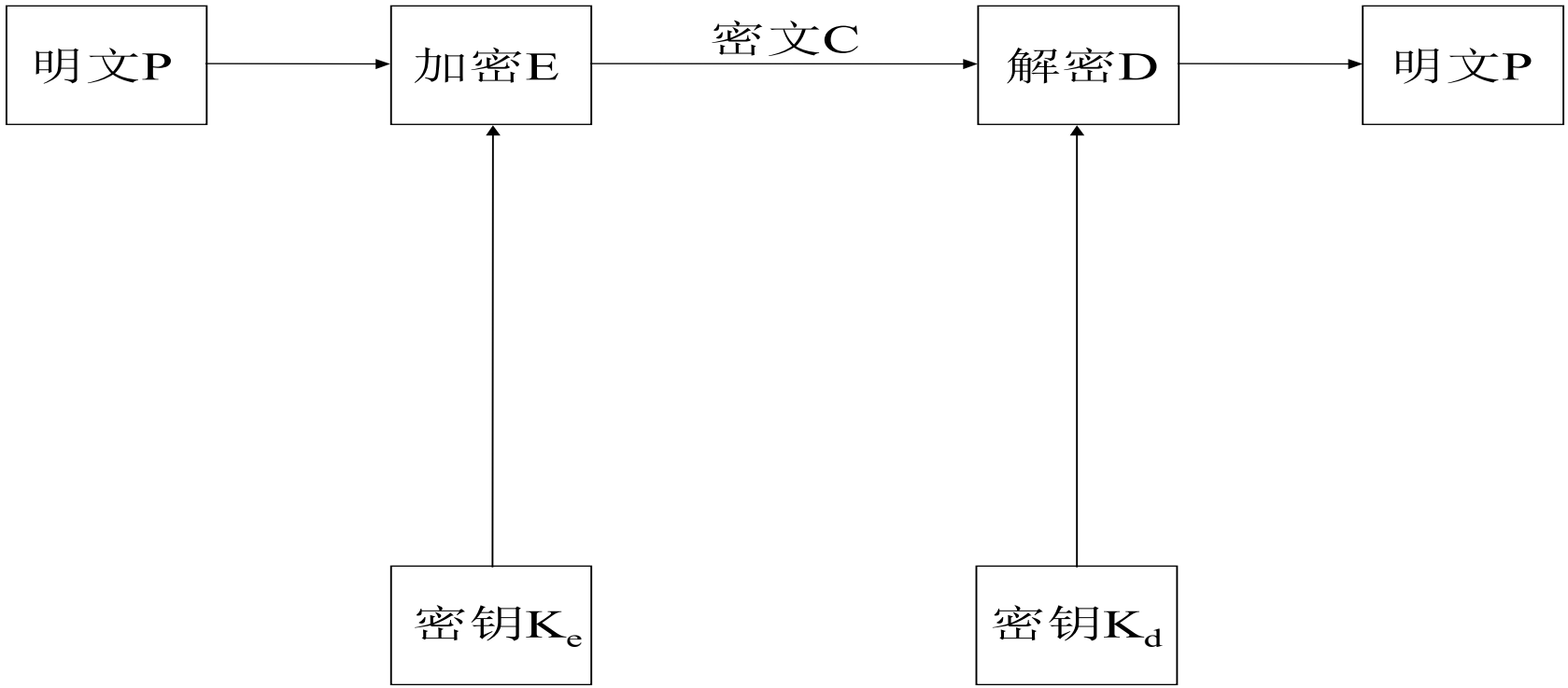
VLSI设计实例

—AES密码处理器设计

1. 加解密原理简介

1.1 数据加密原理

- 数据加密的基本过程如下图所示：



- 加密的基本思想是伪装机密信息，待伪装的信息称为明文(plain text)，伪装以后的信息称为密文(cipher text)。对信息进行伪装的操作称为加密，为密文解除伪装即将密文还原成明文的操作称为解密。加/解密时所使用的信息变换规则称为密码算法。通常把一个密码系统采用的基本工作方式称为密码体制。构成一个密码体制的两个基本要素是密码算法和密钥。
- 密码算法是将明文(密文)转换为密文(明文)的一系列公式、法则或程序，从数学的角度来看，密码算法是一个明文(密文)到密文(明文)的变换(或者称为映射或函数)，而且这种变换是可逆的，否则，将无法将明文(密文)还原为密文(明文)。密钥可以看作是密码算法中的可变参数，改变了密钥也就改变了明文和密文之间的函数关系。

- 一个密码系统可以用数学符号描述如下：

$$S=\{P，C，K，E，D\}$$

其中，P是明文空间，表示全体可能出现的明文集合。C是密文空间，表示全体可能出现的密文的集合。K是密钥或密钥空间。E是加密算法，D是解密算法。

- 当给定的密钥$k\in K$时，加/解密算法分别记作E_k、D_k，于是密码体制表示为：

$$S_k=\{P，C，k，E_k，D_k\}$$

各符号之间有如下关系：

$$C= E_k(P)$$

$$P= D_k(C)= D_k(E_k(P))$$

分别表示用加密算法E_k对明文P加密得到密文C，用解密算法D_k对密文C解密得到明文P。显然，加密变换E_k和解密变换D_k是互逆的，即：

$$D_k= E_k^{-1}且E_k= D_k^{-1}$$

- 实际上，加密变换E_k和解密变换D_k都是由一系列子变换复合而成，即：

$$E_k=e_1\cdot e_2\cdots e_n,$$

$$D_k=d_1\cdot d_2\cdots d_n$$

其中，每一个子变换e_i或d_i构成加/解密算法中的一个步骤，常用的子变换有移位、置换、逐位异或运算、模乘加运算、S盒变换等。

1.2 密码体制的分类

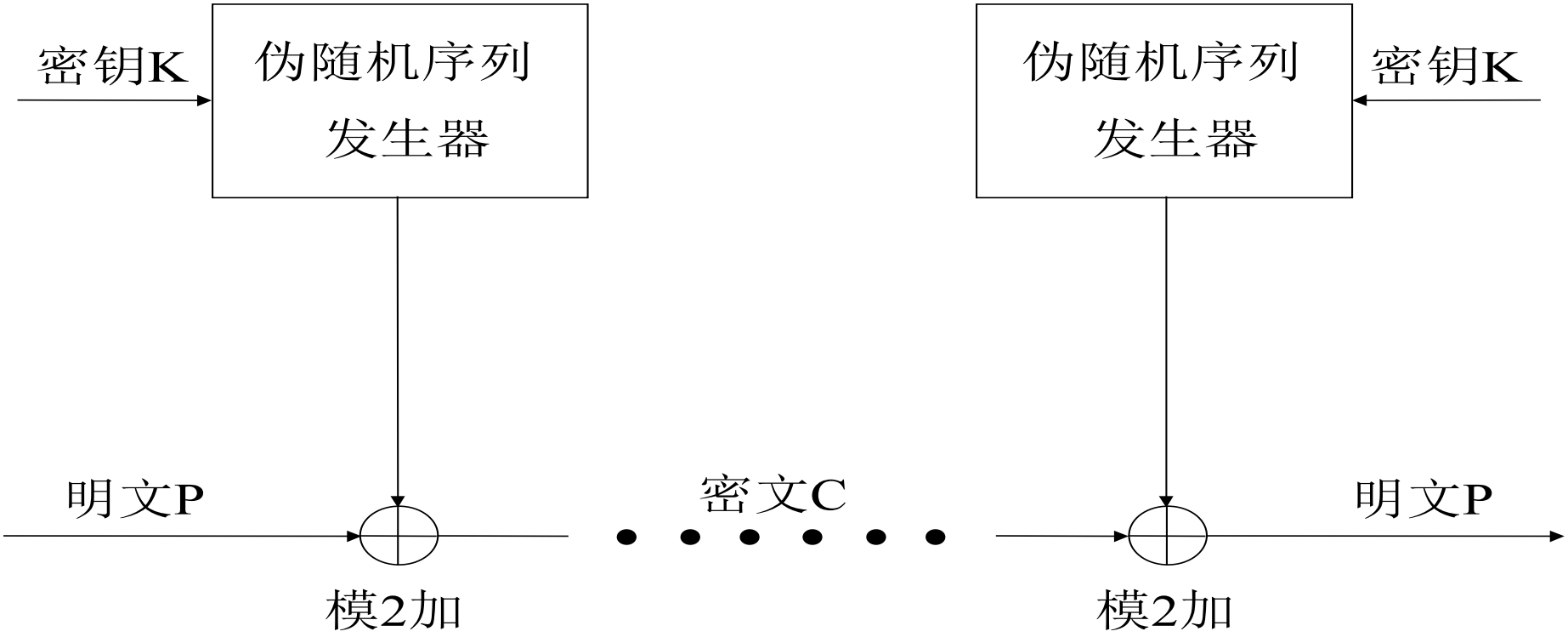
（1）根据密码算法所使用的加密密钥和解密密钥是否相同、能否由加密过程推导出解密过程(或者由解密过程推导出加密过程)，可将密码体制分为**对称密码体制**(也叫作**单钥密码体制**、**秘密密钥密码体制**、**对称密钥密码体制**)和**非对称密码体制**(也叫作**双钥密码体制**、**公开密钥密码体制**、**非对称密钥密码体制**)。

- 如果一个密码系统的加密密钥和解密密钥相同，或者虽然不相同但是由其中的任意一个可以很容易地推导出另一个，则称该密码系统采用的是**对称密码体制**。
- 对称密码体制的优点是**加/解密速度快**(例如DEC公司开发的一个样品DES芯片的加/解密速度达到1Gb/s)且具有很高的保密强度，可以达到经受国家级破译力量的分析和攻击。但它的密钥必须经过安全可靠的途径传递，**密钥管理**成为影响系统安全的关键性因素，使它难以满足系统的开放性要求。

- 如果一个密码系统的加密密钥和解密密钥不同，并且由加密密钥推导出解密密钥(或者由解密密钥推导出加密密钥)是计算上不可行的，则称该密码系统采用的是非对称密码体制。采用非对称密码体制的每个用户都有一对选定的密钥，其中一个是可以公开的，一个由用户自己秘密保存。
- 非对称密码体制的出现是现代密码学研究的一项重大突破，它的主要优点是可以适应开放性的使用环境，密钥管理问题相对简单，可以方便、安全地实现数字签名和验证。但它的加/解密速度较慢(例如目前最快的RSA芯片的加/解密速度为2Mb/s)，而且其保密强度目前还远远达不到对称密码体制的水平。

（2）根据密码算法对明文信息的加密方式，可分为序列密码体制和分组密码体制。

- 如果经过加密(解密)所得到的密文(明文)仅与给定的密码算法和密钥有关，与被处理的明文(密文)数据段在整个明文(密文)中所处的位置无关，就叫作分组密码体制；如果经过加密(解密)所得到的密文(明文)不仅与最初给定的密码算法和密钥有关，而且也与被处理的明文(密文)数据段在整个明文(密文)中所处的位置有关，就叫作序列密码体制。
- 通常的情况下，序列密码体制总是以明文的比特为加密的单位。加密时，将一段类似于噪声的伪随机序列与明文序列模2加后作为密文序列，这样即使对于一段全“0”或全“1”的明文序列，经过序列密码加密后也会变成类似于随机噪声的乱数流。在接受端，用相同的伪随机序列与密文序列模2加就可以恢明文序列。序列密码的关键技术是伪随机序列产生器(即伪随机序列产生函数)的设计。序列密码的基本形式如下图所示：



- 通常分组密码体制总是以大于64比特的数据块为加密单位，给定相同的明文数据块加密后便得到相同的密文数据块。
- 分组密码和序列密码各有其优缺点，应根据不同的应用要求采用不同的体制。

（3）按照在加密过程中是否注入了客观随机因素，可分为确定型密码体制和概率密码体制。

- 如果一个加密过程可以描述为：当明文、密钥被确定后，密文的形式也就唯一地确定，就称之为确定型密码体制。
- 如果一个加密过程可以描述为：当明文、密钥被确定后，密文的形式仍然是不确定的，或者说对于给定的明文和密钥，存在一个很大的密文集合与之对应，而最后产生出来的密文则是通过客观随机因素在这个密文集合中随机地选择出来的，就称之为概率密码体制。采用概率密码体制的目的是为了有效地提高保密强度，因为概率密码体制中存在着大量的不确定因素，使破译的代价成指数地增加。

（4）按照是否能进行可逆的加密变换，可分为单向函数密码体制和双向变换密码体制。

- 单向函数密码体制是一类特殊的密码体制，其性质是可以容易地把明文转换成密文，但把密文转换成明文却是困难的(有时甚至是不可能的)。单向函数只适用于某种特殊的、不需要解密的场合。
- 如果某一密码体制能够进行可逆的双向加/解密变换，则称该密码体制为双向变换密码体制。

1.3 典型密码算法-DES算法简介

- DES是美国数据加密标准算法，英文全称是Data Encryption Standard。
- DES是迄今为止世界上最为广泛使用和流行的一种分组密码算法，它是由美国IBM公司研制的，是早期的称做Lucifer密码的一种发展和修改。
- DES在1975年3月17日首次被公布在联邦记录中，在做了大量的公开讨论后于1977年1月15日正式批准并为美国联邦处理标准即FIPS_46。
- DES是一种为二进制编码设计的，可以对计算机数据进行密码保护的数学运算。
- DES通过密钥对64 位的二进制信息进行加密，把明文的64位信息加密成密文的64位信息。
- 由于DES的加密算法是公开的，所以加密强度取决于密钥的保密程度。
- 加密后的信息可用加密时所用的同一密钥进行逆变换得到对应的明文。
- DES的设计中，将64位密钥中的56位用于加密过程，其余8位用于奇偶校验位。确切的说，密钥分成八个8位的字节，在每一个字节中的7位用于加密算法，第八位用于奇偶校验。

- 在DES算法中，64位的明文PT首先经过一个64*64的初始置换IP，并将置换结果分成左右两个32位的数据，分别记为L_0、R_0，即：

$$\{L_0, R_0\}=IP(PT)$$

- 然后对$\{L_0, R_0\}$进行一系列的加密变换得到新的64位数据$\{L_1, R_1\}$：

$$L_1=R_0$$
$$R_1=XOR(L_0, P(S(XOR(E(R_0), K_1))))$$

其中：E是32*48扩展置换，K_1是第一轮子密钥，XOR是48位逐位异或运算，S是由8个6*4S盒构成的代替变换，P是32*32的P盒置换。

- 再对$\{L_1, R_1\}$进行相同的变换得到$\{L_2, R_2\}$，…，依次类推，共进行16轮相同的加密变换得到$\{L_{16}, R_{16}\}$。
- 最后对$\{L_{16}, R_{16}\}$进行初始置换IP的逆变换IP^{-1}就得到了64位的密文：

$$CT= IP^{-1}\{L_{16}, R_{16}\}$$

- 综上所述，若将第i轮的加密变换表示为T_i，即：

$$\{L_i, R_i\} = T_i\{L_{i-1}, R_{i-1}\}$$
$$= \{R_{i-1}, XOR(L_{i-1}, P(S(XOR(E(R_{i-1}), K_i))))\},$$
$$i=1,2,\ldots,16,$$

则DES算法的加密过程可以表示为：

$$DES_E(PT) = IP^{-1}T_{16}T_{15}\ldots T_2T_1IP(PT)$$

- DES算法的解密过程和加密过程完全类似，只不过将16轮的子密钥逆序使用，因此DES算法的解密过程可以表示为：

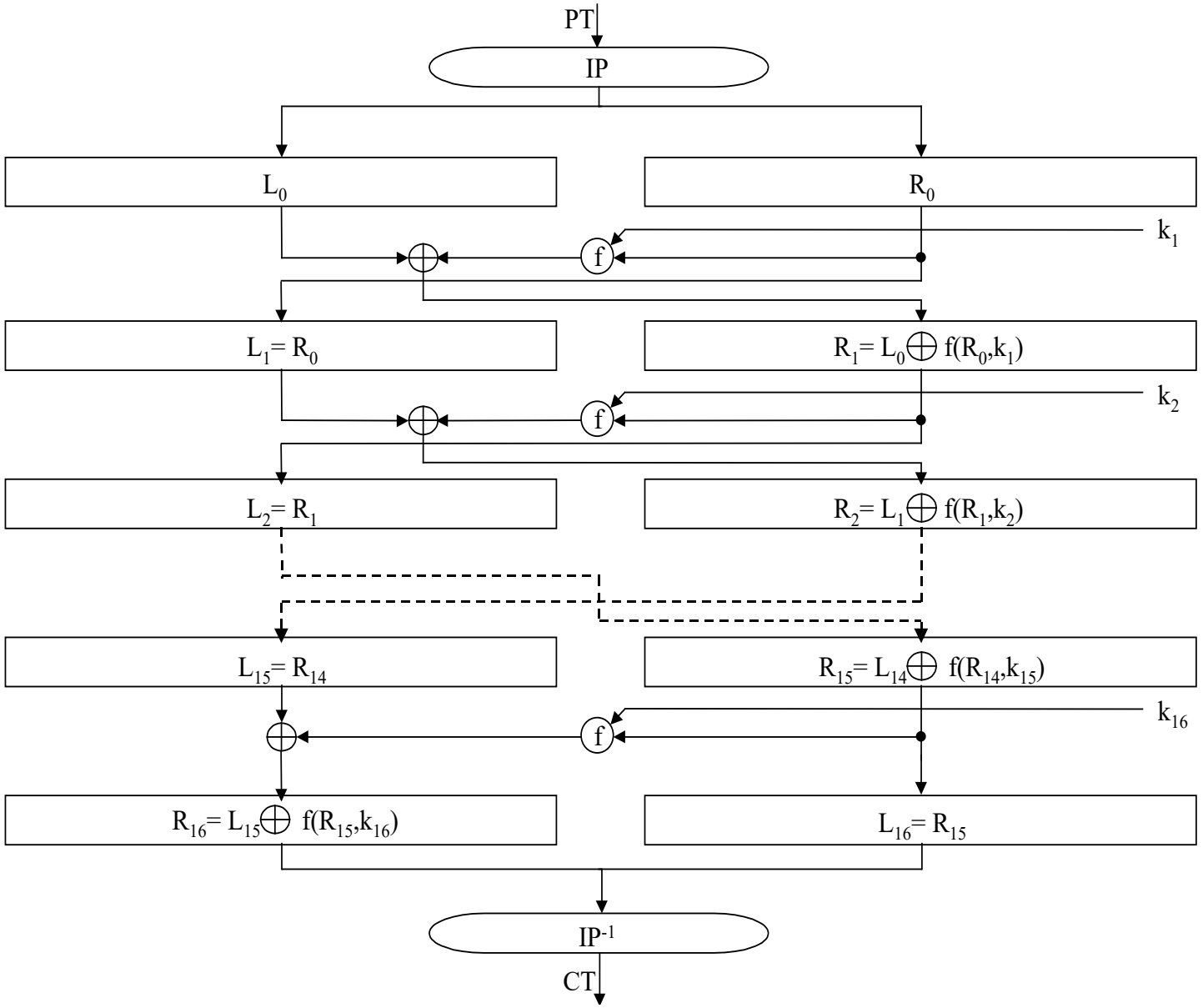
$$DES_D(CT) = IP^{-1}T_1T_2\ldots T_{15}T_{16}IP(CT)$$

