实验报告

曾锦鸿
2021k8009929009

实验目的：

1、熟悉 verilog 编程、调试
2、熟悉简单比较器的工作原理
3、通过简单模块例化、连线实现复杂的数字电路

实验环境：

vivado版本号为2019.2

原理说明：

1、4位无符号数比较器的原理

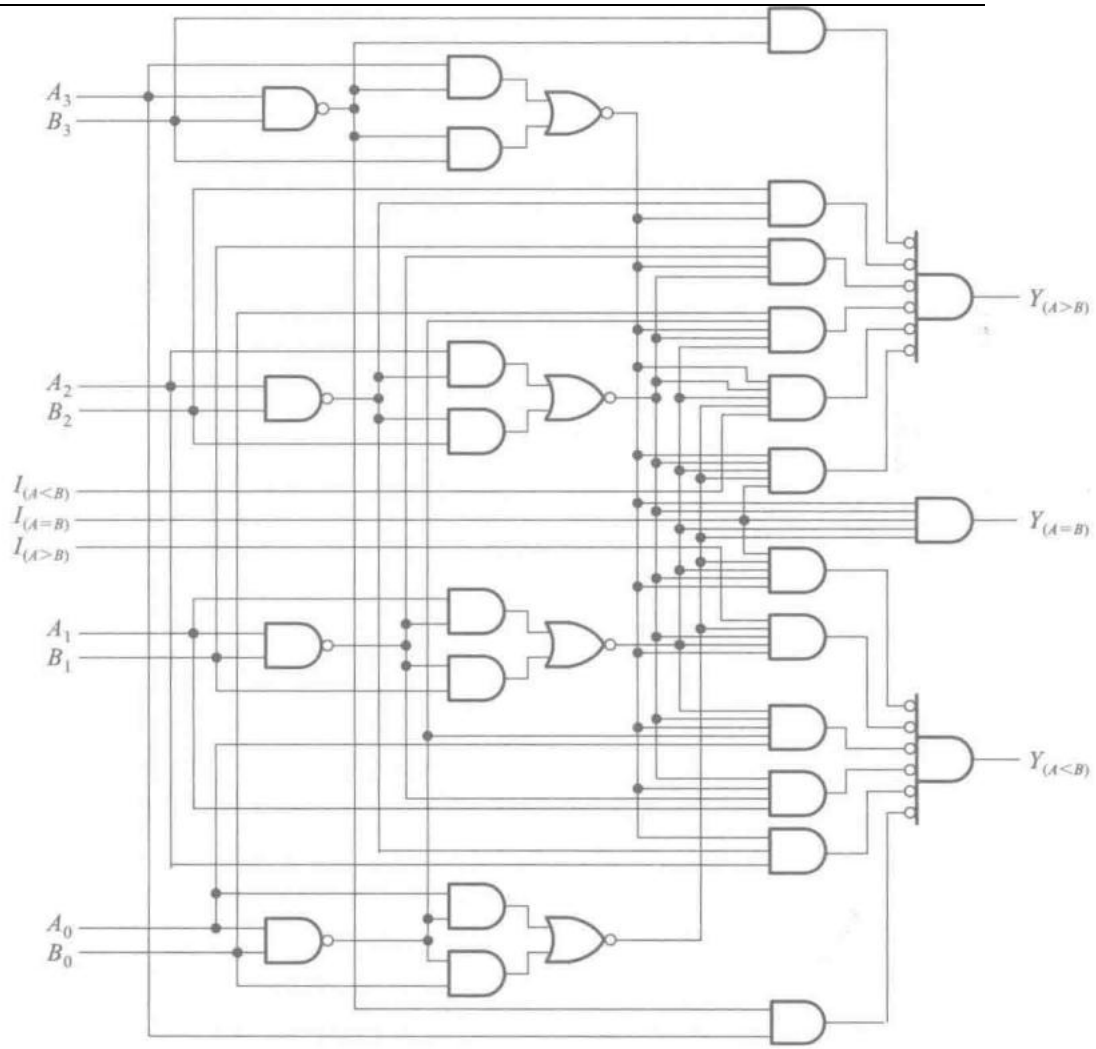
具体逻辑运算如图所示：

$$\begin{aligned}Y_{(A>B)} &= A_3B_3' + (A_3 \odot B_3)A_2B_2' + (A_3 \odot B_3)(A_2 \odot B_2)A_1B_1' \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0B_0' \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)I_{(A>B)}\end{aligned}$$

