实验一 运算器与排序

noname

2020 年 5 月 5 日

1 实验目的

设计一算术逻辑运算单元（ALU），实现 $+, -, \&, |, \wedge$ 功能, 并利用前述的 ALU 模块与适当的硬件电路, 完成四个有符号数的排序功能。

2 逻辑设计

2.1 运算器 (ALU)

编写 ALU 代码并进行功能仿真

这里的 ALU 没有添加 sf 标志符,sf 即为 $y[N-1]$

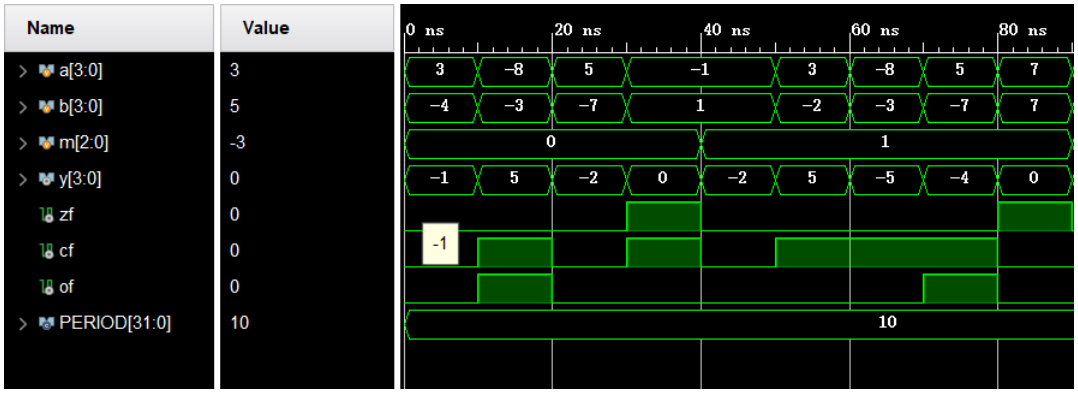

图 1: 有符号数算术运算功能仿真结果

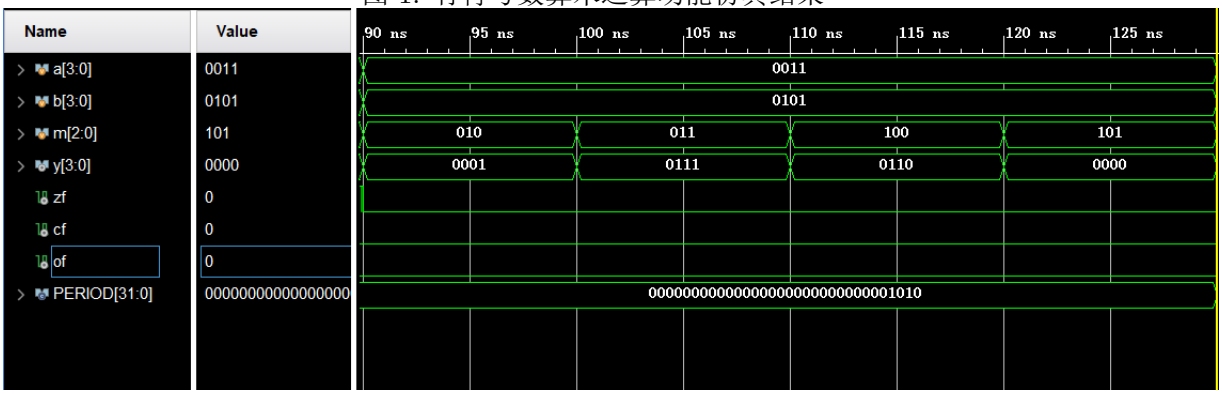

图 2: 有符号数算术逻辑功能仿真结果

2.2 排序器

2.2.1 数据通路与状态图

画出数据通路与状态图 如图所示:

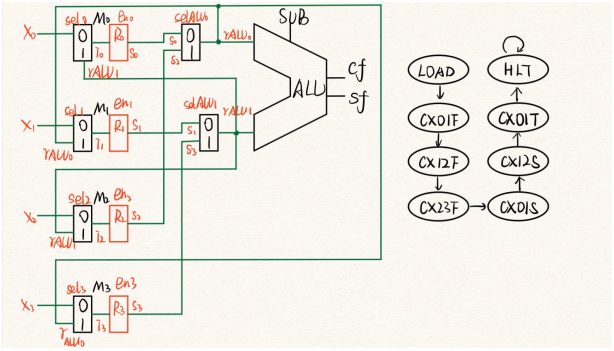

图 3: 排序电路 数据通路

状态	m_0	m_1	m_2	m_3	$mALU_0$	$mALU_1$	en_0	en_1	en_2	en_3
LOAD	0	0	0	0	X	X	1	1	1	1
CX01F	1	1	X	X	0	0	X<Y	X<Y	0	0
CX12F	X	1	1	X	1	0	0	X≥Y	X≥Y	0
CX23F	X	X	1	1	1	1	0	0	X<Y	X<Y
CX01S	1	1	X	X	0	0	X<Y	X<Y	0	0
CX12S	X	1	1	X	1	0	0	X≥Y	X≥Y	0
CX01T	1	1	X	X	0	0	X<Y	X<Y	0	0
HLT	done = 1									

图 4: 状态图

2.2.2 实现所需的寄存器、多选器

代码较为简单且老师已给出, 作为附件, 实验报告内不展示

2.2.3 仿真结果

如图所示:

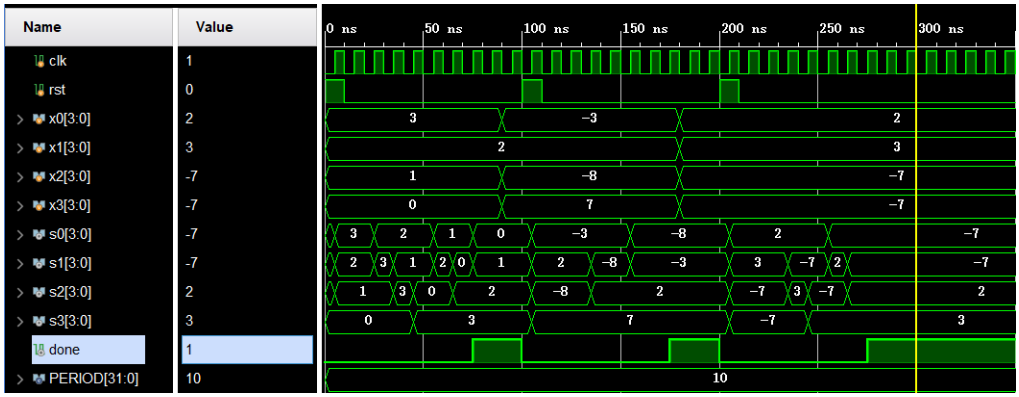

图 5: 排序电路仿真结果

3 实验结果

经仿真检验,ALU 与排序器的功能均正确实现。没有 FPGA, 无法检验在 FPGA 上的正确性

4 思考题

4.1 如果要求排序后的数据是递减顺序，电路如何调整？

若要求排序后数据为递减顺序, 只需将排序电路中需要用到的寄存器使能信号取反即可

4.2 如果为了提高性能，使用两个 ALU，电路如何调整？

首先对 x0,x1 和 x2,x3 排序使得 x0>x1,x2>x3 此时最小值在 x1,x3 中，最大值在 x0,x2 中
再对 x0,x2 和 x1,x3 排序使得 x0 为最大值，x3 为最小值
最后排序 x1,x2 使得 $x0 \geqslant x1 \geqslant x2 \geqslant x3$
新的排序电路如下:

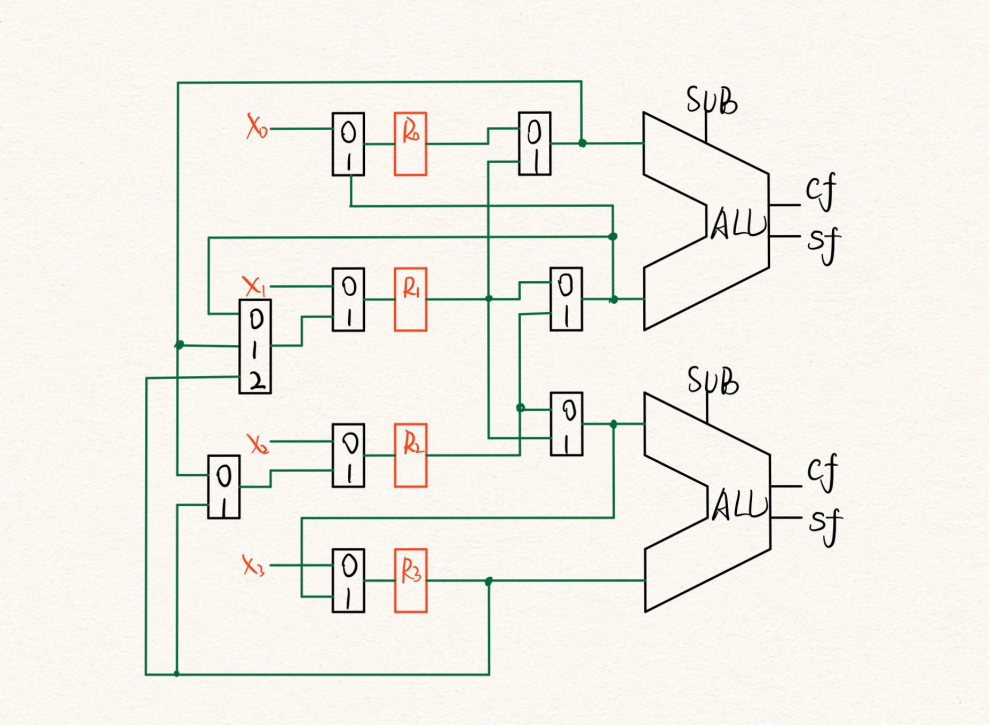

图 6: 新的排序电路

5 意见与建议

本次实验较为简单，主要是让学生复习 Verilog 的相关知识，为后面的实验打下了良好的基础

6 附件

```
module ALU
#(parameter N = 4)
(
input [N-1:0] a,b,
input [2:0] m,

output reg [N-1:0] y,
output reg cf,of,zf,sf
);

always@(*)
begin
case(m)
3'b000:
begin
{cf,y} = a+b;
of = (~a[N-1] & ~b[N-1] & y[N-1]) | (a[N-1] & b[N-1] & ~y[N-1]);
zf = ~|y; sf = y[N-1];
end
3'b001:
begin
{cf,y} = a-b;
of = (~a[N-1] & b[N-1] & y[N-1]) | (a[N-1] & ~b[N-1] & ~y[N-1]);
zf = ~|y; sf = y[N-1];
end
3'b010:
begin
{cf,y} = a&b;
of = 0;
zf = ~|y; sf = y[N-1];
end
3'b011:
begin
{cf,y} = a|b;
of = 0;
zf = ~|y; sf = y[N-1];
end
3'b100:
begin
{cf,y} = a^b;
of = 0;
zf = ~|y; sf = y[N-1];
end
default:
begin
y = 4'b0000;
```

```
                cf = 0;
                of = 0;
                zf = 0;sf = y[N-1];
                end
                endcase
                end
                endmodule
```

```
                module sort
                #(parameter N = 4,
                LOAD = 3'b000,                                          //加载阶段
                CX01F = 3'b001,CX12F = 3'b010,CX23F = 3'b011,           //第一次排序
                CX01S = 3'b100,CX12S = 3'b101,                          //第二次排序
                CX01T = 3'b110,                                         //第三次排序
                HLT = 3'b111)
                (
                input [N-1:0]x0,x1,x2,x3,
                input  clk,rst,
                output [N-1:0] s0,s1,s2,s3,
                output reg done
                );


                reg     [2:0]CUR_STATE,NEXT_STATE;
                reg     en0,en1,en2,en3,                    //寄存器使能
                sel0,sel1,sel2,sel3,                //寄存器前多选器信号
                sel_ALU_0,sel_ALU_1;                //ALU前多选器信号

                wire    [N-1:0] i0,i1,i2,i3, i_ALU_0,i_ALU_1;       //寄存器及ALU数
                    据
                wire    of,sf,cf;                           //ALU标志符

                //DataPath
                register    R0 (.data(i0), .en(en0), .clk(clk), .q(s0)),
                R1 (.data(i1), .en(en1), .clk(clk), .q(s1)),
                R2 (.data(i2), .en(en2), .clk(clk), .q(s2)),
                R3 (.data(i3), .en(en3), .clk(clk), .q(s3));

                ALU #(N)  alu (.a(i_ALU_0), .b(i_ALU_1), .m(CX01F), .of(of), .cf(
                    cf), .sf(sf));

                mux #(N)  M0 (.s(sel0), .a(x0), .b(i_ALU_1), .out(i0)),
                M1 (.s(sel1), .a(x1), .b(i_ALU_0), .out(i1)),
                M2 (.s(sel2), .a(x2), .b(i_ALU_1), .out(i2)),
                M3 (.s(sel3), .a(x3), .b(i_ALU_0), .out(i3)),

                M_ALU_0 (.s(sel_ALU_0), .a(s0), .b(s2), .out(i_ALU_0)),
                M_ALU_1 (.s(sel_ALU_1), .a(s1), .b(s3), .out(i_ALU_1));
```

```
92
93                  //////////Control Unit//////////
94                  always @(posedge clk or posedge rst)
95                  if (rst) CUR_STATE <= LOAD;
96                  else
97                  CUR_STATE <= NEXT_STATE;
98
99                  ////////////////////////////////
100                 always @(*)
101                 begin
