AES circuit 设计调研

Version	Update	Description
1.0	2024/06/07	Initial version is created

目录

1.研究背景：简要介绍 AES（Advanced Encryption Standard）算法的重要性及其在电路设计中的应用。

1.1 AES 简介及重要性

AES（Advanced Encryption Standard，高级加密标准）是一种对称密钥加密算法，于 2001 年由美国国家标准与技术研究院（NIST）发布，分组密码 Rijndael 算法取代了之前使用的 DES（Data Encryption Standard）算法，成为 AES 的标准。 AES 标准和 DES 标准主要区别就是密钥长度，DES 使用 56 位密钥，AES 支持 128、192、256 位密钥，更安全，AES 主要采用 128 位的长度。另外 AES 算法的软件实现更加高效。

值得注意的是，在当前量子计算发展迅速的情况下，AES 是一种对称加密算法，受到量子计算攻击 Grover 算法的威胁，Grover 算法可以将 AES 的密钥搜索速度提升平方根倍（对于密钥空间位 2^n 的密码执行一次完全的密钥搜索只需要 $2^{(n/2)}$步）。为应对量子计算的威胁，AES 需要加长密钥长度，例如从 128 位增加到 256 位，就可以保证在现有的量子计算算法下需要数十年才能破解。当然 NIST 也在尝试建立一种叫做后量子密码学（PQC）的标准，研究的是是抵御量子计算机攻击的新型公钥加密算法。具体可以参考 Post-Quantum Cryptography | CSRC (nist.gov)。

1.2 AES 在电路设计中的应用

在电路设计中，对于数据交换，通讯等领域既要保证数据的安全性，也要实现高速低延迟，通过硬件加速 AES 算法是一个比较好的选择。

2.模块设计：详细说明 AES 电路以下各个模块的功能，重点放在电路实现和门级电路数量上。（注：以下各模块的 RTL 实现参考了日本东北大学 Aoki 研的开源代码：Academic Publications (tohoku.ac.jp)）。

2.1 AES 原理及系统功能与设计

AES 为分组密码，即将待加密明文分为长度相等的组（AES 中分组只能为 128 位，即 16 字节），每次加密一组数据直至全部加密完成。加密密钥长度可以为 128 位、192 位、256 位，密钥长度不同加密轮数不同。之下只考虑 AES-128。

AES	密钥长度	分组长度	加密轮数
AES-128	4	4	10

对于 AES-128，密钥长度 128 位，加密轮数 10，加密和解密的整体流程，如图 1 所示，明文矩阵的加密过程如图 2 所示，密钥轮密钥加的过程如图 3 所示：

- 明文进行轮密钥加（密钥为 W[0,3]）
- 9 轮加密，每一轮包括：字节代换、行移位、列混合、轮密钥加（密钥依次为 W[4,7],⋯,W[36,39]）
- 第 10 轮加密，包括：字节代换、行移位、轮密钥加（密钥为 W[40,43]）

AES 解密流程就是以上的逆过程：

- 密文进行轮密钥加（密钥为 W[40,43]）

- 9 轮解密，每一轮包括：逆行移位、逆字节代换、轮密钥加（密钥依次为 W[36,39],…,W[4,7]）、逆列混合
- 第 10 轮解密，包括逆行移位、逆字节代换、 轮密钥加（密钥为 W[0,3]）

