گزارش آزمایش سوم

دانشگاه صنعتی شریف – زمستان ۹۹

آزمایشگاه طراحی سیستم‌های دیجیتال – دکتر اجلالی

نویسندگان:
سروش جهان‌زاد – ۹۸۱۰۰۳۸۹
علی حاتمی تاجیک – ۹۸۱۰۱۳۸۵

۱ مقدمه

در این سند سعی شده تا گزارشی بر نحوه انجام آزمایش سوم درس آزمایشگاه طراحی سیستم‌های دیجیتال ارائه شود. کدهای مربوط به هر بخش، فایل تست بنچ آن و مدیا مربوط به گزارش در کنار این گزارش آمده است. از صبر و بردباری شما در مطالعه این سند بسیار سپاسگزاریم.

با احترام

سروش جهان‌زاد، علی حاتمی تاجیک – بهار ۱۴۰۰

۲ مقایسه‌کننده ترکیبی

مقایسه‌کننده‌ی تک بیتی (Cascadable 1-bit Comparator)

طراحی مدار

برای این بخش نیاز به یک ماژول داریم که با گرفتن بیت‌های ورودی A و B که باید مقایسه شوند و سه بیت ورودی G_in و E_in و L_in، خروجی G_o و E_o و L_o را به ما بدهد. فرض می‌کنیم بیت A و B به ترتیب بیت با ارزش دو عدد N و M هستند. حروف G و E و L به ترتیب نشان دهنده حالات N > M و N = M و N < M هستند که در آن حالت متناظر برابر ۱ و در غیر آن برابر ۰ خواهند بود.

در طراحی این ماژول فرض می‌کنیم که حاصل مقایسه‌ی دو عدد N و M بدون در نظر گرفتن بیت پر ارزششان (A و B)، در سه بیت ورودی G_in و E_in و L_in داده می‌شود. این ورودی‌ها قابلیت cascadable بودن را ممکن می‌کنند. اگر N و M تک بیتی باشند، چون هیچ بیتی پیش از A و B مقایسه نشده است، دو عدد در حالت یکسانی قرار دارند و بنابراین باید E_in برابر ۱ و G_in و L_in برابر ۰ باشند.

حال مشاهده می‌کنیم که در هر حالت کدام خروجی باید فعال و کدام باید غیرفعال باشد.

اگر A از B بزرگتر باشد، چون این دو بیت با ارزش‌ترین بیت‌های N و M هستند، بدون توجه به بیت‌های کم‌ارزش‌تر می‌توان گفت N > M برقرار است و باید G_o برابر ۱ و دو خروجی دیگر برابر ۰ باشند.

مشابها اگر A از B کوچکتر باشد، چون این دو بیت با ارزش‌ترین بیت‌های N و M هستند، بدون توجه به بیت‌های کم‌ارزش‌تر می‌توان گفت N < M برقرار است و باید L_o برابر ۱ و دو خروجی دیگر برابر ۰ باشند.

در حالتی که A و B برابر باشند، با توجه به بیت‌های کم‌ارزش‌تر که قبلا مقایسه شده‌اند باید تصمیم بگیریم. در این حالت هر کدام از سه ورودی G_in و E_in و L_in که فعال باشد خروجی متناظر با آن فعال خواهد بود

کد

با توجه به حالات گفته‌شده، می‌توانیم جریان داده برای خروجی‌ها را به صورت زیر تعریف کنیم:

```
assign G_o = (A > B) || ((A == B) && G_in);
assign E_o = (A == B) && E_in;
assign L_o = (A < B) || ((A == B) && L_in);
```

بنابراین ماژول طراحی‌شده در نهایت به صورت زیر خواهد بود:

```
module one_bit_comparator (
  A,
  B,
  G_in,
  E_in,
  L_in,
  G_o,
  E_o,
  L_o
);

input wire A, B, G_in, E_in, L_in;
output wire G_o, E_o, L_o;
assign G_o = (A > B) || ((A == B) && G_in);
assign E_o = (A == B) && E_in;
assign L_o = (A < B) || ((A == B) && L_in);


endmodule
```

