تا سر این صفحه scan شد.

جلسه نوزدهم:

ساده‌ترین مدار در مدار ترتیبی فلیپ‌فلاپ است. یک مدار با یک ff، implement می‌کنیم.

```
module simpleff (In, out, Clk, Reset)

reg out;
always @ (posedge clk or posedge Reset)
if (Reset)
    out <= 0;
  else
  out <= In;
endmodule
```

برای implement کردن یک پروژه جدید در ISE ایجاد می‌کنیم. و با Add Source، کد وریلاگ فوق را به پروژه اضافه می‌کنیم. حال implement می‌کنیم

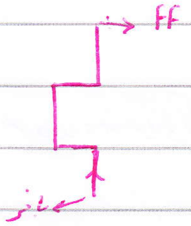
به سمت FPGA Editor می‌رویم.

پایه خروجی FPGA با نام PAD مشخص شده است.

IBUFG: بافری که پالس ساعت وارد می‌شود.

در FPGA بعضی از پایه‌ها را می‌توان به clk متصل کرد. (فقط پایه‌های متصل به IBUFG)

سمت قرمز رنگ، net، کلاک اصلی است که در کل FPGA پخش شده است به شکل مسیر زیر:



BUFG : بافری که به FPGA Routing Resource ها متصل می‌شوند.

لاین‌های مربوط به net کلاک از سایر لاین‌ها جدا است و باعث می‌شود سرعت بالاتری داشته باشیم.

می‌خواهیم مشخص کنیم با فرکانس 200MHz کار می‌کند.

User Constraint ⟶ Creat timing Constraint

clock Domains مربوط به کلاک

input

output

سیستم clock را به صورت اتوماتیک تشخیص می‌دهد.

Clk ⟶ double click ⟶ specify time : period of clk

فایل UCF را می‌توان edit کرد. دوباره implement design را run می‌کنیم.

فرکانس ما را ماکزیمم سرعت سوییچ FF تعیین می‌کند.

Console → place & Route report

slack ، تفاضل بین تاخیری که ما می خواهیم و تاخیری که به آن می رسیم.

slack { >0 برآورده می شود
<0 نمی شود

می خواهیم روی تأخیر ورودی و خروجی این ff ، constraint بگذاریم.

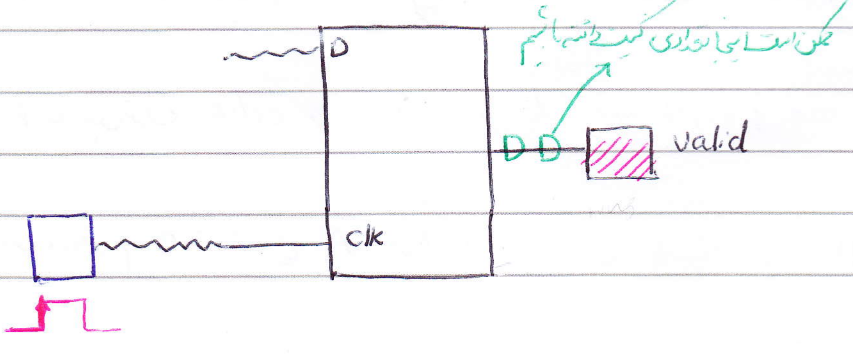
طوری آن را implement می کند که تأخیر مسیر ورودی بیش از 10ns نشود.
از ff تا پایه خروجی

synthesizer با نگاه کردن به event ها مشخص می کند کدام سیگنال است که بقیه سیگنال ها با آن trigger می شود.

FPGA ، optimum پیاده سازی می کند و فقط می خواهیم بسنجیم که آیا به requirement ها می رسیم
اند.

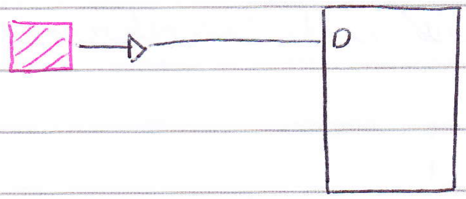
output → Double Click

*External clock to pad(offset out)
فاصله زمانی لبه بالا رونده clk روی پایه خودش تا زمانی که داده در pad خروجی قرار می گیرد.



! هیچ وقت * را بیشتر از پریود clk نمی‌گذاریم .

inputs → { In → Double Click
Reset



Clock Edge → ✓

single Data rate (SDR)

Double 〃 〃 (DDR)

system synchronous: clk سیستم با FPGA برابر است .

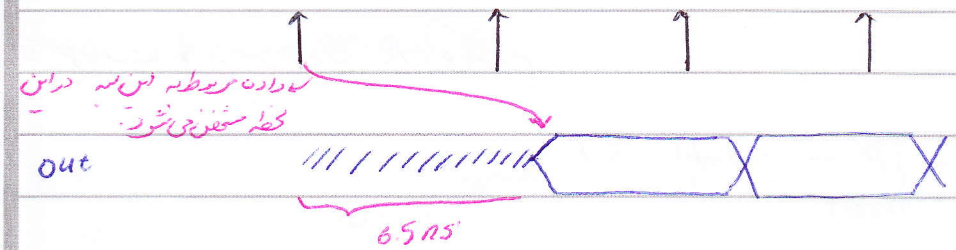
External setup time (offset in)

تأخیر از پایه FPGA تا لبه بالارونده clk

! معمولاً constraint را همیشه نصف پریود بگذارید .

