1.1 四位超前进位加法器（CLA）

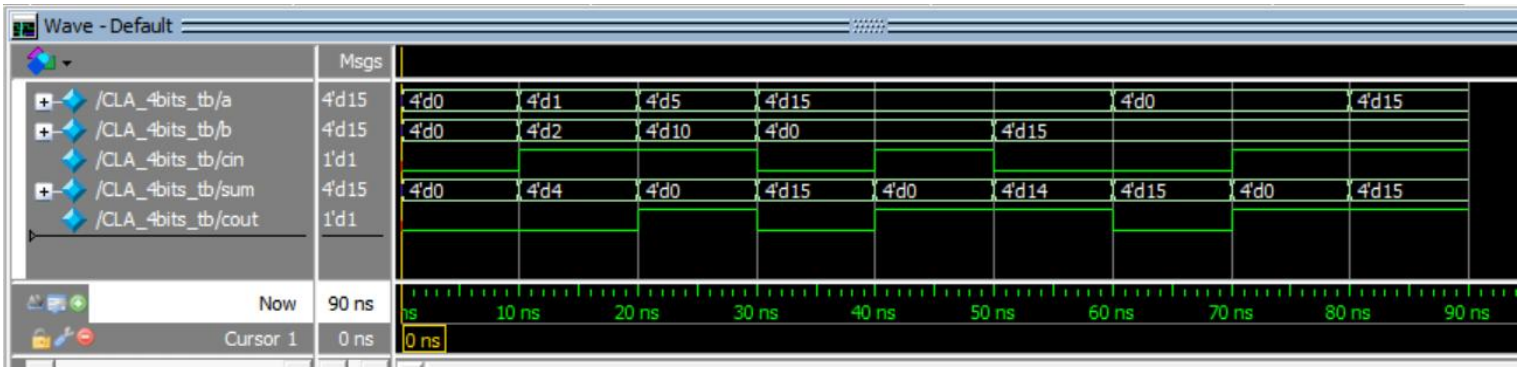
1. 功能模块代码：

```
//4位超前进位加法器
module CLA_4bits(
    //input
        cin,
        a,
        b,
    //output
        s,
        cout
    );

    parameter N = 4;

    input          cin;
    input [N-1:0]    a;
    input [N-1:0]    b;

    output       cout;
    output [N-1:0]  s;

    wire [N-1:0]  p;
    wire [N-1:0]  g;
    wire [N:0]   co;

        assign g = a & b;
        assign p = a ^ b;


    begin assign co[0] = cin; end
    begin assign co[1] = ( co[0] && p[0] ) || g[0]; end
    begin assign co[2] = ( co[0] && p[0] && p[1]) || ( p[1] && g[0] ) || g[1]; end
    begin assign co[3] = ( co[0] && p[0] && p[1] && p[2]) || ( p[2] && p[1] && g[0] ) ||( p[2] && g[1] ) || g[2]; end
    begin assign co[4] = ( co[0] && p[0] && p[1] && p[2] && p[3]) || ( p[3] && p[2] && p[1] && g[0] ) ||( p[3] && p[2] && g[1] ) || ( p[3] && g[2] ) || g[3]; end
    begin assign cout = co[4]; end

    begin assign s[0] = co[0] ^ p[0]; end
    begin assign s[1] = co[1] ^ p[1]; end
    begin assign s[2] = co[2] ^ p[2]; end
    begin assign s[3] = co[3] ^ p[3]; end

endmodule
```

2. 测试模块代码：

```
//CLA_4bits_testbench
`timescale 1ns/1ns
module CLA_4bits_tb;

    reg [3:0] a;
    reg [3:0] b;
    reg     cin;

    wire [3:0] sum;
    wire      cout;

    CLA_4bits CLA_4bits_test(
            .a(a)       ,
            .b(b)       ,
            .cin(cin)   ,
            .cout(cout),
            .s(sum)
    );

    initial begin
            a = 4'd0; b = 4'd0; cin = 1'd0;
    #10     a = 4'd1; b = 4'd2; cin = 1'd1;
    #10     a = 4'd5; b = 4'd10; cin = 1'd1;
    #10     a = 4'd15; b = 4'd0; cin = 1'd0;
    #10     a = 4'd15; b = 4'd0; cin = 1'd1;
    #10     a = 4'd15; b = 4'd15; cin = 1'd0;
    #10     a = 4'd0; b = 4'd15; cin = 1'd0;
    #10     a = 4'd0; b = 4'd15; cin = 1'd1;
    #10     a = 4'd15; b = 4'd15; cin = 1'd1;
    #10 $finish;
    end

    initial begin
        $monitor("%d+%d+%d: %b %b", a, b, cin, cout, sum);
        $dumpfile("CLA_4bits.vcd");
        $dumpvars;
    end

endmodule
```

