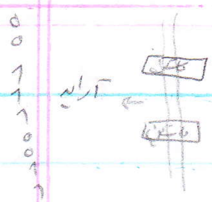
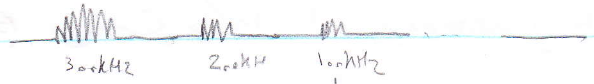


5 کافی است هرگاه ۱۰ از ۱ به صفر برویم counter یکی زیاد شود

```
module car(out, In, clk, resetL, start);
    output [3:0] out;
    input In;
    input clk, resetL,
    input start
    reg [3:0] out;
    reg r;
    always @ (posedge clk or negedge resetL)
        if (!resetL) begin
            r <= 1;
            out <= 0;
                end
        else begin
            if (start) begin
                r <= 1; out <= 0; end
            else begin
                if (!In && r)
                    out <= out+1;
            r <= In
                end
```

16



پس 10000ns = 10ms

50ns است ← 10000/50 = 200 cycle

← یعنی اگر 200cycle تغییر نکرد در نظر می‌گیریم.

در ISE: وقتی core، block ها را نداریم در FPGA گروه‌های FPGA است و از slice ها تولید کنید.

```
assign counterth = 1 << (20_bitlocation - 1);
```

| ادامه core:

می‌خواهیم امروز core تولید کرده و آن را شبیه‌سازی کنیم.

برای core run: Run در coregen

مراحل به صورت زیر است.

file new project ; family: FPGA نوع

Device xc3s400

package pq..

Design Entry → verilog

Flow → یعنی ISE

Simulation ; Behavi..

در tab : Advance : apply, ok.

دقت کنید که این core ها بسیار بهینه‌ترند.

memory & storage:

/Block memory → در کنار FPGA

Distributed → از ترکیب logic cell است

component name: نام

یک پورت داریم یا دوتا. دو پورت: دوتا سیگنال آدرس و clk ... دارد و هم‌زمان ← مثل حافظه

هر دو پورت به حافظه دسترسی دارند.

این کارها با HDL نمی‌توان کرد. ← هر دو پورت کار دارد، true Dual port ram:

next

سپس عرض حافظه را مشخص می‌کنیم. می‌توان 16 - 16 بیت نوشت و 8 بیت خواند.

A = 32 bit
B = 8 bit } → در این حالت هر کدام مختلف BUS است.

اگر با clk بخواهیم که همزمان هم بخوانیم هم بنویسیم مشخص می‌کنیم که اول کدام باشد. read first

next

memory-initialization-r-

اگر بخواهیم که FPGA بالا می‌آید دیتا در اولیه داشته باشد: برای ساخت این فایل ←

MEMORY-INITIALIZATION_RADIX = (16 / مبنا)

بین مقادیر کاما می‌خواهد ,VECTOR =
" " " underline یا

abcDEF ;

این را save کرده در همان core که directory است اینجا است. .coe

سپس همان پنجره

load Init file

fill rem. ← برای مشخص کردن ریم حافظه است

سپس generate ← next.

- یک فایل pdf، تمام اطلاعات مربوط به این core در آن است.
- یک فایل veo برای استفاده از core کافی است اطلاعات داده شده را copy و paste می‌کنیم.
- فایلی که design مشخص شده در آن است => در coregen : format ngc است.

کد وریلاگ : (اولین "design" با clk).

```
module coretest ( In, clkIn, out, clkout, resetL);
   input [31:0] In;
   input clkIn;
   output [7:0] out;
   input clkout, resetL;
```

سپس کد core را استنتاج کرده و پورت‌ها را به طور مناسب وصل می‌کنی.

اگر بخواهیم هر 4 سیکل یک بار داده بخوانید => باید هر 4 سیکل یک بار باید wen را فعال کرد:

```
reg [1:0] incounter; reg [4:0] inad
always @ ( posedge clkIn)
   if (!reset) incounter <= 0; inad <= 0 end
   else begin
      incounter <= incounter + 1; if (incounter == 3) inadres <= inadres + 1;
   end
```

.wea (& incounter) یا .wea((incounter == 3)) <= wea را

و این سیگنال inad را به adda وصل می‌کنیم

و چون با پورت B می‌خواهی بنویسیم => 0 1'b یعنی از آن استفاده نمی‌شود. ← چون کد

را نداری دقیقاً یک اضافه می‌کنم

برای adrb کد شبیه بالا نوشته اما با clkb کار کرده و جای ... اضافه شود.

و چون در b کد داخل نمی‌کنی dinb را صفر می‌کنی.

برای شبیه‌سازی: یک counter می‌سازیم:

```
reg [31:0] In;
reg clkIn, clkout, resetL;
// reset :
 initial begin
        In=0;
        resetL=0;
        #100 resetL=1;
        end
//clkout
 initial begin
      clkout=0;
forever #25 clkout=~clkout;
 end
always @(posedge clkIn)
      In<=In+1;
endmodule
```

→ به جای ورودی‌ها نوشتن overwrite شده‌ها را نمی‌خوانیم.

بعد top:

```
module top
wire clkIn, clkout, resetL;
wire [31:0] In;
wire [7:0] out;
coretest_tester ins (.In(In), .clkIn(clkIn), .out(out), .clk(out), .resetL(reset));
```

باز کردن مدلسیم ← آدرس File / changedirectory /

vlib work

vlog .. hdl

vsim top

برای ۲۷۸۲ دارد چون نوع core که به مدلسیم داده‌ایم:

در خود پروژه یک فایل ۳ داریم. باید در دل آن یک برنامه است که اضافه کنیم (folder)

فایل‌های دیگری که core داریم ← در folder verilog / source در xilinx : فایل‌ها

core های هستند که باید به پروژه اضافه کنیم.

با اضافه کردن این فایلها به فلدر پروژه

vlog ../xilinx/*.v -work xilinx

compxlib = در RUN ...
این کار compxlib کردن

vsim work.top -L xilinx

run 3000 ns

سپس باز کردن را synplify میدهیم

File / new / .. choose ;

add file core.v → RUN : داریم

باید این فایل وریلاگ که اضافه کرد.

سینپلیفای فقط فایل‌های که می‌بیند. سینتاکس که ثابت دستور دارد.

synthesis را این که می‌بیند

سپس edf که می‌سازد.

با باز کردن ISE => file newproject / آدرس / inputdesign file را edif آدرس می‌دهیم.

finish کنیم error می‌دهد. سپس Implement

باید فایل ngc. را به فایلی که E [illegible] تولید شده است اضافه کنیم.

که core سنتز شده.

از آن core که ساختیم دوباره استفاده می‌کنیم:

memory initialization file .mif = فایلی که مقادیر اولیه را در خود نگه می‌دارد، که محتوای همان فایل coe. است.

این فایل جزو پارامترهایی است که مدلسیم آن را می‌خواند.

اگر این فایل به فایل شبیه سازی اضافه نشود مدلسیم ایراد می‌گیرد.

Vsim top_L xilinx

در این حالت پنجره‌ی [illegible] باز می‌شود. اما

در این حالت چون mif. نیست، مدلسیم نمی‌تواند فایل را باز کند لذا run 1ns

دستور [illegible] می‌دهد.

برای حل این مشکل باید فایل e.mif در فلدر شبیه سازی باید اضافه شود.

restart = برای دوباره شبیه‌سازی کردن

برای سنتز: synplify : فایل EDF فقط برای فایل آخر تولید می‌کند.

برای حل مشکل blackbox یک سری دستور هست که الان مهم نیست.

با باز کردن ISE و کپی ngc. و EDF. در فلدر Imp.

الان شبیه‌سازی post rout simulation داریم: که یک Generate post rout simulation وجود دارد.