$$\begin{aligned}Y_{(A<B)} &= A_3'B_3 + (A_3 \odot B_3)A_2'B_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1'B_1 \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0'B_0 \\ &\quad + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)I_{(A<B)}\end{aligned}$$

$$Y_{(A=B)} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)I_{(A=B)}$$

电路图如下所示：



2、16位无符号数比较器的原理

本次实验要求例化4位无符号数比较器，组成一个16位无符号数比较器。所以我们只需要分别以4位为一组，低一级的小组的$Y_{(A>B)}$、$Y_{(A=B)}$、$Y_{(A<B)}$，就是高一级的小组的$I_{(A>B)}$、$I_{(A=B)}$、$I_{(A<B)}$。最高级的小组的$Y_{(A>B)}$、$Y_{(A=B)}$、$Y_{(A<B)}$就是最终的结果。

3、4位超前进位加法器的原理

超前进位加法器的思想是并行计算进位以缩短关键路径。

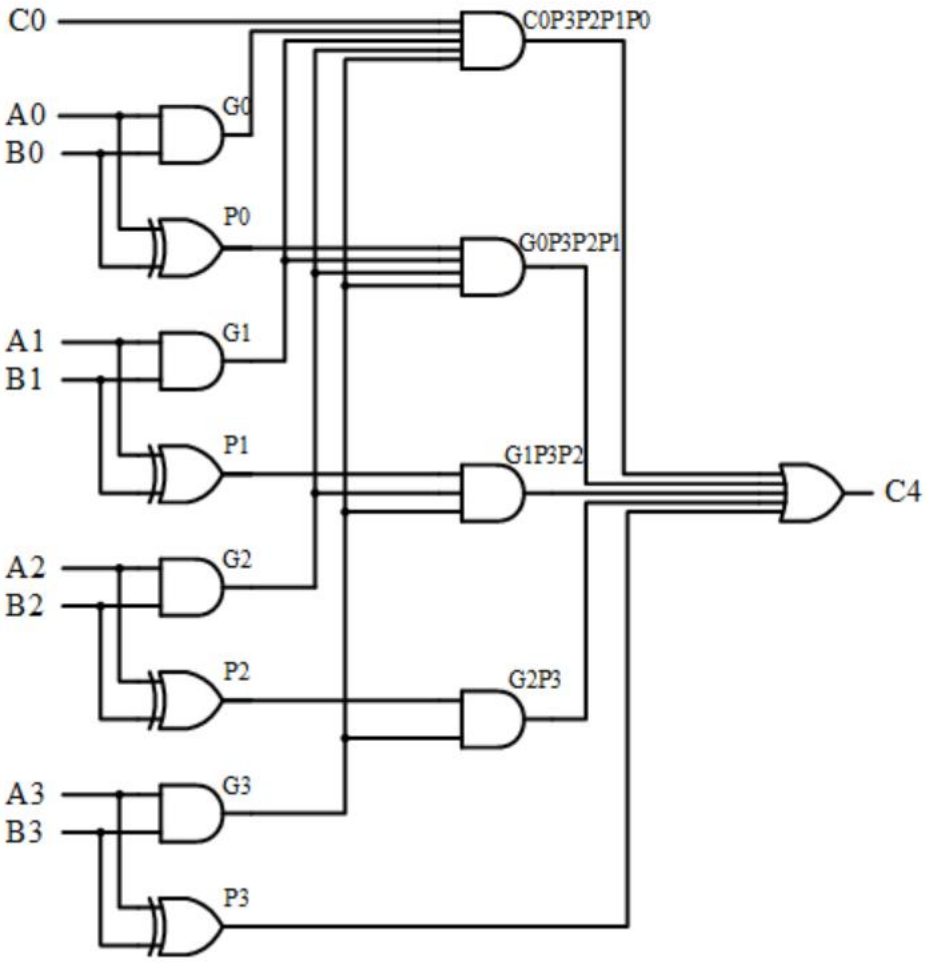
具体逻辑运算如图所示：

$$\begin{cases} P_k = A_k \oplus B_k, & \text{k=0,...,N-1} \\ G_k = A_k B_k & \text{k=0,...,N-1} \end{cases}$$

