

باسمه تعالی

جزوه درس طراحی خودکار سیستم های دیجیتال

مدرس: رضا سعیدی نیا

برای مشاهده سایر آثار رضا سعیدی نیا به سامانه زیر مراجعه کنید:

<http://www.idars.ir>

بخش دانشگاهی - فنی مهندسی

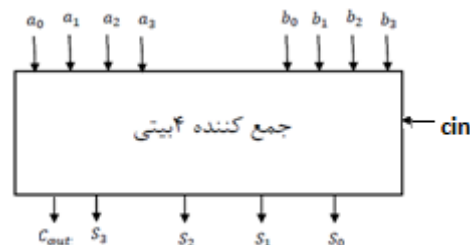
فصل اول مقدمه

در این درس هدف ارائه روش های طراحی خودکار سیستم های دیجیتال می باشد. برای طراحی سخت افزار محیط های نرم افزاری مثل Modelsim و MaxPlus II ایجاد شده است. طراح قبل از ارائه طرح نهایی ابتدا طرح را در این محیط ها آماده می کند و مرحله به مرحله طرح خود را تست و خطایابی می کند در صورت عدم وجود خطا به مراحل پیشرفته تر گام می گذارد. در انتها طرح نهایی را به کارخانه سازنده آی سی ارائه می کند. در این درس ما از زبان برنامه نویسی توصیف سخت افزار^۱ (HDL) برای انجام این مراحل استفاده می کنیم. مهمترین زبان های توصیف سخت افزار VHDL و Verilog می باشد که ما زبان Verilog را ارائه می دهیم. برای طراحی، شبیه سازی و مدل سازی از نرم افزار modelsim استفاده می کنیم.

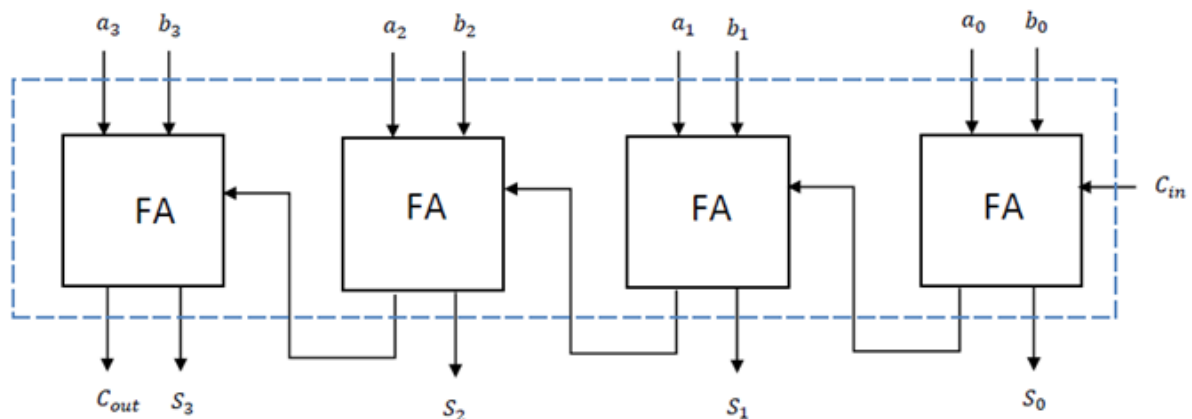
برای طراحی یک سخت افزار سطوح مختلفی وجود دارد که آن را با یک مثال ساده ارائه می کنیم.

سطوح سخت افزار: (مثال) سخت افزاری طراحی کنید که ۲ عدد ۴ بیتی را با هم جمع کند؟

۱- سطح رفتاری^۲: در این مرحله عملکرد مدار مشخص می شود. یعنی $S=a+b$.



۲- سطح مسیر داده^۳: در این مرحله مسیر داده بین قطعات مشخص می شود. مسیر داده بیشتر بین ثبات ها، گذرگاه ها و حافظه مطرح است. اما به خاطر تبیین اولیه مساله، مسیر تبادل بین بخش های مختلف را مد نظر قرار می دهیم.



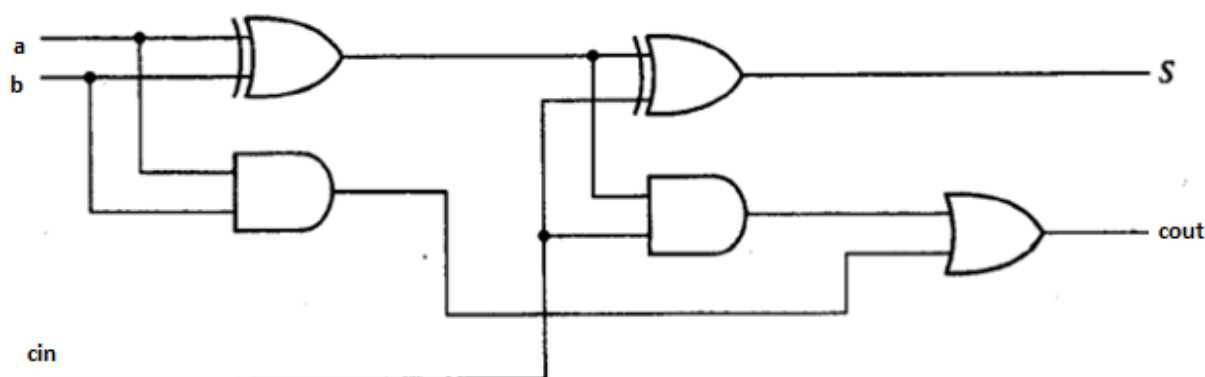
۳- سطح ساختاری^۴: در این سطح قطعات مختلف مدار طراحی می شوند. در این سطح یا از قطعات موجود استفاده می شود یا هر قطعه به مرحله گیت های منطقی می رسد. بنابراین شکل قبلی خود در سطح ساختاری می باشد با فرض وجود قطعات FA در کتابخانه قطعات. اما می توانیم هر FA را از گیت های تشکیل دهنده آن بسازیم.

¹ Hardware Description Language

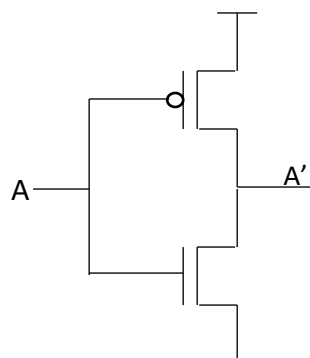
² Behavioral Level

³ Datapath level

⁴ structural Level



۴- سطح ترانزیستور^۱: در این سطح ترانزیستورهای سازنده هر گیت مشخص می شوند مثلاً یک گیت Not در تکنولوژی CMOS بصورت زیر ساخته می شود:



۵- سطح لایه بندی^۲: در این سطح طرح نهایی به کارخانه ساخت عرضه می شود و آی سی ساخته می شود.

بطور کلی طراحی یک سیستم دیجیتال در مراحل زیر انجام می شود



چه موقع یک طرح کامل است؟

نیازی نیست که برای طراحی سخت افزار همه مراحل تا سطح گیت و ترانزیستور از ابتدا انجام شود. این مراحل توسط نرم افزار انجام می شود. یک طرح زمانی کامل است که یک ایده به توصیف معماری مسیر داده یا ساختاری برسد.

DA^۳ چیست؟

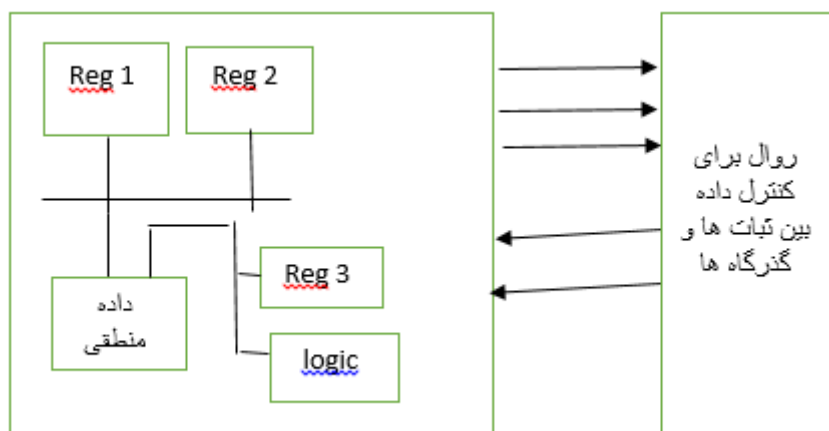
فعالیت هایی مثل تبدیل یک سطح طراحی به سطح دیگر، بررسی خروجی یک مرحله طراحی یا تولید داده های تست توسط کامپیوتر را خودکار سازی طراحی می گویند.

¹ Transistor level

² Layout Level

³ Design Automation

مثالی از طراحی مسیر داده: در طراحی یک CPU ثبات ها و ALU مشخص می شود و ارتباط بین آنها و گذرگاه های آن مشخص می شود به این مرحله مسیر داده می گویند.



فصل دوم مدل سازی سخت افزاری

چه سطح از مدل سازی مورد نیاز است؟

- ۱- اگر مدل مورد استفاده برای مستند سازی عملکرد مدار مورد استفاده شود مدل سطح رفتاری مورد نیاز است.
- ۲- اگر مدل برای تست جزییات زمانی مدار مورد استفاده باشد جزییات بیشتری در طراحی مورد استفاده است (سطح مسیر داده و گیت).

ابزار های مدل سازی در دسترس برای یک مهندس

- ۱- قلم و کاغذ
- ۲- برنامه های ایجاد شماتیک ORCAD
- ۳- قابلیت های بردبرد bread board
- ۴- زبان های توصیف سخت افزار HDL(hardware description language)

وظیفه HDL چیست؟

برای توصیف سخت افزار به منظور شبیه سازی (simulation) مدل سازی، تست، طراحی و مستند سازی سیستم های دیجیتال استفاده می شود.

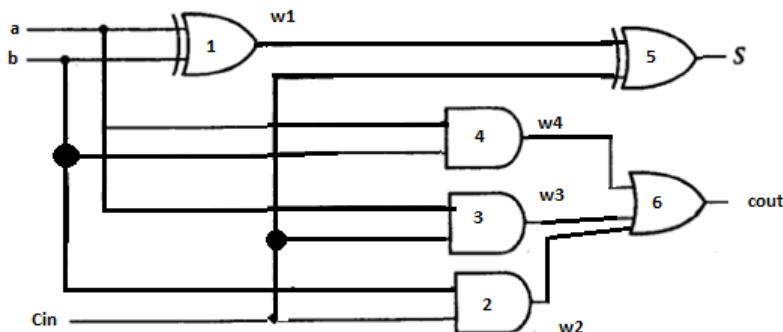
انواع نرم افزار های HDL

- ۱- برنامه شبیه سازی: می تواند برای تست طراحی استفاده شود (بررسی verification)
- ۲- برنامه سنتز (synthesizer): برای تولید سخت افزار به صورت خودکار

نمونه های زبان های توصیف سخت افزار Verilog, VHDL می باشند.

مثال) با استفاده از Verilog در سطح گیت یک fulladder طراحی کنید؟

رابطه بین ورودی ها و خروجی های cout و S بصورت $S = a \oplus b \oplus cin$ و $Cout = a.b + a.Cin + b.Cin$ می باشد. برای طراحی آن با زبان VHDL با فرض وجود گیت ها بصورت یک کامپوننت می توان بصورت زیر نوشت:



```
entity fulladder is
    port (a,b,Cin: in bit ; s, Cout : out bit)
end fulladder
```

```

architecture gate_level of fulladder is
    component and2 port (I1,I2 : IN BIT ,O1: out bit)
    end component;
    component xor2 port (I1,I2 : IN BIT ,O1: out bit)
    end component;
    component or3 port (I1,I2,I3 : IN BIT ,O1: out bit)
    end component;
    signal w1,w2,w3,w4 : bit;
begin
    G1: xor2 portmap (a,b,w1);
    G2: and2 portmap (b,Cin,w2);
    G3: and2 portmap (a,Cin,w3);
    G4: and2 portmap (a,b,w4);
    G5: xor2 portmap (w1,Cin,s);
    G6: or3 portmap (w4,w2,w3, Cout);
end
end gate_level

```

عناصر وریلاگ: دو عنصر اصلی برای نوشتن سخت افزار در وریلاگ module, primitive می باشند. برای تعریف ماژول از ساختاری بصورت زیر استفاده می کنیم:

عنصر ماژول:

```

module module-name
    List of ports
    Declarations
    Specification of the functionality of module
endmodule

```

یک ماژول را می توان در سطوح مختلف طراحی کرد:

۱- سطح رفتاری

```

module module-i
    تعریف پورت ها
    توصیف
    عملکرد رفتاری ماژول
    ماژول بدون توصیف زمانی
endmodule;

```

۲- سطح ساختاری

```

module module-i
    تعریف پورت ها
    توصیف
    توصیف ساختاری
    ماژول با اطلاعات زمانی
endmodule;

```

primitive name

پورت های سطح بیت

توصیف

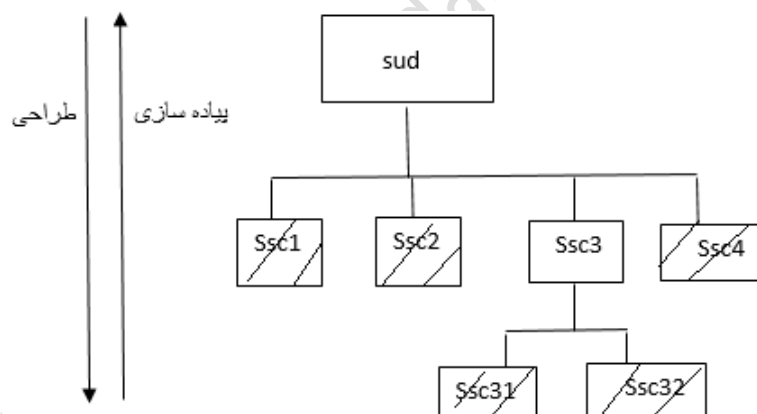
عملکرد گیت به صورت جدولی

end primitive;

طراحی بالا به پایین

برای طرح های بزرگ به جای حل کردن کل سیستم به یک باره از روش تقسیم و غلبه استفاده می کنیم. آن قدر تقسیم می کنیم تا به یک جز قابل مدیریت برسیم. جزیی قابل مدیریت است که در کتابخانه های نرم افزار موجود باشد یا به سطح گیت برسد یا از قطعاتی که از قبل طراحی کرده ایم برسیم و قابل سنتز باشد.

درخت طراحی به روش بالا به پایین : در طراحی بالا به پایین سیستم اولیه را سیستم تحت طراحی^۱ می گویند. با تقسیم آن به زیر قطعات^۲ مرحله دوم درخت طراحی بالا به پایین ایجاد می شود. اگر به قطعه ای به سطح قابل مدیریت رسید طراحی تمام می شود در غیر این صورت تقسیم ادامه می یابد. مثلاً در شکل زیر فرض می کنیم Ssc1, Ssc2, Ssc4 به سطح قابل مدیریت رسیده اند و تقسیم ادامه نمی یابد. ولی قطعه Ssc3 هنوز قابل تقسیم است آن را به زیر قطعات دیگر مثل Ssc31, Ssc32 تقسیم می کنیم. هنگامی که مراحل تقسیم تمام شد طراحی بالا به پایین تمام می شود و سپس مرحله پیاده سازی انجام می شود و قطعه از پایین به بالا پیاده سازی می شود. در شکل قطعاتی که قابل تقسیم نیستند را هاشور زده ایم.



روال تقسیم بازگشتی

Partition (system)

If hardware mapping of (system) is done then

Save hardware of system

Else for every functionally-design part-I of system

Partition(part-I)

End for

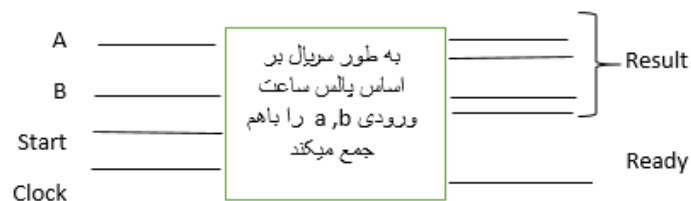
End if

End Partition

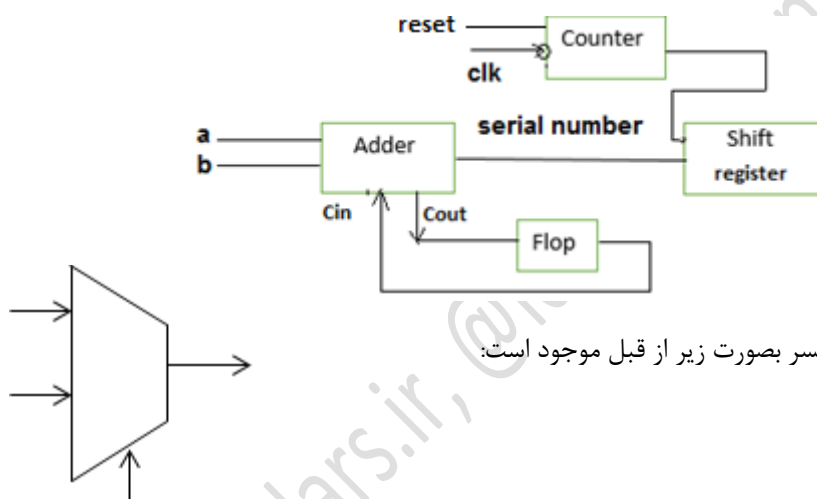
¹ System Under Design (SUD)

² System Sub Component (SSC)

مثال طراحی بالا به پایین: می‌خواهیم یک جمع کننده سریال طراحی کنیم خروجی جمع کننده ۸ بیتی خواهد بود. فرض می‌کنیم کتابخانه یک مالتی پلکسر و یک دی فلیپ فلاپ دارد. کلیت مدار بصورت زیر است:



ورودی A, B بصورت سریال تک بیت، تک بیت وارد می‌شود هر بیت ورودی در یک پالس ساعت به مدار وارد می‌شود. شروع عمل با یک شدن سیگنال start اتفاق می‌افتد. هنگامی که نتیجه ۸ بیت آماده شد خروجی آن در result قرار می‌گیرد. و سیگنال ready یک می‌شود تا به مدار مصرف کننده نتیجه اعلام خاتمه عمل نماید. طرح اولیه جمع کننده را بصورت زیر در نظر می‌گیریم که باید به روش طراحی بالا به پایین به مراحل قابل مدیریت یعنی گیت ها و مالتی پلکسر و فلیپ فلاپ برسند.



فرض می‌کنیم ماژول مالتی پلکسر بصورت زیر از قبل موجود است:

```

ModelSim PE Student Edition (2014)
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
Layout NoDesign Columns AllColumns
//multiplexer 2 to 1
timescale 1ns/100ps

module mux2_1(sel,data1,data0,z);
    input sel,data1,data0;
    output z;
    assign #6 z=(sel==1'b1)? data1:data0;
endmodule

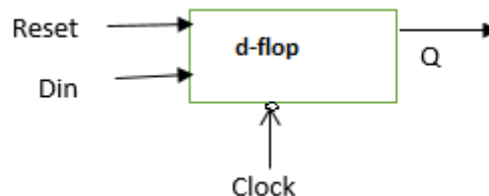
-- Compile of Dec3x8RTL.V was successful.

```

در این ماژول sel, data1, data0 ورودی مالتی پلکسر و z خروجی آن می‌باشد.

دستور assign #6 z=(sel==1'b1)? data1:data0; برای انتساب به خروجی Z استفاده می‌شود. عملگر ? برای مقدار دهی شرطی استفاده می‌شود. در این مثال اگر sel برابر ۱ باشد z=data1 خواهد شد و اگر z=0 شود z=data0 می‌شود. timescale 1ns/100ps مقیاس زمانی را مشخص می‌کند که یک نانو ثانیه است با دقت صد پیکو ثانیه. بعد از دستور assign مقدار #6 مشخص کننده تاخیر عمل بر نانو ثانیه است. یعنی Z بعد از ۶ نانوثانیه مقدار می‌گیرد.

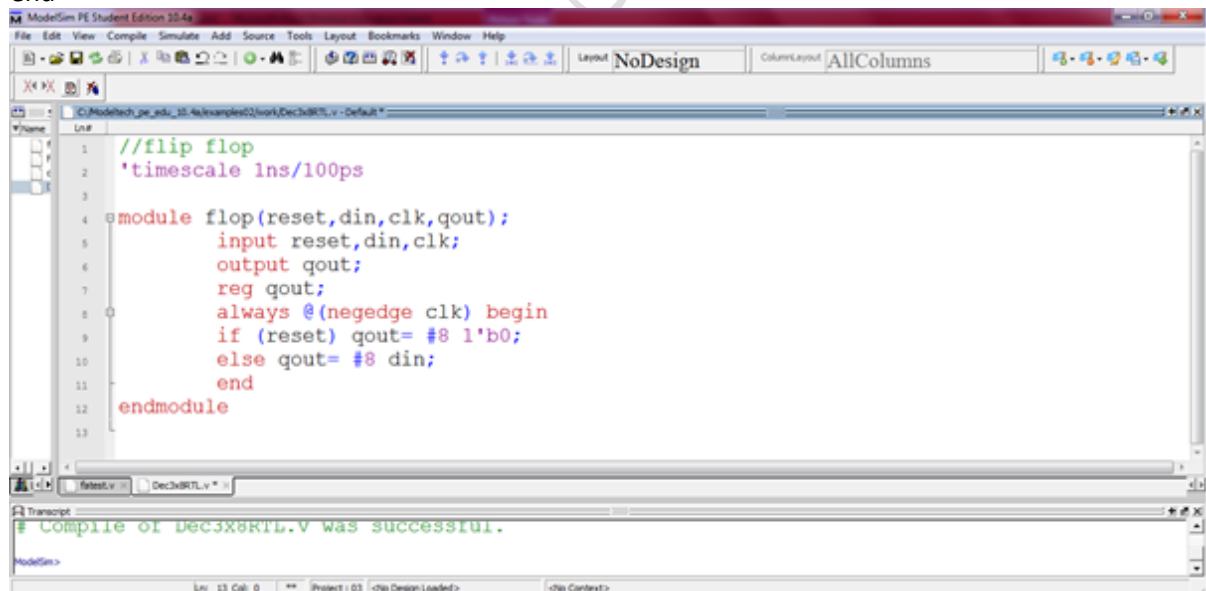
فلیپ فلاپ: فرض می‌کنیم ماژول فلیپ فلاپ هم بصورت زیر آماده است:



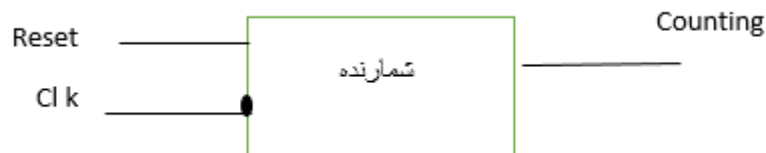
در ماژول ارائه شده reset, din, clk ورودی هستند و qout خروجی می‌باشد. در این ماژول reg qout برای تعریف qout بصورت ثبات می‌باشد. این نوع داده برای داده‌هایی استفاده می‌شوند که نقش ورودی و خروجی دارند یا عنصر حافظه هستند. قطعه کد زیر یک بلوک ایجاد می‌کند که حلقه ای دائمی (always) را ایجاد می‌کند. بدنه این حلقه به شرطی اجرا می‌شود که لبه پایین رونده clk اتفاق بیافتد. @(negedge clk). اگر بلوک شامل چند دستور باشد نیازمند begin-end می‌باشد.

```

always @(negedge clk)
begin
    بدنه
end
  
```



برای این مدار نیازمند یک شمارنده هستیم که تعداد بیت های ورودی را (۷-۰) بشمارد. این مدار ترتیبی است با ورودی reset برای صفر کردن آن و خروجی counting که در صورت مجاز بودن شمارش یک خواهد بود. یعنی هنوز ۸ بیت وارد نشده اند.



```

ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
Layout: NoDesign ColumnsLayout: AllColumns

//counter
timescale 1ns/100ps

module counter(reset,clk,counting);
    input reset,clk;
    output counting;
    reg counting;
    integer count;
    integer limit; initial limit = 8;
    always @(negedge clk) begin
        if(reset) count=0;
        else begin
            if(count<limit)count=count+1;
            end
            if(count==limit)#8 counting=0;
            else #8 counting=1;
            end
        end
    endmodule

```

با توجه به مطالب مطرح شده جمع کننده سریال در سطح رفتاری بصورت ماژول شکل زیر خواهد بود. در این ماژول عملکرد بخش های مدار با هم در سطح رفتاری مطرح شده است. بخش هایی از ماژول امکان تقسیم شدن دارند. مثلاً برای انجام شیفت به راست؛ `result={sum,result[7:1]}` استفاده شده است. عملکرد `{}` برای ادغام استفاده می شود.

```

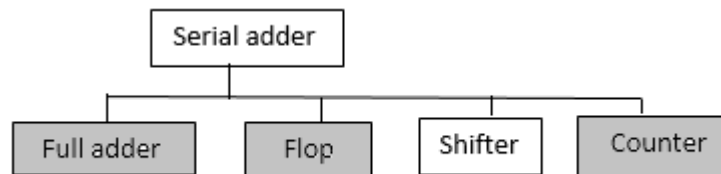
ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
Layout: NoDesign ColumnsLayout: AllColumns

//serial adder
timescale 1ns/100ps

module serial_adder(a,b,start,clk,ready,result);
    input a,b,start,clk;
    output ready;
    output [7:0] result;
    reg [7:0] result;
    reg sum,carry,ready;
    integer count;
    always@(negedge clk)begin
        if(start) begin
            count=0; carry=0;result=0;
        end else begin
            if(count<8) begin
                count = count+1;
                sum=a^b^carry;
                carry=(a&b)|(a&carry)|(b&carry);
                result={sum,result[7:1]};
            end end
            if(count==8)ready=1;
            else ready=0;
        end
    end
endmodule

```

با توجه به طراحی بالا به پایین قطعات مدار را بصورت زیر رسم می کنیم در این شکل shifter بدون هاشور است یعنی هنوز قابل تقسیم به قطعات کوچکتر است.



