

# 使用状态机的方式实现booth乘法器


- 使用状态机的方式实现booth乘法器
  - Booth乘法器原理
    - Booth乘法器设计思路
    - Booth乘法器状态机
    - Booth乘法器部分积生成
    - Booth乘法器部分积压缩
      - 64位超前进位加法器原理

## Booth乘法器原理

课堂上讲解的Booth乘法器原理如图

### Booth 算法:免去特殊操作(减)

X被乘数,Y乘数


$$XY = \sum_{i=0}^{N-1} \underbrace{(-Y_i + Y_{(i-1)})}_{PP_i} 2^i X = \sum_{i=0}^{N-1} PP_i 2^i \quad (3-5)$$

部分积

$$PP_i = (-Y_i + Y_{(i-1)})X$$

对无符号数乘法:

乘数A可表示为

$$A = \sum_{n=0}^{N-1} (A_{[n]} * 2^n) \quad (3-10)$$

$$D = A * B$$

$$= \sum_{n=0}^{N-1} ((A_{[n]} * B) * 2^n) \quad (3-12)$$

式(3-5)、式(3-12)表达形式是相同的,  
无符号数与符号数的乘法就统一起来了



## 经典booth算法

---

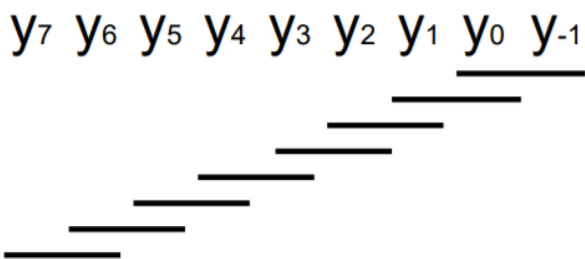
- 经典的Booth算法（1951，A.D. Booth）主要是为了解决有符号运算中复杂的符号修正问题而研究的
  - 对于补码表示的两数相乘无需进行符号位特殊操作
  - 经典Booth算法通过每次在乘数中交叠地取两位 $(Y_{i+1}, Y_i)$ 来产生部分积 $PP_i = (-Y_{i+1} + Y_i)X$ ， $X$ 是被乘数



# 经典booth算法

X被乘数,Y乘数

$$PP_i = (-Y_{i+1} + Y_i)X$$



Booth算法取数操作

| $y_{i+1}$ | $y_i$ | $PP_i$ |
|-----------|-------|--------|
| 0         | 0     | +0     |
| 0         | 1     | +X     |
| 1         | 0     | -X     |
| 1         | 1     | +0     |

表1.1: Booth算法规则

其中： $Y_{-1}=0$ 是附加考察位,帮助分析 $Y_0$   
 $X$ 是被乘数

## Booth乘法器设计思路

Booth乘法器不会加快计算速度，只是单纯解决了有符号运算中复杂的符号修正问题，所以需要进行32次乘加操作。而这32次乘加操作我们可以在一个周期内完成，但这样会造成**关键路径延时过高**；我们也可以分成数级流水线，通过流水线的方式打断关键路径，但是可能会**面积和功耗升高，达不到要求**；所以我们选择使用**状态机**的方式完成Booth乘法器的设计，虽然会降低功耗和面积，也不会有很长时间的**关键路径**，但是**会有很长的输入输出延时**，但是Spec中并没有对此做出要求，所以我们认为这是一个不重要的参数，**无论输入输出延时多长都可以接受**。如下图：