- DES算法的加密变换和解密变换是互逆的，即：

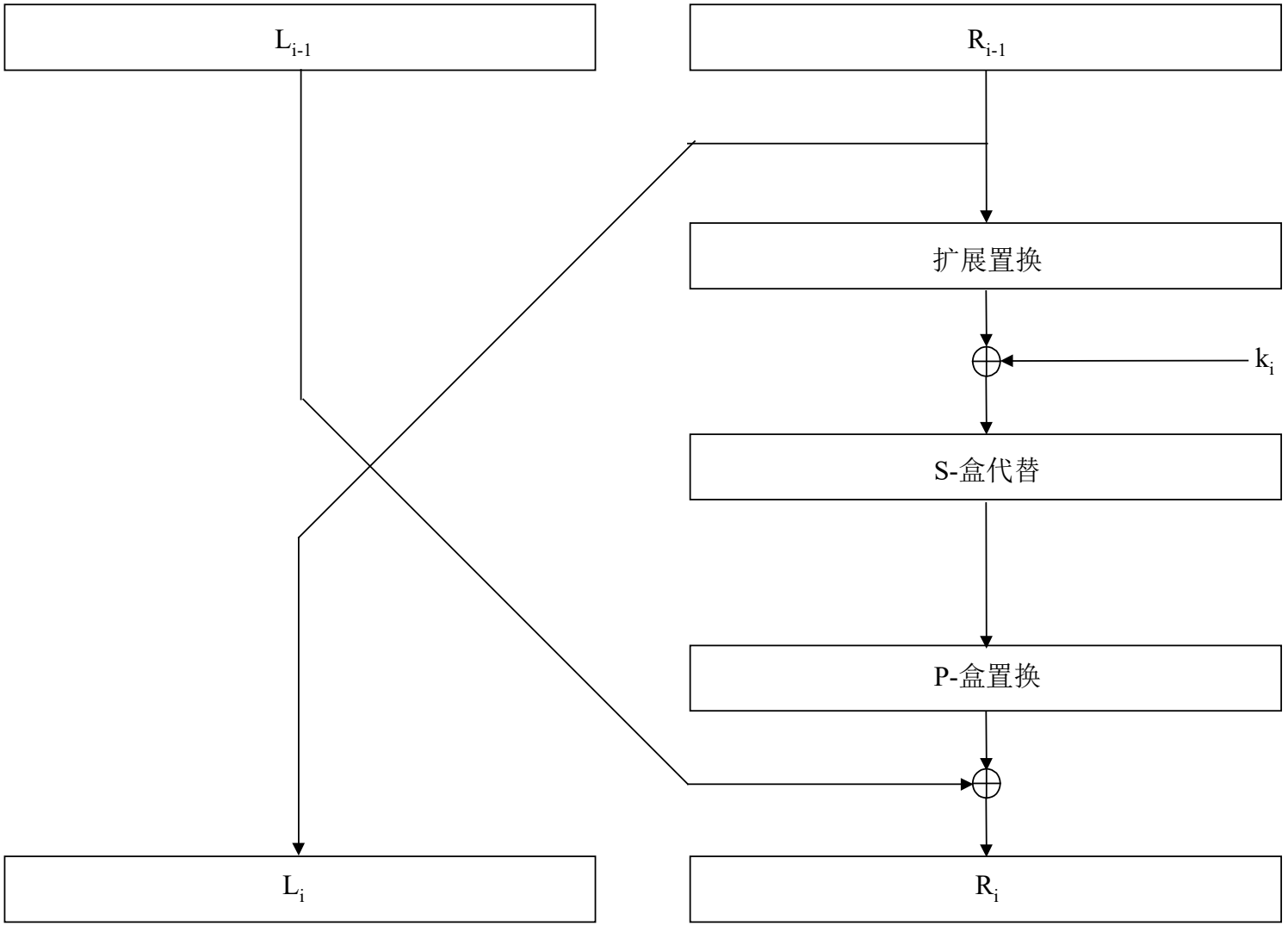
$$DES_D = DES_E^{-1}$$

- DES的加密过程、子密钥产生过程可以用下列流程图来表示：

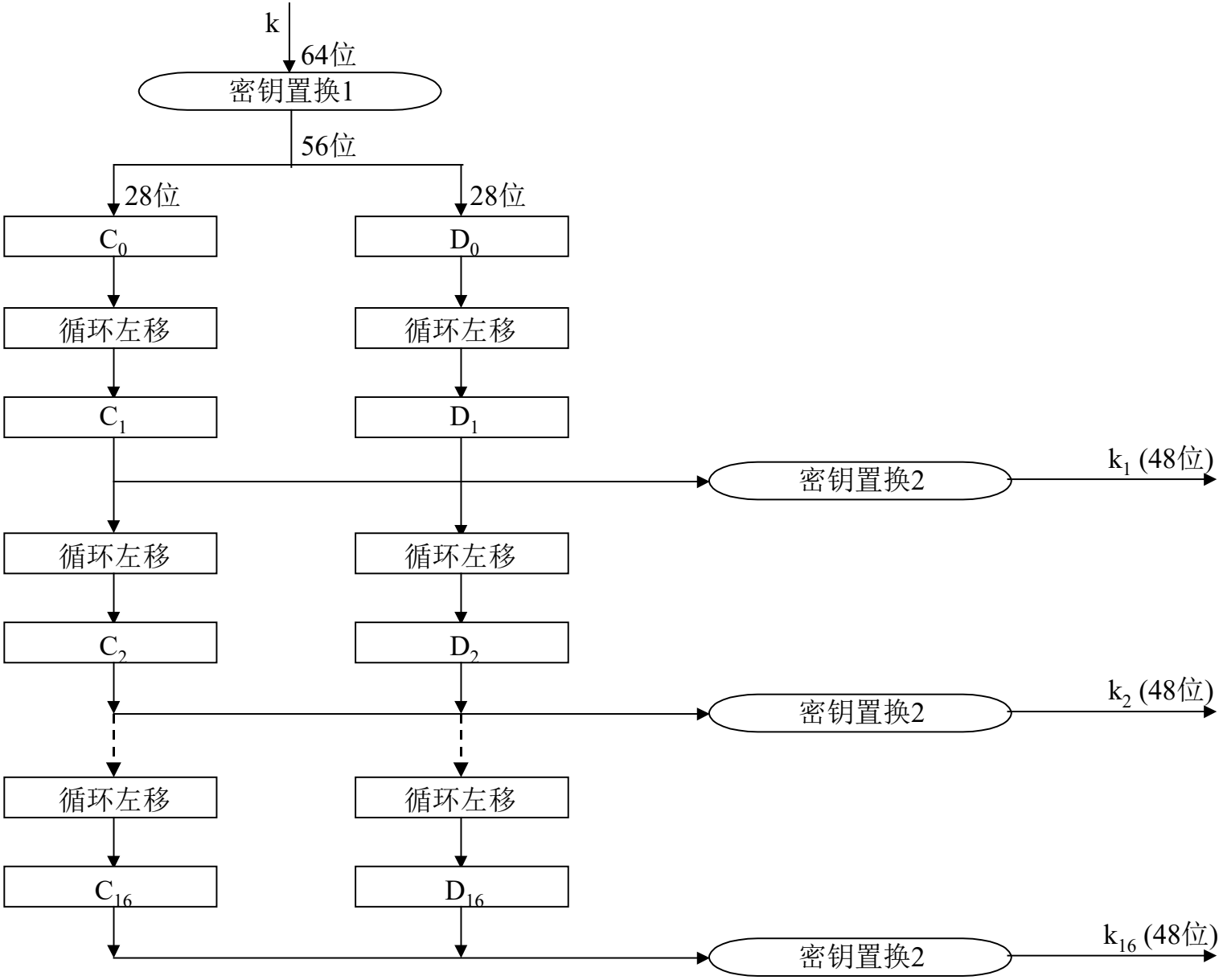
DES加密流程图


PT
IP
L_0
R_0
k_1
f
$L_1 = R_0$
$R_1 = L_0 \oplus f(R_0, k_1)$
k_2
f
$L_2 = R_1$
$R_2 = L_1 \oplus f(R_1, k_2)$
$L_{15} = R_{14}$
$R_{15} = L_{14} \oplus f(R_{14}, k_{15})$
k_{16}
f
$R_{16} = L_{15} \oplus f(R_{15}, k_{16})$
$L_{16} = R_{15}$
IP^{-1}
CT

DES的一轮加密流程图



DES的子密钥产生流程图



2. AES加密算法介绍

2.1 AES算法的数学知识

（1）有限域

群

一个交换群$(G,*)$由集合G和一个二元运算$*$：$G\times G\to G$构成，且满足下列性质：

(i) 结合律：对于所有的$a,b,c\in G$，有$a*(b*c)=(a*b)*c$。

(ii) 存在单位元：存在一个元素$e\in G$，使得$a*e=e*a=a$，对于所有的$a\in G$。

(iii) 存在逆元素：对于每一个$a\in G$，都存在一个元素$b\in G$，使得$a*b=b*a=e$，称元素b为a的逆元素。

(iv) 交换律：对于所有的$a,b\in G$，有$a*b=b*a$。

群的运算通常称为加法(+)或乘法(·)，并分别称为加法群和乘法群。定义了加法运算的群，称为加法群。加法群的单位元通常用0来表示，加法群的元素a的逆元素通常用$-a$来表示。定义了乘法运算的群，称为乘法群。乘法群的单位元通常用1来表示，乘法群的元素a的逆元素通常用a^{-1}来表示。若G是有限集合，则称其为有限群，并称G中元素的个数为群的阶。

域是对常见的数系（如有理数$\mathbb{Q}$、实数$\mathbb{R}$和复数$\mathbb{C}$）及其基本特性的抽象。域由一个集合$\mathbb{F}$和两种运算共同组成，这两种运算分别为加法（用$+$表示）和乘法（用$\cdot$表示），并且满足下列算术特性：

(i) $(\mathbb{F},+)$对于加法运算构成加法交换群，单位元用0表示。

(ii) $(\mathbb{F}\setminus\{0\},\cdot)$对于乘法运算构成乘法交换群，单位元用1表示。

(iii) 分配律成立：对于所有的$a,b,c\in\mathbb{F}$，都有$(a+b)\cdot c=a\cdot c+b\cdot c$。

若集合$\mathbb{F}$是有限集合，则称域为有限域。

➢在密码学中，集合中的元素个数有限，则如上定义的域为有限域，常用GF(p)表示，其中p表示域中的元素个数。在AES算法中用到的有限域是GF(2^8)，该域中的元素是由8bit组成的字节。它们可以通过异或做加法运算，也可以通过某种方式做乘法运算，而且每一个字节都有逆元。

（2）有限域的多项式

➢有限域中的元素可以用多种不同的方式表示，AES算法采用传统的多项式表示法，将$b_7b_6b_5b_4b_3b_2b_1b_0$构成的字节b看成系数在{0，1}上的多项式:$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$

例如：十六进制数{57}对应的二进制数为01010111，该字节对应的多项式为$x^6+x^4+x^2+x+1$。

（3）有限域的加法运算

- 在有限域的多项式表示中，GF(2^8)上两个元素的和仍然是一个次数不超过7的多项式，其系数等于两个元素对应系数的模2加（比特异或）。
- 由于AES算法中的字节被看作有限域上的元素，因此有限域中的加法运算相当于对两个字节进行异或操作。
- 例如：在GF(2^8)域中，16进制数57和83所表示的多项式的和为：

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$$

用二进制表示为

$$\{01010111\} \oplus \{10000011\} = \{11010100\}$$

用十六进制表示为

$$\{57\} \oplus \{83\} = \{d4\}$$

（4）有限域的乘法运算

- 要计算域GF(2^8)上的乘法，必须首先确立一个GF(2)上的8次不可约多项式；域GF(2^8)上的两个元素的乘积就是这两个多项式的模乘，以此8次不可约多项式为模。
- 在AES算法中，规定这个8次不可约多项式为：

$$m(x)=x^8+x^4+x^3+x+1$$

- 例如：16进制数{57}和{83}所表示的多项式的乘积等于{c1}，计算如下：

$$(x^6+x^4+x^2+x+1)\cdot(x^7+x+1)=$$

$$x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1$$

$$= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$$

而$x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$

$$=x^7+x^6+1 (\mathrm{mod}(x^8+x^4+x^3+x+1))$$

（5）有限域的X乘运算

- GF(2^8)上还定义了一种运算，称之为x乘法，其定义为：

假设$b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$是GF($2^8$)上的多项式，则

$$x\cdot b(x)=b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x \mod(m(x))$$

- 如果$b_7=0$，求模结果不变，否则为乘积结果减去m(x)，即求乘积结果与m(x)的异或。
- 由此得出x（十六进制{02}）乘b(x)可以先对b(x)在字节内左移一位（最后一位补0）；如果$b_7=1$，再与十六进制数{1b}做逐比特异或来实现。该运算记为b=xtime(x)。
- x的幂乘运算可以重复应用xtime函数来实现。
- 例如：{57}·{13}可按如下方式实现：

{57}·{02}=xtime({57})={ae}

{57}·{04}=xtime({ae})={47}

{57}·{08}=xtime({47})={8e}

{57}·{10}=xtime({8e})={07}

{57}·{13}={57}·({01}⊕{02}⊕{10})

={57}⊕{ae}⊕{07}

={fe}

(6) 四字节运算

- 定义一个4字节字{$a_3a_2a_1a_0$ }（a_i为一个字节）和一个次数小于4的多项式$a_3x^3 + a_2x^2 + a_1x + a_0$相对应。
- 两个4字节字的加法等于对应多项式相应系数之和，即

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

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

- 两个4字节字的乘法为对应的两个多项式相乘再模$x^4 + 1$，即

$(a(x) \otimes b(x)) \bmod (x^4 + 1) = d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

其中：

$d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3)$

$d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$

$d_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3)$

$d_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)$.

- 用矩阵表示为：

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

2.2 状态矩阵和轮数

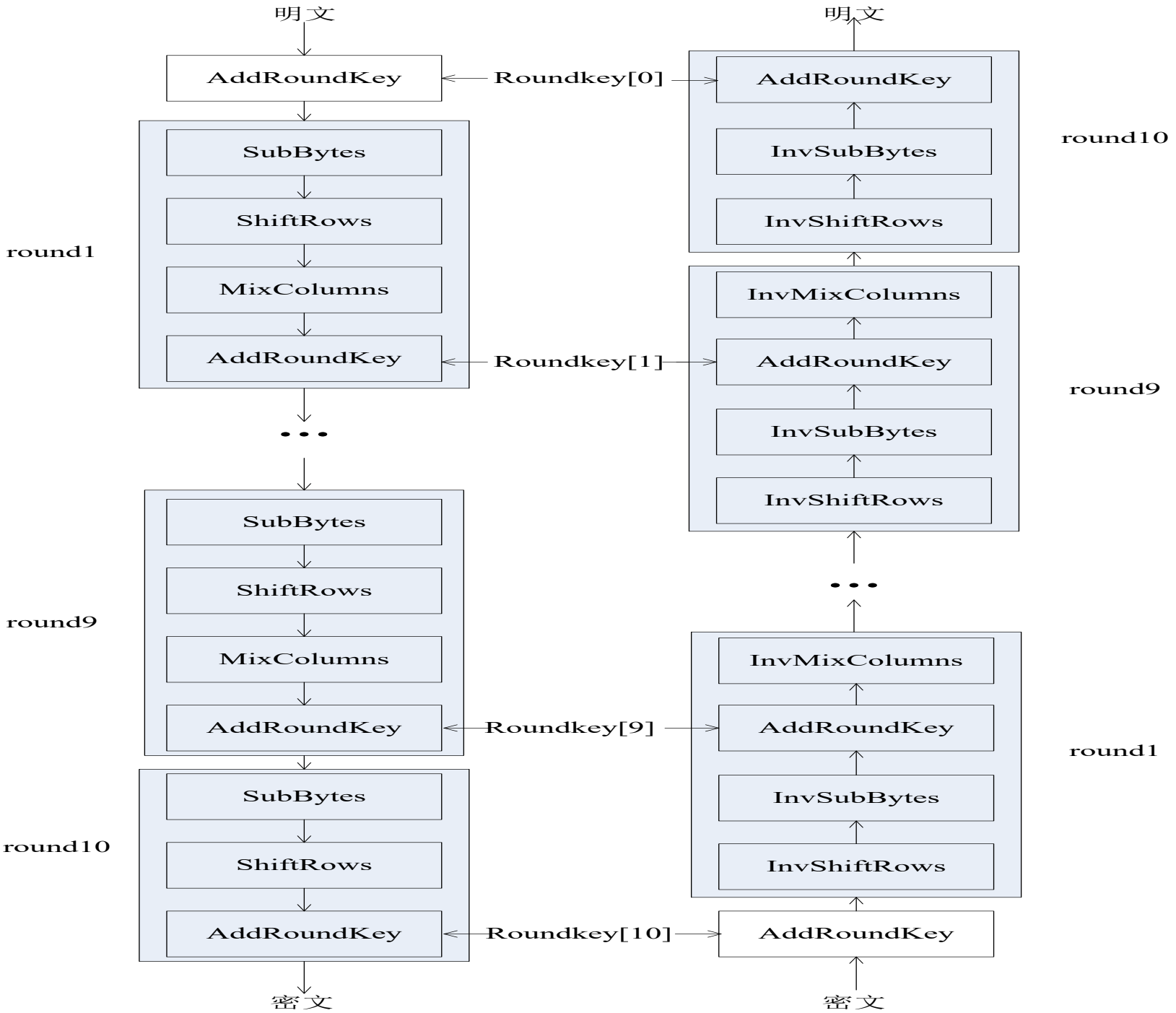
- AES算法是一种迭代型分组密码算法，NIST确定的AES算法标准将分组长度固定为128位，支持128位、192位和256位的密钥长度。
- AES算法由加密、解密和密钥扩展三种基本运算构成，所有的运算都是基于完整的字节（byte）操作。
- 类似于明文分组和密文分组，算法的中间结果也必须分组，称加密和解密过程中所产生的中间结果的分组为状态（state）。
- 状态可以用一个以字节为元素的二维矩阵来表示，即状态矩阵，该矩阵的行数为4，列数为Nb(Nb=4)。
- 加密时将输入字节按照$a_{0,0}$，$a_{1,0}$，$a_{2,0}$，$a_{3,0}$；$a_{0,1}$，$a_{1,1}$，$a_{2,1}$，$a_{3,1}$...的顺序映射到状态矩阵上，加密的最后以同样的顺序提取状态矩阵从而得到输出字节。
- 类似地，密钥也可以看成是由字节构成的二维数组，该数组也可以用一个行数为4，列数为Nk的矩阵来表示，其中Nk的值为密钥长度除以32。
- AES加密运算和解密运算都采取轮迭代结构，运算的轮数记为Nr，与Nb，Nk有关，它们之间的关系如下表所示。

- AES算法的密钥长度、分组长度、轮数之间的关系

密钥长度（Nk）	分组长度（Nb）	轮数（Nr）
4	4	10
6	4	12
8	4	14

2.3 AES算法的整体结构

- AES加密算法由三部分组成：初始密钥加，Nr-1轮循环加密，最后一轮加密。其中每一轮有4个基本的运算步骤，并且，每一轮都有一个由初始密钥生成的轮密钥。
- AES算法的加密流程和解密流程如下图所示。


明文
AddRoundKey
Roundkey[0]
SubBytes
ShiftRows
MixColumns
AddRoundKey
round1
Roundkey[1]
SubBytes
ShiftRows
MixColumns
AddRoundKey
round9
Roundkey[9]
SubBytes
ShiftRows
AddRoundKey
round10
Roundkey[10]
密文
明文
AddRoundKey
InvSubBytes
InvShiftRows
round10
InvMixColumns
AddRoundKey
InvSubBytes
InvShiftRows
round9
InvMixColumns
AddRoundKey
InvSubBytes
InvShiftRows
round1
AddRoundKey
密文

➢AES加密过程可用伪代码描述为：

```
Cipher(byte in[4*4],byte out[4*4],word w[4*(Nr+1)])
  Begin
    Byte state[4,4]
    State=in
    AddRoundKey(state,w[0,3])
    For round=1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state,w[round*4,(round+1)*4-1])
    End for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state,w[Nr*4,(Nr+1)*4-1])
    Out=state
  End
```

➢AES解密过程是加密过程的逆过程，可用伪代码描述为：

```
InvCipher(byte in[4*4],byte out[4*4],word w[4*(Nr+1)])
  Begin
    Byte state[4,4]
    State=in
    AddRoundKey(state, w[Nr*4, (Nr+1)*4-1])
    For round= Nr-1 step -1 to 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state,w[round*4,(round+1)*4-1])
      InvMixColumns(state)
    End for
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[0,3])
    Out=state
  End
```

2.4 AES算法描述

- AES算法采取轮迭代结构，每轮变换由四个不同的变换组成。
- 加密过程的这四个变换分别是字节变换（SubBytes），行移位变换（ShiftRows），列混合变换（MixColumns）和轮密钥加变换（AddRoundKey）。
- 解密过程所对应的轮变换分别是这四个变换的逆变换。

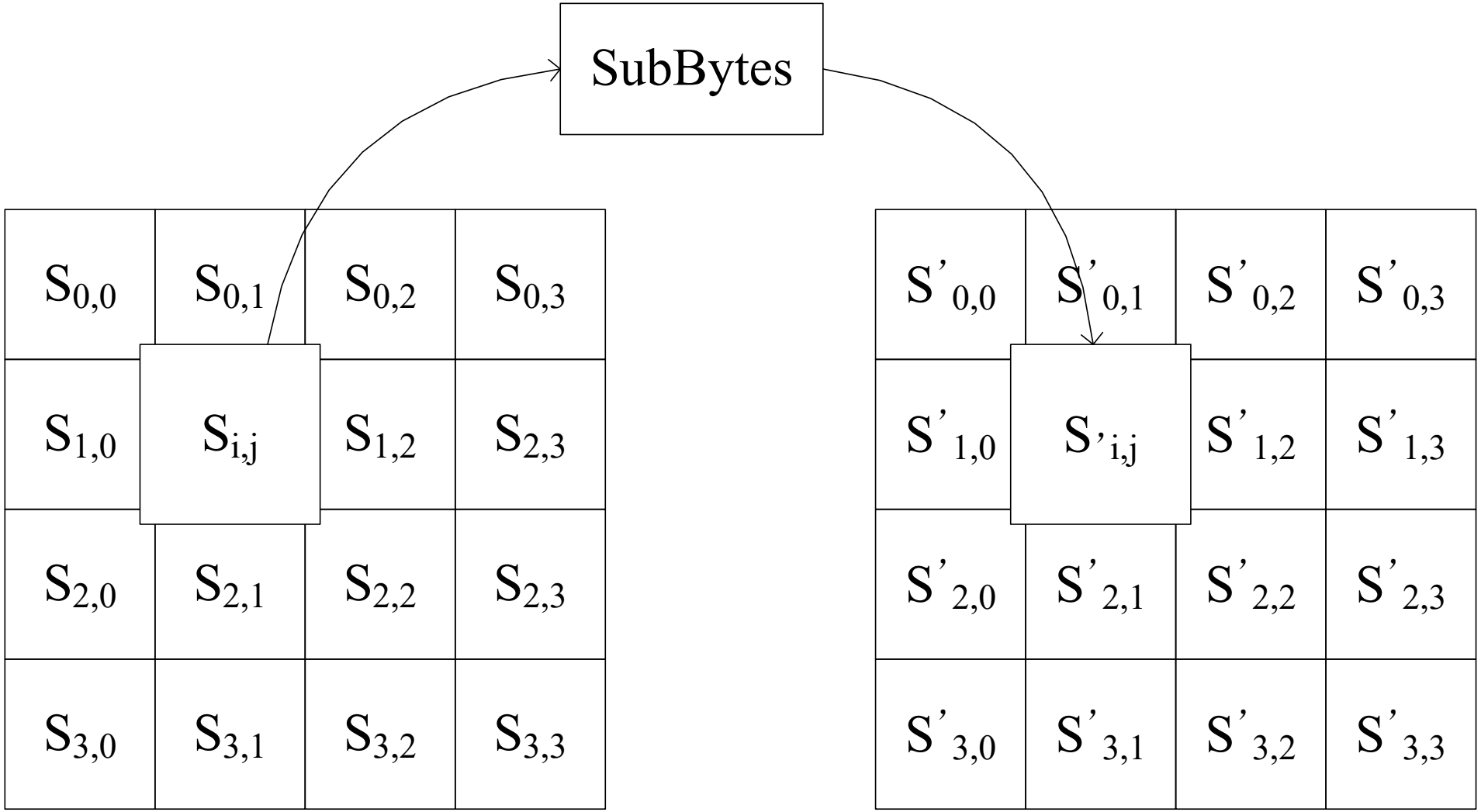
（1）字节变换（SubBytes）和逆字节变换（InvSubBytes）

- 字节变换是一种非线性、可逆变换，它将状态矩阵中的每一个字节进行变换。它由以下两个变换的合成得到。

- 首先，取有限域GF(2^8)中的乘法逆，即把每一个字节的值用它的乘法逆代替。其中，“00”的逆是其本身。
- 然后，在GF(2^8)中作下式所示的仿射变换。

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

- 下图说明了SubBytes作用在状态矩阵的结果。


SubBytes
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{i,j}$ $S_{1,2}$ $S_{2,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S'_{0,0}$ $S'_{0,1}$ $S'_{0,2}$ $S'_{0,3}$
$S'_{1,0}$ $S'_{i,j}$ $S'_{1,2}$ $S'_{1,3}$
$S'_{2,0}$ $S'_{2,1}$ $S'_{2,2}$ $S'_{2,3}$
$S'_{3,0}$ $S'_{3,1}$ $S'_{3,2}$ $S'_{3,3}$

- 将以上变换作用于所有可能字节，便可构造出SubBytes的查找表，如表2-2所示，该表被称为S盒。
- 通过查找S盒可以直接得到SubBytes的结果。如状态矩阵中$S_{1,1}$={53}，则S盒中第5行第3列的字节{ed}即为SubBytes后字节$S'_{1,1}$的值。
- 逆字节替换InvSubBytes是字节替换SubBytes的逆变换,它与SubBytes变换的运算过程刚好相反，通过如下两步实现：首先进行与SubBytes相同的仿射变换的逆变换，最后求其逆。
- 同样的，InvSubBytes也可以用S盒查找表的方式实现，该查找表被称为逆S盒，如表2-3所示。

