摘 要

本文从AES的算法原理和基于ARM核嵌入式系统的开发着手，研究了AES算法的设计原则、数学知识、整体结构、算法描述以及AES存在的优点和局限性。

针对ARM核的体系结构及特点，对AES算法进行了优化设计，提出了从AES算法本身和其结构两个方面进行优化的方法，在算法本身优化方面是把加密模块中的字节替换运算、列混合运算和解密模块中的逆列混合运算中原来的复杂的运算分别转换为简单的循环移位、乘和异或运算。在算法结构优化方面是在输入输出接口上采用了4个32位的寄存器对128bits数据进行了并行输入并行输出的优化设计；在密钥扩展上的优化设计是采用内部扩展，即在进行每一轮的运算过程的同时算出下一轮的密钥，并把下一轮的密钥暂存在SRAM里，使得密钥扩展与加/解密运算并行执行；加密和解密优化设计是将轮函数查表操作中的四个操作表查询工作合并成一个操作表查询工作，同时为了使加密代码在解密代码中可重用，节省硬件资源，在解密过程中采用了与加密相一致的过程顺序。

根据上述的优化设计，基于ARM核嵌入式系统的ADS开发环境，提出了AES实现的软硬件方案、AES加密模块和解密模块的实现方案以及测试方案，总结了基于ARM下的高效编程技巧及混合接口规则，在集成开发环境下对算法进行了实现，分别得出了初始密钥为128bits、192bits和256bits下的加密与解密的结果，并得到了正确验证。在性能测试的过程中应用编译器的优化选项和其它优化技巧优化了算法，使算法具有较高的加密速度。

关键词：AES，ARM，算法，嵌入式，优化

Abstract

This thesis introduces the algorithm principles of AES and ARM Embedded System. The research and analysis of system are focused on the design principles of AES, the fundamental of mathematics, the structure of AES, the description of the algorithm and the advantage and limitation of AES.

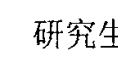
The design of AES is optimized according to the ARM's system structure and characteristic. The optimization method of AES is presented from AES algorithm itself and AES's structure .In the optimization method of AES algorithm itself, the complex calculation is converted to the simple cycle shift, multiplication and XOR calculation from Subwords, Mixcolumns and InvMixcolumns. In the optimization method of AES's structure, the four 32bits registers which are used for the parallel 128bits data input and output are adopted on the I/O interface. The interior expanding method is used on the optimization design of KeyExpansion. The optimization design of decryption and encryption is to combine the four operation tables of Round function to one operation table. At the same time, the same process sequence of encryption is used in the decryption process in order to reuse the encryption code in the decryption, therefore to save the hardware resource.

According to the optimization design method discussed above, based on the development platform of ADS of ARM embedded system, the hardware, software, encryption, decryption and test plans are suggested. The high efficiency programming skill and standard of the mixture interface are concluded. After that, the algorithm is realized in the integrated development platform. The results of the encryption and decryption under the initial key of 128bits, 192bits and 256bits are achieved and validated. AES algorithm is optimized by the optimal option in compiler and other optimization skill, which enhances the encryption speed of algorithm.

Keywords: AES, ARM, algorithm, Embedded System, optimization

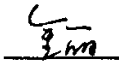
东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，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，也不包含为获得冬南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了谢意。

研究生签名：  日　期：2007.5.10

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学研究生院办理。

研究生签名：  导师签名：  日　期：2007.5.10

第一章 绪论

1.1 AES 概述

随着计算机运算能力的飞速发展，以及互联网所带来的巨大并行计算能力，同时为了确保信息系统中硬件、软件及正在处理、存储、传输信息的保密性、完整性和可用性，第一代私钥加密标准 DES 安全性及可靠性已逐渐减弱。1997 年 4 月 15 日，美国国家标准技术研究所（NIST）向全世界范围内公开发起征集先进加密标准 AES（Advanced Encryption Standard）[1-4]的活动，目的是确定一种保护敏感信息的公开的、免费的且全球通用的算法作为 AES，以代替 DES。在征集公告中，NIST 对算法的基本要求是：算法必须是私钥体制的分组密码，支持 128 位分组长度和 128bits、192bits、256bits 密钥长度。NIST 对 AES 进行评估的主要准则是安全性，效率和算法的实现特性。安全性是第一位的，候选算法应当抵抗已知的密码分析方法，没有明显的安全缺陷。在满足安全性的条件下，效率是最重要的评估因素，包括算法在不同平台上的计算速度和对内存空间的需求等。算法的实现特性包括灵活性，如在不同类型的环境中能安全和有效的运行等。另外，算法必须能够用软件和硬件两种方法实现。经过二轮遴选，Rijndael 最终胜出。2000 年 10 月 2 日，NIST 宣布 Rijndael 算法作为新一代高级加密标准。2001 年 11 月 26 日联邦信息处理标准出版社发布了正式的 AES 标准[3]，即 FIPSPUBS197，其中指出制定的标准生效时间为 2002 年 5 月 26 日。

AES 算法是一种可变分组长度和密钥长度的迭代型对称密钥分组密码，它的分组长度和密钥长度均可独立地指定为 128bits、192bits、256bits，它以其安全性和多方面的优良性能，成为 AES 的最佳选择。AES 算法能抵抗现在的所有已知密码攻击；无论使用反馈模式还是无反馈模式，它在广泛的计算环境中的硬件和软件实现性能都表现得非常优秀；它的密钥建立时间极短且灵活性强；它极低的内存要求使其非常适合在存储器受限的环境中使用，并且表现出很好的性能。

1.2 ARM 嵌入式系统概述

嵌入式系统是指以应用为中心，以计算机技术为基础，软件硬件可裁剪，适应应用系统对功能、可靠性、成本、体积和功耗严格要求的专用计算机系统。在当前数字信息技术和网络技术高速发展的后 PC 时代,嵌入式系统技术已经广泛地渗透到科学研究、工程设计、军事技术、各类产业和商业文化艺术以及人们的日常生活等方方面面中,成为目前最热门的技术之一。随着嵌入式产品逐渐占领市场。高性能，低功耗，低成本是这些嵌入式处理器的主要特点。在这些 32 位嵌入式处理器市场中，ARM 占有 78.6%的份额。

ARM嵌入式处理器是一种高性能、低功耗的RISC芯片。它由英国ARM公司设计，世界上几乎所有的主要半导体厂商都生产基于ARM体系结构的通用芯片，或在其专用芯片中嵌入ARM相关技术。基于ARM的芯片都是由ARM的半导体合作伙伴通过ARM授权进行设计、制造和销售，各个合作厂商在ARM技术的基础上融入各自的特色，并在多个领域进行应用。

目前基于ARM技术的处理器已经被广泛应用于各种电子产品中。截至2004年，ARM的合作伙伴共运付了近30亿个含有ARM内核的芯片，ARM已成为移动通信、手持计算、多媒体数字消费等嵌入式解决方案的RISC标准。

自从1991年RISC CPU推出，ARM结构体系发生了很大演变和提高。在十几年的发展过程中，共发展出7个版本的结构体系，每一个结构体系版本代表了一套指令集定义和相应的功能框架，且所有的结构体系都保持了良好的向下兼容性。不同版本ARM结构体系的特征不同，不同结构体系版本之间极明显的结构差异。ARM体系结构从VI～V3版本开始到目前的Cortex(V7)，每个版本都推动着控制器的进步。

本论文用的是ARM7系列的处理器，使用该系列中的CPU代表是ARM7TDMI，ARMTTDMI采用冯•诺依曼结构，具有3级流水线，可以提供0.9MIPS的性能。该内核用于嵌入式控制或简单的应用系统。

1.3 AES 研究现状

目前AES算法的理论研究主要集中在设计原理、安全性能分析和统计性能分析上。

对于设计原理，主要研究算法设计遵循的原则和整体结构。AES算法所遵循的是安全性和实现性原则，在整体结构上采用的是SP网络结构。

对于安全性能，主要研究AES算法抵抗现有已知密码攻击的能力。当前主要攻击手段有：强力攻击、差分密码分析、线性密码分析、Square攻击和插值攻击等。目前密码分析又有了新的进展，积分分析、功耗分析和代数攻击成为新的研究方向。

对于统计性能，主要研究算法随机化数据的能力，目前国内外研究都比较少。

当前AES算法的实现研究主要集中在软件PC实现、硬件实现和DSP实现三个方面。

软件PC实现主要是用高级语言实现算法，并测试不同工作模式下的性能，商用的软件加密产品还未见到。

对于硬件实现，国外一些机构和大学推出了供测试的AES核(core)，如美国GMU、NSA、Helion技术有限公司等大学和机构。他们提供的AES核大都支持NIST所要求的三种密钥长度，具有较快的加、解密速度。

DSP实现目前主要还是AES算法在DSP上的性能测试研究较多。由于其32位运算特性，算法的执行速度很快，所以AES算法在32位处理器上有着比较优良的性能。

1.4 本论文的研究意义

随着计算机通信技术的飞速发展以及互联网的广泛应用，人们对信息安全的需求也越来越高。信息安全已成了当今尤为重要的且紧迫需解决的问题。目前高级加密标准（AES）产品正处于使用初期，市场上的AES产品不多，随着具有高性能、低功耗、低成本特点的嵌入式系统的不断广泛应用，预计不久的将来基于嵌入式的AES产品将占有很大的市场需求空间。若将AES研究和开发本土化，具有一定的实用价值，不但可提高它的速度，而且具有高度的保密性，将在一定程度上保证了通信的安全。

目前江苏省专用集成电路设计重点实验室实现的AES加密芯片，用原来的算法花费了大量的运行时间，而且占用了空间大，为了克服这些不足，所以基于ARM核嵌入式系统上对AES算法的优化研究具有较大的理论意义和实践意义。

1.5 本论文研究的内容及目标

本论文研究的内容如下：

1. 研究AES算法原理结构和特点，对AES算法的实现进行分析；
2. 研究ARM的体系结构与编程，对AES算法进行优化设计；
3. 采用基于ARM7TDMI核的嵌入式系统的ADS开发环境，实现AES算法。
4. 结合ARM体系的特点，对AES算法进行代码优化及性能测试。

本论文研究的目标是提出AES算法优化设计方案，并在基于ARM核的嵌入式系统的开发环境中进行优化实现，总结AES在ARM核中的实现经验并提出优化原则。

第二章 AES 算法

2.1 AES 算法设计原则

AES 算法在设计时遵循两方面重要原则：一是安全性原则；二是实现性原则。

2.1.1 安全性原则

安全性原则主要是指香农(Shannon)[5-12]所提出的混乱原则和扩散原则。

混乱原则是指人们所设计的密码应使得密钥和明文以及密文之间的依赖关系相当复杂，以至于这种依赖关系对密码分析者来说无法利用。

扩散原则是指人们所设计的密码应使得密钥的每一位数字影响密文的许多位数字，以防止对密钥进行逐段破译，而且明文的每一位数字也应该影响密文的许多位数字，以便隐蔽明文数字的统计特性。

现代实用分组密码算法通常采用轮函数多次迭代的结构，如果轮函数设计适当，经过若干次迭代后就可以提供必要的混乱和扩散，这种分组密码称为迭代分组密码。而 AES 算法属于迭代分组密码，在进行多轮迭代后就提供了必要的混乱和扩散，有效的抵抗了通用攻击。在多轮迭代的同时也消除了 AES 算法面向字节处理的不安全因素，有效地抵抗了对算法的专用攻击。

2.1.2 实现性原则

对于一个好的密码算法，具有良好的实现性的特点为：设计简单、结构合理紧凑、易于软件或硬件实现。硬件实现的优点是可获得高速率，软件实现的优点是灵活性强，代价低。

软件实现的设计原则：尽量使用子模块和简单的基本运算。密码算法设计时针对某一特定的长度进行，子模块的长度应尽可能地适应软件编程，如采用 8 位、16 位、32 位的子块，同时密码算法应采用一些易于软件实现的简单运算，如加、减、乘、移位运算等。

硬件实现的设计原则：加密和解密结构应尽可能一致，即加密和解密的过程仅在密钥的使用方式上不同，以便同样的器件既可用来加密又可用来解密。

基于以上实现性原则，AES 算法设计完全符合上述原则，因此具有良好的实现性。

2.2 AES 算法数学基础

在 AES 算法中，各种操作都是以字节为单位进行的，字节表示有限域 $GF(2^8)$[13-18]中的元素。有限域的元素可以进行加和乘的运算。

2.2.1 有限域 $GF(2^8)$

有限域是具有有限个元素的域，元素的个数称为域的阶。q 阶域存在当且仅当 q 是某素

数的幂，即存在某个整数 n 和素数 p，使得 $q=p^n$，P 称为有限域的特征。有限域记为 $GF(p^n)$，AES 算法中涉及两类如下形式的特征域上的多项式。

$$b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0, b_i \in GF(2) \qquad (2.1)$$

$$a(x)=a_3x^3+a_2x^2+ax+a_0, a_i \in GF(2^8) \qquad (2.2)$$

其中 $b(x)$ 表示算法中的单字节(Byte)数据，$a(x)$ 表示算法中的四字节(4Bytes)或字(Word)数据。单字节也可以用数据向量 $\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}$ 或十六进制数 $\{mn\}$ 表示。在 AES 的描述中均使用以 2 为特征的有限域。

2.2.2 单字节运算

在 AES 算法描述中，我们把字节看作多项式。单字节的运算是如(2.1)所示的多项式的运算。

1. 单字节加：在有限域 $GF(2^8)$中，通过将两个多项式中的对应幂的系数进行“加”取 2 的模来实现的，这里的“加”指的是“异或”操作，用符号⊕表示。即 1⊕1=0，1⊕0=1，以及 0⊕0=0。因此对于两个字节 $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ 和 $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$，和为 $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$，我们可把它描述为 $c_j=a_j \oplus b_j$。

例如，

$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$　多项式表示

$\{01010111\}\oplus\{10000011\}=\{11010100\}$　二进制表示

$\{57\}\oplus\{83\}=\{d4\}$　十六进制表示

2. 单字节乘：在有限域 $GF(2^8)$中，两个多项式的乘积是将两个多项式相乘的结果再模一个次数为 8 的不可约分多项式 $M(x)=x^8+x^4+x^3+x+1$，用十六进制表示为：{01}{1b}。

例如：

$$(x^6+x^4+x^2+x+1)\bullet(x^7+x+1)$$
$$=(x^{13}+x^{11}+x^9+x^8+x^7)+(x^7+x^5+x^3+x^2+x)+(x^6+x^4+x^2+x+1)$$
$$=x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$$
$$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\bmod(x^8+x^4+x^3+x+1)=x^7+x^6+1$$

即 $\{57\}\bullet\{83\}=\{c1\}$

3. 单字节乘 x：用 x 乘以一个多项式再模 M(x)，记为 b=xtime(a)。即通过字节级的左移紧跟一个有条件的与 M(x)进行比特异或来实现。

例如，$\{57\}\bullet\{13\}=\{fe\}$ 因为

$$\{57\}\bullet\{02\}=xtime(\{57\})=\{ae\}$$
$$\{57\}\bullet\{04\}=xtime(\{ae\})=\{47\}$$
$$\{57\}\bullet\{08\}=xtime(\{47\})=\{8e\}$$
$$\{57\}\bullet\{10\}=xtime(\{8e\})=\{07\}$$

因此，

$$\{57\}\bullet\{13\}=\{57\}\bullet(\{01\}\oplus\{02\}\oplus\{10\})$$
$$=\{57\}\oplus\{ae\}\oplus\{07\}$$
$$=\{fe\}$$

2.2.3 四字节运算

四字节运算为如(2-2)所示的多项式的运算。

1. 四字节加：把两多项式中具有相同的 x 的幂的有限域系数相加。

例如：$a(x)=a_3x^3+a_2x^2+a_1x+a_0, b(x)=b_3x^3+b_2x^2+b_1x+b_0$

则 $a(x)+b(x)=(a_3\oplus b_3)x^3+(a_2\oplus b_2)x^2+(a_1\oplus b_1)x+(a_0\oplus b_0)$

2. 四字节乘：乘法要通过两步来实现。在第一步中，多项式的积被代数扩展，相同次数的幂进行相加；在第二步中，把相加的结果对一个次数为 4 的多项式 M(x)取模，$M(x)=x^4+1$，这样使得次数小于 4 的多项式的乘积仍然是一个次数小于 4 的多项式。

3.四字节乘 x：乘 x 是对应向量内字节的一个左循环移位。

2.3 AES 的整体结构

分组加密算法是由一种叫做轮变换的函数通过多次迭代构成的，轮变换的构成包括非线性层，扩散层和密钥调度等元素。这些设计是基于香农提出的设计原则：非线性替代与线性混合函数交替进行。这样结合的结果导致来自密码的重要的统计特性必须是高度相关的和敏感的类型，即通过混合变换的扩散和混乱产生充分的混合，使加密后的分组统计特性分布更均匀。AES 在整体结构设计上采用的是替代/置换(SP)网络的迭代结构方式，这种结构在安全性方面能抵抗各种攻击。