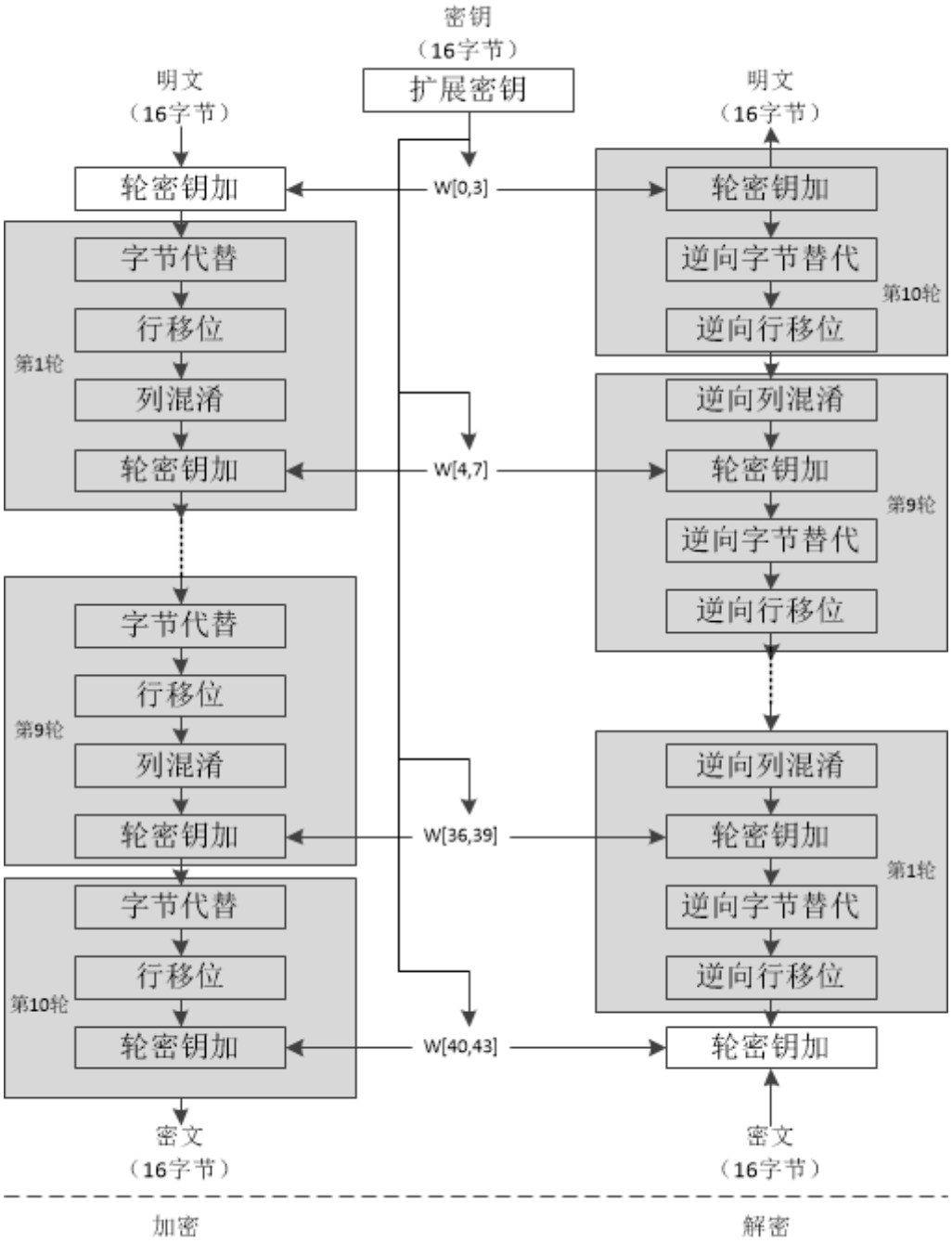


图 1 AES 的加密框架 [1]

基于上述原理，分别设计加密解密的各个模块 RTL，最后集成。

2. 2 SubBytes:

SubBytes 字节代换操作模块，其实就是一个根据 S-Box 查找表对每个字节进行替换，这是 AES 加密的非线性变换。AES 会先定义一个 S-Box，和逆 S-Box，分别是一个 16*16 的查找表。S-box 和逆 S-box 的构造涉及数论和有限域的概念。例如使用 GF(2^8)有限域上的乘法逆元。状态矩阵中的元素按照下面的方式映射为一个新的字节：把该字节的高 4 位作为行值，低 4 位作为列值，取出 S-box 和逆 S-box 中对应的行的元素作为输出。

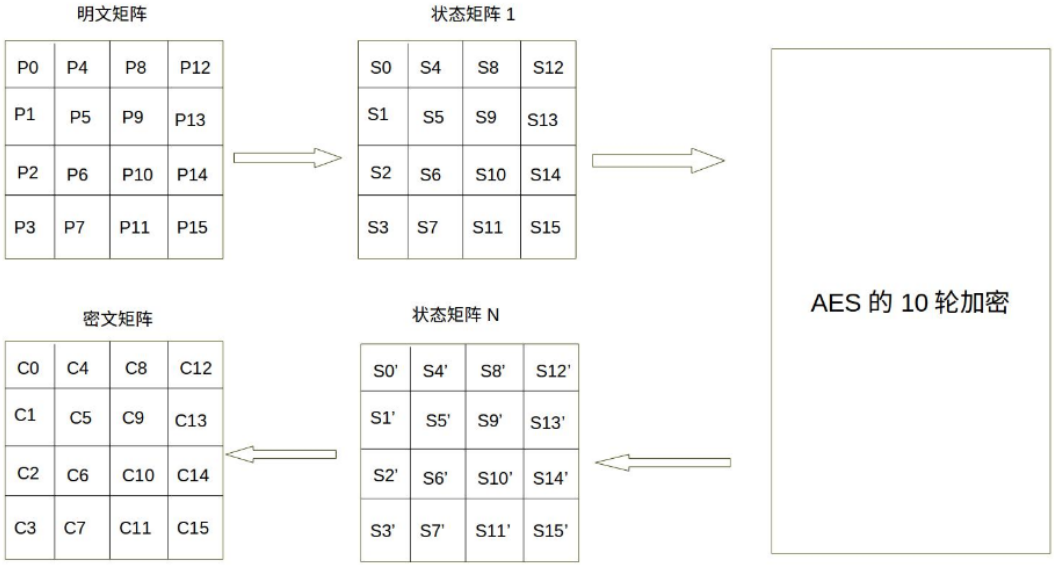


图 2 明文加密原理

硬件实现的话，通常会采用查找表来实现 S-Box，FPGA 也有大量的 LUT 资源。同时东北大学给出其他实现 S-Box 的方法，S-boxes are implemented based on Composite field, LUT, ANF (Arithmetic Normal Form), and 3-stage PPRM。对着这几种方法：