表2-2 S盒

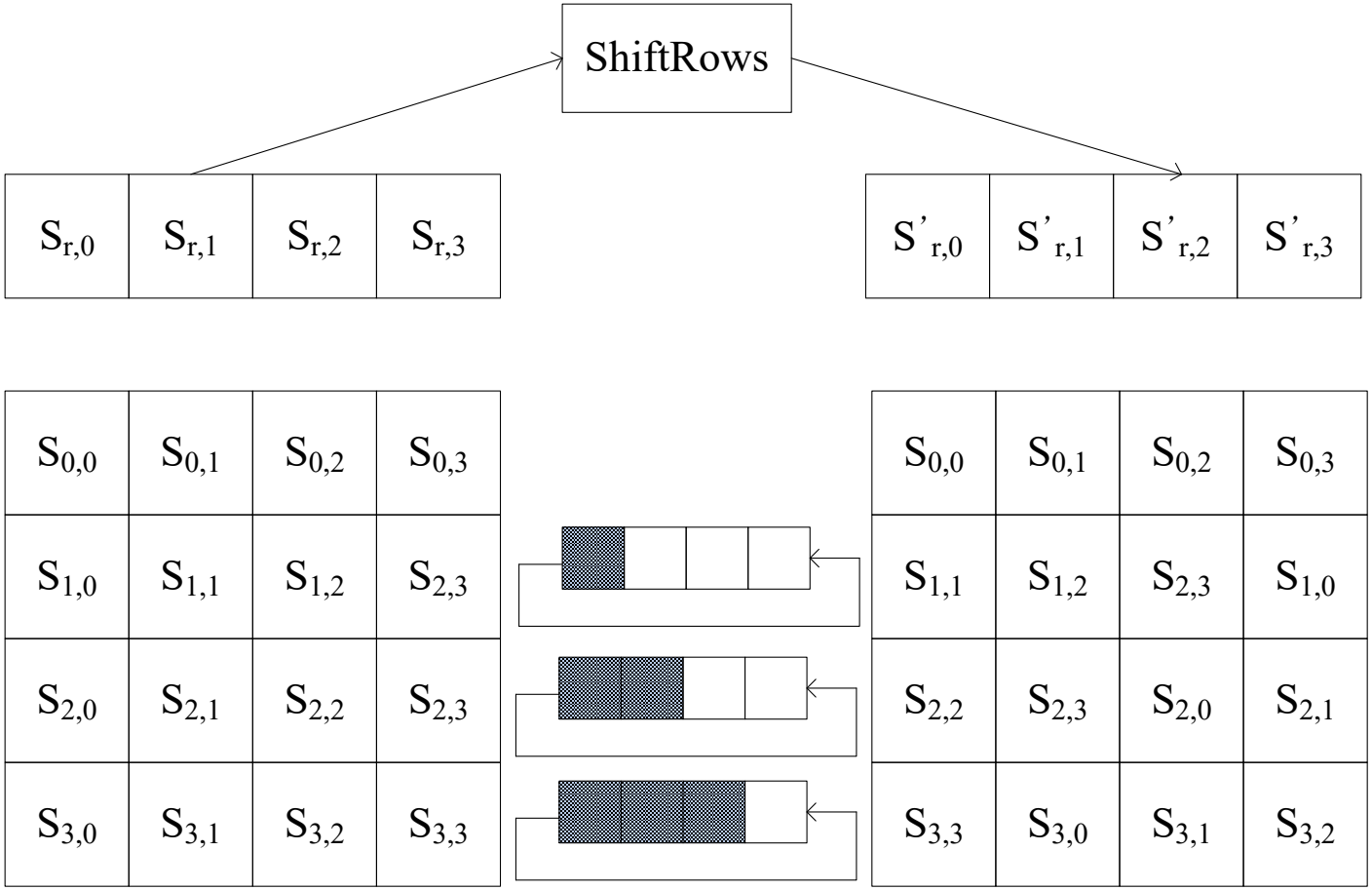
		列号y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
行号 x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	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

表2-3 逆S盒

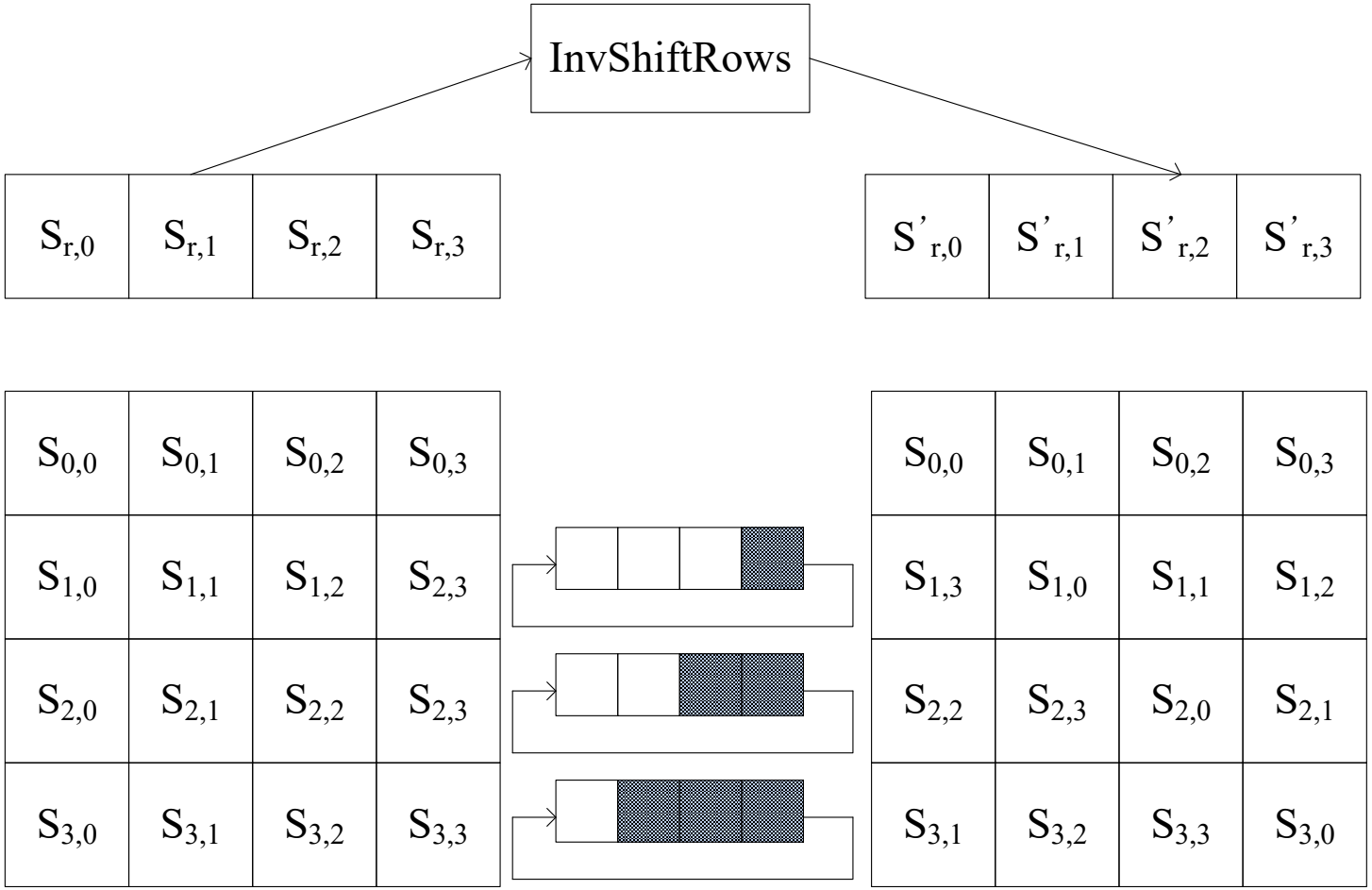
		列号y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
行号 x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	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）行移位变换（ShiftRows）和逆行移位变换（InvShiftRows）

- 行移位变换是将状态矩阵的各行进行左循环移位，如下图所示。
- 不同状态行的位移量不同，第0行不移动，第1行循环左移1个字节，第二行循环左移2个字节，第三行循环左移3个字节。



➢逆行移位变换InvShiftRows是行移位变换ShiftRows的逆过程。
➢即状态矩阵的后3行依次左循环移动3，2，1个字节，如下图所示。



（3）列混合变换（MixColumns）和逆列混合变换（InvMixColumns）

- 列混合变换是将状态矩阵的各列进行变换，如图2-5所示。
- 在变换中，它将状态矩阵的每一列看作系数在GF(2^8)上的多项式，与固定多项式a(x)进行模(x^4+1)相乘，而$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$。
- 这种运算可以用矩阵乘法来表示，记为$s'(x)=a(x)\cdot s(x)$，如式2-2所示。

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix} \quad 0 \le c \le Nb \quad \text{式（2-2）}$$

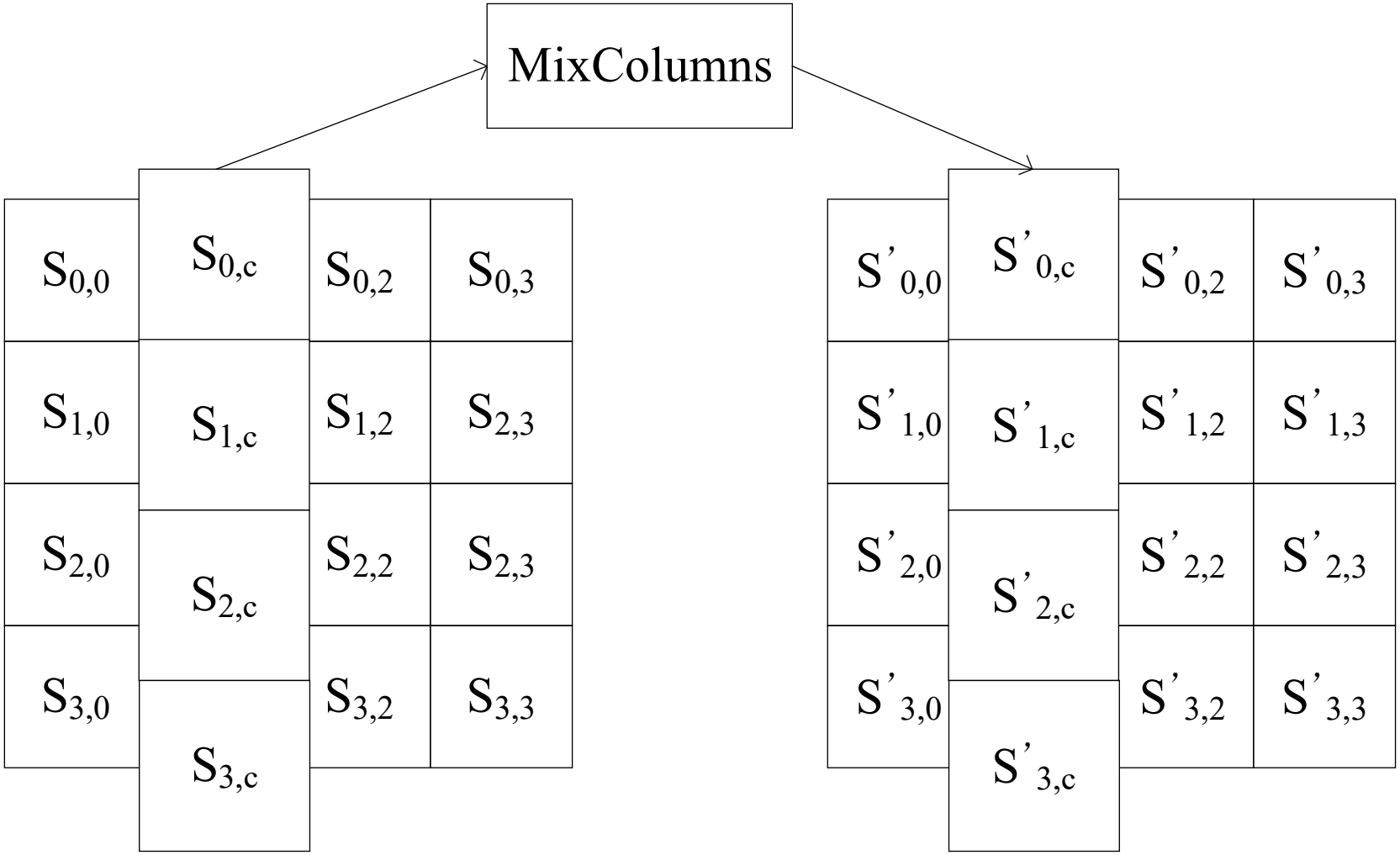


图2-5 MixColumns变换

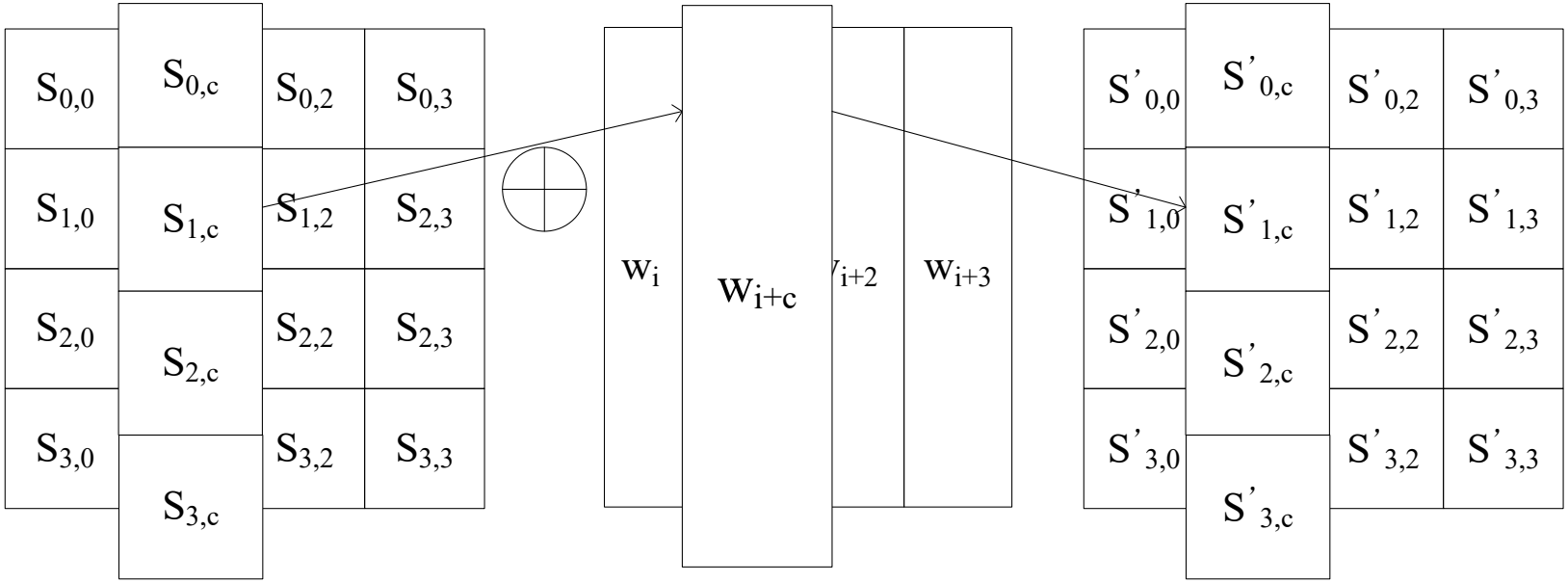
- 逆列混合变换InvMixColumns与列混合变换MixColumns相似，只是固定多项式变为a(x)的逆$a^{-1}(x)$，则$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\}$。
- 同样基于以上的状态转换，解密运算的InvMixColumns可以表示为式2-3。

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = \begin{vmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{vmatrix} \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix} \quad 0 \le c \le Nb$$

式（2-3）

（4）轮密钥加变换（AddRoundKey）

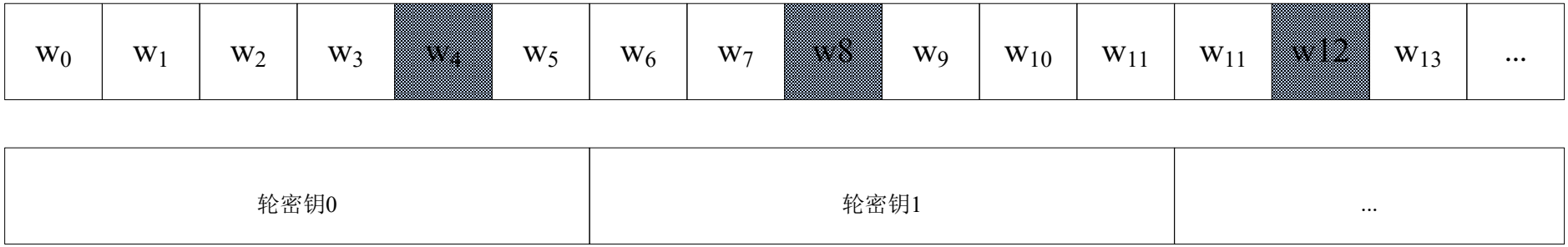
- 轮密钥加变换是将轮密钥与状态矩阵进行逐位异或运算，如下图所示，而轮密钥加变换的逆变换仍然是其本身。
- 轮密钥按顺序取自扩展密钥，而扩展密钥是由初始密钥经密钥扩展算法得到。
- 虽然初始密钥长度不同，但是在加解密过程中，轮密钥的长度必须与加密数据分组长度相同，因此，轮密钥的选取也是至关重要的。



i=round*Nb

2.5 密钥扩展

➢AES算法首先得到初始密钥K后，执行一个密钥扩展程序以产生所有的轮密钥。密钥扩展共产生Nb（Nr+1）个32bit字，算法初始需要一个Nb个32bit字的集合，接着每个轮操作都需要Nb个32bit字的密钥数据。



➢密钥扩展程序涉及RotWord()、SubWord()和Rcon()变换。它们的工作方式如下：

(1)RotWord()：把一个4字节的输入序列（a0，a1，a2，a3）循环左移一个字节后输出。例如将（a0，a1，a2，a3）循环左移一个字节后输出为（a1，a2，a3，a0）。

(2)SubWord()：把一个4字节的输入序列（a0，a1，a2，a3）的每一个字节进行S盒变换，然后作为输出。

(3)Rcon[]：Rcon[]是一个具有10个元素的常量数组，Rcon[i]是一个32比特字符串（x^{i-1},00,00,00）。这里x=(02)，x^{i-1}是x=(02)的（i-1）次幂的十六进制表示，即x^0=(01),x=(02),x^i={02}·x^{i-1}。这里“·”表示有限域GF(2^8)中的乘法。

- 密钥扩展前Nk个字就是外部密钥CipherKey，以后的字w[i]等于它前一个字w[i-1]与前Nk个字w[i-Nk]经过运算得到，

  w[i]=w[i-Nk]⊕SubWord(RotWord(w[i-1]))⊕Rcon[i/Nk]。

- 密钥扩展的整个过程可以用下面的程序段描述：

```
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
     while (i < Nb * (Nr+1))
         temp = w[i - 1]
         if ( i mod Nk = 0 )
            temp = Subword(Rotword(temp)) xor Rcon[i/Nk]
         else if (Nk > 6 and i mod Nk = 4)
            temp = Subword (temp)
         end if
         w [i] = w[i - Nk] xor temp
         i = i +1
       end while
 end
```

2.6 AES算法工作模式

- 所谓分组密码的工作模式是指以该分组密码为基础构造的一个密码系统。
- 1980年12月NIST公布了DES的4种工作模式：电子密码本（ECB）模式，密码分组链接（CBC）模式，密码反馈（CFB）模式，和输出反馈（OFB）模式。
- 这些工作模式可以适用于任何分组密码。因此在AES算法诞生之后，这4种模式便被定义为AES的常用工作模式。现分别讨论这四种工作模式。

（1）电子密码本（ECB）模式

- ECB(Electronic Code Book)模式是最简单的工作模式。它每次处理128位的明文分组，并且每个明文分组都用同一个初始密钥加密。
- 下图是ECB模式的示意图。加密过程中首先把明文分成128位的数据分组$P_1P_2...P_N$，如果最后一个分组P_N不足128位，则需进行填充。它们对应的密文分组就是$C_1C_2...C_N$。解密过程也是每次对一个分组解密，而且每次解密都是用相同的密钥。

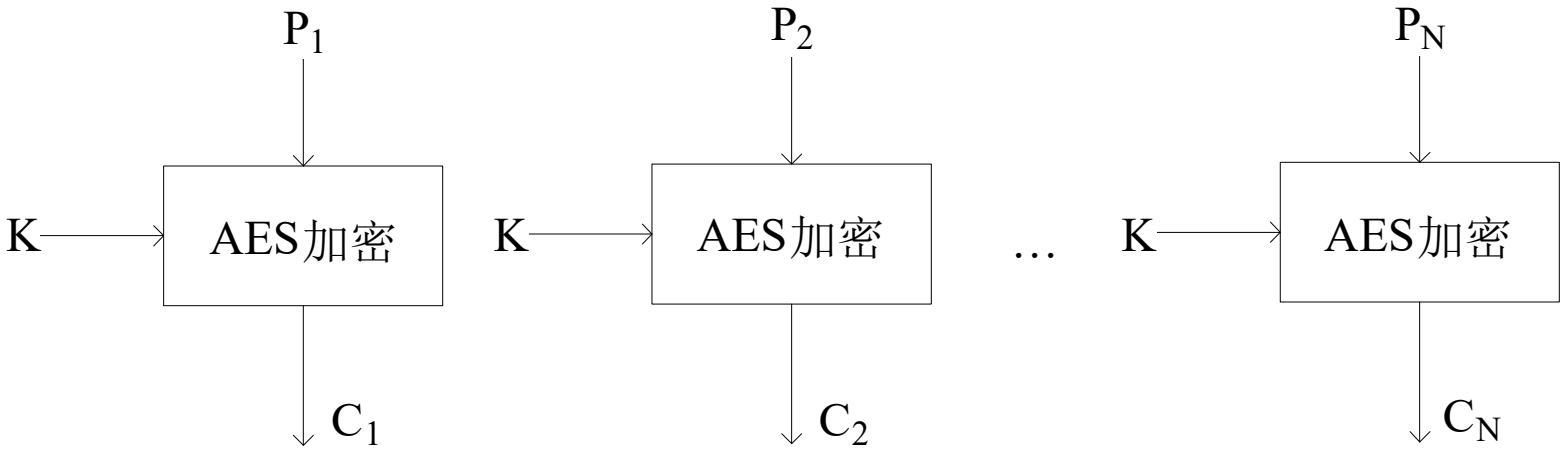

P1
P2
PN
K
AES加密
K
AES加密
…
K
AES加密
C1
C2
CN
加密
C1
C2
CN
K
AES解密
K
AES解密
…
K
AES解密
P1
P2
PN
解密

图1 ECB模式示意图

（2）密码分组链接（CBC）模式

➢CBC(Cipher Block Chaining)模式克服了ECB模式的缺陷，使相同的明文在重复出现时可以获得不同的密文。

➢下图是CBC模式的示意图。加密时，它使用同一个密钥对每一个明文分组加密，但是加密时的输入是当前明文和前面密文异或的结果，因此每次加密时的输入不会显示出与本次明文分组之间的固定关系，所以重复的明文分组会产生不同的密文分组。同样在解密时，每一个密文分组在解密后，需要与前一个密文分组进行异或，从而产生出明文分组。

➢使用CBC模式还存在一个问题，即产生第一个密文分组时，需要有一个初始向量IV，将它与第一个明文分组异或。解密时，IV与第一个密文分组的解密输出进行异或，从而得到第一个明文分组。由于IV对于收发双方是已知的，因此应像密钥一样被保护。

