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、38译码器

代码：

module Decoder38(I, Y);

input [2:0] I;

output reg[7:0] Y;

always@(\*) begin

case(I)

3'b000: Y = 8'b0000\_0001;

3'b001: Y = 8'b0000\_0010;

3'b010: Y = 8'b0000\_0100;

3'b011: Y = 8'b0000\_1000;

3'b100: Y = 8'b0001\_0000;

3'b101: Y = 8'b0010\_0000;

3'b110: Y = 8'b0100\_0000;

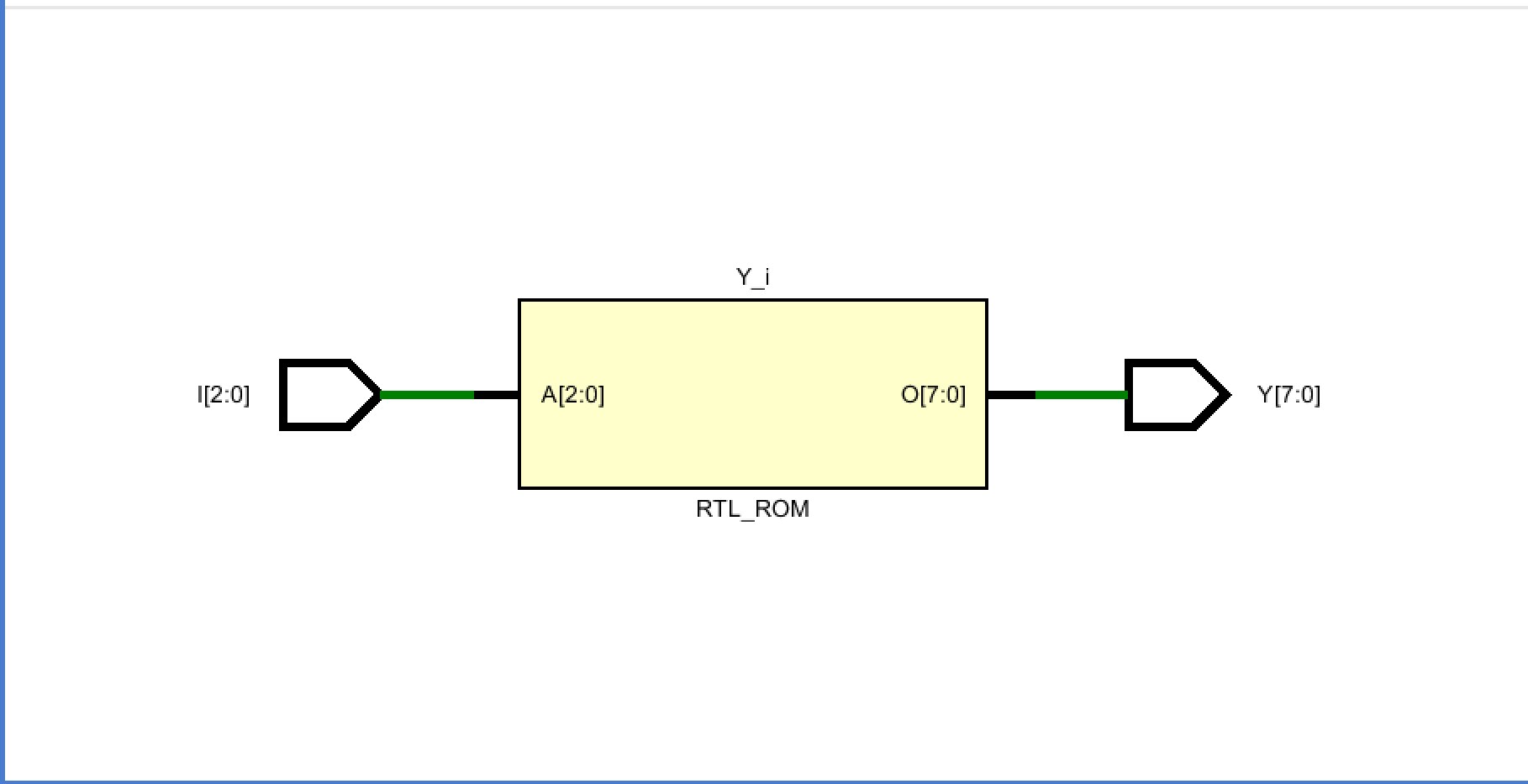
3'b111: Y = 8'b1000\_0000;

