实验三——AES128算法的加解密的硬件实现

实验分工：

姓名	学号	分工
安茂祥	202000180071	加密、解密模块编写，实验报告编写
廖健有	202000180071	实验报告编写，testbench编写
尹文浩	202000180069	密钥扩展模块编写

表1

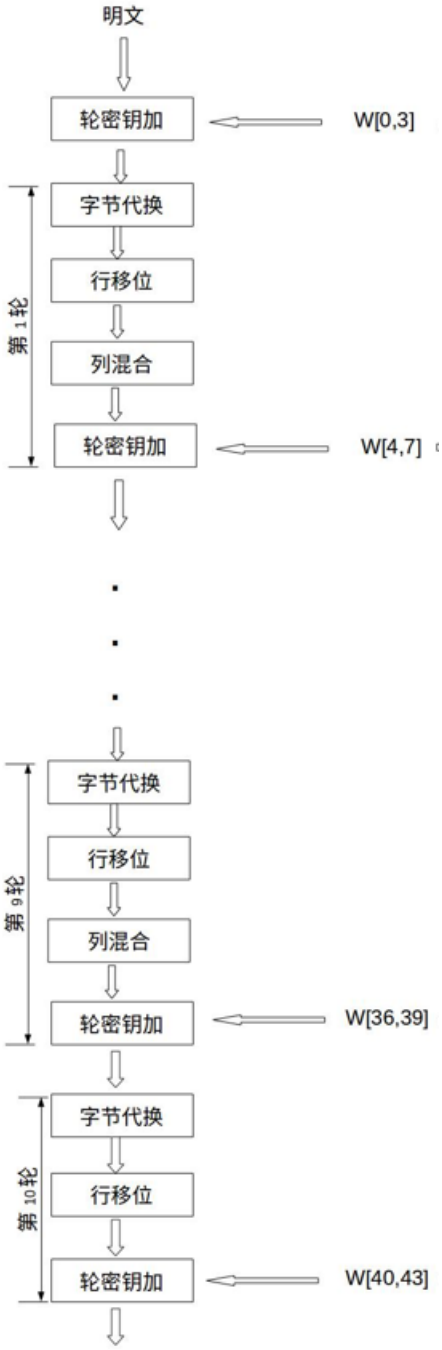
一、AES128算法加解密描述

AES的全称是Advanced Encryption Standard，意思是高级加密标准。其版本有128bit，192bit，256bit对应的是其密钥长度分别为128bit,192bit以及256bit。本次实验主要是对AES128版本的加密解密的硬件实现。AES128密钥长度为128bit，分组长度也为128bit.