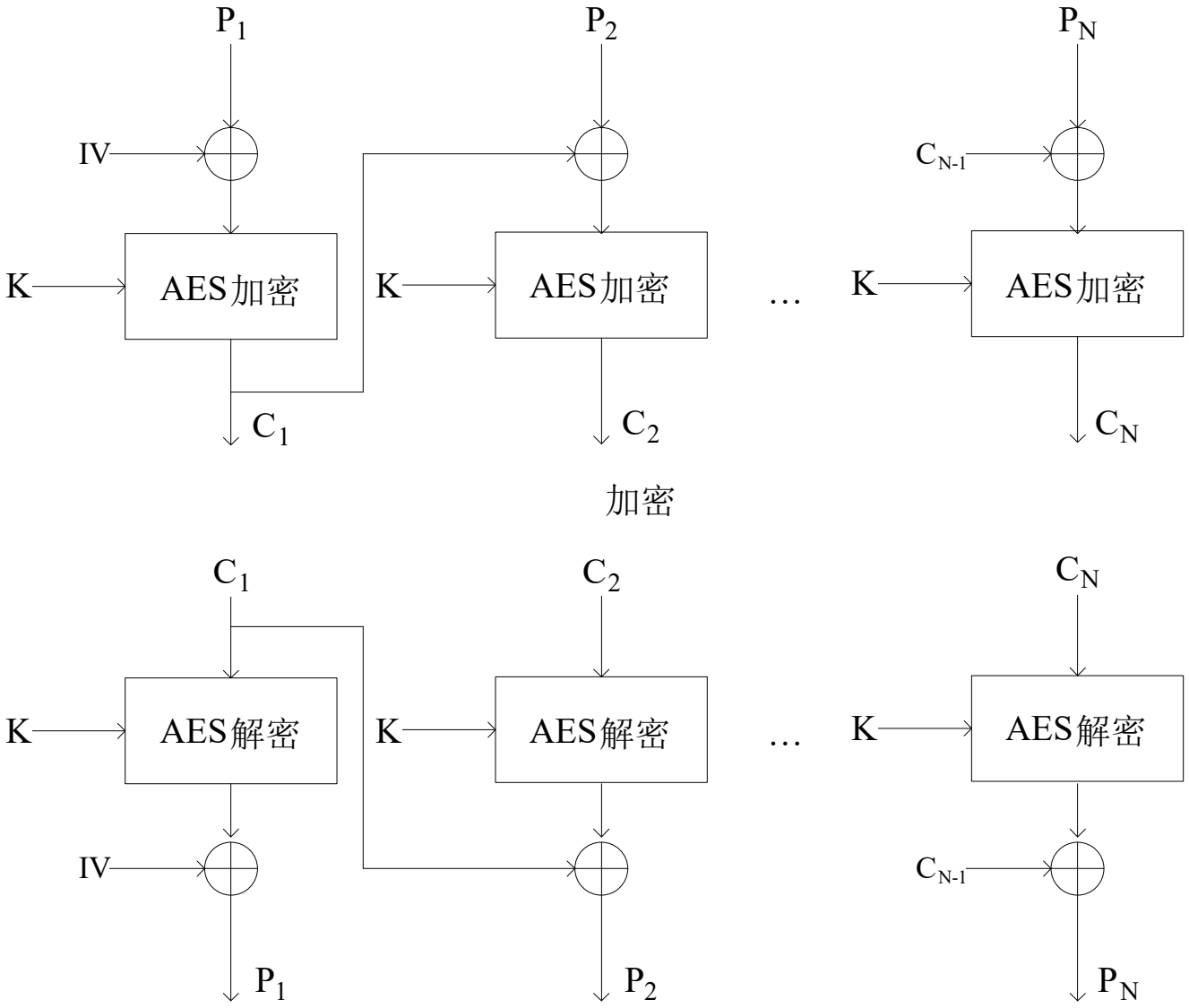

P1
P2
PN
IV
CN-1
K
AES加密
K
AES加密
…
K
AES加密
C1
C2
CN
加密
C1
C2
CN
K
AES解密
K
AES解密
…
K
AES解密
IV
CN-1
P1
P2
PN
解密

图2 CBC模式示意图

（3）密码反馈（CFB）模式

- 如上所述，ECB模式和CBC模式有一个共有的问题，即只对一个完整的分组进行加密或解密。而CFB(Cipher Feedback)模式是一个利用分组生成随机位的流密码工作模式。流密码不需要对消息进行填充，而且运行是实时的。
- 下图是CFB模式的示意图。加密时，加密算法的输入是128位移位寄存器，其初始值是初始向量IV。加密算法输出的最左j比特（取j=8）与j比特的明文P1异或，从而产生出第一个密文单元C1。然后将移位寄存器左移j位，并将C1送入移位寄存器的最右j位，用同样的加密方式产生出密文单元C2，直到所有的明文单元都被加密为止。解密时，将密文单元与解密的输出进行异或。值得注意的是，解密时仍然使用加密算法。

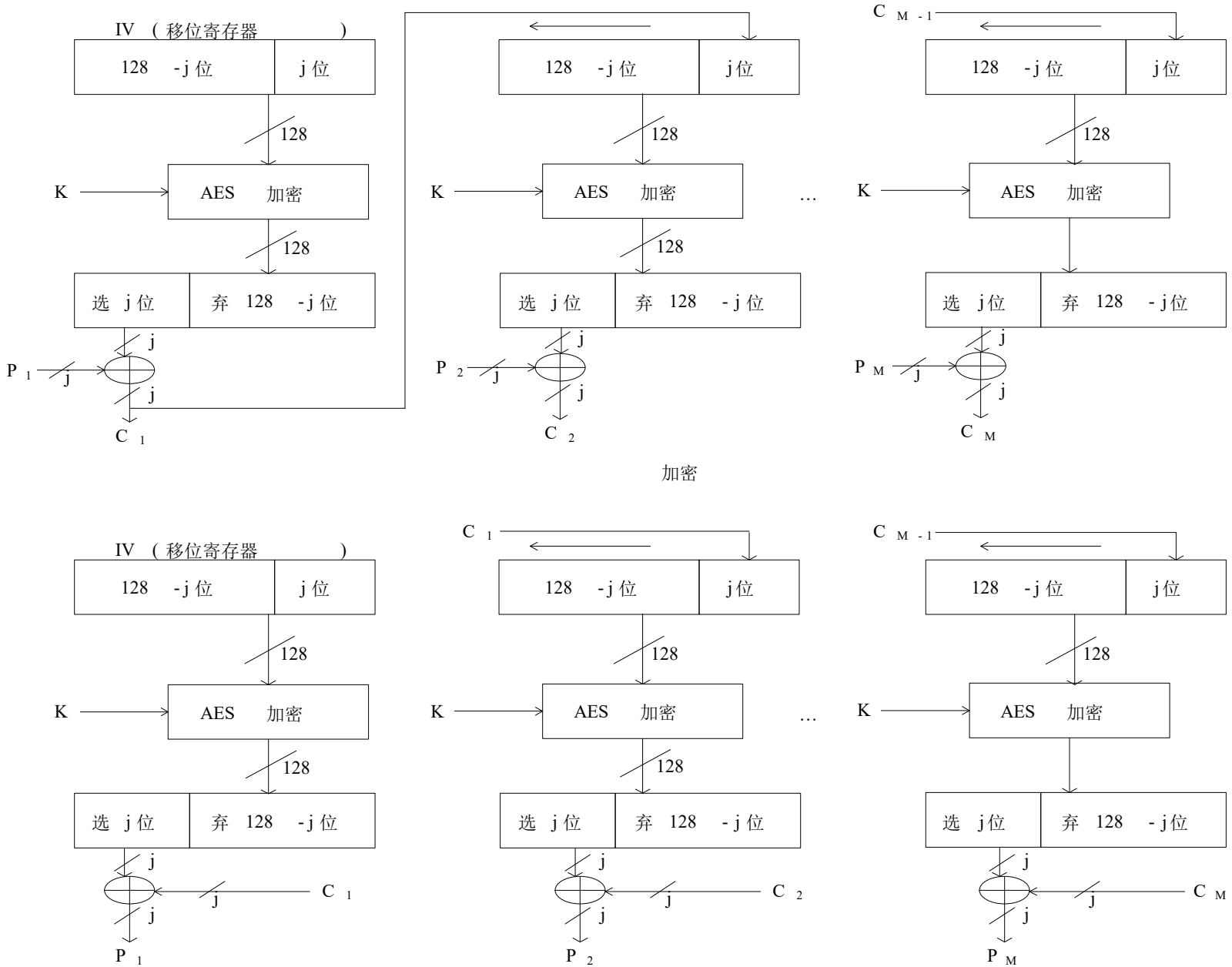
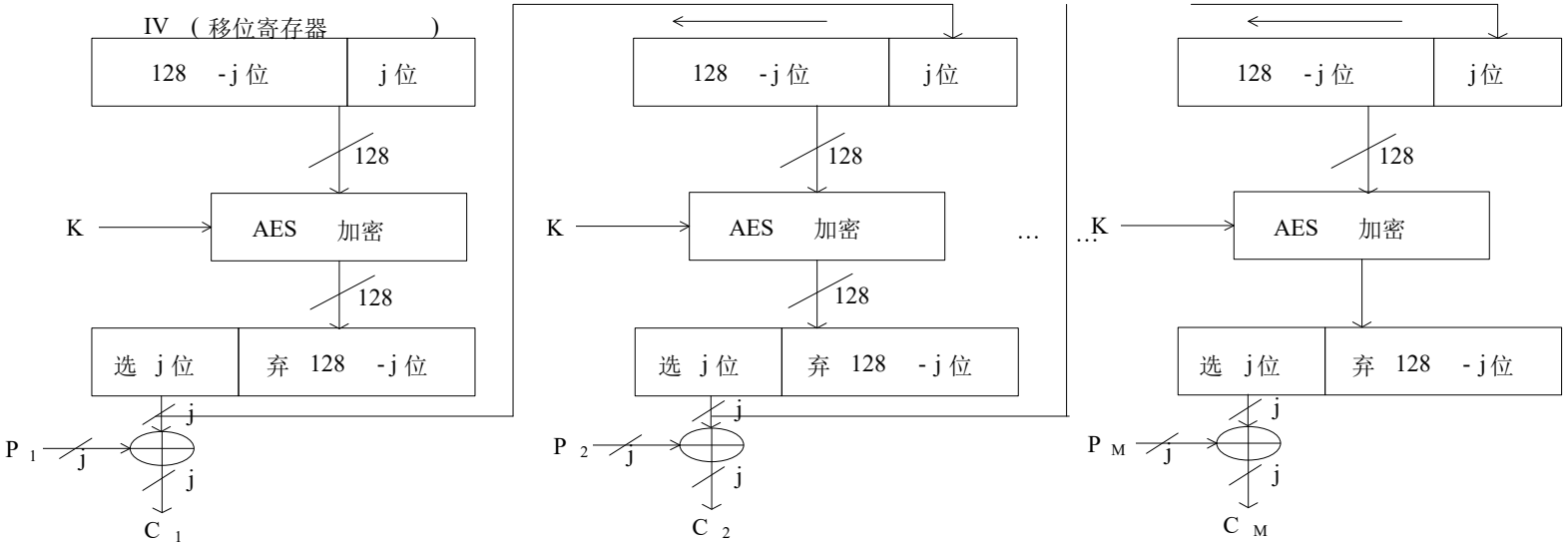

IV （移位寄存器 ）
128 -j 位
j 位
128
K
AES 加密
128
选 j 位
弃 128 -j 位
j
P 1
C 1
P 2
C 2
C M - 1
P M
C M
…
加密
C 1
C 2
C M - 1
解密

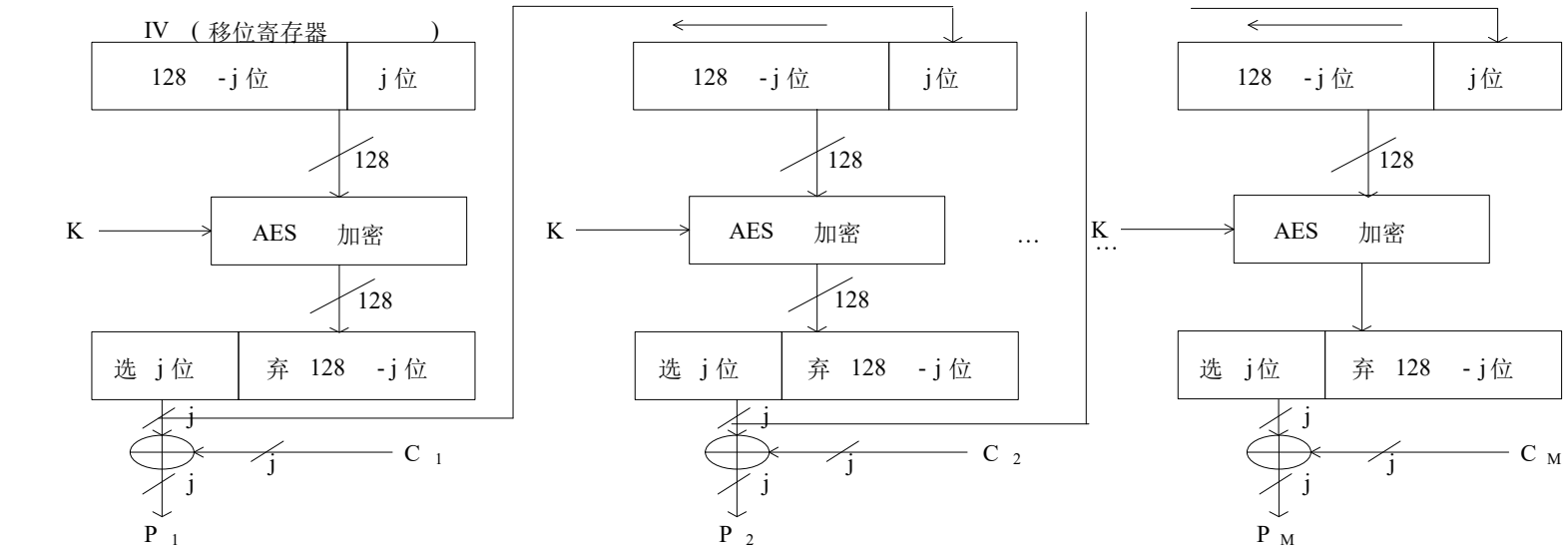
图 3 CFB 模式示意图

（4）输出反馈（OFB）模式

- OFB(Output Feedback)模式是另一个流密码工作模式的例子，其结构与CFB模式相类似，如下图所示。不同之处如下：OFB模式是将加密算法的输出反馈到移位寄存器，而CFB模式是将密文单元反馈到移位寄存器。
- 相对于CFB模式，OFB模式的优点是传输过程中的错误不会被传播。例如解密过程中C1中出现了错误，而解密结果中只有P1受到了影响，以后的各个明文单元则不受影响。但是在CFB模式中，由于C1被作为移位寄存器的输入，因此它的错误将会影响解密结果中所有的明文单元的值。



加密



解密

图 4 OFB 模式示意图

3.AES密码处理器的体系结构设计

3.1 AES密码处理器框图及外部信号说明

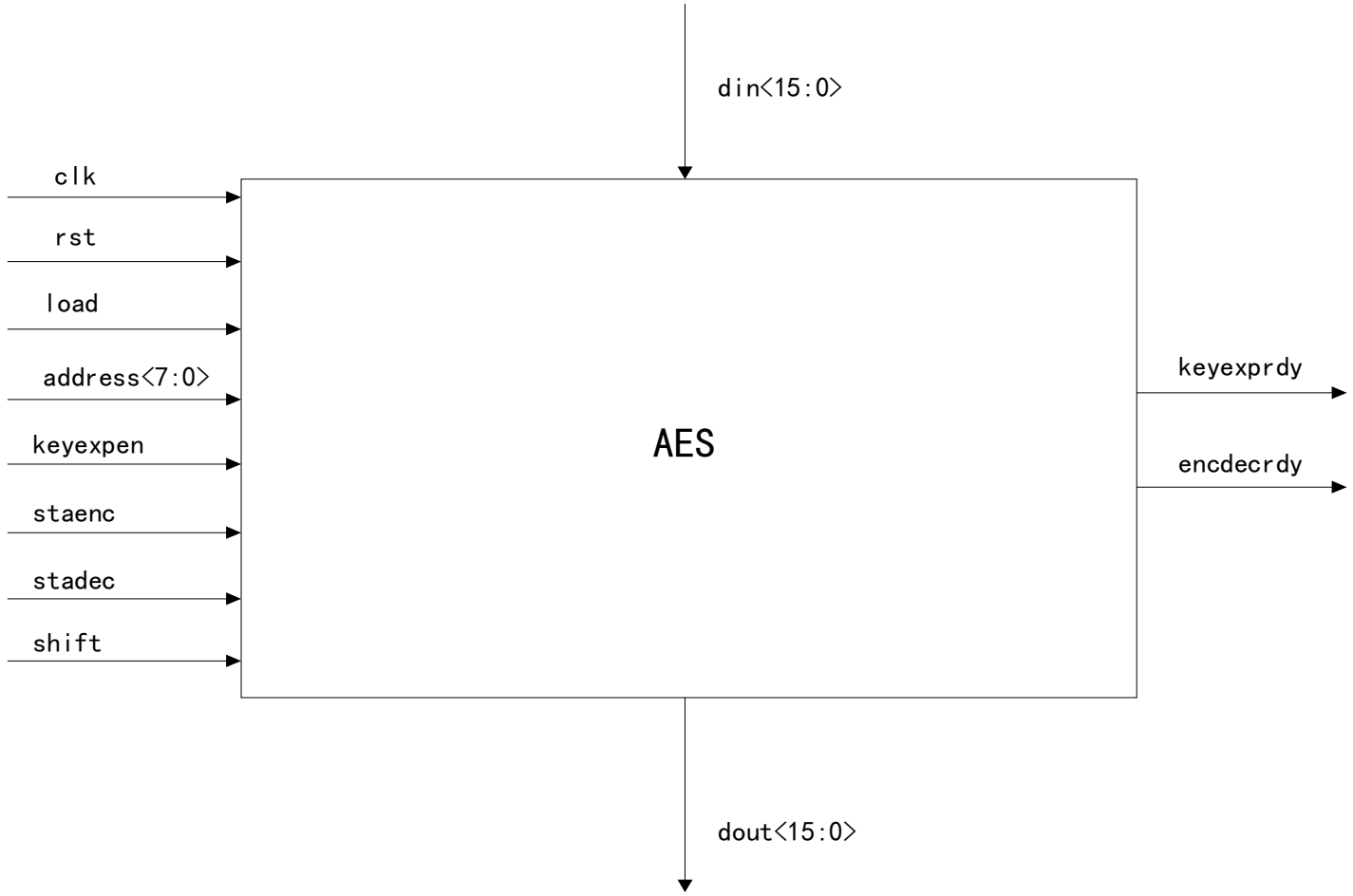
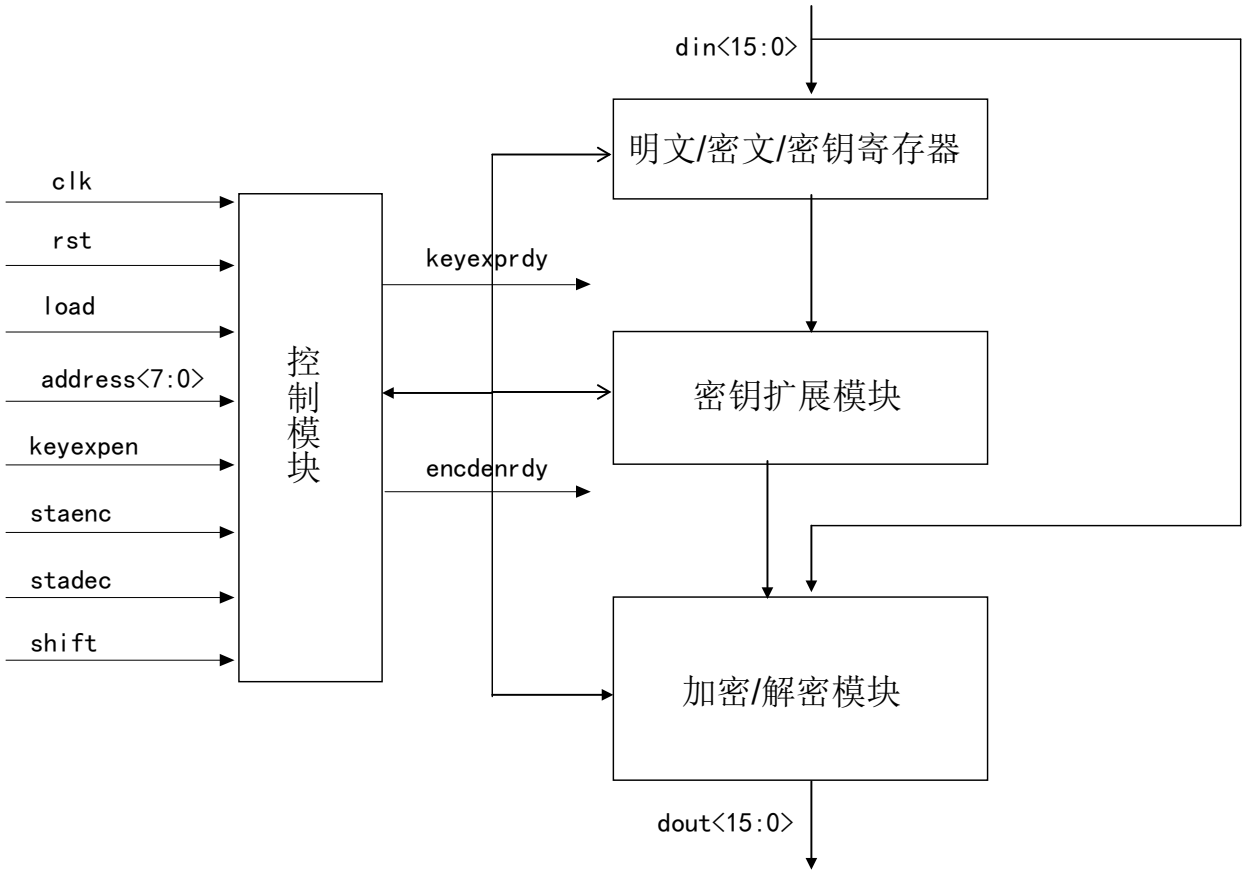


表1 AES密码处理器外部信号说明

信号名称	信号宽度	传输方向	信号含义
clk	1位	输入	时钟信号
rst	1位	输入	复位信号，1有效。
load	1位	输入	数据装载使能信号，用于控制输入明/密文、密钥、S盒配置数据等，1有效。
address	8位	输入	寄存器或RAM地址，用于表示明文/密文/密钥寄存器、S盒RAM单元。
keyexpen	1位	输入	密钥扩展使能信号，1有效。
keyexprdy	1位	输出	密钥扩展完成标识信号，1有效。
staenc	1位	输入	开始加密使能信号，1有效。
stadec	1位	输入	开始解密使能信号，1有效。
encdecrdy	1位	输出	加/解密运算完成标识信号，1有效。
din	16位	输入	输入数据总线，用于输入明/密文、密钥、配置数据等。
dout	16位	输出	输出数据总线，用于输出加/解密结果。
shift	1位	输入	结果移位输出使能信号，1有效。有效时，每个时钟周期移位输出16位结果。

