آزمایشگاه معماری کامپیوتر

COMPUTER ARCHITECTURE LAB

به همراه زبان توصیف سخت افزاری VHDL

مدرس : مسعود دلدار

ویرایش دوم پاییز ۱۳۹۴

فهرست

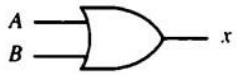
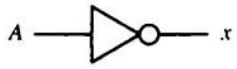
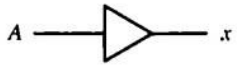
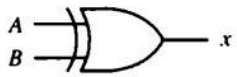
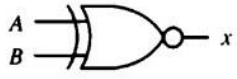
آلبرت انیشتین : به خاطر داشته باشید هر چه در مدارس خود یاد می گیرید نتیجه کار نسل های بیشماری است که در اثر کوشش آرزومندانه همه مردم جهان به ثمر رسیده است و سپرده ای است در دست شما ، از آن استفاده ببرید و به آن بیفزایید تا روزی که آنرا با کمال امانت و وفاداری به فرزندانتان بسپارید .

خواننده گرامی این جزوه در دست ویرایش می باشد در صورت مشاهده هرگونه ایراد یا اشکال تایپی و یا هرگونه انتقاد و پیشنهاد با ایمیل اینجانب تماس بگیرید deldar_edu@yahoo.com

ویرایش اول : بهار ۹۴

📖 دانشجوی گرامی جدول زیر را در هر جلسه به همراه داشته باشید این جدول ارزیابی شما در تمرینات می باشد .

ردیف	موضوع تمرین	شماره صفحه	ملاحظه
۱			
۲			
۳			
۴			
۵			
۶			
۷			
۸			
۹			
۱۰			
۱۱			
۱۲			
۱۳			
۱۴			
۱۵			
۱۶			
۱۷			
۱۸			
۱۹			
۲۰			
۲۱			
۲۲			
۲۳			
۲۴			
۲۵			
۲۶			
۲۷			
۲۸			
۲۹			
۳۰			
۳۱			
۳۲			
۳۳			
۳۴			

Name	Graphic symbol	Algebraic function	Truth table
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0
Buffer		$x = A$	A | x 0 | 0 1 | 1
NAND		$x = (AB)'$	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$x = (A + B)'$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

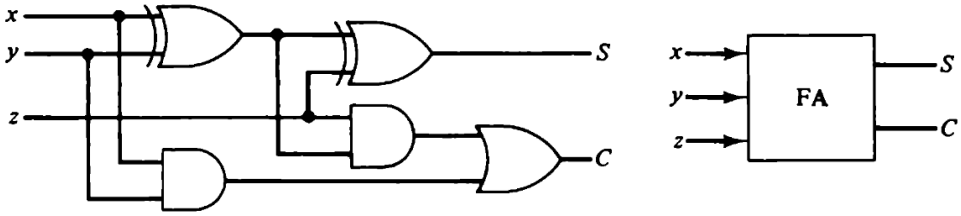
Digital logic gates.

آزمایش شماره ۱ : تست چند گیت ساده

ابتدا با کمک کدهای انتهای جزوه سعی کنید کدهای مربوط به گیت های AND و OR و XOR را نوشته و آنها را تست کنید .

📖 تمرین : سعی کنید کدهای مربوط به گیتهای NOT و NAND و NOR و XNOR و BUFFER را نوشته و آنها را تست نمایید .

آزمایش شماره ۲ : مدار تمام جمع کننده ***FULL ADDER***

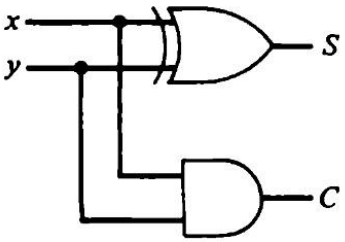


Logic diagram Block diagram

📖 تمرین ۱ : مدار نیم جمع کننده

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

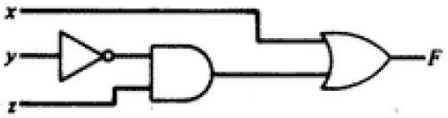
(a) Truth table



(b) Logic diagram

Half-adder.

📖 تمرین ۲ : مدار تابع رسم شده



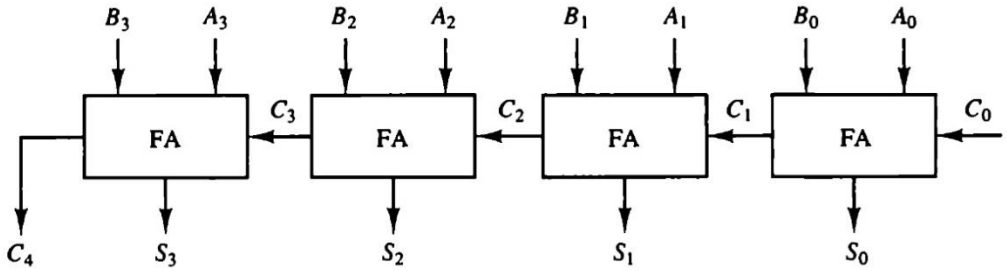
(ب) دیاگرام منطقی

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(الف) جدول درستی

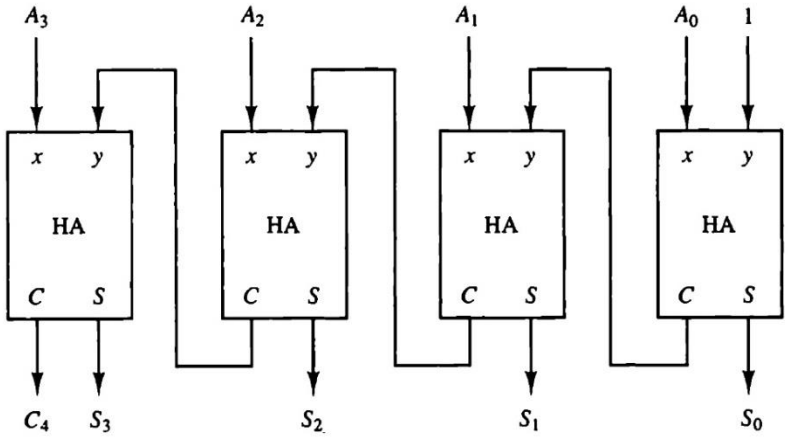
جدول درستی و دیاگرام منطقی برای $F = x + y'z$

آزمایش شماره ۳ : مدار جمع کننده چهاربیتی



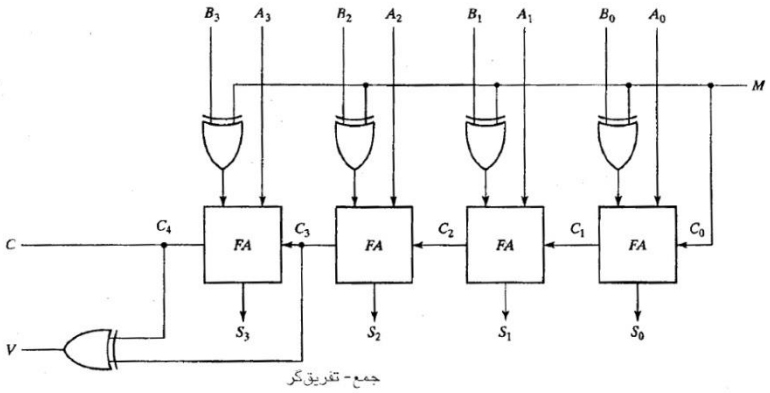
4-bit binary adder.

📖 تمرین ۱ : مدار افزایشگر دودویی با کمک نیم جمع کننده ها



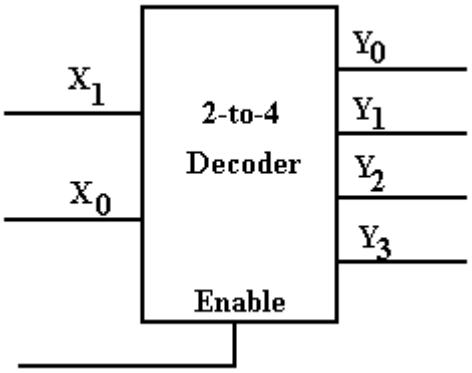
4-bit binary incrementer.

📖 تمرین ۲ : مدار جمع کننده – تفریق گر به همراه پرچم های CF , OF , SF , ZF

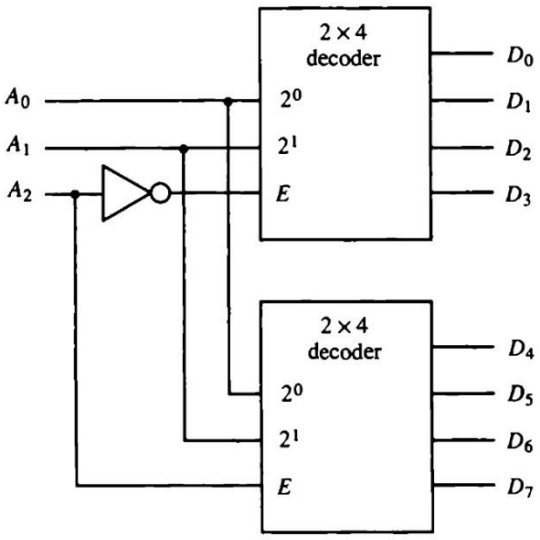


آزمایش شماره ۴ : دیکدر

❖ آزمایش های زیر را انجام و نتایج را ثبت کنید :



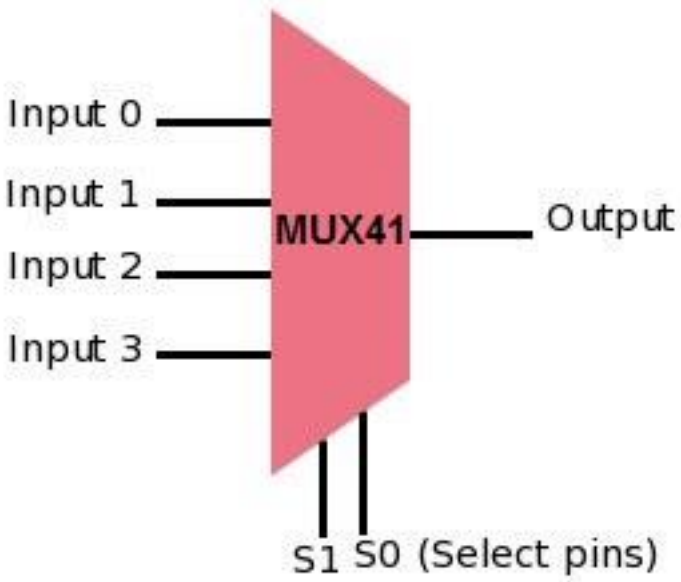
📖 تمرین۱ : به کمک دو دیکدر ۴×۲ یک دیکدر ۸×۳ طراحی کنید



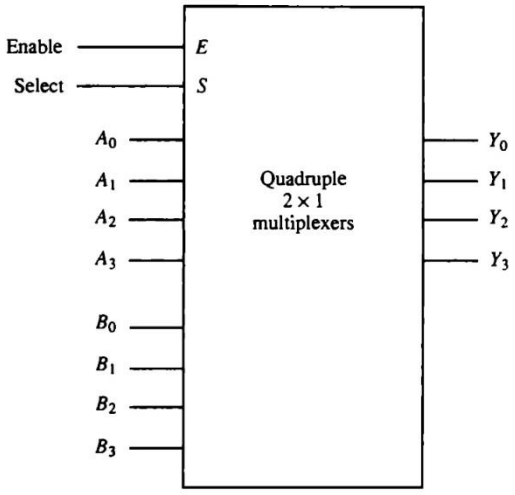
📖 تمرین۲ : ساخت دیکدر ۵ به ۳۲ با دیکدر های ۳ به ۸ و ورودی فعالساز و یک دیکدر ۲ به ۴

آزمایش شماره ۵ : مولتی پلکسر

❖ آزمایش های زیر را انجام و نتایج را ثبت کنید :



📖 تمرین۱ : به کمک ۴ مولتی پلکسر ۱×۲ یک مولتی پلکسر ۱×۲ چهار تایی طراحی و تست کنید .



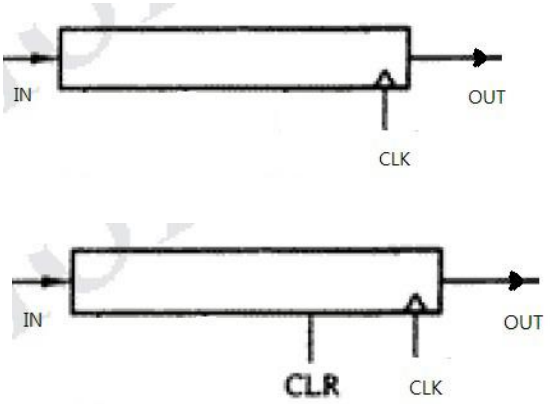
Block diagram

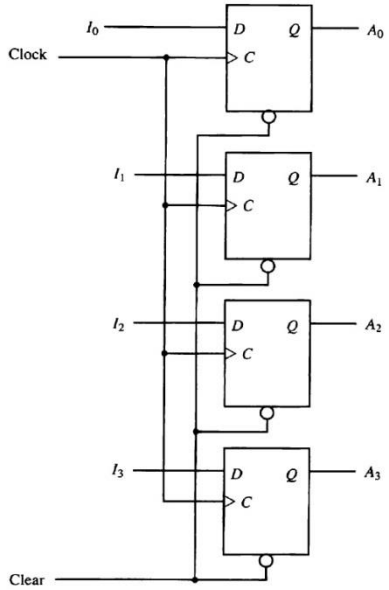
E	S	Y
0	×	All 0's
1	0	A
1	1	B

Function table

آزمایش شماره ۶ : ثبات ***register***

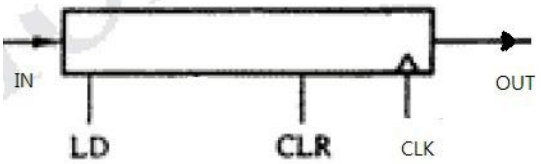
❖ ثبات ۴ بیتی زیر را به کمک فلیپ فلاپ D طراحی و تست کنید .

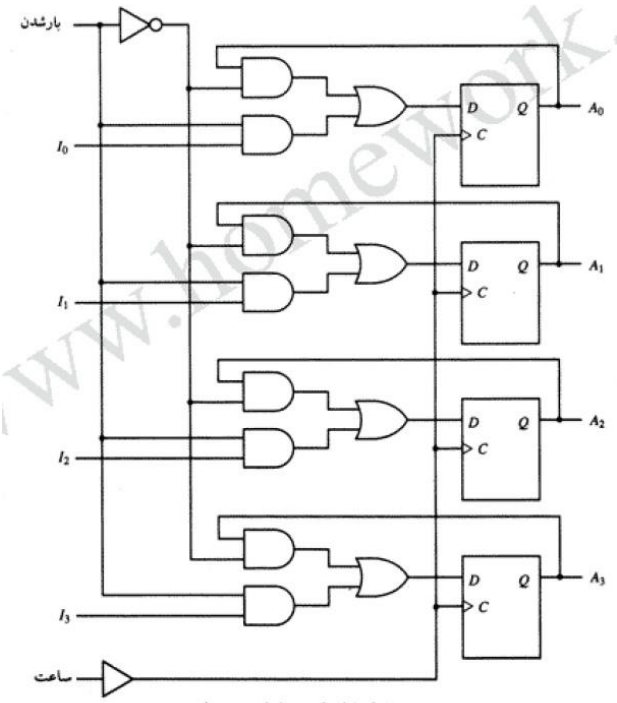




4-bit register.

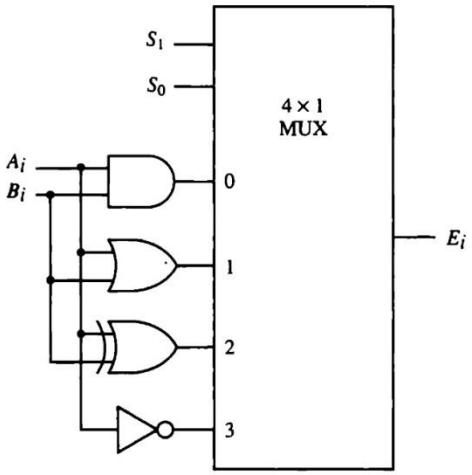
📖 تمرین۱ : ثبات ۴ بیتی زیر را تست کنید .





یک ثبات 4 بیت با بارشدن موازی

آزمایش شماره ۷ : طراحی مدار منطق



(a) Logic diagram

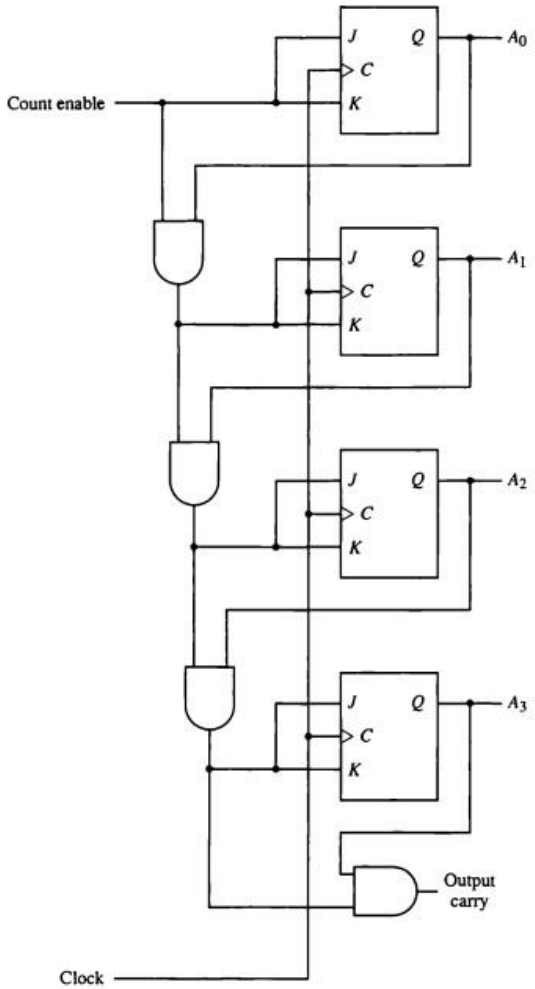
S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

📖 تمرین۱ : مدار منطقی طراحی کنید که جدول زیر را تحقق بخشد .