ماژول تمام جمع کننده بصورت زیر قابل نوشتن است:

```

1 //Full Adder
2 `timescale 1ns/100ps
3
4 module fulladder(a,b,cin,sum,cout);
5     input a,b,cin;
6     output sum,cout;
7     assign sum=a^b^cin;
8     assign cout=(a&b) | (a&cin) | (b&cin);
9 endmodule
10
  
```

ماژول شیفت دهنده را می‌توان بصورت شکل زیر در نظر گرفت. در این ماژول دستور استفاده شده برای مقدار دهی به خروجی (parout) بصورت $\text{parout} = (\text{reset}) ? 8'b0 : ((\text{enable}) ? \{\text{sin}, \text{parout}[7:1]\} : \text{parout})$ می‌باشد. این یک انتساب شرطی است و معادل عبارت زیر است:

```

if (reset==1) begin
    parout=8'b0; // parout= 8 bit 0 or 00000000;
else if (enable==0) parout={sin,parout[7:0]}
else parout=parout;
end
  
```

```

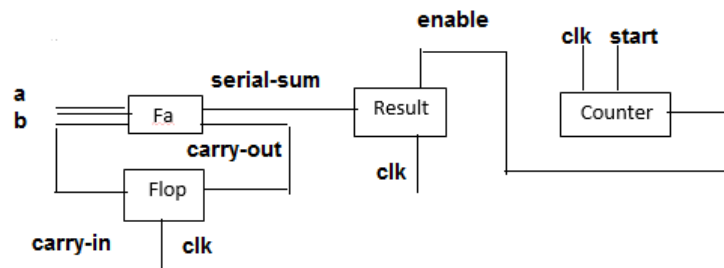
1 //shifter
2 `timescale 1ns/100ps
3
4 module shifter(sin,reset,enable,clk,parout);
5     input sin,reset,enable,clk;
6     output [7:0] parout;
7     reg [7:0] parout;
8     always@(negedge clk)
9         parout=(reset)?8'b0:((enable)?{sin,parout[7:1]}:parout);
10 endmodule
11
  
```

با توجه به اینکه جمع کننده سریال از ۴ قطعه تشکیل شده است می‌توان آن را بر اساس زیرقطعاتش بصورت زیر نوشت:

```

1 //serialadder
2 `timescale 1ns/100ps
3
4 module serial_adder(a,b,start,clk,ready,resault);
5     input a,b,start,clk;
6     output ready;
7     output [7:0] resault;
8     wire serial_sum,carry_in,carry_out,counting;
9     fulladder u1(a,b,carry_in,serial_sum,carry_out);
10    flop u2(start,carry_out,clk,carry_in);
11    counter u3(start,clk,counting);
12    shifter u4(serial_sum,start,counting,clk,resault);
13    assign ready =~ counting;
14 endmodule
15
  
```

شکل مدار تاکنون می تواند بصورت زیر طراحی شود:



خود فلیپ فلاپ استفاده شده در مدار عملکردی بصورت زیر دارد:

```

1 //flipflop D
2 'timescale 1ns/100ps
3
4 module der_flop(din,reset,enable,clk,qout);
5     input din,reset,enable,clk;
6     output qout;
7     reg qout;
8     always@(negedge clk)begin
9         if(reset) qout= #8 1'b0;
10        else
11            if(enable) qout= #8 din;
12        end
13    endmodule
14

```

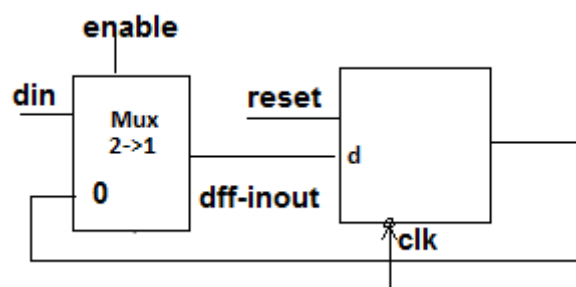
با توجه مطالب گفته شده می توان مدار شیفتر دهنده را بصورت زیر طراحی کرد:

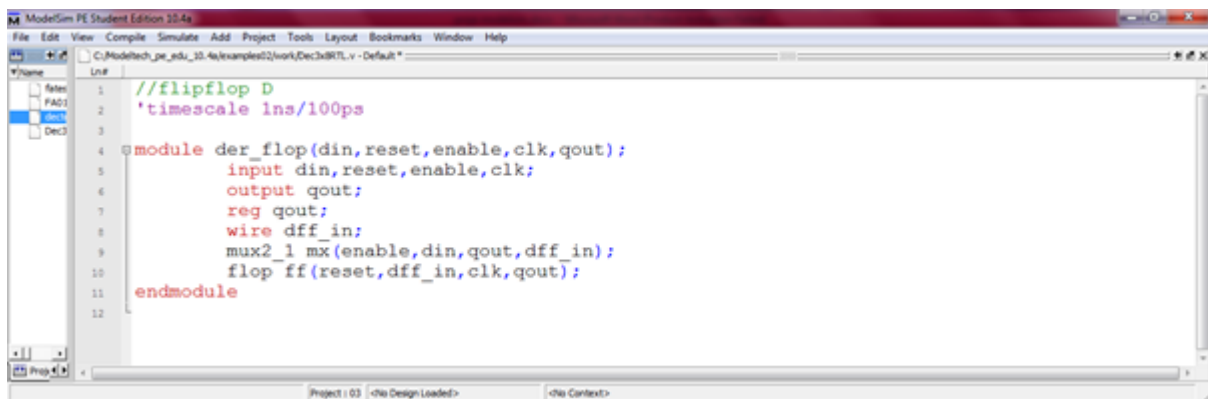
```

1 //SHIFTER
2 'timescale 1ns/100ps
3
4 module shifter(sin,reset,enable,clk,parout);
5     input sin,reset,enable,clk;
6     output [7:0] parout;
7     reg [7:0] parout;
8     der_flop b7(sin,reset,enable,clk,parout[7]);
9     der_flop b6(parout[7],reset,enable,clk,parout[6]);
10    der_flop b5(parout[6],reset,enable,clk,parout[5]);
11    der_flop b4(parout[5],reset,enable,clk,parout[4]);
12    der_flop b3(parout[4],reset,enable,clk,parout[3]);
13    der_flop b2(parout[3],reset,enable,clk,parout[2]);
14    der_flop b1(parout[2],reset,enable,clk,parout[1]);
15    der_flop b0(parout[1],reset,enable,clk,parout[0]);
16
17 endmodule

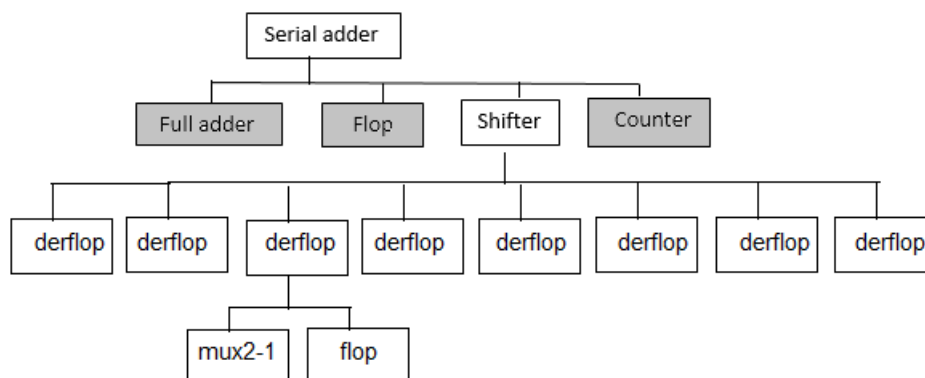
```

برای طراحی der_flop می توان از یک فلیپ فلاپ D و یک مالتی پلکسر بصورت زیر استفاده کرد:



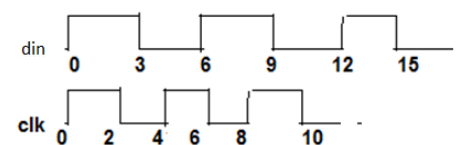
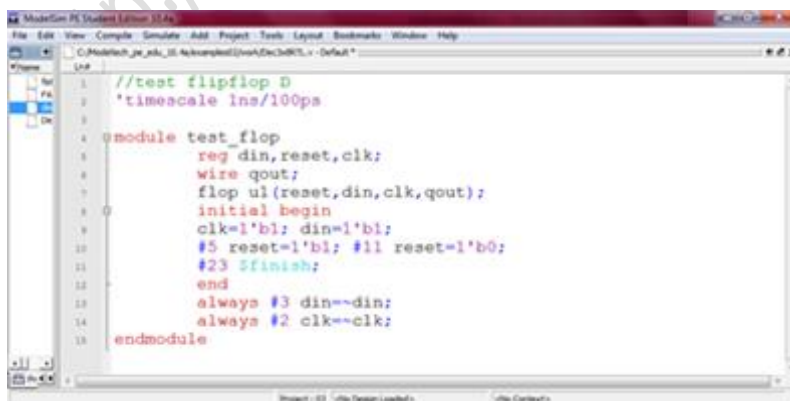


با توجه به مراحل طراحی بالا به پایین شکل نهایی مدار بصورت زیر است:



نوشتن ماژول تست: برای اینکه بتوانیم عملکرد هر مداری را بسنجیم باید برای آن تست بنویسیم. در ماژول تست باید ورودی ها را تغییر دهیم و نتایج خروجی ها را مشاهده کنیم. به عنوان مثال ماژول تست فلیپ فلاپ بصورت زیر است:

```
module flop(reset ,din , clk , qout );
    input reset , din , clk;
    output qout;
    reg qout;
    always @(negedge clk)
    begin
        if(reset) qout=# 8 1` b 0;
        else qout=# 8 din;
    end
endmodule;
```



فصل سوم عملگرهای وریلاگ

عملیات منطقی رایجترین نوع عملیات برای توصیف عملکرد اجزای سخت افزاری در سطح گیت می باشد. علاوه بر این عملیات، عملیات دیگری نیز برای توصیف عملیات سخت افزار در سطح رفتاری وجود دارند. اکثر عملیات پوشش داده شده در نرم افزارهای برنامه نویسی در وریلاگ نیز پشتیبانی می شوند. عملگرها را براساس عملکردشان دسته بندی می کنیم. ابتدا عملگرهای پایه و رابطه ای را لیست می کنیم.

نتیجه	توصیف	عمل	عملیات پایه
چند بیت	چهار عمل اصلی ریاضی	$+, -, *, /$	ریاضی
یک بیت	مقایسه	$<, >, =, >=, <=$	رابطه ای

عملگرهای ریاضی روی همه انواع داده ها کار می کنند و معنی رایجشان را دارند. عملگرهای رابطه ای و تساوی مقادیر ۰ و ۱ را به عنوان نتیجه به معنی درست و غلط بر میگردانند.

نتیجه	توصیف	عمل	برابری
یک بیت	شامل X, Z نیستند (برابری و نابرابری)	$==, !=$	منطقی
یک بیت	شامل X, Z (برابری و نابرابری)	$===, !==$	حالت

سه گروه عملگرهای منطقی شامل عملیات منطقی برای بردارها، اعداد اسکالر و کاهش بردار به عدد اسکالر می باشند.

نتیجه	توصیف	عمل	منطقی
یک بیت	منطقی ساده	$\&\&, \ \ , !$	اتصال دهنده
چند بیت	عملیات منطقی برداری	$\sim, \wedge, \vee, \&$	بیتی
یک بیت	انجام عملیات روی همه بیت ها	$\sim, \wedge, \vee, \&, \sim\&, \ \sim, \wedge, \sim\wedge$	کاهش

در عملگرهای شیفت عدد سمت راست مشخص کننده تعداد شیفت ها می باشد.

نتیجه	توصیف	عمل	شیفت
چند بیت	از سمت راست صفر وارد می شود n بیت به راست شیفت می کند	$<<n$	چپ
چند بیت	از سمت چپ صفر وارد می شود n بیت به چپ شیفت می کند	$>>n$	راست

سایر عملیات در جدول زیر لیست شده است.

نتیجه	توصیف	عمل	سایر
چند بیت	محاسبه باقی مانده تقسیم	$\%$	ماژول
چند بیت	اتصال بیت	$\{\}$	الحاق
چند بیت	اتصال و تکرار	$\{\{\}\}$	تکرار
چند بیت	If then else	شرط	؟

همانطور که قبلاً گفتیم وریلاگ حساس به حروف نیست و عبارات طولانی می توانند در چند سطر نوشته شوند. برای نوشتن توضیحات از $//$ استفاده می کنیم و برای ایجاد چندین سطر توضیح آنها را بین $/*$ و $*/$ قرار می دهیم.

مفاهیم پایه در وریلاگ

تفاوت زبان های برنامه نویسی نرم افزار و زبان های برنامه نویسی سخت افزار چیست؟

- در زبان های نرم افزاری به صورت ترتیبی (sequential) فکر کنیم. یعنی یک دستور به شرطی اجرا می شود که دستورات قبل از آن تمام شده باشد. اما در سخت افزار (به ویژه درکد ساختاری) به صورت همروند^۱ فکر می کنیم. یعنی دستورات با هم اجرا می شوند. (اجرای همزمان اجزای سخت افزاری مختلف).
- تفاوت دوم این است که در نرم افزار برای اجرای دستورات زمان^۲ قرار نمی دهیم ولی در سخت افزار از زمان استفاده می کنیم. زیرا مقادیر بین اجزای سخت افزاری از طریق سیم ها و گذرگاه ها عبور می کنند و سیم ها به دلیل وجود اثرات خازنی و مقاومتی تاخیرهای متفاوتی دارند. البته همیشه در مقدار دهی وریلاگ از توصیف زمانی استفاده نمی شود.

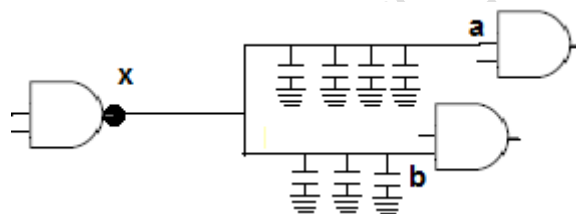
A:=x

B:=x

در نرم افزار

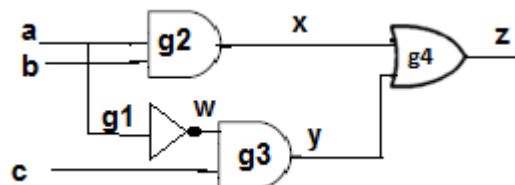
سخت افزار همروند

```
assign #(4* unit-dalay) a=x;
assign #(۳* unit-dalay) b=x;
```



همانطور که در شکل مشاهده می شود x بطور همروند به دو ورودی a, b منتقل می شود ولی به توجه به اینکه فاصله سیمی آنها برابر نیست اثرات خازنی ورودی a بیشتر است. بنابراین x یک واحد زمانی تاخیر دیرتر به a می رسد. هرگاه در کد وریلاگ در سطح ساختاری کد بنویسیم به صورت همروند کار می کند اما در سطح رفتاری به صورت ترتیبی کار می کند.

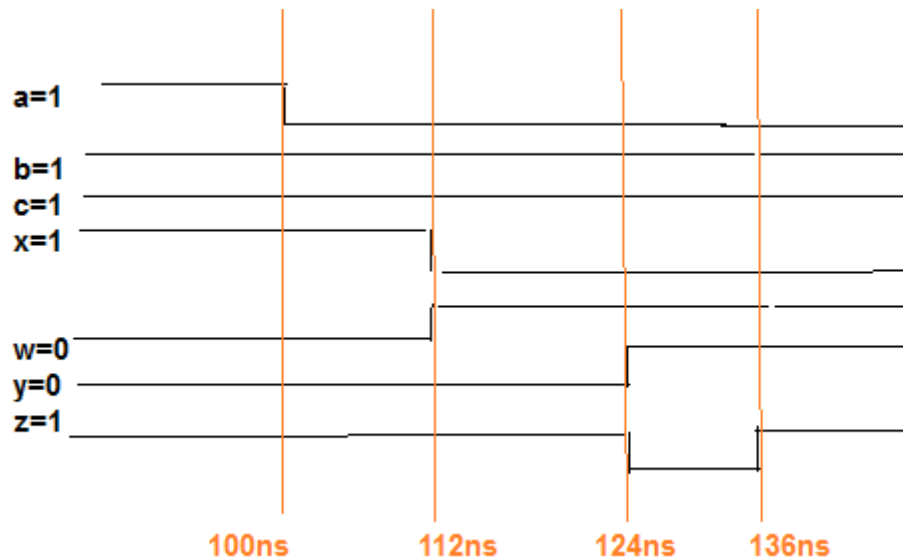
مثال (مدار زیر را در نظر بگیرید و همروندی سیگنال ها را مشخص کنید. فرض کنید هر گیت تاخیر 12ns دارد. سپس کد وریلاگ بنویسید که عملکرد مدار را نشان می دهد.



فرض کنید در ابتدا a, b, c یک باشند و در زمان 100ns مقدار a=0 شود. نمودار مربوطه را رسم کنید.

¹ concurrent

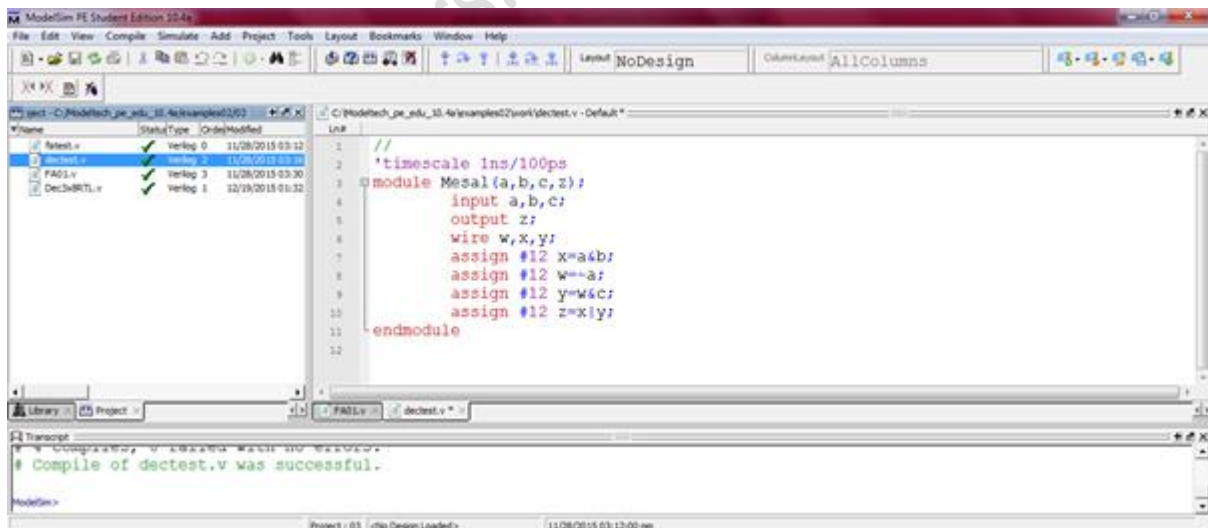
² Timing



با تغییر a در زمان 100ns به صفر ورودی های $g1$ و $g2$ تغییر میکند و بعد از 12ns خروجی این گیت ها یعنی x و y تغییر میکند (زمان 112ns). یعنی $g1$ و $g2$ همروند هستند و در زمان 112ns x و y تغییر کرده اند. بنابراین ورودی گیت های $g3$ و $g4$ تغییر کرده اند و در زمان 124ns خروجی y و z تغییر کرده اند و $g3$ و $g4$ همروند می باشند. چون در زمان 124ns تغییر کرده اند. ورودی $g4$ تغییر داشته است. خروجی z در زمان 136ns تغییر می کند. بنابراین یک پالس هزارد (ایستای سطح صفر) در خروجی z داریم.

$$\left. \begin{array}{l} z=1, abc=111 \\ z=1, abc=011 \end{array} \right\} z=ab+dc \text{ بدون زمان}$$

برنامه وریلاگ معادل مدار بصورت زیر است:



تعریف متغیرها در Verilog: قوانین تعریف متغیرها در وریلاگ مشابه زبان های دیگر است به اضافه موارد زیر:

- ۱- اولین کاراکتر حرف می باشد.
- ۲- علامت \$ قبل از یک شناسه به عنوان فراخوانی کارهای سیستمی و توابع می باشد.
- ۳- علامت ` برای راهنماهای کامپایلری است.

`timescale 1ns/100 ps

۴- comment به صورت زیر است: یک خط توضیح //

/* توضیحات */

اعداد : مقادیر منطقی در وریلاگ دارای چهار مقدار هستند. x,z,0,1

مقدار دهی اعداد با حروف b باینری- d مبنای ۱۰ - o مبنای ۸ - h مبنای ۱۶ می باشد و ساختار کلی مقداردهی بشکل زیر است.