حال دوباره Design را implement می‌کنیم .

حدودی constraint هایی که meet شده * می‌گذارد .



این حالت خوبی در طراحی نیست. باید پریود کلاک را افزایش دهیم.

معمولاً تأخیر خروجی نسبت به setup time عدد بزرگتری است.

تأخیر خروجی تابع مسیر clk، تأخیر ff، تأخیر net (خروجی ff)، تأخیر پین که در راه خروجی است و تأخیر بار خارجی خود پایه FPGA است.

تا زمانی که constraint برآورده نشده، نمی‌توان در مورد اعداد داده شده صحبت کرد. آنها شرط تحمیلی هستند و می‌گویند اگر این طوری می‌شد خوب می‌شد.

constraint های مختلف روی هم تأثیر می‌گذارند.

Black Bird:

سال ۱۹۶۲ تولید و در سال ۱۹۶۴ اعلام وجود کرد. تا سطح جو زمین می‌تواند پرواز کند.

سرعت: ۳۵۰۰ km/h

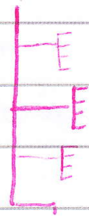
سیستم دفاعی: گریز سرعت

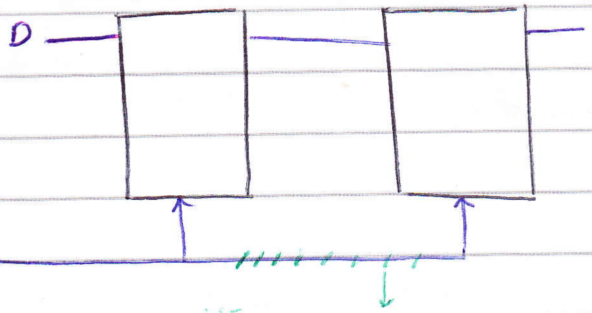
حالا می‌خواهیم یک 256bit counter پیاده‌سازی کنیم.

```
module simplecounter(



always @ (posedge clkN or posedge Reset)
    if (Reset)
        r <= 0;
        else
         r <= r+1;
 assign out: r[255];
endmodule
```

FPGA Editor

256 فلیپ‌فلاپ زیرهم قرار دارد. clk به وسط FPGA منتقل می‌شود. چرا؟





فرض این net تأخیر دارد.

(1ns)

6ns 200MHz

* الآن که این فلیپ فلاپ کلاک خورد این لبه بالارونده 1ns بعد به فلیپ فلاپ بعدی می‌رسد.

! چون می‌خواهیم تمامی ff ها تا می‌شود همزمان کلاک بخورند، کلاک را به این صورت پخش می‌کنیم.

اگر در FPGA Editor، DELAY را نشان دهیم تأخیرهای ورودی تا ff ها را مشاهده می‌کنیم.

1.035
1.014
} همه ff ها دقیقاً همزمان clk می‌خورند.

clk، تأخیر در حد 1ns است.

و skew در حد 0.02ns است.

clk skew:

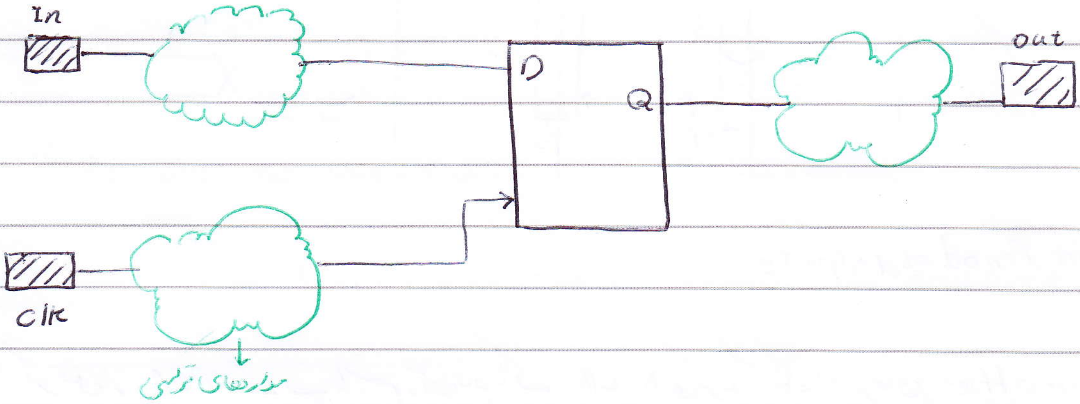
تأخیری که با آن تفاوت تأخیر clk به بقیه قسمت‌های مدار می‌رسد.

