آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش شماره ۵: طراحی ضرب‌کننده

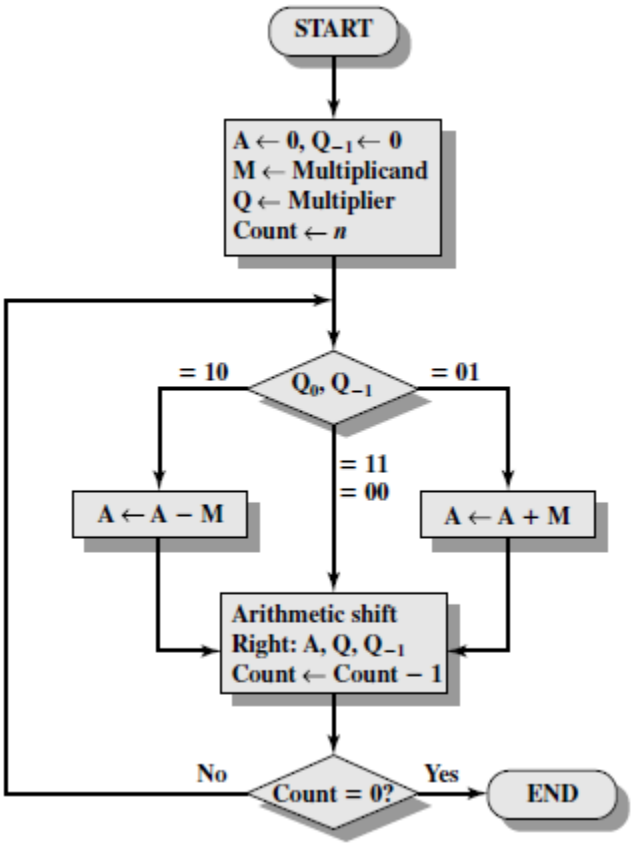
اعضای گروه:

روژین تقی‌زادگان ۴۰۱۱۰۵۷۷۵
رادین شاه‌دایی ۴۰۱۱۰۶۰۹۶
باربد شهرآبادی ۴۰۱۱۰۶۱۲۵

استاد: دکتر انصاری

هدف از این آزمایش طراحی یک واحد ضرب‌کننده است که برای انجام عمل ضرب از روش ضرب booth استفاده می‌کند.
برای انجام این آزمایش مسیر داده و واحد کنترل جداگانه طراحی می‌شود و سپس با اتصال آن‌ها به یکدیگر یک ضرب‌کننده ایجاد می‌کنیم.
در این طراحی، هنگامی که مضروب‌فیه (Multiplier) را به سمت راست شیفت می‌دهیم لازم است که واحدشیفت‌دهنده توان انجام شیفت بیش از یک بیت در یک پالس ساعت را داشته باشد تا بتواند نسبت به ضرب عادی (shift and add) تسریع داشته باشد.

الگوریتم booth:



نکات پیاده‌سازی الگوریتم:

I. به محض برخورد با اولین بیت ۱ کم‌ارزش در رشته ۱ها در مضروب‌فیه، مضروب از حاصل‌ضرب جزئی کم می‌شود.
II. به محض برخورد با اولین صفر (به شرطی که قبل از آن ۱ باشد) در رشته‌ای از ۰ها در مضروب‌فیه، مضروب با حاصل‌ضرب جزئی جمع می‌شود.
III. وقتی که دو بیت متوالی مضروب‌فیه مانند هم باشند، حاصل‌ضرب جزئی تغییر نمی‌کند.

ماژول booth:

```
module booth (
input [3:0] multiplicand,
input [3:0] multiplier,
input rst,
input clk,
output [7:0] result,
output done);

wire [2:0] A_shift_amount;
wire [2:0] B_shift_amount;
wire [3:0] B;

controlUnit CU (B, rst, clk, A_shift_amount, B_shift_amount, op,
done);
datapath DP (multiplicand, multiplier, rst, clk, A_shift_amount,
B_shift_amount, op, done, result, B);

endmodule
```

در این ماژول دو instance از control unit و datapath گرفته ایم و برای راه ارتباطی این دو نیز یک سری wire تعریف کرده‌ایم. (A_shift_amount, B_shift_amount, B) دقت شودکه خروجی ۸ بیتی در result ذخیره می‌شود و یک شدن سیگنال done، به معنای پایان محاسبه و valid شدن مقدار result می‌باشد.

