Subject: ۱

Year : Month : Day : ( )

بسمه تعالی

زبان های توصیف سخت افزار
(FPGA)

محمد صادق عبدی

FPGA :

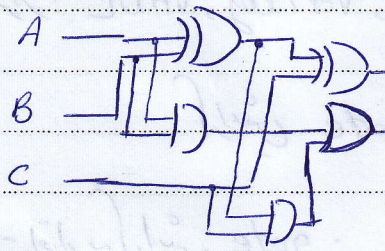
Field Programmable Gate Array

fpga یک مدار مجتمع قابل برنامه ریزی شامل دریایی از گیت های منطقی خام است.

نحوه طراحی مدارات fpga :

مدار جمع کننده :

Full Adder




طرح شماتیک :

کشیدن طرح شماتیک مشکل است.

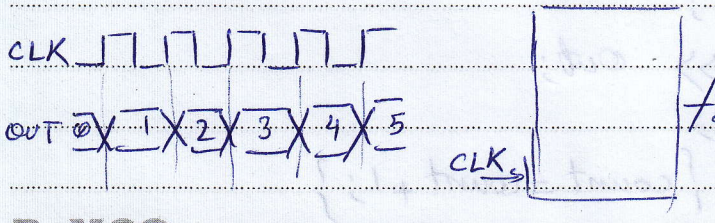
نرم افزاری می خواهیم که :

با چیدن
جعبه ای به آن می گوییم را انجام
با آرایش گیتی مدارها آشنا باشد

شمارنده ۸ تایی :

مداری می خواهیم که :

خروجی اش در هر لبه بالارونده یکی
زیاد شود.




PAYCO

Subject: ۲

Year : Month : Day : ( )

همیشه ـ در هر لبه بالا رونده ـ خروجی یکی زیاد شود.

always @(posedge CLK)
Out <= Out+1;

زبان نرم افزار،

نوشتن برنامه (کامل نیست):

module 8bit Counter (input Clk, Output [7:0] Out);
always @(posedge Clk)
Out <= Out+1;
end module

/

برای Verilog خاص برای نوشتن برنامه زبان برنامه نویسی syntax

زبان های توصیف سخت افزار: System_C , Verilog, VHDL

Verilog
VHDL
System_C
→ Synthesis → gate آرایش

Synthesizer : gate نرم افزار مبدل زبان سخت افزار به آرایش

نوشتن برنامه بالا به زبان System_C :
(کانترِ بیتی)

۴ بیتی

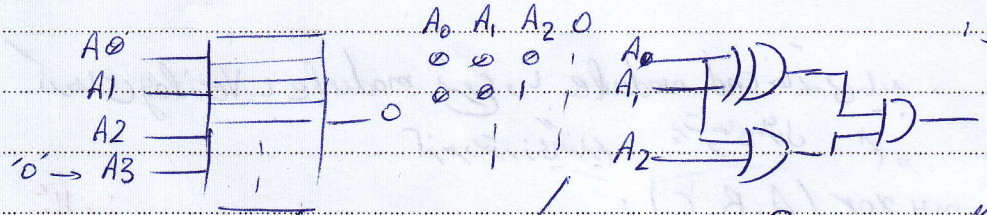
SC_MODULE (Counter) {
SC_in_clk Clock;
SC_out <sc_uint<4>> Out;
sc_uint<4> count;
Void inc_counter () { count = count + 1; }

Subject: ۳

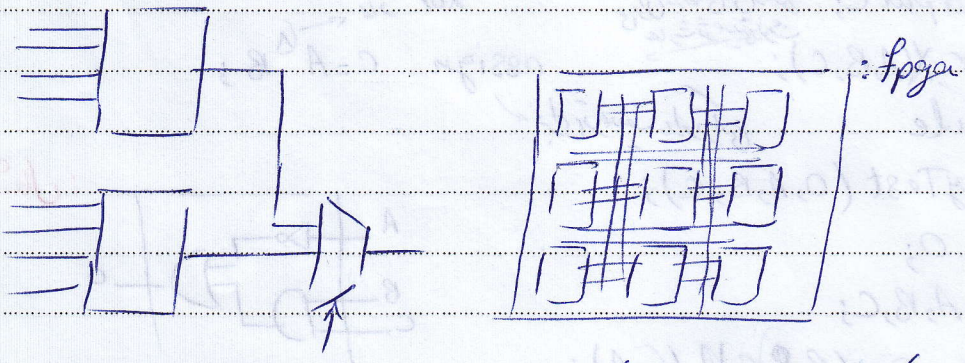
Year : Month : Day : ( )

```
SC_CTOR (counter) {
    SC_METHOD (inc_counter);
    sensitive << clock.pos();
}
}
```

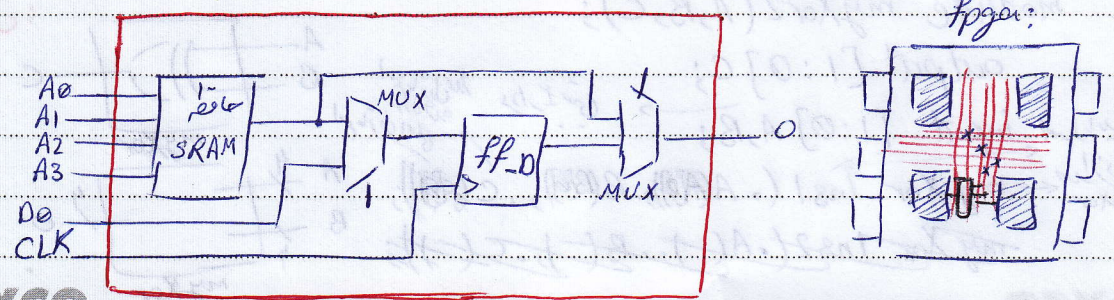
ساختار داخلی fpga به این صورت است که به جای گیت ها مدار مورد نظر روی حافظه ها پیاده می شود. مثلاً:




در داخل fpga هیچ گیتی وجود ندارد و آنچه موجود است بلوک های حافظه است که به تعداد زیاد در کنار هم قرار گرفته اند و امکان اتصالات آن ها به هم وجود دارد.




