实验报告

实验题目：**运算器与寄存器**　　日期：2019 年 3 月 22 日

姓名:廖洲洲　　学号:PB17081504　　成绩:_____

实验目的：

设计一个运算器与寄存器，使用设计的运算器与寄存器实现三个功能：

- 比较两个数的大小关系
- 求多个数的累加和
- 求给定两个初始数的斐波拉契数列

（一）ALU:

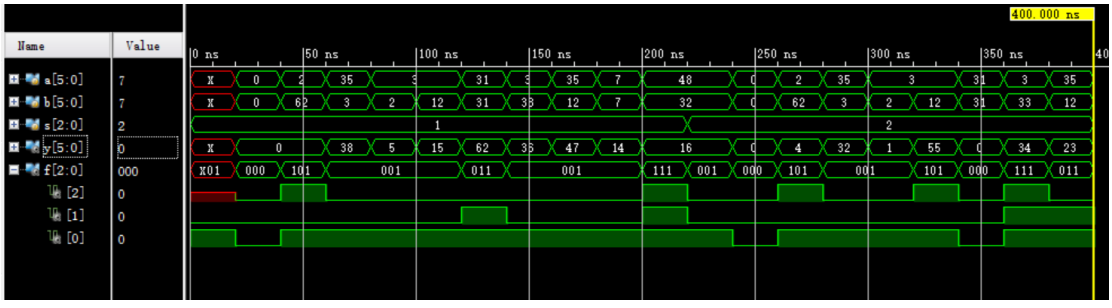
设计逻辑：设计了恒等、加、减、与、或、非、异或和判零功能。其中输出 f 是标志位，f[2]是进借位标志，f[1]是溢位标志，f[0]是判零标志。其中溢位标志的确定是按：当两个相同符号数相加，而运算结果的符号与原数据符号相反时，产生溢出。当两个相反符号位相减，若运算结果符号与被减数符号相反时，产生溢出。进借位标志则直接按运算结果的超前进位判定。

核心代码：

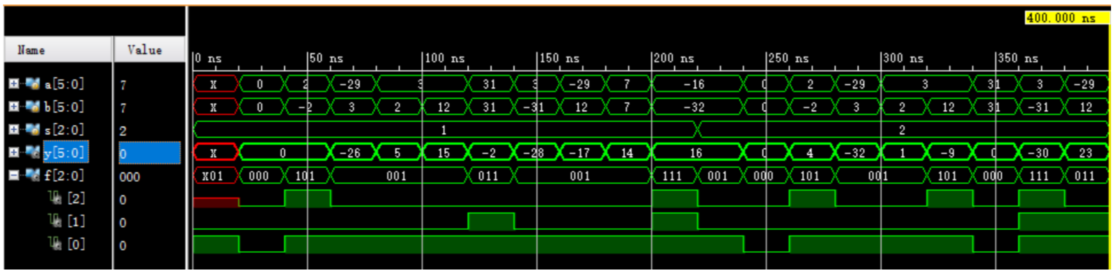
```
always @(a or b or s)
    begin
    f=3'b001;
    case(s)
    3'b000:y <= a;//==
    3'b001:
        begin
       {f[2], y} = a+b;//+  f[2]为进借位 加 减进借位时为 1
       //只有当两个相同符号数相加，而运算结果的符号与原数据符号相反时，产生溢出；f[1]为 1 时溢出
```

```
            if(a[5]==b[5] & y[5]==~a[5])
                f[1]=1'b1;
            else
                f[1]=1'b0;
            if(y==0 & f[2]==0 & f[1]==0)
                f[0]=1'b0;
            else
                f[0]=1'b1;
             end
        3'b010:
             begin
             {f[2],y} = a-b;//-
             //当两个相反符号位相减，若运算结果符号与被减数符号相反时，
产生溢出
             if(a[5] == ~b[5] & y[5] == ~a[5])
                 f[1]=1'b1;
             else
                 f[1]=1'b0;
             if(y==0 & f[2]==0 & f[1]==0)
                 f[0]=1'b0;
             else
                 f[0]=1'b1; //为什么加了 else 之后 f[0]会变为 1 了，若不
加都为 0
             end
        3'b011:if(a>0 & b>0) f <= 3'b1; else f <= 3'b0;//and      f[0]
为 0 表示为 0
        3'b100:if(a==0 & b==0) f <= 3'b0;else f <= 3'b1;//or
        3'b101:if(a>0) f <= 3'b0; else f <= 3'b1;//not
        3'b110:if(a==b) f <= 3'b0;else f <= 3'b1;//nor 异或
        3'b111:if(a==0) f <= 3'b0;else f <= 3'b1;//judge 0
        default: ;
        endcase
        end
```

