



دانشکده مهندسی برق

مدارهای منطقی و سیستم‌های دیجیتال

آزمایش ۳ - آشنایی با Verilog
و تراشه Zynq و FPGA

آزمایش ۳ - آشنایی با زبان توصیف سخت افزار Verilog - شروع کار با مدار Vivado و Zynq، تراشه FPGA ترکیبی در PL زینک

هدف از این آزمایش آشنایی اولیه با Verilog، FPGA، نرم افزار Vivado و تراشه Zynq است. شما در این آزمایش ضمن یادگیری نحوه کار با زینک، مداراتی ترکیبی را با وریلاغ توصیف کرده و در بخش FPGA زینک تست می کنید.

قطعات و تجهیزات مورد نیاز

- بورد Zynq (AXPZ7010)

- یک عدد کابل USB و سیم پاور بورد

- نرم افزار Vivado 2019.1

۱- پیش‌مطالعه

برای آشنایی با مفاهیمی چون HDL، FPGA و تراشه Zynq به **بخش اطلاعات تکمیلی آزمایش سوم** در انتهای این فایل مراجعه کنید.

۲- پیش‌گزارش

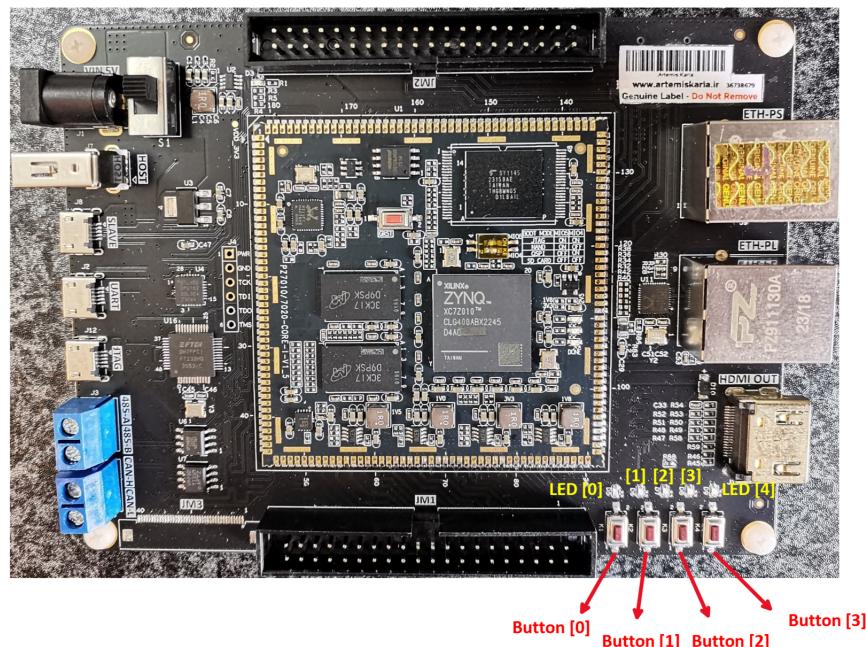
پیش از شروع، لازم است برای آمادگی قبل از آزمایش، حتماً ویدیوهای هفته اول تا سوم **این جدول برنامه** را مشاهده کرده باشید.

۳- دستور کار

۱-۳ آشنایی با بورد

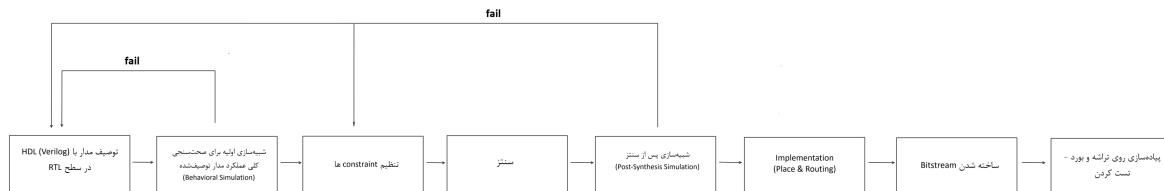
بوردی که در آزمایشگاه با آن کار می‌کنید شامل تراشه Zynq 7010 می‌یاشد. برای دیدن نتیجه و تست مدارات ساده ساخته شده و برقراری ارتباط با دنیای خارج، لازم است بتوانیم ورودی و خروجی‌های مدارها را توسط IO های بورد مدیریت و مشاهده کنیم؛ در واقع ورودی‌های مدار را به یک سری کلید متصل کرده و خروجی‌ها را به LED متصل می‌کنیم تا عملکرد مدار را پس از پیاده‌سازی روی تراشه تست کنیم. برای این منظور بورد FPGA مورد استفاده در آزمایشگاه (که دارای یک چیپ مدل Zynq 7010 ساخته شده شرکت Xilinx است). دارای چهار کلید ورودی به صورت ساده (push button) و پنج LED تک می‌باشد که همگی به پین‌های طرف (PL) تراشه متصل شده‌اند.

! توجه کنید که LED‌ها Active Low و کلیدها Active High هستند.



شکل ۳-۱ : بورد آزمایشگاه مدار منطقی (AXPZ7010)، دارای تراشه Zynq 7010

هر کدام از کلیدها و LED‌ها در PCB بورد به یکی از پین‌های FPGA متصل شده‌اند که در ادامه به طور دقیق به این موضوع اشاره خواهیم کرد.



شکل ۲-۳: مراحل FPGA design Flow

اولین مرحله در کار با FPGA‌ها توصیف مدار مورد نظر با زبان وریل‌اگ است.

در کلیه بخش‌های این آزمایش لازم است فقط از گیت‌های درونی و عبارت‌های Continuous Assignment استفاده گردد.

نام مازول و ورودی خروجی‌های مازول شما می‌تواند هر اسم دلخواهی باشد، اما در این آزمایش برای سهولت اتصال ورودی خروجی‌های مازول به کلیدها و LED‌های بورد (که در ادامه توضیح خواهیم داد)، از فرمت کلی زیر استفاده کنید (این فرمت برای Top Module هر قسمت است که بناسن به IO‌های بورد وصل شود).

```

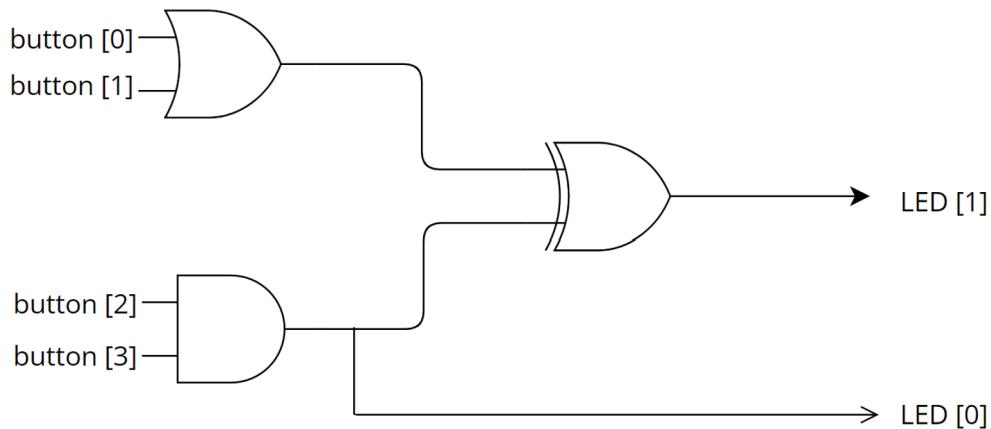
1 `timescale 1ns / 1ps
2 module top_module(
3     input wire [3:0] button,
4     output wire [4:0] LED);
5 // Your Code Here
6 endmodule

```

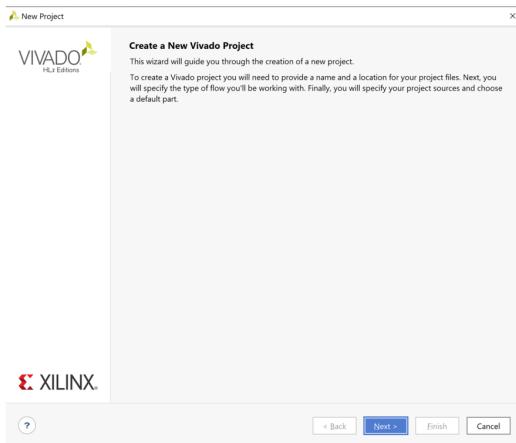
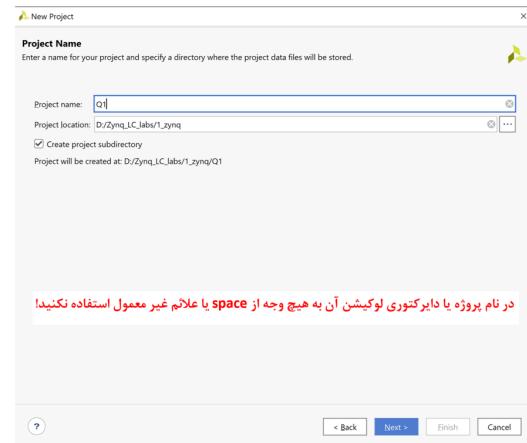
این آزمایش دارای پنج بخش کلی است؛ در بخش اول آزمایش توضیح انجام مراحل با جزئیات کامل داده شد و هدف این است که شما برای اولین تجربه کار با Vivado مشکل نداشته باشید و منبعی برای مراجعه دوباره شما به جهت انجام مراحل وجود داشته باشد. پس از آن در بخش‌ها و آزمایش‌های بعدی، صرفاً تسک مربوط به هر بخش توضیح داده شده و برای انجام مراحل به صورت پیش‌فرض به توضیحات بخش اول دستور کار این آزمایش مراجعه کنید.

۲-۳ - بخش اول (زمان تقریبی: ۳۰ دقیقه)

مدار زیر را به زبان وریل‌اگ و فقط با استفاده از عبارات Continuous Assignment توصیف کنید.

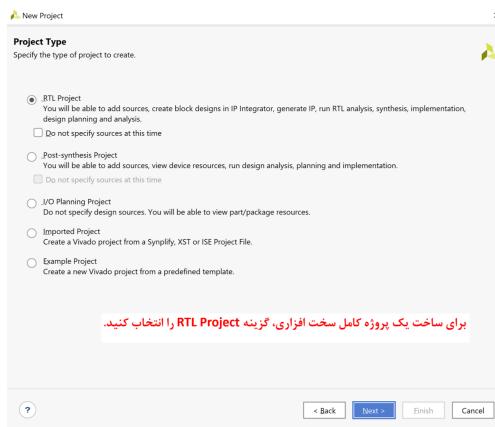


توصیه می‌شود از **VSCCode** به عنوان ادیتور برای نوشتگی کد وریل‌اگ استفاده کنید. سپس وارد نرم‌افزار Vivado شده و مرحله زیر را طی کنید.

1.

2.




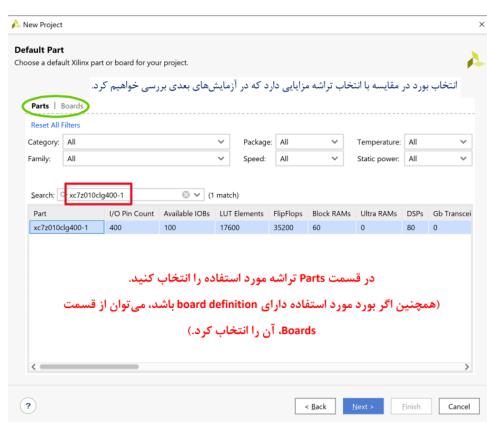
3.



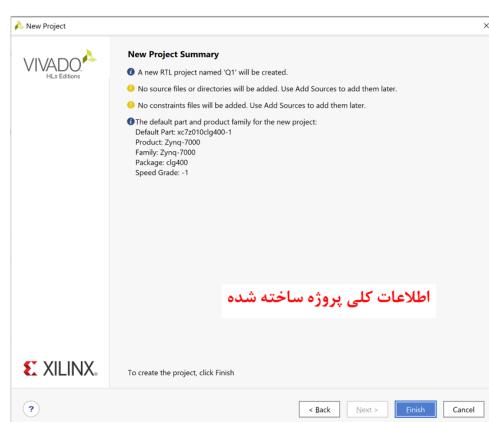
4.