S2	S1	S0	out
0	0	0	**AND**
0	0	1	**OR**
0	1	0	**XOR**
0	1	1	**XNOR**
1	0	0	**NAND**
1	0	1	**NOR**
1	1	0	**NOT A**
1	1	1	**BUFFER**

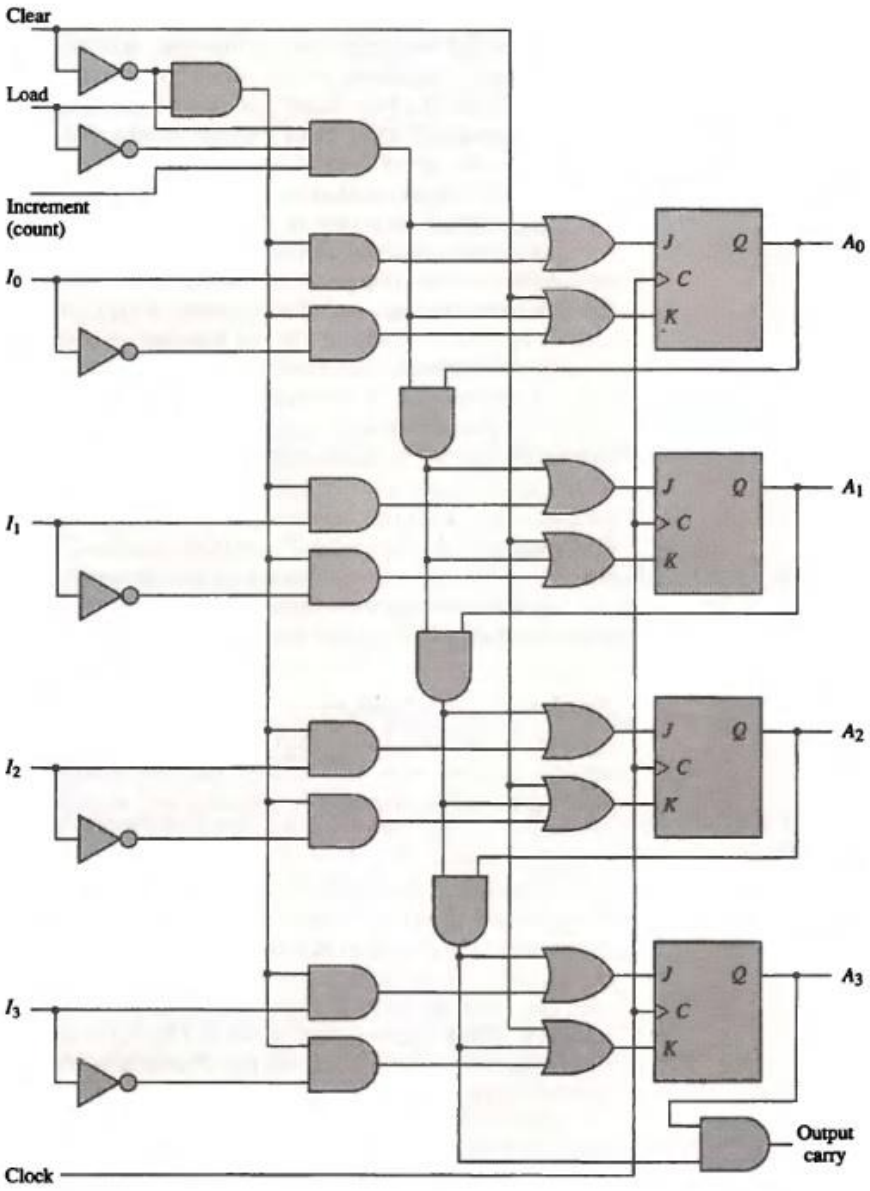
آزمایش شماره ۸ : شمارنده



4-bit synchronous binary counter.

📖 تمرین۱ : *یک شمارنده از صفر تا ۹۹طراحی کنید .*

📖 تمرین۲ : *شمارنده زیر را تست کنید . این شمارنده قابلیت شمارش از یک عدد مشخص را دارد .*



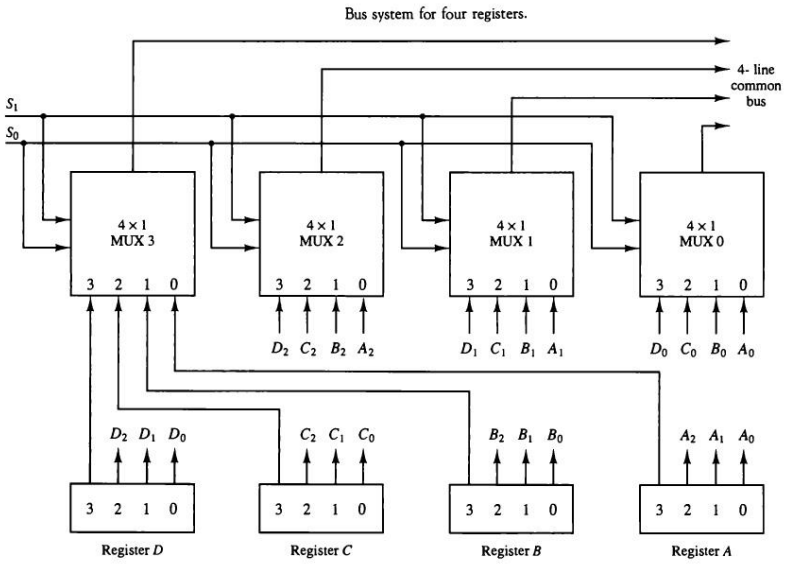
4-bit binary counter with parallel load and synchronous clear.

Function Table for the Register

Clock	Clear	Load	Increment	Operation
↑	0	0	0	No change
↑	0	0	1	Increment count by 1
↑	0	1	×	Load inputs I_0 through I_3
↑	1	×	×	Clear outputs to 0

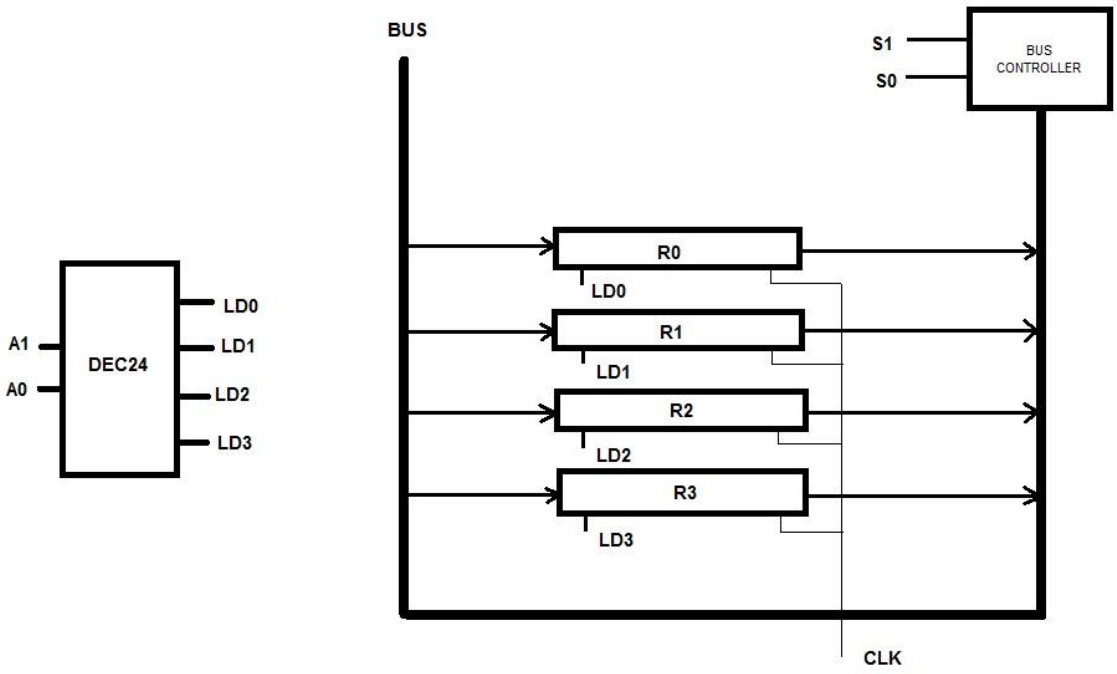
آزمایش شماره ۹ : طراحی مدار گذرگاه

❖ مدار شکل زیر یک گذرگاه ۴ بیتی را نشان می دهد .

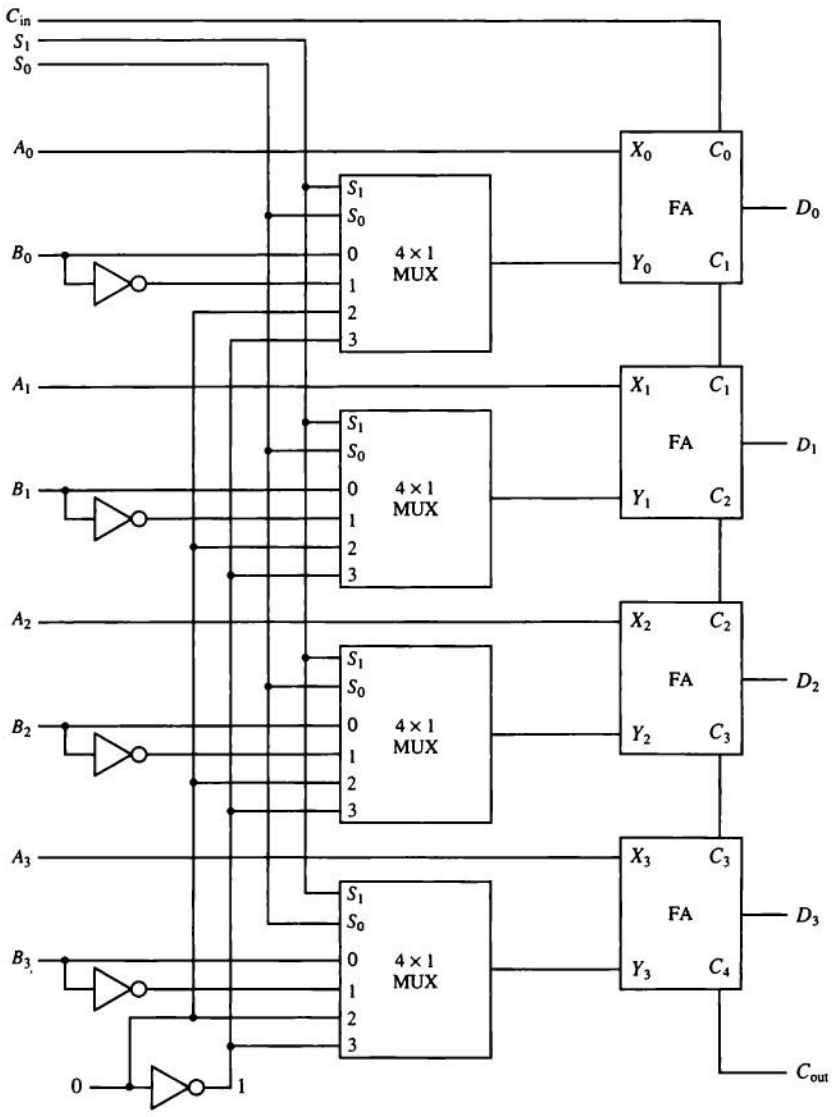


Function Table for Bus

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D



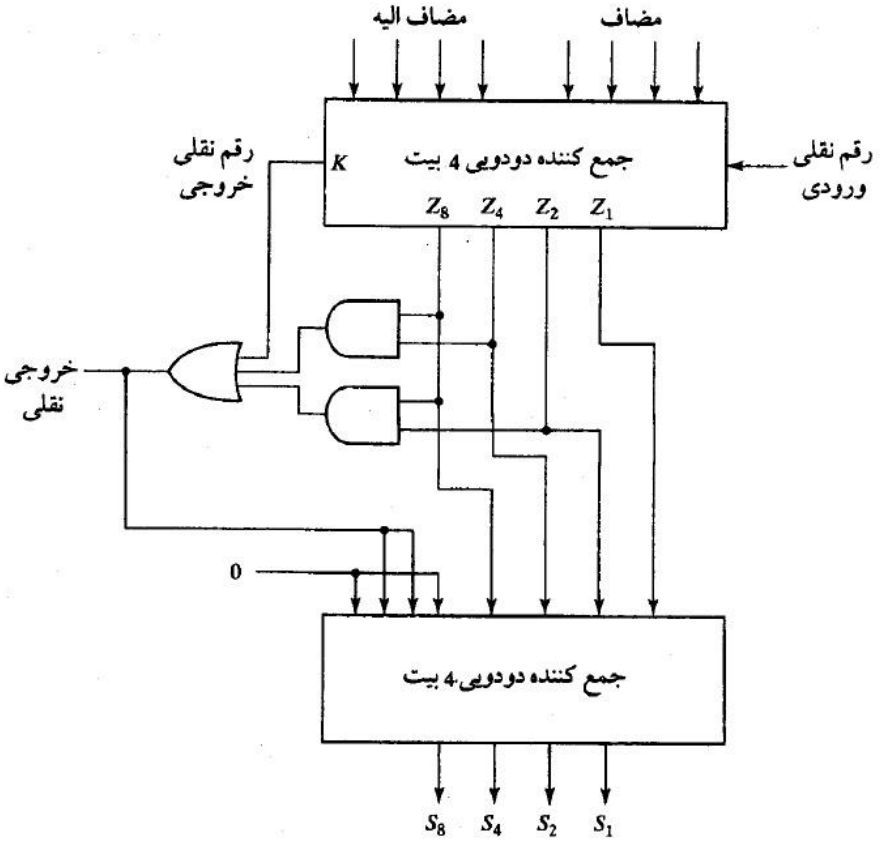
آزمایش شماره ۱۰ : طراحی مدار حساب



Arithmetic Circuit Function Table

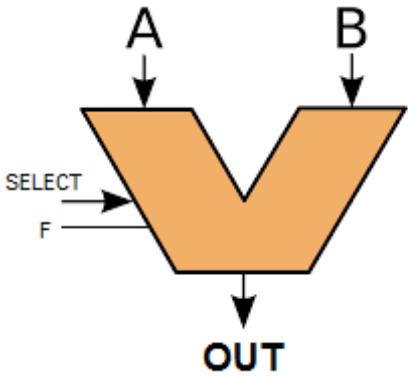
Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

📖 تمرین۱ : جمع کننده BCD



نمودار بلوکی یک جمع‌کننده BCD

آزمایش شماره ۱۱ : طراحی مدار محاسبه و منطق *ALU*



❖ یک واحد محاسبه و منطق ۴ بیتی بصورت زیر طراحی کنید

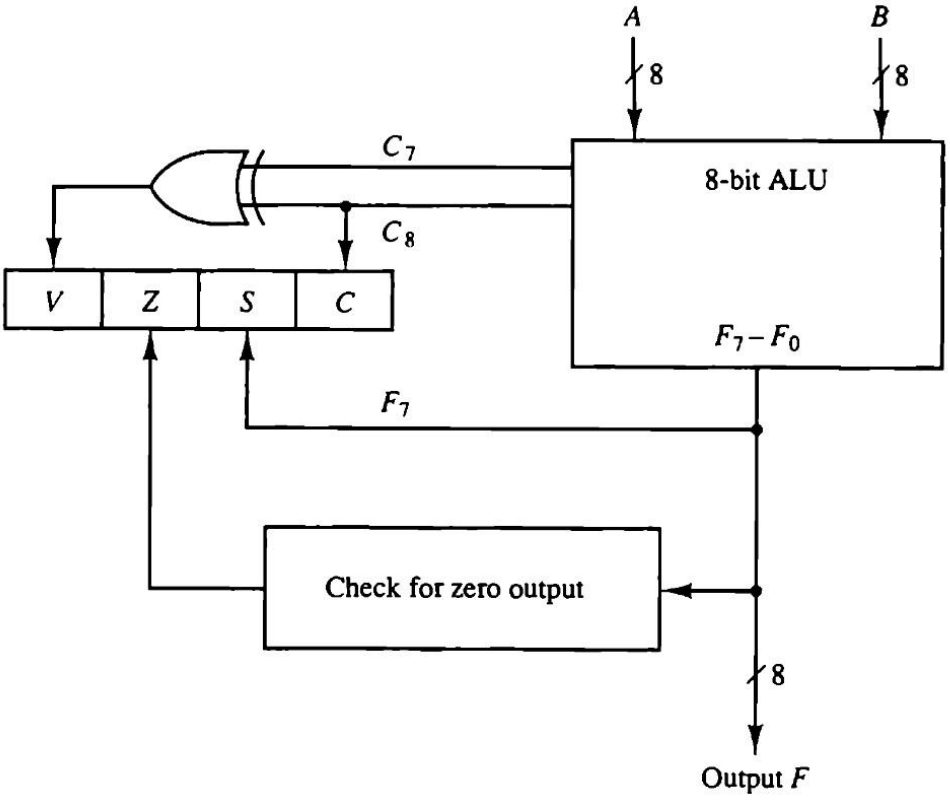
F	S1	S0	OUT
0	0	0	**AND**
0	0	1	**OR**
0	1	0	**NOT A**
1	1	0	**A+B**
1	1	1	**A-B**

📖 تمرین۱ : یک ALU هشت بیتی بصورت زیر طراحی کنید .

