# تمرین سری چهارم DSD

**متین باقری** **402105727**

---

خواسته شده که یک 15x15 Array Multiplier بیتی طراحی کنیم.

FullAdder را به صورت رفتاری طراحی کردیم:

```
module FullAdder(input  a, b, cin, output sum, cout);
    assign sum  = a ^ b ^ cin;
    assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

سپس ArrayMultiplier را به صورت ساختاری نوشتیم:

```
module array_multiplier(input  [14:0] a, b, output [29:0] result);
    wire [14:0] pp[14:0];     // partial products
    wire [29:0] sum [14:0];   //internal sum
    wire [29:0] carry [14:0]; //carry arrays

    genvar i, j;
    generate
        // Generate partial product bits
        for (i = 0; i < 15; i = i + 1) begin : GEN_PP
            for (j = 0; j < 15; j = j + 1) begin : GEN_AND
                   and(pp[i][j], a[i], b[j]);
            end
        end

        // Initialize row 0
        for (j = 0; j < 15; j = j + 1) begin : FIRST_ROW_ASSIGN
            buf(sum[0][j], pp[0][j]);
            // assign 0 :
            buf(sum[0][j+15], 0);
            buf(carry[0][2*j], 0);
            buf(carry[0][2*j+1], 0);
        end
        //assign sum[0][29:15]= 15'b0;
        //assign carry[0] = 30'b0;

        // rows 1 to 14
        for (i = 1; i < 15; i = i + 1) begin : GEN_ROWS
            for (j = 0; j < 30; j = j + 1) begin : GEN_COLS
                if (j < i) begin // pass down previous data
                    buf(sum[i][j], sum[i-1][j]);
```

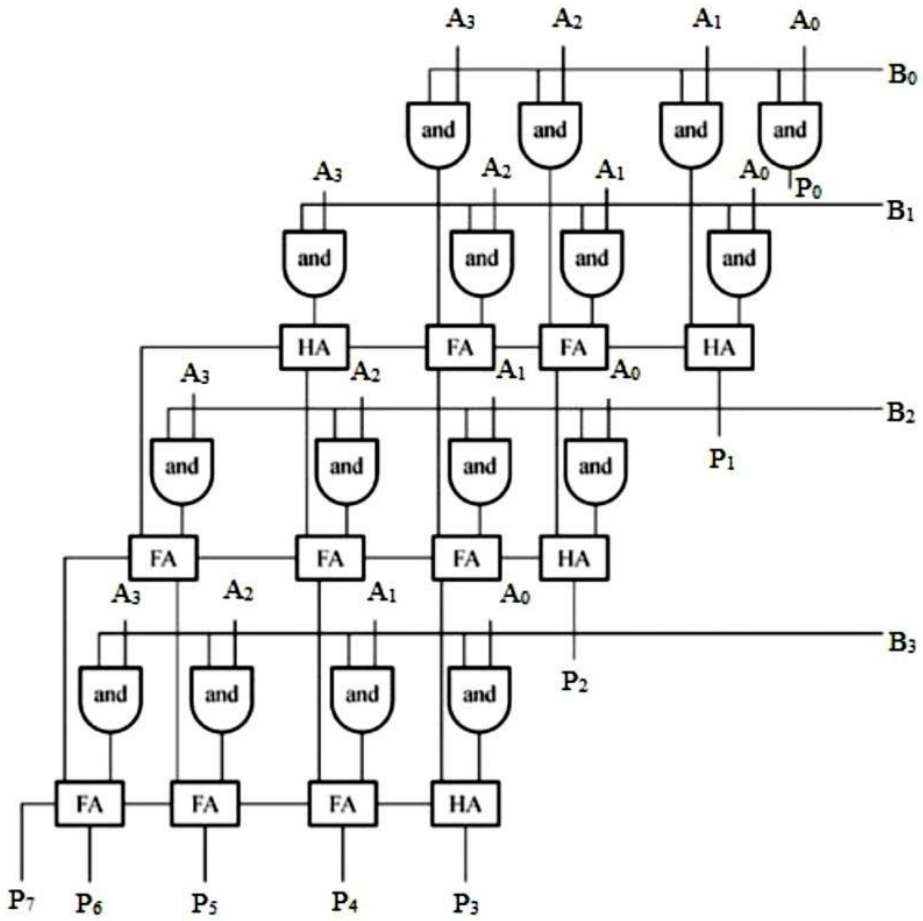
```
                    buf(carry[i][j], 1'b0);
                end
                else if (j < i + 15) begin // use FullAdder
                    FullAdder fa (
                    .a(pp[i][j-i]), .b(sum[i-1][j]), .cin(carry[i-1][j-1]),
                    .sum(sum[i][j]), .cout(carry[i][j])
                    );
                end
                else begin // if no new pp bit is available, combine the previous
sum and carry
                    xor(sum[i][j], sum[i-1][j], carry[i-1][j-1]);
                    and(carry[i][j], sum[i-1][j], carry[i-1][j-1]);
                end
            end
        end
    endgenerate