（一）加密

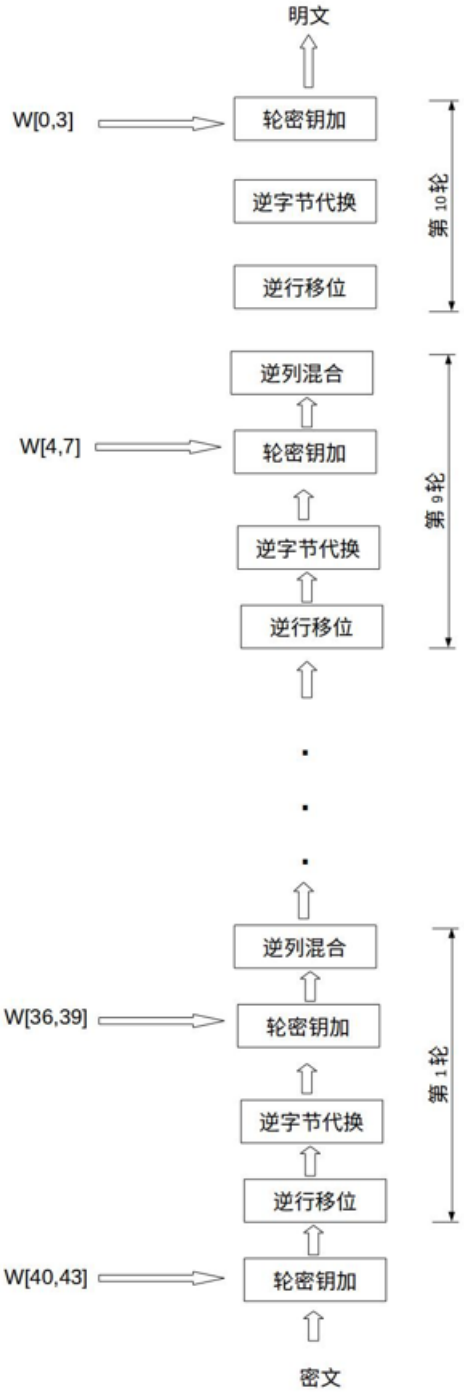
AES128一共有10轮操作，除最后一轮外每一轮包括以下四个步骤：

字节代换，行移位，列混合，异或密钥。最后一轮不含有列混合操作即只包含：字节代换，行移位，异或密钥。此外在算法开始前明文要首先异或原始密钥。下图表示了加密流程。



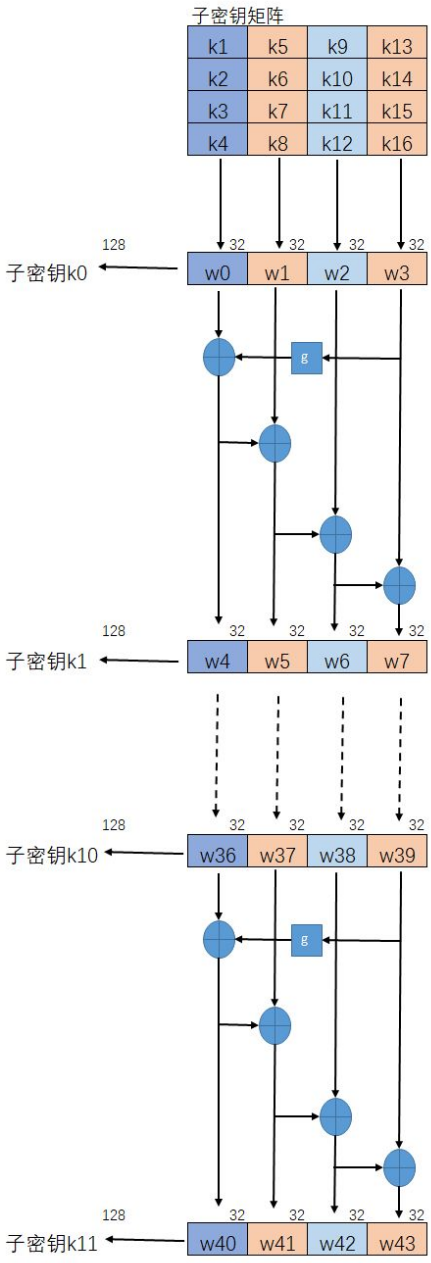
（二）解密

AES128解密操作与加密操作相反，可用下图表述，值得注意的是加解密算法对于轮数的划分并不完全相同。

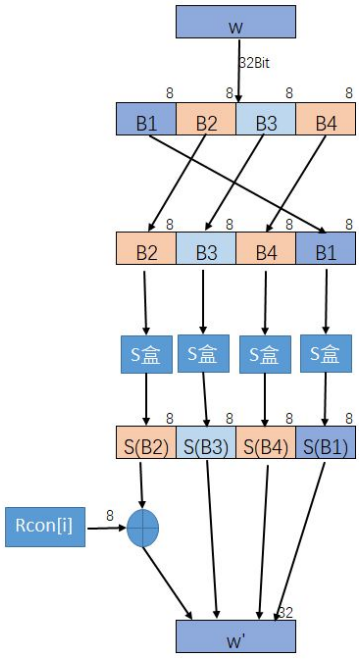


（三）密钥扩展

AES 128除了初始密钥外，还要用到10个128bit轮密钥，因此需要密钥扩展算法进行实现，该算法描述如下图所示：



密钥生成大致流程



G函数处理流程

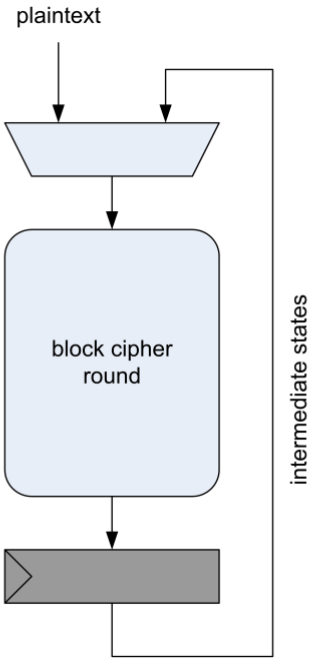
二、硬件实现

对于加解密模块的实现我们总体采用的结构如下所示，这种结构的优点有如下方面：

1、可以有效减少电路所用的门的数目，减小电路的平铺面积。

2、该结构实现的电路可以有较快的时钟周期。

2、该结构可以在11个时钟周期完成一次加密或者解密操作，效率比较高。



（一）密钥扩展模块

对于密钥扩展模块为了使加解密可以共用该模块代码，我们选择一次性生成10个轮密钥。

verlog 代码描述如下：

```
1
2  module gen_key(
3  output [127:0]
   key_r0,key_r1,key_r2,key_r3,key_r4,key_r5,key_r6,key_r7,key_r8,key
   _r9,key_r10,
4  input [127:0]key);
5
6          wire [31:0]
   w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,w18,
   w19,w20,w21,w22,w23,w24,w25,w26,w27,w28,w29,w30,w31,w32,w33,w34,w3
   5,w36,w37,w38,w39,w40,w41,w42,w43;
7
8          wire[31:0]
   out_g1,out_g2,out_g3,out_g4,out_g5,out_g6,out_g7,out_g8,out_g9,out
   _g10;
9
10
11         assign  w0=key[127:96];
12         assign  w1=key[95:64];
13         assign  w2=key[63:32];
14         assign  w3=key[31:0];
```

```
15
16          function_g G1(w3,4'd1,out_g1);
17
18          assign  w4=w0^out_g1;
19          assign  w5=w1^w0^out_g1;
20          assign  w6=w2^w0^out_g1^w1;
21          assign   w7=w3^w0^out_g1^w1^w2;
22
23          function_g G2(w7,4'd2,out_g2);
24
25          assign  w8=w4^out_g2;
26          assign  w9=w4^w5^out_g2;
27          assign  w10=w4^w5^w6^out_g2;
28          assign  w11=w4^w5^w6^w7^out_g2;
29
30          function_g G3(w11,4'd3,out_g3);
31
32          assign  w12=w8^out_g3;
33          assign  w13=w8^w9^out_g3;
34          assign  w14=w8^w9^w10^out_g3;
35          assign  w15=w8^w9^w10^w11^out_g3;
36
37          function_g G4(w15,4'd4,out_g4);
38
39          assign  w16=w12^out_g4;
40          assign  w17=w12^w13^out_g4;
41          assign  w18=w12^w13^w14^out_g4;