این گزینه یک فایل وریلاگ جدید تولید می‌کند که از روی design می‌سازد. که در آن primary element ها

را به صورت ماژول صدا کرده، آنها را خودش به هم وصل می‌کند. در این حالت به

simulation عمل همه اینها را می‌خواهیم که بابسونه SDF: standard delay format است. که

شامل تأخیرهاست.

```
#1 reset=1;  →  #(1,1,2) reset=1;
```

(عرض فایل؛ mindelay، maxdelay، typical delay)

وجود این سه نوع تأخیر به علت آن است که به شدت وابسته به دماست در این حالت با maxdelay

کار کرده تا مطمئن شویم. برای مطمئن شدن از شدت تأخیر و درست بودن برنامه است.

```
initial $sdf_annotate(
```

در این فایل task است که تأخیرها را از روی sdf خوانده و در مدار اعمال می‌کند.

دوباره به شبیه‌سازی با postsim و باز کردن مدلسیم =

vlib work لایبرری می‌سازیم

vlog فایل‌ها

فایل‌های sdf و وریلاگ را در postsim باز می‌کنیم

در این حالت دوباره باید همه vhdl ها را دوباره compile کنیم.

باید در این حالت باید همه قسمت‌های داخل FPGA را اضافه کنیم.

برای این کار داخل فلدر ISE، داخل verilog و sorce

simprims را کپی کرده داخل فلدر پروژه می‌آوریم.

vlib simprims   فایل‌ها در داخل simprims بریزد

vlog ../simprims/*.v -work simprims

فایل‌ها کجا بریزد؟ لایبرری که دادیم!

vsim top -L simprims

در این حالت غلط داریم: وقتی bitstream رو ریختیم اول مقادیر اولیه FF مشخص شود خروجی ها را از highamp خارج نکند. دو نوع سیگنال داریم. golobnt : مال I/O های FPGA را از H.A. خارج می کند.
glbl : مقدار اولیه

تنها جایی که دو تا باهم compile می کنیم
vsim top L simprims glbl

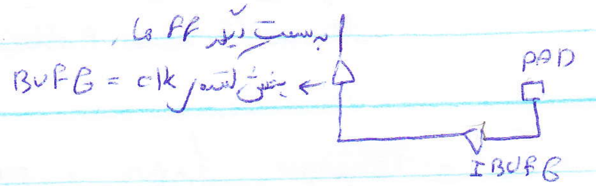
موارد تشخیص clk => user constraint → timing constraint

فلیپ فلاپ:

با نوشتن برنامه فلیپ فلاپ implement کردن:

imp / place and rout / Edit FPGA /

PAD : پایه FPGA که دارد FF نمره است. → این پایه به بافر IBUFG است. => پالس ساعت فقط توسط یک پایه خاص به FPGA می رود.
از این بافر در FPGA پخش می شود: ?. شکل علت صاف نیست



هر سیگنالی که قرار است کلی FF ها را درایو کند باید با BUFG وصل باشد. لایه های که شامل clk است لایه های خاص دارد. و این باعث سرعت بالا رفتن می شود.

می خواهیم مطمئن شویم که با clk های خاص کار کند در timing constraint:

1: clk domain

2: input

2. output

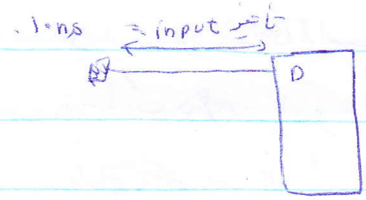
با Imp کردن در console tab : slack => تفاوت زمانی است که ما خواسته‌ایم و آنچه که FF لازم است ! اگر + باشه می‌تواند انجام دهد با - یعنی design ایراد دارد .

همین حال برای input : طوری طراحی شود که از ورودی گرفتن چقدر طول بکشه . همین کار رو با output هم می‌کنیم یعنی از FF تا output چقدر طول بکشه .

external clock to pad : زمانی که طول می‌کشه clk از ورودی تا خروجی برسه . = offset out

فاصله لبه بالارونده از clk تا تولید شدن output :

این عدد باید بزرگتر از پریود FPGA مان باشد ، یا برابر یا کمتر بگیر .



external setup time (offset in)

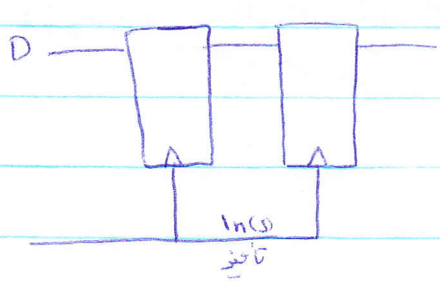
از پایه FPGA تا لبه بالارونده clk که FF قرار هست با آن کار کنه .

این مقدار رو نصف مقدار پریود بگذار .

معمولاً setup time ، input و output زیاد است که تاخیرات شامل net و از آن مهمتر ظرفیت خازنی خروجی است

علت مشکلات نوسان clk :

اگر net & clk ورودی تاخیر داشته باشه یعنی همزمان رخ نمی‌دهد و تبدیل می‌شود

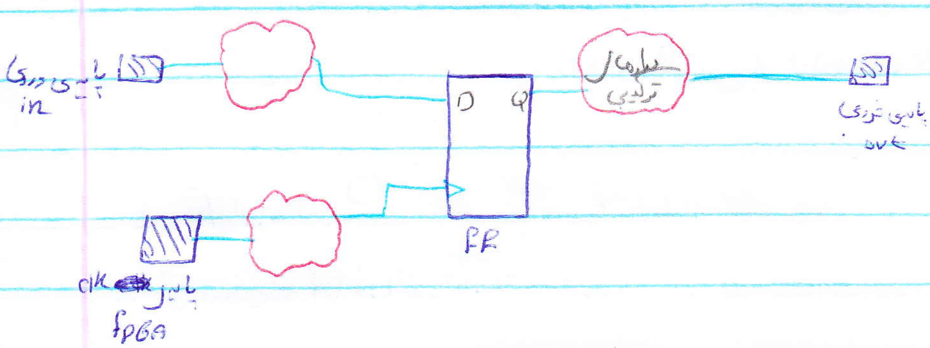


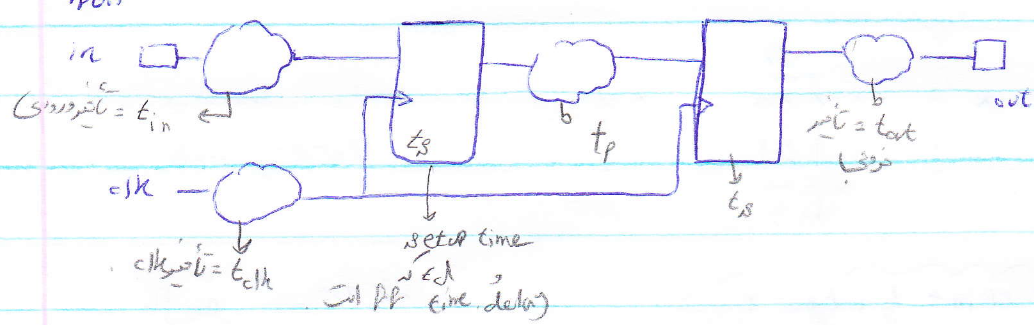
در واقع مسیر صاف نیست چون می‌خواهیم تا آنجا که می‌شود هم‌زمان همه clk بخورند.

یک سری Delet در FPGA Editor ⇐ تأخیر clk ها تا هر FF می‌بینیم.

تأخیرها 1,035 Q ← تفاوت زمانی که در آن clk به FF ها متفاوت می‌رسد.

- ورودی که مدار با آن کاری که هیچ ربطی به offset ندارد.

