Booth 乘法器

一、Booth 乘法器原理：

在计算两个补码相乘时，可以通过 Booth 算法来实现定点补码一位乘的功能。布斯(Booth)算法采用相加和相减的操作计算补码数据的乘积，Booth 算法对乘数从低位开始判断，根据后两个数据位的情况决定进行加法、减法还是仅仅进行移位操作。讨论当相乘的两个数中有一个或二个为负数的情况，在讨论补码乘法运算时，对被乘数或部分积的处理上与原码乘法有某些类似，差别仅表现在被乘数和部分积的符号位要和数值一起参加运算。

Booth 乘法规则如下：

假设 X、Y 都是用补码形式表示的机器数，[X]补和[Y]补=Ys.Y1Y2…Yn，都是任意符号表示的数。比较法求新的部分积，取决于两个比较位的数位，即 Yi+1Yi 的状态。

首先设置附加位 Yn+1=0，部分积初值[Z0]补=0。

当 n≠0 时，判断 YnYn+1，

若 YnYn+1=00 或 11，即相邻位相同时，上次部分积右移一位，直接得部分积。

若 YnYn+1=01，上次部分积加[X]补，然后右移一位得新部分积。

若 YnYn+1=10，上次部分积加[-X]补，然后右移一位得新部分积。

当 n=0 时，判 YnYn+1(对应于 Y0Y1)，运算规则同(1)只是不移位。即在运算的最后一步，乘积不再右移。

二、设计思路

程序首先进行判 0 操作，如果乘数中有一个或两个为 0，则直接输出结果 0，否则进入程序主体。

程序主体分成三个判断模块进行，当乘数最低位与次低位值相等时，先将乘数右移一位，再将原部分累加和右移一位至乘数最高位，同时部分积累加和的最高位根据次高位正负补 0 或 1；如果乘数最低位与次低位分别为 1，0 时，将原部分累加和加上被乘数 X 补后，再右移一位至乘数最高位，同时部分积累加和的最高位根据次高位正负补 0 或 1；如果乘数最低位与次低位分别为 0，1 时，将原部分累加和减去被乘数 X 补后，再右移一位至乘数最高位，同时部分积累加和的最高位根据次高位正负补 0 或 1。

每次比较一次乘数的最末两位，进行相应运算后，共循环 4 次。循环结束后，再进行一次判断，如果乘数最低位与次低位分别为 1、0，将原部分累加和加上被乘数 X 补。如果乘数最低位与次低位分别为 0、1，将原部分累加和减去被乘数 X 补。此时，最终累加和就是乘积的高位结果，取乘数的高四位作为低位结果，拼接即为最终乘法结果。

booth乘法器有个重要的加码运算。来看一下

B[0]	B[-1]	加码结果
0	0	0（无操作）
0	1	1（+被乘数）
1	0	$\underline{1}$（一被乘数）
1	1	0（无操作）

B[-1]就是B的零位右边的位。是假想的位。如0010 0 B[-1]就是0。

做booth乘法器又引入了p空间。

B[0]	B[-1]	加码结果
0	0	无操作，右移一位
0	1	+被乘数，右移一位
1	0	-被乘数，右移一位
1	1	无操作，右移一位

图 1

一个具体的例子（以 7（0111）*2（0010）为例）：

1. 一开始先求出 -1 (被乘数)	A = 0111，$\underline{A}$ = 1001
2. 然后初始化 P 空间，默认为 0	P = 0000 0000 0
3. P 空间的 [4..1] 填入乘数	P = 0000 0010 0
4. 判断 P[1:0]，是 2'b00 亦即“无操作”	P = 0000 0010 0
5. 判断 P[8]，如果是逻辑 0 右移一位补 0，反之右移一位补 1。	P = 0000 0001 0
6. 判断 P[1:0]，是 2'b10 亦即“-被乘数”。	P = 0000 0001 0
7. P 空间的[8..5] 和 被乘数 $\underline{A}$ 相加。	P = 0000 0001 0 + 1001
	P = 1001 0001 0
8. 判断 P[8]，如果是逻辑 0 右移一位，补 0，反之右移一位补 1	P = 1100 1000 1
9. 判断 P[1:0]，是 2'b01 亦即“+被乘数”。	P = 1100 1000 1
10. P 空间的[8..5] 和 被乘数 A 相加。	p = 1100 1000 1 + 0111 P = 0011 1000 1 无视最高位溢出
11. 判断 P[8]，如果是逻辑 0 右移一位补 0，反之右移一位补 1	P = 0001 1100 0
12. 判断 P[1:0]，是 2'b00 亦即“无操作”	P = 0001 1100 0
13. 判断 P[8]，如果是逻辑 0 右移一位，补 0，反之右移一位补 1	P = 0000 1110 0
14. 最终 P 空间的[8..1] 就是最终答案。	P = 0000 1110 0

