法使用了上述所有的运算部件。

（1）RC6 算法混合密钥生成过程的伪代码表示

```
S[0]=Pw
for i=1 to t-1 do
S[i]=S[i-1]+Qw
输入比特数大小为 8，密钥长度为 b 的用户密钥 K[0]至 K[b-1]
转换 K[0]至 K[b-1]为数组长度为 c，比特数为 w 的 L[]数组
i=j=0 x=y=0
do 3×max(t,c)times:
S[i]=(S[i]+x+y)<<<3;X=S[i];i=(i+1)mod t
L[j]=(L[j]+x+y)<<<(x,y);X=L[j];j=(j+1)modC
```

其中 c=[b×8/w]方括号表示上取整运算，t=2r+4，当 w 分别为 16、32、64 时，常数 Pw、Qw 分别如表 3.3 所列。

表 3.7 常数 Pw、Qw 取值表

W	16	32	64
Pw	0xB7E1	0xB7E15163	0xB7E151628AED2A6B
Qw	0x9E37	0x9E3779B9	0x9E3770B97F4A7C15

（2）RC6 加密算法过程伪代码表示

```
Input(A,B,C,D)
B=B+S[0]D=D+S[1]
for i=1 to r do
t=(B×(2B+1))<<<log2w
u=(D×(2D+1))<<<log2w
A=((A⊕t)<<<t)+S[2i]
C=((C⊕u)<<<u)+S[2i+1]
```

C=((C⊕u)<<<u)+S[2i+1]

(A, B, C, D)=(B, C, D, A)

A=A+S[2i+2]C=C+S[2i+3]

Output(A, B, C, D)

其中初始的 A、B、C、D 分别为要加密的四个比特数为 w 的数据，最终的 A、B、C、D 分别为加密好的四个比特数为 w 的数据。

(3) RC6 解密算法过程的伪代码表示

Input(A, B, C, D)

C=C-S[2i+3]A=A-S[2i+2]

for i=1 to r do

(A, B, C, D)=(D, A, B, C)

u=(D×(2D+1))<<<log2w

t=(B×(2B+1))<<<log2w

C=((C-S[2(r-i)+3])>>>t)⊕u

A=((A-S[2(r-i)+2])>>>u)⊕t

D=D-S[1] B=B-S[0]

Output(A, B, C, D)

其中初始的 A、B、C、D 分别为已经被加密的四个比特数为 w 的数据，最终的 A、B、C、D 分别为解密后的四个比特数为 w 的数据。

2、RC6 算法资源分析与资源分析

由上可以看出，RC6 算法操作过程如下：

子密钥产生过程：输入 128 位用户密钥、模 2^{32} 加、32 位循环移位、循环、数据传输、子密钥结果保存。

加密/解密过程：输入 128 位明文、模 2^{32} 加、模 2^{32} 减、32 位循环左移位、32 位异或、模 2^{32} 乘、循环、数据传输、结果保存。

在下表中我们给出了用到的器件名称、规格、功能及器件数量。表 3.8：

表 3.8：GOST 算法资源表

IP 名称	规格	功能	数量
Mod2n32add	模 2 的 32 次方加	实现 2-32 输入到 1-32 输出的模 2 的 32 次方加	2 个
Mod2n32dec	模 2 的 32 次方减	实现 2-32 输入到 1-32 输出的模 2 的 32 次方减	2 个
SHFT_32	32 位循环移位单元	实现 32 位数据的循环移位操作	2 个
XOR_32	32 位逐位异或	实现 32 位数据的逐位异或操作	2 个

第四章 重组元素设计

在本章重组元素的设计中，我们遵从让每个重组元素实现一个或几个特定的密码变换，尽量使得每个重组元素具有可重组的价值。为可重组算法逻辑芯片提供可编程的 IP 核。

4.1 S 盒设计

S 盒是大多数分组密码算法中唯一的非线性结构，它的密码强度直接决定了密码算法的好坏。本课题中 S 盒在 3DES、GOST、RIJNDAEL 中都有用到，3DES 中用到 6*4 规格的 S 盒，GOST 中用到 4*4 规格的 S 盒，RIJNDAEL 中用到 8*8 规格的 S 盒。本设计就是实现了三种规格的 S 盒变换，它能实现 8*8，6*4 和 4*4 三种规格 S 盒的任意布尔函数变化。

1、S 盒[4][26][28]的设计原理

S 盒的功能就是一种简单的“代替”操作。一个 n 输入、m 输出的 S-盒所实现的功能是从二元域F_2上的 n 维向量空间F_2^n到二元域F_2上的 m 维向量空间F_2^m的映射：$F_2^n \longrightarrow F_2^m$，我们称这个映射为 S 盒代替函数。构造 S 盒常用的方法有如下 3 种[23]：①随机选择；③人为构造；④数学方法构造。为保证 S 盒的密码强度，S 盒应越大越好。一般来讲，n 和 m 越大，随机性越好，S 盒的密码强度就越大。但 n 和 m 越大，S 盒的规模和可控编码的宽度也就越大。S 盒的随机性可以通过可控编码来实现，通过改变可控编码，一个 n*m 的 S 盒能够实现$F_2^n \longrightarrow F_2^m$的所有的代替函数。

输入为$x=\sum_{i=0}^{n-1} x_i 2^i = (x_{n-1}, x_{n-2}, \ldots\ldots x_1, x_0)_2$的 n*m S 盒布尔函数的通式[28]为

$$f(x)=\sum_{i=0}^{2^n-1} c_i x_{n-1}^{i_{n-1}} x_{n-2}^{i_{n-2}} \ldots\ldots x_2^{i_2} x_1^{i_1} x_0^{i_0} \qquad (4.1)$$

其中，系数$c_i \in \{0,1\}$，$i=\sum_{k=0}^{n-1} i_k 2^k = (i_{n-1}, i_{n-2}, \ldots\ldots i_1, i_0)_2$。通项记为$p_i(x)=\prod_{l=0}^{n-1} x_l^{i_l} = x_{n-1}^{i_{n-1}} x_{n-2}^{i_{n-2}} \ldots\ldots x_2^{i_2} x_1^{i_1} x_0^{i_0}$，则$f(x)=\sum_{i=0}^{2^n-1} c_i p_i(x)$。从（4.1）式可以看出，当输入数据后，$p_i(x)$作为最小项就确定下来了，对输出$f(x)$产生影响的只有最小项系数$c_i$，通过改变$c_i$就可以使 S 盒实现任意布尔函数的变换。

2、S 盒的设计方法

根据（4.1）式可以知道 S 盒代替变换实际上就是一组布尔逻辑函数，S 盒的输入 X 就是布尔逻辑函数的自变量，S 盒的输出 f(x)就是函数值。不同的加/解密算法所使用的 S 盒代替变换是不同的，为了 S 盒能够支持不同的加/解密算法，必须能够通过编程改变 S 盒模块实现的函数关系。在电路规模允许的条件下，我们应该使 S 盒模块实现的函数个数尽可能多，最好能够实现输入变量的任意的布尔函数。下面我们给出一个 n 输入 m 输出的可编程 S 盒的设计方法。该 S 盒能够实现 n 个输入变量的任意布尔函数[27]。

设 $x_1, x_2, \cdots, x_n$ 是 n 个布尔变量，$f_1(x_1, x_2, \cdots, x_n)$、$f_2(x_1, x_2, \cdots, x_n)$、…、$f_m(x_1, x_2, \cdots, x_n)$ 是 $x_1, x_2, \cdots, x_n$ 的 m 个布尔函数，则 $f_1(x_1, x_2, \cdots, x_n)$、$f_2(x_1, x_2, \cdots, x_n)$、…、$f_m(x_1, x_2, \cdots, x_n)$ 都可以表示为下列最小项之和的形式：

$$f_1(x_1, x_2, \cdots, x_n)=k_{11}(\overline{x_1x_2}\cdots\overline{x_n})+k_{12}(\overline{x_1x_2}\cdots x_n)+\cdots+k_{12^n}(x_1x_2\cdots x_n) \quad (4.2)$$

$$f_2(x_1, x_2, \cdots, x_n)=k_{21}(\overline{x_1x_2}\cdots\overline{x_n})+k_{22}(\overline{x_1x_2}\cdots x_n)+\cdots+k_{22^n}(x_1x_2\cdots x_n) \quad (4.3)$$

… … … … …

$$f_m(x_1, x_2, \cdots, x_n)=k_{m1}(\overline{x_1x_2}\cdots\overline{x_n})+k_{m2}(\overline{x_1x_2}\cdots x_n)+\cdots+k_{m2^n}(x_1x_2\cdots x_n) \quad (4.4)$$

其中，$\overline{x_1x_2}\cdots\overline{x_n}$、$\overline{x_1x_2}\cdots x_n$、$x_1x_2\cdots x_n$ 是 n 个变量的 2^n 个最小项，$k_{ij}\in\{0,1\}$，i=1，2，…，m，j=1，2，…2^n。

对于任意的布尔函数 $f_i(x_1, x_2, \cdots, x_n)$（$1\le i\le m$），其 2^n 个最小项（n 项之积）是固定不变的，其表达式的结构（2^n 项之和）也是不变的，其函数关系的改变完全依赖于最小项的系数（即控制编码）$k_i=(k_{i1},k_{i2},\ldots,k_{i2^n})$ $(1\le i\le m)$ 的改变。因此我们在设计逻辑电路的时候，应该把“n 项之积”和“2^n 项之和”作为固定的电路结构，而把控制编码所对应的电路结构作为可变结构。通过对 $k_i=(k_{i1}, k_{i2}, \cdots, k_{i2^n})$（$1\le i\le m$）赋以不同的值，就可以实现不同的布尔逻辑函数。该子模块的电路图如图 4.1 所示[27]。

对每个 $f_i(x_1, x_2, \cdots, x_n)$（$1\le i\le m$），我们通过改变控制编码能够实现 n 个变量的全部（共 2^{2^n} 个）布尔逻辑函数，因此上述电路能够实现任意的 n 输入、m 输出的函数：$f(x_1, x_2, \cdots, x_n)=(f_1(x_1, x_2, \cdots, x_n), f_2(x_1, x_2, \cdots, x_n), \cdots f_m(x_1, x_2, \cdots, x_n))$，其个数为 2^{m2^n}。

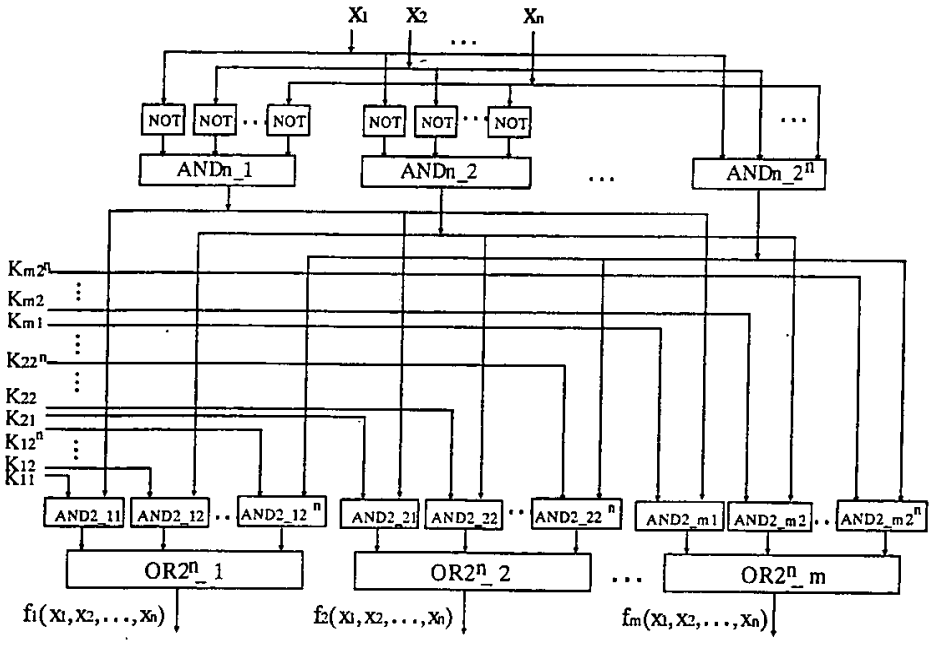

图 4.1　n 输入 m 输出的可编程 S 盒的电路图

当取 n=8、m=8，上述电路就能实现 1 组 8*8 的 S 盒代替变换。通过电路设计，这套电路资源可以兼容 4 组 6*4 的 S 盒代替变换，和 16 组 4*4 的 S 盒代替变换。