مقدار یکی از حروف فوق ' تعداد بیت ها

یعنی متغیر ۴ بیتی است با مقدار ۵=۱۰۱۰۱

X=4'd 5 => 00000101

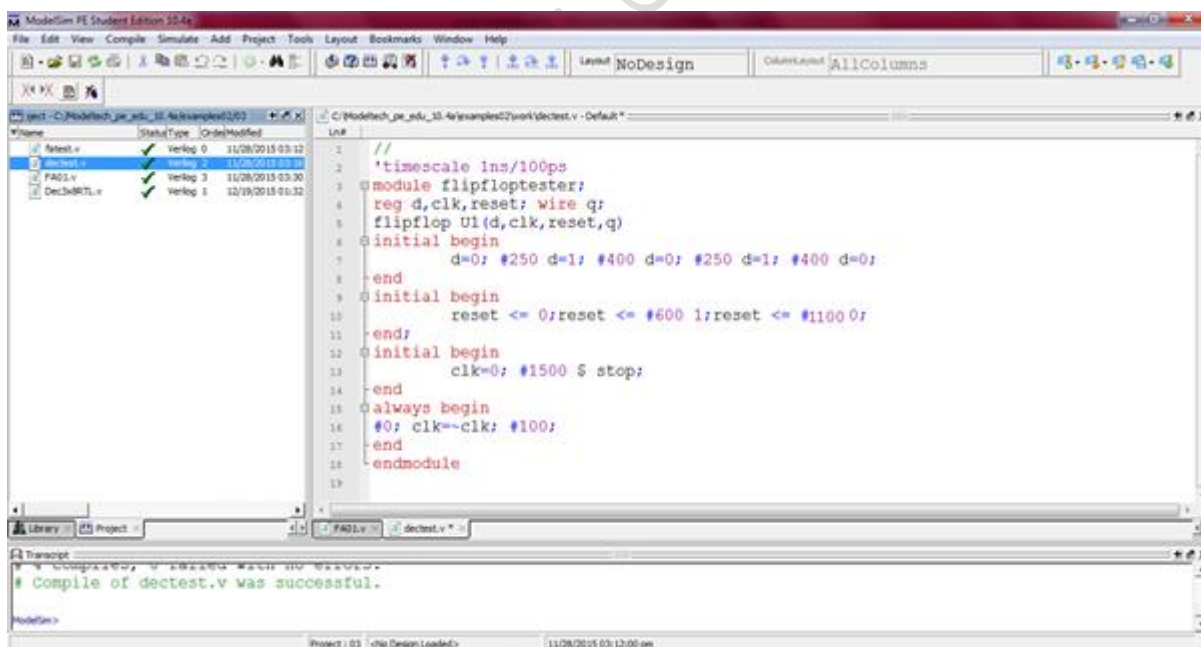
X=-8'b 101 => 11111011

X=10'o 752 => 0 111 101 010

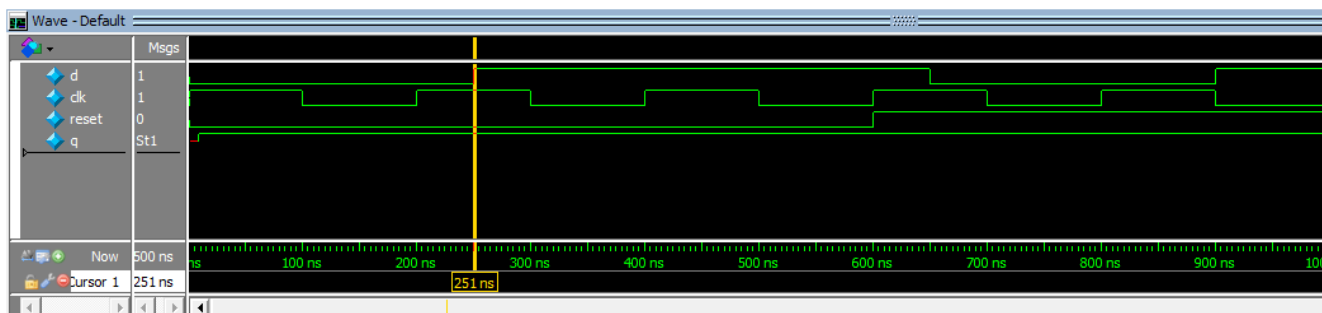
X=8'h f => 00001111

X=12'h xf => xxxxxxxx1111

مثال: همانطور که گفتیم برای چک کردن عملکرد یک مدار باید برای آن تست نوشت با توجه به مطالب گفته شده یک نمونه تست دیگر برای برای فلیپ فلاپ بصورت زیر است:



وقتی در یک ماژول تست چند بلوک initial و always نوشته می شوند بصورت همروند اجرا می شوند و داخل آنها وابسته به نوع نوشتار است. اگر در مقدار دهی از عملگر = استفاده شود مقدار دهی ترتیبی است و اگر از عملگر <= استفاده شود مقدار دهی همروند است. بنابراین موج ایجاد شده روی d, clk, reset بصورت زیر خواهد بود:

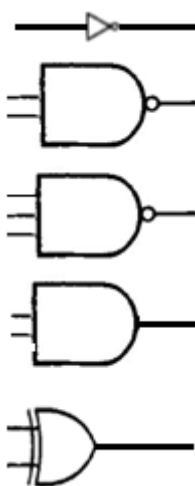


خروجی که مشاهده می کنید نتیجه شبیه سازی تست در نرم افزار modelsim می باشد. همانطور که مشاهده می شود کلاک در زمان ۰ یک می شود و هر ۱۰۰ نانو ثانیه ممتد می شود.

فصل چهارم توصیف ساختاری سخت افزار

نکات سطح ساختاری:

۱. دستورات Verilog به صورت همروند اجرا می‌شوند.
۲. سیگنال‌های زمانی در طراحی مدار استفاده می‌شود.
۳. ساختارهای آماده‌ی Verilog در این سطح شامل گیت‌های (and, nand, or, xor, xnor, not, buf) می‌باشد.



not #4 (z,a)

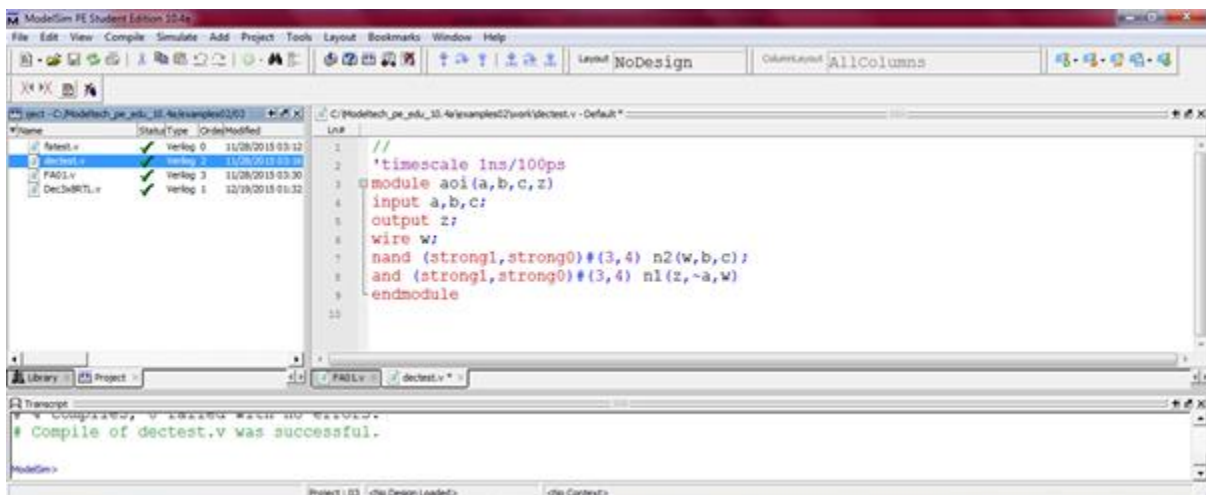
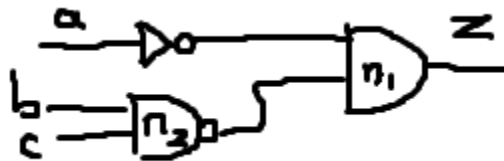
nand #5(z,a,b)

nand #6(z,a,b,c)

and #5(z,a,b)

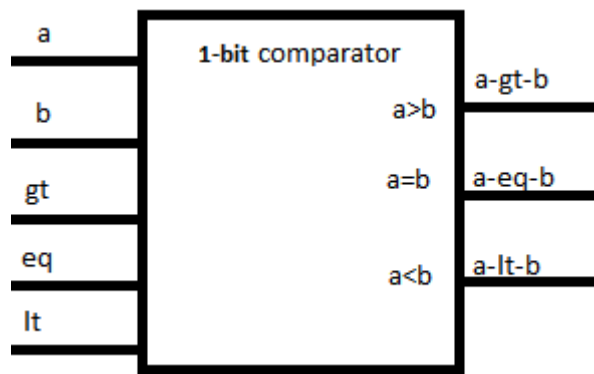
xor #5(z,a,b)

مثال: توصیف ساختاری برای محاسبه‌ی عبارت $z = a' \cdot (bc)'$ را بنویسید؟



نکته: به جای strong میتوان highz0, weak0, pull0, supply0 را نیز نوشت.

مثال: یک مقایسه‌گر تک بیتی طراحی کنید که قابل توسعه باشد؟



$$a\text{-gt-}b = ab' + ab.gt + a'b'.gt = a.gt + ab' + b'.gt$$

$$a\text{-eq-}b = a'b'.eq + ab.eq$$

$$a\text{-lt-}b = a'b + a'b'.lt + ab.lt = a'b + b.lt + a'.lt$$

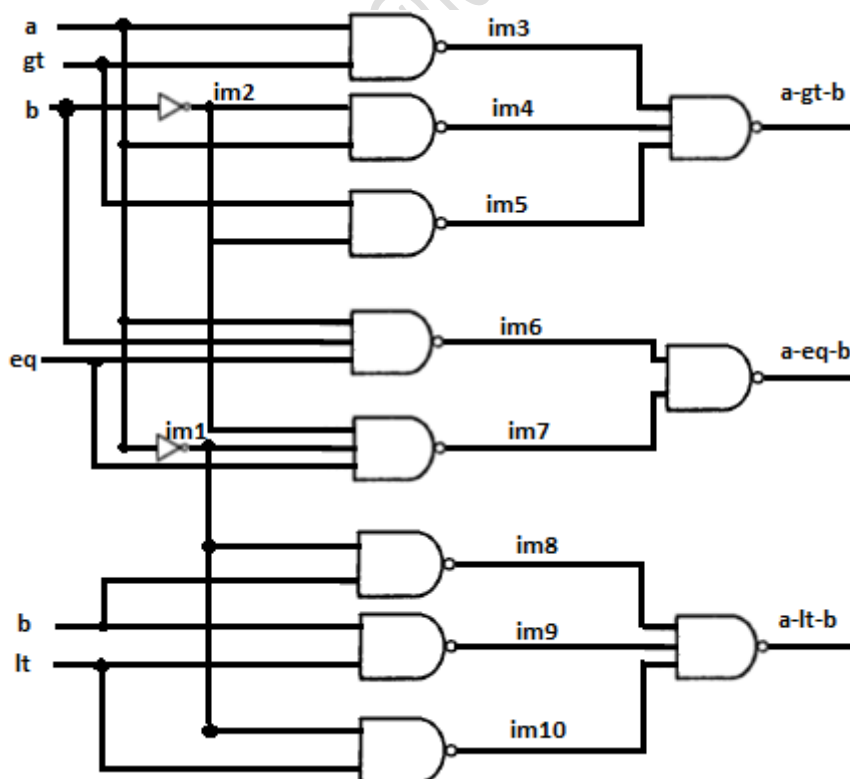
تبدیل عبارات به NAND:

$$a_gt_b = a.gt + a.b' + b'.gt = ((a.gt)'.(a.b')'.(b'.gt)')'$$

$$e_eq_b = a.b.eq + a'b'.eq = ((a.b.eq)'.(a'b'.eq)')'$$

$$a_lt_b = a'b + b.lt + a'.lt = ((a'b)'.(b.lt)'.(a'.lt)')'$$

شکل مدار بصورت زیر است:



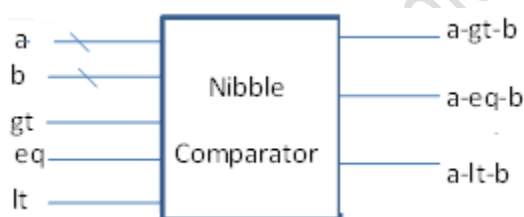
با توجه به شکل مدار کد وریلاگ ساختاری آن بصورت زیر است:

```

ModelSim ALTERA STARTER EDITION 10.1b
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
C:/altera/12.1/modelsim_ase/bit_comparator.v (/test_nibble_comparator/u1/c3) - Default
Ln#
1 `timescale 1ns/1ns
2 module bit_comparator(a,b,gt,eq,lt,a_gt_b,a_eq_b,a_lt_b);
3 input a,b,gt,eq,lt;
4 output a_gt_b,a_eq_b,a_lt_b;
5 wire im[1:10];
6 not #(4) g0(im[1],a);
7 not #(4) g1(im[2],b);
8 nand #(5) g2(im[3],a,gt);
9 nand #(5) g3(im[4],a,im[2]);
10 nand #(5) g4(im[5],im[2],gt);
11 nand #(6) g5(a_gt_b,im[3],im[4],im[5]);
12 nand #(6) g6(im[6],a,b,eq);
13 nand #(6) g7(im[7],eq,im[2],im[1]);
14 nand #(5) g8(a_eq_b,im[6],im[7]);
15 nand #(5) g9(im[8],im[1],b);
16 nand #(5) g10(im[9],b,lt);
17 nand #(5) g11(im[10],im[1],lt);
18 nand #(6) g12(a_lt_b,im[8],im[9],im[10]);
19 endmodule

```

استفاده از bit comparator و طراحی یک مقایسه گر ۴ بیتی:



```

ModelSim ALTERA STARTER EDITION 10.1b
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
C:/altera/12.1/modelsim_ase/nibble_comparator.v (/test_nibble_comparator/u1) - Default
Ln#
1 module nibble_comparator(a,b,gt,eq,lt,a_gt_b,a_eq_b,a_lt_b);
2 input[3:0] a,b;
3 input gt,eq,lt;
4 output a_gt_b,a_eq_b,a_lt_b;
5 wire im[0:8];
6 bit_comparator c0(a[0],b[0],gt,eq,lt,im[0],im[1],im[2]);
7 bit_comparator c1(a[1],b[1],im[0],im[1],im[2],im[3],im[4],im[5]);
8 bit_comparator c2(a[2],b[2],im[3],im[4],im[5],im[6],im[7],im[8]);
9 bit_comparator c3(a[3],b[3],im[6],im[7],im[8],a_gt_b,a_eq_b,a_lt_b);
10 endmodule

```

- برای ساخت یک مقایسه گر ۴ بیتی می‌بایست چهار بار یک مقایسه گر تک بیتی را فراخوانی نمود.
- برای ساخت مقایسه گر ۸ بیتی نیز می‌توان یا ۸ بار مقایسه گر تک بیتی را فراخوانی کرد و یا دو بار مقایسه گر ۴ بیتی را فراخوانی کرد.
- Wire را می‌توان در قالب آرایه نیز تعریف کرد.
- نحوه ساخت و کار این مدار موازیست.

نوشتن تست برای مقایسه گر ۴ بیتی:

```

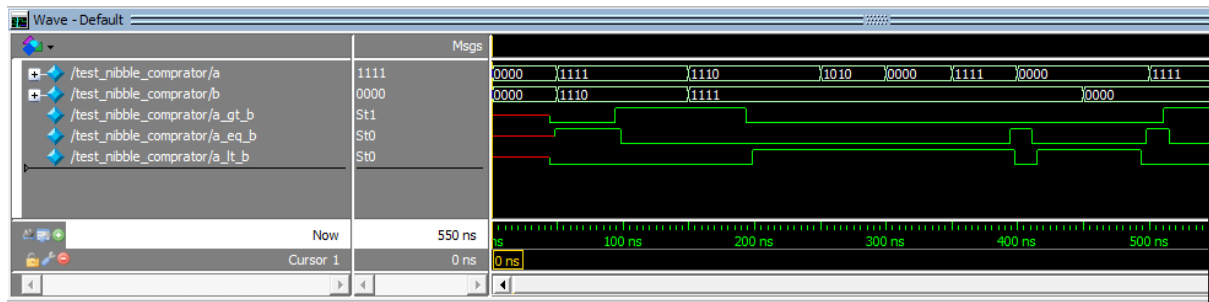
ModelSim ALTERA STARTER EDITION 10.1b
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
C:/altera/12.1/modelsim_ase/test_nibble_comprator.v (/test_nibble_comprator) - Default

Ln#
1  `timescale 1ns/1ns
2  module test_nibble_comprator;
3  reg[3:0] a, b;
4  supply0 gt,lt;
5  supply1 eq;
6  wire a_gt_b,a_eq_b,a_lt_b;
7  nibble_comprator u1(a,b,gt,eq,lt,a_gt_b,a_eq_b,a_lt_b);
8  initial
9  begin
10 a= 4 'd 0;b=4 'd 0;
11 #50 a=4'd15; b=4'd14;
12 #50; #50 a=4'd14;b=4'd15;
13 #50; #50 a=4'd10;
14 #50 a=4'd0; b=4'd15;
15 #50 a=4'd15; #50 a=4'd0;
16 #50 b=4'd0; #50 a=4'd15;
17 #50 $finish;
18 end
19 endmodule
  
```

مقادیر اختصاص داده شده به a,b بصورت جدول زیر می باشد.

Time -ns	a	B
0	0	0
50	15	14
100	15	14
150	14	15
200	14	15
250	10	15
300	0	15
350	15	1
400	0	15
450	0	0
500	15	0
550	end	End

خروجی سیگنال ها تا زمان ۵۵۰ نانو ثانیه بصورت نمودار زیر است:

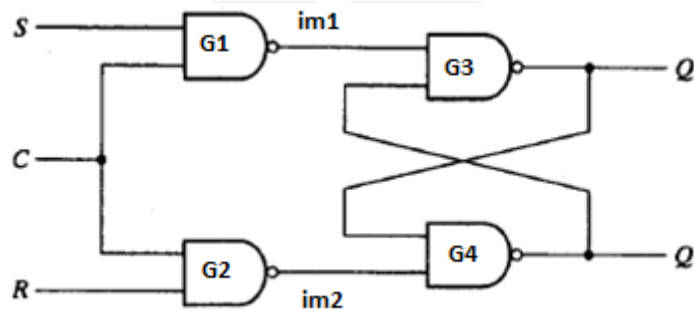


تمرین ۱:

۱. توصیف ساختاری یک فول ادر را با استفاده از گیت‌های ارایه شده بنویسید.
۲. توصیف ساختاری یک جمع کننده ۸ بیتی را با استفاده از فول ادر طراحی شده بنویسید.
۳. یک جمع کننده ۸ بیتی با استفاده از گیت‌های XOR و جمع کننده ۸ بیتی طراحی کنید.

طراحی یک فلیپ فلاپ در سطح ساختاری، طراحی یک مدار ترتیبی در سطح ساختاری:

***طراحی فلیپ فلاپ: S_r



ماژول:

```

ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
C:\Modeltech_pe_edu_10_4a\examples2\work\dectest.v - Default
//sr latch
timescale 1ns/100ps
module sr_latch(s,r,c,q)
input s,r,c;
output q;
wire im1,im2,im3;
nand #(5)
g1(im1,s,c);
g2(im2,c,r);
g3(q,im1,im3);
g4(im3,q,im2);
endmodule

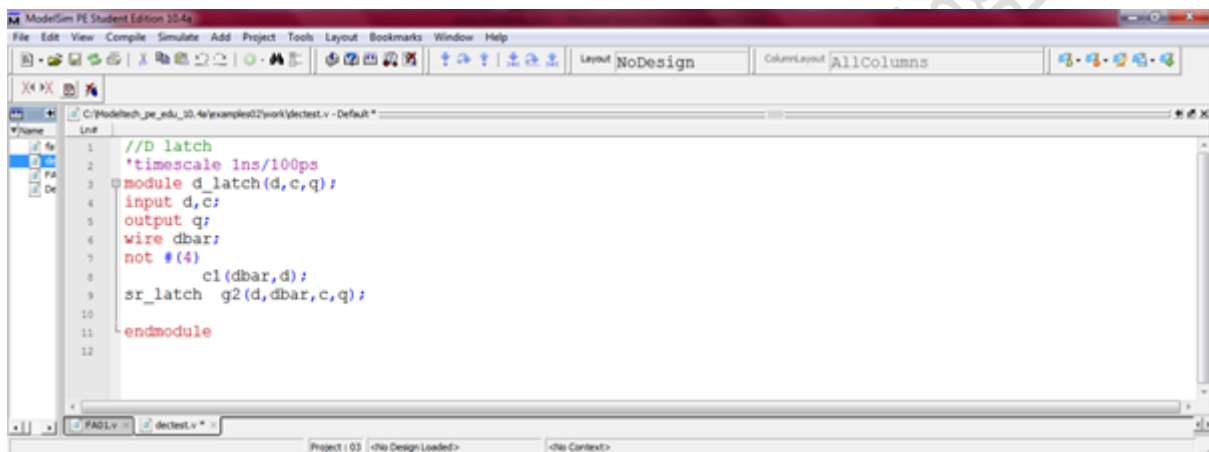
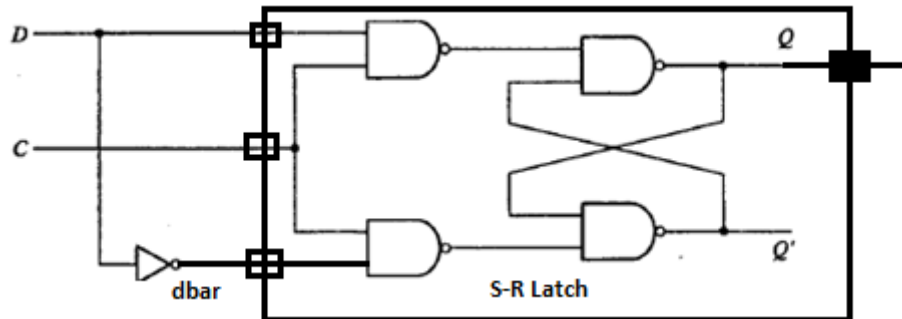
```

کلاک سطح بالا

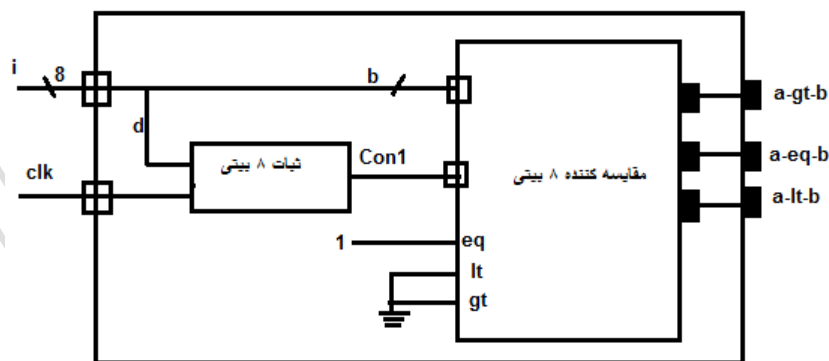
***طراحی فلیپ فلاپ d با استفاده از sr:

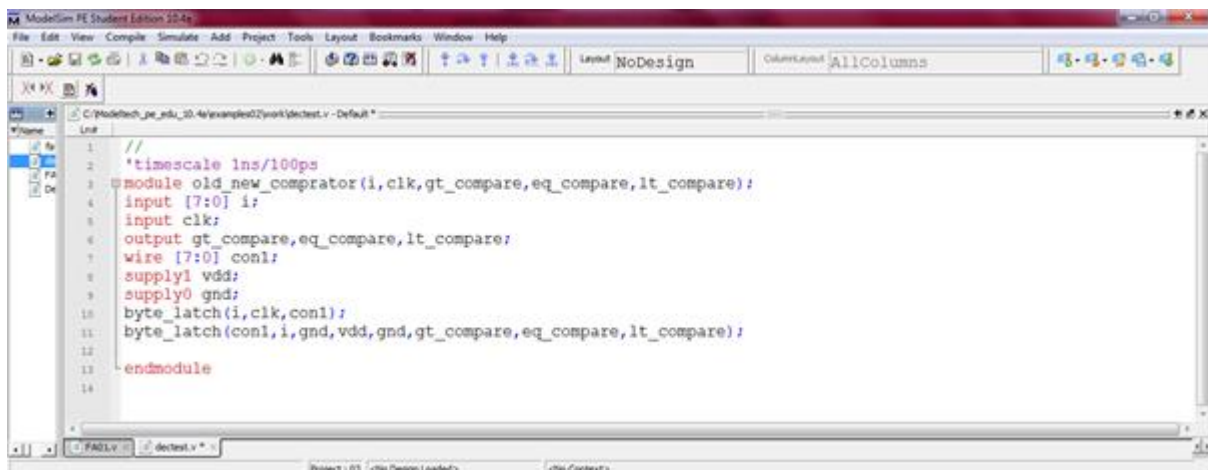
برای رسم فلیپ فلاپ d کافیست که ورودی d را به s و dbar را به r در فلیپ فلاپ sr بدهیم.