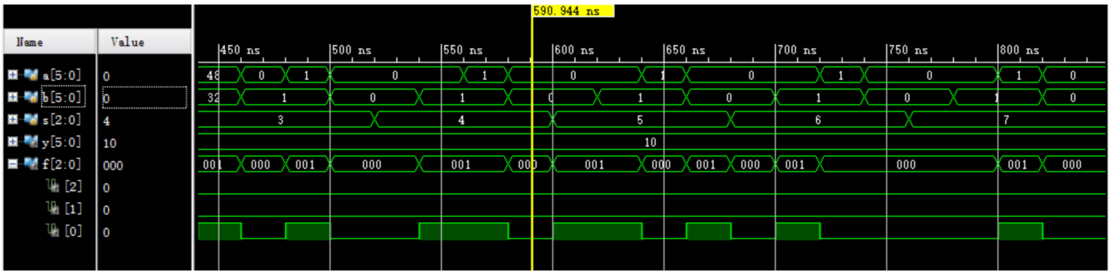
仿真截图：



无符号加、减 （2+62 48+32 进位（2-62 3-12 3-33）借位



有符号加、减 （31+31 -16+-32 3—31 -29-12）溢出



逻辑功能仿真　 与 或 非 异或

下载照片：



（二）Register:

核心代码

```
always@(posedge clk)
begin
if(rst)
    out=6'b0;
else if(en)
    out=in;
end
```