3、功能描述

我们用 Verilog 硬件描述语言在 ModelSim 上实现了 1 组 8*8 的 S 盒，并且兼容 4 组 6*4 和 16 组 4*4 的 S 盒。

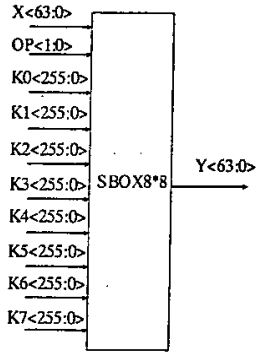

图 4.2　SBOX8*8 外观图

S 盒的外观电路如图 4.2 所示。X 是输入，64 位；OP 是功能选择输入，2 位；K0, K1, K2, K3, K4, K5, K6, K7 是控制编码，都分别是 256 位；Y 是输出，64 位。当 OP=00 时，电路实现 1 组 8*8 的 S 盒，输入的 8 位数据是 X 的低 8 位，输出对应 Y 的低 8 位；当 OP=01 时，电路能同时实现 4 组 6*4 的 S 盒，4 组 6 位输入从 X 的低位依次排列，输出对应输出 Y 的 4 组 4 位输出。当 OP=10 时，电路能同时实现 16 组 4*4 的 S 盒，14 组 4 位输入从 X 的低位依次排列，输出对应输出 Y 的 16 组 4 位输出。通过改变控制编码 K1, K2……K7，S 盒就可以实现 8*8，6*4，4*4 规格的任意布尔函数的变化，实现不同算法中要求不同的 S 盒功能。用表格说明更能清晰的看出它的功能和实现方式，如表 4.1。

OP	规格	输入 X	静态编码 K0, K1, K2, K3, K4, K5, K6, K7 的使用	输出 Y
00	8*8	X<7:0>	K0<255:0>, K1<255:0>, K2<255:0>, K3<255:0>, K4<255:0>, K5<255:0>, K6<255:0>, K7<255:0>	Y<7:0>
01	第 1 组 6*4	X<5:0>	K0<63:0>, K1<63:0>, K2<63:0>, K3<63:0>	Y<3:0>
	第 2 组 6*4	X<11:6>	K0<127:64>, K1<127:64>, K2<127:64>, K3<127:64>	Y<7:4>
	第 3 组 6*4	X<17:12>	K0<191:128>, K1<191:128>, 2<191:128>, K3<191:128>	Y<11:8>
	第 4 组 6*4	X<23:18>	K0<255:192>, K1<255:192>, 2<255:192>, K3<255:192>	Y<15:12>
10	第 1 组 4*4	X<3:0>	K0<15:0>, K1<15:0>, K2<15:0>, K3<15:0>	Y<3:0>
	第 2 组 4*4	X<7:4>	K0<31:16>, K1<31:16>, K2<31:16>, K3<31:16>	Y<7:4>
	第 3 组 4*4	X<11:8>	K0<47:32>, K1<47:32>, K2<47:32>, K3<47:32>	Y<11:8>
	第 4 组 4*4	X<15:12>	K0<63:48>, K1<63:48>, K2<63:48>, K3<63:48>	Y<15:12>
	第 5 组 4*4	X<19:16>	K0<79:64>, K1<79:64>, K2<79:64>, K3<79:64>	Y<19:16>
	第 6 组 4*4	X<23:20>	K0<95:80>, K1<95:80>, K2<95:80>, K3<95:80>	Y<23:20>
	第 7 组 4*4	X<27:24>	K0<111:96>, K1<111:96>, K2<111:96>, K3<111:96>	Y<27:24>

	第 8 组 4*4	X<31:28>	K0<127:112>, K1<127:112>, K2<127:112>, K3<127:112>	Y<31:28>
	第 9 组 4*4	X<35:32>	K0<143:128>, K1<143:128>, K2<143:128>, K3<143:128>	Y<35:32>
	第 10 组 4*4	X<39:36>	K0<159:144>, K1<159:144>, K2<159:144>, K3<159:144>	Y<39:36>
	第 11 组 4*4	X<43:40>	K0<175:160>, K1<175:160>, K2<175:160>, K3<175:160>	Y<43:40>
	第 12 组 4*4	X<47:44>	K0<191:176>, K1<191:176>, K2<191:176>, K3<191:176>	Y<47:44>
	第 13 组 4*4	X<51:48>	K0<207:192>, K1<207:192>, K2<207:192>, K3<207:192>	Y<51:48>
	第 14 组 4*4	X<55:52>	K0<223:208>, K1<223:208>, K2<223:208>, K3<223:208>	Y<55:52>
	第 15 组 4*4	X<59:56>	K0<239:224>, K1<239:224>, K2<239:224>, K3<239:224>	Y<59:56>
	第 16 组 4*4	X<63:60>	K0<255:240>, K1<255:240>, K2<255:240>, K3<255:240>	Y<63:60>
11 及其它	无	X<63:0>	无	Y<63:0>全是 0

表 4.1　8*8 S 盒的功能说明

4、实现方法

S 盒电路是一个组合电路。根据在第 3 部分(S 盒的硬件实现方法)中的介绍，1 组 8*8 的 S 盒有 256 个最小项，而 1 组 6*4 的 S 盒有 64 个最小项，因此 1 组 8*8 S 盒的资源可以实现 4 组 6*4 的 S 盒，这就是设计中的资源复用，即一套资源实现多种规格的 S 盒功能。图 4.3 给出了具体电路实现单元。电路实现中，共分为 4 组，1 组有 64 个最小项，由 64 个相同的电路单元实现。op=1 时，实现 1 组 8*8 S 盒的最小项构造，OP=0 时，实现 4 组 6*4 S 盒的最小项构造，之后将最小项与静态编码相应位进行与操作，再将结果按位进行或操作，就得到输出 Y。

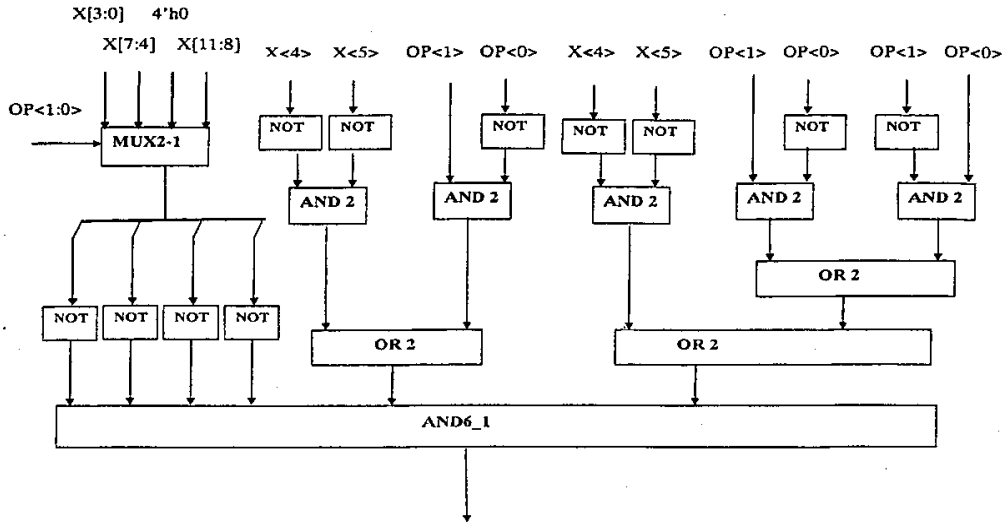

图 4.3 电路实现单元

5、接口说明

接口信号比较多，输入数据，输入静态编码，输入控制信号，输出数据，详见表 4.2。

名称	类型	描述	备注
X	input	64 位输入数据	
OP	input	2 位输入数据	选择实现 S 盒的规格
K0	input	256 位输入数据	静态编码
K1	input	256 位输入数据	静态编码
K2	input	256 位输入数据	静态编码
K3	input	256 位输入数据	静态编码
K4	input	256 位输入数据	静态编码
K5	input	256 位输入数据	静态编码
K6	input	256 位输入数据	静态编码
K7	input	256 位输入数据	静态编码
Y	output	64 位输出数据	

表 4.2 SBOX88 模块接口

6、设计与仿真

S 盒的功能仿真在 Modelsim 上进行，在仿真过程中，S 盒的输入数据通过激励赋值，分别针对 8*8 规格、6*4 规格和 4*4 给定数据，且 OP=00 时，预期要实现 1 组 8*8 的 S 盒变换，OP=01 时，预期要实现 4 组 6*4 的 S 盒变换，OP=11 时，预期要实现 16 组 4*4 的 S 盒变换，最终得到的仿真结果与预期结果相吻合。仿真图如下：

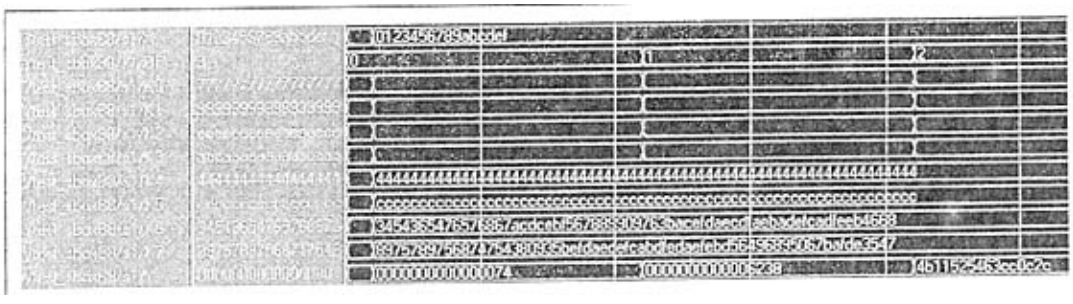

图 4.4　S 盒功能仿真图

4.2 模加单元设计

1、功能介绍

模加在 IDEA、GOST、RC6 中都有用到，IDEA 中用到模 2^{16} 加运算，GOST 和 RC6 中都用到模 2^{32} 加运算。本设计就是同一个模块中兼容实现了两种规格的模加运算，它们是模 2^{16} 加运算和模 2^{32} 加运算。模加运算可控接点图如下：

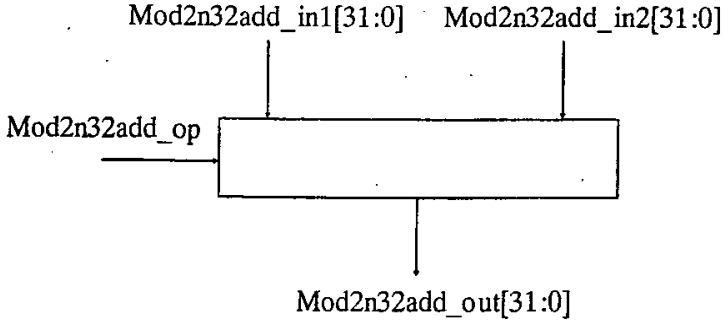

图 4.5　模加运算可控接点图

Mod2n32add_op 是功能控制节点，Mod2n32add_in1 和 Mod2n32add_in2 是数据来源，

Mod2n32add_out 是数据结果输出。当 Mod2n32add_op＝1 时，模块执行模 2^{32} 加运算：Mod2n32add_out＝（Mod2n32add_in1＋Mod2n32add_in2） mod 2^{32}；当 Mod2n32add_op＝0 时，模块执行两组模 2^{16} 加运算：Mod2n32add_out[15:0] ＝（Mod2n32add_in1[15:0]＋Mod2n32add_in2[15:0]） mod 2^{16}，Mod2n32add_out[31:16]＝（Mod2n32add_in1[31:16]＋Mod2n32add_in2[31:16]） mod 2^{16}。

