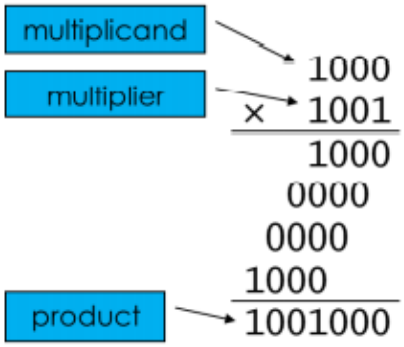
به نام خدا

آزمایش چهارم – ضرب کننده

امین ساوه دورودی – محمدرضا اسکینی – کامیاب عابدی

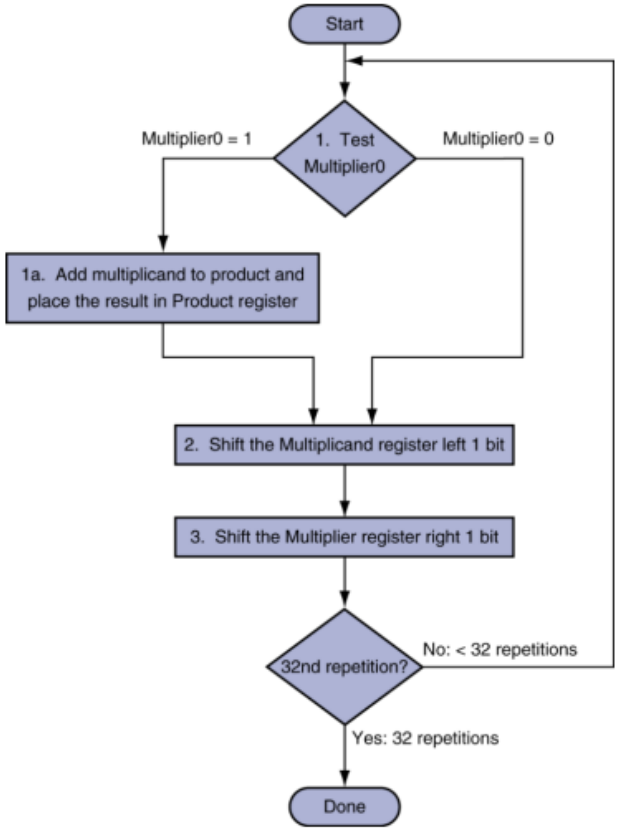
شرح اولیه

در این آزمایش, ضرب کننده با کمک شیفت و جمع به صورت Generic پیاده سازی میشود. در این ضرب کننده, همانند ضرب اعداد بر مبنای ده , از کم ارزش ترین رقم یکی از اعداد شرع کرده و رقم به رقم در کل رقم های عدد دیگر ضرب میکنیم و همین مراحل را برای دیگر ارقام ادامه میدهیم و به شکل خاصی آنها را جمع میکنیم. در اعداد باینری از آنجا که فقط با صفر و یک کار میکنیم, طی این مراحل ساده تر میباشد. از کم ارزش ترین بیت یکی از ارقام شروع میکنیم و در صورتی که این رقم یک باشد, کل بیت های عدد دوم به عنوان جواب این مرحله انتخاب میشود و در صورتی که رقم انتخاب شده صفر باشد, جواب مرحله صفر میباشد.تصویر(1) ضرب دو عدد باینری 4 بیتی را به صورت ضرب اعداد بر مبنای ده نشان میدهد.



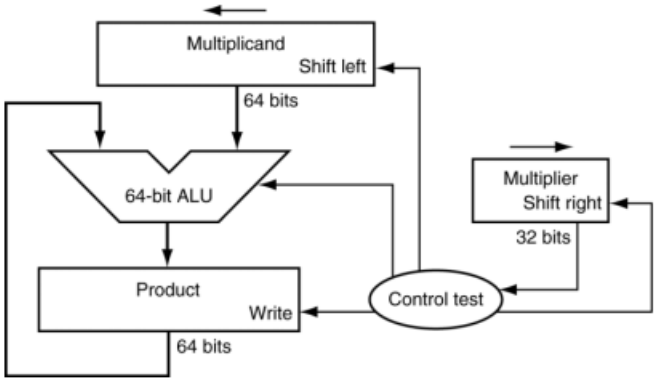
تصویر(1)

