fpga: googoolia.com

VHDL :

refrence: Verilog Quick start.

pasword = Saeed

fpga : Field programmable (Gate Array)

در زمینه‌های مختلف — قابل برنامه ریزی — آرایه‌ای از گیت‌ها

یعنی می‌توان آنرا در محل داشته باشیم و با یک کابل ساده پروگرام کرد.

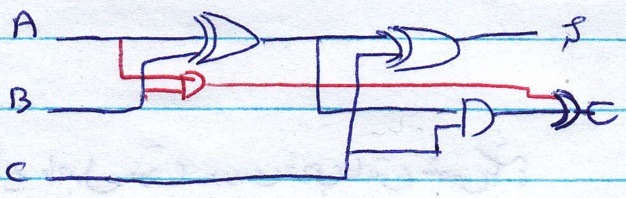
Lab	← از شبیه‌سازی یک آزمایشگاه	System
on	کاملا شبیه در یک فضای	on
a	محدود.	A
chip		chip

field programable Lab on a chip = یعنی بتوان آزمایش شبیه‌سازی انجام داد که قابل برنامه ریزی باشد در زمینه‌های مختلف

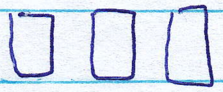
- چگونه یک مدار جمع کننده‌ای را ساخت؟

پایه‌ترین کار این است که شکل را بکشی و روی fpga بریزیم

Fulladder:



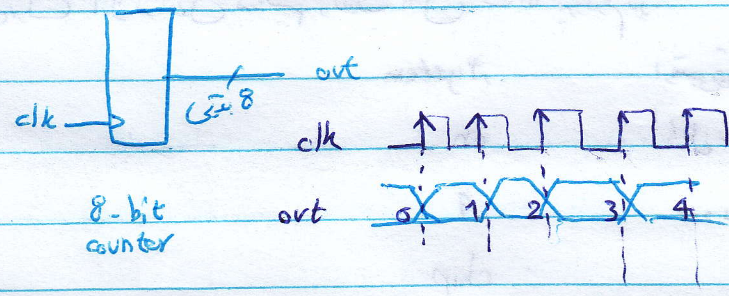
همچنین برای تعداد بیشتر بیت‌ها هر یک را درست کرد و پشت سر هم ردیف کرد.



این حالت چون بزرگ است مدیریت و انجام آن کار مشکلی است مثل چیپ های.

حال اگر من بتوانم نرم افزاری را اختراع کرد که بتواند خود نرم افزار fulladder را بیاورد. یعنی با هدش باشد. حالتی که می گوییم را بفهمد و با آرایش گیتی مدارها آشنا باشد ⇐ یعنی کامی است مدار را توصیف کرد.

اگر بخواهیم یک شمارنده 8 بیتی بسازیم: باید توصیف کرد و خود نرم افزار شماتیک را بفهمد.



مداری می خواهیم که خروجی اش در هر لبه ی بالا رونده یکی زیاد شود.

که باید syntax، نرم افزار را رعایت کرد:

مداری را می خواهیم که همیشه در هر لبه ی بالا رونده خروجی یک زیاد شود:

همیشه — لبه ی بالا رونده ی کلاک

```
always @ ( posedge clk )
out <= out + 1 ;
```

out یکی زیاد کن.

که باید ماجول فوق ورودی خروجی هایش مشخص باشد:

```
module 8bitcounter ( input clk , output [7:0] out )

end module
```

یعنی خروجی 8 بیت است.

حال هر طور که بتوانی توصیف کنی syntax خودش را دارد. که مطابق با زبان Verilog است. حال اگر نرم افزاری باشد که اگر با C نوشتیم باز هم بفهمد خوب است. یعنی برای توصیف سخت افزار زبان های متفاوتی هست. مثل VHDL، یا System-C کد.

اسم نرم افزار Synthesizer ← (مرحله ای که کد برنامه توصیفی به آرایش گیت تبدیل می شود) ← synthesis کردن

Verilog --synthesis--> gat

توصیف --سنتز--> آرایش گیت ها

پس امروز یاد گرفتیم که به جای شماتیک کشیدن، مدار را توصیف کردیم تا نرم افزار فوق به گیت تبدیل کند. که نرم افزار باهوش تره یعنی مداری که درست می کند بهتر از شماتیک ما است :)

حال اگر با system-c کد باشه: کانتر 4 بیتی.