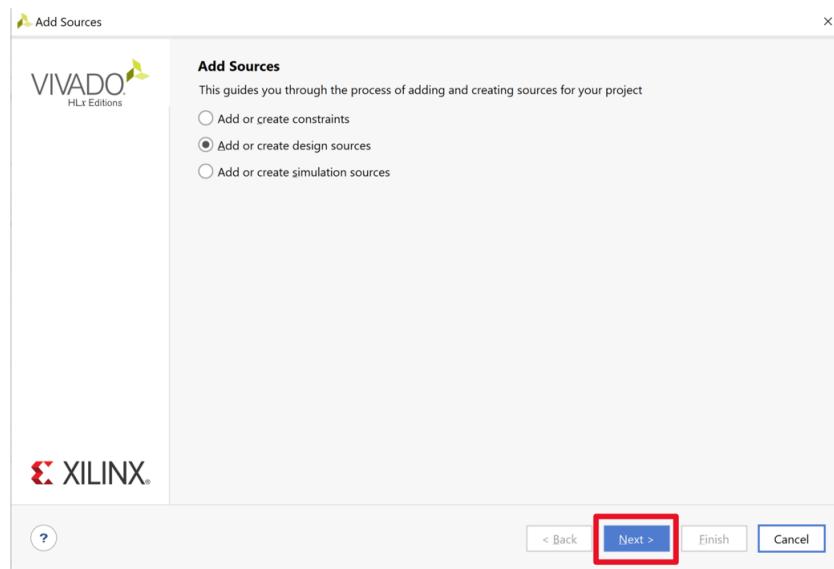
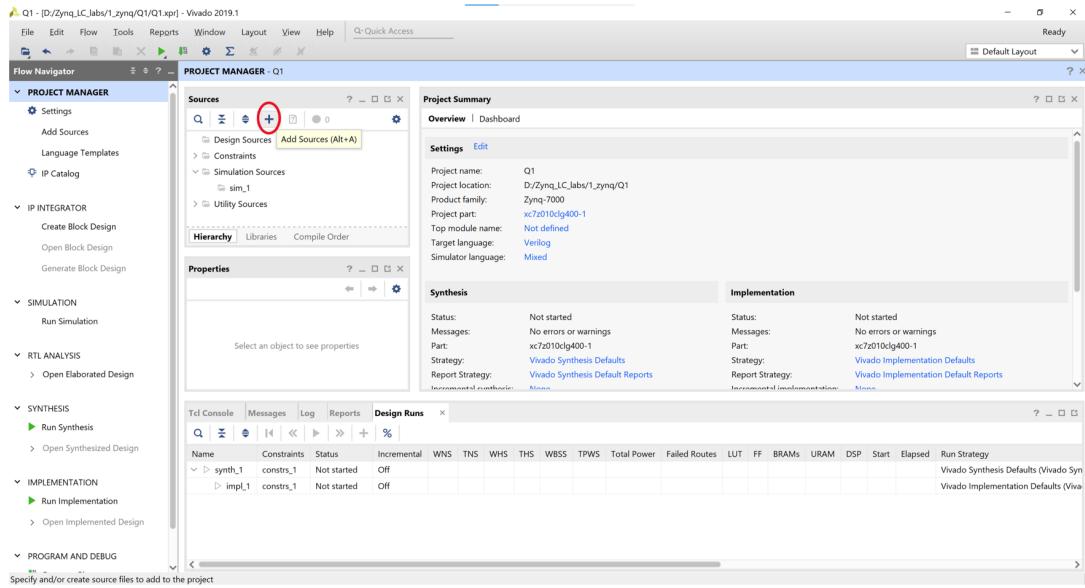
5.

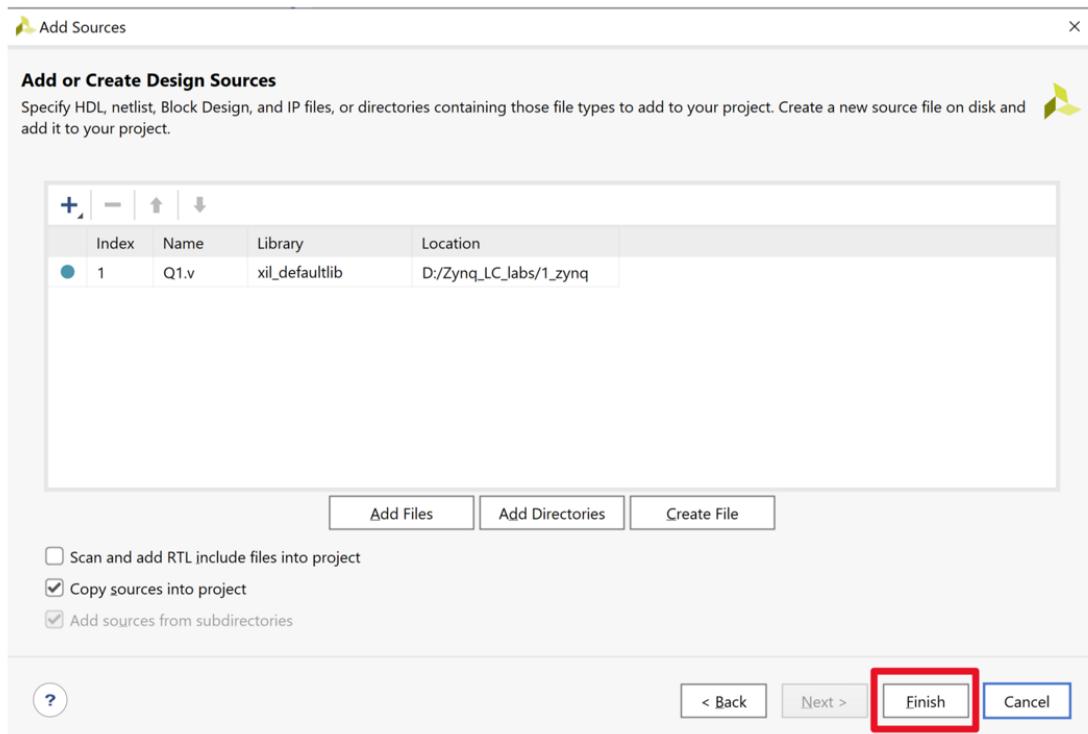
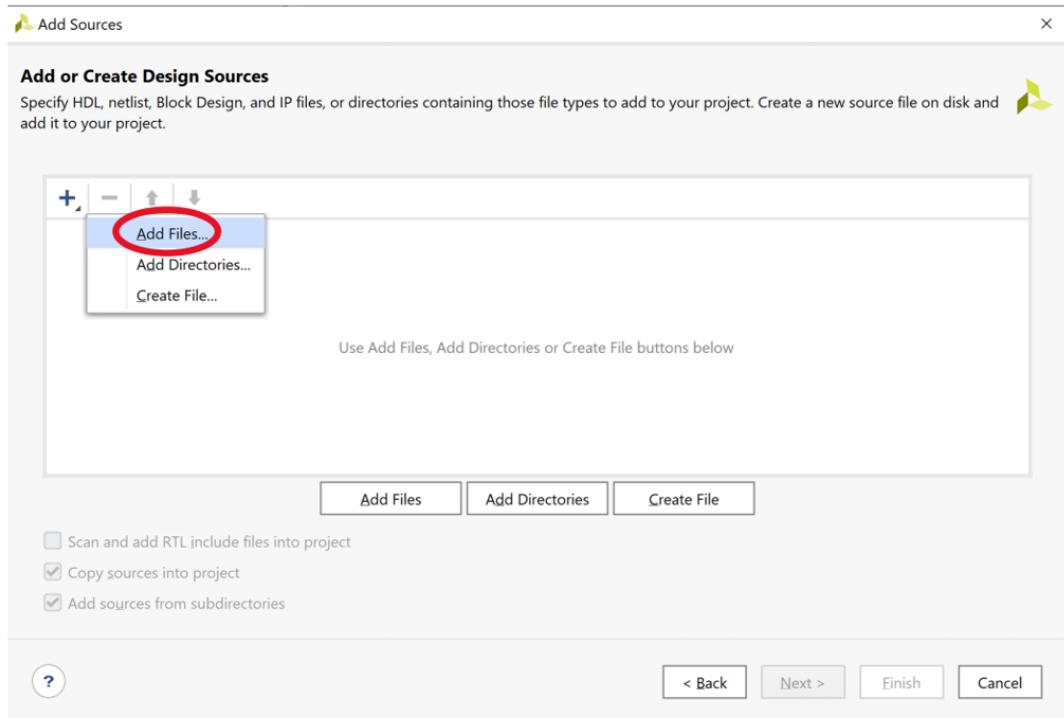


6.



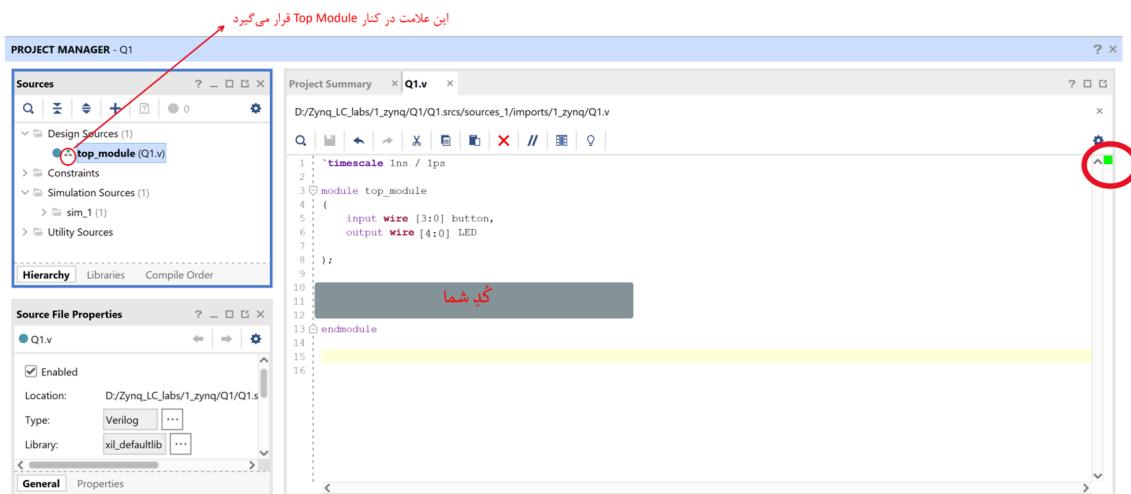
حال مانند تصاویر زیر فایل وریلاغ نوشته شده را به عنوان Source به پروژه اضافه کنید.







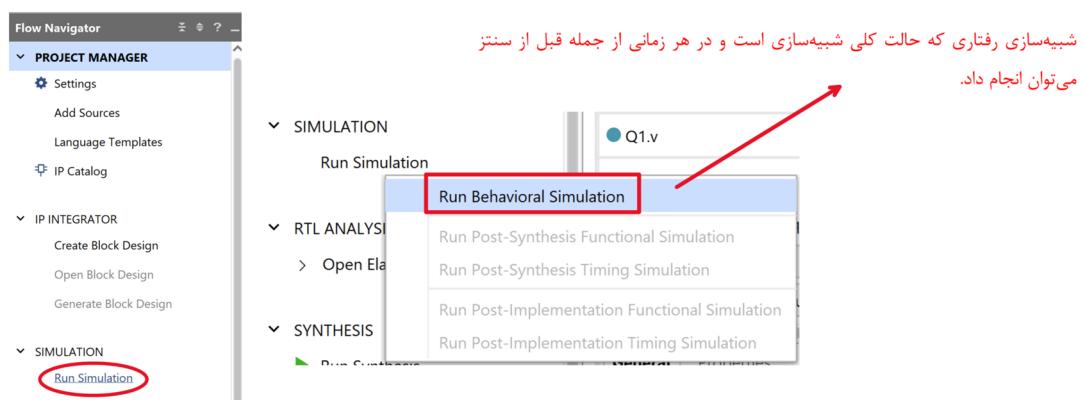
ادیتور Vivado به صورت آنلاین ایرادات سیستکسی کد شما را بررسی می‌کند و علامت سبز رنگ نشان‌دهنده صحت کد از این جهت می‌باشد:



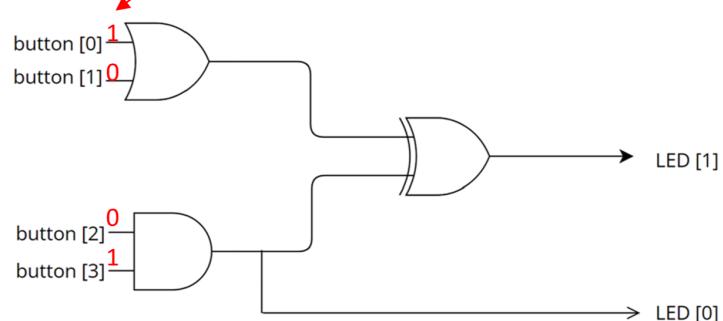
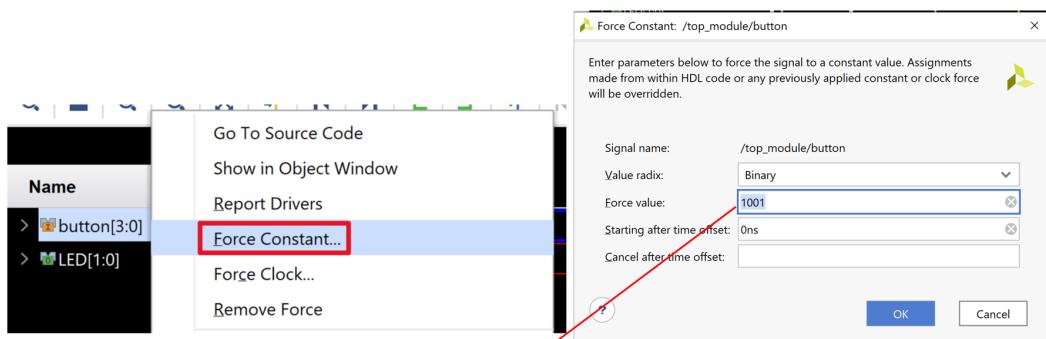
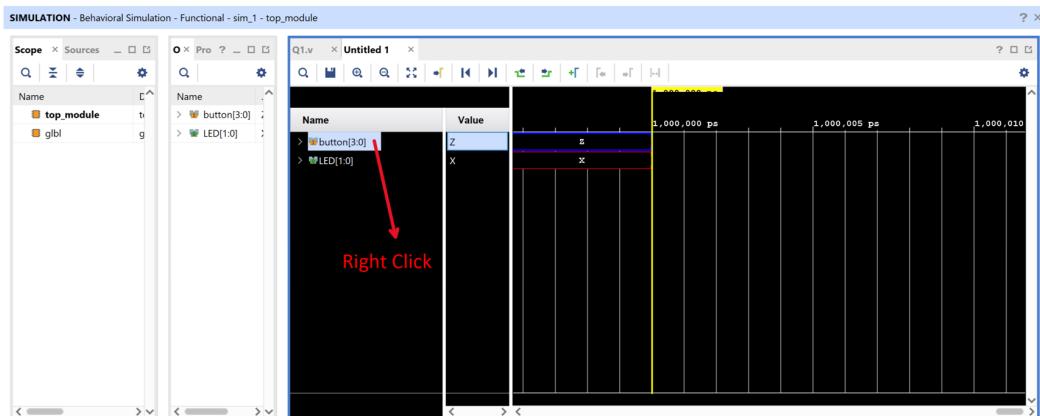
ابتدا کد خود را شبیه‌سازی کنید تا از صحت عملکرد مدار حاصل از آن مطمئن شوید.

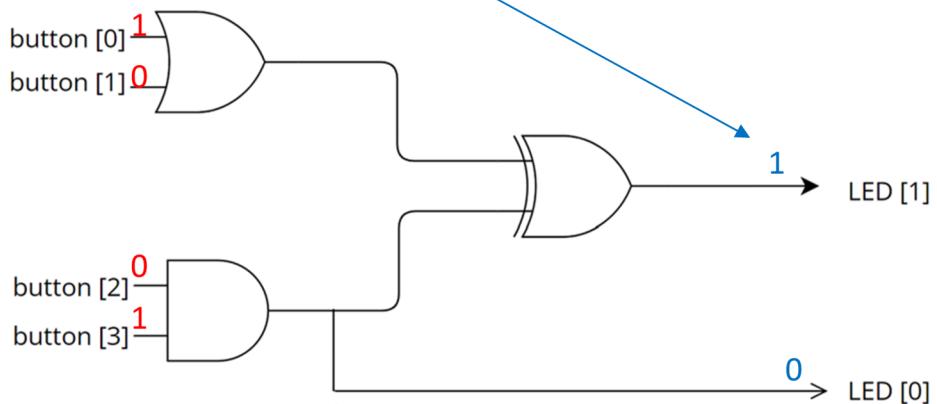
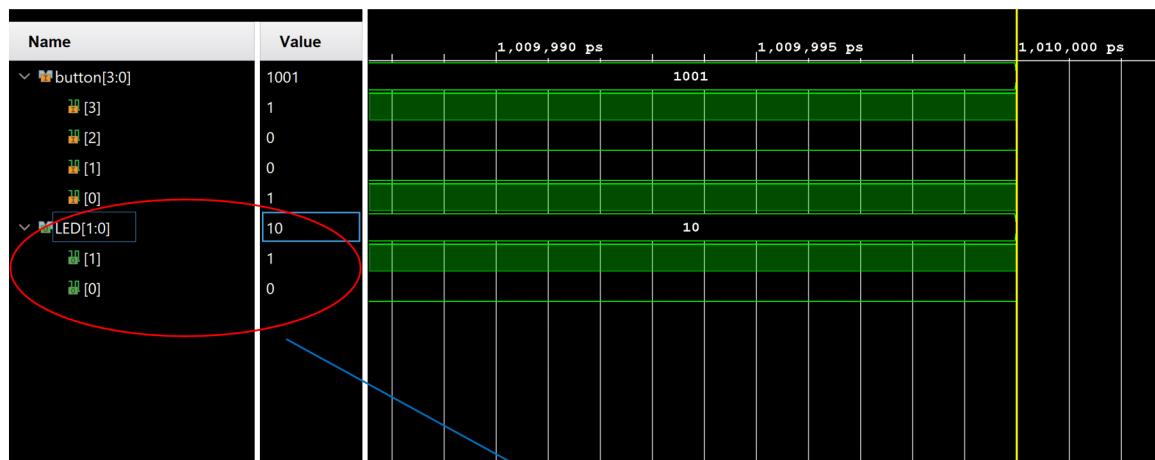
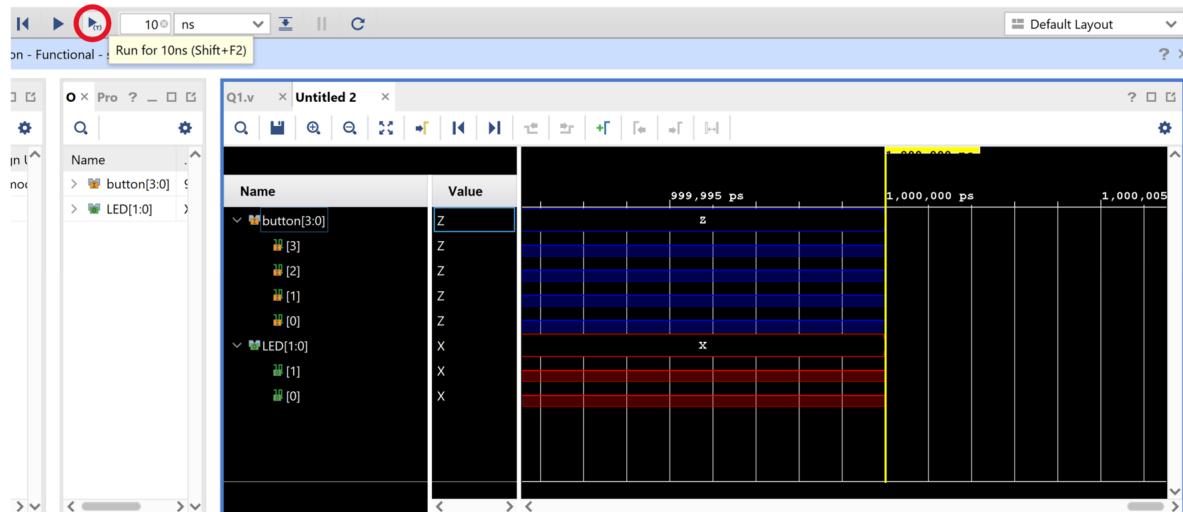
۲-۲-۳ - مراحل شبیه‌سازی در Vivado

(این مراحل برای شبیه‌سازی بدون testbench است که در شبیه‌سازی‌های محدود و مدارات ساده کاربرد دارد)



نحوه دادن مقدار به ورودی‌ها در شبیه‌سازی:



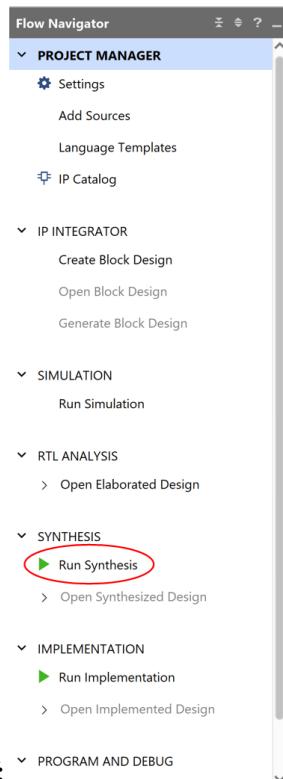


برای اطمینان از صحیح عملکردی کد، چند مقدار دیگر را به صورت دستی امتحان کنید. سپس، مراحل پیاده‌سازی آن روی بورد را مانند زیر پیش ببرید.

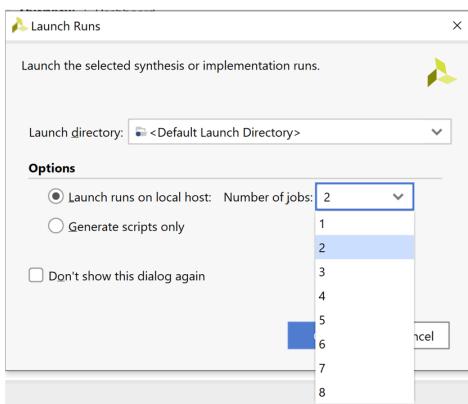
۳-۲-۳ - مراحل پیاده‌سازی روی بورد

الف) سنتز: در مرحله اول باید کد وریل‌اگ نوشته شده توسط شما به مداری شامل گیت‌ها و عناصر منطقی (LUT‌ها و ...) تبدیل شود. هم‌چنین فرآیندهای بهینه‌سازی Logic به صورت خودکار انجام می‌گیرد. به این فرآیند سنتز مدار گفته می‌شود که در این مثال توسط Synthesis Tool نرم‌افزار Vivado انجام می‌شود. در این مرحله اهمیتی ندارد که از چه چیزی استفاده می‌کنید یا ورودی خروجی‌های مدار شما به چه پین‌هایی از تراشه متصل خواهد شد، چرا که صرفاً قرار است شماتیکی از مدار شما طراحی شود؛ پیاده‌سازی آن روی تراشه برای مراحل بعدی است.

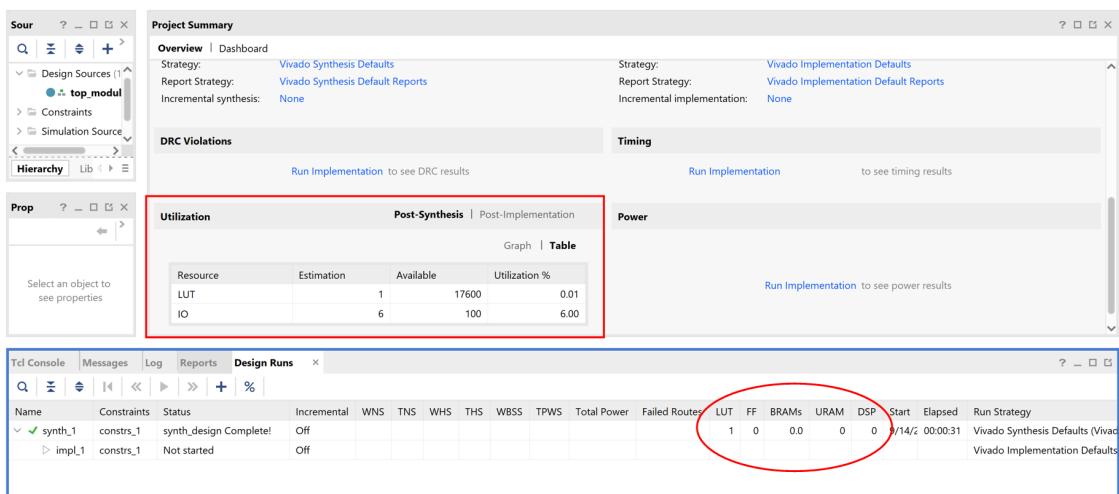
برای سنتز شدن کد در Vivado بر روی گزینه زیر در بخش SYNTHESIS از پنجره Flow Navigator کلیک کنید.