endcase

end

endmodule

逻辑仿真图：



代码分析：

使用always@(\*)语句，即只要检测到输入信号变化，使用case语句改编为对应的case值，38译码器已经实现过多次，此处不赘述。

2、32位加法器

代码：

module ADD(F, CF, A, B, EN);

parameter SIZE = 32;

output reg [SIZE-1:0] F;

output reg CF;

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

input EN;

always @(A, B, EN) begin

if (EN) begin

{CF,F}=A+B;

end

else begin

F <= 32'bz;

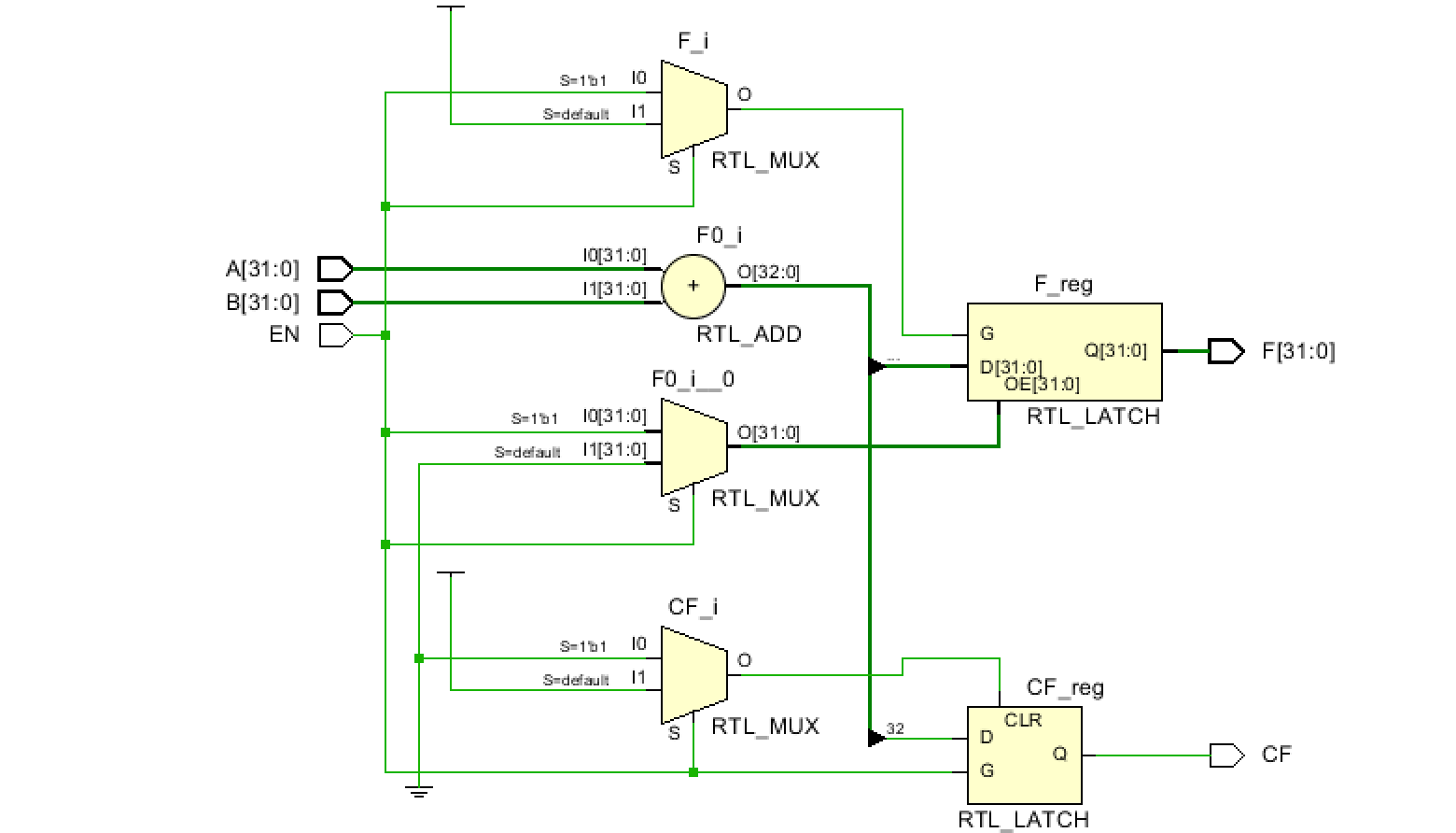
CF <= 1'bz;