2.3.1 SP 网络迭代结构

SP 网络迭代结构是近几年来应用比较广泛的一种结构，这种网络的结构简单清晰，每一轮由非线性 S 层和线性层 P 层组成，SP 型密码的一轮加密过程如图 2-1 所示：

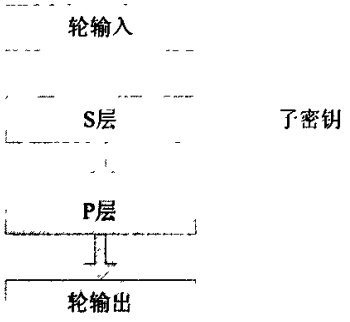


图 2-1 SP 网络结构

SP 型密码的每一轮变换中，首先将 S 层作用于轮输入使其混乱，然后经过 P 层作用使之得到扩散。SP 型密码具有的优点是给定 S 层和 P 层的密码指标后，可以从理论上给出抵抗差分密码攻击和线性攻击的能力。此外，经一轮变换后，轮输入的每一位均得到了扩散，这个比其它类型的结构扩散的速度快。

2.3.2 AES 算法结构

AES 算法结构设计合理紧凑，其加密/解密算法结构如图 2.7(a)(b)所示。AES 算法结构属于 SP 结构，组成其每一轮变换的 4 个函数分别属于 S 层、P 层和密钥加层。

S 层是由字节替换函数 SubBytes( )组成，即是非线性层，它的作用主要是确保多轮迭代后结果的高度混乱，其目标就是通过一个较小的非线性元素得到一个大的非线性构件。

P 层由行移位函数 ShiftRows( )和列混合函数 MixColoumns( )组成，即是线性层，它的作用是确保多轮迭代后的高度扩散，其目标就是使非线性元素取得尽可能简单。

密钥加层密钥加函数 AddRoundkey( )组成，该层主要实现了子密钥与明文或密文的结合。

2.4 AES 算法描述

AES 为分组密码算法，数据块长度为 128 位，密钥长度为 128、192、256 位可选，以下分别称之为 AES-128、AES-192 和 AES-256。

2.4.1 状态、密钥种子和轮数

AES 算法的明文分组以及每次变换的中间结果分组叫作状态（state）。状态可表示为二维字节数组，它有 4 行 4 列，如表 2.1 所示：

表2.1 列数N_b=4的状态

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

输入按，$a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$,....的字节顺序映射到状态中。加密结束后，密文按同样的顺序从状态中抽取。

密钥种子类似的用二维字节数组表示，该数组有 4 行，列数记为 Nk，Nk 等于分组长度除以 32，如表 2.2 所示：

表 2.2 N_k=4 密钥种子

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
K3,0	K3,1	K3,2	K3,3

算法转换的轮数由表2.3所示：

表 2.3 密钥长度、数据块长度以及加密轮数的组合

参数 / 分类	密钥长度 N_k (单位：32 比特字)	数据块长度 N_b （单位：32 比特字）	轮数 N_r
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

2.4.2 轮变换

AES 算法加密过程中的轮变换由四个变换组成，它们分别是：字节代替（SubByte）、行移位（ShiftRow）、列混合（MixColumn）、密钥加（AddRoundKey）。轮变换的伪 C 代码如下：

```
第一轮之前执行 AddRoundKey(State,RoundKey)
Round(State,RoundKey)
{
 SubBytes(State);
 ShiftRow(State);
 MixColumn(State);
 AddRoundKey(State,RoundKey);
}
```

加密算法的最后一个轮同稍有不同。它由以下代码定义：

```
FinalRound(State,RoundKey)
{
 SubBytes(State);
```

```
ShiftRow(State);
AddRoundKey(State,RoundKey);
}
```

下面就轮变换中的四种不同变换分别进行介绍：

1. SubBytes( )变换

SubBytes( )变换即 S－盒运算，是将每个字节通过 S－盒（如表 2.4 所示）做非线性运算。S－盒选取的是有限域 $GF(2^8)$中的乘法逆运算。$GF(2^8)$中的乘法（以●表示）是多项式的模 2 乘积通过免去进位，再模一个次数为 8 的不可约多项式约化得到。S-盒是可逆转的，它通过两步变换得到：

(1)在有限域 $GF(2^8)$中求得乘逆，其中‘00’的逆就是它本身。

(2)运用下面的仿射变换(GF(2)中)：

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \qquad (2.3)$$

用矩阵形式表示，仿射变换结果如下：

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

SubBytes( )变换通常是通过查表方式来实现，它的十六进制表示形式如表2.4所示：

表2.4 S－盒（十六进制形式）

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

状态上的表示形式如图2-2所示：

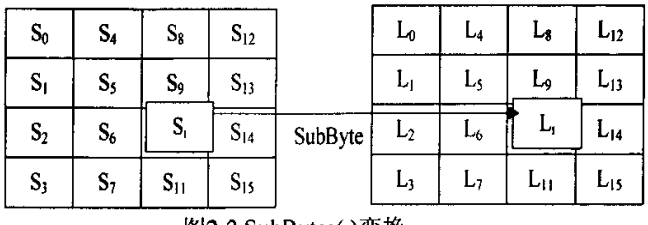


图2-2 SubBytes( )变换

2. ShiftRows( )变换

ShiftRows( )变换是进行行变换，即对状态矩阵进行移位操作，第一行保持不变，第二行循环左移 P_1 字节，第三行循环左移 P_2 字节，第四行循环左移 P_3 字节。位移量 P_1、P_2 和 P_3 与分组长度 N_b 有关，具体如表 2.5 所示：

表 2.5 位移量与 N_b 的关系

分组长度 N_b	P_1	P_2	P_3
4	1	2	3
6	1	2	3
8	1	3	4

图 2-3 表示为 ShiftRows( )变换。

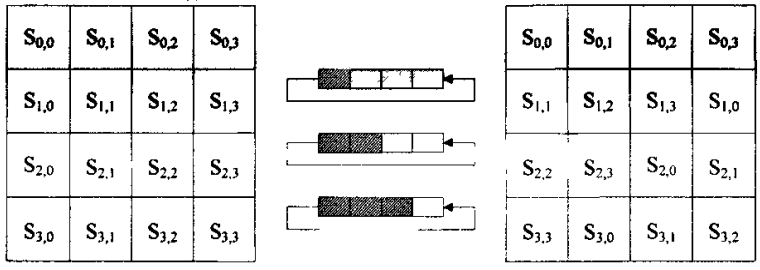


图 2-3 ShiftRows（）变换

3. MixColumns( )变换

MixColumns( )运算是把状态中的每一列看作GF(2^8)上的多项式与一固定多项式 $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$ 相乘然后模多项式 x^4+1，根据矩阵的乘法运算，设 $s'(x)=a(x)\oplus s(x)$,可得到：

$$s'_{0,c}=(\{02\}\bullet s_{0,c})\oplus(\{03\}\bullet s_{1,c})\oplus s_{2,c}\oplus s_{3,c} \quad (2.4)$$

$$s'_{1,c}=s_{0,c}\oplus(\{02\}\bullet s_{1,c})\oplus(\{03\}\bullet s_{2,c})\oplus s_{3,c} \quad (2.5)$$

$$s'_{2,c}=s_{0,c}\oplus s_{1,c}\oplus(\{02\}\bullet s_{2,c})\oplus(\{03\}\bullet s_{3,c}) \quad (2.6)$$

$$s'_{3,c}=(\{03\}\bullet s_{0,c})\oplus s_{1,c}\oplus s_{2,c}\oplus(\{02\}\bullet s_{3,c}) \quad (2.7)$$

因此一列中的四个字节替换如图2-4所示：

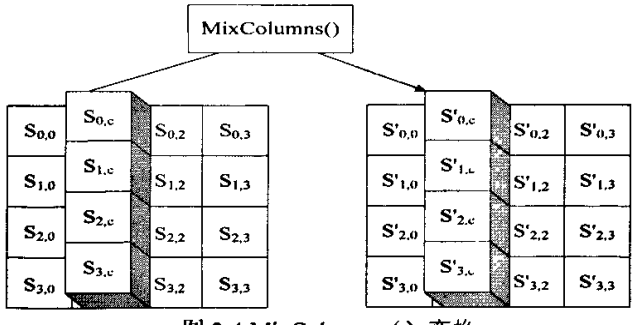


图 2-4 MixColumns（）变换

4. AddRoundkey( )变换

AddRoundKey( )变换是与扩展密钥的异或运算，即根据加密的轮数用相应的扩展密钥的四个数据项和中间状态矩阵上的列进行按位异或：

$$[s'_{0,c},s'_{1,c},s'_{2,c},s'_{3,c}]=[s_{0,c},s_{1,c},s_{2,c},s_{3,c}]\oplus[w_{round*Nb+c}]$$

具体如图2-5所示：

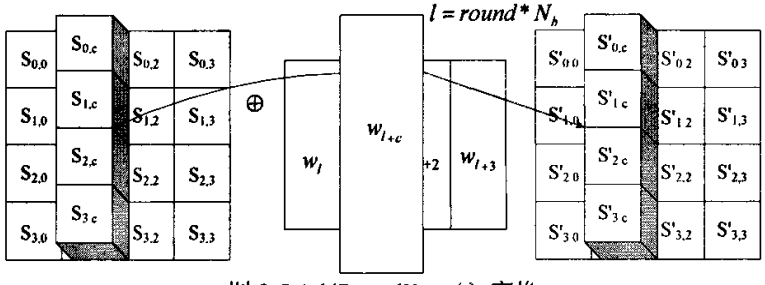


图 2-5 AddRoundKey（）变换

2.4.3 密钥调度

密钥调度有两个组成部分：密钥扩展(Key Expansion)和轮密钥选择(Round KeySelection)。其基本原则是：

(1)轮密钥的总位数等于分组长乘以(1+Nr)，如分组长为 128bit，轮数为 10，则轮密钥的总长为 128* (10+1) =1408bit；

(2)种子密钥扩展为扩展密钥，种子密钥长度为 4*N_k 个字节；

(3)轮密钥由以下方法从扩展密钥中获得:对第 1 轮密钥由前 N_b 个字构成；第 2 轮密钥由第二个 N_b 个字即第 N_b+1 个字到第 2N_b 个字构成；以下依次类推。

1. 密钥扩展(Key Expansion)

密钥扩展算法首先利用初始密钥产生出加密所需要的所有密钥，然后对该密钥序列中的除第一个和最后一个字外的密钥字进行逆列混合处理，得到解密过程所需要的所有密钥。

密钥扩展总共产生 N_b(Nr+1)个双字:算法初始需要一个 N_b 个双字的集合，接着每个轮操作都需要 N_b 个双字的密钥数据。最终的密钥流程共包含了一个 4 字节双字的线性数组，用 w[i]表示，整个密钥扩展算法用 C 伪码表示如下：