تست

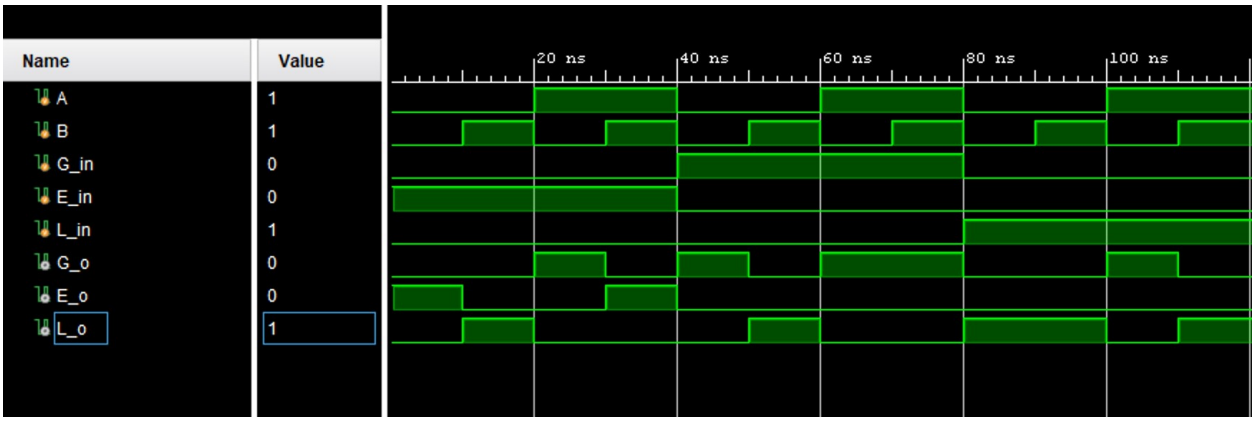
برای تست این ماژول، تست‌بنچی می‌نویسیم تا تمام ورودی‌های ممکن را بررسی کند. دقت می‌کنیم که در هر لحظه تنها یکی از سه ورودی G_in و E_in و L_in می‌تواند برابر یک باشد. در ادامه کد مورد استفاده برای تست این ماژول را مشاهده می‌کنید.

```
module one_bit_comparator_tb;
  reg G_in, E_in, L_in, A, B;
```

```
 wire G_o, E_o, L_o;
 one_bit_comparator comparator(
   .A(A),.B(B),.G_in(G_in),.E_in(E_in),.L_in(L_in),.G_o(G_o),.E_o(E_o),.L_o(L_o));

 initial begin
  G_in <= 0;
  E_in <= 1;
  L_in <= 0;
  A <= 0;
  B <= 0;
 #10;
  A <= 0;
  B <= 1;
  #10;
  A <= 1;
  B <= 0;
  #10;
  A <= 1;
  B <= 1;
  #10;

  G_in <= 1;
  E_in <= 0;
  L_in <= 0;
  A <= 0;
  B <= 0;
  #10;
  A <= 0;
  B <= 1;
  #10 A <= 1;
  B <= 0;
  #10 A <= 1;
  B <= 1;
 #10;

  G_in <= 0;
  E_in <= 0;
  L_in <= 1;
  A <= 0;
  B <= 0;
 #10 A <= 0;
  B <= 1;
 #10 A <= 1;
  B <= 0;
  #10 A <= 1;
  B <= 1;
  #10;
 end
endmodule
```

نتیجه تست

خواهیم دید که شکل موج‌ها به صورت زیر در می‌آید و در نتیجه طراحی ما به درستی عمل می‌کند.



شکل ۱: نتیجه تست مقایسه کننده یک بیتی گسترش پذیر

مقایسه‌کننده ۴ بیتی

طراحی مدار

در این بخش با طراحی سلسله‌مراتبی و با استفاده از ماژول طراحی شده در بخش قبل، یک مقایسه‌کننده‌ی ۴ بیتی طراحی می‌کنیم.