    //assign result = sum[14] + {carry[14], 1'b0};
    wire [30:0] adder_c;
    assign adder_c[0] = 1'b0;
    FullAdder fa (.a(sum[14][0]), .b(1'b0), .cin(adder_c[0]),
        .sum(result[0]), .cout(adder_c[1]));
    genvar k;
    generate
      for (k = 1; k < 30; k = k + 1) begin : FINAL_ADD
        FullAdder fa (.a   (sum[14][k]), .b   (carry[14][k-1]), .cin
(adder_c[k]),
          .sum (result[k]), .cout(adder_c[k+1]));
      end
    endgenerate

endmodule
```

برای رعایت حالت ساختاری کد، از گیت های buffer, and, or, xor, ... استفاده کردیم. منطق کد به طور خلاصه در تصویر زیر آمده. البته که این تصویر برای 4 بیت است، ما آن را به 15 بیت گسترش دادیم و 15 ردیف از گیت and و FullAdder استفاده کردیم. نهایتا آخرین ردیف sum و carry به دست آمده را با این منطق با هم جمع کردیم تا به جواب نهایی رسیدیم:

```
//assign result = sum[14] + {carry[14], 1'b0};
```




در حین نوشتن کد، برای بررسی صحت جواب ها از این test bench استفاده کردیم:

تعداد محدودی تست کیس به صورت دستی آماده کردیم. سعی شد از اعداد کوچک مانند 0 و 1 تا اعداد بزرگ در این تست کیس ها گنجانده شود.

```
// First Test Bench:
module tb_array_multiplier;
    reg [14:0] a, b;
    wire [29:0] result;

    array_multiplier uut(.a(a), .b(b), .result(result));

    initial begin
        // Test cases and their expected results
        test_case(15'd0,     15'd12345,   30'd0);
        test_case(15'd1,     15'd12345,   30'd12345);
        test_case(15'd123,   15'd456,     30'd56088);       // 123 * 456 = 56088
        test_case(15'd32767, 15'd32767,   30'd1073676289);  // 32767 * 32767 =
1,073,676,289
```

```
        test_case(15'd32767, 15'd1,       30'd32767);
        test_case(15'd2345,  15'd6789,    30'd15920205);     // 2345 * 6789 =
15,920,205
        test_case(15'd64,    15'd128,     30'd8192);
        test_case(15'd1,     15'd1,       30'd1);
        test_case(15'd12345, 15'd678,     30'd8369910);      // 12345 * 678 =
8,369,910
        test_case(15'd30000, 15'd20000,   30'd600000000);    // 30000 * 20000 =
600,000,000
    end

    task test_case;
        input [14:0] a_in, b_in;
        input [29:0] expected;
        begin
            a = a_in;
            b = b_in;
            #1;
            $display("a = %5d, b = %5d, result = %10d (Expected: %10d) %s",
                     a, b, result, expected,
                     (result === expected) ? "PASS" : "FAIL");
        end
    endtask
endmodule
```

بعد از تکمیل کد به کمک test bench بالا، test bench دیگری نوشتیم که 100 ورودی رندوم تولید کرده و عملکرد ضرب کننده را با آن‌ها می‌سنجد:

```
// Second Test Bench:
module tb_random;
    reg  [14:0] a, b;
    wire [29:0] result;
    integer     i;
    integer     pass_count;
    reg  [29:0] expected;

    array_multiplier uut (.a(a), .b(b), .result(result));

    initial begin
        pass_count = 0;
        $display("Starting random testbench...");

        // Loop over 100 random test cases
        for (i = 0; i < 100; i = i + 1) begin
            // generate two random 15-bit numbers
            a = $urandom_range(0, 15'h7FFF);
            b = $urandom_range(0, 15'h7FFF);
            expected = a * b;

            #1;  // wait a delta cycle for result to settle
```

```
            if (result === expected) begin
                pass_count = pass_count + 1;
                $display("Test %0d: a=%5d, b=%5d => result=%10d (exp=%10d) PASS",
                         i, a, b, result, expected);
            end else begin
                $display("Test %0d: a=%5d, b=%5d => result=%10d (exp=%10d) FAIL",
                         i, a, b, result, expected);
            end
        end

        // Summary
        $display("Random testing complete: %0d out of 100 tests passed.",
pass_count);
    end
endmodule
```

نهایتا برای اینکه حالت های خاص بیشتری را بررسی کنیم، یک test bench با unit coverage نوشتیم که به این صورت است:

```
// Third Test Bencch:
module tb_unit_coverage;
    reg  [14:0] a, b;
    wire [29:0] result;
    reg  [29:0] expected;
    integer     pass_count;
    integer     total_tests;

    array_multiplier uut(.a(a), .b(b), .result(result));

    task test_case;
        input [14:0] a_in, b_in;
        input [29:0] expected_in;
        begin
            a = a_in;
            b = b_in;
            expected = expected_in;
            #1;
            total_tests = total_tests + 1;
            if (result === expected) begin
                pass_count = pass_count + 1;
                $display("PASS: a = %5d, b = %5d, result = %10d", a, b, result);
            end else begin
                $display("FAIL: a = %5d, b = %5d, result = %10d (Expected:
%10d)", a, b, result, expected);
            end
        end
    endtask

    initial begin
        pass_count = 0;
        total_tests = 0;
```

```
        $display("Starting unit coverage tests...");

