实验二——A5/1流密码verilog 实现

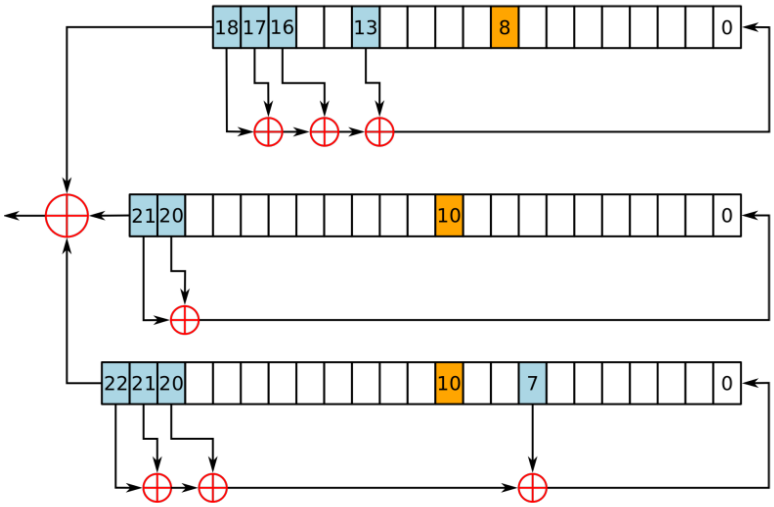
实验分工：

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。

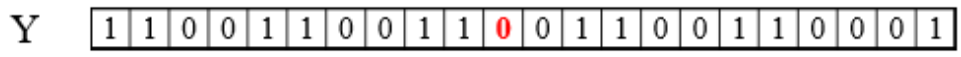
2、之后进行22个循环，每个循环的操作与1类似，将22位公开密钥也加入线性反馈移位寄存器中。

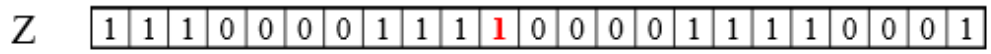
3、之后进行100次循环，每次每个线性反馈移位寄存器正常更新，此时输出打的密钥流是无效的。

（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：



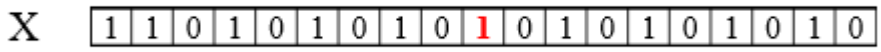


（1）首先找到$X_8=1$，$Y_{10}=0$，$Z_{10}=1$

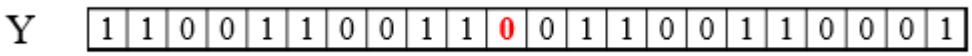
（2）取数量最多的作为结果M=Maj（$X_8=1$，$Y_{10}=0$，$Z_{10}=1$）= Maj（1，0，1）=1；（如果是M=Maj（0,0,1）=0）

（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

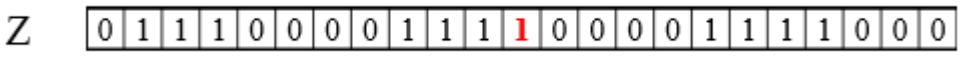


同理，再看Y寄存器，因为$M=1$，$Y_{10}=0$，所以Y寄存器不需要进行移位，保持不变.

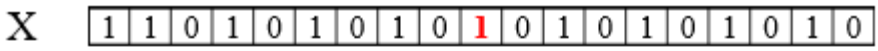


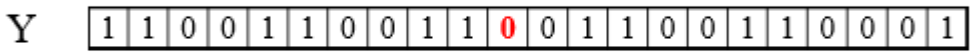
同理，Z寄存器，因为$M=1$，$Z_{10}=1$，所以Z寄存器需要左移一位，第一位$Z_0=z_{22}+z_{21}+z_{20}+z_7=0$.

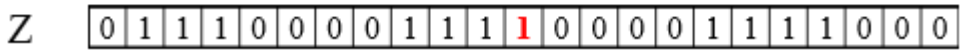
所以最后Y寄存器变为



所以最后得到的三个移位寄存器是







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：

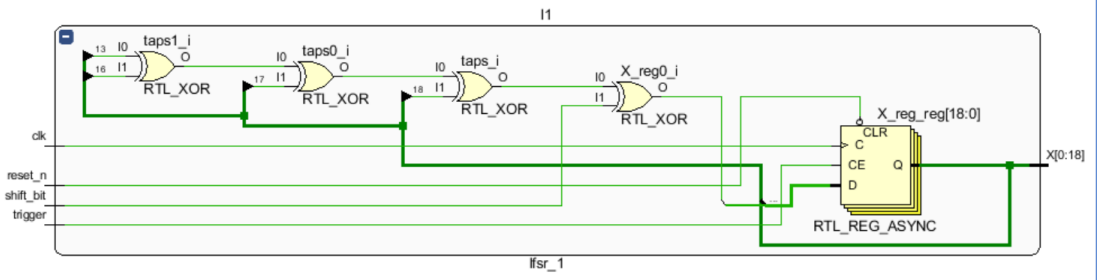
```
 module lfsr_1(
 input shift_bit,
   input trigger,
   input clk,
   input reset_n,
   output [0:18] X
   );
reg [0:18] X_reg=0;
wire taps;

   assign taps = X_reg[13]^X_reg[16]^X_reg[17]^X_reg[18];
   always @(posedge clk,negedge reset_n)
   begin
       if(~reset_n)
           X_reg <= 'b0;
       else if (trigger)
           X_reg <= {taps^shift_bit,X_reg[0:17]};
   end

assign X = X_reg;