$$\begin{cases} S_k = P_k \oplus C_k, & \text{k=1,...,N} \\ C_k = G_{k-1} + C_{k-1} P_{k-1} & \text{k=1,...,N} \\ C_N = C_{out} \\ C_0 = C_{in} \end{cases}$$

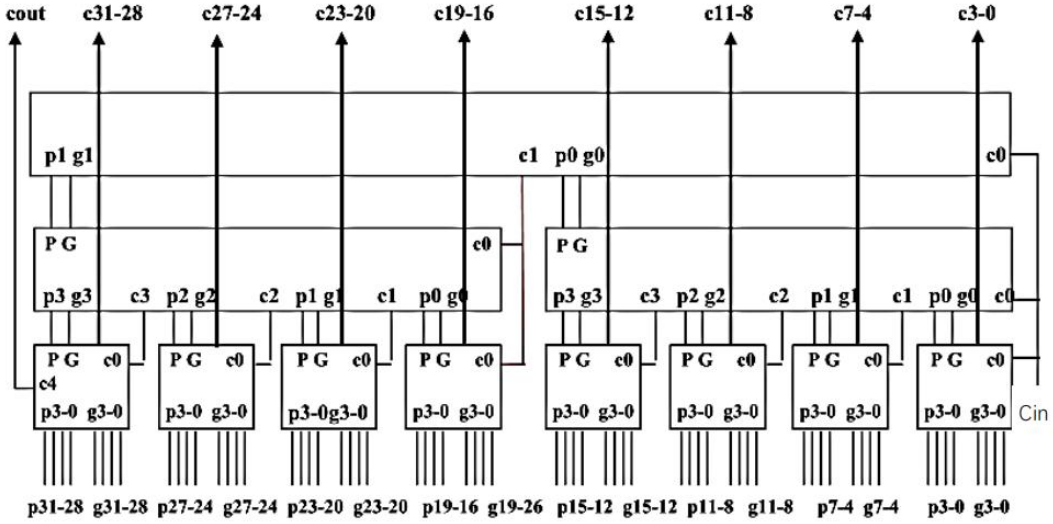
$$\begin{cases} C_1 = G_0 + C_0 P_0 \\ C_2 = G_1 + C_1 P_1 = G_1 + G_0 P_1 + C_0 P_1 P_0 \\ C_3 = G_2 + C_2 P_2 = G_2 + G_1 P_2 + G_0 P_2 P_1 + C_0 P_2 P_1 P_0 \\ C_4 = G_3 + C_3 P_3 = G_3 + G_2 P_3 + G_1 P_3 P_2 + G_0 P_3 P_2 P_1 + C_0 P_3 P_2 P_1 P_0 \end{cases}$$

电路图如下所示：



4、32位超前进位加法器的原理：

本次实验要求例化4位超前进位加法器，组成32位超前进位加法器，且讲义中给了对应的电路图：



这里我们可以看到，4位超前进位加法器模块中除了本位生成的p,g,还需要output出之前4位超前进位未设计的P、G。那么这个P、G究竟为何物呢？

下面我解释一下P、G的含义：

P表示进位的传递信号

G表示进位的生成信号

它们的逻辑表达式为：

$$P = p_0 \,\&\, p_1 \,\&\, p_2 \,\&\, p_3$$

$$G = (p_3 \,\&\, p_2 \,\&\, p_1 \,\&\, g_0) \,|\, (p_3 \,\&\, p_2 \,\&\, g_1) \,|\, (p_3 \,\&\, g_2) \,|\, g_3$$