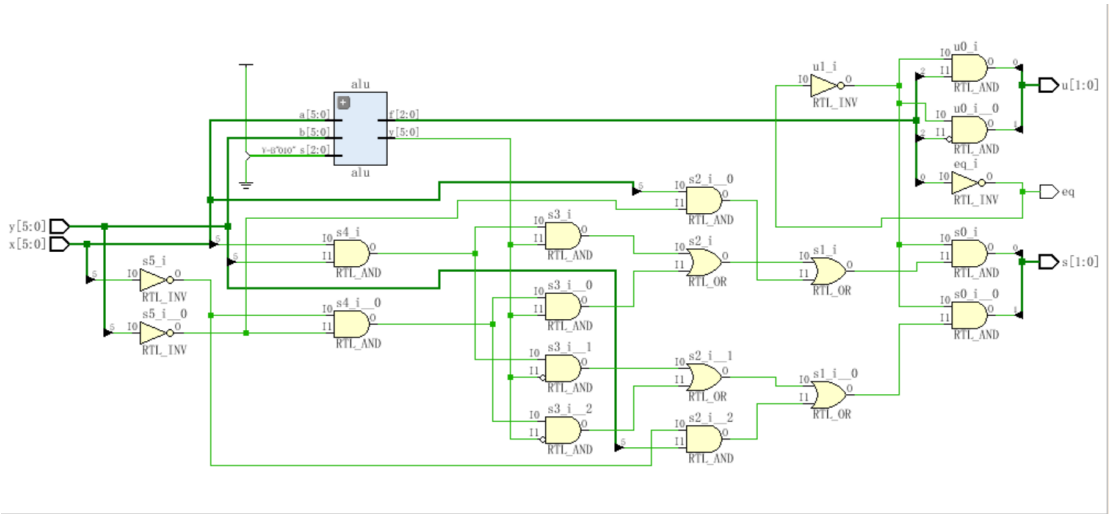
（三）CMP：

设计逻辑：利用 ALU 的减功能来判断大小。若输出标志判零位为 0（即减结果为 0），则令 eq 为 1，即两数相等。当 eq 为 0（即两数不等），对于无符号数 x-y，若无借位，则 x 更大，若有借位，则 x 更小。当 eq 为 0（即两数不等），对于有符号数 x-y，当 x、y 同号且 result>0，或 x 正 y 负时则 x 更大，当 x、y 同号且 result<0 或 x 为负 y 为正时则 x 更小。

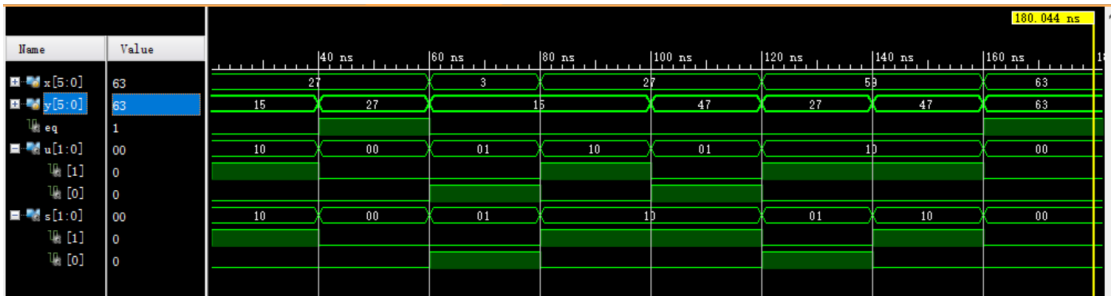
核心代码：

```
    reg [3:0]sel=3'b010;
    wire [5:0]result;
    wire [2:0]flag;

    alu alu(x,y,sel,result,flag);


    assign eq = ~flag[0]; //eq 为 1 相等
    assign u[1]= ~eq &~flag[2];//x 更大，未借位
    assign u[0]= ~eq &flag[2];//x 更小，借位了
    assign s[1]= ~eq & ((x[5]&y[5]&~result[5]) | (~x[5]&~y[5]&~result[5])
| (~x[5]&y[5]));//x 更大，当 xy 同号且 result>0，或 x 正 y、负
    assign s[0]= ~eq &( (x[5]&y[5]&result[5]) | (~x[5]&~y[5]&result[5])
| (x[5]&~y[5]) );
```

