میان‌ترم DSD

سوال 1

متین باقری **402105727**

در module karatsuba_multiplier ابتدا مقدار قدر مطلق ورودی ها را محاسبه کردیم و با آنها کار می‌کنیم. سپس در انتهای این بخش، علامت خروجی را با توجه به علامت ورودی ها تعیین می‌کنیم. هر ورودی را به ۲ بخش ۶۴ بیتی تقسیم کرده و به ۴ روش ممکن به module karatsuba_multiplier ورودی می‌دهیم. سپس نتایج را که z0 تا z3 هستند به کمک این فرمول جمع کرده:

$$
\begin{aligned}
xy &= \left(x_H b^{\frac{n}{2}} + X_L\right) \times \left(y_H b^{\frac{n}{2}} + Y_L\right) \\
&= x_H y_H b^n + (x_H y_L + x_L y_H) b^{\frac{n}{2}} + x_L y_L
\end{aligned}
$$

و قدر مطلق حاصل نهایی را به دست می‌آوریم.نهایتا همانطور که گفته شد، علامت خروجی را با توجه به علامت ورودی ها تعیین می‌کنیم.

```
module karatsuba_multiplier(
    input signed [127:0] a,
    input signed [127:0] b,
    output signed [255:0] product,
    input wire clk
);
    // get abs values
    wire [127:0] abs_a = a[127] ? -a : a;
    wire [127:0] abs_b = b[127] ? -b : b;
    // result sign
    wire result_sign = a[127] ^ b[127];

    // Split 128-bit inputs into 64-bit halves
    wire [63:0] a_high = abs_a[127:64];
    wire [63:0] a_low = abs_a[63:0];
    wire [63:0] b_high = abs_b[127:64];
    wire [63:0] b_low = abs_b[63:0];

    // Intermediate products needed for Karatsuba
    wire [127:0] z0, z1, z2, z3;

    // Compute the four partial products using 64-bit multipliers
    add_shift_multiplier mult_low     (.a(a_low), .b(b_low), .product(z0),
.clk(clk));
    add_shift_multiplier mult_cross1  (.a(a_low), .b(b_high), .product(z1),
.clk(clk));
    add_shift_multiplier mult_cross2  (.a(a_high), .b(b_low), .product(z2),
.clk(clk));
    add_shift_multiplier mult_high    (.a(a_high), .b(b_high), .product(z3),
.clk(clk));

    // Final product assembly
    wire [255:0] p = (z3 << 128) + (z2 << 64) + (z1 << 64) + z0;
    assign product = result_sign ? -p : p;

endmodule
```

در module add_shift_multiplier از یک شمارنده استفاده کردیم تا در ۶۴ مرحله اگر LSB شیفت راست داده شده ی ورودی اول ۱ بود، حاصل ضرب را با شیفت چپ داده شده ی ورودی دوم جمع کنیم.

```
module add_shift_multiplier (
    input wire [63:0] a,
    input wire [63:0] b,
    output reg [127:0] product,
    input wire clk
);

    reg [63:0] aa;
    reg [127:0] bb;
    reg signed [7:0] counter;

    // Initialize everything at the beginning (only once)
    initial begin
        aa = 0;
        bb = 0;
        product = 0;
        counter = 0;
    end

    // Load inputs on first clock cycle when counter is 0
    always @(posedge clk) begin
        if (counter == 0) begin
            aa <= a;
            bb <= {64'b0, b};
            product <= 0;
            counter <= 64;
        end else if (counter > 0) begin
            if (aa[0]) begin
                product <= product + bb;
            end
            aa <= aa >> 1;
            bb <= bb << 1;
            counter <= counter - 1;
            if (counter == 0) begin
                counter <= counter - 1;
            end
        end
    end

endmodule
```

در نهایت تست بنچی طراحی کردیم که ۱۰۰ ورودی رندوم ۳۲ بیتی و ۲۰ ورودی رندوم ۱۲۸ بیتی و ۱۳ ورودی edge-case تولید کرده و صحت جواب تولید شده را می‌سنجد و تعداد کل محاسبات صحیح را پرینت می‌کند. در طراحی edge-case ها سعی شد از اعداد کوچک صفر و یک تا بزرگترین اعداد مثبت و منفی که در ورودی می‌گنجند استفاده شود.