Function Table for Arithmetic Logic Shift Unit

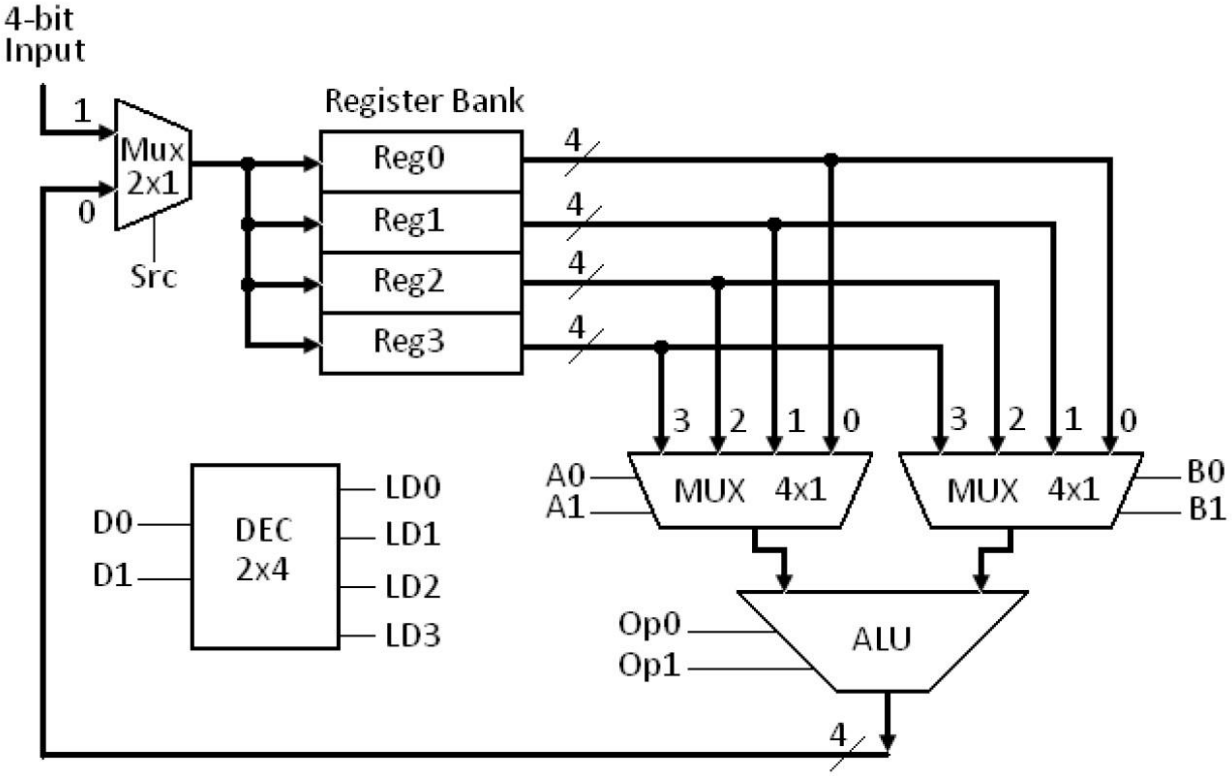
Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

📖 تمرین۲ : یک **ALU** هشت بیتی به همراه ثبات پرچم طراحی کنید . (از تمرین قبل استفاده کنید)

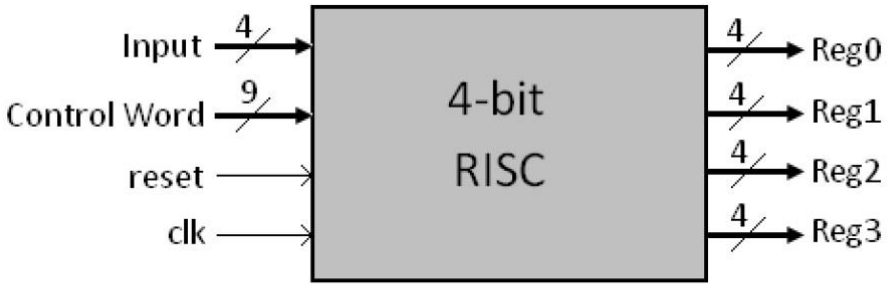


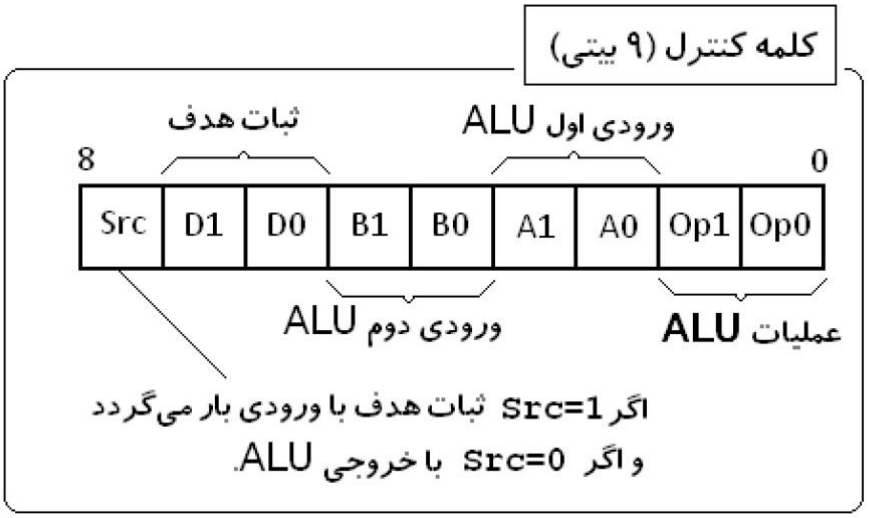
آزمایش شماره ۱۲ : طراحی واحد کنترل

❖ طرح زیر نمای ساده یک کامپیوتر RISC به همراه control word را نشان می دهد .



❖ دیاگرام واحد کنترل





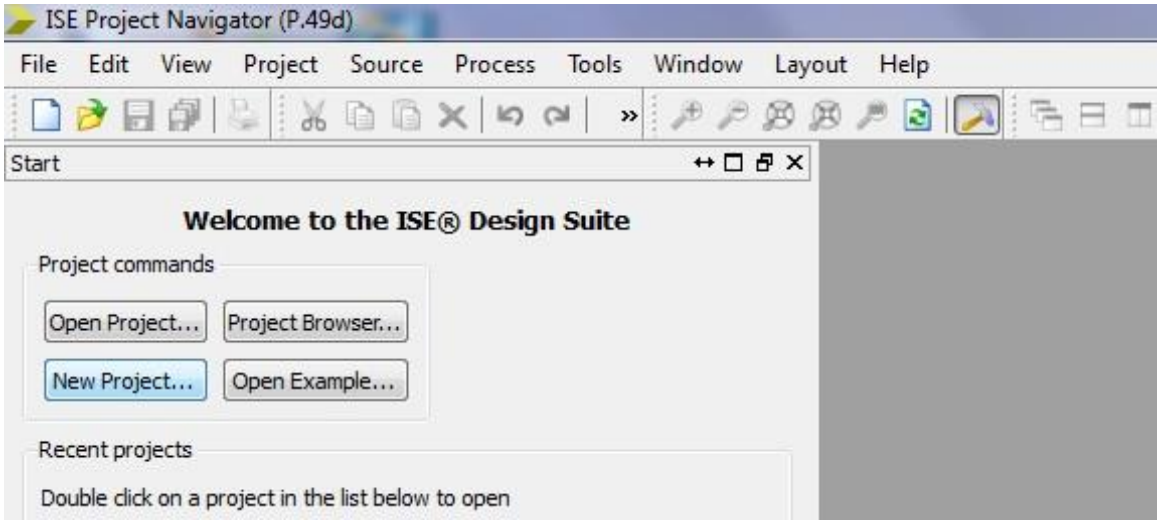
طراحی و شبیه سازی چیپ های شرکت XILINX به کمک نرم افزار ISE14.4 :

فرایند طراحی :

ابتدا بر روی آیکن زیر کلیک کنید

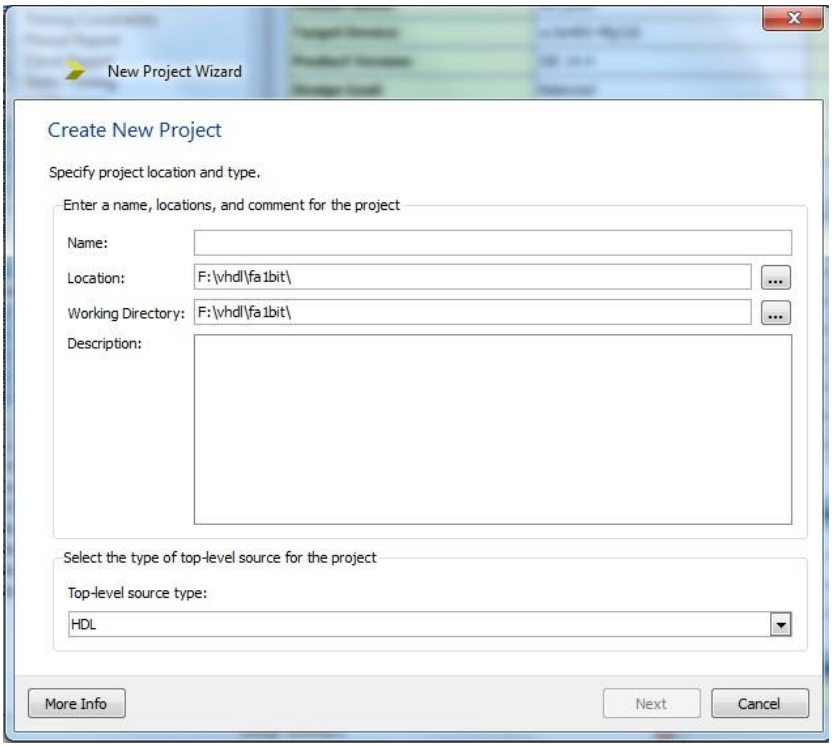
پنجره زیر ظاهر می شود . در صورت عدم ظاهر شدن پنجره به فرم فوق ابتدا از مسیر زیر پروژه در حال انجام را بسته و سپس پروژه جدید را باز کنید .

File > Cloce Project…



سپس از مسیر زیر پروژه جدید را شروع می کنیم .

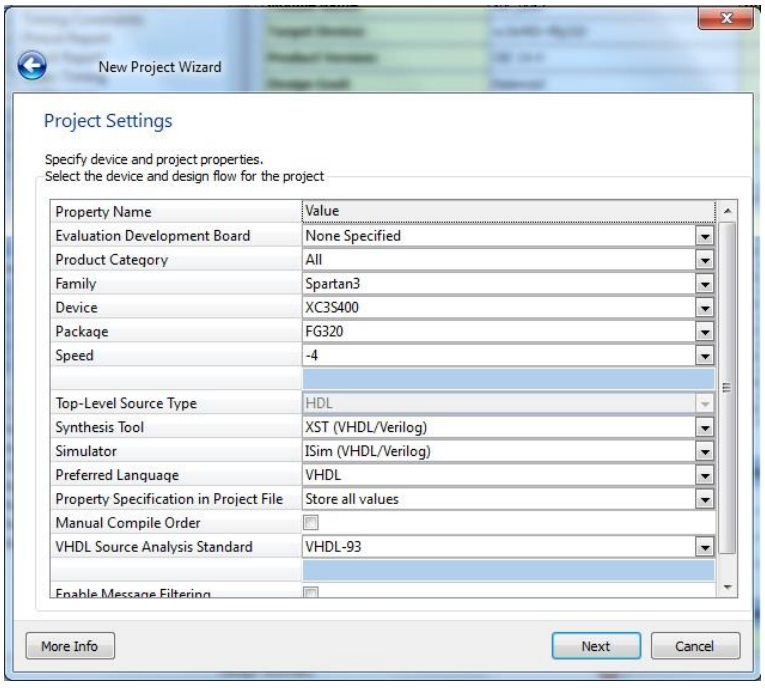
New Project…



در قسمت Name می توان یک اسم دلخواه و در قسمت Location مسیر ذخیره پروژه را انتخاب و مشخص می کنیم.

همچنین در قسمت Top-level source type می توان نوع یا روش توصیف طراحی را مشخص کرد .

با کلیک بر روی Next به مرحله بعد می رویم :



در قسمت Family نوع خانواده چیپ ها و در قسمت Device شماره چیپ و در قسمت Package نوع بسته بندی چیپ و در قسمت Speed فرکانس کاری چیپ را مشخص می کنیم.

پارامترهای دیگری نیز برای تنظیم کردن وجود دارند مثلاً نوع شبیه ساز و یا ابزار سنتز و ...

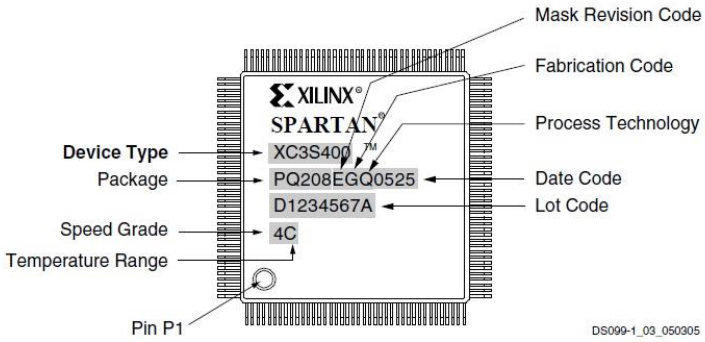
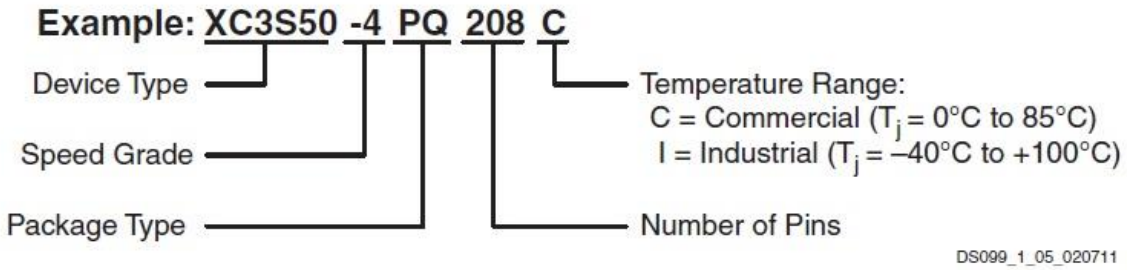


Figure 2: **Spartan-3 FPGA QFP Package Marking Example for Part Number XC3S400-4PQ208C**

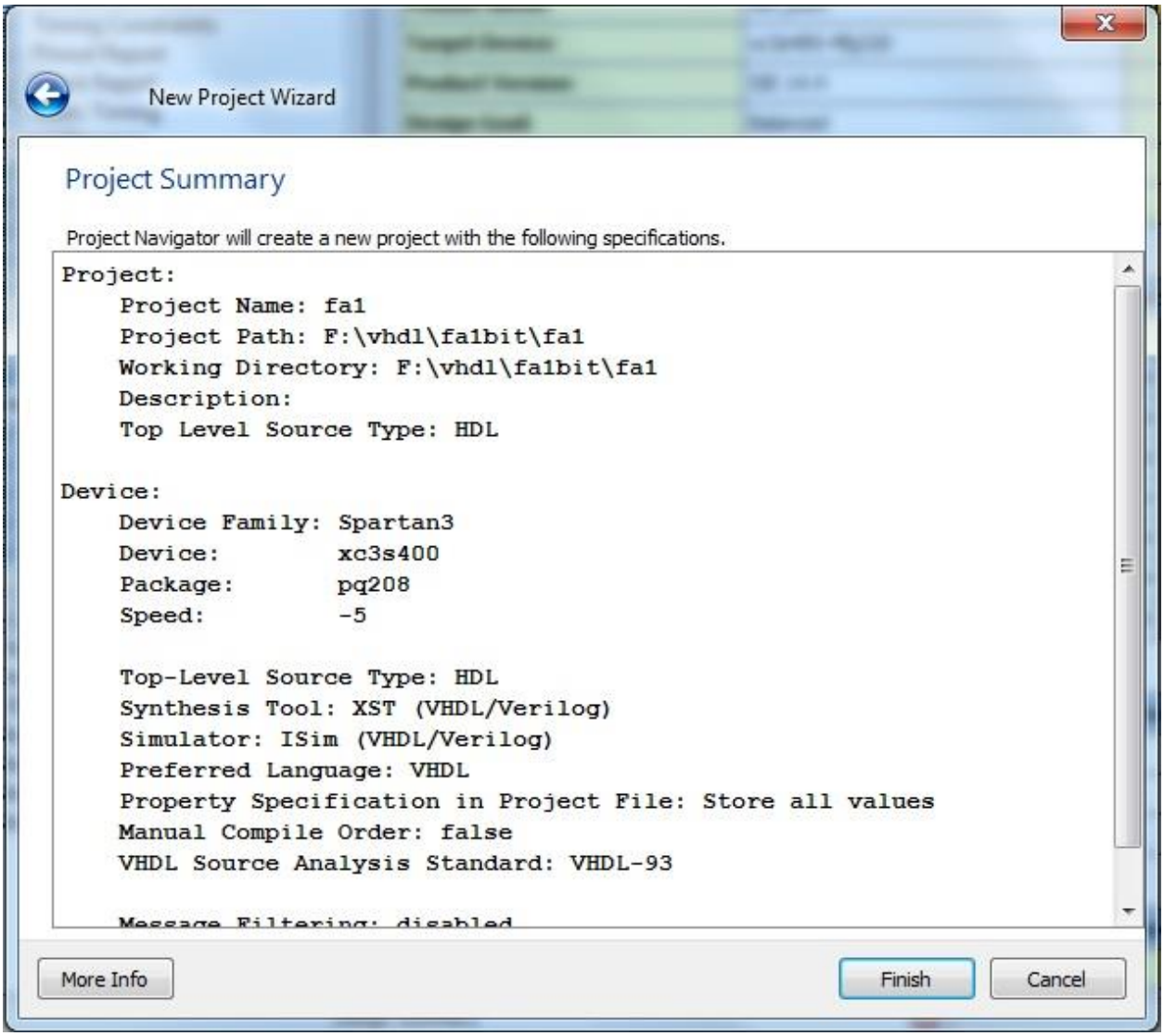
DS099 (v3.1) June 27, 2013
Product Specification
www.xilinx.com
5



-4 > 630MHz

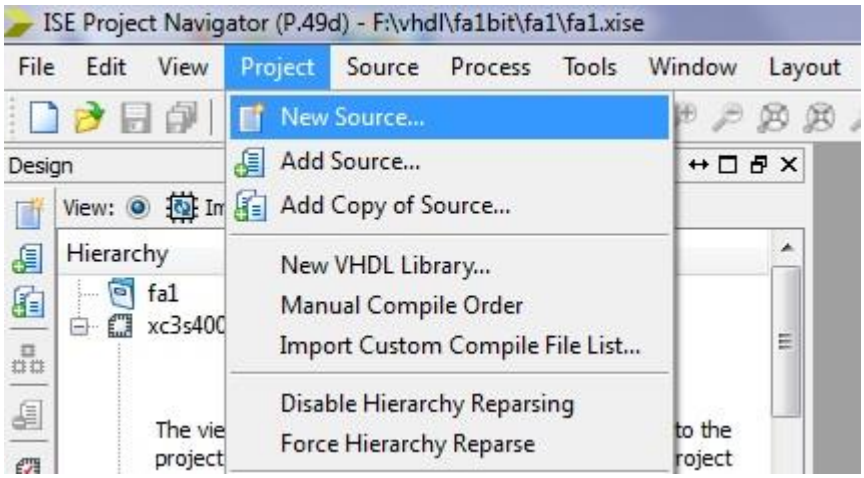
-5 > 725MHz

با کلیک بر روی Next به مرحله بعد می رویم :



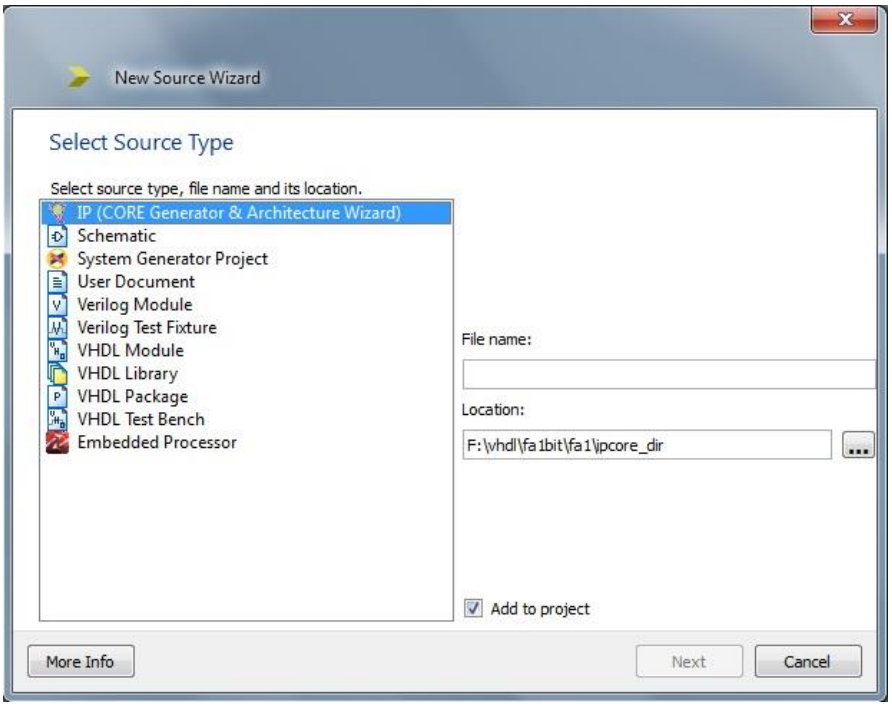
در شکل فوق یک گزارش و خلاصه ای از تنظیمات انجام شده مشاهده می گردد . بر روی Finish کلیک می کنیم .

