实验三 层次结构设计方法及应用

**一、实验目的**

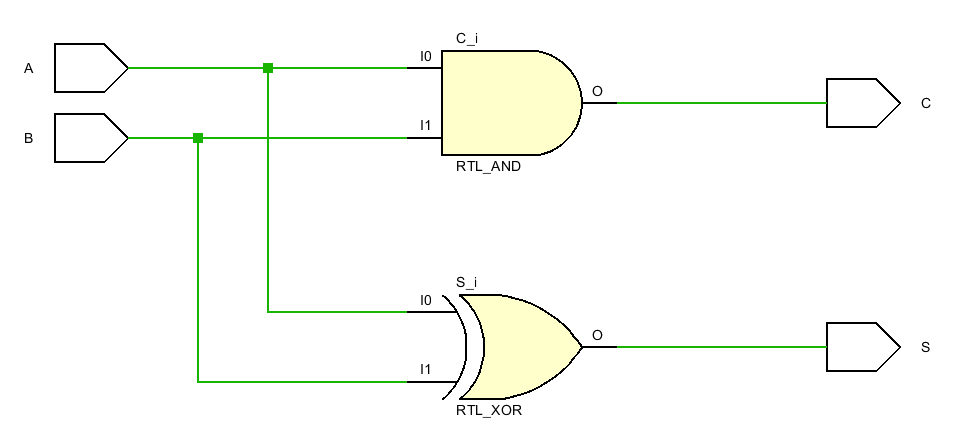
1. 进一步掌握Verilog中的基本语法和语句。
2. 熟悉Logisim软件的使用。

**二、实验内容**

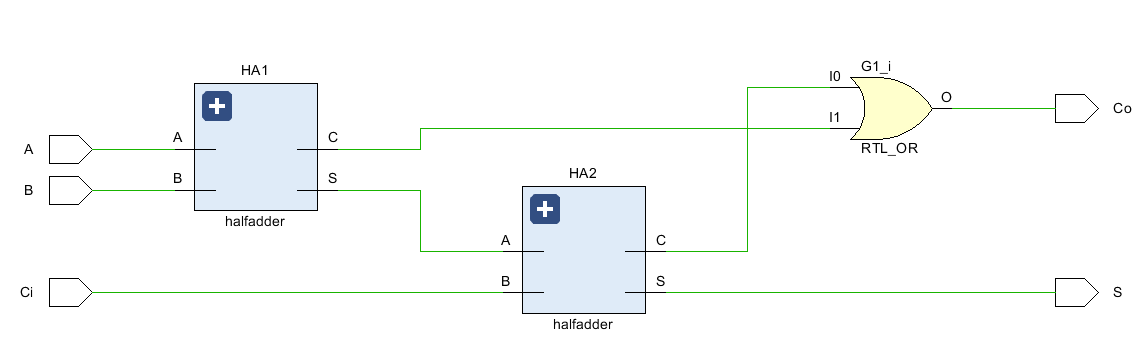
1. 掌握Verilog三种描述方式的使用。
2. 完成1位半加器、32位全加器模块的设计。
3. 设计一个基本的32位算术逻辑运算（ALU）模块。
4. 观察记录分析仿真波形。
5. 或者在Logisim中完成设计并验证。