接口定义：

1、4位无符号数比较器接口：

名称	方向	位宽	功能描述
A	IN	4	第一个比较数
B	IN	4	第二个比较数
in_A_G_B	IN	1	低位大于输入
in_A_E_B	IN	1	低位等于输入
in_A_L_B	IN	1	低位小于输入
out_A_G_B	OUT	1	输出大于
out_A_E_B	OUT	1	输出等于
out_A_L_B	OUT	1	输出小于

2、16位无符号数比较器接口：

名称	方向	位宽	功能描述
A	IN	16	第一个比较数
B	IN	16	第二个比较数
out_A_G_B	OUT	1	输出大于
out_A_E_B	OUT	1	输出等于
out_A_L_B	OUT	1	输出小于

3、4位超前进位加法器接口：

名称	方向	位宽	功能描述
A	IN	4	第一个加数
B	IN	4	第二个加数
CIN	IN	1	低位进位输入
SUM	OUT	4	和
COUT	OUT	1	高位进位输出
POUT	OUT	1	高位进位传递信号输出
GOUT	OUT	1	高位进位生成信号输出

4、32位超前进位加法器接口

名称	方向	位宽	功能描述
A	IN	32	第一个加数
B	IN	32	第二个加数
CIN	IN	1	低位进位输入
SUM	OUT	32	和
COUT	OUT	1	高位进位输出

调试过程及结果波形：

本次实验较为简单，写完后并未出过bug，所以无调试过程。

本次实验我在例化完4位无符号数比较器后，直接进行了16位无符号数比较器的实验，而16位无符号数比较器的正确性直接来自于4位无符号数比较器模块，故直接展示4位无符号数比较器的波形。

1、16位无符号数比较器：

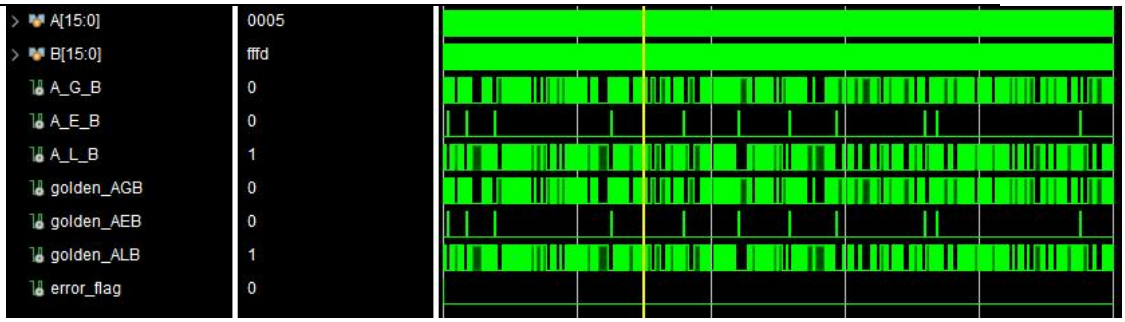
激励文件：

```
module test_comp_16(

    );
    reg [15:0] A;
    reg [15:0] B;
    wire A_G_B;
    wire A_E_B;
    wire A_L_B;
    comp16 u_comp16(
        .A(A),
        .B(B),
        .A_G_B(A_G_B),
        .A_E_B(A_E_B),
        .A_L_B(A_L_B)
    );
    initial begin
        A=16'b1;
        B=16'b0;
    end
    always begin
        #2;
        A=$random()%16;
        B=$random()%16;
    end
    wire golden_AGB;
    wire golden_AEB;
    wire golden_ALB;
    assign golden_AGB = A > B;
    assign golden_AEB = (A == B);
    assign golden_ALB = (A < B);
    wire error_flag;
    assign error_flag = (A_G_B != golden_AGB) | (A_E_B != golden_AEB) | (A_L_B != golden_ALB);
endmodule
```

我并未使用老师们所给的check，而是自己借用vivado中自带的比较符号生成了金标准。然后用error_flag代表自编写的16位无符号数比较器模块的结果是否与金标准一致，若一致则为0，否则为1。所有查看波形时，只需要看其一直未被拉高，则结果正确。