از مسیر زیر یک source جدید ایجاد می کنیم :

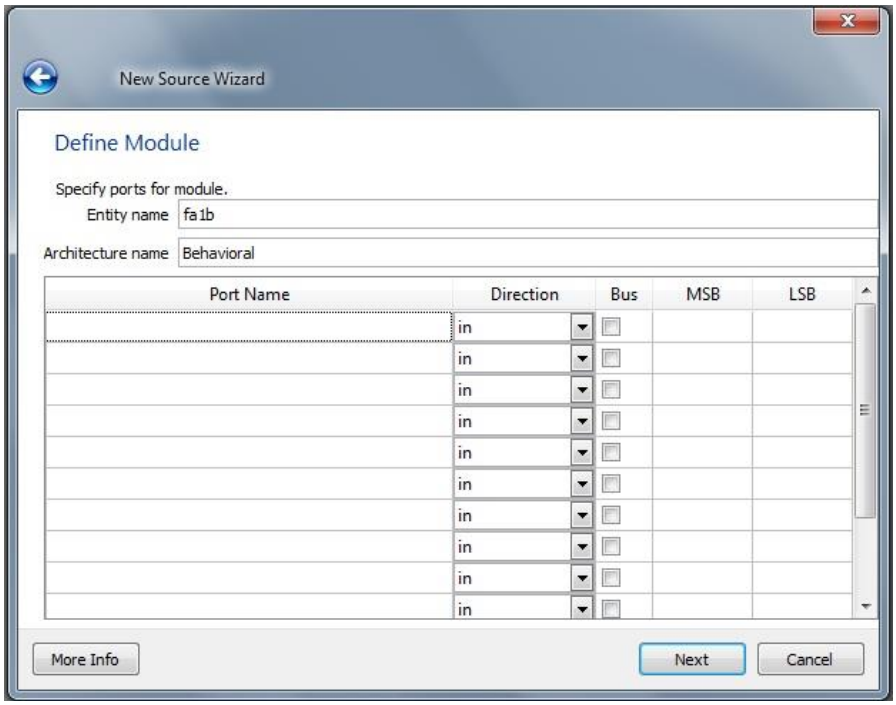


در پنجره ظاهر شده نوع عملیات را انتخاب می کنیم مثلا VHDL Module

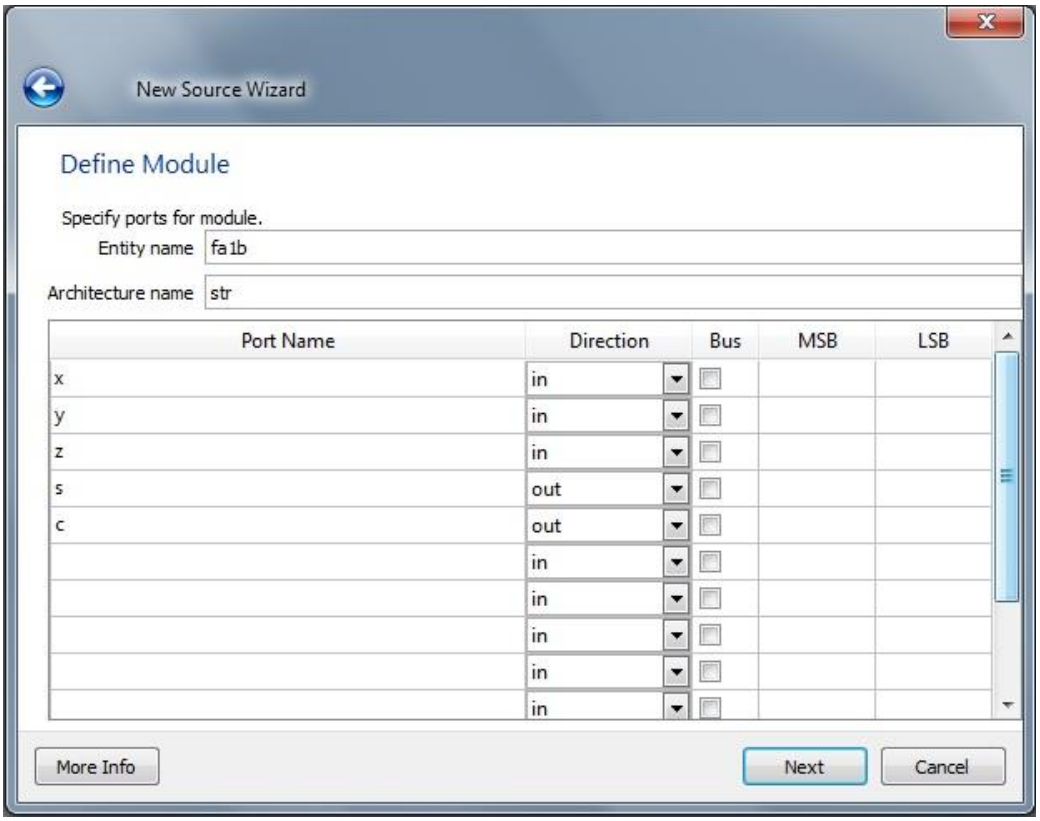
همچنین یک اسم برای این نوع عملیات انتخاب و مسیر ذخیره نیز مشخص می شود ، دقت شود نام فایل بایستی با نام Entity یکسان باشد :(نکته اینکه با انتخاب User Document می توان یک فایل متن مثل توضیحات به طرح اصلی اضافه نمود و System Generator Project جهت ارتباط با نرم افزار متلب و Embedded Processor جهت ساخت پردازنده میکروبلیزر می باشد)



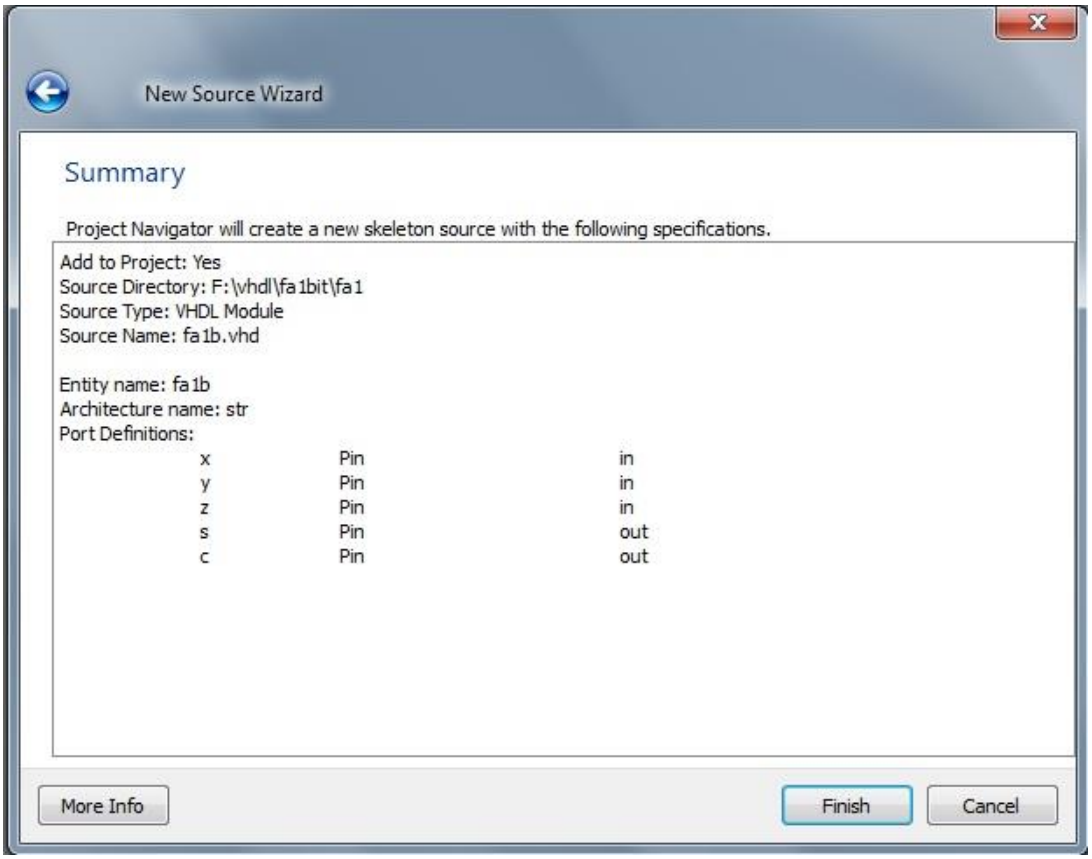
با کلیک بر روی Next به مرحله بعد می رویم :



در این مرحله بایستی پورت های ورودی و خروجی یا دو طرفه را مشخص کنیم مثلا به صورت زیر :



با کلیک بر روی Next به مرحله بعد می رویم :

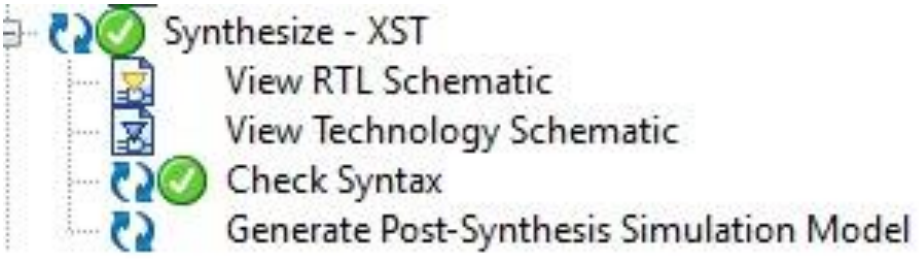


در شکل فوق یک گزارش و خلاصه ای از تنظیمات پورتها مشاهده می گردد . بر روی Finish کلیک می کنیم .

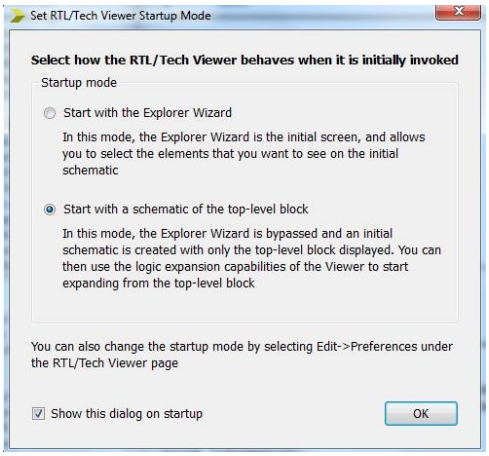
در پنجره ظاهر شده می توان کدهای VHDL را تایپ نمود مثلا برای یک گیت AND بصورت زیر :

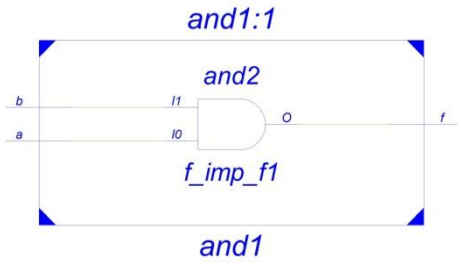

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity and1 is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           f : out  STD_LOGIC);
end and1;

architecture Behavioral of and1 is

begin

f<=a and b;

end Behavioral;
```

پس از تایپ کدها ابتدا بایستی کدها از نظر خطای نگارش بررسی و سپس عملیات سنتز انجام شود بدین منظور ابتدا بر روی Check Syntax کلیک و سپس بر روی Synthesize - XST کلیک می کنیم . خطا های احتمالی را نیز برطرف کرده و مجددا عملیات فوق را تکرار می کنیم . **Xilinx Synthesis Technology (XST)**



با کلیک بر روی View RTL Schematic می توان نتیجه عملیات سنتز را مشاهده کرد مثلا برای یک گیت AND نتیجه زیر ظاهر شده است :

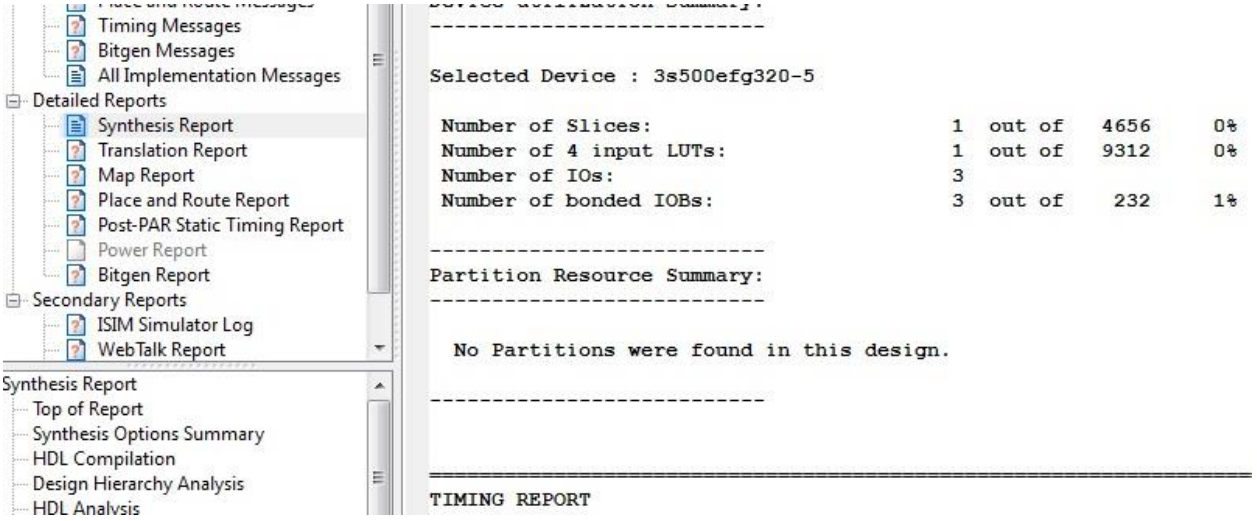




از قسمت زیر می توان تعداد قطعات استفاده شده از چیپ و قطعات باقیمانده را مشاهده کرد :

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1	4656	
Number of 4 input LUTs	1	9312	
Number of bonded IOBs	3	232	

از این قسمت می توان جهت گزارش گیری از عملیات سنتز استفاده نمود :

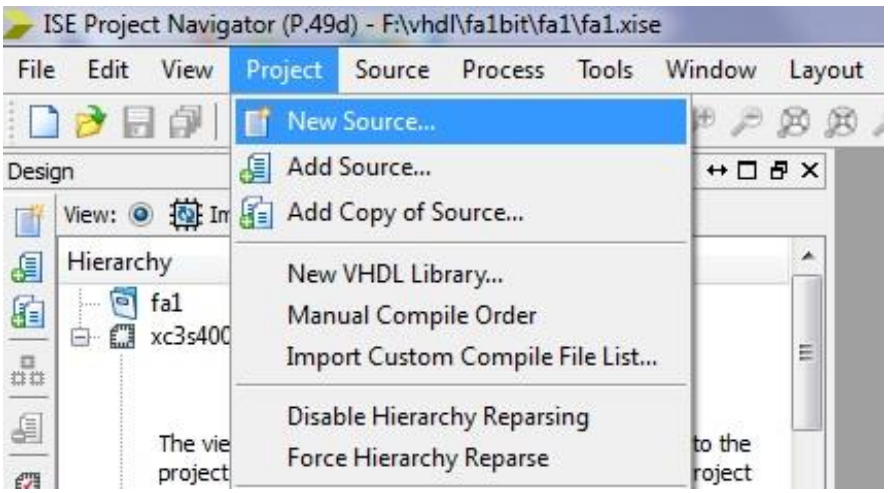


فرایند شبیه سازی :

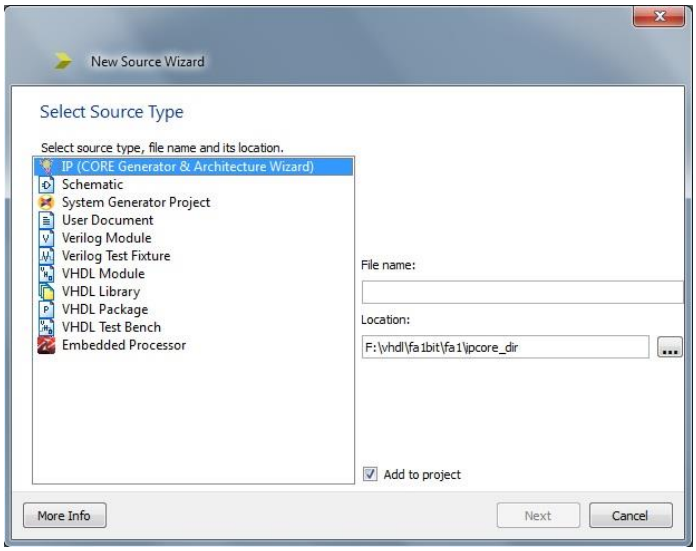
ابتدا بر روی ایکن زیر کلیک می کنیم

به دو روش می توان شبیه سازی را انجام داد البته از طریق نرم افزار Modelsim نیز شبیه سازی انجام می شود که بعداً به آن خواهیم پرداخت .

در این روش که روش حرفه ای تری می باشد از یک برنامه به نام Test Bench برای شبیه سازی استفاده می شود بدین صورت که ابتدا بایستی از پنجره زیر یک Source جدید ایجاد کرد



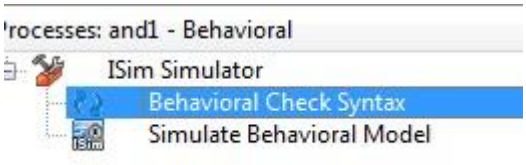
سپس از پنجره زیر گزینه VHDL Test Bench را انتخاب کرد



با انتخاب این گزینه و انتخاب اسم و مکان یک پنجره ظاهر شده که در آن بایستی کد VHDL مربوط به برنامه شبیه سازی نوشته شود . مثلا یک نمونه برنامه تست جهت تست گیت AND به صورت زیر است :

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY test1 IS
END test1;
ARCHITECTURE behavior OF test1 IS
    COMPONENT and_2input
    PORT(
         a : IN  std_logic;
         b : IN  std_logic;
         f : OUT  std_logic );
    END COMPONENT;
   signal a : std_logic := '0';
   signal b : std_logic := '0';
   signal f : std_logic;
BEGIN
 uut: and_2input PORT MAP (
          a => a,
          b => b,
          f => f  );


 a <= '1' after 100ns,'0' after 200ns,'1' after 300ns,'0' after
400ns,'1' after 500ns,'0' after 600ns;
     b <= '1' after 200ns,'0' after 400ns,'1' after 600ns;


END;
```

پس از پایان نوشتن ابتدا بایستی از طریق ایکن Behavioral Check Syntax خطای نگارشی بررسی و سپس از گزینه Simulate Behavioral Model عملیات شبیه سازی انجام می شود منتها این بار بطور اتوماتیک شبیه سازی انجام خواهد شد



با کلیک بر روی ایکن زیر کل فضای شبیه سازی قابل رویت می باشد

برخی گزینه های شبیه سازی :

از سمت چپ restart و run all و step (جهت مشاهده اجرای خط به خط) و Re-launch (اگر در برنامه اصلی تغییری دادیم با اعمال این گزینه تغییرات را به شبیه سازی منتقل می کنیم)

روش استفاده از IP CORE ها

Core چیست ؟

هرگاه پروژه مشترکی توسط عده ای متفاوت انجام می شود هر کدام از طراحان ماژول خود را تا سه مرحله طراحی و شبیه سازی و سنتز پیش میبرند و تحویل طراح نهایی می دهند . طراح نهایی ماژول های تست و سنتز شده مابقی را گرفته و اتصالات بین ماژول ها را برقرار و Implement نهایی را انجام خواهد داد .

