به نام خداوند بخشنده مهربان

طراحی سیستم‌های دیجیتال — پاییز ۱۴۰۰

تمرین ۳، سوال ۲ — آرین احدی نیا، ۹۸۱۰۳۸۷۸

توجه بفرمایید که عددی باینری که طولش توانی از دو باشد را می توان به شکل زیر نوشت. (طول بزرگ تر از ۳۲)

$$\begin{aligned} a_{32n-1}a_{32n-2}a_{32n-3}\dots a_0 &= a_{32n-1}a_{32n-2}a_{32n-3}\dots a_{32(n-1)} \times 2^{32(n-1)} \\ &+ a_{32(n-1)-1}a_{32(n-1)-2}a_{32(n-1)-3}\dots a_{32(n-2)} \times 2^{32(n-2)} + \dots \\ &+ a_{31}a_{30}a_{29}\dots a_0 \times 2^0 \end{aligned}$$

بنابرین میتوانیم عدد را به صورت بلوک‌های ۳۲ بیتی بنویسیم.

از طرفی توجه بفرمایید که $x \times 2^k$ برابر $x \ll k$ خواهد بود.

حال اگر عدد b را نیز به همین صورت بنویسیم و دو عبارت $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times 2^i$ را در $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+}\ b_{30+j}b_{29+j}\dots b_j \times 2^{i+j}$ ضرب کنیم، حاصل برابر $b_{31+j}b_{30+j}b_{29+j}\dots b_j \times 2^j$ خواهد شد. بنابرین اگر همه عبارت‌های دو عدد رو نظیر به نظیر در هم ضرب کنیم، خواهیم داشت

$$\begin{aligned} a \times b &= \sum_{i=0}^{n}\sum_{j=0}^{n} a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j \times 2^{i+j} \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n} a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j \ll (i+j) \end{aligned}$$

توجه بفرمایید که $a_{31+i}a_{30+i}a_{29+i}\dots a_i \times b_{31+j}b_{30+j}b_{29+j}\dots b_j$ را می‌توانیم با ضرب کننده ۳۲ بیتی که در اختیار داریم محاسبه کنیم و با استفاده از شیفت دادن، حاصل ضرب‌های پاره‌ای را به صورت تجمعی با جواب آخر جمع کنیم. و به این صورت حاصل را به دست بیاوریم.

قطعه کد زیر دقیقا همین روند را انجام می‌دهد.

```
module multiplier (clock, start, in1, in2, ready, out);
    parameter N = 128;

    input clock, start;
    input [N-1:0] in1, in2;

    output reg ready;
    output reg [2*N-1:0] out;

    reg [15:0] i, j;

    reg [31:0] op1, op2;
    wire [63:0] res;

    DSP high_speed_mult (res, op1, op2);

    always @(posedge start) begin
        ready = 1'b0;
        out = 1'b0;
        i = 16'b0;
        j = 16'b0;
    end
```

```
    always @(posedge clock) begin
        if (start == 1'b1 && ready == 1'b0) begin
            op1 = in1 >> i;
            op2 = in2 >> j;
        end
    end

    always @(negedge clock) begin
        if (start == 1'b1 && ready == 1'b0 && res !== 64'bx) begin
            out = out + (res << (i + j));
            j = j + 32;
            if (i == N) begin
                ready = 1'b1;
            end else if (j == N) begin
                i = i + 32;
                j = 0;
            end
        end
    end
endmodule
```

در ماژول فوق، یک ضرب کننده ۳۲ بیتی داریم. با شروع عملیات و ۱ شدن سیگنال start مقدار صفر به خروجی‌ها و برخی مقادیر کمکی نسبت داده می‌شود.

در این مدار ۳۲ بیت ۳۲ بیت از دو ورودی جدا می کنیم و در لبه بالارونده در op1 و op2 قرار می‌دهیم. سپس در لبه پایین رونده حاصل را دریافت کرده و با اعمال شیفت مناسب، با خروجی جمع میکنیم. در هر مرحله نیز i و j را به عنوان متغیرهایی پیماینده پایش می‌کنیم تا در زمان اتمام، سیگنال‌های مورد نظر را فعال کنیم.

برای تست، دو عدد را به صورت رندوم تولید می‌کنیم. توجه کنید که تابع $random یک عدد تصادفی ۳۲ بیتی خروجی می‌دهد. بنابرین برای اینکه عدد تصادفی ۱۲۸ بیتی داشته باشیم، میتوانیم ۴ بار از این تابع استفاده کنیم و خروجی‌ها را به هم متصل کنیم. سپس حاصل ضرب این دو عدد را یکبار به کمک ضرب داخلی خود وریلاگ و یکبار به کمک ماژولی که نوشتیم محاسبه می‌کنیم. در صورتی که دو پاسخ یکی باشد، ماژول به ازای این دو عدد صحیح کار میکند. با تغییر seed میتوانیم اعداد مختلفی تولید کنیم و به ازای چندین ترکیب مختلف ورودی برنامه خود را تست کنیم.

```
module test;
    wire clock;

    reg start;
    reg [127:0] in1, in2;

    wire ready;
    wire [255:0] out;

    integer i;

    reg [255:0] expected;

    clock_generator clk_ins (clock);
    multiplier mult_ins (clock, start, in1, in2, ready, out);

    initial begin
        in1 = 128'b0;
        in2 = 128'b0;
        for (i = 0; i < 4; i = i + 1) begin
            in1 = in1 + ($random << (32 * i));
            in2 = in2 + ($random << (32 * i));
        end
        $display("%H", in1);
```

```
        $display("%H", in2);
        start = 1'b1;
    end

    always @(posedge ready) begin
        expected = in1 * in2;
        $display("ex:\t%H%H", expected[255:128], expected[127:0]);
        $display("ac:\t%H%H", out[255:128], out[127:0]);
    end

    initial begin
        #200 $finish;
    end
endmodule
```