تست بنچ:

```
module tb_karatsuba_multiplier();

    reg signed [127:0] a, b;
    wire signed [255:0] product;
    reg clk;

    integer i;
    integer correct_32 = 0;
    integer correct_128 = 0;
    integer correct_edge = 0;

    integer pass;
    reg signed [127:0] ra, rb;
    reg signed [127:0] edge_a [0:13];
    reg signed [127:0] edge_b [0:13];

    // Instantiate the multiplier
    karatsuba_multiplier uut (.a(a), .b(b), .product(product), .clk(clk));

    // Clock generation
    initial begin
```

```
        clk = 0;
        forever #1 clk = ~clk;
    end

    // Task to test multiplication
    task test_multiplication;
        input signed [127:0] ta, tb;
        output integer pass_flag;
        begin
            a = ta;
            b = tb;
            #130; // 64 cycles + margin
            if (product === ta * tb) begin
                $display("PASS: a = %0d, b = %0d, product = %0d", ta, tb,
product);
                pass_flag = 1;
            end else begin
                $display("FAIL: a = %0d, b = %0d, expected = %0d, got = %0d", ta,
tb, ta * tb, product);
                pass_flag = 0;
            end
        end
    endtask

    initial begin
        // --------- 100 random 32-bit tests ---------
        $display("\n--- 100 Random 32-bit Tests ---");
        for (i = 0; i < 100; i = i + 1) begin
            ra = $random; // 32-bit signed
            rb = $random;
            test_multiplication(ra, rb, pass);
            correct_32 = correct_32 + pass;
        end

        // --------- 20 random 128-bit tests ---------
        $display("\n--- 20 Random 128-bit Tests ---");
        for (i = 0; i < 20; i = i + 1) begin
            ra = {$random(), $random(), $random(), $random()};
            rb = {$random(), $random(), $random(), $random()};
            if ($random % 2) ra = -ra;
            if ($random % 2) rb = -rb;
            test_multiplication(ra, rb, pass);
            correct_128 = correct_128 + pass;
        end

        // --------- 14 edge case tests ---------
        $display("\n--- 14 Edge Case Tests ---");
        edge_a[0]  = 0;               edge_b[0]  = 0;
        edge_a[1]  = 1;               edge_b[1]  = 1;
        edge_a[2]  = -1;              edge_b[2]  = -1;
        edge_a[3]  = 1;               edge_b[3]  = -1;
        edge_a[4]  = 127;             edge_b[4]  = 127;
        edge_a[5]  = -127;            edge_b[5]  = -127;
        edge_a[6]  = (2**64 - 1);     edge_b[6]  = (2**64 - 1);
        edge_a[7]  = -(2**64 - 1);    edge_b[7]  = (2**64 - 1);
        edge_a[8]  = (2**127 - 1);    edge_b[8]  = 1;
        edge_a[9]  = -(2**127);       edge_b[9]  = 1;
        edge_a[10] = -(2**127);       edge_b[10] = -1;
        edge_a[11] = (2**126);        edge_b[11] = 2;
        edge_a[12] = 1;               edge_b[12] = (2**127 - 1);
        edge_a[13] = (2**127 - 1);    edge_b[13] = (2**127 - 1);

        for (i = 0; i < 14; i = i + 1) begin
            test_multiplication(edge_a[i], edge_b[i], pass);
            correct_edge = correct_edge + pass;
        end

        // --------- Summary ---------
        $display("\n--- TEST SUMMARY ---");
        $display("Random 32-bit tests passed : %0d / 100", correct_32);
        $display("Random 128-bit tests passed: %0d / 20",  correct_128);
        $display("Edge case tests passed     : %0d / 14",  correct_edge);

        $finish;
    end

endmodule
```

خروجی تست بنچ به این صورت است:

```
#
# --- 100 Random 32-bit Tests ---
# PASS: a = 303379748, b = -1064739199, product = -323020309878341852
# PASS: a = -2071669239, b = -1309649309, product = 2713160187332905851
# PASS: a = 112818957, b = 1189058957, product = 134148391340247849
# PASS: a = -1295874971, b = -1992863214, product = 2582501559649216794
# PASS: a = 15983361, b = 114806029, product = 1834986206483469
        …       …       …       …
# PASS: a = 1062902654, b = -309493541, product = -328961506124757814
# PASS: a = -406604081, b = 1022176121, product = -415620982299349801
# PASS: a = -47651590, b = -1329795487, product = 63366869330374330
# PASS: a = 194251031, b = -789219167, product = -153306636874711177
#
# --- 20 Random 128-bit Tests ---
# PASS: a = 134643879490799594544712154506645237370, b =
69111000537326723426710001329100912087, product =
9305373227836405374831746493022015985531414803416541805809752491001617091190
# PASS: a = 58094000079208327242587351101560658196, b = -
17007877478122577396021100391769899607, product = -
988055635561218536233414207903976101488486551680804343490415522467761728972
…       …       …       …       …
# PASS: a = 96080807841733150643962682956863629188, b = -
56680478222256477472982908161207123689, product = -
5445906136450165230607976231855971142115904637171610801301326026015346634532
# PASS: a = 145534277178804962543628503711976108614, b =
156326102477184092322491690810340965582, product =
22750806328196778770079411736114764951305886649368681494413008731583667723348
#
# --- 14 Edge Case Tests ---
# PASS: a = 0, b = 0, product = 0
# PASS: a = 1, b = 1, product = 1
# PASS: a = -1, b = -1, product = 1
# PASS: a = 1, b = -1, product = -1
# PASS: a = 127, b = 127, product = 16129
# PASS: a = -127, b = -127, product = 16129
# PASS: a = 18446744073709551615, b = 18446744073709551615, product =
340282366920938463426481119284349108225
# PASS: a = -18446744073709551615, b = 18446744073709551615, product = -
340282366920938463426481119284349108225
# PASS: a = 170141183460469231731687303715884105727, b = 1, product =
170141183460469231731687303715884105727
# PASS: a = -170141183460469231731687303715884105728, b = 1, product = -
170141183460469231731687303715884105728
# PASS: a = -170141183460469231731687303715884105728, b = -1, product =
170141183460469231731687303715884105728
# PASS: a = 85070591730234615865843651857942052864, b = 2, product =
170141183460469231731687303715884105728
# PASS: a = 1, b = 170141183460469231731687303715884105727, product =
170141183460469231731687303715884105727
```

```
# PASS: a = 170141183460469231731687303715884105727, b =
170141183460469231731687303715884105727, product =
28948022309329048855892746252171976962977213799489202546401021394546514198529
#
# --- TEST SUMMARY ---
# Random 32-bit tests passed : 100 / 100
# Random 128-bit tests passed: 20 / 20
# Edge case tests passed     : 14 / 14
```