2、设计及仿真

模加运算一般被称为“时钟运算”，也就是求余数运算。模加用加法器来实现.在本次的设计中，两个加数全部是 32 位或 16 位的数据，模数分别是 2^{32} 或 2^{16}。在实现的过程中只有取数据的低 32 位或 16 位即可。下面是模加的功能仿真图：

图 4.6 模加功能仿真图

4.3 模加逆单元设计

1、功能说明

模加逆在 IDEA 中用到，并且还是模 2^{16} 加逆运算，但我们为了系统有更好的扩展性，我们还是在本模块的设计中同时兼容了模 2^{16} 加逆和模 2^{32} 加逆运算，模加逆运算可控接点图如下：

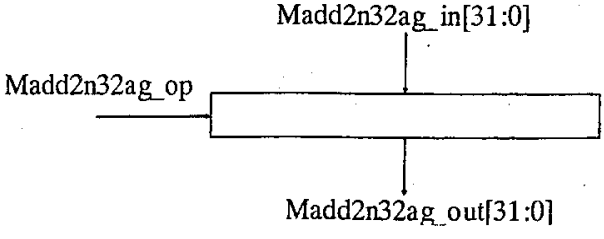

图 4.7 模加逆运算可控接点图

Madd2n32ag_op 是功能控制节点，Madd2n32ag_in 是数据来源，Madd2n32ag_out 是数据结

果输出。当 Madd2n32ag_op=1 时，模块执行模 2^{32} 加逆运算：(Madd2n32ag_out+Madd2n32ag_in) mod 2^{32} = 1 mod 2^{32}；当 Madd2n32ag_op = 0 时，模块执行两组模 2^{16} 加逆运算：(Madd2n32ag_in[15:0] + Madd2n32ag_out[15:0]) mod 2^{16} = 1 mod 2^{16}，(Madd2n32ag_in[31:16]+Madd2n32ag_out[31:16]) mod 2^{16}=1 mod 2^{16}。

2、设计及仿真

模加逆的实现就是在模加的基础上进行实现。功能仿真在 Modelsim 上进行，仿真图如下：

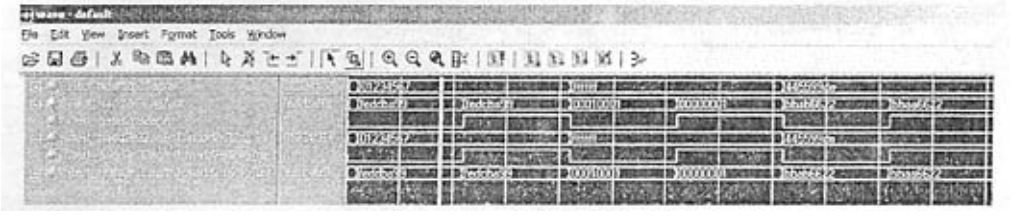

图 4.8 模加逆功能仿真图

4.4 模减单元设计

1、功能说明

本课题中模减运算在 RC6 中用到，并且还是模 2^{32} 减运算，但我们为了系统有更好的扩展性，我们还是在本模块的设计中同时兼容了模 2^{32} 减和模 2^{16} 减运算，模减运算可控接点图如下：

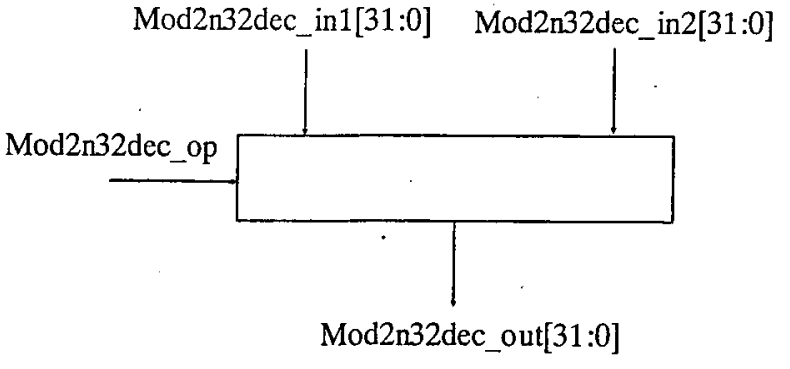

图 4.9 模减运算可控接点图

Mod2n32dec_op 是功能控制节点，Mod2n32dec_in1 和 Mod2n32dec_in2 是数据来源，Mod2n32dec_out 是数据结果输出。当 Mod2n32dec_op = 1 时，模块执行模 2^{32} 加运算：

Mod2n32dec_out=（Mod2n32dec_in1+Mod2n32dec_in2） mod 2^{32}；当Mod2n32dec_op=0时，模块执行两组模2^{16}加运算：Mod2n32dec_out[15:0]=（Mod2n32dec_in1[15:0]+Mod2n32dec_in2[15:0]） mod 2^{16}，Mod2n32dec_out[31:16]=（Mod2n32dec_in1[31:16]+Mod2n32dec_in2[31:16]） mod 2^{16}。其中减法器的实现方法在[31]和[32]中都有讲述。

2、设计与仿真

功能仿真在Modelsim上进行，仿真图如下：

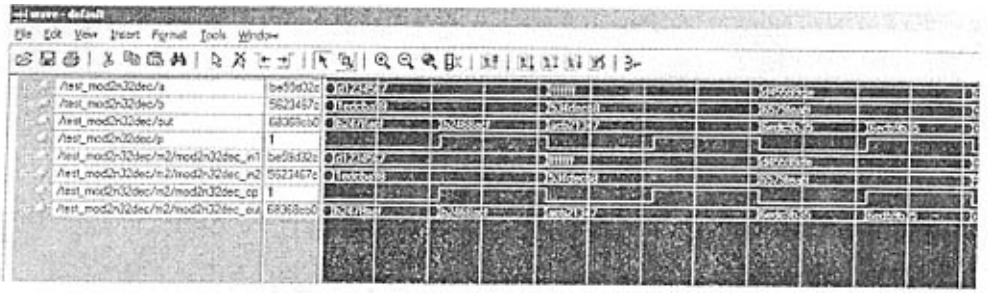

图 4.10 模减功能仿真图

4.5 模2^{32}乘设计

1、功能说明

本课题中模2^{32}乘运算在RC6中用到，但其他可能扩展的密码算法也会应用的到，我们还是作为一个IP来实现，并且也兼容实现模2^{16}乘运算。模2^{32}乘运算可控接点图如下：

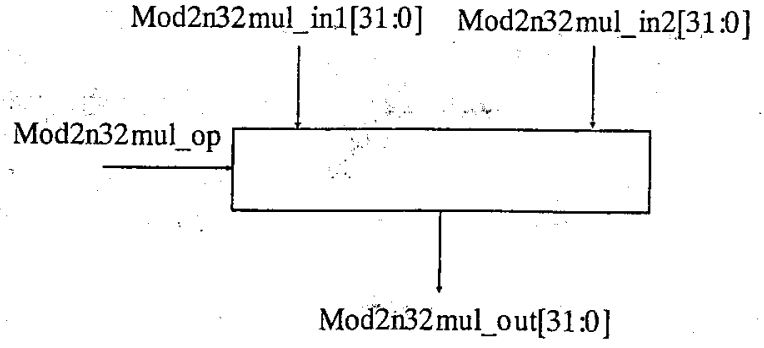

图 4.12 模2^{32}乘运算可控接点图

Mod2n32mul_op 是功能控制节点，Mod2n32mul_in1 和 Mod2n32mul_in2 是数据来源，Mod2n32mul_out 是数据结果输出。当 Mod2n32mul_op＝1 时，模块执行模 2^{32} 乘运算：Mod2n32mul_out＝（Mod2n32mul_in1×Mod2n32mul_in2） mod 2^{32}；当Mod2n32mul_op=0 时，模块执行两组模 2^{16} 乘运算：Mod2n32mul_out[15:0] ＝（Mod2n32mul_in1[15:0] × Mod2n32mul_in2[15:0]） mod 2^{16}，Mod2n32mul_out[31:16]＝（Mod2n32mul_in1[31:16]× Mod2n32mul_in2[31:16]） mod 2^{16}。其中乘法器的实现方法在[32]和[33]中都有讲述。

2、设计与仿真

在模乘的实现过程中，乘法器用加法进行实现，其设计思想是将模乘转换为一系列加法，算法中为使最终结果 C 小于模数 N，每次计算都需对累加中间结果 C 进行模简化，Blakley[42]算法如下：

```
Blakley(A,B,N)
1:C=0
2:For i=0 to k-1
3:C=2C+Ak-i-1*B
4:if(C>=N) C=C-N
5: if(C>=N) C=C-N
6:return C
```

Blakley 算法需 k 次 k-Bit 移位、$\frac{k}{2}$ 次 k-Bit 加法、2k 次 k-Bit 次比较和平均 k 次 k-Bit 减法，如果数据位数过多，实现的效率不够理想。在本模块的设计中，只是实现 32 位和 16 位的数据模乘，并且模数也是 2^{32} 和 2^{16} 比较特殊的数据。在模块实现中，我们对算法进行改进，利用硬件实现过程的位数特点，每次加法后，只取低 32 位或低 16 位的数据，即为模 2^{32} 或 2^{16} 乘结果，这样就省去了减法的过程。模块设计完成在 Modelsim 上功能仿真通过，仿真图如下：

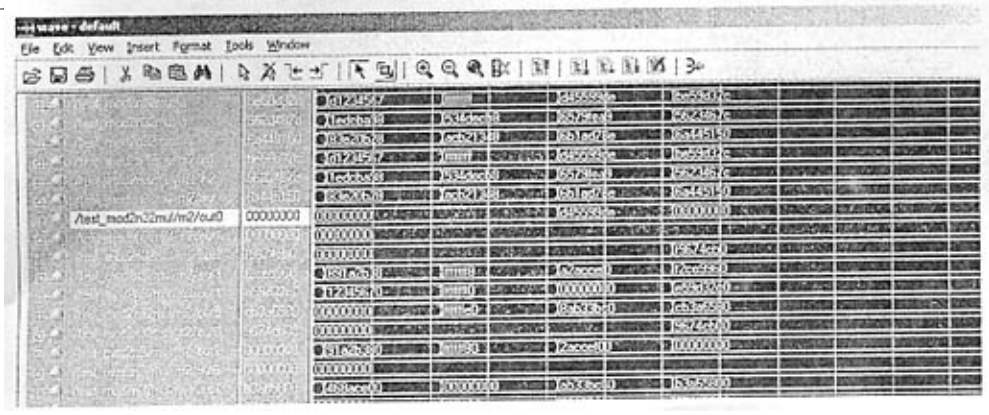

图 4.13 模乘功能仿真图

4.6 模 $2^{16}+1$ 乘设计

1、功能说明

模 $2^{16}+1$ 乘：(a×b)mod （$2^{16}+1$）其实就是求 a×b 的结果除以 $2^{16}+1$ 的余数。模 $2^{16}+1$ 乘运算可控接点图如下：

