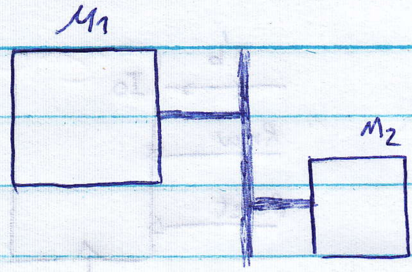
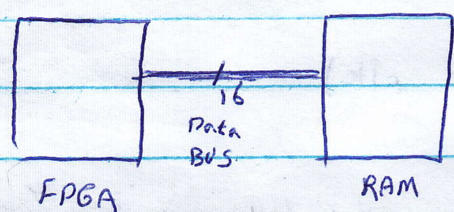
وقتی یک Bus مشترک داریم:
در هر لحظه فقط یکی از M_1 یا M_2 باس را drive می‌کند که احتیاج به بافر سه حالته داریم.




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

→ یعنی گاهی اوقات که هیچ کدام از شرط‌ها برقرار نشد مدار به صورت high ampedance می‌رود!

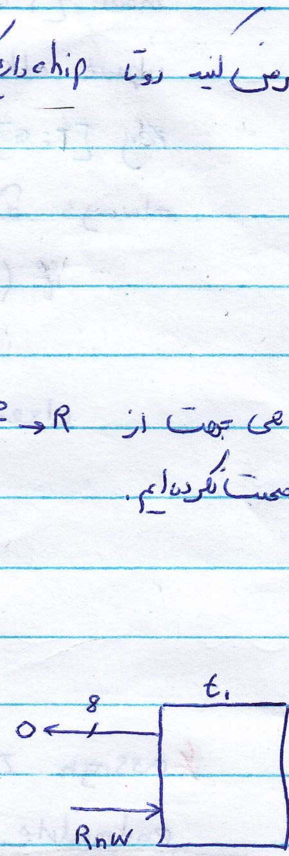
فرض کنید دو تا chip داریم که به هم متصل‌اند:



گاهی جهت از F → R ، گاهی F ← R . تا الان درمورد پورتی که هم ورودی باشه هم خروجی صحبت نکرده‌ایم.

* inout * برای تعریف هم ورودی هم خروجی

if RnW == 0 → O = 8'hff . این ماجول کارش این است:
if RnW == 1 → O = high ampedance

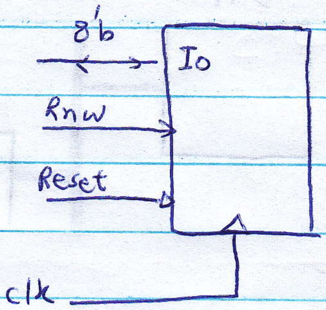



```
module t1(O, Rnw);
  inout [7:0] O;
  input Rnw;
  assign O = (!Rnw) ? 8'hff : 8'bz;
endmodule
```

high ampedance

8'bz → این یعنی همه بیت‌ها را high amp می‌کنیم.

در هر لبه‌ی بالا رونده‌ی ساعت اگر Rnw=0 بود، داده‌ای که روی درگاه IO قرار دارد را بخوان و ذخیره کن.
در لبه دیگر اگر Rnw=1 بود داده‌ای که ذخیره کرده‌ای را در خروجی قرار بده.



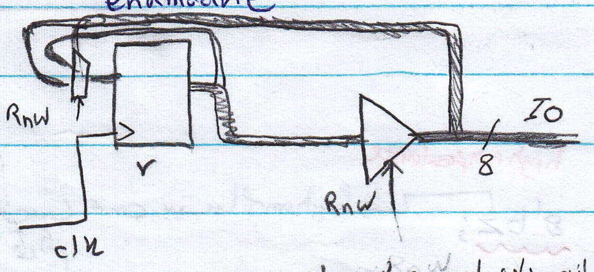
پس باید داخل ماجول یک رجیستر تعریف کرد که بتوان روی آن ذخیره کرد و سپس بازیابی کرد.