حال به شرح control unit پردازیم.

```
module controlUnit (
input [3:0] B,
input rst,
input clk,
output [2:0] A_shift_amount, output [2:0] B_shift_amount, output
op,
output done);

reg [2:0] shifted;
reg first_clock;
wire [1:0] one_index;
wire [2:0] zero_index;

find_one first_one(B,one_index);
find_zero first_zero (B, zero_index);

assign op = B[0] & (~first_clock);
assign B_shift_amount = op ? zero_index : {1'b0, one_index};
assign A_shift_amount = shifted + B_shift_amount;
assign done = shifted + B_shift_amount >= 4;

always @(posedge clk) begin
    if (rst) begin
        shifted <= 0;
        first_clock <= 1;
    end
    else begin
        first_clock <= 0;
        shifted <= shifted + B_shift_amount;
    end
end

endmodule
```

در صورت ریست شدن، مقدار shifted برابر صفر شده و در کلاک اول پس از شروع محاسبات قرار میگیریم. در صورت posedge clk، مقدار shifted با B_shift_amount جمع شده و دیگر در اولین کلاک قرار نداریم .

استفاده از دو ماژول find_zero و find_one اندیس اولین بیت 0 یا 1 در B را پیدا میکنم که ما را برای انجام چندین شیفت در یک پالس ساعت یاری می‌کند. با توجه به تعریف، op کم ارزش ترین بیت B را ذخیره میکند (دقت شود که B خروجی datapath و شیفت راست داده‌شده اکستند شده multiplier می‌باشد که همان مضروب فیه در ماژول datapath می‌باشد) همچنین با توجه به نحوه assign شدن op، توجه به ~first_clock اگر در کلاک اول شیم op همان صفر در نظر گرفته شود. با توجه به تعریف زیر:

```
assign B_shift_amount = op ? zero_index : {1'b0, one_index};
```

اگرop برابر ۱ شد ما به دنبال اولین اولین بیت صفر پس از آن، و در غیر این صورت به دنبال اولین بیت یک پس از آن هستیم. مقدار shifted که برابر مقدار شیفت خوردن B کلاک قبلیست نیز با B_shift_amount جمع زده می‌شود تا A_shift_amount مشخص شود، یعنی مقداری که A برای جمع یا منها شدن حاصل نهایی باید به چپ شیفت بخورد . اگر مقدار B_shift_amount + shifted برابر 4 یا بیشتر شود، یعنی شیفت به راست دادن B به این مقدار، تمام بیتهای B بررسی شده است. پس محاسبات به اتمام رسیده است.

حال به شرح datapath می‌پردازیم.

```
module datapath (
input [3:0] multiplicand,
input [3:0] multiplier,
input rst,
input clk,
input [2:0] A_shift_amount,
input [2:0] B_shift_amount,
input op,
input done,
output reg [7:0] result,
output reg [3:0] B);

reg[7:0] A;

always @(posedge clk) begin
    if (rst) begin
        A <= {{4{multiplicand[3]}}, multiplicand};
        B <= {{4{multiplier[3]}}, multiplier};
        result <= 0;
    end

    else if (~done) begin
        B <= B >> B_shift_amount;
        if(op == 1)
            result <= result + (A << A_shift_amount);
        if(op == 0)
            result <= result - (A << A_shift_amount);
    end
end

endmodule
```