3. ModelSim 仿真结果及分析：




```
VSIM 3> run -all
#  0+ 0+0: 0 0000
#  1+ 2+1: 0 0100
#  5+10+1: 1 0000
# 15+ 0+0: 0 1111
# 15+ 0+1: 1 0000
# 15+15+0: 1 1110
#  0+15+0: 0 1111
#  0+15+1: 1 0000
# 15+15+1: 1 1111
# ** Note: $finish    : H:/study_master/Digital_Integrated_Circuit_Design/homework/ALU/ADDER/CLA_4bits/src/CLA_4bits_tb.v(30)
#    Time: 90 ns  Iteration: 0  Instance: /CLA_4bits_tb
```

由输出波形和输出文本分析可知，四位超前进位加法器设计正确。

1.2 32 位加法器

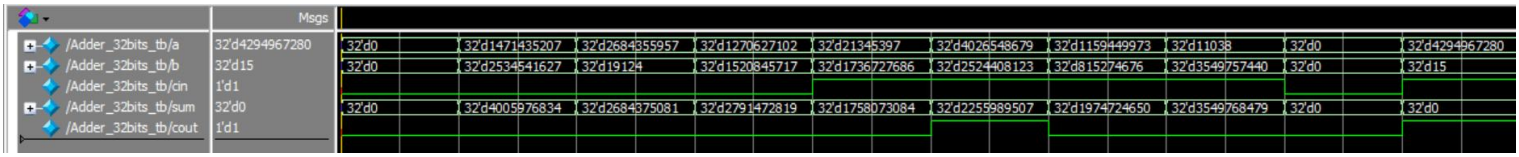
1. 功能模块代码：

```
1   //32位加法器，4位一组，组内使用超前进位，组间使用行波进位
2   module Adder_32bits(
3       //input
4       a_in,
5       b_in,
6       c_in,
7       //output
8       sum_o,
9       c_o
10      );
11
12      input [31:0] a_in;
13      input [31:0] b_in;
14      input        c_in;
15
16      output [31:0] sum_o;
17      output          c_o;
18
19      wire [7:0] cout;
20
21      CLA_4bits #(4) adder0(
22                          .a    (a_in[3:0]),
23                          .b    (b_in[3:0]),
24                          .cin  (c_in),
25                          .s    (sum_o[3:0]),
26                          .cout(cout[0])
27                          );
28      CLA_4bits #(4) adder1(
29                          .a    (a_in[7:4]),
30                          .b    (b_in[7:4]),
31                          .cin  (cout[0]),
32                          .s    (sum_o[7:4]),
33                          .cout(cout[1])
34                          );
35      CLA_4bits #(4) adder2(
42      CLA_4bits #(4) adder3(
49      CLA_4bits #(4) adder4(
56      CLA_4bits #(4) adder5(
63      CLA_4bits #(4) adder6(
70      CLA_4bits #(4) adder7(
77
78      assign c_o = cout[7];
79
80  endmodule
```