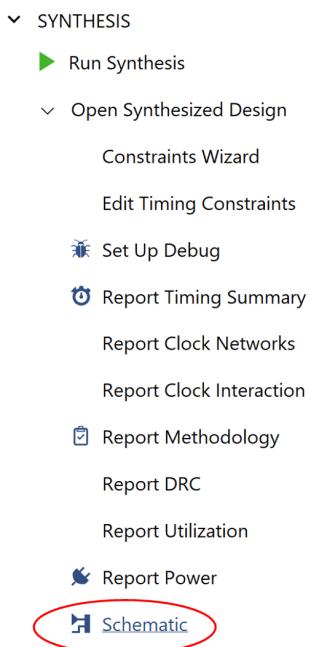
سپس پنجره‌ای باز شده و از شما می‌خواهد مشخص کنید فرآیند سنتز بر روی چند هسته از پردازنده کامپیوتر شما اجرا شود. این انتخاب از این جهت مهم است که در پروژه‌های بزرگتر که شامل چندین IP Core (در ساده‌ترین حالت یک ماژول از پیش‌آماده وریلاغ تصورش کنید) و ماژول سخت‌افزاری هستند، Vivado می‌تواند برای سنتز هر کدام از آن‌ها سناریوی جداگانه‌ای در نظر گرفته و آن‌ها را به صورت موازی روی هسته‌های مختلف کامپیوتر انجام دهد تا سرعت سنتز بالاتر برود. در این بخش چون تنها یک ماژول وریلاغ داریم، تفاوتی ندارد این تنظیم را روی چند هسته قرار دهیم (توجه کنید که هیچ‌گاه از تمام هسته‌های پردازنده‌تان برای سنتز استفاده نکنید؛ چرا که سرعت کلی کار کامپیوتر در تعامل با سایر برنامه‌هایی که باز هستند، کم می‌شود).



پس از سنتز، میزان منابع مصرفی استفاده شده توسط Synthesis tool برای ساخت مدار را می‌توانید از پنجره Project Summary یا تب Design Runs مشاهده کنید. در خصوص تعداد منابع مصرف شده بحث کنید و آن را در گزارش آزمایش قید کنید.



حال با کلیک بر روی Schematic در بخش SYNTHESIS، شماتیک مدار خود را پس از سنتز مشاهده کرده و با شکل مدار رسم شده مقایسه کنید (به این تفاوت و نوع عناصر استفاده شده در مدار سنتز شده توسط Synthesis Tool دقต کنید و در گزارش بنویسید.)



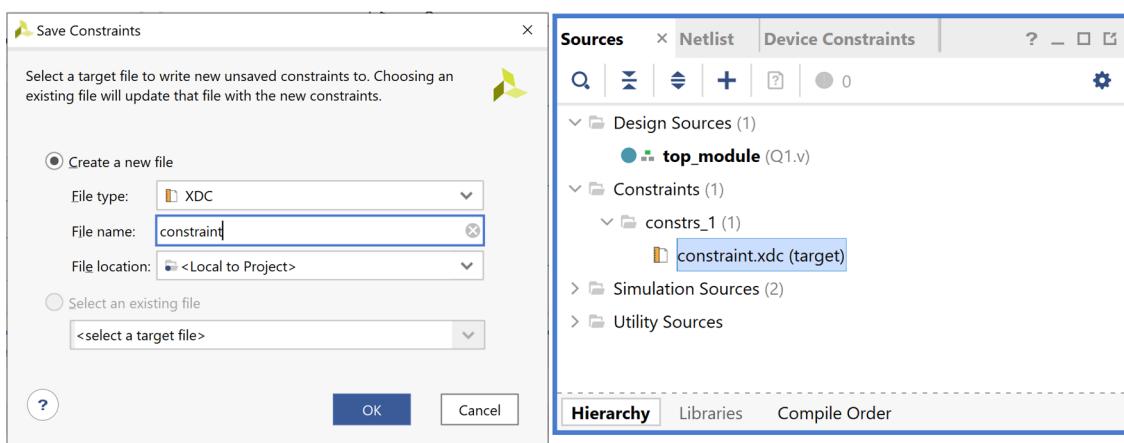
این پورت‌هایی (ورودی خروجی‌هایی) که تا به حال در کد وریلاگ با آن‌ها کار کردید، پورت‌های انتزاعی هستند و برای پیاده‌سازی مدار بر روی FPGA باید آن‌ها را به کلیدها، LED‌ها و به طور کلی Interface‌های روی بورد متصل کنید. لذا هر پورت مدار خود را به پینی از تراشه متصل می‌کنید که از نظر سخت‌افزاری روی PCB بورد به یکی از این Interface‌ها متصل است.

در جدول زیر مشخص شده که هر کدام از این کلیدها و LED‌ها در PCB بورد به کدامیک از پین‌های FPGA وصل شده‌اند. از این جدول شما باید برای تنظیم Constraint در Vivado استفاده کنید؛ یعنی مشخص کنید که هر ورودی و خروجی ماثول وریلاگ (مدار) شما به چه پینی از FPGA متصل شود.

Name	Package Pin	Description - I/O Std
Button[0]	T10	
Button[1]	T11	BANK34
Button[2]	T15	FIX 3.3V
Button[3]	T14	
LED[0]	V20	
LED[1]	W19	
LED[2]	W18	LVCMOS33
LED[3]	V18	
LED[4]	V17	

جدول : 1-3 اطلاعات پین های متصل به کلید ها و LED های بورد

حال در بخش I/O Ports ورودی خروجی های مدار سنتز شده را مطابق جدول به پین های مناسبی از تراشه (که از نظر ساخت افزاری و PCB بورد، به کلید ها و LED های بورد متصل آند) Assign کنید. سپس با ذخیره کردن آن، فایلی با پسوند و فرمت XDC تولید می شود که اصطلاحاً به آن Constraint File می گوییم.



Constraint File (constraint.xdc) content:

```

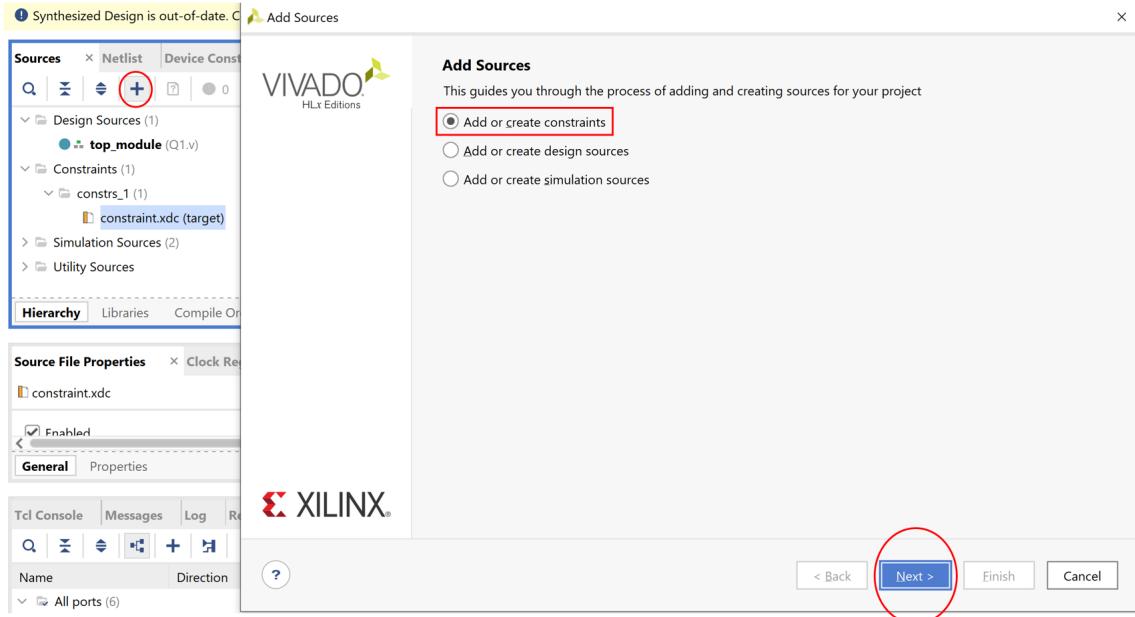
1 set_property PACKAGE_PIN T10 [get_ports {button[0]}]
2 set_property PACKAGE_PIN T11 [get_ports {button[1]}]
3 set_property PACKAGE_PIN T15 [get_ports {button[2]}]
4 set_property PACKAGE_PIN T14 [get_ports {button[3]}]
5 set_property PACKAGE_PIN V20 [get_ports {LED[0]}]
6 set_property PACKAGE_PIN W19 [get_ports {LED[1]}]
7 set_property IOSTANDARD LVCMS33 [get_ports {button[2]}]
8 set_property IOSTANDARD LVCMS33 [get_ports {button[1]}]
9 set_property IOSTANDARD LVCMS33 [get_ports {button[3]}]
10 set_property IOSTANDARD LVCMS33 [get_ports {button[0]}]
11 set_property IOSTANDARD LVCMS33 [get_ports {LED[1]}]
12 set_property IOSTANDARD LVCMS33 [get_ports {LED[0]}]
13
14

```

Annotations:

- Line 1: button[0] is connected to pin T10 via a Zynq package.
- Line 2: button[1] is connected to pin T11 via a Zynq package.
- Line 3: button[2] is connected to pin T15 via a Zynq package.
- Line 4: button[3] is connected to pin T14 via a Zynq package.
- Line 5: LED[0] is connected to pin V20 via a Zynq package.
- Line 6: LED[1] is connected to pin W19 via a Zynq package.
- Line 7: button[2] is connected to pin T10 via an LVCMS33 standard.
- Line 8: button[1] is connected to pin T11 via an LVCMS33 standard.
- Line 9: button[3] is connected to pin T15 via an LVCMS33 standard.
- Line 10: button[0] is connected to pin T14 via an LVCMS33 standard.
- Line 11: LED[1] is connected to pin W19 via an LVCMS33 standard.
- Line 12: LED[0] is connected to pin V20 via an LVCMS33 standard.

در Constraint File تمام محدودیت‌ها و قیودی که برای ساخته‌شدن مدار و قرارگیری آن درون اعمال می‌کنیم، با فرمتی که برای Vivado قابل فهم باشد نوشته شده است؛ یکی از این قیود همان اتصال پورت‌های مدار به پین‌های مناسب تراشه است (البته Constraint File فقط برای این کار نیست).

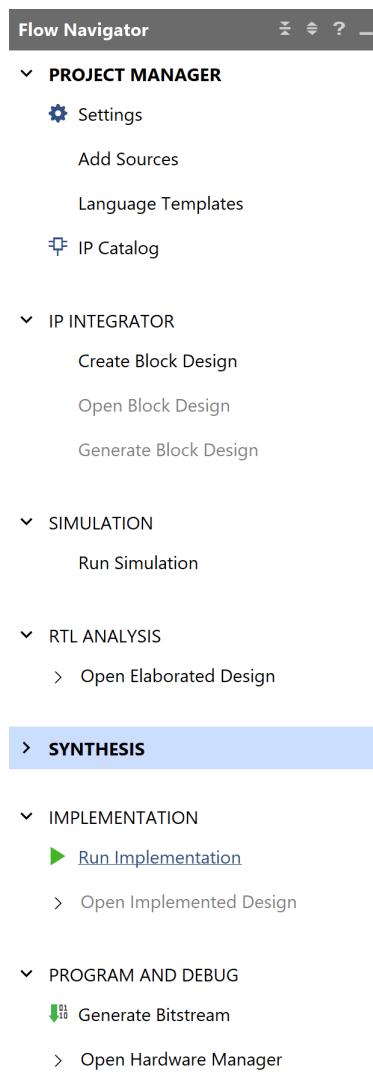


تنظیم اتصال ورودی خروجی‌ها به پین مناسب به صورت گرافیکی و از طریق تابع I/O Ports امکانی است که برای راحتی کار در اختیار ما گذاشته و سپس خودش را از روی آن می‌سازد. در غیر این صورت با نوشتن فرمت یک فایل XDC، می‌توانیم خودمان مستقیماً این فایل را از طریق پنجره Source ایجاد یا اضافه کنیم و درون آن کدزنی کنیم.