در ابتدای هر مرحله نیز جواب هر مرحله یک بیت به سمت چپ شیفت پیدا میکند. در ضرب اعداد باینری جواب حاصل عددی باینری با حداکثرتعداد دو برابر بیت های دو عدد اولیه میباشد. و از آنجا که تمامی ارقام یکی از اعداد انتخاب میشوند و در کل عدد دیگر ضرب میشوند, پس به تعداد بیت های اعداد, مراحل بالا تکرار میشود و در هر مرحله جواب با جواب مراحل قبل جمع میشود. در تصویر (2) الگوریتم این ضرب کننده به طور مثال برای 32 بیت, نمایش داده شده است.



تصویر(2) : الگوریتم ضرب 32 بیتی با کمک شیفت و جمع

در تصویر(3) نیز ساختار سخت افزاری این ضرب کننده برای دو عدد 32 بیتی نمایش داده شده است.



تصویر(3): سخت افزار ضرب دو عدد با شیفت و جمع

پیاده سازی

با استفاده از توضیحات بالا و به کمک تصویر (4) که الگوی این ضرب کننده برای محاسبه حاصل ضرب دو عدد 4 بیتی را نشان میدهد, این ضرب کننده را با جمع و شیفت پیاده سازی میکنیم.

Multiplier.txt

ماژول ضرب کننده پیاده سازی شده

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

تصویر(4): مثالی از ضرب دو عدد به همراه مقادیر ثبات ها در هر مرحله

این ماژول ضرب کننده, به صورت پارامتری تعداد بیت های دو عدد اولیه(N) را میگیرد و سپس دو عدد (N) بیتی را به عنوان ورودی میگیرد و در اخر یک خروجی (2N) بیتی به عنوان حاصل ضرب این دو عدد , تولید میکند.

از آنجا که ساختار این ضرب کننده به صورت مرحله ای و تکرار میباشد از بلاک always استفاده کردیم. در هر کلاک, با توجه به فعال بودن ضرب کننده (start) و همچنین با توجه به مقدار اولیه (init) مراحل مشخص شده در بلاک always تکرار میشود. در این مراحل از شیفت های منطقی چپ و راست و جمع استفاده شده است. همچنین متغیر counter تعداد مراحل انجام شده را در خود ذخیره میکند. حاصل ضرب آخر این عملیات بعد از تکرار N مرحله مشخص میشود و در خروجی قرار میگیرد. (تصویر 5)

```
module MultiPlier #(parameter N =32)(
input start,
input clk,
input [N-1:0] A,
input [N-1:0] B,
output reg [(2*N)-1:0] out
);
    reg [N-1:0] multiplier;
    reg [(2*N)-1:0] multiplicand,result;
    reg [5:0] counter;
    reg init;

always @(posedge clk)
    begin
        if(start)
        begin
            if(init)
            begin
                if(N>counter)
                begin
                    if(multiplier[0]==1)
                        result <= result + multiplicand;


                multiplicand <= multiplicand<<1;
                multiplier<=multiplier>>1;
                counter<=counter+1;
                end
                else
                begin
                out<= result;
                end
            end
        else
        begin
            init <=1;
            counter<=0;
            multiplier <=B;
            multiplicand <=A;
            result <=0;
        end
    end
    else
    begin
    init <=0;
    result <=0;
    out <=0;
    end
end
endmodule
```

تصویر(5)

تست

جهت اطمینان از حصول پیاده سازی این ضرب کننده , تست های انجام می دهیم. در تست بنچ مربوطه به این ضرب کنند دو نمونه 32 بیتی و 8 بیتی در نظر گرفته شده است تا صحت عملکرد این ضرب کننده با توجه به پارامترهای مختلف نیز مورد بررسی قرارگیرد.

Multiplier_Test.v

تست بنچ ماژول ضرب کننده

تصویر (6) تست بنچ ماژول ضرب کننده را نشان میدهید که با تست های مختلف را با تغییر متناوب کلاک و با استفاده از ریست کردن هر بار ضرب کننده, انجام میدهیم.

