داده‌ها از ENG گرفتیم.

ftp:

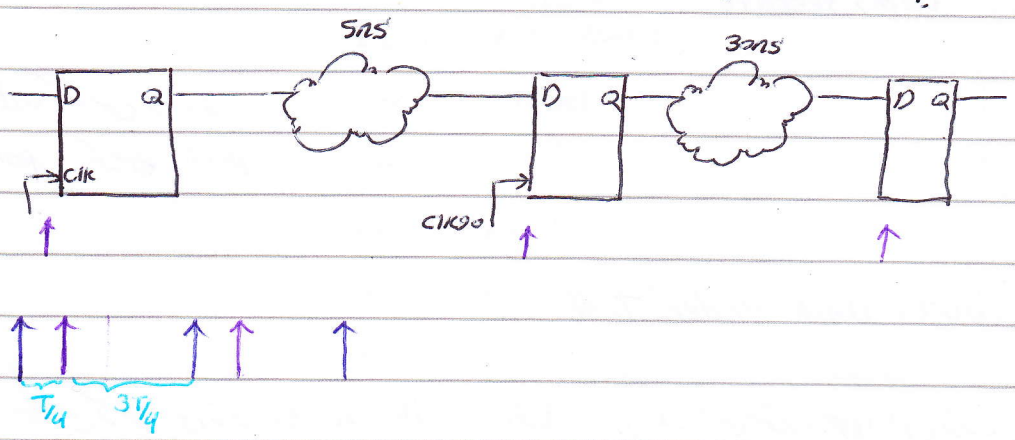
172.16.3.36

172.16.3.103

جلسه بعد رسیدیم.

کوییز:

ماکزیمم فرکانس کاری مدار؟



$\frac{1}{4}T \geq 12ns \longrightarrow T \geq 48ns$

$\Rightarrow T \geq 46.7ns$

$\frac{3T}{4} \geq 35ns \longrightarrow T \geq 46.7ns$

آزمایشگاه FPGA یک امتحان کتبی ساده دارد (از نرم‌افزارها الات نرم‌افزاری)

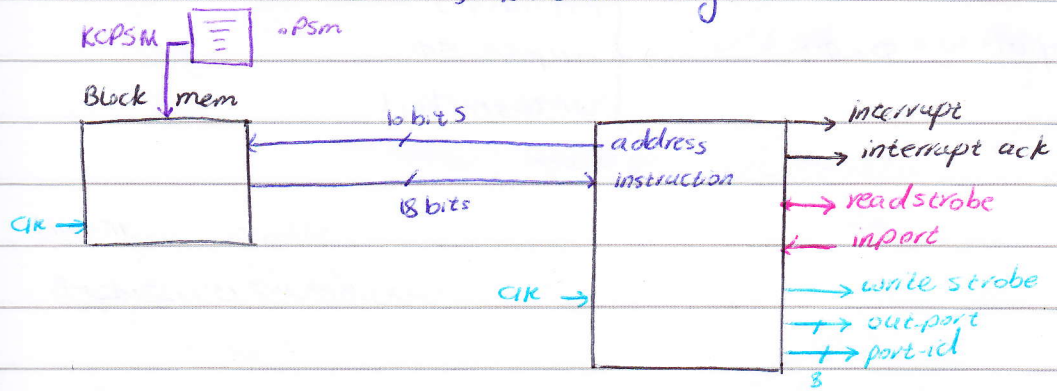
امتحان آزمایشگاه: شنبه ساعت ۱۰:۳۰ و یکشنبه ساعت ۱۰:۳۰

امتحان عملی محدودیت وقت ندارد. (از دو آزمایش آخرد post route simulation می‌آید)

عملی یا دو هفته دیگر یا بعد از امتحانات پایان ترم

در جلسه قبل با core جدیدی به نام Pico Blaze آشنا شدیم.

Pico Blaze: CPU 8bits, 16x8 bit reg



برنامه‌ای که می‌نویسیم PB اجرا می‌شود در Block mem

برای ساختن Block mem از دستورات اسمبلی پیروی می‌کند .psm استفاده می‌شود.

KCPSM زبان اسمبلی را به VHDL یا Verilog تبدیل می‌کند.

Port id:

۸ بیت است و شبیه آدرس در mem است.

وزارت بازرگانی شرکت بازرگانی دولتی ایران

Power Pc ۱۴ تا Address Bus دارد.

PB با عددی که روی Port . ID می‌گذارد می‌گوید که می‌خواهد با کدام بخش ارتباط داشته باشد.

write - strobe:

وقتی می‌خواهد داده‌ای را قرار دهد آن را یک می‌کند.

با اجرای دستور output محتوای رجیستری که مشخص کرده‌ام روی out port خروجی قرار می‌دهد.

Output S0, 05 → { Port-id = 05, outport = S0, writestrobe = 1 }

in-port:

8 بیت است برای خواندن به کار می‌رود.

read-strobe

به مدت ۲ سیکل ساعت یک می‌شود.

INPUT S0, 01 → port-id : 01 → readstrobe : 1

مقدار inport داخل reg S0 ریخته می‌شود.

عملاً از read strobe استفاده نمی‌کنیم.

interrupt:

وقتی int رخ می‌دهد Program Coun. به آخرین آدرس می‌رود و باید در این جا یک jump قرار داد.

باشیم در int. service routine برود.

```
ISR:

RETURNI

0xff [JUMP ISR]
```

زمانیکه PicoBlaze ، int را تشخیص دهد.

دو دستور اسمبلی که به فعال و غیرفعال کردن int کمک می‌کند:

interrupt enable

interrupt disable

می‌خواهیم در مورد Port id و کاربرد آن صحبت کنیم.

با Port-id مشخص می‌کنیم به کدام قسمت از کد verilog می‌خواهیم دسترسی داشته باشیم و در آن بنویسیم یا از آن بخوانیم

معمولاً در آخر برنامه یک spin loop می‌گذاریم تا هیچ وقت به 0xff نرسد.

با Port id به بازه‌هایی که خودمان نوشته‌ایم و در FPGA قرار دارد دسترسی داشته باشیم.

incredibles: کار تک

ابتدا design حل تمرین می‌کنیم.

