بسمه تعالی

گزارش کار پنجم آزمایشگاه طراحی سیستم های دیجیتال

ضرب کننده

استاد

دکتر انصاری

نویسنده

سیدعماد امام جمعه – ۴۰۰۱۰۸۷۷۴

آرش ضیایی رازبان – ۴۰۰۱۰۵۱۰۹

محمدعرفان سلیما – ۴۰۰۱۰۵۰۱۴

دانشگاه صنعتی شریف

تابستان 1402

مقدمه

در این آزمایش هدف طراحی یک ضرب کننده 4 بیتی است. این کار با استفاده از DataPath و ControlUnit صورت گرفته و در Verilog پیاده سازی شده است. کد ها در VSCode زده شده و WaveForm ها توسط WaveTrace تولید شده اند.

پیاده سازی

این ضرب کننده شامل 3 ماژول Multiply, DataPath, ControlUnit است که در ادامه به توضیح آنها پرداخته ایم.

Multiply

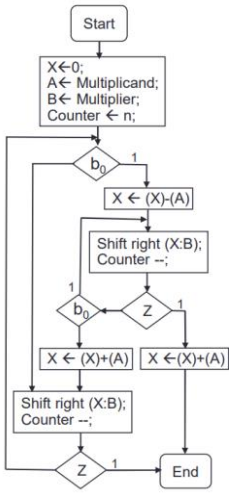
وظیفه این ماژول که بالاترین ماژول نیز هست، صرفا متصل کردن DP با CU است که مدار به درستی کار کند. کد وریلاگ این ماژول به شکل زیر است:

```
module Multiply (
    input wire[3:0] multiplier, multiplicand,
    input rstn, clk,
    output [7:0]result,
    output finish
);

wire[2:0] shift, counter;

ControlUnit CU(counter, result, rstn, clk, shift, op, finish);
DataPath DP(multiplier, multiplicand, rstn, clk, shift, op, finish, result,
counter);

endmodule
```

لازم به ذکر است که rstn به صورت active low است و برای شروع کار مدار باید دکمه متناظر آنرا فشار دهیم و رها کنیم.

Control Unit

این بخش از مدار بر اساس ASM Chart زیر طراحی شده است:



که البته در نسخه ما طبق دستور کار آزمایش از شیفت چندتایی استفاده میکنیم و به همین دلیل state های شیفت در نمودار بالا با state ها جمع و تفریق ترکیب شده و یه یک استیت تشکیل میدهند.

در این ماژول سیگنال های کنترلی مدار تولید میشوند که به شکل زیر است:

- Op: نشان دهنده عملیات در حال انجام است. اگر 1 باشد جمع و اگر 0 باشد تفریق.
- Finish: نشان میدهد که مدار حاصل نهایی را محاسبه کرده است.
- Shift: نشان میدهد که چند واحد شیفت باید در C:B رخ دهد. (در چارت معادل X:B است.)

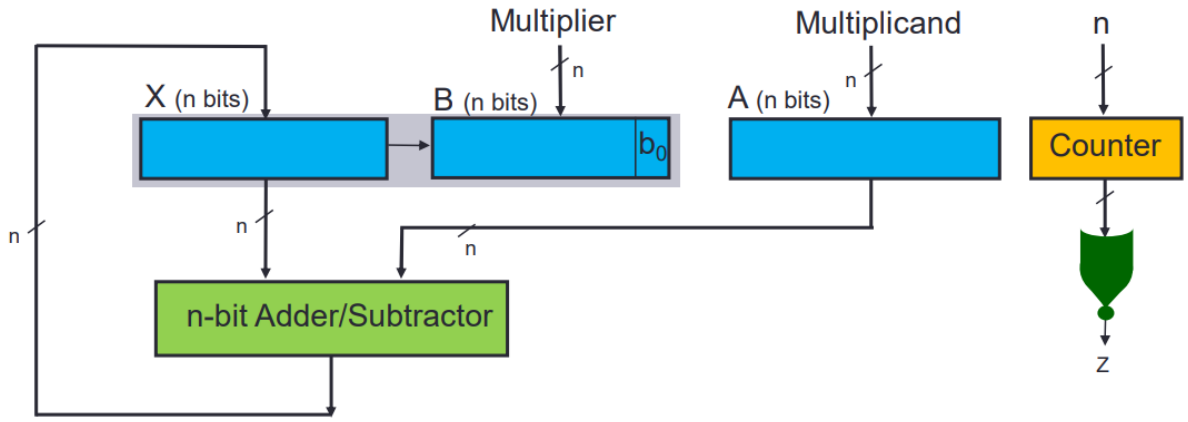
کد وریلاگ این ماژول به شکل زیر است:

```
module ControlUnit (
    input wire [2:0]counter,
    input [7:0] result,
    input rstn, clk,
    output reg [2:0] shift,
    output reg op,
    output reg finish
);

always @(posedge clk or negedge rstn) begin
    if(!rstn) begin
        finish = 0;
        shift = 0;
        op = 1;
    end
    else begin
        if(!finish) begin
            if(counter == 0) finish = 1;
            if(result[0] == op || counter == 0) shift = 0;
            else if(result[1] == op || counter == 1) shift = 1;
            else if(result[2] == op || counter == 2) shift = 2;
            else if(result[3] == op || counter == 3) shift = 3;
            else shift = 4;
            op = ~op;
        end
    end
end


endmodule
```

توجه کنید که در منطق این نکته لحاظ شده که هیچگاه shift از تعداد عملیات باقی مانده (counter) بیشتر نشود.