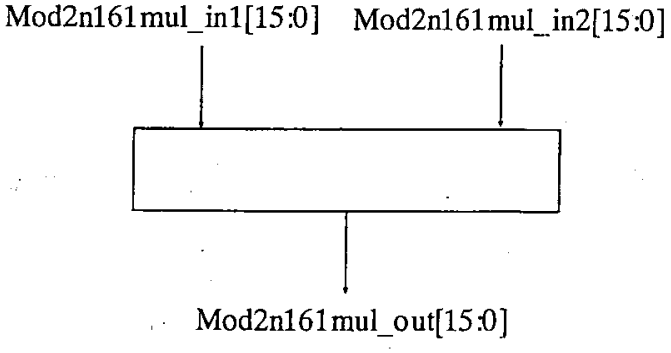

图 4.14 模 $2^{16}+1$ 乘运算可控接点图

Mod2n161mul_in1 和 Mod2n161mul_in2 是数据来源，Mod2n161mul_out 是数据结果输出。模块执行模 $2^{16}+1$ 乘运算：Mod2n161mul_out[15:0] = （Mod2n161mul_in1[15:0] × Mod2n161mul_in2[15:0]） mod （$2^{16}+1$）。

2、设计与仿真

实现方案有两种：第一种是经过数学推导得到(a×b) mod （$2^{16}+1$）= (a×b) mod 2^{16}

－（a×b）div 2^{16}。第二种是根据其定义一步步实现，即 a×b 再除以 $2^{16}+1$ 得到其余数就是结果。两种方案都要实现一个 16 位的乘法器，和一个除法器。前一种方案实现起来比较容易，并且延时只有 9.417ns，但这种方法在 a=1000, b=2000 等的特殊情况时得不到正确结果。而第二种方案是根据其定义实现，在任何情况下都能得到正确结果，只是延时大些，足有 35.870ns。在本设计中我们选择第二种方案，保证结果的正确性。

在模块的实现过程中，采取加法器和减法器来实现模乘，由于模数是 $2^{16}+1$ 比较特殊的数据，在实现过程中采取特殊的处理方法。第一步通过加法实现乘法 a×b, 实现方法上面已经介绍过；第二步用减法器实现（a×b）mod （$2^{16}+1$）。

除法可以由多个减法来实现，从左到右一次取被除数的各位，组成部分被除数。判断该部分被除数是否够除数减。如果不够，则在该位上商 0，同时补充被除数的下一位，组成更大的数；如果够减，则该位商 1，在部分被除数中减去除数，然后补充被除数的下一位。直至被除数的每一个位都使用了为止。

由此可以看出在除法进行过程中，存在着一个怎样判断部分被除数是否够除数减的问题。有两种方式来解决这个问题。第一种方式是使用专门的数据比较器来比较部分被除数是否比除数大。然后，根据比较的结果采用相应操作。第二种方式是不管部分被除数是否够除数减，先用部分被除法减去余数，然后判断产生的差是否为负数。如果为负数，则表明不够减，否则，

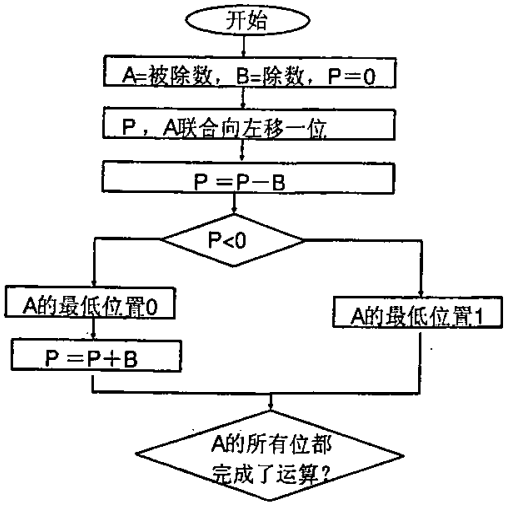

图 4.15　恢复余数法运算流程图

表明够减。当够减时，这种方式就同时判断了是否够减和完成了减法操作，一举两得。与之相

比，第一种方式则浪费了专门的数据比较电路，同时，也加大了运算的延迟，降低了运算的速度。在第二种方式中，当完成了减法操作之后，如果发现不够减，就存在是否要把除数加回到产生的差，以恢复原来的部分被除数上。这个问题也有两种方式来解决。一种方式是加回，这种方式称为恢复余数法。另一种方式是不加回，把产生的差留给下一次运算时再处理，这种方式称为不恢复余数法[32] [33]。由于在本模块的设计中是求余数，我们采用恢复余数法来实现。下面介绍恢复余数法的流程。恢复余数法的运算过程如图 4.15 的运算流程所示。表 4.3 为恢复余数法的算法演示，表中为 1101/0011。

表 4.3 恢复余数法算法演示

运算次数	余数 P	被除数 A	商 Q	操作说明
1	0000 0001 −0011 1110 +0011 0001	1011 011	0	PA 左移一位 P=P-B P<0, 商 0 恢复余数
2	0010 −0011 1111 +0011 0010	1	00	PA 左移一位 P=P-B P<0, 商 0 恢复余数
3	0101 −0011 0010	1	001	PA 左移一位 P=P-B P>=0, 商 1 恢复余数
4	0101 −0011 0010		0011	PA 左移一位 P=P-B P>=0, 商 1 恢复余数

本模块利用这种方法，把数据扩展到 16 位进行实现，并在 Modelsim 上功能仿真通过，仿

真图如下：

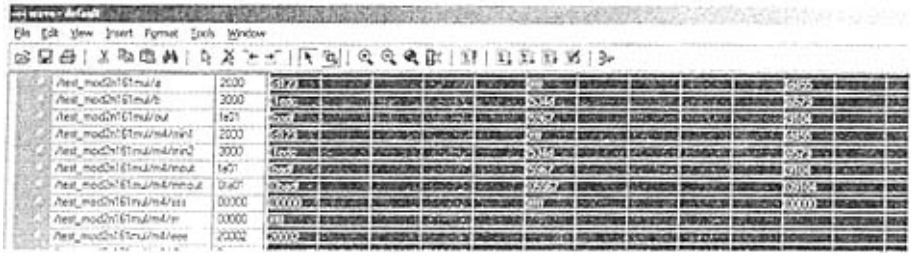

图 4.16　模 $2^{16}+1$ 乘功能仿真图

4.7 模 $2^{16}+1$ 乘逆设计

1、设计与功能介绍

模 $2^{16}+1$ 乘逆就是求数据的模 $2^{16}+1$ 的乘法逆元。例如求 a 的模 $2^{16}+1$ 的乘法逆元，那将表示为 $1=(a\times x) \bmod 2^{16}+1$，也可写作 $a^{-1}\equiv x(\bmod 2^{16}+1)$。其中 x 就是 a 的模 $2^{16}+1$ 的乘法逆元。在本设计中用欧几里德算法实现模 $2^{16}+1$ 乘逆。模 $2^{16}+1$ 乘逆运算可控接点图如下：

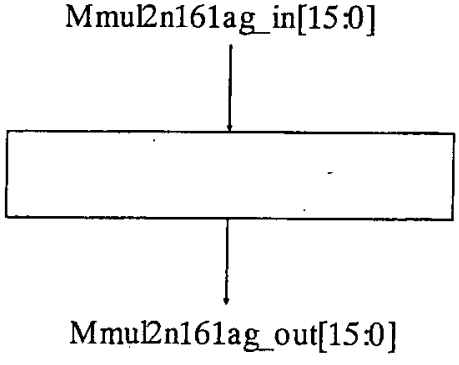

图 4.17　模 $2^{16}+1$ 乘逆运算可控接点图

Mmul2n161ag_in 是数据来源，Mmul2n161ag _out 是数据结果输出。模块执行模 $2^{16}+1$ 乘逆运算：1＝（Mmul2n161ag _in×Mmul2n161ag _out） mod （$2^{16}+1$）。

2、设计与仿真

IDEA中模乘逆的运算法则是：对于16位数X，求一16位数Y，使(X·Y)mod($2^{16}+1$)＝1。由

于模乘逆运算只用于密钥扩展，不要求实时高速，故可采用Eucid算法，分多时钟周期实现。用伪语言表示如下：

```
If(x< 2)
Return X;
Else
T1:=65537 div X;
Y:=65537 rood X;
If(y=1)
Return (65537 sub t1);
Else
TO:=1;
While(y!=1)
{q:=X div Y;
X:=X mod Y;
To:=q mul t1 add to;
if(x =1)
return t0 ;
else
q : =Y div X;
Y : = Y mod X;
t1: =q mul t0 add t1; )
return (65537 sub t1);
```

4.8 逻辑单元设计

1、功能说明

逻辑异或运算在所有的对称算法中都会用到，在本课题中的五种算法也不例外。在 IDEA 算法中用到 16 位异或运算，GOST 算法中用到 32 位异或运算，RC6 算法中用到 32 位异或运算，RIJNDAEL 中用到 32 位异或运算，3DES 中用到 48 位和 32 位异或。那么我们在逻辑单元的设计中是实现 64 位数据的逻辑处理，并且兼容与、或、非、异或 4 中数据处理。逻辑单元可控接点

如图 4.18：

Log64_op 是功能控制节点，Log64_in1 和 Log64_in2 是数据来源，Log64_out 是数据结果输出。当 Log64_op＝11 时，模块执行异或运算：Log64_out＝Log64_in1 ⊕Log64_in2；当 Log64_op＝10 时，模块执行非运算：Log64_out＝～Log64_in1；当 Log64_op＝01 时，模块执行或运算：Log64_out＝Log64_in1^Log64_in2；当 Log64_op＝00 时，模块执行与运算：Log64_out＝Log64_in1&Log64_in2。要进行 48 位、32 位或 16 位的数据处理时，只要在两个数据来源 Log64_in1 和 Log64_in2 以低位输入需要处理的数据，在结果输出端 Log64_out 获得对应位的结果即可。

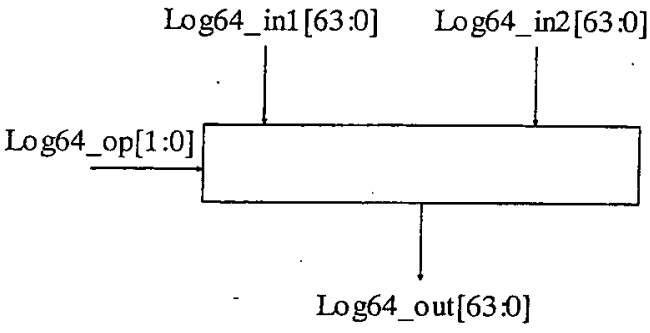

图 4.18　逻辑单元可控接点图

2、设计与仿真

功能仿真在 Modelsim 上进行，仿真图如下：

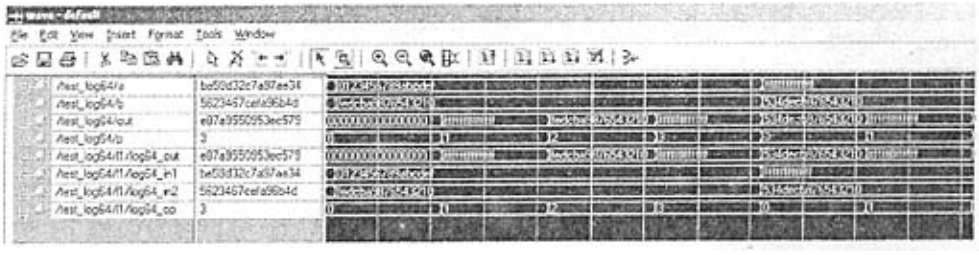

图 4.19　逻辑单元功能仿真图

4.9　置换模块设计

1、功能说明

置换模块在对称算法中应用也很广，在本课题的五种算法中有 3DES 用到，并且有多种规格，置换 64*64、置换 32*32、压缩置换 64*48、扩展置换 32*48。在置换模块的设计中，我们用实现置换 128*128 规格的电路同时兼容置换 64*64、置换 32*32、压缩置换 64*48、扩展置换 32*48

