实验三 算术逻辑单元设计

**一、实验目的**

1 掌握Verilog语言和Vivado、Logisim开发平台的使用；

2 掌握算术逻辑单元的设计和测试方法。

**二、实验内容**

1 运算模块的设计与测试

2 算术逻辑单元设计与测试

**三、实验要求**

1 掌握Vivado或Logisim开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog代码/电路图/表达式/真值表，Vivado仿真结果，Logisim验证结果等）；

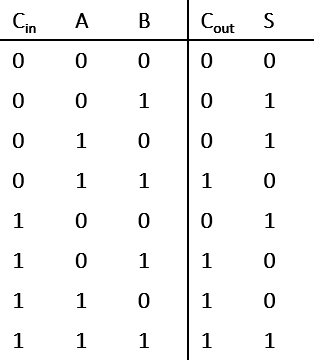
3 分析Vivado仿真波形/Logism验证结果，注重输入输出之间的对应关系。

**四、实验过程及分析**

1 加法器模块

一位全加器的输入和输出关系为：

-->



输入为两个加数A，B，以及低位的进位Cin,输出本位和S以及向高位的进位Cout。可以看出，只要三个输入中2个以上为1，则产生进位，Cout输出为1（这可以用三个输入两两相与再将三个结果或实现）。对于本位和S，三个输入中有1个或3个为1时，输出1（这可以用三个输入与或来实现）。设计代码：

module adder\_1bit(

Cin,A,B,C,Cout

);

input wire A,B,Cin;

output wire C,Cout;

assign C=A^B^Cin;

assign Cout=(A&B)|(A&Cin)|(B&Cin);

endmodule

一位全加器通过级联可以实现多位串行加法器。具体实现方法是，低位的全加器输出的进位值作为高一位全加器的进位输入值，这样逐级连接的最高位向外输出进位值，最低位接受一个外来低位进位输入值。这可以用模块化设计自底向上逐级封装实现。（这里先组成8位加法器，再由8位加法器级联成32位加法器）。设计代码：

module adder\_8bit(

Cin,A,B,C,Cout

);

input Cin;

input [7:0] A,B;

output wire Cout;

output wire[7:0]C;

wire [7:0] carry;

adder\_1bit add0(Cin,A[0],B[0],C[0],carry[0]);

adder\_1bit add1(carry[0],A[1],B[1],C[1],carry[1]);

adder\_1bit add2(carry[1],A[2],B[2],C[2],carry[2]);

adder\_1bit add3(carry[2],A[3],B[3],C[3],carry[3]);

adder\_1bit add4(carry[3],A[4],B[4],C[4],carry[4]);

adder\_1bit add5(carry[4],A[5],B[5],C[5],carry[5]);

adder\_1bit add6(carry[5],A[6],B[6],C[6],carry[6]);

adder\_1bit add7(carry[6],A[7],B[7],C[7],carry[7]);

assign Cout=carry[7];

endmodule

上面实现8位加法器。

module adder\_8bit(

Cin,A,B,C,Cout

);

input Cin;

input [7:0] A,B;

output wire Cout;

output wire[7:0]C;

wire [7:0] carry;

adder\_1bit add0(Cin,A[0],B[0],C[0],carry[0]);

adder\_1bit add1(carry[0],A[1],B[1],C[1],carry[1]);

adder\_1bit add2(carry[1],A[2],B[2],C[2],carry[2]);

adder\_1bit add3(carry[2],A[3],B[3],C[3],carry[3]);

adder\_1bit add4(carry[3],A[4],B[4],C[4],carry[4]);

adder\_1bit add5(carry[4],A[5],B[5],C[5],carry[5]);

adder\_1bit add6(carry[5],A[6],B[6],C[6],carry[6]);

