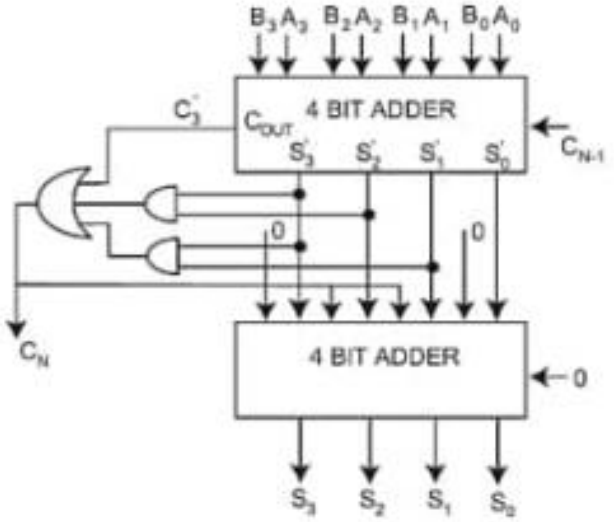
# به نام خدا

# آزمایش اول

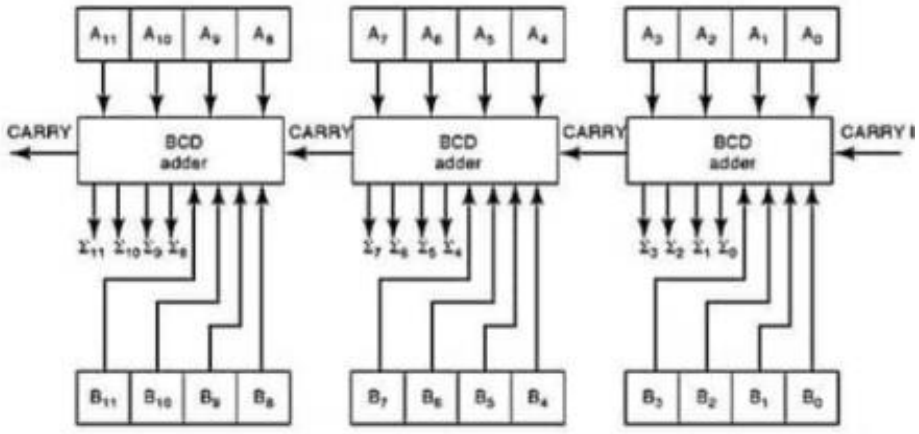
**امین ساوه دورودی – محمدرضا اسکینی – کامیاب عابدی**

## آزمایش الف

از در درس معماری و مدار منطقی میدانیم که سیستم BCD یا همان Binary Coded Decimal صرفا دارای اعداد 0 تا 9 میباشد. ( یا به عبارت دیگر در مبنای 2 به میتوان را دامنه اعداد را از 0000 تا 1001 در نظر گرفت)
از نکات بالا میتوان نتیجه گرفت که در این سیستم و در 4 بیت نمی توان اعداد بزرگتر از 9 ( 10 به بالا) را در 4 بیت نمایش داد. پس به بیت های بیشتر نیاز داریم. برای مثال برای عدد 10 به 8 بیت ( یک بایت ) نیاز داریم تا برای هر رقم آن، 4 بیت در نظر بگیریم.و هر کدام از اعداد را در سیستم BCD حساب و در کنار هم قرار دهیم.




شکل(۱): نمودار بلوکی جمع کننده BCD




شکل(۲): جمع‌کننده سه رقمی با بلوک BCD

همانطور که در دستور کار آورده شده است شکل های ۱ و ۲ در بالا مشخص شده اند که طبق آن به طراحی ماژول ها میپردازیم و در ادامه از آنها برای ساختن جمع کننده سه رقمی استفاده میکنیم.

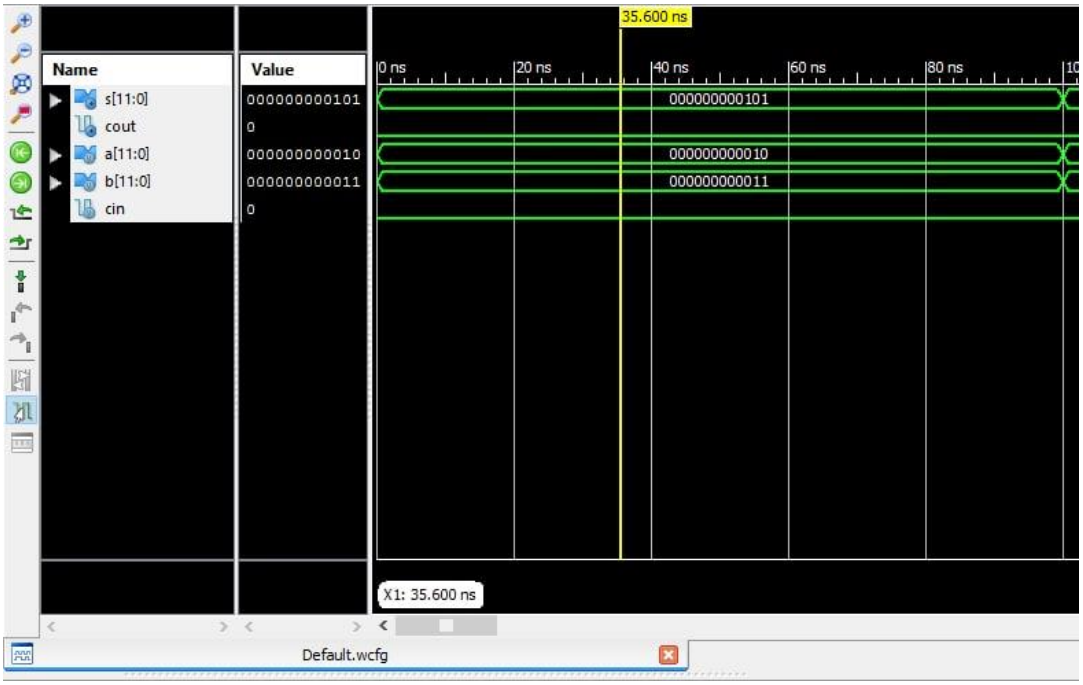
تست :

در ادامه شماتیک های داده شده در دستور کار را با کمک Verilog شبیه سازی و بررسی میکنیم. در زیر تعدادی از خروجی ها و شبیه سازی ها آورده شده است:

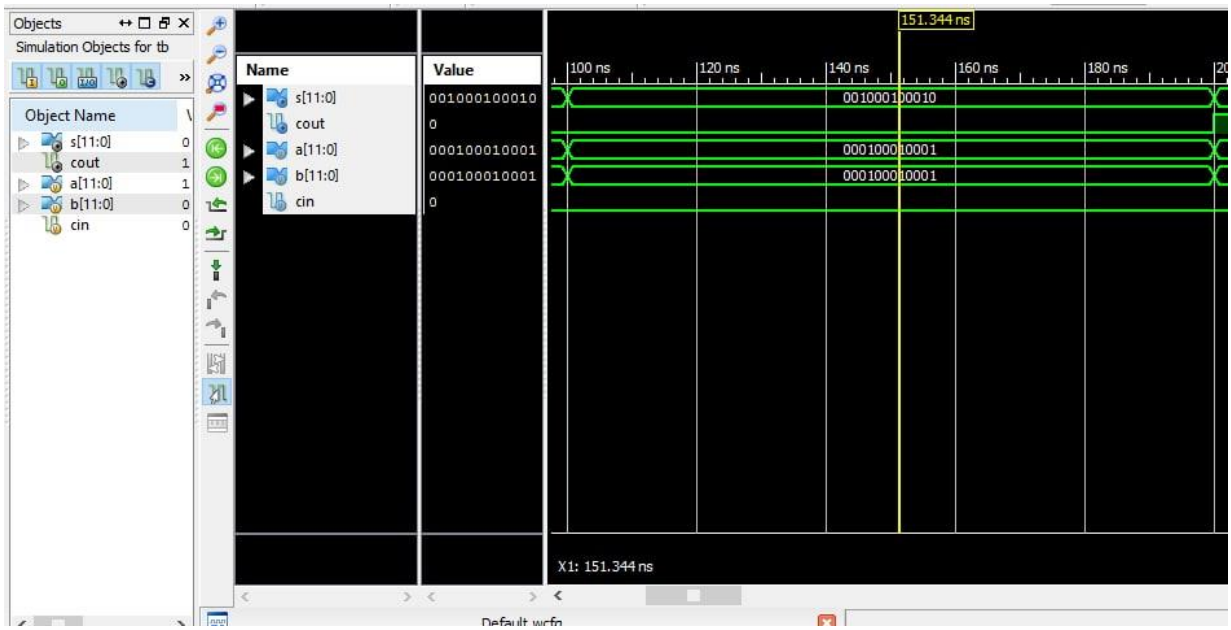
```
initial
begin
        cin = 0;
        a = 12'b0000_0000_0010; //2
        b = 12'b0000_0000_0011; //3
        #100;
        a = 12'b0001_0001_0001; //111
        b = 12'b0001_0001_0001; //111
        #100;
        a = 12'b1000_1000_1000; //888
        b = 12'b0011_0011_0011; //333   overflow
        #100;
end
```

داده های ورودی در تست بنچ

* در خروجی های زیر، Radix بر روی Binary ست شده است. چرا که کدینگ ما به صورت BCD است.




خروجی برای جمع دو عدد 2 و 3 = 5

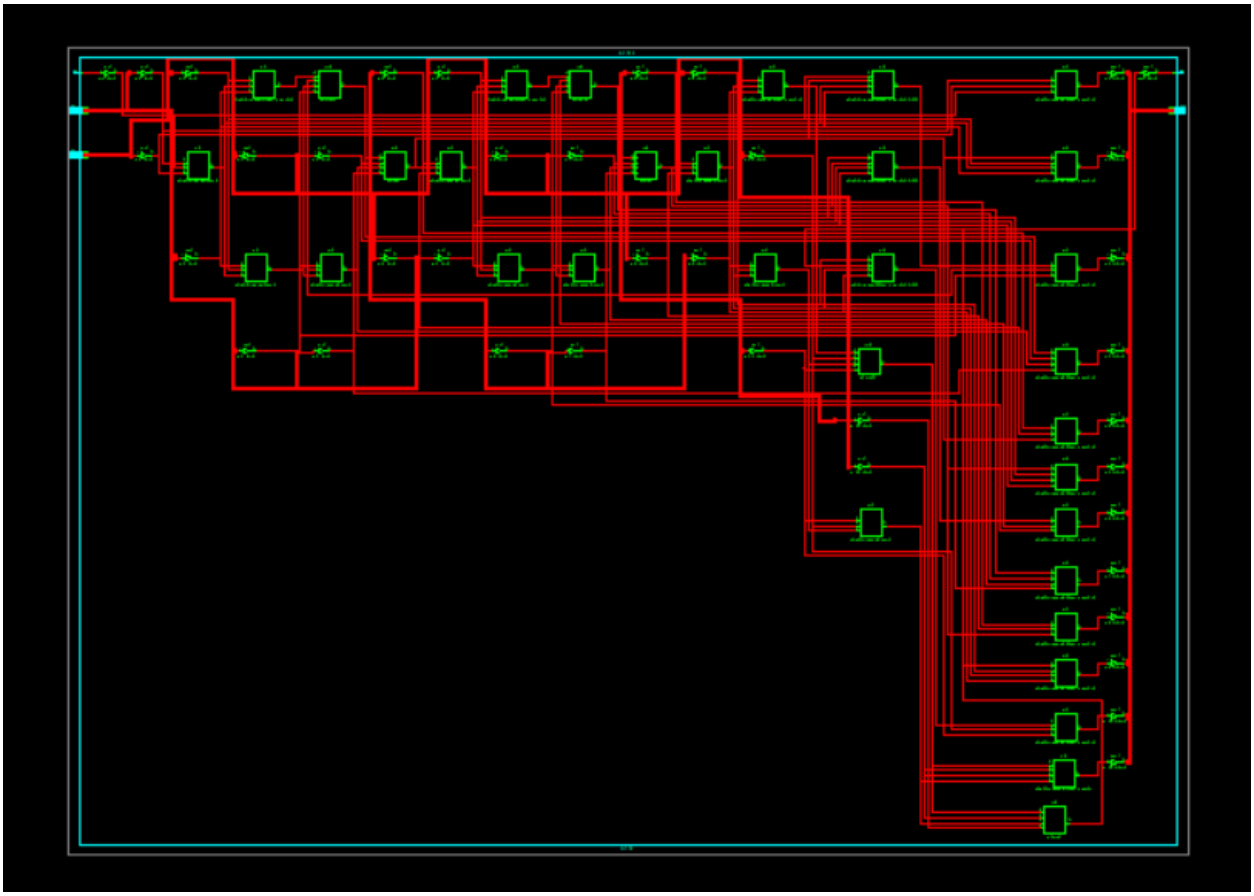



خروجی برای جمع دو عدد 111 و 111 = 222


خروجی برای جمع دو عدد 333 و 888=1221

در این خروجی همان طور که دیده میشود حاصل جمع چهار رقمی میباشد و overflow رخ داده است که همان طور که مشخص است Cout نیز 1 شده است.

همانطور که در دستور کار نیز گفته شد است، قرار دادن مدار های سنتز شده کاملا اختیاری است برای این تمرین اما ما آن ها را نیز خروجی گرفتیم .