3.2 AES密码处理器总体结构

➢AES加密芯片包括明/密文和密钥寄存器、密钥扩展、加/解密运算、控制供4个子模块，其总体结构如下图所示。



3.3 明文/密文/密钥寄存器设计

➢明文/密文/密钥寄存器的外部信号如下表所示。

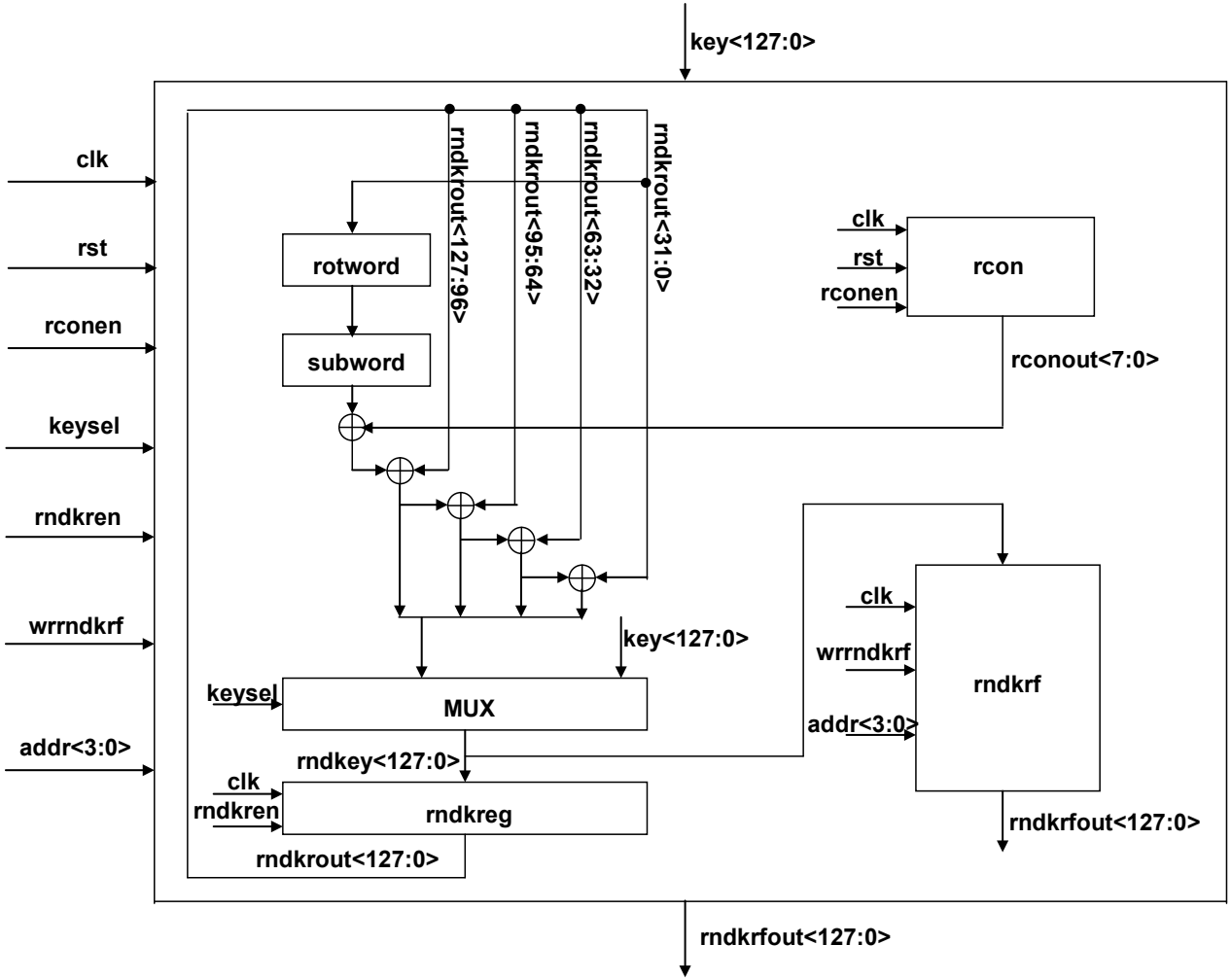
信号名称	信号宽度	传输方向	信号含义
clk	1位	输入	时钟信号。
write	1位	输入	寄存器写使能信号，1有效。
din	16位	输入	输入数据总线，用于输入明/密文、密钥、配置数据等。
dout	128位	输出	输出数据总线，用于输出加/解密结果。

➢明文/密文/密钥寄存器是一个128位的寄存器，用于保存从外部输入的明文/密文/密钥。为了与USB接口芯片的数据总线宽度相匹配，其输入数据宽度定为16位，为了提高AES模块内部的处理速度，同时为了与AES分组长度相匹配，其输出数据的宽度定为128位。

➢其详细功能描述如下：在时钟信号clk的上升沿，若寄存器写使能信号write有效，则将输入数据总线din上的16位数据写入明文/密文/密钥寄存器的高16位，同时将寄存器原来的数据右移16位放入寄存器的低112位。若寄存器写使能信号write无效，则寄存器保持原来的数据不变。由此可见，1个128位的数据需要经过8个时钟周期，通过8次写操作才能装入寄存器。

3.4 密钥扩展模块设计

➢密钥扩展模块的电路结构如下图所示：



密钥扩展模块的外部信号如下表所示：

信号名称	信号宽度	传输方向	信号含义
clk	1位	输入	时钟信号。
rst	1位	输入	复位信号，1有效。有效时使轮常数产生模块输出第一个轮常数。
keysel	1位	输入	轮密钥选择信号，keysel=0选择密钥寄存器中的种子密钥，否则，选择经过密钥扩展变换的轮密钥。
rndkren	1位	输入	轮密钥寄存器写使能信号，1有效。
wrrndkrf	1位	输入	轮密钥寄存器堆写使能信号，1有效。
addr	4位	输入	轮密钥寄存器堆地址，表示11个轮密钥寄存器的地址。
rconen	1位	输入	轮常数寄存器写使能信号，1有效。当有效时将下一个轮常数写入轮常数寄存器。
key	128位	输入	种子密钥，来自于密钥寄存器的输出。
rndkrfout	128位	输出	轮密钥寄存器堆的输出，为加解密模块提供需要的轮密钥。

- 密钥扩展模块由1个128位的轮密钥寄存器rndkreg、1个11*128的轮密钥寄存器堆rndkrf、1个轮常数产生模块rcon、1个字节代替模块subword、1个32位的循环左移移位器rotword、1个128位的二选一选通器MUX以及5个32位异或器构成。
- 其电路工作原理如下：
- 在进行密钥扩展之前，首先令复位信号rst=1,以便使轮常数产生模块输出第1个轮常数。
- 然后，在第1个时钟周期，通过选通器选择种子密钥key作为第1个子密钥，并在本周期结束时的时钟上升沿，将其同时保存到轮密钥寄存器rndkreg和轮密钥寄存器堆rndkrf。
- 在第2个时钟周期，对保存在轮密钥寄存器rndkreg中的第1个子密钥先后进行循环左移、字节代替变换、异或、选通操作得到第2个子密钥，并在本周期结束时的时钟上升沿，将其同时保存到轮密钥寄存器rndkreg和轮密钥寄存器堆rndkrf。
- 重复第2个时钟周期的操作10次，就可以得到AES第1-10轮迭代所需要的轮密钥。
- 这样，经过11个时钟周期之后，AES加密/解密所需的全部11个子密钥就都产生出来了，并且被保存在轮密钥寄存器堆rndkrf中。

- 其中，循环左移运算实现对一个32位的数据循环左移8位，在电路实现时通过硬件连线即可实现，无须设计专门的移位电路。
- 字节代替变换由4个8*8 S盒实现，S盒采用查表方式实现。为了充分利用cyclone FPGA中的RAM资源，减少LE（逻辑单元）的使用数量，我们利用cyclone FPGA中的RAM构建了一个8*8的ROM作为S盒，该ROM共有256个的存储单元，每个存储单元存储一个字节，这256个字节就是实现S盒变换的查找表。该ROM可以利用quartus II的megafunction工具自动生成，但需要注意的是该ROM的地址要经寄存器锁存以后才能进入ROM，因此在地址有效的下一个周期才能在ROM的输出端口得到读出的数据。
- 二选一选通器MUX用于选择外部输入的种子密钥或者内部逻辑产生的数据作为子密钥。
- 轮密钥寄存器rndkreg是1个128位的寄存器，用于暂时保存当前产生的子密钥，以便产生下一个子密钥时使用。
- 轮密钥寄存器堆rndkrf共有11个128位的存储单元，用于保存密钥扩展以后的全部11个子密钥，供AES加密/解密使用。
- 轮常数产生模块rcon由1个8位寄存器和轮常数产生逻辑构成，当复位时，寄存器输出第一个轮常数，当轮常数寄存器写使能信号rconen有效时，将由轮常数产生逻辑产生的下一个轮常数写入轮常数寄存器。

3.5 加密/解密模块设计

- AES加密算法和解密算法所使用的变换大多相同或相似，因此其电路结构也非常类似，有很多资源可以共享。为了减少电路规模，我们采用一套电路分时实现AES加密和解密。
- AES加密过程由一个初始密钥加(异或)变换和十个轮变换构成，其中除第10个轮变换外，每个轮变换都是一样的，都是由字节代替(S盒变换)、行移位、列混合、密钥加4个子变换组成，第10个轮变换由字节代替、行移位、密钥加3个子变换组成，不包括列混合变换。
- 为了进一步减少电路的规模，我们仅实现一个轮变换的电路，用循环迭代的方式实现十轮变换。
- 设 $a_{i,j}\ (0 \le i \le 3, 0 \le j \le 3)$ 表示每一轮变换的输入字节，

$$a_j = \begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} \quad (0 \le j \le 3)$$

表示由4个输入字节构成的一个32位字，它是输入状态矩阵中的一列，

$a=(a_0,a_1,a_2,a_3)$表示输入状态矩阵。

令$b_{i,j}\ (0\le i\le 3,0\le j\le 3)$ 表示字节代替变换(记为S)后的字节，

$c_{i,j}\ (0\le i\le 3,0\le j\le 3)$ 表示行移位变换后的字节，

$d_{i,j}\ (0\le i\le 3,0\le j\le 3)$ 表示列混合变换后的字节，

$e_{i,j}\ (0\le i\le 3,0\le j\le 3)$ 表示每一轮变换后的输出字节，

$k_{i,j}\ (0\le i\le 3,0\le j\le 3)$ 表示每一轮变换的密钥字节。

则根据AES加密算法的描述，对于第1-9轮变换，有下列式子成立：

$$b_{i,j}=s(a_{i,j})\qquad (0\le i\le 3,0\le j\le 3) \tag{1}$$

$$\begin{pmatrix}c_{0,j}\\c_{1,j}\\c_{2,j}\\c_{3,j}\end{pmatrix}=\begin{pmatrix}b_{0,j}\\b_{1,(j+1)\bmod 4}\\b_{2,(j+2)\bmod 4}\\b_{3,(j+3)\bmod 4}\end{pmatrix}\quad (0\le j\le 3) \tag{2}$$

$$\begin{pmatrix}d_{0,j}\\d_{1,j}\\d_{2,j}\\d_{3,j}\end{pmatrix}=\begin{pmatrix}02&03&01&01\\01&02&03&01\\01&01&02&03\\03&01&01&02\end{pmatrix}\begin{pmatrix}c_{0,j}\\c_{1,j}\\c_{2,j}\\c_{3,j}\end{pmatrix}\quad (0\le j\le 3) \tag{3}$$

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix} \oplus \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \quad (0 \le j \le 3) \qquad (4)$$

➢将（1）式代入（2）式，（2）式代入（3）式，（3）式代入（4）式，得

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s(a_{0,j}) \\ s(a_{1,(j+1) \bmod 4}) \\ s(a_{2,(j+2) \bmod 4}) \\ s(a_{3,(j+3) \bmod 4}) \end{pmatrix} \oplus \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix}$$

$$= \begin{pmatrix} 02 \bullet s(a_{0,j}) \oplus 03 \bullet s(a_{1,(j+1) \bmod 4}) \oplus s(a_{2,(j+2) \bmod 4}) \oplus s(a_{3,(j+3) \bmod 4}) \oplus k_{0,j} \\ s(a_{0,j}) \oplus 02 \bullet s(a_{1,(j+1) \bmod 4}) \oplus 03 \bullet s(a_{2,(j+2) \bmod 4}) \oplus s(a_{3,(j+3) \bmod 4}) \oplus k_{1,j} \\ s(a_{0,j}) \oplus s(a_{1,(j+1) \bmod 4}) \oplus 02 \bullet s(a_{2,(j+2) \bmod 4}) \oplus 03 \bullet s(a_{3,(j+3) \bmod 4}) \oplus k_{2,j} \\ 03 \bullet s(a_{0,j}) \oplus s(a_{1,(j+1) \bmod 4}) \oplus s(a_{2,(j+2) \bmod 4}) \oplus 02 \bullet s(a_{3,(j+3) \bmod 4}) \oplus k_{3,j} \end{pmatrix} \qquad (5)$$

➢在上式中，分别令j=0, 1, 2, 3，我们就得到了经过一轮变换后的所有输出字节。

➢对于初始密钥加变换（可以看成是第0轮变换），其输出字节与输入字节之间的函数关系为：

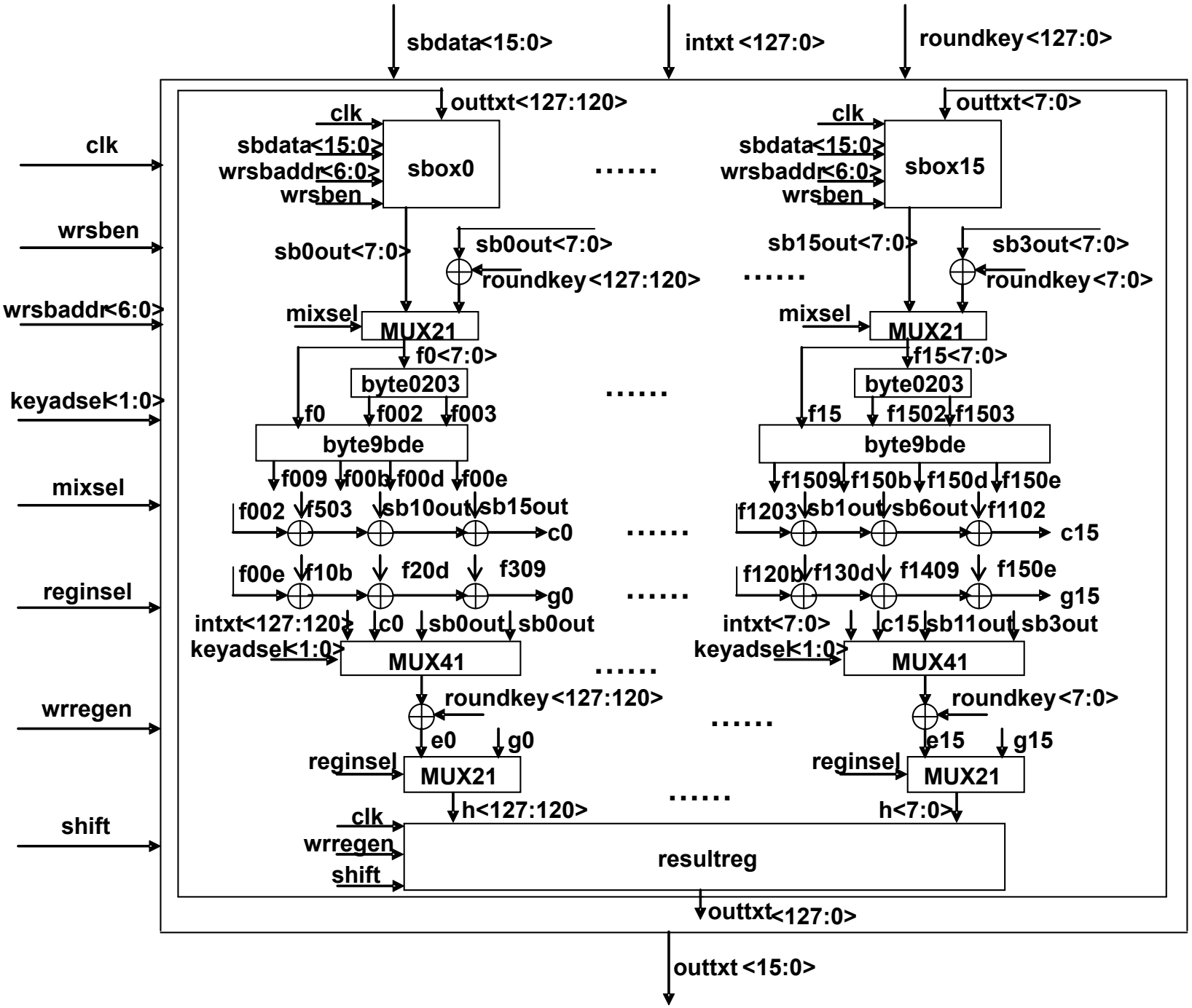
$$e_{i,j} = a_{i,j} \oplus k_{i,j} \qquad (0 \le i \le 3, 0 \le j \le 3) \qquad (6)$$

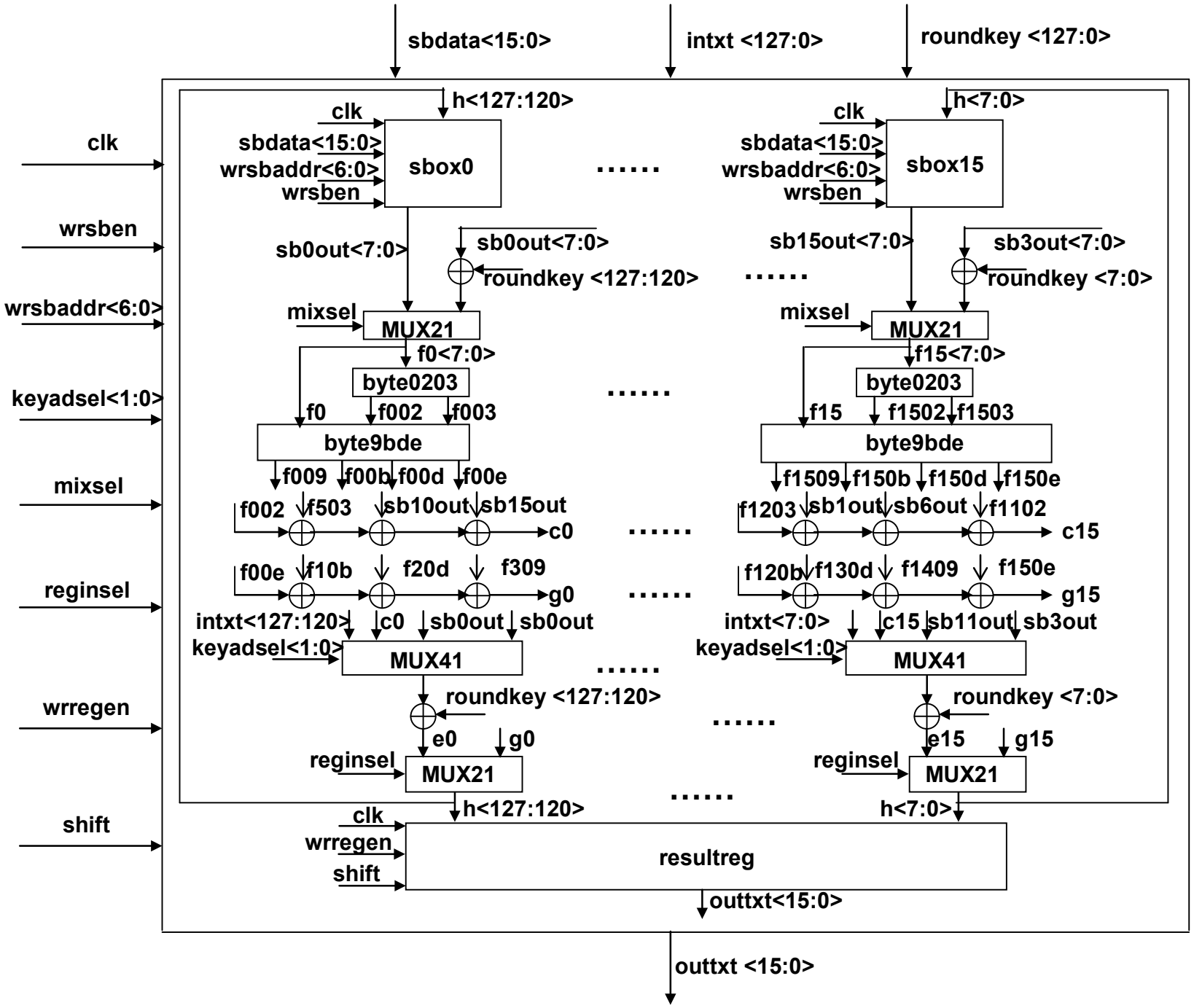
➢对于第10轮变换，其输出字节与输入字节之间的函数关系为：

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} s(a_{0,j}) \oplus k_{0,j} \\ s(a_{1,(j+1) \bmod 4}) \oplus k_{1,j} \\ s(a_{2,(j+2) \bmod 4}) \oplus k_{2,j} \\ s(a_{3,(j+3) \bmod 4}) \oplus k_{3,j} \end{pmatrix} \qquad (0 \le j \le 3) \qquad (7)$$

➢由（5）、（6）、（7）式可以看出，AES加密过程包括字节代替（S盒）、02乘字节、03乘字节、异或共4种操作，因此只要在电路中设置相应的电路模块就可以实现加密功能。为了充分利用FPGA中的RAM资源，减少LE的资源占用，我们采用查表方式实现S盒变换。同时，为了与AES加密算法自身的并行性相匹配，我们在电路中设置了16个8*8S盒，16个02乘字节、03乘字节模块。另外，为了保存每轮加密变换的结果，在电路中还应该设置一个128位的寄存器。

