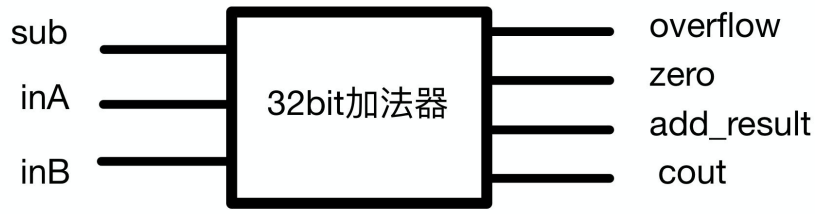
32 位超前进位加法器

1.功能特性

（1）实现有符号、无符号数的加减法。

（2）具有溢出标志（有符号数），进/借位标志（无符号数），零标志位

2.输入输出端口说明

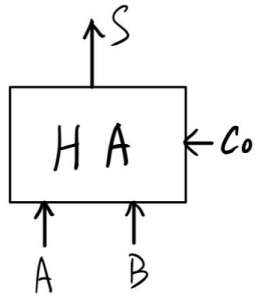


名称	类型	描述
inA	输入/wire	32bit/操作数 0
inB	输入/wire	32bit/操作数 1
sub	输入/wire	1bit/减法标志位
overflow	输出/wire	1bit/溢出标志位
zero	输出/wire	1bit/置零标志位
cout	输出/reg	1bit/进位标志位
add_result	输出/wire	32bit/运算结果

3、设计思路

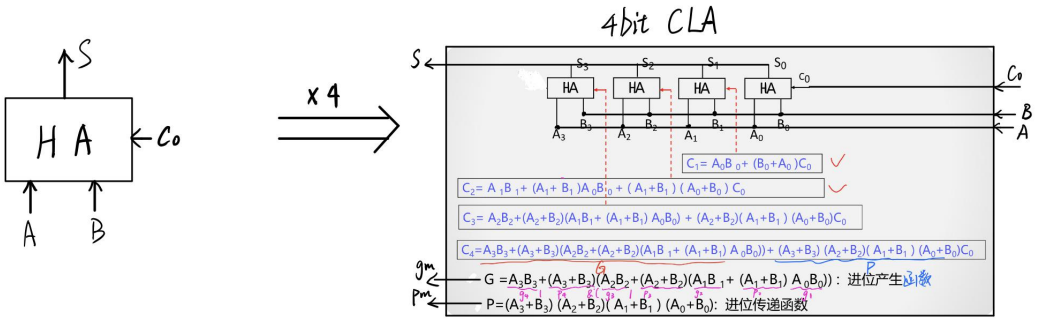
我们采取了自底向上的思路：

（1）我们先通过门级实现了半加器（因为每一位的进位会先行产生，所以每一位的加法不需要向下一级进位，用半加器就可以了）



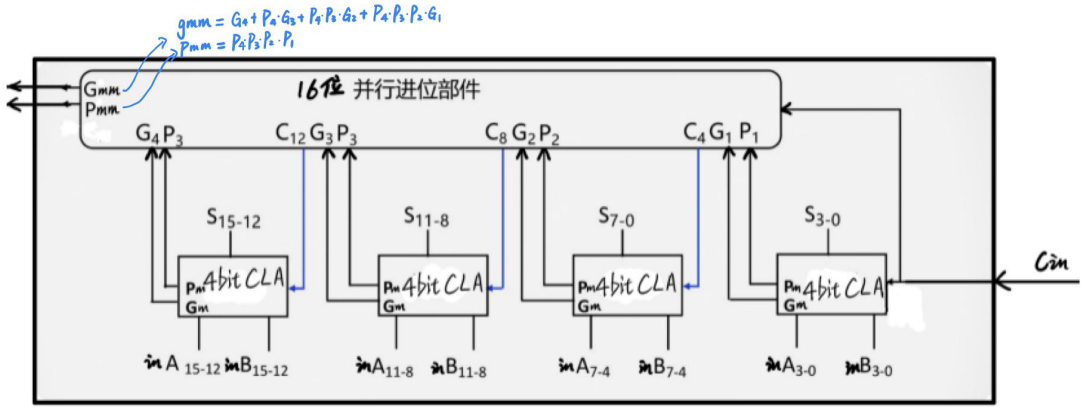

```
131 module half_adder(x,y,c0,f);  //半加器，因为不用向下一位进位
132 input x;
133 input y;
134 input c0;
135 output f;
136 assign f=(x^y)^c0;
137 endmodule
```

