# 实验二——A5/1流密码verilog 实现

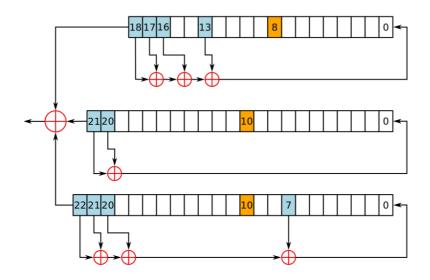
#### 实验分工:

姓名	学号	分工							
安茂祥	202000180071	顶层模块编写, 实验报告编写							
廖健有	202000180071	majority 模块编写,实验报告编写							
尹文浩	202000180069	线性反馈移位寄存器模块编写,testbench编写							

表1

### 一、算法原理

A5/1流密码算法使用了三个线性反馈移位寄存器称为X, Y, Z, 其位数分别为为19, 22, 23。其反馈多项式分别为:  $x_{18} + x_{17} + x_{16} + x_{13} + 1$ ,  $y_{21} + y_{20} + 1$ ,  $z_{22} + z_{21} + z_{20} + z_{7} + 1$ 。每轮密钥流的生成由三个线性反馈移位寄存器的最高位18, 21, 22共同异或确定。具体流程可由以下步骤概括:



# (一) 初始化

首先三个反馈移位寄存器的值均设置为0,此时要将64位会话密钥K,以及22位的公开密钥以一定方法放入三个线性反馈移位寄存器中,具体方法为:

1、在前64个循环中, $R'[0] = R[0] \oplus K[i]$ ,  $0 \le i < 64$ , 其中R[0] 由其响应的线性 反馈多项式确定,每一轮每个线性反馈移位寄存器用此方法计算出的R'[0]进行移位、更新。

- 2、之后进行22个循环,每个循环的操作与1类似,将22位公开密钥也加入线性 反馈移位寄存器中。
- 3、之后进行100次循环,每次每个线性反馈移位寄存器正常更新,此时输出打的密钥流是无效的。

#### (二)密钥流生成

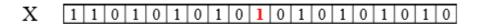
经过初始化阶段后, 要进行密钥流的生成。

假设初始化后三个寄存器目前的状态是这样:



- (1) 首先找到 $X_8 = 1$ ,  $Y_{10} = 0$ ,  $Z_{10} = 1$
- (2) 取数量最多的作为结果M=Maj( $X_8=1$ ,  $Y_{10}=0$ ,  $Z_{10}=1$ )= Maj(1,0, 1)=1; (如果是M=Maj(0,0,1)=0)
- (3) 因为M=1,且 $X_8=1$ , $M=X_8$ ,所以X寄存器需要左移一位,那么因为左移,第一位就空出来了,则  $X_0=x_{18}+x_{17}+x_{16}+x_{13}$  (移位前的)=1

所以最后X寄存器变为



同理,再看Y寄存器,因为M=1,  $Y_{10}=0$ , 所以Y寄存器不需要进行移位,保持不变.



同理, Z寄存器, 因为M=1,  $Z_{10}=1$ , 所以Z寄存器需要左移一位, 第一位 $Z_0=z_{22}+z_{21}+z_{20}+z_7=0$ .

所以最后Y寄存器变为

Z 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0

所以最后得到的三个移位寄存器是

- Y 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
- Z 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0

秘钥是 $S_0 = x_{18} \oplus y_{21} \oplus z_{22} = 1$ ,

就这样得到了第一位秘钥,如果需要64位秘钥,则按照上述进行循环64次操作即可。

# 二、硬件实现

对于A5/1算法的硬件实现主要分位三部分

- 1、三个线性反馈移位寄存器模块实现
- 2、majority模块实现
- 3、顶层模块实现

### (一) 三个线性反馈移位寄存器模块实现

以线性反馈寄存器X实现为例, 定义模块接口如下:

```
module lfsr_1(
input shift_bit,
input trigger,
input clk,
input reset_n,
output [0:18] X
);
```

其中shift\_bit 用于在初始化过程中,将密钥输入。

trigger表示是否移位, trigger=1代表此时进行移位, 否则不变。

clk表示时钟信号。

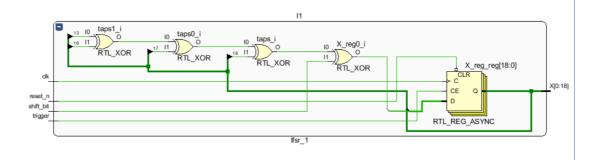
reset\_n表重置信号,低电平有效。

X 输出寄存器的18bit 状态。

内部实现过程代码如下:

```
1
     module lfsr_1(
 2
     input shift_bit,
 3
       input trigger,
 4
       input clk,
 5
       input reset_n,
 6
       output [0:18] X
 7
       );
 8
    reg [0:18] X_reg=0;
 9
    wire taps;
10
11
       assign taps = X_reg[13]^X_reg[16]^X_reg[17]^X_reg[18];
12
       always @(posedge clk,negedge reset_n)
13
       begin
            if(~reset_n)
14
15
                X_reg <= 'b0;
16
            else if (trigger)
17
                X_reg <= {taps^shift_bit,X_reg[0:17]};</pre>
18
       end
19
    assign X = X_reg;
20
21
```

通过vivado综合后,实现后的电路如下:



### (二) majority模块实现

majority模块的功能是输出 $X_8$ ,  $Y_{10}$ ,  $Z_{10}$ 中出现数目最多的数。模块接口设计如下:

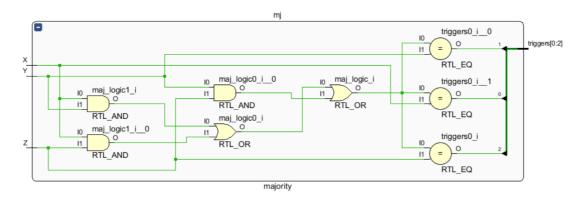
```
1  module majority(
2   input X,
3   input Y,
4   input Z,
5   output reg [0:2]triggers
6  );
```

X, Y, Z分别代表 $X_8, Y_{10}, Z_{10}$ ,输出trigger三个bit 分别代表三个线性反馈移位寄存器是否要进行移位。

#### 实现如下:

```
wire maj_logic;
    assign maj_logic = (X&Y) | (X&Z) | (Y&Z);// 根据真值表得出 三个数出
 2
    现最多的数。
 3
        always @(X,Y,Z)
 4
        begin
 5
            if(maj logic == X)
                 triggers[0] <= 1'b1;</pre>
 6
 7
            else
 8
                 triggers[0] <= 1'b0;
 9
            if(maj_logic == Y)
                 triggers[1] <= 1'b1;</pre>
10
11
            else
12
                 triggers[1] <= 1'b0;
            if(maj_logic == Z)
13
                 triggers[2] <= 1'b1;
14
            else
15
                 triggers[2] <= 1'b0;</pre>
16
17
        end
```

vivado综合后电路如下:



### (三) top 模块实现

顶层模块包括了对线性反馈移位寄存器模块以及majority模块的实例化以及对加密过程的控制。其模块接口如下:

```
1 module top(
2 input clk,
3 input [0:63]secret,
4 input rst,
5 output init_ok,
6 output keystream
7 );
```

clk为时钟信号。

secret为会话密钥,即私钥。

rst重置标志, 低电平有效。

init\_ok为初始化结束的标志,只有其为1时,此时输出的密钥流keystream才有效。

keystream为单比特密钥流,每个时钟周期上升沿输出。

#### 代码如下:

```
1  reg [21:0]public=22'b1101001110000110010001;
2  reg init=0;
3  reg shift_bit;
4  reg [2:0] trigger=3'b111;
5  wire [2:0]temp;
6  wire [0:18]LFSR_1;
7  wire [0:21]LFSR_2;
```

```
8
     wire [0:22] LFSR_3;
 9
    reg [8:0] counter=8'b0;
10
    reg [2:0]reset=0;
    lfsr_1 l1(.shift_bit(shift_bit)
11
    ,.trigger(trigger[0]),.clk(clk),.reset_n(reset[0]),.X(LFSR_1));
12
     lfsr_2 l2(.shift_bit(shift_bit)
    ,.trigger(trigger[1]),.clk(clk),.reset_n(reset[1]),.Y(LFSR_2));
13
     lfsr_3 l3(.shift_bit(shift_bit)
    ,.trigger(trigger[2]),.clk(clk),.reset_n(reset[2]),.Z(LFSR_3));
14
     majority
    mj(.X(LFSR_1[8]),.Y(LFSR_2[10]),.Z(LFSR_3[10]),.triggers(temp));
15
     always@(posedge clk)
16
     begin
     if(~rst)begin
17
18
    counter<=0;
19
     reset<=3'b000;
20
     init<=0;
21
     end
22
     end
23
    always@(posedge clk)begin
    if(counter==0)begin
24
25
    shift bit<=secret[0];</pre>
26
    else if(counter<=63)begin</pre>
27
28
    reset<=3'b111;
29
    shift_bit<=secret[counter];</pre>
30
    trigger<=3'b111;
31
    end
32
    else if(counter<=85) begin</pre>
33
    shift_bit<=public[counter-64];</pre>
34
   trigger<=3'b111;
35
    end
36
    else if(counter<=185)begin</pre>
37
    shift bit<=0;</pre>
38
    trigger<=temp;
39
    end
40
    else
           init<=1;
    if(init==1)begin
41
    shift bit<=0;</pre>
42
43
    trigger<=temp;</pre>
44
    counter<=1;</pre>
45
    end
46
    counter<=counter+1;</pre>
47
     end
48
     assign keystream=LFSR 1[18]^LFSR 2[21]^LFSR 3[22];
     assign init ok=init;
49
```

# 三、运行仿真结果

编写testbench执行仿真,

		1, 865 ps									
Name	Value	0.0 ns	0.5 ns	1. 0 ns	1.5 ns		2. 0 ns	2.5 ns	3. 0 ns	3.5 ns	4.0 ns
1 clk	1										
¼ init_ok	1										
> Numbrasida > public[21:0]	1101001110000110010001					101	0111000011001000				
> 🔣 secret[0:63]	01001110001011110100110101111			010011100010	11101001101	111	1000001111010111	0010001011001110	10		
la keystream	1					Ш					
> 🎇 LFSR_1[0:18]	0111110000100100101										
> MI LFSR_2[0:21]	0100011101111101001000										
> MLFSR_3[0:22]	00010100101111111000100					_		1 1			
> 👹 counter[8:0]	010111011										
			ž∏ <i>†</i> -	化阶段							
			TVJX	TOPTEX					输出密钥流	段	

四、实验心得

通过本次实验初步认识到了密码算法的硬件实现流程,尤其是针对本次基于LFSR的流密码算法的实现,通过本次实现我们团队进一步了解了verilog 语言与实现基本方法与思想。我们一致认为本次实验是充满意义的。