运算器及其应用

李卓 pb19000064

实验目的

掌握算术逻辑单元（ALU）的功能

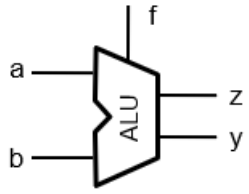
掌握数据通路和控制器的设计方法

掌握组合电路和时序电路，以及参数化和结构化的 Verilog 描述方法

了解查看电路性能和资源使用情况

实验原理

一， ALU

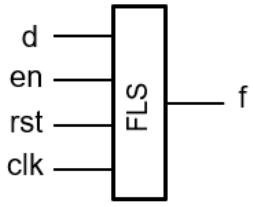


ALU 模块功能表

f	y	z
000	a + b	*
001	a - b	*
010	a & b	*
011	a | b	*
100	a ^ b	*
其他	0	1

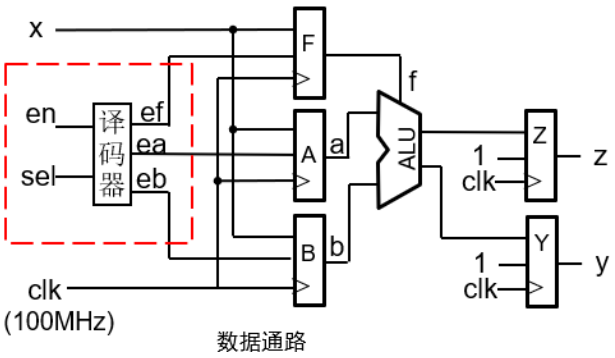
a b 为两个 32/6 位操作数，F 为操作功能数，y 为 32/6 结果, z 为 0 标志

二， FLS



en 为输入输出使能信号，rst 复位，d 输入数列初始项，f 输出数列

三， ALU 分时复用



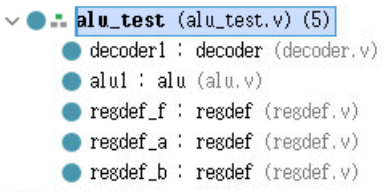
数据通路

操作数 a, b 和功能 f 复用开关输入 x[5:0]。方法：通过 sel 和 en ，生成译码电路，将开关输入 x[5:0]分时存入寄存器 F(x[2:0])，A(x[5:0])，B(x[5:0])

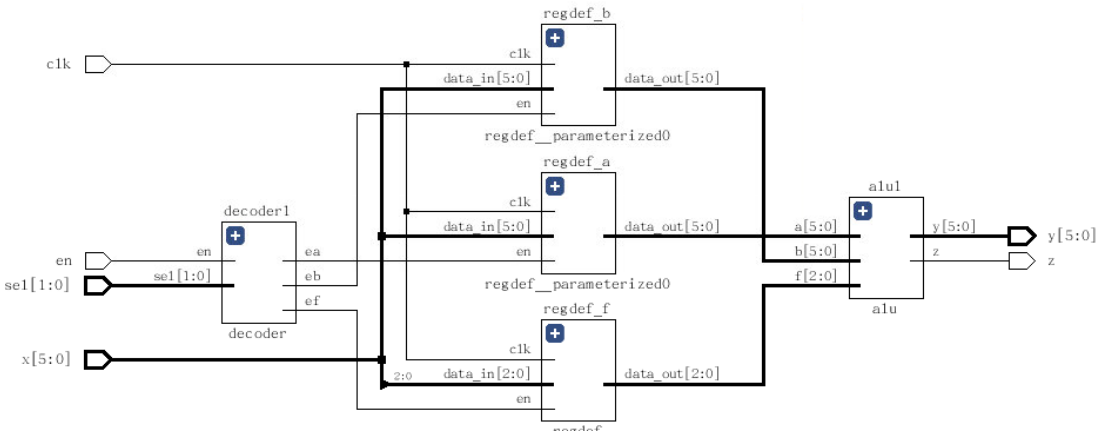
实验思路

一， ALU

文件结构如下



数据通路如下：



二，　FLS

一个寄存器 S 存储当前状态

状态转移图：s0->s1->s2->s3<->s4

s0 为初始态，复位后为 s0，s1 s2 接受输入的 d 值，

s3 s4 计算 F(n) F(n-1) F(n-2)的值

F(n-1) F(n-2)存储在寄存器 a b 中　输出 f=a+b

中间寄存器 tmp0 tmp1 存储 a b 相关的转移值

实验代码

1.

```
module alu#(
    parameter WIDTH = 6
)(
    input[WIDTH-1:0] a,b,
    input[2:0] f,
    output reg [WIDTH-1:0] y,
    output reg z
    );

always@(*)
begin
    case(f)
```

```
            3'b000: y=a+b;
            3'b001: y=a-b;
            3'b010: y=a&b;
            3'b011: y=a|b;
            3'b100: y=a^b;
            default: y=0;
        endcase
        if(!y)
            z=1;
        else
            z=0;
end

endmodule
```