- 通过类似分析，可以得出AES解密过程包括逆S盒变换、09乘字节、0b乘字节、0d乘字节、0e乘字节、异或共6种操作。
- 其中，逆S盒变换可以使用与S盒变换相同的存储器电路，只是需要装入不同的初始值；
- 09乘字节、0b乘字节、0d乘字节、0e乘字节也可以在02乘字节、03乘字节的基础上实现。
- 另外，保存每轮解密变换结果的寄存器也与加密过程所使用的寄存器相同。
- 由此可见，只需在AES加密电路上增加少许电路，就可以实现AES解密功能。
- 综上所述，我们得到AES加密/解密模块的电路结构如下图所示：


sbdata<15:0>
intxt <127:0>
roundkey<127:0>
clk
wrsben
wrsbaddr<6:0>
keyadsel<1:0>
mixsel
reginsel
wrregen
shift
outtxt<127:120>
outtxt<7:0>
sbox0
sbox15
sb0out<7:0>
sb15out<7:0>
sb3out<7:0>
roundkey<127:120>
roundkey<7:0>
MUX21
f0<7:0>
f15<7:0>
byte0203
byte9bde
f0
f002
f003
f15
f1502
f1503
f009
f00b
f00d
f00e
f1509
f150b
f150d
f150e
f503
sb10out
sb15out
c0
f1203
sb1out
sb6out
f1102
c15
f10b
f20d
f309
g0
f120b
f130d
f1409
g15
intxt<127:120>
sb0out
intxt<7:0>
c15
sb11out
MUX41
e0
e15
h<127:120>
h<7:0>
resultreg
outtxt<127:0>
outtxt <15:0>


sbdata<15:0>
intxt <127:0>
roundkey <127:0>
clk
wrsben
wrsbaddr<6:0>
keyadsel<1:0>
mixsel
reginsel
wrregen
shift
h<127:120>
h<7:0>
sbox0
sbox15
sb0out<7:0>
sb15out<7:0>
sb3out<7:0>
roundkey <127:120>
roundkey <7:0>
MUX21
f0<7:0>
f15<7:0>
byte0203
byte9bde
f0
f002
f003
f15
f1502
f1503
f009
f00b
f00d
f00e
f1509
f150b
f150d
f150e
sb10out
sb15out
sb1out
sb6out
f1102
f1203
c0
c15
f10b
f20d
f309
f120b
f130d
f1409
g0
g15
intxt<127:120>
intxt<7:0>
sb0out
sb11out
MUX41
e0
e15
resultreg
outtxt<15:0>
outtxt <15:0>

➢AES加密/解密模块的外部信号如下表所示：

信号名称	信号宽度	传输方向	信号含义
clk	1位	输入	时钟信号。
wrsben	1位	输入	向S盒写入配置数据的使能信号，1有效。
wrsbaddr	7位	输入	S盒配置数据的地址，一个S盒包含256个字节，一次写入16位，因此共需要写128次，需要128个地址，所以需要7位地址码。
sbdata	16位	输入	S盒配置数据。
keyadsel	2位	输入	密钥加操作输入数据选择信号，
mixsel	1位	输入	选择是否进行逆列混合变换，在进行第1-9轮解密变换时为1，其余时间为0。
reginsel	1位	输入	结果寄存器输入数据选择信号。
wrregen	1位	输入	结果寄存器写使能信号，1有效。
intxt	128位	输入	外部输入数据（明文或密文）。
roundkey	128位	输入	子密钥。
shift	1位	输入	结果寄存器移位使能信号，1有效，有效时将128位结果寄存器中的数据右移16位。
outtxt	16位	输出	输出数据（加密/解密结果）。

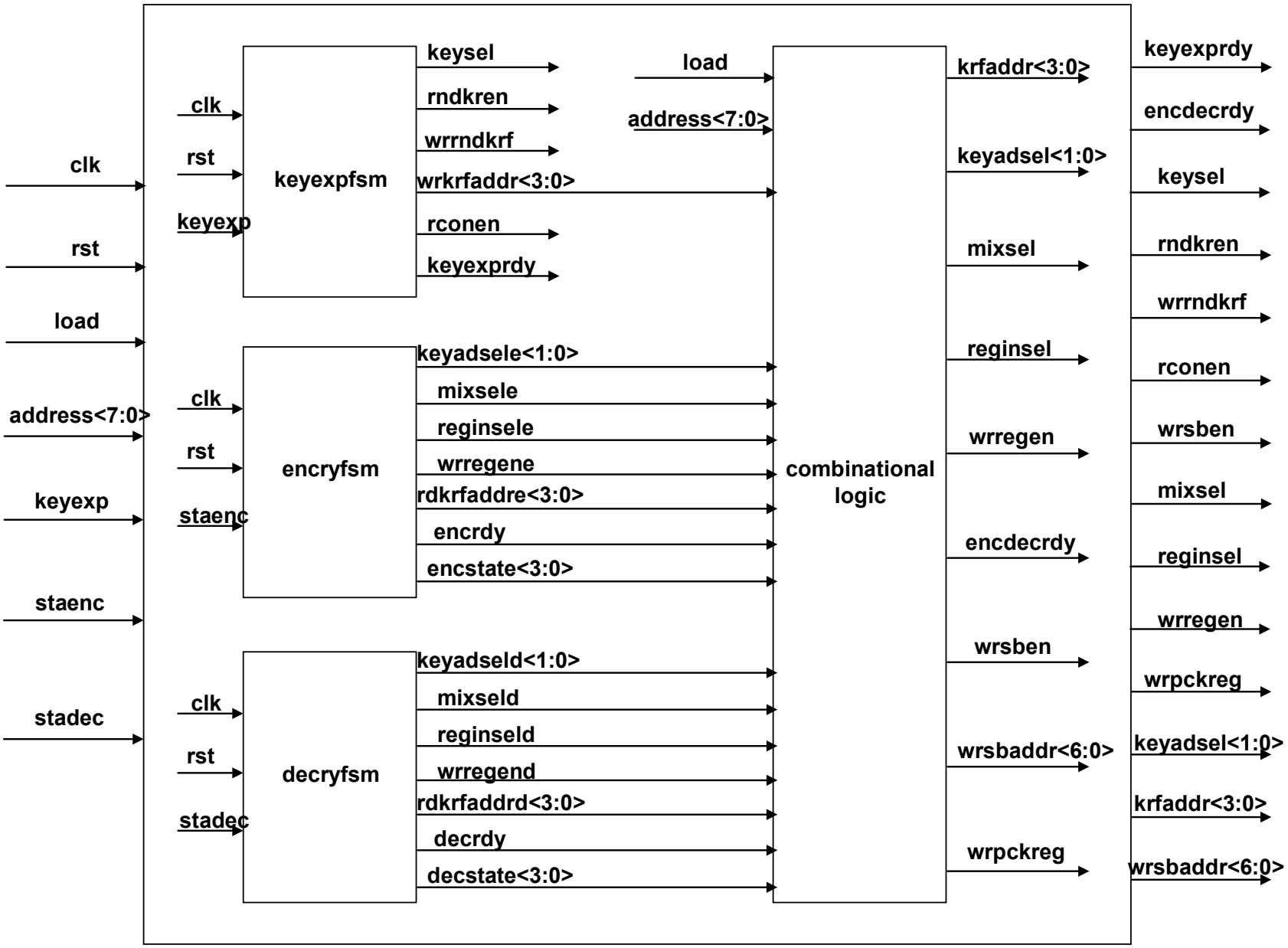
➢AES加密/解密模块的工作原理如下：

（1）加密流程：首先将S盒配置为加密S盒，即在使能信号wrsben和地址信号wrsbaddr的控制下，通过S盒配置数据端口sbdata将加密S盒配置数据写入16个S盒sbox0~sbox15。然后实现初始密钥加变换，即在选择信号keyadsel的控制下，通过四选一选通器选择外部输入明文数据intxt，与初始子密钥roundkey进行异或操作，并在选择信号reginsel的控制下，通过二选一选通器将异或操作的结果e0~e15保存到S盒的输入寄存器。接下来进行第一轮加密变换，即初始密钥加变换的结果经sbox0~sbox15完成S盒变换后，在选择信号mixsel的控制下，通过二选一选通器进入byte0203模块，完成02乘字节和03乘字节运算，然后进行（5）式中前4项的异或运算，得结果c0~c15，在选择信号keyadsel的控制下，通过四选一选通器选择c0~c15与第一轮子密钥进行异或操作，从而得到第一轮加密变换的结果e0~e15，并将其保存到S盒的输入寄存器，作为下一轮加密变换的输入数据。依次类推，可以完成第1-9轮加密变换。最后进行第10轮加密变换，即第9轮加密变换变换的结果经sbox0~sbox15完成S盒变换后，在选择信号keyadsel的控制下，通过四选一选通器选择恰当的S盒输出与第10轮子密钥进行异或操作，即可得到密文，最后将其保存到结果寄存器resultreg。初始密钥加变换和每轮加密变换都在一个周期内完成，因此上述加密过程共需要11个时钟周期。

➢（2）解密流程：首先将S盒配置为解密S盒，配置过程与加密S盒配置过程一样，只是配置数据不同。然后实现初始密钥加变换，即在选择信号keyadsel的控制下，通过四选一选通器选择外部输入密文数据intxt，与初始子密钥roundkey进行异或操作，并在选择信号reginsel的控制下，通过二选一选通器将异或操作的结果e0~e15保存到S盒的输入寄存器。接下来进行第一轮解密变换，即初始密钥加变换的结果经sbox0~sbox15完成逆S盒变换后，再与第一轮子密钥进行异或操作，然后在选择信号mixsel的控制下，通过二选一选通器进入byte0203模块和byte9bde模块，完成进行逆列混合变换所需要的字节乘法运算（即09乘字节、0b乘字节、0d乘字节和0e乘字节），然后通过一系列异或运算得列混合变换的结果g0~g15，在选择信号reginsel的控制下，通过二选一选通器选择g0~g15输出，从而得到第一轮解密变换的结果h，并将其保存到S盒的输入寄存器，作为下一轮解密变换的输入数据。依次类推，可以完成第1-9轮解密变换。最后进行第10轮解密变换，即第9轮加密变换变换的结果经sbox0~sbox15完成逆S盒变换后，在选择信号keyadsel的控制下，通过四选一选通器选择恰当的S盒输出与第10轮子密钥进行异或操作，即可得到明文，最后将其保存到结果寄存器resultreg。初始密钥加变换和每轮解密变换都在一个周期内完成，因此上述解密过程共需要11个时钟周期。需要注意的是，解密过程使用的子密钥与加密过程使用的子密钥相同，但使用顺序恰好相反。

3.6 控制模块设计

- AES控制模块由密钥扩展状态机、加密状态机、解密状态机以及其它少量组合逻辑构成。
- 其中密钥扩展状态机用于控制密钥扩展过程的执行，加密状态机用于控制加密过程的执行，解密状态机用于控制解密过程的执行。
- AES控制模块的电路结构图如下：


clk
rst
load
address<7:0>
keyexp
staenc
stadec
keyexpfsm
clk
rst
keyexp
keysel
rndkren
wrrndkrf
wrkrfaddr<3:0>
rconen
keyexprdy
encryfsm
clk
rst
staenc
keyadsele<1:0>
mixsele
reginsele
wrregene
rdkrfaddre<3:0>
encrdy
encstate<3:0>
decryfsm
clk
rst
stadec
keyadseld<1:0>
mixseld
reginseld
wrregend
rdkrfaddrd<3:0>
decrdy
decstate<3:0>
load
address<7:0>
combinational logic
krfaddr<3:0>
keyadsel<1:0>
mixsel
reginsel
wrregen
encdecrdy
wrsben
wrsbaddr<6:0>
wrpckreg
keyexprdy
encdecrdy
keysel
rndkren
wrrndkrf
rconen
wrsben
mixsel
reginsel
wrregen
wrpckreg
keyadsel<1:0>
krfaddr<3:0>
wrsbaddr<6:0>

➢AES控制模块的外部信号表：

信号名称	信号宽度	传输方向	信号含义
clk	1位	输入	时钟信号
rst	1位	输入	复位信号，1有效。
load	1位	输入	数据装载使能信号，用于控制输入明/密文、密钥、S盒配置数据等，1有效。
address	8位	输入	寄存器或RAM地址，用于表示明文/密文/密钥寄存器、S盒RAM单元。
keyexp	1位	输入	密钥扩展使能信号，1有效。
staenc	1位	输入	开始加密使能信号，1有效。
stadec	1位	输入	开始解密使能信号，1有效。
keyexprdy	1位	输出	密钥扩展完成标识信号，1有效。
encdecrdy	1位	输出	加/解密运算完成标识信号，1有效。
keysel	1位	输出	轮密钥选择信号，keysel=0选择密钥寄存器中的种子密钥，否则，选择经过密钥扩展变换的轮密钥。
rndkren	1位	输出	轮密钥寄存器写使能信号，1有效。

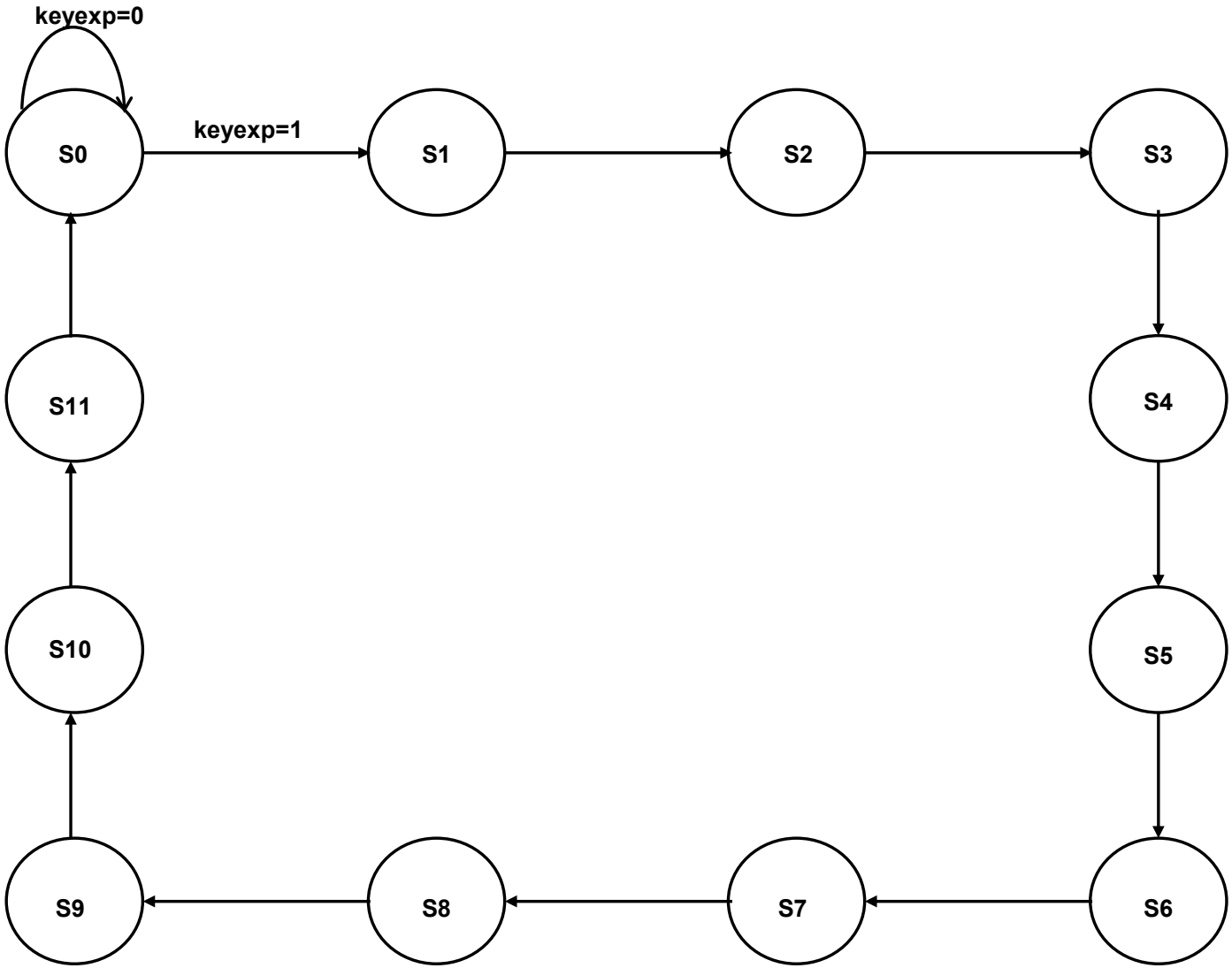
➢AES控制模块的外部信号表-续：

信号名称	信号宽度	传输方向	信号含义
wrrndkrf	1位	输出	轮密钥寄存器堆写使能信号，1有效。
rconen	1位	输出	轮常数寄存器写使能信号，1有效。当有效时将下一个轮常数写入轮常数寄存器。
wrsben	1位	输出	向S盒写入配置数据的使能信号，1有效。
mixsel	1位	输出	选择是否进行逆列混合变换，在进行第1-9轮解密变换时为1，其余时间为0。
reginsel	1位	输出	结果寄存器输入数据选择信号。
wrregen	1位	输出	结果寄存器写使能信号，1有效。
wrpckreg	1位	输出	明文/密文/密钥寄存器的写使能信号，1有效。
keyadsel	2位	输出	密钥加操作输入数据选择信号，2'b00选择外部输入数据，2'b 01选择列混合变换的结果，2'b10选择第10轮加密变换所需的S盒输出，2'b11选择第10轮解密变换所需的S盒输出。
krfaddr	4位	输出	轮密钥寄存器堆地址，表示11个轮密钥寄存器的地址。
wrsbaddr	7位	输出	S盒配置数据的地址。

➢密钥扩展状态机用于产生密钥扩展过程中所使用的控制信号，它由12个状态构成，其状态的划分和定义如下表所示：

状态名称	状态编码	状态定义
S0	0000	空闲状态，保持所有寄存器写使能信号无效。当rst=1时状态机处于空闲状态。
S1	0001	产生并保存初始密钥加变换所需要的子密钥。
S2	0010	产生并保存第1轮加密变换所需要的子密钥。
S3	0011	产生并保存第2轮加密变换所需要的子密钥。
S4	0100	产生并保存第3轮加密变换所需要的子密钥。
S5	0101	产生并保存第4轮加密变换所需要的子密钥。
S6	0110	产生并保存第5轮加密变换所需要的子密钥。
S7	0111	产生并保存第6轮加密变换所需要的子密钥。
S8	1000	产生并保存第7轮加密变换所需要的子密钥。
S9	1001	产生并保存第8轮加密变换所需要的子密钥。
S10	1010	产生并保存第9轮加密变换所需要的子密钥。
S11	1011	产生并保存第10轮加密变换所需要的子密钥。

➢AES密钥扩展状态机的状态转移图如下：



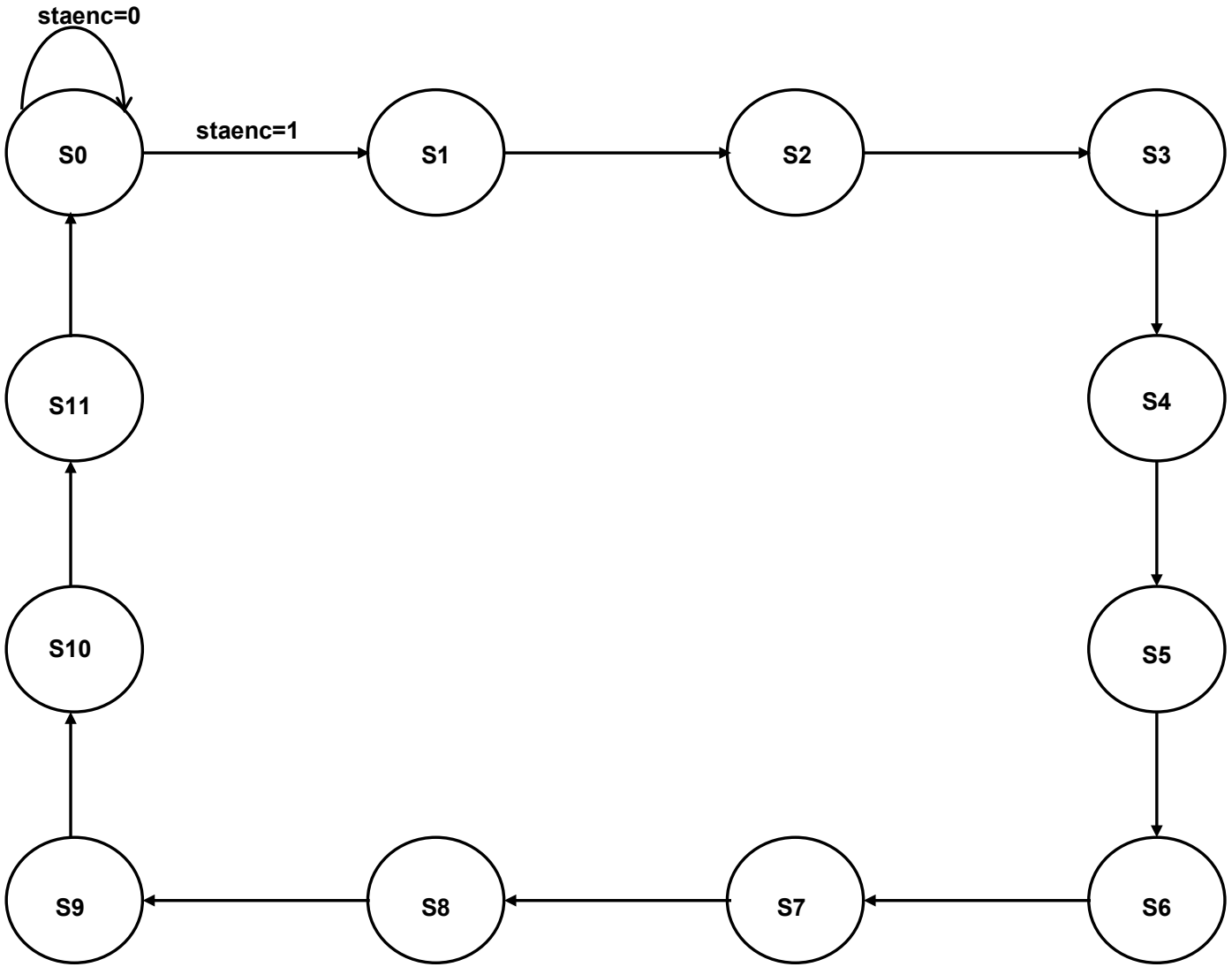
➢AES密钥扩展状态机的状态转移及控制信号取值表如下：

当前状态	当前输入	下一状态	当前输出
S0	keyexp=0	S0	keysel=0,rndkren=0,wrrndkrf=0,wrkrfaddr=4'd0, rconen=0, keyexprdy= (state_delay= =S11)。
S0	keyexp=1	S1	keysel=0,rndkren=0,wrrndkrf=0,wrkrfaddr=4'd0, rconen=0, keyexprdy= (state_delay= =S11)。
S1	x	S2	keysel=0,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd0, rconen=0, keyexprdy=0。
S2	x	S3	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd1, rconen=1, keyexprdy=0。
S3	x	S4	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd2, rconen=1, keyexprdy=0。
S4	x	S5	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd3, rconen=1, keyexprdy=0。
S5	x	S6	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd4, rconen=1, keyexprdy=0。
S6	x	S7	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd5, rconen=1, keyexprdy=0。
S7	x	S8	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd6, rconen=1, keyexprdy=0。
S8	x	S9	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd7, rconen=1, keyexprdy=0。
S9	x	S10	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd8, rconen=1, keyexprdy=0。
S10	x	S11	keysel=1,rndkren=1,wrrndkrf=1, wrkrfaddr=4'd9, rconen=1, keyexprdy=0。
S11	x	S0	keysel=1,rndkren=1,wrrndkrf=1,wrkrfaddr=4'd10, rconen=1, keyexprdy=0。