با توجه به طراحی ماژول بخش قبل، در مقایسه‌ی دو عدد با استفاده از آن کافیست از سمت راست بیت‌ها را در نظر بگیریم و بیت i ام را به i امین مقایسه‌کننده بدهیم و خروجی‌های G_o و E_o و L_o از مقایسه‌کننده‌ی i ام را به ورودی‌های G_in و E_in و L_in برای مقایسه‌کننده‌ی $i+1$ ام بدهیم. در مقایسه‌ی بیت سمت راست (صفرم) اعداد هم چون هیچ بیتی پیش از آن مقایسه نشده است، دو عدد در حالت یکسانی قرار دارند و بنابراین باید E_in برابر ۱ و G_in و L_in برابر ۰ باشند.

با این روش، با منطق در نظر گرفته شده حاصل مقایسه از بیت ۰ تا i در مقایسه‌کننده‌ی i ام تولید می‌شود و بنابراین خروجی‌های G_o و E_o و L_o از مقایسه‌کننده‌ی n ام حاصل نهایی مقایسه‌ی دو عدد n بیتی هستند که آنها را به ترتیب G و E و L نام‌گذاری می‌کنیم. ورودی‌های ماژول هم اعداد n بیتی A و B خواهند بود که بیت i ام هر کدام، به ورودی متناظر A و B از مقایسه‌کننده‌ی i ام وصل می‌شوند.

حال برای طراحی یک مقایسه‌کننده‌ی ۴ بیتی، کافیست ۴ نمونه از ماژول بخش قبل را به روشی که توضیح دادیم و به صورت زنجیره‌ای به یکدیگر متصل کنیم و بیت‌های متناظر در ورودی‌های ۴ بیتی A و B را به آن‌ها متصل کنیم. توصیف مدار حاصل به صورت زیر خواهد بود.

```
module four_bit_comparator (
  A,
  B,
  G,
  E,
  L
);
  input wire [3:0] A;
  input wire [3:0] B;
  output wire G, E, L;
  wire [3:0] G_in;
  wire [3:0] E_in;
  wire [3:0] L_in;

  assign G_in[0] = 0;
  assign E_in[0] = 1;
  assign L_in[0] = 0;

  one_bit_comparator c0(
    .A(A[0]), .B(B[0]),
    .G_in(G_in[0]), .E_in(E_in[0]), .L_in(L_in[0]),
    .G_o(G_in[1]), .E_o(E_in[1]), .L_o(L_in[1])
  );
  one_bit_comparator c1(
    .A(A[1]), .B(B[1]),
    .G_in(G_in[1]), .E_in(E_in[1]), .L_in(L_in[1]),
    .G_o(G_in[2]), .E_o(E_in[2]), .L_o(L_in[2])
  );
  one_bit_comparator c2(
    .A(A[2]), .B(B[2]),
    .G_in(G_in[2]), .E_in(E_in[2]), .L_in(L_in[2]),
    .G_o(G_in[3]), .E_o(E_in[3]), .L_o(L_in[3])
  );
  one_bit_comparator c3(
    .A(A[3]), .B(B[3]),
    .G_in(G_in[3]), .E_in(E_in[3]), .L_in(L_in[3]),
    .G_o(G), .E_o(E), .L_o(L)
  );

endmodule
```