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

توجه کن که فقط در بلوک always باید رجیسترها مقدارشون تغییر کنه.

در این مدار نباید هم زمان Ram و FPGA روشن شوند چون می‌سوزد.

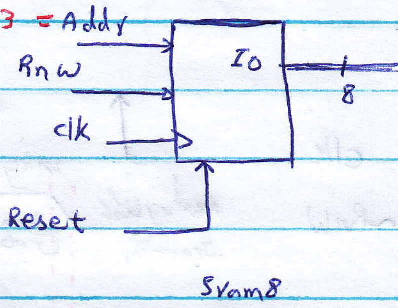
* این عبارت یعنی اگر منطقی که داری



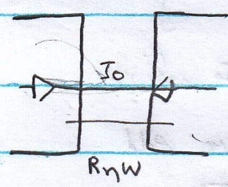
وقتی Rnw=0 است یه نفر از خارج داده‌ای به ما می‌دهد و IO درایو نشود.

وقتی Rnw=1 است می‌خواهیم از بیرون مقدار را بخوانیم و داده باید روی باس بیاید.

Addr = 3 بیت چون 8 بیت داریم که می‌توان با سه بیت آدرس دهی کرد.




```
module sRam8(        );
  inout [7:0] Io;
  input [2:0] Addr;
  input clk, Reset, RnW;
  reg [7:0] r1, r2, r3, r4, ..., r8;   ← آرایه دو بعدی →   reg [7:0] r [7:0];
                                                          عرض    تعداد عناصر
  always @(posedge clk)
     if (Reset)
        begin
          r[0] <= 0;   ← توجه کن که فقط به کل 8 بیت با این طریق دسترسی داری.
          r[1] <= 0;
          r[2] <= 0;
          ...
          r[7] <= 0;
        end
     else
        begin
          if (!RnW)
             r[Addr] <= Io;
        end
  assign Io = (!RnW)? 8'bz : r[Addr]
```



در واقع این RnW تعیین می‌کند که کدام chip ، Io را درایو کند. به این علت از دو بافر برای ورودی و خروجی جدا استفاده می‌کنیم چون محدودیت پایه داریم.

این دستورها را می‌توان این‌گونه نوشت:

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

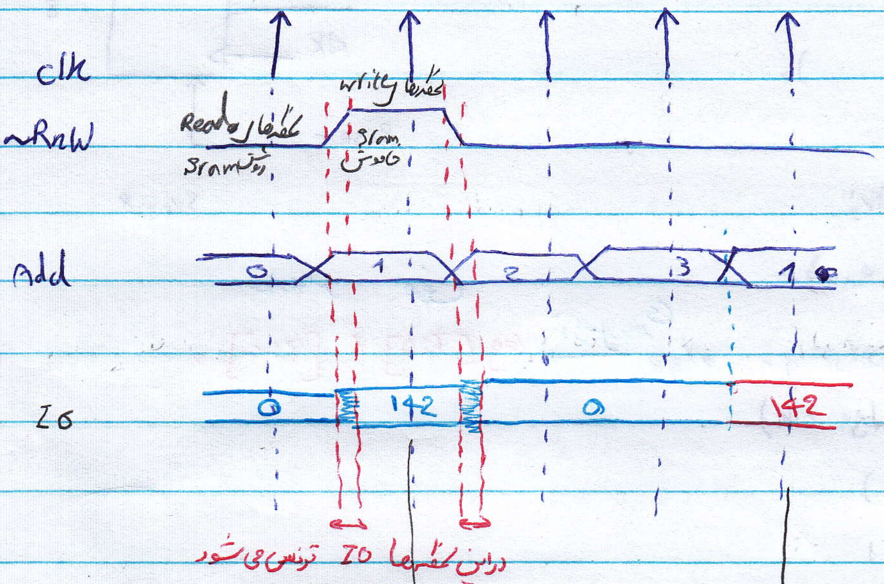
این خود سخت افزار است.