➢AES加密状态机用于产生加密过程中所使用的控制信号，它由12个状态构成，其状态的划分和定义如下表所示。

状态名称	状态编码	状态定义
S0	0000	空闲状态，保持所有寄存器写使能信号无效。当rst=1时状态机处于空闲状态。
S1	0001	进行初始密钥加变换。
S2	0010	进行第1轮加密变换。
S3	0011	进行第2轮加密变换。
S4	0100	进行第3轮加密变换。
S5	0101	进行第4轮加密变换。
S6	0110	进行第5轮加密变换。
S7	0111	进行第6轮加密变换。
S8	1000	进行第7轮加密变换。
S9	1001	进行第8轮加密变换。
S10	1010	进行第9轮加密变换。
S11	1011	进行第10轮加密变换，并保存密文。

➢AES加密状态机的状态转移图如下：



➢AES加密状态机的状态转移及控制信号取值表如下：

当前状态	当前输入	下一状态	当前输出
S0	**staenc=0**	**S0**	**wrregen=0, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd0, encrdy=(state_delay= =S11)。**
S0	**staenc=1**	**S1**	**wrregen=0, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd0, encrdy=(state_delay= =S11)。**
S1	**x**	**S2**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd0, encrdy=0。**
S2	**x**	**S3**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd1, encrdy=0。**
S3	**x**	**S4**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd2, encrdy=0。**
S4	**x**	**S5**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd3, encrdy=0。**
S5	**x**	**S6**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd4, encrdy=0。**
S6	**x**	**S7**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd5, encrdy=0。**
S7	**x**	**S8**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd6, encrdy=0。**
S8	**x**	**S9**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd7, encrdy=0。**
S9	**x**	**S10**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd8, encrdy=0。**
S10	**x**	**S11**	**wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b01, rdkrfaddr=4'd9, encrdy=0。**
S11	**x**	**S0**	**wrregen=1,mixsel=0,reginsel=0,keyadsel=2'b10, rdkrfaddr=4'd10, encrdy=0。**

➢AES解密状态机用于产生解密过程中所使用的控制信号，它由12个状态构成，其状态的划分和定义如下表所示。

状态名称	状态编码	状态定义
S0	0000	空闲状态，保持所有寄存器写使能信号无效。当rst=1时状态机处于空闲状态。
S1	0001	进行初始密钥加变换。使用第10轮加密变换的子密钥。
S2	0010	进行第1轮解密变换。使用第9轮加密变换的子密钥。
S3	0011	进行第2轮解密变换。使用第8轮加密变换的子密钥。
S4	0100	进行第3轮解密变换。使用第7轮加密变换的子密钥。
S5	0101	进行第4轮解密变换。使用第6轮加密变换的子密钥。
S6	0110	进行第5轮解密变换。使用第5轮加密变换的子密钥。
S7	0111	进行第6轮解密变换。使用第4轮加密变换的子密钥。
S8	1000	进行第7轮解密变换。使用第3轮加密变换的子密钥。
S9	1001	进行第8轮解密变换。使用第2轮加密变换的子密钥。
S10	1010	进行第9轮解密变换。使用第1轮加密变换的子密钥。
S11	1011	进行第10轮解密变换，并保存明文。使用加密过程的初始密钥加变换的子密钥。

➢AES解密状态机的状态转移图与加密状态机的状态转移图类似，其状态转移及控制信号取值如下表所示：

当前状态	当前输入	下一状态	当前输出
S0	stadec=0	S0	wrregen=0, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd0, decrdy=(state_delay= =S11)。
S0	stadec=1	S1	wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd0, decrdy=(state_delay= =S11)。
S1	x	S2	wrregen=1, mixsel=0, reginsel=0,keyadsel=2'b00, rdkrfaddr=4'd10, decrdy=0。
S2	x	S3	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd9, decrdy=0。
S3	x	S4	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd8, decrdy=0。
S4	x	S5	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd7, decrdy=0。
S5	x	S6	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd6, decrdy=0。
S6	x	S7	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd5, decrdy=0。
S7	x	S8	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd4, decrdy=0。
S8	x	S9	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd3, decrdy=0。
S9	x	S10	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd2, decrdy=0。
S10	x	S11	wrregen=1, mixsel=1, reginsel=1,keyadsel=2'b00, rdkrfaddr=4'd1, decrdy=0。
S11	x	S0	wrregen=1,mixsel=0,reginsel=0,keyadsel=2'b11, rdkrfaddr=4'd0, decrdy=0。

4. AES密码处理器的Verilog模型设计

```
module aes(clk,rst,load,address,keyexpen,staenc,stadec,din,keyexprdy,encdecrdy,dout);
  output [127:0] dout;
  output keyexprdy,encdecrdy;
  input  clk,rst,load,keyexpen,staenc,stadec;
  input  [4:0] address;
  input  [127:0] din;
  wire  wrpckreg,keysel,rndkren,wrrndkrf,rconen,wrsben,mixsel,reginsel,wrregen;
  wire [127:0] pckregout,roundkey;
  wire [3:0] krfaddr,wrsbaddr;
  wire [1:0] keyadsel;
  reg_128     pckreg(clk,wrpckreg,din,pckregout);
  aescontrol  control(clk,rst,load,address,keyexpen,staenc,stadec,keyexprdy,encdecrdy,
                 keysel,rndkren,wrrndkrf,krfaddr,rconen,
                 wrsben,wrsbaddr,keyadsel,mixsel,reginsel,wrregen,wrpckreg);
  keyexp  keyexp(clk,rst,keysel,rndkren,wrrndkrf,krfaddr,rconen,pckregout,roundkey);
  crydap  crydap(clk,wrsben,wrsbaddr,din,keyadsel,mixsel,reginsel,
                      wrregen,pckregout,roundkey,dout);
endmodule
```

```
module aes(clk,rst,load,address,keyexpen,staenc,stadec,din,keyexprdy,encdecrdy,dout,shift);

   output [15:0] dout;
   output keyexprdy,encdecrdy;
   input  clk,rst,load,keyexpen,staenc,stadec,shift;
   input  [7:0] address;
   input  [15:0] din;

   wire  wrpckreg,keysel,rndkren,wrrndkrf,rconen,wrsben,mixsel,reginsel,wrregen;
   wire [127:0] pckregout,roundkey;
   wire [3:0] krfaddr;
   wire [6:0] wrsbaddr;
   wire [1:0] keyadsel;

   inreg     pckreg(clk,wrpckreg,din,pckregout);
   aescontrol  control(clk,rst,load,address,keyexpen,staenc,stadec,keyexprdy,encdecrdy,
                          keysel,rndkren,wrrndkrf,krfaddr,rconen,
                          wrsben,wrsbaddr,keyadsel,mixsel,reginsel,wrregen,wrpckreg);
   keyexp  keyexp(clk,rst|keyexprdy,keysel,rndkren,wrrndkrf,krfaddr,rconen,pckregout,roundkey);
   crydap  crydap(clk,wrsben,wrsbaddr,din,keyadsel,mixsel,reginsel,wrregen,
                       pckregout,roundkey,dout,shift);

endmodule
```

```
module aescontrol(clk,rst,load,address,keyexp,staenc,stadec,keyexprdy,encdecrdy,
                  keysel,rndkren,wrrndkrf,krfaddr,rconen,
                  wrsben,wrsbaddr,keyadsel,mixsel,reginsel,wrregen,wrpckreg);
output keyexprdy,encdecrdy,keysel,rndkren,wrrndkrf,rconen;
output wrsben,mixsel,reginsel,wrregen,wrpckreg;
output [1:0] keyadsel;
output [3:0] krfaddr,wrsbaddr;
input clk,rst,load,keyexp,staenc,stadec;
input[4:0] address;
wire [3:0] wrkrfaddr,rdkrfaddre,rdkrfaddrd,encstate,decstate;
wire [1:0] keyadsele,keyadseld;
wire mixsele,reginsele,wrregene,encrdy,mixseld,reginseld,wrregend,decrdy;
assign krfaddr=(encstate != 4'd0)? rdkrfaddre:((decstate != 4'd0)?rdkrfaddrd:wrkrfaddr);
assign keyadsel=(encstate != 4'd0)? keyadsele:keyadseld;
assign mixsel=(encstate != 4'd0)? mixsele:mixseld;
assign reginsel=(encstate != 4'd0)? reginsele:reginseld;
assign wrregen=(encstate != 4'd0)? wrregene:wrregend;
assign encdecrdy=encrdy & decrdy;
keyexpfsm  keyexpfsm(clk,rst,keyexp,keysel,rndkren,wrrndkrf,wrkrfaddr,rconen,keyexprdy);
encryfsm   encryfsm(clk,rst,staenc,keyadsele,mixsele,reginsele,wrregene,rdkrfaddre,encrdy,encstate);
decryfsm   decryfsm(clk,rst,stadec,keyadseld,mixseld,reginseld,wrregend,rdkrfaddrd,decrdy,decstate);
assign wrsben=load & ~address[4];
assign wrsbaddr=address[3:0];
assign wrpckreg=load & address[4] & ~address[3] & ~address[2] & ~address[1] & ~address[0];
endmodule
```

```
module
keyexpfsm(clk,rst,keyexp,keysel,rndkren,wrrndkrf,wrkrfaddr,rconen,keyexprdy);
output keysel,rndkren,wrrndkrf,rconen,keyexprdy;
output [3:0] wrkrfaddr;
input clk,rst,keyexp;
reg [3:0] state,next_state,wrkrfaddr;
reg keysel,rndkren,keyexprdy;
always @(posedge clk)
        begin
                if(rst)
                        state<=4'd0;
                else
                        state<=next_state;

        end
```

```
always @ (state or keyexp)
                    case(state)
                              4'd0:     if(keyexp == 1)
                                                next_state = 4'd1;
                                        else
                                                 next_state = 4'd0;
                              4'd1: next_state = 4'd2;
                              4'd2: next_state = 4'd3;
                              4'd3: next_state = 4'd4;
                              4'd4: next_state = 4'd5;
                              4'd5: next_state = 4'd6;

                              4'd6: next_state = 4'd7;
                              4'd7: next_state = 4'd8;
                              4'd8: next_state = 4'd9;
                              4'd9: next_state = 4'd10;
                              4'd10: next_state = 4'd11;
                              4'd11: next_state = 4'd0;
                              default: next_state = 4'd0;
                    endcase
```

```
always @ (state)
                case(state)
                        4'd0:       keysel=0;
                        4'd1:       keysel=0;
                        4'd2:       keysel=1;
                        4'd3:       keysel=1;
                        4'd4:       keysel=1;
                        4'd5:       keysel=1;
                        4'd6:       keysel=1;
                        4'd7:       keysel=1;
                        4'd8:       keysel=1;
                        4'd9:       keysel=1;
                        4'd10:      keysel=1;
                        4'd11:      keysel=1;

                        default: keysel=0;
                endcase
```

```
always @ (state)
                    case(state)
                                4'd0:     rndkren=0;
                                4'd1:     rndkren=1;
                                4'd2:     rndkren=1;
                                4'd3:     rndkren=1;
                                4'd4:     rndkren=1;
                                4'd5:     rndkren=1;
                                4'd6:     rndkren=1;
                                4'd7:     rndkren=1;
                                4'd8:     rndkren=1;
                                4'd9:     rndkren=1;
                                4'd10:    rndkren=1;
                                4'd11:    rndkren=1;

                                default: rndkren=0;
                    endcase
assign wrrndkrf=rndkren;
```

```
always @ (state)
                case(state)
                        4'd0:       wrkrfaddr=4'd0;
                        4'd1:       wrkrfaddr=4'd0;
                        4'd2:       wrkrfaddr=4'd1;
                        4'd3:       wrkrfaddr=4'd2;
                        4'd4:       wrkrfaddr=4'd3;
                        4'd5:       wrkrfaddr=4'd4;
                        4'd6:       wrkrfaddr=4'd5;
                        4'd7:       wrkrfaddr=4'd6;
                        4'd8:       wrkrfaddr=4'd7;
                        4'd9:       wrkrfaddr=4'd8;
                        4'd10:      wrkrfaddr=4'd9;
                        4'd11:      wrkrfaddr=4'd10;

                        default: wrkrfaddr=4'd0;
                endcase
assign rconen=keysel;
always @ (state)
                case(state)
                        4'd0:       keyexprdy=1;

                        default:    keyexprdy=0;
                endcase
endmodule
```

```
module encryfsm(clk,rst,staenc,keyadsel,mixsel,reginsel,wrregen,rdkrfaddr,encrdy,state);
output wrregen,mixsel,reginsel,encrdy,state;
output [1:0] keyadsel;
output [3:0] rdkrfaddr;
input clk,rst,staenc;
reg [3:0] state,next_state,rdkrfaddr;
reg wrregen,encrdy;
reg [1:0] keyadsel;
always @(posedge clk)
        begin
                if(rst)
                        state<=4'd0;
                else
                        state<=next_state;

        end
```

```
always @ (state or staenc)
                case(state)
                        4'd0:     if(staenc == 1)
                                        next_state = 4'd1;
                                  else
                                        next_state = 4'd0;
                        4'd1: next_state = 4'd2;
                        4'd2: next_state = 4'd3;
                        4'd3: next_state = 4'd4;
                        4'd4: next_state = 4'd5;
                        4'd5: next_state = 4'd6;
                        4'd6: next_state = 4'd7;
                        4'd7: next_state = 4'd8;
                        4'd8: next_state = 4'd9;
                        4'd9: next_state = 4'd10;
                        4'd10: next_state = 4'd11;
                        4'd11: next_state = 4'd0;
                        default: next_state = 4'd0;
                endcase
```

```
always @ (state)
                    case(state)
                                4'd0:     wrregen=0;
                                4'd1:     wrregen=1;
                                4'd2:     wrregen=1;
                                4'd3:     wrregen=1;
                                4'd4:     wrregen=1;
                                4'd5:     wrregen=1;
                                4'd6:     wrregen=1;
                                4'd7:     wrregen=1;
                                4'd8:     wrregen=1;
                                4'd9:     wrregen=1;
                                4'd10:    wrregen=1;
                                4'd11:    wrregen=1;

                                default: wrregen=0;
                    endcase
assign mixsel=0;
assign reginsel=0;
```

```
always @ (state)
                case(state)
                        4'd0:     keyadsel=2'b00;
                        4'd1:     keyadsel=2'b00;
                        4'd2:     keyadsel=2'b01;
                        4'd3:     keyadsel=2'b01;
                        4'd4:     keyadsel=2'b01;
                        4'd5:     keyadsel=2'b01;
                        4'd6:     keyadsel=2'b01;
                        4'd7:     keyadsel=2'b01;
                        4'd8:     keyadsel=2'b01;
                        4'd9:     keyadsel=2'b01;
                        4'd10:    keyadsel=2'b01;
                        4'd11:    keyadsel=2'b10;

                        default: keyadsel=2'b00;
                endcase
```

```
always @ (state)
                case(state)
                        4'd0:       rdkrfaddr=4'd0;
                        4'd1:       rdkrfaddr=4'd0;
                        4'd2:       rdkrfaddr=4'd1;
                        4'd3:       rdkrfaddr=4'd2;
                        4'd4:       rdkrfaddr=4'd3;
                        4'd5:       rdkrfaddr=4'd4;
                        4'd6:       rdkrfaddr=4'd5;
                        4'd7:       rdkrfaddr=4'd6;
                        4'd8:       rdkrfaddr=4'd7;
                        4'd9:       rdkrfaddr=4'd8;
                        4'd10:      rdkrfaddr=4'd9;
                        4'd11:      rdkrfaddr=4'd10;

                        default: rdkrfaddr=4'd0;
                endcase
always @ (state)
                case(state)
                        4'd0:      encrdy=1;

                        default: encrdy=0;
                endcase
endmodule
```

```
module decryfsm(clk,rst,stadec,keyadsel,mixsel,reginsel,wrregen,rdkrfaddr,decrdy,state);
output wrregen,mixsel,reginsel,decrdy,state;
output [1:0] keyadsel;
output [3:0] rdkrfaddr;
input clk,rst,stadec;
reg [3:0] state,next_state,rdkrfaddr;
reg wrregen,decrdy,reginsel;
reg [1:0] keyadsel;
always @(posedge clk)
        begin
                if(rst)
                        state<=4'd0;
                else
                        state<=next_state;

        end
```

```
always @ (state or stadec)
                    case(state)
                              4'd0:     if(stadec == 1)
                                        next_state = 4'd1;
                                                  else
                                                       next_state = 4'd0;
                              4'd1: next_state = 4'd2;
                              4'd2:     next_state = 4'd3;
                              4'd3: next_state = 4'd4;
                              4'd4: next_state = 4'd5;
                              4'd5:     next_state = 4'd6;

                              4'd6: next_state = 4'd7;
                              4'd7:     next_state = 4'd8;
                              4'd8: next_state = 4'd9;
                              4'd9:     next_state = 4'd10;
                              4'd10: next_state = 4'd11;
                              4'd11: next_state = 4'd0;
                              default: next_state = 4'd0;
                    endcase
```

```
always @ (state)
                case(state)
                        4'd0:     wrregen=0;
                        4'd1:     wrregen=1;
                        ……
                        4'd11:    wrregen=1;
                        default: wrregen=0;
                endcase
always @ (state)
                case(state)
                        4'd0:     reginsel=0;
                        4'd1:     reginsel=0;
                        4'd2:     reginsel=1;
                        ......
                        4'd10:    reginsel=1;
                        4'd11:    reginsel=0;
                        default:    reginsel=0;
                endcase
assign mixsel=reginsel;
```

```
always @ (state)
                case(state)
                        4'd0:       keyadsel=2'b00;
                        4'd1:       keyadsel=2'b00;
                        4'd2:       keyadsel=2'b11;
                        ……
                        4'd11:      keyadsel=2'b11;
                        default:    keyadsel=2'b00;
                endcase

always @ (state)
                case(state)
                        4'd0:       rdkrfaddr=4'd0;
                        4'd1:       rdkrfaddr=4'd10;
                        4'd2:       rdkrfaddr=4'd9;
                        ……
                        4'd10:      rdkrfaddr=4'd1;
                        4'd11:      rdkrfaddr=4'd0;
                        default:    rdkrfaddr=4'd0;
                endcase
always @ (state)
                case(state)
                        4'd0:      decrdy=1;
                        default: decrdy=0;
                endcase
endmodule
```

```
module reg_128(clk,write,din,dout);
output [127:0] dout;
input  clk,write;
input  [127:0] din;
reg [127:0] dout;
always @ (posedge clk)
        begin
                if(write)
                        dout<=din;
                else
                        dout<=dout;
        end
endmodule
```

```
module keyexp(clk,rst,keysel,rndkren,wrrndkrf,addr,rconen,key,rndkrfout);
output[127:0] rndkrfout;
input clk,rst,keysel,rndkren,wrrndkrf,rconen;
input[3:0] addr;
input[127:0] key;
wire [127:0] rndkey,rndkrout,rndkrfout;
wire [31:0] w4,w5,w6,w7,rotword,subword,xorrcon;
wire [7:0] rconout;
assign rndkey=(keysel==0) ? key:{w4,w5,w6,w7};
reg_128 rndkreg(clk,rndkren,rndkey,rndkrout);
rndkrf rndkrf(clk,wrrndkrf,addr,rndkey,rndkrfout);
assign rotword={rndkrout[23:0],rndkrout[31:24]};
sbox_mux sbox0(rotword[31:24],subword[31:24]);
sbox_mux sbox1(rotword[23:16],subword[23:16]);
sbox_mux sbox2(rotword[15:8],subword[15:8]);
sbox_mux sbox3(rotword[7:0],subword[7:0]);
rcon rcon(clk,rst,rconen,rconout);
assign xorrcon=subword^{rconout,24'h000000};
assign w4=xorrcon^rndkrout[127:96];
assign w5=w4^rndkrout[95:64];
assign w6=w5^rndkrout[63:32];
assign w7=w6^rndkrout[31:0];
endmodule
```

```
module sbox_mux(in,out);
output[7:0] out;
input[7:0] in;
reg [7:0] out;
always@(in)
      case(in)
          8'h00: out=8'h63;
          8'h01: out=8'h7c;
          8'h02: out=8'h77;
          8'h03: out=8'h7b;
          ……
          8'hfa: out=8'h2d;
          8'hfb: out=8'h0f;
          8'hfc: out=8'hb0;
          8'hfd: out=8'h54;
          8'hfe: out=8'hbb;
          8'hff: out=8'h16;
      endcase

endmodule
```

```
module rndkrf(clk,wrrndkrf,addr,rndkey,rndkrfout);
input clk,wrrndkrf;
input [3:0] addr;
input [127:0] rndkey;
output [127:0] rndkrfout;
reg [10:0] decout;
wire [10:0] write_reg;
wire [127:0] reg0out,reg1out,reg2out,reg3out,reg4out,reg5out,reg6out,reg7out,reg8out,reg9out,reg10out;
reg [127:0] rndkrfout;
always @ (addr)
case(addr)
4'd0: decout=11'b000_0000_0001;
4'd1: decout=11'b000_0000_0010;
4'd2: decout=11'b000_0000_0100;
4'd3: decout=11'b000_0000_1000;
4'd4: decout=11'b000_0001_0000;
4'd5: decout=11'b000_0010_0000;
4'd6: decout=11'b000_0100_0000;
4'd7: decout=11'b000_1000_0000;
4'd8: decout=11'b001_0000_0000;
4'd9: decout=11'b010_0000_0000;
4'd10: decout=11'b100_0000_0000;
default: decout=11'b000_0000_0000;
endcase
```

```
assign write_reg=decout &
{wrrndkrf,wrrndkrf,wrrndkrf,wrrndkrf,wrrndkrf,wrrndkrf,wrrndkrf,
wrrndkrf,wrrndkrf,wrrndkrf,wrrndkrf};

reg_128 reg0(clk,write_reg[0],rndkey,reg0out);
reg_128 reg1(clk,write_reg[1],rndkey,reg1out);
reg_128 reg2(clk,write_reg[2],rndkey,reg2out);
reg_128 reg3(clk,write_reg[3],rndkey,reg3out);
reg_128 reg4(clk,write_reg[4],rndkey,reg4out);
reg_128 reg5(clk,write_reg[5],rndkey,reg5out);
reg_128 reg6(clk,write_reg[6],rndkey,reg6out);
reg_128 reg7(clk,write_reg[7],rndkey,reg7out);
reg_128 reg8(clk,write_reg[8],rndkey,reg8out);
reg_128 reg9(clk,write_reg[9],rndkey,reg9out);
reg_128 reg10(clk,write_reg[10],rndkey,reg10out);
```

```
always @(addr or reg0out or reg1out or reg2out or reg3out or reg4out
or reg5out or reg6out or reg7out or reg8out or reg9out or reg10out)
case(addr)
4'd0: rndkrfout=reg0out;
4'd1: rndkrfout=reg1out;
4'd2: rndkrfout=reg2out;
4'd3: rndkrfout=reg3out;
4'd4: rndkrfout=reg4out;
4'd5: rndkrfout=reg5out;
4'd6: rndkrfout=reg6out;
4'd7: rndkrfout=reg7out;
4'd8: rndkrfout=reg8out;
4'd9: rndkrfout=reg9out;
4'd10: rndkrfout=reg10out;
default: rndkrfout=reg10out;
endcase
endmodule
```

```
module rcon(clk,rst,write,rconout);

    output [7:0] rconout;
    input  clk,rst,write;
    reg [7:0] rconout;

    always @ (posedge clk)
        begin
            if(rst)
                rconout<=8'h01;
             else if(write)
                rconout<=(rconout[7]==0)? (rconout<<1):((rconout<<1)^{8'h1b});
            else
                rconout<=rconout;
        end

endmodule
```

```
module
crydap(clk,wrsben,wrsbaddr,sbdata,keyadsel,mixsel,reginsel,wrregen,intxt,roundkey,outtxt);
output [127:0] outtxt;
input clk,wrsben,wrregen,mixsel,reginsel;
input [1:0] keyadsel;
input [3:0] wrsbaddr;
input [127:0] sbdata,intxt,roundkey;
wire [7:0] sb0out,sb1out,sb2out,sb3out,sb4out,sb5out,sb6out,sb7out;
wire [7:0] sb8out,sb9out,sb10out,sb11out,sb12out,sb13out,sb14out,sb15out;
wire [7:0] a0,b0,c0,a1,b1,c1,a2,b2,c2,a3,b3,c3,a4,b4,c4,a5,b5,c5;
wire [7:0] a6,b6,c6,a7,b7,c7,a8,b8,c8,a9,b9,c9,a10,b10,c10,a11,b11,c11;
wire [7:0] a12,b12,c12,a13,b13,c13,a14,b14,c14,a15,b15,c15;
wire [7:0] d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15;
wire [7:0] e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15;
wire [7:0] f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15;
wire [7:0] g0,g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12,g13,g14,g15;
wire [7:0] i0,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15;
wire [7:0] j0,j1,j2,j3,j4,j5,j6,j7,j8,j9,j10,j11,j12,j13,j14,j15;
wire [7:0] f002,f003,f009,f00b,f00d,f00e;
wire [7:0] f102,f103,f109,f10b,f10d,f10e;
……
wire [7:0] f1502,f1503,f1509,f150b,f150d,f150e;
wire [127:0] d,e,g,h;
```

```
sbox  sbox0(clk,wrsben,wrsbaddr,sbdata,outtxt[127:120],sb0out);
sbox  sbox1(clk,wrsben,wrsbaddr,sbdata,outtxt[119:112],sb1out);
……
sbox  sbox15(clk,wrsben,wrsbaddr,sbdata,outtxt[7:0],sb15out);
mux21_8  mux21_8_0(mixsel,sb0out,e0,f0);
mux21_8  mux21_8_1(mixsel,sb1out,e1,f1);
……
mux21_8  mux21_8_15(mixsel,sb15out,e15,f15);
byte0203 byte0203_0(f0,f002,f003);
byte0203 byte0203_1(f1,f102,f103);
……
byte0203 byte0203_15(f15,f1502,f1503);
byte9bde byte9bde_0(f0,f002,f003,f009,f00b,f00d,f00e);
byte9bde byte9bde_1(f1,f102,f103,f109,f10b,f10d,f10e);
……
byte9bde byte9bde_15(f15,f1502,f1503,f1509,f150b,f150d,f150e);
assign a0=f002^f503;
assign b0=sb10out^sb15out;
assign c0=a0^b0;
mux41_8  mux41_8_0(keyadsel,intxt[127:120],c0,sb0out,sb0out,d0);
```

```
assign a1=sb0out^f502;
assign b1=f1003^sb15out;
assign c1=a1^b1;
mux41_8  mux41_8_1(keyadsel,intxt[119:112],c1,sb5out,sb13out,d1);

assign a2=sb0out^sb5out;
assign b2=f1002^f1503;
assign c2=a2^b2;
mux41_8  mux41_8_2(keyadsel,intxt[111:104],c2,sb10out,sb10out,d2);
……

assign a14=sb12out^sb1out;
assign b14=f602^f1103;
assign c14=a14^b14;
mux41_8  mux41_8_14(keyadsel,intxt[15:8],c14,sb6out,sb6out,d14);

assign a15=f1203^sb1out;
assign b15=sb6out^f1102;
assign c15=a15^b15;
mux41_8  mux41_8_15(keyadsel,intxt[7:0],c15,sb11out,sb3out,d15);
```

```
assign d={d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15};
assign e={e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15};
assign g={g0,g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12,g13,g14,g15};
assign {e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15}=d^roundkey;
assign i0=f00e^f10b;
assign j0=f20d^f309;
assign g0=i0^j0;
assign i1=f009^f10e;
assign j1=f20b^f30d;
assign g1=i1^j1;
…….
assign i14=f120d^f1309;
assign j14=f140e^f150b;
assign g14=i14^j14;
assign i15=f120b^f130d;
assign j15=f1409^f150e;
assign g15=i15^j15;
mux21_128  mux21_128_0(reginsel,e,g,h);
reg_128 resultreg(clk,wrregen,h,outtxt);
endmodule
```

```
module sbox(clk,write,wr_addr,din,rd_addr,dout);
input clk;
input write;
input [3:0] wr_addr;
input [127:0] din;
input [7:0] rd_addr;
output [7:0] dout;
reg [15:0] decout;
wire [15:0] write_reg;
wire [127:0] reg0out,reg1out,reg2out,reg3out,……,reg15out;
reg [7:0] dout;
always @ (wr_addr)
case(wr_addr)
4'd0: decout=16'b0000_0000_0000_0001;
4'd1: decout=16'b0000_0000_0000_0010;
4'd2: decout=16'b0000_0000_0000_0100;
4'd3: decout=16'b0000_0000_0000_1000;
……
4'd14: decout=16'b0100_0000_0000_0000;
4'd15: decout=16'b1000_0000_0000_0000;
endcase
```

```
assign write_reg=decout &
{write,write,write,write,write,write,write,write,write,write,write,write,write,write,write,wr
ite,write};

reg_128 reg0(clk,write_reg[0],din,reg0out);
reg_128 reg1(clk,write_reg[1],din,reg1out);
reg_128 reg2(clk,write_reg[2],din,reg2out);
reg_128 reg3(clk,write_reg[3],din,reg3out);
reg_128 reg4(clk,write_reg[4],din,reg4out);
reg_128 reg5(clk,write_reg[5],din,reg5out);
reg_128 reg6(clk,write_reg[6],din,reg6out);
reg_128 reg7(clk,write_reg[7],din,reg7out);
reg_128 reg8(clk,write_reg[8],din,reg8out);
reg_128 reg9(clk,write_reg[9],din,reg9out);
reg_128 reg10(clk,write_reg[10],din,reg10out);
reg_128 reg11(clk,write_reg[11],din,reg11out);
reg_128 reg12(clk,write_reg[12],din,reg12out);
reg_128 reg13(clk,write_reg[13],din,reg13out);
reg_128 reg14(clk,write_reg[14],din,reg14out);
reg_128 reg15(clk,write_reg[15],din,reg15out);
```

```
always @(rd_addr or reg0out or reg1out or reg2out or reg3out or reg4out or reg5out
or reg6out or reg7out or reg8out or reg9out or reg10out or reg11out or reg12out or
reg13out or reg14out or reg15out)
case(rd_addr)
8'd0: dout=reg0out[127:120];
8'd1: dout=reg0out[119:112];
8'd2: dout=reg0out[111:104];
8'd3: dout=reg0out[103:96];
8'd4: dout=reg0out[95:88];
8'd5: dout=reg0out[87:80];
8'd6: dout=reg0out[79:72];
8'd7: dout=reg0out[71:64];
…….
8'd251: dout=reg15out[39:32];
8'd252: dout=reg15out[31:24];
8'd253: dout=reg15out[23:16];
8'd254: dout=reg15out[15:8];
8'd255: dout=reg15out[7:0];
endcase
endmodule
```

```
module mux21_8(sel,a,b,c);
output[7:0] c;
input[7:0] a,b;
input sel;
reg [7:0] c;
always@(sel or a or b)
      case(sel)
          1'b0: c=a;
          1'b1: c=b;
      endcase
endmodule
module byte0203(a,a02,a03);
output[7:0] a02,a03;
input[7:0] a;
wire [7:0] b,c;
assign b={a[6:0],1'b0};
assign c=b^{8'h1b};
assign a02=(a[7]==0)? b:c;
assign a03=a02^a;
endmodule
```

```
module byte9bde(a,a02,a03,a09,a0b,a0d,a0e);
    output[7:0] a09,a0b,a0d,a0e;
    input[7:0] a,a02,a03;
    wire [7:0] a04,a08,b,c;
    byte02  byte02_0(a02,a04);
    byte02  byte02_1(a04,a08);
    assign a09=a08^a;
    assign a0b=a08^a03;
    assign b=a04^a;
    assign c=a04^a02;
    assign a0d=a08^b;
    assign a0e=a08^c;
endmodule
module byte02(a,a02);
    output[7:0] a02;
    input[7:0] a;
    wire [7:0] b,c;
    assign b={a[6:0],1'b0};
    assign c=b^{8'h1b};
    assign a02=(a[7]==0)? b:c;
endmodule
```

```
module mux41_8(sel,a,b,c,d,e);
    output[7:0] e;
    input[7:0] a,b,c,d;
    input [1:0] sel;
    reg [7:0] e;
    always@(sel or a or b or c or d)
       case(sel)
          2'b00: e=a;
          2'b01: e=b;
          2'b10: e=c;
          2'b11: e=d;
      endcase
endmodule
module mux21_128(sel,a,b,c);
    output[127:0] c;
    input[127:0] a,b;
    input sel;
    reg [127:0] c;
    always@(sel or a or b)
        case(sel)
           1'b0: c=a;
           1'b1: c=b;
        endcase
endmodule
```