（2）随后，将 4 个半加器以及超前进位产生函数 G 和进位传递函数 P 组合，形成了一个 4 位超前进位运算器，结构如下：




```
module aheadAdder_4(gm,pm,f,x,y,c0);    //4位超前进位加法器
input [4:1]x;                           //四位x值
input [4:1]y;                           //四位y值
input c0;                               //上一的级进位
output [4:1]f;                          //四位加和f
output gm,pm;

wire [3:1]c;                            //超前进位
wire [4:1]p;
wire [4:1]g;
assign p=x|y;  // P，进位传递函数中的每一和项(A3+B3)，(A2+B2)，(A1+B1)，(A0+B0)
assign g=x&y;  //G，  进位产生函数中的积项，  A3B3,A2B2,A1B1,A0B0

assign pm=p[4]&p[3]&p[2]&p[1];  //生成下一个单元的G和P
assign gm=g[4]|p[4]&g[3]|p[4]&p[3]&g[2]|p[4]&p[3]&p[2]&g[1];

// 产生超前进位
assign c[1]=g[1]|p[1]&c0;
assign c[2]=g[2]|p[2]&g[1]|p[2]&p[1]&c0;
assign c[3]=g[3]|p[3]&g[2]|p[2]&p[3]&g[1]|p[3]&p[2]&p[1]&c0;

//得到计算结果
half_adder H1(x[1],y[1],c0,f[1]);
half_adder H2(x[2],y[2],c[1],f[2]);
half_adder H3(x[3],y[3],c[2],f[3]);
half_adder H4(x[4],y[4],c[3],f[4]);
endmodule
```

(3) 随后，将 4 个超前进位运算器和 16 位并行进位部件组合，构造了 16 位并行运算单元，结构如下：

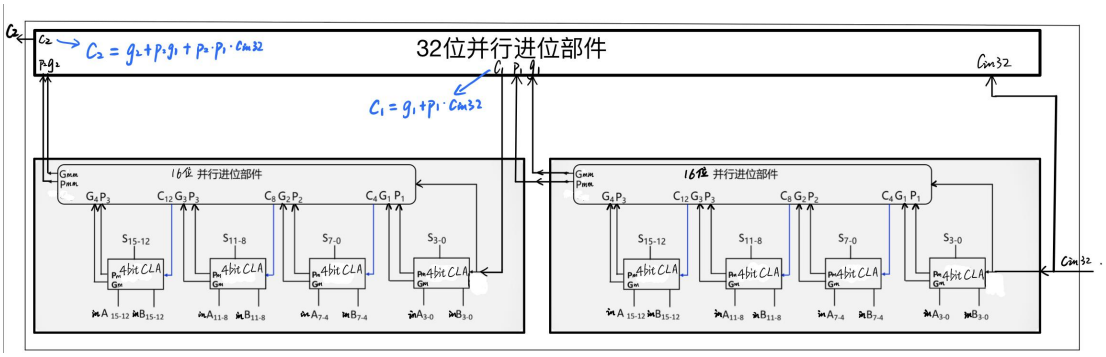



```
78 module aheadAdder_16(gmm, pmm, f16, x16, y16, cin);//16位加法器先行进位加法器
79  input [16:1]x16;
80  input [16:1]y16;
81  input cin;
82  output [16:1]f16;
83  output gmm, pmm;
84  wire [4:1]c;
85  wire [4:1]p;
86  wire [4:1]g;
87
88  aheadAdder_4 ah_1(g[1], p[1], f16[4:1], x16[4:1], y16[4:1], cin);
89  aheadAdder_4 ah_2(g[2], p[2], f16[8:5], x16[8:5], y16[8:5], c[1]);
90  aheadAdder_4 ah_3(g[3], p[3], f16[12:9], x16[12:9], y16[12:9], c[2]);
91  aheadAdder_4 ah_4(g[4], p[4], f16[16:13], x16[16:13], y16[16:13], c[3]);
92
93  parrallel_carry c14(c, p, g, cin);
94
95  assign pmm=p[4]&p[3]&p[2]&p[1];
96  assign gmm=g[4]|p[4]&g[3]|p[4]&p[3]&g[2]|p[4]&p[3]&p[2]&g[1];
97 endmodule
```