تمرین: یکبار In1 را بخوانیم و یکبار In2. برای این کار از port id استفاده می‌کنیم.

```
CONSTANT      OUTPUT_PORT1, 01        <OUTPUT_PORT5, 05
CONSTANT      INPUT_PORT2, 02
CONSTANT      INPUT_PORT3, 03
              INPUT_PORT4, 04
```

عددهایی که در کد verilog است با کد اسمبلی باید هماهنگ باشند

```
start :
        INPUT     S2, INPUTPORT3 *
        INPUT     S0, INPUT_PORT2
        INPUT     S1, INPUT_PORT3
        ADD       S1,S0
        OUTPUT    S1, OUTPUT_PORT1
        JUMP      start
```

In3 را به صورت command اضافه می‌کنیم

```
assign in_port = (port_id=2 ) ? In1:
                 (port_id =3 ) ? In2:
                 (port_id =4) ? In3:
                 0 ;
```

* In3 را از بیرون می‌خواند و می‌ریزد در S2.

```
LOAD    S5,SC
LOAD    S6,S1
ADD     S6,S5
SUB     S1,S4
COMPARE S2,00
JUMP    Z,somewhere
OUTPUT  S1,OUTPUT-PORT1
JUMP    end-loop
somewhere:
OUTPUT  S6, out-port1
end-loop:
JUMP    start
```

OUTPUT S6, output-Port5
1

```
always @ (posedge clk)
if (Reset)
out <=0

else begin
  if (write-strobe) begin
  if (port-id = = 01)
      Out <= out-port;
  else if (port-id = = 05)
      out2<= outport;
```

assemble کنیم ← دوباره ... اسمبلی را تغییر دادیم

KCPSM3

ISE → simulate Behavioral mode

هر دستور PB تقریباً دو سیکل طول می‌کشد تا اجرا شود

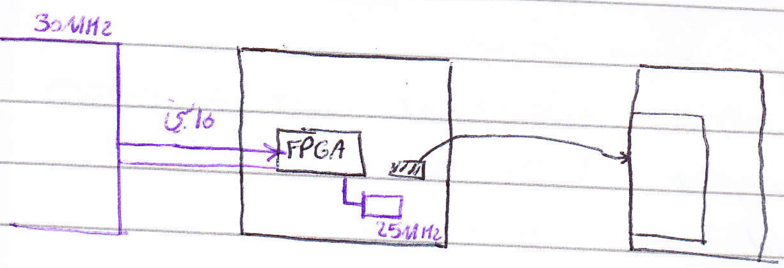
می‌خواهیم یک مسئله دیگر حل کنیم

از یک فرستنده داده‌های ۱۶ بیتی با فرکانس 30MHz به FPGA ارسال می‌شود همراه داده‌ها یک سیگنال valid هم ارسال می‌شود که یک بودن آن به معنای valid بودن داده است.

سیستمی طراحی کنید که داده‌ها را از روی پورت سریال به کامپیوتر منتقل کند.

Crystal Oscillator روی برد 25MHz است.

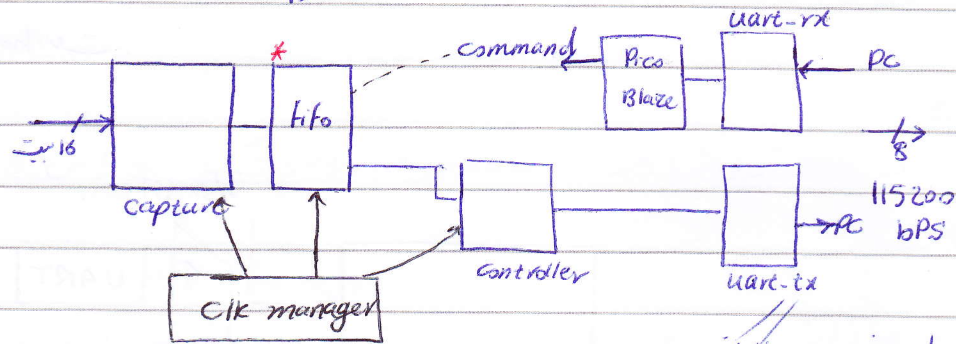
می‌خواهیم کاربر طرف کامپیوتر قادر باشد از طریق RS-232 بورد را کنترل کند.
با دستور R سیستم Reset شود.
با دستور S ارسال داده متوقف و شروع شود.



طراحی top-down (بالا به پایین) است.

سخت‌افزارهای مورد نیاز :

۱- سخت‌افزاری که از روی فرکانس 25 فرکانس های دیگری بسازیم و بتوانیم داده ها را capture کنیم .



یک بار باید عرض داده ها را کم کنیم

داده ها همیشه نمی آید ولی وقتی می آید Burst می آید .

* نرخ خروج و ورود داده متفاوت است .

جلسه بیست و چهارم :

هر field را که در نظر گرفتید تا آخر در همان بمانید . معمولاً switch کردن بسیار دشوار است . ابزاری apply کردن !

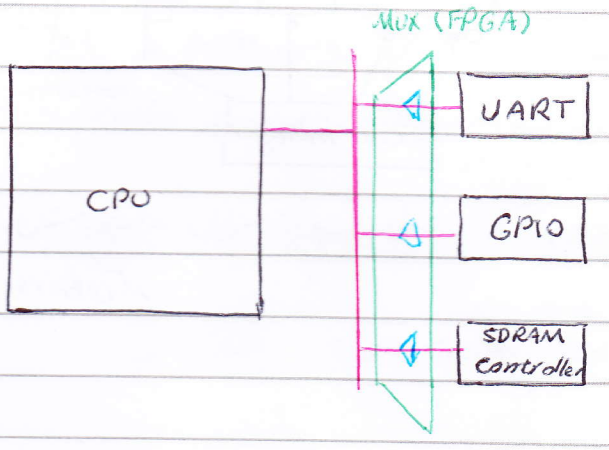
امروز می خواهیم در مورد Embeded design صحبت کنیم

- Micro Blaze
- Power PC

Power Pc :

این را در اصل IBM طراحی و پیاده سازی کرده و در داخل FPGA گذاشته می شود. بخش عظیمی از شرکت ها وابسته به IBM هستند.

hard core است.