102                 case (CUR_STATE)
103                 LOAD: NEXT_STATE <= CX01F;
104                 CX01F: NEXT_STATE <= CX12F;
105                 CX12F: NEXT_STATE <= CX23F;
106                 CX23F: NEXT_STATE <= CX01S;
107                 CX01S: NEXT_STATE <= CX12S;
108                 CX12S: NEXT_STATE <= CX01T;
109                 CX01T: NEXT_STATE <= HLT;
110                 default: NEXT_STATE <= HLT;
111                 endcase
112                 end
113
114                 ////////////////////////////////
115                 always @(*)
116                 begin
117                 case (CUR_STATE)
118                 LOAD: begin{sel0,sel1,sel2,sel3,en0,en1,en2,en3,done} = 9'
                        b000011110;end
119                 CX01F, CX01S, CX01T:
120                 begin
121                 sel0 = 1;sel1 = 1;
122                 sel_ALU_0 = 0;sel_ALU_1 = 0;
123                 //不加~号，无符号从大到小、有符号从小到大
124                 //en0 = cf; en1 = cf;                                    //无符号数
125                 en0 = (sf&of)|(~sf&~of);en1 = (sf&of)|(~sf&~of);   //有符号数
126                 en2 = 0;en3 = 0;
127                 end
128                 CX12F, CX12S:
129                 begin
130                 sel1 = 1;sel2 = 1;
131                 sel_ALU_0 = 1;sel_ALU_1 = 0;
132                 //en1 = ~cf; en2 = ~cf;                                  //无符号数
133                 en1 = ~((sf&of)|(~sf&~of));en2 = ~((sf&of)|(~sf&~of));     //有符
                        号数
134                 en0 = 0;en3 = 0;
135                 end
136                 CX23F:
137                 begin
```

```
138                 sel2 = 1;sel3 = 1;
139                 sel_ALU_0 = 1;sel_ALU_1 = 1;
140                 //en2 = cf; en3 = cf;                                   //无符号数
141                 en2 = (sf&of)|(~sf&~of);en3 = (sf&of)|(~sf&~of);   //有符号数
142                 en0 = 0;en1 = 0;
143                 end
144                 HLT:
145                 begin
146                 done = 1;
147                 {en0,en1,en2,en3} = 4'b0000;
148                 end
149                 default: {sel0,sel1,sel2,sel3, sel_ALU_0,sel_ALU_1,en0,en1,en2,
                        en3} = 10'b0;
150                 endcase
151                 end
152                 //assign s0 = r0,s1 = r1,s2 = r2,s3 = r3;
153                 endmodule
```

```
154                 module register
155                 #(parameter N = 4)
156                 (    input clk,en,
157                 input [N-1:0] data,
158                 output reg [N-1:0]q = 0
159                 );
160
161                 always@(posedge clk)
162                 begin
163                 if(en)
164                 q <= data;
165                 end
166
167                 endmodule
```

```
168                 module mux
169                 #(parameter N = 4)
170                 (
171                 input [N-1:0]a,b,
172                 input s,
173                 output [N-1:0] out
174                 );
175
176                 assign out = (s==0 ? a : b);
177
178                 endmodule
```