end

end

endmodule

逻辑仿真图：

 代码分析：

本模块主要需要注意的地方有：

A+B可能会产生进位，则除32位输出F外还需多加一位输出CF标示进位，并使用{CF,F}=A+B语句，当A+B不产生进位时CF=0，F=A+B，当产生进位时，CF=1存储进位，F为低32位值。

在使能信号EN为低时，要求输出为高阻态，不影响其他部件输出。故使用if语句，如果EN=0，则{CF,F}=33’z，EN=1时，才正常工作。

3、32位减法器

代码：

module SUB(F, CF, A, B, EN);

parameter SIZE = 32;

output reg [SIZE-1:0] F;

output reg CF;

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

input EN;

always @(A, B, EN) begin

if (EN) begin

{CF,F}=A-B;

end

else begin

F <= 32'bz;

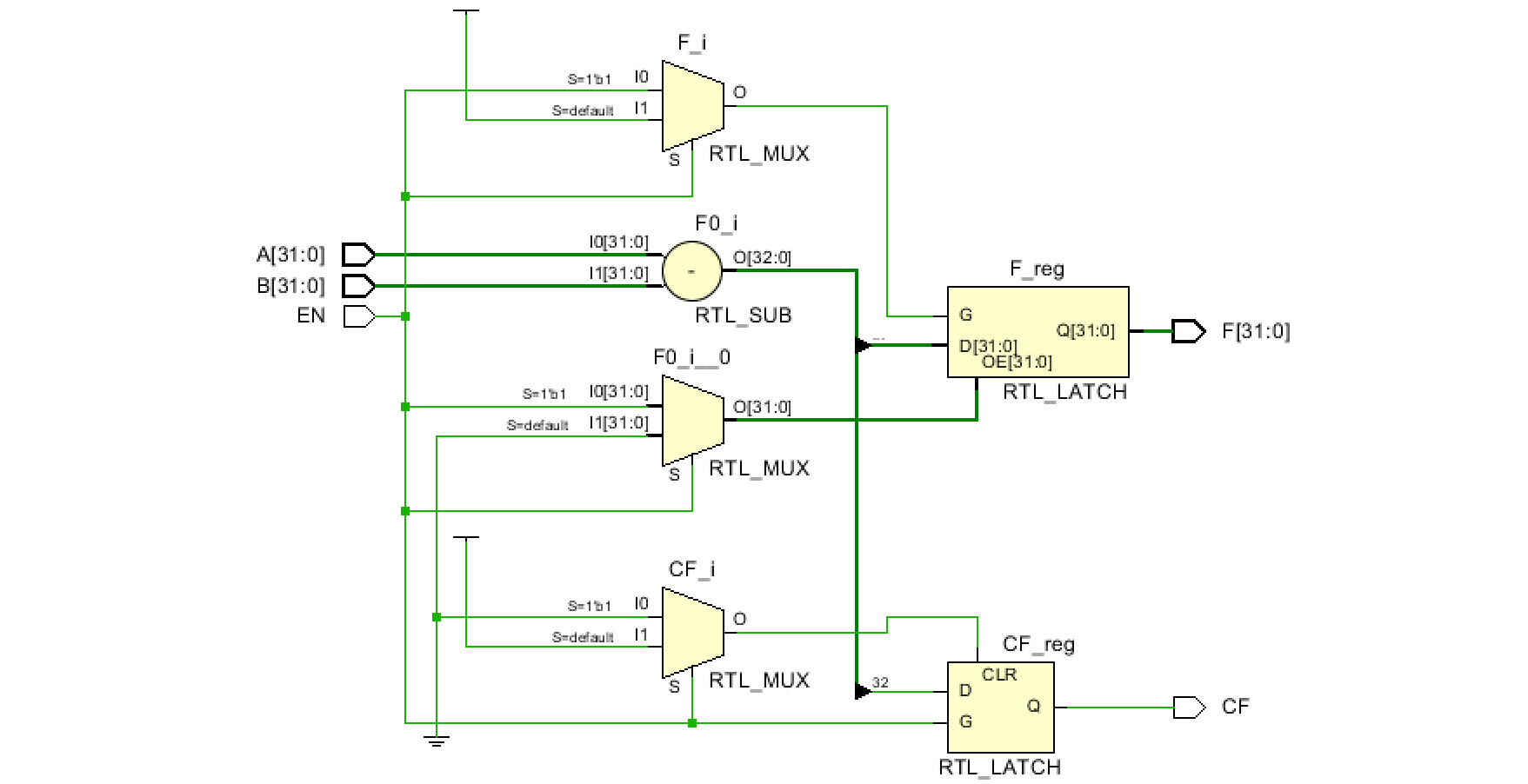
CF <= 1'bz;