42          assign  w19=w12^w13^w14^w15^out_g4;
43
44          function_g G5(w19,4'd5,out_g5);
45
46          assign  w20=w16^out_g5;
47          assign  w21=w16^w17^out_g5;
48          assign  w22=w16^w17^w18^out_g5;
49          assign  w23=w16^w17^w18^w19^out_g5;
50
51          function_g G6(w23,4'd6,out_g6);
52
53          assign  w24=w20^out_g6;
54          assign  w25=w20^w21^out_g6;
55          assign  w26=w20^w21^w22^out_g6;
56          assign  w27=w20^w21^w22^w23^out_g6;
57
58          function_g G7(w27,4'd7,out_g7);
59
60          assign  w28=w24^out_g7;
61          assign  w29=w24^w25^out_g7;
62          assign  w30=w24^w25^w26^out_g7;
```

```
        assign  w31=w24^w25^w26^w27^out_g7;

        function_g G8(w31,4'd8,out_g8);

        assign  w32=w28^out_g8;
        assign  w33=w28^w29^out_g8;
        assign  w34=w28^w29^w30^out_g8;
        assign  w35=w28^w29^w30^w31^out_g8;

        function_g G9(w35,4'd9,out_g9);

    assign  w36=w32^out_g9;
    assign  w37=w32^w33^out_g9;
    assign  w38=w32^w33^w34^out_g9;
    assign  w39=w32^w33^w34^w35^out_g9;

    function_g G10(w39,4'd10,out_g10);

    assign  w40=w36^out_g10;
    assign  w41=w36^w37^out_g10;
    assign  w42=w36^w37^w38^out_g10;
    assign  w43=w36^w37^w38^w39^out_g10;


        assign key_r0={w0,w1,w2,w3};
        assign key_r1={w4,w5,w6,w7};
        assign key_r2={w8,w9,w10,w11};
        assign key_r3={w12,w13,w14,w15};
        assign key_r4={w16,w17,w18,w19};
        assign key_r5={w20,w21,w22,w23};
        assign key_r6={w24,w25,w26,w27};
        assign key_r7={w28,w29,w30,w31};
        assign key_r8={w32,w33,w34,w35};
        assign key_r9={w36,w37,w38,w39};
        assign key_r10={w40,w41,w42,w43};
endmodule
module function_g(input [31:0] w,input [3:0] i,output [31:0]
D_out);

    wire [31:0] shift_w ;
    wire [7:0] S_wire;
    reg [7:0] RC;

    assign shift_w[31:24] = w[23:16];//w0
    assign shift_w[23:16] = w[15:8];//w1
    assign shift_w[15:8] = w[7:0];//w2
    assign shift_w[7:0] = w[31:24];//w3