نکته: در کتاب وریلاگ ورودی را با □ و خروجی را با ■ نمایش میدهند.

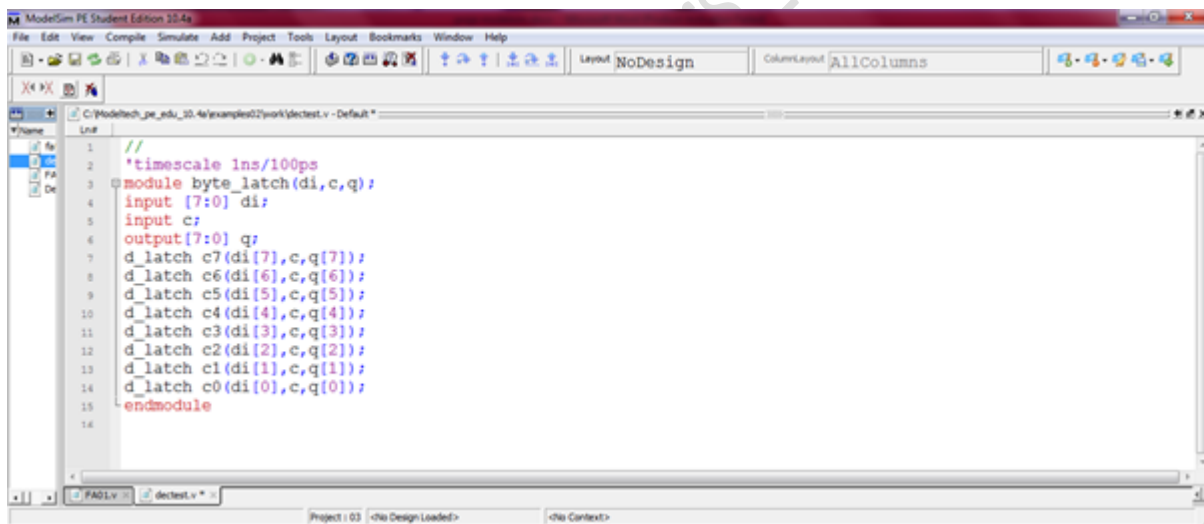
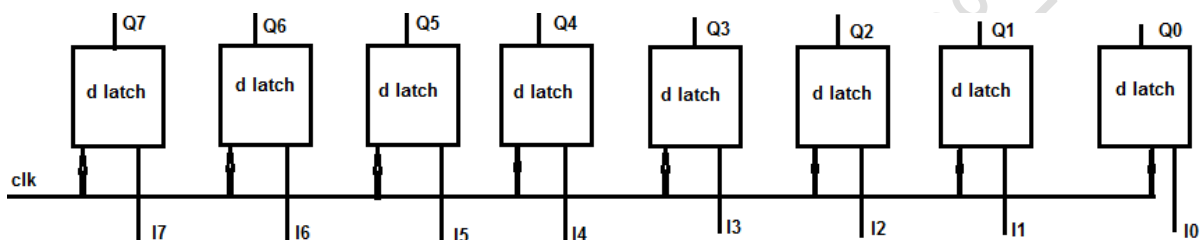


مدار مقایسه کننده سریال با استفاده از مقایسه کننده هشت بیتی با بار موازی:



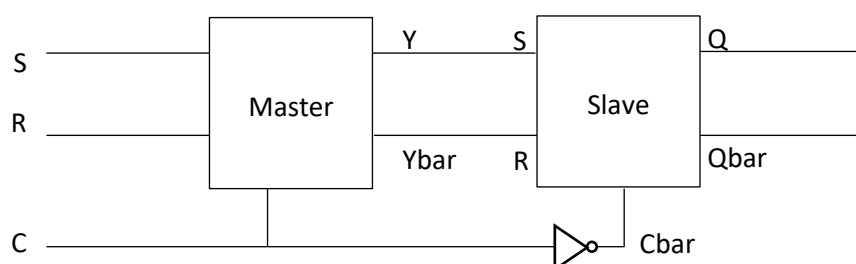


*****طراحی بایت لچ: ۸ بار فراخوانی d-latch که تک بیتی است:

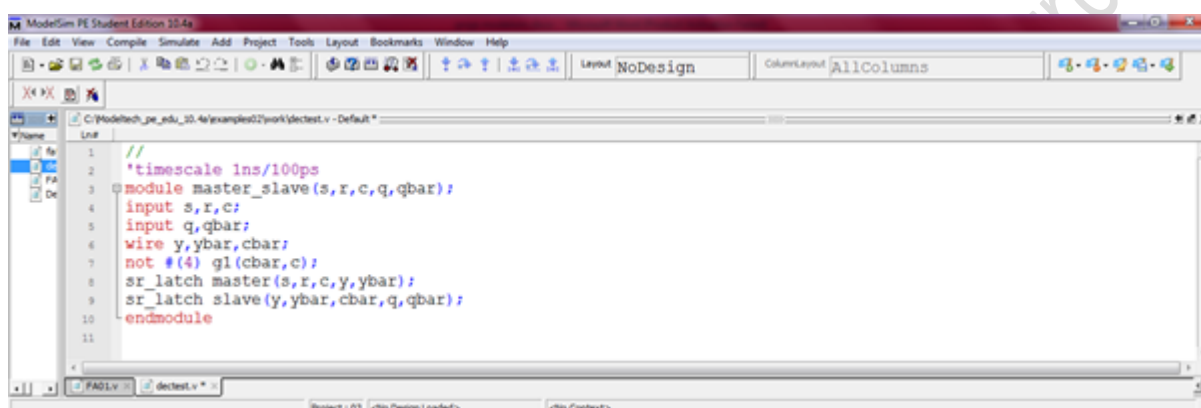
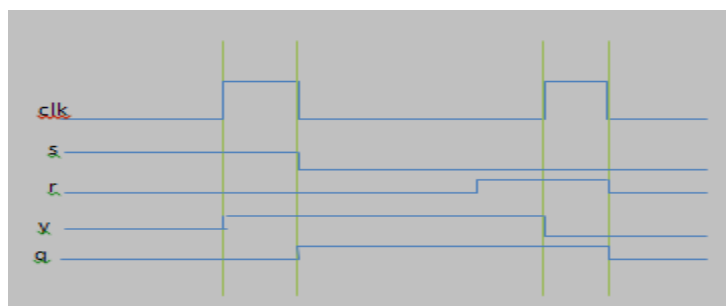


استفاده از فلیپ فلاپ master-slave برای طراحی کلاک لبه پایین رونده و بالا رونده:

اگر دو تا فلیپ فلاپ sr را کنار هم بگذاریم و به یکی 'clk' را وصل کنیم و به دیگری 'clk' در اینصورت کلاک با لبه ایجاد می شود اگر به فلیپ فلاپ master کلاک و به 'slave', 'clk' وصل شود لبه پایین رونده ایجاد می شود و اگر به 'slave', 'clk' و به 'master', 'clk' متصل شود لبه بالارونده ایجاد می شود.



نمودار زمانی:



فصل ۵: سازماندهی طراحی

تعریف تابع^۱

Function اسم تابع

تعریف ورودی‌ها

Begin

بدنه تابع

خروجی=اسم تابع

End

End function

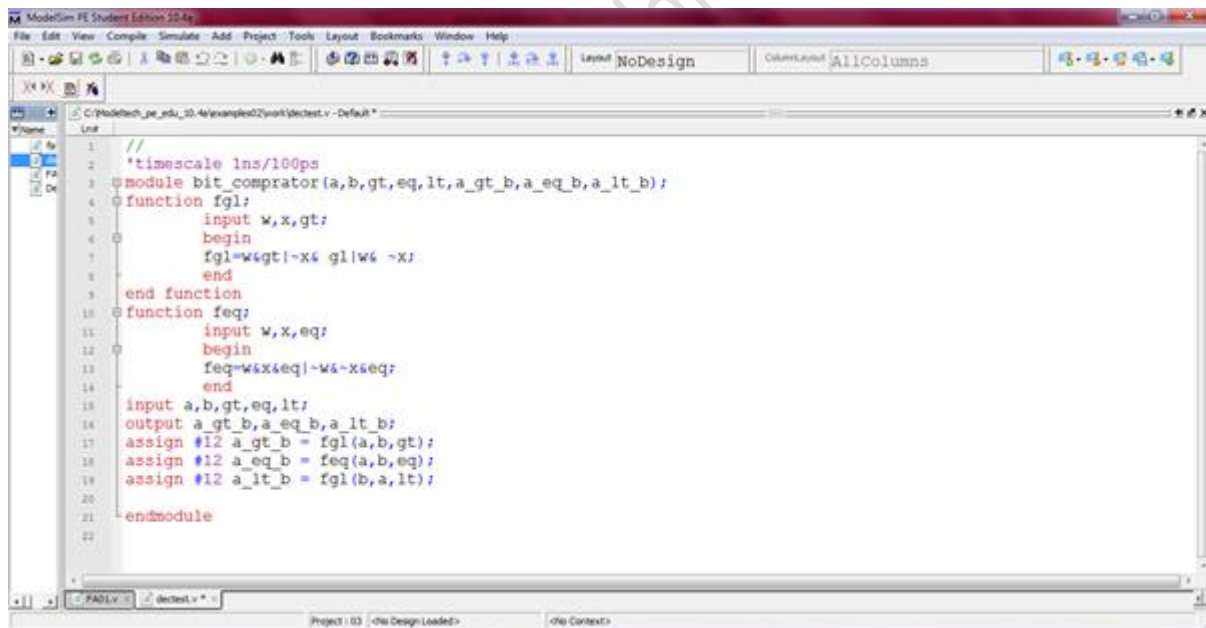
مثال: طراحی مقایسه گر تک بیتی با تابع:

$a_gt_b = a \text{ gt } a' + b' \text{ gt } b$

$e_eq_b = a.b.eq + a'b'eq$

$a_lt_b = a'b + b.lt + a'.lt$

برای طراحی مقایسه گر دو تابع با نام های fgl(greater-less) و feq (equal) تعریف می کنیم و سپس آنها را برای محاسبه سه خروجی صدا میزنیم.



- $a_gt_b = fgl(a,b,gt)$ برای محاسبه بزرگتر بودن a از b می باشد و با جابجایی a, b و صدا زدن آن کوچکتر از b بدست می آید.
- تاخیر تابع صفر می باشد. به همین خاطر تابع را با تاخیر زمانی ۱۲ واحد فراخوانی کرده ایم.

¹ function

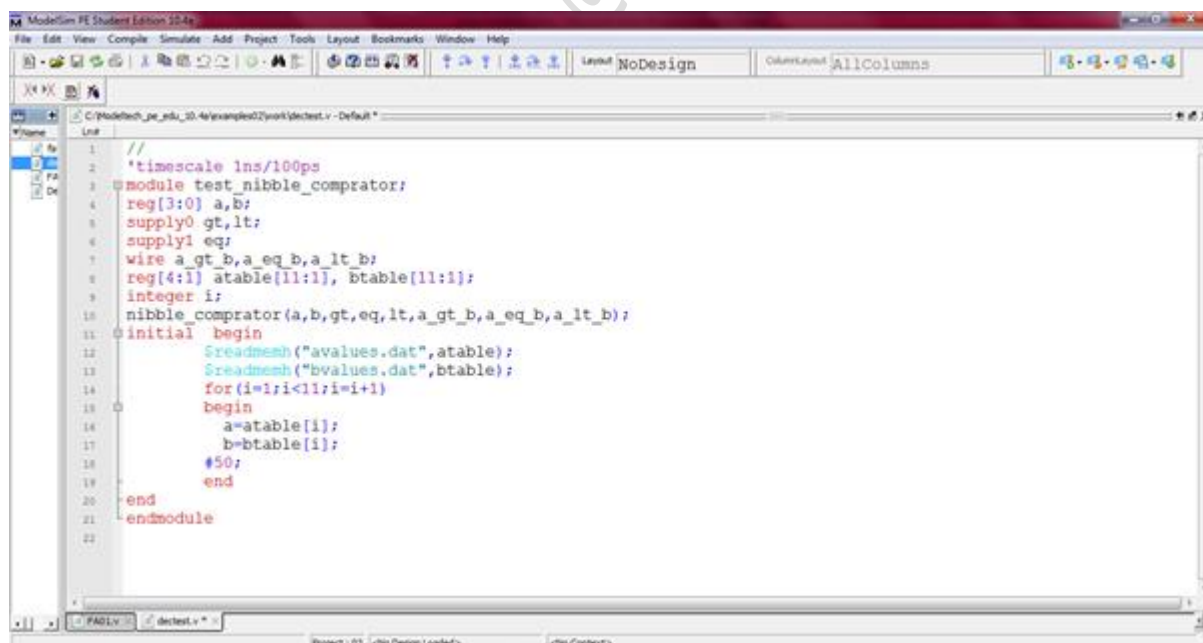
Task: دو نوع task داریم:

۱- سیستمی که task های موجود سیستمی را فراخوانی می کنیم.

۲- کاربر: که توسط برنامه نویس تولید و فراخوانی می شود.

روش دیگر نوشتن تست **nibble-comparator**: یک فایل (notepad) برای مقادیر a ایجاد و نام avalues.dat را می گذاریم. همچنین یک فایل برای مقادیر b ایجاد و نام bvalues.dat را می گذاریم. حالا یک تست می نویسیم که این دو فایل را صدا بزند.

a	b
0	0
F	E
F	E
E	F
E	F
A	F
0	F
F	F
0	f
0	0
f	0



برای ذخیره فایل ها می توان از پسوند txt. نیز استفاده کرد.

- لازم نیست که آرایه های table را داخل کد دقیقا هم اندازه فایل های اصلی تعریف کنیم، می توان یک فایل values هزارتایی داشت ولی فقط ۱۰ تای آنرا در قالب table استفاده کنیم.

- Readmemh(file name,array name,start index,end index)
- Index ها در readmemh اختیاری اند، اگر گذاشته نشوند کل فایل values خوانده می شود.

ساختار task:

اسم کار Task;

تعریف ورودی و خروجی Task

Begin:name

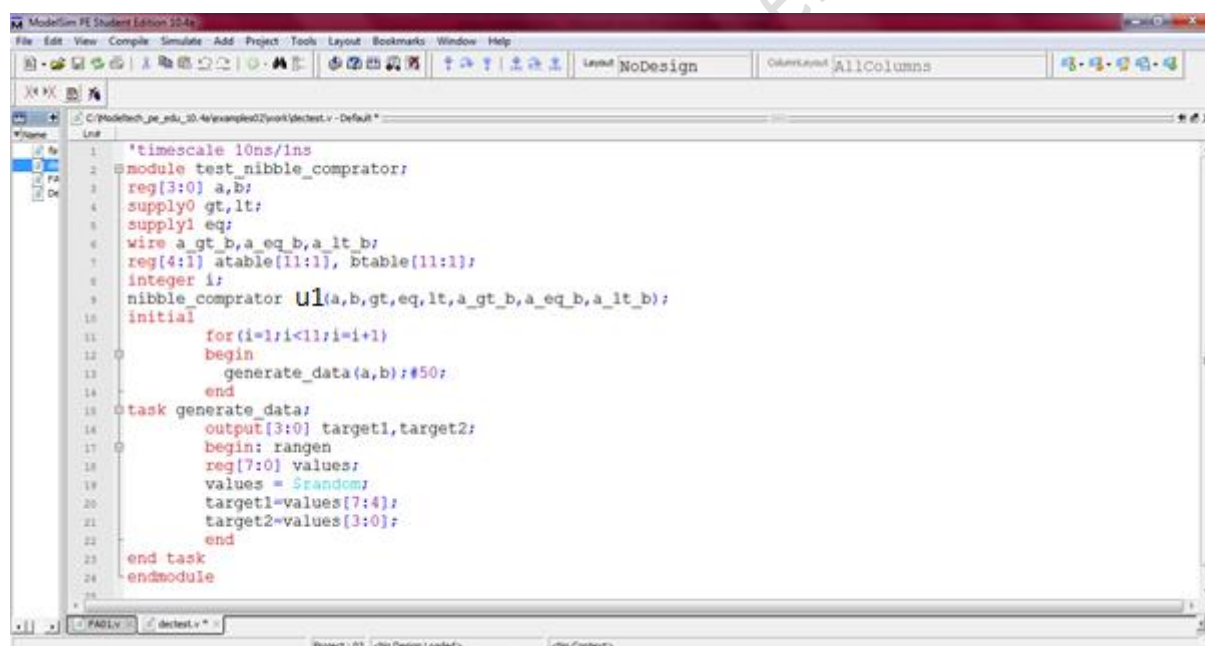
task بدنه

End

End task

مثال: ایجاد یک تست برای nibble-comparator به طوری که اعداد تصادفی را خود سیستم برای ما تولید کند.

تولید ۱۱ عدد تصادفی با task تعریف شده توسط کاربر:



- می توان task را در فایلی به اسم f1.v بنویسیم و سپس در بالای initial با کمک module f1.v فراخوانی اش کنیم.
- می توانستیم به جای تعریف متغیر values یکجا به target ها با کمک random مقدار دهیم.

تمرین ۲:

۱. یک تابع برای ایجاد نقلی خروجی یک تمام جمع کننده (FA) بنویسید.
۲. یک تابع برای ایجاد sum خروجی یک تمام جمع کننده بنویسید.

۳. با استفاده از عبارات بولی، یک تابع به نام inc-bit را بنویسید که ورودی آن ۴ بیتی و به مقدار ورودی یک واحد افزایش دهد.

فصل ۶ تعریف انواع

انواع پورتها می تواند input, output, inout باشد. برای تعریف پورت های ورودی و خروجی از reg نیز می توان استفاده کرد. مقدار نوع reg را می توان به inout انتساب داد.

input a1;

Input a2;

A1=a2;

تعریف پارامتر:

Parameter delay 5;

برای تعریف زمان، توان و اندازه می توان استفاده کرد.

تبدیل نوع: اگر یک متغیر integer از real مقدار بگیرد به نزدیکترین عدد صحیح گرد می شود. نوع reg را نیز می توان در real قرار داد ولی در real مقادیر X,Z نداریم، به جای آنها صفر می گذاریم.

Real a1;

Rega2='x';

A1=a2;

A1='0';

نوع دیگر: signal, wire, wor, برای تعریف سیم و متغیرهای میانی

تعریف بردار (آرایه):

Wor[7:0] abus,bbus;

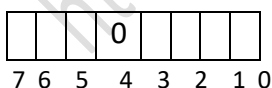
Reg[15,0] breg;

سمت چپترین بیت (با

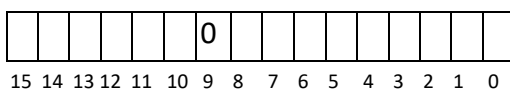
کم ارزشترین

مقدار دهی: طبق ادامه مثال بردار:

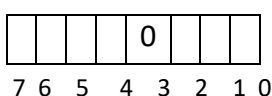
Abus[4]=1'b0



Bbus[3]=breg[9];



breg



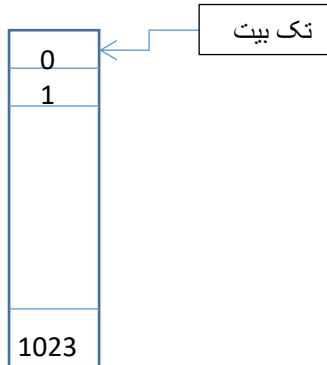
bbus

اگر از محدوده خارج شویم X برمیگردد.

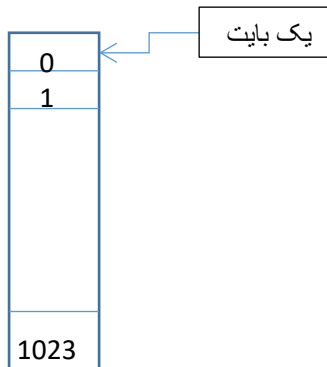
حافظه:

Reg[word reg] mem[memory reg]

Ex) reg mem1k[0:1023]



Ex) reg[7:0] mema[0:1023]



میتوان با اندیس به خانه های حافظه اشاره کرد، اما نمیتوان چند کامه حافظه را در یک زمان استفاده کرد: چون حافظه فقط یک گذرگاه داده دارد:

Mem1k[10] نوشتار درست هست

Mem1k[10:20] نوشتار غلط دسترسی به چند خانه حافظه بطور همزمان

Mema[100] نوشتار درست هست

Mema[100:101] نوشتار غلط دسترسی به چند خانه حافظه بطور همزمان

Bbus=mema[20]; محتوای خانه ۲۰ حافظه mema در bbus قرار می گیرد

- در مقدار دهی، اگر متغیر کوچکتر در بزرگتر قرار گیرد، بخش پر ارزش، صفر می شود.

- اگر متغیر بزرگتر در کوچکتر قرار گیرد، بخش پر ارزش تر برش می خورد.

Wire[3:0] a;

Wire [5:0] aa;

اگر مقدار اولیه aa=001110 در اینصورت:

Assign a=aa;>>>>>>>a=1110

اگر مقدار a=0110 آنگاه

Assign aa=a;>>>>>>>aa=000110

<http://www.idars.ir>, @idars_elearning_group

فصل ۷: طراحی مدارات در سطح رفتاری

در مواردی مثل توابع -کارها- برنامه های تست -ثبات ها و ماشین های حالت می توان از سطح رفتاری استفاده کرد.

در سطح رفتاری بدنه به صورت زیر روال (procedure) می باشد و به صورت زیر است:

Always Begin end دستورات	Always یک دستور
initial Begin end دستورات	initial یک دستور

رفتاری یا باید داخل always, initial باشد تمام تستها رفتاری بودند.

دستورات initial یکبار تکرار و دستورات always بینهایت تکرار می شود (چند بار). دستورات داخل بدنه initial و always خط به خط اجرا می شوند (ترتیبی) و در زمان صفر شروع به اجرا می کنند اگر تاخیر نداشته باشند. اما چند بلوک initial و always به صورت همروند اجرا می شوند.