1. Composite Field

- **实现方式**：将 GF(2^8)上的运算分解为较小的 GF(2^4)上的运算，通过分层结构实现 S-Box。
- **门级电路数量**：优化后使用约 800-1200 个门。

2. LUT（查找表）

- **实现方式**：使用 ROM 查找表直接存储 256 个预计算值，输入字节作为地址进行查找。
- **门级电路数量**：需要 256x8 位的存储单元，大约需要 2000 个门。

3. ANF（Arithmetic Normal Form）

- **实现方式**：基于代数范式，通过逻辑电路实现 S-Box 的多项式表示。
- **门级电路数量**：约 1000-1500 个门，取决于优化程度。

4. 3-stage PPRM（3 级部分积减少法）

- **实现方式**：使用三阶段的方法，通过分步减少部分积来实现 S-Box。
- **门级电路数量**：约 700-900 个门。

比较一下：

- **Composite Field** 和 **3-stage PPRM** 在优化后门级电路数量最少。
- **LUT** 实现简单，但使用更多的存储单元。
- **ANF** 基于数学表示，门级电路数量中等，但实现复杂度较高。

基于 LUT 的 RTL 实现如下所示，通过 case 语句会综合成查找表。根据输入的明文来替换字节。

```
`timescale 1ns / 1ps

module SubBytes(x, y);
    input  [31:0] x;
    output [31:0] y;

    function [7:0] S;
    input    [7:0] x;
      case (x)
          0:S= 99;   1:S=124;   2:S=119;   3:S=123;
         ……
        252:S=176; 253:S= 84; 254:S=187; 255:S= 22;
      endcase
    endfunction

    assign y = {S(x[31:24]), S(x[23:16]), S(x[15: 8]), S(x[ 7: 0])};
endmodule
```

加密中过程中的第一步字节替换如下：

```
      SubBytes SB3 (di[127:96], sb[127:96]);
      SubBytes SB2 (di[ 95:64], sb[ 95:64]);
      SubBytes SB1 (di[ 63:32], sb[ 63:32]);
      SubBytes SB0 (di[ 31: 0], sb[ 31: 0]);
```

逆 S-box 如下：

```
module InvSubBytes(
    x, y
    );
    input  [31:0] x;
    output [31:0] y;

    function [7:0] S;
    input    [7:0] x;
      case (x)
          0:S= 82;   1:S=  9;   2:S=106;   3:S=213;
          ……
```

```
          252:S= 85; 253:S= 33; 254:S= 12; 255:S=125;
      endcase
    endfunction

    assign y = {S(x[31:24]), S(x[23:16]), S(x[15: 8]), S(x[ 7: 0])};
    //明文根据上述 S-box 置换字节
endmodule
```

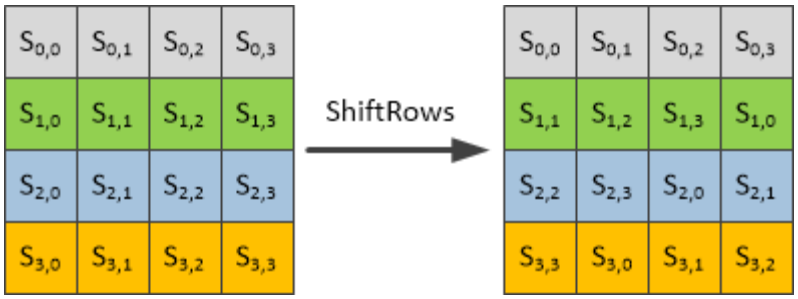
解密过程中的字节替换如下:

```
      InvSubBytes SB3 (sr[127:96], sb[127:96]);
      InvSubBytes SB2 (sr[ 95:64], sb[ 95:64]);
      InvSubBytes SB1 (sr[ 63:32], sb[ 63:32]);
      InvSubBytes SB0 (sr[ 31: 0], sb[ 31: 0]);
```

2.3 ShiftRows: 描述移位操作的硬件实现。

ShiftRows模块负责对状态矩阵的行进行循环移位操作,这是AES加密的扩散过程之一。通过线的重新排列实现,每行的字节按预定的偏移量移位。此模块的硬件实现相对简单,移位的 RTL 代码会被综合成多路选择器,对于 128 位的数据需要消耗 128 个 8 位的 MUX。其 RTL 如下:

加密过程中循环左移操作,状态矩阵的第 0 行左移 0 字节,第 1 行左移 1 字节,第 2 行左移 2 字节,第 3 行左移 3 字节,:




```
      assign sr = {sb[127:120], sb[ 87: 80], sb[ 47: 40], sb[  7:  0],
      //第一行保持不变
                   sb[ 95: 88], sb[ 55: 48], sb[ 15:  8], sb[103: 96],
      //第二行循化左移一位
                   sb[ 63: 56], sb[ 23: 16], sb[111:104], sb[ 71: 64],
      //第三行循环左移两位
                   sb[ 31: 24], sb[119:112], sb[ 79: 72], sb[ 39: 32]};
      //第四行循环左移三位
```