برای راحتی کار شما، Constraint File مربوط به کلیدها و LED‌های روی بورد در قالب یک فایل به فرمت XDC در **این لینک** قابل دسترسی است. این فایل را به عنوان Constraint در تمام بخش‌های این آزمایش اضافه کنید.

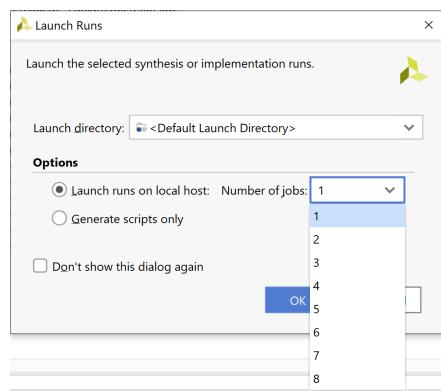
 توجه کنید که ورودی خروجی‌های Top Module شما حتما باید به شکل **این کد** باشد (حتی اگر در بخشی از آزمایش از برخی از کلیدها یا LED‌ها استفاده نمی‌کنید) تا فایل Constraint به درستی اعمال شود.

(ب) **Implementation**: در این مرحله عملاً مدار سنتز شده قرار است بر روی FPGA قرار گیرد. در ساده‌ترین حالت همانند قرار دادن گیت‌ها بر روی برد بورد و سپس سیم‌کشی بین آنهاست. پس مهم است که از چه تراشه‌ای با چه امکانات و میزان منابع مصرفی می‌کنیم. الگوریتم‌های به کار رفته برای implement کردن هم نسبتاً پیچیده و از موضوع درس خارج است، ولی خوب‌بختانه توسط کامپیوتر و Vivado انجام و بسیار ساده‌تر از استفاده از عناصر فیزیکی است.

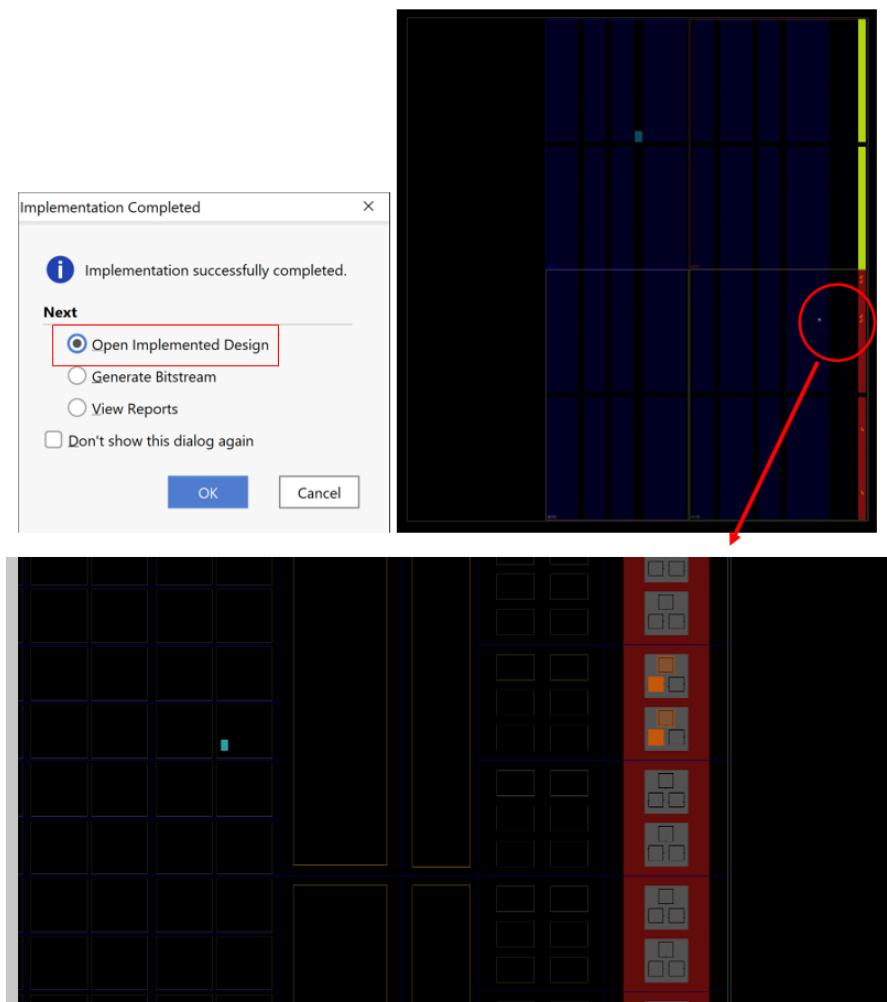


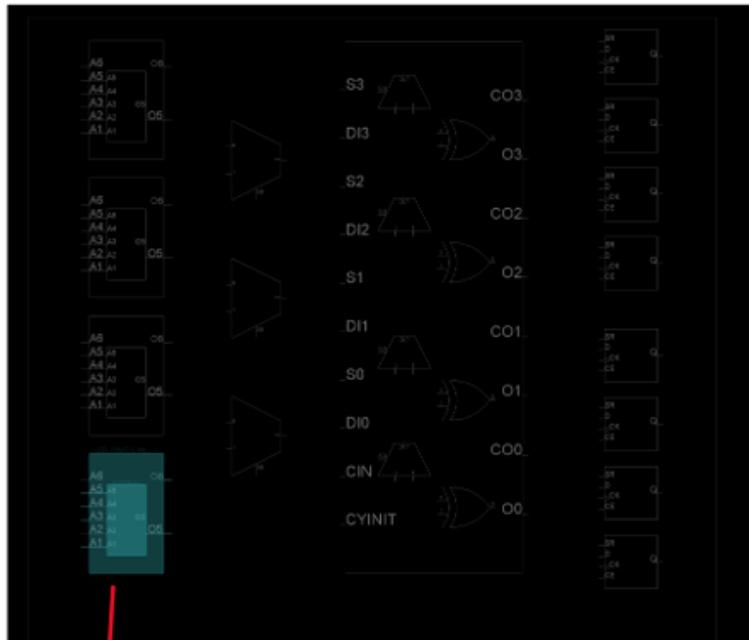
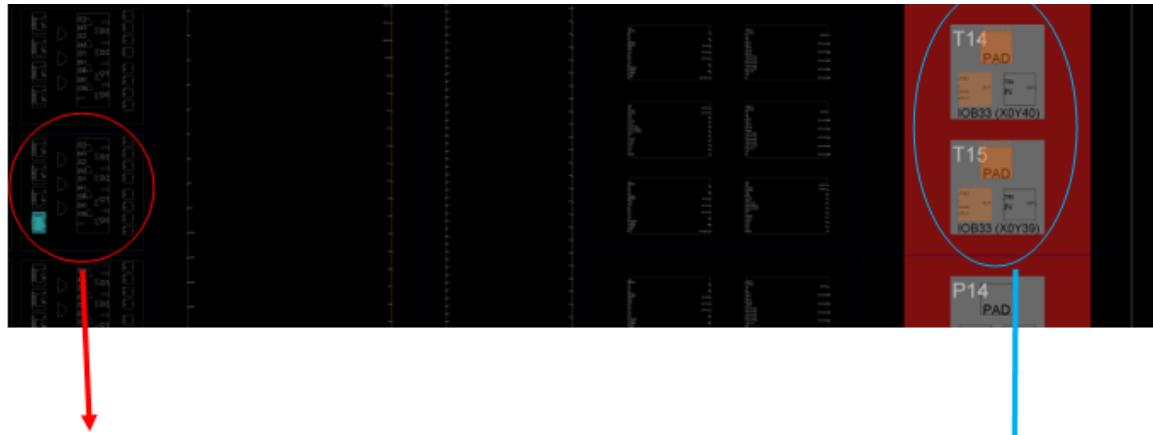
فرایند Implementation به جهت الزامات سیم‌کشی یا برقراری ارتباط بین بخش‌های مختلف، یک فرایند ترتیبی است؛ لذا تعداد هسته پردازنده که به آن اختصاص می‌دهید اهمیتی ندارد و با همان ۱ یا ۲ هسته به ماکزیمم سرعت انجام این فرایند می‌رسیم. (بعضًا اگر تعداد هسته‌های زیادی را در گیر کنید به جهت درگیری بیجای آنها سرعت کلی کامپیوتر شما و به تبع آن فرایند implementation

کنترل هم می شود).



پس از اتمام implementation با زدن گزینه زیر می توانید مشاهده کنید که مدار شما به چه صورت و در چه بخشی از تراشه (FPGA) پیاده شده است:



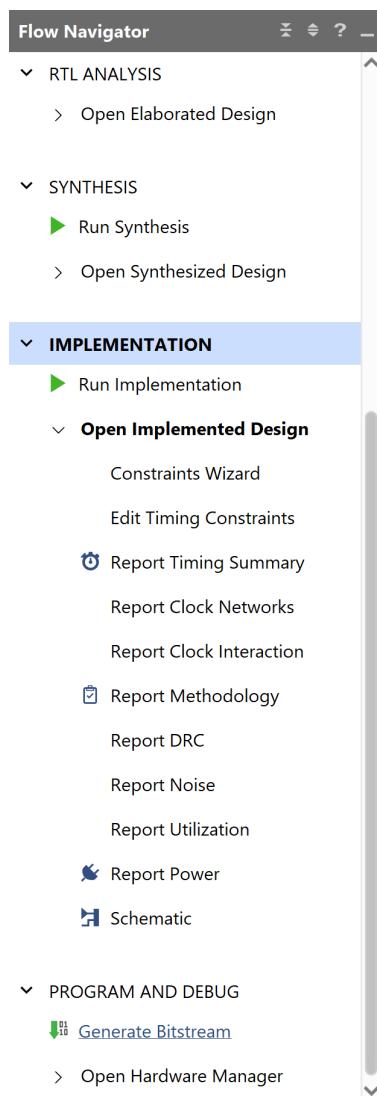


مورد استفاده برای پیاده‌سازی گیت‌های مدار LUT (Look Up Table)

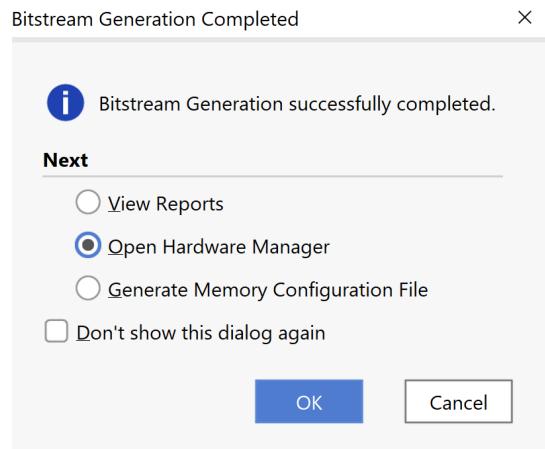
بین‌هایی از تراشه که به عنوان IO مورد استفاده واقع شده اند و ورودی خروجی‌های مدار به آنها وصل شده اند. (پین T14 و T15 که طبق جدول پین‌ها، از نظر سخت‌افزاری در PCB بورد به [2] و Button[3] و Button[2] متصل اند)



