乘累加器

原理：

一位乘法相加，若乘数第 i 位为 1，则加上被乘数的 2 的 i 次幂，若乘数第 i 位为 0，则被乘数加上 0.

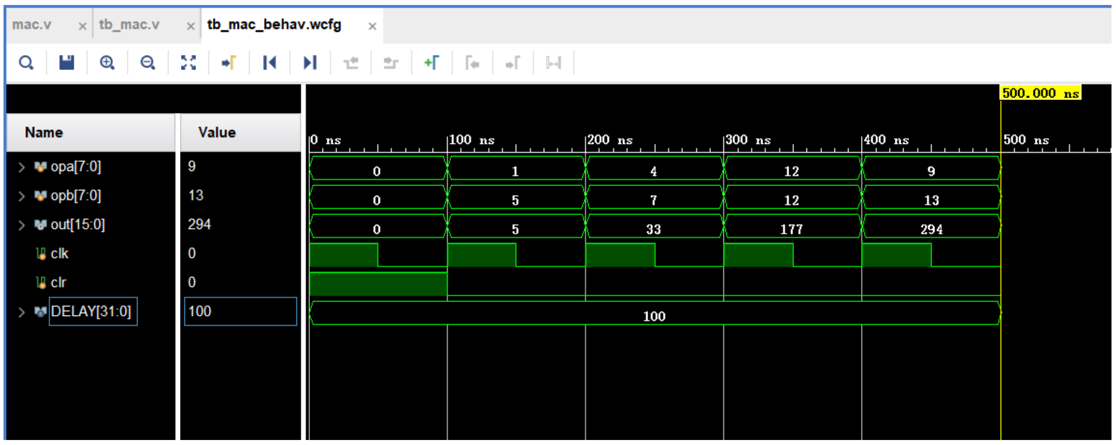
代码：

```
module mac(
	input clk,
	input clr,
	input [7:0] opa,
	input [7:0] opb,
	output [15:0] out
	);

reg [15:0] out;
wire [15:0] sum;
function [15:0] mult;
input [7:0] opa, opb;
reg [15:0] result;
integer index;
begin
	result = opa[0] ? {{8'b0}, opb} : 16'b0;
	for (index = 1; index < 8; index = index + 1)
	begin
		if (opa[index] == 1) result = result + (opb<<index);
		else result = result + {16'b0};
	end;
	mult = result;
end
endfunction
assign sum = mult(opa, opb) + out;
always @(posedge clk or posedge clr)
begin
	if (clr)
		out <= 0;
	else
		out <= sum;
end
endmodule
```

测试代码：

```
module tb_mac();
reg [7:0] opa, opb;
reg clk, clr;
wire [15:0] out;
parameter DELAY = 100;
mac u1(.clk(clk), .clr(clr), .opa(opa), .opb(opb), .out(out));
initial begin
	clk = 1;
	clr = 1;
	opa = 8'd0;
	opb = 8'd0;
	#DELAY clr = 0; opa = 8'd1; opb = 8'd5;
	#DELAY clr = 0; opa = 8'd4; opb = 8'd7;
	#DELAY clr = 0; opa = 8'd12; opb = 8'd12;
	#DELAY clr = 0; opa = 8'd9; opb = 8'd13;
	#DELAY $stop;
end
always #(50) clk = ~clk;
initial $monitor($time,,, "clr = %d opa = %d opb = %d out = %d", clr, opa, opb, out);

endmodule
```

仿真结果：



Name	Value
opa[7:0]	9
opb[7:0]	13
out[15:0]	294
clk	0
clr	0
DELAY[31:0]	100

	0 ns	100 ns	200 ns	300 ns	400 ns
opa[7:0]	0	1	4	12	9
opb[7:0]	0	5	7	12	13
out[15:0]	0	5	33	177	294
DELAY[31:0]	100				

移位相加乘法器

代码 1：（有问题）

```
module mult_shift_add #(parameter WIDTH = 8) (
    input [WIDTH - 1 : 0] S_data1,
    input [WIDTH - 1 : 0] S_data2,
    output reg [2 * WIDTH - 1 : 0] F_mult
    );

    reg [2 * WIDTH - 1 : 0] temp;
    //wire [WIDTH - 1 : 0] S_data1, S_data2;
    reg [2 * WIDTH - 1 : 0] S_data2_temp;
    function [15:0] mult;
    input [WIDTH - 1 : 0] S_data1, S_data2;
    integer index;
    reg [2 * WIDTH - 1 : 0] result;
    begin
//          F_mult = 0;
        S_data2_temp = {{WIDTH{1'b0}}, S_data2}; //Expand
        for (index = 0; index < WIDTH; index = index + 1)
        begin
            temp = {2 * WIDTH{S_data1[index]}} & {S_data2_temp};
            result = result + temp << index;
            $monitor("index = %d", index);
        end
//          $monitor("S_data1 = %d", S_data1);
//          $monitor("S_data2 = %d", S_data2_temp);
    mult = result;
    end
    endfunction
    always @(*) F_mult = mult(S_data1, S_data2);
endmodule
```