اسم یک کلاس است (counter)

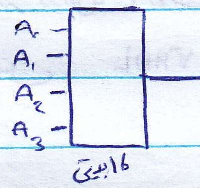
```
SC_MODULE (counter) {
    sc_in_clk clock;
    sc_out<sc_uint<4>> out;
    sc_uint<4> counter;
    void inc_counter() { counter = count+1; }
    SC_CTOR (counter)

    SC_METHOD (inc_counter) {
    sensitive << clock_pos();
  }
}
```

احتیاجی به یاد گرفتن نیست :) (sc_out خط)

کانستراکتور کلاس counter است (SC_CTOR)

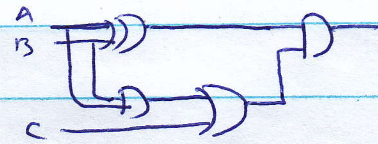
هر بار باید تکرار شود (SC_METHOD)

* ساختار داخلی FPGA:



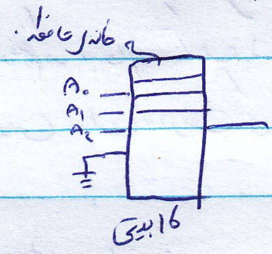
با داشتن چنین حافظه‌ای می‌توان تابعی 4 متغیره را تعریف کرد.

$f(A_0, A_1, A_2, A_3)$



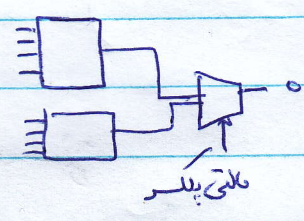
این شکل را می‌توان به حافظه بالا انتقال داد:

حال اگر به ABC مقدار دهیم و خروجی را به ازای هرکدام حساب کرده و به حافظه داد:



در خانه شماره صفر مقدار خروجی به ازای صفر صفر صفر را می‌دهیم.

یعنی خروجی این حافظه با مدار فوق یکی است.



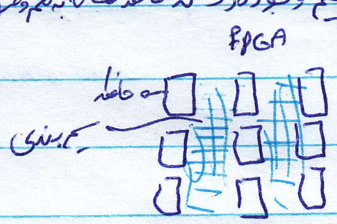
یعنی اگر با این چیپ که حافظه دار است می‌توان استفاده کرد

یعنی داخل FPGA اصلاً گیت نیست بلکه

خانه حافظه وجود دارد که 16 بیتی هستند و کنار هم

قرار دارند و سیم بندی مناسبی وجود دارند به طوری که

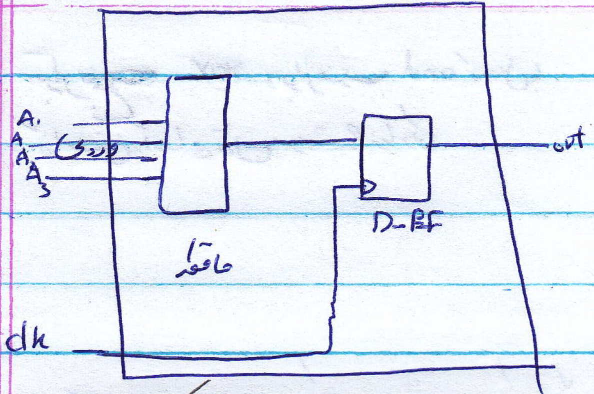
همه می‌توانند با هم در ارتباط باشند؛ یعنی داخل FPGA حجم زیادی سیم وجود دارد که حافظه‌ها را به هم وصل می‌کند.