8'bz: یعنی داریم رایت می‌کنیم پس کاری با Io نداریم.
r[Addr]: وگرنه آنچه درون r است را در خانه r شماره addr ذخیره کن.

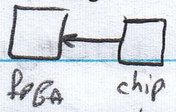
توجه کن دستور for در HDL زمانی استفاده می‌شود که سخت افزاری کاملاً یکسان تکرار شود. با for در C متفاوت است. یعنی for در یک سیکل زمانی انجام نمی‌شود!

برای تعریف i از کلمه کلیدی integer استفاده کرد و ظهور سخت افزاری ندارد و اندازه‌ی آن 32 بیت.

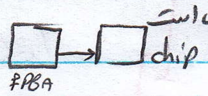
شکل موج :



در این لحظه مقدار 142 را در خانه شماره 1 Add save می‌شود



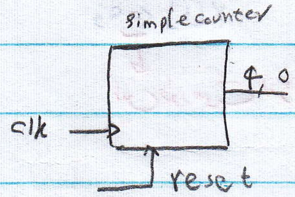
در این لحظه محتوای خانه 1 که در قبل 142 شده است 142 شده است



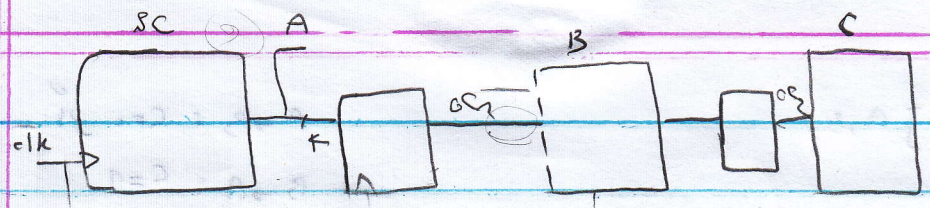
توجه کنید read آسون است اما write مشکله.

! به هر متغیری در داخل always که می‌خواهی مقداردهی کنی باید reg تعریف کنی.

تمرین 3 : counter سه رقمی که تا 10 بشمارد.




```
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 $oc_1 = ((A==9) \&\& clk) ? 0 : 1;$

assign $oc_2 = ((A==9) \&\& (B==9) \&\& clk) ? 1 : 0;$

```
module counte (A, B, C, clk, Reset);
  output [3:0] A, B, C;
  input  clk, Reset;
  reg [3:0] A, B, C;
  always @(posedge clk or posedge reset)
    if (reset) begin  A <= 0;
                      B <= 0;
                      C <= 0;
               end
    else  begin
          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
                      B <= 0;
                      A <= 0;
               end
          end
endmodule
```

always ← يا بت

ريست اسنكرون

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

اگر A→B : C=0

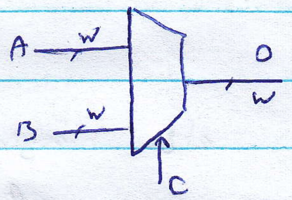
B→A : C=1

```
assign B=(C) ? 8'bz : A;
```

! NOTE: سعی کنید clk جدید نسازید. یا روی clk عمل logic انجام ندهید. هر جا احتیاج به چنین چیزی بود یک Enable بساز و بعد با clk ، and کن.

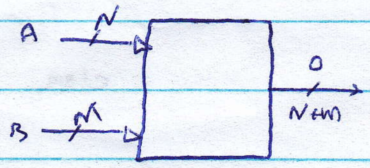
– می‌خواهیم مالتی پلکسری طراحی کنیم که خصوصیات آن قابل تغییر باشد:

```
`define W 32;
module mux(O,A,R,B,C);
  output [`W-1:0]O;
  input [`W-1:0] A,B;
  assign o =(c)? A,B;
endmodule
```



! ~~فقط با عوض کردن مقدار define می‌توان زیر ماژول‌های مختلف با عرض‌های متفاوت استفاده کرد~~

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



! نمی‌توانی با حالت بالا دو ماژول بالا را با دو عرض متفاوت انجام دهی:

اگر بخواهی هر بار که برنامه را صدا می‌کنی عرض متفاوت باشه یعنی در صدا کردن آن را تغییر دهی داریم:

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