```

```
    S_box S3(.c( D_out[7:0]), .a(shift_w[7:0]));
    S_box S2(.c( D_out[15:8]), .a(shift_w[15:8]));
    S_box S1(.c( D_out[23:16]), .a(shift_w[23:16]));
    S_box S0(.c( S_wire), .a(shift_w[31:24]));

    always@(*)
        begin

        case (i)

         4'd01: RC = 8'h01;
         4'd02: RC = 8'h02;
         4'd03: RC = 8'h04;
         4'd04: RC = 8'h08;
         4'd05: RC = 8'h10;
         4'd06: RC = 8'h20;
         4'd07: RC = 8'h40;
         4'd08: RC = 8'h80;
         4'd09: RC = 8'h1B;
         4'd10: RC = 8'h36;
         default:
            RC = 8'h01;
            endcase
            end

        assign D_out[31:24]=S_wire^RC;

endmodule
```

（二）加密模块

1. 字节替换操作

字节替换操作为了提高加密效率，使一个时钟周期完成一个轮操作，我们选择在此处将16个Sbox电路进行平铺设计。verilog 描述如下：

```

module Subbytes(
input [127:0] data,
output [127:0] sb
);
     S_box q0( .a(data[127:120]),.c(sb[127:120]) );
     S_box q1( .a(data[119:112]),.c(sb[119:112]) );
     S_box q2( .a(data[111:104]),.c(sb[111:104]) );
     S_box q3( .a(data[103:96]),.c(sb[103:96]) );

```

```
    S_box q4( .a(data[95:88]),.c(sb[95:88]) );
    S_box q5( .a(data[87:80]),.c(sb[87:80]) );
    S_box q6( .a(data[79:72]),.c(sb[79:72]) );
    S_box q7( .a(data[71:64]),.c(sb[71:64]) );

    S_box q8( .a(data[63:56]),.c(sb[63:56]) );
    S_box q9( .a(data[55:48]),.c(sb[55:48]) );
    S_box q10(.a(data[47:40]),.c(sb[47:40]) );
    S_box q11(.a(data[39:32]),.c(sb[39:32]) );

    S_box q12(.a(data[31:24]),.c(sb[31:24]) );
    S_box q13(.a(data[23:16]),.c(sb[23:16]) );
    S_box q14(.a(data[15:8]),.c(sb[15:8]) );
    S_box q16(.a(data[7:0]),.c(sb[7:0]) );

endmodule
```