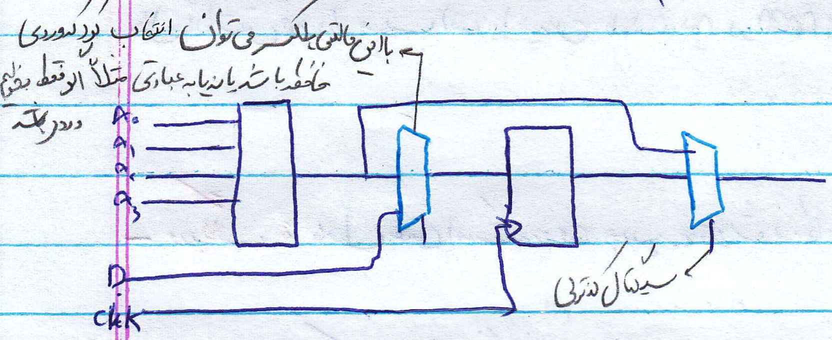


یعنی باید حافظه‌ها را مشخص کرد و سپس نوع سیم بندی‌ها را معین کرد.

که در این حالت مدارها ترکیبی بود. حال اگر بخواهیم ترتیبی باشه: باید یک فلیپ فلاپ اضافه کرد.

پس اگر حافظه خواستیم استفاده کرد از D-FF .





باید حافظه معین شود.

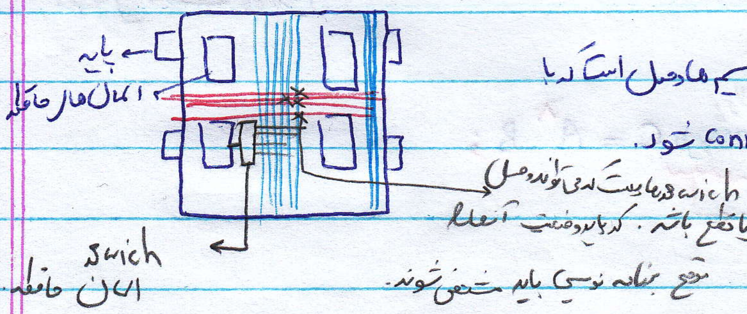
باید از signal control استفاده شود تا بگوید که از کدام حالت باید استفاده کرد.

حال هر کدام بلوک ها را در FPGA می گذاریم. پس باید اول حافظه سپس مقدار حالت بگذاریم و سپس سیم بندی داخل FPGA که چه طوری به هم وصل شده اند.

و باید FPGA ، قابلیت باشه.

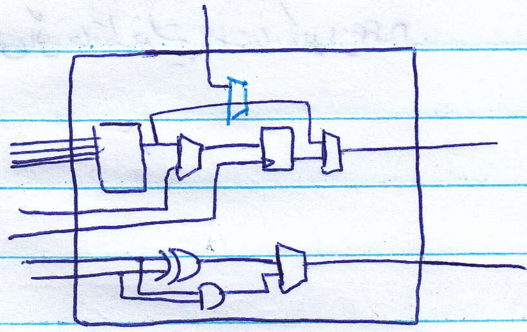
این حافظه یک Ram است که اگر برق برود از بین می رود.

برای سیم بندی: موقع ساخت به طور منظم المان ها را کنار هم می گذارند. سپس پایه ها را می گذارند. بعد همه آن ها را به هم سیم می کشند.



کنار هر حافظه سوییچ وجود دارد که به تمام سیم ها وصل است که با برنامه نویسی می توان گفت که چه گونه connect شود.

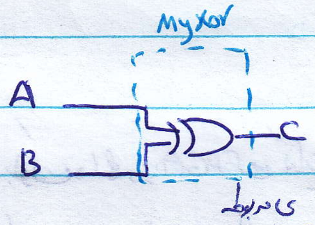
بعضی از المان ها در تمام مدار ها استفاده می شود مثل جمع و ضرب و حافظه. حالا می آیند یکسری gate به مدار بلوک اضافه می کنند تا بهتر شود.



برای جمع xor و برای ضرب and که حالا با گذاشتن آنها می‌توان سرعت را زیاد کرد.

حال با گذاشتن خود مدار ضرب که شده یا چندین بلوک که در FPGA می‌گذاریم تا سرعت را بالا ببریم.

— وریلاگ: ماژول بخشی است که روی چیپ ریخته می‌شود و کار مربوطه را انجام دهد.



می‌خواهیم توصیف کنیم:

مثل C که هر دستور ; بگذاریم.