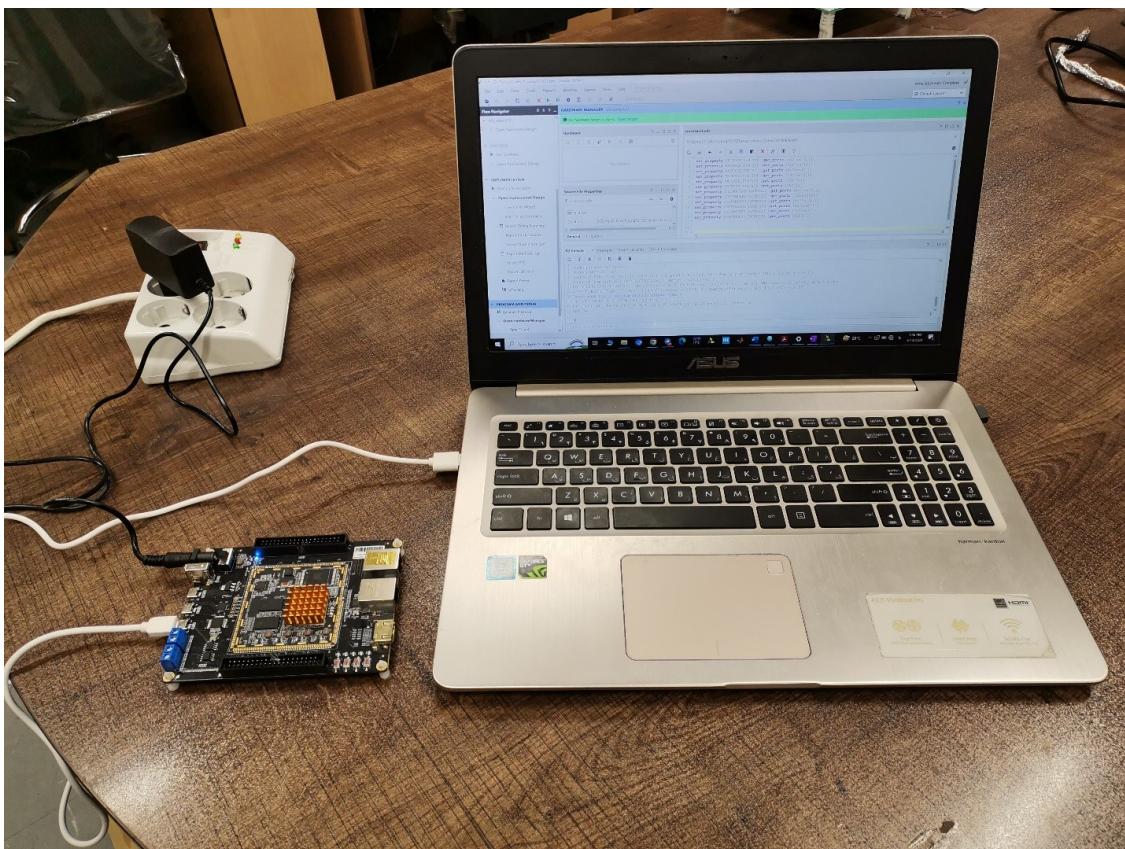
ج) ساخته شدن Bitstream: در مرحله بعد باید اطلاعات مدار implement شده به قالبی در بیاید که پروگرمر تراشه آن را بشناسد و آن را برنامه‌ریزی کند. به این قالب مشخص File و به فایل حاصل از این مرحله Bitstream گفته می‌شود که با فرمت bit. ساخته و ذخیره می‌شود. در ساده‌ترین حالت تصور کنید این فایل شامل تعدادی ۰ و ۱ است که پروگرمر بر اساس این استریم از بیت‌ها متوجه می‌شود کدام دو گیت درونی FPGA را به هم متصل بکند.



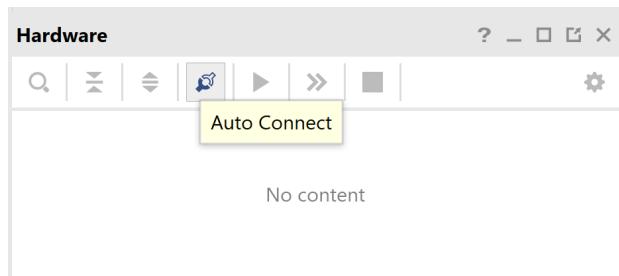
د) پروگرم کردن تراشه و بورد: پس از ساخته شدن Bitstream با انتخاب گزینه زیر وارد Hardware Manager شوید: (Hardware Manager را از قسمت PROGRAM AND DEBUG در Flow Navigator نیز می‌توان باز کرد).



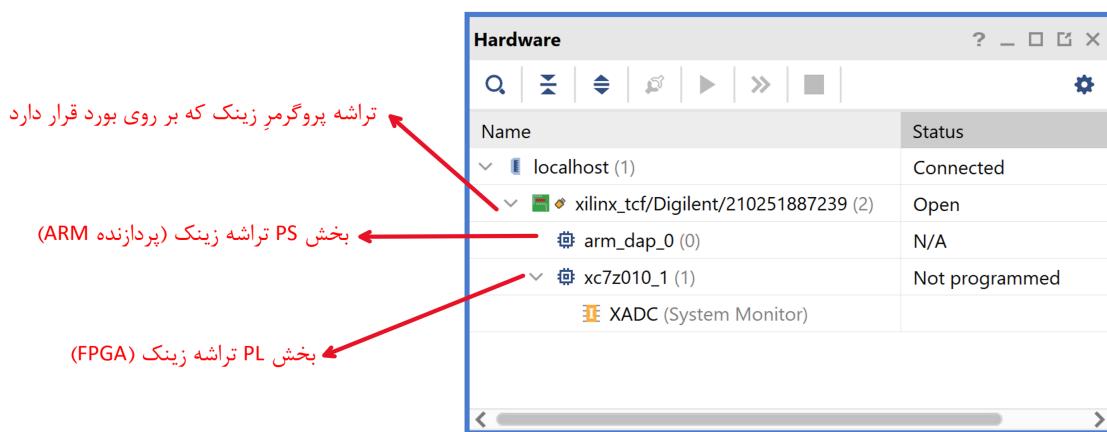
حال بورد را از طریق پورت JTAG به کامپیوتر خود متصل کرده و پاور بورد را نیز روشن کنید:



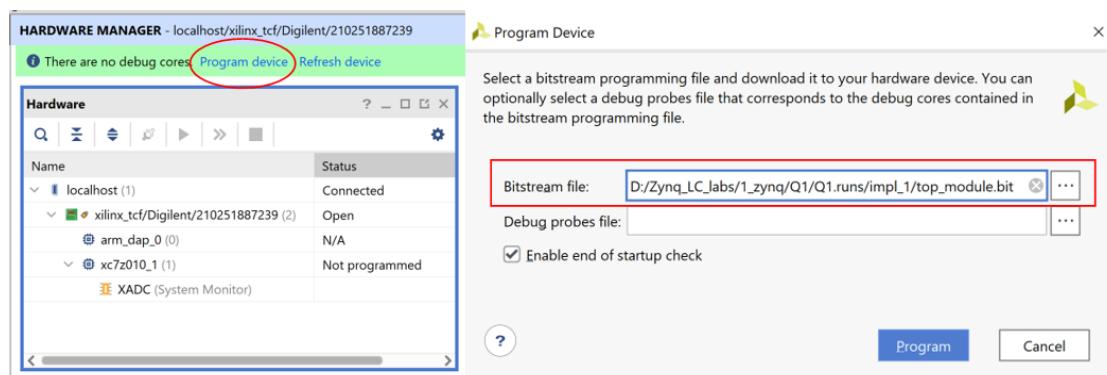
سپس Auto Connect را بزنید تا به طور خودکار تراشه را بشناسد:



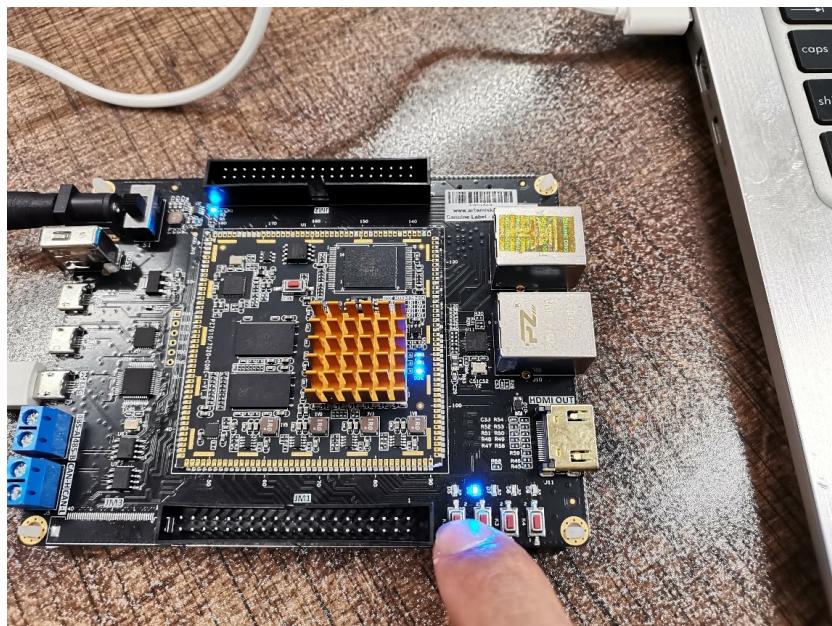
Vivado (کامپیوتر شما) با اتصال به پورت JTAG بورد و از طریق آن، ابتدا IC پروگرم تراشه زینک را می‌شناسد که از لحاظ سخت‌افزاری به پورت JTAG متصل است؛ سپس به واسطه پروگرم، بخش PS و PL زینک را پیدا می‌کند:



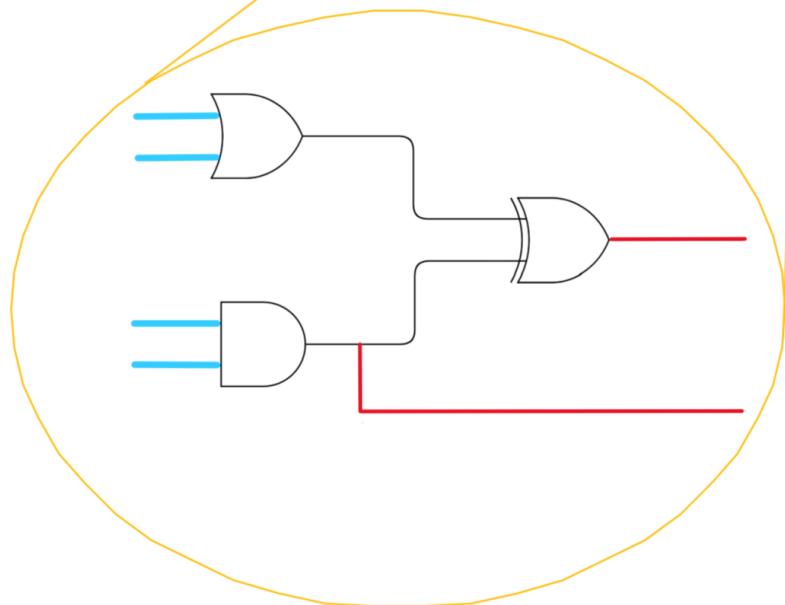
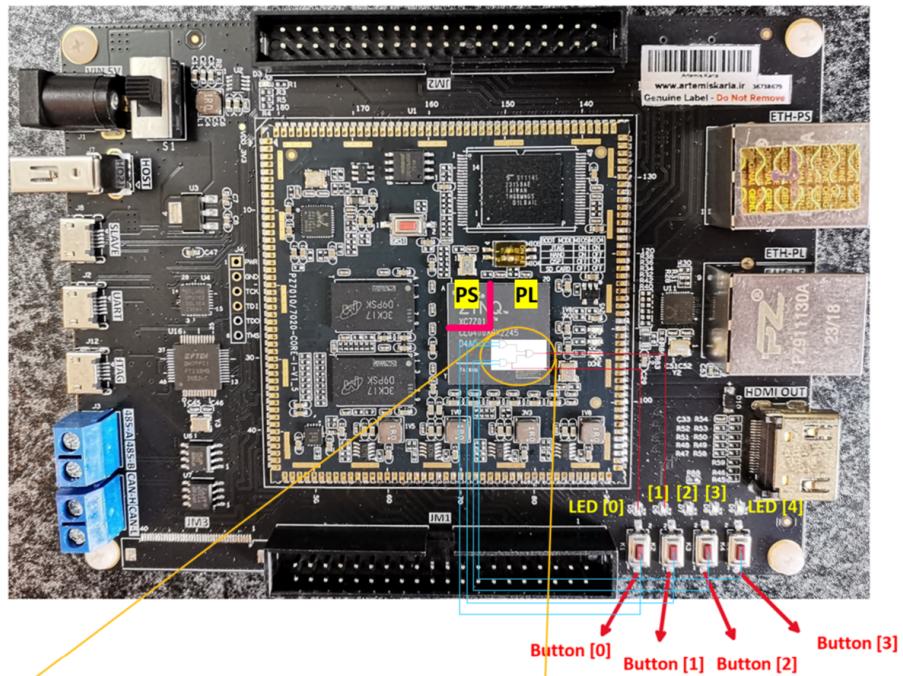
با زدن گزینه device Program و تنظیم درست فایل bitstream پروژه، تراشه را پروگرم کنید:



۵) تست پروژه روی بورد:

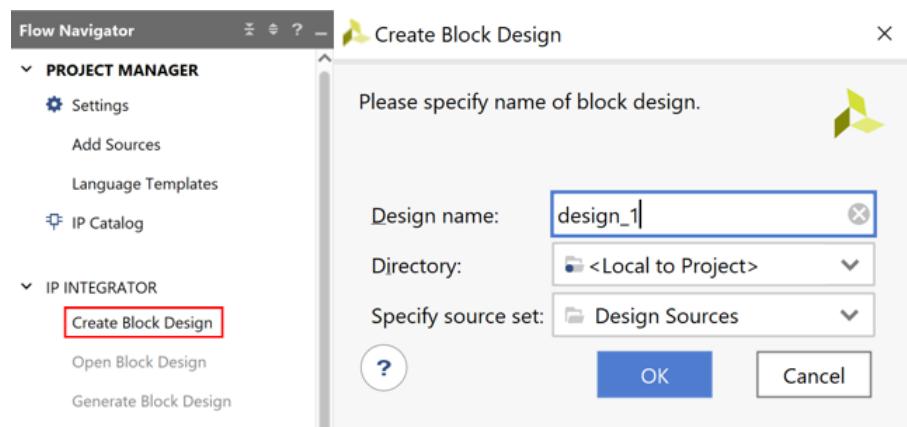


شکل کلی مدار در بورد پس از پروگرم کردن FPGA به این شکل خواهد بود: (توجه کنید که low active و کلیدها high active LEDها هستند)

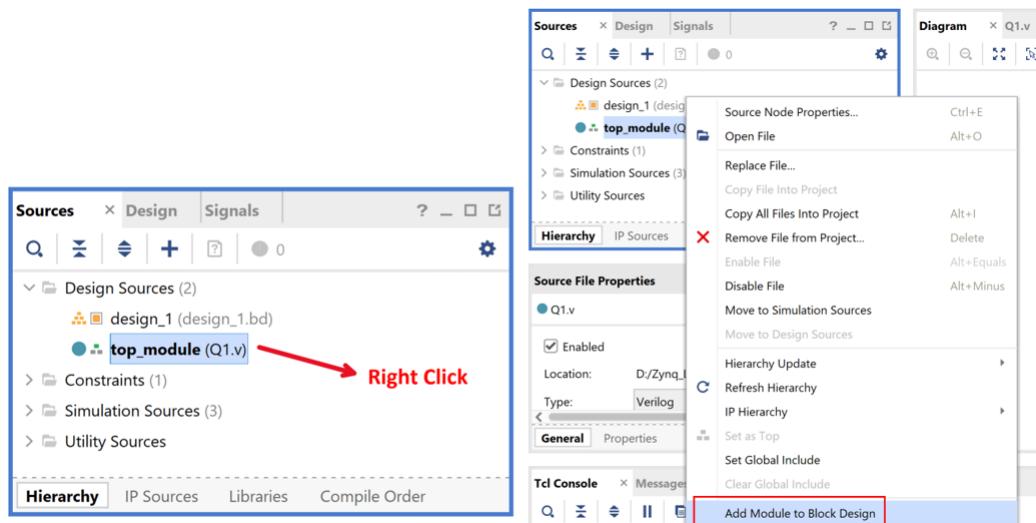


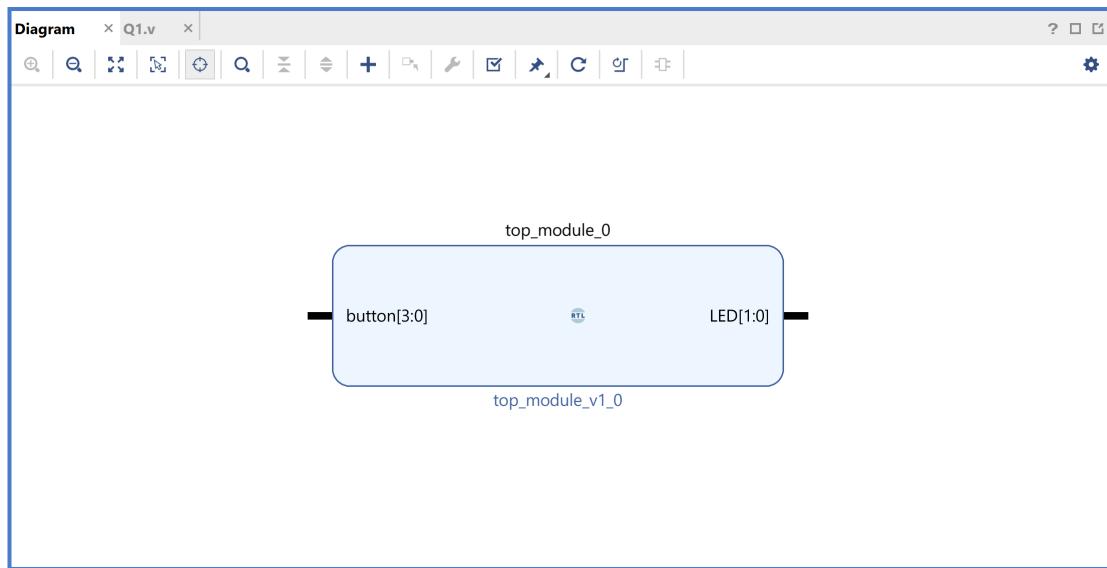
۴-۲-۳ - کار با Block Design در Vivado

در این مرحله از فضای گرافیکی (Block Design or Diagram) در Vivado استفاده می‌کنید. این محیط گرافیکی را برای راحتی کار شما قرار داده تا بتوانید به راحتی مازول‌های خود و IP‌های Xilinx Core را به آن اضافه کرده و به هم متصل کنید و شهودی تصویری و گرافیکی از سخت‌افزار نهایی که در FPGA ساخته می‌شود، داشته باشید. ابتدا یک دیاگرام بسازید:

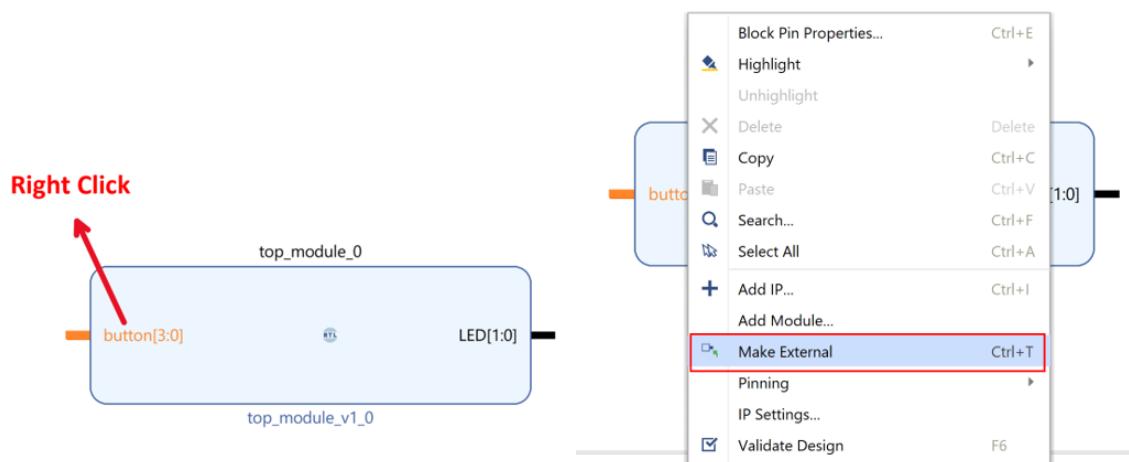


ماژول خود را به محیط گرافیکی Vivado اضافه کنید:

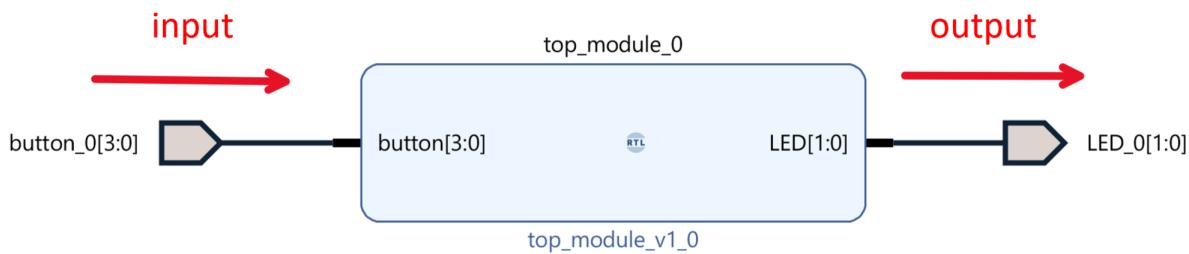
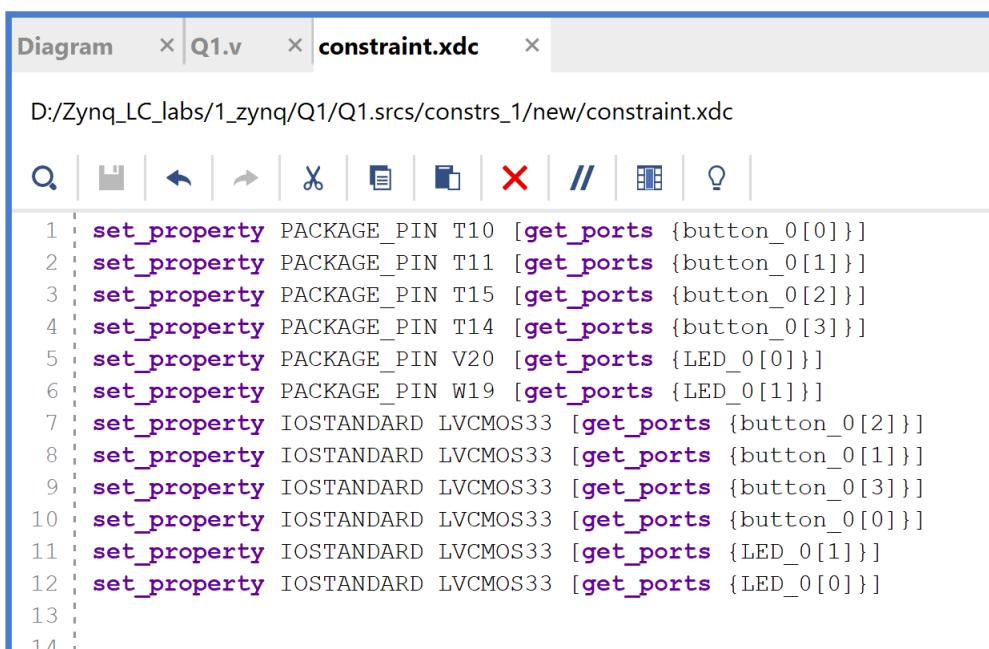




ورودی خروجی‌های ماژول را آماده اتصال به پین‌های تراشه باشد:



حال متناسب با نام جدیدی که External Port هر پورت ورودی یا خروجی گرفته، Constraint File خود را آپدیت کنید: (همچنین می‌توانید با کلیک بر روی پورت اکسترناł شده، نام آن را از پنجه Properties External Port به دلخواه تغییر دهید)