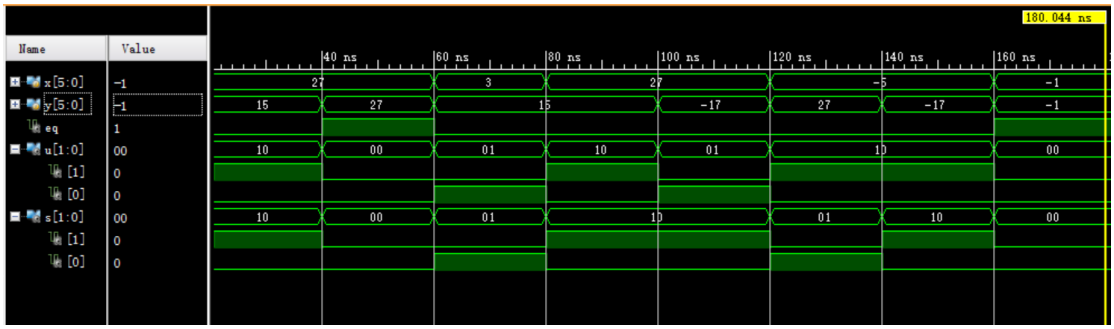
原理图：



仿真截图：



无符号数大小比较 如 27>15　27==27 3<15



有符号数大小比较 如 -5<27 -5>-17

下载照片

000011 大于 000001 1 00011 与 000001 有大（无符号）有小（有符号）

（四）ACM：

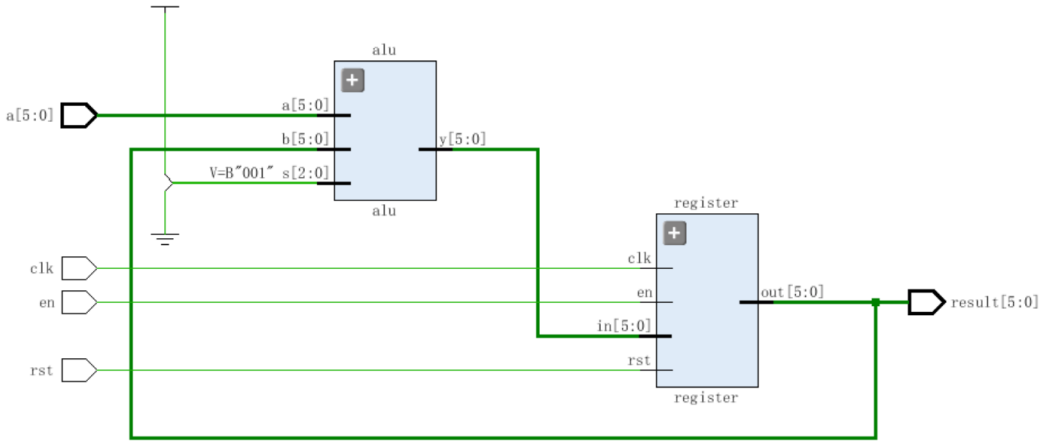
设计逻辑：累加的输入分时输入到 alu 的 a 输入中，同时将 alu 的结果输出输入到寄存器的输入中，并把寄存器的输出连接到 alu 的 b 输入中，寄存器开始时需通过 rst 置 0，即可实现累加。

核心代码：

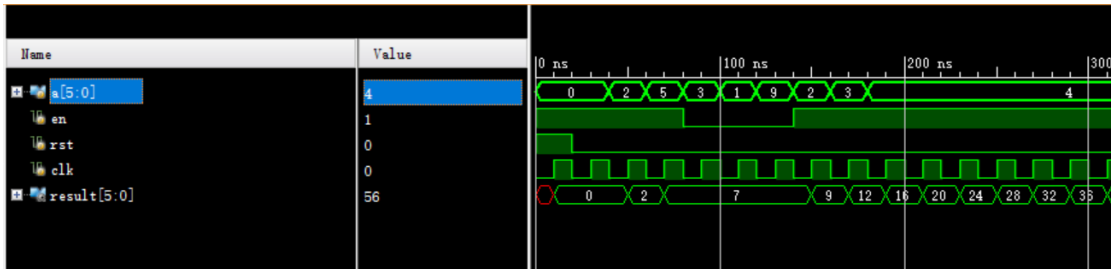
```
reg [3:0] sel=3'b001;
wire [5:0] y;
wire [5:0] b;
alu alu(.a(a),.b(b),.s(sel),.y(y));
register register(.in(y),.en(en),.rst(rst),.clk(clk),.out(b));

always@(*)
begin
    result=b;
```