解密过程中相反的循环右移操作,状态矩阵的第 0 行右移 0 字节,第 1 行右移 1 字节,第 2 行右移 2 字节,第 3 行右移 3 字节:

```
    assign sr = {dx[127:120], dx[ 23: 16], dx[ 47: 40], dx[ 71: 64],
                 dx[ 95: 88], dx[119:112], dx[ 15:  8], dx[ 39: 32],
                 dx[ 63: 56], dx[ 87: 80], dx[111:104], dx[  7:  0],
                 dx[ 31: 24], dx[ 55: 48], dx[ 79: 72], dx[103: 96]};
```

2.4 MixColumns：讲解有限域上的矩阵乘法实现。

MixColumns 模块对每一列进行有限域上的矩阵乘法操作，进一步扩散字节间的关系。这个操作涉及 GF(2^8)上的多项式乘法和加法。根据矩阵的乘法可知，在列混淆的过程中，每个字节对应的值只与该列的 4 个值有关系。此处的乘法和加法都是定义在 GF(2^8)上的，需要注意如下几点：

1) 将某个字节所对应的值乘以 2，其结果就是将该值的二进制位左移一位，如果原始值的最高位为 1，则还需要将移位后的结果异或 00011011

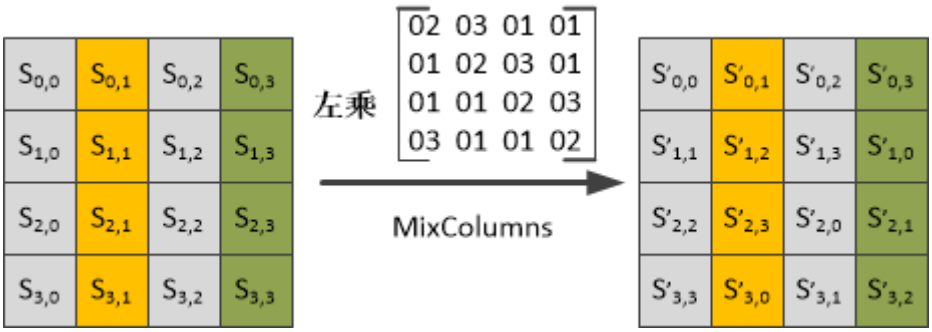
$$(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0) = \begin{cases} (a_6a_5a_4a_3a_2a_1a_00), a_7 = 0 \\ (a_6a_5a_4a_3a_2a_1a_00) \oplus (00011011), a_7 = 1 \end{cases}$$

2) 乘法对加法满足分配率，乘以(0000 0011)可以拆分成先分别乘以(0000 0001)和(0000 0010)，再将两个乘积异或：

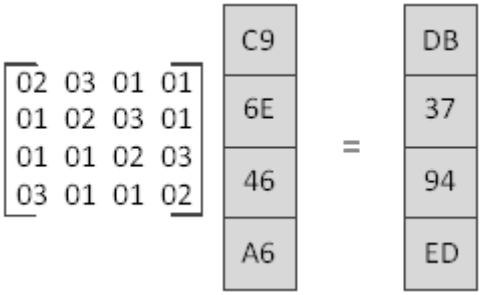
$$\begin{aligned}(00000011) * (a_7a_6a_5a_4a_3a_2a_1a_0) &= [(00000010) \oplus (00000001)] * (a_7a_6a_5a_4a_3a_2a_1a_0) \\ &= [(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0)] \oplus (a_7a_6a_5a_4a_3a_2a_1a_0)\end{aligned}$$

因此，我们只需要实现乘以 2 的函数，其他数值的乘法都可以通过组合来实现。

3) 各个值在相加时使用的是模 2^8 加法（异或运算）。



过程比较难以理解，这里举一个具体的例子：



$$S'_{0,0} = (02 \bullet C9) \oplus (03 \bullet 6E) \oplus (01 \bullet 46) \oplus (01 \bullet A6)$$

其中：

$$02 \bullet C9 = 02 \bullet 11001001_B = 10010010_B \oplus 00011011_B = 10001001_B$$

$$03 \bullet 6E = (01 \oplus 02) \bullet 6E = 01101110_B \oplus 11011100_B = 10110010_B$$

$$01 \bullet 46 = 01000110_B$$

$$01 \bullet A6 = 10100110_B$$

则：

$$S'_{0,0} = 10001001_B \oplus 10110010_B \oplus 01000110_B \oplus 10100110_B$$

$$= 11011011_B = DB$$

在计算 02 与 C9 的乘积时，由于 C9 对应最左边的比特为 1，因此需要将 C9 左移一位后的值与(0001 1011)求异或。同理可以求出另外几个值。

基于以上分析，完成 RTL 代码设计，一个比较容易想到的方法是先建立一个 MixColumns 计算矩阵的多维数组，然后在建立一个输入混淆矩阵的多维数字，然后进行上述有限域的乘法和加法。也是是考虑到这种方法消耗的资源比较多，这也采用了直接采用组合逻辑电路的方法。**输入分割**：将 32 位输入 x 分割成四个 8 位字节 a3, a2, a1, a0。**中间变量计算**：计算中间变量 b3, b2, b1, b0，它们是相邻字节的按位异或结果。**最终结果计算**：使用中间变量和按位操作来计算输出 y 的每个字节。