اگر Pico Blaze داشتیم استاده از Port-id یا وسایل جانبی ارتباط برقرار می کرد. ولی در طرحهای پیشرفته از Address Bus برای این کار استفاده می کنیم.

transaction :

هر عملی که به طور کامل انجام می شود.

Bus Master :

initiate می تواند transaction

ممکن است چند Master داشته باشیم و یک Arbiter برای هماهنگ کند.

در طرح کشیده شده، CPU، master است.

System-on-a-chip (System-on-a-programmable-chip)

SOC Bus:

باسی که در SOC تعبیه می‌شود چون integration دارد سرعت آن نسبت به باس PCB بیشتر است.

در PCB ها باید tri-state-buffer داشته باشیم اما در FPGA ما تو ادوار این منابع routing داریم. به هر تعداد که بخواهیم می‌توانیم connection برقرار کنیم.

در mux در هر لحظه، آن دو طرفی که تبادل داده دارند با هم ارتباط دارند.

در FPGA عرض Bus می‌تواند خیلی بزرگ باشد و در نتیجه rate انتقال داده بیشتر است. چون سیم کشی راحت است عرض افزایش می‌یابد.

چند شرکت بیشتر SOC ندارند.

چند نمونه از SOC ها:

1) Core Connect → IBM

IBM برای CPU های خودش طراحی کرده است.

2) ABMBA → ARM

Advanced Micro.... Bus Architecture

ARM:

core به صورت silicon وجود ندارد و فقط design آن موجود می باشد. در واقع کمپانی ARM، License آن را می فروشد پس licensable است. در شرکت های Atmel، Apple و برای میکروکنترلر خود استفاده می کنند.

3. Wish Bone (open source)

متعلق به کمپانی خاصی نیست و همه می توانند از آن استفاده کنند.

کمپانی هایی که وسایل جانبی را می سازند حتماً یک نسخه wish bone آن را نیز می سازند که مبتنی بر استاندارد wish bone هستند

علت استفاده از SOC:

ماژول‌های منطقی که می خواهند به صورت یک سیستم پیاده سازی شوند و اگر بخواهیم آنها را به صورت efficient پیاده سازی کنیم آنها را به صورت SOC پیاده می کنیم

www.opencores.org

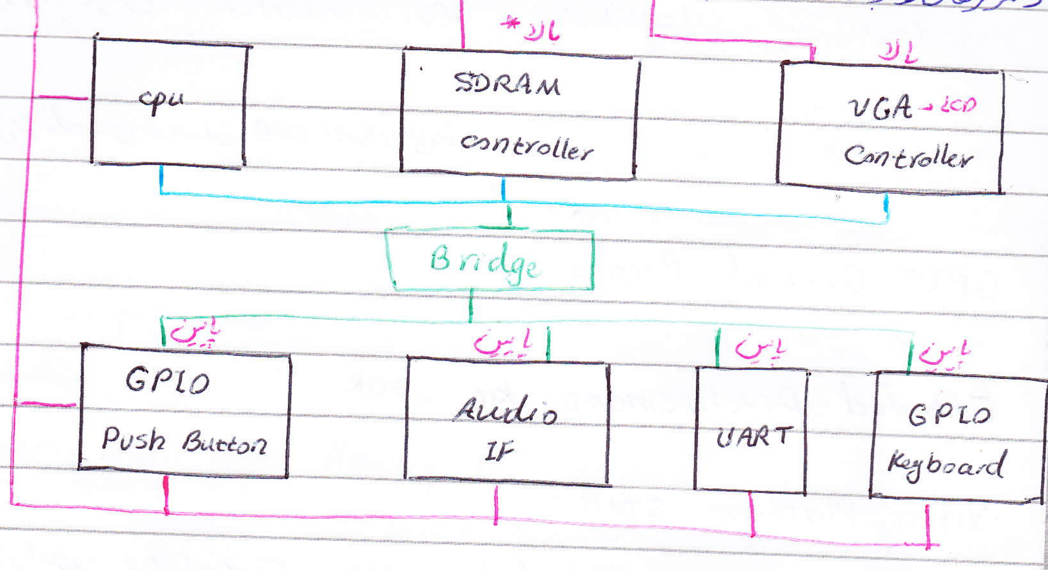
Xilinx از باس core connect استفاده می کند.

حجم انتقال داده ماژول‌های مختلف متصل به یک Bus متفاوت است.

فرض برای وصل کردن speaker و microphone تعدادی DAC نداریم.

به ازای هر ورودی خروجی متصل به board یک ماژول داریم پس یک ارتباط یک به یک بین Peripheral ها

و کنترلرهای موجود در FPGA وجود دارد.



Peripheral ≡ FPGA Controller

* ریت انتقال داده

تمام سازنده‌های فوق می‌خواهند با CPU حرف بزنند ولی حجم انتقال داده در آنها متفاوت است.

پس لازم نیست همه آنها را روی یک Bus متصل کنیم و دو Bus می‌گذاریم. یکی برای سازنده‌های با ریت بالا، دیگری برای سازنده‌های با ریت پایین و این ایده Core Connect است.

PLB: Processor Local Bus

OPB: On chip Pripheral Bus

DCR : Device Control register

منظور داده‌های کنترلی را ارسال می‌کند سرعت آن خیلی پایین است.

CPU بستقیم به OPB متصل نیست. Bridge که رابط بین CPU و سازنده‌های با سرعت پایین است.

CPU می توانند همزمان interfce PLB & OPB را داشته باشند. (مثلاً با mux)

در design های جدید از OPB استفاده نمی شود.

GPIO: General Purpose I/O

Embeded Developement kit ; EDK

Xilinx Platform studio:

یک محیط جدید که با سرعت بسیار زیادی می توانند Embeded system را بر روی FPGA بریزند

Base System Builder wizard:

مثلاً کل طرحی که در صفحه قبل کشیده شده است یک Base System است.

I would like to create . . . → next

, , . . a system for the

→ xilinx

Board name → virtex 4 ML403 .. → next

تعیین تعداد processor ← next

فرکانس کاری مدار ← next

حالا پنجره ای داریم که لیست Pripheral هایی که نشان داده شده است.