波形图如下所示：



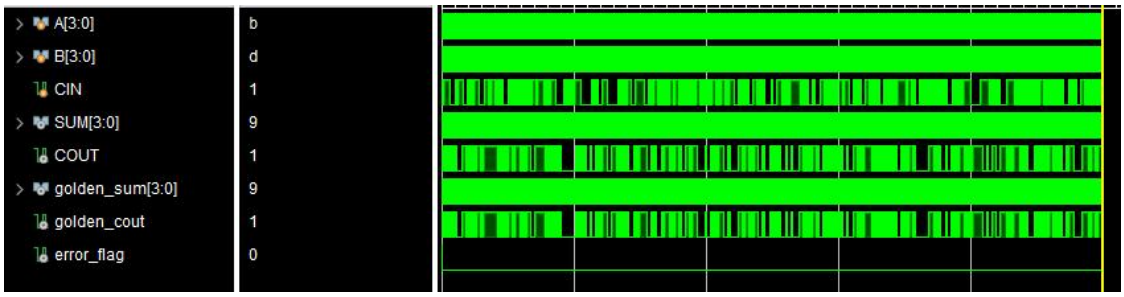
2、4位超前进位加法器：

激励文件：

```
module test_add_4();
    reg [3:0] A;
    reg [3:0] B;
    reg CIN;
    wire [3:0] SUM;
    wire COUT;
    add4 u_add4(
        .A(A),
        .B(B),
        .CIN(CIN),
        .SUM(SUM),
        .COUT(COUT)
    );
    initial begin
        A = 4'h1;
        B = 4'h0;
        CIN = 1'd0;
    end
    always begin
        #2;
        A = $random() % 16;
        B = $random() % 16;
        CIN = $random() % 2;
    end
    wire [3:0] golden_sum;
    wire golden_cout;
    wire error_flag;
    assign {golden_cout,golden_sum} = A + B + CIN;
    assign error_flag = (golden_sum != SUM) || (golden_cout != COUT);
endmodule
```

关于golden_reference的生成就不再赘述，仍然是调用vivado自带的加号。

波形图如下所示：



3、32位超前进位加法器：

激励文件：

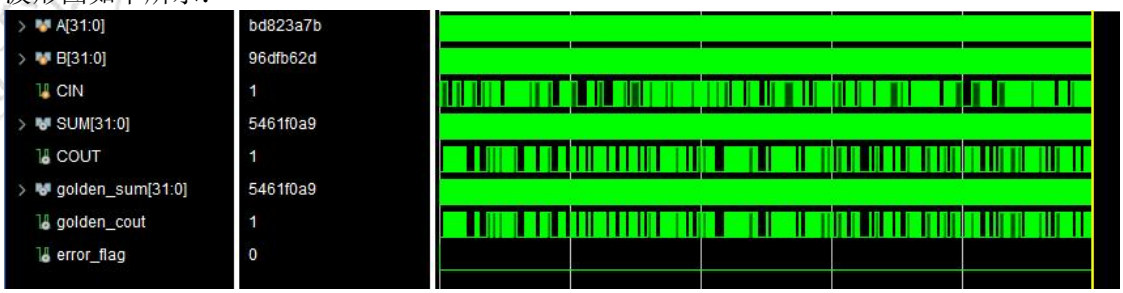
```
module test_add32(

    );
    reg [31:0] A;
    reg [31:0] B;
    reg CIN;
    wire [31:0] SUM;
    wire COUT;
    add32 u_add32(
        .A(A),
        .B(B),
        .CIN(CIN),
        .SUM(SUM),
        .COUT(COUT)
    );
    initial begin
        A = 32'd1;
        B = 32'd0;
        CIN = 1'd0;
    end
    always begin
        #2;
        A = $random() % 33'h100000000;
        B = $random() % 33'h100000000;
        CIN = $random() % 2;
    end
    wire [31:0] golden_sum;
    wire golden_cout;
    wire error_flag;
    assign {golden_cout, golden_sum} = A + B + CIN;
    assign error_flag = (golden_sum != SUM) || (golden_cout != COUT);

endmodule
```