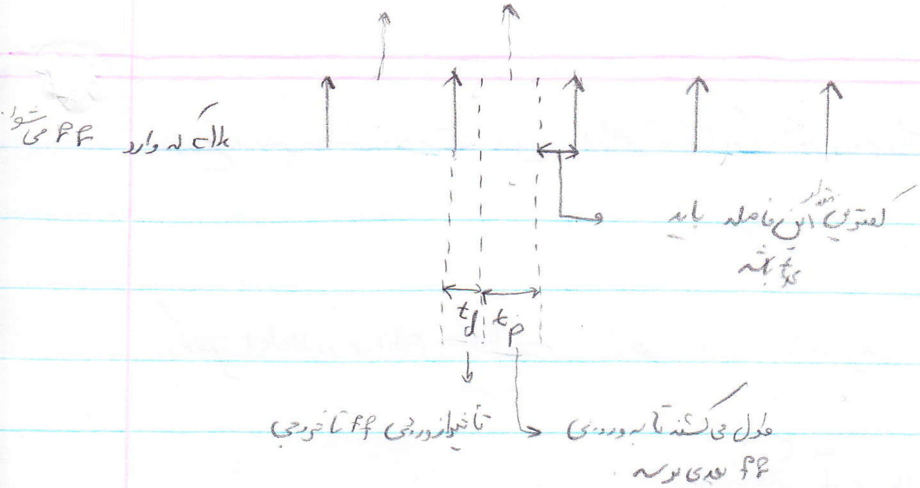




مدار مسئل

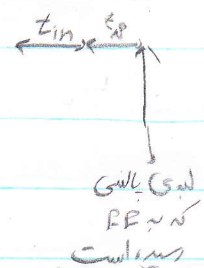
max frequency ← از روی t_p ساخته می‌شود

یا به عبارتی بالاترین ... که ورودی به FF اول رسیده و با یک تأخیر روی net قرار گرفته با تأخیری به FF دوم رسیده و این FF با تأخیری داده را روی خروجی می‌گذارد.

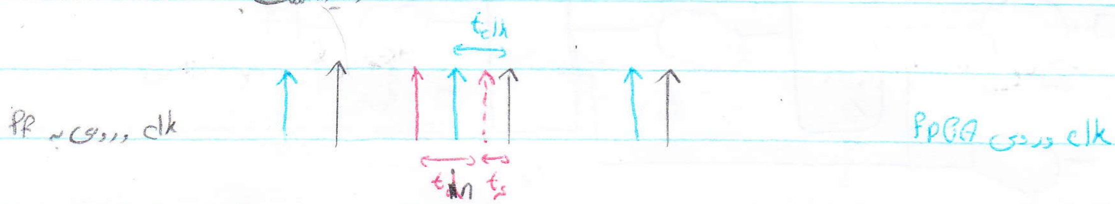


⇒ min period = $t_d + t_p + t_s$ min period = $t_p + t_d + t_s$

min offset in = چه قدر وقت قبل از لبه‌ی بالارونده‌ای که به FF می‌رسه لازم است داده‌ی ورودی در این جا valid شده باشه. (stable)

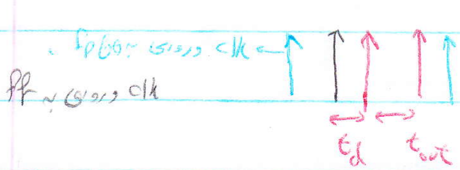


min offset in = $t_{in} + t_s$ یعنی حداقل این مقدار زمان قبل ورودی رو نگه داشته باشیم و تغییرش ندیم.



min off-set in = $t_s + t_{in} - t_{clk}$ ⇐ حال اگر ↑ refrence باشه

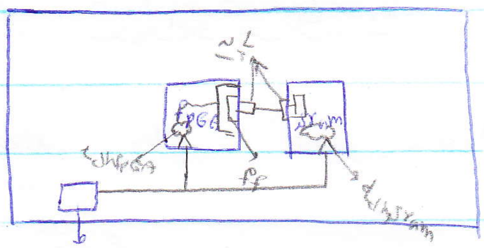
offset out = $t_d + t_{out}$ ، نسبت به clk ای که به FF وارد می‌شود.



offset out = $t_d + t_{clk} + t_{out}$

t_h = holdtime $\cong$ ۰ چه‌قدر وقت بعد از لبه clk ورودی به ff باید

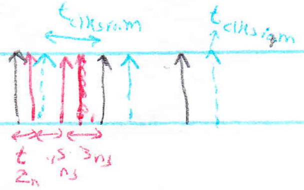
داده باقی بماند تا ff به‌درستی آن‌ها capture کند.



$t_{net} = 0.5\ ns$ ، $t_{s_{Sram}} = 3ns$ ، $t_{s_{FPGA}} = 2ns$

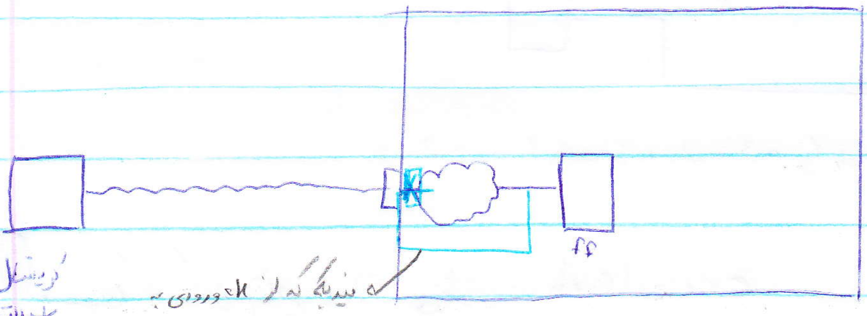
اگر $t_{clk_{FPGA}} = t_{clk_{Sram}}$ باشه یعنی لبه‌ها هم‌زمان به ffهای Sram و FPGA برسه: $\min_{period} = 2ns + 0.5 + 3ns = 5.5\ ns$

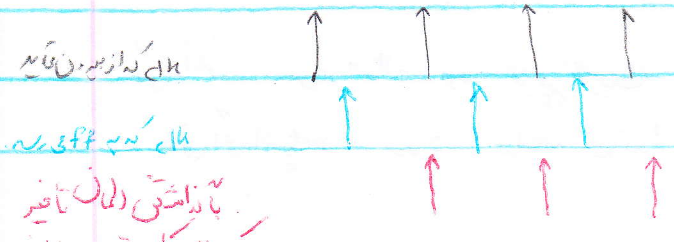
اگر $t_{clk_{FPGA}} = 0$ و $t_{clk_{Sram}}$ مقدار دارد $\Leftarrow$ $2 + 0.5 + 3 - t_{clk_{Sram}}$



اگر $t_{clk_{Sram}} \neq t_{clk_{FPGA}}$ مخالف صفر: $2 + 0.5 + 3 + t_{clk_{FPGA}} - t_{clk_{Sram}}$

در عمل سعی می‌شود $t_{clk_{Sram}} = t_{clk_{FPGA}}$ باشه.





phase combination = فاز دو clk هم یکسان می‌کند تا لبه‌های clk داخل FPGA با کریستال اسیلاتور هم‌زمان شود.

imp / place and rout / analize

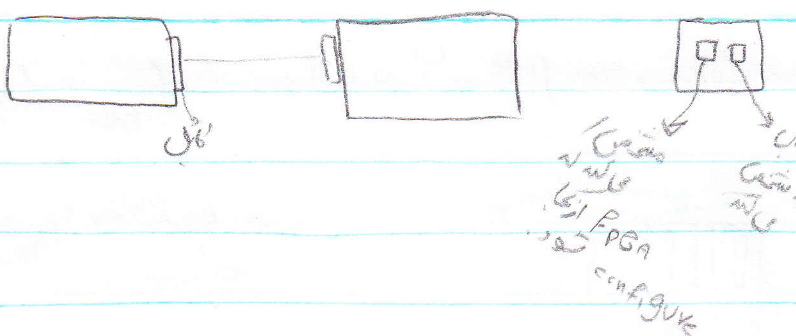
اگر در tools ، FPGA Editor را run کنیم و روی پنجره analize روی link کلیک کنیم

مسیرها در FPGA نشان می‌دهد.