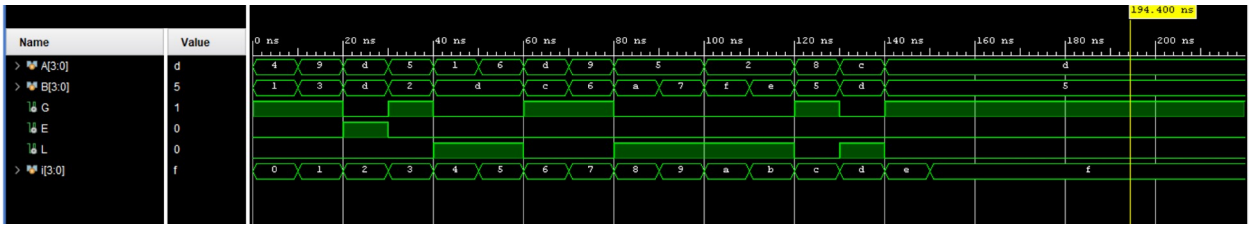
تست

برای بررسی صحت کارکرد این مدار، تست‌بنچ زیر را در نظر می‌گیریم. با توجه به تعداد زیاد حالات ممکن برای ورودی‌های A و B، از مقادیر رندوم برای آن‌ها استفاده می‌کنیم.

```
module four_bit_comparator_tb;
  reg [3:0] A;
  reg [3:0] B;
  wire G, E, L;

  four_bit_comparator comparator(.A(A),.B(B),.G(G),.E(E),.L(L));

  reg [3:0] i;
  initial begin
    for(i = 0; i < 15; i = i + 1) begin
      A = $random;
      B = $random;
      #10;
    end
  end
endmodule
```

نتایج تست

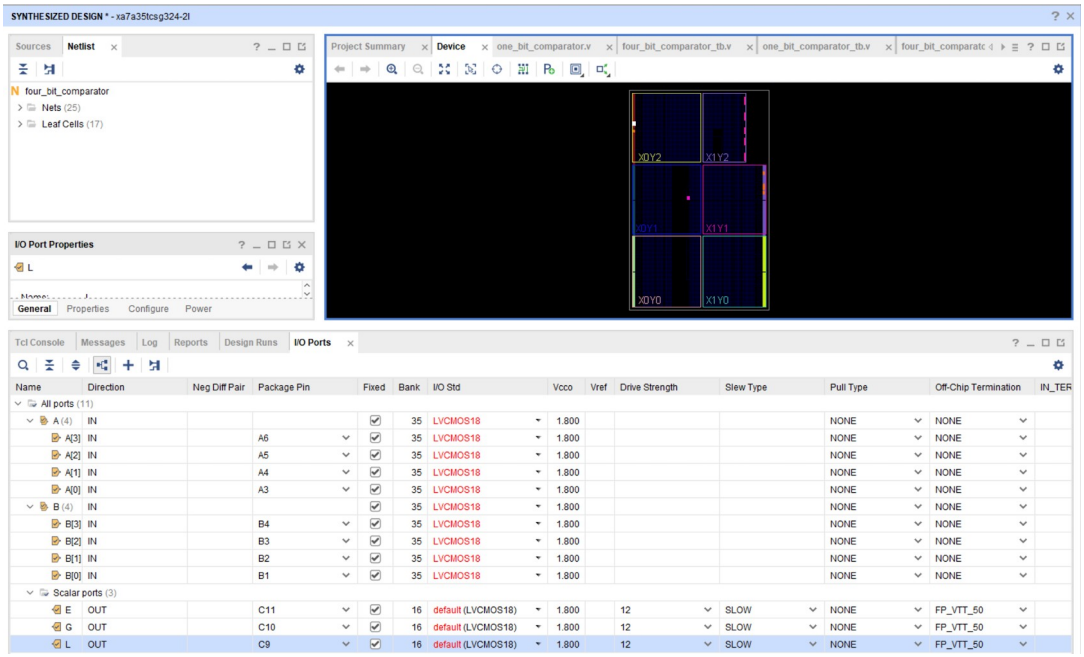
پس از اجرای شبیه‌سازی، مشاهده می‌کنیم که شکل موج‌ها به صورت زیر در می‌آید و در حالات بررسی شده، عملکرد مدار درست و طبق انتظار است.



شکل ۲: نتیجه تست مقایسه کننده چهاربیتی

سنتز

در نهایت برای اطمینان از درستی کامل و سنتزپذیری مدار، با استفاده از ابزار سنتز ویوادو آن را برای یکی از نمونه‌های موجود در این نرم افزار سنتز می‌کنیم. در این آزمایش از xa7a15tcsg324-2l استفاده می‌کنیم. پس از انجام سنتز، مشاهده می‌کنیم که همه چیز به درستی پیش رفته و آزمایش موفقیت‌آمیز بوده است.



