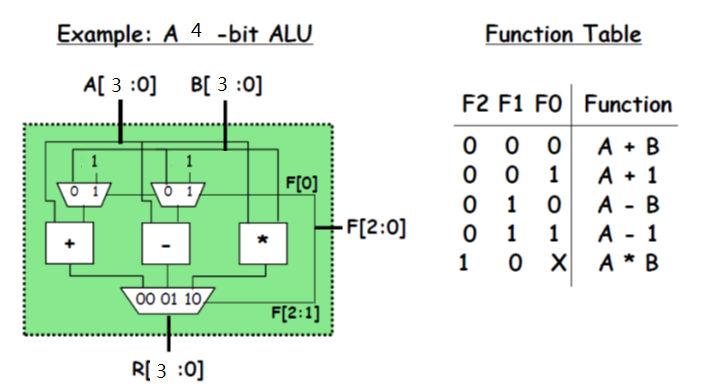
**实验报告 Lab 2**

**任务** 设计具有以下功能的ALU：



（1）设计（4比特）2-1选择器，3-1选择器，加法器，减法器，乘法器（不考虑溢出）。

（2）基于（1）中的模块，设计ALU的顶层结构。

（3）在FPGA板上实现该电路。其中数据输入A[3:0],B[3:0],控制输入F[2:0]连接开关，输出R[3:0]连接到LED。

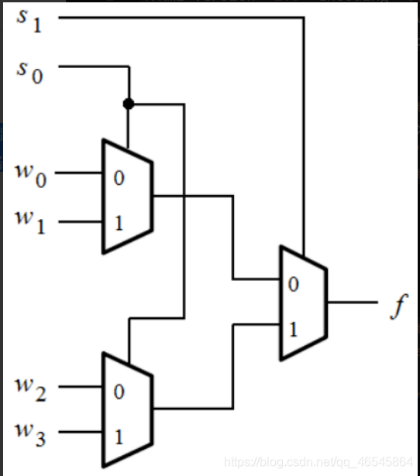
（4）（选做）解决溢出问题。

（5）（选做）增加新功能。

**一、模块准备**

1、4比特2-1选择器

代码实现与Lab 1中的选择器大体相同，不过输入和输出都需要扩充成四位，并在约束文件里对每个输入/输出量的每一位指定好管脚，否则比特流跑不通。



*w*2

2、4比特3-1选择器

可以通过对4-1选择器做减法来得到（当然要先扩充位数）。如图所示，将左下角的选择器去掉，当S1置1时，W2启用，此时无论S0置0或1都不起作用，完美契合给定的真值表。

3、4比特加法器

先设计全加器，给出全加器逻辑表达式：

编写模块:

module fulladder(

input A,

input B,

input Ci,

output S,

output Co

);

assign S=A^B^Ci;

assign Co=(A & B)| ((A|B) & Ci);

endmodule

接下来设计4比特加法器，直接将四个全加器进行级联，同时需要三个进位的中间变量，以及最低位进位输入和最高位进位输出，代码如下：

module adder(

input [3:0]A,

input [3:0]B,

input Ci,

output [3:0]S,

output Co

);

wire C1,C2,C3;

fulladder FA0(.A(A[0]),.B(B[0]),.Ci(Ci),.S(S[0]),.Co(C1));

fulladder FA1(.A(A[1]),.B(B[1]),.Ci(C1),.S(S[1]),.Co(C2));

fulladder FA2(.A(A[2]),.B(B[2]),.Ci(C2),.S(S[2]),.Co(C3));

fulladder FA3(.A(A[3]),.B(B[3]),.Ci(C3),.S(S[3]),.Co(Co));

endmodule

更近一步地，直接让输入A和B进行算术运算，这将大大简化代码（后面乘法器使用了这种写法），如：

module adder(

input [3:0]A,

input [3:0]B,

output reg [4:0]S

);

always @(\*)

begin S = A + B;

end

endmodule

4、4比特减法器

减法运算相当于减数取补码（按位取反再加1）再与被减数相加。减法器当然也可以由加法器来实现，不过输入得改一改，如：

module subtracter(

input [3:0]A,

input [3:0]B,

output [3:0]D,

output Co

);