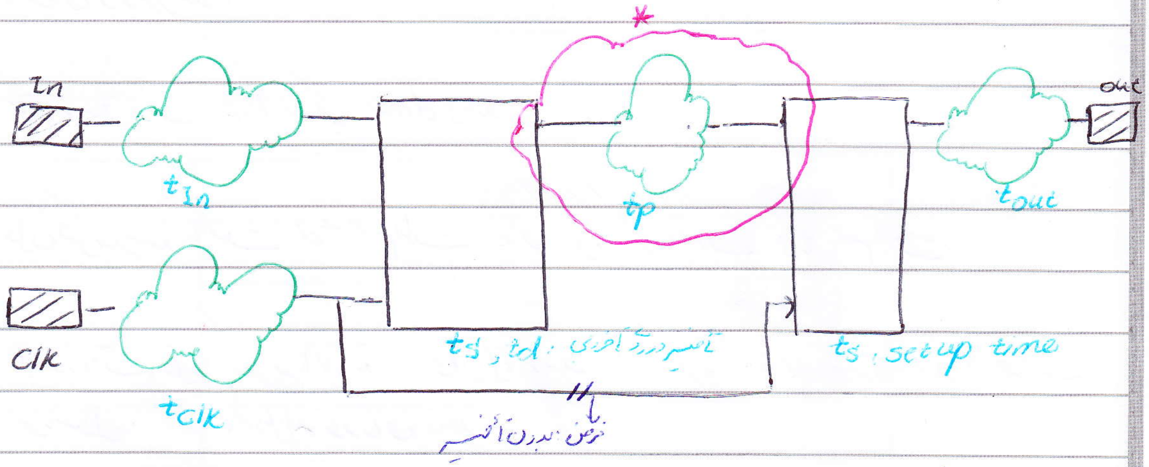
جلسه سیم:

پریودی که مدار می تواند با آن کار کند هیچ ربطی به تأخیر ورودی و خروجی ندارد.

کوییز:

در یک طرح FPGA، بیشینه مقدار offset in (یعنی تأخیر ورودی) 5ns و بیشینه مقدار offset out (تأخیر خروجی) 7ns است. ماکزیمم فرکانس کاری این طرح چه قدر می تواند باشد؟

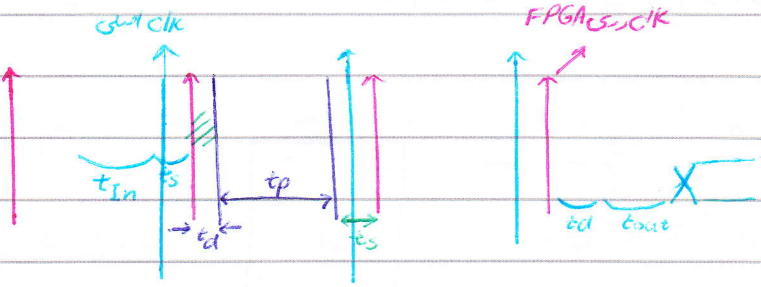

In
D
Q
out
clk
مدارهای ترکیبی


*
In
t_{In}
tp
out
t_{out}
clk
t_{clk}
ts , td : تأخیر درز آخری
ts : setup time
فرض : بدون تأخیر

ماکزیمم فرکانس که مدار می تواند با آن کار کند از * به دست می آید.

از زمانی که clk می خورد یه مدتی طول می کشد تا داده به خروجی اولین ff برسد بعد از آن به اندازه tp طول می کشد تا داده به ورودی دومین ff می رسد و مدت زمانی طول می کشد تا دومین ff ، داده را capture کند.

td : از وقتی که clk آمد چه قدر طول می کشد داده از ورودی به خروجی برسد



$$\text{min Period} = t_d + t_p + t_s$$

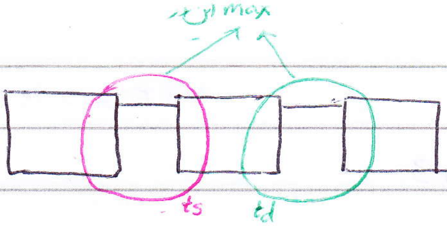
اگر قبل از FPGA یک چیپ داشتیم و آن دو با یک clk کار می کردند آنگاه تأخیرهای offset ورودی و خروجی نیز مهمتر باشد.

چون In را یک chip دیگر drive می کند؛

برای کسی که فرستنده است td مهم است و برای کسی که گیرنده است ts مهم است.

چه قدر وقت قبل از اینکه لبه بالارونده روی ff برسد، باید داده stable شده باشد؟ جواب این سوال را offset in به ما می دهد.

$$\text{offset in} = t_s + t_{In} - t_{clk}$$


از این max
ts
td

فعلاً با clk ورودی پایه FPGA کاری نداریم و فقط به clk ورودی ffها کار داریم؟

تعریف offset in درست نسبت به clk ورودی پایه است.

از لبه بالارونده‌ای که ورودی دستش ff است چه قدر طول می‌کشد تا داده valid روی خروجی قرار گیرد.