```
module top ..
    mux #(مقدار پارامتر) mymux(.A(), ... )
```

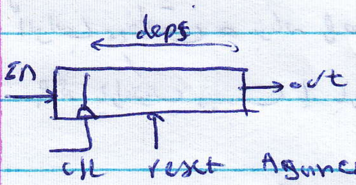
mux: اسم ماجول — #(مقدار پارامتر): پارامتر — mymux: Ins اسم — (.A(), ...): اتصال پورت

اگر قسمت مقدار پارامتر را نگذاری همان مقدار تعریف شده در mux است.

```
#(.W
```

```
mux mymux2 (پورت);
    defparam mymux2.W = 5;
    :
```

mymux2: Instance name



Aguneren

کوییز 4: یک شیفت register طراحی کنید متغیر deps

```
module SR(out, In, clk, regester L);
    parmeter Depth = 8;
    output out;
    Input In, clk, reset;
    reg [Depth-1:0] r;
    always @(posedge clk or negedge Reset)
```

```
if (! resetL)
    r <= 0;
else
    r <= {r[Depth-2 :0], In};
assign out = r[Depth-1];
endmodule
```

شیفت می‌دهی از خانه صفر تا یکی مونده به آخری

با for ⇐

```
integer i;
always @ (posedge clk or negedge reset)
    if (! resetL)
        r <= 0;
    else begin
        for (i=1; i<depth; i = i+1)
            r[i] <= r[i-1];
        r[0] <= In; end
assign out = r[Depth-1];
endmodule
```

اگر اینجا می‌نوشتی در برنامه اصلی (ازش) SR SRIn (.out(), .In(), .clk(), .reset()) اینجا با

SR که پارامتریک نیست یا پارامترها نمی‌خواهی عوض کنی.

حال اگر بخواهی عوض کنی: SR #(4) SRIn(.out(), .In(), .clk(), .reset());

#(4, 5) —(اگر ترتیب پارامترها یادت نمی‌آید)→ #(.Depth(4), .W(7))

اگر چند تا پارامتر بود به ترتیب اولین پارامتر و بعدی دومین پارامتر

- روش دوم این است که اگر خواستی شکل قبلی ماژول را جدید بنویسی :

```
SR   SRIn ( .out( ), .In( ), ..)
defparam   SRIn . Depth=4 ;
```

همان اسم اینستنس گذاری .

- داخل بلوک begin end می توانی If و elseif داریم : (وخارج آن نمی توانی)

