آزمایش پنجم: طراحی ضرب کننده

ضرب کننده‌ی بوث

همانطور که در توضیحات آزمایش آمده است، این ضرب کننده دارای یک بخش مسیر داده و یک بخش کنترل است. همچنین ما یک فایل constants.v تعریف کردیم که در آن مقادیر ثابت مقداردهی شده‌اند. در نهایت نیز ماژولی بنام booth_mutiplier وجود دارد که بخش کنترل را به بخش مسیر داده متصل می‌کند. برای این آزمایش یک تست بنچ نیز نوشته شده است که در انتها آن را بررسی می‌کنیم. در بخش سیمولیشن ما پهنای multiplier و multiplicand را برابر ۵ در نظر گرفتیم و یکبار اعداد ۴ و ۱۰ و یکبار -۳ و ۵ را در آن تست کردیم. ولی برای بخش سنتز برای سهولت کار پهنای بیت را ۴ بیت در نظر گرفتیم تا اختصاص پین ها ساده‌تر باشد. در ادامه کدهای وریلاگ مربوط به هر بخش آمده است:

کد مربوط به booth_multiplier :

```
`include "constants.v"

module booth_multiplier(multiplier, mutiplicand, clk, rst, start, result);

    parameter size = `size;

    input [size-1:0] multiplier;
    input [size-1:0] mutiplicand;
    input clk;
    input rst;
    input start;

    output [2*size-1:0] result;

    //=======================================

    wire data_path_en;
    wire first_round;
    wire last_round;
    wire [2*size-1:0] temp_result;

    //=======================================

    assign result = (last_round) ? temp_result : 0;

    booth_controler controler(
                                .clk(clk),
```

```
                                .rst(rst),
                                .start(start),
                                .first_round(first_round),
                                .last_round(last_round),
                                .data_path_en(data_path_en)
                            );

    booth_multiplier_data_path data_path(
                                            .en(data_path_en),
                                            .clk(clk),
                                            .rst(rst),
                                            .multiplier(multiplier),
                                            .mutiplicand(mutiplicand),
                                            .first_round(first_round),
                                            .result(temp_result)
                                        );


endmodule
```

کد مربوط به بخش کنترلر:

```
`include "constants.v"

module booth_controler(clk, rst, start, first_round, last_round, data_path_en);

    parameter size = `size;

    input clk;
    input rst;
    input start;

    output first_round;
    output last_round;
    output data_path_en;

    //=========================================

    reg [size-1:0] counter =0;
    reg first_round;
    reg last_round;
    reg data_path_en;

    //=========================================

    always @(posedge clk, posedge rst)begin
        if (rst == 1'b1)begin
```

```
            counter = 0;
            last_round = 0;
            first_round = 0;
            data_path_en = 0;
        end
        else begin
            if (start == 1'b1) begin
                data_path_en = 1;
                if (counter == 0)begin
                    first_round = 1;
                end
                else begin
                    first_round = 0;
                end

                if (counter == size)begin
                    last_round = 1;
                end
                else begin
                    last_round = 0;
                end

                if(counter == size)begin
                    counter = 0;
                end
                else begin
                    counter = counter + 1;
                end
            end
            else begin
                data_path_en = 0;
            end
        end
    end

endmodule
```

کد مربوط به بخش مسیر داده:

```
`include "constants.v"

module booth_multiplier_data_path(en, multiplier, mutiplicand, clk, rst, first_round,
 result);

    parameter size = `size;

    input en;
```

```
    input [size-1:0] multiplier;
    input [size-1:0] mutiplicand;
    input clk;
    input rst;
    input first_round;

    output [2*size:1] result;

    //=======================================

    reg signed  [2*size:0] temp_result;
    reg [1:0]      op;

    //=======================================

    assign result       = temp_result[2*size:1];

    always @(posedge clk, posedge rst)begin
        if (rst == 1'b1) begin
            temp_result =   0;
        end
        else begin
            if (en) begin
                if(first_round == 1'b1)begin
                    temp_result[size:0] = (mutiplicand<<1);
                    temp_result[2*size:size+1] = 0;
                    op = temp_result[1:0];
                end
                case (op)
                    `NOOP:  begin
                                temp_result = (temp_result >>> 1);
                                op = temp_result[1:0];
                            end

                    `ADD:   begin
                                temp_result[2*size:size+1] = temp_result[2*size:size+
1] + multiplier;
                                temp_result = (temp_result >>> 1);
                                op = temp_result[1:0];
                            end

                    `SUB:   begin
                                temp_result[2*size:size+1] = temp_result[2*size:size+
1] - multiplier;
                                temp_result = (temp_result >>> 1);
                                op = temp_result[1:0];
                            end
                    default: begin
                                temp_result = (temp_result >>> 1);
```

```
                                op = temp_result[1:0];
                            end
                endcase
            end
        end
    end



endmodule
```