اگر به خروجی حافظه یک فلیپ فلاپ اضافه کنیم می توان مدارهای ترتیبی نیز داشت.



Subject: ٤

Year : Month : Day : ( )

در عمل بلوک های پایه fpga کامل تر از آنچه نشان داده شد هستند و شامل لیت های منطقی برای اعمال ماشه خرب و جمع هستند.

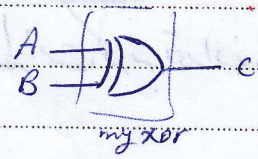
ضمن این که fpga علاوه بر بلوک های اصلی نشان داده شده بلوک های دیگری مانند ضرب کننده نیز وجود دارد.

برنامه نویسی به زبان Verilog:

کدهای Verilog با module شروع و با end module خاتمه می یابد.

```
module myXor (A,B,C);          ← اسم ماژول | پرت های ماژول | هر دستور را با ; پایان می دهیم
  input A;
  input B;
  output C;                    ← آرگومان های این دستور
  xor X(A,B,C);                ← عبارت توصیف کننده
endmodule                      ← اسم (برای صدا زدن بعدی)
```

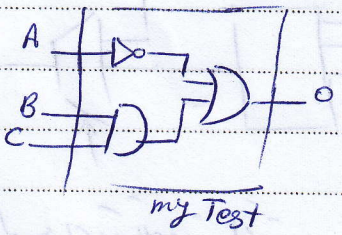
مثال:




xor یا ⊕

```
assign C = A ^ B;              ← برای توصیف مدار ترکیبی
```

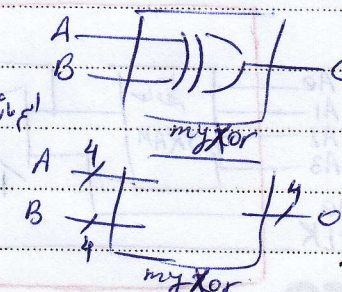
مثال:




```
module myTest (O,A,B,C);
  output O;
  input A,B,C;
  assign O = (B & C) | (~A);
endmodule
```

(& → and, | → or, ~ → not)

مثال:




```
module myXor2 (A,B,C);
  output [1:0] C;
  input [1:0] A,B;                 ← بردار 2 بیتی
  myXor Ins1 (.A(A[0]), .B(B[0]), .C(C[0]));
  myXor Ins2 (.A(A[1]), .B(B[1]), .C(C[1]));
```

اسم پایه یا شاخه ماژول myXor — اسم ماژول ساخته شده — اسم ماژول اصلی — صدا زدن ماژول myXor

Subject: ۵

Year : Month : Day : ( )

myXor Ins2(.A(A[1]), .B(B[1]), .C(C[1])); ← می‌توان پرت‌های ماژول صدا زده شده را اینگونه (در این صورت ترتیب باید رعایت شود): myXor Ins2(A[1], B[1], C[1])

endmodule

~~myXor4~~ module myXor4 (In1, In2, Out);

input [3:0] In1, In2;

output [3:0] Out;

myXor2 Ins1 (.A(In1[1:0]), ← می‌توان به خط بعد رفت

.B(In2[1:0]),

.C(Out[1:0]));

myXor2 Ins2 (.A(In1[3:2]), .B(In2[3:2]), .C(Out[3:2]));

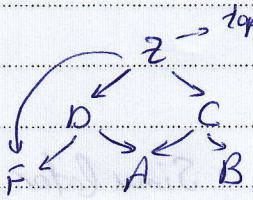
endmodule

به جای این حل می‌توان جواب را به صورت زیر به دست آورد:

```
module myXor4 (In1, In2, Out);
  input [3:0] In1, In2;
  output [3:0] Out;
  assign C = A ^ B;
endmodule
```

top module : ۱۷

ماژولی به ماژول دیگران را صدا می‌زند.



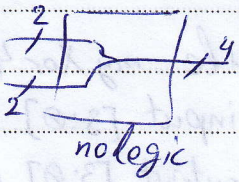

مثال: ضرب کننده‌ی بیتی

```
module Mult4(In1, In2, Out);
  input [3:0] In1, In2;
  output [7:0] Out;
  assign Out = In1 * In2;
endmodule
```

مثال: جمع ۴ بیتی:

```
module Adder4 (In1, In2, Out, C);
    input [3:0] In1, In2;
    output [3:0] Out;
    output C;
    wire [4:0] W;          // برای تعریف سیگنال جدید
    assign W = In1 + In2;
    assign C = W[4];
    assign Out = W[3:0];
end module
```

مثال:




```
module nologic (I1, I2, O);
    input [1:0] I1, I2;
    output [3:0] O;
    assign O = {I2, I1};
end module
```

این دستور I2 را الصاق هم ترکیبی دهد (Concatenation)

نکته:

در جمع ۴ بیتی می‌توان به جای تعریف سیگنال محلی W نوشت:

```
    assign {C, Out} = In1 + In2;
end module
```

Simulator: نرم افزارهایی که فرمت Verilog را فرمت و عمل شبیه‌سازی را انجام می‌دهد.

شبیه‌سازی برای تشخیص صحت عملکرد مدار ← Functional Simulation: ایده‌آل

در این مرحله اثرات غیرایده‌آل نداریم.

Subject: ۷

Year : Month : Day : ( )

مراحل طراحی:

طراحی → Functional Verification → Synthesis → Post Synthesis Simulation (شبیه‌سازی بعد از سنتز)

Post Route Simulation ← Place & Route ← (بخش سوم فوجر)

(ایرادات مدار را در عمل نشان می‌دهد)

نرم‌افزارهای شبیه‌ساز:

۱- Active HDL ← Aldec ... (تحت ویندوز) ... برای پروژه‌های بزرگ استفاده می‌شود.