```
begin
   if (cond1) act1;
   else if (cond2) act2;
     ;
   else
      act f;
end
```

act f; → بهتر است else آخر باشد تا خودش latch نگذارد / نزند .

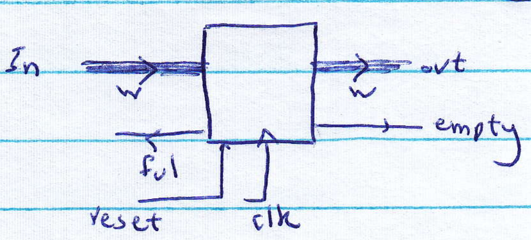
اینجا هم می توانی با case بنویسی :

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

2'b00 : r <= 1; → اگر به ازای یک شرط چند عمل قرار بود انجام شود باید begin end بگذاری .

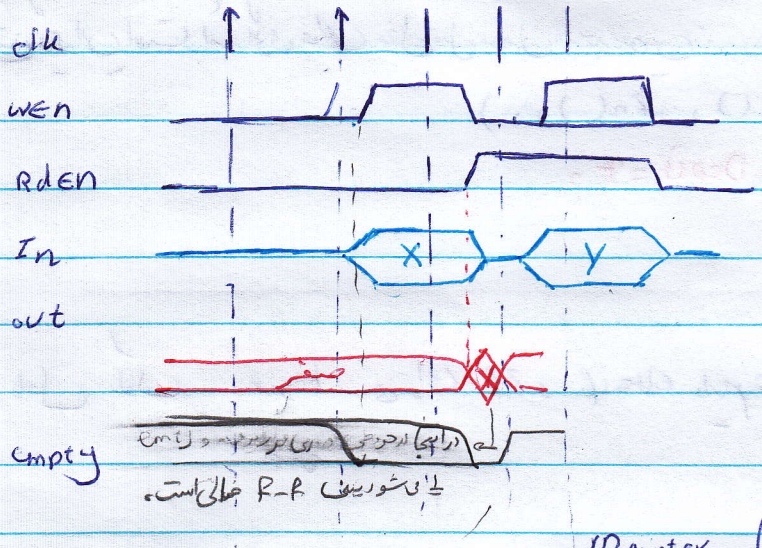
باید case در always باشد !

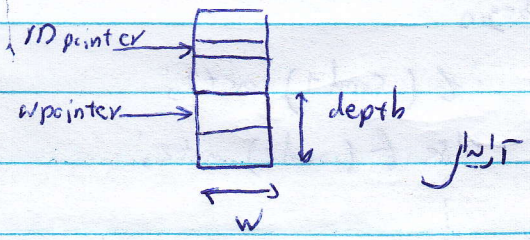
* یک F-F طراحی کنید : عرض F-F ، w متغیر است .



با هر لبه بالارونده ساعت که wen یک بود In را روی F.F save کند .

با هر لبه ساعت که rd en یک بود داده‌ای که در F.F بود در out رود .





آرایه‌ای

تعریف آرایه reg [w-1:0] r[Depth-1:0];

reg [D:0] wpointer و rpointer;

reg [D':0] count; ← تعداد خونه‌ای که براست یا نگذی دارد.

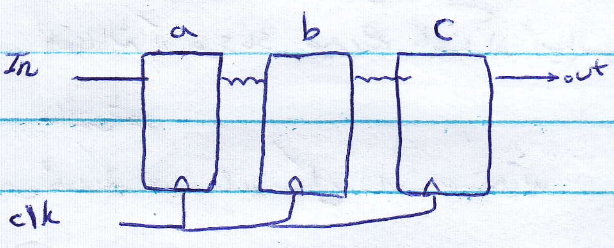
```
Case ( {RdEn, WEn} )
    2'b 10 : count <= count - 1;   ← کلک است → begin if(!empty) count<=count-1; end
    2'b 01 : count <= count + 1;   → begin if(!full) count<=count+1; end
    default : count <= count;
endcase
```

تفاوت <= و = :

اگر یک shr داشته باشیم که سه بیتی است در هر حالت بررسی می‌کنیم :