کد مربوط به تست بنچ:

```
`include "constants.v"

module test_booth();

    parameter size = `size;
    reg [size-1:0] multiplier;
    reg [size-1:0] mutiplicand;
    reg clk;
    reg rst;
    reg start;

    wire [2*size-1:0] res;

    always #5 clk = !clk;

    booth_multiplier booth_multiplier(
                                        .multiplier(multiplier),
                                        .clk(clk),
                                        .rst(rst),
                                        .mutiplicand(mutiplicand),
                                        .start(start),
                                        .result(res)
                                    );

    initial begin
        rst = 1;
        clk = 0;
        mutiplicand = -3;
        multiplier = 5;
        #15
        rst =0;
        start =1;
    end
endmodule
```

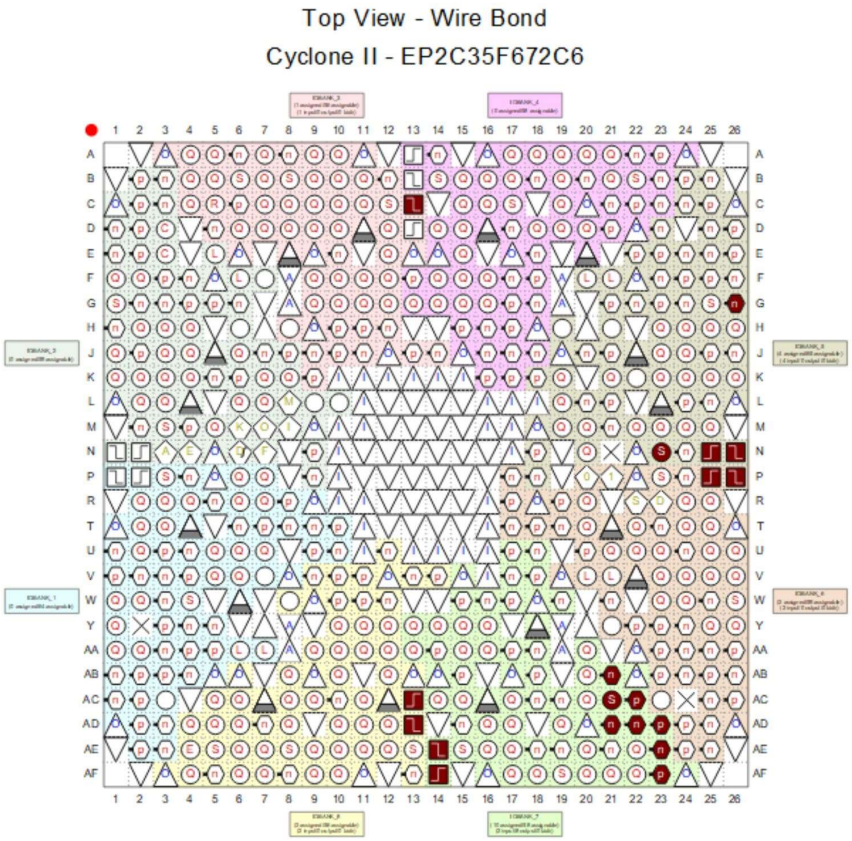
کد مربوط به constants:

```
`define size    4
`define NOOP    2'b00
`define ADD     2'b01
`define SUB     2'b10
```

ماژول booth_multiplier به اینصورت عمل می‌کند که که هر ۵ کلاک یکبار (به اندازه پنای بیت ورودی ها) نتیجه حاصل ضرب را نمایش می‌دهد. به این صورت که کنترلر دو سیگنال first_round و last_round تولید می‌کند که با assert شدن سیگنال first_round مسیر داده ورودی ها را لود می‌کند و کار خود را شروع می‌کند. سپس کنترلر یک متغیر counter دارد که می‌شمرد چند کلاک گذشته و در کلاک آخر که نتیجه ضرب آماده شده است (counter ==size) سیگنال last_round را assert می‌کند و نتیجه محاسبه شده در بخش مسیرداده را بر روی خروجی booth_multiplier قرار می‌دهد. تا زمانی هم که نتیجه ضرب آماده نشده است مقدار صفر را برای خروجی درنظر می‌گیریم.