(RTL)

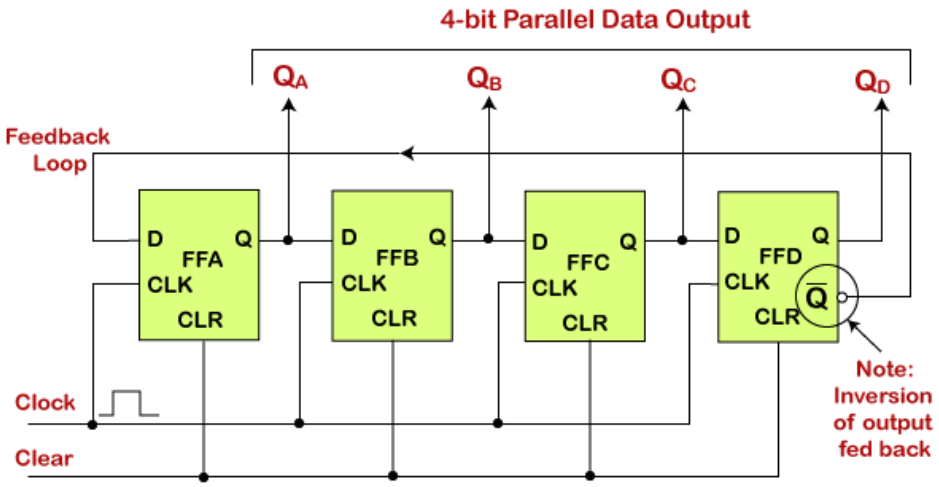
خلاصه طراحی :

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slices | 19 | 960 | 1% |
| Number of 4 input LUTs | 33 | 1920 | 1% |
| Number of bonded IOBs | 38 | 83 | 45% |

## آزمایش ب

# Johnson Counter

در این قسمت به پیاده سازی و شبیه سازی Johnson Counter به کمک زبان توصیف سخت افزار Verilog می پردازیم:
شیوه کاری این شمارنده به این صورت است که بیت آخر را (MSB) به صورت معکوس شده به بیت اول ( LSB) بر میگرداند.
به همین دلیل است که یک الگوی 8 تایی در طول زمان این شمارشگر ( با 4 بیت خروجی) ایجاد میکند که این 8 الگو در صورت دستور کار آورده شده است.




شماتیک

| CP | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 |

جدول درستی

```
module JohnsonCounter(

    input clk,
    input  reset,
    output reg [3:0]out
);

    always @(posedge clk)
    begin
        if (reset)
            out <= 4'b0000;
        else
        begin
            out[3]<=~out[0];
            out[2]<=out[3];
            out[1]<=out[2];
            out[0]<=out[1];
        end
    end
endmodule
```

کد استفاده شده برای شبیه سازی شمارنده

همانطور که در شکل بالا دیده میشود، ما از روش توصیف رفتاری استفاده کرده ایم. و همینطور یک سیگنال **reset** نیز برای شروع کار این دستگاه و راه اندازی مجدد آن در نظر گرفته شده است.

در و در ادامه در هر لبه بالارونده کلاک، 4 بیت را یک واحد به سمت چپ شیف میدهیم. نکته مهم در این قسمت این است که بیت آخر (که معمولا دور ریخته میشود) در اینجا نقیض و به بیت اول منتقل میشود..

مزایا:

برای پیاده سازی یک الگوی تکرار شونده در 2n کلاک تنها به n فلیپ فلاپ نیاز داریم. در مقایسه با Ring Counter که در پایین توضیح داده خواهد شد نیز حالت های بیشتری را میتواند نشان دهد چرا که  میتوانیم همزمان چند بیت با مقدار یک داشته باشیم.

معایب :

تمامی حالات ممکن از ۱۵ حالت را پوشش نمیدهد و تنها ۸ حالت را در بر میگیرد و تمامی حالاتی که چهاربیت میتوانند باشند را شامل نمیشود.

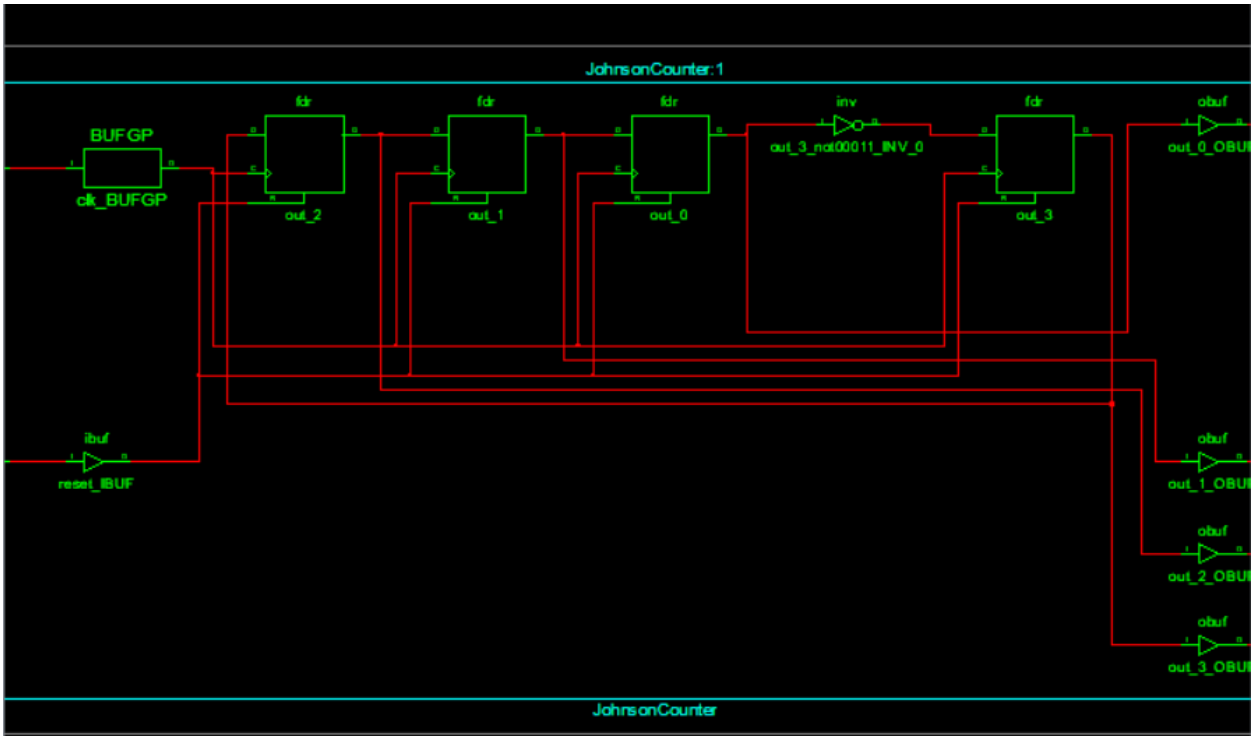
تست :

```
module tb;
    // Inputs
    reg clk;
    reg reset;
    // Outputs
    wire [3:0] out;
    JohnsonCounter john(clk,reset,out);
    always #5 clk = ~clk;

    initial
    begin
        clk = 0;
        #5 reset = 1;
        #100 reset = 0;
    end
endmodule
```