۲- Model Sim ← Mentor ... (تحت Linux و ویندوز، یکپارچه‌ساز C-C)

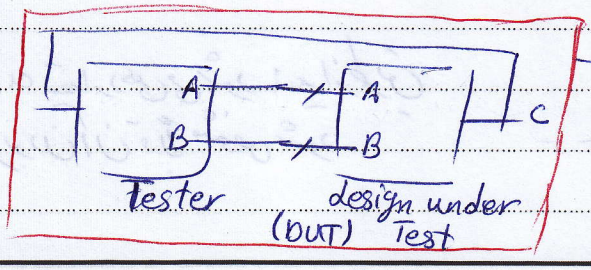
۳- IUS ← Cadence ... (تحت Linux) ... قدرتمندترین نرم‌افزار برای پروژه‌های بسیار بزرگ

Insieve unified Simulator

۴- VCS ← Synopsys

۵- System C

پس از نوشتن ماژول برنامه برای ارزیابی صحت آن در نرم‌افزار ماژول تست کننده می‌نویسیم که سیگنال‌های ورودی ماژول اصلی را تولید کند و آن را با یک top module صدا می‌زنیم.



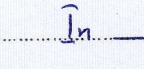
Subject: ۸

Year : Month : Day : ( )

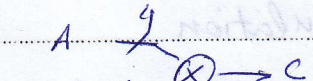
موضوع تحقیق:

physics Engine چیست؟ قابلیت‌های آن و توابعی که در آن پیاده شده است

برای مدارات ترکیبی:




assign Out = In;

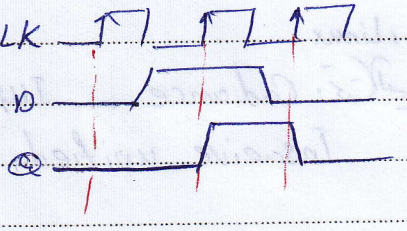


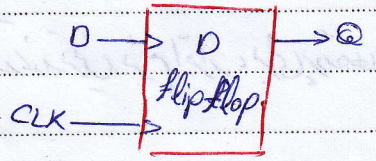

assign C = A & B;

→ Continous Assignment

می خواهیم مدارات ترتیبی را با fpga پیاده سازی کنیم:

مثال:




```
module dFF (D, Clk, Q);
    input D, Clk;
    output Q;
    always @ (posedge clk)
        Q <= D;
end module
```

کد کلی برای پیاده سازی مدارات ترتیبی

چون Q یک رجیستر است باید رجیستر بودن آن مشخص شود:

```
output Q;
reg Q;
always ---
```

متغیری که در always مقداردهی می شود باید از این دستور reg، رجیستر بودن آن قبلاً مشخص شود.

مثال:

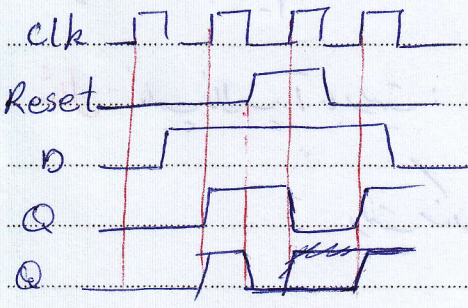
```
module tFF (Clk, Q);
    input Clk;
    output Q;
    reg Q;
    always @ (posedge Clk)
        Q <= ~Q;
end module
```

نکات:

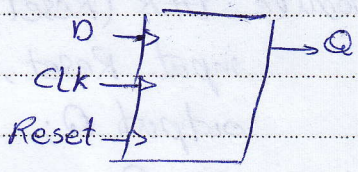
negedge ← posedge : اگر خواستیم تغییرات در لبه پایین رونده اتفاق افتد.

مدارات ترتیبی احتیاج به reset دارند.

مثال: فلیپ فلاپ D با ریست:




در حالت ریست سنکرون
در حالت ریست آسنکرون




ریست سنکرون:

```
module DFFR (D, Clk, Reset, Q);
    input D, Clk, Reset;
    output Q;
reg Q;
    always @ (posedge Clk)
        if (Reset)
            Q <= 0;
        else
            Q <= D;
end module
```

دستور if مانند زبان C می باشد

Subject: 10

Year : Month : Day : ( )

ریست آسنکرون:

```
module DffR (D, Clk, Reset, Q);
    input D, clk, Reset;
    output Q;
    reg Q;
    always @(posedge Clk or posedge Reset)
        if (Reset)
            Q <= 0;
        else Q <= D;
endmodule
```

نکته: هر متغیری فقط در یک always مقدار دهی می‌شود (در مثال بالا نباید برای Q دو بلوک always بنویسیم)

مثال: طلب فلاپ آ با ریست:

ریست سنکرون:

```
module tffR (Reset, clk, Q);
    input Reset, clk;
    output Q;
    reg Q;
    always @(posedge clk) or posedge Reset)
        if (Reset) Q <= 0;
        else Q <= ~Q;
endmodule
```

برای ریست آسنکرون

برای ریست active low ←

مثال: کانتر ۸ بیتی:

```
module Counter (Out, Clk, Reset);
    output [7:0] Out;
    input Clk, Reset;
    reg [7:0] Out;
```

Subject:

Year : Month : Day : ( )

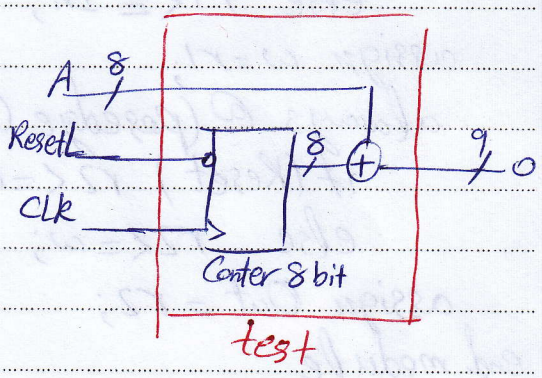
```
    always @ (posedge Clk)
        if (Reset)  Out <= 0;
        else  Out <= Out + 1;
endmodule
```

ارجو اسم متغیر مثبت یا منفی به مقدار قبلی اضافه شود.

```
module Counter (Out, Clk, Reset, value, updown);
    output [7:0] Out;
    input Clk, Reset, updown;
    input [3:0] value;
    reg [7:0] Out;
    always @ (posedge clk)
        if (Reset) Out <= 0;
        else if (updown)  Out <= Out + value;
        else Out <= Out - value;
endmodule
```