16 位并行进位部件：

```
140 module parrallel_carry(c, p, g, c0);//并行进位部件，用于把4个4位超前进位加法器->16位超前进位加法器。根据G，P和C0，生成每一位
141  input [4:1]p;
142  input [4:1]g;
143  input c0;
144  output [4:1]c;
145  assign c[1]=g[1]|p[1]&c0;  // C4进位
146  assign c[2]=g[2]|p[2]&g[1]|p[2]&p[1]&c0; //C8进位
147  assign c[3]=g[3]|p[3]&g[2]|p[2]&p[3]&g[1]|p[3]&p[2]&p[1]&c0; // C12进位
148  assign c[4]=g[4]|p[4]&g[3]|p[4]&p[3]&g[2]|p[4]&p[3]&p[2]&g[1]|p[4]&p[3]&p[2]&p[1]&c0; // C16进位
149 endmodule
```

（4）将两个 16 位的超前进位加法器和 32 位并行进位部件组合，构成 32 位超前进位加法器，结构如下：




```
58  module aheadAdder_32(f32,c2,x32,y32,cin32);//32位超前进位加法器
59  input [32:1]x32;
60  input [32:1]y32;
61  input cin32;//进位输入
62  output [32:1]f32;
63  output c2;   //进位输出
64  wire c1;
65  wire [2:1]p;
66  wire [2:1]g;
67
68  // 生成两个16位的加法器的进位
69  assign c1=g[1]|p[1]&cin32;  // C16 = G0+P0*C0
70  assign c2=g[2]|p[2]&g[1]|p[2]&p[1]&cin32; // C32 = G1 + P1*G0 + P1*P0*C0
71
72  aheadAdder_16 ah1(g[1],p[1],f32[16:1],x32[16:1],y32[16:1],cin32);
73  aheadAdder_16 ah2(g[2],p[2],f32[32:17],x32[32:17],y32[32:17],c1);
74
75  endmodule
```

（5）最后，我们在 32 位超前进位加法电路的基础上，加入了求补运算，判断零标志，溢出标志，进/借位标志的语句，最终形成了符合需求的 32 位加法运算模块：

1>求补运算实现：当 sub 为 1 时做减法，需要求出减数的补码，即取反再加上 1（sub 输入最低位进位）

2>零标志：将结果的每一位相与，最后取反。

3>溢出标志：对于有符号运算，使用 $\mathbf{V = X_0 Y_0 \overline{S}_0 + \overline{X}_0 \overline{Y}_0 S_0}$ 判断是否溢出，X0Y0 分别是两个输入的最低位，S0 是计算结果的最低位。

4>进/借位标志：对于无符号运算，做加法（sub=0）时，加法器的进位即为进位标志；做减法（sub=1）时，加法的进位取反即为借位标志。

```
// 带溢出检测功能的32位加减法运算器
module Adder_detectedOverflow(sub, inA, inB, overflow, cout, zero, add_result);
    input sub; //是否做减法
    input [31:0]inA;
    input [31:0]inB;

    output overflow; //有无溢出（仅对有符号运算有用）
    output zero;     //结果是否为0
    output [31:0]add_result; //运算结果
    output reg cout; // 是否有进位（无符号数加法），借位（无符号减法）
    wire [31:0]inB_r; //B取反
    wire C2;  //进位结果
    wire C_r;  //进位结果取反
    reg [31:0]inB_final; //最终的B

    assign inB_r = ~inB;
    assign C_r = ~C2;
    always@(*)
    begin
    if(sub==1)
        begin
            inB_final = inB_r;
            cout = C_r; //如果是无符号减法，那么借位通过把进位取反得到
        end
    else
        begin
            inB_final = inB;
            cout = C2;
       end
   end
   aheadAdder_32 adder(add_result, C2, inA, inB_final, sub); // 如果做减法，那么把B取反后，加上sub（相当于
   assign zero = ~(add_result[0]|add_result[1]|add_result[2]|add_result[3]|add_result[4]|add_result[5]|ad
   assign overflow = inA[31]&inB_final[31]&~add_result[31]|~inA[31]&~inB_final[31]&add_result[31];
endmodule
```