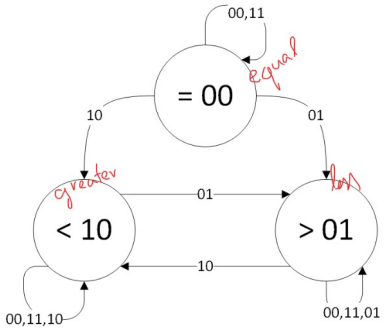
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TER
All ports (11)													
A (4)	IN			☑	35	LVCMOS18	1.800				NONE	NONE	
A[3]	IN		A6	☑	35	LVCMOS18	1.800				NONE	NONE	
A[2]	IN		A5	☑	35	LVCMOS18	1.800				NONE	NONE	
A[1]	IN		A4	☑	35	LVCMOS18	1.800				NONE	NONE	
A[0]	IN		A3	☑	35	LVCMOS18	1.800				NONE	NONE	
B (4)	IN			☑	35	LVCMOS18	1.800				NONE	NONE	
B[3]	IN		B4	☑	35	LVCMOS18	1.800				NONE	NONE	
B[2]	IN		B3	☑	35	LVCMOS18	1.800				NONE	NONE	
B[1]	IN		B2	☑	35	LVCMOS18	1.800				NONE	NONE	
B[0]	IN		B1	☑	35	LVCMOS18	1.800				NONE	NONE	
Scalar ports (3)													
E	OUT		C11	☑	16	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
G	OUT		C10	☑	16	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
L	OUT		C9	☑	16	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	

شکل ۳: نتیجه سنتز و امپلیمنتیشن مدار

۳ مقایسه‌کننده سریال

طراحی مدار

این مدار به صورت زیر طراحی شده است که یک مدار ترتیبی سه وضعیتی است. به طور خلاصه اگر در هر وضعیتی بودیم اگر بیت با ارزش بیشتر بعدی با هم برابر بود در همان وضعیت میمانیم و غیر از این حالت اگر بیت اول صفر و دیگری یک بود به حالت کوچکتر و اگر برعکس بود به حالت بزرگتر خواهیم رفت. اگر از همان ابتدا نیز بیتها با یکدیگر برابر بودند در استیت صفر یا همان برابری باقی خواهیم ماند.



شکل ۴–حالتهای مقایسه کننده سریال

حال برای حالتهای بالا با استفاده از دو فلیپفلاپ نوع دی، جدول حالت و جداول کارنو ساده سازی ها را انجام میدهیم از آنجایی که تضمین شده است که قبل از مقایسه حتما ماژول ریست خواهد شد به همین خاطر حالتی را که فلیپفلاپها هردو یک باشند را در نظر نمیگیریم

Q_1	Q_0	a	b	Q_1^+	Q_0^+
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	X

جدول ۱–جدول حالت مقایسه کننده سریال

Q_1Q_0\ab	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	1	1

جدول ۲–جدول کارنو برای *D1*

Q_1Q_0\ab	00	01	11	10
00	0	1	0	0
01	1	1	1	0
11	X	X	X	X
10	0	1	0	0

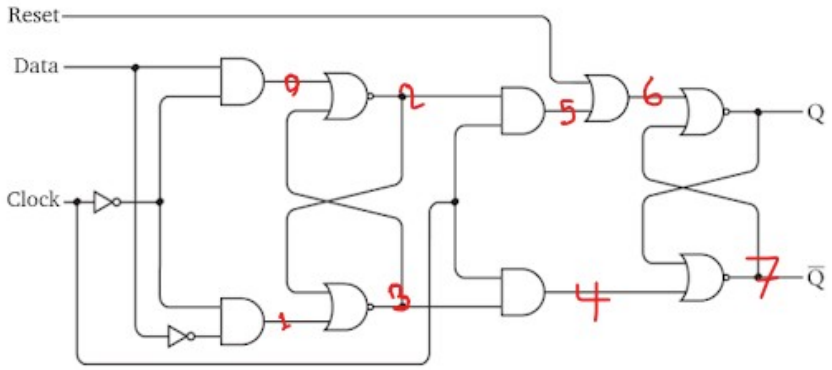
جدول ۳-جدول کارنو برای *D0*

که نتایج این دو جدول به صورت روبه‌رو خواهد بود:

$$D_1 = a\,\bar{b} + Q_1(a + \bar{b})$$
$$D_0 = \bar{a}\,b + Q_0(\bar{a} + b)$$

ساخت DFF

برای نگهداری از وضعیت فعلی مدار به دو فلیپ فلاپ نوع دی نیازمندیم. در ادامه شکلی از شماتیک فلیپ فلاپ پیادهسازی شده آمده است (ایندکس وایرها که داخل کد از آن‌ها به عنوان واسط استفاده شده است آمده است). پس از آن کد مربوط به یک فلیپ فلاپ آورده شده است که در ماژول مقایسه کننده این کد دو بار استفاده شده‌است و ورودی آن‌ها یعنی D1 و D2 با استفاده از خروجی آن‌ها تعیین شده است. در ابتدا نیز تمام سیمها را صفر میکنیم تا به مشکلی برنخوریم. برای صحت کارایی فلیپ فلاپ، آنرا به صورت یک ماژول درآورده‌ایم تا آنرا تست کنیم (فایل ماژول dff و تست آن نیز در پوشه مربوط به این بخش از آزمایش موجود است) ولی در ماژول اصلی دو دفعه از این کد آمده است تا فلیپ فلاپ ها ساخته شوند.



شکل ۵-*D Flip-Flop*

```
wire connectorFF[7:0];
wire q;
wire D;
assign connectorFF[0] = D && (~clk);
assign connectorFF[1] = (~D) && (~clk);
assign connectorFF[2] = ~(connectorFF[0] || connectorFF[3]);
assign connectorFF[3] = ~(connectorFF[1] || connectorFF[2]);
assign connectorFF[4] = connectorFF[3] && clk;
assign connectorFF[5] = connectorFF[2] && clk;
assign connectorFF[6] = connectorFF[5] || reset;
assign connectorFF[7] = ~(connectorFF[4] || q);
assign q              = ~(connectorFF[6] || connectorFF[7]);
```

ساخت مدار نهایی

حال که چهار سیم D_0, D_1 را داریم با توجه به نتایج بدست آمده از قسمت طراحی مدار این چهار سیم را مقدار دهی میکنیم و پس از آن نیز خروجی ها را اساین کرده (که عبارت بولی آن طبق شکل ۴ واضح است) و کارمان به پایان میرسد.

```
assign D1 = (a && (~b)) || (q1 && (a || (~b)));
assign D0 = (b && (~a)) || (q0 && (b || (~a)));
assign less = (~q1) && q0;
assign greater = (~q0) && q1;
assign equal = (~q1) && (~q0);
```

کد نهایی

```
module serial_comparator(
    input clk,
    input a,
    input b,
    input reset,
    output greater,
    output equal,
    output less
    );

    // Assigning D0, D1
   wire D1;
   wire D0;
   wire q1;
   wire q0;
   assign D1 = (a && (~b)) || (q1 && (a || (~b)));
   assign D0 = (b && (~a)) || (q0 && (b || (~a)));

   // Q1 D-Flip-Flop Assignments
   wire connectorFF[7:0];
   assign connectorFF[0] = D1 && (~clk);
   assign connectorFF[1] = (~D1) && (~clk);
   assign connectorFF[2] = ~(connectorFF[0] || connectorFF[3]);
   assign connectorFF[3] = ~(connectorFF[1] || connectorFF[2]);
   assign connectorFF[4] = connectorFF[3] && clk;
   assign connectorFF[5] = connectorFF[2] && clk;
   assign connectorFF[6] = connectorFF[5] || reset;
   assign connectorFF[7] = ~(connectorFF[4] || q1);
   assign q1             = ~(connectorFF[6] || connectorFF[7]);

   // Q1 D-Flip-Flop Assignments
   wire connectorFF0[7:0];
   assign connectorFF0[0] = D0 && (~clk);
   assign connectorFF0[1] = (~D0) && (~clk);
   assign connectorFF0[2] = ~(connectorFF0[0] || connectorFF0[3]);
   assign connectorFF0[3] = ~(connectorFF0[1] || connectorFF0[2]);
   assign connectorFF0[4] = connectorFF0[3] && clk;
   assign connectorFF0[5] = connectorFF0[2] && clk;
   assign connectorFF0[6] = connectorFF0[5] || reset;
   assign connectorFF0[7] = ~(connectorFF0[4] || q0);
   assign q0              = ~(connectorFF0[6] || connectorFF0[7]);


   // Assigning Outputs
   assign less = q0;
   assign greater = q1;
   assign equal = (~q1) && (~q0);
endmodule
```