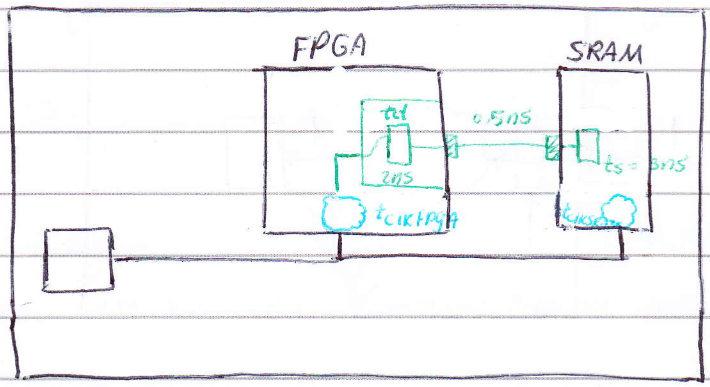
td + tout

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

ffها hold time هم دارند ولی چون بسیار کوچک است اکثراً صرف نظر می‌شود؟

hold time: th: مدت زمانی که باید داده بعد از لبه stable باشد.

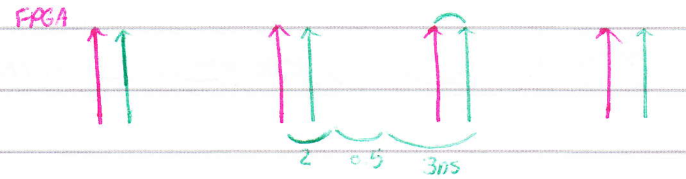
Example:


FPGA
SRAM
td
2ns
0.5ns
ts = 3ns
tclkFPGA
tclkSRAM

با فرض برابر بودن $t_{clkSRAM}$ و $t_{clkFPGA}$:

min Period = $2 + 0.5 + 3$

با فرض $t_{clkFPGA} = 0$ و $t_{clkSRAM} \neq 0$



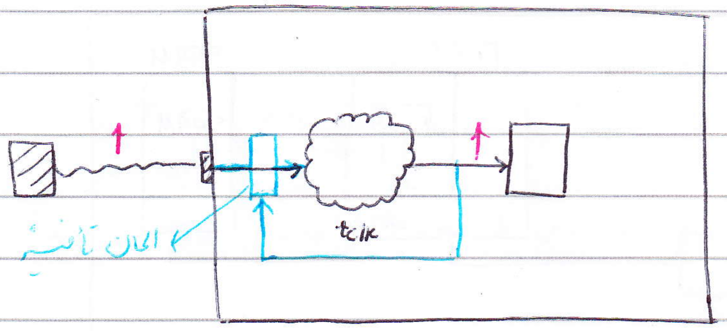
$2 + 0.5 + 3 - t_{clkSRAM}$

با فرض وجود داشتن و مخالف بودن $t_{clkSRAM}$ و $t_{clkFPGA}$

$2 + 0.5 + 3 + t_{clkFPGA} - t_{clkSRAM}$

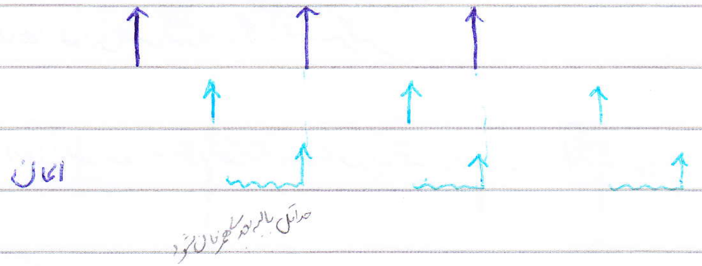
فرض کردیم که فقط در SRAM می‌نویسیم و از آن داده می‌خوانیم.

در عمل کاری می‌کنیم که t_{clk} ها همزمان باشند.



علت تأخیر: وجود داشتن بار خازنی بسیار زیاد و علت FF ها و ...

می‌خواهیم کاری کنیم که ↑ همزمان شوند



Allignment

Phase Compensation

همه FPGA ها المانی دارند که این کار را انجام می‌دهد و ما در عمل حتماً از این المان استفاده می‌کنیم.

می‌توانیم کاری کنیم که تأخیر بین لبه‌ها یک تناوب باشد.

ادامه شمارنده 256 بیتی جلسه قبل (پس از مشاهده کارتون)

نرم‌افزار دیگری در ISE وجود دارد که برای زمان‌بندی به کار می‌رود:

implement design → place & Route → Analyze Timing

در timing Analyzer نشان می‌دهد که در کدام مسیرها تأخیر بیشتر و مقدار طول است چرا؟

Tools → FPGA Editor

در timing Analyzer روی مسیر کلیک می‌کنیم و به طور اتوماتیک مسیر در FPGA Editor به ما نشان داده می‌شود.

net‌های carry chain ، FPGA همواره عمودی است .