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)],Nk)
begin
    word temp
    i=0
    while (i<Nk)
        w[i]=word(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3])
          i=i+1
    end while
  i=Nk
  end if
        w[i]=w[i-Nk] xor temp
         i=i+1
    end while
end
```

从以上代码可以看出：扩展密钥的 N_k 个字由种子密钥构成，随后的字 w[i]等于字 w[i-1]经一些变换后得到的字 temp 和字 w[i-N_k]异或而成;而且对位置为 N_k 倍数的字变换中不仅运用了循环左移变换 RotByte。和子字节变换 SubByte，还运用了轮常数 Rcon。

Rcon[i]=[x^{i},{00},{00},{00}],其中 x={02}，它是 GF(2^8)上的元素，这里 i=1,2,3,...。

RotWord 操作从密钥序列中取一个字$[a_0,a_1,a_2,a_3]$，执行字节循环移位，输出字$[a_1,a_2,a_3,a_0]$。

SubWord 操作作用在 RotWord 操作后的字上，对该字中的每个字节应用 S 盒产生一个输出字$[a_1,a_2,a_3,a_0]$。

如下表 2.6、表 2.7 和表 2.8 所示分别为 128-bits、192-bits 和 256-bits 的密钥扩展。

(1)设 Cipher Key =2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c，密钥长度 N_k=4，于是得到：

w_0 =2b7e1516　w_1=28aed2a6　w_2 = abf71588　w_3=09cf 4f3c

表 2.6 128-bits 密钥扩展

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943
10	7a96b943					23a33939	5935807a
11	5935807a					2a6c7605	7359f67f
12	7359f67f	59f67f73	cb42d28f	04000000	cf42d28f	f2c295f2	3d80477d
13	3d80477d					7a96b943	4716fe3e
14	4716fe3e					5935807a	1e237e44
15	1e237e44					7359f67f	6d7a883b
16	6d7a883b	7a883b6d	dac4e23c	08000000	d2c4e23c	3d80477d	ef44a541
17	ef44a541					4716fe3e	a8525b7f
18	a8525b7f					1e237e44	b671253b
19	b671253b					6d7a883b	db0bad00
20	db0bad00	0bad00db	2b9563b9	10000000	3b9563b9	ef44a541	d4d1c6f8
21	d4d1c6f8					a8525b7f	7c839d87
22	7c839d87					b671253b	caf2b8bc
23	caf2b8bc					db0bad00	11f915bc

24	11f915bc	f915bc11	99596582	20000000	b9596582	d4d1c6f8	6d88a37a
25	6d88a37a					7c839d87	110b3efd
26	110b3efd					caf2b8bc	dbf98641
27	dbf98641					11f915bc	ca0093fd
28	ca0093fd	0093fdca	63dc5474	40000000	23dc5474	6d88a37a	4e54f70e
29	4e54f70e					110b3efd	5f5fc9f3
30	5f5fc9f3					dbf98641	84a64fb2
31	84a64fb2					ca0093fd	4ea6dc4f
32	**4ea6dc4f**	**a6dc4f4e**	**2486842f**	**80000000**	**a486842f**	**4e54f70e**	**ead27321**
33	**ead27321**					**5f5fc9f3**	**b58dbad2**
34	b58dbad2					84a64fb2	312bf560
35	312bf560					4ea6dc4f	7f8d292f
36	7f8d292f	8d292f7f	5da515d2	1b000000	46a515d2	ead27321	ac7766f3
37	ac7766f3					b58dbad2	19fadc21
38	19fadc21					312bf560	28d12941
39	28d12941					7f8d292f	575c006e
40	575c006e	5c006e57	4a639f5b	36000000	7c639f5b	ac7766f3	d014f9a8
41	d014f9a8					19fadc21	c9ee2589
42	c9ee2589					28d12941	e13f0cc8
43	e13f0cc8					575c006e	b6630ca6

(2)设 Cipher Key =8e 73 b0 f7 da 0e 64 52 c8 10 f3 2b 80 90 79 e5 62 f8 ea d2 52 2c 6b 7b ，密钥长度 N_k=6，于是得到：

w_0 =8e73b0f7　w_1 = da0e6452　w_2 = c810f32b　w_3 =809079e5

w_4 =e562f8ea　w_5 =522c6b7b

表 2.7 192-bits 密钥扩展

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
6	522c6b7b	2c6b7b52	717f2100	01000000	707f2100	8e73b0f7	fe0c91f7
7	fe0c91f7					da0e6452	2402f5a5
8	2402f5a5					c810f32b	ec12068e

9	ec12068e					809079e5	6c827f6b
10	6c827f6b					62f8ead2	0e7a95b9
11	0e7a95b9					522c6b7b	5c56fec2
12	5c56fec2	56fec25c	b1bb254a	02000000	b3bb254a	fe0c91f7	4db7b4bd
13	4db7b4bd					2402f5a5	69b54118
14	69b54118					ec12068e	85a74796
15	85a74796					6c827f6b	e92538fd
16	e92538fd					0e7a95b9	e75fad44
17	e75fad44					5c56fec2	bb095386
18	bb095386	095386bb	01ed44ea	04000000	05ed44ea	4db7b4bd	485af057
19	485af057					69b54118	21efb14f
20	21efb14f					85a74796	a448f6d9
21	a448f6d9					e92538fd	4d6dce24
22	4d6dce24					e75fad44	aa326360
23	aa326360					bb095386	113b30e6
24	113b30e6	3b30e611	e2048e82	08000000	ea048e82	485af057	a25e7ed5
25	a25e7ed5					21efb14f	83b1cf9a
26	83b1cf9a					a448f6d9	27f93943
27	27f93943					4d6dce24	6a94f767
28	6a94f767					aa326360	c0a69407
29	c0a69407					113b30e6	d19da4e1
30	d19da4e1	9da4e1d1	5e49f83e	10000000	4e49f83e	a25e7ed5	ec1786eb
31	ec1786eb					83b1cf9a	6fa64971
32	6fa64971					27f93943	485f7032
33	485f7032					6a94f767	22cb8755
34	22cb8755					c0a69407	e26d1352
35	e26d1352					d19da4e1	33f0b7b3
36	33f0b7b3	f0b7b333	8ca96dc3	20000000	aca96dc3	ec1786eb	40beeb28
37	40beeb28					6fa64971	2f18a259
38	2f18a259					485f7032	6747d26b
39	6747d26b					22cb8755	458c553e
40	458c553e					e26d1352	a7e1466c
41	a7e1466c					33f0b7b3	9411f1df
42	9411f1df	11f1df94	82a19e22	40000000	c2a19e22	40beeb28	821f750a
43	821f750a					2f18a259	ad07d753

(3)设 Cipher Key =60 3d eb 10 15 ca 71 be 2b 73 ae f0 85 7d 77 81 1f 35 2c 07 3b 61 08 d7 2d 98 10 a3 09 14 df f4 ，密钥长度 N_k=8，于是得到：

w_0 =603deb10 w_1 = 15ca71be w_2 = 2b73aef0 w_3 =857d7781

w_4 =1f352c07 w_5 =3b6108d7 w_6 =2d9810a3 w_7 =0914dff4

表 2.8 256-bits 密钥扩展

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
8	0914dff4	14dff409	fa9ebf01	01000000	fb9ebf01	603deb10	9ba35411
9	9ba35411					15ca71be	8e6925af
10	8e6925af					2b73aef0	a51a8b5f
11	a51a8b5f					857d7781	2067fcde
12	2067fcde		b785b01d			1f352c07	a8b09c1a
13	a8b09c1a					3b6108d7	93d194cd
14	93d194cd					2d9810a3	be49846e
15	be49846e					0914dff4	b75d5b9a
16	b75d5b9a	5d5b9ab7	4c39b8a9	02000000	4e39b8a9	9ba35411	d59aecb8
17	d59aecb8					8e6925af	5bf3c917
18	5bf3c917					a51a8b5f	fee94248
19	fee94248					2067fcde	de8ebe96
20	de8ebe96		1d19ae90			a8b09c1a	b5a9328a
21	b5a9328a					93d194cd	2678a647
22	2678a647					be49846e	98312229
23	98312229					b75d5b9a	2f6c79b3
24	2f6c79b3	6c79b32f	50b66d15	04000000	54b66d15	d59aecb8	812c81ad
25	812c81ad					5bf3c917	dadf48ba
26	dadf48ba					fee94248	24360af2
27	24360af2					de8ebe96	fab8b464
28	fab8b464		2d6c8d43			b5a9328a	98c5bfc9
29	98c5bfc9					2678a647	bebd198e
30	bebd198e					98312229	268c3ba7
31	268c3ba7					2f6c79b3	09e04214
32	09e04214	e0421409	e12cfa01	08000000	e92cfa01	812c81ad	68007bac
33	68007bac					dadf48ba	b2df3316
34	b2df3316					24360af2	96e939e4
35	96e939e4					fab8b464	6c518d80

36	6c518d80		50d15dcd			98c5bfc9	c814e204
37	c814e204					bebd198e	76a9fb8a
38	76a9fb8a					268c3ba7	5025c02d
39	5025c02d					09e04214	59c58239
40	59c58239	c5823959	a61312cb	10000000	b61312cb	68007bac	de136967
41	de136967					b2df3316	6ccc5a71
42	6ccc5a71					96e939e4	fa256395
43	fa256395					6c518d80	9674ee15
44	9674ee15		90922859			c814e204	5886ca5d
45	5886ca5d					76a9fb8a	2e2f31d7
46	2e2f31d7					5025c02d	7e0af1fa
47	7e0af1fa					59c58239	27cf73c3
48	27cf73c3	cf73c327	8a8f2ecc	20000000	aa8f2ecc	de136967	749c47ab
49	749c47ab					6ccc5a71	18501dda
50	18501dda					fa256395	e2757e4f
51	e2757e4f					9674ee15	7401905a
52	7401905a		927c60be			5886ca5d	cafaaae3
53	cafaaae3					2e2f31d7	e4d59b34
54	e4d59b34					7e0af1fa	9adf6ace
55	9adf6ace					27cf73c3	bd10190d
56	bd10190d	10190dbd	cad4d77a	40000000	8ad4d77a	749c47ab	fe4890d1
57	fe4890d1					18501dda	e6188d0b
58	e6188d0b					e2757e4f	046df344
59	046df344					7401905a	706c631e

2. 轮密钥选择(Round KeySelection)

由于在进行密钥加时，密钥长必须与分组长相等，因此第 i 轮的密钥与分组长度有关，并且由扩展密钥的字构成。如图 2-6 所示轮密钥 i 是由轮密钥缓冲字 W[N_b*i]到 W[N_b*(i+1)]给出的。

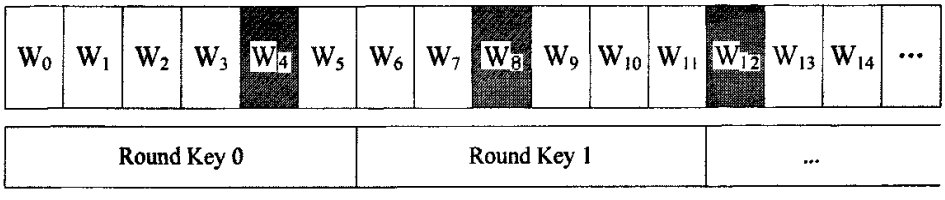
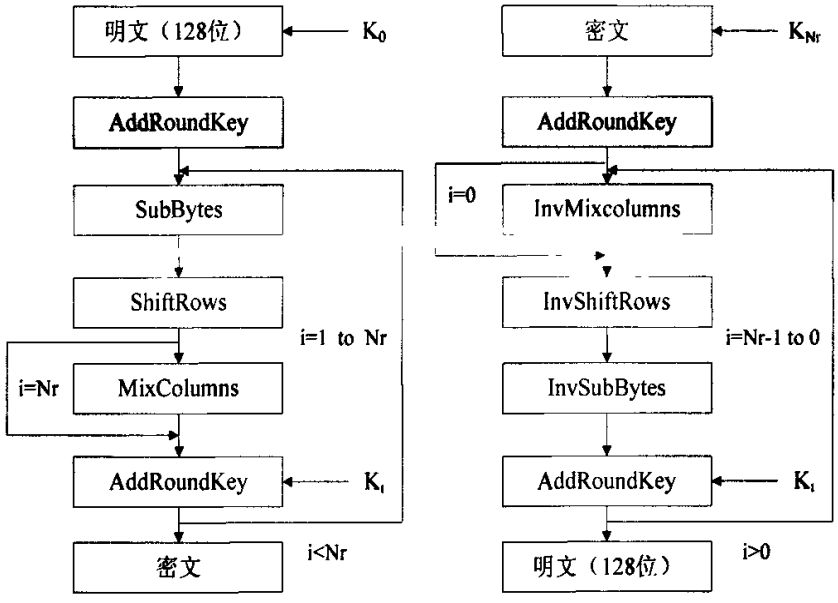


图 2-6 N_b=6，N_k=4 密钥扩展与轮密钥选择

2.4.4 AES 加密算法

AES 加密的每一轮由三层组成：非线性层，由 16 个 S－盒并置而成，进行 Subbytes( ) 运算，起到混淆的作用；线性混合层，进行 ShiftRows( )运算和 Mixcolumns( )运算，确保多轮之上的高度扩散；密钥加层，进行 AddRoundKey( )运算，子密钥简单的异或到中间状态上[2]。其加密和解密算法原理流程图如图 2-7 所示：



(a)加密算法流程　　(b)解密算法流程

图2-7 加密和解密算法原理流程图

AES加密过程包括一个初始密钥加法，记作AddRoundkey ，接着进行Nr-1次轮变化round，最后再使用一个轮变换。具体加密流程C伪码表示如下：

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4,Nb]
state=in
AddRoundkey(state,w[0,Nb-1])
for round=1 step 1 to Nr-1
  SubBytes(state)
  ShiftRows(state)
  MixColumns(state)
AddRoundkey( state, w[round*Nb, (round+1 )*Nb-1])
enf for
SubBytes(state)
ShiftRows(state)
AddRoundkey(stae,w[Nr*Nb, (Nr+1)*Nb-1])
out=state
end
```

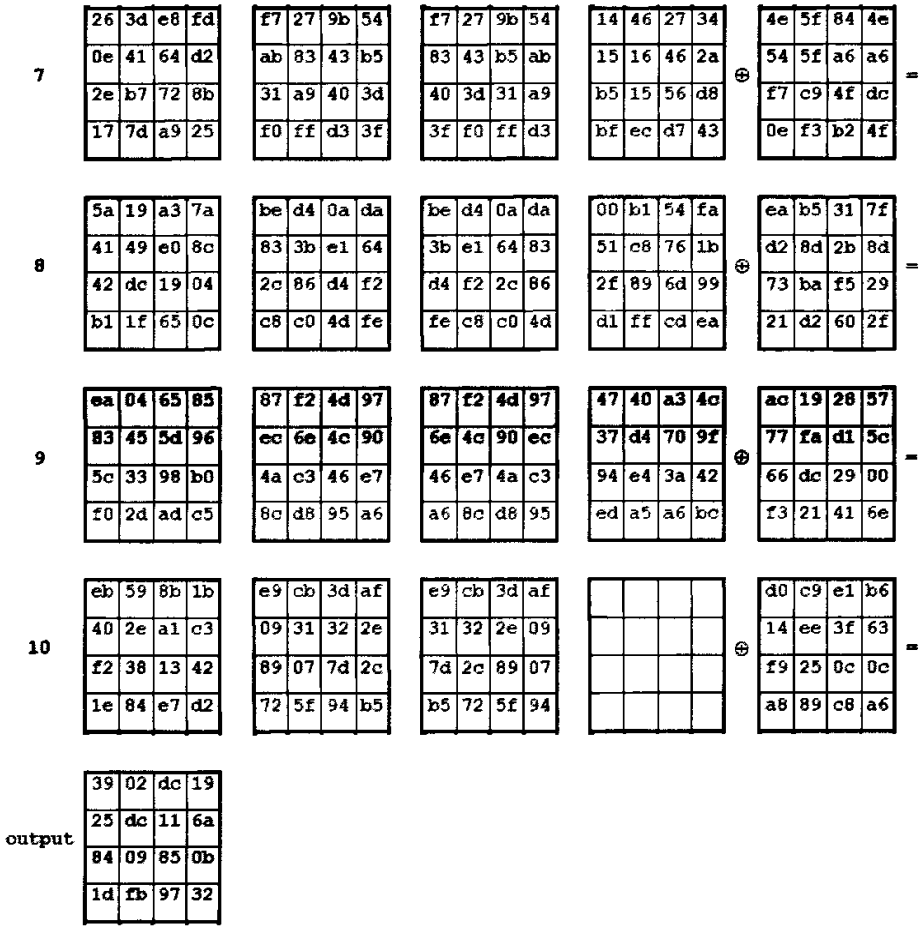
下面我们对以128-bits为例，说明AES加密过程。其中$N_b=4$，$N_k=4$。

设初始明文输入input=32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

密钥为Cipher Key=2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

轮密钥在密钥扩展运算如2.4.3节所述，加密过程如下：

Round Number	Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key Value
input	32 88 31 e0 43 5a 31 37 f6 30 98 07 a8 8d a2 34				⊕ 2b 28 ab 09 7e ae f7 cf 15 d2 15 4f 16 a6 88 3c =
1	19 a0 9a e9 3d f4 c6 f8 e3 e2 8d 48 be 2b 2a 08	d4 e0 b8 1e 27 bf b4 41 11 98 5d 52 ae f1 e5 30	d4 e0 b8 1e bf b4 41 27 5d 52 11 98 30 ae f1 e5	04 e0 48 28 66 cb f8 06 81 19 d3 26 e5 9a 7a 4c	⊕ a0 88 23 2a fa 54 a3 6c fe 2c 39 76 17 b1 39 05 =
2	a4 68 6b 02 9c 9f 5b 6a 7f 35 ea 50 f2 2b 43 49	49 45 7f 77 de db 39 02 d2 96 87 53 89 f1 1a 3b	49 45 7f 77 db 39 02 de 87 53 d2 96 3b 89 f1 1a	58 1b db 1b 4d 4b e7 6b ca 5a ca b0 f1 ac a8 e5	⊕ f2 7a 59 73 c2 96 35 59 95 b9 80 f6 f2 43 7a 7f =
3	aa 61 82 68 8f dd d2 32 5f e3 4a 46 03 ef d2 9a	ac ef 13 45 73 c1 b5 23 cf 11 d6 5a 7b df b5 b8	ac ef 13 45 c1 b5 23 73 d6 5a cf 11 b8 7b df b5	75 20 53 bb ec 0b c0 25 09 63 cf d0 93 33 7c dc	⊕ 3d 47 1e 6d 80 16 23 7a 47 fe 7e 88 7d 3e 44 3b =
4	48 67 4d d6 6c 1d e3 5f 4e 9d b1 58 ee 0d 38 e7	52 85 e3 f6 50 a4 11 cf 2f 5e c8 6a 28 d7 07 94	52 85 e3 f6 a4 11 cf 50 c8 6a 2f 5e 94 28 d7 07	0f 60 6f 5e d6 31 c0 b3 da 38 10 13 a9 bf 6b 01	⊕ ef a8 b6 db 44 52 71 0b a5 5b 25 ad 41 7f 3b 00 =
5	e0 c8 d9 85 92 63 b1 b8 7f 63 35 be e8 c0 50 01	e1 e8 35 97 4f fb c8 6c d2 fb 96 ae 9b ba 53 7c	e1 e8 35 97 fb c8 6c 4f 96 ae d2 fb 7c 9b ba 53	25 bd b6 4c d1 11 3a 4c a9 d1 33 c0 ad 68 8e b0	⊕ d4 7c ca 11 d1 83 f2 f9 c6 9d b8 15 f8 87 bc bc =
6	f1 c1 7c 5d 00 92 c8 b5 6f 4c 8b d5 55 ef 32 0c	a1 78 10 4c 63 4f e8 d5 a8 29 3d 03 fc df 23 fe	a1 78 10 4c 4f e8 d5 63 3d 03 a8 29 fe fc df 23	4b 2c 33 37 86 4a 9d d2 8d 89 f4 18 6d 80 e8 d8	⊕ 6d 11 db ca 88 0b f9 00 a3 3e 86 93 7a fd 41 fd =



从上述加密过程可看出，首先分别把输入（input）的明文数组和初始密钥数组转换为二维数组，依次按列存放，以四字节为一基本单位，则128位数据在二维数组中表示为四行四列，如加密过程开始的两个四行四列的二维数组所示。对于例子中的128位初始密钥而言，完成一次加密过程总共需要进行10轮变换，其中前9轮分别进行字节替换（SubBytes）、行移位（ShiftRows）、列混合（MixColumns）和轮密钥加（MixColumns后的结果与Round key的值相异或），最后一轮分别进行字节替换（SubBytes）、行移位（ShiftRows）和轮密钥加，而没有列混合。整个加密的过程首先是先把明文和初始密钥进行异或加运算，将运算的结果进行字节替换，即将二维数组中的十六进制数通过映射到S盒进行查表，得出它们的字节替换后的结果，比如十六进制数“19”，按x=1，y=9，查S盒（表2.4）可得出字节替换结果为十六进制数“d4”；将字节替换后的结果进行行移位，即第一行不变，第二行左移一位，第三行左移两位，第四行左移三位；将行移位的结果接着进行列混合，列混合是把每一列看成一个多项式，依次与一个固定的多项式 $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$ 进行相乘运算，如果结果次数大于4，则将结果再模一个不可约多项式 x^4+1；最后将列混合的结果与每一轮密钥扩展后的子密钥进行异或加。在进行了10轮变换后最后将加密的密文进行输出，如二维数组output所示。

2.4.5 AES 解密算法

解密过程是加密过程的逆转，解密流程中分别运用了InvShiftRows( )、InvSubBytes( )、InvMixColumns( )和AddRoundKey( )四个变换，对状态矩阵进行了处理。其算法是加密算法反序执行，结构类似。

Rijndael解密算法的伪C代码描述如下：

```
I_Rijndael(State,CipherKey)
{
 I_KeyExpansion(CipherKey,I_ExpandedKey) ;
 AddRoundKey(State,I_ExpandedKey+ Nb*Nr);
 For( i=Nr-1 ; i>0 ; i-- )I_Round(State,I_ExpandedKey+Nb*i) ;
 I_FinalRound(State,I_ExpandedKey);
}
```