RTL 代码如下，通过这种方法会需要大量的异或 XOR 单元，大约几百个门级电路。

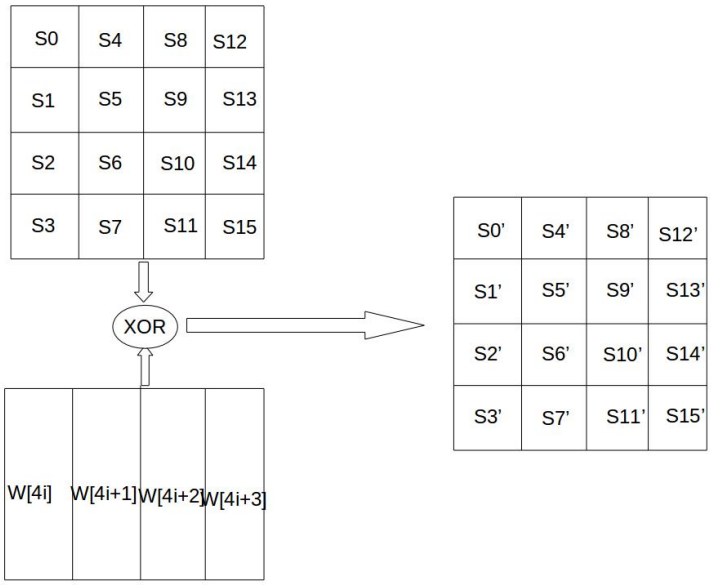
```
module MixColumns(
    x, y
    );
    input  [31:0]  x;
    output [31:0]  y;

    wire [7:0] a3, a2, a1, a0, b3, b2, b1, b0;

    assign a3 = x[31:24];
    assign a2 = x[23:16];
    assign a1 = x[15: 8];
    assign a0 = x[ 7: 0];
```

```
    assign b3 = a3 ^ a2;
    assign b2 = a2 ^ a1;
    assign b1 = a1 ^ a0;
    assign b0 = a0 ^ a3;

    assign y = {a2[7] ^ b1[7] ^ b3[6],
                ……
                a1[0] ^ b3[0] ^ b0[7]};
endmodule
```

逆列混淆就是做成一个逆矩阵来恢复原文：

2.5 AddRoundKey： 解释按位异或操作的实现。

AddRoundKey 轮密钥加相对简单，是将 128 位轮密钥 Ki 同状态矩阵中的数据进行逐位异或操作，如下图所示。



RLT 代码如下，需要注意的是要判断一下是否是第十轮加密，如果是就跳过；列混合，直接 AddRound，会综合按位 XOR 运算器。主要由 XOR 门组成，每个字节 8 个 XOR 门，共 128 个 XOR 门。

```
assign do = ((Rrg[0] == 1)? sr: mx) ^ ki;
```

解密的时候，由于异或的逆运算也是异或，所以代码相同

```
 assign do = sb ^ ki;
```

2.6 KeyExpansion： 描述密钥扩展的硬件实现。

KeyExpansion 模块负责从初始密钥生成所有轮密钥，密钥扩展过程说明：

将种子密钥按下图的格式排列，其中 k_0、k_1、……、k_{15}依次表示种子密钥的一个字节；排列后用 4 个 32 比特的字表示，分别记为 w[0]、w[1]、w[2]、w[3]；

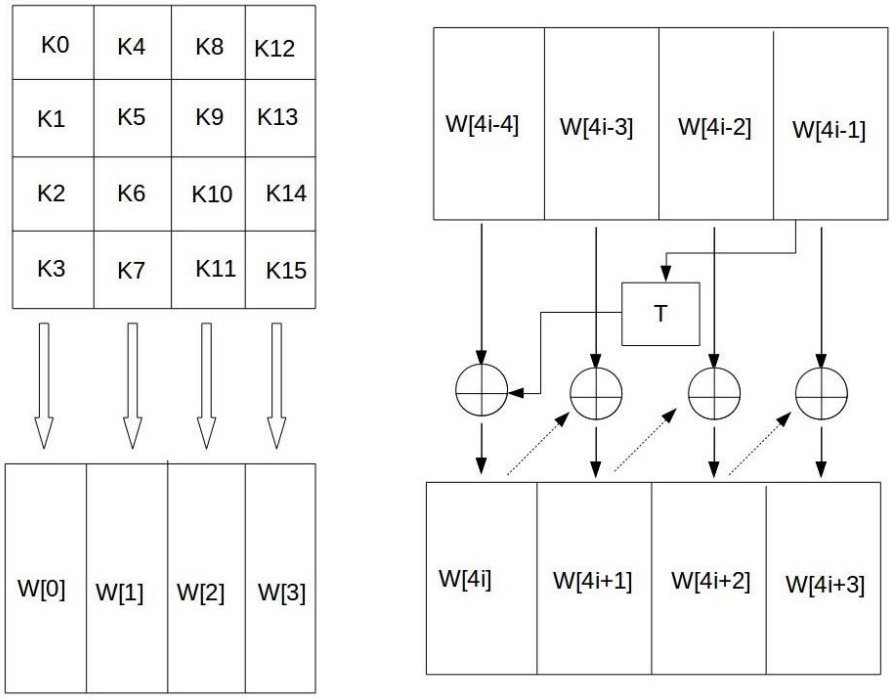
接着，对 W 数组扩充 40 个新列，构成总共 44 列的扩展密钥数组。新列以如下的递归方式产生：
1.如果 i 不是 4 的倍数，那么第 i 列由如下等式确定：
W[i]=W[i-4]⊕W[i-1]
2.如果 i 是 4 的倍数，那么第 i 列由如下等式确定：
W[i]=W[i-4]⊕T(W[i-1])