四种置换规格。不同规格置换的实现是通过静态编码的变换来实现的。置换模块的可控接点图如下：

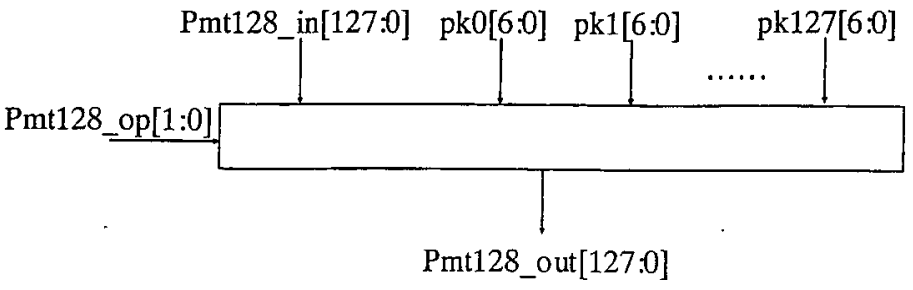

图 4.20 置换模块可控接点图

Pmt128_in 是数据来源，pk0、pk1、……pk127 是静态编码，Pmt128_op 是功能控制节点，通过功能控制节点和静态编码的变化，置换模块可以实现多种规格的置换功能，并且是输入数据任意位的置换。当 Pmt128_op＝11 时，实现两组扩展置换 32*48；当 Pmt128_op＝10 时，实现四组置换 32*32；当 Pmt128_op＝01 时，实现两组置换 64*64 或压缩置换 64*48；当 Pmt128_op＝00 时，实现一组置换 128*128；

2、设计与仿真

置换模块的实现单元如图 4.21：

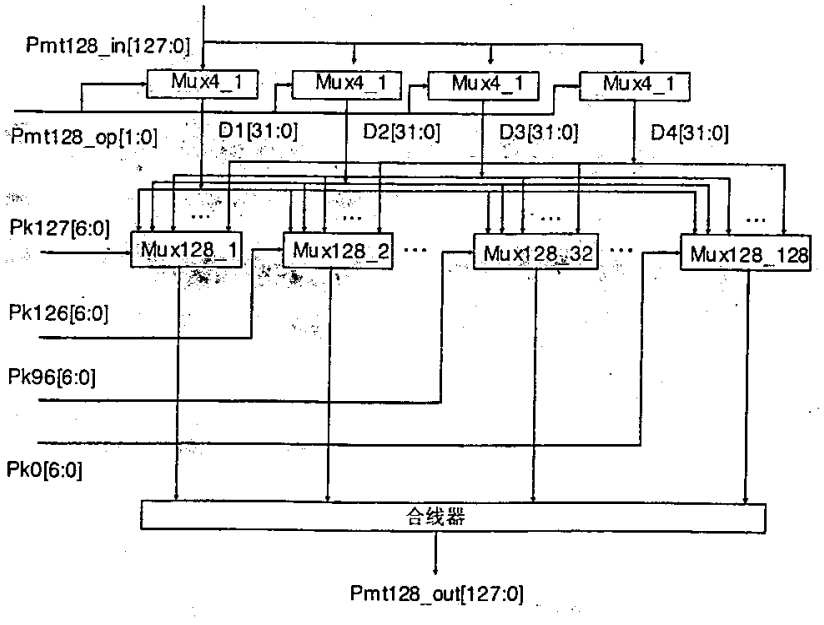

图 4.21 置换模块的实现单元

随着 pk0、pk1、……pk127 的变化，置换单元可以实现输入数据的任意置换，而 Pmt128_op 的变换，可以实现不同规格和个数的置换。输入数据和输出数据遵从低位原则，即原始数据从 Pmt128_in 低位开始输入，置换结果从对应输出端口取数。

置换单元的功能仿真在 Modelsim 上进行，仿真图如下：

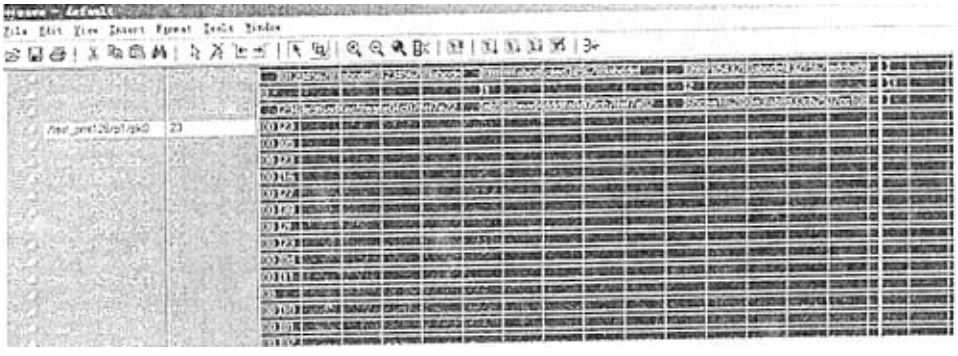

图 4.22　置换单元功能仿真图

4.10 移位单元设计

1、功能说明

对称算法中几乎一半的算法都用到移位单元，在本课题中的五种算法都用到移位单元，共有三种规格，分别是：28 位的循环移位单元、32 位循环移位单元和 128 位循环移位单元。我们在设计本单元时，不单单实现循环移位的功能，还实现逻辑移位的功能。128 位移位单元的可控接点图如下：

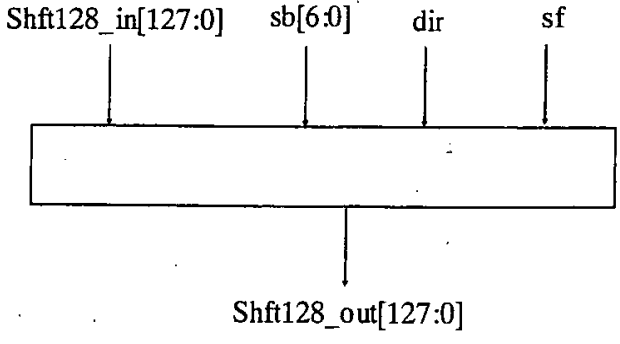

图 4.23　移位单元可控接点图

Shft128_in 是数据来源，Shft128_op 是功能控制节点，sb 是移位位数数据来源，dir 是移位方向控制节点，sf 是移位方式控制节点，Shft128_out 是数据结果输出。sf＝0 时，模块循环

移位；sf=1 时，模块逻辑移位。dir=0 时，模块执行左移操作；dir=1 时，模块执行右移操作。

2、设计与仿真

移位单元分为两部分实现，一部分是通过错位排列法实现循环移位；另一部分是过滤电路，用来纠正循环移位电路，从而实现逻辑移位。而循环右移可以通过循环左移来实现。循环左移与循环右移关系如表 4.4：

表 4.4 循环左移与循环右移关系

移位位数（sb[6:0]）	循环左移位数	循环右移位数
0000000	1	127(128-sb+1)
0000001	2	126
0000010	3	125
…	…	…
i	i+1	(128-i+1)
…	…	…
1111111	128	0

移位单元实现电路如图 4.24

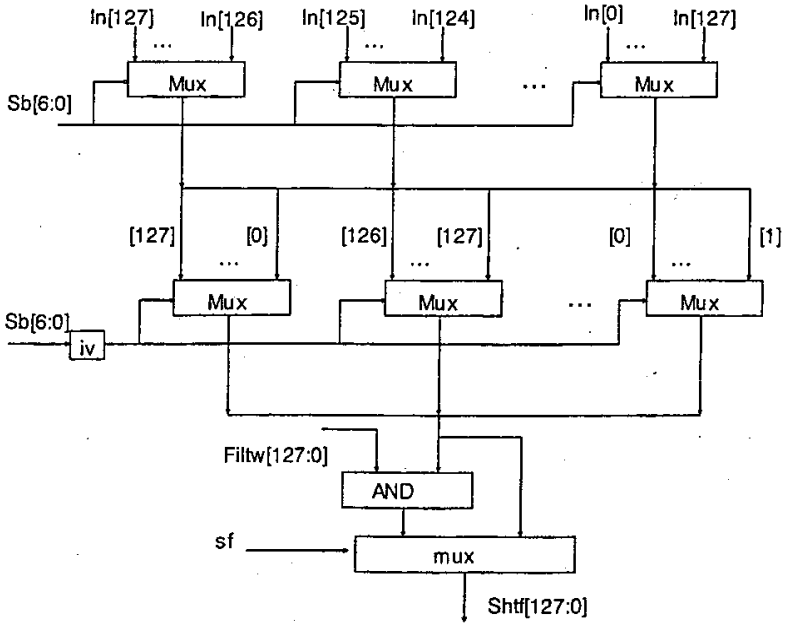

图 4.24 移位单元实现电路

其中 Filtw 是过滤电路的输出，它的实现电路如图 4.25。在逻辑移位中，逻辑左移时右边相应位补零，逻辑右移时左边相应位补零，过滤电路就是产生过滤字，纠正循环电路，达到逻辑移位的效果。

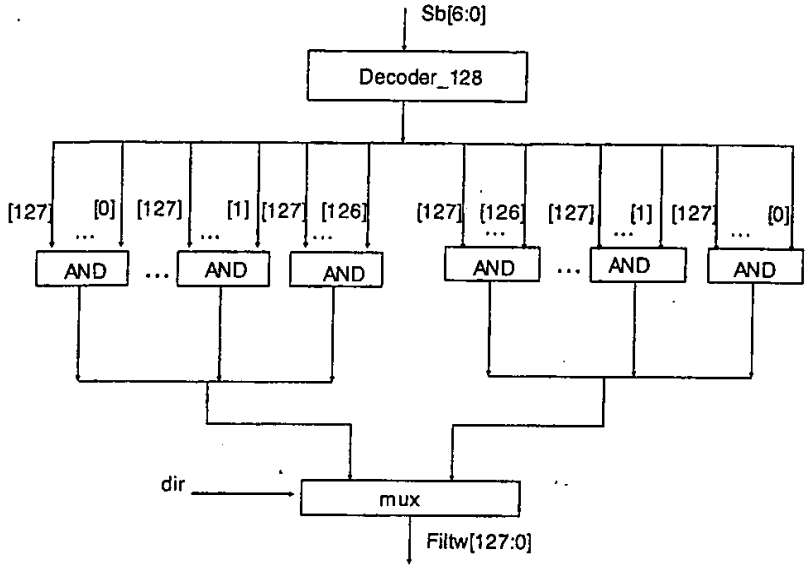

图 4.25　过滤电路

移位单元的功能仿真在 Modelsim 上进行，仿真图如下：

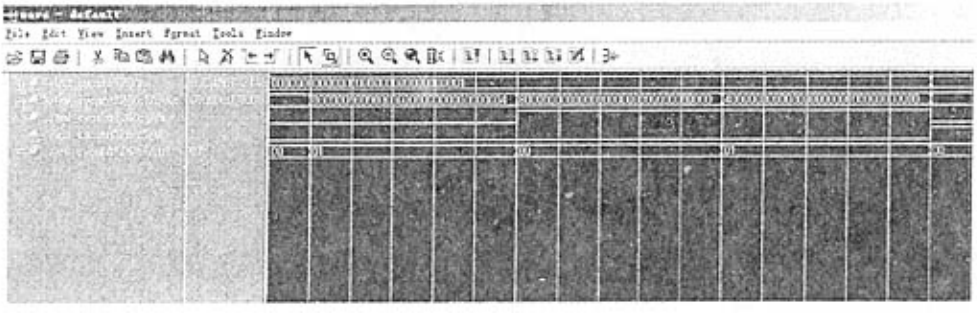

图 4.26　移位单元功能仿真图

4.11 模 8 多项式模块设计

模 8 多项式模块在 RIJNDAEL 算法中应用到，我们在设计时将模块设计成可扩展的模多项式通用模块。

1、模 8 多项式模块功能说明

模 8 次多项式模块可控接点如下图表示：

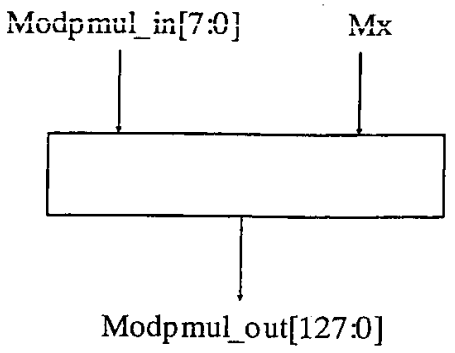

图 4.27 模 8 次多项式模块可控接点图

本课题系统中有 4 个模 8 次多项式运算单元，因此可以同时进行 4 个模 8 次多项式运算操作。Modpmul_in 是数据来源输入，Modpmul_in 是数据结果输出，Mx 是模数多项式，可置为静态可控接点，Mx 可以改变，实现不同模 8 次多项式的运算。在 RIJNDAEL 算法中 Mx 赋值为 11B。