逆密钥扩展的伪C代码描述如下：

```
I_KeyExpansion(CipherKey,I_ExpandedKey)
{
 KeyExpansion(CipherKey,I_ExpandedKey);
 for( i=1 ; i < Nr ; i++ )
 InvMixColumn(I_ExpandedKey + Nb*i) ;
}
```

解密过程中的逆轮变换定义为：

```
I_Round(State,I_RoundKey)
{
 InvShiftRow(State);
 InvByteSub(State);
 AddRoundKey(State,I_RoundKey);
 InvMixColumn(State);
}
```

解密过程中的最后一轮逆轮变换略有不同，定义为：

```
I_FinalRound(State,I_RoundKey)
{
 InvShiftRow(State);
 InvByteSub(State);
 AddRoundKey(State,I_RoundKey);
}
```

解密过程中的逆变换：

1. 逆字节代替变换InvSubBytes( )

逆字节代替变换是SubBytes( )的逆变换，是应用SubBytes( )中表的逆表的字节代替，这通过先用仿射变换的逆变换作用，再在GF(2^8)中取乘法逆得到。同样通过建立并查找逆S盒实现。SubBytes( )中用到的逆S－盒如表2.9所示：

表2.9 逆S－盒（十六进制形式）

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

2. 逆行移位变换 InvShiftRows( )

逆行移位变换是 ShiftRows( )的逆变换，它对一个状态的每一行循环右移不同的位移量。第0行不移动，第一行循环右移C_1个字节，第二行循环右移C2个字节，第三行循环右移C_3个字节。位移量(C_1，C_2，C_3)的选取与Nb有关，当Nb=4或6时一般取(C_1，C_2，C_3)=(1，2，3)。如图2-8所示：

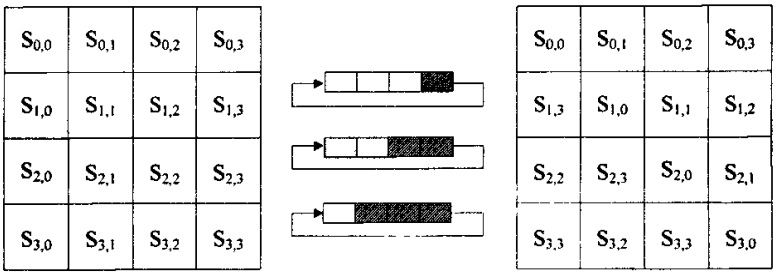


图 2-8 InvShiftRows（）变换

3. 逆列混合变换 InvMixColumns( )

逆列混合变换是 MixColumns( )的逆变换，InvMixColumns()变换是一列一列地对状态矩阵进行变换，它把每一列视为一个四项多项式，每一列被视为系数在GF(2^8)上的多项式，与一个固定多项式$a^{-1}(x)$相乘后并对x^4+1取模。$a^{-1}(x)$由下式定义：

$$a^{-1}(x)='0b'x^3+'0d'x^2+'09'x+'0e' \tag{2.8}$$

由于 $$s'(x)=a^{-1}(x)\otimes s(x) \tag{2.9}$$

由此得出矩阵乘法形式为：

$$\begin{bmatrix} s'_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{2.10}$$

作为乘法的结果，一列中的四个字节用下式代替：

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{0,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{0,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

4. 密钥加AddRoundKey( )

AddRoundKey是自身的逆。

2.5 AES 算法的优点和局限性

1. 优点

(1) 具有最佳的差分特征概率和最佳线性逼进偏差，分析结果显示7轮以上的AES对“Square”攻击是免疫的，能抵抗现有的已知密码的攻击；

(2) 没有复杂的算术运算，在各种软硬件平台上都具有良好的性能；

(3) 该算法完全是“自力更生”的，没有和其他密码进行变换，密码设计简单紧凑；

(4) 具有可变的分组长度（128－bit，192－bit，256－bit），分组长度和密钥长度可以步长32－bit为单位在128到256－bit内进行扩充变化；

(5) 既消除了DES中出现的弱密钥和半弱密钥存在的可能性，又消除了在IDEAL中发现的弱密性，对密钥的选择没有限制；

(6) 该算法没有将其安全性或部分安全性建立在算术运算之间模糊的和不好理解的相互作用的基础之上；

(7) 在安全性方面，重码率低，出现相同号码的概率小于10^{-19}

2. 局限性

(1) 在软件实现中，该算法及其逆运算使用不同的代码和 S 盒。

(2) 在硬件实现中，该算法的逆运算只能共用实现正运算的部分电路。

2.6 小结

本章对 AES 算法的设计原则、数学基础知识、整体结构和 AES 算法描述进行详细的介绍，在 AES 算法的描述中对 AES 加密/解密算法的原理及各部分算法变换的进行了数学推导及伪代码描述，最后对 AES 算法优点和存在的局限性进行了总结。

第三章 基于 ARM 核嵌入式系统的 AES 算法优化设计

3.1 优化设计方案

AES 算法可以用软件或硬件来实现。硬件实现的优点是可获得高速率，而软件实现的优点是灵活性强、代价低。基于软件和硬件的不同性质，AES 的设计原则可根据预定的实现方法来考虑。基本的要求是：设计简单、结构合理、易于用软件或硬件实现，即有着良好的实现性。

一个密码算法若要用软件实现，则尽量使密码运算针对某一长度的子块进行，子块的长度应尽可能的适应软件编程，如采用 8 位、16 位、32 位的子块，这是因为在软件实现中，单个比特之间的置换是难于实现的;除了使用子块外，密码算法应采用一些易于软件实现的运算，如标准处理器能直接处理的加法、乘法、移位运算等。密码算法若是要用硬件实现，那么就要求密码算法的结构尽可能紧凑，轮变换尽可能一致，这样便于用一个功能模块同时实现加密和解密过程。

AES 算法的运算是针对字节进行的，密码运算比较简单，并且结构紧凑，每轮变换也基本一致，所以易于用软件或是硬件实现，为了使算法设计能够适应 ARM 核的 ADS 开发环境，在进行算法设计时考虑方案如下：

1. 算法要同时支持 128bit、192bit 和 256bit 三种密钥长度；

2. 优化[19]算法核心运算部分，优化加解密中轮函数算法。

3. 在存储器空间允许下，设计时涉及到大量数据时尽可能采用查表方式。以提高算法执行效率。即在 S－box 设计时，可先计算出所有的 X 和 Y 的值，把 S 表存在存储空间里。

4. 在加、解密之前先进行密钥扩展，把密钥扩展中计算出的所有结果都存储在 ARM 核中 SRAM 存储器中，以便加/解密时随时可调用，从而加快加/解密速度。

5. 由于 ARM 体系[20-26]支持 32 位的字长，故采用 32 位设计输入，以四字节为一基本单位。

6. 在程序设计时，利用 ARM 的体系特点采用自动递增递减的寻址方式，以优化程序循环。同时 load 和 store 多条指令，以增加数据吞吐率。

3.2 AES 算法整体结构设计

AES 算法整体设计采用自顶而下的设计方法，根据算法原理[27-39]，为了节省内存空间，加快密钥扩展的速度，在设计中采用了密钥调度模块，与通用的硬件密码系统相比，我们不包括密钥发生器，所用的密钥是通过输入接口输入，加解密后的数据经由输出接口输出。密钥扩展模块负责对输入的初始密钥进行扩展，并将扩展的结果按一定的方式传送到加/解密运算模块。输入接口单元用来输入数据和密钥，控制单元用来输出所有的控制信号。我们将 AES 算法实现分为以下五个模块，其设计结构如图 3-1 所示：

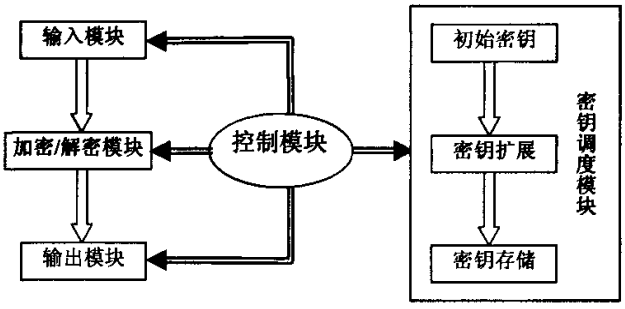


图 3-1 AES 算法设计结构图

1. **输入模块：用来输入和存储加/解密的数据和密钥；**
2. **密钥调度模块：对初始密钥进行密钥扩展和加/解密过程中各轮子密钥的存储；**
3. 加密/解密模块：加密模块是对明文进行加密运算，解密模块是对加密过的数据进行解密运算，在控制信号作用下把结果送到输出模块；
4. 控制模块：产生输入/输出接口模块、密钥调度模块及加/解密模块等模块的控制信号；
5. 输出模块：用来输出和存储加/解密的数据和密钥。

3.3 各模块的具体优化设计

3.3.1 输入输出接口模块的设计

输入输出模块数据传输主要是采用 32 位设计，以四个字节为基本单位。如果输入 128 位数据，我们就使用四个并行的 32 位的寄存器进行暂存或传输。对加密来说，输入是一个明文分组和一个密钥，输出是一个密文分组。对解密而言，输入是一个密文分组和一个密钥，而输出是一个明文分组。在控制信号的作用下，其输入输出接口结构图如图 3-2 所示：

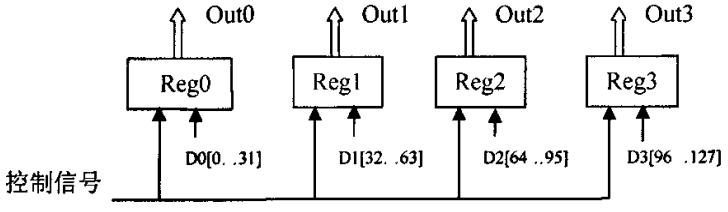


图 3-2 输入输出接口结构

3.3.2 密钥扩展模块的设计

密钥扩展模块分两部分，产生加密密钥部分和解密密钥部分。无论是加密密钥还是解密密钥，它们都分别经过字节代换、字节移位、计算圈常数、计算密钥这四步运算。为了节省 ARM 核处理器的内存，所以密钥扩展采用内部扩展，密钥扩展与加/解密运算并行执行，即给定初始密钥后，在进行每一轮的运算过程的同时算出下一轮的密钥，并把下一轮的密钥扩展

的运算如图 3-3 所示：

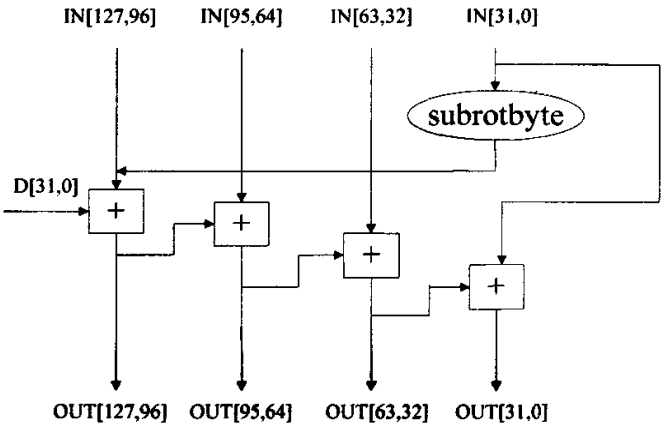


图 3-3 密钥扩展运算

IN[127,0]为输入密钥，OUT[127,0]是产生的扩展密钥。Subrotbyte 变换如图图 3-4 中所示，将进行 Rotword（一字节循环左移），Subword（S 盒替换）等操作，然后再将操作结果与轮常数 Rcon[k]异或，将异或后的结果与 w[i-Nk]的密钥异或即产生出新的密钥，具体密钥扩展流程如图 3-5 所示：

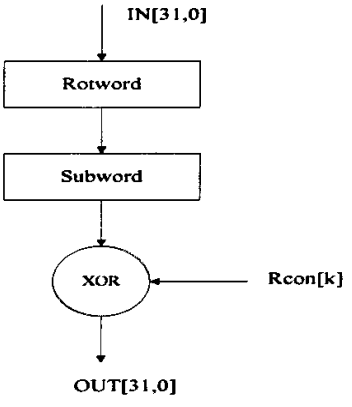


图 3-4 Subrotbyte 变换

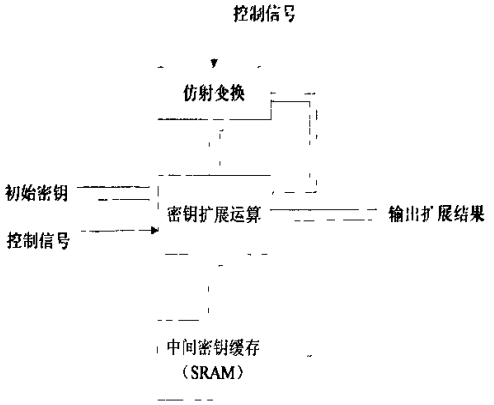


图 3-5 密钥扩展流程图

3.3.3 控制模块的设计

在控制模块设计时是由ARM处理器根据不同的时序向各个模块发出控制信号。在图3-6控制模块的输入输出端口示意图中，clk是来自ARM处理器的全局时钟，rst是来自ARM处理器的系统恢复信号，当key_in为高时，表示为有密钥输入信号，它来自ARM寄存器，当data_in为时，表示有新数据输入信号，它也来自ARM寄存器，当alg_work为高时，表示加密/解密工作模块使能有效，当keyexp_work为高时，表示密钥扩展工作模块使能有效。

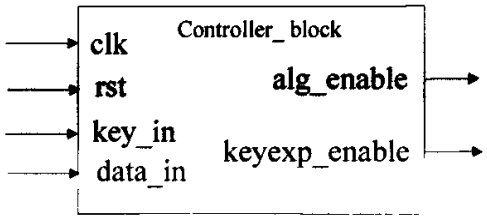


图 3-6 控制模块的输入输出端口示意图

具体ARM对数据流的传输控制流程如图3-7所示，ARM产生的时序控制信号通过数据通路对各个模块分别进行握手传输数据，即主要是对加/解密模块和密钥扩展模块以及数据输入输出的控制。当有新的密钥输入时，控制模块对密钥扩展模块进行启动运行。当有新明文数据输入时，控制模块启动数据通路传输数据至加/解密模块，进行数据加密/解密运算。

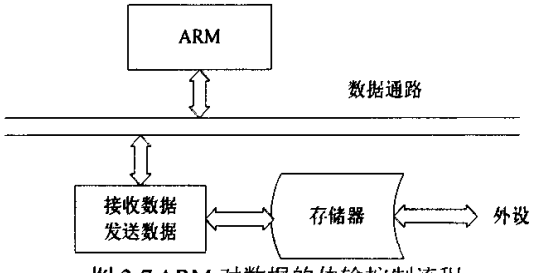


图 3-7 ARM 对数据的传输控制流程

3.3.4 加密模块的设计

加密模块是整个模块的重要部分之一，加/解密阶段都需要进行九轮至十轮的函数循环，加密前九轮中每一轮都包括 SubBytes( )变换、ShiftRows( )变换、MixColumns( )变换以及AddRoundKey( )变换，最后一轮不包括 MixColumns( )变换。其单轮加密的结构如图 3-8 所示：