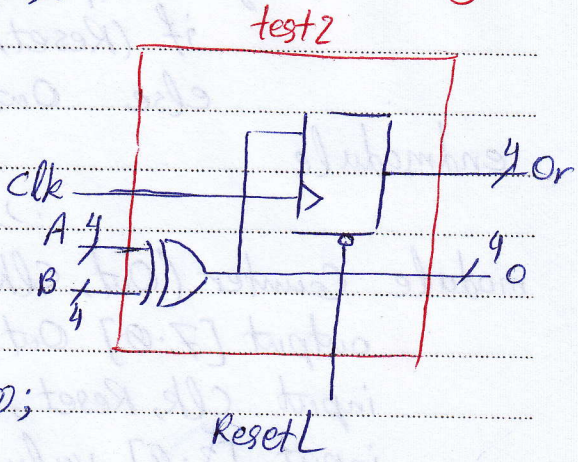
مثال: برنامه‌ای بنویسید که رو به رو را انجام دهد.

```
module test (O, A, Clk, ResetL);
    output [8:0] O;
    input [7:0] A;
    input Clk, ResetL;
    reg [7:0] r;
    always @ (posedge Clk)
        if (!Reset) r <= 0;
        else r <= r + 1;
    assign O = A + r;
endmodule
```



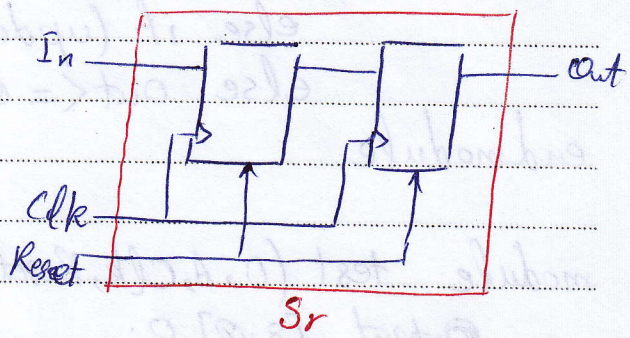
```
module test2(Or,O,Clk,ResetL,A,B);
    output [3:0] Or,O;
    input [3:0] A,B;
    input Clk,ResetL;
    reg [3:0] Or;
    assign O = A ^ B;
    always @(posedge Clk)
        if (!ResetL) Or<=0;
        else Or<=O;
endmodule;
```

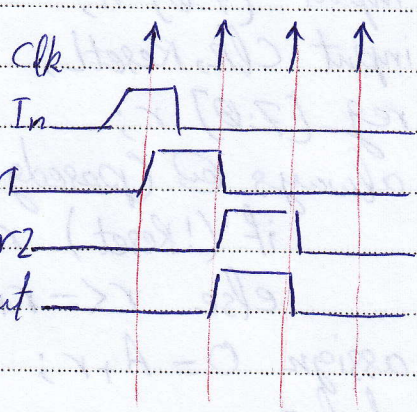
مثال:




```
module Sr(Out,In,Clk,Reset);
    output Out;
    input In,Clk,Reset;
    wire w;
    reg r1,r2;
    always @(posedge Clk)
        if (Reset) r1<=0;
        else r1<=In;
    assign w=r1;
    always @(posedge Clk)
        if (Reset) r2<=0;
        else r2<=w;
    assign Out=r2;
end module
```

مثال:

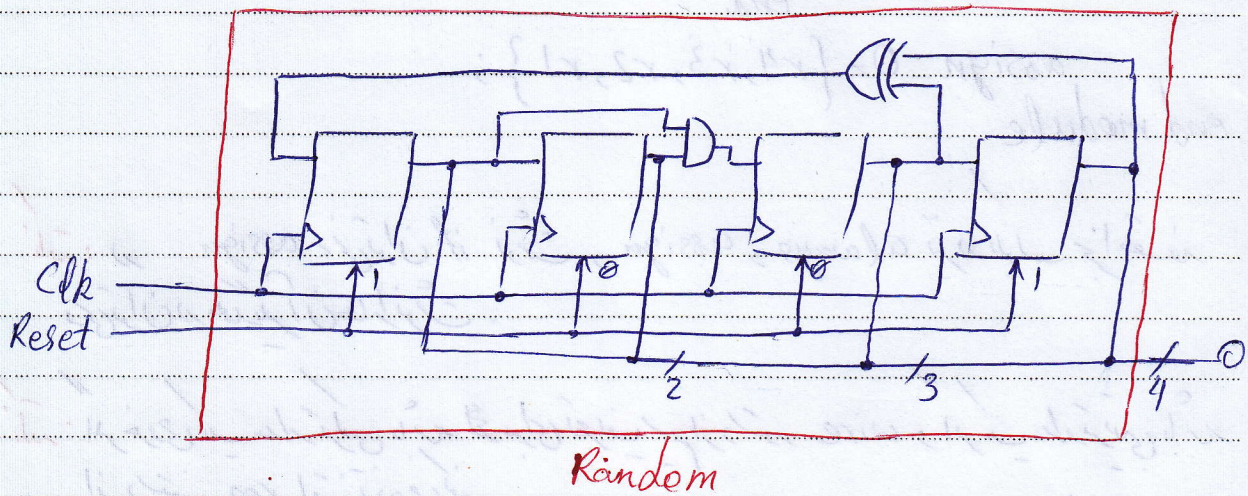


نکته: مثال قبل را می‌توان با دستور begin , end به صورت روبرو نوشت:

```
module Sr(Out, In, Clk, Reset);
    output Out;
    input Clk, Reset, In;
    reg r; Out;
    always @(posedge Clk)
        if (Reset) begin
            Out <= 0;
            r <= 0;
        end
        else begin
            r <= In;
            Out <= r;
        end
end module
```

مثال: برنامه‌ای بنویسید که مدار زیر را تحقق دهد.



Subject: ۱۴

Year : Month : Day : ( )

```
module Random (Clk, Reset, O);
    input Clk, Reset;
    output [3:0] O;
    reg r1, r2, r3, r4;
    always @ (posedge Clk)
        if (Reset) begin
                r1 <= 1;
                r2 <= 0;
                r3 <= 0;
                r4 <= 1;
            end
        else begin
                r1 <= r3 ^ r4;
                r2 <= r1;
                r3 <= r2 & r1;
                r4 <= r3;
            end
    assign O = {r4, r3, r2, r1};
end module
```

نکته: در always ، assign ، if نمی‌توان نوشت ، نمی‌توان در همدیگر آن‌ها را نوشت.