DDR SRAM - Ethernet MAC - RS232 Uart - xps - Brm - if-cntires

سایر موارد را remove می کنیم. ← next

تا اینجا درمورد سخت‌افزار Base System پرسیده بود.

Base System را می‌توانید برای application خالی ایجاد کنید ← next

حالا یک report به ما می‌دهد ← Finish

نشان دادن چگونگی ارتباط اجزا ← Block Diagram

MPMC: Multi-Port Memory Controller

* روی باس PLB قرار گرفته است! (برخلاف مطلب گفته شده).

رابط SDRDRAM و CPU است

Slaves of plB:

Ethernet - Interrupt Controller - Uart

CPU از یک طرف به باس PLB متصل بود و از یک طرف به حافظه.

* علت: نرخ انتقال داده بین حافظه و CPU زیاد است. اگرچه آنها از روی PLB بیایند سهم پهنای باند زیادی به آن اختصاص داده می‌شود.

از باس OPB هیچ خبری نیست. ماژول‌های ما یا به PLB متصلند یا به MPMC.

Ports: ماژول‌ها، لیست پورت‌ها

Addresses: ماژول‌ها، آدرس آنها

Hardware → Generate bit stream

Hardware مشکلی باشد.

چ software ی لازم است ش PicoBlaze

Software →

تنظیمات نرم افزار

جلسه بیست پنجم:

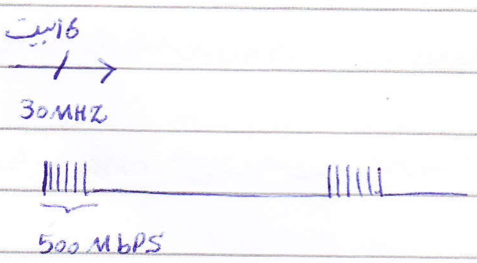
کاربردی: استاندارد PCI EXPRESS

Spartan6 & Viotex6 این رابط را درخود دارند.

172.16.3.177

172.16.3.80

ادامه جلسه بیست دوم:



داده ها به صورت Burst می آیند مانند شکل فوق.

در واقع داده ای ارسال می دهیم

→ UART

115200 bPS

valid

16 bits

30MHz

25MHz

هیچ کلاکی با داده می آید

دقیقترین راهش برای انتقال داده این است که با داده یک سیگنال سنکرون (CLK) برای مبنای نمونه برداری هر

لبه آن داده را برداریم . حتی پارچیپ ها این کار را می کنند که با هر داده یک CLK می فرستند مانند :

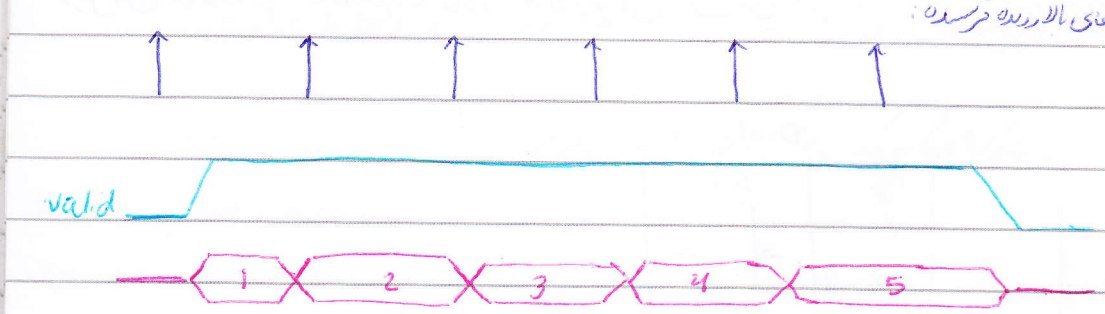
PDR - SDRAM

دارای حجم سریع برداریش بالا است .

Source Synchron Data ← design هایی که در آن ها CLK همراه داده می آید

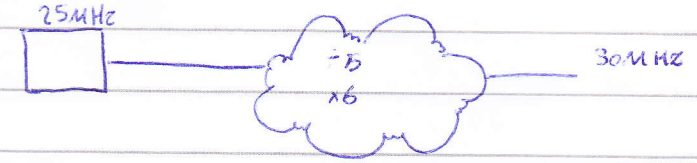
گفته می شود .

لبه های بالا رونده فرستنده :



اول باید با CLKDLL فرکانس 25 را به 30 تبدیل کنیم

clock manager



یک راه حل می‌تواند به صورت زیر باشد:

با فرکانس 3 برابر نمونه برداری می‌کنیم می‌توان اطمینان داشت که حتماً در ناحیه data بدون Valid

سه لبه داریم.

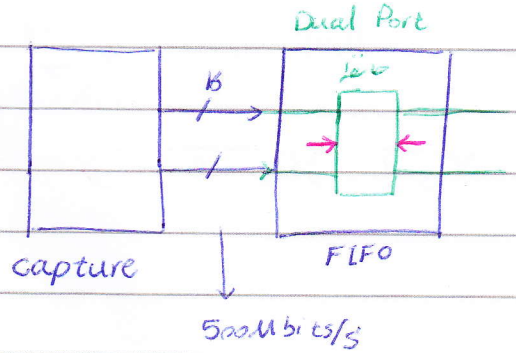
اگر هر لبه ای که valid یک بود در لبه‌ی داده را برداریم حتماً داده درست است.

می‌توان با فرکانس 60MHz به صورت زیر عمل کرد:

با لبه بالارونده‌ای که دیدی valid یک است با لبه پایین رونده‌ی داده را بردار.

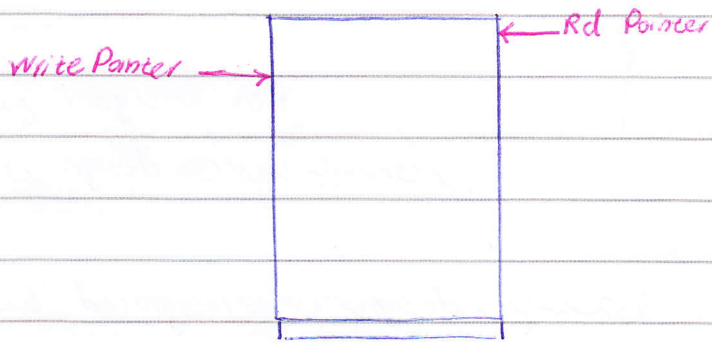
فرکانس SATA HARD ، 3GHz است.