**三、实验要求**

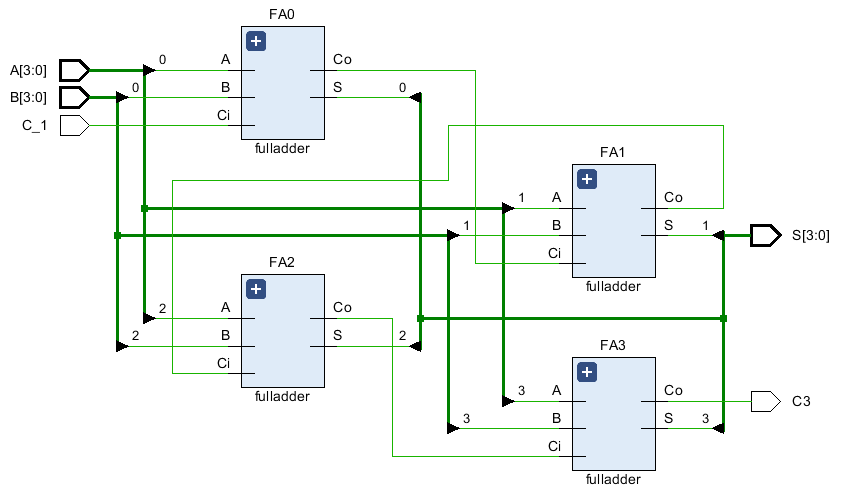
1. 画出模块的电路图。
   1. 1位半加器



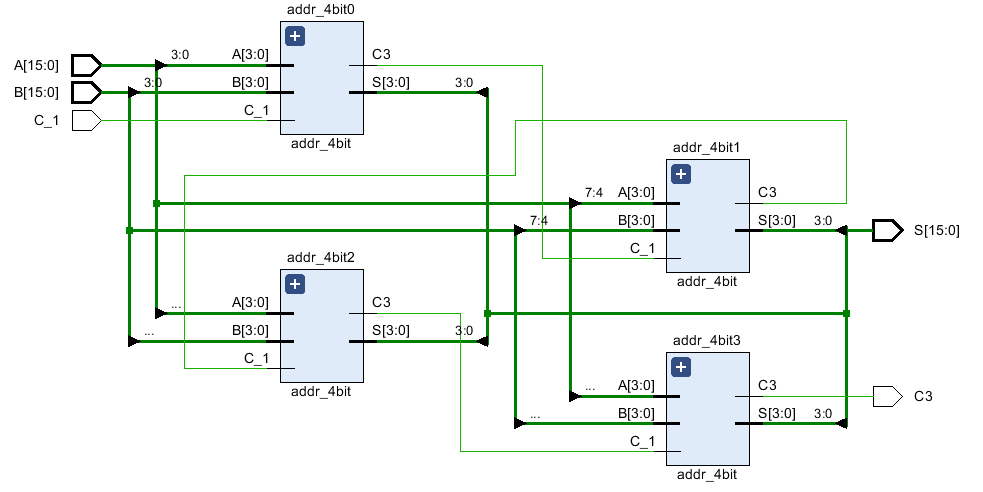
* 1. 1位全加器



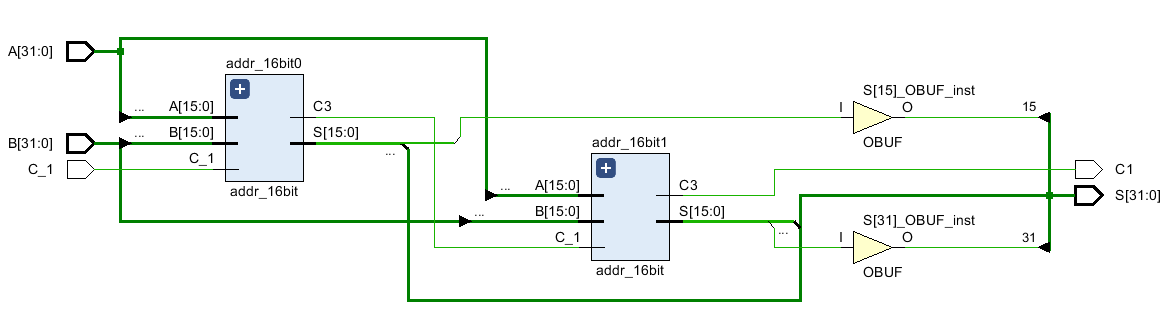
* 1. 4位全加器



* 1. 16位全加器

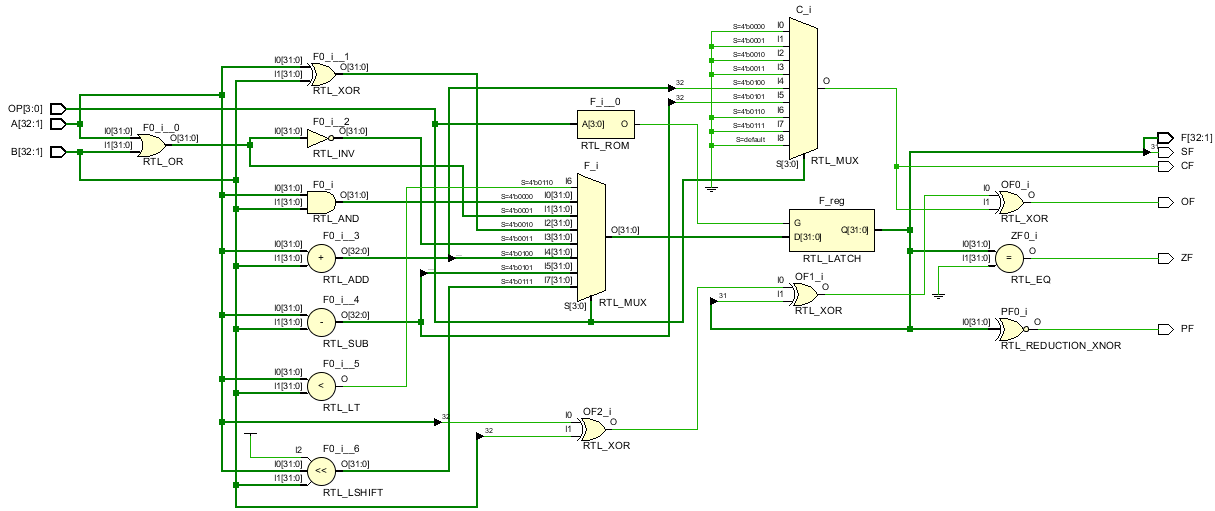


* 1. 32位全加器

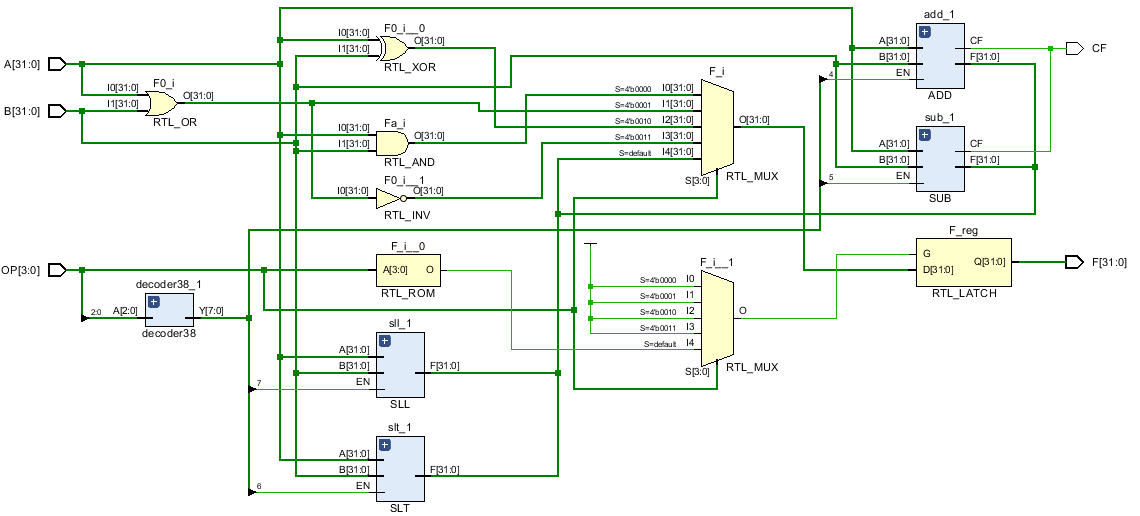


* 1. 32位算术逻辑运算（ALU）

Case语句



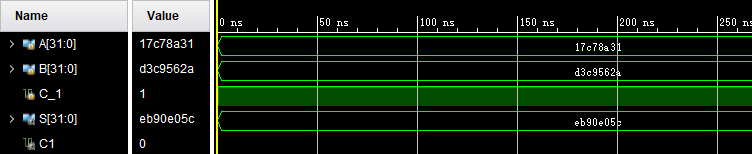
利用38译码器

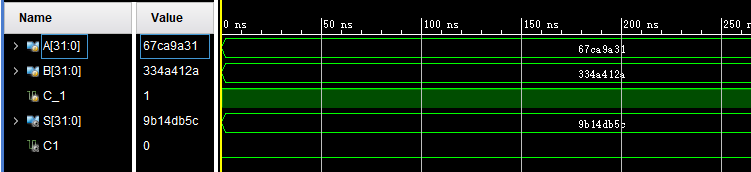


1. 分析电路的仿真波形/真值表，标出关键的数值。

由于32位全加器是基于前面所有模块的基础上设计的，故仅验证32位全加器即可

* 1. 32位全加器

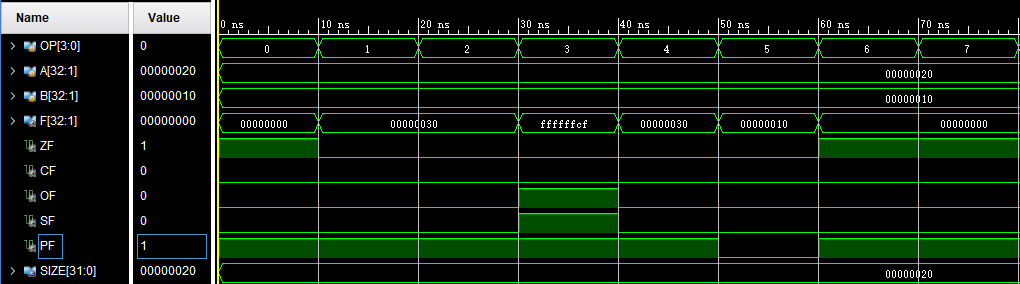




分别输入两组数据，验算均正确。

* 1. 32位算术逻辑运算（ALU）

Case实现

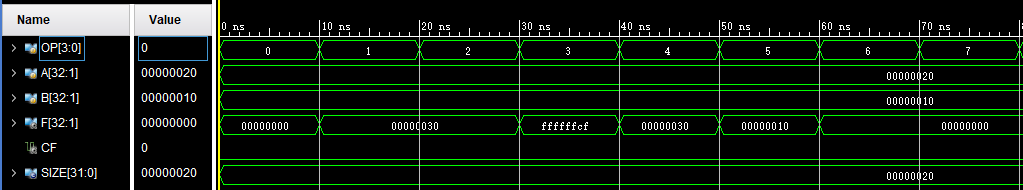


为方便观察，将A设为32，仅有从右往左数第6位为1，B设为16，仅有从右往左数第5位为1一次按op输入加一观察各功能情况。其中ZF为0标志位，CF为进位标志位，OF为溢出标志位，SF为符号标志位，PF为奇偶标志位。