نکته: اگر خروجی یک مدار ترکیبی باشد ما برای تعریف سیم از دستور wire و اگر مدار ترتیبی باشد از دستور reg استفاده می‌کنیم.

Subject: 18

Year : Month : Day : ( )

## Numbers In Verilog :

عدد مبنای ده

```
assign  W = A + 1230;
```

برای مشخص کردن عدد باینری و سایر مبناها:

8'd124 ← تعداد بیت ، مبنای عدد ، خود عدد

هر سه نمایشگر عدد 124 هستند:

8'b 01111100

8'h 7C

مثال:

8'b 11110000 = {4'b1111, 4'b0} = {4{1'b1}, 4'b0}

= 8'hf0

برای نمایش مبنای ۸:

8'o17 ← Octal

هر سیم می‌تواند ۴ مقدار داشته باشد: ( 0, 1, z, x )

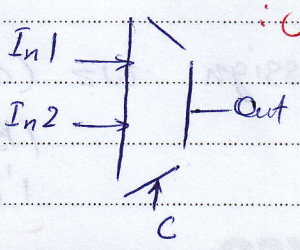
نامشخص

معمولا در مواقعی بوجود می‌آید که برای فلیپ فلاپ سیگنال Reset قرار ندهیم.

مثال:

```
module mux2(output Out, In1, In2, C);
  input In1, In2, C;  output Out;
    assign Out = (C) ? In1 : In2;
end module
```

شرط — شرط درست — شرط نادرست

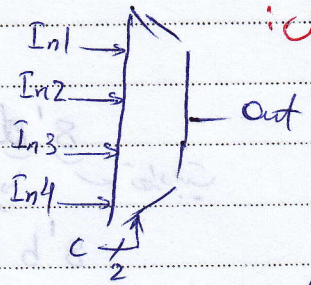



Procedual Continous Assignment

نحوه ی assign زدن:

assign خروجی = (شرط) ? S1 : S2;

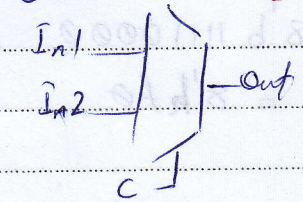
مثال:




```
assign Out = (C==0) ? In1 :
             (C==1) ? In2 :
             (C==2) ? In3 : In4;
```

هرگاه خواستیم در مدارهای ترکیبی از ساختار if استفاده کنیم نحوه ی assign زدن استفاده می کنیم.

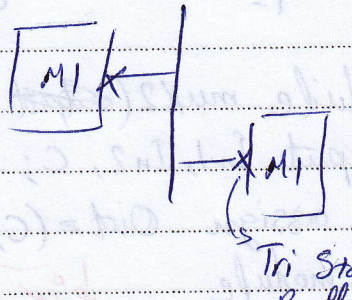
مثال: کدی برای توان به صورت رویدادی نوشت:




```
module mux2(Out, In1, In2, C);
  output Out;
  input In1, In2, C;
  reg Out;
  always @(In1 or In2 or C)
    if (C) Out<=In1;
    else Out<=In2;
endmodule
```

مثال: ابزار MUX سیم زیر را ادا بوکنید.

در این حالت در verilog باید در مواقع خاصی از حالت high Imp. استفاده کرد. در مواقعی باید M1 و در مواقعی M2 سیم را ادا بوکنید.




```
assign wz (C) ? 1 :
          (B) ? 0 :
          1'bz;
```

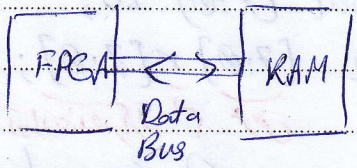
1'bz → high Impedance

Subject: IV

Year : Month : Day : ( )

بعضی ادوات احتیاج دارم به پورت ها هم خروجی و هم ورودی باشند. و بعضی ادوات خروجی و بعضی ادوات دیگر ورودی باشند:

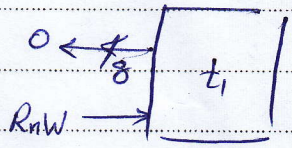
برای این منظور باید پورت را به صورت ورودی خروجی تعریف کنیم.




```
inout x;
```

مثال:

```
module t1(O, RnW);
    inout [7:0] O;   → می‌تواند هم output تعریف شود
    input RnW;
    assign O = (!RnW) ? 8'hff : 8'bz;
endmodule
```

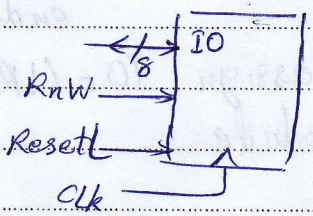



RnW=0 → O=0xff

RnW=1 → O=z

مثال:

```
module t2(IO, RnW, ResetL, clk)
    input RnW, ResetL, clk;
    inout [7:0] IO;
    reg [7:0] r;
    always @(posedge clk)
        if (!ResetL)
            r<=0;
        else begin
            if (!RnW) r<=IO;
        end
    assign IO = (!RnW) ? 8'bz : r;
endmodule
```



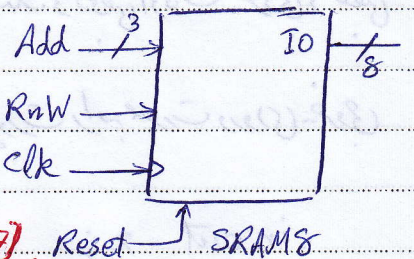

در هر لبه بالارونده: اگر RnW=0 داده‌ای که روی پورت IO قرار دارد را بخوان و ذخیره کن. اگر RnW=1 داده‌ای که ذخیره شده را در خروجی قرار ده.