کد استفاده شده در تست بنچ

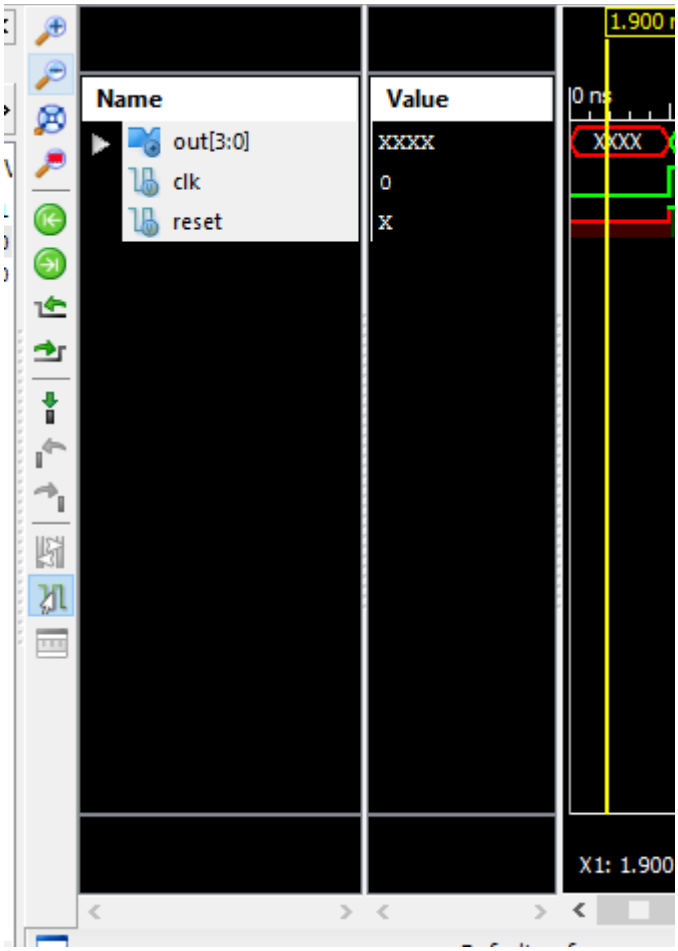
برای کلاک، یک سیگنال که هر 5 واحد زمانی عوض می شود در نظر گرفته ایم.

در ادامه به مدت یک لبه کلاک، مقدار reset را 1 کرده ایم تا دستگاه ما مقدار دهی اولیه شود.

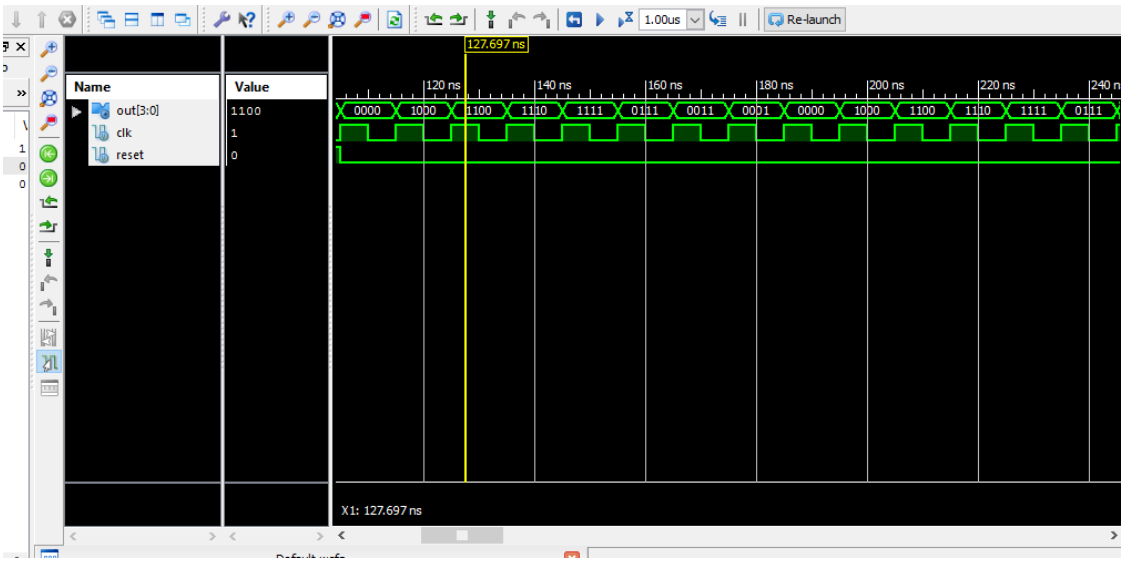

JohnsonCounter:1
BUFGP
clk_BUFGP
fdr
out_2
fdr
out_1
fdr
out_0
inv
out_3_not00011_INV_0
fdr
out_3
obuf
out_0_OBUF
ibuf
reset_IBUF
obuf
out_1_OBUF
obuf
out_2_OBUF
obuf
out_3_OBUF
JohnsonCounter

(RTL)

شبیه سازی:




و همانطور که معلوم است در 5 واحد زمانی اول، چون مقدار خروجی ست نشده است، مقدار X نمایش داده میشود.




و بعد از آن که شمارنده شروع به کار می کند داریم که الگو های ذکر شده در صورت سوال، همگی خودشان را نشان میدهند.

خلاصه طراحی :

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Flip Flops | 4 | 1,920 | 1% | |
| Number of occupied Slices | 2 | 960 | 1% | |
| Number of Slices containing only related logic | 2 | 2 | 100% | |
| Number of Slices containing unrelated logic | 0 | 2 | 0% | |
| Number of bonded IOBs | 6 | 83 | 7% | |
| Number of BUFGMUXs | 1 | 24 | 4% | |
| Average Fanout of Non-Clock Nets | 2.40 | | | |

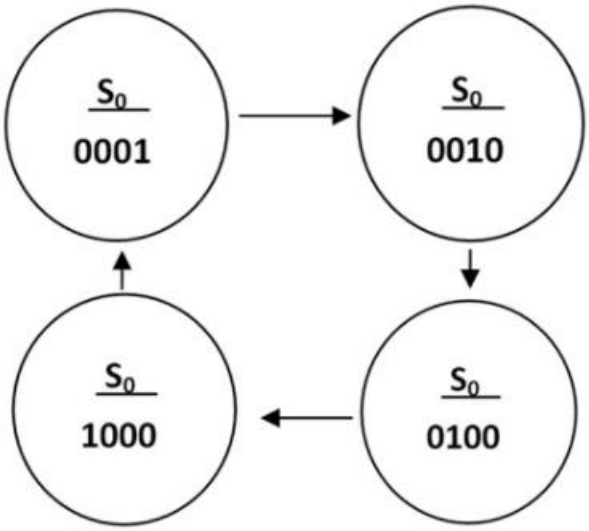
# Ring Counter

طبق صورت پروژه, مدار تایمینگ هشت حالته مداری است که هشت خروجی دارد ( یک پایانه هشت بیتی ) که در هر لحظه فقط یکی از بیت ها میتواند یک و بقیه بیت ها صفر باشند. پس نیاز به کلاکی داریم که در هر کلاک فقط و فقط یکی از سیگنال ها روشن و بقیه خاموش باشند. با استفاده از الگوی شمارنده جانسون Ring Counter را پیاده سازی میکنیم.