```
module SR ( In, out, clk, resetL );
    :
    :
reg a, b, c;
always @ ( posedg clk ) begin
        a <= In;
        b <= a;
        c <= b;
                end
assign out = c;
```

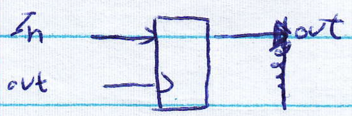
اگر نوشته شود نوع <= سنتز کند :



nonblocking <= : اجرای دستور بعدی را بلوک نمی‌کند. مقدار الان را نمونه برداری کن و چند هم تاخیر بعد تخصیص بده، یعنی در آن لحظه نمونه برداری می‌کند و بعد تخصیص می‌دهد که یعنی همان موقع تخصیص نمی‌دهد.

اگر به جای <= از = استفاده کرده بودیم : علامت مساوی blocking assignment ، از سیگنال نمونه برداری می‌کند و همان موقع تخصیص می‌دهد. مدار به شکل زیر می‌شود :

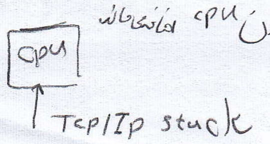
عملاً a و b کاری انجام نمی‌دهد که معمولاً منظورمون <= است!



اعمالی که در بلوک begin end هستند پشت سرهم انجام می‌شود

یک بلوک دیگر هم داریم fork join

اگر C ما باشه که کدهای Tcp ما انجام دهد بجای cpu خیلی عالیه



پیاده‌سازی سخت‌افزاری TcpIIp است.

hardware implimention

که معادل always می‌آید <=

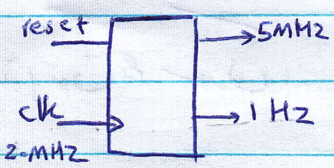
```
fork
    a = In;
    b = a;
    c = b;
join
```

این‌ها shr یعنی همه با هم انجام می‌دهد.

! بهتری از ها begin end بلوک‌ها واژه <= استفاده کنی ✓

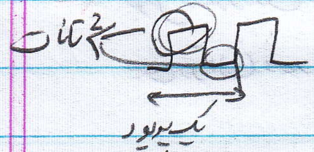
✱ حل تکلیف تحویلی 8

5) هر 4 سیکل clk یک سیکل برای 5MHz می‌شود. پس اگر
کانتری باشه که هر دو تا لبه‌ی بالا رو ثابت کند یک پالس
ساخته می‌شود.



پس نگاه می‌کنیم نسبت ها به سیکل ساعت بدست می‌آید

$\frac{20}{5} = 4/2 = 2 - 1 = 1$



تقسیم بر دو به علت این است که هر 2 تای آن باید ثابت شود!! مثال
یک جور اوضح شماری

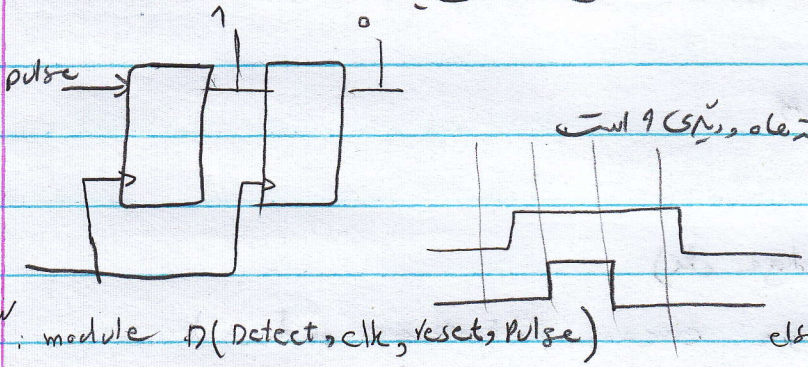
```
module M (o, clk, reset);
  output o;
  input clk, reset;
  parameter TH = 1;
  reg [31:0] r;
  reg a;
  always @ (posedge clk)
     if (reset) begin
```

با دیباگ صدا کردن برنامه و مقداردادن به th می‌توان
5، 1 و 2 تولید کرد.

```
r<=0;
a<=0;
end
else begin
  if (r == TH)
      r<=0;
  else
      r<=r+1;
  if (r == TH/2)
    a<=~a;
  end
assign o = a;
endmodule
```

مهم مهم مهم!

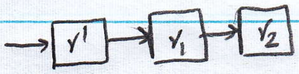
- توجه کن که پریود پالس کمتر از ساعت است و آن را می‌توان دید.



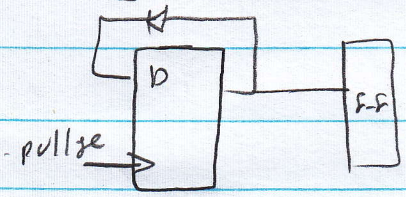
راه‌حل مهم است که یکی از رجیستر ها و دیگری ۹ است