其中S_box模块只有一个简单的查表操作，在此就省略了。

2. 行移位

行移位的实现也比较简单，只是一个拉线的过程，verilog描述如下：

```
module shiftRaw(
input [127:0] sb,
output [127:0] sr
);
assign         sr[127:120] = sb[127:120];
assign         sr[119:112] = sb[87:80];
assign         sr[111:104] = sb[47:40];
assign         sr[103:96] = sb[7:0];

assign          sr[95:88] = sb[95:88];
assign          sr[87:80] = sb[55:48];
assign          sr[79:72] = sb[15:8];
assign          sr[71:64] = sb[103:96];

assign          sr[63:56] = sb[63:56];
assign          sr[55:48] = sb[23:16];
assign          sr[47:40] = sb[111:104];
assign          sr[39:32] = sb[71:64];

assign          sr[31:24] = sb[31:24];
assign          sr[23:16] = sb[119:112];
assign          sr[15:8] = sb[79:72];
assign          sr[7:0] = sb[39:32];
endmodule

```

3. 列混合

对于列混合运算由于加密过程中的矩阵乘法中的乘数只有1，2，3三种数，并且这三种数在GF(2^8)上的运算都比较简单，因此我们可以通过推导出某一列经过列混合后得到的8bit数的逻辑表达式。

$$[2 \quad 3 \quad 1 \quad 1] \times \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} = m_1$$

那么m_1的8bit关于i_1, i_2, i_3, i_4的逻辑表达式可以写为：

$$\begin{aligned}
&m_1[7] = i_1[6] \oplus i_2[6] \oplus i_2[7] \oplus i_3[7] \oplus i_4[7];\\
&m_1[6] = i_1[5] \oplus i_2[5] \oplus i_2[6] \oplus i_3[6] \oplus i_4[6];\\
&m_1[5] = i_1[4] \oplus i_2[4] \oplus i_2[5] \oplus i_3[5] \oplus i_4[5];\\
&m_1[4] = i_1[3] \oplus i_1[7] \oplus i_2[3] \oplus i_2[4] \oplus i_2[7] \oplus i_3[4] \oplus i_4[4];\\
&m_1[3] = i_1[2] \oplus i_1[7] \oplus i_2[2] \oplus i_2[3] \oplus i_2[7] \oplus i_3[3] \oplus i_4[3];\\
&m_1[2] = i_1[1] \oplus i_2[1] \oplus i_2[2] \oplus i_3[2] \oplus i_4[2];\\
&m_1[1] = i_1[0] \oplus i_1[7] \oplus i_2[0] \oplus i_2[1] \oplus i_2[7] \oplus i_3[1] \oplus i_4[1];\\
&m_1[0] = i_1[7] \oplus i_2[7] \oplus i_2[0] \oplus i_3[0] \oplus i_4[0];
\end{aligned}$$