جدول درستی و دیاگرام وضعیت Ring Counter به عنوان مثال چهارحالته به شکل زیر میباشد:

| $Q_0$ | $Q_1$ | $Q_2$ | $Q_3$ |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

جدول درستی



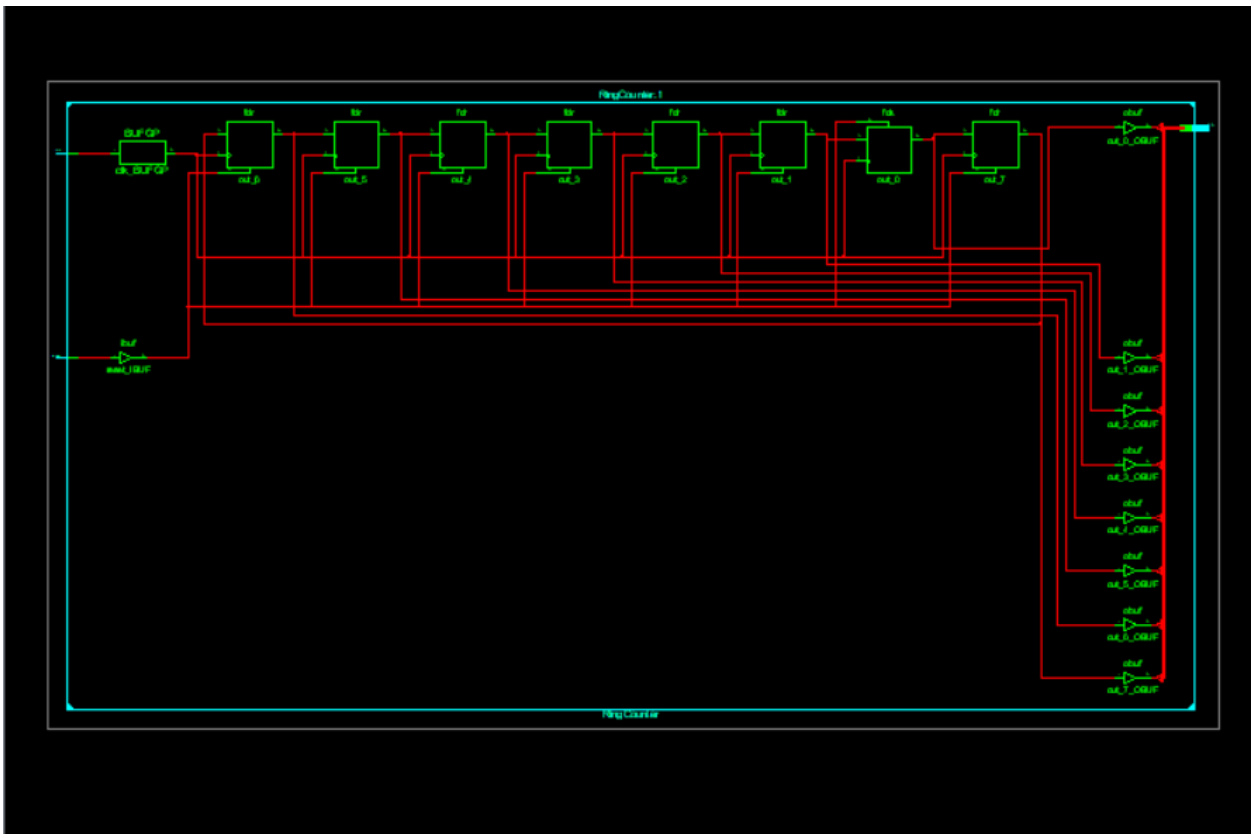

دیاگرام وضعیت

مزایا :

به تنهایی یک دیکودر میباشد چرا که در هر کلاک تنها یک بیت مقدار یک را دارد. میتواند با انواع فلیپ فلاپ ها ساخته شود.

معایب :

فقط ۸ حالت از ۲۵۵ حالت ممکن برای مقدار بیت ها را میتواند تولید کند .



(RTL)

پیاده سازی :

```
module ringcounter(input  clk,
input  reset,
output reg [7:0]out);

always @(posedge clk)
begin
    if (reset)
        out <= 8'b00000001;
    else
    begin
        out[7]<=out[0];
        out[6]<=out[7];
        out[5]<=out[6];
        out[4]<=out[5];
        out[3]<=out[4];
        out[2]<=out[3];
        out[1]<=out[2];
        out[0]<=out[1];
    end
end

endmodule
```

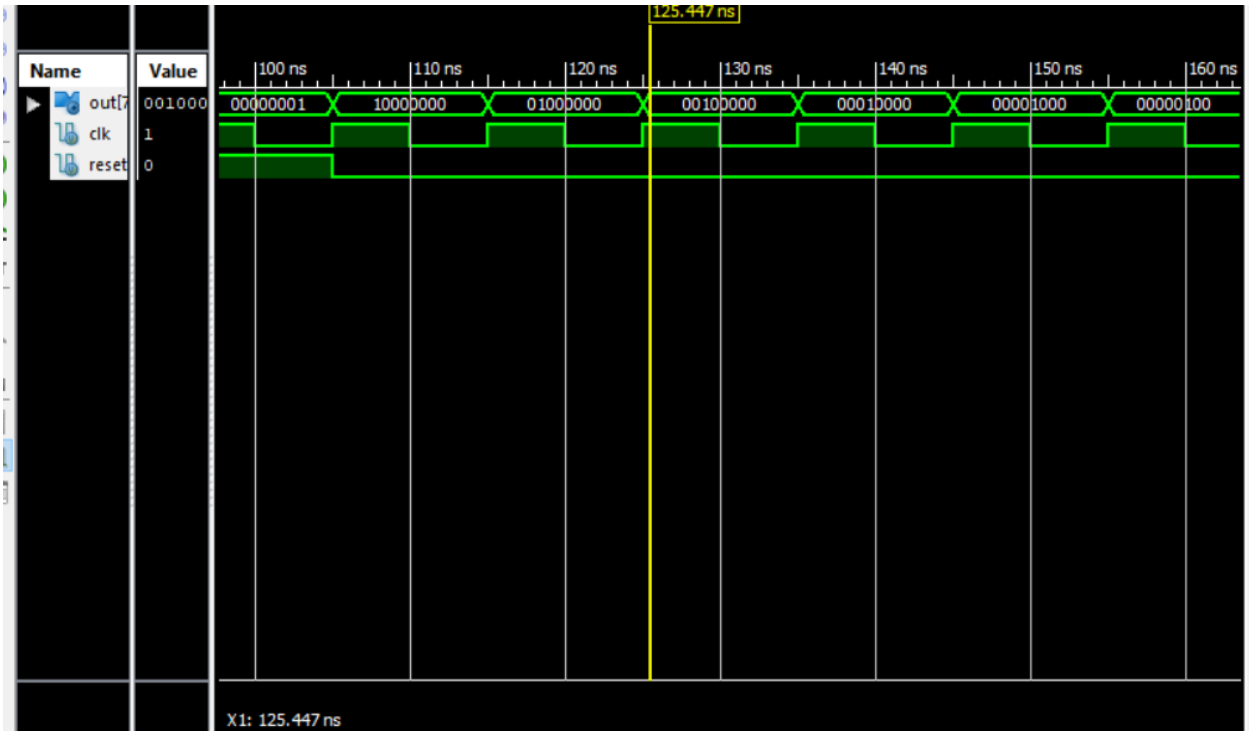
همانطور که مشاهده میشود با ریست کردن LSB خروجی برابر یک خواهد بود و همانند آنچه که در شمارنده جانسون پیاده سازی کرده ایم LSB خروجی به MSB خروجی داده میشود و یک حلقه ایجاد شده که در هر کلاک با حرکت مقدار ها به یک بیت به سمت LSB و دوباره تکرار حلقه به یک RingCounter یا مدار تایمینگ هشت حالته (هشت بیت خروجی) دست پیدا کرده ایم.