以32 bit×32 bit有符号乘法器设计为例，进行半定制芯片设计流程训练。

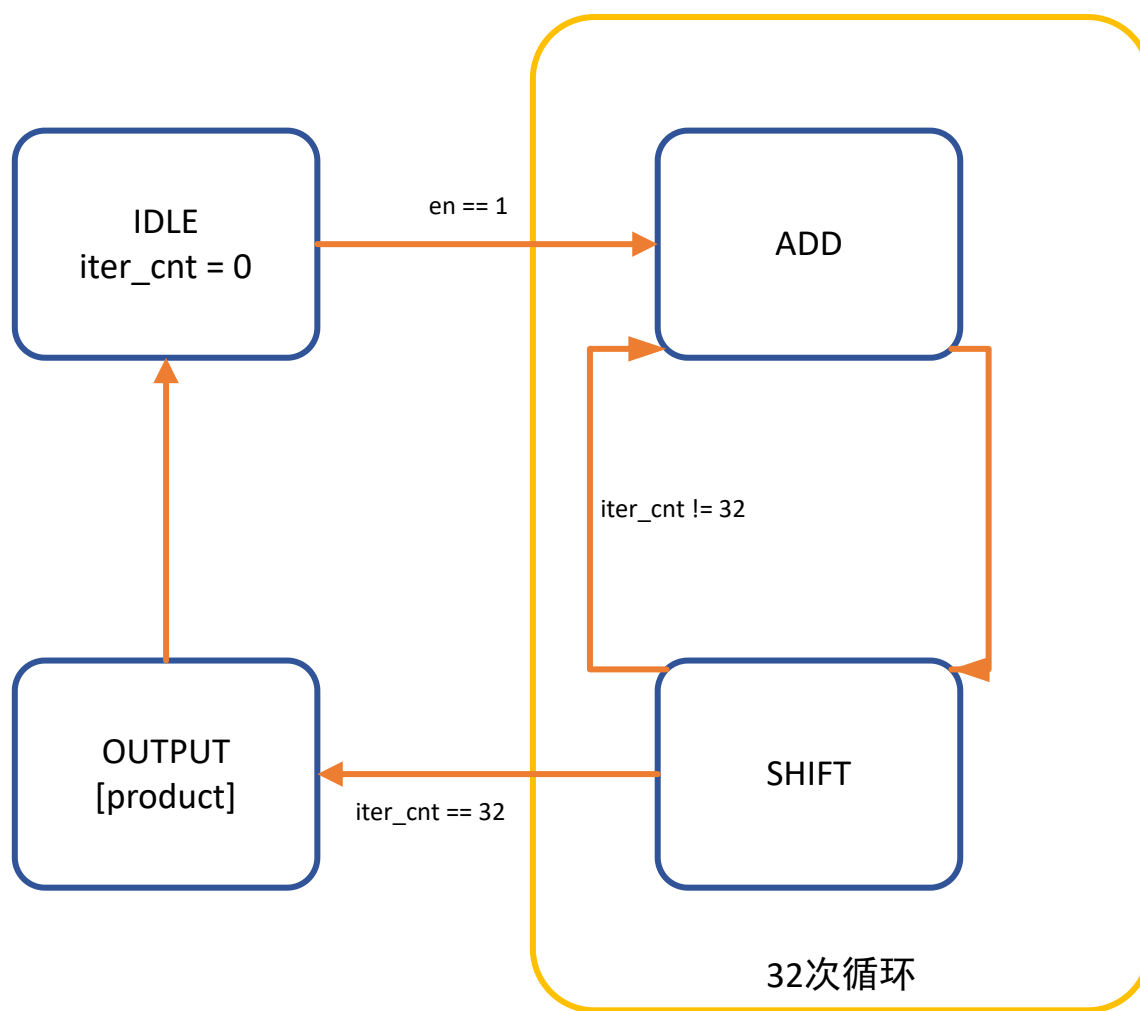
## 一、 参考目标

SMIC 180nm工艺下，在典型情况下，逻辑综合（DC）报告数据：关键路径延迟小于3.6ns，功耗小于32.7mw，面积小于0.2mm<sup>2</sup>

## Booth乘法器状态机

通过Booth算法原理，我们发现Booth算法完成一项部分积的计算需要一次加法和移位操作（移位操作指乘2），如果要完成整个乘法计算，我们需要进行32次加法和移位。

状态机图片如下



## Booth乘法器部分积生成

在状态机的IDLE阶段生成部分积，分别保存在 `a_reg` 和 `s_reg`。 `a_reg` 保存正的部分积，相当于做了加法； `s_reg` 保存负的部分积，相当于做了减法。两个寄存器的宽度为66位，我们将部分积保存在高位，部分积压缩后，进行右移，最终可以发现我们进行的部分积为最低位的部分积。

```

.....
reg  [2*DATAWIDTH+1:0]  a_reg,s_reg,p_reg,sum_reg;  // computational values.
.....
assign multiplier_neg = -{multiplier[DATAWIDTH-1],multiplier};
//取补码，相当于每一位取反，然后加1
.....
always @(posedge clk or negedge rstn) begin
.....
    case (current_state)
    IDLE :  begin
        a_reg    <= {multiplier[DATAWIDTH-1],multiplier,{(DATAWIDTH+1){1'b0}}};
        s_reg    <= {multiplier_neg,{(DATAWIDTH+1){1'b0}}};
        p_reg    <= {{(DATAWIDTH+1){1'b0}},multiplicand,1'b0};
        iter_cnt <= 0;
        done     <= 1'b0;
    end
    endcase
.....
end
.....