مثلا net های carrychain همگی عمودی‌اند. لذا چیدمان عمودی است نه افقی.

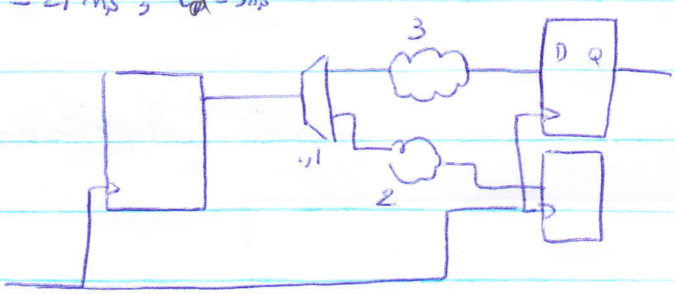
اختلاف: clk90 = فاز ۹۰ ، clk0 = هم‌فاز و هم‌فرکانس اسیلاتور ، clk2x = دو برابر | clkDV = فرکانس تقسیم بر دو

دو روش برای program کردن هست: 1: prom (↓ xcf4) 2. کابلی که حال پرتابل است.



ماکزیمم فرکانس کاری:

$t_{su} = 2.1ns$ و $t_d = 3ns$



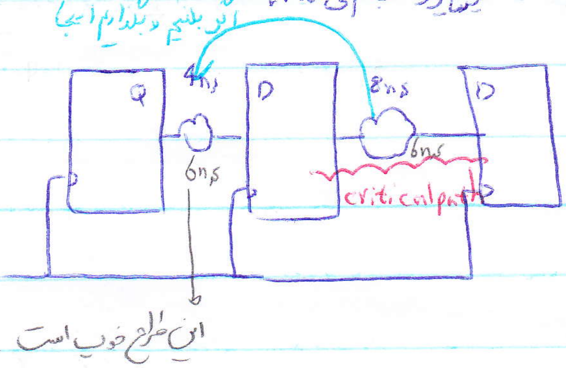
$t_d + t_{mux} + t_{route_p} + t_{su} = 3 + 0.1 + 3 + 2.1 = 8.2ns$

مسیری که ماکزیمم تاخیر را دارد critical path می‌شود.

اگر تاخیر برابر ... clk=9ns بود. هیچ تغییری نمی‌کرد.

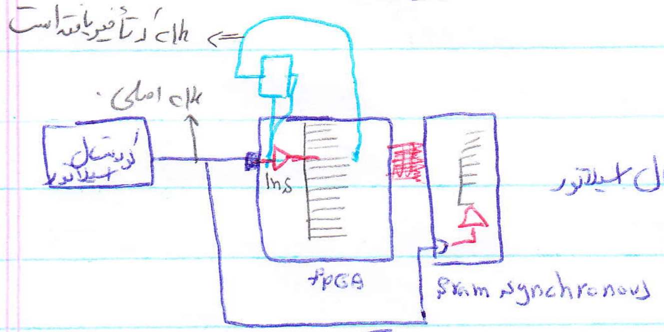
SQ clk: حال اگر مال net_2 بود ۵ بود برابر می‌شود!

بهترین design آن است که همزمان همه انجام دهند. اگر تکه‌ای از کار 3ns هم قبل از FF1 می‌گذاشتیم بهتر بود! این عمل هم سنتزایزر انجام می‌دهد:



retiming: عملی که سنتیسایزر می‌کند تا مینیمم تأخیر ایجاد شود.

همه‌ی نتایج یکی است اما نتایج function و post سنتیسایزر متفاوت است!

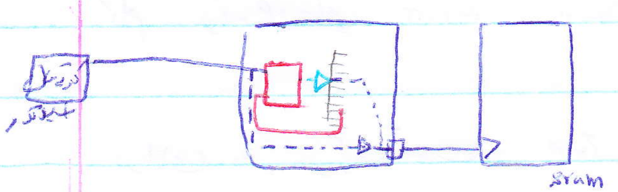


* clkDLL:
علت استفاده از بافر: تا اثر بار روی کریستال اسیلاتور نیفتد.

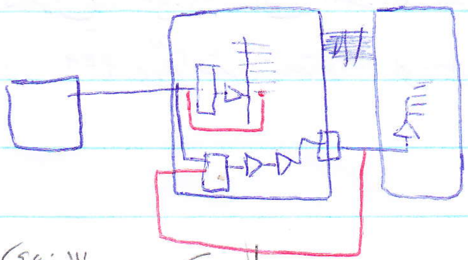
اگر کلاک اصلی با clk تأخیر یافته یکسان نبود یک عنصر تأخیر اضافه می‌کند تا با یک سیکل عقب افتادن یکسان شود. (از تأخیر net نیز صرف‌نظر می‌شود)
کارشناس

به این المان که تأخیر ایجاد می‌کند Delay locked loop گفته می‌شود. این المان کارهای دیگری هم انجام می‌دهد.

اگر clksram کمی از clk زودتر FPGA بیاید.
این حالت برای تأخیر پیاکوین بد است.



اگر از سر بافر به پایه وصل شود هم بد است چون تأخیر پایه هم علت بار خارجی بسیار بد است!



کاربرد دیگر clkDLL و یعنی همان لبه‌ی بالارونده که FPGA دارد دیگران هم کار کنند.

با این فیدبک لبه‌ی بالارونده‌ی پایه‌ی خروجی FPGA و لبه‌ی بالارونده‌ی درون FPGA یکسان می‌شود

* نرم‌افزار ISE و بکار بردن اتراین clkDLL :

```
module counter(clk,reset,out);
 input clk,reset;
 output
 reg [2:0] r;
 always @ (posedge clk)
   if(reset)  r<=0;
   else r<=r+1;
assign out = r[2];
endmodule
```

با سنتز کردن و Imp کردن.

حال یک rightclk کردن روی ... پروژه یک test fixture می‌سازی دیگه که کدام برنامه‌ها را می‌خواهی test کنی. که خودش یکی از برنامه‌ها نوشته است.

در بالای ... پروژه دو تست Imp و Sim دارد. با انتخاب sim دیگر گزینه
Isim simulator: post rout simulation . Behavior

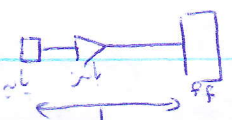
uut = clkBUFG

برای اضافه کردن clkDLL:

```
wire clktoDLL, clktoBUFG, myclk;
IBUFG inputclockinc(.I(clk), .O(clktoDLL))
DCM DCMIns(.clkin(clktoDLL), .clk0(clktoBufG), .clkFB(myclk));
BUFG BUFGIns(.I(clktoBufG), .O(myclk));
```

سپس به جای clk در always ها myclk می گذاریم!

با گذاشتن این clkDLL زمان clk to out کاهش می یابد، پریود تغییری نمی کند. تأخیر ورودی افزایش می یابد!



این clkDLL تأخیر را از بین می برد.

clkFX = فرکانس هم فرکانس ورودی ضرب در $\frac{m}{n}$ است. که m, n را می توان تعیین کرد.

تقسیم کننده فرکانس clkDV

فرکانس clkFX ارتباطی به Feedback ندارد. اما clk0 و clk90 و clk180
clkDV احتیاج به این فیدبک دارد.

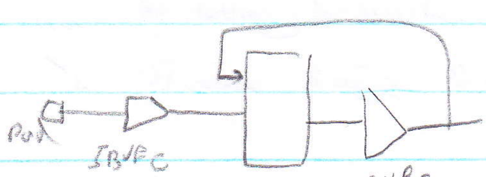
برای ایجاد فاز و شیفت می توان از clkDLL استفاده کرد. طرز ساخت: clk2x = 0, 180, 90

کوئیز:

1. عمل retiming و مدار؟ لم بالانس کردن

2. تأخیر offset out

3. BUFG ها.



* امبدد سیستم => سیستم‌های که مالتی تسک هستند. مهم‌ترین پارت این سیستم CPU است. برای پیاده کردن CPU:

همان طور که قبلاً گفته شده است دو نوع CPU هستند: 1. از المان‌های FPGA استفاده شود: نرم
2. خود CPU هست: سخت.