2.

```
module alu_test#(
parameter WIDTH = 6
)(
input en,     //button
input clk,    //
```

```
input [1:0] sel,       //sw7~6
input [WIDTH - 1:0] x,    //sw5~0
output z,
output [WIDTH - 1:0] y
      );

wire ef;
wire ea,eb;
wire [2:0] a_f;
wire [WIDTH-1:0] a_a,a_b;

reg [31:0] num;
reg [2:0] cnt;
reg [3:0]hexplay_an;
reg [3:0]hexplay_data;

always @(posedge clk)
begin
      if(num[20]==1)
      begin
            cnt<=cnt+1;
            num<=0;
```

```
        end
        else
            num<=num+1;
end
always @(*)
begin
        case(cnt)
        3'b000: begin hexplay_an<=0;hexplay_data<=5;end
        3'b001: begin hexplay_an<=1;hexplay_data<=9;end
        3'b010: begin hexplay_an<=2;hexplay_data<=0;end
        3'b011: begin hexplay_an<=3;hexplay_data<=3;end
        3'b100: begin hexplay_an<=4;hexplay_data<=1;end
        3'b101: begin hexplay_an<=5;hexplay_data<=8;end
        3'b110: begin hexplay_an<=6;hexplay_data<=2;end
        3'b111: begin hexplay_an<=7;hexplay_data<=6;end
        endcase
end



decoder #(.WIDTH(2))decoder1(.en(en),.sel(sel),.ef(ef),.ea(ea),.eb(eb));


alu #(.WIDTH(6))alu1(.a(a_a),.b(a_b),.f(a_f),.y(y),.z(z));
```

```
regdef #(.WIDTH(3))regdef_f(.clk(clk),.en(ef),.data_in(x[2:0]),.data_out(a_f));

regdef #(.WIDTH(6))regdef_a(.clk(clk),.en(ea),.data_in(x),.data_out(a_a));

regdef #(.WIDTH(6))regdef_b(.clk(clk),.en(eb),.data_in(x),.data_out(a_b));


endmodule
```

3.

```
module decoder#(

parameter WIDTH = 2

)(

input en,

input [WIDTH-1:0] sel,

output reg ef,

output reg ea,

output reg eb

);


always@(*)

begin
```

```
    if(!en)
    begin
        ef = 0; ea = 0; eb = 0;
    end
    else
        case(sel)
            2'b10: begin ef=1; ea=0; eb=0; end
            2'b00: begin ef=0; ea=1; eb=0; end
            2'b01: begin ef=0; ea=0; eb=1; end
            default: begin ef=0; ea=0; eb=0; end
        endcase
end
endmodule
```

4.

```
module fls#(
parameter WIDTH = 7
)(
    input clk,
```

```
        input rst,
        input en,
        input [WIDTH-1:0] d,
        output [WIDTH-1:0] f
);

reg [WIDTH-1:0] a,b;
reg [WIDTH-1:0] tmp1,tmp2;
reg flag=0;
reg [2:0] curr_state;
reg [2:0] next_state;

parameter [2:0] ALU_ADD = 3'b000;
parameter S0=3'b000;
parameter S1=3'b001;
parameter S2=3'b010;
parameter S3=3'b011;
parameter S4=3'b100;

reg en_r1,en_r2;
wire en_edge;
always@(posedge clk)
```

```
    en_r1<=en;
always@(posedge clk)
    en_r2<=en_r1;
assign en_edge = en_r1&(~en_r2);

alu #(.WIDTH(7))alu1(.a(a),.b(b),.f(ALU_ADD),.y(f),.z());

always@(*)
begin
    case(curr_state)
    S0: next_state=S1;
    S1: next_state=S2;
    S2: next_state=S3;
    S3: next_state=S4;
    S4: next_state=S3;
    default: next_state=S0;
    endcase
end

always@(posedge clk)
begin
    if(!en_edge)
```

```
            flag<=0;
        else
            flag<=1;
end
always@(posedge clk)
begin
        if(rst)
            curr_state <= S0;
        else if(!flag)
                if(en_edge)
                    curr_state <=next_state;
end


always@(posedge clk)
begin
        if(en_edge&(!flag))
        begin
            case(curr_state)
            S0:begin a=0;b=0;tmp1=0;tmp2=0;end
            S1:begin a=d;b=0;tmp1=a;end
            S2:begin a=0;b=d;tmp2=b;end
            S3:begin a=tmp1;b=tmp2;tmp1=a+b;end
```

```
            S4:begin a=tmp1;b=tmp2;tmp2=a+b;end
            endcase
        end
end
endmodule
```