اختصاص پین و کامپایل

برای پیاده‌سازی این مدار از دستگاه EP2C35F672C6 از خانواده Cyclone II استفاده خواهد شد. با توجه به راهنمای این دستگاه، پین‌ها را مطابق شکل 1 اختصاص می‌دهیم.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_P26	6	B6_N0	PIN_P26	3.3-V LV...default)		24mA (default)	
multiplier[3]	Input	PIN_N25	5	B5_N1	PIN_N25	3.3-V LV...default)		24mA (default)	
multiplier[2]	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)		24mA (default)	
multiplier[1]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)		24mA (default)	
multiplier[0]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)		24mA (default)	
mutiplicand[3]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)		24mA (default)	
mutiplicand[2]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)		24mA (default)	
mutiplicand[1]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)		24mA (default)	
mutiplicand[0]	Input	PIN_C13	3	B3_N0	PIN_C13	3.3-V LV...default)		24mA (default)	
result[7]	Output	PIN_AC21	7	B7_N0	PIN_AC21	3.3-V LV...default)		24mA (default)	
result[6]	Output	PIN_AD21	7	B7_N0	PIN_AD21	3.3-V LV...default)		24mA (default)	
result[5]	Output	PIN_AD23	7	B7_N0	PIN_AD23	3.3-V LV...default)		24mA (default)	
result[4]	Output	PIN_AD22	7	B7_N0	PIN_AD22	3.3-V LV...default)		24mA (default)	
result[3]	Output	PIN_AC22	7	B7_N0	PIN_AC22	3.3-V LV...default)		24mA (default)	
result[2]	Output	PIN_AB21	7	B7_N0	PIN_AB21	3.3-V LV...default)		24mA (default)	
result[1]	Output	PIN_AF23	7	B7_N0	PIN_AF23	3.3-V LV...default)		24mA (default)	
result[0]	Output	PIN_AE23	7	B7_N0	PIN_AE23	3.3-V LV...default)		24mA (default)	
rst	Input	PIN_G26	5	B5_N0	PIN_B13	3.3-V LV...default)		24mA (default)	
start	Input	PIN_N23	5	B5_N1	PIN_A13	3.3-V LV...default)		24mA (default)	

شکل 1 - نحوه‌ی اختصاص پین‌ها

به جز سیگنال rst و start که از Push-button استفاده می‌کند، سایر ورودی‌ها به کمک Switch داده شده‌اند. خروجی‌های result نیز به LED متصل شده است.

حال، پروژه را کامپایل می‌کنیم. نتیجه‌ی تحلیل و سنتز پس از کامپایل پروژه در شکل 2 نشان داده شده است.