        // Zero and One Cases
        test_case(15'd0,     15'd0,       30'd0);
        test_case(15'd1,     15'd0,       30'd0);
        test_case(15'd0,     15'd1,       30'd0);
        test_case(15'd1,     15'd1,       30'd1);

        // Max/Min Value Edge Cases
        test_case(15'd32767, 15'd1,       30'd32767);
        test_case(15'd1,     15'd32767,   30'd32767);
        test_case(15'd32767, 15'd32767,   30'd1073676289);
        test_case(15'd16384, 15'd2,       30'd32768);

        // Powers of Two
        test_case(15'b000000000000001, 15'd3, 30'd3);
        test_case(15'b000000000000010, 15'd3, 30'd6);
        test_case(15'b000000000000100, 15'd3, 30'd12);
        test_case(15'b000000000001000, 15'd3, 30'd24);
        test_case(15'b000000000010000, 15'd3, 30'd48);
        test_case(15'b000000000100000, 15'd3, 30'd96);
        test_case(15'b000000001000000, 15'd3, 30'd192);
        test_case(15'b000000010000000, 15'd3, 30'd384);
        test_case(15'b000000100000000, 15'd3, 30'd768);

        // Alternating bits
        test_case(15'b010101010101010, 15'd1, 30'd10922);
        test_case(15'b101010101010101, 15'd1, 30'd21845);

        // Random and edge
        test_case(15'd12345, 15'd6789, 30'd83810205);
        test_case(15'd1000,  15'd2000, 30'd2000000);
        test_case(15'd2,     15'd16383, 30'd32766);
        test_case(15'd32767, 15'd2, 30'd65534);
        test_case(15'd255,   15'd255, 30'd65025);
        test_case(15'd1023,  15'd1023, 30'd1046529);

        $display("Unit coverage testing complete: %0d out of %0d tests passed.",
pass_count, total_tests);
    end
endmodule
```

خروجی test bench به صورت کوتاه شده به این صورت اند:

```
//FIRST TEST BENCH:
# a =     0, b = 12345, result =          0 (Expected:          0) PASS
# a =     1, b = 12345, result =      12345 (Expected:      12345) PASS
# a =   123, b =   456, result =      56088 (Expected:      56088) PASS
# a = 32767, b = 32767, result = 1073676289 (Expected: 1073676289) PASS
# a = 32767, b =     1, result =      32767 (Expected:      32767) PASS
# a =  2345, b =  6789, result =   15920205 (Expected:   15920205) PASS
# a =    64, b =   128, result =       8192 (Expected:       8192) PASS
# a =     1, b =     1, result =          1 (Expected:          1) PASS
# a = 12345, b =   678, result =    8369910 (Expected:    8369910) PASS
# a = 30000, b = 20000, result =  600000000 (Expected:  600000000) PASS
```

```
//SECOND TEST BENCH:
# Starting random testbench...
# Test 0: a= 2531, b= 4651 => result=  11771681 (exp=  11771681) PASS
# Test 1: a=19798, b=30721 => result= 608214358 (exp= 608214358) PASS
# Test 2: a=17537, b= 6340 => result= 111184580 (exp= 111184580) PASS
# Test 3: a=22611, b= 7134 => result= 161306874 (exp= 161306874) PASS
        ...
        ...
# Test 97: a= 4715, b=  873 => result=   4116195 (exp=   4116195) PASS
# Test 98: a= 8917, b=25913 => result= 231066221 (exp= 231066221) PASS
# Test 99: a=31963, b= 3018 => result=  96464334 (exp=  96464334) PASS
# Random testing complete: 100 out of 100 tests passed
```

```
//THIRD TEST BENCH:
# Starting unit coverage tests...
# PASS: a =     0, b =     0, result =          0
# PASS: a =     1, b =     0, result =          0
# PASS: a =     0, b =     1, result =          0
# PASS: a =     1, b =     1, result =          1
# PASS: a = 32767, b =     1, result =      32767
# PASS: a =     1, b = 32767, result =      32767
# PASS: a = 32767, b = 32767, result = 1073676289
# PASS: a = 16384, b =     2, result =      32768
        …       …
# PASS: a = 21845, b =     1, result =      21845
# PASS: a = 12345, b =  6789, result =   83810205
# PASS: a =  1000, b =  2000, result =    2000000
# PASS: a =     2, b = 16383, result =      32766
# PASS: a = 32767, b =     2, result =      65534
# PASS: a =   255, b =   255, result =      65025
# PASS: a =  1023, b =  1023, result =    1046529
# Unit coverage testing complete: 25 out of 25 tests passed.
```