四位超前进位加法器

原理如下：

根据一位全加器的表达式可以推理出，四位全加器每一位的计算通式。

$sum_i = a_i \oplus b_i \oplus cin_i$

$co_i = a_i b_i$

令 $g_i = a_i b_i$

$p_i = a_i \oplus b_i$

则 $sum_i = g_i \oplus cin_i$

1. $p_0 = a_0 \oplus b_0$

   $g_0 = a_0 b_0$

   $sum_0 = p_0 \oplus cin$

   $co_0 = g_0 + p_0 cin$

2. $p_1 = a_1 \oplus b_1$

   $g_1 = a_1 b_1$

   $sum_1 = p_1 \oplus （g_0 + p_0 cin）$

   $co_1 = g_1 + p_1（g_0 + p_0 cin）$

   $= g_1 + p_1 g_0 + p_1 p_0 cin$

3. $p_2 = a_2 \oplus b_2$

   $g_2 = a_2 b_2$

   $sum_2 = p_2 \oplus （g_1 + p_1（g_0 + p_0 cin））$

   $co_2 = g_2 + p_2（g_1 + p_1（g_0 + p_0 cin））$

   $= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 cin$

4. $p_3 = a_3 \oplus b_3$

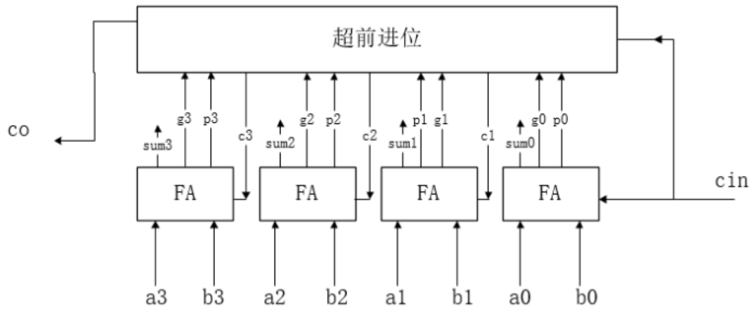
$g_3 = a_3 b_3$

$sum_3 = p_3 \oplus （g_2 + p_2（g_1 + p_1（g_0 + p_0 cin）））$

$co_3 = g_3 + p_3(g_2 + p_2（g_1 + p_1（g_0 + p_0 cin）))$

$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 cin$

为了达到并行的效果，通过公式化简，得到co和sum的表达式，每一位之间的运算都是独立的，和低位的运算没有关系，这样做达到完全并行，这就是四位超前进位全加器。



有了四位全加器，我们可以以通过四位超前进位全加器设计出16位、32位、64位超前进位全加器，有些设计方法是将四个四位超前进位全加器串联起来，即将低四位的co连接到高四位的cin上，但这样并不能算得上是真正的并行，每四位是并行的，但每四位之间是串行的，博主目的是要达到完全并行的设计，继续往下看吧。

这里的g是generation（生成）的意思，当a和b同时为1时，1位全加器必然产生进位，p表示propagation（传到）的意思，当a或b中有一个是1时，如果进位cin位1，1位全加器必然产生进位，利用这个原理即可做出四位超前进位全加器之间的超前进位。

四位超前进位全加器的co进位输出端为

$co_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 cin$

这与一位全加器的进位类似，

$co_0 = g_0 + p_0 cin$

令 $G = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$P = p_3 p_2 p_1 p_0$

此时c_3表示为

$c_3 = G_0 + P_0 cin$

我们可以推出如下关系式

$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$G_1 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4$

$G_2 = g_{11} + p_{11} g_{10} + p_{11} p_{10} g_9 + p_{11} p_{10} p_9 g_8$

$G_3 = g_{15} + p_{15} g_{14} + p_{15} p_{14} g_{13} + p_{15} p_{14} p_{13} g_{12}$

$P_0 = p_3 p_2 p_1 p_0$

$P_1 = p_7 p_6 p_5 p_4$

$P_2 = p_{11} p_{10} p_9 p_8$

$P_3 = p_{15} p_{14} p_{13} p_{12}$

$C_0 = G_0 + P_0 cin$

$C_1 = G_1 + P1G0 + P1P_0 cin$

$C_2 = G_2 + P2G1 + P2P1G0 + P2P1P_0 cin$

$C_3 = G_3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P_0 cin$

在 Verilog 中表示为:

```
module full4_adder3(
        input [3:0] a,b,
        input cin,
        output [3:0] sum,
        output cout
        );

        wire [3:0] G, P;
        wire [3:0] sum;

        assign G[0] = a[0] & b[0];
        assign P[0] = a[0] ^ b[0];
        assign sum[0] = P[0] ^ cin;
        assign P[1] = a[1] ^ b[1];
        assign G[1] = a[1] & b[1];
        assign sum[1] = P[1] ^ (G[0] | (P[0] & cin));
        assign P[2] = a[2] ^ b[2];
        assign G[2] = a[2] & b[2];
        assign sum[2] = P[2] ^ (G[1] | (P[1] & (G[0] | (P[0] & cin))));
        assign P[3] = a[3] ^ b[3];
        assign G[3] = a[3] & b[3];
        assign sum[3] = P[3] ^ (G[2] | P[2] & (G[1] | (P[1] & (G[0] | (P[0] & cin)))));
        assign cout = 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] & cin);
endmodule
```