2、模 8 多项式模块实现方法

$GF(2^8)$域 8 次多项式乘法算法原理：[4, 7, 14, 35]

设 a(x)和 b(x)分别为 $GF(2^8)$ 域 7 次多项式，则：

$c(x)=(a(x)\times b(x))\bmod m(x)=(c14x^{14}+c13x^{13}+c12x^{12}+c11x^{11}+c10x^{10}+c9x^9+c8x^8+c7x^7+c6x^6+c5x^5+c4x^4+c3x^3+c2x^2+c1x+c0) \bmod m(x)$。在 RIJNDAEL 中多项式 $m(x)=x^8+x^4+x^3+x+1$

$GF(2^8)$域 8 次多项式乘法原理分析：

$a(x)\ b(x) = a(x)\ (b7x^7 + b6x^6 + b5x^5 + b4x^4 + b3x^3 + b2x^2 + b1x + b0)$

$= b7x^7\ a(x) + b6x^6\ a(x) + b5x^5\ a(x) + b4x^4\ a(x) + b3x^3\ a(x) + b2x^2\ a(x) + b1x\ a(x) + b0a(x) =$

$b7x(x(x(x(x(x(x\ a(x)))))))) + b6x(x(x(x(x(x\ a(x)))))) + \cdots + b1x\ a(x) + b0\ a(x)$

对于任意 1 个 x a(x)，若 a7=0，$x\ a(x)=a6x^7 + a5x^6 + a4x^5 + a3x^4 + a2x^3 + a1x^2 + a0x$，即逻辑左移 1 位。若 a7=1，$x\ a(x)=(a6x^7 + a5x^6 + a4x^5 + a3x^4 + a2x^3 + a1x^2 + a0x)$，mod m(x)，即逻辑左移 1 位后，与模数按位（逐项次系数）异或。

$GF(2^8)$域 8 次多项式乘法操作步骤：

将 a(x)逻辑左移 1 位，通过 a7 判断 x a(x)结果为左移 1 位，或异或 M(x)，保留该结果；将 x a(x)结果逻辑左移 1 位，通过移出位判断 x (x a(x))结果为左移 1 位，或异或 M(x)，保留该结果，即 x^2 a(x)结果；如此循环 7 次，分别算得 x (x (x a(x)))、x (x (x (x a(x))))、x (x (x (x (x a(x)))))、x (x (x (x (x (x a(x))))))、x (x (x (x (x (x (x a(x)))))))的结果，

即 x^3 a(x)、x^4 a(x)、x^5 a(x)、x^6 a(x)、x^7 a(x)的结果；将 a(x)、x a(x)、x^2 a(x)、x^3 a(x)、x^4 a(x)、x^5 a(x)、x^6 a(x)、x^7 a(x)分别与 b0、b1、b2、b3、b4、b5、b6、b7 位扩展的 8 位数（即为 0 时，得 00H；为 1 时，得 FFH）相与，可知 a(x)、xa(x)、x^2a(x)、x^3a(x)、x^4a(x)、x^5a(x)、x^6a(x)、x^7a(x)各次项系数是否存在，即：b7x(x(x(x(x(x(xa(x)))))))、b6x(x(x(x(x(xa(x)))))))、…、b1x a(x)、b0a(x)的值；将上述 8 个与操作结果按位异或，即为如下 8 个多项式相加的结果：b7x(x(x(x(x(x(xa(x)))))))) + b6x(x(x(x(x(x a(x)))))))+ …+ b1xa(x)+ b0a(x)。

完成 xa(x)逻辑模型设计：

a(x)多项式的系数左移一位，构成新的多项式结果；（将系数的最高位链接选择控制，次高位之后的 7 位系数补 0，形成输出的 8 位系数结果）；a(x)多项式的系数左移一位，构成新的多项式结果与 m(x)多项式系数异或计算的结果；a(x)多项式的系数最高位选择控制结果；为 0 时，选择 a(x)多项式的系数左移一位的结果；为 1 时，选择 a(x)多项式的系数左移一位的结果与 m(x)多项式系数异或计算的结果；

完成 x^7a(x)逻辑模型设计：

将 xa(x)逻辑模型的计算结果串行输入与输出链接，形成 7 组（7 次多项式）xa(x)逻辑模型的计算结构，得到 7 组计算结果；加上 a(x)原始输入的系数，共计 8 组数据结果；

判定保留项及结果输出：

扩展 b(x)系数位，判定保留项（按位扩展和输出项按位与计算）；将保留项的输出结果 8 组，组组按位异或，输出结果即为（a(x)×b(x)）mod m(x)；

4.12 寄存器堆设计

在算法系统中，寄存器堆的作用是保存工作子密钥和中间计算结果。接下来我们就讨论在可重组算法逻辑系统的寄存器堆中的寄存器的位长、寄存器堆所包括的寄存器的数量、寄存器堆的输入、输出端口数、以及输入、输出端口的数据来源数等。

3DES 算法的工作子密钥是 48 位的，RIJNDAEL 算法的工作子密钥是 32 位的，IDEA 算法的工作子密钥是 16 位的，GOST 算法的工作子密钥是 32 位的，IDEA 算法的工作子密钥是 16 位，RC6 算法的工作子密钥是 32 位的，另外，系统的重组元素的输入、输出数据的宽度主要有 16 位、32 位、64 位、128 位等几种类型，因为 32、48、64、128 均是 16 的倍数，因此我们将寄存器堆中的寄存器的位长确定为 16 位，这样有利于提高寄存器堆的资源利用率。

RIJNDAEL 算法有 60 个 32 位的工作子密钥，需要用 120 个 16 位的寄存器保存，3DES 算法有 48 个 48 位的工作子密钥，需要用 96 个 16 位的寄存器保存，IDEA 算法有 88 个 16 位的工作子密钥，需要用 88 个 16 位的寄存器保存，GOST 算法有 8 个 32 位的工作子密钥，需要用 16 个 16 位的寄存器保存，RC6 算法有 36 个 32 位的工作子密钥，需要用 72 个 16 位的寄存器保存，另外还有一些中间计算结果需要保存，因此我们在寄存器堆中共设置 144 个 16 位的寄存器。

通过对 5 个样本算法所需要的读/写寄存器堆操作进行分析，我们发现，写寄存器堆操作最多需要同时进行 8 个（RIJNDAEL、IDEA、RC6），读寄存器堆操作最多也需要同时进行 8 个（RIJNDAEL、IDEA、RC6），为了使所有的样本算法达到自身所具有的最大的并行度，我们为寄存器堆设置 8 个写端口、8 个读端口，每个写端口可以将任一重组元素的输出或者外部输入的数据写入到寄存器堆中，每个读端口可以将寄存器堆中的任一寄存器的值读出。

这样寄存器堆有两种设计方案，方案一是设计一个包含 128 个 16 位的寄存器、拥有 8 个写端口、8 个读端口的寄存器堆，其中每个写端口可以将 64 个数据来源中的任何一个写入到 128 个寄存器中的任何一个，每个读端口可以将 128 个寄存器中的任何一个的值读出。方案二是用 8 个 16*16 寄存器堆的组合来实现的寄存器堆，每个 16*16 寄存器堆包含 16 个 16 位的寄存器、拥有 1 个写端口、1 个读端口，其写端口可以将 64 个数据来源中的任何一个写入到 16 个寄存器中的任何一个、读端口可以将 16 个寄存器中的任何一个的值读出，8 个 16*16 寄存器堆的输出经过 8 个通路单元产生 8 条数据总线，每个通路单元可以从 8 个 16*16 寄存器堆的输出中选择任何一个输出到对应的总线上。方案一和方案二均能满足 RELOG_RDIGCGG 的要求，两个方案的区别在于：方案一的寄存器堆比方案二的寄存器堆使用起来更加灵活、方便，但方案一的寄存器堆的规模比方案二的寄存器堆的规模大得多。比如说，方案一的寄存器堆的 8 个读端口能够同时读出 128 个寄存器中的任何 8 个寄存器的值；而方案二的 8 条数据总线虽然也能同时读出 8 个不同的数据，但这 8 个不同的数据必须位于不同的 16*16 寄存器堆，若两个不同的数据位于同一个 16*16 寄存器堆，则这两个数据不能被同时读出。关于两个方案的寄存器堆的规模的评估与比较可见按照方案一设计的寄存器堆需要 2048 个 D 触发器、38656 个 2 选 1 的选通器（假定所有的多路选通器都用 2 选 1 的选通器实现）；按照方案二设计的寄存器堆需要 2048 个 D 触发器、10880 个 2 选 1 的选通器（假定所有的多路选通器都用 2 选 1 的选通器实现）。

可见按照方案一设计的寄存器堆的规模比按照方案二设计的寄存器堆的规模要大的多，方案一对应的寄存器堆所需要的选通器的数量约是方案二对应的寄存器堆所需要的选通器的数量的 3.5 倍。为了减少寄存器堆的规模，我们采用方案二设计的寄存器堆。

第五章 体系结构设计分析与仿真验证

在第四章中，我们介绍了系统所用到 IP 的设计，其中有的 IP 在本系统中是重组元素，有的在本课题中只是单独的算法在用，但也可以作为系统扩展算法的重组元素来使用，所以这些所有的 IP 构成的集合我们就称为重组元素集合，下面我们就以此为基础来介绍系统体系结构的设计。

5.1 重组元素的设置

在上一章中我们得到了 3DES、IDEA、GOST、RIJNDAEL、RC6 五个样本算法所需要的重组元素集合，这些重组元素集合的并集就是可重组算法逻辑系统所需要的重组元素的集合。在这个重组元素集合中包括重组元素有：逻辑单元（兼容与、或、非、异或逐位操作），128*128 置换单元（兼容 2 个 64*64 置换，2 个压缩置换 64*48，2 个扩展置换 32*48，4 个 32*32 置换，完成任意置换），28 位循环移位单元，8*8S 盒（兼容 4 个 6*4S 盒，16 个 4*4S 盒），128 位移位单元（兼容 4 个 32 位移位单元），模 2^{32} 加法器（兼容 2 个摸 2^{16} 加），模 2^{32} 减法器（兼容 2 个摸 2^{16} 减），模 $2^{16}+1$ 乘单元，模 $2^{16}+1$ 乘逆单元，模 2^{32} 乘单元（兼容 2 个摸 2^{16} 乘），模 8 次多项式乘法运算单元，16*16 寄存器堆，通路单元}。

各个重组元素的名称、功能见下表：

表 5.1 可重组算法逻辑包括的重组元素的名称、功能

序号	名称	功能
1	28 位循环移位单元 SHIFT_28	能对 28 位数据进行循环移 1 位或 2 位的操作。
2	128 位移位单元 SHFT_128	能对 128 位数据进行逻辑左移、逻辑右移、循环左移、循环右移任意 n(n≤128)位的操作。并兼容 4 个 32 位移位单元
3	128*128 置换单元 PMT128	能够实现输入数≤128、输出数≤128 的任意的置换，包括一一对应置换、扩展置换、压缩置换。
4	8*8S 盒 Sbox8*8	能够实现 8 输入 8 输出的任意的布尔逻辑函数。并兼容 4 个 6*4S 盒，16 个 4*4S 盒。