```

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：

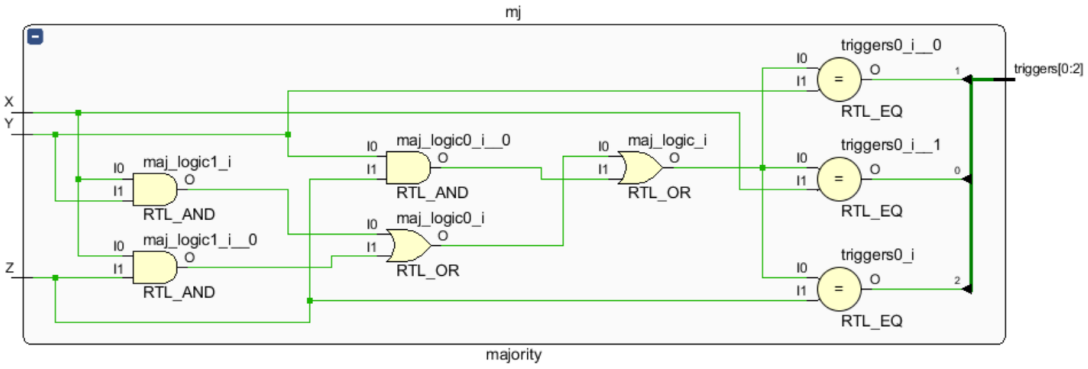
```
module majority(
    input X,
    input Y,
    input Z,
    output reg [0:2]triggers
    );
```

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 三个数出现最多的数。
    always @(X,Y,Z)
    begin
        if(maj_logic == X)
            triggers[0] <= 1'b1;
        else
            triggers[0] <= 1'b0;
        if(maj_logic == Y)
            triggers[1] <= 1'b1;
        else
            triggers[1] <= 1'b0;
        if(maj_logic == Z)
            triggers[2] <= 1'b1;
        else
            triggers[2] <= 1'b0;
    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：

```
module top(
input clk,
input [0:63]secret,
input rst,
output  init_ok,
output keystream
    );
```

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。

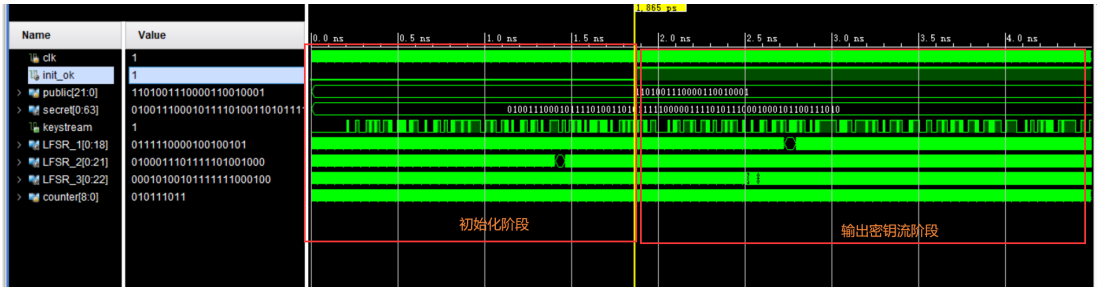
代码如下：

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

```
 wire [0:22]LFSR_3;
 reg [8:0]counter=8'b0;
 reg [2:0]reset=0;
 lfsr_1 l1(.shift_bit(shift_bit)
,.trigger(trigger[0]),.clk(clk),.reset_n(reset[0]),.X(LFSR_1));
 lfsr_2 l2(.shift_bit(shift_bit)
,.trigger(trigger[1]),.clk(clk),.reset_n(reset[1]),.Y(LFSR_2));
 lfsr_3 l3(.shift_bit(shift_bit)
,.trigger(trigger[2]),.clk(clk),.reset_n(reset[2]),.Z(LFSR_3));
 majority
mj(.X(LFSR_1[8]),.Y(LFSR_2[10]),.Z(LFSR_3[10]),.triggers(temp));
 always@(posedge clk)
 begin
 if(~rst)begin
 counter<=0;
 reset<=3'b000;
 init<=0;
 end
 end
always@(posedge clk)begin
if(counter==0)begin
shift_bit<=secret[0];
end
else if(counter<=63)begin
 reset<=3'b111;
 shift_bit<=secret[counter];
 trigger<=3'b111;
 end
else if(counter<=85) begin
shift_bit<=public[counter-64];
trigger<=3'b111;
end
else if(counter<=185)begin
shift_bit<=0;
trigger<=temp;
end
else   init<=1;
if(init==1)begin
shift_bit<=0;
trigger<=temp;
counter<=1;
end
counter<=counter+1;
 end
  assign keystream=LFSR_1[18]^LFSR_2[21]^LFSR_3[22];
  assign init_ok=init;
```

三、运行仿真结果

编写testbench执行仿真，



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。