پورت ها

module اسم ماژول (A, B, C);

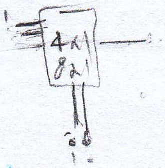
(اسم ماژول ← myxor)

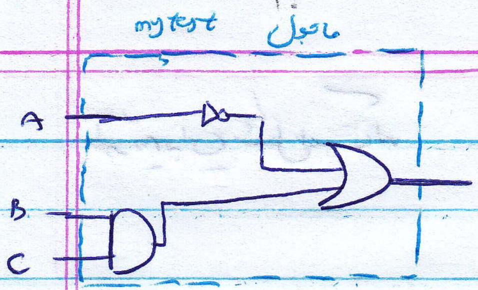
input A; ← معرفی پورتها (input / output)
input B;
output C;
xor X(A, B, C); ← المان‌های اولیه — (X: اسم المان)
End module

به المان اسم انتصاب می‌دهیم تا اگر بعد خواستیم صدا کنیم یا اگر چند xor داریم که می‌خواهیم آن‌ها را از هم جدا کنیم پس اسم‌های متفاوت می‌گذاریم.

(xor X(A, B, C); → می‌توان این گونه نوشت تا توصیفش کنیم:) assign C = A ^ B;

assign ← برای این است که نشان دهد مدار ترکیبی است



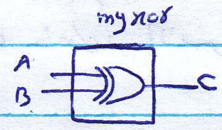


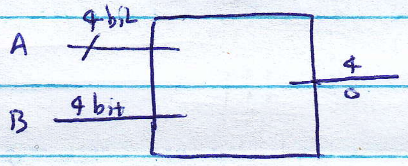

```
module mytest (o, A, B, C);
output o;
input A, B, C;

assign o = (B&C) | (~A);
endmodule
```

علامت مثل یا

ماجول هایی نوشتیم دیگه را صدا کند :

اینو داریم:





o = A nor B

اول یک nor دو بیتی می سازیم :

اسم ماجول جاری

```
module mynor2 (A, B, C);
output [1:0] C;
input [1:0] A, B;
myxor Ins1 (.A(A[0]), .B(B[0]), .C(C[0]));
mynor Ins2 (.A(A[1]), .B(B[1]), .C(C[1]));
endmodule
```

بردار دو بیتی
اول عدد بزرگتر و سپس کوچکتر.

اول اسم خودش که در تعریف ماجول
استفاده شده (یعنی در ماجول mynor)
با نقطه بگذار.

برای nor دو بیتی دو بار باید mynor را
صدا کرد که باید به آن اسم داد. یعنی
یک بار اسم خودش را صدا می کنیم و بعد
اسم جاری را می گذاری تا بعد بتوانی
با این اسم جدید صدایش کنی.

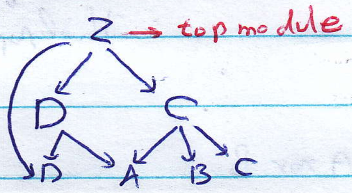
برای nor 4 بیتی باید دوتا nor 2 بیتی را صدا کنی :

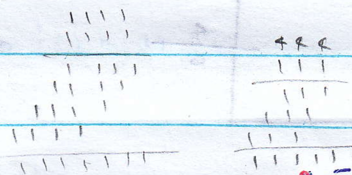
```
module mynor4 (in1, in2, out);
output input [3:0] in1, in2;
       output [3:0] out;
mynor2 ins1 (.A(in1[1:0]), .B(in2[1:0]), .C(out[1:0]));
mynor2 ins2 (.A(in1[3:2]), .B(in2[3:2]), .C(out[3:2]));
endmodule
```

طور کنیم به این شکل است که

```verilog
module mynor2(A,B,C);
output [1:0]C;
input [1:0] A,B;
assign C=A^B;
endmodule
```

top module = آن ماجولی که بالاسر همه است دیگر کسی آن را صدا نمی‌کند Topmodule است :





* ضرب کننده 4 بیتی :

```verilog
module Mult4(in1,in2,out);
input [3:0] In1,In2;
output [7:0] out;
assign out = In1 * In2;
endmodule
```

* جمع کننده 8 بیتی :

```verilog
module Adder1(In1,In2,out,C)
input[3:0] In1,In2;
output [3:0] out;
output C;
wire [4:0]W;
assign W = In1 + In2;
assign C = W[4];
assign out = W[3:0];
end module
```

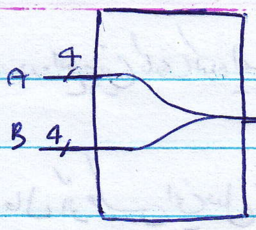
همه سیگنال‌ها دنبال آن 4 بیتی‌اند.