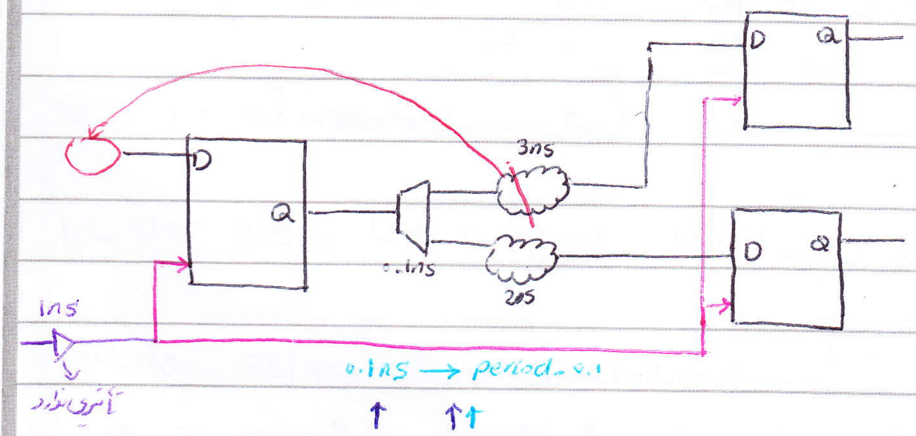
چرا همه المان‌ها عمودی چیده شده‌اند؟ تأخیر کمتر

در حلقه بعد المان گفته شده به مدار اضافه خواهد شد در نتیجه offset out کمتر می‌شود زیرا تأخیر clk صفر می‌شود

جلسه بیست و یکم :

تمرین :

در یک FPGA ، set up time مربوط به ff ها 2.1ns می‌باشد تأخیر ff ها 3ns می‌باشد
ماکزیمم فرکانس کاری مدار مقابل چه قدر می‌تواند باشد؟


D
Q
D
Q
D
Q
3ns
0.1ns
2ns
1ns
تأثیری ندارد
0.1ns → period - 0.1

critical path :

مسیری که بیشترین تأخیر را دارد .

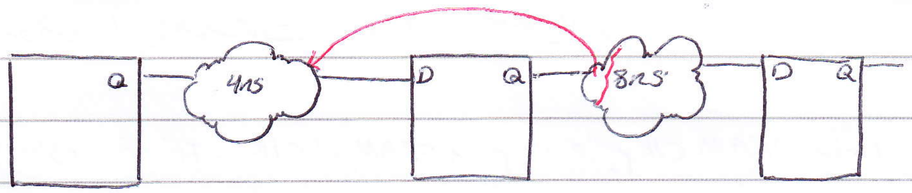
۳+ ۰/۱+ ۳+ ۲/۱ = ۸/۲ ns

clk skew :

تأخیری که بین لبه‌های clk ای رسیده به المان‌ها وجود دارد .

این طرح خوبی نیست زیرا pipeline stage باید بهتر است تأخیرهای برابری داشته باشند.

پس بهتر است مقداری از تأخیر موجود در 8ns را به قبل از ff انتقال دهیم.



Retiming :

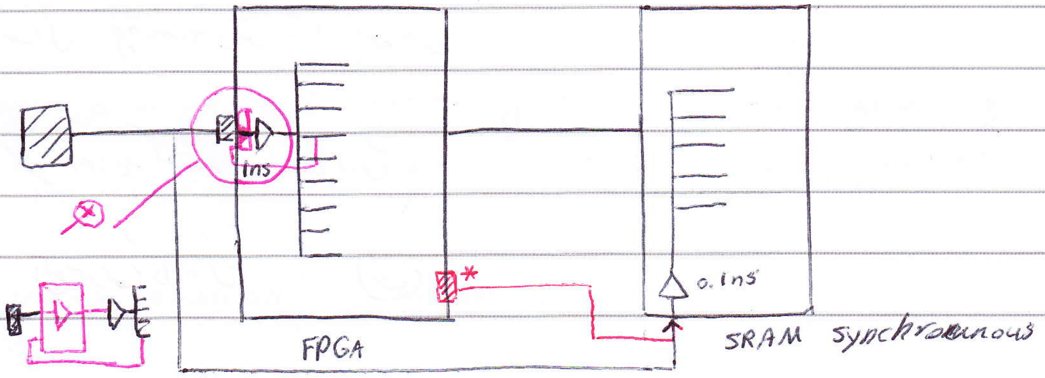
بالانس کردن pipeline از لحاظ تأخیر که توسط synthesizer رخ می‌دهد.

در هنگام استفاده از Retiming، نتایج Functional sim با post route sim متفاوت خواهد شد.

همواره باید clk را بافر کنیم و به تراشه‌های خود می‌دهیم.

علت بافر کردن :
باعث می‌شود که load طبقات بعد روی این اسیلاتور نیفتد در صورت بافر نکردن بار روی اسیلاتور می‌افتد

و ممکن است دیگر نوسان نکند یا لبه‌ها تیز نباشد.