end

end

endmodule

逻辑仿真图：



代码分析

注意事项与ADD模块基本相同，只需要把{CF,F}=A+B改为{CF,F}=A-B即可。

4、比较器

代码：

module SLT(F, A, B, EN);

parameter SIZE = 32;

output reg [SIZE-1:0] F;

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

input EN;

always @(A, B, EN) begin

if (EN)

F <= (A < B) ? 1'b1 : 1'b0;

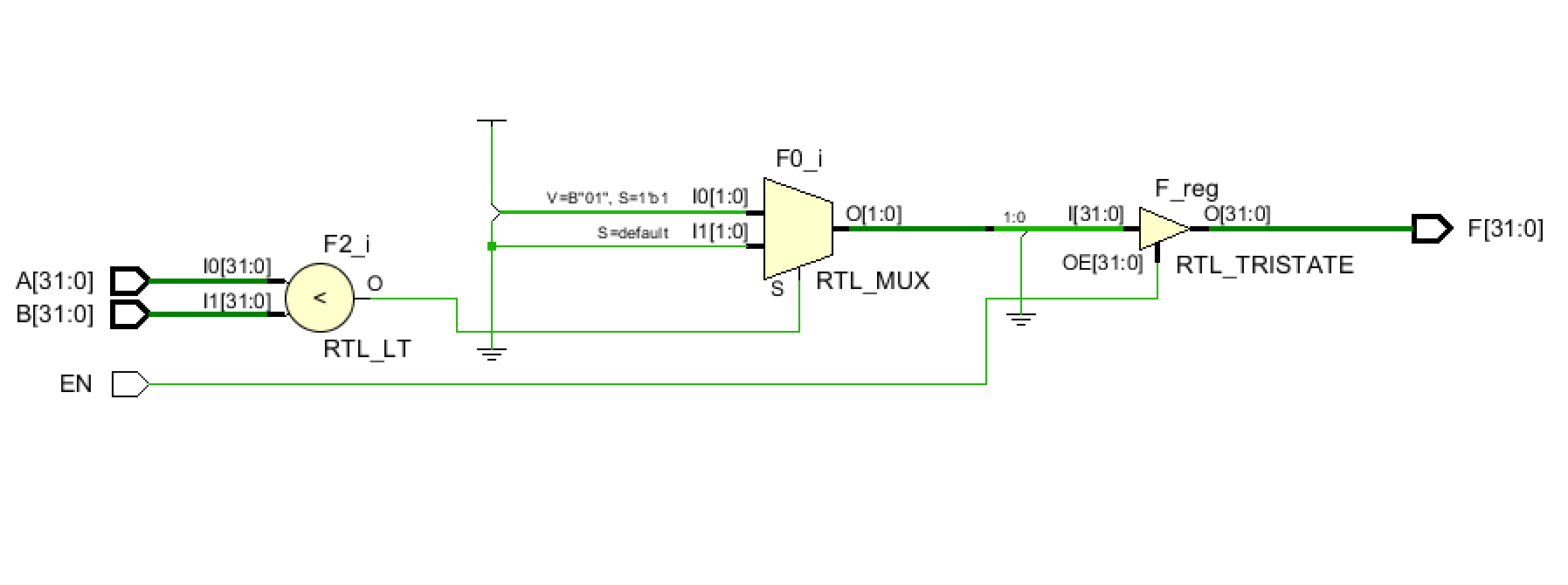
else

F <= 32'bz;