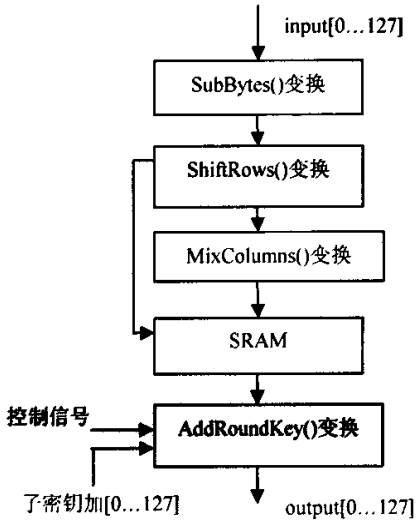


图 3-8 单轮加密结构图

1. 加密模块中 SubBytes( )变换优化设计

由 2.2.2 节可知，SubBytes( )变换即 S—盒运算是可逆转的，它通过求乘逆和仿射变换得到如式子（3.1）：

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.1}$$

从以上式子看，直接实现需要较大的运算，为优化实现我们可通过以下式子循环移位实现。

$$b' = \begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_0 \end{bmatrix} + \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_0 \\ b_1 \end{bmatrix} + \begin{bmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_0 \\ b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.2}$$

$$= b + (b << 1) + (b << 2) + (b << 3) + (b << 4) + '63'$$

2. 加密模块中 MixColumns( )变换的优化设计

由 2.2.2 节可知 MixColumns( )变换中可得到如下矩阵式子：

$$\begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{3.3}$$

转换为：

$$\Rightarrow \quad \begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 2s_{0,c} \\ s_{0,c} \\ s_{0,c} \\ 3s_{0,c} \end{bmatrix} + \begin{bmatrix} 3s_{1,c} \\ 2s_{1,c} \\ s_{1,c} \\ s_{1,c} \end{bmatrix} + \begin{bmatrix} s_{2,c} \\ 3s_{2,c} \\ 2s_{2,c} \\ s_{2,c} \end{bmatrix} + \begin{bmatrix} s_{3,c} \\ s_{3,c} \\ 3s_{3,c} \\ 2s_{3,c} \end{bmatrix}$$

$$= 2\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} + 3\begin{bmatrix} s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{0,c} \end{bmatrix} + \begin{bmatrix} s_{2,c} \\ s_{3,c} \\ s_{0,c} \\ s_{1,c} \end{bmatrix} + \begin{bmatrix} s_{3,c} \\ s_{0,c} \\ s_{1,c} \\ s_{2,c} \end{bmatrix} \tag{3.4}$$

$$= 2s + 3(s << 1) + (s << 2) + (s << 3)$$

根据推导的式子，我们发现算法优化了，可用循环移位和倍乘运算（3s=2s+s 即 3s 是倍乘 2s 与 s 的和）来实现。

3. 加密模块中轮函数优化设计

轮函数是整个加/解密算法的复杂且核心部分。为了便于实现加密运算，我们对轮函数进行优化。我们知道每一轮操作包括以下四步变换：SubBytes( )变换、ShiftRows( )变换、MixColumns( )变换以及 AddRoundKey( )变换。我们可以把这四步优化合并成一个简单的查表操作。具体优化方法如下：

设一个字节的 $S_{r,c}$ 的 S 盒操作为 $S[S_{r,c}]$，那么状态阵列中一列经过 SubBytes( )变换和 ShiftRows( )变换，得到：

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} S[s_{0,c}] \\ S[s_{1,c}] \\ S[s_{2,c}] \\ S[s_{3,c}] \end{bmatrix} \quad , \quad \begin{bmatrix} s''_{0,c} \\ s''_{1,c} \\ s''_{2,c} \\ s''_{3,c} \end{bmatrix} = \begin{bmatrix} S[s_{0,c(0)}] \\ S[s_{1,c(1)}] \\ S[s_{2,c(2)}] \\ S[s_{3,c(3)}] \end{bmatrix} \tag{3.5}$$

将（3.5）式接着经过 MixColumns()变换和 AddRoundKey()变换，得到：

$$\begin{bmatrix} s''_{0,c} \\ s''_{1,c} \\ s''_{2,c} \\ s''_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[s_{0,c(0)}] \\ S[s_{1,c(1)}] \\ S[s_{2,c(2)}] \\ S[s_{3,c(3)}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,c} \\ k_{1,c} \\ k_{2,c} \\ k_{3,c} \end{bmatrix} \tag{3.6}$$

(3.6)式的等效变换如下式所示：

$$\begin{bmatrix} s''_{0,c} \\ s''_{1,c} \\ s''_{2,c} \\ s''_{3,c} \end{bmatrix} = S\left[s_{0,c(0)}\right]\begin{bmatrix}02\\01\\01\\03\end{bmatrix} \oplus S\left[s_{1,c(1)}\right]\begin{bmatrix}03\\02\\01\\01\end{bmatrix} \oplus S\left[s_{2,c(2)}\right]\begin{bmatrix}01\\03\\02\\01\end{bmatrix} \oplus S\left[s_{3,c(3)}\right]\begin{bmatrix}01\\01\\03\\02\end{bmatrix} \oplus \begin{bmatrix} k_{0,c} \\ k_{1,c} \\ k_{2,c} \\ k_{3,c} \end{bmatrix} \quad (3.7)$$

由（3.7）式可构成以下四张表：

$$T_{0[x]} = \begin{bmatrix} S[x]02 \\ S[x] \\ S[x] \\ S[x]03 \end{bmatrix} \qquad T_{1[x]} = \begin{bmatrix} S[x]03 \\ S[x]02 \\ S[x] \\ S[x] \end{bmatrix}$$

$$T_{2[x]} = \begin{bmatrix} S[x] \\ S[x]03 \\ S[x]02 \\ S[x] \end{bmatrix} \qquad T_{3[x]} = \begin{bmatrix} S[x] \\ S[x] \\ S[x]03 \\ S[x]02 \end{bmatrix}$$

因此轮函数的变换可表示为：

$$\begin{bmatrix} s''_{0,c} & s''_{1,c} & s''_{2,c} & s''_{3,c} \end{bmatrix}^T = T_0\left[s_{0,c(0)}\right] \oplus T_1\left[s_{1,c(1)}\right] \oplus T_2\left[s_{2,c(2)}\right] \oplus T_3\left[s_{3,c(3)}\right] \oplus k_{round,c} \quad (3.8)$$

其中 C(r)=[c+h(r,N_b)]modN_b，h(r, N_b)为 r 行行移位的字节数。$K_{round,c}$ 是 C 列所对应的轮密钥。

以上每个表占 1k 字节，四个表占 4K 字节，因此每一轮的变换只需要 4 次查表和 4 次异或操作，这个优化方法大大提高了算法的效率。我们分析可知，

$$T_i[x] = Rotbyte(T_{i-1}[x]) \quad (3.9)$$

从而得到以下公式：

$$\begin{bmatrix} s''_{0,c} & s''_{1,c} & s''_{2,c} & s''_{3,c} \end{bmatrix}^T = k_{round,c} \oplus T_0\left[s_{0,c(0)}\right] \oplus Rotbyte\left(T_0\left[s_{1,c(1)}\right]\right) \oplus Rotbyte\left(Rotbyte\left(T_0\left[s_{2,c(2)}\right]\right)\right) \oplus Rotbyte\left(Rotbyte\left(Rotbyte\left(T_0\left[s_{3,c(3)}\right]\right)\right)\right) \quad (3.10)$$

因此，由上分析可知，只要构造出了一张 T_0 表，其余三个表通过依次字节移位就易得出了。我采用了此优化方法。

3.3.5 解密模块设计

解密模块是加密模块的逆[27-30]。即前九轮先分别进行 InvShiftRows( )变换、InvSubBytes( )变换、AddRoundKey( )变换和 InvMixColumns( )变换，最后一轮分别进行 InvByteSub( )变换、InvShiftRow( )变换、AddRoundKey( )变换，同样不包括 InvMixColumns( )变换。在轮函数实现部分进行了等效解密优化，在 3.5.2 节中进行详细介绍。

1. 解密模块中 InvMixColumns( )变换优化设计

由 2.2.5 节中可知，InvMixColumns( )变换的得到的矩阵如下：

$$\begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \tag{3.11}$$

上式可转换为：

$$\Rightarrow \begin{bmatrix} s_{0,c}' \\ s_{1,c}' \\ s_{2,c}' \\ s_{3,c}' \end{bmatrix} = \begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \left(\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \right) \tag{3.12}$$

设把式子（3.12）中括号内的运算结果看成一列，它与上括号外的矩阵可转换为：

$$\begin{bmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a+4(a+c) \\ b+4(b+d) \\ c+4(c+a) \\ d+4(d+b) \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} + \begin{bmatrix} 4(a+c) \\ 4(b+d) \\ 4(c+a) \\ 4(d+b) \end{bmatrix} \tag{3.13}$$

由上式可看出，此逆运算可优化为异或运算及乘运算来实现，大大提高了算法的运算效率。

2. 解密模块中轮函数优化设计

在解密阶段，轮函数的变换顺序为：

{InvShiftRows( )变换
InvSubBytes( )变换
AddRoundKey( )变换
InvMixColumns( )变换
}

这与加密阶段不一致，为了使加密阶段的代码能在解密阶段可重用，我们采用了对算法结构进行优化，具体算法优化方法为：

(1) InvShiftRows 和 InvSubBytes 的次序可互换。

由于 InvShiftRows 只对字节进行换位，而不影响字节的值。InvSubBytes 作用在单字节

上，而与这些字节的位置无关。因此这两步可互相交换。

即 InvShiftRows(InvSubBytes( ))=InvSubBytes(InvShiftRows( ))，即先进行 InvSubBytes( ) 变换后进行 InvShiftRows( )变换与先进行 InvShiftRows( )变换后进行 InvSubBytes( )变换等价。

(2)如果调整相应的轮密钥，AddRoundKey 和 InvMixColumns 的次序可互换。

因为对于任意线性交换 $A: x \to y = A(k)$，有定义：

$$A(x \oplus k) = A(x) \oplus A(k) \qquad (3.14)$$

由于步骤 AddRoundKey 仅将常量 ExpandedKey[i]与其输入相加，而且 InvMixColumns 是一个线性运算，因此步骤

这样，等效的解密算法轮函数的次序变换为：

{InvSubBytes( )变换

InvShiftRows( )变换

InvMixColumns( )变换

AddRoundKey( )变换

}

因此结构与加密算法顺序一致。从而使算法的结构得到了优化。

3.4 小结

本章首先针对 ARM 核提出了系统优化方案，然后介绍了 AES 的整体设计模块以及各模块的具体设计，在各模块具体设计中进行了算法本身及结构上的部分优化，使原来的复杂运算变为简单的移位、异或及乘的操作，从而节省了时间及空间，提高了系统的性能。

第四章 基于ARM核嵌入式系统的AES算法的实现

4.1 实现方案

4.1.1 硬件方案

硬件选择方面，考虑到系统的可实现性，以及系统的可扩展性。在本次设计硬件电路中，选择使用以ARM7TDMI为其CPU的SEP3203微处理器来实现AES加密/解密中的核心算法，**如密钥扩展运算可以用ARM中的运算器进行运算，主要进行的是移位运算和加算法，而ARM中提供的运算器的功能有移位和加，所以用硬件实现能够提高整个算法运算的执行效率。**同时在设计中再辅以适当的外围电路来实现。因为嵌入式CPU具有体积小、质量轻、成本低、可靠性高的优点。和通用CPU相比较，在同样的速度下，嵌入式CPU可以少用30%的硬件，这表明嵌入式CPU价格更便宜，或者说在同样的集成度下效率更高。而且，我们打算以后从事实现某一功能的专用芯片，因此在通用CPU和嵌入式CPU之间我们选择使用嵌入式CPU。

1.SEP3203微处理器

SEP3203微处理器是由东南大学国家ASIC工程中心针对低成本、低功耗手持多媒体应用的SOC芯片设计而成的，它采用的是0.25um标准的CMOS的设计工艺，使用ARM7TDMI作为该SOC的CPU内核。它的处理器结构如图4-1所示：

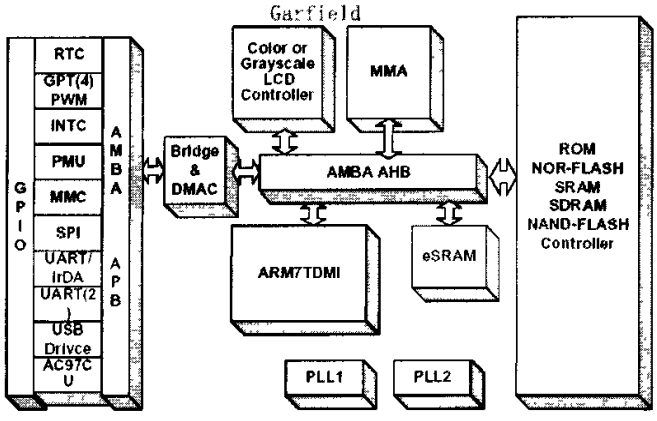


图4-1 SEP3203微处理器结构

它的性能指标为：

- ARM7TDMI处理器内核；
- AMBA 2.0片上总线；
- 主频：75MHz～100MHz；
- 电源：2.5V(内核)以及3.3V（IO）；
- 功耗：250mw（典型）、150uw（休眠）；
- 四种工作模式：Normal，Slow，Idle，Sleep；
- 使用片外SDRAM存储器（75MHz）性能为26.25MIPS；
- 使用片内ESRAM存储器（75MHz）性能为66.75MIPS。

SEP3203微处理器的结构中的处理器内核（ARM7TDMI）是业界标准的32位RISC处理

器内核，处理器内核频率为75MHz，支持16位的Thumb指令集、具有32×8的硬件乘法器，片上ICE支持各种基于JTAG软件调试方案。

由于ARM体系结构的突出优点是在低功耗方面以及每瓦功耗所产生MIPS(每秒钟执行百万条指令数)方面，其产品已被证明在工业界中处于领先水平。所以使用ARM核在其相关的领域中迅速开发出新一代的消费类产品，在市场上具有一定的竞争力。基于以上的考虑，我们采用了基于ARM的嵌入式系统。

2、外围电路

外围电路主要使用ARM开发套件中的ASIC开发板上的电路来实现。如输入输出缓冲我们就是用SEP3203微处理器所带的指令/数据缓存来实现，存储器则采用在核模块的SRAM DIMM插槽上接入64M的SDRAM来实现的。

4.1.2 软件方案

AES算法的整个软件设计流程如图4-2所示：

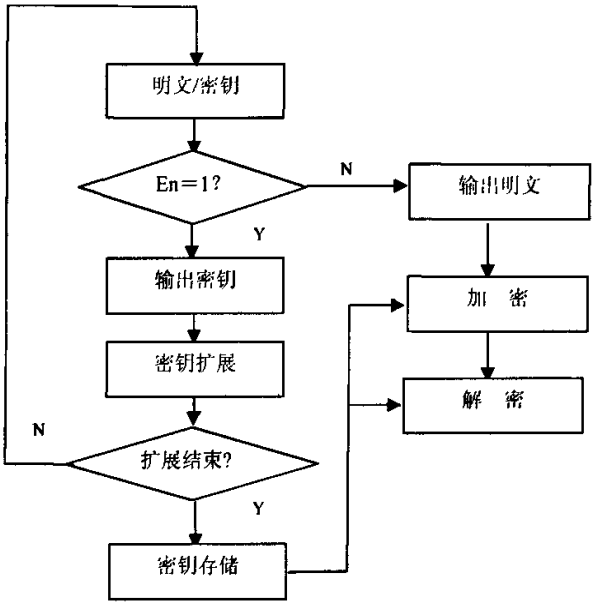


图 4-2 AES 算法软件设计流程图

在实现过程中，整个软件设计流程基本完全按照此流程进行，在某些具体部分的实现时灵活的采用了不同的方式完成。在某些情况下，ARM 的 C 编译器的优化功能可以使 C 代码的效率大大增加，但是 C 程序代码的效率与汇编代码的效率相比，汇编的代码执行效率明显占优势，所以在一些设计实现中，考虑到采用 C 语言和汇编语言混合编程实现。由于时间的关系，在对于 ARM 指令的使用和掌握比较有限，因此，在对主要框架控制流程实现时，采用 C 语言编程完成整个流程的控制，涉及到具体的模块中的各个部分采用 ARM 汇编语言实现。

4.1.3 AES 加密模块的实现方案

AES 加密模块实现方案流程框图如图 4-3 所示：

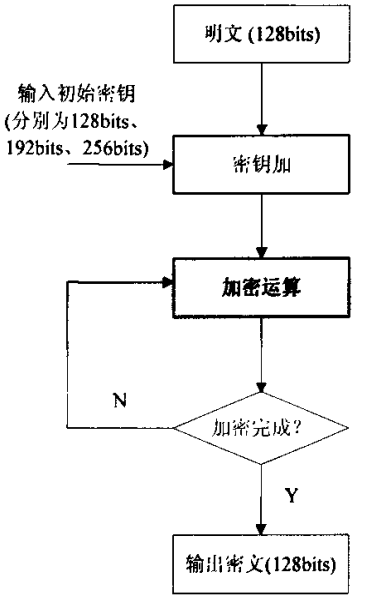


图 4-3 加密模块实现流程框图

加密模块实现的主程序如下：

```
void main(void)
eAES(int in[4] [4], int ck[12][4],char inc[33],char ckc[65])
{/*以下两行代码表示分别将用户从键盘输入的明文和密钥存储在二维数组中*/
convert(inc,in,4);
convert(ckc,ck,l);
/*由于上述两个二维数组中明文和密钥在存储时是逐行存储的，而 AES 中明文和密钥是按列分别存储在二维数组中的，所以必须进行矩阵的转置，以下代码分别实现了存储明文矩阵的转置*/
for(r=0;r<3;r++)
    for(c=3;c>r;c--)
        {i=in[r][c];in[r][c]=in[c][r];in[c][r]=i;}
……//存储密钥的二维数组的转置与以上过程相似，代码略。
/*以下三行代码表示初始密钥与明文的异或，即密钥加*/
for(r=0;r<4;r++)
    for(c=0;c<4;c++)
        in[c][r]=in[c][r]^ck[r][c];
/*以下三行代码表示当密钥长度 l 的值为 4、6 及 8 时，调用相应的加密函数*/
if(l==4)calculate4(in,ck);
if(l==6)calculate6(in,ck);
if(l==8)calculate8(in,ck);
}
```