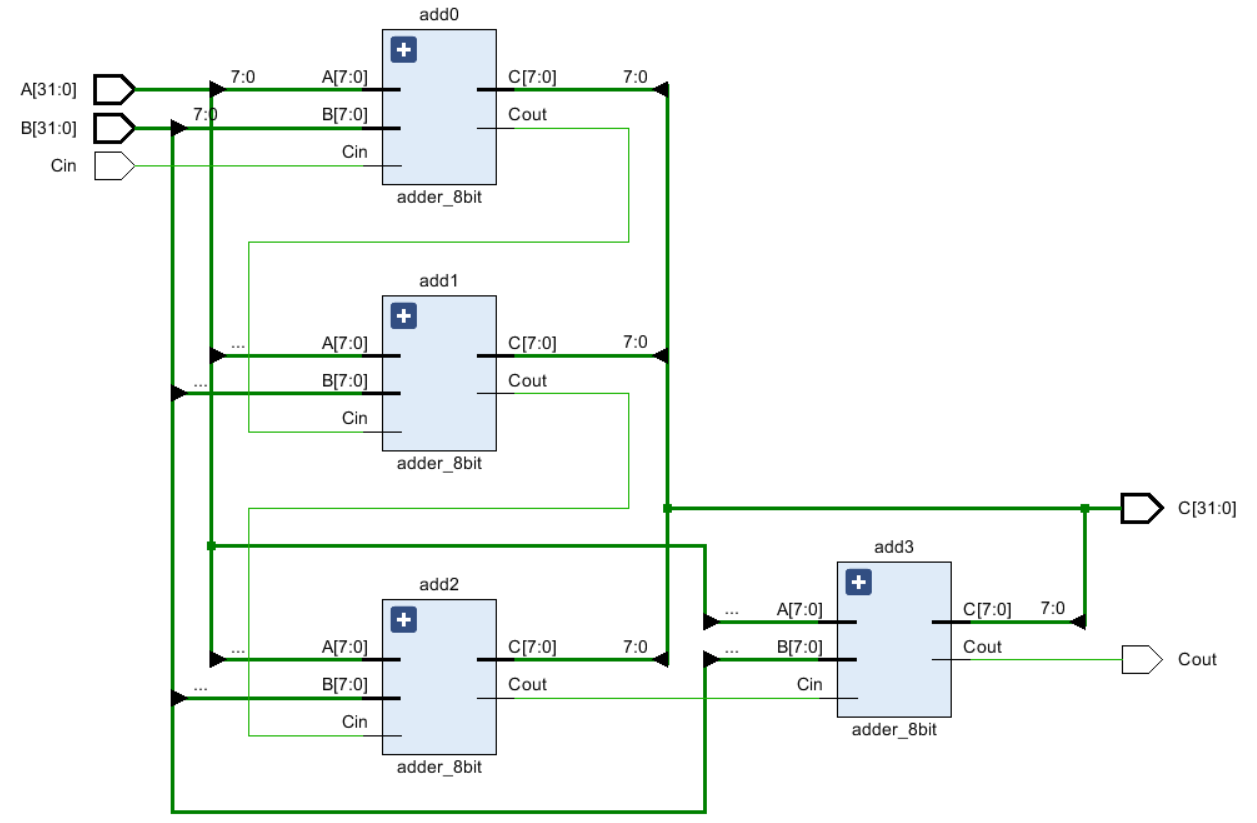
adder\_1bit add7(carry[6],A[7],B[7],C[7],carry[7]);

assign Cout=carry[7];

endmodule

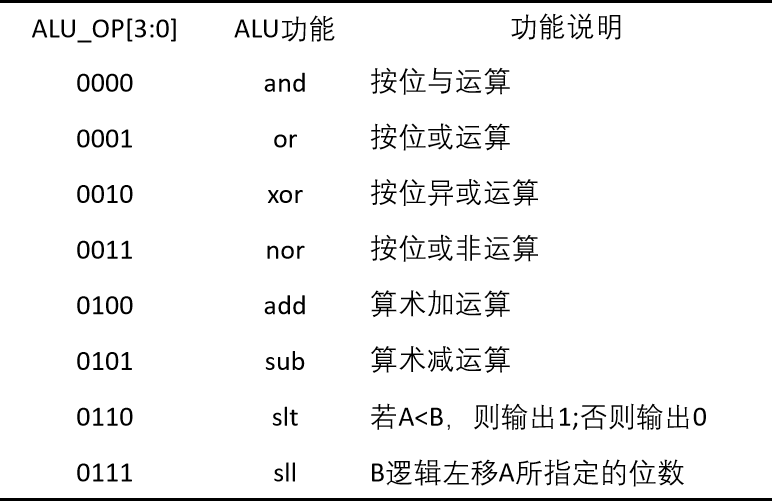
上面实现32位加法器。

RTL电路图：



2 算术逻辑单元

8功能32位ALU，通过4根控制线ALU\_OP[3:0]来选择其8种功能，功能见表所示。-->



采用模块化的设计方式，顶层模块代码：

module ALU\_8(

F,CF,A,B,OP

);