end

endmodule

逻辑仿真图



代码分析：

使用always@(A,B,EN)语句，每当有一个信号发生变化时更新状态。

使用if else 语句，在EN为0时输出高阻态（原因与加法器、减法器中置高阻态相同），EN有效时，根据A,B大小调整输出

5、左移器

代码：

module SLL(F, A, B, EN);

parameter SIZE = 32;

output reg [SIZE-1:0] F;

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

input EN;

always @(A, B, EN) begin

if (EN)

F <= B << A;

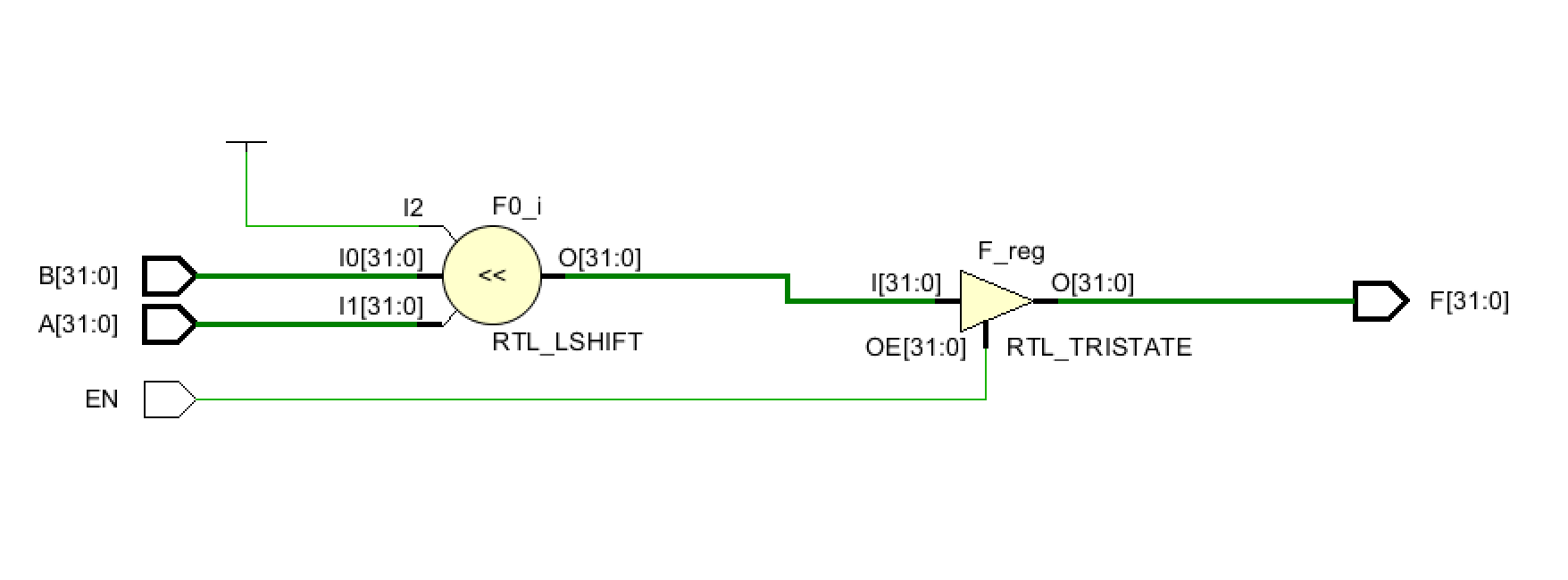
else

F <= 32'bz;

end

endmodule

逻辑仿真图：



代码分析：

基本思路与其他模块相同，只是在EN有效的情况下对F的处理不同。此模块做移位运算，先使用B<<A将B左移A位，再使用非阻塞赋值F<=B更新F的值。

6、ALU\_8

代码：

module ALU\_8(F, CF, A, B, OP);

parameter SIZE = 32;

output reg [SIZE-1:0] F;

output CF;