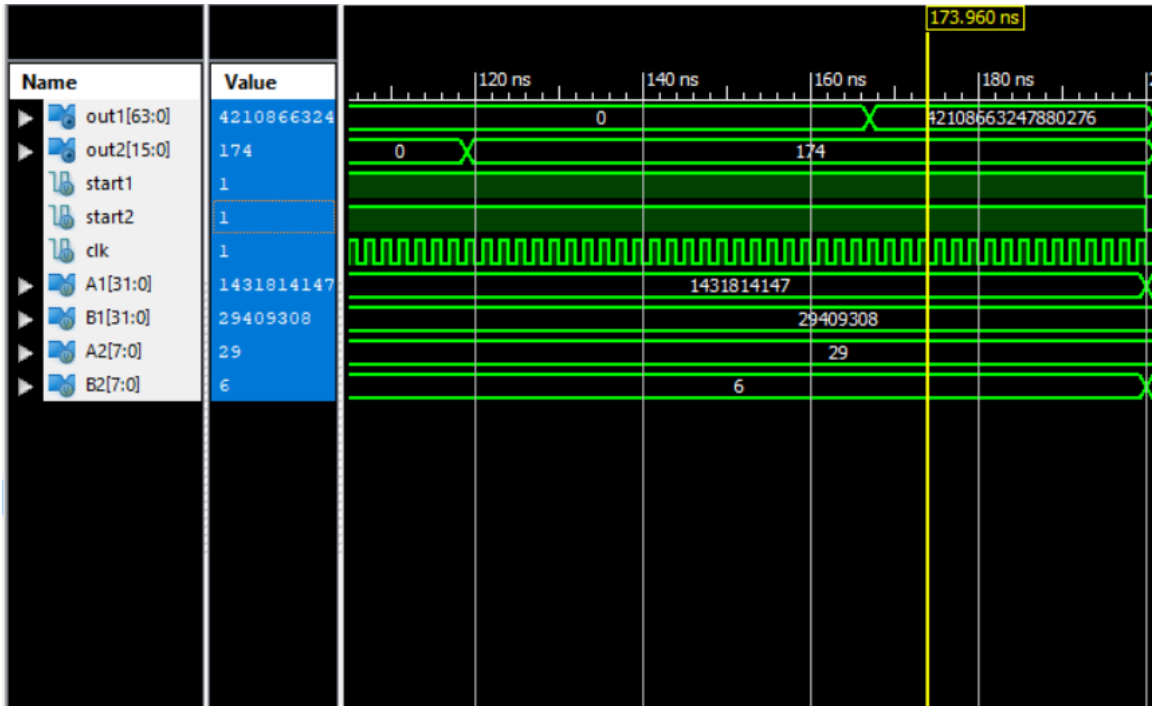
```
`timescale 1ns / 1ps
module MultiPlier_Test;
    // Inputs
    reg start1;
    reg start2;
    reg clk;
    reg [31:0] A1;
    reg [31:0] B1;
    reg [7:0] A2;
    reg [7:0] B2;
    wire [63:0] out1;
    wire [15:0] out2;

    MultiPlier #(32) m32bit(
        .start(start1),
        .clk(clk),
        .A(A1),
        .B(B1),
        .out(out1)
    );
    MultiPlier #(8) m8bit(
    .start(start2),
        .clk(clk),
        .A(A2),
        .B(B2),
        .out(out2)
);
    always #1 clk =~clk;
    initial begin
        start1 = 0;
        start2 =0;
        clk = 0;
        A1 = 32'b 01010101010101111100000000000011;
        B1 = 32'b 00000001110000001100000000011100;
        A2 = 8'b 00011101;
        B2 = 8'b 00000110;
        #100
        start1=1;
        start2=1;
        #100
        start1=0;
        start2=0;
        A1 = 32'b 00000000000000000000000001110011;
        B1 = 32'b 00000001110000001100000000011100;
        A2 = 8'b 00011101;
        B2 = 8'b 00000001;
      #100
        start1=1;
        start2=1;
    end
endmodule
```

تصویر(6)

