密码工程实验报告

实验名称 Block cipher

目录

1 实验目的

本实验目的是使用硬件实现一个分组加密算法，本实验将以 AES 算法为例进行实现，硬件实现的算法具有以下的特点.

1. 有逻辑门和电路组成，内部存在并行
2. 由组合逻辑电路和时序逻辑电路构成
3. 在电路大小和效率上作平衡

2 实现过程

AES 是以 SPN 结构位基础设计的算法，这里以 128bit 的 AES 为例.

AES 的操作是在一个 4×4 的 State 矩阵上进行的，首先要把 128bit 的明文解析进 State 矩阵.(按 column 竖向排列),AES 将明文转化为密文通过不断迭代更新 state 进行.

每次轮函数过程包含四步

AddRoundKey:State 中的每一个字节和轮密钥中的对应字节做异或运算

SubBytes: State 中的每一个字节按一定规则用 S 盒中的元素进行替换

ShiftRows: State 中第 i 行的元素左移 i-1 个字节

MixColumns: State 每一列的四个字节被混合通过可逆的线性运算

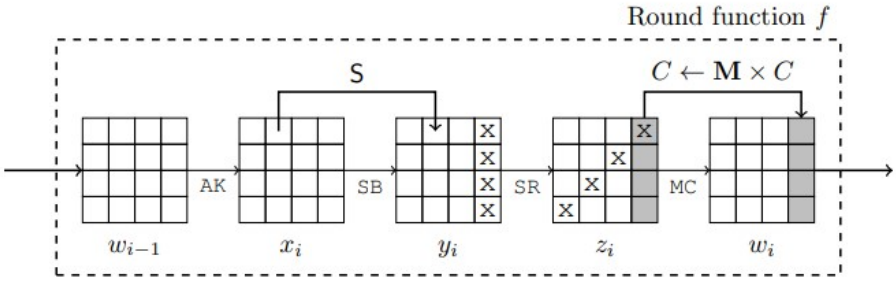

图 1: Round Function

2.1 电路设计

朴素的 AES 组合逻辑电路设计思路是将明文经过多轮的连续运算得到密文，我在这里应用的优化设计是将多轮的 AES 加密过程变成一轮的 AES，多轮的控制则有时序逻辑实现，这样的实现方法能够大大减少硬件逻辑电路的规模，能使用更少的电路完成 AES 加密过程.

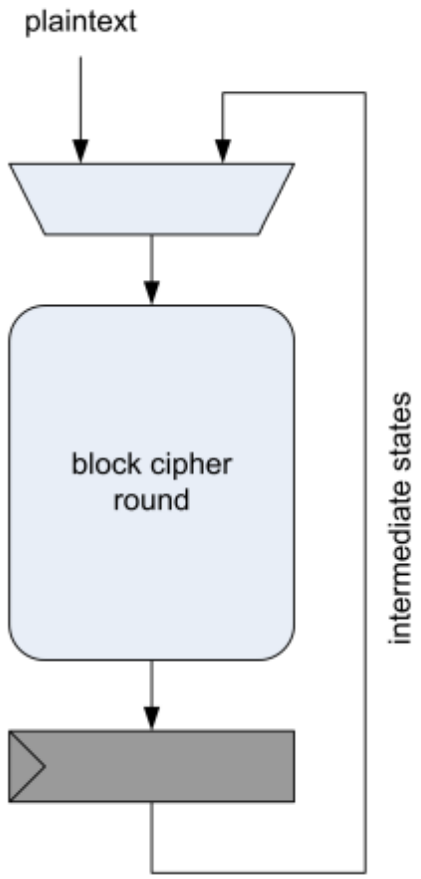

图 2: structure

2.2 S 盒实现

在 S 盒的实现中，我们将 S 盒设计为一个查表运算电路，且将 S 盒设计为一个 module，这样能在 AES 顶层模块中进行例化重复调用，这样能够减少硬件空间的面

积.

这里我们实际可以做一个 trade-off，在 S 盒 module 中，我们使用 4 组 8bit 数据做一个并行的查表运算，这样一个时钟周期可以完成 4 次查表。而在顶层模块设计中，我们采用串行模块设计，每次将一个 32bit 过 S 盒然后存到寄存器中，然后下一个时钟周期再使用同一个 S 盒进行查表运算，这样可以减少大量实例化 S 盒带来的空间开销.