wire [3:0]Bcom;

assign Bcom=(~B+1);

adder FA(.A(A),.B(Bcom),.S(D),.Ci(0),.Co(Co));

endmodule

但第五位的溢出很难处理，哪怕是1111-0001=1000这个算式，这个加法器会给出11100这种结果，第五位出现了意向不到的结果。还记得刚才加法器写出的简单写法吗？用在这里也是类似的，如：

module subtracter(

input [3:0]A,

input [3:0]B,

output reg [4:0]D

);

always @(\*)

begin D = A - B;

end

endmodule

5、4比特乘法器

就A[3:0]×B[3:0]而言，对于Bi（i=0，1，2，3），若Bi=1，则这一位乘法结果为A3A2A1A0与i个0相结合；若Bi=0，则这一位乘法结果为0，这一原理和按位与非常相像，因为，所以只要让A3A2A1A0分别和BiBiBiBi按位与，再在后面添上相应个数的0就可以了。然后将这四部分和相加即得最终的乘法结果。

module multiplier(

input [3:0]A,

input [3:0]B,

output reg [7:0]P

);

wire [7:0] pp [3:0];

assign pp[0] = {4'b0000, A & {4{B[0]}}}; //

assign pp[1] = {3'b000, A & {4{B[1]}}, 1'b0}; //

assign pp[2] = {2'b00, A & {4{B[2]}}, 2'b00}; //

assign pp[3] = {1'b0, A & {4{B[3]}}, 3'b000};

always @(\*)

begin P = pp[0] + pp[1] + pp[2] + pp[3];

end

endmodule

这里索性将乘法器输出设成了8位（省的考虑溢出问题）。

**二、顶层结构设计**

根据任务给出的电路图，可以写出顶层结构的代码：

module ALU(

input [3:0]A,

input [3:0]B,

input [2:0]F,

output [7:0]R

);

wire [3:0]B0;

mux2 mux21(.A(B),.B(1),.sel(F[0]),.Y(B0));

wire [3:0]B1;

mux2 mux22(.A(B),.B(1),.sel(F[0]),.Y(B1));

wire [4:0]Sout;

adder adder0(.A(A),.B(B0),.S(Sout));

wire [4:0]Dout;

subtracter subtracter0(.A(A),.B(B1),.D(Dout));

wire [7:0]Pout;

multiplier multiplier0(.A(A),.B(B),.P(Pout));

mux3 mux30(.A(Sout),.B(Dout),.C(Pout),.S1(F[1]),.S2(F[2]),.Y(R));

endmodule

板子现象如下：

|  |  |
| --- | --- |
|  |  |
| F0=[0,0,0],0000+0000=0000 | F0=[0,0,0],0000+0001=0001 |
|  |  |
| F0=[0,0,0],0001+0001=0010 | F0=[0,0,0],0011+0001=0100 |
|  |  |
| F0=[0,0,0],0111+0001=1000 | F0=[0,0,0],1111+0001=10000 |
|  |  |
| F0=[0,0,1],A+1,B不起作用 | F0=[0,0,0],1011+0110=10001 |
|  |  |
| F0=[0,1,0],0000-0001=11111(-1) | F0=[0,1,0],0101+0010=0011 |
|  |  |
| F0=[0,1,0],0111-0010=0101 | F0=[0,0,0],0011+0001=0100 |
|  |  |
| F0=[0,0,0],0111+0001=1000 | F0=[0,1,1],A-1,B不起作用 |
|  |  |
| F0=[0,1,1],0010-0001=0001 | F0=[0,1,1],1101-0001=1100 |

|  |  |
| --- | --- |
|  |  |
| F0=[1,0,0],0000×1111=0000 | F0=[1,0,0],0001×1111=1111 |
|  |  |
| F0=[1,0,0],0111×1111=1101001 | F0=[1,0,0],0011×1111=101101 |
|  |  |
| F0=[1,0,0],1111×1111=11100001 |  |