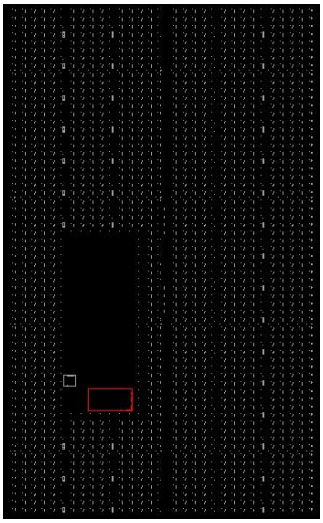
بطور خلاصه Core یک ماژل تست و سنتز شده است و شرکتهای تولید کننده Core در حقیقت طرح سنتز شده را به کاربر می دهند که با این کار Source Code آنها حفظ خواهد شد .

انواع Core :

۱- Source Code
۲- فایل سنتز شده با پسوند edif.
۳- Implement Core
۴- Hard Core

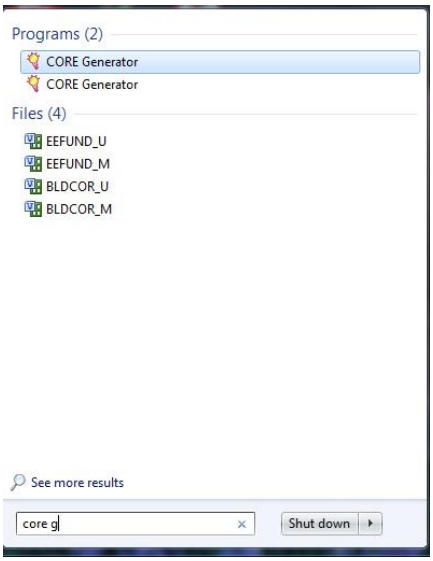
سه مورد اول Soft Core هستند در واقع کاربر می تواند این Core ها را گرفته و در قسمتی از FPGA قرار دهد در حقیقت چیپ FPGA که شامل یک مشت گیت خام بود اکنون بخشی از گیت های آن صرف ساختن Core شده است .

اما مورد چهارم Core از ابتدا در داخل FPGA قرار دارد یعنی در داخل این مدل FPGA یک مشت گیت خام به همراه یک Core سخت افزاری موجود است . در برخی طراحی ها به دلیل محدودیت ایجاد شده نمی توان از Core های آماده استفاده کرد و مجبور هستیم از Core سخت افزاری استفاده کنیم . شکل زیر نمای داخلی چیپ XC4VFX12 خانواده vertix4 می باشد که شامل بلوک های SLICE بلوک های RAM شانزده کیلو بایتی ، یک HARD CORE پردازنده۳۲ بیتی Power Pc405که تا حدود 400MHz کار می کند و یک HARD CORE مربوط به لایه MAC گیگابیت اترنت و سایر قسمتها دیده می شود .

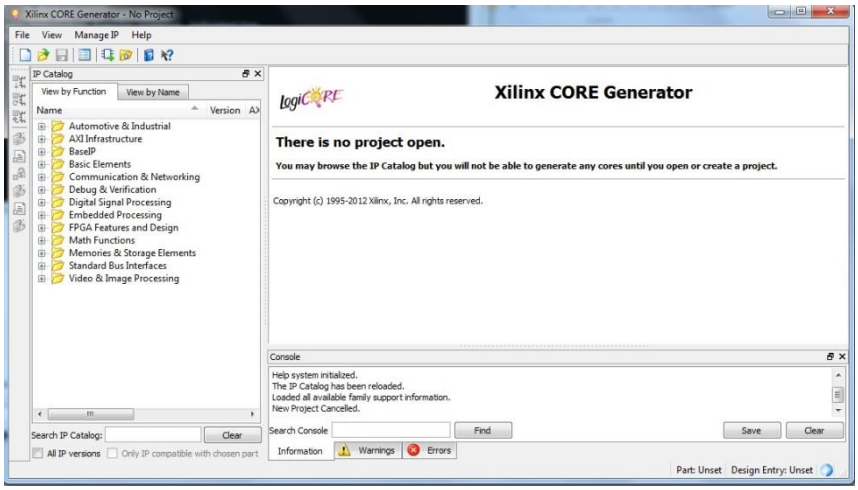


جهت ایجاد یک core بصورت زیر عمل می کنیم :

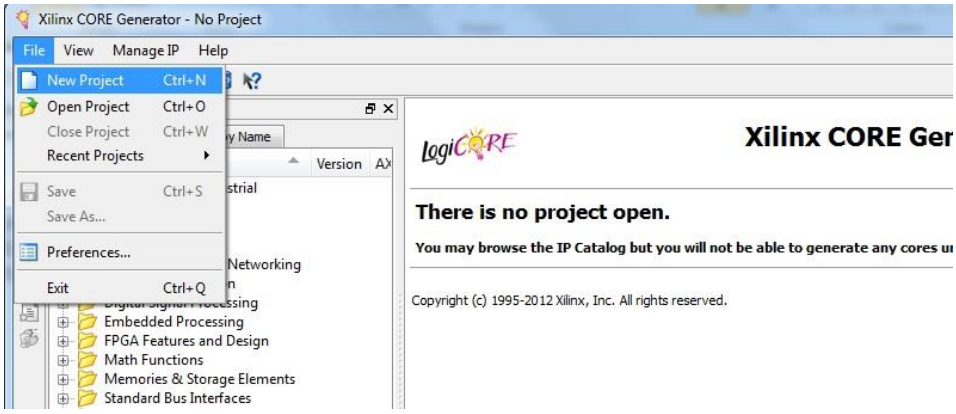
ابتدا یک پوشه جدید ایجاد کرده و سپس از قسمت زیر نرم افزار core generator را اجرا می کنیم :



پنجره اصلی نرم افزار باز می شود :

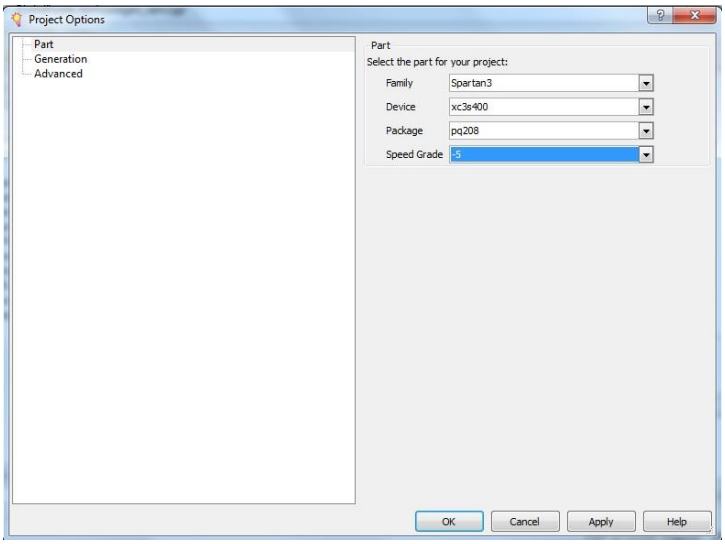


یک پروژه جدید ایجاد می کنیم :

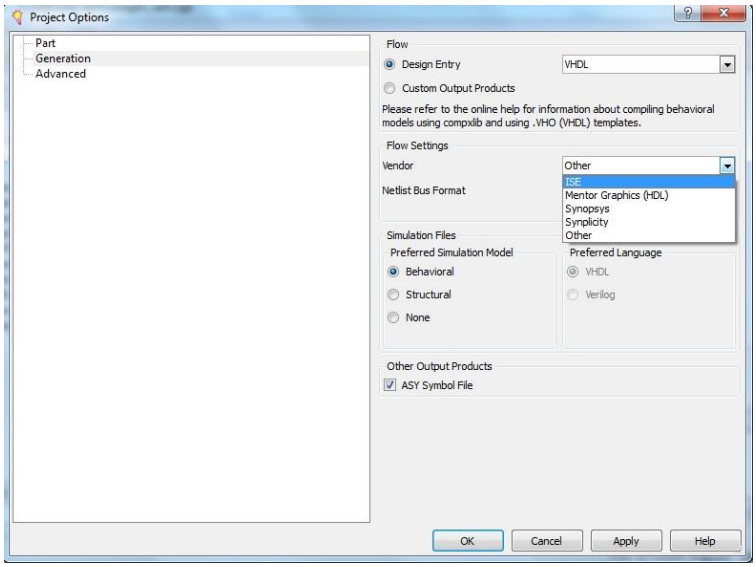


در این مرحله در سمت چپ سه قسمت دیده می شود

ابتدا از قسمت Part نوع چیپ را انتخاب می کنیم



از قسمت generation از قسمت Vendor می توان نوع شرکتی که core را ارائه کرده انتخاب کرد .



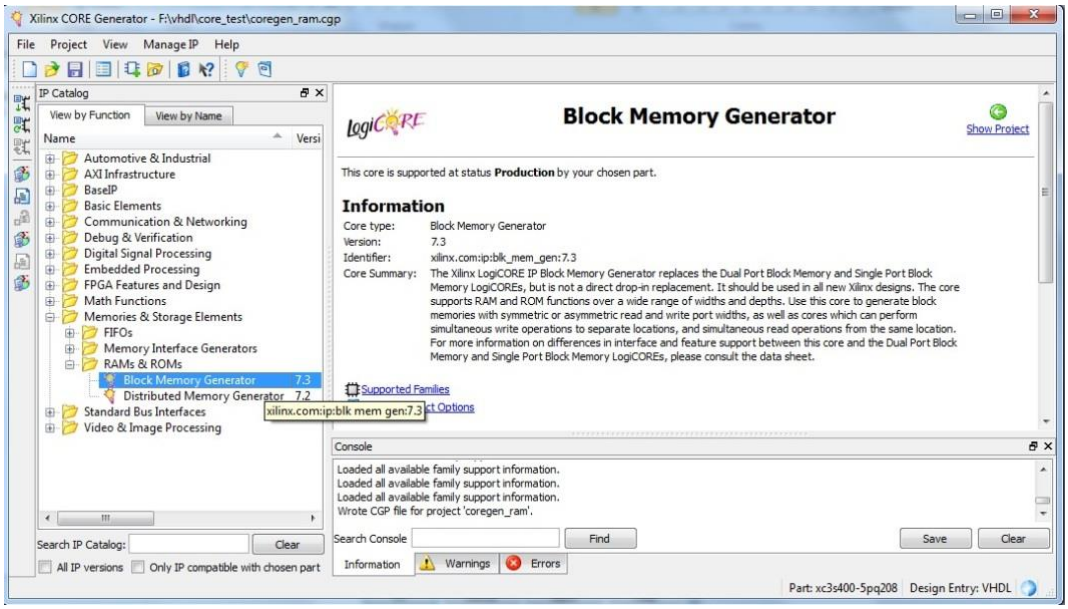
فعلا با قسمت Advance کاری نداشته و Ok می کنیم .

حال فرض کنید بخواهیم یک ماژول RAM طراحی کنیم معمولا در داخل FPGA ها به دو طریق می توان ماژول RAM را طراحی کرد

تکنیک اول استفاده از بلوک های RAM آماده سخت افزاری که در داخل FPGA قرار دارد که به آن Block Memory گویند

تکنیک دوم استفاده از گیت ها و فلیپ فلاپ ها و SRAM های موجود در LUT ها و یا استفاده از SLICE ها می باشد که به آن Distributed Memory گویند .

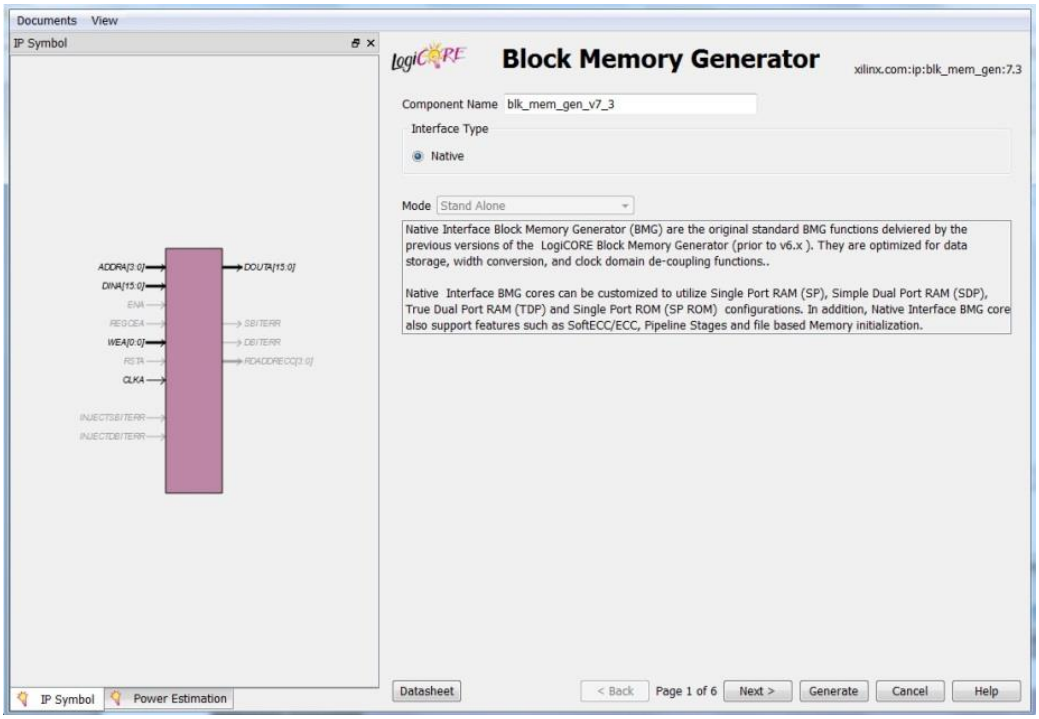
در شکل زیر از قسمت نشان داده شده core را انتخاب می کنیم



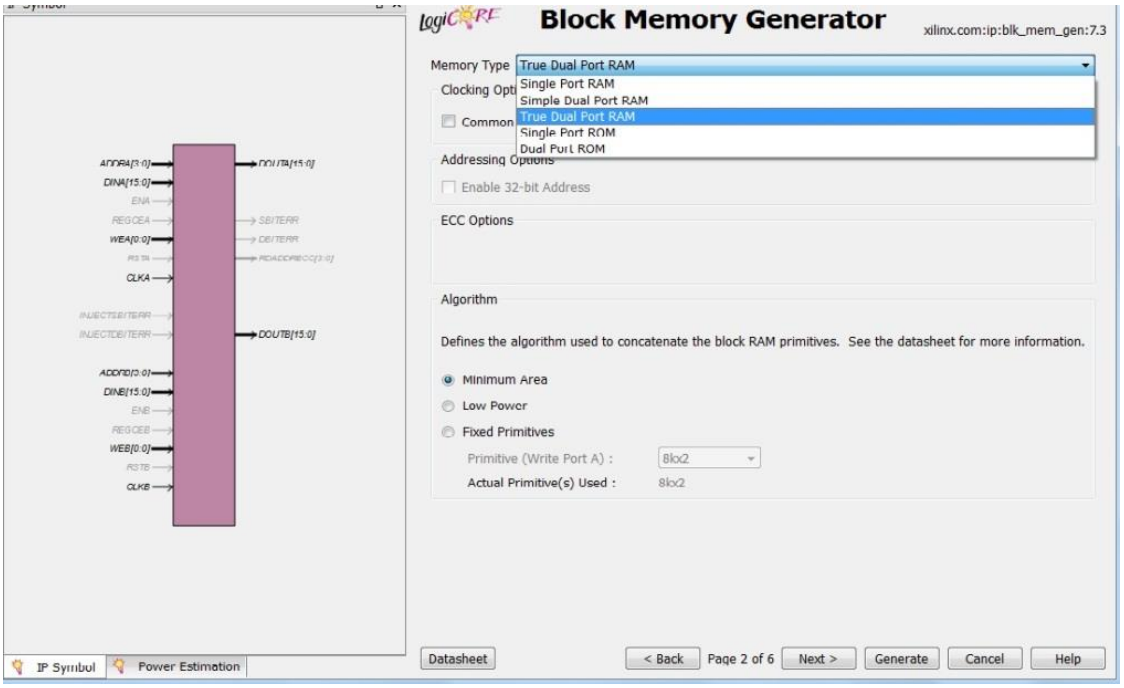
پنجره جدید مربوط به RAM انتخابی باز خواهد شد .

در ادامه تنظیمات مربوط به یک حافظه دو پورت را انجام خواهیم داد این حافظه شامل دو پورت دیتا دو پورت آدرس دو پایه clock دو پایه خواندن دو پایه نوشتن و دو پایه enable می باشد . یک فضای مشترک از دو جهت بطور همزمان قابل دسترسی می باشد . البته لزومی به مساوی بودن مشخصات دو پورت نیست .

