**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

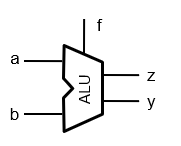
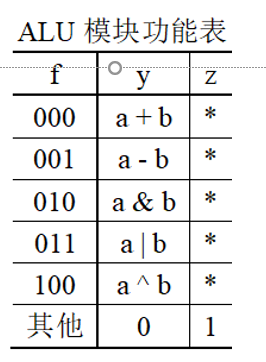
掌握数据通路和控制器的设计方法

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

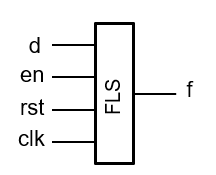
**实验原理**

1. ALU

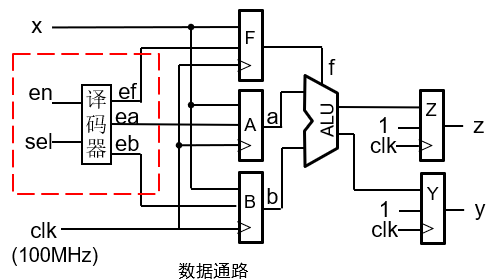
a b为两个32/6位操作数，F 为操作功能数，y为32/6结果, z为0标志

1. 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

1. ALU分时复用

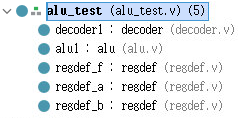


操作数a, b和功能f 复用开关输入x[5:0]。方法：通过sel和en ，生成译码电路，将开关输入x[5:0]分时存入寄存器 F(x[2:0])，A(x[5:0])，B(x[5:0])

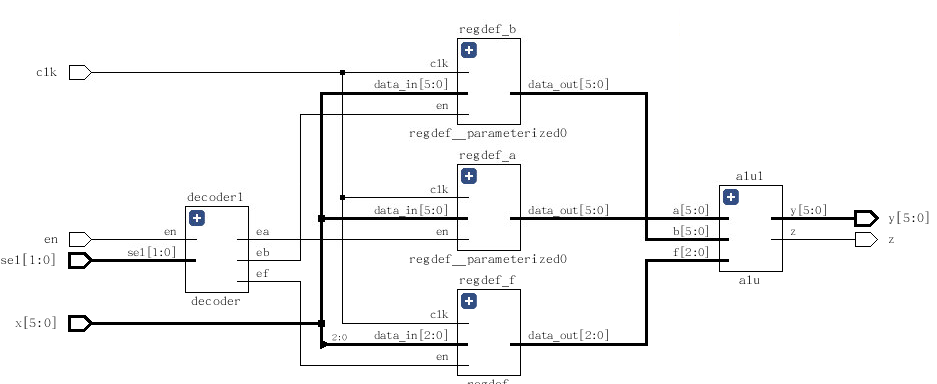
**实验思路**

1. ALU

文件结构如下



数据通路如下：



1. 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