نتایج تست ها

تست اول:

```
start1 = 0;
start2 =0;
clk = 0;
A1 = 32'b 01010101010101111100000000000011;
B1 = 32'b 00000001110000001100000000011100;
A2 = 8'b 00011101;
B2 = 8'b 00000110;
#100
```

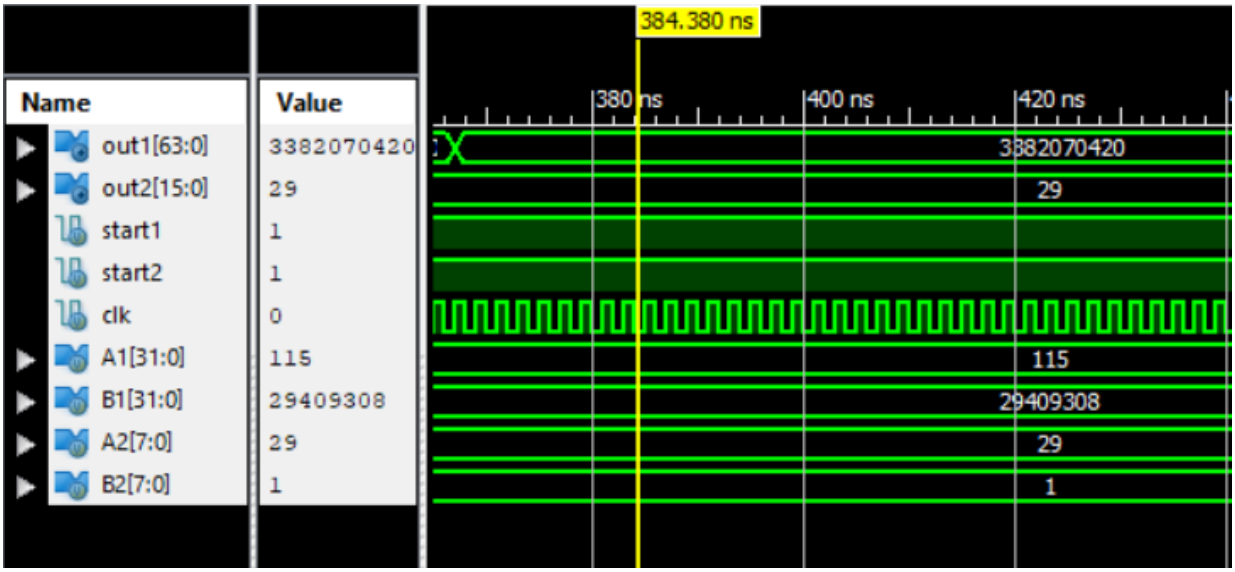


تصویر(7)

در تصویر(7) ابتدا ورودی های این تست نشان داده است و در پایین نتیجه شبیه سازی تست بر روی دو نمونه 8 و 32 بیتی نشان داده شده است که جواب ها با جواب های مورد انتظار ما , یکسان بوده است.

تست دوم:

```
A1 = 32'b 00000000000000000000000001110011;
B1 = 32'b 00000001110000001100000000011100;
A2 = 8'b 00011101;
B2 = 8'b 00000001;
```