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

input [3:0] OP;

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;

parameter ALU\_SLL = 4'b0111;

wire [7:0] EN;

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

assign Fa = A & B;

always@(\*) begin

case(OP)

ALU\_AND: begin F <= Fa; end

ALU\_OR: begin F <= A|B; end

ALU\_XOR: begin F <= A^B; end

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

default: F = Fw;

endcase

end

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

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

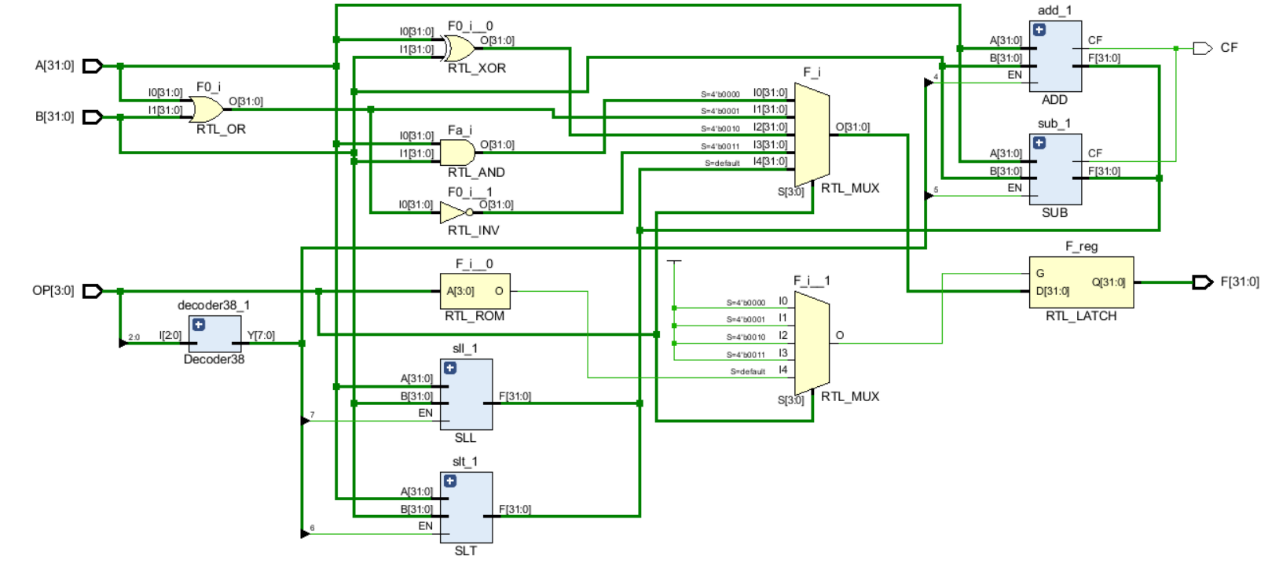
SUB sub\_1(Fw, CF, A, B, EN[5]);