其中入口参数是：inc 和 ckc 两个字符型数组，分别为输入的明文和初始密钥；

出口参数是： 整型二维数组 in[c][r]，用来存放输出的密文。

AES 加密模块测试主程序如下：

```
eAEStest(int in[4] [4], int ck[12][4],char inc[33],char ckc[65])
{printf("Please confirm length of cipher key(128/192/256):\n");
scanf("%d",&l);
printf("\n");
l=l/32;
printf("Please enter plain text:\n");
scanf("%s",inc);
printf("\n");
printf("Please enter cipher key:\n");
scanf("%s",ckc);
printf("\nThe result:\n");
……..//加密模块，代码略。
for(r=0;r<4;r++)
    for(c=0;c<4;c++)// 密文输出部分代码
        {   in[c][r]=in[c][r]^ck[r][c];
            printf("%4x",in[c][r]);
        }
```

以128bits密钥长度为例，如果输入明文：00112233445566778899aabbccddeeff，输入密钥：000102030405060708090a0b0c0d0e0f。所得的密文输出与FIPS 197中标准输出完全一致为：69c4e0d86a7b0430d8cdb78070b4c55a。

4.1.4 AES 解密模块的实现方案

AES解密模块是加密模块的逆过程，它的实现方案的流程框图如图4-4所示：

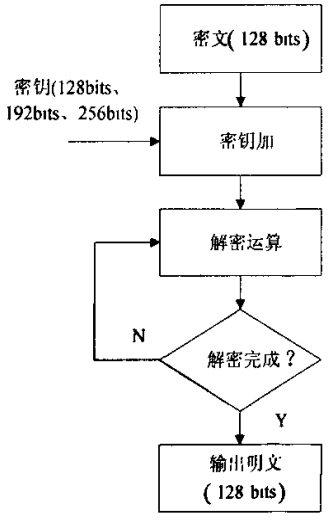


图4-4 解密模块实现流程框图

解密模块的实现的主程序如下：

```
dAES(int in[4] [4], int ck[12][4],char inc[33],char ckc[65])
 {/*以下两行代码表示分别将用户从键盘输入的密文和解密密钥存储在二维数组中*/
   convert(inc,in,4);
   convert(ckc,ck,l-8);
   /*由于上述两个二维数组中密文和解密密钥在存储时是逐行存储的，而 AES 中密文和
解密密钥是按列分别存储在二维数组中的，所以必须进行矩阵的转置，以下代码分别实现了
存储密文矩阵的转置*/
   for(r=0;r<3;r++)
       for(c=3;c>r;c--)
           {
               i=in[r][c];
               in[r][c]=in[c][r];
               in[c][r]=i;
           }
   /*以下三行代码表示密文与解密密钥的异或，即密钥加*/
   for(r=0;r<4;r++)
       for(c=0;c<4;c++)
           in[c][r]=in[c][r]^ck[r][c];
   /*以下三行代码表示当 l 的值为 12、14 及 16 时，调用相应的解密函数*/
   if(l==12) calculate4(in,ck);
   if(l==14) calculate6(in,ck);
   if(l==16) calculate8(in,ck);
   for(r=0;r<4;r++)
       for(c=0;c<4;c++)// 明文输出部分代码
           printf("%4x",in[c][r]);
   }
   其中入口参数是：inc 和 ckc 两个字符型数组，分别为输入的密文和密钥；
   出口参数是： 整型二维数组 in[c][r]，用来存放输出的明文。
   AES 解密模块测试主程序如下：
   dAEStest(int in[4] [4], int ck[12][4],char inc[33],char ckc[65])
  {printf("Please confirm length of cipher key(128/192/256):\n");
   scanf("%d",&l);
   printf("\n");
   l=l/32+8;
   printf("Please enter cipher text:\n");
   scanf("%s",inc);
   printf("\n");
   printf("Please enter cipher key:\n");
   scanf("%s",ckc);
   printf("\nThe result:\n");
   ……//解密模块，代码略。
  for(r=0;r<4;r++)
       for(c=0;c<4;c++)// 明文输出部分代码
           printf("%4x",in[c][r]);
```

}

以 128bits 密钥长度为例，如果输入密文：69c4e0d86a7b0430d8cdb78070b4c55a，
输入密钥：000102030405060708090a0b0c0d0e0f，
输出明文：00112233445566778899aabbccddeeff。

4.1.5 AES 测试方案

1.AES 正确性测试方案

本 AES 算法实现支持 128bits、192bits 和 256bits 密钥长度，所有的测试向量均来自 FIPS PUBS 197。AES 加密模块主要测试是在输入明文后，验证加密模块输出密文的正确性，然后再以密文为输入，验证解密模块的输出正确性。加密和解密模块的输入、输出均用十六进制数表示的字符数组组成，且每一字符均用两位十六进制数表示。具体加密模块和解密模块的测试主程序分别见 4.1.3 节和 4.1.4 节中所述。

如果测试产生的输出数据与 FIPS PUBS 197 给出的结果完全相同，则说明加密、解密模块完全正确。

2.性能测试方案

因为解密是加密的逆过程，在关键代码上有很多的相似性，所以本章本节中以介绍加密性能的测试方案为例，解密性能同样可采用此方案。

（1）测试环境及开发环境

主频：75MHz

开发环境：ARM ADS C 语言 32 位编程

（2）测试方法

由于加密一个 128bits 分组所需时间在本测试环境下为 10^{-6} 秒数量级，一个分组的加密时间无法测试，所以采用对同一分组进行 n 次加密(即 128×nbits 明文)的方法来实现加密时间测试。

假设测到的时间为 t，则加密一个 128bits 分组所需时间就为 t/n 秒，测试取 $n=2\times10^8$。每一种密钥下测试三次数据，得到三个速度值，然后取平均值即为该密钥加密时的速度。

（3）测试程序

```
time_t start,stop;
double timeused;
time(&start); //获取系统开始时间
{for(int j=1; j<=200000000; j++)
    加密或者解密算法模块}
time(&stop); //获取结束时间
timeused=difftime(stop,start);  //计算程序执行所花费的总时间
printf("Total timeused:%f\n",timeused);
```

在测试程序中，定义两个 time_t 类型的变量 start 和 stop，分别表示加密的起始时间和结束时间，整个加密过程用的时间为结束时间与起始时间的差。所得的加密速度为加密文件的大小（单位为 Mbit）除以加密的时间 t（单位为 s）。其中加密结果的衡量指标就是它的加密速度，单位为 Mbit/s。

4.2 基于 ARM 核嵌入式系统的 AES 算法的实现

4.2.1 基于 ARM 核嵌入式系统的 ADS 集成开发环境

在整个系统设计实现过程中，我们使用了基于 ARM 的嵌入式的 ADS 集成开发环境。

ARM ADS 全称为 ARM Developer Suite，它是由 ARM 公司提供的专门用于 ARM 相关应用开发和调试的综合性软件，是一款功能强大又易于使用的开发工具。

ARM ADS 由六部分组成：

1. 代码生成工具（Code Generation Tools）

代码生成工具由源程序编译、汇编、链接工具集组成。ADS 代码生成工具是一个用于 ARM 指令的 C 和嵌入式 C++编译器、汇编器和连接器。用于使用 C 和 C++语言编写的用户程序进行编译和连接，生成 ARM 内核的机器代码。代码生成工具是整个开发系统的核心。它具有如下特征：

- 从速度和长度上对程序进行优化；
- 可选择调试和优化的层次；
- 可针对不同处理器进行程序优化；
- 支持位置无关的代码和数据；
- 目标文件和调试表格式采用 ELF 和 DWARF2 工业标准；
- 编译器指示器能指示程序和数据的精确位置；
- 功能强大的 ARM 指令宏汇编；

ADS 的 C 和嵌入式 C++（EC++）编译器（如图 4-5 所示）提供了良好的程序与 ARM 核之间的配合，因此能有效的缩短程序的长度。此外，特殊的优化方法还提供了适合硬件内核的程序优化结果，进一步降低了程序的规模。

图 4-5　ADS 的 C 编译器界面

2. 集成开发环境（CodeWarrior IDE from Metrowerks）

CodeWarrior IDE 是 Metrowerks 公司一套比较有名的集成开发环境，有不少厂商将它作为界面工具集成在自己的产品中。CodeWarrior IDE 包含工程管理器、代码生成接口、语法

敏感编辑器、源文件和类浏览器、源代码版本控制系统接口、文本搜索引擎等，其功能与 Visual Studio 相似，但界面风格比较独特。ADS 仅在其 PC 版本中集成了该 IDE。用户可以使用 ADS 的 CodeWarrior IDE 为 ARM 和 Thumb 处理器开发用 C，C++，或 ARM 汇编语言的程序代码。

3. 调试器（Debuggers）

调试器分为两个调试器：ARM 扩展调试器 AXD（ARM eXtended Debugger ）、ARM 符号调试器 Armsd（ARM symbolic debugger）。

AXD 基于 Windows9X/NT 风格，AXD 窗口如图 4-6 所示，它具有一般意义上调试器的所有功能，包括简单和复杂断点设置、栈显示、寄存器和存储区显示、命令行接口等。AXD 运行时的调试界面如图 4-7 所示。

图 4-6　AXD 窗口

图 4-7　AXD 运行时的调试界面

Armsd 作为一个命令行工具辅助调试或者用在其他操作系统平台上。

4. 指令集模拟器（Instruction Set Simulators）

用户使用指令集模拟器无需任何硬件即可在 PC 机上完成一部分调试工作。

5. ARM 开发包（ARM Firmware Suite）

ARM 开发包由一些底层的例程和库组成，帮助用户快速开发基于 ARM 的应用和操作系统。具体包括系统启动代码、串行口驱动程序、时钟例程、中断处理程序等，Angel 调试软件也包含在其中。

6. ARM 应用库（ARM Applications Library）

ADS 的 ARM 应用库完善和增强了 SDT 中的函数库，同时还包括一些相当有用的提供了源代码的例程。使用 ARM ADS 开发应用程序与使用 ARM SDT 完全相同，同样是选择配合 Angel 驻留模块或者 JTAG 仿真器进行，目前大部分 JTAG 仿真器均支持 ARM ADS。

4.2.2 基于 ARM 核嵌入式系统的软件开发流程

ARM ADS 软件开发流程如图 4-8 所示，整个开发流程分为三个阶段。

第一阶段：根据各模块设计编写 C 源代码。编译运行，确认正确性，再利用 ADS 开发环境中的 AXD 调试器测试程序的运行周期，若不能满足要求，则进入第二阶段。

第二阶段：优化 C 源代码。选择 ADS 开发环境所提供的优化项及其高效编程技巧，优化代码，若仍不能满足效率要求，则进入第三阶段。

第三阶段：编写汇编代码。将上一阶段测试到的影响效率的关键代码段抽取出来，改用 ARM 汇编语言编写，利用汇编优化器优化。若性能仍不能达到要求，继续修改汇编代码，直到达到要求为止。这种 C 语言和汇编语言同时使用的编程方法称为混合编程。汇编优化器的作用是让开发人员在不考虑 ARM7TDMI 流水线结构和分配其内部寄存器的情况下编写 ARM 汇编语言程序，汇编优化器通过分配寄存器和循环优化，将汇编语言程序转化为利用流水线方式的高速并行汇编程序，这里我采用了 ARM ADS 集成开发环境作为开发工具。在 ADS 下，开发者可以对软件进行编辑、编译、调试、代码性能测试和进行项目管理等所有工作。

我们采用了 ARM ADS 的 CodeWarrior IDE 集成开发环境作为开发工具，可以对软件进行编辑、编译、调试、代码性能测试和进行项目管理等所有工作，除此之外，ADS 还提供了实时分析和数据可视化功能，从而大大降低了系统的开发难度，使开发者可以将精力集中在应用开发上。

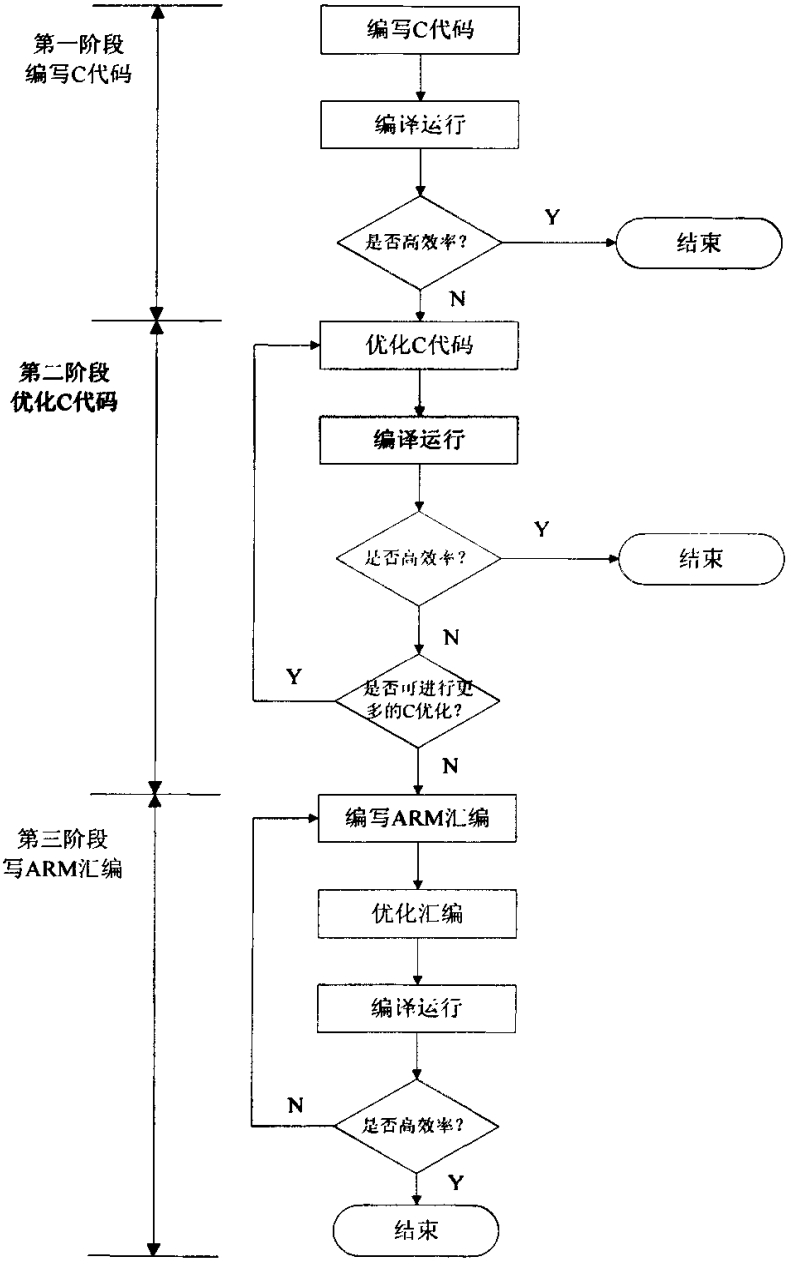


图 4-8 ADS 软件开发流程图

4.2.3 基于 ARM 核嵌入式系统的高效率程序开发技巧及接口规则

嵌入式系统对应用软件的质量要求很高，因而在嵌入式开发中尤其须注意对代码进行优化，尽可能地提高代码的效率，减少代码的大小，开发出高效率的程序，开发高效的程序涉及很多方面，包括编程风格、算法实现、针对目标的特殊优化等等，实现这些方面主要从

ARM的体系结构特点出发，以下介绍需用到的几种方法。

1. 基于 ARM 的高效 C 编程方法

(1)使用 switch-case 语句

在程序中经常会使用 switch-case 语句，每一个由机器语言实现的测试和跳转仅仅是为了决定下一步要做什么，就浪费了处理器时间。为了提高速度，可以把具体的情况按照它们发生的相对频率排序。即把最可能发生的情况放在第一，最不可能发生的情况放在最后，这样会减少平均的代码执行时间。