按位与结果为0，ZF为1，1个数为偶PF为1。按位或仅第五第六位为1，即为00000030h，1个数为偶PF为1。按位异或也仅第五第六位为1，1个数为偶PF为1。按位或非仅第五第六位为0，即为ffffffcfh，产生溢出OF为1，SF为符号位为1，1个数为偶PF为1。加法运算结果为48，即00000030h，1个数为偶PF为1。减法运算结果为16，即00000010h，1个数为奇PF为0。比较中A>B，故输出0，1个数为偶PF为1。位移中将B逻辑左移A即32位，结果全为0，1个数为偶PF为1。

利用38译码器



与Case实现相同。

1. 记录设计和调试过程。

在1位半加器的基础上，设计出1位全加器，进而一层一层生成4位全加器，16位全加器，32位全加器。

ALU中op输入对应不同功能情况，适合用case选择情况，可以直接在case中判断op信号，也可用38译码器将op信号译后再判断。

**四、实验代码及结果**

1. 1位半加器

halfadder.v

module halfadder(S, C, A, B);

input A, B;

output S, C;

xor(S, A, B);

and(C, A, B);

endmodule

1. 1位全加器

fullader.v

module fulladder(S, Co, A, B, Ci);

input A, B, Ci;

output S, Co;

wire S1, D1, D2;

halfadder HA1(S1, D1, A, B);

halfadder HA2(.A(S1), .B(Ci), .S(S), .C(D2));

or G1(Co, D2, D1);

endmodule

1. 4位全加器

addr\_4bit.v

module addr\_4bit(S, C3, A, B, C\_1);

input [3:0]A, B;

input C\_1;

output [3:0]S;

output C3;

wire C0, C1, C2;

fulladder FA0(S[0], C0, A[0], B[0], C\_1),

FA1(S[1], C1, A[1], B[1], C0),

FA2(S[2], C2, A[2], B[2], C1),

FA3(S[3], C3, A[3], B[3], C2);

Endmodule

1. 16位全加器

addr\_16bit.v

module addr\_16bit(S, C3, A, B, C\_1);

input [15:0]A, B;

input C\_1;

output [15:0]S;

output C3;

wire C0, C1, C2;

addr\_4bit addr\_4bit0(S[3:0], C0, A[3:0], B[3:0], C\_1),

addr\_4bit1(S[7:4], C1, A[7:4], B[7:4], C0),

addr\_4bit2(S[11:8], C2, A[11:8], B[11:8], C1),

addr\_4bit3(S[15:12], C3, A[15:12], B[15:12], C2);

endmodule

1. 32位全加器

addr\_32bit.v

module addr\_32bit(S, C1, A, B, C\_1);