函数 T 由 3 部分组成：字循环、字节代换和轮常量异或，这 3 部分的作用分别如下。

a.字循环：将 1 个字中的 4 个字节循环左移 1 个字节。即将输入字[b0, b1, b2, b3]变换成[b1,b2,b3,b0]。

b.字节代换：对字循环的结果使用 S 盒进行字节代换。

c.轮常量异或：将前两步的结果同轮常量 Rcon[j]进行异或，其中 j 表示轮数。



RTL 代码如下，会综合成较多的 S-Box 查找和 XOR 操作，大约几百到一千个门级电路。

```
function [7:0] rcon;//生成密钥扩展中的 T 函数中的 rcon[j]
input [9:0] x;//x 对应当前加密轮数
  casex (x)
    10'bxxxxxxxxx1: rcon = 8'h01;
    10'bxxxxxxxx1x: rcon = 8'h02;
    10'bxxxxxxx1xx: rcon = 8'h04;
    10'bxxxxxx1xxx: rcon = 8'h08;
    10'bxxxxx1xxxx: rcon = 8'h10;
    10'bxxxx1xxxxx: rcon = 8'h20;
    10'bxxx1xxxxxx: rcon = 8'h40;
    10'bxx1xxxxxxx: rcon = 8'h80;
    10'bx1xxxxxxxx: rcon = 8'h1b;
    10'b1xxxxxxxxx: rcon = 8'h36;
  endcase
endfunction

SubBytes SBK ({ki[23:16], ki[15:8], ki[7:0], ki[31:24]}, so);//字循环后 S
盒字节代换，结果保存在 so 中
```

```
assign ko[127:96] = ki[127:96] ^ {so[31:24] ^ rcon(Rrg), so[23: 0]};//对于W[4i]的扩展
assign ko[ 95:64] = ki[ 95:64] ^ ko[127:96];//W[4i+1],W[4i+2],w[4i+3]
assign ko[ 63:32] = ki[ 63:32] ^ ko[ 95:64];
assign ko[ 31: 0] = ki[ 31: 0] ^ ko[ 63:32];
//以上过程生成下一轮加密的轮密钥
```

3.设计流程：简要概括从算法到硬件实现的设计流程，包括使用 Matlab 等工具仿真 AES 算法，验证正确性、用 Verilog/VHDL 描述各个模块以及系统验证方案等。

3.1 算法仿真：

使用 Matlab 等工具对 AES 算法进行仿真，验证算法的正确性。

个人对 Python 比较了解，仿真使用了 GitHub 中的开源项目：[mohib181/AES_Simulation: Implementation of AES in python (github.com)]

其加密模块入下

```python
def encrypt(block):
    # round 0
    state = []
    for i in range(len(block)):
        state.append([x ^ y for x, y in zip(w[i], block[i])])
    # print('round 0:', print_matrix(state))

    # round 1-10
    for r in range(1, total_rounds):

        # byte substitute
        for state_col in state:
            for j in range(len(state_col)):
                state_col[j] = byte_substitute(state_col[j])
        # print('byte sub:', print_matrix(state))

        # shift row
        state = left_shift(state)
        # print(row shift', print_matrix(state))

        # mix columns
        if r != total_rounds - 1:
```

```
            state = matrix_multiplication(Mixer, state)
            # print('mix column', print_matrix(state))

        # add round key
        for i in range(len(state)):
            state[i] = [x ^ y for x, y in zip(state[i], w[i + (4 *
r)])]
        # print('round', r, print_matrix(state))

    return state
```