parameter size=32;

output reg [size-1:0]F;//output result

output CF;//ouput carry flag

input [size-1:0] A,B;

input [3:0]OP;//select function

parameter ALU\_AND=4'b0000;

parameter ALU\_OR=4'b0001;

parameter ALU\_XOR=4'b0010;

parameter ALU\_NOR=4'b0011;

parameter ALU\_ADD=4'b0100;

parameter ALU\_SUB=4'b0101;

parameter ALU\_SLT=4'b0110;//compare,if A<B,output1;otherwise 0

parameter ALU\_SLL=4'b0111;//B shift left A bits

wire [7:0] EN;

wire [size-1:0]Fw,Fa;

assign Fa=A&B;

always @(\*) begin

case(OP)

ALU\_AND:F<=Fa;

ALU\_OR:F<=A|B;

ALU\_XOR:F<=A^B;

ALU\_NOR:F<=~(A|B);

default:F=Fw;

endcase

end

Decoder38 decoder38\_1(OP[2:0],EN);

ADD add\_1(Fw,CF,A,B,EN[4]);

SUB sub\_1(Fw,CF,A,B,EN[5]);

SLT slt\_1(Fw,A,B,EN[6]);

SLL sll\_1(Fw,A,B,EN[7]);

Endmodule

输入32位加数A，B以及4位功能选择信号OP，ALU根据OP执行对应的功能计算。对于所有的按位逻辑运算、移位运算以及加减运算，都会输出一个32位计算结果F；除此之外，在加减法运算中，如果出现进位或者借位，还会输出标志位CF=1，否则输出0.由于CF并不是在每个功能中都要被用到，因此当不需要输出CF时期被置高阻态。这里所有8个功能中，前四个按位逻辑运算用case语句描述对应的输出结果。对于其他四个功能，则统一定义中间变量Fw作为输出，这四个功能分别用模块调用的方式实现，四个子模块的选择信号由38译码器根据OP的低三位译码来获得选择信号。

38译码器模块：

module Decoder38(

OP,EN

);

input wire[2:0] OP;

output reg[7:0] EN;

always @(OP) begin

case(OP)

3'b000:EN<=8'b0000\_0001;

3'b001:EN<=8'b0000\_0010;

3'b010:EN<=8'b0000\_0100;

3'b011:EN<=8'b0000\_1000;

3'b100:EN<=8'b0001\_0000;

3'b101:EN<=8'b0010\_0000;

3'b110:EN<=8'b0100\_0000;

3'b111:EN<=8'b1000\_0000;

endcase

end

endmodule

用case语句分别描述输入输出关系。这里译码结果是8位二进制数中与输入3位二进制数对应的一位为1，其余为0.

算数加法模块：

module ADD(

F,C,A,B,EN

);

output reg [31:0]F;

output reg C;

input [31:0]A,B;

input EN;

wire[31:0] Ftmp;

wire Ctmp;

adder\_32bit add1(0,A,B,Ftmp,Ctmp);

always @(A,B,EN) begin

if (EN==0) begin C=1'bz;F=32'bz;end

//else {C,F}=A+B;

else begin C=Ctmp;F=Ftmp;end

end

endmodule

这里调用上述32位加法器模块实现。当使能EN（由以上38译码器译码输入）有效时，将32位加法结果和进位结果从ALU模块中输出。EN无效时，为了不干扰其他模块占用相同的数据线输出，就会产生高阻态信号。