end

原理图：



仿真截图：



下载照片：

（五）FIB：

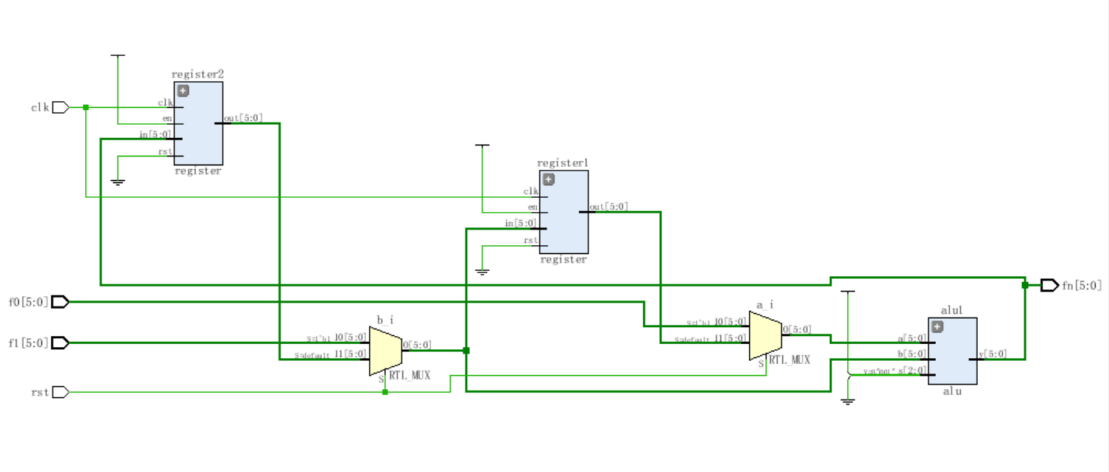
设计逻辑：使用两个寄存器和一个 alu 完成设计，两个寄存

器的输入分别接 alu 输出和其输入 b，输出分别借 alu 的 b 和 a，即可实现后一位输入与结果不断相加。即可计算斐波那契数列。

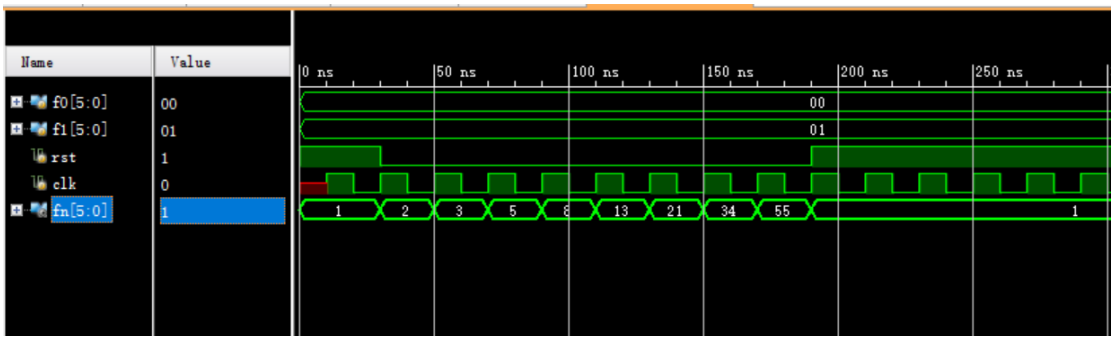
核心代码：

```
reg [2:0] s=3'b001;
reg grst=0;
reg gen=1;
wire [5:0]out1,out2,y;
reg [5:0] a,b;
alu alu1(.a(a),.b(b),.s(s),.y(y));
register register1(b,gen,grst,clk,out1);
register register2(y,gen,grst,clk,out2);
always@(*)
begin
    if(rst)
    begin
        a=f0;
        b=f1;
    end
    else
    begin
        a=out1;
        b=out2;
    end
end
assign fn=y;
```

原理图：



仿真截图：



0 1 1 2 3 5 8 13 21 34 55

下载照片：

1+1=2

1+2=3

2+3=5

5+3=8

8+5=13

实验总结

通过本次实验完成了 alu 和 register 的设计，同时利用这两个模块实现了 3 个功能。通过这次实验对进位、借位和溢位有了更深理解，同时在实验中发现了：

1.阻塞型赋值和非阻塞型赋值需根据不同情况下选择使用,不然可能会出错

2.注意a[5:0]与a[0:5]的区别,前者最高位是a[5],后者时还是a[0]