5. AES密码处理器的功能仿真

```
`timescale 1ns / 1ns
module  byte0203_tb;
    wire [7:0] a02,a03;
    reg [7:0] a;
    reg clk;
    byte0203 byte0203(a,a02,a03);
    //clock generation
   initial clk = 1;
   always #50 clk = ~clk;

   initial
           begin
                #20    a=8'h98;
                #200  $finish;
           end
endmodule
```

```
`timescale 1ns / 1ns
module crydap_tb;
    wire [127:0] outtxt;
    reg clk,wrsben,wrregen,mixsel,reginsel;
    reg [1:0] keyadsel;
    reg [3:0] wrsbaddr;
    reg [127:0] sbdata,intxt,roundkey;
    crydap   crydap(clk,wrsben,wrsbaddr,sbdata,keyadsel,mixsel,
                        reginsel,wrregen,intxt,roundkey,outtxt);
    //clock generation
    initial clk = 1;
    always #50 clk = ~clk;
    initial
            begin
                        //decryption sbox configuration.
                        #20  wrsben=1;
                              wrsbaddr=4'd0;
                              sbdata=128'h52096ad53036a538bf40a39e81f3d7fb;
                        #100 wrsben=1;
                              wrsbaddr=4'd1;
                              sbdata=128'h7ce339829b2fff87348e4344c4dee9cb;
```

```
#100 wrsben=1;
        wrsbaddr=4'd2;
        sbdata=128'h547b9432a6c2233dee4c950b42fac34e;
#100 wrsben=1;
        wrsbaddr=4'd3;
        sbdata=128'h082ea16628d924b2765ba2496d8bd125;
.............

#100 wrsben=1;
        wrsbaddr=4'd13;
        sbdata=128'h60517fa919b54a0d2de57a9f93c99cef;
#100 wrsben=1;
        wrsbaddr=4'd14;
        sbdata=128'ha0e03b4dae2af5b0c8ebbb3c83539961;
#100 wrsben=1;
        wrsbaddr=4'd15;
        sbdata=128'h172b047eba77d626e169146355210c7d;
#100 wrsben=0;
```

```
//decryption.
    #100            wrregen=1;
                    keyadsel=2'b00;
                    mixsel=1;
                    reginsel=0;
                    intxt=128'h3925841d_02dc09fb_dc118597_196a0b32;
                    roundkey=128'hd014f9a8_c9ee2589_e13f0cc8_b6630ca6;

    #100            wrregen=1;
                    keyadsel=2'b11;
                    mixsel=1;
                    reginsel=1;
                    roundkey=128'hac7766f3_19fadc21_28d12941_575c006e;

    #100            wrregen=1;
                    keyadsel=2'b11;
                    mixsel=1;
                    reginsel=1;
                    roundkey=128'head27321_b58dbad2_312bf560_7f8d292f;
    ……
```

```
#100
                wrregen=1;
                keyadsel=2'b11;
                mixsel=1;
                reginsel=1;
                roundkey=128'hf2c295f2_7a96b943_5935807a_7359f67f;
 #100
                wrregen=1;
                keyadsel=2'b11;
                mixsel=1;
                reginsel=1;
                roundkey=128'ha0fafe17_88542cb1_23a33939_2a6c7605;
 #100
                wrregen=1;
                keyadsel=2'b11;
                mixsel=1;
                reginsel=0;
                roundkey=128'h2b7e1516_28aed2a6_abf71588_09cf4f3c;

 #100           wrregen=0;
```

```
//encryption sbox configuration.
                    #100 wrsben=1;
                        wrsbaddr=4'd0;
                        sbdata=128'h637c777bf26b6fc53001672bfed7ab76;
                    #100 wrsben=1;
                        wrsbaddr=4'd1;
                        sbdata=128'hca82c97dfa5947f0add4a2af9ca472c0;
                    #100 wrsben=1;
                        wrsbaddr=4'd2;
                        sbdata=128'hb7fd9326363ff7cc34a5e5f171d83115;
                    #100 wrsben=1;
                        wrsbaddr=4'd3;
                        sbdata=128'h04c723c31896059a071280e2eb27b275;
                    .................

                    #100 wrsben=1;
                        wrsbaddr=4'd14;
                        sbdata=128'he1f8981169d98e949b1e87e9ce5528df;
                    #100 wrsben=1;
                        wrsbaddr=4'd15;
                        sbdata=128'h8ca1890dbfe6426841992d0fb054bb16;
                    #100 wrsben=0;
```

```
//encryption.
    #100
                        wrregen=1;
                        keyadsel=2'b00;
                        mixsel=0;
                        reginsel=0;
                        intxt=128'h3243f6a8_885a308d_313198a2_e0370734;
                        roundkey=128'h2b7e1516_28aed2a6_abf71588_09cf4f3c;

    #100
                        wrregen=1;
                        keyadsel=2'b01;
                        mixsel=0;
                        reginsel=0;
                        roundkey=128'ha0fafe17_88542cb1_23a33939_2a6c7605;

    #100
                        wrregen=1;
                        keyadsel=2'b01;
                        mixsel=0;
                        reginsel=0;
                        roundkey=128'hf2c295f2_7a96b943_5935807a_7359f67f;
```

```
#100
                wrregen=1;
                keyadsel=2'b01;
                mixsel=0;
                reginsel=0;
                roundkey=128'h3d80477d_4716fe3e_1e237e44_6d7a883b;
 #100
                wrregen=1;
                keyadsel=2'b01;
                mixsel=0;
                reginsel=0;
                roundkey=128'hef44a541_a8525b7f_b671253b_db0bad00;
 #100
                wrregen=1;
                keyadsel=2'b01;
                mixsel=0;
                reginsel=0;
                roundkey=128'hd4d1c6f8_7c839d87_caf2b8bc_11f915bc;
```

```
#100
                wrregen=1;
                keyadsel=2'b01;
                roundkey=128'h6d88a37a_110b3efd_dbf98641_ca0093fd;
#100
                wrregen=1;
                keyadsel=2'b01;
                mixsel=0;
                reginsel=0;
                roundkey=128'h4e54f70e_5f5fc9f3_84a64fb2_4ea6dc4f;
#100
                wrregen=1;
                keyadsel=2'b01;
                mixsel=0;
                reginsel=0;
                roundkey=128'head27321_b58dbad2_312bf560_7f8d292f;
```

```
    #100
                    wrregen=1;
                    keyadsel=2'b01;
                    mixsel=0;
                    reginsel=0;
                    roundkey=128'hac7766f3_19fadc21_28d12941_575c006e;
     #100
                    wrregen=1;
                    keyadsel=2'b10;
                    mixsel=0;
                    reginsel=0;
                    roundkey=128'hd014f9a8_c9ee2589_e13f0cc8_b6630ca6;

     #100           wrregen=0;

     #200           $finish;
  end

endmodule
```

```
`timescale 1ns / 1ns
module aescontrol_tb;
wire  keyexprdy,encdecrdy,keysel,rndkren,wrrndkrf,rconen;
wire  wrsben,mixsel,reginsel,wrregen,wrpckreg;
wire  [1:0] keyadsel;
wire  [3:0] krfaddr,wrsbaddr;
reg   clk,rst,load,keyexp,staenc,stadec;
reg   [4:0] address;
aescontrol  aescontrol(clk,rst,load,address,keyexp,staenc,stadec,keyexprdy,encdecrdy,
                  keysel,rndkren,wrrndkrf,krfaddr,rconen,
                  wrsben,wrsbaddr,keyadsel,mixsel,reginsel,wrregen,wrpckreg);
//clock generation
initial clk = 1;
always #50 clk = ~clk;
initial
            begin
                        #20         rst=1;
                            load=0;
                            address=5'd0;
                            keyexp=0;
                            staenc=0;
                            stadec=0;
```

```
#200 rst=0;
        load=1;
        address=5'd0;
        keyexp=0;
        staenc=0;
        stadec=0;

    #100 rst=0;
        load=1;
        address=5'd1;
        keyexp=0;
        staenc=0;
        stadec=0;

    #100 rst=0;
        load=1;
        address=5'd2;
        keyexp=0;
        staenc=0;
        stadec=0;
```

```
#100 rst=0;
        load=1;
        address=5'd3;
        keyexp=0;
        staenc=0;
        stadec=0;

    #100 rst=0;
        load=1;
        address=5'd4;
        keyexp=0;
        staenc=0;
        stadec=0;

    #100 rst=0;
        load=1;
        address=5'd5;
        keyexp=0;
        staenc=0;
        stadec=0;
```

```
        #100  rst=0;
                load=1;
                address=5'd6;
                keyexp=0;
                staenc=0;
                stadec=0;

#100  rst=0;
                load=1;
                address=5'd7;
                keyexp=0;
                staenc=0;
                stadec=0;

#100  rst=0;
                load=1;
                address=5'd8;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
        #100  rst=0;
                load=1;
                address=5'd9;
                keyexp=0;
                staenc=0;
                stadec=0;

#100  rst=0;
                load=1;
                address=5'd10;
                keyexp=0;
                staenc=0;
                stadec=0;

#100  rst=0;
                load=1;
                address=5'd11;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd12;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd13;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd14;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd15;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd16;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd17;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd18;
                keyexp=0;
                staenc=0;
                stadec=0;

    #100 rst=0;
                load=1;
                address=5'd19;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd20;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                    load=1;
                    address=5'd21;
                    keyexp=0;
                    staenc=0;
                    stadec=0;

          #100  rst=0;
                    load=1;
                    address=5'd22;
                    keyexp=0;
                    staenc=0;
                    stadec=0;
    #100  rst=0;
                    load=1;
                    address=5'd23;
                    keyexp=0;
                    staenc=0;
                    stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd24;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd25;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd26;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd27;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd28;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd29;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=1;
                address=5'd30;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=1;
                address=5'd31;
                keyexp=0;
                staenc=0;
                stadec=0;

        #100  rst=0;
                load=0;
                address=5'd0;
                keyexp=1;
                staenc=0;
                stadec=0;
```

```
#100  rst=0;
                load=0;
                address=5'd1;
                keyexp=0;
                staenc=0;
                stadec=0;

        #1200 rst=0;
                load=0;
                address=5'd2;
                keyexp=0;
                staenc=1;
                stadec=0;

        #100  rst=0;
                load=0;
                address=5'd3;
                keyexp=0;
                staenc=0;
                stadec=0;
```

```
#1200 rst=0;
                        load=0;
                        address=5'd16;
                        keyexp=0;
                        staenc=0;
                        stadec=1;

            #100  rst=0;
                        load=0;
                        address=5'd17;
                        keyexp=0;
                        staenc=0;
                        stadec=0;
            #1200 $finish;

          end

endmodule
```

```
`timescale 1ns / 1ns
module aes_tb;
wire [127:0] dout;
wire keyexprdy,encdecrdy;
reg  clk,rst,load,keyexpen,staenc,stadec;
reg  [4:0] address;
reg  [127:0] din;
aes  aes(clk,rst,load,address,keyexpen,staenc,stadec,din,keyexprdy,encdecrdy,dout);
//clock generation
initial clk = 1;
always #50 clk = ~clk;
initial
        begin
                #20         rst=1;       //reset.

                #200 rst=0;       //load key.
                    load=1;
                    address=5'd16;
                    din=128'h2b7e1516_28aed2a6_abf71588_09cf4f3c;
                    keyexpen=0;
                    staenc=0;
                    stadec=0;
```

```
#100 rst=0;
        load=0;
        address=5'd0;
        keyexpen=1;   //key expansion.
        staenc=0;
        stadec=0;

    #100 rst=0;
        load=0;
        address=5'd0;
        keyexpen=0;
        staenc=0;
        stadec=0;

    #1000 rst=0;       //encryption sbox configuration.
        load=1;
        address=5'd0;
        din=128'h637c777bf26b6fc53001672bfed7ab76;
        keyexpen=0;
        staenc=0;
        stadec=0;
```

```
#100 address=5'd1;
    din=128'hca82c97dfa5947f0add4a2af9ca472c0;
#100 address=5'd2;
    din=128'hb7fd9326363ff7cc34a5e5f171d83115;
#100 address=5'd3;
    din=128'h04c723c31896059a071280e2eb27b275;
#100 address=5'd4;
    din=128'h09832c1a1b6e5aa0523bd6b329e32f84;
#100 address=5'd5;
    din=128'h53d100ed20fcb15b6acbbe394a4c58cf;
#100 address=5'd6;
    din=128'hd0efaafb434d338545f9027f503c9fa8;
#100 address=5'd7;
    din=128'h51a3408f929d38f5bcb6da2110fff3d2;
#100 address=5'd8;
    din=128'hcd0c13ec5f974417c4a77e3d645d1973;
…….
#100 address=5'd14;
    din=128'he1f8981169d98e949b1e87e9ce5528df;
#100 address=5'd15;
    din=128'h8ca1890dbfe6426841992d0fb054bb16;
```

```
#100 load=1;
        address=5'd16;
        din=128'h3243f6a8_885a308d_313198a2_e0370734;   // load plain text.

#100 load=0;
        staenc=1;   // start encryption.

#100 staenc=0;

#1200 load=1;   //decryption sbox configuration.
          address=5'd0;
          din=128'h52096ad53036a538bf40a39e81f3d7fb;
#100 address=5'd1;
        din=128'h7ce339829b2fff87348e4344c4dee9cb;
#100 address=5'd2;
        din=128'h547b9432a6c2233dee4c950b42fac34e;
…….
#100 address=5'd15;
        din=128'h172b047eba77d626e169146355210c7d;
```

```
                #100  load=1; // load cipher text.
                      address=5'd16;
                      din=128'h3925841d_02dc09fb_dc118597_196a0b32;

                #100 load=0;
                      stadec=1;   // start decryption.
                #100 stadec=0;

          #1500 $finish;

        end

endmodule
```