同样对于乘 1 2 3 1的列只需要颠倒i_1, i_2, i_3, i_4的位置即可。

因此我们的列混合verilog代码描述如下：

```
1
2  module mixColumn(
3  input [127:0] a,
4  output [127:0] mcl
5  );
6
7
8  assign mcl[127:120]= mixcolumn32
   (a[127:120],a[119:112],a[111:104],a[103:96]);
9  assign mcl[119:112]= mixcolumn32
   (a[119:112],a[111:104],a[103:96],a[127:120]);
10 assign mcl[111:104]= mixcolumn32
   (a[111:104],a[103:96],a[127:120],a[119:112]);
11 assign mcl[103:96]= mixcolumn32
   (a[103:96],a[127:120],a[119:112],a[111:104]);
12
13 assign mcl[95:88]= mixcolumn32
   (a[95:88],a[87:80],a[79:72],a[71:64]);
```

```
14  assign mcl[87:80]= mixcolumn32
    (a[87:80],a[79:72],a[71:64],a[95:88]);
15  assign mcl[79:72]= mixcolumn32
    (a[79:72],a[71:64],a[95:88],a[87:80]);
16  assign mcl[71:64]= mixcolumn32
    (a[71:64],a[95:88],a[87:80],a[79:72]);
17
18  assign mcl[63:56]= mixcolumn32
    (a[63:56],a[55:48],a[47:40],a[39:32]);
19  assign mcl[55:48]= mixcolumn32
    (a[55:48],a[47:40],a[39:32],a[63:56]);
20  assign mcl[47:40]= mixcolumn32
    (a[47:40],a[39:32],a[63:56],a[55:48]);
21  assign mcl[39:32]= mixcolumn32
    (a[39:32],a[63:56],a[55:48],a[47:40]);
22
23  assign mcl[31:24]= mixcolumn32 (a[31:24],a[23:16],a[15:8],a[7:0]);
24  assign mcl[23:16]= mixcolumn32 (a[23:16],a[15:8],a[7:0],a[31:24]);
25  assign mcl[15:8]= mixcolumn32 (a[15:8],a[7:0],a[31:24],a[23:16]);
26  assign mcl[7:0]= mixcolumn32 (a[7:0],a[31:24],a[23:16],a[15:8]);
27
28
29  function [7:0] mixcolumn32;//单列进行列混合
30  input [7:0] i1,i2,i3,i4;
31  begin
32  mixcolumn32[7]=i1[6] ^ i2[6] ^ i2[7] ^ i3[7] ^ i4[7];
33  mixcolumn32[6]=i1[5] ^ i2[5] ^ i2[6] ^ i3[6] ^ i4[6];
34  mixcolumn32[5]=i1[4] ^ i2[4] ^ i2[5] ^ i3[5] ^ i4[5];
35  mixcolumn32[4]=i1[3] ^ i1[7] ^ i2[3] ^ i2[4] ^ i2[7] ^ i3[4] ^
    i4[4];
36  mixcolumn32[3]=i1[2] ^ i1[7] ^ i2[2] ^ i2[3] ^ i2[7] ^ i3[3] ^
    i4[3];
37  mixcolumn32[2]=i1[1] ^ i2[1] ^ i2[2] ^ i3[2] ^ i4[2];
38  mixcolumn32[1]=i1[0] ^ i1[7] ^ i2[0] ^ i2[1] ^ i2[7] ^ i3[1] ^
    i4[1];
39  mixcolumn32[0]=i1[7] ^ i2[7] ^ i2[0] ^ i3[0] ^ i4[0];
40  end
41  endfunction
42  endmodule
```

4. 异或密钥

由于该模块操作比较简单，直接将其放入单轮函数中进行了。

5. 一轮加密模块

有了上述字节替换，行移位，列混合，异或密钥后，便可以写出一整轮的加密模块。

verilog 描述如下：

```
1
2  module round(
3      input [3:0]rc,//当前加密轮数,
4  input [127:0]ctx,//本轮输入
5  input [127:0]key,//本轮的密钥
6  output [127:0]ret//本轮加密结果
7      );
8  wire[127:0] sb,sr,mc;
9  reg [127:0]final;
10 Subbytes SUB(.data(ctx),.sb(sb));
11 shiftRaw SR(.sb(sb),.sr(sr));
12 mixColumn MC(.a(sr),.mcl(mc));
13 always@(*)begin
14 if(rc==10)
15 final<=key^sr;
16 else
17 final<=key^mc;
18 end
19 assign ret=final;
20 endmodule
21
```