DataPath

در این ماژول رجیسترها و ماژول های نگهداری اطلاعات و سیم کشی بین آنها قرار گرفته. که بر مبنای شکل زیر درست شده است:



تنها تفاوت این است که ما به جای z برای راحتی کد وریلاگ، کل Counter را به CU ورودی میدهیم. و اینکه به جای X (چون X در وریلاگ معنای دیگری دارد) از نام گذاری C استفاده کرده ایم.

کد وریلاگ این ماژول به شکل زیر است:

```
module DataPath (
    input wire [3:0] multiplier, multiplicand,
    input rstn, clk,
    input [2:0] shift,
    input op,
    input finish,
    output wire [7:0] result,
    output reg [2:0] counter
);

reg signed [3:0] A, B, C;

assign result = {C, B};

always @(posedge clk or negedge rstn) begin
    if(!rstn) begin
        A = multiplicand;
        B = multiplier;
        C = 0;
        counter = 3'b100;
    end
    else begin
        #1
        if(!finish) begin
            if(shift == 1) {C, B} = {C[3], C, B[3:1]};
```

```
            else if(shift == 2) {C, B} = {C[3], C[3], C, B[3:2]};
            else if(shift == 3) {C, B} = {C[3], C[3], C[3], C, B[3]};
            else if(shift == 4) {C, B} = {C[3], C[3], C[3], C[3], C};
            counter = counter - shift;
            $display("pre %b ? %b", result, A);
            if(counter >= 0) begin
                if(op) C = C + A;
                else C = C - A;
            end
            $display("post %b", result);
        end
    end
end

endmodule
```

توجه کنید که در اینجای کار در ماژول از یک تاخیر 1 واحده استفاده شده که قابل سنتز نیست. دلیل این تاخیر این است که پس از آماده شدن سیگنال های کنترلی DP شروع به فعالیت کند تا با سیگنال اشتباه کار نکند. (در غیر این صورت مشکل خواهیم داشت.) اما این مشکل هنگام پیاده سازی بر روی FPGA رفع خواهد شد.

$display ها برای دیباگ به کار میروند و تاثیری در عملکرد مدار ندارند.

تست کردن

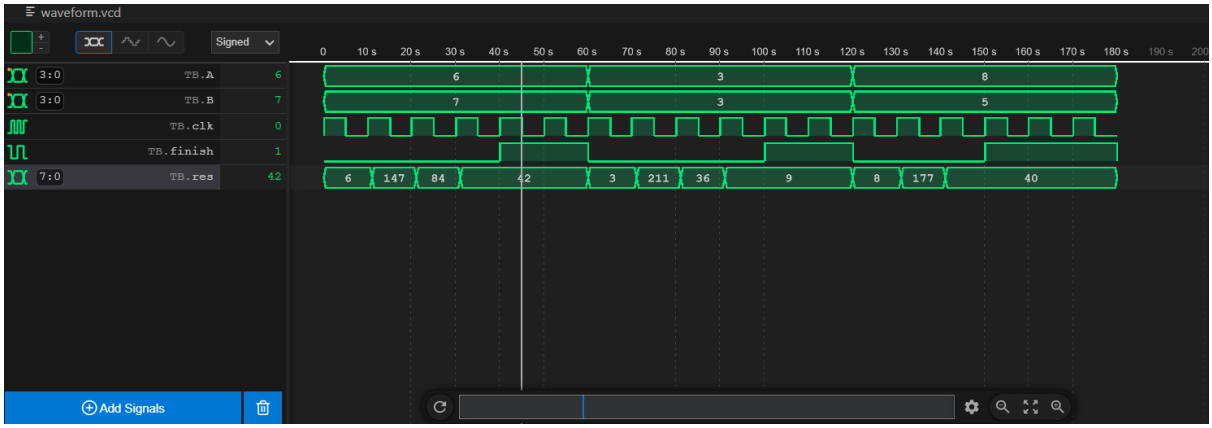
TestBench

در نهایت برای تست ضرب کننده، Test Bench زیر طراحی شده که حاصل 3 ضرب را بررسی میکند.

```
module TB;
    reg [3:0] A, B;
    reg rstn = 1, clk = 1;
    wire [7:0] res;
    wire finish;

    Multiply mul(A, B, rstn, clk, res, finish);

    always #5 clk = ~clk;

    initial begin
        $dumpfile("waveform.vcd");
        $dumpvars(0, TB);
        rstn = 0;
        A = 6; B = 7;
        #10
        rstn = 1;
        #50
        rstn = 0;
        A = 3; B = 3;
        #10
```

```
        rstn = 1;
        #50
        rstn = 0;
        A = 8; B = 5;
        #10
        rstn = 1;
        #50
        $finish();
    end

endmodule
```

پس از شبیه سازی کردن تست بنچ بالا WaveForm زیر تولید خواهد شد:



که همانطور که میبینید، همه خروجی ها به درستی تولید شده اند:

$6 * 7 = 42$ $3 * 3 = 9$ $8 * 5 = 40$

پس نتیجه میگیریم که ماژول به درستی کار میکند.

منابع:

Mano, Morris. *Computer system architecture*. Prentice-Hall of India, 2003.