2. 测试模块代码：

```
//Adder_32bits_testbench
`timescale 1ns/1ns
module Adder_32bits_tb;
    reg [31:0] a;
    reg [31:0] b;
    reg cin;

    wire [31:0] sum;
    wire cout;

    Adder_32bits Adder_32bits_test(
                        .a_in   (a),
                        .b_in   (b),
                        .c_in (cin),
                        .sum_o(sum),
                        .c_o  (cout)
                        );

    initial begin
         a = 32'd0;         b = 32'd0;        cin = 0;
    #10  a = 32'h57b451c7; b = 32'h9712093b;cin = 0;
    #10  a = 32'ha0000575; b = 32'h00004ab4;cin = 0;
    #10  a = 32'h4bbc3b1e; b = 32'h5aa64395;cin = 0;
    #10  a = 32'h0145b475; b = 32'h67845c86;cin = 1;
    #10  a = 32'hf00041c7; b = 32'h9677693b;cin = 1;
    #10  a = 32'h451bcd75; b = 32'h30981ab4;cin = 1;
    #10  a = 32'h00002b1e; b = 32'hd3950000;cin = 1;
    #10  a = 32'h0;        b = 32'h0;        cin = 0;
    #10  a = 32'hfffffff0; b = 32'hf;        cin = 1;
    #10  $finish;
    end

    initial begin
        $monitor("%d + %d + %d: %d %d", a, b, cin, cout, sum);
        $dumpfile("Adder_32bits.vcd");
        $dumpvars;
    end

endmodule
```

3. ModelSim 仿真结果及分析：




```
VSIM 10> run -all
#          0 +          0 + 0: 0          0
# 1471435207 + 2534541627 + 0: 0 4005976834
# 2684355957 +      19124 + 0: 0 2684375081
# 1270627102 + 1520845717 + 0: 0 2791472819
#   21345397 + 1736727686 + 1: 0 1758073084
# 4026548679 + 2524408123 + 1: 1 2255989507
# 1159449973 +  815274676 + 1: 0 1974724650
#      11038 + 3549757440 + 1: 0 3549768479
#          0 +          0 + 0: 0          0
# 4294967280 +         15 + 1: 1          0
# ** Note: $finish    : H:/study_master/Digital_Integrated_Circuit_Design/homework/ALU/ADDER/Adder_32bits/src/Adder_32bits_tb.v(30)
#    Time: 100 ns  Iteration: 0  Instance: /Adder_32bits_tb
# 1
# Break in Module Adder_32bits_tb at H:/study_master/Digital_Integrated_Circuit_Design/homework/ALU/ADDER/Adder_32bits/src/Adder_32bits_tb.v line 30
```

由输出波形和输出文本结果可知，32 位加法器设计正确。

1.3 32 位 ALU 设计

1. 功能模块代码：

```
//32位ALU
module ALU_core#(
    parameter n = 32
    )(
    //input
        opA,     //操作数A
        opB,     //操作数B
        S,       //工作模式选择信号
        M,       //逻辑操作控制信号
        Cin,     //进位输入信号
    //output
        DO,      //数据输出
        C,       //进位输出
        V,       //溢出指示输出信号
        N,       //DO符号位输出信号
        Z,       //DO为全0指示信号
    );

    input [n-1:0] opA;
    input [n-1:0] opB;
    input [3:0]     S;
    input           M;
    input         Cin;

    output [n-1:0] DO;
    output          C;
    output          V;
    output          N;
    output          Z;

    //将ALU扩展为33位计算
    wire [n:0] opA_ex;
    wire [n:0] opB_ex;
    wire [n:0]  DO_ex;
    wire [n:0]      g;
    wire [n:0]      p;
    wire [n:0]     Ci;

    wire [n:0] S3_joint;
    wire [n:0] S2_joint;
    wire [n:0] S1_joint;
    wire [n:0] S0_joint;
    wire [n:0] M_joint;

    assign S3_joint = {33{S[3]}};
    assign S2_joint = {33{S[2]}};
    assign S1_joint = {33{S[1]}};
    assign S0_joint = {33{S[0]}};
    assign M_joint = {33{M}};

    assign opA_ex = {1'b0, opA};
    assign opB_ex = {1'b0, opB};


    assign g =   (S3_joint & opA_ex & opB_ex)|(S2_joint & opA_ex & (~opB_ex))|(~M_joint);
    assign p = ~((S3_joint & opA_ex & opB_ex)|(S2_joint & opA_ex & (~opB_ex))|(S1_joint & opB_ex & (~opA_ex))|(S0_joint & (~opA_ex) & (~opB_ex)));

    assign Ci[0] = Cin;

    genvar i;
    generate
        for(i=1;i<=32;i=i+1)
            begin: gen_Ci
            assign Ci[i] = ( Ci[i-1] & p[i-1] ) | g[i-1];
            end
    endgenerate

    assign DO_ex = p ^ Ci;


    assign DO = DO_ex[31:0];
    assign  C = (M)?DO_ex[32]:0;
    assign  V = (M)?(DO_ex[32] ^ DO_ex[31]):0;
    assign  N = (M)?DO_ex[31]:0;
    assign  Z = (DO == 0);

endmodule
```