1. PicoBlaze : 8 بیت، soft core

سه نوع CPU های Xilinx ساخته: 2. MicroBlaze : 32 بیت، soft core

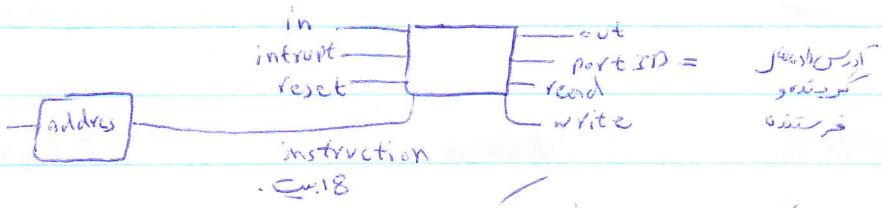
3. power PC : 64 بیت، hard core

دو نوع اول را در هر نوع FPGA می‌توان پیاده‌سازی کرد.

نوع سوم در این FPGA ها یافت می‌شود:
Virtex-II pro
Virtex-4 Fx
Virtex-5, 6 Fx

برای پیاده‌سازی دو نوع core آخر احتیاج به Embeded development kit است = EDK.

شرکت Altra => NIOS II که 32bit نرم است: می‌توان با استفاده از این CPU ها می‌توان لینوکس را بالا آورد.



در حد 60 اسلایس استفاده می‌کند.

که 16 رجیستر 8 بیتی و 64 بیت برای ذخیره‌ی داده.

دستور test = دو رجیستر و and می کند آن عبارت Zero Flag را set کند. اگر تعداد یک ها فرد بود دوباره set می شود وگرنه هیچ. تغییری روی رجیستر نمی دهد.

در Read and Write هر دو سیکل عمل انجام می شود.

Writstrobe = وقتی بخواهد بیرون بگذارد. وقتی بخواهد در خودش Read، save کند.

باید از سایت xilinx باید Download کرد.

باید در فلدر HDL ، فایل لازم و Add کرد.

```
توضیح ; this is a simble
تعریف مقدار  Constant output_port1 , 1
شروع برنامه  start
        Load  S0 , 00        load S0 مکان مقدار 0 در reg
loop1:
        output  S0, output_port1     مقدار S0 را به output
        ADD  S0 , 1          S0 + 1
        Compare  S0 , 05     Flag Z اگر برابر باشد
        jump  Z, loop1_cont  set می شود.     →  if (S) → loop1_cont
                                               else → loop1
        jump  loop1
loop1_cont:
        load  S0 , 00
        jumb  loop1
```

با save کردن این برنامه در Folder ، asm ، فایل های اسمبلر را ذخیره کرده ایم که پسوند PSM است.

خالی که می‌سازد وریلاگ می‌سازد: با بارگذاری ISE و ساخت پروژه‌ی جدید:

add source: ۱. kpcm را در فلدر HDL داشتیم. و بعد که ساخته شده،

حالا باید یک کدی بنویسیم که صدا کند.

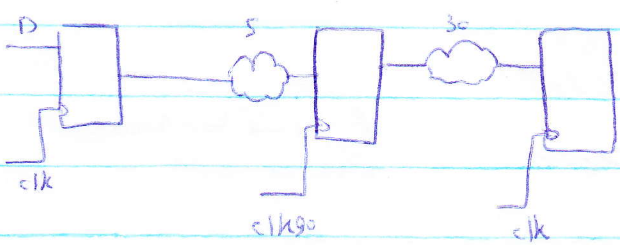
```
module top(clk, reset, out);
    input clk, reset;
    output [7:0] out;   → 8 بیت pico output
    reg [7:0] out;
```

اسم my_code را حافظه صدا می‌کنیم، و حافظه‌ها به picoblaze وصل می‌کنیم.

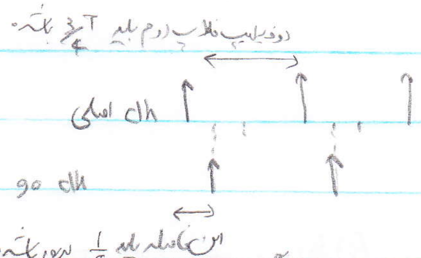
```
always @ (          )
    if reset out <= 0;
    else if (writestrobe && portID0)
        out <= outport;
```

tester را هم می‌نویسیم؟

کوئیز: الگوریتم پریود؟



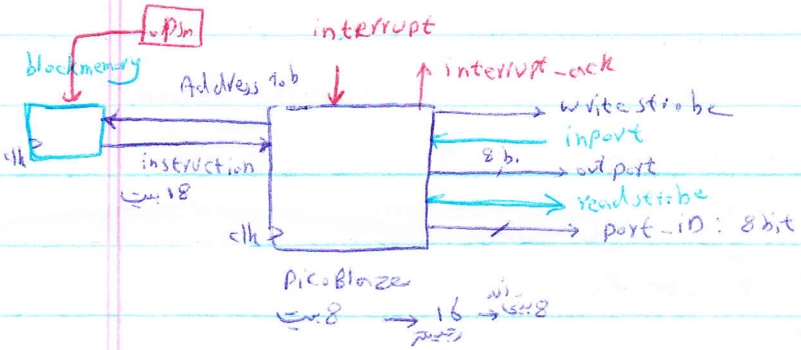
$t_{su} = 2ns$, $t_{delay} = 3$



$\frac{1}{4} T \geq 10ns \rightarrow T \geq 40ns \Leftarrow 3 + 5 + 2 = 10ns$ دو فلیپ فلاپ اول =>

دوفلیپ فلاپ دوم ⇐ 30 + 2 + 3 = 35ns ⇐ $\frac{3}{4}T \geq 35ns \rightarrow T \geq 46$

$\Rightarrow T = 46\ ns$



برای ساخت Blockmemory باید اسمبلی نوشت با پسوند psm. که آن را به یک اسمبلر به نام kcpsm می‌دهیم
که مبرد Blockmemory می‌ریزد.

در این حالت دو تا AddressBus دارد.

با portID مشخص می‌کنیم که با کدام بخش logic می‌خواهیم حرف بزنیم.

وقتی چیزی روی outpot هست writestrobe را تیک می‌کنیم.

با چه دستوری writstrobe را 1 می‌کنیم؟ output s0 , 01 ⇒ port id = 01

outport = s0

writestrobe به مدت 2 سیکل 1 است.

output s0 , 05 → port id = 05

outport = s0

writestrobe = 1

ورودی پورت inport و readstrobe ⇐ Input s0 , 01 ⇒ portID = 01

آنچه روی inport است توی s0 ریخته می‌شود

و readstrobe 1 می‌شود.

بعداً از readstrobe استفاده می‌کنیم.

وقتی interrupt رخ می‌دهد به آخرین آدرس حافظه پرش می‌کند.

```
ISR: ≡
return I
0xff [jump ISR]
```

interruptack رو زمانی می‌دهد که بگوید من interrupt شما رو دیدم.

که دو دستور interrupt enable و disable مهم است که صدا می‌کند.

تعدادی از portID:

تو را است تک باز ← In2 ، In1 یکبار assign in-port = (port id = 2) ? In1 : In2

در دستور اصلی همین پورت که انتخاب کردی.