关于golden_reference的生成就不再赘述，仍然是调用vivado自带的加号。

波形图如下所示：



实验总结：

本次实验总体来说并不困难，4位无符号数比较器的逻辑在阎石先生所著的《数字电子技术基础（第六版）》的177页已给出。16位的无符号数比较器只是对4位无符号数比较器模块的组装。4位超前进位加法器的逻辑在阎石先生所著的《数字电子技术基础（第六版）》的175页已给出。唯一需要考虑的是32位超前进位加法器中的P、G的处理，这也是本次实验唯一的难点，其具体公式我以在上述报告中写出，故不再赘述。

源代码：

1、4位无符号数比较器：

```
module comp4(
    input [3:0] A,
    input [3:0] B,
    input in_A_G_B,
    input in_A_E_B,
    input in_A_L_B,
    output out_A_G_B,
    output out_A_E_B,
    output out_A_L_B
    );
assign out_A_G_B = (A[3]&(~B[3])) | (~(A[3]^B[3])&A[2]&(~B[2])) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&A[1]&(~B[1])) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&~(A[1]^B[1])&A[0]&(~B[0])) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&~(A[1]^B[1])&~(A[0]^B[0])&in_A_G_B);
assign out_A_L_B = ((~A[3])&B[3]) | (~(A[3]^B[3])&(~A[2])&B[2]) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&(~A[1])&B[1]) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&~(A[1]^B[1])&(~A[0])&B[0]) |
                   (~(A[3]^B[3])&~(A[2]^B[2])&~(A[1]^B[1])&~(A[0]^B[0])&in_A_L_B);
assign out_A_E_B = ~(A[3]^B[3])&~(A[2]^B[2])&~(A[1]^B[1])&~(A[0]^B[0])&in_A_E_B;

endmodule
```

2、16位无符号数比较器：

```
module comp16(
    input [15:0] A,
    input [15:0] B,
    output A_G_B,
    output A_E_B,
    output A_L_B
    );
    wire [4:0] temp_A_G_B;
    wire [4:0] temp_A_E_B;
    wire [4:0] temp_A_L_B;
    assign temp_A_G_B[0]=0;
    assign temp_A_E_B[0]=1;
    assign temp_A_L_B[0]=0;
    comp4 u_comp4_1(
        .A(A[3:0]),
        .B(B[3:0]),
        .in_A_G_B(temp_A_G_B[0]),
        .in_A_E_B(temp_A_E_B[0]),
        .in_A_L_B(temp_A_L_B[0]),
        .out_A_G_B(temp_A_G_B[1]),
        .out_A_E_B(temp_A_E_B[1]),
        .out_A_L_B(temp_A_L_B[1])
    );
    comp4 u_comp4_2(
        .A(A[7:4]),
        .B(B[7:4]),
        .in_A_G_B(temp_A_G_B[1]),
        .in_A_E_B(temp_A_E_B[1]),
        .in_A_L_B(temp_A_L_B[1]),
        .out_A_G_B(temp_A_G_B[2]),
        .out_A_E_B(temp_A_E_B[2]),
        .out_A_L_B(temp_A_L_B[2])
    );
    comp4 u_comp4_3(
        .A(A[11:8]),
        .B(B[11:8]),
        .in_A_G_B(temp_A_G_B[2]),
        .in_A_E_B(temp_A_E_B[2]),
        .in_A_L_B(temp_A_L_B[2]),
        .out_A_G_B(temp_A_G_B[3]),
        .out_A_E_B(temp_A_E_B[3]),
        .out_A_L_B(temp_A_L_B[3])
    );
    comp4 u_comp4_4(
        .A(A[15:12]),
        .B(B[15:12]),
        .in_A_G_B(temp_A_G_B[3]),
        .in_A_E_B(temp_A_E_B[3]),
        .in_A_L_B(temp_A_L_B[3]),
        .out_A_G_B(temp_A_G_B[4]),
        .out_A_E_B(temp_A_E_B[4]),
        .out_A_L_B(temp_A_L_B[4])
    );
```

```
    assign A_G_B = temp_A_G_B[4];
    assign A_E_B = temp_A_E_B[4];
    assign A_L_B = temp_A_L_B[4];
endmodule
```