```

## Booth乘法器部分积压缩

因为我们采用状态机方式设计，所以，每个状态机循环进行一次部分积压缩，即使用加法器对其应用加法，并将结果进行移位。因为我们将部分积保存在了高位，所以每次压缩完对结果右移；这样做，相当于将部分积保存在低位，每次压缩完，将部分积寄存器左移。

```

.....
adder_66 adder1(
    .A(data_in1),
    .B(data_in2),
    .S(sum),
    .c66()
);
.....
assign data_in1 =      p_reg;
assign data_in2 =      (p_reg[1:0]==2'b01) ? a_reg:// + multiplier
                        (p_reg[1:0]==2'b10) ? s_reg:// - multiplier
                        0;// nop

// algorithm implemenation details.
always @(posedge clk or negedge rstn) begin
.....
    case (current_state)
        IDLE :  begin
.....
            end
        ADD  :  begin
            sum_reg <= sum;
            iter_cnt <= iter_cnt + 1;
        end
        SHIFT :  begin
            p_reg <= {sum_reg[2*DATAWIDTH+1],sum_reg[2*DATAWIDTH+1:1]}; // right shift
        end
        OUTPUT : begin
            product <= p_reg[2*DATAWIDTH:1];
            done <= 1'b1;
        end
    endcase
end
end

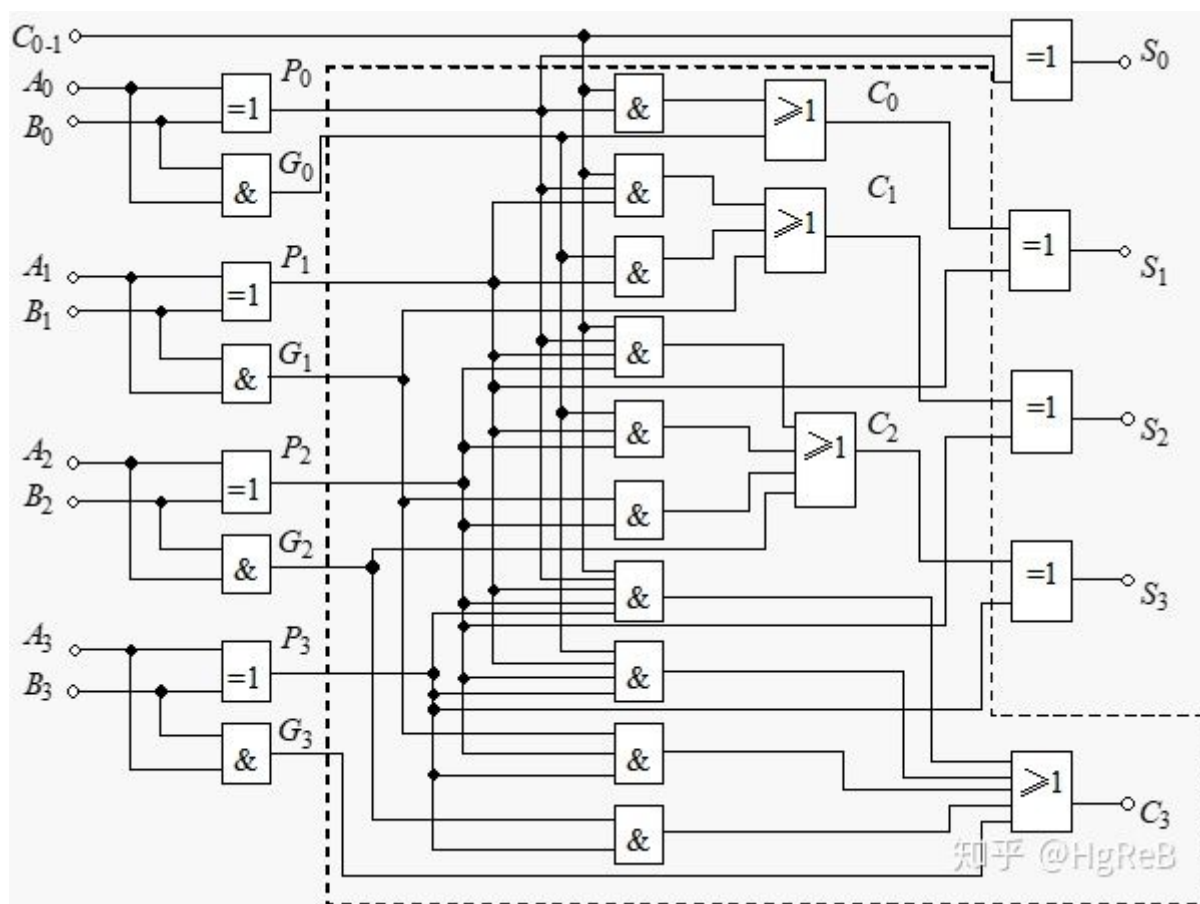
```