思路：将 ALU 扩展为 33 位进行计算。将参与位运算的数据（M、S 等）均进行位扩展。输出取低 32 位。

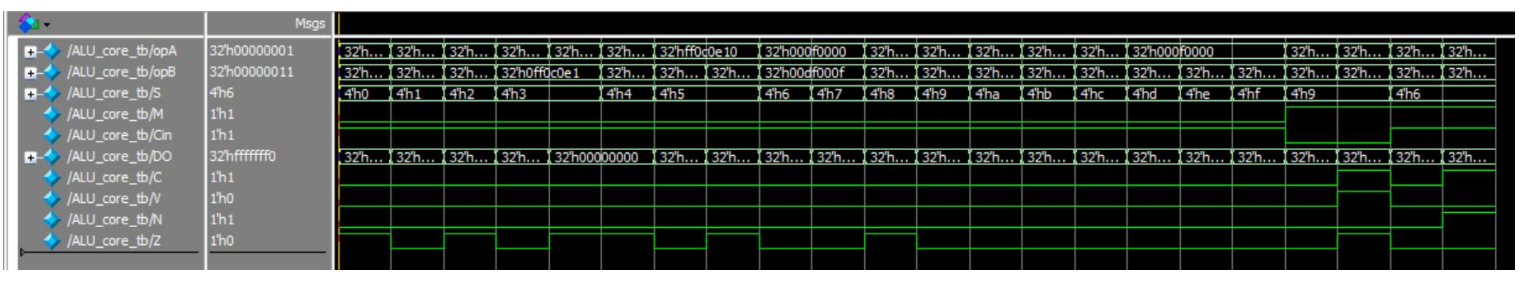
标志位的确定：逻辑运算时，进位输出、溢出指示、符号位输出均设为 0，视为无效；只有算术运算时有效，进位输出为 33 位结果的最高位，符号位输出为 33 位结果的次高位，溢出指示为 33 位结果的高两位进行异或后的值。

