密码工程实验报告

实验名称　　流密码算法的硬件实现

目录

1 流密码介绍

流密码 (stream cipher)，是一种对称加密算法，加密和解密双方使用相同伪随机加密数据流（pseudo-random stream）作为密钥，明文数据每次与密钥数据流顺次对应加密，得到密文数据流。实践中数据通常是一个位（bit）并用异或（xor）操作加密。

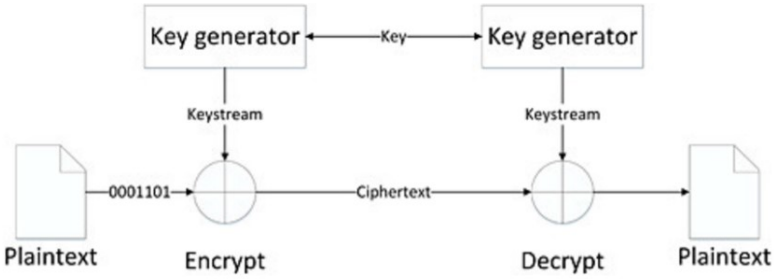

图 1: Stream Cipher

伪随机密钥流 (keystream) 由一个随机的种子（seed）通过算法（PRNG，pseudo random number generator）得到，k 作为种子，则 $G(k)$ 作为实际使用的密钥进行加密解密工作。

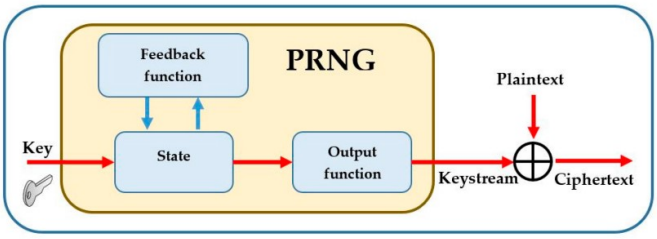

图 2: PRNG

PRNG 主要通过原始状态反馈生成随机的数字序列，PRNG 通过设定随机种子可以从任意初始值开始生，同样的初始值总是生成同样的序列。PRNG 算法具有一定的周期，如果 PRNG 的内部状态包含 n 位，那么它的周期不会超过 2^n，PRNG 常使

用线性反馈移位寄存器 (Linear feedback shift register，LFSR) 生成随机序列,LSFR 是指给定前一状态的输出，将该输出的线性函数再用作输入的移位寄存器。异或运算是最常见的单比特线性函数：对寄存器的某些位进行异或操作后作为输入，再对寄存器中的各比特进行整体移位。

赋给寄存器的初始值叫做 seed key，因为线性反馈移位寄存器的运算是确定性的，所以由寄存器所生成的数据流完全决定于寄存器当时或者之前的状态。而且由于寄存器的状态是有限的，它最终肯定会是一个重复的循环。通过本原多项式，线性反馈移位寄存器可以生成看起来是随机的且循环周期非常长的序列。(M-序列，抽头与本原多项式对应，不包含 0 状态)

2 A5/1 算法实现

我们流密码硬件实现以 A5/1 算法为例，A5/1 基于具有不规则的三个线性反馈移位寄存器 (LFSR) 的组合，三个 LFSR 的指定如下：

LFSR number	Length in bits	Feedback polynomial	Clocking bit	Tapped bits
1	19	$x^{19}+x^{18}+x^{17}+x^{14}+1$	8	13,16,17,18
2	22	$x^{22}+x^{21}+1$	10	20,21
3	23	$x^{23}+x^{22}+x^{21}+x^{8}+1$	10	7,20,21,22

表 1: character of LFSR

最初，寄存器每一位设计为 0，然后在 64 个周期中，根据以下方案混合 64 位密钥：在周期 $0 \leqslant i \leqslant 64$, 第 i 位使用异或将密钥位添加到每个寄存器的最低有效位，然后将寄存器进行移位，接着用同样的方法混合 22 位的帧号，这里实现中默认初始化过程已经完成，只进行密钥输出的过程。

每个寄存器都有一个相关的钟控位。在每个周期，检查三个寄存器的钟控位，并确定多数位 (0 或者 1)。如果钟控位与多数位一致，则对寄存器作移位操作。因此，在每个步骤中至少要对两个或三个寄存器进行移位，并且每个寄存器移位的概率是 3/4。