另外，我们的加法器为66位加法器，我们66位加法器的实现方式是将64位超前进位加法器和两个1bit全加器级联而成，下面是我们64bit超前进位加法器的设计。

## 64位超前进位加法器原理

64位超前进位加法器由4位超前进位加法器级联而成。

4位超前进位加法器电路图如下



代码如下



//一位全加器

```
module adder(X,Y,Cin,F,Cout);
```

```
    input X,Y,Cin;
```

```
    output F,Cout;
```

```
    assign F = X ^ Y ^ Cin;
```

```
    assign Cout = (X ^ Y) & Cin | X & Y;
```

```
endmodule
```

///四位CLA部件

```
module CLA(c0,c1,c2,c3,c4,p1,p2,p3,p4,g1,g2,g3,g4);
```

```
    input c0,g1,g2,g3,g4,p1,p2,p3,p4;
```

```
    output c1,c2,c3,c4;
```

```
    assign c1 = g1 ^ (p1 & c0),
```

```
           c2 = g2 ^ (p2 & g1) ^ (p2 & p1 & c0),
```

```
           c3 = g3 ^ (p3 & g2) ^ (p3 & p2 & g1) ^ (p3 & p2 & p1 & c0),
```

```
           c4 = g4 ^ (p4 & g3) ^ (p4 & p3 & g2) ^ (p4 & p3 & p2 & g1) ^ (p4 & p3 & p2 & p1 & c0);
```

```
endmodule
```

//四位超前进位加法器

```
module adder_4(x,y,c0,c4,F,Gm,Pm);
```

```
    input [4:1] x;
```

```
    input [4:1] y;
```

```
    input c0;
```

```
    output c4,Gm,Pm;
```

```
    output [4:1] F;
```

```
    wire p1,p2,p3,p4,g1,g2,g3,g4;
```

```
    wire c1,c2,c3;
```

```
    adder adder1(
```

```
        .X(x[1]),
```

```
        .Y(y[1]),
```

```
        .Cin(c0),
```

```
        .F(F[1]),
```

```
        .Cout()
```

```
    );
```

```
    adder adder2(
```

```
        .X(x[2]),
```

```
        .Y(y[2]),
```

```
        .Cin(c1),
```

```
        .F(F[2]),
```

```
        .Cout()
```

```
    );
```

```
    adder adder3(
```

```

        .X(x[3]),
        .Y(y[3]),
        .Cin(c2),
        .F(F[3]),
        .Cout()
    );

```

```

adder adder4(
    .X(x[4]),
    .Y(y[4]),
    .Cin(c3),
    .F(F[4]),
    .Cout()
);

```

```

CLA CLA(
    .c0(c0),
    .c1(c1),
    .c2(c2),
    .c3(c3),
    .c4(c4),
    .p1(p1),
    .p2(p2),
    .p3(p3),
    .p4(p4),
    .g1(g1),
    .g2(g2),
    .g3(g3),
    .g4(g4)
);

```

```

assign p1 = x[1] ^ y[1],
       p2 = x[2] ^ y[2],
       p3 = x[3] ^ y[3],
       p4 = x[4] ^ y[4];

```

```

assign g1 = x[1] & y[1],
       g2 = x[2] & y[2],
       g3 = x[3] & y[3],
       g4 = x[4] & y[4];

```

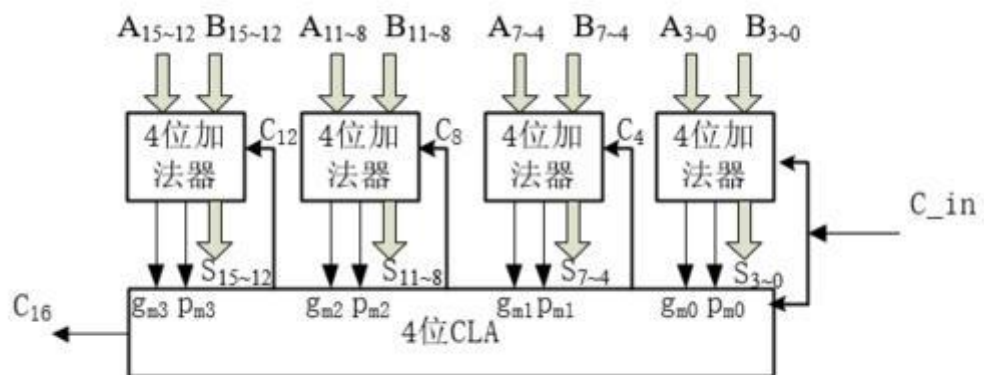
```

assign Pm = p1 & p2 & p3 & p4,
       Gm = g4 ^ (p4 & g3) ^ (p4 & p3 & g2) ^ (p4 & p3 & p2 & g1);

