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。

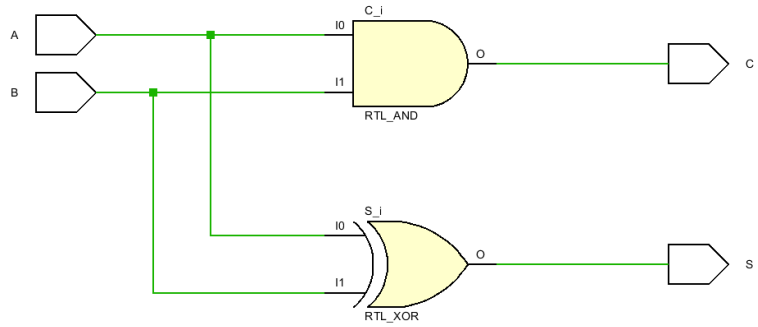
二、实验内容

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。

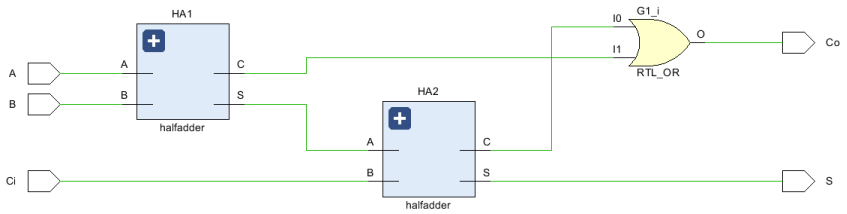
三、实验要求

1. 画出模块的电路图。

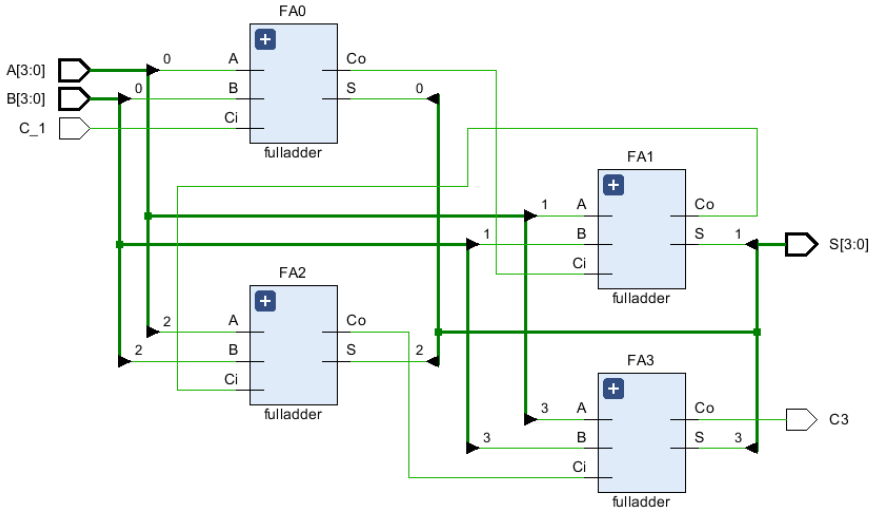
1.1 1 位半加器



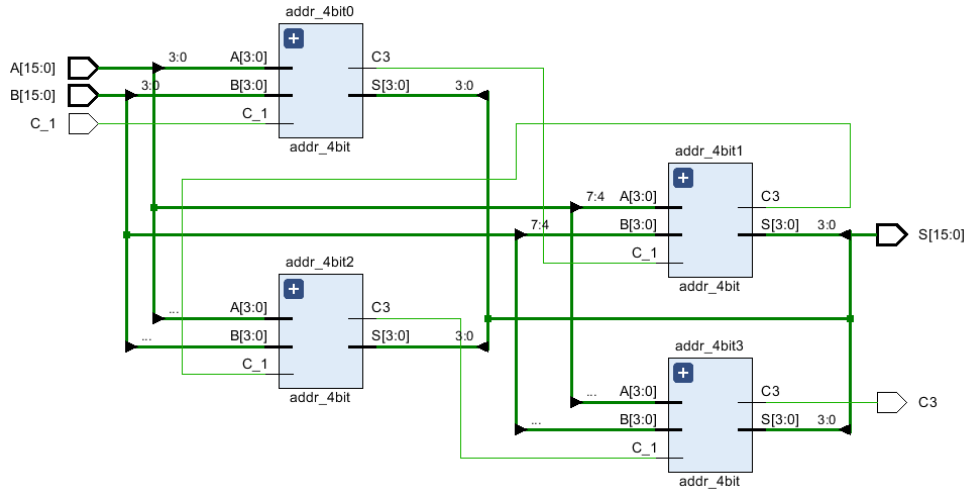
1.2 1 位全加器



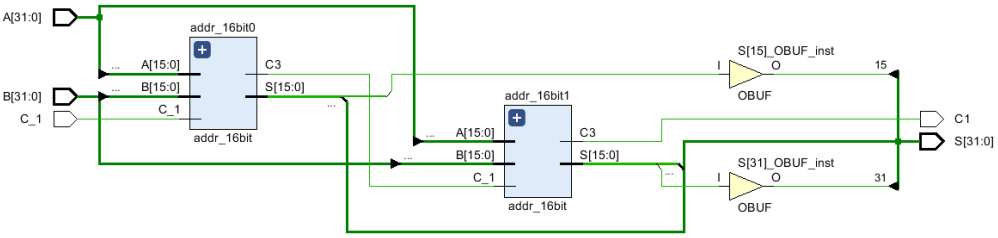
1.3 4 位全加器



1.4 16 位全加器

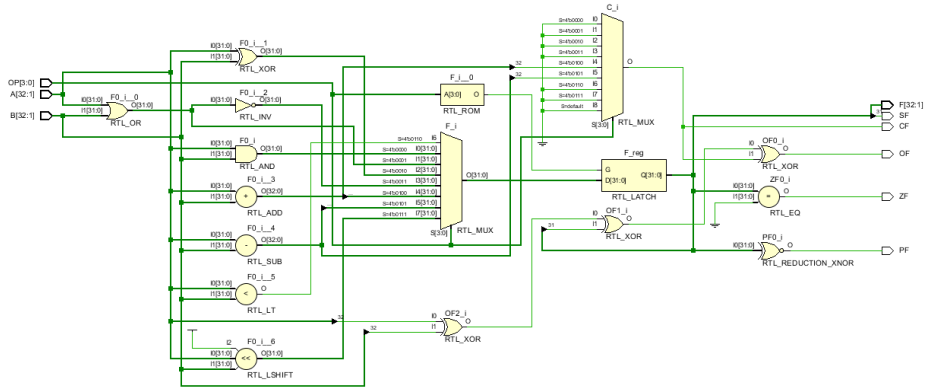


1.5 32 位全加器

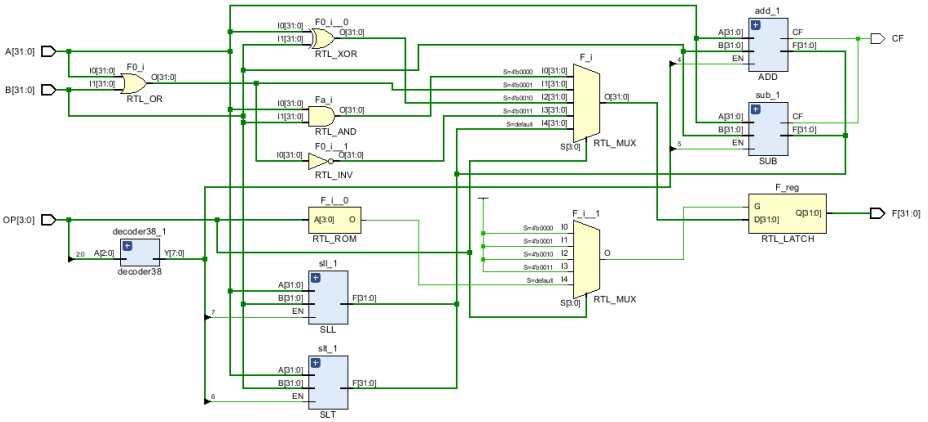


1.6 32 位算术逻辑运算（ALU）