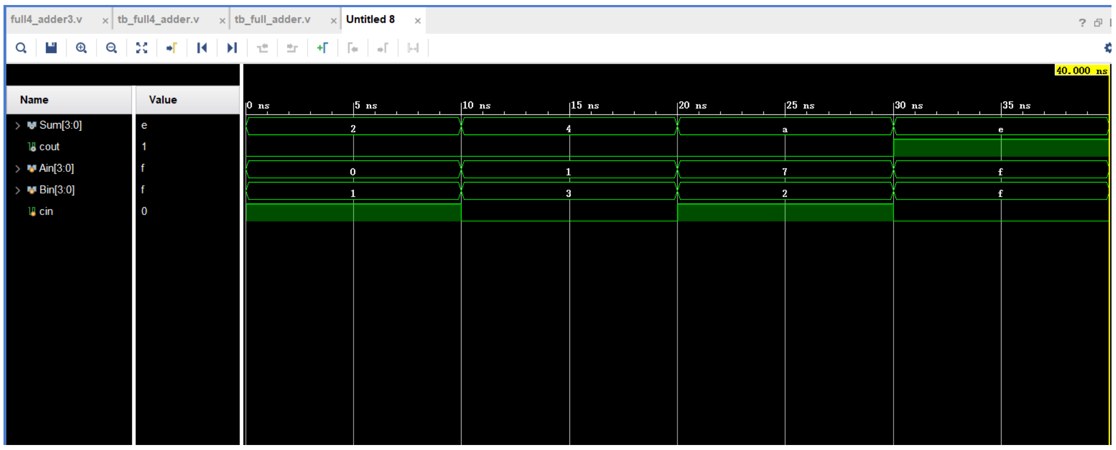
测试文件为:（就用了四个样例，没有全部测试）

```
module tb_full4_adder();
wire [3:0] Sum;
wire cout;
reg [3:0] Ain;
reg [3:0] Bin;
reg cin;

initial
        begin
                //0000 0001 1
                Ain = 4'b0000;
                Bin = 4'b0001;
                cin = 1;
                //0001 00110
                #10
                Ain = 4'b0001;
                Bin = 4'b0011;
                cin = 0;
                //0111 0010 1
                #10
                Ain = 4'b0111;
                Bin = 4'b0010;
                cin = 1;
                //1111 1111 0
                #10
                Ain = 4'b1111;
                Bin = 4'b1111;
                cin = 0;
                #10 $stop;
        end

        full4_adder3 f4adder3(Ain, Bin, cin, Sum, cout);
endmodule
```

仿真结果为:



乘法器

串行乘法器:

```
module multiply1(clk, x, y, result);
    input clk;
    input [7:0] x, y;
    output [15:0] result;
    reg [15:0] result;
    parameter s0 = 0, s1 = 1, s2 = 2;
    reg [2:0] count = 0;
    reg [1:0] state = 0;
    reg [15:0] P, T;
    reg [7:0] y_reg;
    always @(posedge clk) begin
        case (state)
            s0: begin
                count <= 0;
                P <= 0;
                y_reg <= y;
                T <= {{8{1'b0}}, x};
                state <= s1;
            end
            s1: begin
                if(count == 3'b111)
                    state <= s2;
                else begin
                    if(y_reg[0] == 1'b1)
                        P <= P + T;
                    else
                        P <= P;
                    y_reg <= y_reg >> 1;
                    T <= T << 1;
                    count <= count + 1;
                    state <= s1;
                end
            end
            s2: begin
                result <= P;
                state <= s0;
            end
            default: ;
        endcase
    end
```

```
endmodule
```

流水线乘法器:

```
module multiply2(mul_a, mul_b, clk, rst_n, mul_out);
input [3:0] mul_a, mul_b;
input   clk;
input   rst_n;
output [15:0] mul_out;

reg [15:0] mul_out;
reg [15:0] stored0;
reg [15:0] stored1;
reg [15:0] stored2;
reg [15:0] stored3;
reg [15:0] stored4;
reg [15:0] stored5;
reg [15:0] stored6;
reg [15:0] stored7;

reg [15:0] mul_out01;
reg [15:0] mul_out23;

reg [15:0] add01;
reg [15:0] add23;
reg [15:0] add45;
reg [15:0] add67;

always @(posedge clk or negedge rst_n)
begin
    if(!rst_n)
        begin
            mul_out <= 0;
            stored0 <= 0;
            stored1 <= 0;
            stored2 <= 0;
            stored3 <= 0;
            stored4 <= 0;
            stored5 <= 0;
            stored6 <= 0;
            stored7 <= 0;

            add01 <= 0;
            add23 <= 0;
            add45 <= 0;
            add67 <= 0;
        end
        else begin
            stored0 <= mul_b[0]? {8'b0, mul_a} : 16'b0;
            stored1 <= mul_b[1]? {7'b0, mul_a, 1'b0} : 16'b0;
            stored2 <= mul_b[2]? {6'b0, mul_a, 2'b0} : 16'b0;
            stored3 <= mul_b[3]? {5'b0, mul_a, 3'b0} : 16'b0;
            stored4 <= mul_b[0]? {4'b0, mul_a, 4'b0} : 16'b0;
            stored5 <= mul_b[1]? {3'b0, mul_a, 5'b0} : 16'b0;
            stored6 <= mul_b[2]? {2'b0, mul_a, 6'b0} : 16'b0;
            stored7 <= mul_b[3]? {1'b0, mul_a, 7'b0} : 16'b0;
            add01 <= stored1 + stored0;
            add23 <= stored3 + stored2;
            add45 <= stored5 + stored4;
            add67 <= stored7 + stored6;

            mul_out01 <= add01 + add23;
            mul_out23 <= add45 + add67;

            mul_out <= mul_out01 + mul_out23;
        end
end
endmodule
```