مثال:

Always

A=0,b=1

Begin

#5;

a=~a;

b=~b;

end

initial

begin

#5

a=~a;

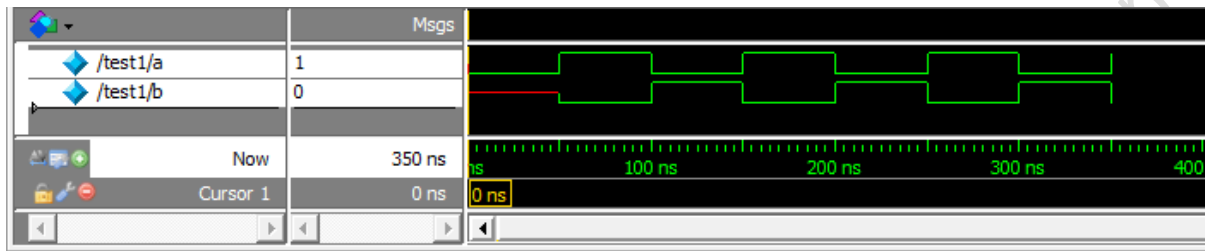
b=~b;

end

```

1  `timescale 10ns/1ns
2  module test1;
3      reg a,b;
4      initial a=0;
5      always
6      begin
7          #5
8          a=~a;
9          b=~a;
10     end
11 endmodule

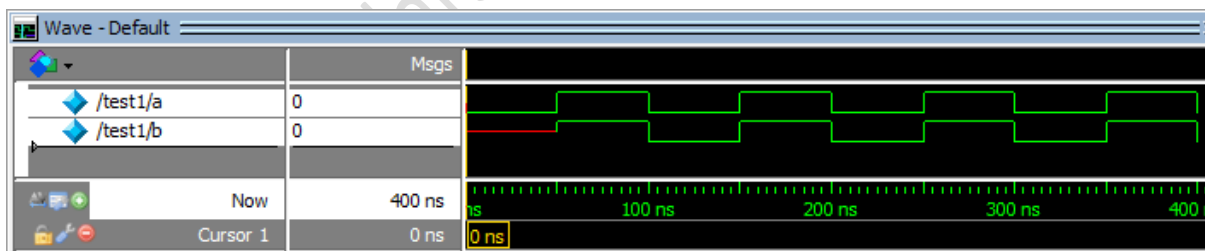
```



```

1  `timescale 10ns/1ns
2  module test1;
3      reg a,b;
4      initial a=0;
5      always
6      begin
7          #5
8          a<=~a;
9          b<=~a;
10     end
11 endmodule

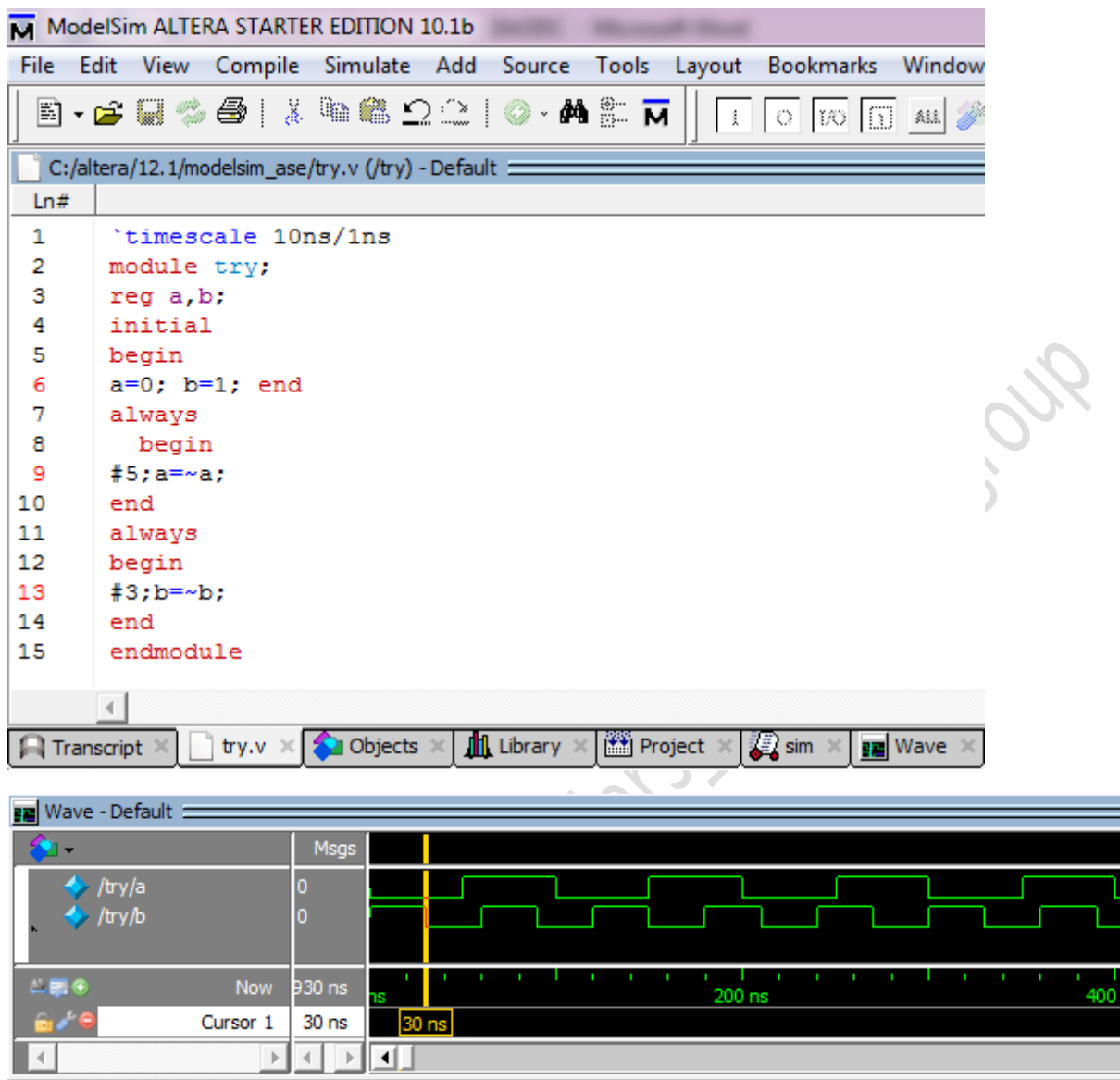
```



```

`timescale 1ns/100ps
initial
begin
#1
a<= #4 0;
b<= #2 1;
end

```



دستورات کنترل زمانی:

#Time : به اندازه time صبر می کند.

#(time-expression) : به اندازه عبارت t-e صبر می کند.

@Variable : منتظر تغییر متغیر می ماند.

@(Variable or expression) : منتظر تغییر متغیر در عبارت یا متغیر می ماند.

@(posedge Variable or expression) : منتظر تغییر صفر به یک عبارت یا متغیر می ماند.

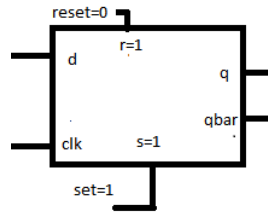
@(negedge Variable or expression) : منتظر تغییر یک به صفر عبارت یا متغیر می ماند.

@(event1 or event2 or ...) : منتظر تغییر یکی از رخدادها می ماند.

Wait(expression) : تا زمانی که عبارت غلط است منتظر می ماند.

Wait(a) : تا زمانی که a=0 است منتظر است به محض اینکه a=1 شود از انتظار خارج میشود.

مثال سطح رفتاری ff-sr:



```

1  `timescale 1ns/100ps
2  module d_sr_ff(d,set,rst,clk,q,qb);
3  input d,set,rst,clk;
4  output q,qb;
5  reg q,qb,state;
6  parameter sq_delay=6,rq_delay=6 ,cq_delay=6;
7  always@ (posedge clk or posedge rst or posedge set)
8  begin
9  if(set)begin
10     state=#sq_delay 1;
11 end else if(rst)
12 begin
13     state=#rq_delay 0;
14 end else if(clk)
15 begin
16     state=#cq_delay d;
17 end
18 end
19 always@ ( state)
20 begin
21     q=state ;
22     qb=~state ;
23 end
24 endmodule

```

طریقه نوشتن case:

در توابع و تسک ها و initial و always قابل استفاده است.

```

case(expression)
1:statement;
2:statement;
...
N :statement ;
default :statement ;
endcase

```

در case عبارت می تواند مقادیر صفر و یک داشته باشد.

اگر از case x استفاده کنیم مقادیر X ,Z نیز اضافه می شود.

اگر از case z استفاده کنیم مقدار Z نیز اضافه می شود.

نوشتن حلقه: ۴ روش برای نوشتن حلقه وجود دارد:

۱: استفاده از forever:

Initial forever		always
Begin	=	begin
بدنه یا چند دستور		بدنه یا چند دستور
End		End

۲: استفاده از repeat(expression):

```
always
begin
  repeat(5)@(posedge clock); // 5 iterations delay
  areg=abus;
end
```

عبارت @(posedge clock), ۵ بار تکرار می‌شود.

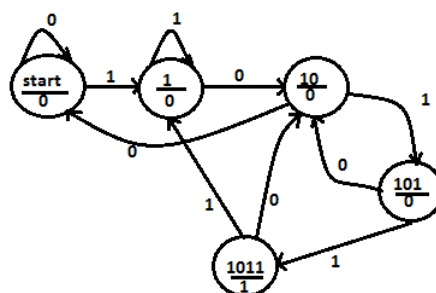
۳: حلقه while:

```
While(expression)
Begin
...
end
```

۴: حلقه for:

```
for( initial_assignment; expression; step_assignment )
Begin
...
end
```

مثال: توصیف رفتاری یک تشخیص دهنده رشته ۱۰۱۱ را بنویسید؟ با استفاده از ماشین حالت دیاگرام تشخیص این رشته را رسم می‌کنیم. سپس کد مربوط به آن را می‌نویسیم. (مدل مور تغییر روی خروجی به حالت وابسته است بنابراین با تغییر کلاک خروجی تغییر می‌یابد).

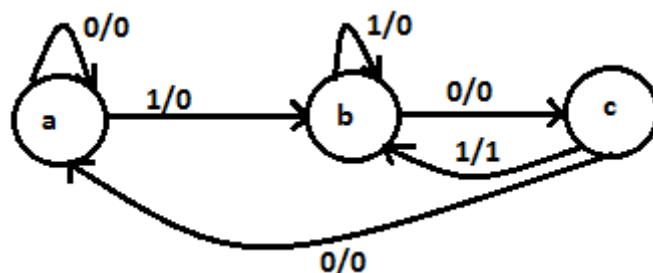


```

`timescale 1ns/100ps
module moore_detector(x,clk,z);
input x,clk;
output z;
reg[2:0] current; reg z;
parameter[2:0]reset=0,got1=1,got10=2,got101=3,got1011=4;
initial current = reset;
always@(posedge clk)
    case(current)
        reset: begin
            if(x==1)    current = got1;
            else        current = reset;
            z=0;
        end
        got1: begin
            if(x==0)    current = got10;
            else        current = got1;
            z=0;
        end
        got10:begin
            if(x==1)    current = got101;
            else        current = reset;
            z=0;
        end
        got101:begin
            if(x==1)
                begin
                    current = got1011;
                    z=1;
                end else
                begin
                    current = got10;
                    z=0;
                end
            end
        end
        got1011:begin
            if(x==1) current=got1;
            else current=got10;
            z=0;
        end
    endcase
endmodule

```

مثال: توصیف رفتاری تشخیص دهنده ۱۰۱ به مدل میلی: در مدل میلی خروجی وابسته به ورودی است. در این مدل تنها با تغییر حالت خروجی عوض نمی شود بلکه با تغییر در ورودی بدون وابستگی به کلاک تغییر اعمال می شود.



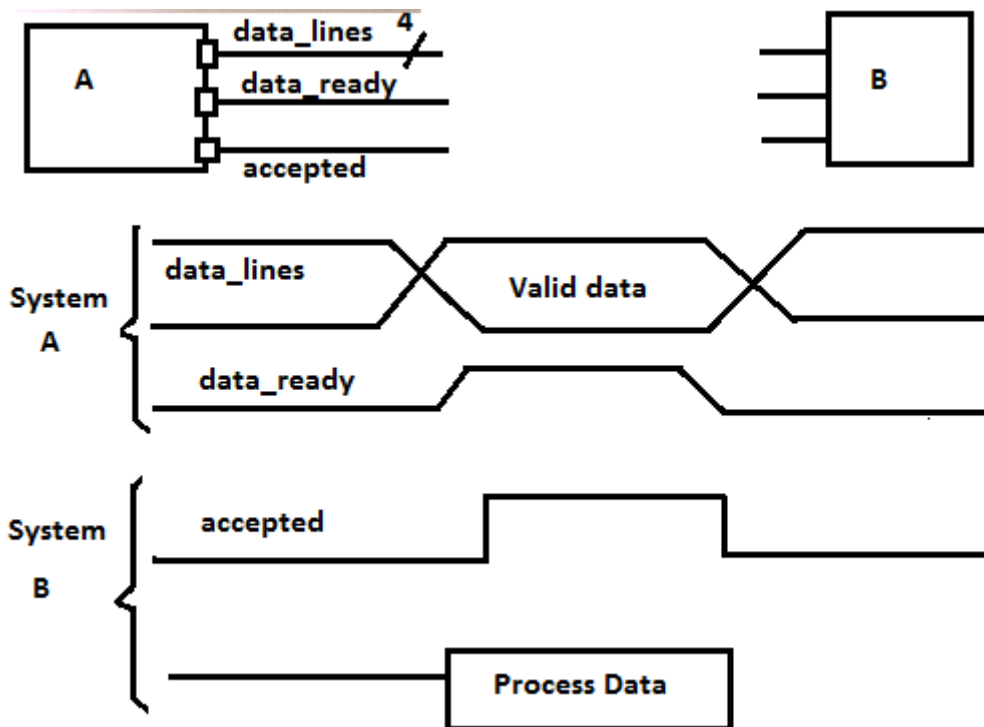
```

Ln#
1   `timescale 1ns/100ps
2   module asynch_reset_detector(x,r,clk,z);
3   input x,r,clk;
4   output z;
5   reg[1:0] nxt,present;
6   reg z;
7   parameter a=0, b=1, c=2;
8   initial nxt = a;
9   always @(posedge clk or posedge r)
10  begin:registring
11    if(r==1) present = a;
12    else present = nxt;
13  end
14  always @(present or x)
15  begin:combinational
16    z=0;
17    case(present)
18    a:
19      if(x==0) nxt = a;
20      else nxt = b;
21    b:
22      if(x==0) nxt = c;
23      else nxt = b;
24    c:
25      if(x==0) nxt = a;
26      else nxt = b;
27    default:
28      nxt = a;
29    endcase
30    if(present == c && x==1) z=1;
31  end
32  endmodule
  
```

مثال: پیاده سازی دست تکان دهی^۱: ارتباط غیر همزمان^۲ بین سیستم‌ها توسط دست تکان دهی انجام می‌شود. دست تکان دهی یعنی سیگنالهایی که بین دو سیستم انجام می‌شود. وقتی که یک سیستم داده‌هایی را برای انتقال به سیستم دیگر آماده می‌کند، فرستنده گیرنده را آگاه می‌کند که داده آماده است. وقتی که گیرنده داده‌ها را می‌پذیرد به فرستنده دریافت آنها را اطلاع می‌دهد.

¹ handshaking

² asynchronous



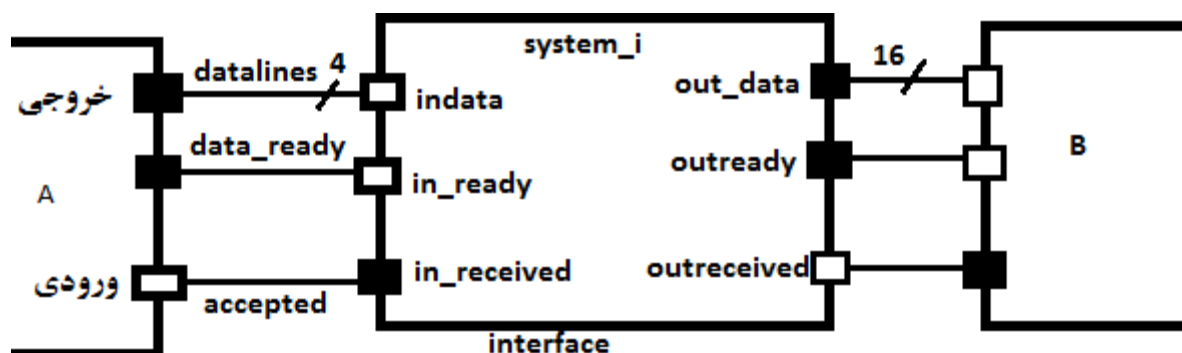
سیستم A بعد از آماده شدن عملیات زیر را انجام می دهد:

- Data-lines = داده های آماده شده روی آن قرار داده می شود.
- Data-ready=1
- accepted مثبت می شود.
- Data-ready=0

سیستم B:

- منظر لبه مثبت data-ready می شود .
- Accepted=1
- داده ها را پردازش می کند.
- منظر لبه منفی data-ready می شود .
- Accepted=0

مثال: فرض کنید سیستم A دارای خط داده ۴ بیتی و سیستم B دارای خط داده ۱۶ بیتی است یک مدار رابط طراحی کنید که به روش دست تکان دهی بین این دو ارتباط برقرار کند؟



```

1  `timescale 1ns/100ps
2  module system_i(in_data,out_data,in_ready,out_ready,in_recieved,out_recieved);
3  input[*:0] in_data;      input in_ready,out_recieved;
4  output[15:0] out_data;   output out_ready, in_recieved;
5  reg[15:0] out_data;      reg out_ready,in_recieved;
6  reg[*:0] word_buffer[*:0]; reg buffer_full,buffer_picked;
7  reg[0:0] i;
8  initial begin i=0; end
9  always begin: atalk
10     @(posedge in_ready);
11     word_buffer[i] = in_data;
12     if(i==3) begin
13         buffer_full = 1;
14     @(posedge buffer_picked);
15     buffer_full = 0;
16     i = 0;
17     end else i=i+1;
18     in_recieved = 1;
19     @(negedge in_ready);
20     in_recieved = 0;
21     end
22 always begin: b_talk
23     wait(buffer_full);
24     out_data={word_buffer[3],
25             word_buffer[2],
26             word_buffer[1],
27             word_buffer[0]};
28     buffer_picked = 1;
29     @(negedge buffer_full);
30     buffer_picked = 0;
31     out_ready = 1;
32     @(posedge out_recieved);
33     out_ready = 0;
34     end
35 endmodule

```

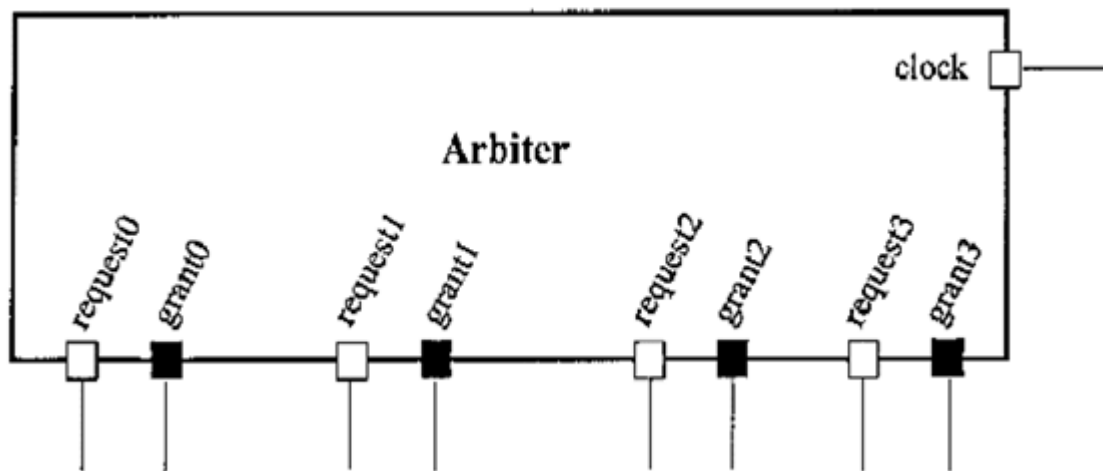
```

1  `timescale 1ns/1ns
2  module test_system_interface;
3      reg [3:0] a_data;
4      reg a_ready;
5      wire a_received;
6      wire [15:0] b_data;
7      wire b_ready;
8      reg b_received;
9      integer i,seed;
10     system_i u1(a_data, b_data, a_ready, b_received, a_received, b_ready);
11     initial begin
12         for (i=1;i<=15; i=i+1) seed<=#(i*3) i;
13         a_ready=0;b_received=0;
14         #40 $stop;
15     end
16     //this models system_a
17     always begin
18         a_data=$random (seed);
19         repeat (4) begin
20             #2 a_data={~a_data[0],a_data[3:1]};
21             a_ready=1;
22             @(posedge a_received);
23             #1 a_ready=0;
24         end
25     end
26     //this models system_b
27     always begin
28         @(posedge b_ready);
29         #1 b_received=1;
30         @(negedge b_ready);
31         #2 b_received=0;
32     end
33 endmodule

```

مثال: طراحی مدار داوری

فرض کنیم ۴ دستگاه داریم که درخواست یک منبع مشترک را دارند. این منبع مشترک میتواند گذرگاه مشترک باشد. با توجه به این که گذرگاه در هر لحظه میتواند به یک دستگاه اختصاص داده شود نیازمند Arbitration می باشد. به پروسه اختصاص منبع مشترک به چند درخواست کننده داوری می گویند. هر دستگاه یک خط request و یک خط grant دارد که درخواست منبع مشترک را روی خط request میدهد و داور در صورتی که بخواهد منبع را به او بدهد خط grant را یک می کند. در این سیستم می خواهیم بالاترین اولویت مربوط به خط ۳ باشد. هنگامی که یک دستگاه با اولویت بالا تر درخواست می دهد ، داور به بقیه دستگاه ها پاسخ نخواهد داد. مدار داور را طراحی کنید.