3.对于有符号数,当其是负数时硬件语言会视其为补码

源代码：

```
module alu(
    input [5:0] a,
    input [5:0] b,
    input [2:0] s,
    output reg[5:0] y,
    output reg[2:0] f
    );
```

```
initial
 begin
 f=3'b001;
 end

 always @(a or b or s)
     begin
     f=3'b001;
     case(s)
     3'b000:y <= a;//==
     3'b001:
         begin
         {f[2], y} = a+b;//+  f[2]为进借位 加 减进借位时为1
         //只有当两个相同符号数相加，而运算结果的符号与原数据符号相反时，产生溢出；f[1]为1时溢出
         if(a[5]==b[5] & y[5]==~a[5])
             f[1]=1'b1;
         else
             f[1]=1'b0;
         if(y==0 & f[2]==0 & f[1]==0)
             f[0]=1'b0;
         else
             f[0]=1'b1;
         end
     3'b010:
         begin
         {f[2],y} = a-b;//-
         //当两个相反符号位相减，若运算结果符号与被减数符号相反时，产生溢出
         if(a[5] == ~b[5] & y[5] == ~a[5])
             f[1]=1'b1;
         else
             f[1]=1'b0;
         if(y==0 & f[2]==0 & f[1]==0)
             f[0]=1'b0;
         else
             f[0]=1'b1; //为什么加了 else 之后 f[0]会变为 1 了，若不加都为0
         end
     3'b011:if(a>0 & b>0) f <= 3'b1; else f <= 3'b0;//and      f[0]为0表示为0
     3'b100:if(a==0 & b==0) f <= 3'b0;else f <= 3'b1;//or
     3'b101:if(a>0) f <= 3'b0; else f <= 3'b1;//not
     3'b110:if(a==b) f <= 3'b0;else f <= 3'b1;//nor 异或
```

```
        3'b111:if(a==0) f <= 3'b0;else f <= 3'b1;//judge 0
        default: ;
        endcase
        end



endmodule




module alu_tb(
    );
    reg [5:0] a,b;
    reg [2:0] s;
    wire [5:0] y;
    wire [2:0] f;
    alu alu(a,b,s,y,f);

    initial
        begin
        s=3'b001;
        #20  a=6'b000000;b=6'b000000;
        #20  a=6'b000010;b=6'b111110;
        #20  a=6'b100011;b=6'b000011;
        #20  a=6'b000011;b=6'b000010;
        #20  a=6'b000011;b=6'b001100;
        #20  a=6'b011111;b=6'b011111;
        #20  a=6'b000011;b=6'b100001;
        #20  a=6'b100011;b=6'b001100;
        #20  a=6'b000111;b=6'b000111;
        #20  a=6'b110000;b=6'b100000;
        #20  s=3'b010;
        #20  a=6'b000000;b=6'b000000;
        #20  a=6'b000010;b=6'b111110;
        #20  a=6'b100011;b=6'b000011;
        #20  a=6'b000011;b=6'b000010;
        #20  a=6'b000011;b=6'b001100;
        #20  a=6'b011111;b=6'b011111;
        #20  a=6'b000011;b=6'b100001;
        #20  a=6'b100011;b=6'b001100;
        #20  a=6'b000111;b=6'b000111;
        #20  a=6'b110000;b=6'b100000;
```

```
        #20  s=3'b011;
        #20  a=0;b=1;
        #20  a=1;b=1;
        #20  a=0;b=0;
        #20  s=3'b100;
        #20  a=0;b=1;
        #20  a=1;b=1;
        #20  a=0;b=0;
        #20  s=3'b101;
        #20  a=0;b=1;
        #20  a=1;b=1;
        #20  a=0;b=0;
        #20  s=3'b110;
        #20  a=0;b=1;
        #20  a=1;b=1;
        #20  a=0;b=0;
        #20  s=3'b111;
        #20  a=0;b=1;
        #20  a=1;b=1;
        #20  a=0;b=0;
        end
endmodule


module register(
    input [5:0] in,
    input en,
    input rst,
    input clk,
    output reg[5:0] out
    );
     always@(posedge clk)
     begin
     if(rst)
         out=6'b0;
     else if(en)