Subject: ۱۸

Year : Month : Day : ( )

مثال :

```
module SRAM8 (Clk, Reset, RnW, Add, IO);
  input RnW, Clk, Reset;
  input [2:0] Add;
  inout [7:0] IO;
  reg [7:0] r[7:0];
always @ (posedge Clk)
  if (Reset) begin
      r[0] <= 0;
      r[1] <= 0;
      r[7] <= 0;
  else begin
    if (!RnW)
      r[Add] <= IO;
  end
assign IO = (!RnW) ? 8'bz : r[Add];
end module
```




تعریف آرایه دوبعدی

تعداد عناصر آرایه اسم عرض

دسترسی پیدا می‌کنیم r[0] یا r[7] ← تعداد عناصر آرایه اسم عرض

به جای تکراری می‌توان از دستور for استفاده کرد:

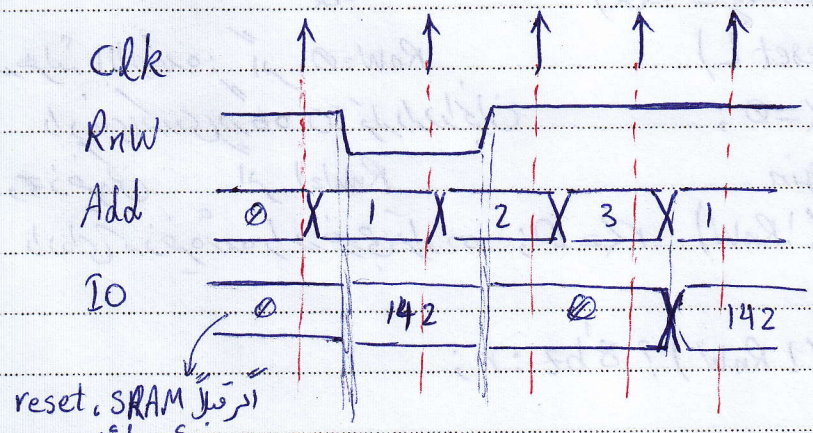
```
for (i=0; i<8; i=i+1)
  r[i] <= 0;
```

ساخت اقرار ← ساخت اقرار

باید i را قبلاً تعریف کرد:

```
integer i;
```

i فقط یک متغیر کمکی است و ظهور ساخت اقراری ندارد.




write به صورت سنکرون با لبه پایین ساعت صورت می‌گیرد ولی read ربطی به لبه ندارد، و آسنکرون می‌باشد.

PAYCO

Subject: 19

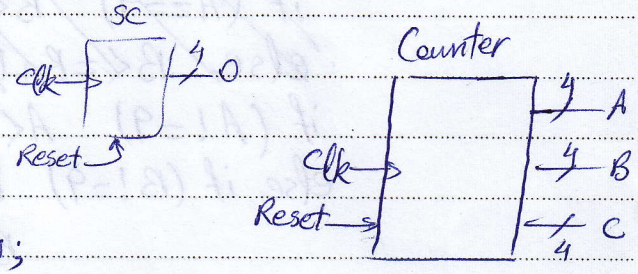
Year : Month : Day : ( )

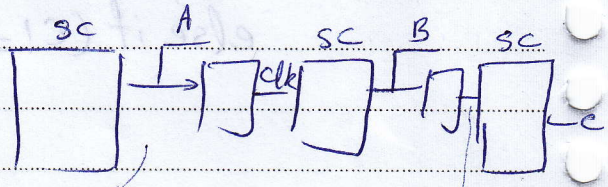
نکته: فقط وقتی حق استفاده از دستور for را داریم که بخواهیم یک تحت اقرار را تکرار کنیم. بنابراین استفاده از for کاملاً محدود است.

سؤال: یک کانتر طراحی نمید که در مبنای ۱۰ بشمارد

ماژول اول

```
module Sc(O,Clk,Reset);
    output [3:0] O;
    input Clk,Reset;
    reg [3:0] O;
    always @ (posedge Clk)
        if (Reset) O<=0;
        else begin
            if (O==9) O<=0;
            else O<=O+1;
        end
endmodule
```




```
assign OC1=((In==9)&&Clk)?1:0;
assign OC2=((In==9)&&(B==9)
    && Clk)?1:0;
```

ماژول دوم

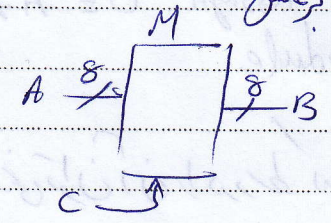
```
module Counter (A,B,C,Reset,Clk);
    input Reset, Clk;
    output [3:0] A,B,C;
    reg [3:0] A,B,C;
    always @ (posedge Clk or posedge Reset)
        if (Reset) begin
            A<=0;
            B<=0;
            C<=0;
        end
```

Subject: 10

Year : Month : Day : ( )

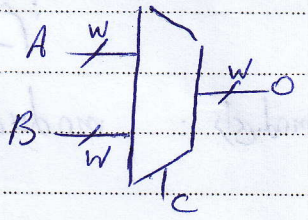
```
else begin
    if((B!=9)&&(A!=9)&&(C!=9))
        C<=0;
    else C<=C+1;
    if (A==9) B<=0;
    else B<=B+1;
    if (A!=9) A<=A+1;
    else if (B!=9) begin
                       B<=B+1;
                       A<=0;
                       end
    else if (C!=9) begin
                       C<=C+1;
                       A<=0;
                       B<=0;
                       end
    else begin
             A<=0;
             B<=0;
             C<=0;
             end
endmodule;end
endmodule
```

سوال: اگر C=1 ، B به B متصل شود و اگر C=0 ، برعکس




```
module M(A,B,C);
    inout [7:0] A,B;
    input C;
    assign B=(!C)? 8'bz : A;
    assign A=(C)? 8'bz : B;
endmodule
```

مثال: mux با عرض متغیر:




```
module mux(O,A,B,C);
    output [W-1:0] O;
    input [W-1:0] A,B;
    assign O=(C)? A:B;
endmodule
```

نکته:

برای تعریف عدد ثابت در Verilog از define استفاده می‌کنیم:

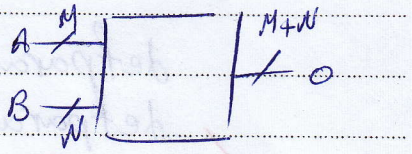
در کد بالا:

```
`define W 32
```

کاربرد:

برای استفاده از sub mod. با عرض داده‌های مختلف می‌توان از define استفاده کرد و sub mod. را عمومی نوشت.