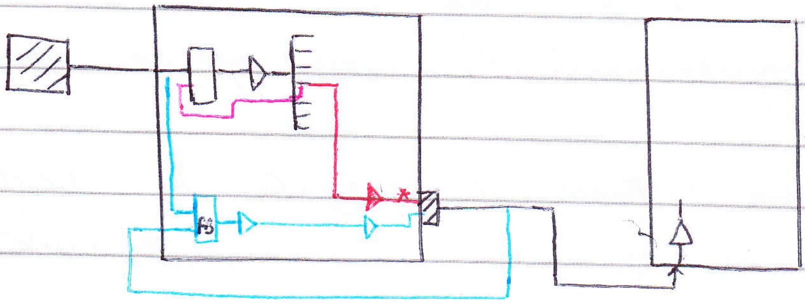


وقتی فرکانس خیلی پایین باشد، به allign شدن لبه‌ها نیاز نداریم.

Delay Locked loop:

المانی که در طرح اضافه می‌شود تا تأخیر مطلوب جهت allign شدن لبه‌ها ایجاد شود

* اگر بخواهیم یکی از پایه‌های FPGA، clk را به SRAM بدهیم، و بخواهیم SRAM clk دقیقاً شبیه FPGA clk باشد.



* این خوب نیست زیرا تأخیر درون پایه FPGA را در نظر نگرفتیم.

می‌توان clk روی بورد را با clk درون FPGA همزمان کرد.

همه المان‌های logic، DLL دارند.

مسائل timing می‌تواند بسیار پیچیده شود.

اگر reset کنیم از lock در می‌آید.

آخ جون کارتون

با نرم افزار ISE ادامه می دهیم :

کد شمارنده 256 بیتی را می نویسیم :

create new project

new source → verilog module → counter

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

حال design را implement می کنیم .

design روی → right click → verilog test fixture → name →

انتخاب ماژولی که می خواهیم تست کنیم .

implementation → counter

simulation → test_fixture → { behavioral
post route ✓

simulate post route & place

ISE خودش یک simulator دارد که قوی نیست .

clk ی که می‌بینیم ، clk ی است که اسیلاتور تولید می‌کند .

counter_test_fixture → UUT → clkBUFG → Add

این سیگنال clk روی پایه FPGA است .

می‌بینیم که بین لبه clk ها فرق 1.9ns تأخیر داریم .

می‌خواهیم clk DLL را اضافه کنیم .

user constraint → timing → { clk domain : 20ns
offset out : 20ns

حال دوباره implement می‌کنیم .

در کد شمارنده :

```
wire    clkToDLL;
 IBUFG   inputClockIns  (.I(clk) , .O (clkToDLL))
```

اینجایی که همیشه clk یکسان وارد می‌شود

```
DCM   DCMIns (.clkIN(clk To DLL), . CLKO (CLK toBUFG)
                 ,. CLKFB(myclk))

wire  clk ToBUFG;
wire  myclk;

BUFG  bufgIns  (.I(clk toBUFG ),.O(myclk));
```

قبل از always اضافه می‌کنیم .

دوباره طرح را implement می‌کنیم .

simulate post place & route

اگر تعدادی از پایه‌ها را نخواهیم صدا بزنیم آنها را رها می‌کنیم .

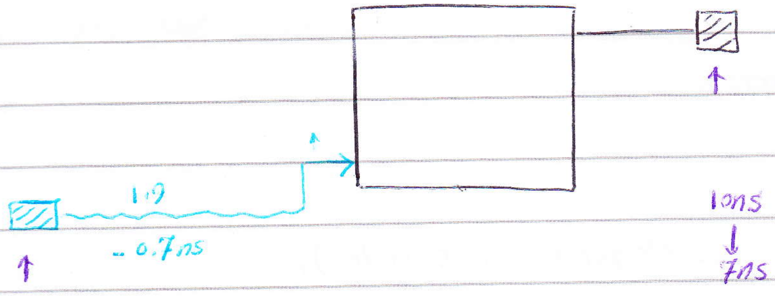
خروجی BUFG را اضافه می‌کنیم .

حالا اختلاف زمانی 740ps می‌شود .

implement → reRun

اگر constraint مربوط به clk to out حذف شده بود به خاطر اختلاف فاز ۹۰ می‌باشد .

در این حالت Clk to Out بیشتر شده است . (7.2)



10 - (1.9 + 0.7) =

In OFFset در این حالت بیشتر می شود.

project ⟶ right click ⟶ New Source ⟶ IP Core

⟶ Name : myclk manager $\xrightarrow{\text{select}}$ FPGA Feature & Design

⟶ clocking ⟶ spartan III

datasheet spartan III، بسیار مهم است (به عنوان یک refrence برای کار ما)

Digital clock Manager = DCM = DLL

در datasheet به قسمت DCM می ریم ۱

خروجی ها : Clk0 , Clk90 , Clk180 , Clk270 , Clk2x , Clk2x180 , ClkDV

Clk2x ← فرکانس دو برابر

ClkDV ← فقط تقسیم

ClkFx , Locked , Status [7:0]

ClkFx → فرکانس ورودی × M/N

$2 < M, N < 32$

Locked → وقتی DLL Locked باشد

تنها خروجی که بدون FB می‌توان داشت CLKFX می‌باشد.