تست :

```
module tb;

    // Inputs
    reg clk;
    reg reset;

    // Outputs
    wire [7:0] out;

    // Instantiate the Unit Under Test (UUT)
    RingCounter ringcounter (
        .clk(clk),
        .reset(reset),
        .out(out)
    );
    always
    #5 clk=~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        #5 reset=1;
        #100 reset=0;

    end

endmodule
```

همانند شمارنده جانسون ابتدا ریست میکنیم( مقدار دهی اولیه خروجی ها) سپس در هر 5 نانوثانیه مقدار کلاک را تغییر میدهیم.

شبیه سازی :

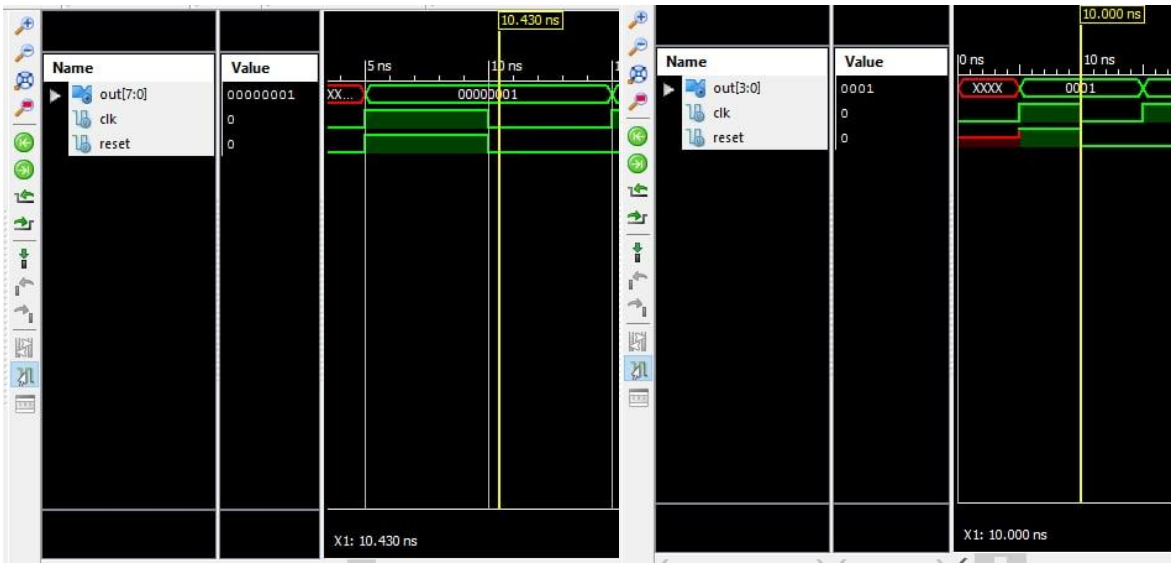



همانطور که انتظار داشتیم در شبیه سازی نیز میتوانیم مشاهده کنیم که در کلاک تنها یک بیت برابر یک است و مانند یک شیفت دهنده عمل میکند.

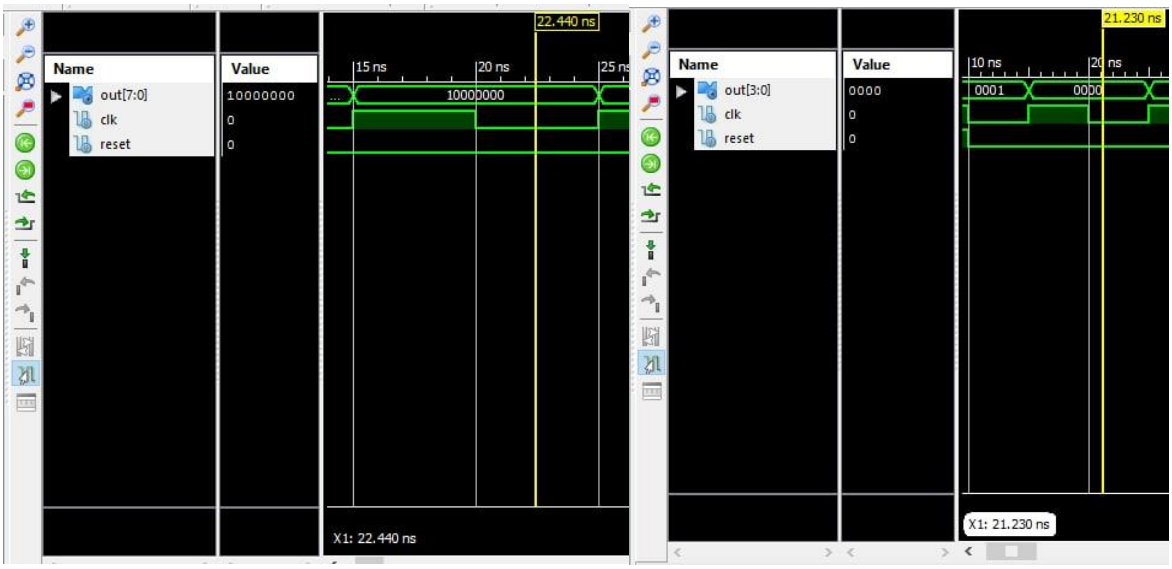
خلاصه طراحی :

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 8 | 126800 | 0% |
| Number of fully used LUT-FF pairs | 0 | 8 | 0% |
| Number of bonded IOBs | 10 | 210 | 4% |
| Number of BUFG/BUFGCTRLs | 1 | 32 | 3% |