فرض کن Data را Capture کردی. حالا باید آن را یک جایی ذخیره کنیم که آرام آرام آن را بخوانیم.



Block Mem { Single Port / Dual Port

تنها محدودیت موجود در حافظ Dual Port ، write همزمان به یک نقطه است. در حافظ Dual Port هر Port می‌تواند به صورت یک interface SRAM کامل عمل کند



یک counter می‌گذاریم و هر وقت خواستیم بنویسیم آن را یکی زیادی کنیم و هر وقت خواندیم آن را یکی کم می‌کنیم

در صورتی که مقدار counter با خانه‌های حافظ برابر باشد نمی‌توان چیزی نوشت.

کسی که دارد می‌نویسد به FULL نگاه می‌کند و کسی که دارد می‌خواند به Empty نگاه می‌کند.

هیچ الزامی ندارد که با فرکانسی که می‌نویسیم با همان فرکانس بخوانیم.

Asynchron Fifo

synchron FiFo

با هر لبه بالارونده ای می خوانیم (می نویسیم) فقط در بعضی لبه ها.

شدنی است سیگنال یک بار فقط یک داده می خوانیم ولی ممکن است در هر سیگنال یک داده بنویسیم.

کارتون

سبک طراحی: Top - Down

نرم افزار جدید: HDL Designer

می توان با آن design های پیچیده را پیاده سازی کرد.

AHDL یک محیط integrated بود یعنی هم design هم شبیه سازی.

HDL Designer محیط برای Design

Graphical view → block digram → Verilog 95 → name

Core های آماده را به پروژه HDL Design اضافه می کنیم.

File → Add → Existing File → download → KCPSM3

→ verilog

یک مستطیل سبز رنگ بالای صفحه وجود دارد ← click

از آن پجره ، uart_rax ، Uart_tx ، KCPSM3 را بطرح می‌کردیم

حالا میخواهیم حافظه‌ی رنگ را برای FIFO ... بطرح می‌کردیم

```
module My FIFO (

Data In,
WEn,
FULL,
Data Out,
RdEn,
Empty,
Data Count
clk,
Reset
);

parameter FIFOWIDTH = 16;
parameter FIFODEPTH = 256;
parameter ADDRWIDTH = 8;
input [FIFOWIDTH-1:0] DataIn;
```

سیگنال‌های سمت write

سیگنال‌های سمت Read

input WEn;

output FULL;

Output [FIFOWIDTH-1:0] DataOut;

input REn;

output Empty;

Output [FIFOWIDTH-1:0] DataCount;

حالا از Core کمک استفاده می‌کنیم.

Coregen ⟶ File ⟶ new Project ⟶ XC3S ⟶ Verilog

Memory & storage ⟶ Block memory ⟶ single Dual port RAM

veo ⟶ open with Notepad++ ⟶ محتوا مربوط به فایل‌هایی را به پروژه می‌کپی می‌کنیم

* myfifo_block_memory block_MemIns

*reg [ADDRWIDTH_1:0] write pointer, Read pointer;

* .clka(clk),
  .wea(WEn && (!FULL)),
  .addra(write pointer),

وزارت بازرگانی شرکت بازرگانی دولتی ایران

```
.dina( DataIn ),
.clkb(clk),
.addrb( readpointer )
.doutb (DataOut));
```

هنگام تولید core باید عمق را مشخص کنیم

```
* reg [ADDRWIDTH-1:0] DataCount;

always @ (posedge clk or posedge Reset)

  if (Reset) begin
    writePointer <= 0;   readPointer <= 0;
                          end
  else   begin
    if ( WEn && (!FULL))
      writePointer <= writePointer+1;

    if (RdEn && (!Empty))
      readPointer <= readPointer+1;
  end

always @ (posedge clk or posedge Reset)

  if (Reset)
    DataCount <= 0;
```

وزارت بازرگانی شرکت بازرگانی دولتی ایران

```
else begin
    if (WEn && (!FULL))
        DataCount <= DataCount+1;
    else if (RdEn && (!Empty))
        DataCount <= DataCount-1;
    else
        DataCount <= DataCount;
end
assign FULL= (DataCount==256)? 1:0;
assign Empty= (DataCount==0)? 1:0;
endmodule
```

if (WEn && (!FULL) && RdEn && (!Empty))
DataCount ← DataCount;

جلسه بیست و سوم:

کوئیز:

با این Core Connect چیست؟

از چه اجزایی تشکیل شده است؟

مزیتی که FIFO آن کلاک ورودی هم است نه مزیت خواندن و نوشتن هم است.

یک SOC متشی بر Power PC (یا میکروبلیز) است با سه گذرگاه:

PLB

OPB

DCR

تمام سیستم‌های Embeded برهمین پایه‌اند.

ادامه طراحی FiFo جلسه قبل :

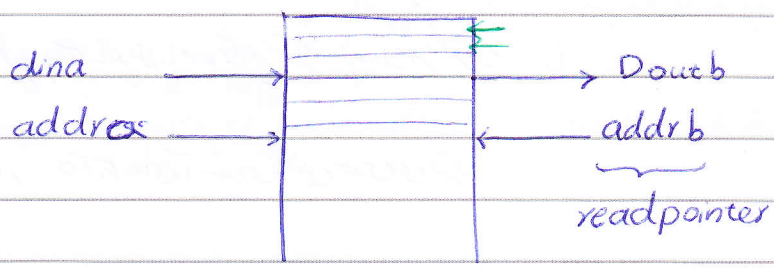
Full signal ← نویسنده

Empty " ← خواننده

اگر FiFo آسنکرون بود، دو تا سیگنال Data Count لازم داشتیم.

ما همواره داریم از memory می‌خوانیم، منتظر زمانی که بخواهیم read address را تولید کنیم اتفاق در (RdEn)

می‌شود.



