Problem description

Implementation of Booth’s multiplication algorithm to multiply two eight bit signed numbers.

Literature Survey

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation.

The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth's algorithm is of interest in the study of computer architecture.

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P.

Let m and r be the multiplicand and multiplier, respectively.

Let x and y represent the number of bits in m and r.

The following are the steps to be followed:

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to (x + y + 1).
    - A: Fill the most significant (leftmost) bits with the value of m. Fill the remaining (y + 1) bits with zeros.
    - S: Fill the most significant bits with the value of (−m) in two's complement notation. Fill the remaining (y + 1) bits with zeros.
    - P: Fill the most significant x bits with zeros. To the right of this, append the value of r. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of P.
    - If they are 01, find the value of P + A. Ignore any overflow.

- If they are 10, find the value of P + S. Ignore any overflow.
- If they are 00, do nothing. Use P directly in the next step.
- If they are 11, do nothing. Use P directly in the next step.

3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.
4. Repeat steps 2 and 3 until they have been done y times.
5. Drop the least significant (rightmost) bit from P. This is the product of m and r.

The Booth algorithm has two attractive features:

a) It handles both positive and negative multipliers uniformly.
b) It achieves some efficiency in the number of additions required when the multiplier has a few large blocks of 1s.

Worst case multiplier:

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 => +1 -1+1 -1+1 -1+1 -1+1 -1+1 -1+1 -1+1 -1

Good multiplier:

0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 => 0 0 0 +1 0 0 0 0 -1 0 0 0 +1 0 0 -1

Circuit diagram

Verilog code snippets

File name: **boothsfinal.v**

```
module Adder(a,b,sum); //Adds two 8 bit signed numbers

        input [7:0] a,b;

        output [7:0]sum;

        wire cout;

        wire [7:0] q;

        fa fa1(a[0],b[0],1'b0,sum[0],q[0]);

        fa fa2(a[1],b[1],q[0],sum[1],q[1]);

        fa fa3(a[2],b[2],q[1],sum[2],q[2]);

        fa fa4(a[3],b[3],q[2],sum[3],q[3]);

        fa fa5(a[4],b[4],q[3],sum[4],q[4]);

        fa fa6(a[5],b[5],q[4],sum[5],q[5]);

        fa fa7(a[6],b[6],q[5],sum[6],q[6]);

        fa fa8(a[7],b[7],q[6],sum[7],cout);

endmodule


module subtractor(a,b,sum); // Subtracts two 8 bit signed numbers

        input [7:0] a,b;

        output [7:0]sum;

        wire [7:0] ib;

        wire cout;
        invert b1(ib[0],b[0]);
        invert b2(ib[1],b[1]);
        invert b3(ib[2],b[2]);
        invert b4(ib[3],b[3]);
        invert b5(ib[4],b[4]);
        invert b6(ib[5],b[5]);
        invert b7(ib[6],b[6]);
        invert b8(ib[7],b[7]);
        wire [7:0] q;
        fa fa1(a[0],ib[0],1'b1,sum[0],q[0]);
        fa fa2(a[1],ib[1],q[0],sum[1],q[1]);
        fa fa3(a[2],ib[2],q[1],sum[2],q[2]);
        fa fa4(a[3],ib[3],q[2],sum[3],q[3]);
        fa fa5(a[4],ib[4],q[3],sum[4],q[4]);
        fa fa6(a[5],ib[5],q[4],sum[5],q[5]);
        fa fa7(a[6],ib[6],q[5],sum[6],q[6]);
        fa fa8(a[7],ib[7],q[6],sum[7],cout);
endmodule


module booth_substep(input wire signed [7:0]a,Q,input wire signed q0,input wire signed
[7:0] m,output reg signed [7:0] f8,output reg signed [7:0] l8,output reg cq0);
```

/* To perform 1 of the 3 possible operations on the accumulator and multiplier based on the last bit of multiplier Q[0] and the bit value of Q_{-1} */

```
wire [7:0] addam,subam;

Adder myadd(a,m,addam);

subtractor mysub(a,m,subam);

        always @(*) begin

        if(Q[0] == q0) begin

                 cq0 = Q[0];

                l8 = Q>>1;

                 l8[7] = a[0];

                 f8 = a>>1;

                if (a[7] == 1)

                f8[7] = 1;

        end

        else if(Q[0] == 1 && q0 ==0) begin

                 cq0 = Q[0];

                        l8 = Q>>1;

                 l8[7] = subam[0];

                 f8 = subam>>1;

                if (subam[7] == 1)

                f8[7] = 1;

        end

        else begin

                 cq0 = Q[0];

                        l8 = Q>>1;
```

```
                    l8[7] = addam[0];

                    f8 = addam>>1;

                    if (addam[7] == 1)

                    f8[7] = 1;

            end

end

endmodule


module boothmul(input signed[7:0]a,b,output signed [15:0] c);

// To multiply two 8 bit signed numbers based on Booths algorithm

        wire signed [7:0]Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7;

        wire signed [7:0] m;

        wire signed [7:0] A1,A0,A3,A2;

        wire signed [7:0] A4,A5,A6,A7;

        wire signed[7:0] q0;

        wire qout;


        booth_substep step1(8'b00000000,a,1'b0,b,A1,Q1,q0[1]);

        booth_substep step2(A1,Q1,q0[1],b,A2,Q2,q0[2]);

        booth_substep step3(A2,Q2,q0[2],b,A3,Q3,q0[3]);

        booth_substep step4(A3,Q3,q0[3],b,A4,Q4,q0[4]);

        booth_substep step5(A4,Q4,q0[4],b,A5,Q5,q0[5]);

        booth_substep step6(A5,Q5,q0[5],b,A6,Q6,q0[6]);
        booth_substep step7(A6,Q6,q0[6],b,A7,Q7,q0[7]);

        booth_substep step8(A7,Q7,q0[7],b,c[15:8],c[7:0],qout);

endmodule
```

File name: **boothtbfinal.v**

```
module tb;

wire signed [15:0] z;

reg signed [7:0] a,b;


boothmul my_booth(.a(a),.b(b),.c(z));


initial begin $dumpfile("tb_boothsalgo.vcd"); $dumpvars(0,tb); end

initial

begin

$monitor($time,"      ",a," *",b," = ", z);

a = 8'b11110000;

b = 8'b11110000;

#10

a = 8'b10010101;

b = 8'b100000;

#10

a = 8'b0111;
```

```
b = 8'b0;

#10

b = 8'b1;

a = 8'b1;

#10

a = 8'b00111100;

b = 8'b0101;

#10

a = 8'b10101010;

b = 8'b100011;

#10

a = 8'b010001;

b = 8'b11100;

#10

a = 8'b1000;

b = 8'b10111111;

end

endmodule


//commands to be executed

//iverilog -o booth boothsfinal.v lib.v boothtbfinal.v

//vvp booth

//gtkwave tb_boothsalgo.vcd
```