تست مدار

تمام کارایی های ماژول را درون یک تست می‌گنجانیم. مدار را برای سه جفت عدد تست می‌کنیم و بین هرکدام از reset ماژول استفاده می‌کنیم.

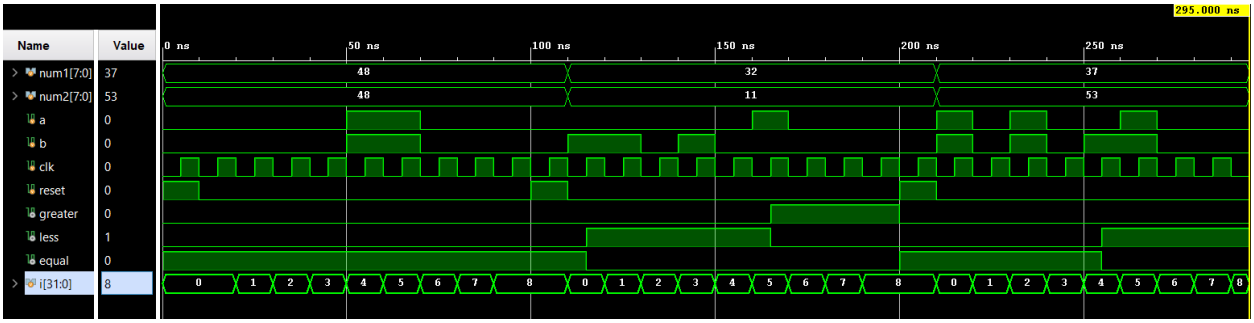
کد تست

در ادامه کد مورد استفاده برای تست ماژول را می‌بینیم.

```
module test_serial_comparator;
    reg [7:0]num1 = 48;
    reg [7:0]num2 = 48;
    reg a=0, b=0, reset = 0, clk = 0;
    wire greater, less, equal;
    serial_comparator uut(clk,a,b,reset,greater,equal,less);
    initial begin
    forever
        #5 clk = ~clk;
    end
    integer i=0;
    initial begin
        reset = 1; #10 reset = 0;
        for (i=0; i<8; i = i+1) begin
           a = num1[i];
           b = num2[i];
           #10;
        end
        #10 reset = 1;#10 reset = 0;
        num1 = 32; num2 = 11;
        for (i=0; i<8; i = i+1) begin
           a = num1[i];
           b = num2[i];
           #10;
        end
        #10 reset = 1;#10 reset = 0;
        num1 = 37;num2 = 53;
        for (i=0; i<8; i = i+1) begin
           a = num1[i];
           b = num2[i];
           #10;
        end
        #5 $finish;
    end
endmodule
```