加密解密 demo:

```
Enter you text: 00112233445566778899aabbccddeeff
key: 0001020304050607 len: 16
data_in_hex:  30303131323233333434353536363737383839396161626263636464656666

cipher text: ca4299dfe4552ded446ea5ce89a8d251c87d4170445371aad61f438c821dc913 [IN HEX]

deciphered text:
30303131323233333434353536363737383839396161626263636464656666 [IN HEX]
00112233445566778899aabbccddeeff [IN ASCII]

Execution Time: [128 bits]
key_scheduling_time: 0.007558584213256836 seconds
encryption_time: 1.2076332569122314 seconds
decryption_time: 1.5663471221923828 seconds

key: 000102030405060708090a0b len: 24
data_in_hex:  30303131323233333434353536363737383839396161626263636464656666

cipher text: 5edcd55a3e945d911c008de65a526f2cd005d9f4319fef41bf1f456815e015858693dd1667a29e205a3299abc7b27538 [IN HEX]

deciphered text:
3030313132323333343435353636373738383939616162626363646465656662020202020202020202020202020202020 [IN HEX]
00112233445566778899aabbccddeeff                [IN ASCII]

Execution Time: [192 bits]
key_scheduling_time: 0.03529930114746094 seconds
encryption_time: 2.338735342025757 seconds
decryption_time: 2.6563503742218018 seconds

key: 000102030405060708090a0b0c0d0e0f len: 32
data_in_hex:  30303131323233333434353536363737383839396161626263636464656666

cipher text: db0307ab4cd9860143016f4e0eddd10c1d45ceb28556611fde46bc219aafd5d6 [IN HEX]

deciphered text:
30303131323233333434353536363737383839396161626263636464656666 [IN HEX]
00112233445566778899aabbccddeeff [IN ASCII]

Execution Time: [256 bits]
key_scheduling_time: 0.03885221481323242 seconds
encryption_time: 1.9496562480926514 seconds
decryption_time: 2.4058783054351807 seconds
```

3.2 系统硬件集成:

具体的各个模块已经在第二节中讨论了，AES 包括加密和解密两个部分，其原理图如下所示。

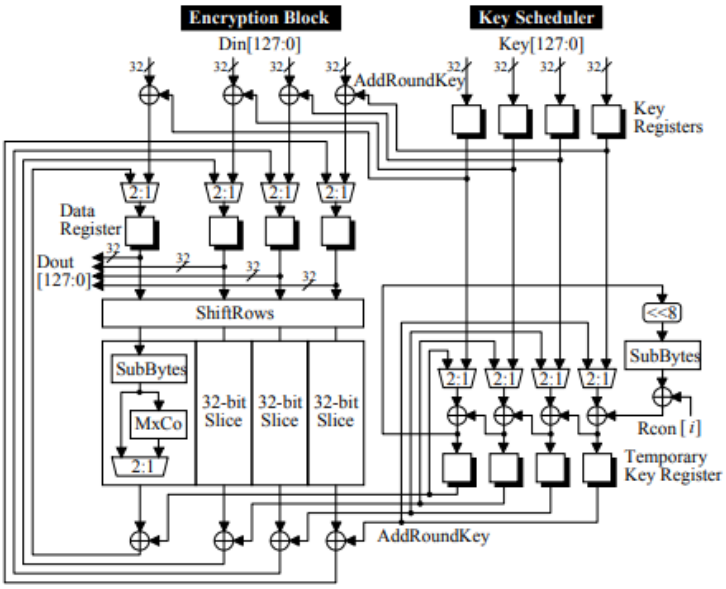


Fig.1 Hardware architecture of the AES encryption module.

其 RTL 代码如下

```
//AES 加密
module AES_ENC(Din, Key, Dout, Drdy, Krdy, RSTn, EN, CLK, BSY, Dvld);
input  [127:0] Din;  // Data input, 128 位明文
input  [127:0] Key;  // Key input, 128 位密钥
output [127:0] Dout; // Data output, 128 位密文

input  Drdy;         // Data input ready
input  Krdy;         // Key input ready
input  RSTn;         // Reset (Low active)
input  EN;           // AES circuit enable
input  CLK;          // System clock
output BSY;          // Busy signal
output Dvld;         // Data output valid

reg  [127:0] Drg;    // Data register, 当前状态矩阵
reg  [127:0] Krg;    // Key register
reg  [127:0] KrgX;   // Temporary key Register, 当前轮密钥
reg  [9:0]   Rrg;    // Round counter, 当前加密轮数，10 位表示 10 轮加密
reg  Dvldrg, BSYrg;
wire [127:0] Dnext, Knext;//下一轮状态矩阵，下一轮轮密钥

EncCore EC (Drg, KrgX, Rrg, Dnext, Knext);//参数依次当前状态矩阵、当前轮密钥、当前加密轮数、下一轮状态矩阵、下一轮轮密钥
```

```
assign Dvld = Dvldrg;
assign Dout = Drg;
assign BSY  = BSYrg;

always @(posedge CLK) begin//CLK 上升沿触发
  if (RSTn == 0) begin//初始化
    Rrg    <= 10'b0000000001;//轮数为 1
    Dvldrg <= 0;//无最终输出
    BSYrg  <= 0;//标志轮加密是否开始
  end//if (RSTn == 0)

  else if (EN == 1) begin

    //还未开始 10 轮加密的准备
    if (BSYrg == 0) begin
     //key input ready=1
     if (Krdy == 1) begin
       Krg    <= Key;//key register=初始密钥
       KrgX   <= Key;//初始轮密钥
       Dvldrg <= 0;//无最终输出
     end//if (Krdy == 1)
     //data input valid
     else if (Drdy == 1) begin
       Rrg    <= {Rrg[8:0], Rrg[9]};//轮数+1
       KrgX   <= Knext;
       Drg    <= Din ^ Krg;//初始时进行轮密钥加
       Dvldrg <= 0;

       BSYrg  <= 1;//置 1,类似于 flag,标志 10 轮加密开始

     end//else if (Drdy == 1)
   end//if (BSYrg == 0)

   //开始 10 轮加密
   else begin
     Drg <= Dnext;//输出的状态矩阵作为下一轮的输入
     if (Rrg[0] == 1) begin//如果是第 10 轮
       KrgX   <= Krg;//轮密钥变为初始密钥
       Dvldrg <= 1;//有最终输出，Dvld = Dvldrg，data output valid
       BSYrg  <= 0;//加密结束 flag 重置为 0
     end//if (Rrg[0] == 1)
     else begin//第 1～9 轮
       Rrg    <= {Rrg[8:0], Rrg[9]};//轮数+1
       KrgX   <= Knext;//输出轮密钥作为下一轮的输入
     end
```