می‌توانیم پالس ساعت ورودی را هر مقداری که بخواهیم شیفت بدهد (مثلاً 60° ....)

Clk2x : xor 0 و 90°

جلسه بیست و دوم:

گوئیز:

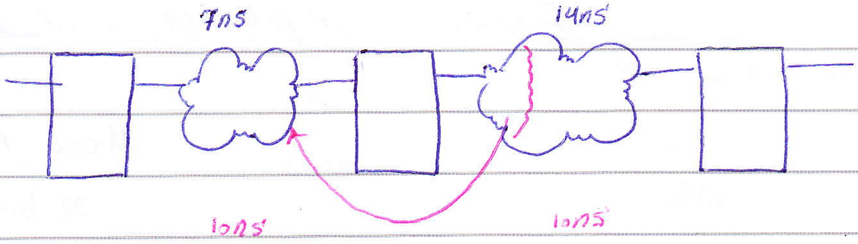
الف) طی عمل Retiming چه تغییری در مدار اتفاق می‌افتد و اثر آن چیست؟ (با شکل)

ب) تأخیر offset out به چه مفهومی است؟

ج) المان BUFG چه کاری در FPGA انجام می‌دهد؟

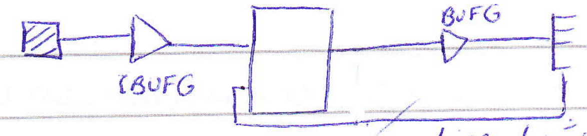
درس امروز بسیار مهم است و کاربردهای بسیار زیادی دارد.

الف) برای بالانس کردن pipeline استفاده می‌شود.



ب) تأخیر پالس ساعت سر پایه clk تا valid شدن داده روی pad خروجی

ح) net clk به المان BUFG متصل است.



BUFG : clk ای که می‌خواهد در FPGA پخش شود را drive می‌کند.

بحث امروز در مورد Embeded system هست.

یک کامپیوتر کوچک برای یک task خاص: مانند پروژکتور، دوربین

یکی از اصلی‌ترین عناصر یک Embeded system، cpu است. امروز می‌خواهیم ببینیم چگونه می‌توان با FPGA، cpu داشت.

در FPGA می‌توان دو نوع core داشت: نرم (المان‌های موجود در FPGA) - سخت (core واقعی cpu)

کمپانی xilinx سه نوع cpu دارد:

۱. Pico Blaze:

8bit soft core

ممکن است سرعت محدود، resource محدود و ... داشته باشد.

۲. Micro Blaze:

32 bit soft core

resource بیشتری در FPGA مصرف می‌کند

۴. Power Pc

64 bits -hard core

توان پردازشی بالا (max freq=500MHz

دو مورد اول در همه FPGAهای خانواده Xilinx وجود دارد ولی مورد سوم در خانواده های زیر وجود دارد:

Virtex - II Pro * Virtex - 4Fx * Virtex - 5,6 Fx

برای استفاده از دو مورد آخر نیاز به یک محیط نرم افزاری (Develpement Env.) داریم که در آن محیط برنامه ریزی می کنیم

EDk = Embeded Developement kit

امروز می خواهیم در مورد Pico Blaze صحبت کنیم

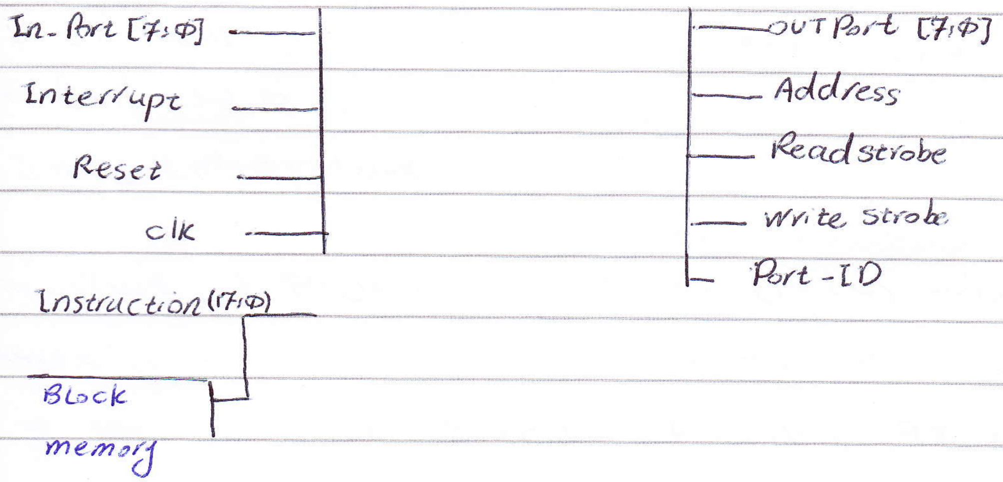
کمپانی Xilinx اولین و کمپانی Ultra دومین فروش FPGA را در جهان دارند.

کمپانی Ultra، یک soft core، 32 bit دارد که نام آن NIOS II است.

سه مورد اخیر در محیط Linux نیز بالا می آید.