序号	名称	功能
5	64 位逻辑单元 Log64	实现字长≤64 位的逐位与、或、非、异或运算。
6	模 2^{32} 加法器 Mod2n32add	实现 32 位模 2^{32} 加和 16 位模 2^{16} 加。
7	模 2^{32} 减法器 Mod2n32dec	实现 32 位模 2^{32} 减和 16 位模 2^{16} 减。
8	模 2^{16}+1 乘法器 Mod2n161mul	实现 16 位模 2^{16}+1 乘法。
9	模 2^{32} 乘法器 Mod2n32mul	实现 32 位模 2^{32} 乘法和 16 位模 2^{16} 乘。
10	模 2^{16}+1 乘法逆 Mmul2n161ag	求 16 位整数的模 2^{16}+1 乘法逆。
11	128 位加/解密结果寄存器 IOREG_128	保存加/解密结果。
12	16*16 寄存器堆 RE_16	保存工作密钥和中间结果，并作为数据传输的中转站。有 16 个 16 位的寄存器组成，一个写端口、一个读端口，可将 64 个 16 位数据中的任何一个通过写端口写入寄存器堆中的任何一个寄存器，也可将寄存器堆中的任何一个寄存器的值通过读端口读出。
13	通路单元 MUX11_1	能够从 8 个寄存器堆的输出中任意选择一个作为功能部件的数据来源。
14	模 8 次多项式乘法单元 Modpmul8	实现模 8 次多项式乘法运算

5.2 连接网络的设计

连接网络采用寄存器堆间接相连型结构，即除寄存器堆以外的任意两个重组元素都通过寄存器堆间接相连。系统中共有 8 个 16*16 寄存器堆；每个 16*16 寄存器堆有一个读端口、一个写端口，写端口能将外部输入的待加/解密数据、密钥和任一重组元素的操作结果写入该寄存器堆的 16 个寄存器中的任何一个，读端口能将该寄存器堆中的 16 个寄存器中 的任何一个寄存器所保存的数据读出；8 个 16*16 寄存器堆的输出经过 8 个通路单元的选择输出到 8 条数据

总线上，每一个通路单元能够从 8 个 16*16 寄存器堆的输出中选择任何一个输出到它所对应的数据总线上。除寄存器堆和通路单元以外的任一重组元素的操作数都只来自这 8 条数据总线中的某几条，且其输出只连接到寄存器堆的输入端口。其操作结果直接写入寄存器堆。[35]

连接网络的结构图如下：

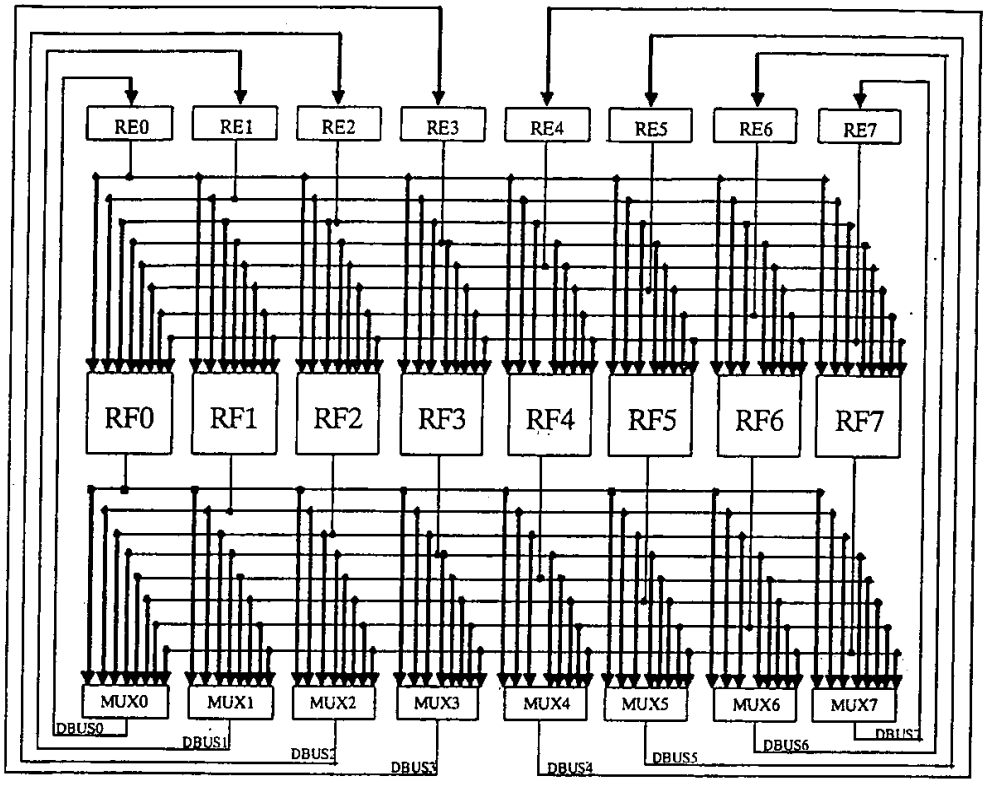

图 5.1　连接网络结构图

图 5.1 中 RE 表示重组元素，RF 表示寄存器堆，MUX 表示多路选通器，DBUS 表示数据总线，为简单起见，我们这里只画出了 8 个重组元素。如果为了实现更多算法，可以按需要在连接网络中扩充重组元素。

5.3 可控接点及其功能与特性

根据 5.1 节、5.2 节确定的可重组算法逻辑的重组元素和连接网络，我们得到了系统的可控接点，其名称、功能、编码宽度及特性见下表：[35]

表 5.2　可重组算法逻辑的可控接点的名称、功能、编码宽度及特性

序号	可控接点名称	功能	编码宽度

序号	可控接点名称	功能	编码宽度
1	保存加/解密结果控制信号 ESTORE	ESTORE=1 时，加/解密结果寄存器打开；ESTORE=0 时，加/解密结果寄存	1 位
2	寄存器堆写使能信号 ERFWi（i=0, 1, ……7）	ERFWi=1，寄存器堆 i 写操作使能；ERFWi=0，寄存器堆 i 写操作不使能。	1 位
3	寄存器堆 i 写控制接点 RFWi（i=0, 1, ……7）	RFWi[5：0]是写入寄存器堆 0 的数据来源的地址，RFWi[9：6]是目标寄存器地址。	10 位
4	寄存器堆 i 读控制接点 RFRPi（i=0, 1, ……7）	RFRPi=0000~1111 分别表示读出寄存器堆 i 的第 1~第 15 个寄存器的值。	4 位
5	第 i 数据总线 DATABUSi 控制接点 DBiCTRL	DBiCTRL=000~111 分别表示将第 0~第 7 个寄存器堆的数据输出到 DATABUSi。	3 位
6	128 位置换单元 PMT128 的静态编码 Pk0~Pk127	实现置换单元的任意位置换。	128*7=896 位
7	128 位置换单元 PMT128 的功能控制接点 POP	选择置换规格。00:128*128 置换；01：64*64 置换或压缩置换 64*48；10:32*32 置换；11：扩展置换 32*48；	2 位
8	8*8S 盒 SBOX88 的静态编码 K0~K255	实现 S 盒的任意布尔函数的变换。	256*8=2048 位
9	8*8S 盒 SBOX88 的功能控制接点 OP	选择不同规格的 S 盒变换。00：8*8 规格；01：6*4 规格；10：4*4 规格；	2 位
10	28 位循环移位单元 SHFT28 的移位位数控制接点，SHFT28_SB	28 位循环移位单元的移位位数 0—3 位。	2 位
11	28 位循环移位单元 SHFT128 的移位位数控制接点，SHFT28_DIR	28 位循环移位单元的移位方向。0：左移；1：右移。	1 位
12	128 位循环移位单元	128 位循环移位单元的移位位数：1—	7 位

序号	可控接点名称	功能	编码宽度
	SHFT128 的移位位数控制接点：SHFT128_SB	128 位。	
13	128 位循环移位单元 SHFT128 的移位方向控制接点：SHFT128_DIR	128 位循环移位单元的移位方向。0：左移；1：右移。	1 位
14	128 位循环移位单元 SHFT128 的移位方式控制接点：SHFT128_SF	128 位循环移位单元的移位方式。0：循环移位；1：逻辑移位。	1 位
15	128 位循环移位单元 SHFT128 的移位功能控制接点：SHFT128_OP	128 位循环移位单元的功能编码。0：32 位移位；1：128 位移位。	1 位
16	逻辑单元 LOG64 操作功能控制接点：LOG64_OP	逻辑单元操作选择。00：与操作；01：或操作；10：非操作；11：异或操作。	2 位
17	模 8 次多项式乘法器的模多项式系数控制接点 MPOLY_8	MPOLY_8 的第 0~7 位分别是模多项式系数的 0 次项~7 次项的系数。	8 位
18	法逆 mod2n161mul 的操作使能控制信号 En	En=0：模 2^{16+1} 乘法逆操作使能； En=1：模 2^{16+1} 乘法逆操作不使能。	1 位
19	模 2^{32} 加	选择不同规格的模加。1：模 2^{32} 加；0：模 2^{16} 加.	1 位
20	模 2^{32} 加逆	选择不同规格的模加逆。1：模 2^{32} 加逆；0：模 2^{16} 加逆.	1 位
21	模 2^{32} 减	选择不同规格的模减。1：模 2^{32} 减；0：模 2^{16} 减.	1 位
22	模 2^{32} 乘 moc2n32mul 功能控制接点 moc2n32mul_op	选择不同规格的模乘。1：模 2^{32} 乘；0：模 2^{16} 乘.	1 位

5.4 主频估计

系统的主频估计主要依据系统中的 IP 电路延时来确定。在本课题中，各个重组元素的电路

延时自接影响了可重组算法逻辑系统的主频。下面我们就给出各个重组元素的电路延时。

表 5.3：重组元素的电路延时

序号	基本密码操作名称	所需时间(纳秒 ns)
1	逻辑单元	7.814
2	模 2^{32} 减	7.782
3	模 2^{32} 加	7.781
4	28 位循环左移操作	4.439
5	模 2^{32} 加逆	7.782
6	模 2^{32} 乘	9.943
7	模 2^{16}+1 乘法运算	35.870
8	128 位循环移位单元	9.984
9	8*8S 盒	12.667
10	模 8 次多项式乘法单元	5.413

由表 6.1 可见，8/10≈80%的基本密码操作可以在 10ns 之内完成，因此我们将系统的时钟周期的长度确定为 10ns，即主频 100Mhz，这样，绝大多数的重组在一周期内完成，从而使系统能够获得较高的性能。

5.5 仿真验证

根据第五章体系结构，我们进行了可重组算法逻辑 VERILOG 模型的建立，但由于本人水平和经验的不足，对这一系统设计无法完成。在重组元素设计的基础上尝试了对单一算法 3DES 进行了实现，3DES 的加密的功能仿真如图 5.2，图 5.3：

图 5.2　3DES 密码算法加密功能仿真图

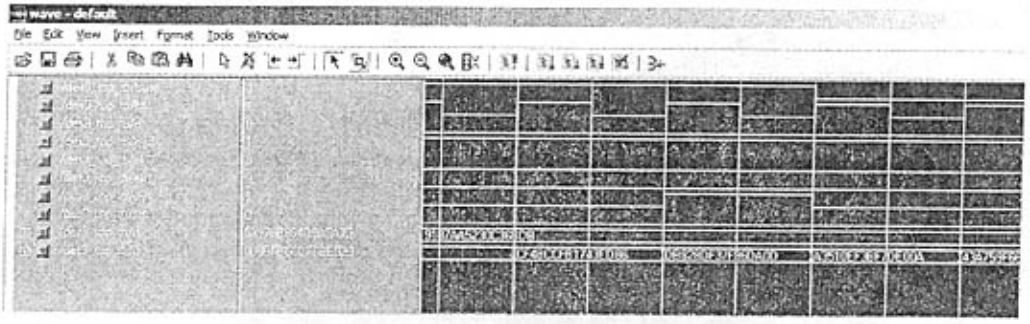

图 5.3　3DES 密码算法解密功能仿真图

本设计中为可重组算法设计提供了 9 个可编程的 IP 模块，有标准的接口，多数能在 10ns 内完成相关操作，并通过变换可控节点实现不同的功能或规格，为以后可重组算法逻辑的实现奠定了基础。