其中always语句主要目的使在最后一轮跳过列混合直接将行移位后的结果异或本轮轮密钥输出。

6. 加密顶层模块

对于顶层模块设计主要包含三部分内容：

1、轮函数实例化

2、密钥扩展实例化

3、加密状态控制

verilog描述如下：

```
1  module top_aes(
2  input clk,//时钟信号
3  input [127:0]din,//明文
4  input [127:0]key,//密钥
```

```
input reset,//重置信号 高位有效
output reg [127:0]ctxt,//密文
output reg done//加密结束标志，done==1 此时ctxt才有效
    );
    wire[127:0] rk[0:10];//记录轮密钥
    reg[3:0]  rc;//记录当前轮数
    reg init=0;//当前加密状态  init=0首先进行初始化
    reg[127:0]ctx,rkey;
    wire[127:0]ctx_new;//存储每轮的最新状态
round R(.rc(rc),.ctx(ctx),.key(rkey),.ret(ctx_new));//轮函数实例化
gen_key
GK(.key(key),.key_r0(rk[0]),.key_r1(rk[1]),.key_r2(rk[2]),.key_r3(
rk[3]),.key_r4(rk[4]),.key_r5(rk[5]),.key_r6(rk[6]),.key_r7(rk[7])
,.key_r8(rk[8]),.key_r9(rk[9]),.key_r10(rk[10])); //密钥扩展模块实
例化
always@(posedge clk,posedge reset)begin
if(reset)begin
 rc<=0;
 init<=0;
 end
else if(init==0)begin
ctx<=din^rk[0];
rkey<=rk[1];
done<=0;
init<=1;
rc<=1;
end
else if(done==1)begin
    rc<=11;
    end
else if(rc<=9)begin
ctx<=ctx_new;
rkey<=rk[rc+1];
rc<=rc+1;
end
else if(rc==10) begin
    done<=1;
    ctxt<=ctx_new;
end
end
endmodule

```

从实例代码中可以看到我们只对轮函数进行了一次实例化，用寄存器存储中间状态值，从而实现对电路门数的减少。本次加密每轮进行一次加密，因此算上初始化用的一个时钟周期，其余10个周期便可以完成一次128bit加密操作。

（三）解密模块

对于解密操作其逆字节替换，与逆行移位的实现思路与加密操作一致，只是逆过程有，因此此处就不再赘述。

1. 逆列混合

逆列混合需要乘上逆矩阵，由于E,B,D,9这些数值都比较大，因此在实现时选择用基于乘二与加一对这些数的乘法完成设计。

如：$a \times E = a \times 2 \times 2 \times 2 + a \times 2 \times 2 + a \times 2$

基于此思想设计逆列混合模块，verilog描述如下：

$$\begin{bmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ E & D & 9 & E \end{bmatrix}$$

逆矩阵

```
1
2  module mixColumn_ins(
3  input [127:0]din,
4  output [127:0]dout
5      );
6  assign
   dout[127:120]=Mult_ebd9(din[127:120],din[119:112],din[111:104],din
   [103:96]);
7  assign
   dout[119:112]=Mult_ebd9(din[119:112],din[111:104],din[103:96],din[
   127:120]);
8  assign
   dout[111:104]=Mult_ebd9(din[111:104],din[103:96],din[127:120],din[
   119:112]);
9  assign
   dout[103:96]=Mult_ebd9(din[103:96],din[127:120],din[119:112],din[1
   11:104]);
10
11
12 assign
   dout[95:88]=Mult_ebd9(din[95:88],din[87:80],din[79:72],din[71:64])
   ;
13 assign
   dout[87:80]=Mult_ebd9(din[87:80],din[79:72],din[71:64],din[95:88])
   ;
```

```
14  assign
    dout[79:72]=Mult_ebd9(din[79:72],din[71:64],din[95:88],din[87:80])
    ;
15  assign
    dout[71:64]=Mult_ebd9(din[71:64],din[95:88],din[87:80],din[79:72])
    ;
16
17  assign
    dout[63:56]=Mult_ebd9(din[63:56],din[55:48],din[47:40],din[39:32])
    ;
18  assign
    dout[55:48]=Mult_ebd9(din[55:48],din[47:40],din[39:32],din[63:56])
    ;
19  assign
    dout[47:40]=Mult_ebd9(din[47:40],din[39:32],din[63:56],din[55:48])
    ;
20  assign
    dout[39:32]=Mult_ebd9(din[39:32],din[63:56],din[55:48],din[47:40])
    ;
21
22  assign
    dout[31:24]=Mult_ebd9(din[31:24],din[23:16],din[15:8],din[7:0]);
23  assign
    dout[23:16]=Mult_ebd9(din[23:16],din[15:8],din[7:0],din[31:24]);
24  assign
    dout[15:8]=Mult_ebd9(din[15:8],din[7:0],din[31:24],din[23:16]);
25  assign
    dout[7:0]=Mult_ebd9(din[7:0],din[31:24],din[23:16],din[15:8]);
26
27  function[7:0]Mult_ebd9(input[7:0]d1,input[7:0]d2,input[7:0]d3,inpu
    t[7:0]d4);
28  begin
29  Mult_ebd9= Mult_e(d1)^Mult_b(d2)^Mult_d(d3)^Mult_9(d4);
30  end
31  endfunction
32
33   function [7:0] Mult2 (input [7:0] i_Din);
34          begin
35              Mult2 = {i_Din[6:0],1'b0}^(8'h1b&{8{i_Din[7]}});
36          end
37   endfunction
38
39   function [7:0]Mult_e(input [7:0]i_Din);
40   begin
41
     Mult_e=Mult2(Mult2(Mult2(i_Din)))^Mult2(Mult2(i_Din))^Mult2(i_Din)
    ;//8+4+2=e
42   end
```

```
 endfunction

 function [7:0]Mult_d(input [7:0]i_Din);
 begin

 Mult_d=Mult2(Mult2(Mult2(i_Din)))^Mult2(Mult2(i_Din))^i_Din;//8+4+1=d
 end
 endfunction

 function [7:0]Mult_9(input [7:0]i_Din);
  begin
         Mult_9=Mult2(Mult2(Mult2(i_Din)))^i_Din;//8+1=9
  end
  endfunction
  function [7:0]Mult_b(input [7:0]i_Din);
   begin

 Mult_b=Mult2(Mult2(Mult2(i_Din)))^Mult2(i_Din)^i_Din;//8+2+1=b
   end
   endfunction

endmodule
```