'time scale 1ns/100ps

module arbiter (request, grant, clock);

input [3:0] request;

input clock;

output[3:0] grant;

reg [3:0] grant;

reg found;

integer i;

initial i = 0;

always begin

@(negedge clock);

#20;

found=0;

for(i=3; i>=0; i=i-1)

if(request[i] == 0) grant[i] = 0; l[i]=0;

if(grant == 4'd0)

for(i=3; i>=0; i=i-1)

if(request[i] && found == 0) begin

grant[i]=1;

found=1;

end

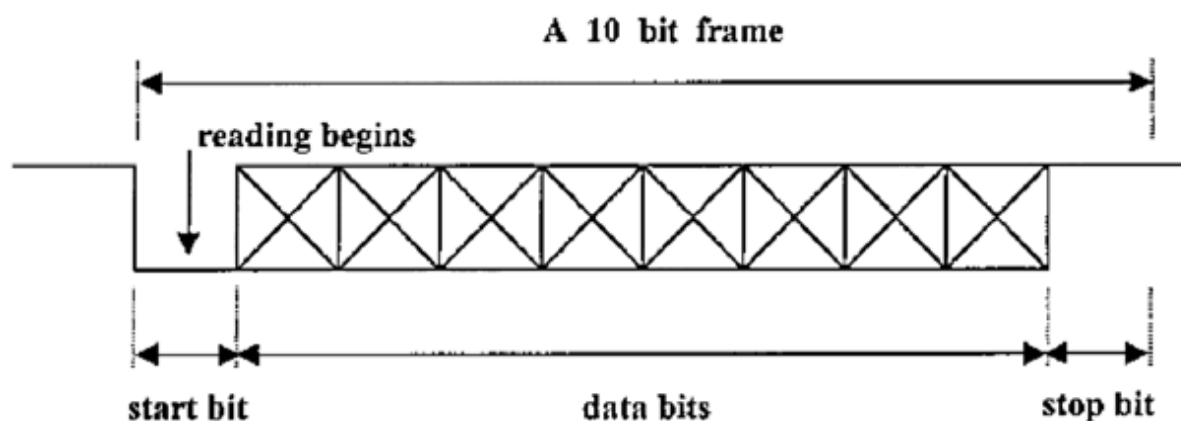
end

end module

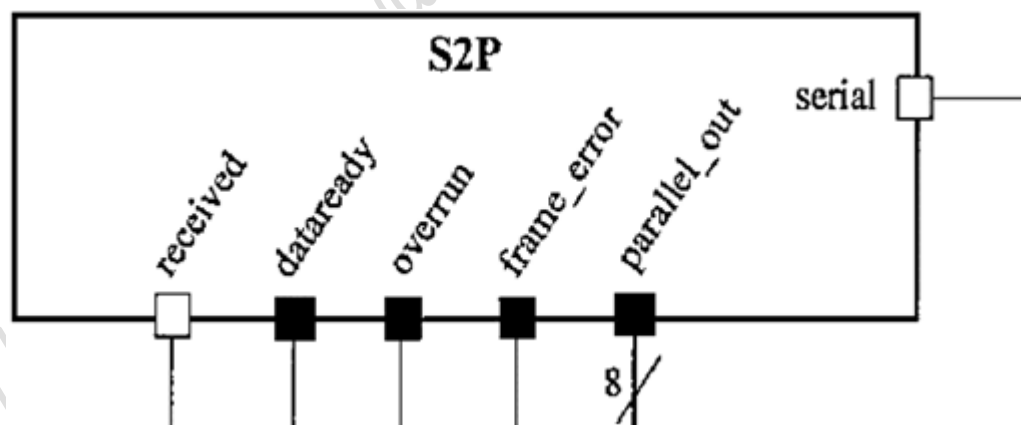
مثال: طراحی رابط سری به موازی

می خواهیم رابطی طراحی کنیم که یک دستگاه سریال را که فرستنده است به یک دستگاه موازی که گیرنده است متصل کند. برای تبادل اطلاعات بین این دو دستگاه خروجی موازی را هشت بیتی در نظر می گیریم.

برای شروع یک start bit با مقدار ۰ ، سپس ۸ بیت داده و در انتها یک stop bit با مقدار ۱ در نظر می گیریم. ساختار تبادلی به شکل زیر است.



برای ارسال هر بیت یک مدت زمان لازم است. که این مدت زمان وابسته به سرعت خطوط می باشد. به عنوان مثال port سریال های قدیمی با سرعت 9600 b/s کار می کردند، و زمان هر بیت 1/9600 می باشد.



این رابط بر اساس استاندارد RS232 کار می کند. خط frame_error زمانی ۱ می شود که بجای مقدار ۱ در stop bit صفر ظاهر شود. خط over_run زمانی اتفاق می افتد که سیستم گیرنده نتواند داده ی موازی را دریافت کند قبل از اینکه مجموعه داده بعدی آماده شود.

timescale 1ns/100ps'

define BPS 10000'

```

module serial2parallel(serial, received, dataready, overrun, frame_error,
parallel_out);
input serial, received;
output dataready, overrun, frame_error;
output [7:0] parallel_out;
reg dataready, overrun, frame_error;
reg [7:0] parallel_out;
reg [7:0] buff;
initial begin
frame_error=0;
dataready=0;
overrun=0;
parallel_out=1'b0;
end
parameter
half_bit=1E9/('BPS*2.0);
full_bit=1E9/'BPS;
always begin: collect
@(negedge serial);
#half_bit;]
repeat(8) #full_bit buff={ serial, buff[7:1]}
#full_bit;
if(serial==0) begin
frame_error=1;
@(posedge serial);
end else begin
frame_error=0;
dataready=1;
parallel_out=buff;

```

```

@(posedge received)@(negedge received);

dataready=0;

end

end //collect

always begin too_fast @(dataready);

if(dataready==1)begin

@(negedge serial) if (dataready==1) overrun=1;

end else

overrun=0;

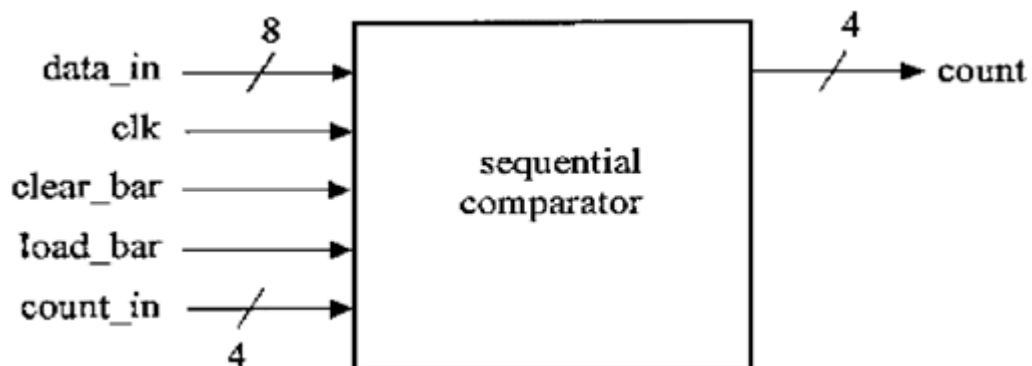
end

endmodule

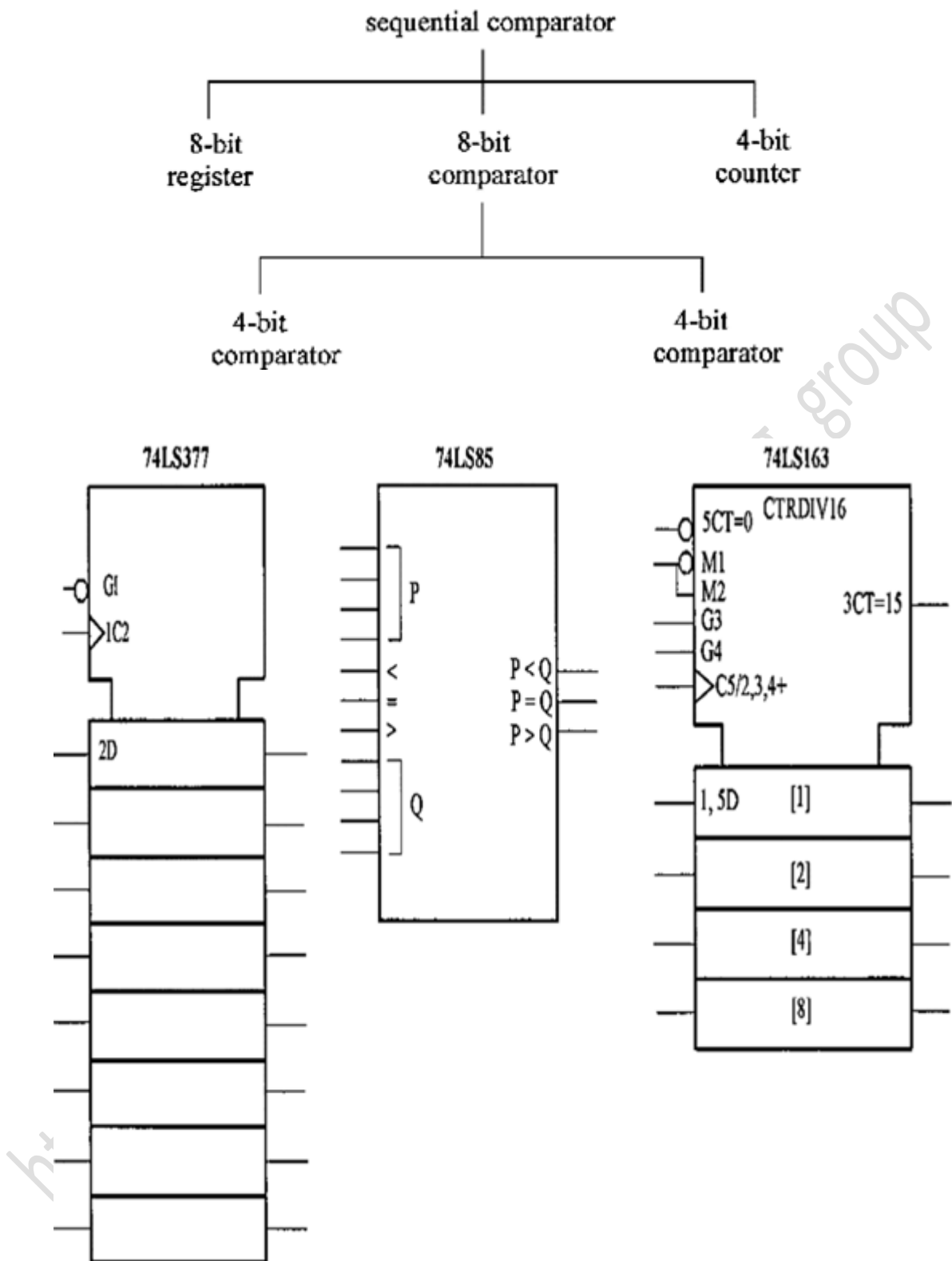
```

مثال: طراحی یک مقایسه کننده با استفاده از خانواده ی 74L500:

در این مقایسه گر میخواهیم مقایسه گر ترتیبی طراحی کنیم که data های ۸ بیتی را باهم مقایسه کند و تعداد تکرار داده های مساوی را مشخص کند.



توجه شود که ورودی های clear_bar , load_bar به صورت low active هستند. یعنی وقتی clear صفر شود، کار می کنند. ورودی clear_bar خروجی count را ۰ می کند و ورودی load_bar ورودی count_in را به شمارنده بار می کند.



توصیف رفتاری ثبات ۸ بیتی 74LS377:

```

'timescale 1ns/100ps
module ls377_register(clk, g_bar, d8, q8);

```



```

parameter n=8;
input clk, g_bar;
input [n-1:0] d8;
output [n-1:0] q8;
reg [n-1:0] q8;
parameter prop_delay=7;
always @(posedge clk)
    if (g_bar==0) q8 = #prop_delay d8;
end module

```

توصیف رفتاری مقایسه گر LS85 :

```

'timescale 1ns/ 100ps
module ls85_comparator(a, b, gt, lt, a_gt_b, a_eq_b, a_lt_b);
parameter n=4;
input [n-1:0] a, b;
input gt, eq, lt;
output a_gt_b, , a_eq_b, a_lt_b;
reg a_gt_b, , a_eq_b, a_lt_b;
parameter prop_delay=10;
always @(gt or eq or lt or a or b)
    if(a>b) begin
        a_gt_b <= #prop_delay 1;
        a_eq_b <= # prop_delay 0;
        a_lt_b <= # prop_delay 0;
    end else if (a<b) begin
        a_gt_b <= #prop_delay 0;
        a_eq_b <= # prop_delay 0;
        a_lt_b <= # prop_delay 1;
    end else if (a == b) begin
        a_gt_b <= #prop_delay gt;

```

```

        a_eq_b <= # prop_delay eq;
        a_lt_b <= # prop_delay lt;
    end
endmodule

```

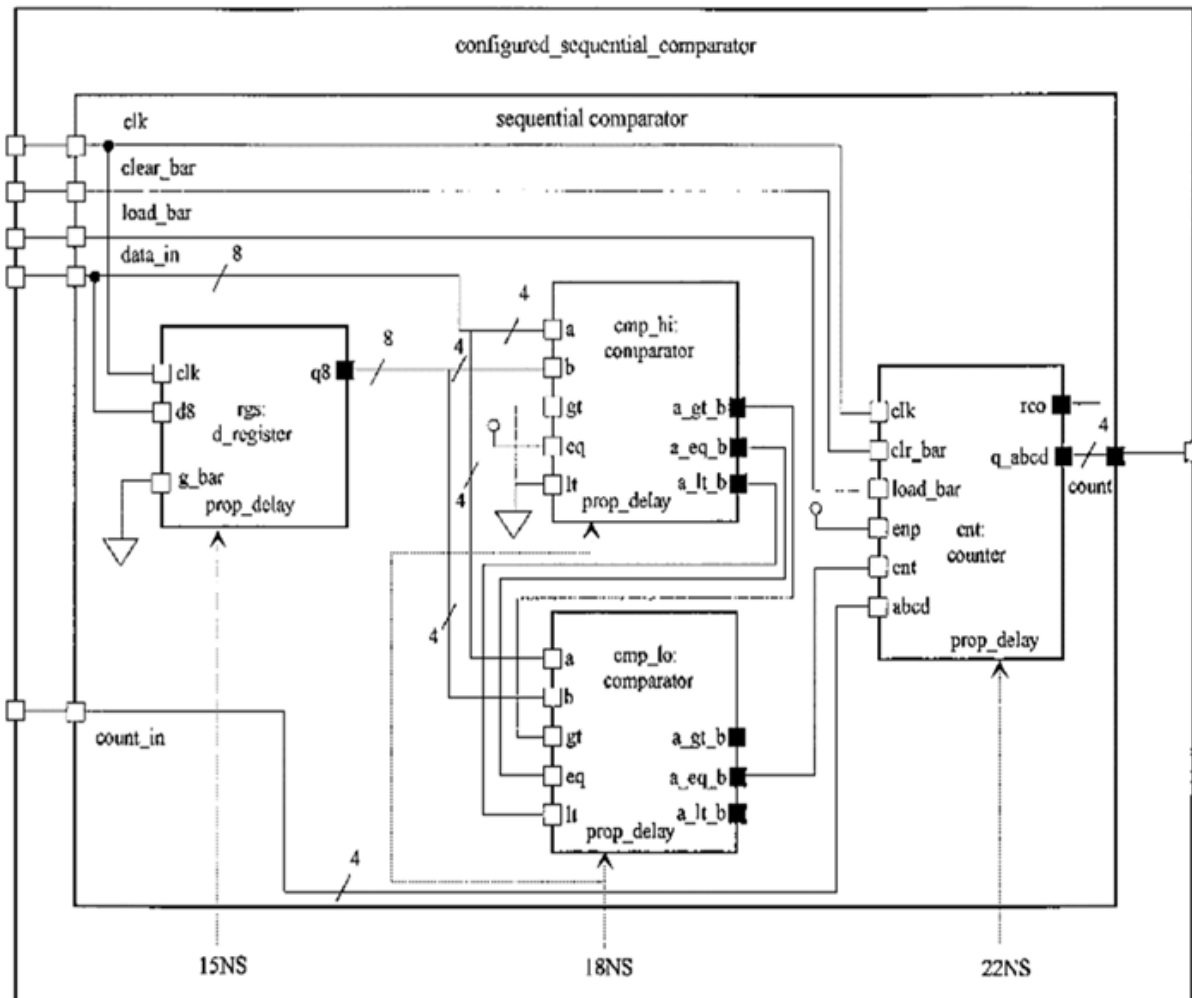
توصیف رفتاری شمارنده ی LS163:

```

'timescale 1ns/100ps
module ls163_counter (clk, clr_bar, ld_bar, enp, ent, abcd, q_abcd, rco);
parameter n=4, m=4'b1111;
input clk, clr_bar, ld_bar, enp, ent;
input [n-1:0] abcd;
output [n-1:0] q_abcd;
output rco;
reg [n-1:0] q_abcd;
reg rco;
parameter prop_delay= 12;
reg [n-1:0] internal_count;
    always @(posedge clk) begin : counying
        if(clr_bar == 0) internal_count=0;
        else if(ld_bar==0) internal_count= abcd;
        else if(enp==1 && ent==1) internal_count=internal_count+1;
        if(internal_count==m && ent==1)
            rco=#prop_delay 1;
        else rco=0;
        q_abcd= #prop_delay internal_count;
    end
endmodule

```

تا اینجا هر بخش را به صورت جداگانه طراحی کردیم حال باید ارتباط بین آن ها را بنویسیم.



'timescale 1ns/100ps

module sequential_comparator (data_in, clk, clear_bar, load_bar, count_in,

count);

input [7:0] data_in;

input clk, clear_bar, load_bar;

input [3:0] count_in;

output [3:0] count;

supply0 gnd; supply1 vdd;

specify

\$width (posedge clk, 1E3);

endspecify

wire [7:0] old_data;

wire compare_out, gt_i, eq_i, lt_i;

```

ls377_register rgs (clk, gnd, data_in, old_data);

ls85_comparator cmp_lo (data_in [3:0], old_data[3:0], gnd,      vdd, gnd,
gt_i, eq_i, lt_i);

ls85_comparator cmp_hi(data_in [7:4], old_data [7:4], gt_i,      eq_i, lt_i,
compare_out, );

ls163_counter crl (clk, clear_bar, load_bar, vdd, compare_out,
count_in, count, );

endmodule

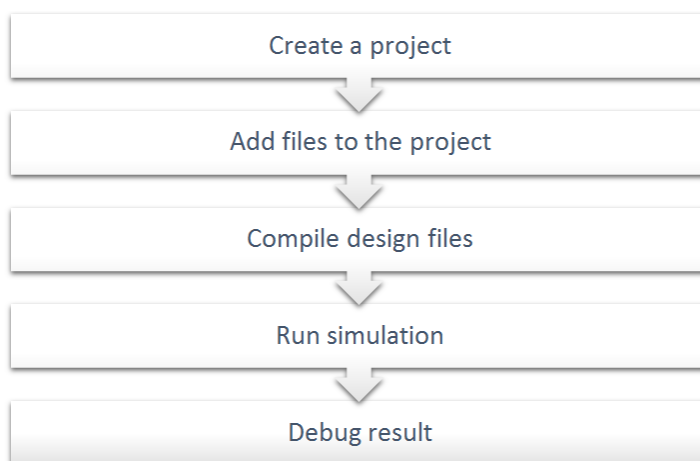
```

ضمیمه:

قصد داریم دو مثال را توسط نرم افزار modelsim پیاده سازی کنیم تا شما نحوه ی کار با این نرم افزار را نیز بیاموزید . پس در ابتدا به سراغ آموزش نرم افزار modelsim می رویم .

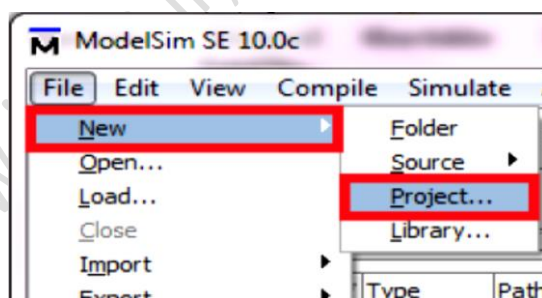
✓ آموزش modelsim :

مراحلی که برای پیاده سازی و اجرا گرفتن از یک کد باید طی کنیم به شکل زیر است :

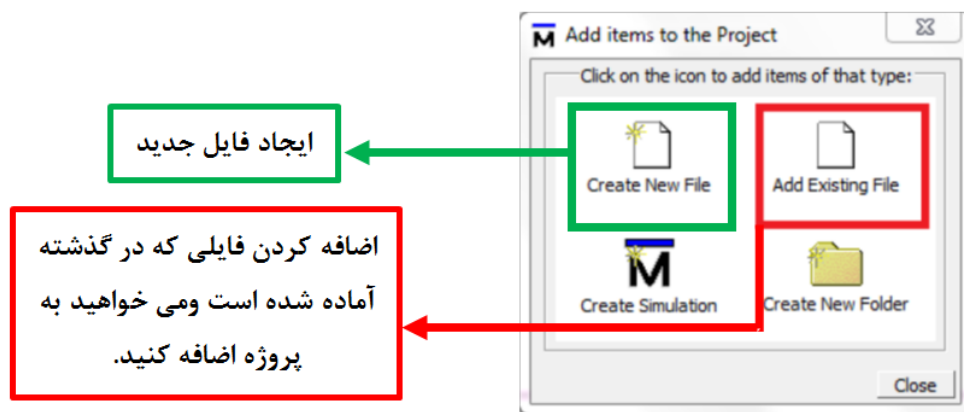


برای ایجاد پروژه جدید به صورت زیر عمل کنید:

File ⇒ New ⇒ Project

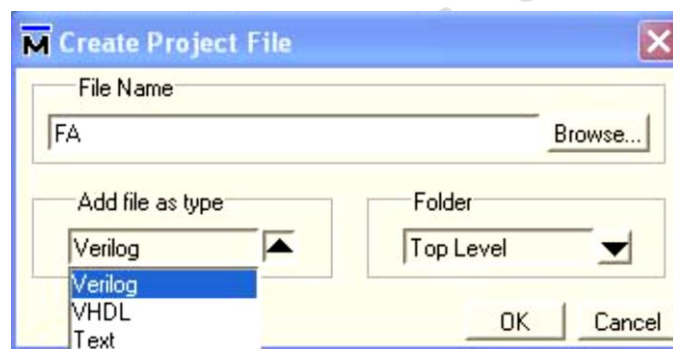


سپس پنجره ای باز خواهد شد که در قسمت Project Name ، نام پروژه و در قسمت Location محل ذخیره پروژه را مشخص می کنید و بعد دکمه OK را می زنید و پنجره زیر باز می شود :

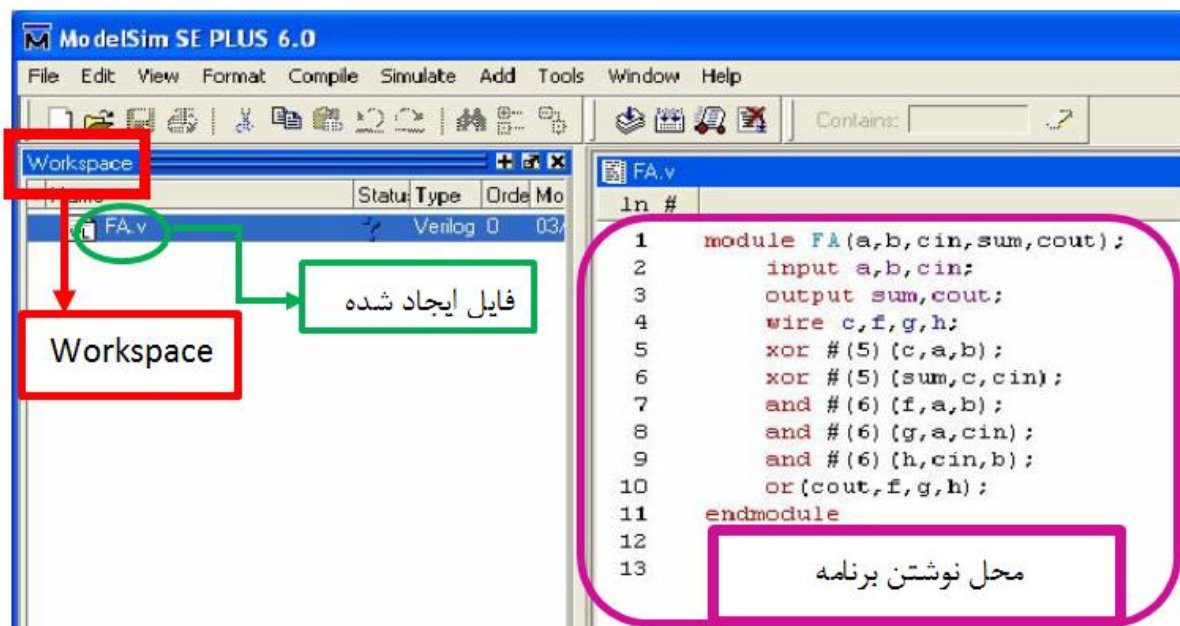


۱. انتخاب Create New File :

بعد از کلیک بر روی Create New File صفحه زیر باز می شود که در آن زبان برنامه نویسی و نام فایل خود را انتخاب می کنید. اگر زبان برنامه نویسی خود را VHDL انتخاب کنید، فایلی با پسوند vhd ایجاد می کند و اگر زبان Verilog را انتخاب کنید، فایلی با پسوند v ایجاد می کند که ما با زبان Verilog کار می کنیم.