减法模块：

module SUB(

F,C,A,B,EN

);

output reg[31:0]F;

output reg C;

input [31:0] A,B;

input EN;

always @(A,B,EN) begin

if (EN==0) begin C=1'bz;F=32'bz;end

else {C,F}=A-B;

end

endmodule

同上，使能EN有效时将减法结果和借位标志输出，无效时置高阻。

比较模块：

module SLT(

F,A,B,EN

);//compare,if A<B,output 1;otherwise 0

output reg[31:0] F;

input [31:0]A,B;

input EN;

always @(A,B,EN) begin

if (EN==0) F<=32'bz;

else begin

if (A<B) F<=1;

else F<=0;

end

end

endmodule

该模块在使能EN有效时比较A和B的大小，如果A<B则输出1，否则输出0.若该功能模块使能无效，同样置高阻态输出。

移位模块：

module SLL(

F,A,B,EN

);//逻辑左移，B左移A指定位数

parameter N=32;

output reg[N-1:0] F;

input [N-1:0] A,B;

input EN;

always@(A,B,EN) begin

if (EN==1) F<=B<<A;

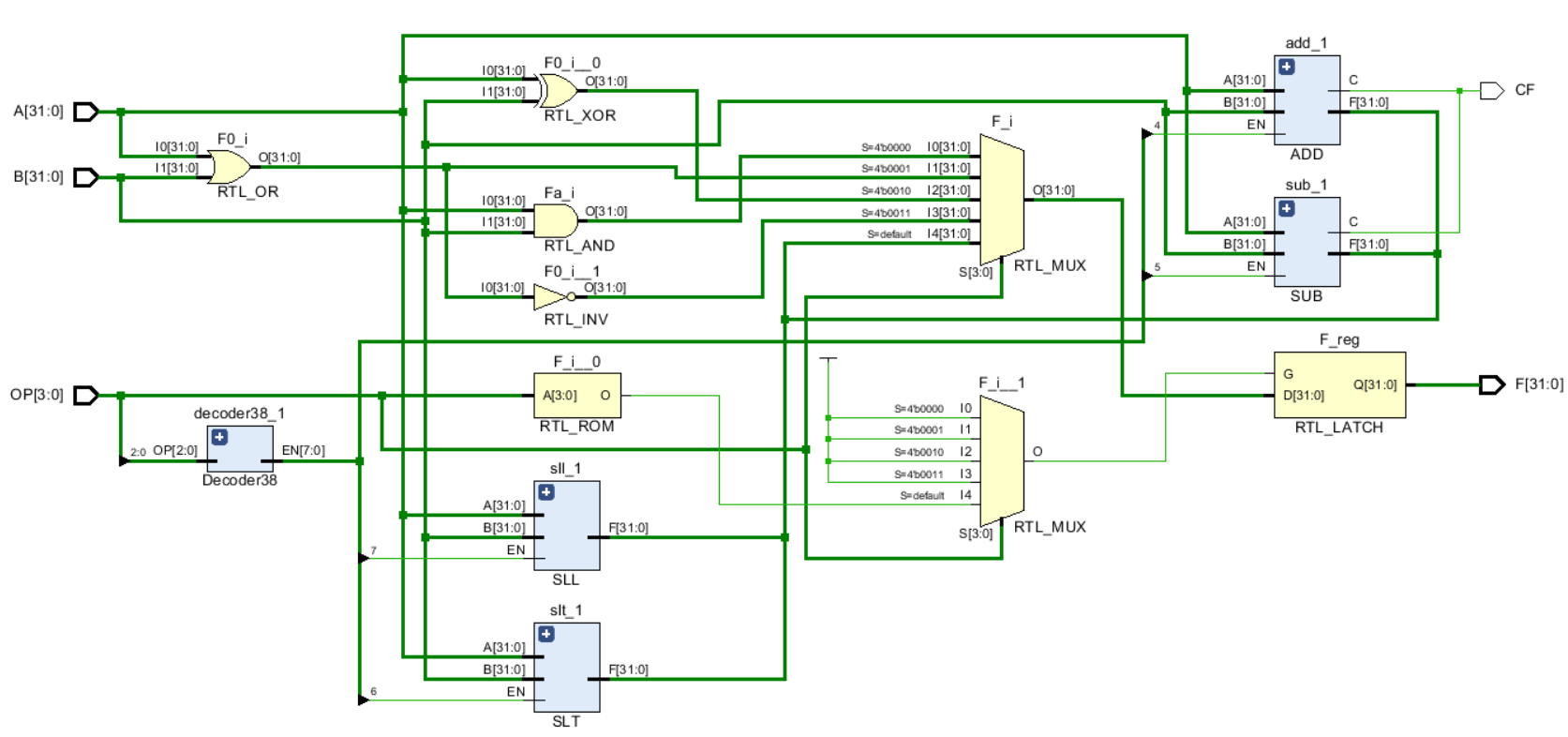
else F<=32'bz;