عملی

```
constant   input-port4 , 04   ——(در وریلاگ)——>   assign in-port = (portID=2) ? in1 :
                                                          (portID= 3) ? in2 :
                                                          (portID=4) ? in3 : 0
```

پس در این حالت in3 که از بیرون می‌گیرد در s2 ذخیره می‌کند.

```
start ← input   s2 , input-port4
        load    s5 , s0
        load    s6 , s1
        ADD     s6 , s5
        SUB     s1 , s0
```

حاصل جمع s6 ، تفریق s1.

```
compare  s2 , 00   ——>
jump  Z , some-where
output  s1 , output-port
jump  end
somewhere  output  s6 , output-port
```

در این حالت اگر In3 صفر است حاصل جمع که بیرون اگر In3 نبود حاصل تفریق که دهد.

برای اینکه هر دو را در خروجی ببریم ←

```
constant  output-port5 , 05

start  output  s6 , outport5
       output  s1 , outport1
```

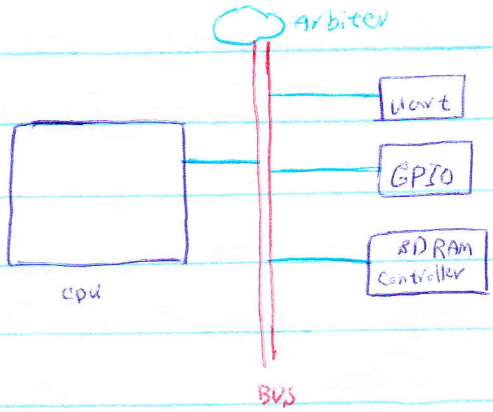
در وریلاگ ←

```
if (writstrob) begin
        if (portid == 01)
                out <= outport
        if (portid == 05)
                out2 <= outport
```

هر دستور پیکو بلیز 2 سیکل ساعت طول می‌کشد.

- Microblaze
- powerpc → IBM

* ارائه‌ی Embeded system



اگر pic مهم بود با port ID مشخص می‌کنیم که به کدام یک دسترسی می‌خواهیم.

مراحل انجام transaction مثلاً در Read.

Master = مشخص می‌کند که چه transaction initiate می‌کند. از کجا مثلاً Read شود.

در اینجا Master ها cpu است.

BUS در SOC.

system on a programmable chip ⟹ SOC

سرعت BUS در روی برد کمتر از آن حالت integer است.

تفاوت BUS که روی برد است با یک Buffer سوهر جیپ که قرار است کار انجام دهد هستند.

چون مشکل routing داریم. اما در FPGA ها mux داریم که مشخص می‌کند کدام کارها انجام شوند و
routing زیادی داریم. اما ماژول‌های مختلف را وصل می‌کند ⇐ SOC BUS که
در FPGA چون Buffer ها حذف می‌شود سرعت بسیار بالاتر است
همچنین در BUS در FPGA زیاد است و محدودیت ندارد. اما روی برد این گونه نیست. که
این به علت امکان نداشتن routing محدودیت است.

تنها چند شرکت SOC BUS ارائه کرده‌اند

مانند core connect که مال IBM است.

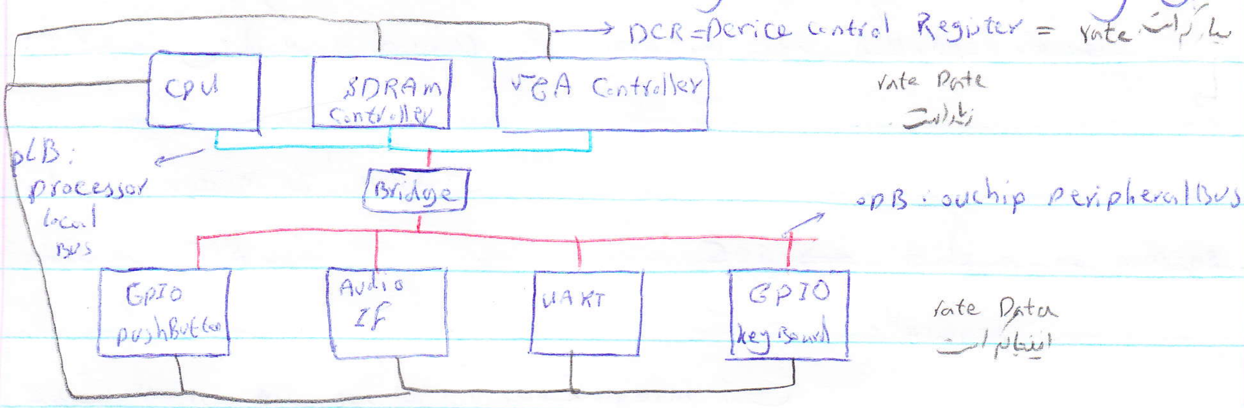
و AMBA : Advanced micro. Bus Architecture مال کمپانی ARM است که مال شرکت philips است. یعنی یک design company ، تولیدکننده‌ی سیلیکونی نیست ولی شرکت‌ها از آن استفاده می‌کنند.

یکی دیگر هم wish Bone است خاصیت آن open source است، برای همین هر کمپانی که مثلاً uart ای می‌سازد، چیز لازم برای wishbone ای می‌سازند که می‌توان از آدرس زیر گرفت.

interface

www.opencores.org

ماژول‌های مختلف در fpga سرعت‌های متفاوتی روی BUS دارند



یک انتقال یک بیک بین peripheral ها و FPGA controller است.

در این حالت سرعت و حجم انتقال داده‌ها بین ماژول‌ها متفاوت است مثلاً بین cpu و sdram زیاد است.

پس دو نوع BUS می‌سازیم تا بین آنها که rate کم دارند و یکی بین آنها که rate زیاد دارند.

سپس یک Bridge داریم تا آن BUS که rate کم دارد به BUS rate بالا متصل شود.

Bridge در واقع یک slave برای pLB است و یک master برای OPB است.

172.16.3.177
172.16.3.80

12

Xilinx platform ~~system~~ studio

• Base system Builder wizard → یک سیستم حداقل Base گویند.

سپس بورد را انتخاب می‌کنی.

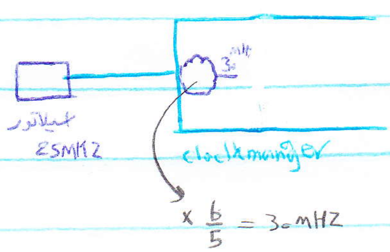
اگر mpmc که حافظه است جدا از PLB وصل باشد خوب است چون PLB فقط در اختیار ماژول‌های دیگر است و سریع عمل می‌کنند.

فرض کنید یک ماژول به طریق زیر طراحی کنیم:

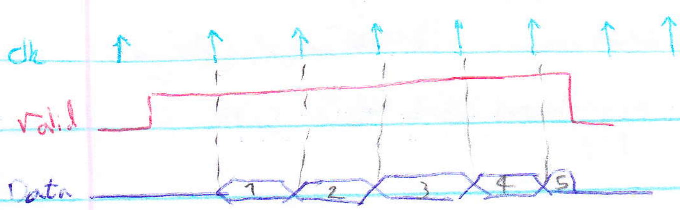
Valid همراه داده،
→
16 Bits
→
30MHz

→
uart
115200 bps

داده‌ها یه لحظه‌ای و منظم به وقت نمی‌آید. rate ورودی 115200 بیش هیچ داده‌ای از بین نمی‌رود. اما در یک لحظه rate خیلی زیاد است.



حالت خوب آن بود که clk همزمان با Data داشتیم. مثل DDR SRAM در حافظه. به این design ها که clk با خود داده می‌آید Source synchronous Data.

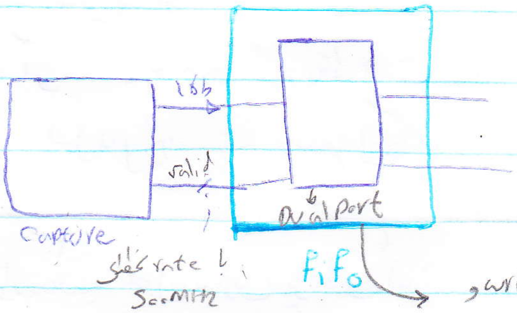


یعنی هر پالس وقتی Valid 1 است Data عوض می‌شود.