```

```
endmodule
```

通过复用CLA电路可以级联完成16位超前进位加法器



//16位加法器

```
module adder_16(A,B,c0,c16,S,px,gx);  
    input [16:1] A;  
    input [16:1] B;  
    input c0;  
    output c16,gx,px;  
    output [16:1] S;  
  
    wire c4,c8,c12;  
    wire Pm1,Gm1,Pm2,Gm2,Pm3,Gm3,Pm4,Gm4;
```

```
    adder_4 adder1(  
        .x(A[4:1]),  
        .y(B[4:1]),  
        .c0(c0),  
        .c4(),  
        .F(S[4:1]),  
        .Gm(Gm1),  
        .Pm(Pm1)  
    );
```

```
    adder_4 adder2(  
        .x(A[8:5]),  
        .y(B[8:5]),  
        .c0(c4),  
        .c4(),  
        .F(S[8:5]),  
        .Gm(Gm2),  
        .Pm(Pm2)  
    );
```

```
    adder_4 adder3(  
        .x(A[12:9]),  
        .y(B[12:9]),  
        .c0(c8),  
        .c4(),  
        .F(S[12:9]),  
        .Gm(Gm3),  
        .Pm(Pm3)  
    );
```

```
    adder_4 adder4(  
        .x(A[16:13]),  
        .y(B[16:13]),  
        .c0(c12),  
        .c4(),  
        .F(S[16:13]),  
        .Gm(Gm4),  
        .Pm(Pm4)  
    );
```

```

CLA CLA(
    .c0(c0),
    .c1(c4),
    .c2(c8),
    .c3(c12),
    .c4(c16),
    .p1(Pm1),
    .p2(Pm2),
    .p3(Pm3),
    .p4(Pm4),
    .g1(Gm1),
    .g2(Gm2),
    .g3(Gm3),
    .g4(Gm4)
);

assign px = Pm1 & Pm2 & Pm3 & Pm4,
        gx = Gm4 ^ (Pm4 & Gm3) ^ (Pm4 & Pm3 & Gm2) ^ (Pm4 & Pm3 & Pm2 & Gm1);

endmodule

```

最终16位加法器级联完成64位超前进位加法器。

//64位并行进位加法器顶层模块

```
module adder_64(A,B,c0,c64,S,px,gx);
    input [64:1] A;
    input [64:1] B;
    input c0;
    output [64:1] S;
    output c64,px,gx;

    wire px1,gx1,px2,gx2,px3,gx3,px4,gx4;
    wire c16,c32,c48,c64;

    adder_16 adder1(
        .A(A[16:1]),
        .B(B[16:1]),
        .c0(c0),
        .c16(),
        .S(S[16:1]),
        .px(px1),
        .gx(gx1)
    );

    adder_16 adder2(
        .A(A[32:17]),
        .B(B[32:17]),
        .c0(c16),
        .c16(),
        .S(S[32:17]),
        .px(px2),
        .gx(gx2)
    );

    adder_16 adder3(
        .A(A[48:33]),
        .B(B[48:33]),
        .c0(c32),
        .c16(),
        .S(S[48:33]),
        .px(px3),
        .gx(gx3)
    );

    adder_16 adder4(
        .A(A[64:49]),
        .B(B[64:49]),
        .c0(c48),
        .c16(),
        .S(S[64:49]),
        .px(px4),
        .gx(gx4)
    );
endmodule
```

```
CLA CLA(  
    .c0(c0),  
    .c1(c16),  
    .c2(c32),  
    .c3(c48),  
    .c4(c64),  
    .p1(px1),  
    .p2(px2),  
    .p3(px3),  
    .p4(px4),  
    .g1(gx1),  
    .g2(gx2),  
    .g3(gx3),  
    .g4(gx4)  
);
```

```
assign px = px1 & px2 & px3 & px4,  
        gx = gx4 ^ (px4 & gx3) ^ (px4 & px3 & gx2) ^ (px4 & px3 & px2 & gx1);
```

```
endmodule
```