end

endmodule

实现逻辑左移功能，在使能EN有效时，B左移A指定的位数作为输出，EN无效时置高阻态。

RTL电路：



仿真测试代码：

module sim\_ALU\_8(

);

reg [3:0]OP;

reg [31:0]A,B;

wire C;

wire [31:0] F;

integer i;

initial begin

A=2;B=32'hFFFF\_FFFE;OP=4'b0000;

for (i=0;i<7;i=i+1) #25 OP=OP+1;

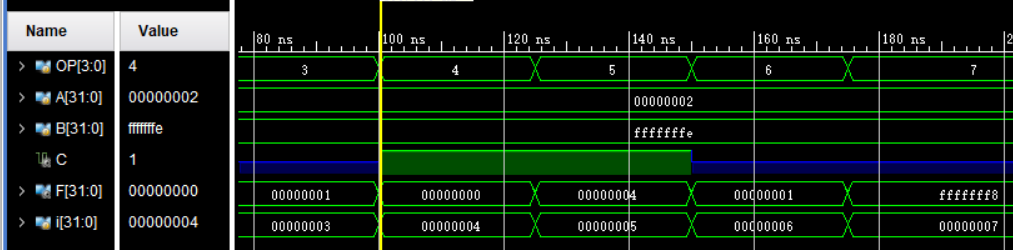
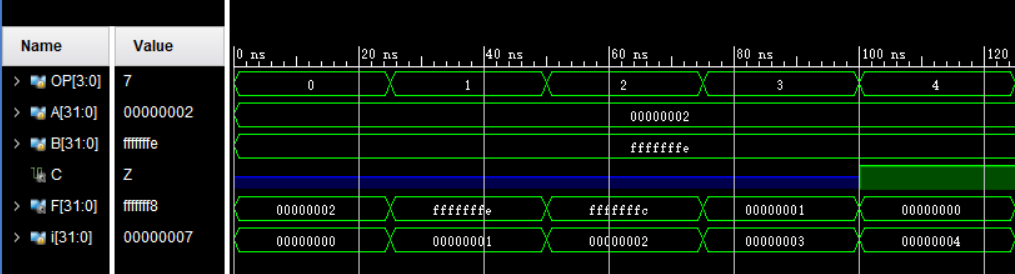
end

ALU\_8 test(F,C,A,B,OP);

Endmodule

这里输入A=2，B=FFFF\_FFFEh，功能选择信号依此遍历上述所有功能，观察仿真输出波形。

仿真波形：



计数值i=0时，按位与运算，A[31:0]中只有A[1]=1，B[31:0]中只有B[0]=0，因此按位与运算后F=A，C未使用为高阻态；i=1时，按位或运算，输出F=B，C仍为高阻态；i=2时，按位异或，因此只有F[1]=0，即F=FFFF\_FFFCh，C高阻；i=3时，按位或非，其结果是i=1时的输出按位取反，值为1，C高阻；i=4时，算数加法，恰好产生进位，本位和F为0，C=1；i=5时，算数减法，显然产生借位C=1，输出F=4正是按照补码的方式的运算结果；i=6时，比较大小，A<B输出F=1，C高阻；i=7时，B左移2位，这样B[2:0]=000，其他位都是1.输出F为FFFF\_FFF8，C高阻。所有测试结果均符合ALU功能描述和设计预期。。

**五、调试和心得体会**

1 在设计实现一些较复杂的功能模块时，可以采用自底向上的设计方法，将所有功能分割为若干功能相互独立的子模块，可以使设计更加清晰、简单。顶级模块只要调用下级的模块，并且可以嵌套调用层层封装，设计代码的可读性也会更好。但是，在设计子模块使要即使分别进行测试排查，确保子模块的设计逻辑正确，否则一旦在上级模块出现错误，排查难度会非常大。