比如在 AES 算法的 subword( )中，进行了如下的编写：

```
void subword(p,i)
 int *p,i;
{
 int k;
 for(k=0;k<i;k++)
 {
   switch(*(p+k))
  {
     case 0x00: *(p+k)=0x63;break;   case 0x01: *(p+k)=0x7c;break;
     case0x02: *(p+k)=0x77;break;   case 0x03 :*(p+k)=0x7b;break;
     case 0x04: *(p+k)=0xf2;break;   case 0x05: *(p+k)=0x6b;break;
     case 0x06: *(p+k)=0x6f;break;   case 0x07:*(p+k)=0xc5;break;
     case0x08: *(p+k)=0x30;break;   case 0x09:*(p+k)=0x01;break;
     case 0x0a: *(p+k)=0x67;break;   case 0x0b:*(p+k)=0x2b;break;
     case 0x0c: *(p+k)=0xfe;break;   case 0x0d: *(p+k)=0xd7;break;
     case 0x0e: *(p+k)=0xab;break;   case 0x0f:*(p+k)=0x76;break;
     case0x10: *(p+k)=0xca;break; case 0x11: *(p+k)=0x82;break;
     case ..............    break;
  }
 }
}
```

(2)循环条件的设置

①中止循环条件

在 C 中,我们在编写累加计数的方法上习惯用以下的 for 循环语句:

```
for(loop=1;loop<=limit;loop++)
```

而相对用以下递减至 0 的计数方法较少:

```
for(loop=limit;loop!=0;loop--)
```

上述两种计数方法在逻辑上并不存在效率上的差异,但是映射到具体的 ARM 体系结构中就产生了很大的区别.如图 4-9 所示：

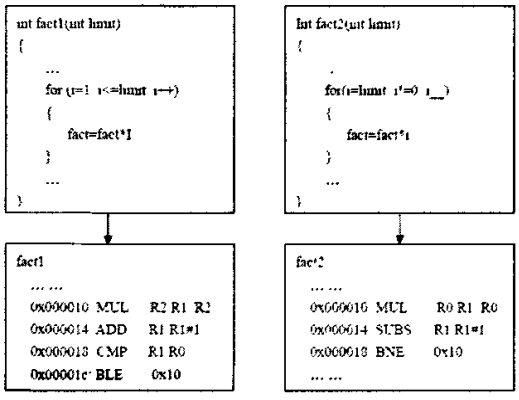


图 4-9 不同循环条件设置比

从图中我们可以看到累加法比递减法多用了一条指令，当循环参数比较大的时候，这两段代码就会在性能上产生明显的差异。因此，在 ARM 的体系结构下编写 C，最好采用递减至 0 的方法来设置循环条件。

②循环展开

小程序的展开能够得到高的运行速度性能，但增加了代码的长度,如果循环执行较少的次数，我们可以把循环完全展开，这样 Loop 的开销就完全消失。这是经典的速度优化，如图 4-10 所示：

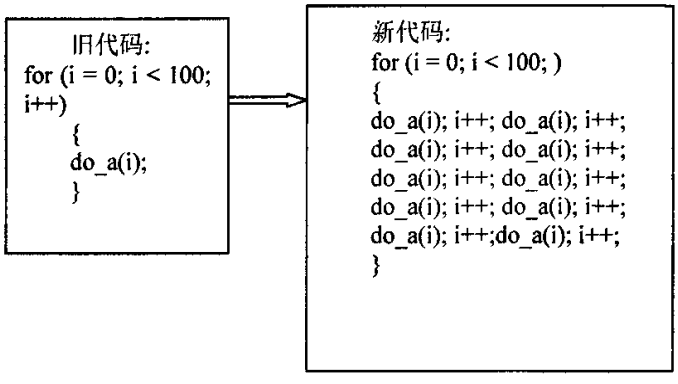


图 4-10 循环展开比较

可以看出，新代码里比较指令由 100 次降低为 10 次，循环时间节约了 90%。

目前 ARM 编译器没有自动展开的功能，展开可以在 C 的源程序里完成。因此展开也为程序的优化提供了新的方法。

(3)进行变量定义

变量定义的次序会导致最终的映像中不同的数据布局，如图 4-11 所示：

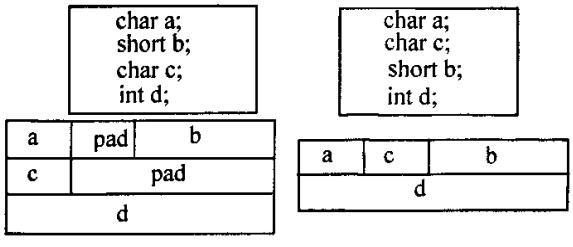


图 4-11 不同的变量定义导致不同的数据布局

由此可见在变量声明的时候，需要考虑怎样最佳的控制存储器布局。当然，便一起在一定程度上能够优化这类问题，但是最好的方法还是在编程的时候，把所有相同类型的变量放在一起定义。

(4)选用合适的变量类型

合适的变量类型可以降低代码长度和数据长度,在 C 中支持的基本类型有：char，short，int 和 long（有符号和无符号），float，double。在使用局部变量时，尽量不要定义为 char 型和 short 型，因为对此类型的数,编译器需要每次赋值后把局部变量大小还原到 8 位或 16 位，对有符号的变量要扩展符号位，对无符号的高位要添 0，故操作 8 位或 16 位的变量比操作 32 位的变量要多用的指令。例如图 4-12 表示的不同类型的局部变量编译结果：

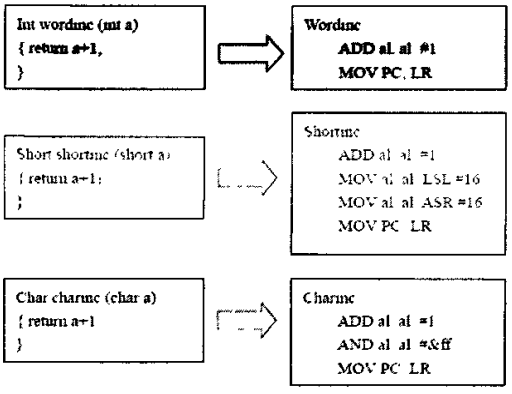


图 4-12 不同类型局部变量编译结果

因此使用合适的变量类型非常重要，它不但缩短代码长度，同时能提高代码运行效率。

(5)利用条件执行

所有的 ARM 指令都是可以带条件的，每条指令都有四位用于条件代码，指令只有在 ARM 的 flag 寄存器的标志位显示出特定的条件成立时才执行。我们可以充分利用 ARM 的条件执行这一特性。来缩短代码长度,优化流程控制。例如：图 4-13 中的流程是求 R_0 和 R_1 最大公约数。

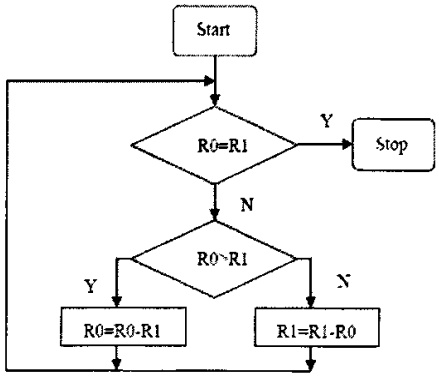


图 4-13 求最大公约数程序流程

流程中两次条件比较,达到最优结果的解决方案是:

```
Start   CMP     R0,R1
        SUBLT   R1,R1,R0
        SUBGT   R0,R0,R1
        BNE     Start
Stop
```

从这个例子我们可以得知,在充分利用条件执行下，可以从一次比较判断得到多个跳转分支。因此有条件执行降低了分支指令的数目，从而提高了代码长度和性能。如果一个条件执行指令组的数目超过 3 条，可以考虑使用跳转指令来进行条件分支，有助于条件判决速度，但是增加了代码长度，属于代码长度与执行性能之间的矛盾平衡问题。如果性能是首要问题，那么在 C 的条件描述语句中（如 if 描述语句），应尽可能简化条件描述。

(6) 查表方式

好的ARM嵌入式软件基本上不会在自己的主循环里进行运算工作，一般是先计算好了，再到循环里查表。如果表很大，就用 init 函数，在循环外临时生成表格。这样加快的程序的运行效率。

2. 基于 ARM 的 C 与汇编语言混合编程的方法

C语言与汇编语言混合编程在追求效率的程序中是经常用到的方法之一，注意在两者互相调用过程中，要遵循ATPCS[40]的定义，这里介绍在C中加入汇编程序的两种方法：

(1)内联汇编

内联汇编是指在C函数定义中插入汇编语句的方法，比如下面的例子：

```
Void enable_IRQ(void)
 {   int temp;
      _asm                                  //内联汇编定义
    { MRS   temp,CPSR
      BIC   temp,temp,#0x08
      MSR   CPSR_c,temp
     }
    }
```

内联汇编与真实的汇编有所区别，它不支持Thumb，内联汇编不能直接访问物理寄存器（CPSR除外）。

(2)嵌入式汇编

嵌入式汇编与内联汇编不同，它同时支持ARM和Thumb，具有真实汇编的所有特性，但不能直接应用C变量定义，数据交换必须通过ATPCS进行，在形式上表现为独立的函数体，就像调用一个名为asm，带有一个字符串型常量为参数的函数，调用形式为asm("汇编语句")，编译时，嵌入的汇编语句指令直接复制到编译器输出的汇编代码中，由编译器进行语法检查。比如下面的例子：

```
_asm int add(int i,int j)     //定义嵌入式汇编
      {   ADD R0, R0, R1
          MOV PC,LR
       }
        void main( )
```

{.....}

3. 接口规则

为了使单独编译的C语言和汇编语言之间能够相互调用，所以为子程序间的调用规定了一定的接口规则。

(1)寄存器使用规则

①子程序间通过寄存器R_0～R_3来传递参数。

②子程序中使用R_4～R_{11}来保存局部变量。

③使用寄存器R_{12}作子程序间的scartch寄存器，记作ip。

④使用寄存器R_{13}作数据栈指针，记作sp。R_{13}不可作其他用途，sp在进入子程序的值和退出子程序的值必须相等。

⑤使用寄存器R_{14}作连接寄存器，记作lr，用作保护子程序的返回地址。

⑥使用R_{15}作程序计数器，记作PC，不能作其他用途。

(2)参数传递规则

参数个数固定的子程序参数传递规则：如果参数包含浮点运算硬件部件，各个浮点运算按顺序处理，首先为每个浮点参数分配FP寄存器，第一个整数参数，通过R_0～R_3进行传递，其他的参数通过数据栈传递。

参数个数可变的子程序参数传递规则：如果参数个数不超过 4 个，则通过 R_0～R_3 进行传递；如果参数个数超过 4 个，则将剩余的字数据传送到数据栈中，入栈的顺序与参数相反，即最后一个字数据先入栈。

子程序结果返回规则：结果为 32 位整数时，通过寄存器返回；结果为 64 位整数时，通过寄存器 R_0 和 R_1 返回；结果为浮点数时，通过浮点运算寄存器 f_0、d_0 来返回。

(3)数据栈使用规则

根据数据栈向内存地址的增加或减小方向分有：Ascending 栈和 Descending 栈，综合这两者特点，可有四种数据栈类型：FD（Full Descending）、ED（Empty Descending）、FA（Full Ascending）、EA（Empty Ascending）。但 ATPCS 规定数据栈用 FD 类型，并且数据栈操作是 8 字节对齐的。

4.2.4 基于 ARM 的 AES 算法的实现验证

根据以上章节的 AES 算法的总体设计及优化设计，在 ARM ADS 开发环境下，对算法的各个模块设计编写 C 源代码，如图 4-14 所示：

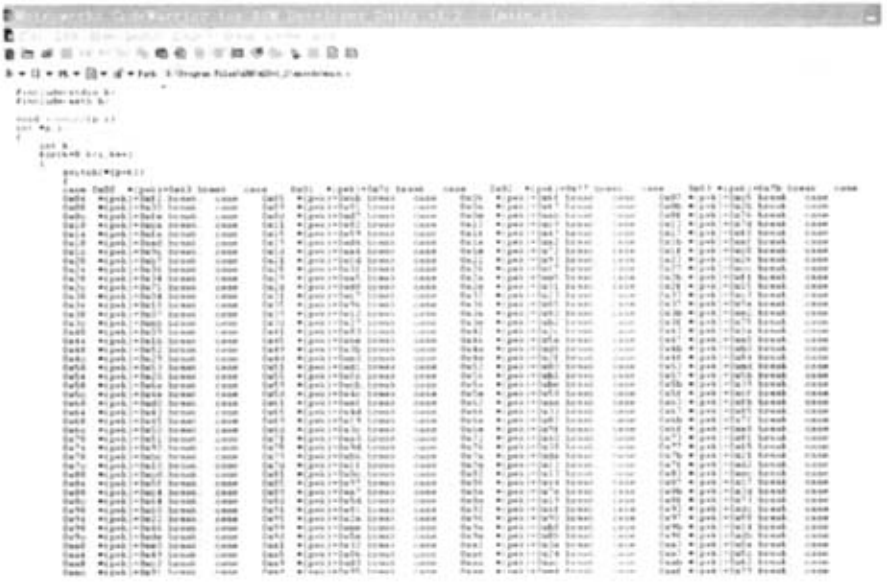


图 4-14 AES 程序 ADS 界面

通过 ARM C 编译器编译运行，在 AXD 调试器中调试后输出的结果与 FIPS PUBS 197 上给出的结果一致，因此验证了算法设计正确，输入密钥分别为 128 位、192 位和 256 位，结果分别如图 4-15、4-16、4-17 所示。

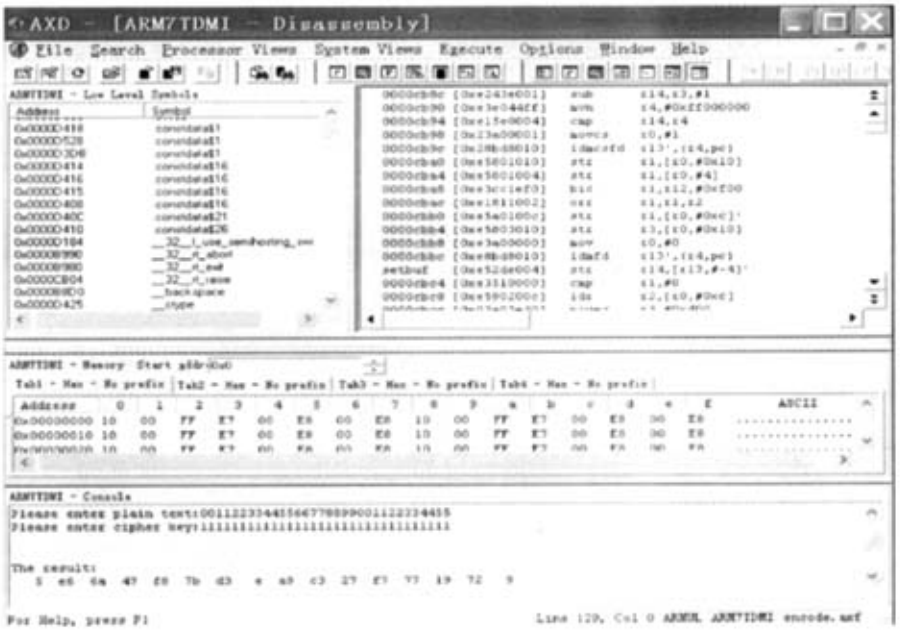


图 4-15 Key＝128bits 的验证实现

图 4-16 Key＝192bits 的验证实现

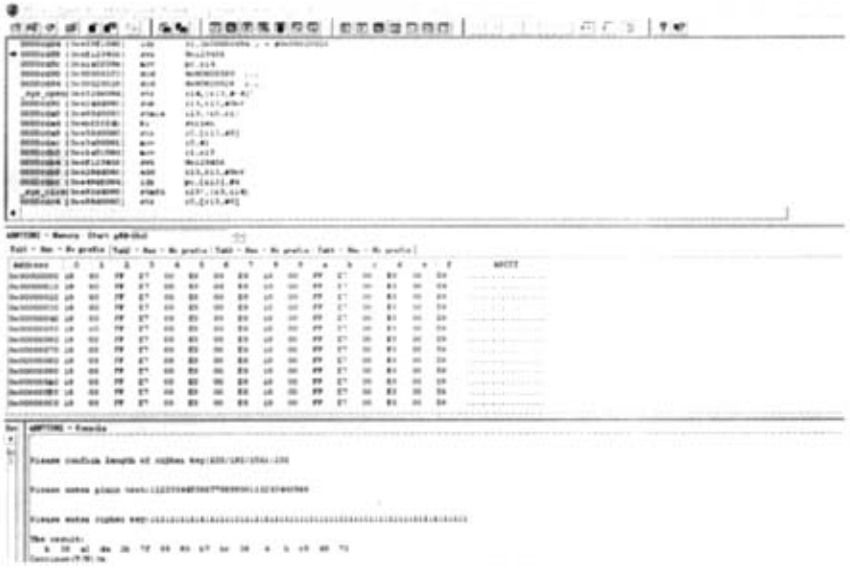


图 4-17 Key＝256bits 的验证实现

明文密钥以及结果在 ARM7TDMI Console 中能正确显示出来。在内存观察区中我们可以看出在地址为 0x000000000 的连续 4 个单元内，分别存放了输入/输出模式控制寄存器的值，可以读出该控制寄存器的内容为 0xE7FF0010。