اگر مثلاً 4 برابر فرکانس فرستنده نمونه‌برداری کنیم دیگر مشکل transient نداریم. حال با گذاشتن ماژول می‌توان مثلاً بگوییم بالای لبه‌ای که Valid 1 بود [illegible] در هر دو لبه بعد از دیدن Valid capture کن.

یا مثلاً با اولین لبه‌ی بالارونده که valid = 1 بود با اولین لبه‌ی پایین‌رونده capture کنیم.

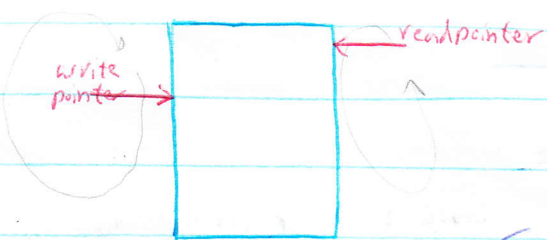
مرحله‌ی بعدی آن است که داده‌های capture شده را جایی save کنیم و بعداً آرام بخوانیم.



چون ماژول‌های write و read جدا داریم،

همانطور که قبلاً داشتیم Block memory
- Single port
- Dual port

در این حالت write همزمان و read همزمان نیاز است. که هم write و هم read هر پورتی دارند که آدرس را مشخص می‌کند. 1. write pointer کجای حافظه بنویسیم 2. readpointer از کجای حافظه بخوانیم.



با نوشتن counter دو سیگنال تولید می‌کنیم full و empty. اگر حافظه پر بود داده را نمی‌نویسی و اگر خالی بود نمی‌توانیم => نویسنده به full و خواننده از empty تبعیت می‌کند.

هم ارزی برای همین که clk نویسنده با خواننده یکی باشد. اگر clk ها یکی باشند می‌گوییم Synchronous fifo.
اگر clk ها یکی نباشند Asynchronous fifo می‌گوییم.

پس با وجود full هیچ داده‌ای overwrite نمی‌شود و داده‌ای از بین نمی‌رود.

طراحی ما از بالا به پایین است: HDL designer ← طراحی‌های پیچیده را انجام می‌دهیم

بسیار شبیه Active HDL است.

این نرم افزار صرفاً برای design است.

new project / system design (ساخت پروژه) (اسم فولدر) system1 (اسم پروژه)

سپس می‌پرسد که چه کاری می‌خواهی بکنی. که اینجا یک Block diagram است.

Finish.

HDL در محیط خود

file / add / existing file / kcpsm / verilog / kcpsm.v
uart_rx
_tx

حال با Drag کردن به بلوک دیاگرام، بلوک‌ها را می‌کشی.

طراحی fifo ⇐ شکل بلوک را اول تولید می‌کنی. حال code HDL را می‌نویسی:

```
module myfifo ( Datain, wen , full , Dataout , Rden , empty , Datacount , clk, reset
```

(wen: وقتی 1 است آنچه در Datain است روی fifo می‌ریزیم) (Rden: 1 که باشد خوانده می‌شود)

```
parameter fifowidth = 16 ;
parameter fifodepth = 256 ;
parameter ADDRwidth = 8 ;  → 256 را با 8 bit می‌خواهیم آدرس دهی
input [fifowidth-1:0] Datain;
input clk, reset      wen;
output                full;
output [fifowidth-1:0] Dataout;
input                 Rden;
output                empty
output [ADDRwidth:0]  Datacount;
```

ادامه‌اش بعد.

حال با ابزار coregen، یک dualport را انتخاب کن که یک فولدر جدید ساخته و با فعال کردن coregen.

در core gen : file
project option / sport3 / verilog /

در کاتالوگ → memories_Block memory Generator / Dual port ram / Generate

فایل‌ها در Folder که ساختیم ما باز می‌کنیم core ها باز کرده و آن قسمت از
فایل که لازم نوشتن است به پروژه خودمان اضافه می‌کنیم

adrb جای addra است و عمل می‌کنیم

```
reg [ADDwidth-1:0] writepointer, readpointer;
```

wen برای dout b به Dataout ، و clka و clkb جای clk خودمان است
↳ wen && (!full)

```
reg [ADDwidth:0] Datacount;

always @ (posedge clk or posedge reset)
    if (reset) begin
        writepointer <= 0   readpointer <= 0;
    else begin
        if (wen && (!full))
            writepointe <= writepointer + 1;

        if (ren && (!empty))
            readpointer <= readpointer + 1;
    end
always @ (posedge clk or posedge reset)
    if (reset)
        Datacount <= 0;
    else begin
        if (wen && (!full)) Datacount <= Datacount + 1;
        else if (Rden && (!empty))
            Datacount <= Datacount - 1;
        else   [illegible]
```

این قسمت [illegible]

(یعنی fifo پر است)

```
assign full = (Datacount == 256) ? 1 : 0;
assign empty = (Datacount == 0) ? 1 : 0;
endmodule
```

(محل سیگنال‌ها در آخر کد) — (یعنی fifo خالی است)

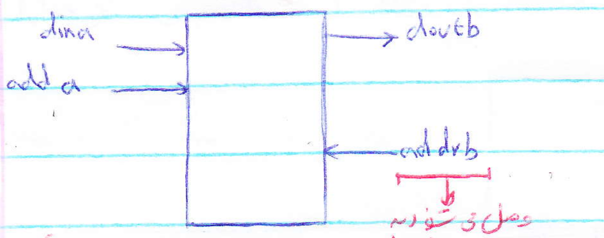
نباید از always استفاده کرد زیرا در این صورت یک سیکل بعد full تغییر می‌کرد و یک سیکل دیتا را از دست داده‌ایم. در واقع باید full با سیکلی که Datacount تغییر کند.

کوئیز:

CoreConnect = یک Bus مبتنی بر powerpc است و شامل دو باس است: DCR و PLB.

* ادامه مطالب قبلی:

در اینجا همواره از بلاک‌های حافظه‌ای که همواره داده روی خروجی هست زمانی خواندن که خودش است که readpointer تأثیر می‌گذارد با توجه به خط قرمز صفحه قبل!



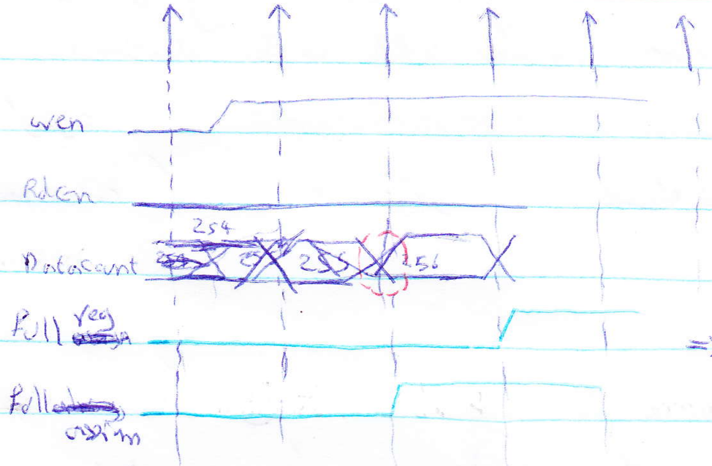
این حالت تقریباً برای wen هم رخ می‌دهد! اما ما همواره write نداریم و همواره read داریم!

readpointer ⇐ هرگاه readenable آمد readpointer را یکی زیاد می‌کنی! پس احتیاجی به خود سیگنال readen نیست که به fifo ببرد.