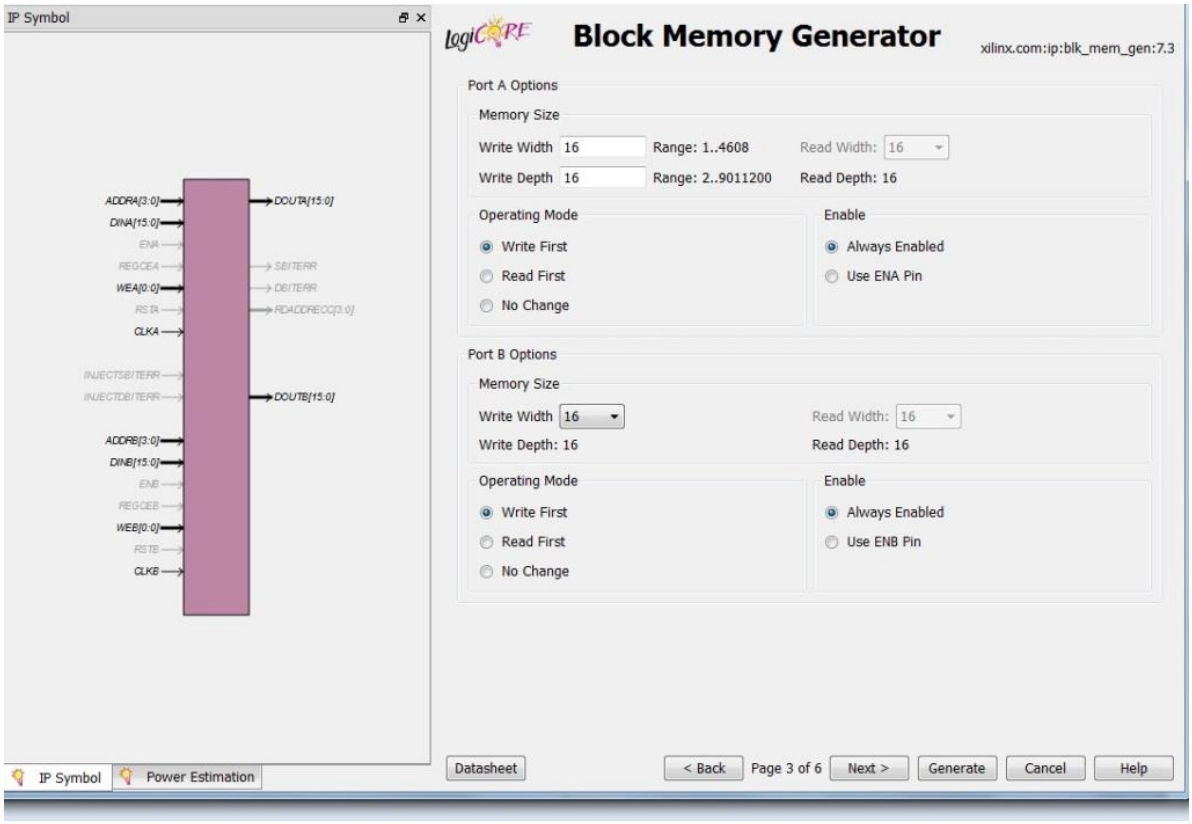
در شکل زیر یک اسم برای ماژول انتخاب می کنیم :



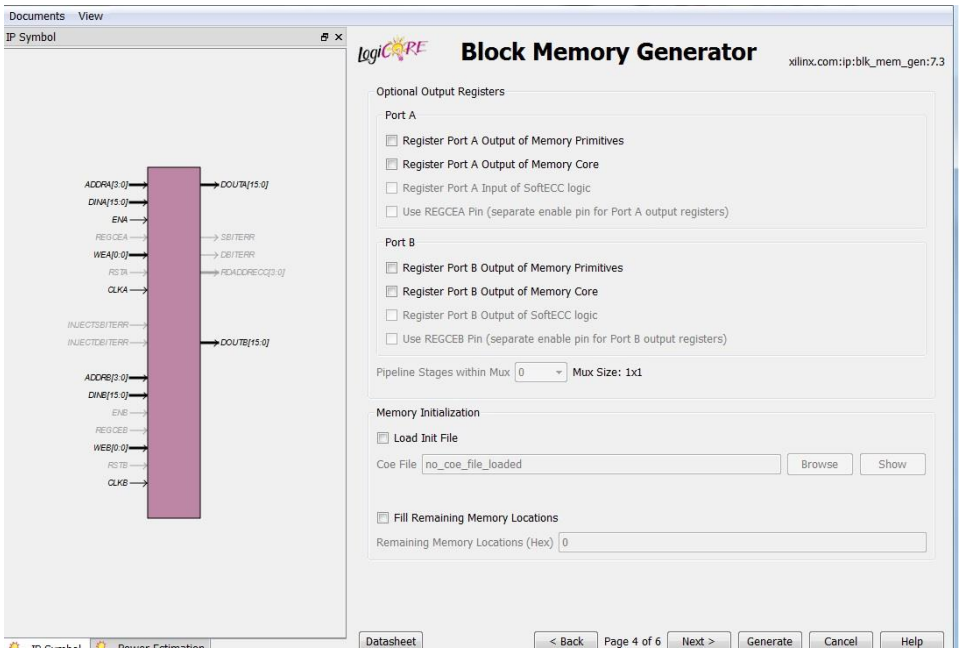
با دکمه Next به مرحله بعد می رویم



با دکمه Next به مرحله بعد می رویم



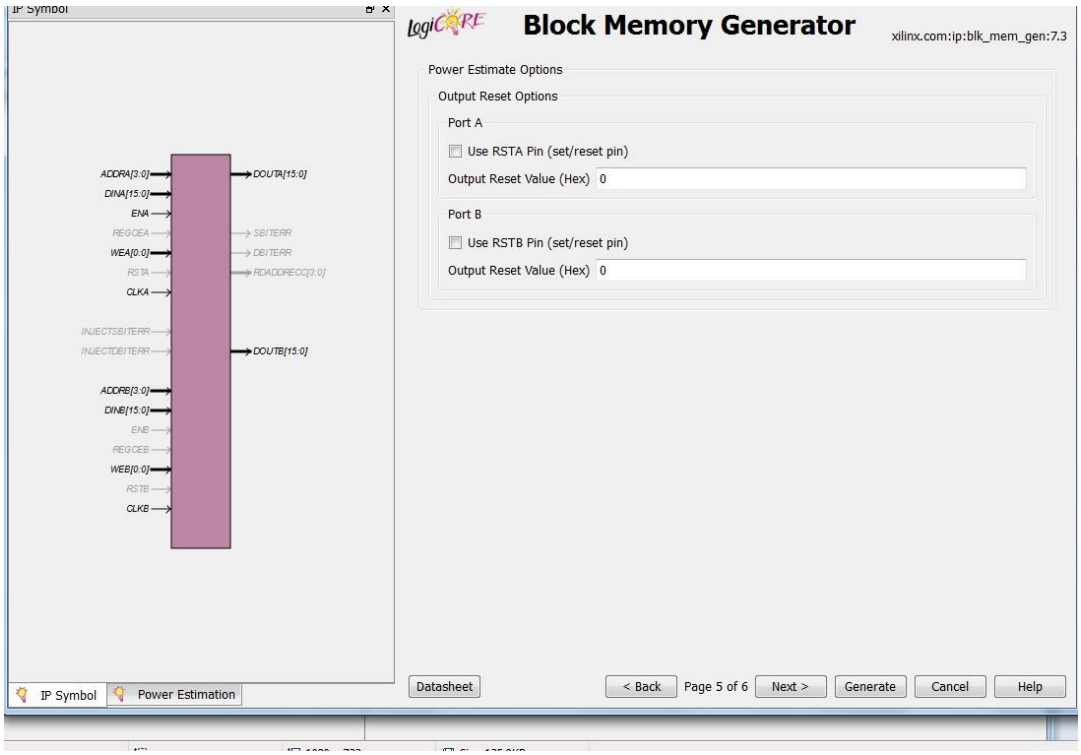
با دکمه Next به مرحله بعد می رویم



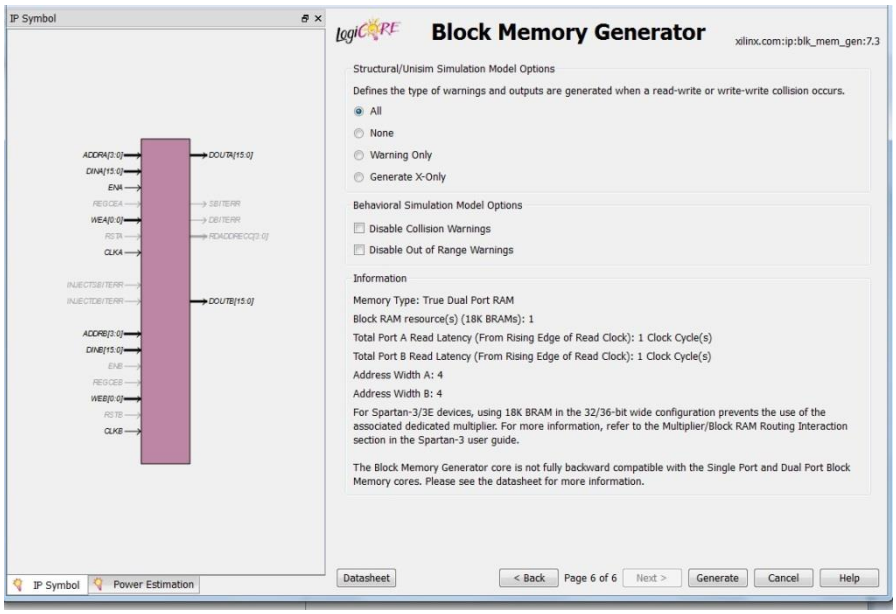
نکته ای که مهم است در شکل فوق می توان به حافظه مقدار اولیه داد : Memory Initialization

در واقع می خواهیم زمانی که FPGA بالا آمد یک مقدار اولیه در حافظه باشد . می توان یک فایل با پسوند Coe نوشت و در این قسمت Load کرد و یا مابقی را با یک عدد مشخص پر کرد .

با دکمه Next به مرحله بعد می رویم



با دکمه Next به مرحله بعد می رویم



دکمه Generate را کلیک تا فرایند تولید Core آغاز شود .

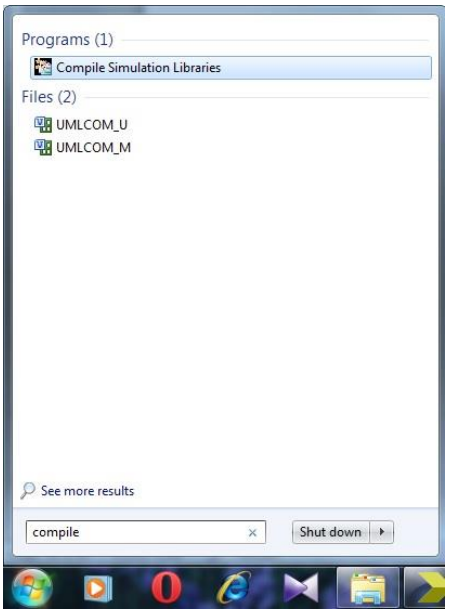
ازجمله فایل های مهمی که پس از فرایند تولید Core ایجاد می شود فایل با پسوند NGC می باشد که در Ise می توان از این فایل برای مرحله Implement استفاده کرد .

نکته مهم : در هنگام شبیه سازی یک فایل VHDL که یک یا چند Core در آنها استفاده شده است از آنجایی که سورس فایل های مربوط به Core وجود ندارند احتمال خطا وجود دارد .

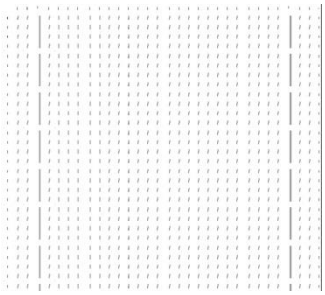
از مسیر زیر کتابخانه مربوط به Core ها را بایستی به طرح اصلی اضافه کرده و کامپایل کرد :

Xilinx > 14.4 > Ise_Ds > Ise > Vhdl > Scr > Xilinx core Lib

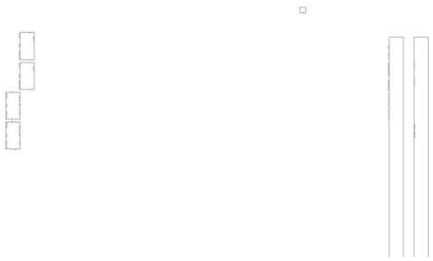
البته از مسیر زیر نیز می توان کمک گرفت :



پس از آنکه این Core را در یک برنامه قرار داده و آزمایش کردیم با ورود به نرم افزار FPGA Editor می توانیم داخل چیپ را مشاهده و نحوه اتصالات را بررسی کنیم و یا حتی قطعات استفاده شده را مشاهده و کنترل کنیم در شکل زیر نمای داخلی چیپ XCS400 خانواده SPARTAN3 را مشاهده می کنید .

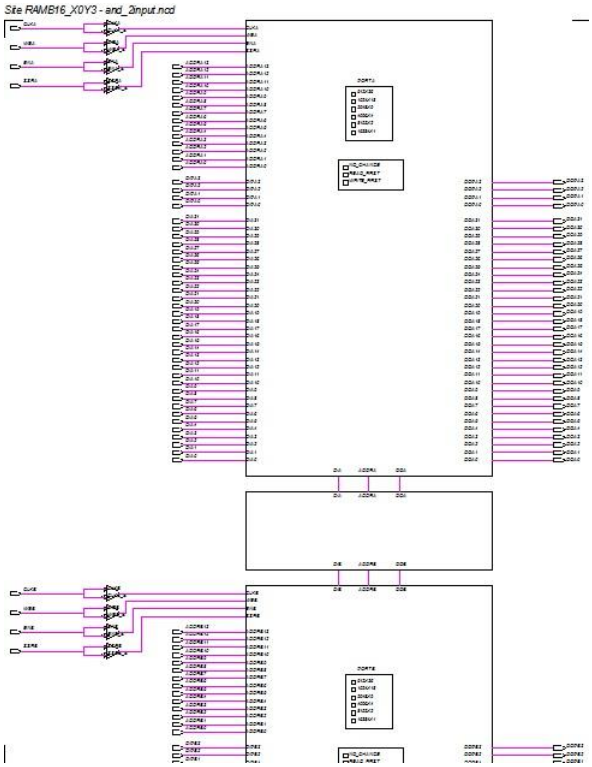


بلوک های کوچک SLICE ها و بلوک های بزرگ که در دو سمت چیپ قرار دارند Block Ram می باشند .

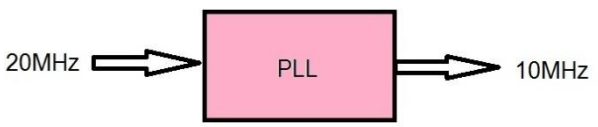


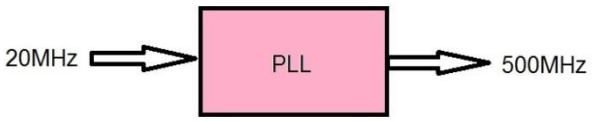
با کمی بزرگنمایی مشاهده می شود بلوک های بزرگ در دوردیف کنار هم قرار دارند . سمت چپ Block Ram و سمت راست یک بلوک ضرب کننده سخت افزاری می باشد .

با دابل کلیک بر روی بلوک Block Ram محتوای داخل آن که شامل یک RAM دو پورته می باشد مشاهده می گردد



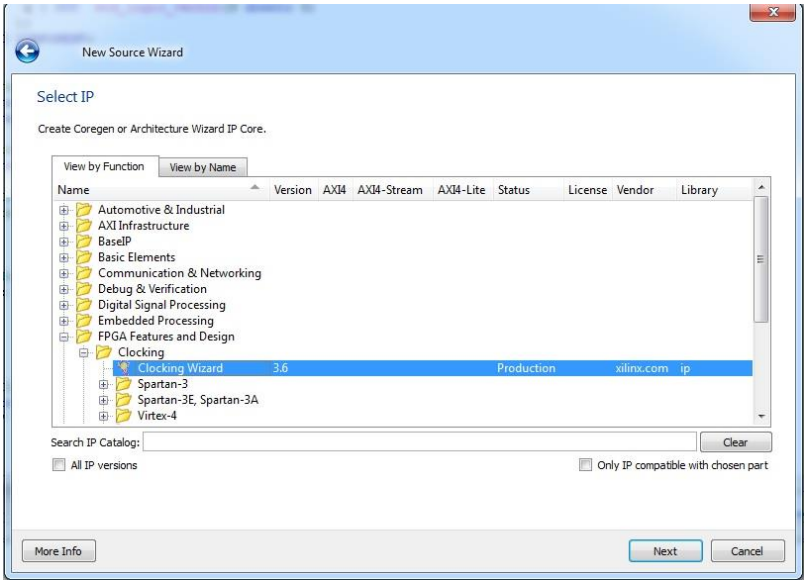
روش تنظیم فرکانس (Spartan6)



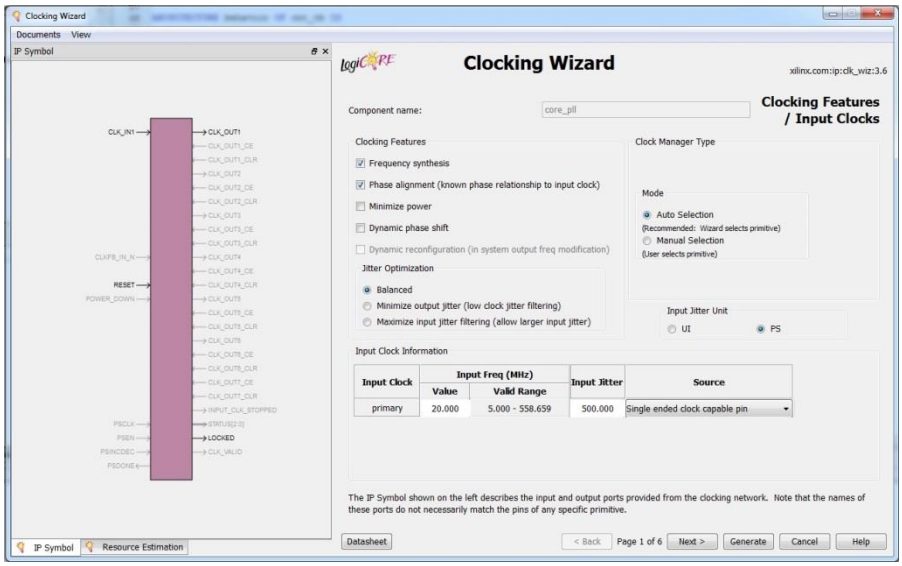


در این قسمت می خواهیم به چگونگی تنظیم و راه اندازی PLL داخلی FPGA بپردازیم ذکر این نکته لازم است که بدانیم واحد PLL جهت ضرب یا تقسیم فرکانس ورودی در یک عدد می باشد.

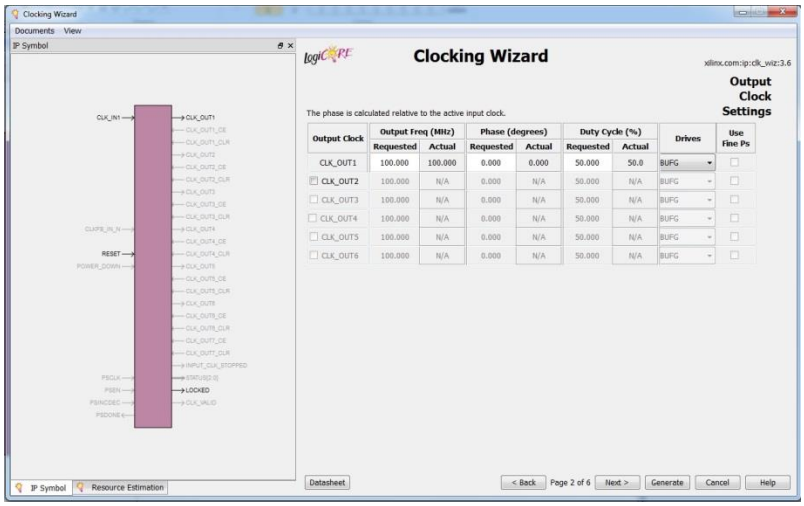
بدین منظور یک فایل Core ایجاد می کنیم . در لیست قسمت Clocking Wizard را انتخاب می کنیم :



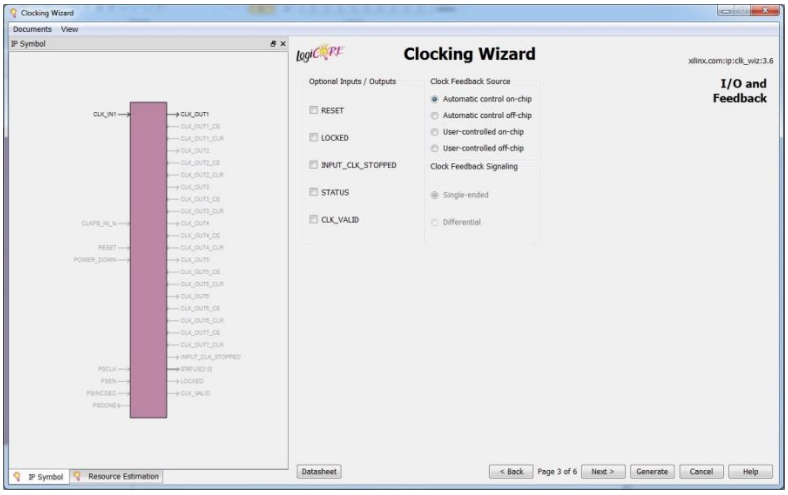
در پنجره زیر فرکانس ورودی که معمولا از روی کلاک اسیلاتور برد وارد FPGA می شود را وارد می کنیم .