مشخصه‌های Pico Blaze:

نویسنده: Ken chapman


In-Port [7:0]
Interrupt
Reset
clk
Instruction (17:0)
Block
memory
OUT Port [7:0]
Address
Readstrobe
Write strobe
Port-ID

S0 - S15

مشخصات : 16 رجیستر 8 بیتی دارد.

ALU _ Program Flow Contrd - حافظه داده 64 بیتی

چطور Pico-Blaze را در کد وریلاگ صدا بزنیم.

KCPSM3 = Pico-Blaze

مجموعه دستورات Pico Blaze:

Compare - SUBC - SUB - ADDC _ ADD - return _ call _ jump

- Store _ rotate _ shift - Test - XOR - OR _ AND . Load

Output . Input - Fetch

دستورات test خوبی برای چک کردن صفر بودن یک reg است.

ریختن یک مقدار ثابت در یک رجیستر

مقدار یک reg در reg دیگر

Load Sx, KK
Load Sx, Sy

AND نیز همین گونه است .

AND Sx, KK

AND Sx, Sy

برای دستور AND ، Flag ، carry همواره صفر می شود و درصورتی که نتیجه zero باشد، zero یک می شود

Test :

یک reg را با یک ثابت AND می کند و آن را در یک temporary reg می ریزد

if temp=0 → zero=1
if تعداد یک ها فرد → carry=1

ورودی خروجی :

OUTPUT Sx, PP
INPUT Sx, PP

اجرای هر دستور ۲ سیکل طول می کشد .

PORT-ID :
شماره پورتی که با آن در ارتباطیم

Read_Strobe :
وقتی فعال است که Pico Blaze می خواهد یک داده بخواند

Write_Strobe :

وقتی فعال است که Pico Blaze یک داده را گذاشته بیرون

کارتون

xilinx website → KCPSM-3 → download

asm → Extract to the directory of project

Source Code → Post Synthesis

Verilog source → copy to the directory of project

فرض کنید FPicoBlaze خواهد به صورت شمارنده کار کند

توضیح ؛ This is a simple PicoBlaze test Program

```
Constant    OUTPUT-PORT1,01
مقدار ثابت
Constant    Input - PORT2,02      { Constant   INPUT_PORT3,03
Start:      شروع برنامه

  LOAD      So,00;

Loop1:                            { INPUT
                                    INPUT
 OUTPUT     So, OUTPUT_PORT1
 ADD        So,01
 COMPARE    So,05
 JUMP       Z, Loop1_Count
JUMP loop1
Loop1_count
LOAD S0,00
JUMP loop1
```

برنامه‌مان را در جایی که asm را ذخیره کرده‌ایم با پسوند psm. ذخیره می‌کنیم.

حالا آن را کامپایل می‌کنیم (در run)

هم coe تولید می‌کند هم کد وریلاگ.

در کد وریلاگ یک ماژول با نام فایل‌مان می‌سازد و مقادیر اولیه Block . memory را مشخص می‌کند.

این فایل وریلاگ را به ISE اضافه می‌کنیم.

New Project → Name → device → .. → Finish

Add source → KCPSM3 & کد وریلاگ

باید یک کد وریلاگ برای صدا زدن Pico-Blaze بنویسیم.

New Source → verilog module

```
                                   ,In1, In2
module  top ( clk, Reset, out  );
input   clk, Reset;          <input  [7:0]  In1,In2;
output   [7:0]  out;
reg      [7:0]   out;
wire     [17:0]  instruction;
wire     [9:0]   address;                  . حافظه را صدا میزنیم Pico Blaze
wire     [7:0]   Port_id   , out_port, in_port
kcpsm3    PicoIns    ( .address (address) ,
                       .instruction (instruction),
                       .port_id  (port_id),
                       .write_strobe (write_strobe),
                       .out_port (outport),
                       .read_strobe() ,
                       .in_port ( ) ,
                       .interrupt( ) ,
                       .          _ack( ) ,
                       .reset (Reset),
                       .clk (clk));

my code    mycodeIns  (.address(address),
.instruction(instruction) , . . . . . .          )
assign  in_port= (port_id ==2) ? In1 : In2;
always  @  (posedge clk or posedge reset)
   if  (reset)
      out <= 0;
```

```
if (write-strobe && port-id = = 01)
        out      ⇐ out-port
```

بعد از نوشتن tester:

implement

○ implemetation   ● simulation ▾
behavioral

حال نتیجه شبیه سازی را خواهیم دید.

```
Isim> run 2000ns
Isim> restart
```

address: آدرس دستور

port-id، آدرس port

UUT: Unit Under Test

هر داده جدیدی که تولید می شود write-strobe به مدت یک سیکل "تک" می شود. یعنی در این لحظه اگر داده روی پورت خروجی را برداریم حتماً داده valid برداشته ایم.

می خواهیم ورودی تعریف کنیم.

سیستم رزکنده video، interface

می خواهیم وقتی port-id=2 داده را از LM بخوانیم در غیر این صورت اگر port-id=3