متغیر Datacount تولیدکننده empty و full هستند. اگر هم wen و هم readen فعال باشند باید در واقع Datacount هیچ تغییری نکند. لذا باید که این‌ها را بنویسیم:

```
else begin
    if (wen && (!full) & RdEn && (!empty))
        Datacount <= Datacount
    elseif (wen && (!full))
        Datacount <= Datacount + 1
    elseif (RdEn && (!empty))
        Datacount <= Datacount - 1;
    else
        Datacount <= Datacount .
```

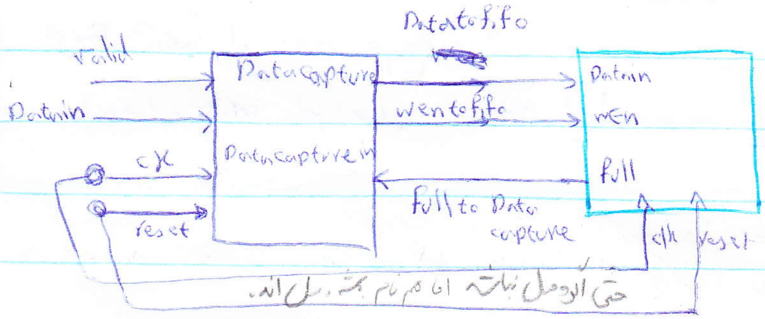
اگر بجای assign از always استفاده کردی ⇐



⇐ اینجا می‌بینی که full باید باشه
یک سیکل عقب‌تر از الان

همین اتفاق برای empty هم هست!

ادامه‌ی design با add کردن وریلاگ myfifo، Addcomponent:



برای نوشتن code capture ⇐

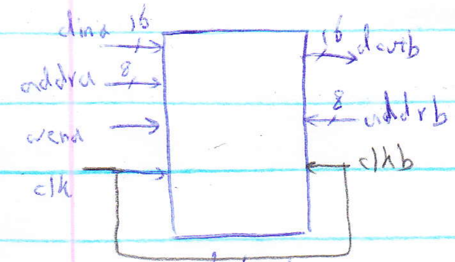
! بهترین کار استفاده از کشیدن شکل موج است : استفاده از نرم افزار timingdesigner:

نوشتن کد — نباید با دو سیکل قبلی خود برداری کنی / اگر یکی دوباره همان R بود تکرار
سیگنالی که درست بود را داشته برداریم

```
wire capturenow;
assign capturenow = (valid) && (validr) && (!capturedR) && (!capturedRR)
reg validR, capturedR, capturedRR;
always @ (posedge clk)
    if (Reset) begin
        validR <= 0;
        capturedR <= 0;
        capturedRR <= 0;
```

```
else begin
    validR <= valid;
    capture <= capturenew
    captureRR <= captureR;
  end
```

— کوئیز: با استفاده از یک حافظه dualport بعرض ۱۶ بیت وعمق 256 خانه یک مدل stack طراحی کنید



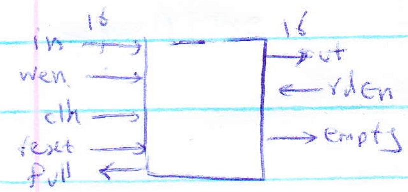
dualport

در FPGA نمی‌توان مثلاً حافظه در حد گیگا داشته باشیم زیرا resource های داخل FPGA کافی نیست؛ با یک sram به بیرون اضافه کنیم در این حالت تنها یک read می‌کنی یا write! هم زمان نمی‌شود!

شرکت‌هایی که حافظه تولید می‌کنند ⇐ cypress . و micron

Dram = samsung و hynix , micron

flash ⇐ toshiba و intel ,



Stack

یک pointer کافی است ⇐ موقع read یکی کم و موقع write یکی زیاد.

بعد از تعریف ماجول

```
reg [8:0] datacount;        reg [7:0] stackpointer;
  always @( posedge clk or negedge reset)
     if(reset) datacount <= 0;  stackpointer <= 0;
     else begin
       if ( wen && (!full) ) begin
            if (rden && (!empty))  stackpointer <= stackpointer
                                   datacount <= datacount;
               stackpointer <= stackpointer + 1;
            else   datacount <= datacount + 1;
                         end
                  stackpointer <= stackpointer - 1;
  elseif (rden && (!empty)) datacount <= datacount - 1;
       stackpointer <= stackpointer
  else   datacount <= datacount;
       end
```

(datacount [8])

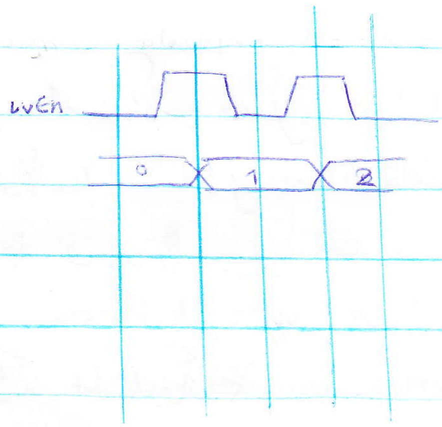
assign full = (datacount == 256)?1 : 0;

assign empty = (datacount == 0)?1 : 0;

(!datacount)

(~| datacount)

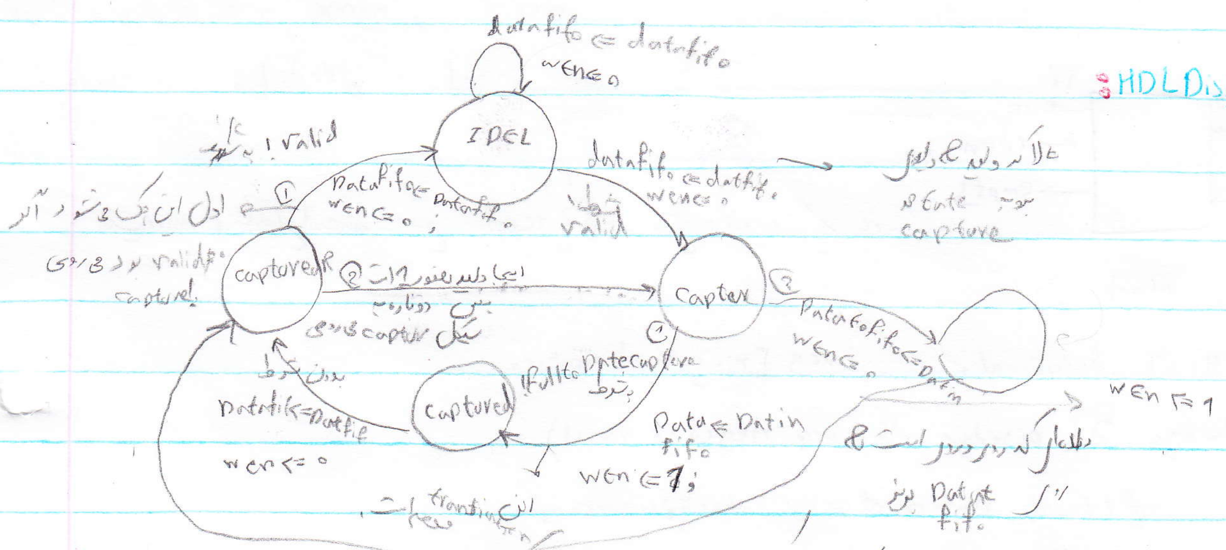
nor کلی & صفر بیت‌ها



addrb = stackpointer - 1;

addra = stackpoint;

HDL Disign



با یک کلاک نمونه‌برداری کرده و با دو کلاک کاری نمی‌کنی دوباره نمونه‌برداری می‌کنی.

با اولین کلاک که valid 1 بود از IDel به capture می‌ری با دومین کلاک capture و

با دو کلاک بعد کاری نمی‌کنی. این عمل تا جایی ادامه می‌دهی که valid صفر شود.

اگر حالتی باشه که در کلاک‌ها می‌خواهی capture کنی fifo پر باشه چطور

دو کلاک ساعت وقت داری.