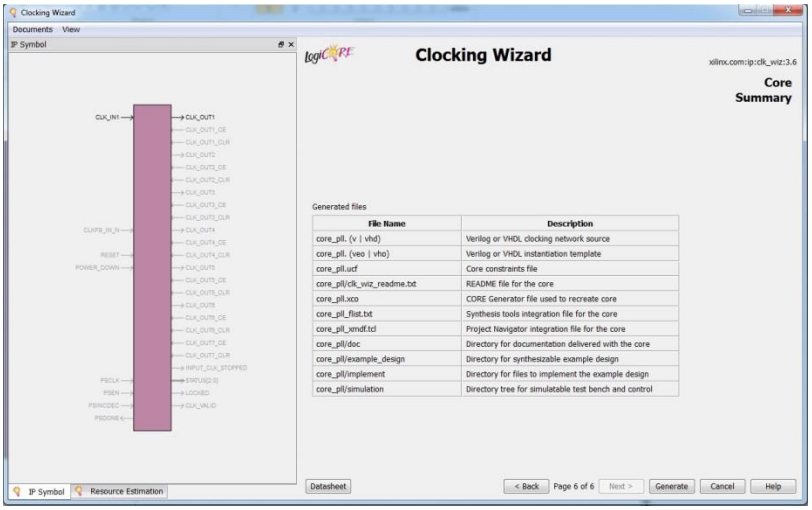
در پنجره زیر فرکانس خروجی (فرکانس مورد نیاز طراح) را وارد می کنیم . در این پنجره می توان کلاک های مختلف نیز تولید کرد .



برای راحتی می توان سیگنال های reset و locked را غیر فعال کرد .(معمولا از لحظه روشن شدن PLL تا آماده شدن سیگنال خروجی یک تاخیری بوجود می آید به محض آماده شدن سیگنال خروجی پایه LOCKED اعلام خواهد کرد)



در انتها خلاصه تنظیمات دیده می شود :



مقدمه ای بر FPGA و زبان توصیف سخت افزاری VHDL :

Field Programmable Gate Array ***FPGA***

انواع طراحی سخت افزار به سبک دیجیتال :

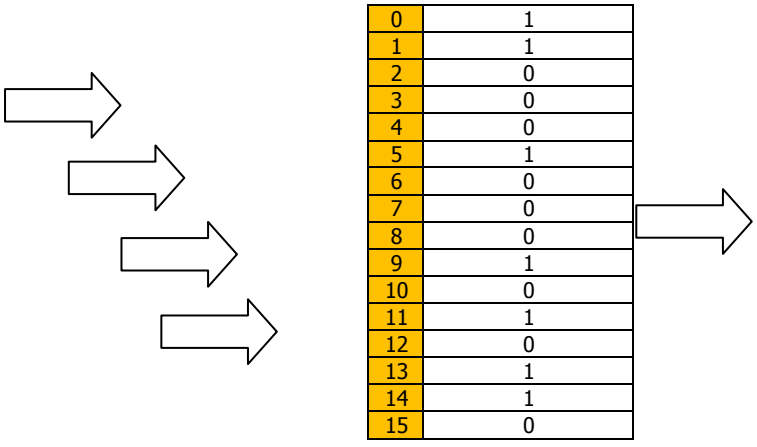
- ترانزیستور
- مدار و گیت
- بلوک و سیستم
- برنامه نویسی

مثال : فرض کنید می خواهیم تابع زیر را تحقق ببخشیم

$F(A,B,C,D)=\sum(0,1,5,9,11,13,14)$

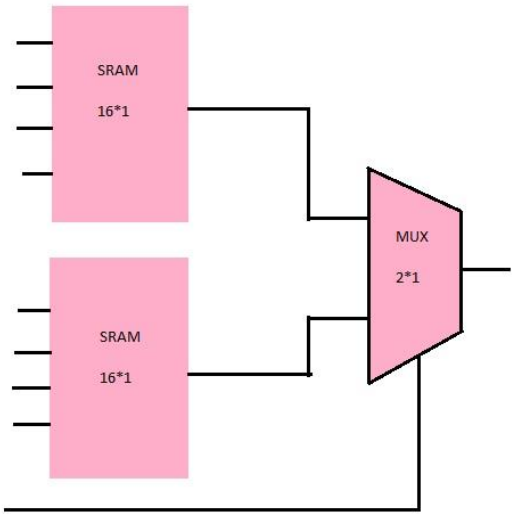
برای این منظور میتوان از جدول کارنو و سپس گیت های منطقی کمک گرفت و یا می توان به کمک مولتی پلکسر و یا دیکدر این طراحی را انجام داد . اما می خواهیم تابع فوق را به کمک چیپ RAM طراحی کنیم .

با توجه به تابع داده شده نیازمند یک SRAM با ۴ خط آدرس و یک خط دیتا هستیم بنابراین اندازه این حافظه 16×1 می شود .

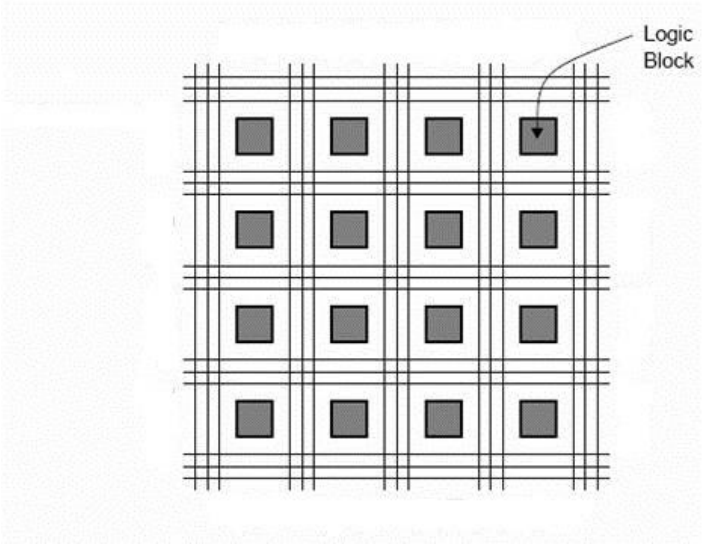


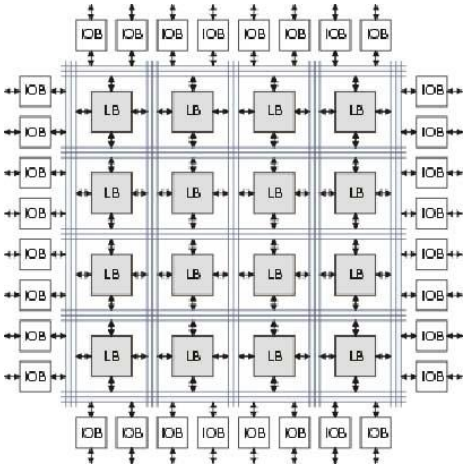
0	1
1	1
2	0
3	0
4	0
5	1
6	0
7	0
8	0
9	1
10	0
11	1
12	0
13	1
14	1
15	0

اگر متغیرهای ورودی کمتر از ۴ بیت باشند می توان پایه های آدرس با ارزش بالاتر را زمین کرد اما اگر متغیرهای ورودی بیشتر از ۴ بیت باشند از شکل زیر استفاده می شود :



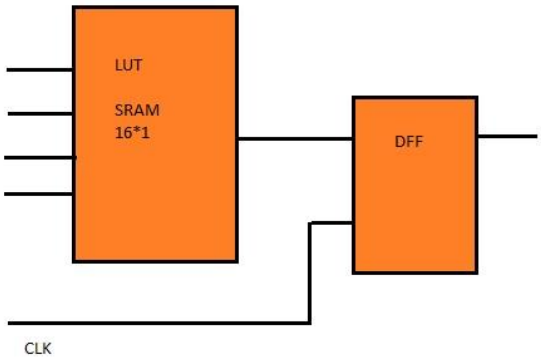
در تراشه های FPGA به هر کدام از این SRAM های 16×1 یک lookup table یا به اختصار LUT گفته می شود . شکل های زیر نحوه قرار گیری SRAM ها را در تراشه FPGA نشان می دهد .



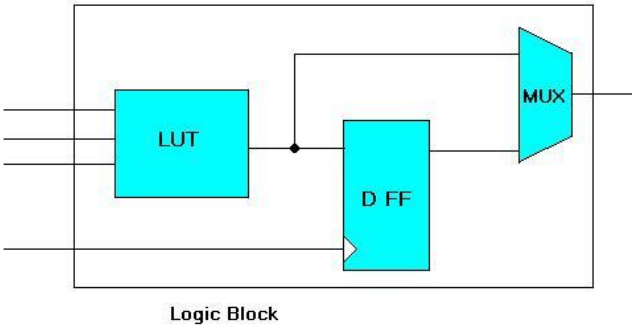


در شکل فوق SRAM ها بخشی از logic block ها هستند فقط تنها ایراد این نوع طراحی آن است که با قطع برق اطلاعات SRAM ها پاک خواهد شد بدین منظور در کنار هر تراشه FPGA یک تراشه ROM نیز قرار می دهند تا به محض وصل شدن برق اطلاعات از ROM وارد بلوک های SRAM شود .

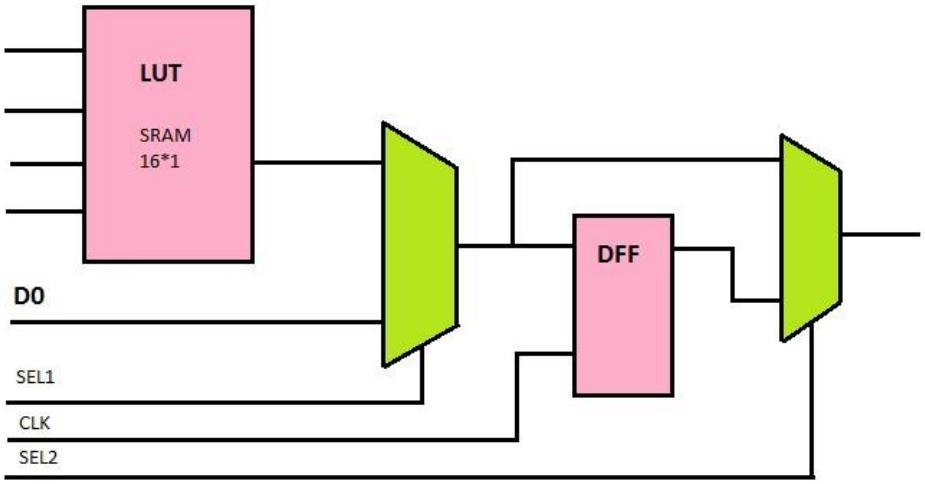
همانطور که در بلوک های SRAM دیده می شود این بلوک ها فقط قادر به پیاده سازی توابع منطقی ترکیبی هستند . برای پیاده سازی توابع منطقی ترتیبی از بلوک منطقی زیر استفاده می شود :



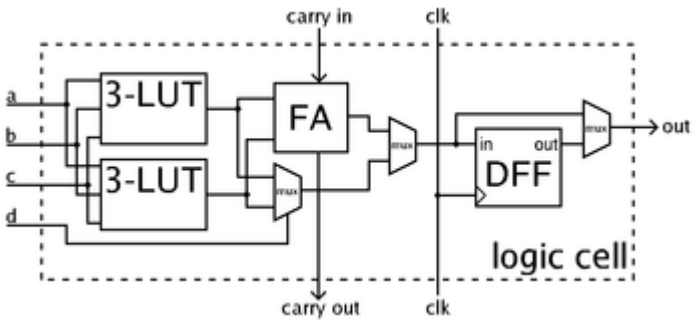
البته که بلوک های انعطاف پذیر بایستی تواماً دارای پیاده سازی ترکیبی و ترتیبی بصورت زیر باشند :



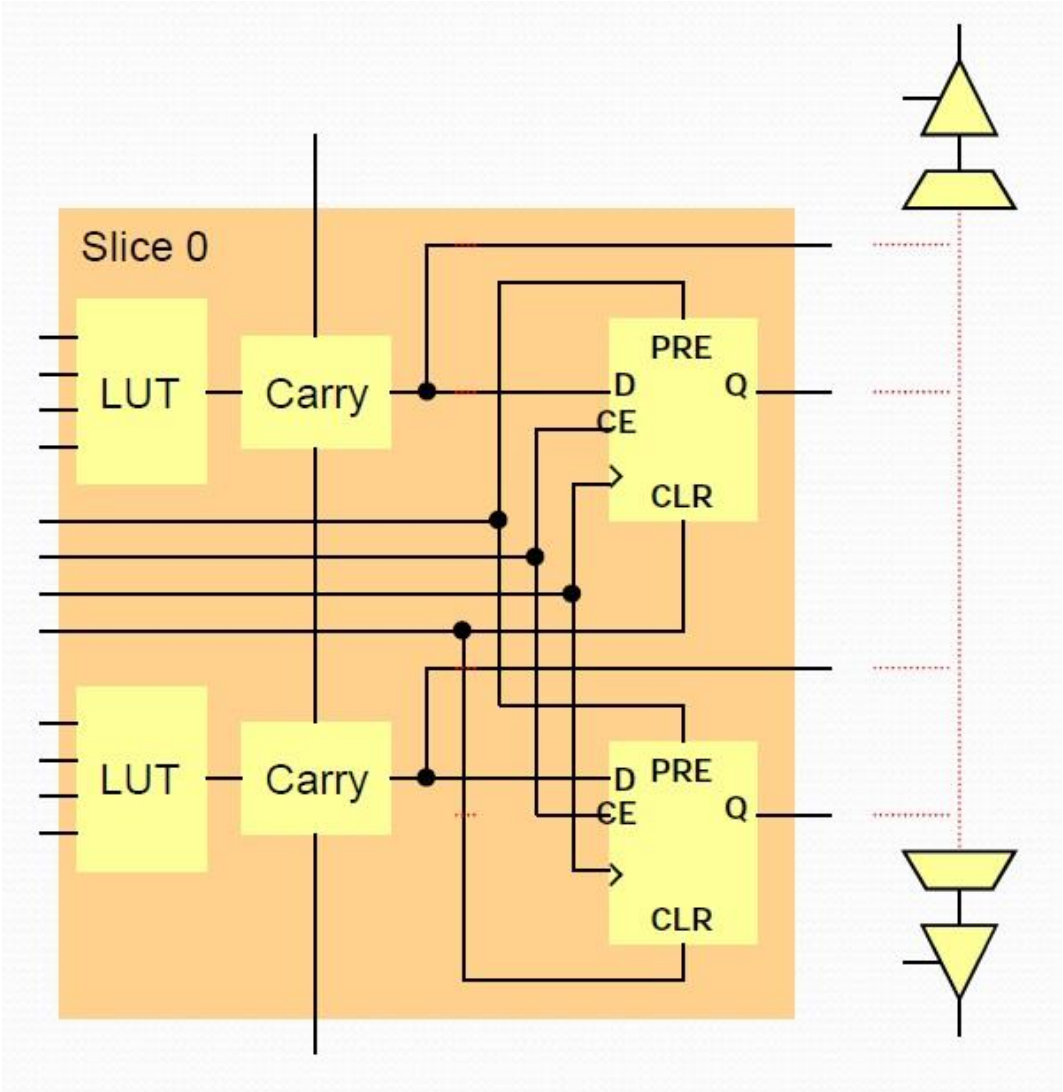
در شکل زیر یک بلوک کاملتری از بلوک های منطقی دیده می شود که به سلول منطقی Logic Cell یا به اختصار LC معروف هستند .