```
module D(Detect, clk, reset, Pulse)
  input pulse, clk, reset;
  output Detect;
  reg r1, r2;
  always @(posedge clk)
    if (reset) begin r1<=0;
                     r2<=0;
               end
    else begin
      r1 <= pulse;
      r2 <= r1;
    end
  assign detect = r1 & (~r2)
endmodule
```

گاهی مهم نیست که همه پالس‌ها رو دیتکت کنه یا یه پالس نبیندش. و ممکن است پالس در زمان setup که فلیپ‌فلاپ آمد، که این باعث نوسان می‌شود. یعنی حالا که Y_1 و Y_2 داریم اگر قبلاً پالس رو رجیستر کنی این پدیده رخ نمی‌ده!

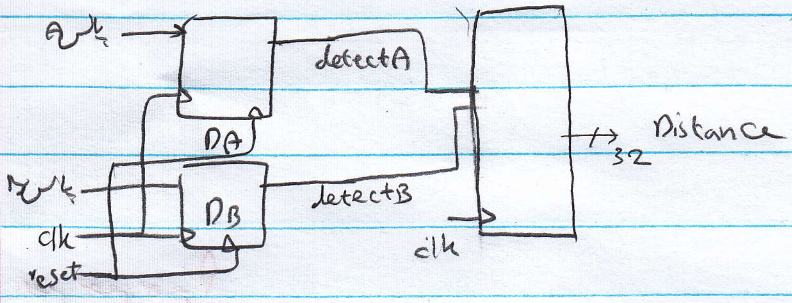


اگه از posedge pulse استفاده کنی باید خروجی رو به یک رجیستر بدهی که با پالس ساعت تغییر کنه



فقط توجه کن که این طور که می‌نویسی تو late می‌شوی.

۱۲ از سؤال قبل استفاده می‌کنیم:




```
module Dist (Distance, DetectA, DetectB, clk, reset);
    ⋮
  reg [31:0] Distance;
  reg en;
  always @(posedge clk)
    if (reset) begin en<=0; distance<=0; end
    else begin
      if (detectA || detectB) en <= ~en;
      if (en) Distance <= Distance + 1;
  end
endmodule
```

برای نگه داشتن حال شمارش است!

سپس این ماژول و ماژول برنامه قبلی رو دوبار صدا کنی.

18

L=255

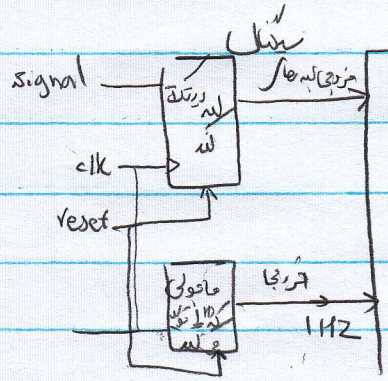
L=0

L=1

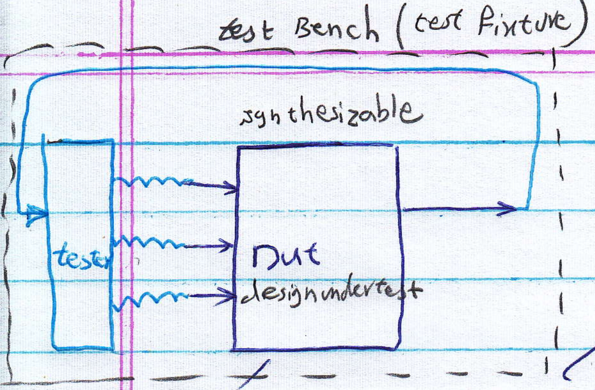
L=10

```
module pwm( o,L,clk, reset);
  :
  parameter TH= 254
  reg [7:0] counter;
  always @ (posedge clk)
    if (reset) counter<=0;
    else if (counter == th) counter <=0;
    else counter <= counter+1;
  assign o=(counter < L) ? 1 :0;
endmodule
```

19 ماجولی که هر ۱ ثانیه ببینیم چند پالس دارد.