3、4位超前进位加法器：

```
module add4(
    input [3:0] A,
    input [3:0] B,
    input CIN,
    output [3:0] SUM,
    output COUT,
    output POUT,
    output GOUT
    );
    wire [4:0] C;
    wire [3:0] P;
    wire [3:0] G;
    assign C[0] = CIN;
    assign P = A | B;
    assign G = A & B;
    assign C[1] = G[0]|(P[0]&C[0]);
    assign C[2] = G[1]|(P[1]&G[0])|(P[1]&P[0]&C[0]);
    assign C[3] = G[2]|(P[2]&G[1])|(P[2]&P[1]&G[0])|(P[2]&P[1]&P[0]&C[0]);
    assign C[4] = G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0])|(P[3]&P[2]&P[1]&P[0]&C[0]);
    assign POUT = P[3]&P[2]&P[1]&P[0];
    assign GOUT = G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0]);
    assign SUM = A^B^C[3:0];
    assign COUT = C[4];
endmodule
```

4、32位超前进位加法器：

先用4个4位超前进位加法器例化出16位的模块

```
module add16(
    input [15:0] A,
    input [15:0] B,
    input CIN,
    output [15:0] SUM,
    output COUT,
    output POUT,
    output GOUT
    );
    wire C[4:0];
    wire P[3:0];
    wire G[3:0];
    assign C[0] = CIN;
    add4 u_add4_1(
        .A(A[3:0]),
        .B(B[3:0]),
        .CIN(C[0]),
        .SUM(SUM[3:0]),
        .POUT(P[0]),
        .GOUT(G[0])
    );
    add4 u_add4_2(
        .A(A[7:4]),
        .B(B[7:4]),
        .CIN(C[1]),
        .SUM(SUM[7:4]),
        .POUT(P[1]),
        .GOUT(G[1])
    );
    add4 u_add4_3(
        .A(A[11:8]),
        .B(B[11:8]),
        .CIN(C[2]),
        .SUM(SUM[11:8]),
        .POUT(P[2]),
        .GOUT(G[2])
    );
    add4 u_add4_4(
        .A(A[15:12]),
        .B(B[15:12]),
        .CIN(C[3]),
        .SUM(SUM[15:12]),
        .POUT(P[3]),
        .GOUT(G[3])
    );
    assign C[1] = G[0]|(P[0]&C[0]);
    assign C[2] = G[1]|(P[1]&G[0])|(P[1]&P[0]&C[0]);
    assign C[3] = G[2]|(P[2]&G[1])|(P[2]&P[1]&G[0])|(P[2]&P[1]&P[0]&C[0]);
    assign C[4] = G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0])|(P[3]&P[2]&P[1]&P[0]&C[0]);
    assign POUT = P[3]&P[2]&P[1]&P[0];
    assign GOUT = G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0]);
    assign COUT = C[4];
endmodule
```

再用两个16位的模块得到32位，并且自行编写最顶层的32位模块的C处理：

```
module add32(
    input [31:0] A,
    input [31:0] B,
    input CIN,
    output [31:0] SUM,
    output COUT
    );
    wire [2:0] C;
    wire [1:0] P;
    wire [1:0] G;
    assign C[0] = CIN;
    add16 u_add16_1(
        .A(A[15:0]),
        .B(B[15:0]),
        .CIN(C[0]),
        .SUM(SUM[15:0]),
        .POUT(P[0]),
        .GOUT(G[0])
    );
    add16 u_add16_2(
        .A(A[31:16]),
        .B(B[31:16]),
        .CIN(C[1]),
        .SUM(SUM[31:16]),
        .POUT(P[1]),
        .GOUT(G[1])
    );
    assign C[1] = G[0]|(P[0]&C[0]);
    assign C[2] = G[1]|(P[1]&G[0])|(P[1]&P[0]&C[0]);
    assign COUT = C[2];
    endmodule
```