4.3 基于 ARM 的 AES 算法代码优化和性能测试

经过上 4.2 节的编译运行，代码基本实现了设计要求，但速度还不能达到预期的要求，故需对代码进行优化。在结构优化设计中，使得加密和解密具有完全相同的结构，所以加密和解密具有完全相同的速度，这里我们仅对 128 位密钥加密模块进行代码优化和性能测试。

1. 编译器优化

在 C 代码优化阶段，ARM C 编译器提供了大量的编译选项供用户选择，这些选项中有一部分直接影响或控制编译器的优化过程，从而会影响编译出的代码性能，所以可通过编译器的部分选项对代码实现初步优化。在测试中选取编译器的优化选项分别为-O_0、-O_1、-O_2以及-O_{time}。

-O_0—清除多余的代码，关闭大多数的优化，得到最好的调试信息，是最少优化。

-O_1—消除了局部和全局共有的子表达式，移掉从未调用的函数，使代码的尺寸也得到优化。

-O_2—进行完全的代码优化，有限的调试信息，最好的代码密度。

-O_{time}—使代码运行速度进行优化。

代码优化后，我们对加密模块的实现做了性能测试，编译器自动代码优化后与原代码优化前测试的加密性能数据相比，结果如表 4.1 所示：

表4.1 基于ARM的C源代码编译器优化后的测试性能表

文件名	原AES_main.c	编译器优化后的AES_main.c
时钟周期数	1210	625
加密速度	32.65Mbit/s	52.34Mbit/s

2. 代码优化

经过编译器优化后，查看了AXD中的整个程序的运行周期，周期数比优化前1210减少到625，但是单凭编译器优化还不够达到要求，还得进行代码优化。首先我们对通过断点设置，统计一段一段函数代码的执行所需要花费的周期数，找到AES_main.c中花费时钟周期较多的函数，对其进行优化。

（1）经过统计分析发现花费周期较多的大部分是多次迭代的代码，比如ShiftRows函数。

```
void ShiftRows(byte state[4][4])
{int i，j；byte temp[4][4]；
  for(i=0；i<4；i++) .
  { for(j=0；j<4；j++)
   {temp[i][j]=state[i][j]；}
   }
  for(j=1；j<4；j++)
  { for(i=0；i<4；i++)
   {state[i][j]=temp[(j+i)%4][j]；}
  }
}
```

在代码优化设计中重点是优化循环部分，优化后的循环代码为：

```
for(i=0; i<4; i++) .
 for(j=0; j<4; j+=2)
 {temp[i][j]=state[i][j];
  temp[i][j+1]=state[i][j+1]; }
```

由于temp[i][j]和temp[i][j+1]、state[i][j]和state[i][j+1]在存储器中是相邻的，因此可通过使用load指令同时读temp[i][j]和temp[i][j+1]、state[i][j]和state[i][j+1]并送入32位寄存器，从而达到进一步优化代码的目的，优化后的测试性能如表4.2所示：

表 4.2 基于 ARM 的代码优化后的测试性能表

文件名	优化后的 AES_main.c
时钟周期数	**472**
加密速度	68.26Mbit/s

（2）通过ARM汇编文件进行优化。

这种针对汇编优化面向的是硬件，首先进行流水线上优化，有些load指令需要多个周期完成，可通过调整指令的顺序来改善性能，同时让程序尽量使用寄存器，弄清楚数据占用寄存器的时间关系，采用寄存器的分时复用。另外，将长度较小的变量合并到一个32位寄存器中保存，以节省寄存器的数量。在函数周期统计过程中，同时我们发现轮函数的时间开销最大，为95个时钟周期，从编译产生的asm汇编文件中发现：在轮函数算法计算出一个字要4次load操作，那一轮就需要16次操作，而我们只需要4次。解决的优化方法是只进行四次load操作，每计算好4个轮输出字写入内存之前先保护好这4个寄存器。具体是对轮函数进行ARM汇编宏定义，将每一轮输出的4个轮输出字直接存在4个寄存器中，这样9轮就节省了时间开销，等第九轮完成后再将4个轮输出字写入相应的内存。

ARM汇编宏定义的轮函数结构如下：

```
MACRO     ; 宏定义开始
$label  Roundfun   $p1,$p2,$p3,$p4 ; 定义宏的 4 个参数
;  code  九轮循环代码段
……
用循环语句 Loop_i  i=0 to 9
 BL   $p1        ; $p1 为一子程序的名称
 BL   $p2        ; $p2 为一子程序的名称
 BL   $p3        ; $p3为一子程序的名称
 BL   $p4        ; $p4 为一子程序的名称
……..
;  code
MEND      ; 宏定义结束
```

在程序中调用该宏

```
AES  Roundfun  SubByte , ShiftRow, MixColumn, AddRoundkey
```

该宏被展开的结果就是

```
;  code
…..
```

```
BL    SubByte          ; 参数 p1 的实际值为 SubByte
BL    ShiftRow         ; 参数 p2 的实际值为 ShiftRow
BL    MixColumn        ; 参数p3的实际值为MixColumn
BL    AddRoundkey      ; 参数 p4 的实际值为 AddRoundkey
.....
;  code
```

ARM汇编优化后性能测试如表4.3所示：

表 4.3 基于 ARM 的代码优化后的测试性能表

文件名	优化后的 AES_macro.c
时钟周期数	**282**
加密速度	**95.02Mbit/s**

经过上述性能优化后，从测试结果看，整个加密的速度得到了较大的提高，在嵌入式微处理器 ARM 上调用 ARM 汇编子程序实现的 AES 算法加密速度比调用 C 子程序实现的 AES 算法加密速度要快 62.37Mbit/s。因此采用上述优化方法提高了算法的运行效率。

4.4 小结

本章首先对 ARM ADS 系统集成开发环境作了介绍，总结了基于 ARM 的软件开发中需用到的高效编程技巧及接口规则，根据各模块设计用 ARM C 语言进行了验证实现，并结合编译器优化及利用代码优化对算法进行了性能上对比测试，同时提出了优化原则。

第五章 总结与展望

5.1 总结

通过对课题的研究，本论文总结如下：

第一章介绍了 AES 算法的产生背景及其研究现状，同时简要介绍了嵌入式系统及其应用，结合二者，提出了本课题的研究意义。

第二章研究了 AES 算法的两个设计原则（即安全性原则和实现性原则）、主要的数学基础知识（即字节上的运算操作及元素在有限域中的加或乘运算）、AES 算法的整体结构（即 SP 网络结构）、AES 的算法描述（即加密和解密的整个详细的算法流程介绍及伪代码实现）以及 AES 算法的存在的优点和局限性。

第三章研究了基于 ARM 核的 AES 优化设计方案，提出了 AES 算法的整体设计，各模块的具体设计及优化，在优化设计中主要从两方面对 AES 算法进行了优化：

一是在算法本身进行了优化，这主要是在加密模块中对字节替换运算、列混合运算和解密模块中的逆列混合运算作了优化，由原来的复杂的运算分别转换为简单的循环移位、乘和异或运算。大大提高了算法在系统中的运算速度。

二是针对 ARM 核硬件角度对算法进行了结构上的优化，输入输出接口上，利用 ARM 核有 32 位的数据接口，为了加快数据传输速度，对 128 位的数据用四个 32 位的寄存器进行了并行输入并行输出的设计；在密钥扩展模块为了节省 ARM 核处理器的内存，所以密钥扩展采用内部扩展，使得密钥扩展与加/解密运算并行执行，大大提高了算法的执行效率；加密/解密轮函数上进行了四个操作表查询工作合并成一个操作表查询工作的优化，节省了存储空间且提高了算法的效率。为了使加密代码在解密代码中可重用，节省硬件资源，在解密过程中采用了与加密相一致的过程顺序。

第四章研究了基于 ARM 核的 AES 算法实现，提出了硬件实现方案、软件实现方案、加密模块和解密模块的实现方案以及测试方案，介绍了基于 ARM 核的嵌入式的 ADS 的软件集成开发环境，总结了基于 ARM 下的高效编程技巧及接口规则，在集成开发环境下对算法进行了实现，分别得出了输入初始密钥为 128bits、192bits 和 256bits 下的加密与解密的结果，并得到了正确验证。在测试的过程中我对算法代码进行了优化及性能上的测试对比，在优化过程中总结了两方面的优化原则：

一是编译器优化，这要根据具体的需求进行优化，并不是所有的优化选项都必须得用，在 AES 算法中，为了达到速度上的优化，可选择“$-O_{time}$”，为了达到代码上的优化，可选择“$-O_0$”或“$-O_1$”或“$-O_2$”，其中“O”后面的“0”、“1”、“2”分别代表优化的级别由低到高。

二是代码优化，在代码优化中重点是优化多次迭代的代码，我们可通过并行流水线法，使用 load 指令把将使用相邻存储器的内容并行送入 32 位寄存器。

另外，在代码优化中我们还可以通过编写汇编程序进行优化，这是面向硬件的，首先进行流水线上优化，有些 load 指令需要多个周期完成，可通过调整指令的顺序来改善性能，同时让程序尽量使用寄存器，弄清楚数据占用寄存器的时间关系，采用寄存器的分时复用。另外，将长度较小的变量合并到一个 32 位寄存器中保存，以节省寄存器的数量。

5.2 展望

此外，本论文中还有一些工作需要完善和深入研究，展望如下：

1. AES 算法的设计使用了很强的代数结构，随着一些代数方法的进展，可能出现新的攻击方法。因此要积极探索新的攻击方法值得研究。

2. 本文虽然在基于 ARM 对 AES 实现作了一些工作，但还是相当的局限，今后还需进一步的进行探索和改进，用 ARM 汇编来完全编写出整个的优化过程，做成基于 ARM 的嵌入式高级加密芯片，投入到实际应用中，即和单片机系统结合起来，形成一个具有高可靠性和安全性的智能 IC 卡。

3. 借鉴 AES 算法在设计原理方面的成功经验开发出本土化的密码算法，尽快制定出我们自己的加密标准。

总之，AES 算法的研究是分组密码乃至整个密码学研究中的一个热点，而基于嵌入式的 AES 算法优化实现上的研究也是应用上的一个重点，需要广大研究人员不断的努力和探索。

致 谢

首先我要感谢我的校内导师李智群老师和校外导师包志华老师，感谢他们在学业和课题上对我进行谆谆教诲。他们严谨的学术作风，踏踏实实的为人态度给我留下了深刻的印象，是我学习、工作的榜样。

其次我要感谢景为平老师和陈海进老师，感谢他们给我提供了良好的研究环境以及在我课题上对我的认真指导和帮助。

再次我要感谢东南大学的嵌入式系统师资培训班的任课教师，感谢你们在课题的ARMADS实现上给我提出的方法和建议。

最后我要衷心感谢我的家人，在我最艰难的时期，他们给了我完成学业的精神动力和物质保障。

参考文献

[1] 谷大武,徐胜波译.Joan Daemen and Vincent.高级加密标准(AES)算法—Rijndael 的设计[M].北京：清华大学出版社.2003

[2] 赵勇. 高级加密标准 AES 的实现研究[D].成都：电子科技大学 2004

[3] 华漫.AES 算法分析及其硬件实现[D].成都：西南交通大学 2004

[4] High Speed AES Encryption Cores,available at http:// www.d-crypt.com

[5] William Stallings 著,杨明,青光辉,齐望东等译.密码编码学与网络安全:原理与实践(第二版)[M].北京:电子工业出版社,2001

[6] 冯登国,吴文玲.分组密码的设计与分析[M].北京:清华大学出版社,2000

[7] 王先平,张爱菊,熊平.新一带数据加密标准--AES [J].计算机工程,2003,29（3）：69-70.

[8] 施奈尔著,吴世忠等译.应用密码学:协议、算法与 C 源程序[M].北京:机械工业出版社,2000.1

[9] 韩沽,周勇,王伟.高级数据加密标准 Rijndael 算法研究及软件实现[J].微型机与应用,2003(3):51-53

[10] 何明星,林昊.AES 算法原理及其实现[[J].计算机应用研究,2002,12：61-63

[11] NIST. Federal Information Processing Standards Publication 197[EB/OL].http://csrc.nist.gov/publications/fips/fips 197/fips-197.pdf

[12] 黄文平.AES 的安全性分析[[J].微型机与应用,2004(2):4-6.

[13] J.Daemen and V.Rijmen. "AES Proposal: Rijndael"[EB/OL]. http://www.cryptosoft.de/docs/Rijndael.pdf

[14] 肖国镇,白恩健,刘晓娟.AES 密码分析的若干新进展[J].电子学报,2003,31(10):1549-1554.

[15] 蔡宇东,沈海斌,严晓浪.AES 算法的高速实现[J].微电子学与计算机,2004(21):83-85

[16] J.Daemen and V.Rijmen, The Proposal:Rijndael, AES Algorithm Submission,September3,1999,http://www.nist.gov/cryptotoolkit.

[17] 韦宝典,刘东苏,王新梅.Rijndael 优化实现研究〔J].计算机工程与应用,2002(20):4-6

[18] 张振权,罗新民,齐春.用 AVR 汇编语言实现 AES 及其优化[J],技术纵横,2005(8):27-29

[19] 潘志铂,郑宝玉等.一种扩展的 Rijndael 算法及其 DSP 快速实现的研究.通信学报,2003,24(9):19-26.

[20] 邵常勇,陈涤,董国锋. 基于 ARM 的嵌入式系统设计方法研究[J].信息技术与信息化,2006 年第 2 期：96－98

[21] 马 林.一种基于ARM核处理器的嵌入式网络温度传感器设计[J].国外电子测量技术,2006,l25（1）：11－13

[22] 高超然,徐成,李仁发.ADS环境下ARM 系统两种初始化方式分析[J].计算机工程,2006,32(2):P73－75

[23] 江俊辉.基于 ARM 的嵌入式系统硬件设计[J].微计算机信息,2005(21):120-122

[24] 白广文,段卫然,俞建新.在 ARM 微处理器上实现 Rijndael 加密算法[J].单片机与嵌入式系统应用,2005(7):20-22

[25] 杨浴光.基于 ARM 的 G723.1 解码的移植和优化[J].电声技术,2005(5):29-30

[26] 杜春雷.ARM 体系结构与编程[M],清华大学出版社,2003

[27] Edward Roback, Morris Dworkin. Conference Report- First Advanced Encryption Standard (AES) Candidate Conference[J]. Journal of Research of the National Institute of Standards andTechnology, Volume 104, Number 1, January-February 1999.

[28] Lauren May, Matthew Henricksen, William Millan. Strengthening the key schedule of the AES[J], in Proceedings of Information Security and Privacy, 7th Australian Conference, ACISP2002[C], Melbourne, Australia, July 3-5, 2002, 226-240.

[29] Stefan Lucks.Atacking seven rounds of Rijndael under 192-bit and 256-bit

keys[EB/OL] ,Conf. New York,2000, Gaithersburg:,NIST,2000:215 -229.2001-10-20.

[30] I.Harvey.The Effects of Multiple Algorithms in the Advanced Encryption Standard[J], in The Third AES Candidate ConferencePrinted by the National Institute of Standards andTechnology,Gaithersburg,MD,April13-14,2000,269-276

[31] 韦宝典,刘东苏等.Rijndael 优化实现研究.计算机工程与应用[J],2002,38(20):4-6

[32] 郎荣玲,夏煜,戴冠中.高级加密标准（AES）算法的研究[J].小型微型计算机系统,2003,24（5）：905-908

[33] S. Murphy and M. Robshaw. New Observations on Rijndael[J], AES Forum comment, August 7, 2000, available at http://www.cs.rhbnc.ac.uk/-seanl/.

[34] 李金花,周大水,李大兴.AES算法在DSP中的实现及优化[J].计算机工程,2004(30):101-102

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

[36] F.L.Bauer著,吴世忠等译.密码编码和密码分析原理与方法[M].北京:机械工业出版社,2001

[37] T. Jakobsen, L.R. Knudsen, The intepolation attack of block cipher[J],Fast Software Encryption'97, INCS 1267, E. Biham, Ed., Springer- Verlag, 1997:28-40

[38] Third AES Candidate Conference. http://csrc.nist. gov/encryption/aes/round2

[39] 黄智颖,冯新喜.高级加密标准 AES 及其实现技巧[J].计算机工程与应用,2002(9):112-115.

[40] 时龙兴,凌明,王学香编著.嵌入式系统—基于SEP3203微处理器的应用开发[M],电子工业出版社,2006