اما C تک بیتی

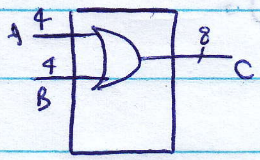
جمع دو عدد 4 بیتی 5 بیتی است
لذا برای assign باید از ترکیب
out و C استفاده کرد. اگر یک
سیگنال جاری ایجاد بکنیم W مشکل
حل است. یعنی فرض می‌کنیم که

خروجی In1+In2 فعلاً در سیم 5 بیت W قرار دارد. سپس با انتصابی دادن بیت‌های W به C و out مشکل حل شد.

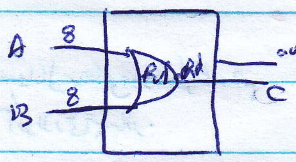
- مثال از عملگر تل:




```
module mytest (A, B, C);
input [3:0] A, B;
output [7:0] C;
assign C = {A, B};

endmodule
```




```
assign C = A | B;
```




```
input[7:0] A, B;
output[7:0] O;
output C;

assign {C, O} = A + B;
```

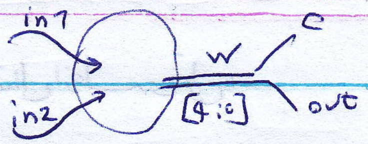
1. Design Entry ⟹ Functional simulation - verification — توصیف تا simulation در مرحله اول طراحی
2. Synthesis ⟶ post synthesis simulation — آرایش‌ها simule که این منظور ایده‌آل است.
3. implementation ⟶ پیاده‌سازی ⟶ post rout simulation — تمام تأخیرها واقعی اند.

نرم افزارهای شبیه سازی متفاوتی وجود دارد؟

1. Active HDL : Aldec
2. ModelSim : mentor
3. IUS : Cadance : Insacive unification Simulator : فقط برای Linux
4. Synopsys VCS . تحت Linux
5. SystemC

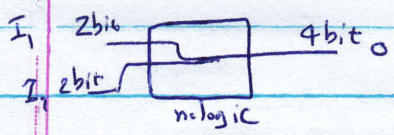
2-3 خیلی قوی اند ⟸ Native Code Simulator

verilog ⟶ opcode x86



درواقع چنین کاری کرده ایم.

حال اگر عکس این عمل را بخواهیم انجام دهیم. به عبارتی دو سیم داریم و می خواهیم یک سیم بکشیم.



concatenation به این عمل گوییم

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

I2: 2 بیت پرارزش — I1: دو بیت کم ارزش

سیگنال های مورد نظر را به هم ← آنها را می چسباند.

درواقع 8 بیتی به جای را بنویسیم =>

```
assign {C, out} = In1 + In2;
```

یعنی 4 بیت کم ارزش را در out و 1 بیت پرارزش را در C می ریزیم. در این حالت دیگر wire احتیاج نیست.

* سیمولاتور: نرم افزاری که بتواند برنامه ها را شبیه سازی کند و ببینیم که همان کاری که می خواهیم سخت افزار انجام می دهد.

0: طراحی

1. عملکرد را چک کنیم (تا پاسخ درست باشد) ← functional simulation یا verification ← همه چیز ایده آل است.
2. سنتز ← مدار تبدیل به گیت.
3. باز می توان گیت هم سیمولیت کرد که عملکرد هاردوار را دید: post synthesis simulation
4. place & route ← ببریم روی fpga.
5. post route simulation ← همه تأخیرها واقعی است. لذا ایرادها نشان دهد.

ok
name

creat an Empty design

default HDL
verilog
fpga نوع
name
finish

add new file → verilog sourse code

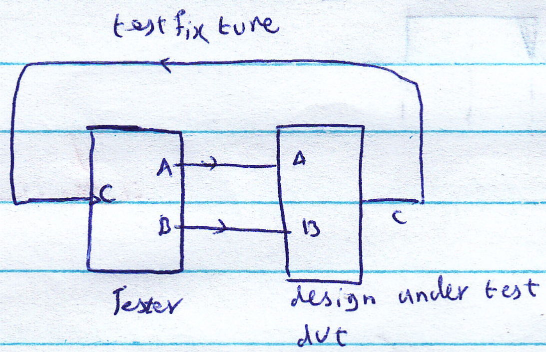
name / ok.