```
reg [21:0] counter;
always @ (posedge clk)
  if(reset) counter<=0;
  else begin if (1Hz) begin @ <=counter;
                            counter<=0;
             end
       else if (reset) counter <= counter +1;
  end
assign freq = o;
```



در ActiveHDL با stimulator که شکل موج ورودی را می دادی.

وقتی ماژول بزرگ می شود تعداد پورت ها زیاد می شود

و حالت های شکل موج های ورودی زیاد می شود. لذا

نمی توان از این تک تک مقدار تعیین کرد. برای هر

شکل یک ماژول دیگری می نویسیم تا آن شکل موج ها را تولید کند و سپس هر دو را با هم اجرا می کنیم.

Reset : در وریلاگ عبارت initial وجود دارد که قابل synthesize نیست، کدی که زیر آن نوشته

می شود را اول اول انجام می دهد:

```
initial  begin    یا    fork
           a = 0;
         end             join
```

```
module tester (A, B, C, out);
  output A, B, C;
  input out;
  reg A, B, C;
  initial begin
      A = 1;
      B = 0;
      C = 1;
    end
endmodule
```

چون مقدار گرفته است و (module tester)

توجه کن که A، B و C رجیستر تعریف شده اند و توجه

کن که initial قابل سنتز نیست! بلکه فقط برای

ماژول سنتز تعریف می شود.

(reg A, B, C;) → ...

A = 1; → نیازی به => نبود.

! اگر بخواهیم در صفر مقدار داشته باشد و بعد از تأخیر ۱۰۰ns مقدارش عوض شود. برای نوشتن چنین کدی

داریم:

عدد
میزان تأخیر

```
initial begin
    A=1;
    B=0;
    C=1;
    #100        ← قابل سنتز نیست
    A=0;
    B=1;
    C=0;
end
```

- برای تعیین واحد میزان تأخیر زمان باید اول کد و بالای کد مشخص می‌کنیم. که مثل define قبل از ماژول می‌آید:

`timescale 1ns / 1ps ← معمولاً ns است

```
initial begin
    A=0;
    #50
    A=1;
    #100.01     → برای تعیین دقت
    A=0;
    #20
    A=1;
end
```

`timescale 100ns/1ns

به طور default میزان time scale به اندازه 1ps است (در ActiveHDL).

- در تاپ ماژول که تعریف کنی همه جا یک scale می‌دهی.

- ماژولی تعریف کنیم که پالس ساعت تولید کند.

```
`timescale 1ns/1ps
  module clkGen(clk);
   output clk;
   reg clk;     → چون در بلوک مقدار می‌گیرد.
   initial begin
        clk=0;
```

```
forever #20 clk=~clk;
end
endmodule
```

یعنی همیشه با تأخیر 20ns ، clk را toggle کند

به نوع دیگر ⇐

```
initial begin
    clk=0;
    end
always begin
    #20 clk=~clk
    end
```

به هیچ event بستگی ندارد. توجه کن که چون تأخیر داری زمان جلو می‌رود. (این بلوک هر بار از اول تا آخر اجرا می‌شود)

این عبارت را همواره انجام می‌دهد.

این دو کد کاملاً هم ارزاند، که می‌توان یعنی بهتر است از forever استفاده شود.

*

```
module A;
  reg a,b,c;
  intial beginfork
    a=#3 2;
    #1 a=2;
    #2 b=3;
    end join
  initial begin
    #2 c=a;
    end
endmodule
```

c=x در این حالت می‌شود. چون تا دو هنوز a مقدار نگرفته است

اگر fork join ←

1	2	3	5
a=2	c=a c=2	b=2	
a=2	b=3 c=a		
	c=a=x	a=2	b=2

اگر a = #1 2; یعنی در این لحظه a را نمونه‌برداری کن و بعداً به اختصاص بده.

در حالت a = #3 2; #2 b = 3; اول در دستور a انجام می‌شود کامل بعد مقدار را عوض می‌شود

2	3
b=3	a=2

اگر a <= #3 2; # b <= 3; ⇐ یعنی بلوک می‌شود. و بعد انجام می‌دهد.