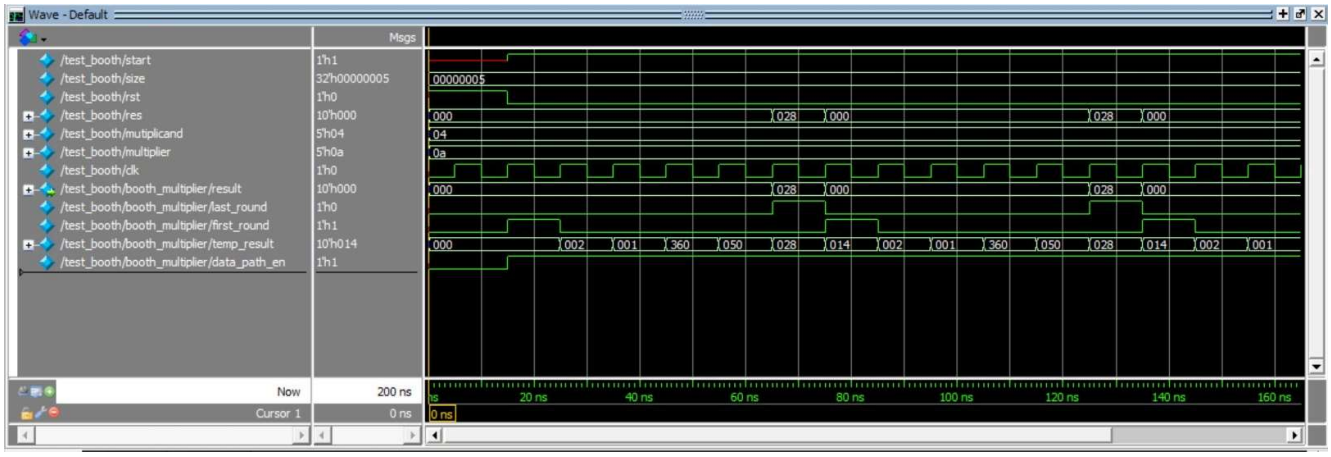
Analysis & Synthesis Summary

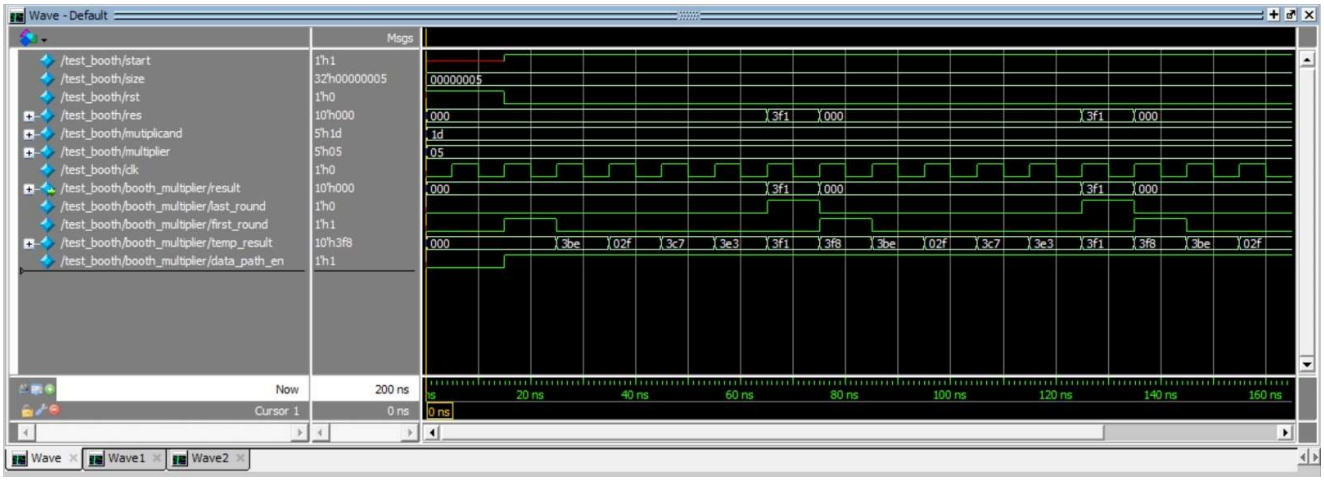
Analysis & Synthesis Status	Successful - Fri Jul 10 16:40:22 2020
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	booth_multiplier
Top-level Entity Name	booth_multiplier
Family	Cyclone II

Task	Time
Compile Design	00:00:23
Analysis & Synthesis	00:00:05
Edit Settings	
View Report	
Analysis & Elaboration	
Partition Merge	
Netlist Viewers	
Design Assistant (Post-Mapping)	
I/O Assignment Analysis	
Early Timing Estimate	
Fitter (Place & Route)	00:00:09
Assembler (Generate programming files)	00:00:05
TimeQuest Timing Analysis	00:00:04
EDA Netlist Writer	
Program Device (Open Programmer)	

شکل 2 - نتیجه‌ی سنتز ماژول ضرب کننده بوث

شکل 3، تست عملکرد ماژول پیاده‌سازی شده به کمک Waveform را نشان می‌دهد.





شکل 3 - شکل موج حاصل از تست عملکرد ماژول تست بنچ ضرب کننده بوث

همانگونه که در دو شکل بالا مشاهده می‌کنید هنگامی که last_round برابر یک است نتیجه ضرب در res ریخته شده است(هر ۵کلاک). نتیجه مرحله به مرحله ضرب کننده را نیز می توانید در سیگنال temp_result مشاهده کنید.