2. 测试模块代码：

```
U_core_testbench
escale 1ns/1ns
le ALU_core_tb;
reg [31:0] opA;
reg [31:0] opB;
reg [3:0]    S;
reg          M;
reg        Cin;

wire [31:0] DO;
wire         C;
wire         V;
wire         N;
wire         Z;

ALU_core ALU_core_test(
//input
          .opA(opA),   //操作数A
          .opB(opB),   //操作数B
          .S(S),       //工作模式选择信号
          .M(M),       //逻辑操作控制信号
          .Cin(Cin),   //进位输入信号
//output
          .DO(DO),         //数据输出
          .C(C),       //进位输出
          .V(V),       //溢出指示输出信号
          .N(N),       //DO符号位输出信号
          .Z(Z)        //DO为全0指示信号
          );

    initial begin
            opA = 32'h0; opB = 32'h0; S = 4'b0; Cin = 1; M = 0; //000010: DO_ex = 0;
        #10 opA = 32'h00000003; opB = 32'h00000002; S = 4'd1; Cin = 1; M = 0; //000110: DO_ex = (!opA_ex)&(!opB_ex);
        #10 opA = 32'hF0000030; opB = 32'hF0000020; S = 4'd2; Cin = 1; M = 0; //001010: DO_ex = (!opA_ex)&opB_ex;
        #10 opA = 32'hF0000000; opB = 32'h0FF0C0E1; S = 4'd3; Cin = 1; M = 0; //001110: DO_ex = !opA_ex;
        #10 opA = 32'hFFFFFFFF; opB = 32'h0FF0C0E1; S = 4'd3; Cin = 1; M = 0;
        #10 opA = 32'hF0000000; opB = 32'hF0DF30FF; S = 4'd4; Cin = 1; M = 0; //010010: DO_ex = opA_ex&(!opB_ex);
        #10 opA = 32'hFF0C0E10; opB = 32'hFF0C0E10; S = 4'd5; Cin = 1; M = 0; //010110: DO_ex = !opB_ex;
        #10 opA = 32'hFF0C0E10; opB = 32'hFFFFFFFF; S = 4'd5; Cin = 1; M = 0;
        #10 opA = 32'h000F0000; opB = 32'h00DF000F; S = 4'd6; Cin = 1; M = 0; //011010: DO_ex = (opA_ex&(!opB_ex))|((!opA_ex)&opB_ex);
        #10 opA = 32'h000F0000; opB = 32'h00DF000F; S = 4'd7; Cin = 1; M = 0; //011110: DO_ex = (!opA_ex)|(!opB_ex);
        #10 opA = 32'h000C0010; opB = 32'h1000300F; S = 4'd8; Cin = 1; M = 0; //100010: DO_ex = opA_ex&opB_ex;
        #10 opA = 32'h000F0000; opB = 32'h00001000; S = 4'd9; Cin = 1; M = 0; //100110: DO_ex = (opA_ex&opB_ex)|((!opA_ex)&(!opB_ex));
        #10 opA = 32'hFFFFE0FF; opB = 32'h70F0C0E0; S = 4'd10; Cin = 1; M = 0; //101010: DO_ex = opB_ex;
        #10 opA = 32'h000F0000; opB = 32'h0000C000; S = 4'd11; Cin = 1; M = 0; //101110: DO_ex = (opA_ex&opB_ex)|((!opA_ex)&opB_ex)|((!opA_ex)&(!opB_e
        #10 opA = 32'hFFFFE0FF; opB = 32'h70F0C0E0; S = 4'd12; Cin = 1; M = 0; //110010: DO_ex = opA_ex;
        #10 opA = 32'h000F0000; opB = 32'h00D00000; S = 4'd13; Cin = 1; M = 0; //110110: DO_ex = (opA_ex&opB_ex)|(opA_ex&(!opB_ex))|((!opA_ex)&(!opB_e
        #10 opA = 32'h000F0000; opB = 32'h0000000F; S = 4'd14; Cin = 1; M = 0; //111010: DO_ex = (opA_ex&opB_ex)|((!opA_ex)&opB_ex)|(opA_ex&(!opB_ex))
        #10 opA = 32'h000F0000; opB = 32'hFFFFE0FF; S = 4'd15; Cin = 1; M = 0; //111110: DO_ex = {33{1}};
        #10 opA = 32'h000000F0; opB = 32'h00000F00; S = 4'd9; Cin = 0; M = 1; //100101: DO_ex = opA_ex + opB_ex + Cin;
        #10 opA = 32'h80000000; opB = 32'h80000000; S = 4'd9; Cin = 0; M = 1;
        #10 opA = 32'h00000011; opB = 32'h00000001; S = 4'd6; Cin = 1; M = 1; //011011: DO_ex = opA_ex - opB_ex - Cin;
        #10 opA = 32'h00000001; opB = 32'h00000011; S = 4'd6; Cin = 1; M = 1;

        #10 $finish;
    end

    initial begin
        $monitor("%b %b %b %b %b : %b %b %b %b %b", opA, opB, S, Cin, M, DO, C, V, N, Z);
        $dumpfile("ALU_core.vcd");
        $dumpvars;
    end

endmodule
```

对 ALU 的 18 种功能逐一测试，算术运算部分各选取两组不同种类的数据进行测试。

3. ModelSim 仿真结果与分析：



以最后三行为例：

（1）32'h80000000 + 32'h80000000，结果 DO 为 0，进位位 C 为 1，溢出位 V 为 1，符号位 N 为 0。

（2）32'h00000011 - 32'h00000001，结果 DO 为 32’h00000010，进位位 C 为 0，溢出位 V 为 0，符号位 N 为 0。

（3）32'h00000001 - 32'h00000011， 结果 DO 为 -32’hFFFFFFF0，进位位 C 为 1，溢出位 V 为 0，符号位 N 为 1。

由波形输出和文本输出可知，DO 和标志位输出正确，ALU 设计正确。