图 2

三、代码实现

```
module mul_Booth(Mx, My, Mout);
input [5:0] Mx, My;
output reg [9:0] Mout;
reg [5:0] a;
reg [5:0] b, c;
reg [3:0] n;
reg p ,q;
always @(Mx, My) begin
    if (Mx == 0 || My == 0) Mout <= 0;
    else begin
        a = 6'b0;
        n = 4'b1111;
        p = 1'b1;
        q = 1'b0;
        b = Mx;
        c = My;
        c = {c[4:0], q};
        while (n) begin
            n = n >> 1;
            if ((c[0] == 0 && c[1] == 0) || (c[0] == 1 && c[1] == 1)) begin
                c = c >> 1;
                c[5] = a[0];
                a = a >> 1;
                if (a[4] == 1) a = {p, a[4:0]};
                else a = a;
            end
            else if (c[0] == 1 && c[1] == 0) begin
                a = a + b;
                c = c >> 1;
                c[5] = a[0];
                a= a >> 1;
                if (a[4] == 1) a = {p, a[4:0]};
                else a = a;
            end
            else if (c[0] == 0 && c[1] == 1) begin
                a = a - b;
                c = c >> 1;
                c[5] = a[0];
                a = a >> 1;
                if (a[4] == 1) a = {p, a[4:0]};
                else a = a;
```

```
            end
        end
        if (c[0] == 1 && c[1]   == 0) a = a + b;
        else if (c[0] == 0 && c[1] == 1) a = a - b;
        Mout = {a, c[5:2]};
    end
end
endmodule
```

四、仿真测试

```
module tb_mul_Booth();
reg clk;
reg [5:0] Mx, My;
wire [9:0] Mout;
initial begin
    clk = 0;
    Mx = 0;
    My = 0;
    #10
    Mx = 9;
    My = 6;
    #10
    Mx = 12;
    My = 12;
    #10 repeat(10) @(posedge clk) begin
        Mx <= {$random} % 16;
        My <= {$random} % 16;
    end
    $stop;
end
always #5 clk = ~clk;
mul_Booth Booth(Mx, My, Mout);
endmodule
```

五、仿真结果

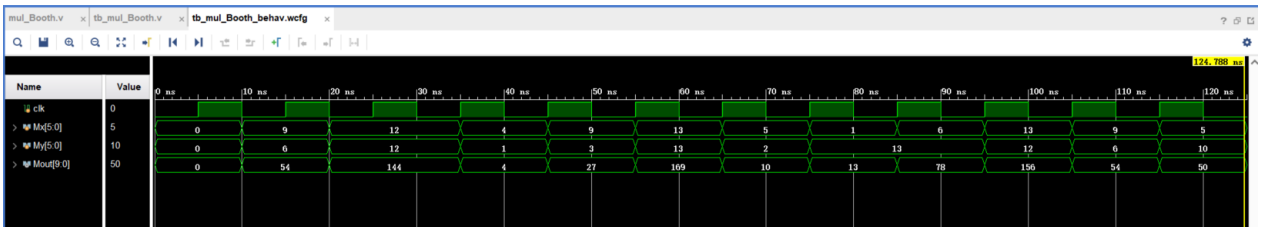
mul_Booth.v
tb_mul_Booth.v
tb_mul_Booth_behav.wcfg
124.788 ns
Name
Value
clk
0
Mx[5:0]
5
My[5:0]
10
Mout[9:0]
50
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
0 9 12 4 9 13 5 1 6 13 9 5
0 6 12 1 3 13 2 13 12 6 10
0 54 144 4 27 169 10 13 78 156 54 50

图 1