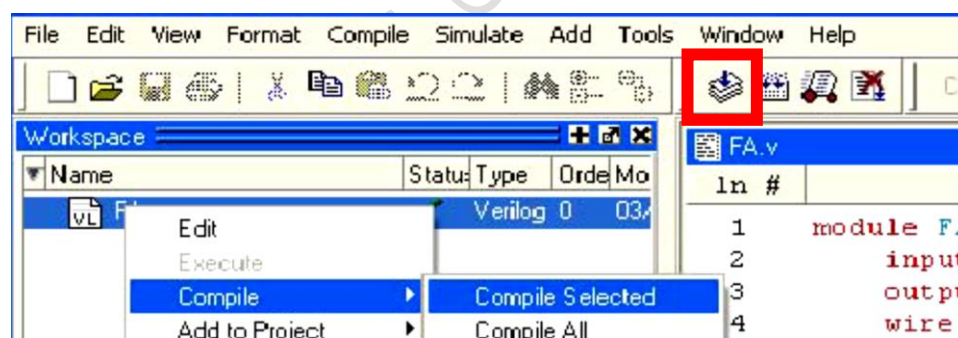


بعد از OK کردن، فایل شما در Project ⇔ Workspace اضافه می شود. حال با دوبار کلیک بر روی نام فایل (مثلاً FA.v)، صفحه ای باز می شود که کد را داخل آن می نویسیم و در انتها Save می کنیم.

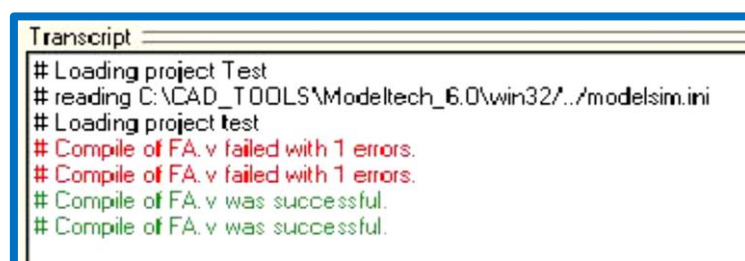


در مرحله بعد باید کد خود را Compile کنیم تا از نظر اصول برنامه نویسی Verilog یا VHDL چک شود. در صورتی که کد شما اشکالی نداشته باشد، شما را با پیغامی که در Transcript می گذارد، آگاه میکند.

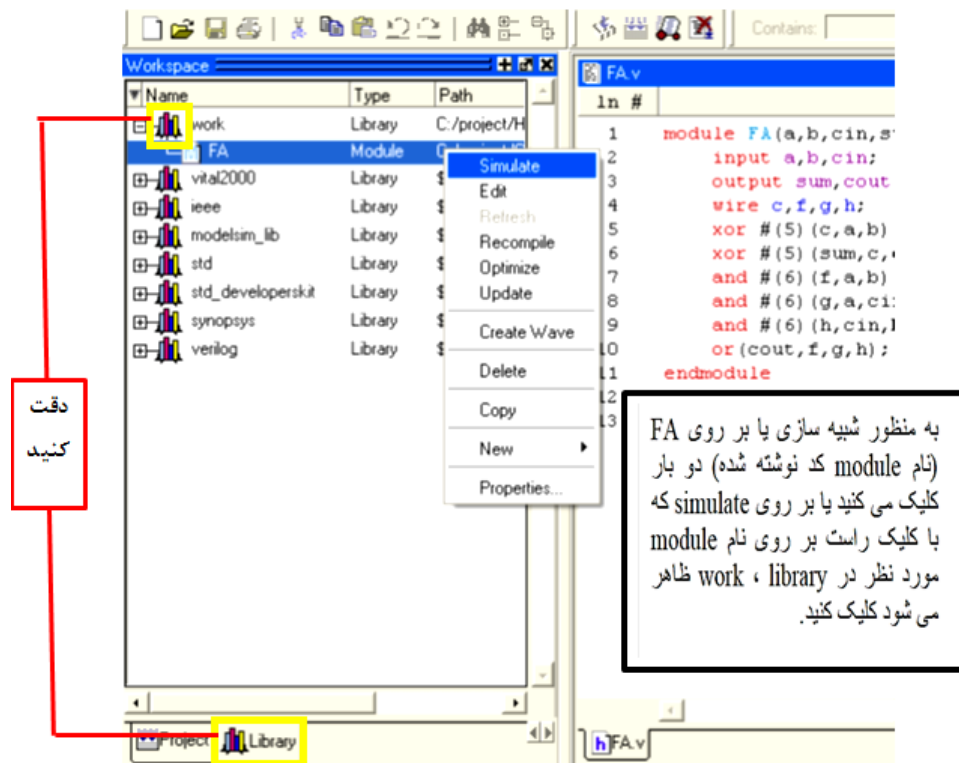
برای این کار دو راه دارید، می توانید روی فایل راست کلیک کرده و گزینه ی Compile را بزنید و یا گزینه Compile را از نوار ابزار انتخاب کنید:



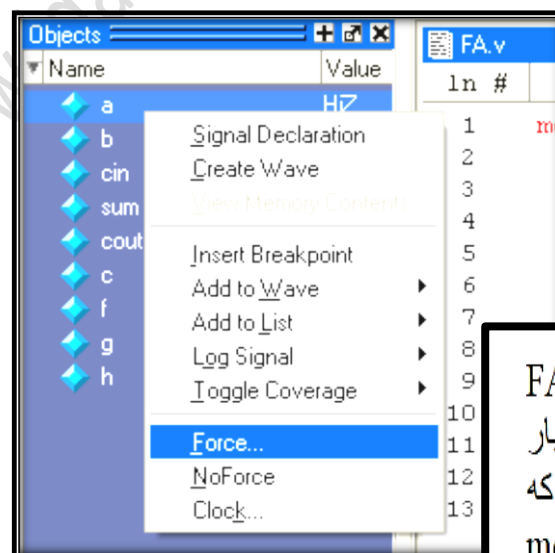
اگر Compile شما موفقیت آمیز باشد در قسمت Transcript به رنگ سبز نشان داده می شود و اگر خطا داشته باشد به رنگ قرمز نشان داده می شود که اگر روی آن دو بار کلیک کنید می توانید از نوع و محل خطا آگاه شوید.



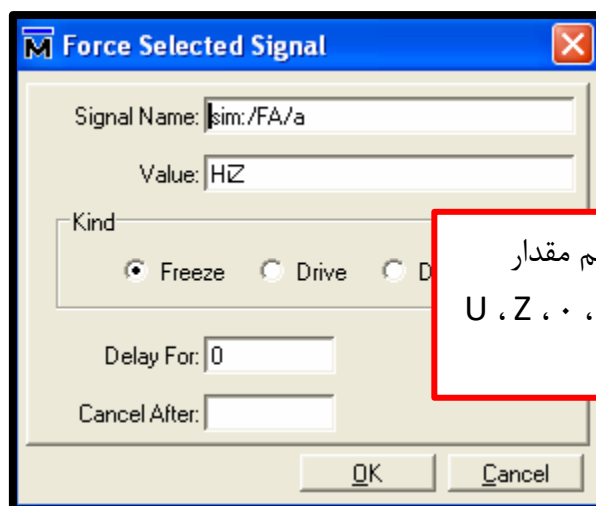
بعد از عمل **Compile** نوبت **Simulation** است .



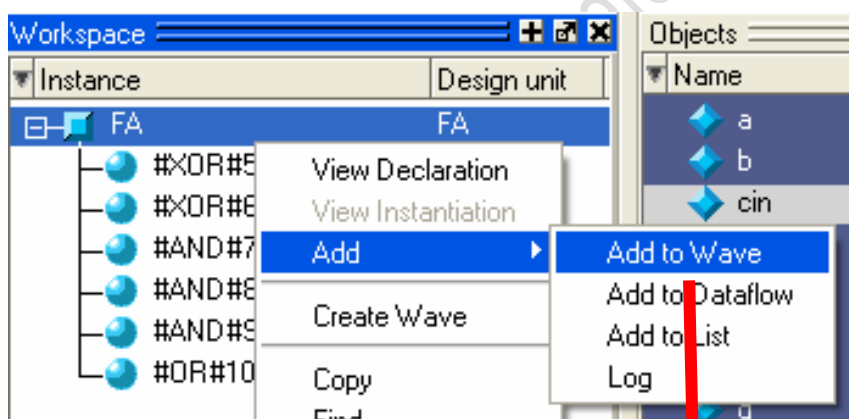
حال نوبت آن است که Port های ورودی را مقداردهی کنیم . با کلیک راست بر روی هر کدام از object های موجود و با انتخاب Force و یا Clock می توانیم آن ها را مقداردهی کنیم . (در بعضی version ها باید روی object کلیک راست کنید و بعد wave را بزنید و بعد در پنجره باز شده دوباره روی object کلیک راست کرده و سپس Force و یا Clock را انتخاب می کنیم .)



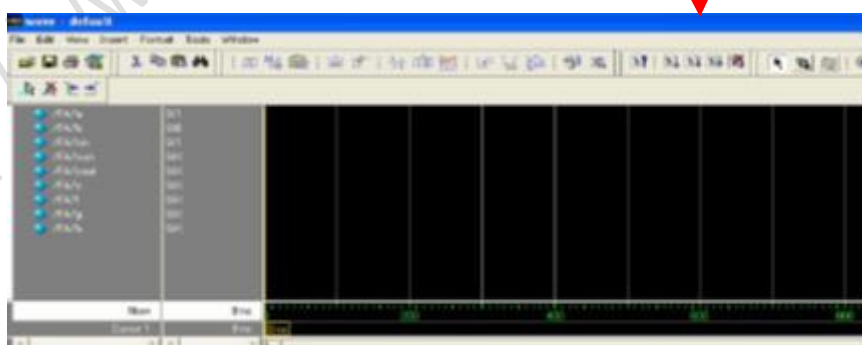
بر باز می شود :



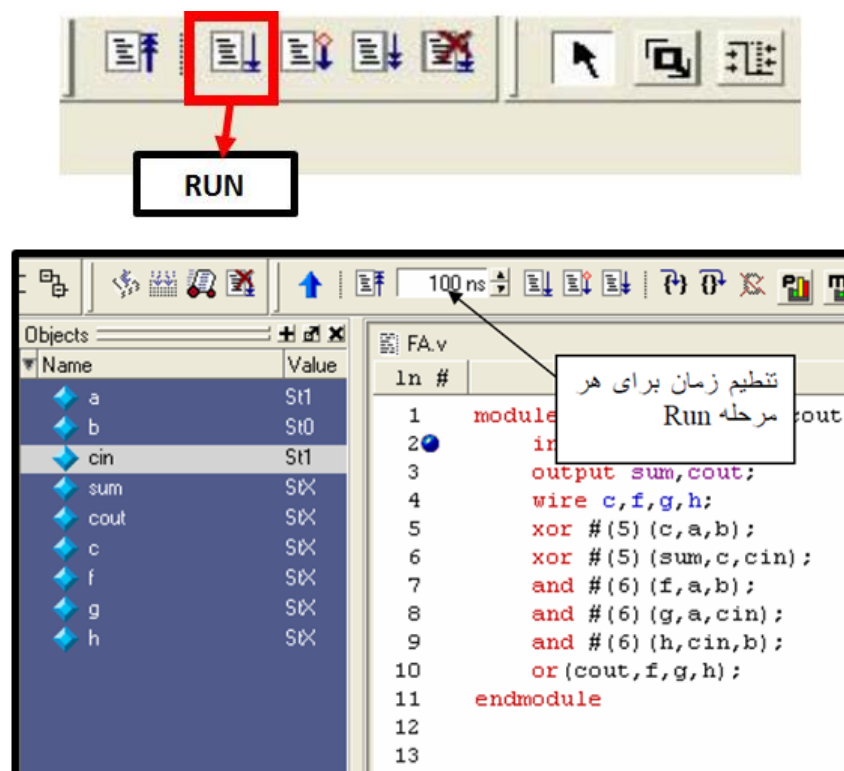
بعد از اینکه object های ورودی را مقداردهی کردید ، نوبت آن است که موج خروجی را ببینید . برای این کار به این روش عمل کنید : راست کلیک روی نام فایل «-- Add --» «Add to Wave» که با کلیک بر روی «Add to Wave» ، پنجره Wave باز می شود .



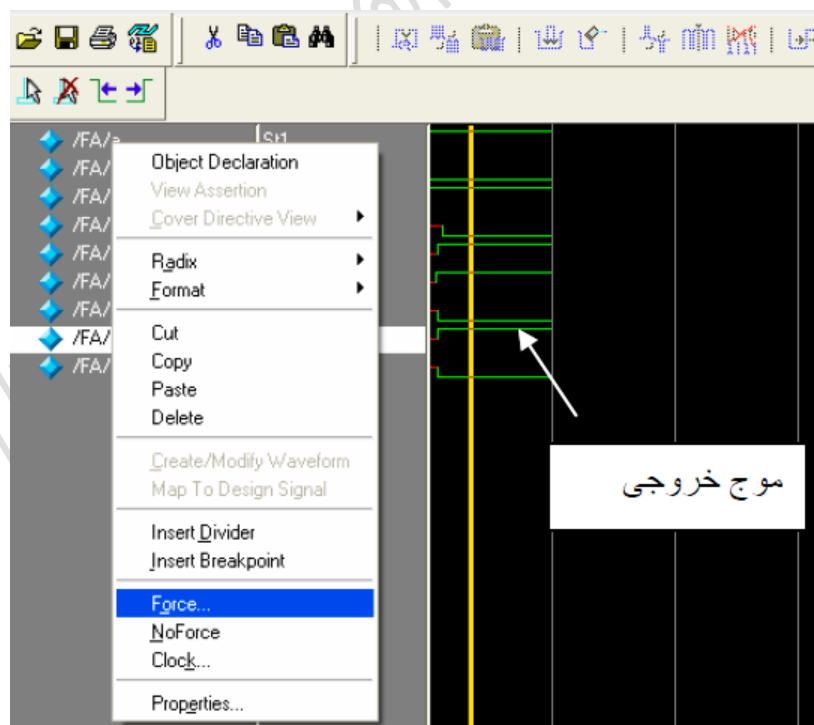
پنجره ی Wave



حال با کلیک بر روی «Run» ، موج خروجی را به اندازه ی زمانی که تعریف شده است (به طور پیش فرض 100ns) رسم می کند و با فشار مجدد ، به اندازه آن زمان دوباره رسم می کند .

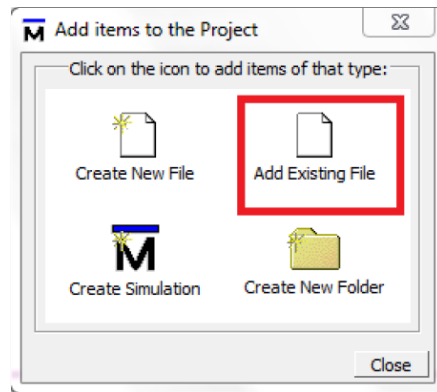


اگر در زمان Run خواستید مقدار یک سیگنال را تغییر دهید می توانید بر روی آن در صفحه Wave کلیک راست کنید و Force را انتخاب کنید.



۲. انتخاب Add Existing File :

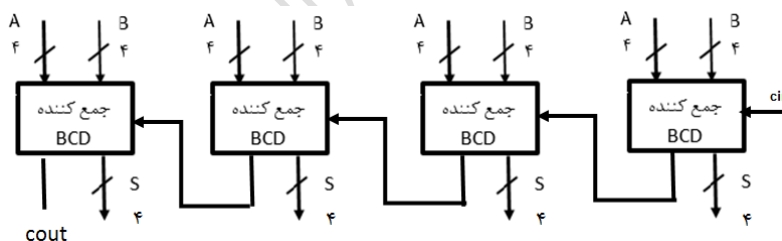
همان طور که قبلا گفتیم زمانی که یک پروژه ایجاد میکنیم ، پنجره زیر باز میشود :



که در قسمت قبل انتخاب **Create New File** را به طور کامل توضیح دادیم. زمانی از گزینه ی **Add Existing File** استفاده میکنیم که بخواهیم برنامه هایی که قبلا نوشته ایم را به پروژه اضافه کنیم و از آنها استفاده کنیم تا مجبور نباشیم یک بار دیگر آن ها را بنویسیم. با کلیک بر روی **Add Existing File** ، پنجره ی **Add File to Project** باز میشود ، **Browse** را می زنیم و برنامه هایی را که قبلا نوشتیم که پسوند **.v** دارند را انتخاب کرده و **OK** میکنیم .

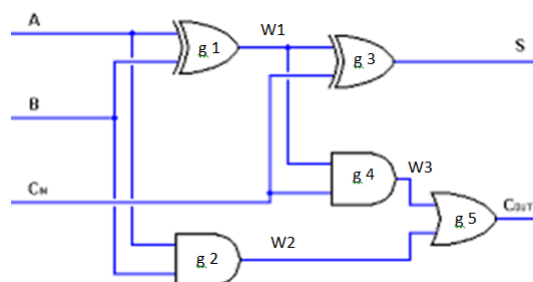
حال در پنجره **Project** فایل ها را می بینید که در قسمت **statues** آن ها علامت سوال وجود دارد که یعنی هنوز **Compile** نشده اند . همان طور که قبلا هم توضیح داده بودیم ، آنها را **Compile** کرده و باقی مراحل هم (simulation , ...) مانند قبل می باشد .

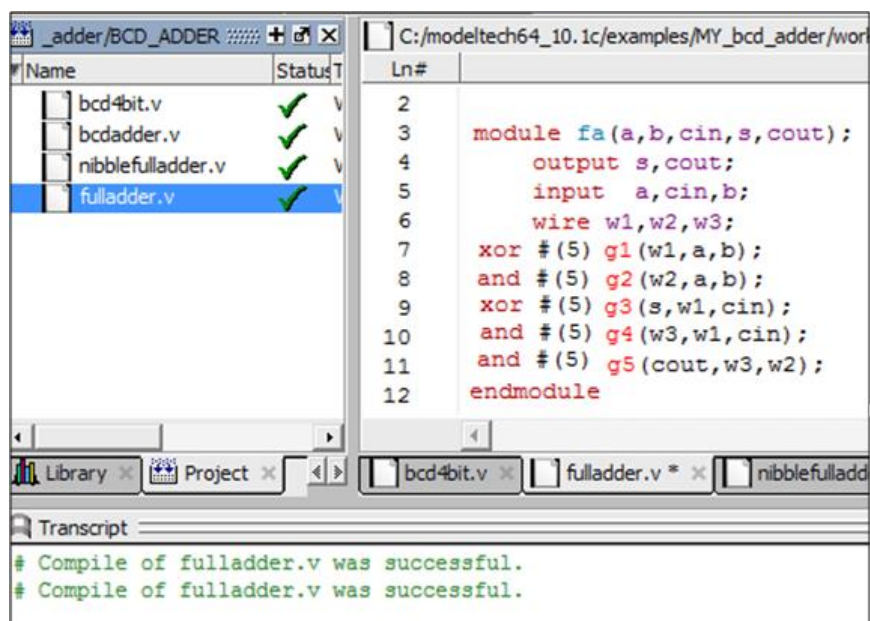
مثال (۱) یک مدار طراحی کنید که دو عدد ۴ رقمی **BCD** را با هم جمع کند .



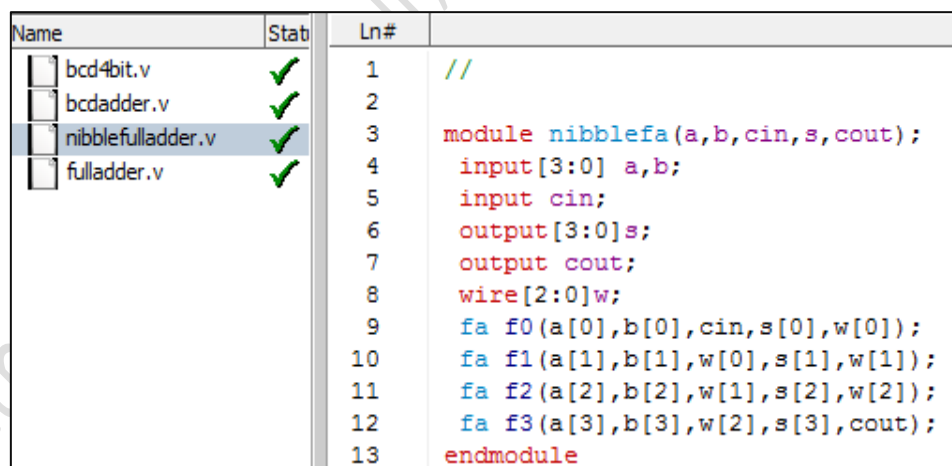
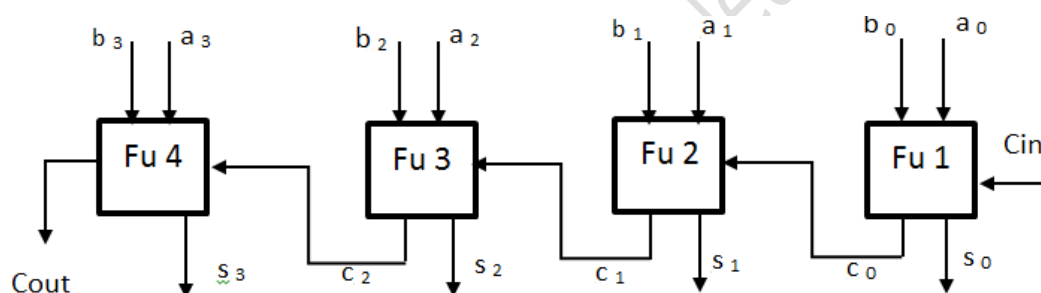
حل (برای حل این سوال ، باید ۴ فایل ایجاد کنیم :

۱- Full Adder

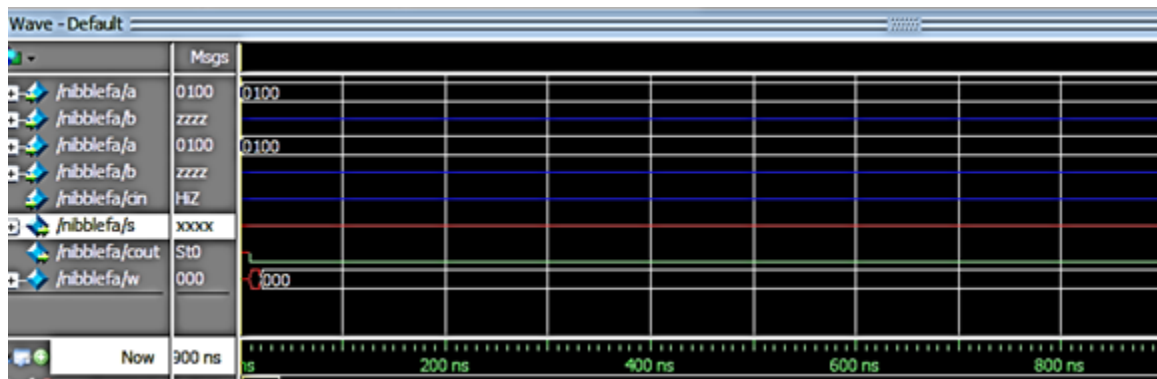




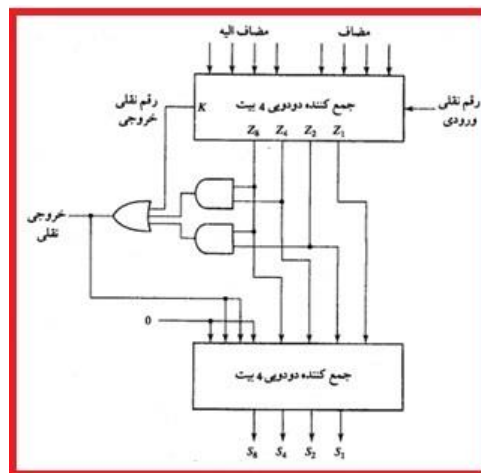
۳- جمع کننده ۴ بیتی (با استفاده از ۴ عدد Full Adder)