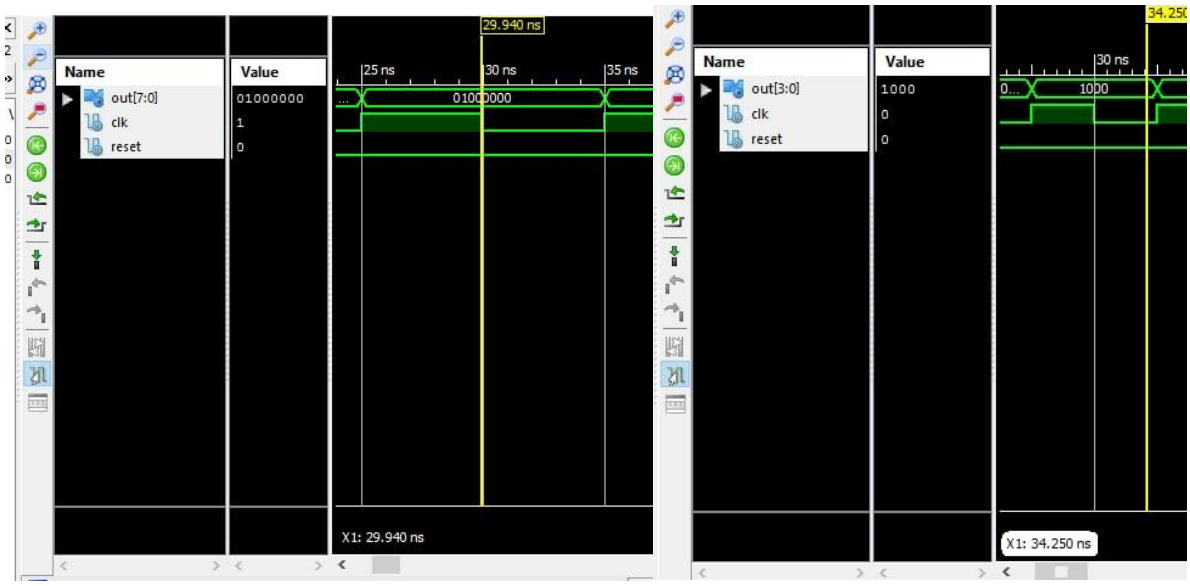
در پایان اگر شبیه سازی Johnson Counter و Ring counter را متناظرا در کنار هم قرار دهیم به راحتی میتوانیم تعداد حرکات Johnson Counter را با استفاده از مدار تایمینگ Ring Counter به دست آوریم.