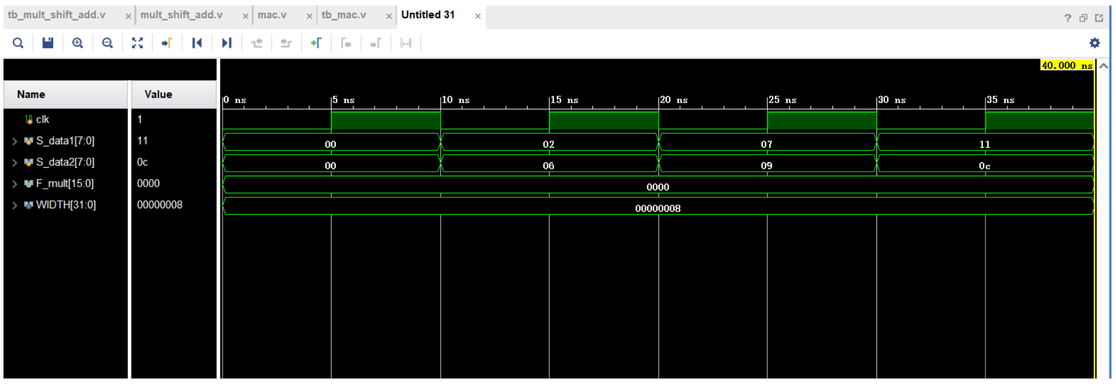
代码 2：（依然有问题）

```
module mult_shift_add #(parameter WIDTH = 8) (
    input [WIDTH - 1 : 0] S_data1,
    input [WIDTH - 1 : 0] S_data2,
    output reg [2 * WIDTH - 1 : 0] F_mult
    );
    reg [2 * WIDTH - 1 : 0] temp;
    //wire [WIDTH - 1 : 0] S_data1, S_data2;
    reg [2 * WIDTH - 1 : 0] S_data2_temp;
    integer index;
    reg [2 * WIDTH - 1 : 0] result;
    always @(*)
    begin
        F_mult = 0;
        S_data2_temp = {{WIDTH{1'b0}}, S_data2}; //Expand
        for (index = 0; index < WIDTH; index = index + 1)
        begin
            temp = {2 * WIDTH{S_data1[index]}} & {S_data2_temp};
            result = result + temp << index;
            $monitor("index = %d", index);
        end
//          $monitor("S_data1 = %d", S_data1);
//          $monitor("S_data2 = %d", S_data2_temp);
    end
endmodule
```

测试代码：（应该没有问题）

```
module tb_mult_shift_add();
parameter WIDTH = 8;
reg clk;
reg [WIDTH - 1 : 0] S_data1,S_data2;
wire [2 * WIDTH - 1 : 0] F_mult;
initial begin
    clk = 0;
    S_data1 = 0;
    S_data2 = 0;
    #10
    S_data1 = 2;
    S_data2 = 6;
    #10
    S_data1 = 7;
    S_data2 = 9;
    #10
    S_data1 = 17;
    S_data2 = 12;
    #10 $stop;
//      #5 repeat (5) @(posedge clk)
//      begin
//              S_data1 <= ($random);
//              S_data2 <= ($random);
//      end
end
always #5 clk = ~clk;
mult_shift_add #(WIDTH) mult_shift_adder(S_data1, S_data2, F_mult);
endmodule
```

仿真结果：（显然也是不对的）



Booth 乘法器（一位补码乘法）

代码：（原理理解，代码实现不了，这个是参考网上的一份代码）

```
module mult_comp(X, Y, out);
input [5:0] X, Y;
output reg [9:0] out;
reg [5:0] a, b, c;
reg [3:0] n;
reg p, q;

always @(X, Y) begin
    if (X == 0 || Y == 0) out <= 0;
    else begin
        a = 6'b0;
        n = 4'b1111;
        p = 1'b1;
        q = 1'b0;
        b = X;
        c = Y;
        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 / 2;
                   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;
         out = {a, c[5:2]};
    end
end
endmodule
```