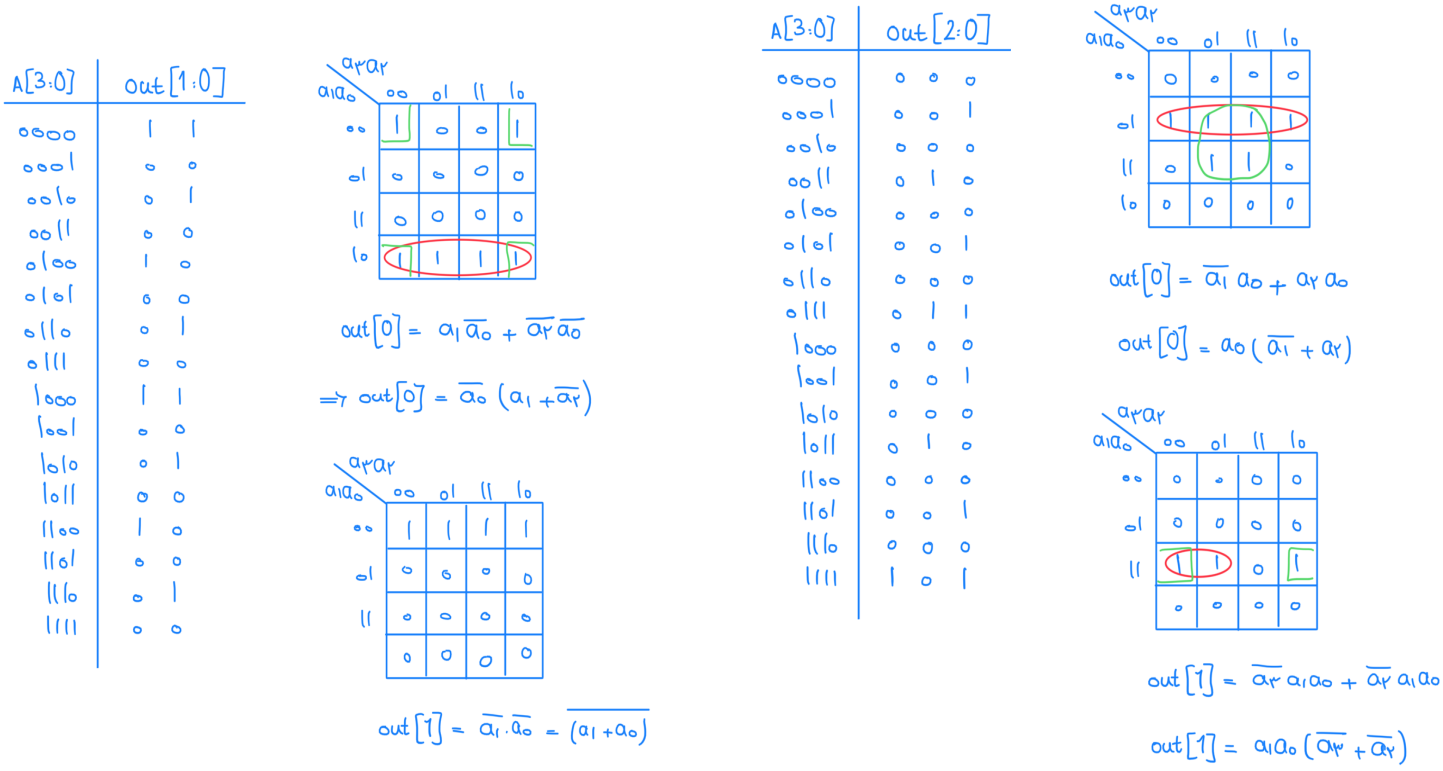
در صورت ریست شدن یعنی یک شدن سیگنال rst ، مقدار ورودی‌های 4 بیتی multiplicand و multiplier به ترتیب در دو رجیستر 8 بیتی A و B به اندازه ی ۴ بیت اکستند شده و ذخیره می‌شوند. در ادامه ی کار نیز در clk posedgeها عملیات شیفت برای multiplier و multiplicand انجام شود. یعنی B به میزان B_shift_amount به راست شیفت داده شود و همچنین ما در الگوریتم booth داریم که با توجه به op که ۱ باشد یا ۰، مضروب باید از حاصل کم شده یا با آن جمع شود و سپس حاصل به سمت راست شیفت داده شود. این عمل معادل حرکت ما یعنی شیفت دادن A به سمت چپ که اکستندشده multiplicand می‌باشد (آن هم به مقدار A_shift_amount) سپس جمع یا کسر آن از result می‌باشد.