input [31:0]A, B;

input C\_1;

output [31:0]S;

output C1;

wire C0;

addr\_16bit addr\_16bit0(S[15:0], C0, A[15:0], B[15:0], C\_1),

addr\_16bit1(S[31:16], C1, A[31:16], B[31:16], C0);

endmodule

sim\_addr\_32bit.v

module sim\_addr\_32bit();

reg[31:0] A, B;

reg C\_1;

wire [31:0]S;

wire C1;

initial begin

A = 32'h67ca9a31;B = 32'h334a412a;C\_1 = 1;

end

addr\_32bit addr\_32bit0(S, C1, A, B, C\_1);

endmodule

1. 32位算术逻辑运算（ALU）

Case语句

ALU.v

module ALU(OP, A, B, F, ZF, CF, OF, SF, PF);

parameter SIZE = 32;

input [3:0] OP;

input [SIZE:1] A;

input [SIZE:1] B;

output reg [SIZE:1] F;

output ZF, CF, OF, SF, PF;

reg C, ZF, CF, OF, SF, PF;

always @(\*)

begin

C = 0;

case(OP)

4'b0000: begin F = A & B; end

4'b0001: begin F = A | B; end

4'b0010: begin F = A ^ B; end

4'b0011: begin F = ~(A | B); end

4'b0100: begin {C, F} = A + B; end

4'b0101: begin {C, F} = A - B; end

4'b0110: begin F = A < B; end

4'b0111: begin F = A << B; end

endcase

ZF = F === 0;

CF = C;

OF = A[SIZE]^B[SIZE]^F[SIZE]^C;

SF = F[SIZE];

PF = ~^F;

end

endmodule

sim\_ALU.v

module sim\_ALU();

parameter SIZE = 32;

reg [3:0] OP;

reg [SIZE:1] A;

reg [SIZE:1] B;

wire [SIZE:1] F;

wire ZF, CF, OF, SF, PF;

initial begin

A = 32;B = 16;OP = 4'b0000;

fork

forever

if(OP <= 4'b0111) #10 OP = OP + 1;

else OP = 4'b0000;

join

end

ALU ALU1(OP, A, B, F, ZF, CF, OF, SF, PF);

endmodule

利用38译码器

ALU\_8.v

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

parameter SIZE = 32;

input [3:0] OP;

input [SIZE-1:0] A;

input [SIZE-1:0] B;

output reg [SIZE-1:0] F;

output CF;

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

decoder38.v

module decoder38(A, Y);

input[2:0] A;

output reg[7:0] Y;

integer i;

always @(A) begin

Y = 8'b1111\_1111;

for(i = 0; i <= 7; i = i + 1)

if(A == i)

Y[i] = 0;

else

Y[i] = 1;

end

endmodule

ADD.v

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

parameter N = 32;

input EN;

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

output reg [N-1:0] F;

output reg CF;

always @(A, B, EN) begin

if(EN == 0) {CF, F} <= A + B;

else begin F <= 32'bz; CF = 0; end

end

endmodule

SUB.v

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

parameter N = 32;

input EN;

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

output reg [N-1:0] F;

output reg CF;

always @(A, B, EN) begin

if(EN == 0) begin {CF, F} <= A - B; end

else begin F <= 32'bz; CF = 0; end

end

endmodule

SLT.v

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

parameter N = 32;

input EN;

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

output reg [N-1:0] F;

always @(A, B, EN) begin

if(EN == 0) F <= A < B;

else F <= 32'bz;

end

endmodule

SLL.v

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

parameter N = 32;

output reg[N-1:0] F;

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

input EN;

always @(A, B, EN) begin

if(EN == 0) F <= B << A;

else F <= 32'bz;

end

endmodule

sim\_ALU\_8.v

module sim\_ALU\_8();

parameter SIZE = 32;

reg [3:0] OP;

reg [SIZE:1] A;

reg [SIZE:1] B;

wire [SIZE:1] F;

wire CF;

initial begin

A = 32;B = 16;OP = 4'b0000;

fork

forever

if(OP <= 4'b0111) #10 OP = OP + 1;

else OP = 4'b0000;

join

end

ALU\_8 ALU\_8\_1(F, CF, A, B, OP);

endmodule

**五、调试和心得体会**

起初发现ALU\_8测试时在后四种情况没有结果，后来想起来38译码器是将对应位数置0，其余位数置1，在修改ADD等模块中if(EN == 1)改为if(EN == 0)后正确。