比如，三个寄存器的钟控位为分别为 0、0、1。此时 0 为多数位，钟控便会移位钟控位为 0 的第一和二个寄存器，而第三个寄存器维持不变。

为了用硬件实现是否进行移位操作，将钟控位输入作为变量，建立移位操作关于钟控位的真值表，然后画出卡诺图进行化简，这里我们以 LFSR1 为例，将钟控位简记为 X,Y,Z. 每一个回合的步骤可以拆分为: 取 Key，反馈多项式，位移.

- **取 key：** 取 key 过程也就是取三个寄存器的最后一位，也就是第 19、22 位和 23 位进行异或操作，得到的结果作为这一轮的 key.
- **Feedback Polynomial:**LFSR 进行位移的前置操作，具体可以分成两步：判定是否需要进行位移和判定新的填充值 **位移:** 采用上述我们介绍过的择多原则，三个寄存器中的钟控信号决定是否进行移位

X \ Y,Z	00	01	11	10
0	1	1	0	1
1	0	1	1	1

表 2: K-map of LFSR1

根据卡诺图的化简规则，我们可以得到相应的逻辑表达式:

$$\text{condition1} = \bar{X}\bar{Y} + XZ + Y\bar{Z}$$

同理，我们可以得到另外两个 LFSR 的条件判断表达式，下面是硬件实现的 module.

```
1
2 module A5(
3     input wire rest,
4     input wire clk,  //时钟信号
5     input wire [63:0] Key,  //初始化密钥
6     output  wire code
7 );
```

```
8
9
10
11 reg[18:0] A;
12 reg[21:0] B;
13 reg[22:0] C;
14 reg X;
15 reg Y;
16 reg Z;
17 reg  re;
18 reg condition1;
19 reg condition2;
20 reg condition3;
21 reg feedback1,feedback2,feedback3;
22
23
24 always @(posedge clk or rest ) begin
25     if(rest)
26     begin
27     A = Key[18:0];
28     B = Key[40:19];
29     C = Key[63:41];
30     end
31     //initialize the key
32     X = A[8];
33     Y = B[10];
34     Z = C[10];
35     //generate round key
36     re = A[18]^B[21]^C[22];
37     //feedback condition judgement
38     //use clocking bit
39     condition1 = ((~X)&(~Y))|(X&Z)|(Y&(~Z));
40     condition2 = ((~Y)&(~Z))|((~X)&Y)|(X&Y);
41     condition3 = ((~Y)&(~Z))|(X&Z)|(Y&(~X));
42     if(condition1 == 1'b1)
```

```
    begin
        feedback1 = A[18]^A[17]^A[16]^A[13];
        A = {A[17:0],feedback1};
    end
    if(condition2 == 1'b1)
    begin
        feedback2 = B[21]^B[20];
        B = {B[20:0],feedback2};
    end
    if(condition3 == 1'b1)
    begin
        feedback3 = C[21]^C[20]^C[22]^C[7];
        C = {C[21:0],feedback3};
    end

end
    assign code = re;

endmodule

/*
addition:
1.the input must wire
2.initial process block is put in textbench module to initialize the value
    of API.
3.The data default use type wire without declaration.
*/
```

下面进行 textbench 的编写，textbench 要对赋值过程进行表示，并将模块进行示例化.

```
`timescale 1ns/1ps
`include "stream_cipher.v"


module text();
```

```
6     reg  clk;
7     reg rest;
8     wire  dataout;
9     reg [63:0] Key;
10
11    //clock generating
12    real CYCLE_200MHZ =5;
13    always begin
14        clk = 0;#(CYCLE_200MHZ/2);
15        clk = 1;#(CYCLE_200MHZ/2);
16    end
17
18    initial begin
19        Key = 64'h3170604015ABCDEF;
20        //initialize the key and h express the hex .
21        rest = 1;
22        #100 //after 100ns end the reset process.
23        rest = 0;
24    end
25
26
27    A5 A5text(
28        .rest      (rest),
29        .clk       (clk),
30        .Key    (Key),
31        .code     (dataout)
32
33    );
34
35    initial begin
36        forever begin
37            #100
38            if($time >= 1000) $finish;
39        end
40    end
```

```
41
42      initial begin
43          $dumpfile("wave.vcd");
44          //check the wave of the module.
45          $dumpvars;
46      end
47
48
49
50  endmodule
```