         out=in;
     end
endmodule


module CMP(
    input [5:0] x,
    input [5:0] y,
```

```
    output  eq,
    output  [1:0] u,  // uh ul
    output  [1:0] s   //sh sl
    );
    reg [3:0]sel=3'b010;
    wire [5:0]result;
    wire [2:0]flag;

    alu alu(x,y,sel,result,flag);


    assign eq = ~flag[0]; //eq 为 1 相等
    assign u[1]= ~eq &~flag[2];//x 更大，未借位
    assign u[0]= ~eq &flag[2];//x 更小，借位了
    assign s[1]= ~eq & ((x[5]&y[5]&~result[5]) | (~x[5]&~y[5]&~result[5])
| (~x[5]&y[5]));//x 更大，当 xy 同号且 result>0，或 x 正 y、负
    assign s[0]= ~eq &( (x[5]&y[5]&result[5]) | (~x[5]&~y[5]&result[5])
| (x[5]&~y[5]) );

endmodule


module CMP_tb(

    );
    reg [5:0] x;
    reg [5:0] y;
    wire eq;
    wire  [1:0] u;
    wire  [1:0] s;
    CMP CMP(x,y,eq,u,s);

    initial
       begin

       #20 x=6'b011011;y=6'b001111;
       #20 x=6'b011011;y=6'b011011;
       #20 x=6'b000011;y=6'b001111;
       #20 x=6'b011011;y=6'b001111;
       #20 x=6'b011011;y=6'b101111;
       #20 x=6'b111011;y=6'b011011;
       #20 x=6'b111011;y=6'b101111;
       #20 x=6'b111111;y=6'b111111;
       end
```

```
endmodule


module ACM(
    input [5:0] a,
    input en,
    input rst,
    input clk,
    output reg [5:0] result
    );

    reg [3:0] sel=3'b001;
    wire [5:0] y;
    wire [5:0] b;
    alu alu(.a(a),.b(b),.s(sel),.y(y));
    register register(.in(y),.en(en),.rst(rst),.clk(clk),.out(b));

    always@(*)
    begin
        result=b;
    end
endmodule

module ACM_tb(

    );
    reg [5:0] a;
    reg en,rst,clk;
    wire [5:0] result;

    ACM ACM(a,en,rst,clk,result);

    initial
        begin
            a=0;
            clk=0;
    forever
        begin
            #10 clk=1;
            #10 clk=0;
        end
        end
```

```
    initial
        begin
            rst=1;
            en=1;
            #20 rst=0;
            #20 a=2;
            #20 a=5;
            #20 a=3;en=0;
            #20 a=1;en=0;
            #20 a=9;
            #20 a=2;en=1;
            #20 a=3;
            #20 a=4;
       end
endmodule


module FIB(
    input [5:0] f0,
    input [5:0] f1,
    input rst,
    input clk,
    output [5:0] fn
    );
    reg [2:0] s=3'b001;
    reg grst=0;
    reg gen=1;
    wire [5:0]out1,out2,y;
    reg [5:0] a,b;
    alu alu1(.a(a),.b(b),.s(s),.y(y));
    register register1(b,gen,grst,clk,out1);
    register register2(y,gen,grst,clk,out2);
    always@(*)
    begin
        if(rst)
        begin
            a=f0;
            b=f1;
        end
        else
        begin
            a=out1;
            b=out2;
        end
```

```
    end
    assign fn=y;
endmodule


module FIB_tb(

    );
    reg [5:0] f0,f1;
    reg rst;
    reg clk;
    wire [5:0] fn;

    FIB FIB(f0,f1,rst,clk,fn);

   initial
        begin
            rst=1;
            f0=0;
            f1=1;
    forever
        begin
            #10 clk=1;
            #10 clk=0;
        end
     end

     initial
     begin
        #30 rst=0;
        #160 rst=1;
     end
endmodule
```