2. 解密top模块

设计方式与加密基本一致，verilog语言描述如下：

```
1
2  module DE_AES_top(
3  input clk,//时钟信号
4  input[127:0] entxt,//128bit密文
5  input reset,//复位信号 高电平有效
6  input [127:0]key,//128bit密钥
7  output reg done,//结束信号
8  output reg [127:0] ptxt//输出明文
9      );
10 reg[127:0] rout;//记录每轮的中间状态
11 wire[127:0]rout_new;
12     wire[127:0] rk[0:10];//记录所有轮密钥
13     reg[127:0] round_key;//当前所需轮密钥
14     reg[3:0]rc=0;//当前轮数
15 reg init=0;//初始化状态
16 gen_key
   GK1(.key(key),.key_r0(rk[0]),.key_r1(rk[1]),.key_r2(rk[2]),.key_r3(rk[3]),.key_r4(rk[4]),.key_r5(rk[5]),.key_r6(rk[6]),.key_r7(rk[7]),.key_r8(rk[8]),.key_r9(rk[9]),.key_r10(rk[10]));//密钥扩展实例化
```

```
    round_ins
RI(.rk(round_key),.rc(rc),.din(rout),.dout(rout_new));//轮函数实例
化
always@(posedge clk,posedge reset)
begin
if(reset)begin
 rc<=0;
 init<=0;
 end
 else if(init==0)begin
 rout<=entxt^rk[10];
 round_key<=rk[9];
 init<=1;
 rc<=1;
 done<=0;
 end
 else if(done==1)begin
 rc<=11;
 end
 else if(rc<=9)begin
 rout<=rout_new;
 round_key<=rk[9-rc];
 rc<=rc+1;
 end
 else if(rc==10)begin
 done<=1;
 ptxt<=rout_new;
 end
end
endmodule
```

三、仿真结果

编写testbench文件，将明文设置为3243f6a8885a308d313198a2e0370734。

其正确密文设置为解密的输入：3925841D02DC09FBDC118597196A0B32。

密钥均为：2b7e151628aed2a6abf7158809cf4f3c。

在iverilog下进行仿真得到结果如下：

可以看到加解密均得到了正确结果。

四、实验心得

本次实验通过小组努力实现了AES128 的加解密操作，也明白了在硬件实现中的对于电路大小与电路速度的权衡。本次实验进一步加强了我们对于硬件实现密码算法的认识与理解，希望通过本次实验可以进一步对硬件实现有更深刻的理解与认识。