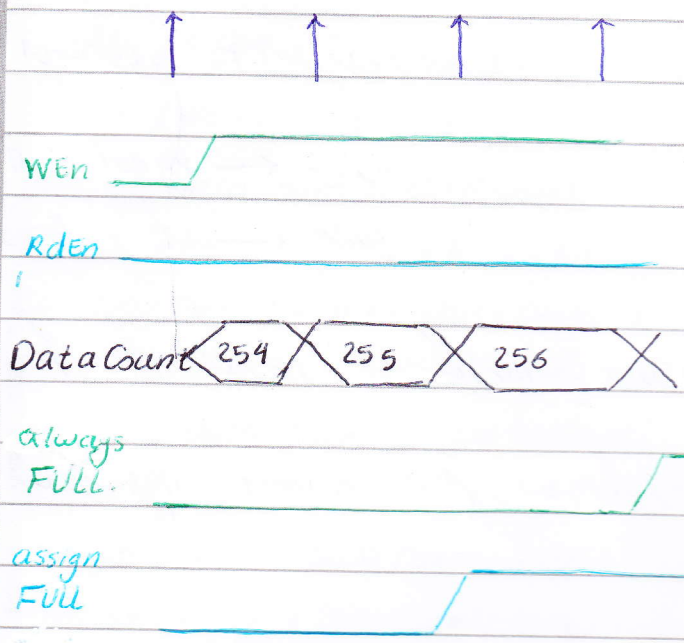
بعد از reset، read pointer صفر است و با هر بار که RdEn می‌آید readpointer

یکی افزایش می‌یابد

Data Count در تولید سیگنال‌های FULL و Empty به ما کمک می‌کند.

ممکن است در یک لحظه RdEn و WEn فعال باشد ← تصحیح برنامه قبل

چرا می‌توان به FULL و Empty در always مقداردهی کرد؟



Empty & FULL باید بلا فاصله توکید شوند. (بدون کوچکترین تأخیر)

این شکلی در FiFo های آسنکرون خیلی جدی می شود.

که FiFo خلاص شد به سراغ ادامه design می ریم.

لازم نرم افزار Designer پروژه خودمان سازیم HDL

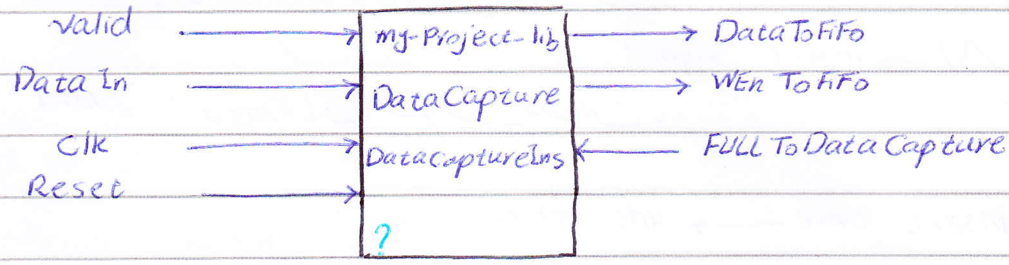
File → new → new project → address → next → Finish

Categories → Graphical view → block diagram

new → → → → →

option → verilog 95

یک نام دیگر instant name برای بلوک دیاگرام رسم می کنیم.



FiFo را insert می کنیم:

Add → Existing

اگر به هم وصل نباشند دو المان مستقل می باشند اگر که به هم متصلند.

روی بلوک فوق Double_Click می کنیم و مشخص می کنیم که می خواهیم کد verilog بنویسیم

Timing Designer:
برای رسم شکل موج

۱) کلاکی که فرستنده با آن می فرستاد. (f=30MHz)

۲) داخل FPGA تولید می کنیم. (f=90MHz) (یک offset به آن اضافه کردیم)

لبه های بالارونده هر دو کلاک را مشخص می کنیم.

سیگنال طرح را اضافه می کنیم: Data In و Valid

پالس

valid ⟶ initiate value

N ⟶ insert signal

insert time ⟶ uns before

wire captureNow;

assign captureNow = (valid) && (validR) && (!capturedR)
&& (!capturedRR);

reg [15:0] Data
reg validR, capturedR, capturedRR;

always @(posedge clk or posedge Reset)

if (Reset) begin
validR <= 0;

end
else begin
valid <= valid;

```
    captureR <= captureNow;
    capturedRR <= capturedR;
end

always @ (po
    if (Reset) begin
        dataR <= 0;
        WENR <= 0;
    end
    else begin
    if (captureNow)
        :
        :
        ;
```

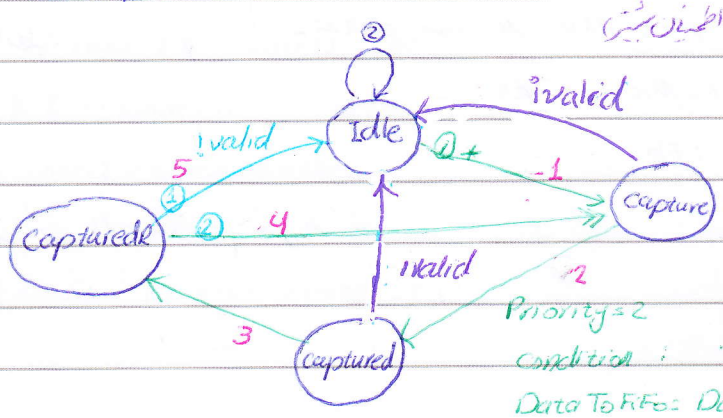
روش دوم استفاده از state machine می‌باشد :

block ⟶ right click ⟶ open as ⟶ new view ⟶ graphical

می‌توان با HDL Designer برای یک ماژول شماتیک view داشت

⟶ state machine

* اضافه کردن برای اطمینان بیشتر



* Valid

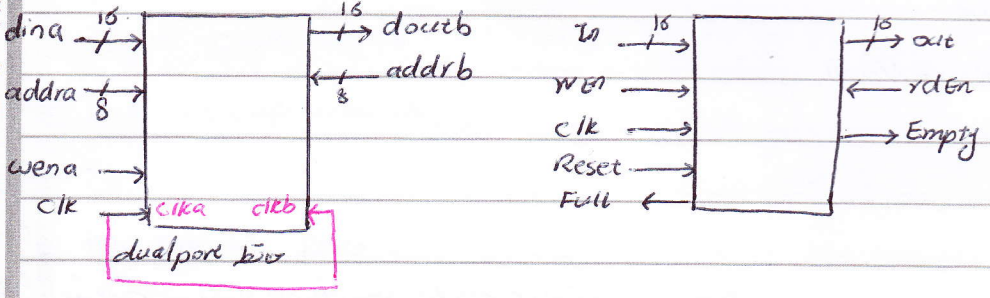
top ⟶

کد در بلاک block diagram, fsm طراحی کند

جلسه بیست و هفتم:

کوئیز:

با استفاده از یک حافظه dual port با عرض 16 بیت و عمق 256 خانه طراحی یک ماژول stack کنید.