2.3 Mixcolumn 优化设计

Mixcolumn 是在有限域上的线性运算，因此在 Mixcolumn 的硬件实现中我们可以使用真值表和布尔代数方法进行实现，但是这样不仅会占用大量的逻辑门，增加电路体积，而且实现上也不高效。

我们可以只使用有限的异或门来实现这个过程，我们可以让 $s = a_7a_6a_5a_4a_3a_2a_1a_0$ 来代表多项式

$$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

在 GF (2^8) 上的不可约多项式为 $x^8 + x^4 + x^3 + x + 1$，那么可以将域上的乘法运算做如下简化

$$\begin{aligned}
& x\left(a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0\right) \\
& \quad \bmod \left(x^8 + x^4 + x^3 + x + 1\right) \\
=& a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \\
& \quad \bmod \left(x^8 + x^4 + x^3 + x + 1\right) \\
=& a_6x^7 + a_5x^6 + (a_4 + a_7)\,x^5 + (a_3 + a_7)\,x^4 + a_2x^3 + a_1x^2 \\
& + (a_0 + a_7)\,x + a_7
\end{aligned}$$

- $2 \cdot s \rightarrow a_6a_5\,(a_4 + a_7)\,(a_3 + a_7)\,a_2a_1\,(a_0 + a_7)\,a_7$
- $3 \cdot s = 2 \cdot s + s$
- $3 \cdot s \rightarrow a_6a_5\,(a_4 + a_7)\,(a_3 + a_7)\,a_2a_1\,(a_0 + a_7)\,a_7 + a_7a_6a_5a_4a_3a_2a_1a_0$

- $3 \cdot s \rightarrow (a_6 + a_7)(a_5 + a_6)(a_4 + a_7 + a_5)(a_3 + a_7 + a_4)(a_2 + a_3)(a_1 + a_2)(a_0 + a_7 + a_1)(a_7 + a_0)$

具体代码实现如下

```
1
2   module MixColumns(x,y);
3   input [31:0] x;
4   output [31:0] y;
5
6   wire [7:0] a3,a2,a1,a0;
7
8   assign a3 = x[31:24];
9   assign a2 = x[23:16];
10  assign a1 = x[15: 8];
11  assign a0 = x[ 7: 0];
12
13  function [7:0] SS;
14      input [7:0] data;
15      SS = {data[6],data[5],data[4],data[3]^data[7],data[2]^data[7],data
            [1],data[0]^data[7],data[7]};
16  endfunction
17  assign y = {SS(a3)^(SS(a2)^a2)^a1^a0,a3^SS(a2)^(SS(a1)^a1)^a0,a3^a2^SS
        (a1)^(SS(a0)^a0),(SS(a3)^a3)^a2^a1^SS(a0)};
18  endmodule
```

2.4 Shiftrow 实现

在行移位的设计中，为了加快硬件算法的执行速度，我们使用数据拼接的方式. 将过 S 盒后的向量按照不同的比特位置装入到新的向量中，这样可以不使用移位运算，能够加快运算速度.

具体的代码实现如下，在代码实现中，我们将列向量按行方向进行装入，因此在进行行移位运算时，我们要对实际向量的列方向进行移位.

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

2.5 密钥扩展算法

AES128 中的原始密钥 Key 为 16 个字节, 运算中包含原始密钥需要 11 个 State 矩阵大小的密钥, 每一列所包含 32 位记为一个 uint32_t W, 所以密钥扩展一共需要产生 44 个列 W, 即 uint32_t W[44].

$$W[n] = \begin{cases} W[n-4] \oplus W[n-1], & \text{if } n! = 4 \text{ 的倍数.} \\ W[n-4] \oplus \mathrm{Mix}(W[n-1]) \oplus \mathrm{rcon}[(n/4)-1], & \text{if } n == 4 \text{ 的倍数.} \end{cases}$$

其中 $Mix(x) = SubWord(RotWord(x))$,$RotWord()$ 为循环左移一位,$SubWord()$ 为字节替换,$rcon$ 为轮常量异或, 在硬件实现中，我们将每一轮的密钥扩展运算和轮加密运算进行结合，在完成轮加密运算后同时完成下一轮的密钥扩展运算

具体实现如下

```
SubBytes SBk ({ki[23:16],ki[15:8],ki[7:0],ki[31:24]},kp);

  assign ko[127:96] = ki[127:96] ^ {kp[31:24] ^ rcon(Round), kp[23: 0]};
  assign ko[ 95:64] = ki[ 95:64] ^ ko[127:96];
  assign ko[ 63:32] = ki[ 63:32] ^ ko[ 95:64];
  assign ko[ 31: 0] = ki[ 31: 0] ^ ko[ 63:32];
```

2.6 轮迭代

我们在整体的实现中采用的是轮模块重复使用的结构，由于最后一轮和前面不同，因此要加入控制位来决定是否进行额外运算.

```
    assign dn = ((Round[0] == 1)? sr:mx)^ki;
//The last round don't have the mixcolumn.
```