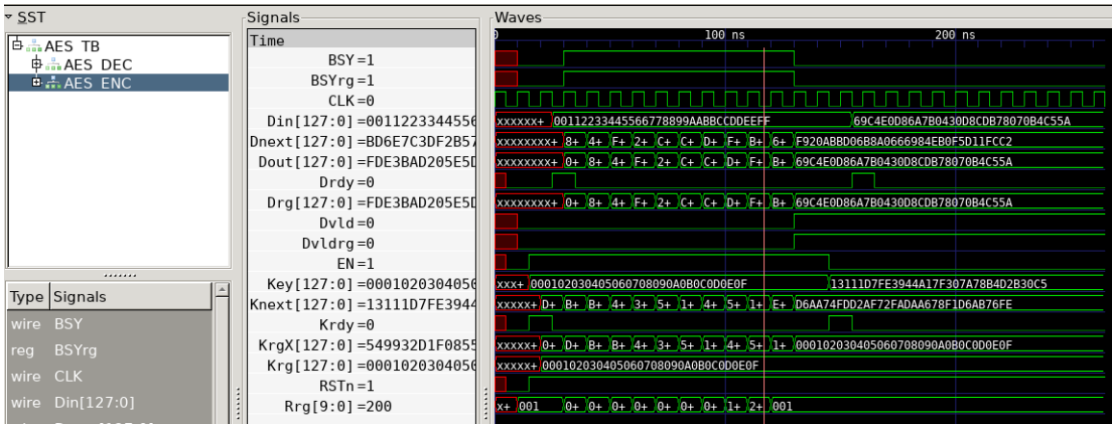
```
    end

  end//else if (EN == 1)

end//always @(posedge CLK) begin
endmodule
```

解码则为相反的过程。

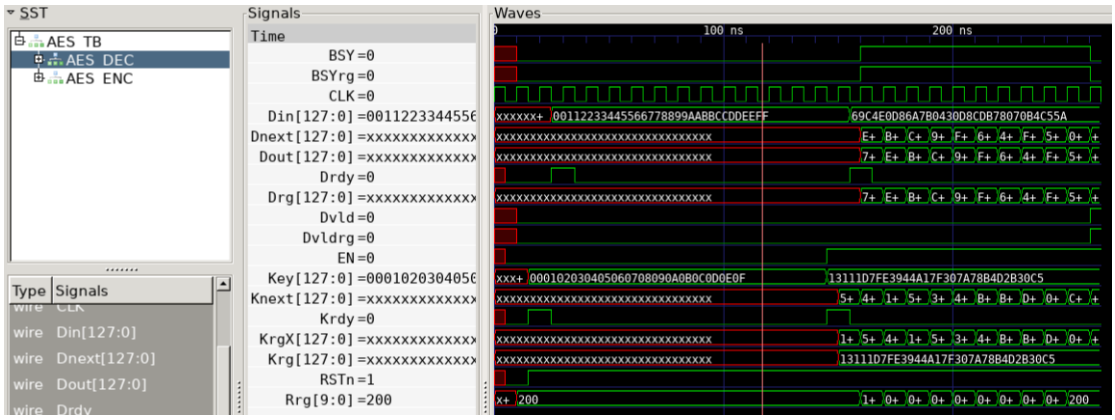
3.3 系统功能验证：

首先对 RTL 进行功能验证，通过开源的综合仿真工具 iverilog+gtkwave 来综合仿真及查看波形。

加密过程：



解密过程



使用 FPGA 进行上板验证验证，查看时序是否真确和使用的资源数。

这边在尝试在 Digilent Nexys A7 或者 PYNQ 板上进行验证，还在尝试中，后续结果会上传 GitHub：[ChuanlaiZang/AES_FPGA_Implement: The FPGA implement of AES（Advanced

Encryption Standard）algorithm (github.com)

参考文献:

[1] Christof Paar, Jan Pelzl 著，马小婷译， 深入浅出密码学。

[2] AES 加密算法原理的详细介绍与实现-CSDN 博客

[3] 密码算法详解——AES - ReadingLover - 博客园 (cnblogs.com)