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

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

endmodule

逻辑仿真图：



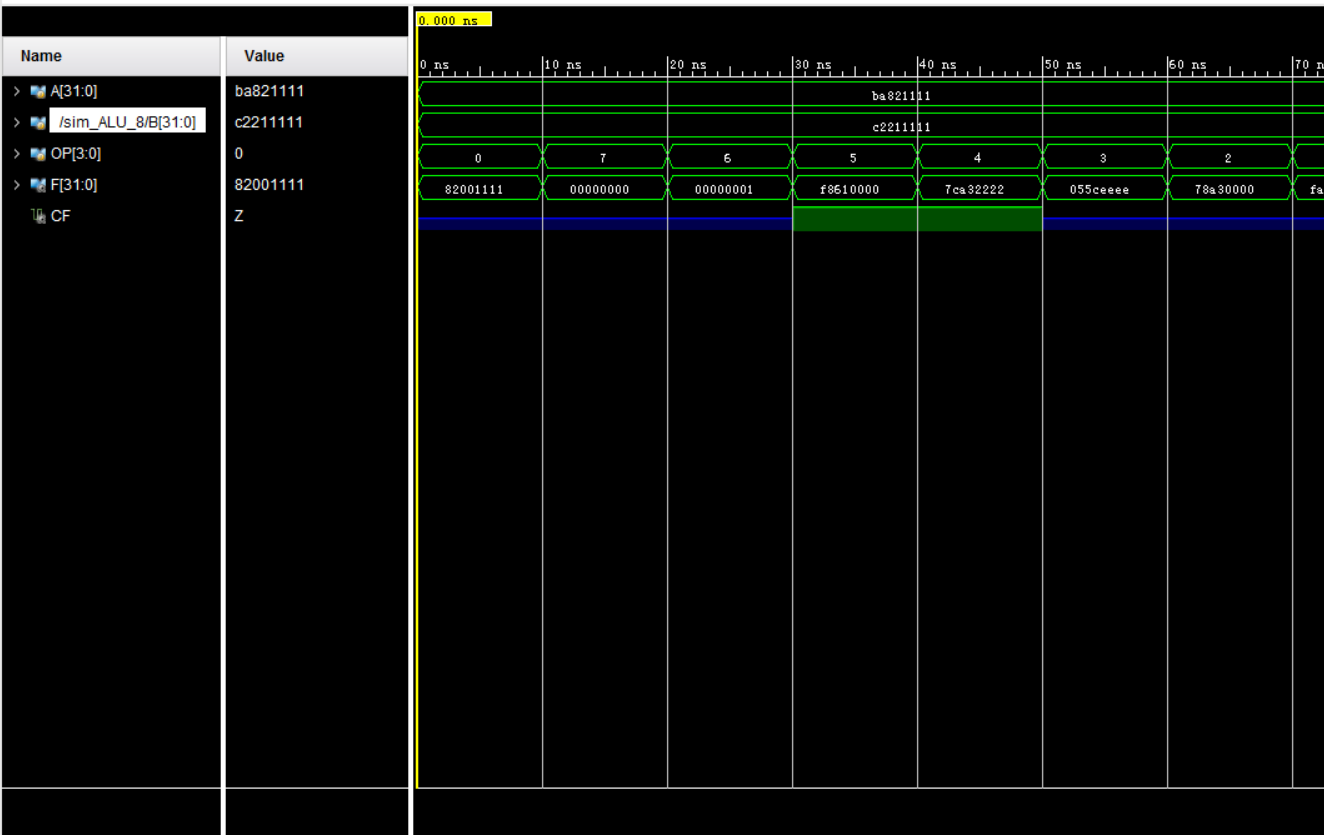
代码分析：

在调用Decoder38模块对3位输入信号进行译码后，根据译码之后的信号EN[8]进行选择。前四种（与、或、按位异或、按位或非）方式之间使用简单逻辑实现，剩下的4位分别作为输入信号进行模块实例化，即当译码器输出的使能信号有效位不在此4位时，此4位使能信号始终为低，4个实例化模块输出高组态，不影响与有效位为高的运算板块发生冲突。当4位信号的某个信号为有效信号时，此实例化模块做有效输出，其他为阻塞状态不影响正常输出。

ALU\_8仿真文件：

module sim\_ALU\_8();  
reg[31:0] A,B;  
reg[3:0] OP;  
wire [31:0] F;  
wire CF;  
initial begin  
A=32&apos;HBA821111;B=32&apos;HC2211111;OP=0;  
repeat(100) #10 if(OP>0) begin OP=OP-1;end else begin OP=7; end  
end  
ALU\_8 ALU(F, CF, A, B, OP);  
endmodule

时序图：



五、调试和心得体会

本次实验让我更加深入的理解了按模块设计文件这一Verilog的语言特性。先分别设计好简单模块的功能，再在主设计文件中实例化各个子模块，类似于面向对象编程里面的面向对象的思想，提高了封装性，也使得整个项目的结构更加清晰。