از آنجایی که عملیات های جمع و ضرب بسیار پرکاربرد هستند در داخل LC ها این بلوک ها برای انعطاف بیشتر قرار داده شده است که یک CLB و یا Configurable Logic Block را تشکیل خواهد داد :

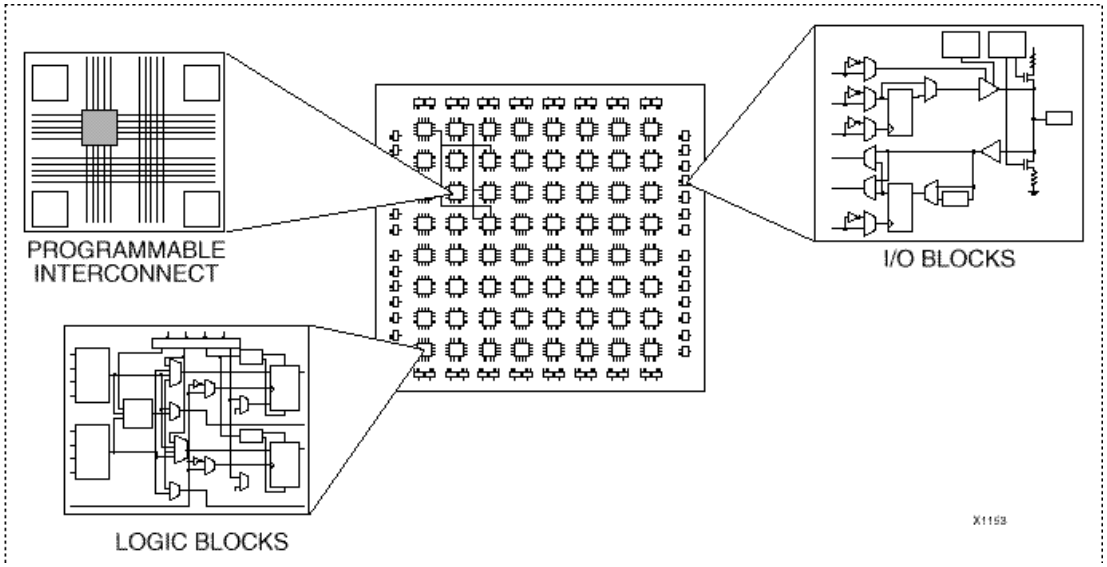


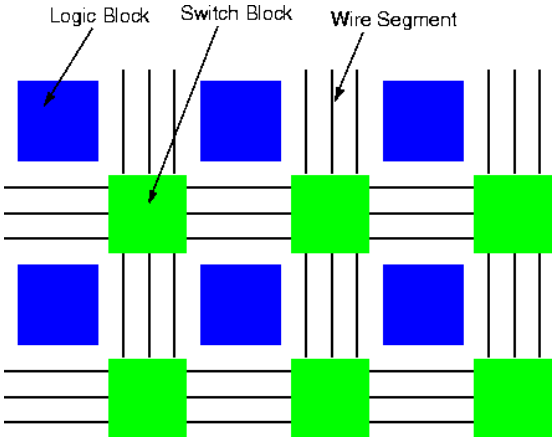
Slice : هر ۲ عدد LC یک SLICE را تشکیل می دهند

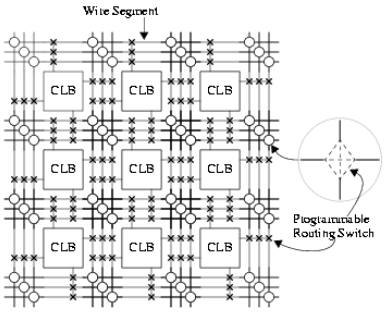


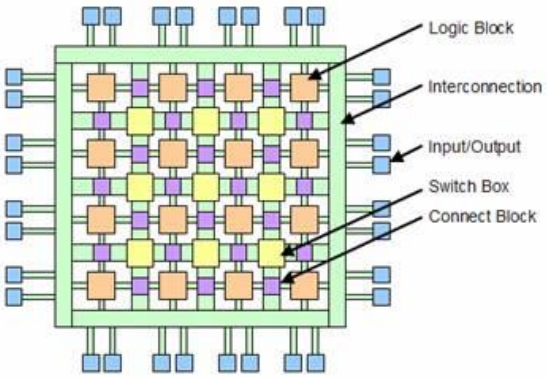
معمولا یک FPGA بطور کلی از سه قسمت اصلی تشکیل شده است :

- LC : هر دو عدد Slice یک CLB و یا Configurable Logic Block را تشکیل خواهد داد
- SM
- IOB



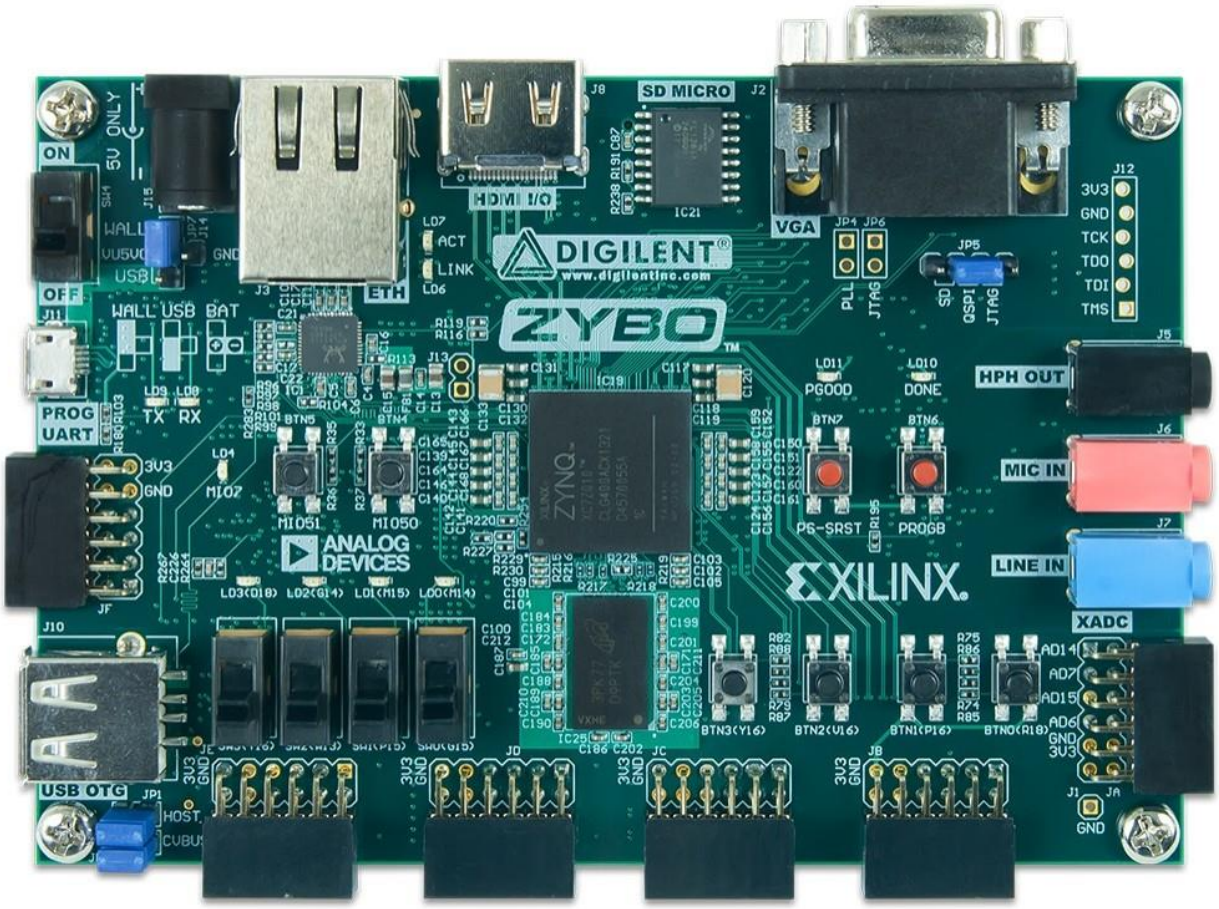






Switch matrix ها برای اتصال بین CLB ها استفاده می شوند .

Configuration در واقع همان عملی است که تمامی mux ها LUT ها و SM ها و تمامی قطعات FPGA مناسب طرح برنامه ریزی می شوند و نهایتاً نشان می دهند که FPGA قرار است چه عملی را انجام دهد .



محاسن استفاده از FPGA

FPGA ها در پیاده سازی توابع نسبتا پیچیده و پیچیده دیجیتال به کار می روند که نیاز به سرعت پردازش بالایی دارند. علاوه بر این کاهش سخت افزار مورد نیاز و همچنین برنامه نویسی ساده و استاندارد نیز از دیگر مزیت های استفاده از FPGA است. آنچه که قابلیت و توانایی FPGA ها را بالا برده است توانایی‌هایی است که پاره‌ای از آنها در زیر آمده است:

- کاربرد اصلی FPGA در ایجاد هسته های پردازشی می باشد .

- مدارهای دیجیتال پیچیده به آسانی در آنها پیاده سازی می شوند.
- تست مدار سریع است.
- برای تولیدات با تیراژ پایین ارزان تمام می شود.
- متناسب با نیاز، تغییرات لازم را در طراحی می توان انجام داد و مجدا FPGA را با ساختار جدید برنامه ریزی نمود.
- قابل برنامه ریزی توسط کاربر است .
- می توان چند تا هسته پردازشی داخل یک FPGA تعریف کرد تا در یک زمان واحد چندتا کار را باهم انجام بدهد مثلا شما می توانید با یک FPGA معمولی حدود ۲۰۰ تا هسته atmega32 تعریف کنید و ۲۰۰ تا کار را همزمان انجام بدهید .
- میکروکنترلرها و DSP ها و میکروپروسسورها به صورت سریال دستورالعمل اجرا می‌کنند و قابلیت پردازش بصورت موازی در آنها وجود ندارد، اما در FPGA قابلیت پردازش موازی وجود دارد و قابلیت انجام عملیات بصورت همزمان را دارد.
- کاربردهای FPGA خیلی تخصصی می‌باشد ودر اکثر موارد به عنوان پردازشگر در مدارات پردازشی استفاده می‌شود. سرعت بالایFPGA ها آنها را مساعد کارهای پردازشی سنگین مثل پردازش تصویر و پردازش صدا می کند و سرعت این پردازش نسبت به سیستم‌های دیگر خیلی بالاتراست.
- FPGA ها در پیاده سازی توابع پیچیده‌ی دیجیتالی به کار می‌روند که نیاز به سرعت پردازش بالایی دارد.
- کاهش سخت‌افزار مورد نیاز و همچنین برنامه‌نویسی ساده و استاندارد نیز از دیگر مزیت‌های استفاده از FPGA است.
- سرعت اجرای توابع منطقی در FPGA ها بسیار بالا و در حد نانو ثانیه است.
- امکان تعریف هر یک از پایه های IC به صورت ورودی یاخروجی یا هر دو
- امکان تعریف وضعیت عملکرد هر پایه در هنگام استفاده یا عدم استفاده.به عنوان مثال عملکرد HIGH امپدانس(Z)در هنگام عدم استفاده و یا قرار گرفتن در یک وضعیت منطقی صفر یا یک در هنگام عدم استفاده.
- امکان تشخیص تغیبیرات سطوح یا لبه های پایین رونده یا بالا رونده منطقی اعمال شده به هر پایه.
- امکان تغییر متناوب معمار ی داخلی با استفاده از سری های Bootable که نقشه معماری آنها در یک حافظه خارجی نگهداری شده و با تغییر آدرس برنامه ریزی می توان IC را بامعماری جدید Boot کرده و از آن استفاده کرد.
- امکان برنامه ریزی در مدار(ISP) که این قابلیت را به وجود می آورد تا بدون اعمال تغییراتی که سخت‌افزاری هستند و تنها از طریق پورت برنامه ریزیJTAG ، معماری داخلی IC را تغییر داد.
- کاهش حیرت انگیز حجم مدار و مجتمع سازی در ابعادی تنها به مساحت چند سانتی متر مربع.

- کاهش یکسان سازی عناصر طراحی و از میان بردن تمامی مشکلات ناشی از عدم تطابق استاندارد های مختلف.(LS,HC,s,AS,...)
- از میان بردن تمامی نویز های ناشی از وجود قطعات مختلف و مجزا در مدار.
- کاهش چشمگیر توان مصرفی و اتلاف توان.
- افزایش سرعت پردازش و خطاهای انتشار به دلیل استفاده از فناوری پیشرفته و دستیابی به خطاهای انتشار تا ۴ nsو فرکانس کلاک فرارتر از ۱۷۸ مگاهرتز.
- کار با دو سطح ولتاژ ۵ Vو ۳,۳ Vجهت استفاده از آنها در دستگاه های قابل حمل مانند گوشی‌های موبایل
- ضریب ایمنی صد در صد به دلیل عدم امکان دستیابی به محتوای داخلی و عدم توان توصیف محتوای داخلی به دلیل انجام ساده‌سازی و فشرده‌سازی بسیار پیچیده.

و بسیاری از قابلیتهای حیرت انگیز دیگر که امکان انجام یک طراحی مجتمع ،کم حجم ،بهینه و سریع را فراهم می آورد.

[بر گرفته از سایت شرکت رهپویان علم و صنعت آوا]

سازندگان FPGA

شامل شرکت‌های بزرگی همچون Xilinx و Altera و Actel می‌باشند که سری‌های مختلفی، از جمله Xilinx شامل Spartan و Virtex و Altra شامل Cyclone و Flex10k و Stratix و .... را به بازار عرضه کرده‌اند. در کشور ما معمولا با CPLD ها و FPGA های شرکت‌های مذکور کار می‌شود که از نظر محبوبیت به صورت زیر هستند :

XILINX
Altera
Actel

کمپانی Xilinx بیش از نیمی از محصولات FPGA در دنیا را تولید می کند و با شرکتهایی همچون IBM مشارکت دارد . مجموعه نرم افزارهای ISE و Vivado تولید این کمپانی هستند .

نگاهی کوتاه به زبان توصیف سخت افزاری VHDL :

- نحوه فراخوانی و بکارگیری یک کتابخانه:

```
LIBRARY نام کتابخانه;
```

- نحوه فراخوانی و بکارگیری یک package از داخل کتابخانه اش:

```
USE نام کتابخانه.نام پکیج کتابخانه.قسمت مورد نظر ;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

کتابخانه های رایج :

IEEE
STD_LOGIC_1164

STD_LOGIC_SIGNED

STD_LOGIC_UNSIGNED

STD_LOGIC_ARITH

NUMERIC_STD

STD
STAN DARD

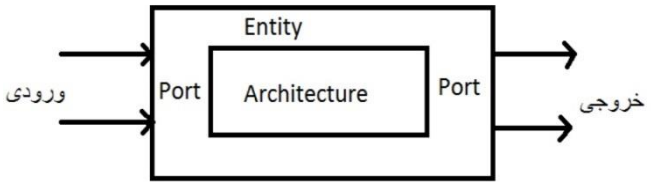
UNISIM
VComponents

- تعریف موجودیت :

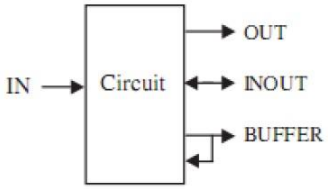
```
Entity نام مدار is

  Port ( معرفی ورودی و خروجی ها ) ;

End نام مدار ;
```

- تعریف معماری طرح :

```
Architecture نام معماری of نام مدار is

Begin

  کد VHDL

End نام معماری ;
```



فرق inout و buffer :



تعریف سیگنال : پایه یک ماژول یا قطعه و یا یک سیم

Signal نام و نوع اتصالات ;

روش نمایش اعداد :

```
56  ,  18  , 4
"10010"
X"5e"
O"47"
```

جدول زیر انواع داده را نشان می دهد :

Bit	'0', '1'
Bit_Vector	Array of bit
STD_LOGIC	'0', '1','X','W','Z','L','H','_'
STD_LOGIC_VECTOR	Array of STD_LOGIC
STD_ULOGIC	'0', '1','X','W','Z','L','H','U','_'
STD_ULOGIC_VECTOR	Array of STD_ULOGIC
SIGNED	Array of STD_LOGIC
UNSIGNED	Array of STD_LOGIC
Boolean	True and False
Character	7-bit ASCII
integer	-2,147,483,647 to +2,147,483,647
NATURAL	0 to +2,147,483,647
POSITIVE	1 to 2,147,483,647
real	Floating point, min:+1e38 to -1e38
string	Array of characters
time	Hr, min, sec, ms, ns, ps, fs

روش تخصیص مقدار اولیه :

روش اول :

```
Signal a:bit:='1';
```

روش دوم :

```
A<='1'
```

پروسس (VHDL ترتیبی) :

Process (لیست سیگنالهایی که با تغییر هر کدام از آنها عبارات داخل پروسس اجرا می شوند)

Begin

عبارات با فرایند اجرای ترتیبی

End process;

برخی عبارات مهم مورد استفاده در پروسس :

❖ **IF :**

نوع ساده :

```
If عبارت۱ then عبارت۲ ;
End if;
```

اگر عبارت۱ درست باشد عبارت۲ انجام می شود درغیراینصورت عملی انجام نشده و از این دستور خارج و به خط بعد از **end if** می رود.

نوع کلی :

```
If عبارت۱ then عبارت۲ ;
Else عبارت۳ ;
End if;
```

فرم کلی :

```
If شرط۱ then عبارت۱ ;
Elsif شرط۲ then عبارت۲ ;
.
.
Else عبارت ;
End if ;
```

❖ **Case :**

```
Case شرط is
    When مقدار شرط۱  => عبارت۱;
    When مقدار شرط۲  => عبارت۲;
    .
    .
    When others    =>  عبارت;
End case;
```

در دستورات فوق اگر شرط با مقدار هر شرطی برابر باشد عبارت روبروی آن اجرا می شود .

مثال های نمونه :

❖ نیم جمع کننده :

```
Entity ha is
        Port( a :in std_logic;
              B :in std_logic;
              S:out std_logic;
              C:out std_logic);
End ha;
Architecture half_adder of ha is
begin
                S<=a xor b;
                C<=a and b;
End half_adder;
```

❖ تمام جمع کننده :

➢ روش ۱

```
Entity Fa is
     port(a,b,Cin:in bit;
          s,Cout:out bit);
End Fa;
Architecture behavioral of Fa is
     Begin
         s<=a xor b xor Cin;
    cout<=(a and b)or(a and Cin)or(b and Cin);
End behavioral;
```

➢ روش ۲

```
entity fa_1bit is
    port (
        a,b,cin: in STD_LOGIC;
        sum,cout: out STD_LOGIC);
end fa_1bit;

architecture fa_1bit_arch of fa_1bit is
        component  ha_1bit
           port(x,y :in std_logic;
                s,c :out std_logic);
        end component;
        signal im1, im2, im3 :std_logic;
begin
        ha1: ha_1bit  port map (x=>a, y=>b, s=>im1, c=>im2);
        ha2: ha_1bit  port map (x=>im1, y=>cin, s=>sum, c=>im3);
        cout <= im2 or im3;
end fa_1bit_arch;
```

❖ جمع کننده ۴ بیت

```
Entity Fa4bit is
     port(a,b:in bit_vector(3 downto 0);
          Cin:in bit;
          Cout:out bit;
            S:out bit_vector(3 downto 0));
End Fa4bit;
Architecture Structural of Fa4bit is
component Fa is
     port(a,b,Cin:in bit;
           s,Cout:out bit);
End component;
Signal c:bit_vector (3 downto 0);
Begin
     fa0:Fa port map(a(0),b(0),Cin,S(0),c(1));
     fa1:Fa port map(a(1),b(1),c(1),S(1),c(2));
     fa2:Fa port map(a(2),b(2),c(2),S(2),c(3));
     fa3:Fa port map(a(3),b(3),c(3),S(3),Cout);
End Structural;
```

❖ دیکدر

➢

```
entity dec24 is
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
           m : out STD_LOGIC_VECTOR (3 downto 0));
end dec24;
architecture Behavioral of dec24 is
begin
m<="0001" when (x="00") else
   "0010" when (x="01") else
   "0100" when (x="10") else
   "1000";
end Behavioral;
```

❖ مولتی پلکسر

```
     entity mux41 is
       Port ( i : in STD_LOGIC_VECTOR (3 downto 0);
              s : in STD_LOGIC_VECTOR (1 downto 0);
            o_4 : out STD_LOGIC);
     end mux41;
     architecture Behavioral of mux41 is
     begin
     o_4<=i(0) when S="00" else
          i(1) when S="01" else
```

```
        i(2) when S="10" else
        i(3);
end Behavioral;
```

❖ فلیپ فلاپ D سنکرون

```
entity dff is
  Port ( d : in STD_LOGIC;
       clk : in STD_LOGIC;
       q : out STD_LOGIC);
end dff;
architecture Behavioral of dff is
begin
process(d,clk)
begin
if (clk'event and clk='1') then q<=d;
end if;
end process;
end Behavioral;
```

❖ رجیستر

```
entity reg4bits is
  Port ( din : in STD_LOGIC_VECTOR (3 downto 0);
       dout : out STD_LOGIC_VECTOR (3 downto 0);
        clk : in STD_LOGIC;
        rst : in STD_LOGIC;
       LOAD : in STD_LOGIC);
end reg4bits;
architecture Behavioral of reg4bits is
begin
process(clk,rst,LOAD)
begin
   if (rst = '1') then dout <= "0000"; -- or (others=>'0')
   elsif(clk'event and clk='1' and LOAD='1')then dout <= din;
        end if;
end process;
end Behavioral;
```

ساخت کلاک

```
clk<=not clk after 10 ns;
```