Case 语句



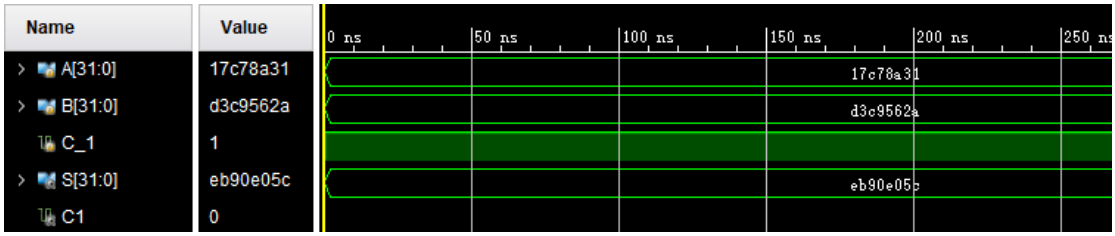
利用 38 译码器

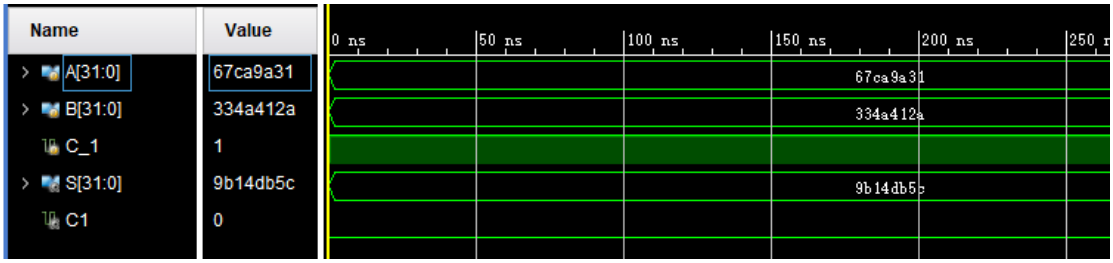


2.　分析电路的仿真波形/真值表，标出关键的数值。

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

2.1 32 位全加器

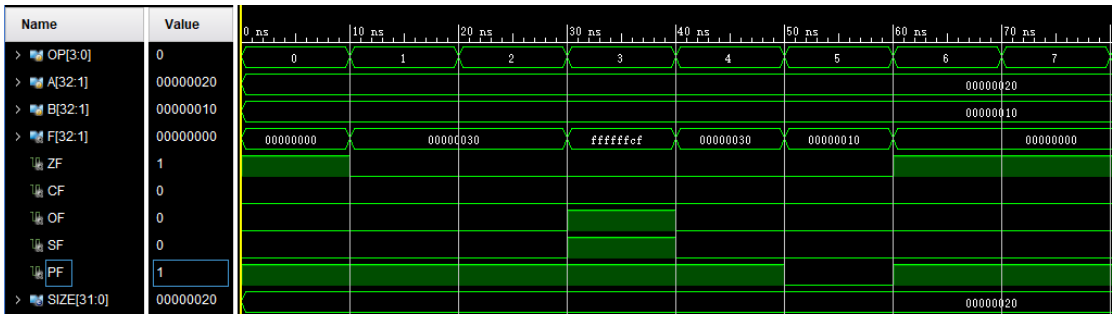




分别输入两组数据，验算均正确。

2.2　32 位算术逻辑运算（ALU）

Case 实现



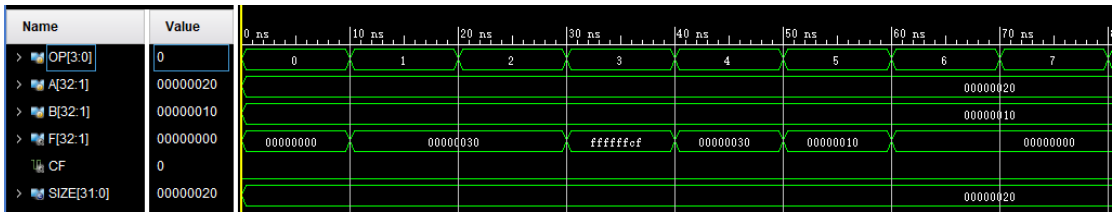
为方便观察，将 A 设为 32，仅有从右往左数第 6 位为 1，B 设为 16，仅有从右往左数第 5 位为 1 一次按 op 输入加一观察各功能情况。其中 ZF 为 0 标志位，CF 为进位标志位，OF 为溢出标志位，SF 为符号标志位，PF 为奇偶标志位。

ALU_OP[3:0]	ALU功能	功能说明
0000	and	按位与运算
0001	or	按位或运算
0010	xor	按位异或运算
0011	nor	按位或非运算
0100	add	算术加运算
0101	sub	算术减运算
0110	slt	若A<B，则输出1;否则输出0
0111	sll	B逻辑左移A所指定的位数

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。

3. 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
```

2. 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
```

3. 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
```

4. 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
```

5. 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
```

6. 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。