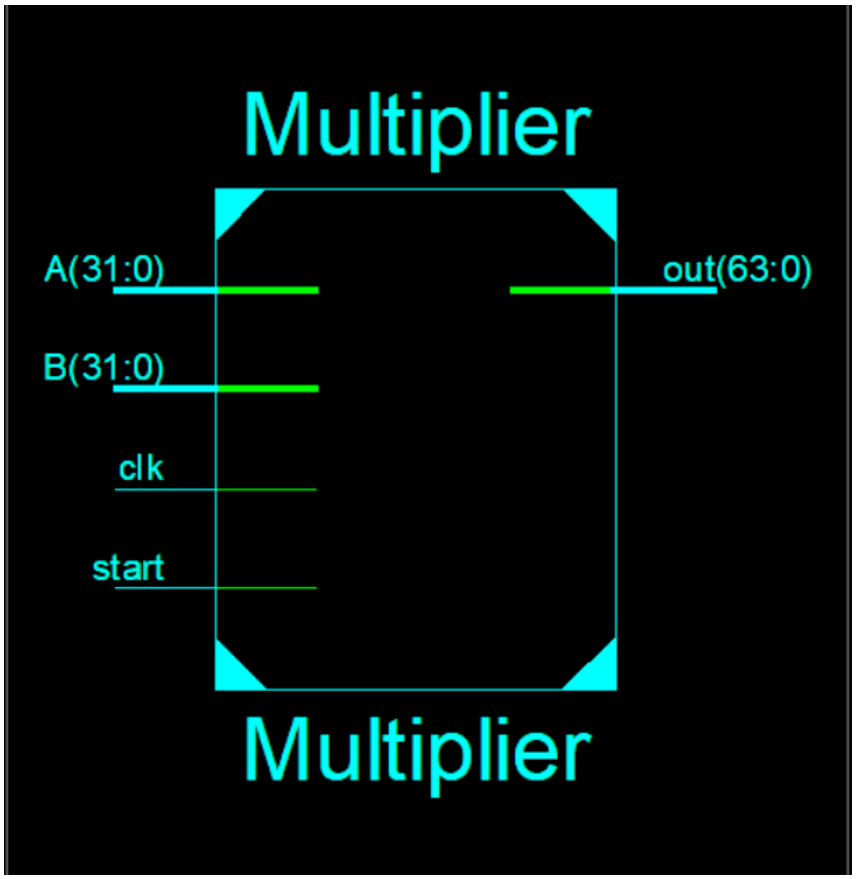
تصویر(8)

در تصویر(8) ابتدا ورودی تست در بالا نشان داده شده است. در پایین تصویر, نتیجه شبیه سازی دو نمونه 8 و 32 بیتی مطابق با انتظارات ما بوده است.

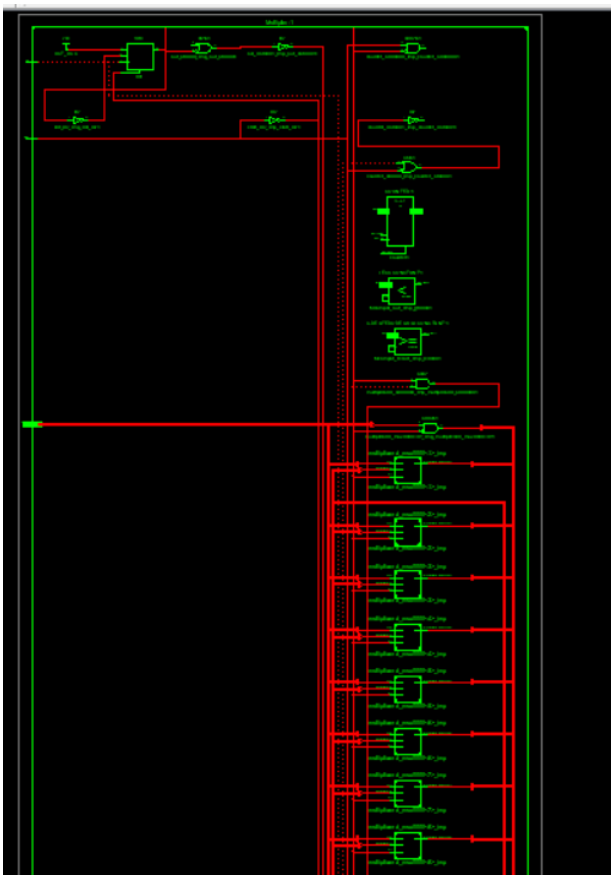
نتیجه گیری:

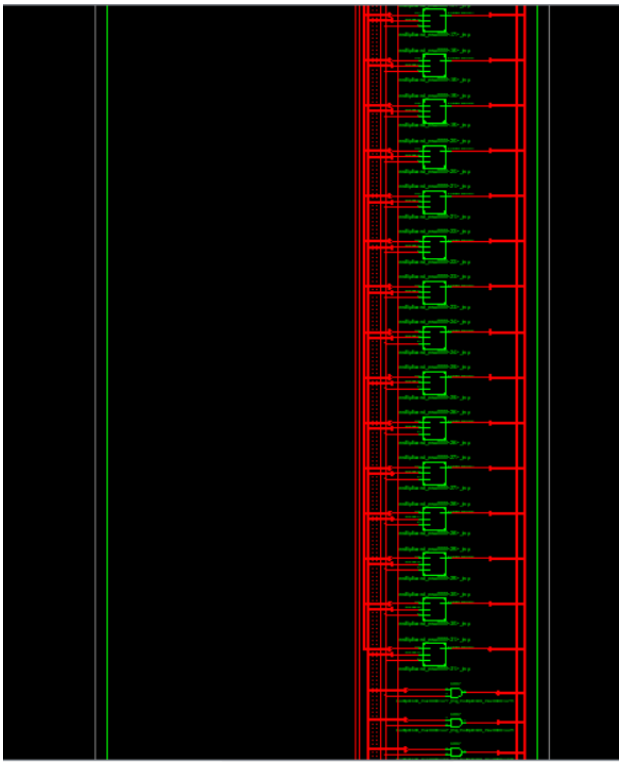
جواب تست های انجام شده بر روی ماژول پیاده سازی شده, مطابق با انتظارات ما بوده است. همچنین این تست صحت عملکرد ضرب کننده برای انواع اعداد با تعداد بیت های مختلف را به ما نشان می دهد.