اگر به حافظه با حجم زیاد نیاز داشتیم باید از حافظه SRAM خارج FPGA استفاده می‌کنیم زیرا این صورت دیگری توان خواندن و نوشتن را همزمان داشت. این حافظه dual port که قابلیت نوشتن و خواندن را به صورت همزمان داشته باشد داریم ولی بسیار گران قیمت است.

تولید کنندگان حافظه:

cypress: dual port RAM. Micron: SRAM.

SAMSUNG, hynix, Micron: DRAM

Toshuba, Intel : Flash

فرض کنید کل سیستم با یک clk واحد به هم متصل هستند.

برای طراحی

```
reg [7:0]   stackpointer;
reg [8:0]   data count;

always @ (posedge clk or posedge Reset)
if (Reset)      datacount <= 0;
else begin  if  (WEn && (!FULL))   begin
   if (rdEn && (!Empty))
      dataCount <= datacount;
   else
      data Count <= datacount + 1;
   end
  else if ( rdEn && (!Empty))
   datacount <= datacount - 1;
 else     datacount <= datacount;
end
```

ضروری نیست

```
assign Full = (dataCount [8]) ? 1 : 0;            (~ |data Count )
assign Empty = (dataCount == 0) ? 1 : 0;   or assign Empty = (datacount == 0);
assign stackPointer = data count - 1;  →
```

اول کار stackpointer ، FFFF

در حین write ، data count زیاد می‌شود و stackpointer هم زیاد می‌گردد به آخرین خانه اشاره می‌کند.

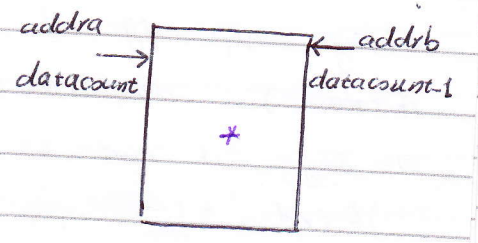
```
always @ (posedge clk or posedge Reset)
  if (Reset)   stackPointer <= 0;
else begin  if (WEn && (!FULL))   begin
  if (rdEn && (!Empty))
    stackpointer <= stackpointer;
  else
    stackpointer <= stackpointer + 1;
 end
```

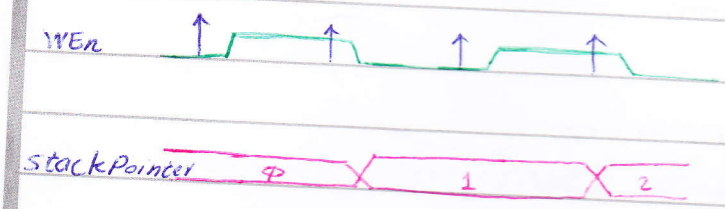
else if (rdEn && (!Empty))

stackPointer <= stackpointer-1;

else stackPointer <= stackPointer;

end





* در صورت استفاده از این حالت دیگر نیازی به تعریف Stack Pointer نیست.

datacount همیشه به آدرس خانه خالی اشاره می کند.

ادامه state machine جلسه قبل:

1: valid : DataToFiFo <= DataToFiFo;

WEnToFiFo <= 0;

2: !FULL To Data Capture: DataToFiFo <= Datain;

WEnToFiFo <= 1;

چون یک transition داریم این شرط هم نداریم. پس شرط ضروری نیست.

3,4: DataToFiFo <= DataToFiFo;

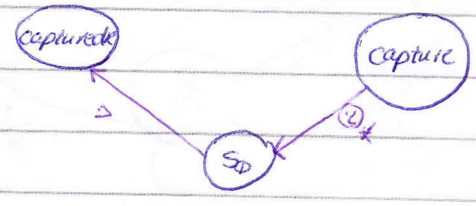
WEnToFiFo <= 0;

اولین حالتی که valid نیست از idle به capture می آییم.

وقتی valid صفر شود دیگه از خونه مربوطی نمی دایم.

فقط در transition از capture به captured خروجی‌گذاری می‌کنیم یک کار درست حسابی می‌کنیم.

ممکن است داده بالا رو زمانی که می‌خواهیم به FIFO بنویسیم FIFO پر باشد ولی در دو سیکل بعدی خالی شود پس می‌توان state machine را این صورت تغییر داد تا قابلیت آن افزایش یابد



*DataToFiFo <= Data In;
WEn <= 0;

نیازی به شرط ندارد زیرا Priority آن دو است.

Δ WEnToFiFo <= 1;

ممکن است FIFO پر باشد ولی عملاً سعی کردیم شرایط را کم خطر کنیم

طراحی با fsm خیلی راحت‌تر است. اگر software را نداشتیم کافی است روی کاغذ fsm را بکشیم و از روی آن کد بنویسیم.

Generate Thorough Component → تولید کد در قالب HDL

همه ورودی‌ها را wire و خروجی‌ها را reg تعریف کرده است به هر state یک عدد parameter نسبت داده است.