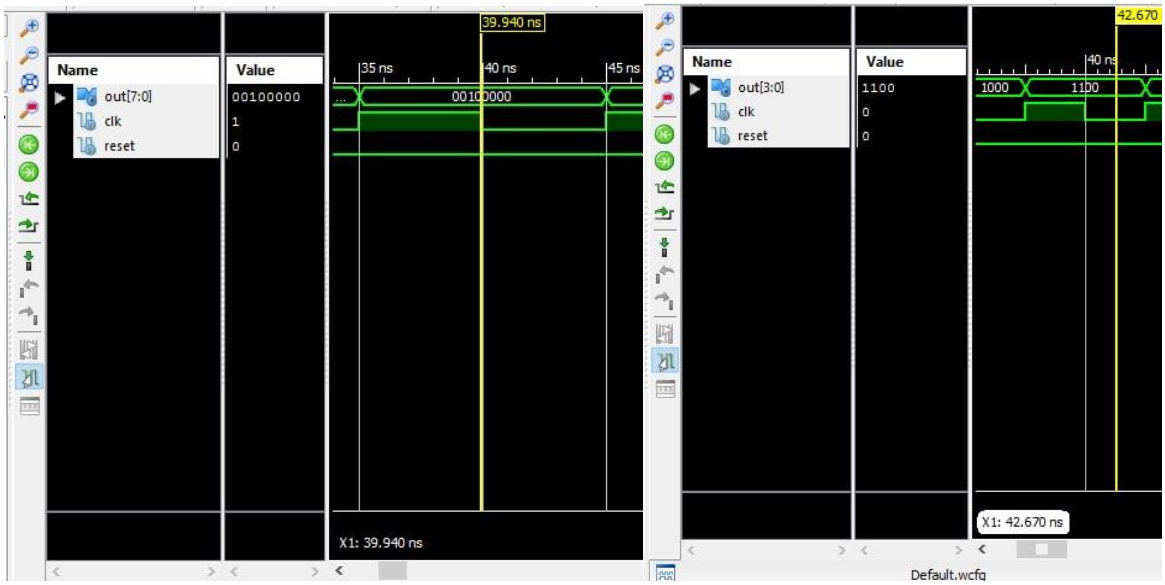

وضعیت اول



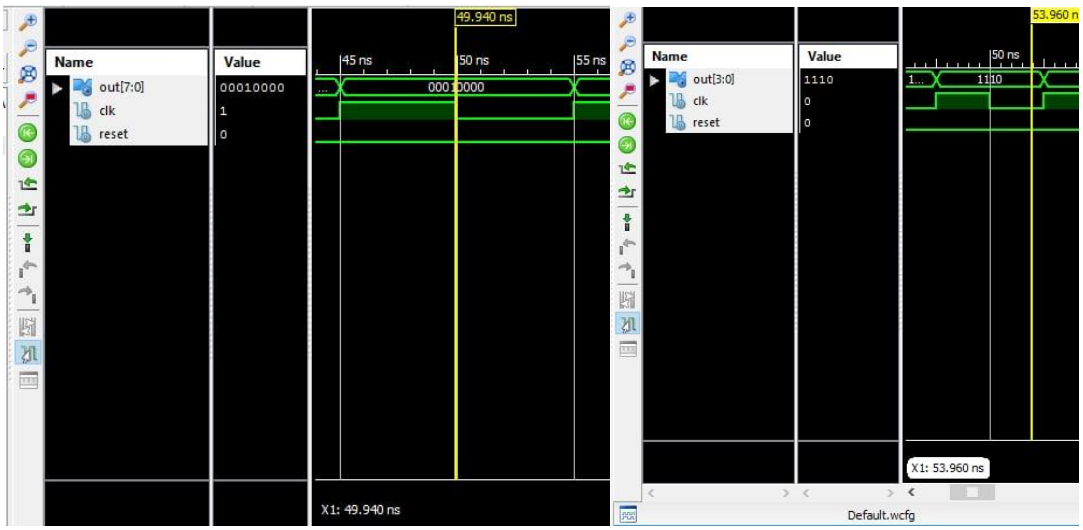

وضعیت دوم



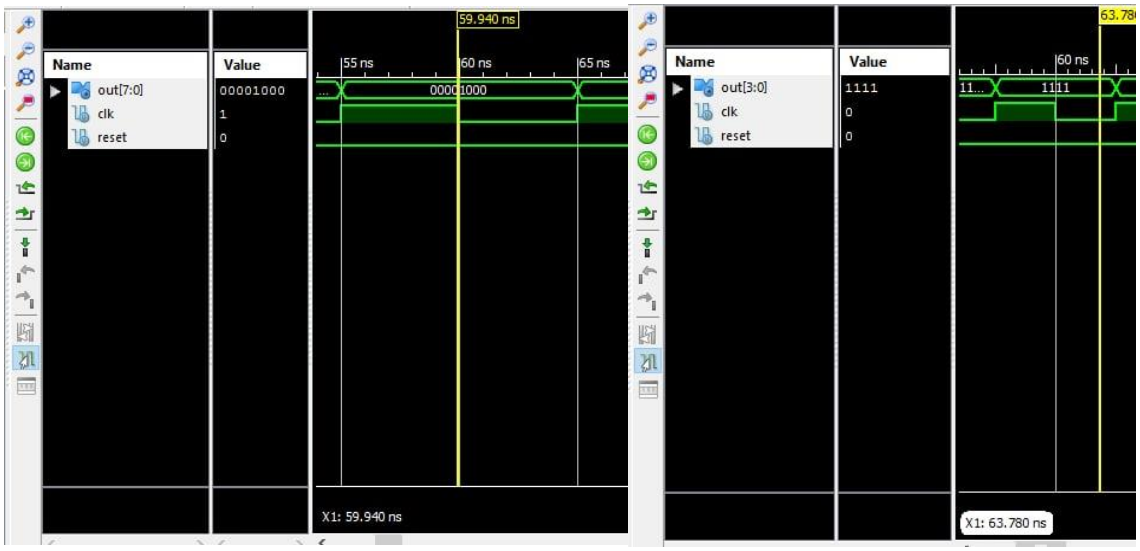

وضعیت سوم



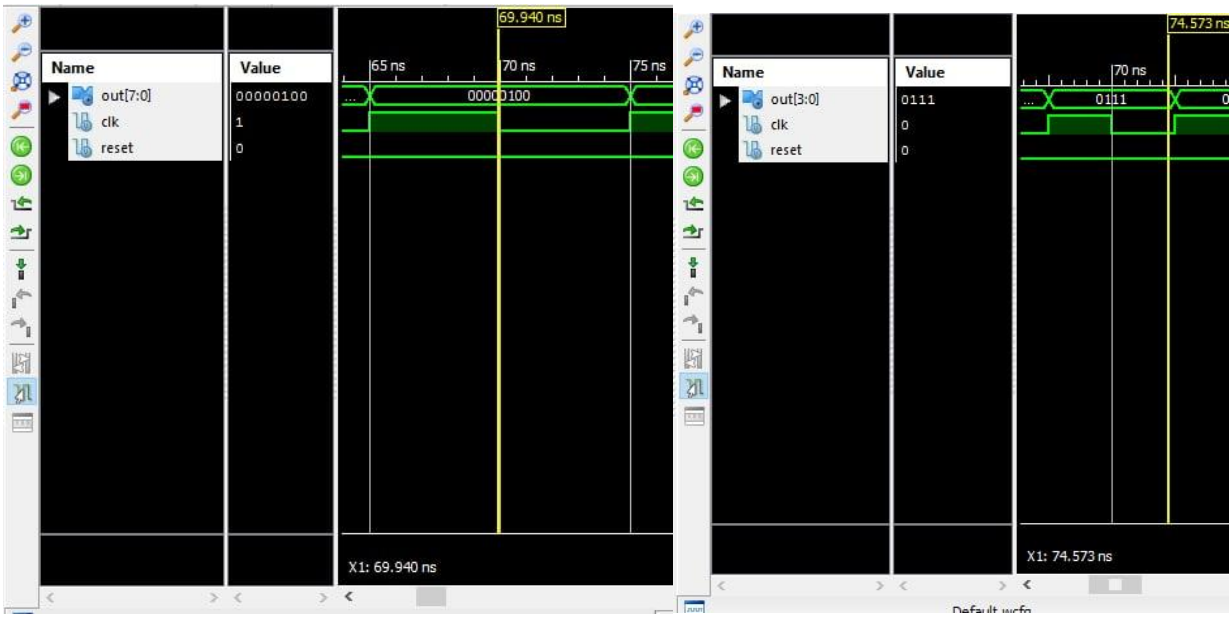

وضعیت چهارم



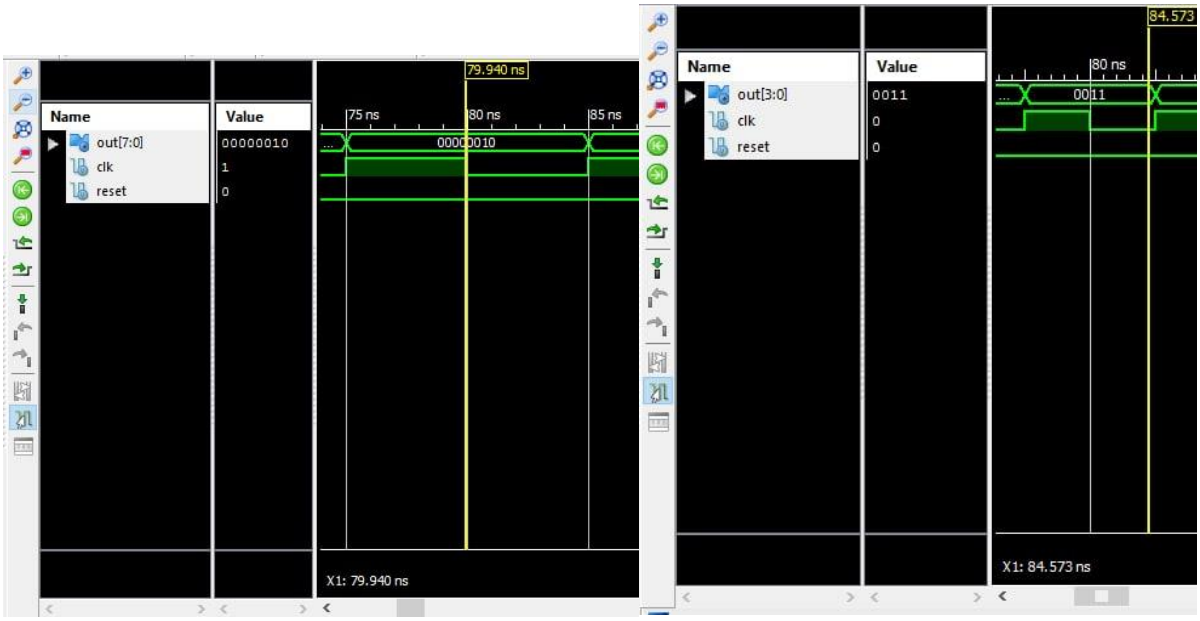

وضعیت پنجم




وضعیت ششم




وضعیت هفتم




وضعیت هشتم

همانطور که میبینم در هشت مرحله حلقه کامل میشود و تکرار میشود پس به درستی شبیه سازی و طراحی انجام شده است.