نتایج تست

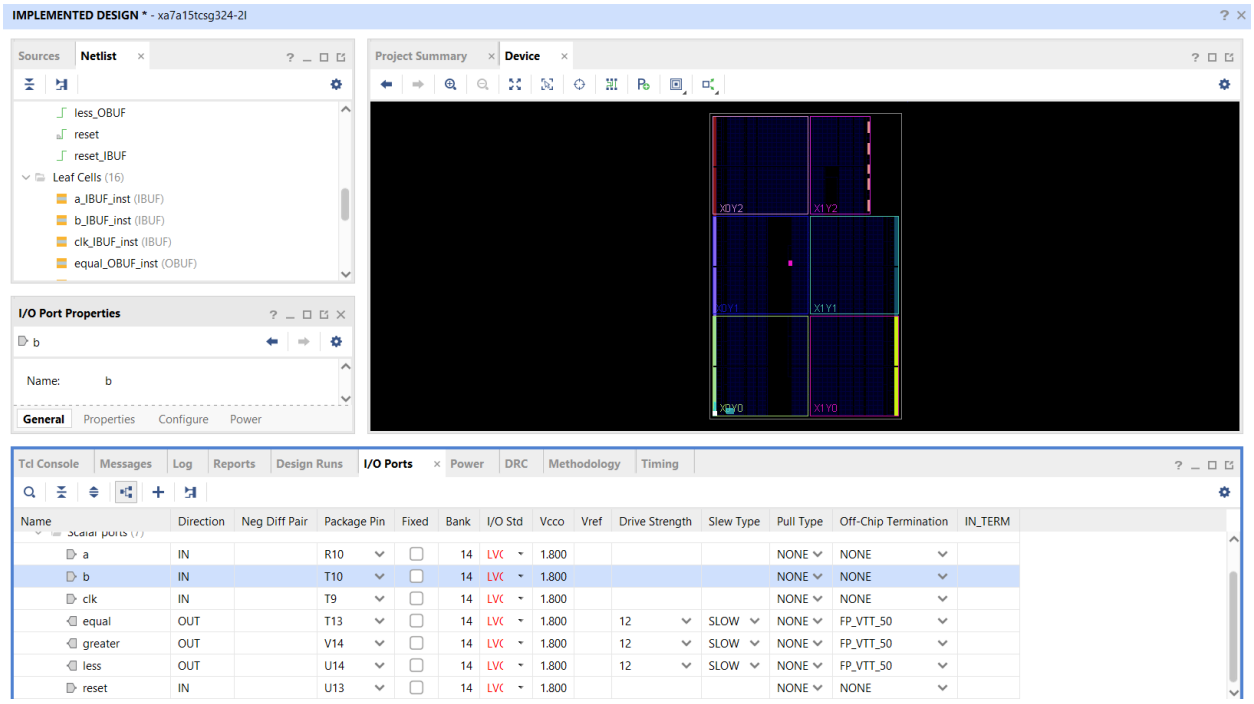
تصویر زیر نیز نتیجه اجرای تست بالاست. همانطور که مشخص است ریست به صورت سریع تأثیر خود را میگـذارد امـا تغیـیرات در a وb تنهـا در لبه‌هـای بالارونده است. همینطور زمانی که I برابر با ۸ می‌شود زمانی است که نتایج نهایی قابل مشاهده هستند. که ابتدا دو عدد برابر هستند، بعد عـدد اول بزرگ‌تـر است و سپس عدد اول کوچتر است. اما این‌ها تنها نتایج نهایی هستند و با توجه به a و b و num1 و num2 مشخص است که تأثیر گذاری بیت بـه بیت است (مثلاً در بخش دوم زمانی که اعداد ۱۱ و ۳۲ مقایسه می‌شوند، عدد ۱۱ در ۵ بیت ابتدایی بزرگ‌تر از عدد ۳۲ است اما در بیت ششم بزرگ‌تر شده است. اینکه در آن مرحله کدام بیت مورد برسی است از مقدار I مشخص می‌شود).



شکل ۶: نتیجه تست بر روی مقایسه‌کننده سریال

سنتز

در نهایت برای اطمینان از درستی کامل و سنتزپذیری مدار، با استفاده از ابزار سنتز ویوادو آن را برای یکی از نمونه‌های موجود در این نرم افزار سنتز می‌کنیم. در این آزمایش از xa7a15tcsg324-2l استفاده می‌کنیم. پس از انجام سنتز، مشاهده می‌کنیم که همه چیز به درستی پیش رفته و آزمایش موفقیت‌آمیز بوده است.



شکل ۷: نتیجه سنتز و امپلیمنتیشن مدار مقایسه کننده سریال