در این ماژول ایندکس اولین بیت از ورودی که برابر صفر است را پیدا می‌کنیم.

```
module find_zero (input [3:0] A, output [2:0] out);

assign out[2] = A[3] & A[2] & A[1] & A[0];
assign out[1] = A[1] & A[0] & (~(A[3] & A[2]));
assign out[0] = A[0] & (~A[1] | A[2]);

endmodule
```

در این ماژول ایندکس اولین بیت از ورودی که برابر ۱ است را پیدا می‌کنیم.

```
module find_one (input [3:0] A, output [1:0] out);

assign out[1] = ~(A[1] | A[0]);
assign out[0] = ~A[0] & (A[1] | ~A[2]);

endmodule
```

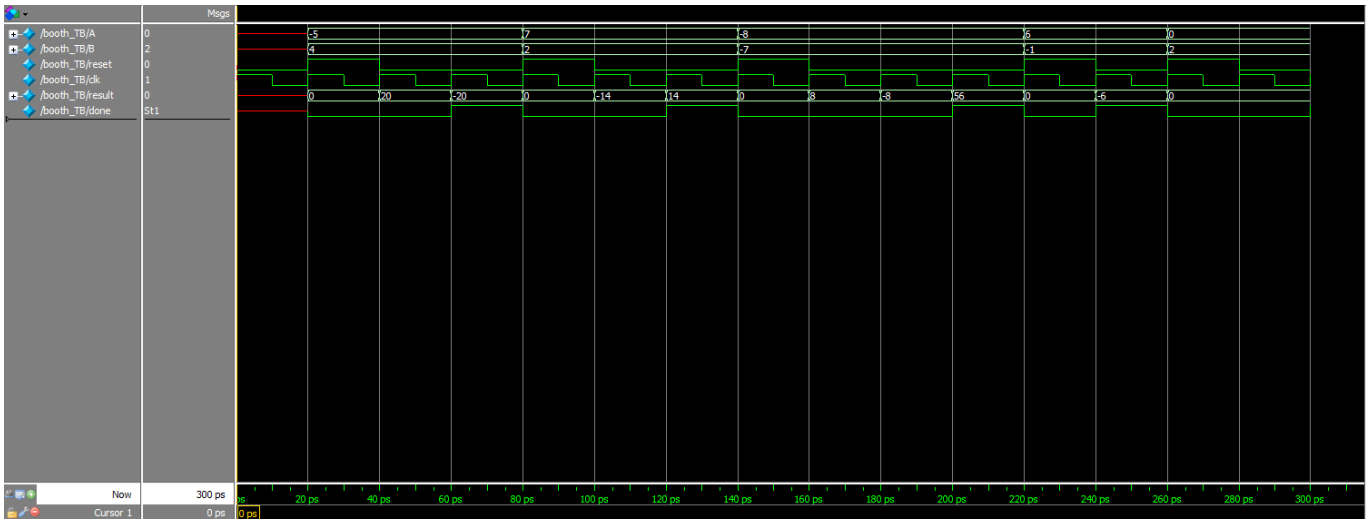


:Test Bench

```
module booth_TB ();
reg signed [3:0] A;
reg signed [3:0] B;
reg reset = 0, clk = 1;
wire signed [7:0] result; wire done;

booth MUL (A, B, reset, clk, result, done);

always #10 clk = ~clk;

initial begin
    #20;
    A=-5; B=4; reset = 1;
    #20 reset = 0;
    wait (done);
    $display("%d * %d = %d", A, B, result);

    #20;
    A=7; B=2; reset = 1;
    #20 reset = 0;
    wait (done);
    $display("%d * %d = %d", A, B, result);

    #20;
    A=-8; B=-7; reset = 1;
    #20 reset = 0;
    wait (done);
    $display("%d * %d = %d", A, B, result);

    #20;
    A=6; B=-1; reset = 1;
    #20 reset = 0;
    wait (done);
    $display("%d * %d = %d", A, B, result);
```

```
    #20;
    A=0; B=2; reset = 1;
    #20 reset = 0;
    wait (done);
    $display("%d * %d = %d", A, B, result);

    $stop;
    $finish;
end
endmodule
```

```
VSIM 10> run -all
#   -5 *    4 =   -20
#    7 *    2 =    14
#   -8 *   -7 =    56
#    6 *   -1 =    -6
#    0 *    2 =     0
```



تغییرات نسبت به پیش‌گزارش:
در پیش‌گزارش از سیگنال rst~ و صفر کردن آن برای ریست کردن ضرب‌کننده استفاده می‌کردیم اما در اینجا از سیگنال reset و یک شدن آن برای ریست کردن استفاده می‌کنیم.