مثال: ضرب‌کننده




```
`define M 2
`define N 3
`define W `M+`N
module mult(O,A,B);
    input [`M-1:0] A;
    input [`N-1:0] B;
```

```
output [W-1:0] O;
assign O = A*B;
endmodule
```

نکته: define فقط یکبار و از ابتدا استفاده می‌شود و نمی‌توان آن را در وسط کد تغییر داد.

define از top mod. برای استفاده از sub mod. با عرض داده مختلف هم دیگر را استفاده می‌کنیم.

در این موارد از parameter استفاده می‌کنیم:

مثال: حالتی یکبار با عرض متغیر:

برای sub mod. :

```
module mux (O, A, B, C);
parameter W=3;
output [W-1:0] O;
input [W-1:0] A,B;
input C;
assign O = (C)? A : B;
endmodule
```

برای top mod.

```
module top ____
mux #(5) myMux (.A( ) ____ );
mux myMux2 ( ____ );
```

مقدار پارامتر را تغییری دهد ← مقدار پارامتر همان مقدار اولیه ۳ است

می‌توان از تکنیک زیر استفاده کرد:

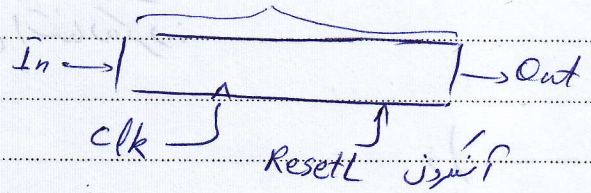
```
defparam myMux2.W = 5
defparam myMux.W = 4
```

با این دستور می‌توان مقدار پارامتر را تعریف کرد.

Subject: ۲۳

Year : Month : Day : ( )

سوال - یک شیفت رجیستر طراحی کنید. (DEPTH باطول DEPTH)




```
module SR(In, clk, Out, ResetL);
    input In, clk, ResetL;
    output Out;
    parameter DEPTH=8;
    reg [DEPTH-1:0] r;
    always @(posedge clk or negedge ResetL)
        if (!ResetL)
            r<=0;
        else
            r<={r[DEPTH-2:0], In};
    assign Out = r[DEPTH-1];
endmodule
```

اگر خواستیم ماژول بالا را صدا بزنیم:

```
SR #(4) SRIns (.Out(), .In(), .clk(), .ResetL());
```

اگر در یک ماژول چند پارامتر داشته باشیم می‌توان به صورت زیر آن را صدا زد.

```
#(4,5)
#(.DEPTH(4), .WIDTH(5))
```

برای صدا زدن یک ماژول پارامتری می توان از دستور defparam بلافاصله پس از صدا زدن ماژول استفاده کرد:

```
SR    SRIns ( _____ );
defparam SRIns.DEPTH = 4;
```

از نظر کدنویسی این روش منظم تر است.

در برنامه نویسی Verilog به جای دستور if می توان از case استفاده کرد.

```
if (Cond1) act1;
else if (Cond2) act2;
else if (Cond3) act3;
else {act N};
```

اگر شرایط بالا برقرار نبود

```
case (variable)
2'b00 : r<=1;
2'b01 : r<=2;
default : r<=N;
endcase
```

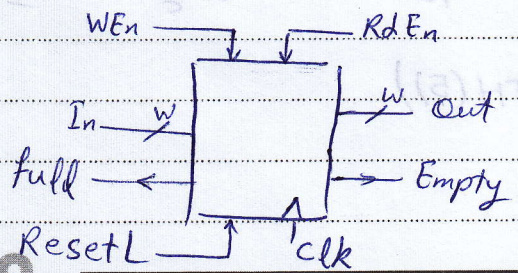
اگر به ازای یک شرط چند کار خواستیم انجام دهیم باید از begin , end استفاده کنیم.

if case ترجمه شد در بلوک always استفاده می شود.

اگر به ازای چند شرط خواستیم یک کار را انجام دهیم، آنها را با بیت بر هم می نویسیم.

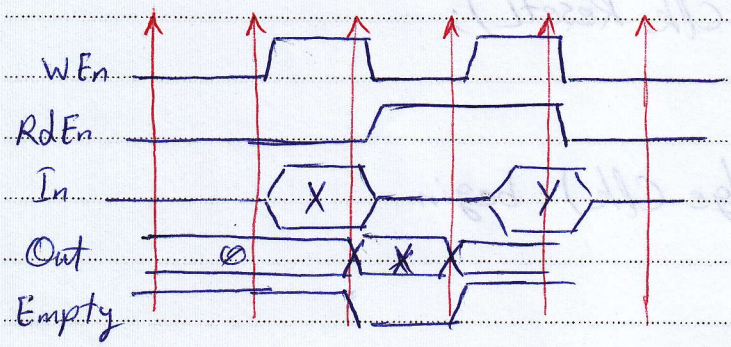
مثال: یک fifo طراحی کنید.

عمق fifo، DEPTH، عرض آن W است.



Subject: ۲۵

Year : Month : Day : ( )




قسمتی از کد مربوط به آن به صورت زیر است:

```
reg [W-1:0] r[DEPTH-1:0];
reg [D:0] wpointer, rpointer;
reg [C:0] Count;
cas ({RdEn, WEn})
    2'b10: Count <= Count-1;
    2'b01: Count <= Count+1;
    default: Count <= Count;
endcase

cas ({RdEn, WEn})
    2'b10: begin if (!Empty) Count <= Count; end
    2'b01: if(!Full) Count <= Count+1
    default: Count <= Count;
endcase
```

نمایندهٔ مربوط به خواندن و نوشتن

برای مدیریت وضعیت خالی

باید شرط حالی در پرون حذف شود

تفاوت = و <= در Verilog:

اگر یک شیفت رجیستر داشته باشیم:

Subject: ۲۴

Year : Month : Day : ( )

```
module SR(In, Out, Clk, ResetL);
    |
reg a,b,c;
  always @(posedge Clk) begin
    |
      a<=In;
      b<=a;
      c<=b; end
  assign Out=c;
```

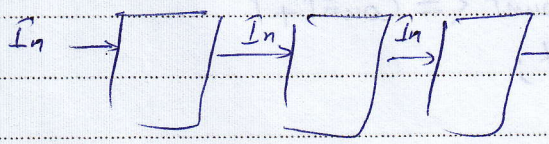
عبارت (<=) اجرای دستور بعدی را بلوک می کند و باعث می شود اجرای هر دستور عملاً هیچ زمانی نگیرد. یعنی در عبارت بالا هر سه دستور در یک لبه بالارونده پالس ساعت اجرا می شوند.