کد در فایل top.v شروع به نوشتن می‌کنیم

rightclick روی top → compile → ✓ سبز (بدون غلط)

منوی simulation → initial simulation

تا کلیک /



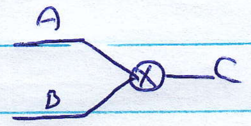
Tester:

```
module tester (A, B, C, O);
output [3:0] A, B;
input c;
input [3:0] O;
assign A = 3;
assign B = 3;
endmodule
```

```
module top test; (X)   داخل بالای پورت نمیذارد
adder Ins1 (.A(A), .B(B), .C(C), .O(O));
adder Tester (.A(A), .B(B), .C(C), .O(O));
end module
```

- massive multiplayer online Game
- physics engine

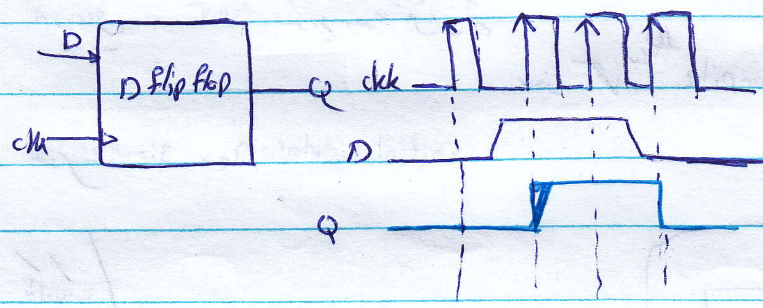
چیزی که ترکیب انجین است و محاسبات فیزیکی در بازی را انجام می‌دهد. بسته مجموعه توابع می‌هستند.



assign C = A * B; → Continous assignment

یعنی هر لحظه خروجی از ورودی ساخته می‌شود.

* مدارهای ترتیبی در وریلاگ:



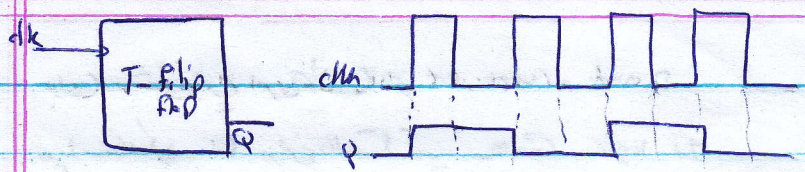
توصیف وریلاگ:

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

کلاً یه دستور است که اتفاقی می‌خواهد (← always @ (posedege clk))

این علامت برای مساوی نا هم است (← <=)

اگر نگوییم سیگنال چیه؟ برای پیش‌فرض wire است.
اما در این حالت output یک register است.
پس آن متغیری که در always مقدار می‌گیرد
باید register تعریف شود. پس در واقع با نوشتن register ، فلیپ فلاپ بودن را
معین می‌کنی.



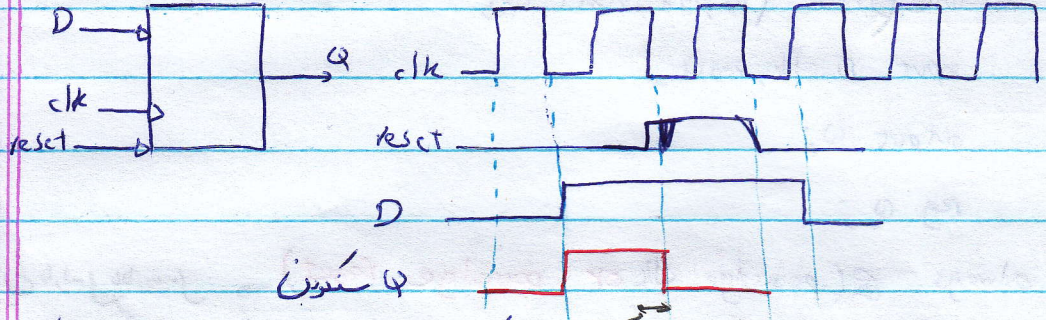

```
module tff(clk, Q);
input clk;
output Q;
reg Q;
always @(posedge clk)
    Q <= ~Q;
endmodule
```