Simulate جمع کننده ۴ بیتی



۳- جمع کننده BCD ۴ بیتی



```

C:\modeltech64_10.1c\examples\My_bcd_adder\work\bcd4bit.v - Default
Name Ln#
bcd4bit.v 1 //bcd for 4 bit
bcdadder.v 2
nibblefullad 3 module BCD4(a,b,cin,s,cout);
fulladder.v 4 input [0:3] a,b;
5 input cin;
6 output [0:3] s;
7 output cout;
8 wire [0:3] z;
9 wire k,w1,w2;
10 supply0 gnd;
11 fa4 f1({a[0],a[1],a[2],a[3]},{b[0],b[1],b[2],b[3]},cin,{s[0],s[1],s[2],s[3]},cout);
12 fa4 f2({z[0],z[1],z[2],z[3]},{gnd,cout,cout,gnd},{s[0],s[1],s[2],s[3]},cout);
13 and#(5)
14 g1(w1,z[3],z[1]);
15 g2(w2,z[3],z[2]);
16 or#(8)
17 g3(cout,k,w1,w2);
18 endmodule
19

```

۴- جمع کننده BCD ۱۶ بیتی (با استفاده از ۴ عدد جمع کننده BCD ۴ بیتی) (شکل صورت مثال)

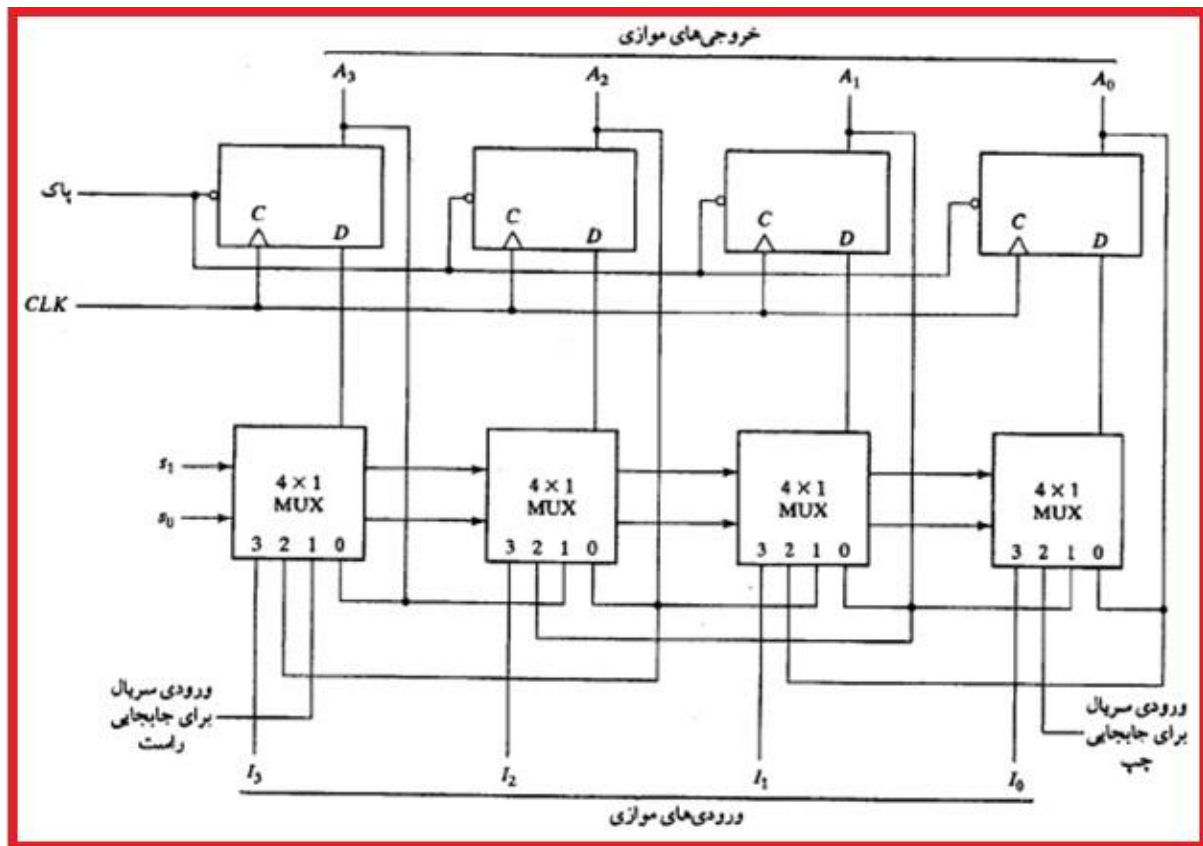
```

1 //adder miner
2
3 module BCDAdder(a,b,cin,s,cout);
4     input [15:0] a,b;
5     input cin;
6     output [15:0] s;
7     output cout;
8     wire w1,w2,w3;
9     BCD4 b1({a[0],a[1],a[2],a[3]}, {b[0],b[1],b[2],b[3]}, cin,s[0],s[1],s[2],s[3],w1);
10    BCD4 b2({a[4],a[5],a[6],a[7]}, {b[4],b[5],b[6],b[7]}, w1,s[4],s[5],s[6],s[7],w2);
11    BCD4 b3({a[8],a[9],a[10],a[11]}, {b[8],b[9],b[10],b[11]}, w2,s[8],s[9],s[10],s[11],w3);
12    BCD4 b4({a[12],a[13],a[14],a[15]}, {b[12],b[13],b[14],b[15]}, w3,s[12],s[13],s[14],s[15],cout);
13
14 endmodule
15

```

مثال ۲) یک شیفت رجیستر ۸ بیتی با قابلیت بار موازی طراحی کنید که قابلیت های شیفت را داشته باشد.

S1	S0	عمل
0	0	بلا تغییر
0	1	شیفت راست
1	0	شیفت چپ
1	1	بار موازی از ورودی



برای حل این مثال باید سه فایل ایجاد کنیم :

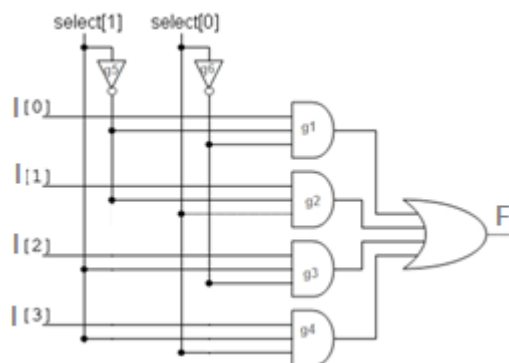
۱. مالتی پلکسر 4×1

۲. D فلیپ - فلاپ

۳. Shift - register

در Shift - register باید هر کدام از مالتی پلکسر 4×1 و D فلیپ - فلاپ را چندین بار فراخوانی کنیم . البته با توجه به این که خروجی مالتی پلکسر ، ورودی D فلیپ - فلاپ میباشد میتوان یک ماژول دیگر ساخت که در آن ، این دو را فراخوانی کنیم و در Shift - register از آن ماژول استفاده کنیم .

❖ مالتی پلکسر 4×1

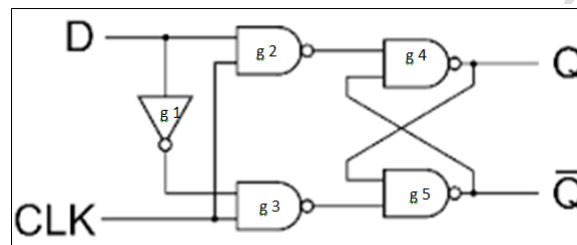


```

1  module mux41(I0,I1,I2,I3,sel0,sel1,f);
2      input I0,I1,I2,I3,sel0,sel1;
3      output f;
4      wire[1:6] w;
5      and #(6) g1(w[1],I0,w5,w6);
6      and #(6) g2(w[2],I1,sel0,w5);
7      and #(6) g3(w[3],I2,sel1,w6);
8      and #(6) g4(w[4],I3,sel1,sel0);
9      or #(8) g5(f,w[1],w[2],w[3],w[4]);
10     not #(4) g6(w[5],sel1);
11     not #(4) g7(w[6],sel0);
12 endmodule

```

❖ D – flip flop :



```

1  module dflop(d,c,q);
2      input d,c;
3      output q;
4      wire r,w1,w2,w3;
5      not #(4) g1(r,d);
6      nand #(5) g2(w1,d,c);
7      nand #(5) g3(w2,r,c);
8      nand #(5) g4(q,w1,w3);
9      nand #(5) g5(w3,w2,q);
10 endmodule

```

❖ Shift – register : (شکل صورت سوال)


```

3  module shiftreg(I,serialR,serialL,sel0,sel1,clk,A);
4  input[0:7] I;
5  input serialR,serialL,sel0,sel1,clk;
6  output[0:7] A;
7  reg[0:7] A;
8  wire[0:7] w;
9  mux41 m0(A[0],A[1],serialL,I[0],sel0,sel1,w[0]);
10 mux41 m1(A[1],A[2],A[0],I[1],sel0,sel1,w[1]);
11 mux41 m2(A[2],A[3],A[1],I[2],sel0,sel1,w[2]);
12 mux41 m3(A[3],A[4],A[2],I[3],sel0,sel1,w[3]);
13 mux41 m4(A[4],A[5],A[3],I[4],sel0,sel1,w[4]);
14 mux41 m5(A[5],A[6],A[4],I[5],sel0,sel1,w[5]);
15 mux41 m6(A[6],A[7],A[5],I[6],sel0,sel1,w[6]);
16 mux41 m7(A[7],serialL,A[6],I[7],sel0,sel1,w[7]);
17 dflop d0(w[0],clk,A[0]);
18 dflop d1(w[1],clk,A[1]);
19 dflop d2(w[2],clk,A[2]);
20 dflop d3(w[3],clk,A[3]);
21 dflop d4(w[4],clk,A[4]);
22 dflop d5(w[5],clk,A[5]);
23 dflop d6(w[6],clk,A[6]);
24 dflop d7(w[7],clk,A[7]);
25 endmodule

```

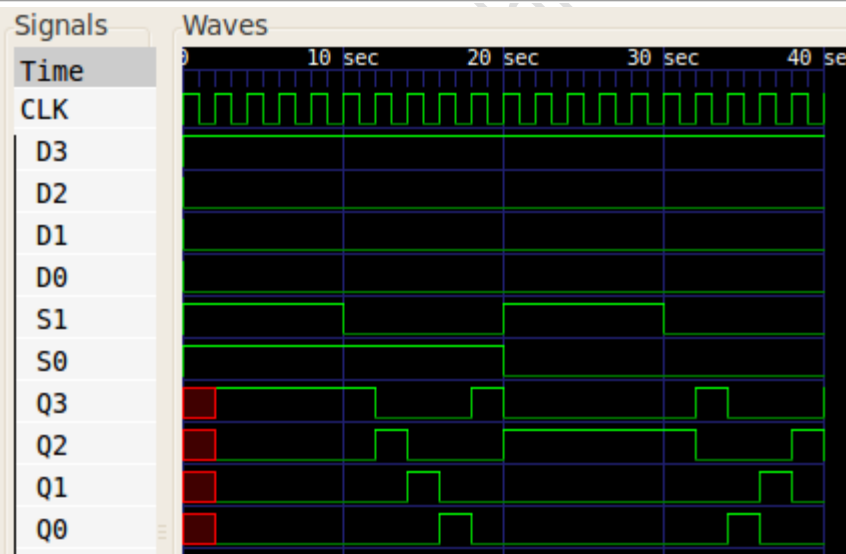


Figure ۱۱ این شکل برای یک شیفت رجیستر ۴ بیتی است

تست برای شیفت رجیستر :

```

C:/modeltech64_10.1c/examples/MY_shift_register/ttt.v (/shifter) - Default
Ln#
1  module shifter(
2      input [31:0] shamt,
3      output reg [31:0] result );
4      reg[31:0] temp;
5      always@(*)begin
6          temp=32'hfff00fff;
7          result[31:0]=temp[31:0]>>>shamt[4:0];
8      end
9  endmodule

```

همان طور که قبلاً گفتیم با توجه به این که خروجی مالتی پلکسر ، ورودی D فلیپ - فلاپ می باشد میتوان یک ماژول دیگر ساخت که در آن ، این دو را فراخوانی کنیم و در Shift - register از آن ماژول استفاده کنیم .

```

Ln#
1  module mux_flop(i0,i1,i2,i3,s0,s1,clk,q);
2      input i0,i1,i2,i3,s0,s1,clk;
3      output q;
4      wire d;
5      mux41 m(i0,i1,i2,i3,s0,s1,d);
6      dflop f(d,clk,q);
7
8  endmodule

```

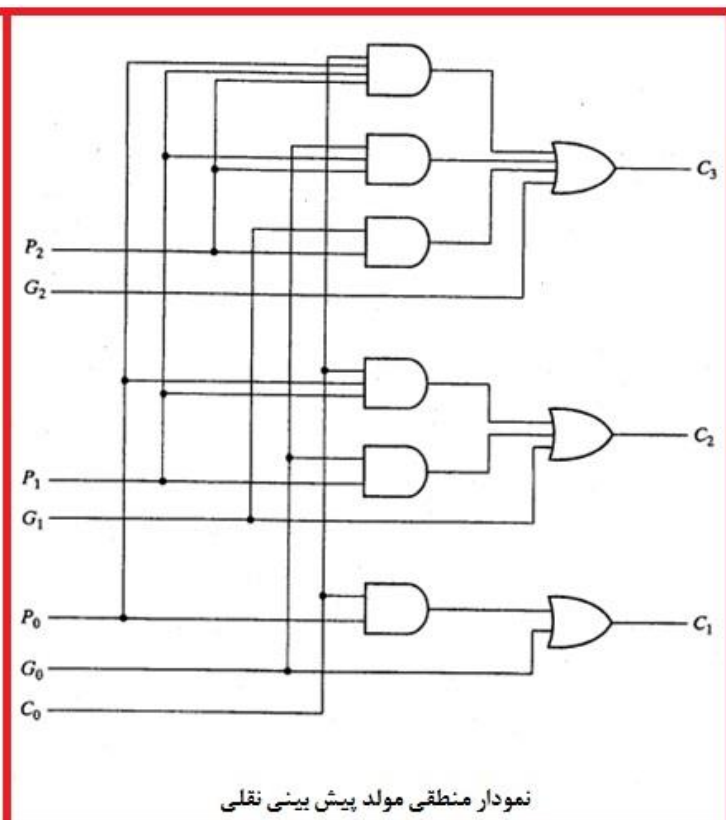
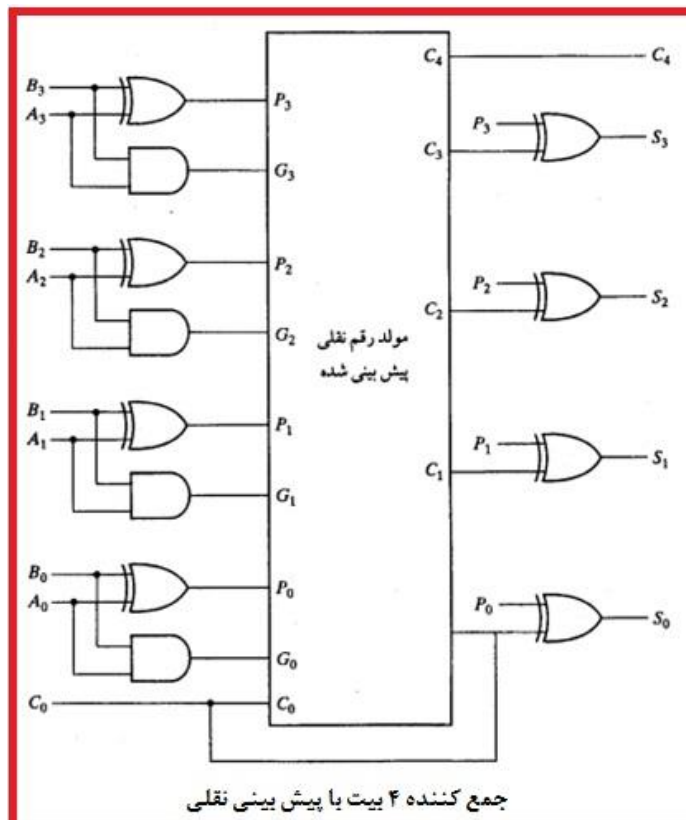
که در این صورت کد شیفت رجیستر به صورت زیر خواهد بود :

```

1  //shiftregister for 8 bit
2  module shiftreg(I,serialR,serialL,sel0,sel1,clk,A);
3      input[0:7] I;
4      input serialR,serialL,sel0,sel1,clk;
5      output[0:7] A;
6      reg[0:7] A;
7      mux_flop m1(A[0],A[1],serialL,I[0],sel0,sel1,clk,A[0]);
8      mux_flop m2(A[1],A[2],A[0],I[1],sel0,sel1,clk,A[1]);
9      mux_flop m3(A[2],A[3],A[1],I[2],sel0,sel1,clk,A[2]);
10     mux_flop m4(A[3],A[4],A[2],I[3],sel0,sel1,clk,A[3]);
11     mux_flop m5(A[4],A[5],A[3],I[4],sel0,sel1,clk,A[4]);
12     mux_flop m6(A[5],A[6],A[4],I[5],sel0,sel1,clk,A[5]);
13     mux_flop m7(A[6],A[7],A[5],I[6],sel0,sel1,clk,A[6]);
14     mux_flop m8(A[7],serialR,A[6],I[7],sel0,sel1,clk,A[7]);
15 endmodule

```

مثال ۴: می خواهیم یک مدار جمع کننده ی CLA ۱۶ بیتی با ۴ عدد مدار جمع کننده ی CLA ۴ بیتی بسازیم. مدار را در سطح گیت پیاده سازی کنید.



$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1$$

$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2$$

$$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

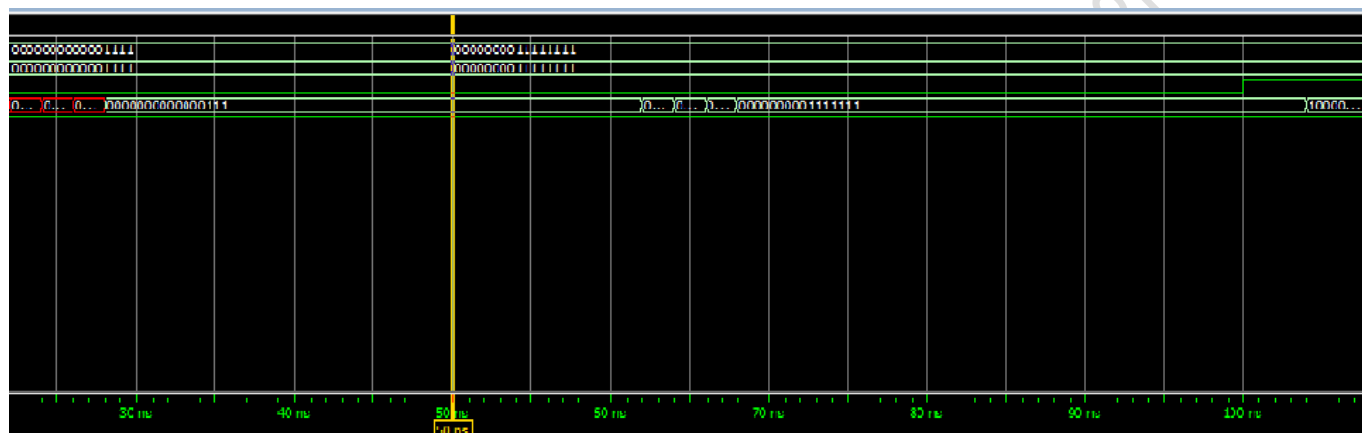
۱- برنامه طراحی ماژول CLA برای ۴ بیت.

da16.v	✓	1	module cla(P,G,cin,C,cout);
test.v	✓	2	input cin;
da4.v	✓	3	input[0:3] P,G;
da.v	✓	4	output[1:3] C;
		5	output cout;
		6	wire[1:10] w;
		7	and # (4) g1(w[1],P[0],cin);
		8	and # (4) g2(w[2],P[1],G[0]);
		9	and # (6) g3(w[3],P[0],P[1],cin);
		10	and # (4) g4(w[4],P[2],G[1]);
		11	and # (6) g5(w[5],P[2],P[1],G[0]);
		12	and # (8) g6(w[6],P[2],P[1],P[0],cin);
		13	and # (4) g7(w[7],P[2],G[3]);
		14	and # (6) g8(w[8],P[3],P[2],G[1]);
		15	and # (10) g9(w[9],P[3],P[2],P[1],P[0],G[0]);
		16	and # (10) g10(w[10],P[3],P[2],P[1],P[0],cin);
		17	or # (4) g11(C[1],w[1],G[0]);
		18	or # (6) g12(C[2],w[2],w[3],G[1]);
		19	or # (8) g13(C[3],w[4],w[5],w[6],G[2]);
		20	or # (10) g14(cout,w[7],w[8],w[9],w[10],G[3]);
		21	endmodule

da16.v	✓	1	module cla4(A,B,cin,S,cout);
test.v	✓	2	input cin;
da4.v	✓	3	input[0:3] A,B;
da.v	✓	4	output[0:3] S;
		5	output cout;
		6	wire[0:3] G,P;
		7	wire[1:3] C;
		8	and # (4) x0(G[0],A[0],B[0]);
		9	and # (4) x1(G[1],A[1],B[1]);
		10	and # (4) x2(G[2],A[2],B[2]);
		11	and # (4) x3(G[3],A[3],B[3]);
		12	xor # (4) x4(P[0],A[0],B[0]);
		13	xor # (4) x5(P[1],A[1],B[1]);
		14	xor # (4) x6(P[2],A[2],B[2]);
		15	xor # (4) x7(P[3],A[3],B[3]);
		16	xor # (4) x8(S[0],P[0],cin);
		17	xor # (4) x9(S[1],P[1],C[1]);
		18	xor # (4) x10(S[2],P[2],C[2]);
		19	xor # (4) x11(S[3],P[3],C[3]);
		20	cla c1(P,G,cin,C,cout);
		21	endmodule

۲. جمع کننده ۴ بیت با پیش بینی رقم نقلی

da16.v	✓	Verilog	1	module cla16(A,B,cin,S,cout);
da4.v	✓	Verilog	2	input[0:15] A,B;
da.v	✓	Verilog	3	input cin;
			4	output[0:15] S;
			5	output cout;
			6	wire[1:3] w;
			7	cla4 c11({A[0:3]},{B[0:3]},cin,{S[0:3]},w[1]);
			8	cla4 c12({A[4:7]},{B[4:7]},w[1],{S[4:7]},w[2]);
			9	cla4 c13({A[8:11]},{B[8:11]},w[2],{S[8:11]},w[3]);
			10	cla4 c14({A[12:15]},{B[12:15]},w[3],{S[12:15]},cout);
			11	endmodule



۳. جمع کننده ۱۶ بیت با پیش بینی رقم نقلی

-تست جمع کننده

```

module test_CLA16;
  reg[0:15] A,B;
  reg cin;
  wire[0:15] S;
  wire cout;
  cla16 u1(A,B,cin,S,cout);
  initial begin
    A = 16'b1111;
    B = 16'b1111;
    cin= 0;
    #50
    A = 16'b11111111;
    B = 16'b11111111;
    cin= 0;
    #50
    cin= 1;

  end
endmodule

```