اگر کد به صورت زیر بود:

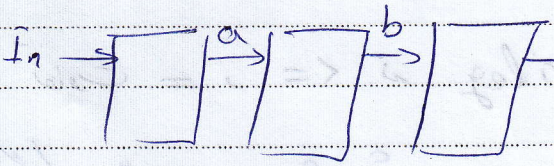
```
a = In;
b = a;
c = b;
```

علامت (=) از سیگنال نمونه برداری می کند و همان لحظه تخصیص می دهد.

اگر کد با = نوشته می شد سیگنال In مستقیماً به Out می رفت




= blocking assignment




<= nonblocking assignment

Subject: ۲۷

Year : Month : Day : ( )

نکته: در Verilog دستوراتی که در بلوک begin, end نوشته می شوند بترتیب هم اجرا می شوند.

بلوک های دیگری نیز مانند fork, join وجود دارند که می توان به جای begin, end استفاده کرد:

always @( ——— )

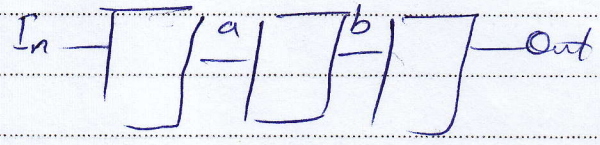
fork

join

در بلوک fork, join دستورات همزمان اجرا می شوند.

بنابراین اگر در بلوک fork, join به جای (=) از (<=) استفاده شود چون دستورات باهم اجرا می شوند می توان همان شیفت رجیستر را پیاده کرد.

```
fork
    a<=In;
    b<=a;
    c<=b;
join
```



Subject: ۲۸

Year : Month : Day : ( )

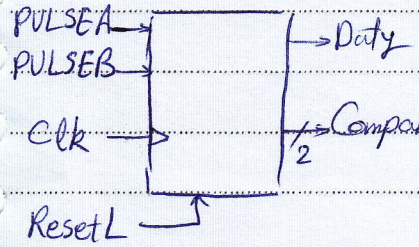
Subject: Y9

Year : Month : Day : ( )

Subject: ۲۰/

Year : Month : Day : ( )

سؤال: پالس های A و B سیگنال های متناوب هم فرکانس




خروجی Duty 1 باشد Duty Cycle پالس A 50٪ باشد

خروجی Compare 1 باشد اگر DC پالس A از B بیشتر باشد

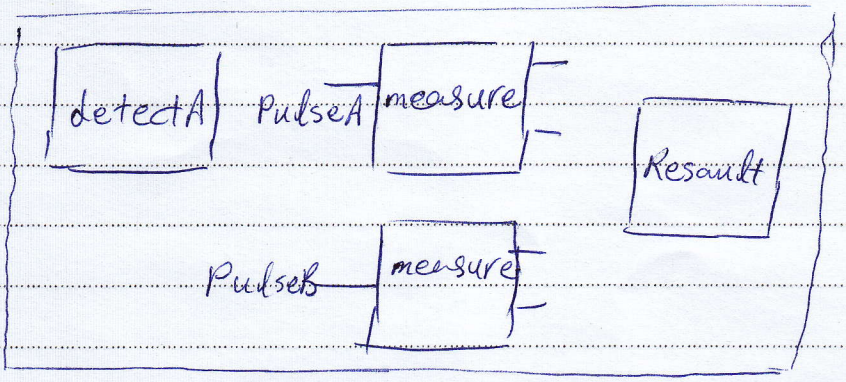
2 اگر B از A بیشتر باشد

3 اگر مساوی باشند.

موضوع تحقیق:

استفاده های fpga در پروژه های lab On a Chip .

برای نوشتن برنامه سؤال بالا به چند ماژول احتیاج داریم




ماژول detectA: تعیین لبه بالارونده پالس A:

```
module Detect (Clk, ResetL, Pulse, edge);
    input Clk, ResetL, Pulse;
    output edge;
    reg r1, r2;
    always @ (posedge Clk or negedge ResetL)
        if (!ResetL) begin
```

Subject: 21

Year : Month : Day : ( )

```
        r1 <= 0;
        r2 <= 0;
    end
    assign edge = (r1 && !r2) ? 1 : 0;
endmodule
```

```
    else begin
        r1 <= PULSE;
        r2 <= r1;
    end
```

: measure 모듈

```
module measure (Clk, ResetL, Pulse, Up, Down, Edge);
    input Clk, ResetL, Edge, Pulse;
    output [31:0] Up, Down;
    reg [31:0] Up, Down;
    always @ (posedge Clk or negedge ResetL)
        if (!ResetL) begin
            Up <= 0;
            Down <= 0;
        end
        else begin
            if (Pulse)
                Up <= Up + 1;
            else
                Down <= Down + 1;
        end
end module
```

```
        else begin
            if (Edge) begin
                Up <= 0;
                Down <= 0;
            end
```

Subject: ٣٢

Year : Month : Day : ( )

```
module Resault (Clk, ResetL, Edge, Duty, Compare, wAU, wBU, wAD);
    input Clk, ResetL, Edge ;
    input [31:0] wAU, wBU, wAD;
    output Duty; output [1:0] Compare;
    reg Duty;
    reg [1:0] Compare;
    always @ (posedge Clk or negedge ResetL)
        if (!ResetL)
            Duty <= 0;
        else begin
            if (Edge) begin
                if (wAU == wAD)
                    Duty <= 1;
                else
                    Duty <= 0;
                end
            end
    always @(posedge Clk or negedge ResetL)
        if (! ResetL)
    e       Compare <= 0;
        else begin
            if (Edge) begin
                if (wAU > wBU)
                    compare <= 1;
                else if (wAU < wBU)
                    compare <= 2;
```