negedge ⟵ ⟶ برای لبه پایین رونده

هر مدار ترتیبی که reset دارد تا مشخص باشد که مقدار اولیه صفر بوده است؟

برای نوشتن reset: ریست سنکرون باشد؟

D.FF ریست دار

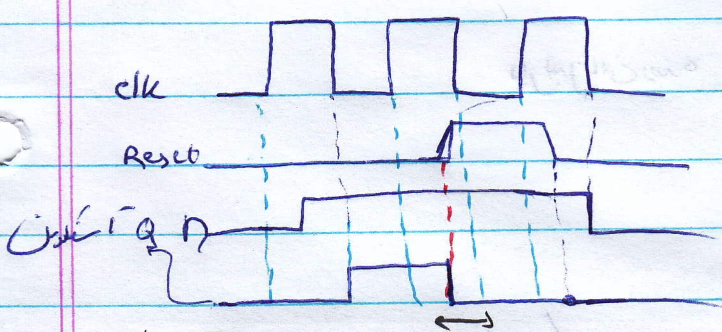


ریست سنکرون در لبه‌های پالس اجرا می‌شود. لذا در فاصله‌ای که ریست داریم هنوز D داریم. به دلیل پالس دیگر Q صفر می‌شود.

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

```
else
    Q <= D;
endmodule
```

برای نوشتن این امر می‌توان از not reset و D and هم استفاده کرد. اما در این حالت یک logic Gate اضافه کرده‌ای که سرعت را می‌آورد پایین.



در اسنکرون منتظر پالس ساعت نمی‌شویم
در همان لحظه Q را صفر می‌کند.

```
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
```

← این بار به لبه بالارونده reset هم تأثیر می‌گذارد

در واقع در این حالت مدار به دو event حساس است.

! هر متغیری رو فقط در یک always مقدار بده! حتی اگر مطمئنی که [illegible] در always های دیگر مقدار نداشته!

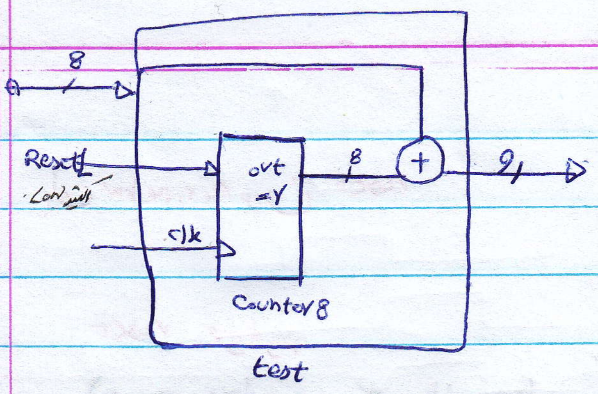
```
module TFF (reset, clk, Q);
input reset, clk;
output Q;
reg Q;
always @ (posedge clk)
  if (reset)
    Q <= 0;
  else
    Q <= ~Q;
endmodule
```

reset يكون Active low يكون

```
always @ (posedge clk or negedge reset)
  if (!reset)
```

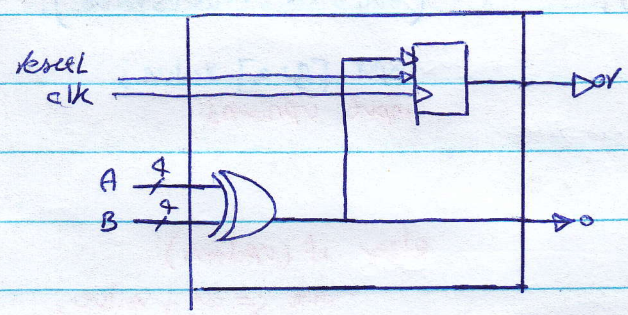
```
module counter (out, clk, reset);
output [7:0] out;
input clk, reset;
reg [7:0] out;
always @ (posedge clk)
  if (reset)
    out <= 0;
  else
    out <= out + 1;
endmodule
```

```
(out, clk, reset, value, upDown);
input [8:0] value;
input upDown;

else if (upDown)
  out <= out + value;
else
  out <= out - value;

out <= out + value;
```