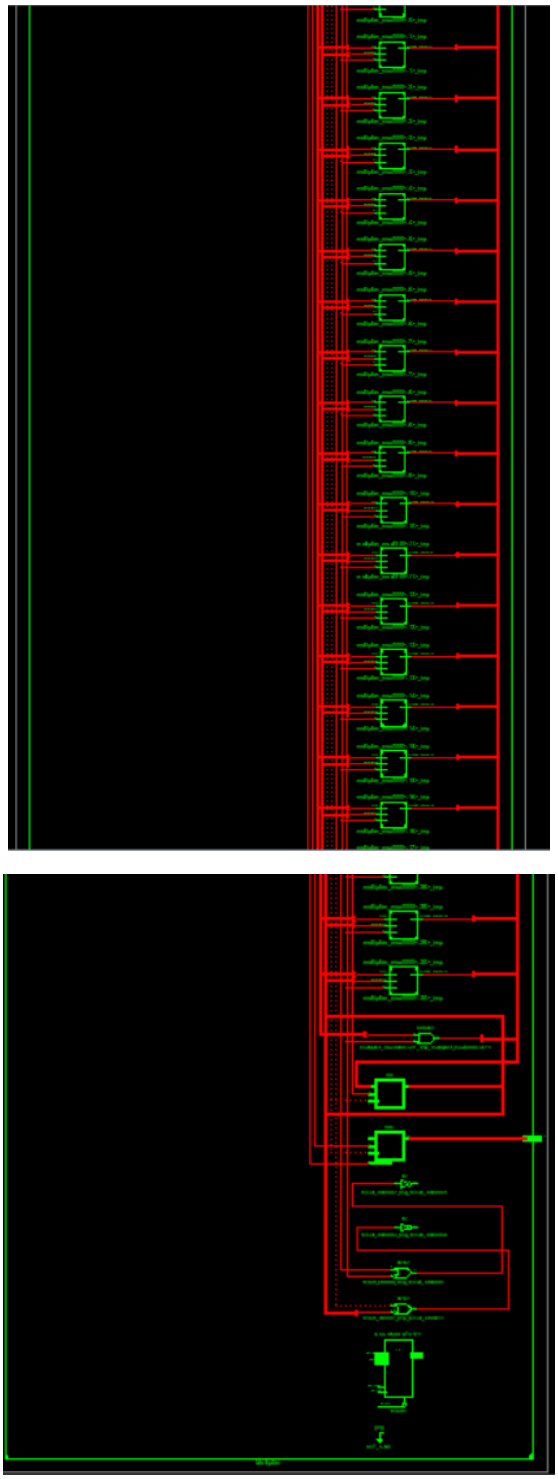
شماتیک مدار سنتز شده



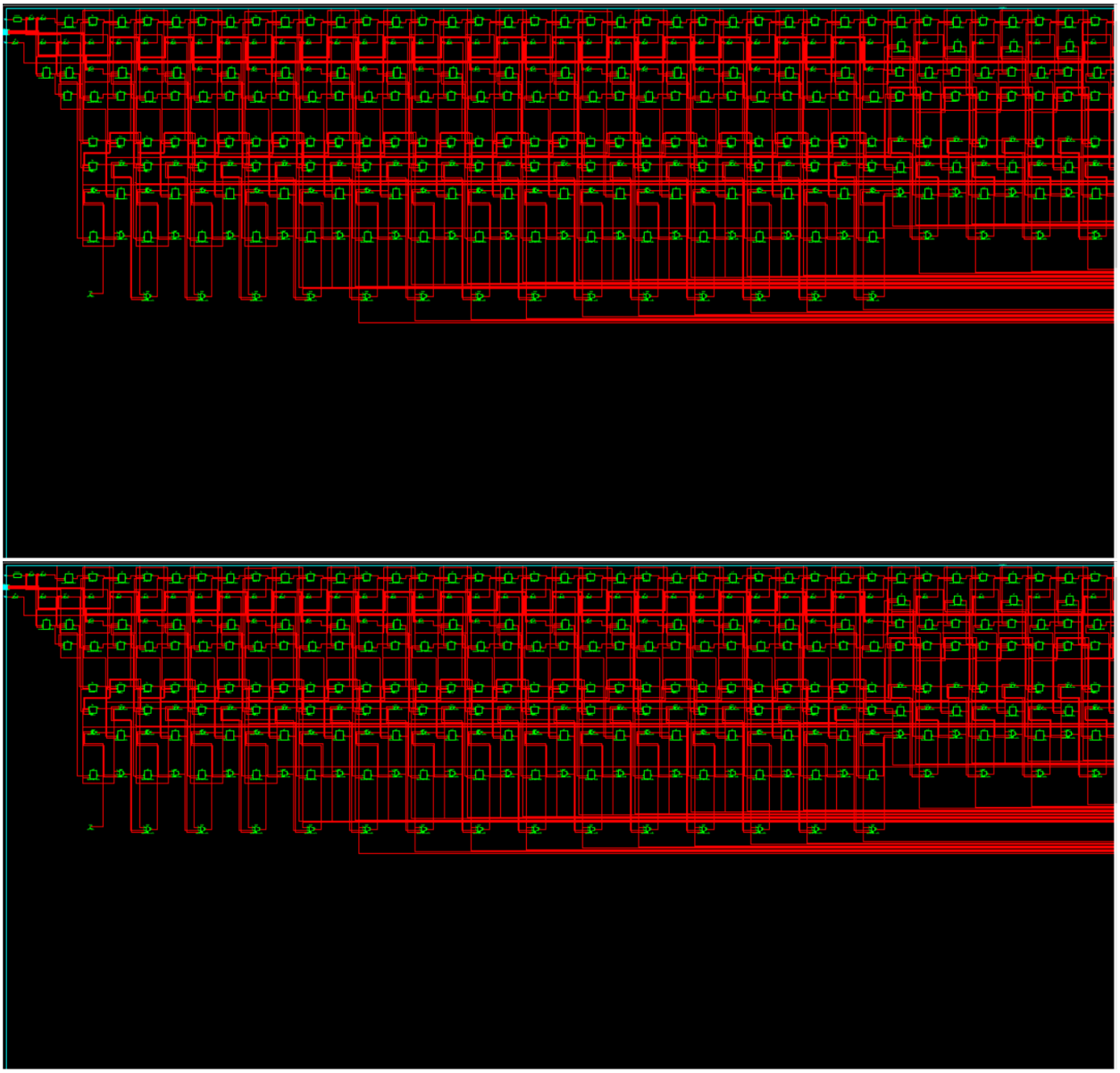
تصویر(9) : ورودی ها و خروجی ماژول ضرب کننده







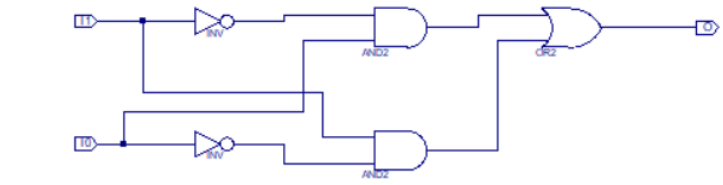
تصویر(10): RTL ماژول ضرب کننده



تصویر(11): Technology Schematic ماژول ضرب کننده

LUT

در Technology schematic ماژول, تصویر(11) , یکی از LUT ها را انتخاب کردیم و موارد آن در پایین نشان داده شده است.