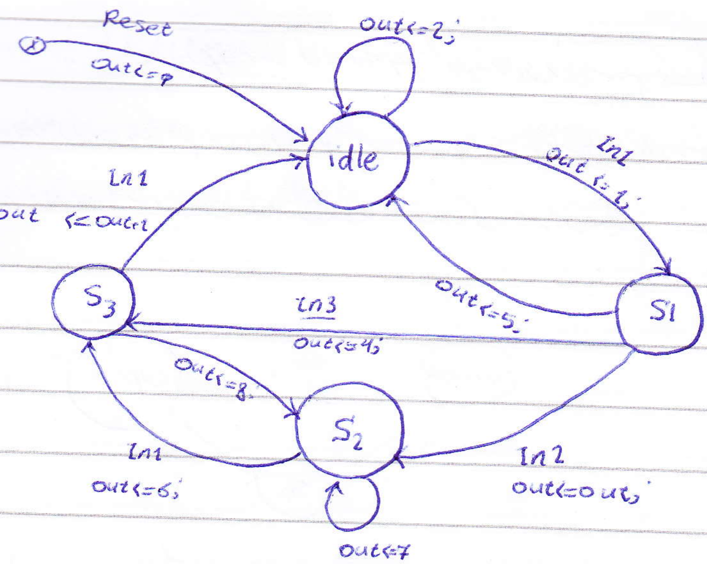
(تغییر دهنده state است)
یک always تولیدکننده خروجی است و یک always تولیدکننده حالت بعدی و یک always به state کنونی و ورودی‌ها next state را تولید می‌کند.

current state _ next state

begin : name

جلسه بیست و هشتم:

کوئیز:



کد وریلاگ معادل با FSM فوق را بنویسید.

ابتدا باید stateهای خود را تخصیص دهیم و این کار را با define انجام می‌دهیم.

```
'define idle 2'b00
'define s1   2'b01
'define s2   2'b10
'define s3   2'b11
```

معمولا این مقادیر هنگام صدا زدن ماژول از بالا تغییر می‌کند

باید با define تعریف شوند نه parameter.

```
wire [1:0] ns;          ns: next state
reg  [1:0] s;
assign ns = ((s == 'idle) && In1) ? 's1 :
            (s == 'idle) ? 'idle :
            ((s == 's1) && In2) ? 's2 :
            ((s == 's1) && In3) ? 's3 :
            (s == 's1) ? 'idle :
```

((S == 'S2) && In1) ? 'S3 ;

⋮

می توانستیم از always @(. . .) استفاده کنیم. (هرکدام از سیگنال های [illegible])

Redundancy: تکرار

```
always @ (posedge clk or posedge Reset)
 if (Reset) begin
   S <= 'idle;
   out <= Φ;
end
else begin
   S <= ns;
   case (S)
   'idle : if (In1) out <= 1; else out <= 2;
   'S1 : if (In2) out out ; else if (In3) out <= 4; else out <= 5;
```

موضوع تحقیق: استفاده های از تکنولوژی در FPGA

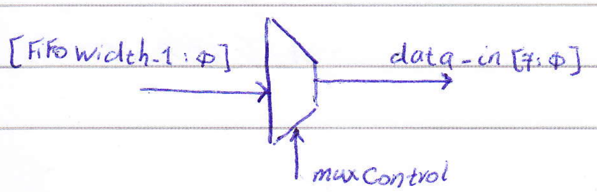
امروز می خواهیم که خواندن از FIFO را کامل کنیم.

(bbfifo - kcuart - uart_rx - uart_tx) ها

را در پروژه اضافه کنیم.

باید یک mux بگذاریم چون خروجی FIFO، 16 بیت است و ورودی uart، 8 بیت است، با mux

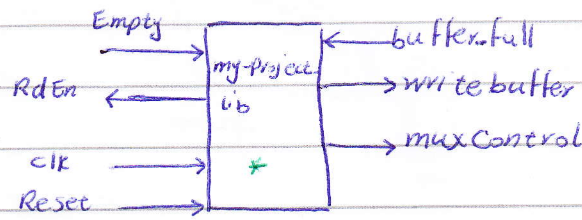
یک بار 8 بیت اول و یک بار 8 بیت دوم را می فرستیم.



mux روی → right click → text → می‌گوییم که در بلاک بنویسیم

assign data-in=(muxControl)? Dataout[15:8] :Dataout[7:0];

هر وقت می‌خواهیم داده‌ای به کامپیوتر ارسال کنیم داده را روی خطوط می‌گذاریم و یک سیکل write-buffer را یک می‌کنیم
از آنجا که داده‌های خروجی ۱۶ بیتی است برای هر بار خواندن یک داده ۱6 بیتی دو بار write-buffer یک می‌شود.



serial out → کامپیوتر

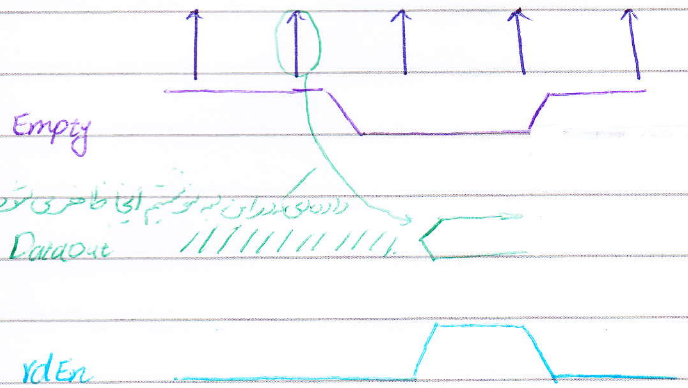
* double click → state dagram → Yes → Name: controler → Finish

idle: اگر FIFO که می‌خواهیم داده را از آن بخوانیم Empty باشد یا uart، FULL باشد در این حالت می‌مانیم و دوری می‌زنیم.

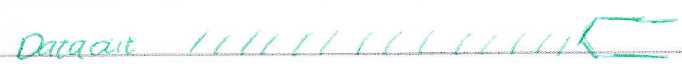
```
idle: Empty || buffer-Full
      RdEn<=0;
      mux-Control<=0;
      Write buffer <=0;
WRITE1: muxcontrol <=0;
      RdEn<=0;
      write buffer<=1;
```

هنوز داده را در uart ننوشته‌ایم که RdEn را یک کنیم
RdEn: چون این داده را خواندیم داده بعدی را بگذارد
ما که را طوری طراحی کرده‌ایم که در FIFO داده‌ای که
باید خوانده شود خودش روی خروجی است و با RdEn
داده بعدی در خروجی قرار می‌گیرد.

مدل FIFO ما :



مدل دیگر FIFO :



وقتی rdEn را می‌دید داده را در خروجی قرار می‌دهد.