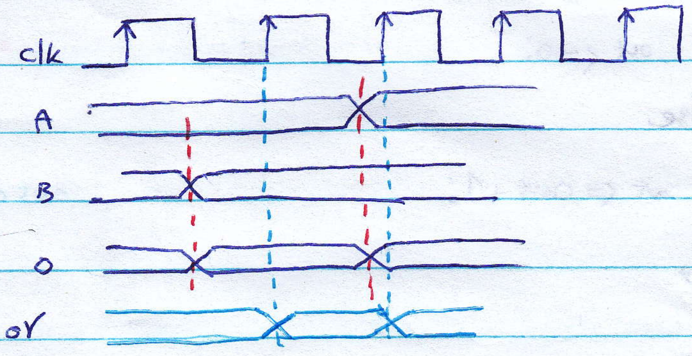

```
module test(o, A, clk, resetL);
  output [8:0] o;
  input [[redacted]:0] A;
  input clk, resetL;
  reg [7:0] r;
  always @(possedge clk)
    if (!reset)
      r <= 0;      → مدار اینجا ترتیبی است
    else r <= r+1;

  assign o = A + r;  → مدار اینجا ترکیبی است لذا ۲ ورودی داریم
endmodule
```

کانتر ۸ بیتی

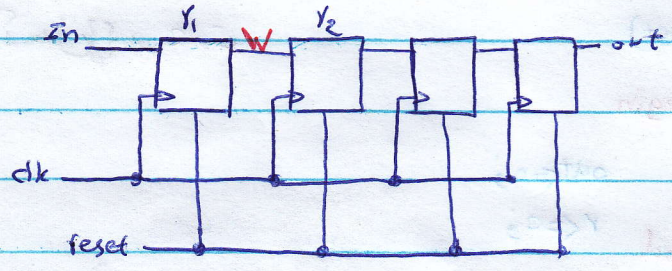


or فقط در لبه ساعت عوض می‌شود اما o در هر لحظه که A و B عوض شود، o عوض می‌شود.

شکل موج

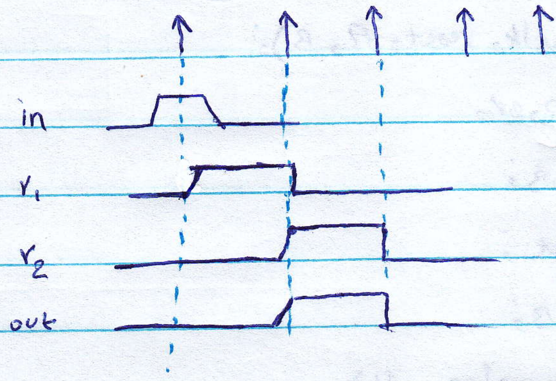


```
module test2 (or, O, clk, reset, A, B);
    output [3:0] O, or;
    input  [3:0] A, B;
    input clk, reset;
    reg [3:0] or;
    assign O = A ^ B;
    always @(posedge clk)
        if (!reset)
            or <= 0;
        else
            or <= O;
endmodule
```



8 Shift register

```
1.  module shiftr ( In, reset, clk, out);
2.  input In, clk, reset;
3.  output out
4.  wire w; reg r1, r2;
6.  always @(posedge clk)
7.      if (rest)
8.          r1 <= 0;
9.      else
10.         r1 <= In;
11. assign w = r1;
12. always @(posedge clk)
13.     if (reset) r2 <= 0;
14.     else r2 <= w;
15. assign out = r2;
16. endmodule
```



در یک لحظه نمونه‌برداری ⟸
کن و در یک لحظه زمان بعد آن را هم انتقال بده.

```
module sr(out, in, clk, Reset);
    output out;
    input clk, reset, In;
    reg out;
    reg r;
    always @(posedge clk)
        if (reset)
            begin
                out <= 0;
                r <= 0;
            end
        else
            begin
                r <= In;
                out <= r;
            end
endmodule
```

می‌خواهیم در Verilog چند دستور بگذاریم
که معادل { } در C است.

– همه برنامه‌های Verilog به این طریق است:

```
always @ (   )
    if reset
        begin
        end
    else
        begin
        end
```