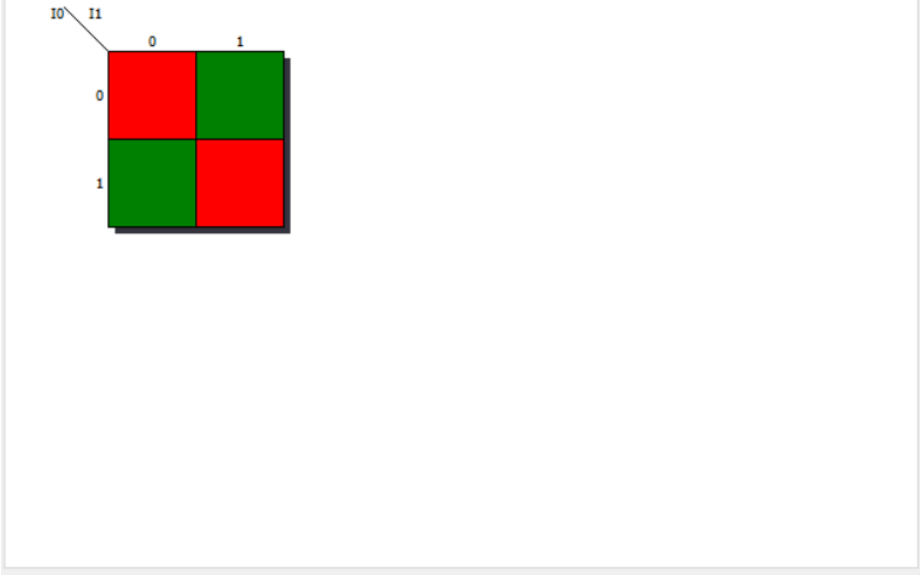
تصویر(12) : شماتیک

```
O = ((!I0 * I1) + (I0 * !I1));
```

تصویر(13): معادله

I1	I0	O
0	0	0
0	1	1
1	0	1
1	1	0

تصویر(14): جدول درستی



تصویر(15) : کارنومپ

خلاصه طراحی

Multiplier Project Status (04/05/2021 - 21:58:25)			
Project File:	Multiplier.xise	**Parser Errors:**	No Errors
Module Name:	Multiplier	**Implementation State:**	Mapped (Failed)
Target Device:	xc3s100e-5cp132	• **Errors:**	X 2 Errors (2 new)
Product Version:	ISE 14.7	• **Warnings:**	1 Warning (1 new)
Design Goal:	Balanced	• **Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	231	1,920	12%	
Number of 4 input LUTs	172	1,920	8%	
Number of occupied Slices	120	960	12%	
Number of Slices containing only related logic	120	120	100%	
Number of Slices containing unrelated logic	0	120	0%	
Total Number of 4 input LUTs	172	1,920	8%	
Number of bonded IOBs	130	83	156%	OVERMAPPED
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.81			

تصویر(16)

تصویر(16) گزارشی کلی و خلاصه از طراحی ما را نشان می دهد. همچنین میزان فضای اشغال شده نیز در تصویر بالا نمایان است.

نتیجه گیری نهایی

در این آزمایش, ماژول ضرب کننده با کمک شیفت و جمع, پیاده سازی شده است. طبق تست های انجام شده, عملکرد ماژول به ازای ورودی های N بیتی مطابق انتظار ما بود. همانطور که در تصویر(16) نیز نشان داده شده است, این ضرب کننده برای اعداد بالای 32 بیت کاربرد دارد و در صورت استفاده برای عددهایی با تعداد بیت کمتر, صرفا هزینه سخت افزاری زیادی مصرف میشود. میتوان از روش های سریعتر دیگر مثل Fast Multiplier برای محاسبه این اعداد استفاده کرد.