5.6 重组元素在安全芯片中的应用

由于可编程 IP 模块具有密码算法重组的特性，它可以应用在安全芯片的 SOC[33][41]设计中，作为安全芯片设计中的 IP 来进行使用，通过编码配置来实现密码算法的重组。下面给出可编程 IP 模块在安全芯片中的应用图。如图 5.4。

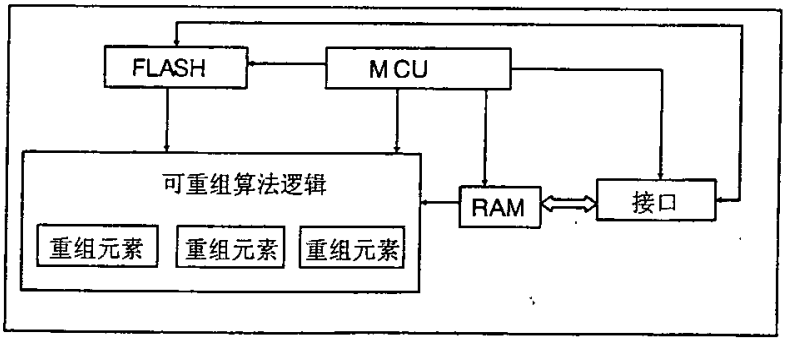

图 5.4 可重组算法逻辑在安全芯片中的应用

芯片在投入使用之前是不含密码算法的白裸片，只有在使用时才由用户将设计好的密码算法装载到芯片上，如通过接口把算法编码注入到芯片 FLASH 中，再通过 MCU 调用算法编码对数据进行加解密处理。这样芯片中的算法可以随时更换，大大降低了算法泄露的可能性，增强了抗攻击性，从而提高了密码系统的安全性。

第六章 总结与展望

6.1 总结

我们确定样本空间为对称算法空间，并对 50 种对称算法进行了需求分析，并详细分析了 RIJNDAEL、3DES、IDEA、GOST、RC6 这五种算法，得到了这些算法的详细实现流程及操作资源和连接关系。并且依据算法流程提炼出重组元素，我们根据资源复用的原则对重组元素进行设计，使得单个重组元素兼容不同规格和功能运算，可以有效的节省芯片规模。并且分析设计了可重组算法逻辑的大概体系结构。在重组元素设计的基础上，对 3DES 算法进行了实现，并进行了功能仿真。

提供的可编程 IP 模块可以作为 IP 应用在安全芯片的 SOC 设计中，重组算法的可控接点编码可以从接口输入存储在 FLASH 中，控制模块调用 FLASH 中算法重组编码来重组重组元素进而实现密码算法重组。

由于本人设计水平和时间的限制，还没完成可重组算法逻辑的完整设计，只能在以后的实践中等待水平提高加以实现。

6.2 展望

在以后的研究中，希望在本文的基础上，搭建完整的可重组算法逻辑系统结构，实现所有五种预定密码算法。在本文中，由于我们只对 RIJNDAEL、3DES、IDEA、GOST、RC6 这五种分组密码算法进行分析，进行重组元素的提炼和设计，那么在密码算法重组时也只能实现固定的几种分组密码算法。对序列密谋算法和散列算法的实现就受到了限制，这也是本设计中的遗憾，希望在以后的研究中，能对尽可能多的密码算法进行分析，扩展更多的重组元素，使得可重组算法逻辑能实现更多的密码算法。

以重组元素为基础的可重组算法逻辑芯片是基于多个对称算法设计的，它可以通过编码配置实现多种密码算法，因此可代替多块专用密码算法芯片，从而大大降低了密码算法芯片的开发和生产费用，这意味着可重组算法芯片具有广泛的市场前景；由于可重组算法芯片能够随时变换算法，所以能够有效地防止被攻破的可能，因此具有极其重要的安全性。基于上述原因，我们有理由相信可重组算法芯片必将在信息安全领域得到广泛的应用和发展。

致谢

本论文是在我的导师傅兴华教授的悉心指导下完成的，在论文的选题、论文的写作、论文的规范等方面傅兴华教授都给予了精心的指导。在三年的学习中，傅兴华教授渊博的知识、严谨求实的治学态度和勤勤恳恳的工作作风给我留下了深刻的印象，让我终身受益。在此，我衷心感谢傅兴华教授对我的谆谆教诲和精心培养。

攻读研究生期间，我的生活和学习也得到了所有授课老师的指导和诸多帮助。在此我向各位老师表示衷心的感谢。

感谢北京多思科技工业园股份有限公司为我提供了实习机会，为论文的设计提供了条件。

在论文的研究和论文的撰写过程中，还得到了各位同学的热情帮助。在这里谨向各位同学致以诚挚的谢意！

最后，我要感谢关心我的家人和朋友，感谢他们一直以来对我的支持和鼓励。

参考文献

[1]Estrin G, Bussel B, Turn R, Bibb J. Parallel Processing in a Restructurable computer System [J]. IEEE Trans. Elect. Comput, 1963:747-755

[2]尹勇生，高明伦，李丽.基于多流水线的可重构系统.微电子学与计算机.2005 年第 22 卷第 10 期.

[3].可重构系统(Reconfigurable System)简介;

http://www.icedu.net/XXLR1.ASP?MENULB=&LB=009FPGA%B8%C5%CA%F6&ID=160&LRPAGESN=4

[4] Bruce Schneier;应用密码学－协议、算法与 C 源程序;机械工业出版社;2004.10

[5]曲英杰，李占才，王沁，涂序彦，“适用于可编程加密芯片的可重组体系结构”， 计算机工程与应用，2001 年 第 37 卷 第 19 期。

[6] 卢开澄;计算机密码学---计算机网络中的数据保密与安全;清华大学出版社

[7] 王衍波,薛通;应用密码学;机械工业出版社

[8] David Salomon(美)著, 蔡建,梁志敏译;数据保密与安全;清华大学出版社;2005.6.1

[9] Atul Kahate 著;邱仲潘等译;密码学与网络安全(cryptography and network security);清华大学出版社;2005.9

[10] 龙冬阳.王常吉.吴丹;应用编码与计算机密码学;清华大学出版社;2005.11

[11] A[fred]。Menezes Paul C. Van Oorschot Scott A. Vanstone; 胡磊，王鹏 等译; 应用密码学手册(Hand book of applied cryptography) ;电子工业出版社、2005.6

[12] 杨义先，钮心忻; 应用密码学; 北京邮电大学出版社; 2005.6

[13] 胡向东，魏琴芳; 应用密码学教程; 电子工业出版社; 2005.1

[14] 蔡乐才，张仕斌，郝文化; 应用密码学; 中国电力出版社; 2005.2

[15] 邱志聪; 加密解密技能百练; 中国铁道出版社; 2005.1

[16] Dded Goldreich;温巧燕 等译; 密码学基础 (Foundations of Cryptography) ;人民邮电出版社; 2005.5

[17] Wade trappe Lawrence C. Washington; 邹红霞，许鹏文，李勇奇 译; 密码学概论 (introduction to cryptography with coding theory) ;人民邮电出版社; 2004.6

[18] 章照止; 现代密码学基础; 北京邮电大学出版社; 2004.4

[19] Steve Burnett Stephen Paine; 冯登国，周永彬，张振峰，李德全 译; 密码工程实践指

南(RSA security' s official Guide to cryptography);清华大学出版社；2001.10

[20] Niels Ferguson Bruce Schneier;张振峰，徐静，李红达译；密码学实践(practical Cryptogaphy);电子工业出版社，2005.8

[21]杨波：现代密码学(Modern Cryptography);清华大学出版社；2003.8

[22]曲英杰，王沁，王昭顺，“可重构体系结构的特征及应用”，计算机工程与应用，2001年 第37卷 第17期。

[23]钱树明,吴灏.信息传输的安全需求与密码算法.www.yesky.com/152/1851652_1.shtml.

[24]算法密码谈.

www.juntuan.net/hkjc/xinshou/n/2005-05-06/4641.html.2005.5.

[25]何明星，林昊；AES算法原理及其实现：四川工业学院计算机科学与工程系；2002.1

[26]韦宝典等.求S盒布尔函数表达式的一种新算法.通信学报,2003.

[27]曲英杰，夏宏，王许书，涂序彦，“可编程S盒及反馈移位寄存器的设计方法”，计算机工程与应用，2001年 第37卷 第11期。

[28]刘景伟等. AES S盒的密码特性分析.西安电子科技大学学报（自然科学版）.2004;

[29] Ranjan著、武传坤译；信息论、编码与密码学；机械工业出版社；2005.1;

[30]牛少彰；信息安全概论；北京邮电大学出版社；2004.4;

[31] Samir Palnitkar著；夏宁闻 等译；Verilog HDL 数字设计与综合；2004.11;

[32]朱子玉、李亚民；CPU芯片逻辑设计技术；清华大学出版社；2005.1;

[33]王彬、任艳颖；数字IC系统设计；西安电子科技大学出版社；2005.9;

[34]任艳颖、王彬；IC设计基础；西安电子科技大学出版社；2004.4;

[35]Richard J. Spillman; CLASSICAL AND CONTEMPORARY CRYPTOLOGY; Publishing House of Qinghua;2005.7;

[36]William Stallings 著，杨明 胥光辉 齐望东 等译，密码编码学与网络安全：原理与实践（第二版），北京：电子工业出版社，2001.4

[37]张亮海编著，数字电路设计 Verilog HDL，北京：人民邮电出版社，2000.10

[38]侯伯亨，顾新编著，VHDL硬件描述语言与数字逻辑电路设计，西安：西安电子科技大学出版社，1999.1

[39]潘松，王国栋编著，VHDL实用教程，北京：电子科技大学出版社，2000.2

[40](美)J.Bhasker编著，徐震林等译，Verilog HDL硬件描述语言，机械工业出版社，2000.7

[41] Virtual Socket Interface alliance homepage. http://www.vsi.org

[42]王金荣、陈勤、丁红；大数模乘算法的分析与研究；计算机工程与应用；2004.24

[43]Rochit Rajsuman,于敦山 等译；SOC 设计与测试；北京航空航天大学出版社；2003.8

[44]L. Dadda. On Parallel Digital Multipliers. Alta Frequenza. 1976, 45(10):574~580

[45] Huey Ling. High-Speed Binary Adder. IBM Journal of Research and Development. 1981, 25(3):156~165

[46] R: Hashemian. A New Design for High Speed and High-Density Carry Select Adders. IEEE Midwest Symposium on Circuits and Systems. East Lansing, 2000:1300~1303

[47]Roland Airiau, Jean-Michel Berge and Vincent Olive,CNET,France Telecom,France，《Circuit Synthesis with VHDL》Kluwer Academic Publishers，Netherlands, 1994

[48]Stanley Mazor and Patricia Langstraat, Synopsys, Inc, 《A Guider to VHDL (Second Edition)》Kluwer Academic Publishers,Netherlands, 1993

[49]Jean-Michel Berge,Alain Fonkoua,Serge Maginot,Jacques Rouillard,《VHDL'92》Kluwer Academic Publishers,Netherlands,1993

附录

[1]苏海亮；多规格 S 盒的硬件实现方法；电子设计应用；已录用。

原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究在做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律责任由本人承担。

论文作者签名：[illegible]　　日　期：2006年6月

关于学位论文使用授权的声明

本人完全了解贵州大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权贵州大学可以将本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、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名：[illegible]　导师签名：[illegible]　日　期：2006年6月

对称算法空间的可重组逻辑研究与SOC设计

作者：苏海亮
学位授予单位：贵州大学