3 结果验证

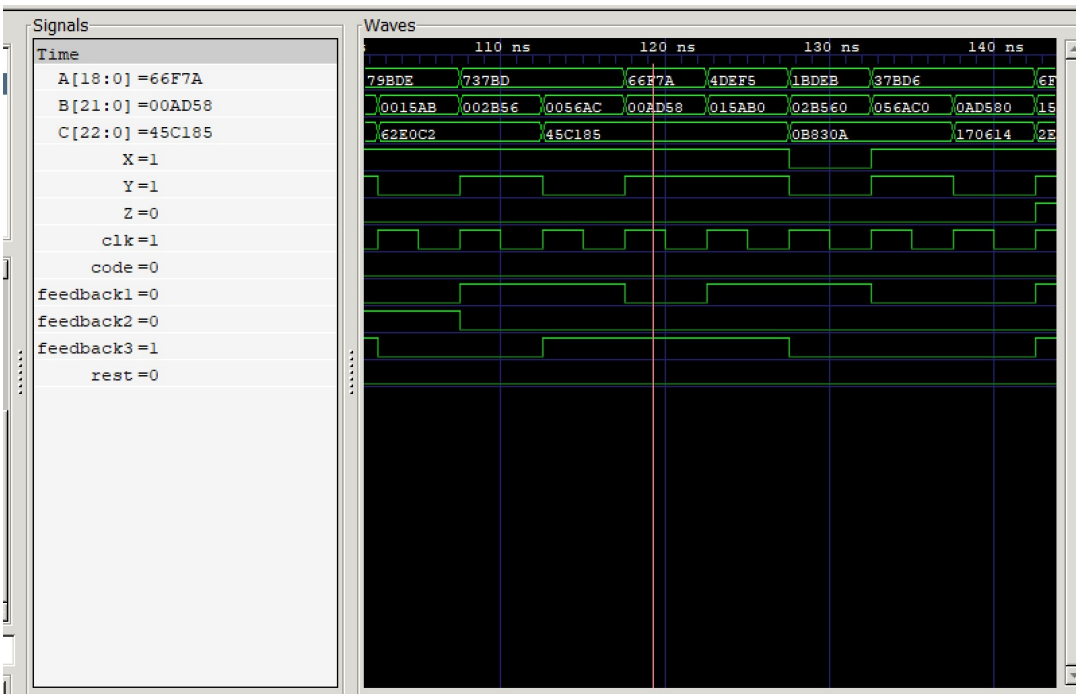

图 3: verify figure

通过 iverilog 对代码进行编译，并使用 gtkwave 对产生的波形图进行查看，任取一个时刻, 我们可以根据移位寄存器相应位置的值验证输出密钥的正确性，可以发现现在钟控位置的值为 1，1，0，前两个 LFSR 需要进行移位，在下个时刻我们可以看到对

应移位结果.

以上图为例，目前 $A = 66F7A, B = 00AD58, C = 45C185$, 输出的 Key 为 $A[19] \oplus B[22] \oplus C[23] = 0$, 这和 code 值一致，根据钟控元素，下个回合 A 和 B 需要移位，可以从波形图中看到下个回合 A,B 都进行了正确的反馈移位，而 C 保持不变.

4 实验心得

通过本次实验进一步了解了流密码算法. 实验中采用 verilog 语言对流密码算法进行硬件实现，在实验中学习并编写了逻辑电路来实现 A5/1 流密码，基本掌握了硬件语言模块的编写和赋值，并编写了 textbench 控制时钟和输入来达到测试效果，还学会通过波形图观察了输出变化. 很重要的一点是通过卡诺图化简变量之间的关系来确定逻辑表达式，这是将数字逻辑课程中学到的知识在实际中进行应用，这让我收获很大.

参考文献

[1] IST.FIPS.197

[2] software_aes_implementation

[3] https://blog.csdn.net/JeronZhou/article/details/115566445