对加密过程的控制采用时序逻辑电路进行，这样能够控制时钟周期来完成加密过程. 在实现中我们加入控制位，首先先初始化轮向量，初始化输出控制位和忙运算位. 然后对忙运算位进行判断，如果忙运算位为 0 则进行向量初始化，放入密钥和明文，并调整输入控制位. 在下面运行过程中，则每轮对轮向量进行判断来确定加密轮数，如过到了第 10 轮，就将输出控制位设为 1，将忙位置 0，准备输出数据. 实现中的核心代码如下：

```
  always @(posedge CLK) begin
if (Reset == 0)
begin
  Rrg <= 10'b0000000001;
  Droutrg <= 0;
  BSYrg <=0;
end
else if(ENK == 1)
begin
  if(BSYrg==0)
  begin
    if (Krin == 1)
    begin
      Krg <= Key;
      KrgT <= Key;
      Droutrg <= 0;
    end
    else if (Drin == 1)
    begin
      Rrg <= {Rrg[8:0],Rrg[9]};
      KrgT <= knext;
      Drg <= Din^Krg;
      Droutrg <= 0;
      BSYrg <=1;
    end
  end
  else begin
    Drg <= Dnext;
```

```
29        if (Rrg[0] == 1)begin
30          KrgT <=Krg;
31          Droutrg <=1;
32          BSYrg<=0;
33        end
34        else begin
35          Rrg <= {Rrg[8:0],Rrg[9]};
36          KrgT <= knext;
37        end
38      end
39    end
40  end
```

3 结果验证

按照 IST.FIPS.197 上给出的附录 B 示例进行验证

Key = 000102030405060708090a0b0c0d0e0f

Plaintext = 00112233445566778899aabbccddeeff

Ciphertext = 69c4e0d86a7b0430d8cdb78070b4c55a

使用 iverilog 编译运行代码，然后使用 gtkwave 来查看输出的波形图发现结果一致. 从上到下依次是轮输出，时钟信号，忙位，输出控制位和轮向量。

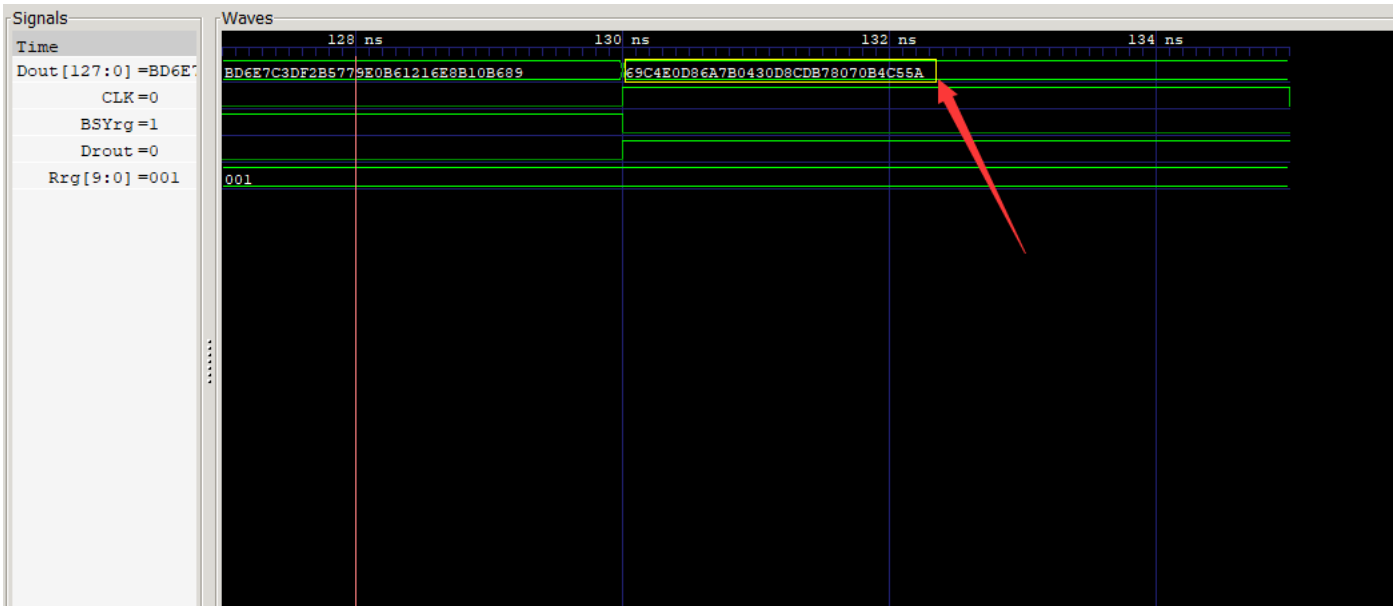

图 3: Verify

4 实验心得

上一个实验我们学习并使用硬件语言实现了流密码，这个实验将使用硬件语言实现分组密码算法. 在这个实验中，我使用硬件描述语言实现了 AES，首先在电路设计中，电路规模是在电路设计中很重要的一点，这里我只实现了一轮的 AES，完成多轮的 AES 过程则通过时序逻辑电路进行控制，时序逻辑电路在这次硬件实现中很重要，它能够对电路中的变量进行控制，根据前一轮的状态对电路做出反应. 另外在实现过程中，对 S 盒和 Mixcolumn 做了适合于硬件的优化设计，通过简化运算只使用有限的异或门来完成运算过程，并通过多个组合逻辑模块进行并行运算，提升了速度.

参考文献

[1] IST.FIPS.197

[2] blockcipher