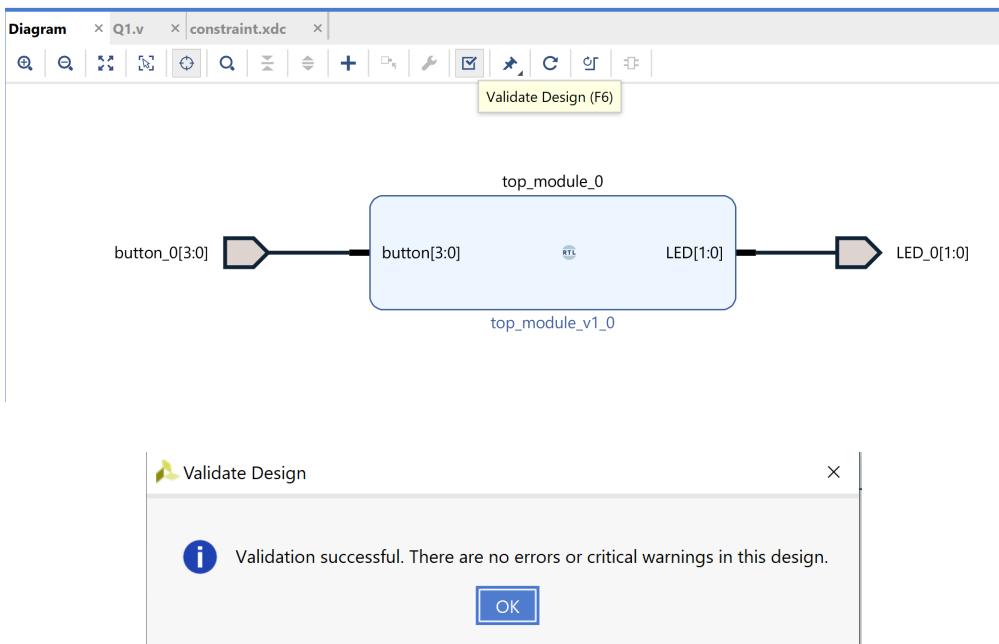
```

D:/Zynq_LC_labs/1_zynq/Q1/Q1.srcts/constrs_1/new/constraint.xdc

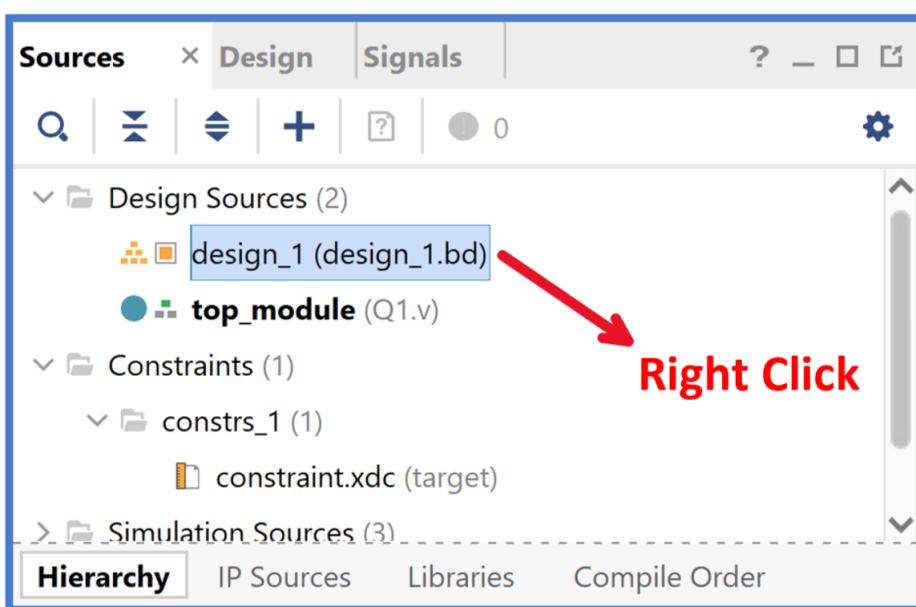
set_property PACKAGE_PIN T10 [get_ports {button_0[0]}]
set_property PACKAGE_PIN T11 [get_ports {button_0[1]}]
set_property PACKAGE_PIN T15 [get_ports {button_0[2]}]
set_property PACKAGE_PIN T14 [get_ports {button_0[3]}]
set_property PACKAGE_PIN V20 [get_ports {LED_0[0]}]
set_property PACKAGE_PIN W19 [get_ports {LED_0[1]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {button_0[2]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {button_0[1]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {button_0[3]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {button_0[0]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {LED_0[1]}]
set_property IOSTANDARD LVC莫斯33 [get_ports {LED_0[0]}]

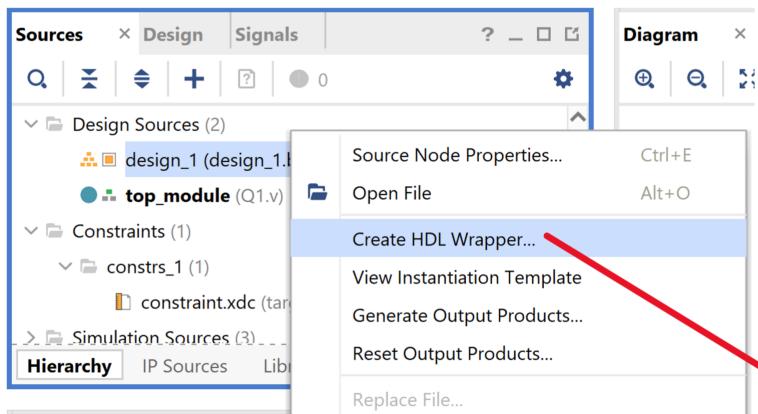
```

با زدن گزینه Design Validate از صحت اولیه دیاگرام گرافیکی خود از نظر نوع اتصالات، تعداد بیت پورت‌های متصل به هم، هماهنگی Low Active و High Active بودن پورت‌های متصل به هم، اتصال صحیح کلک و ریست مدار و... اطمینان حاصل کنید: (توجه کنید که این گزینه صرفاً برای چک کردن صحت موارد ابتدایی دیاگرام است و به هیچ وجه جای سنتز را نمی‌گیرد)

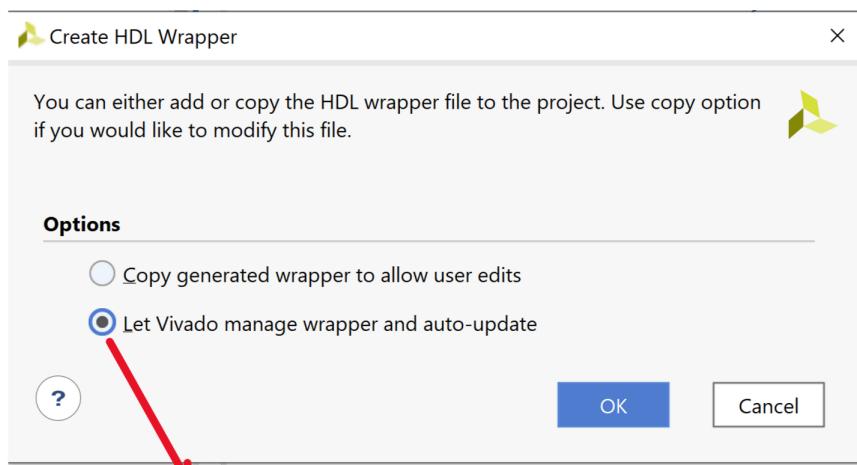


نکته مهمی که در کار با دیاگرام گرافیکی باید بدانید این است که این فضا صرفاً برای راحتی کار شما طراحی شده و tool Synthesis نرمافزار Vivado برای سنتز مدار در نهایت از شما یک کد وریلاغ می‌خواهد؛ پس باید راهی باشد تا سخت‌افزار طراحی شده در دیاگرام را به یک کد وریلاغ تبدیل و با آن توصیفش کنیم:



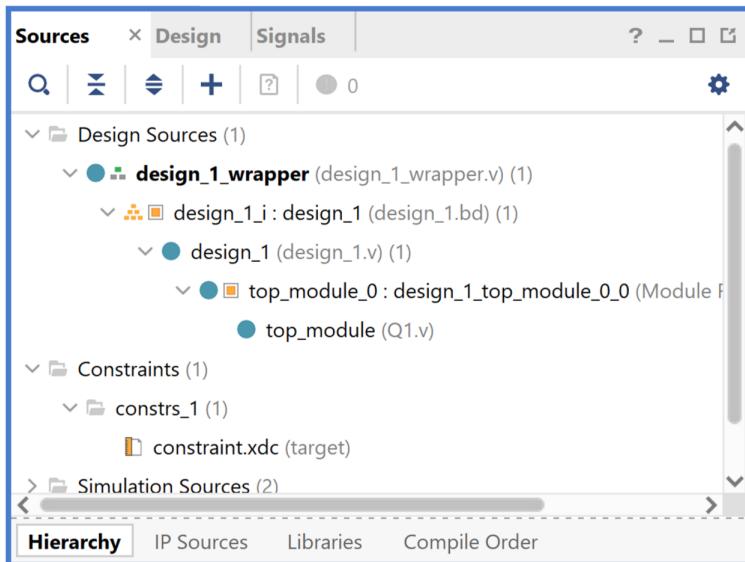


ساختن یک HDL Wrapper به عنوان مازول وریلاغ توصیف کننده دیاگرام گرافیکی



HDL Wrapper اجازه دهد در صورت تغییر دیاگرام، خودش به صورت خودکار Vivado توصیف کننده آن را آپدیت کند.

فایل وریلág به عنوان top module نهایی پروژه design_1_wrapper.v



حال لازم است مراحل پیاده‌سازی روی بورد را مجدداً طی کنید و نتیجه را تست کنید.

Synthesis and Implementation Out-of-date [details](#) 

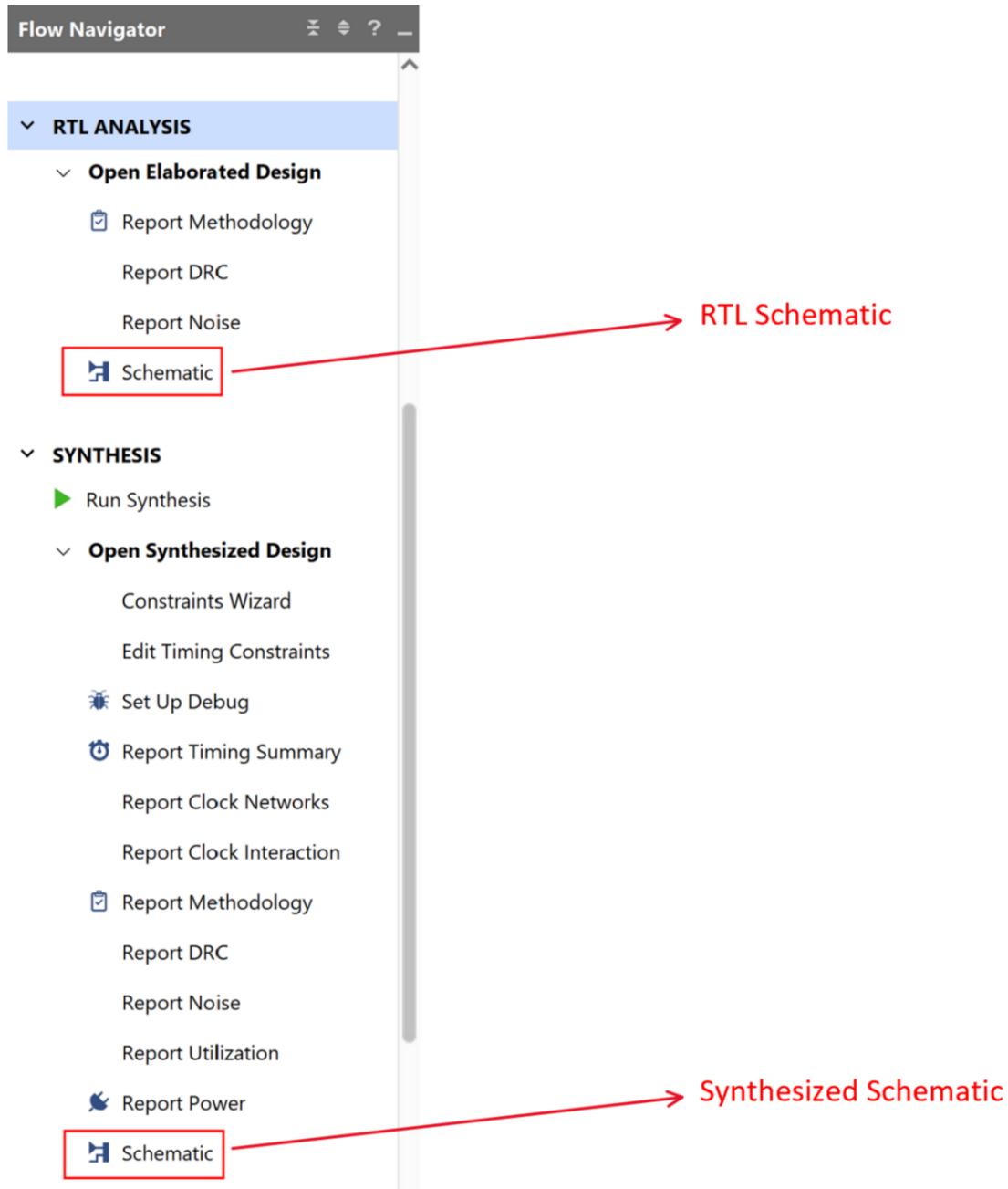
۳-۳- بخش دوم (زمان تقریبی: ۲۵ دقیقه)

کد وریلاغی بنویسید که مدار تابع منطقی زیر (که همان تابع بخش ۲-۶ آزمایش اول است) را توصیف کند و سپس آن را در FPGA پیاده‌سازی کنید.

$$f = \Sigma(0, 2, 4, 5, 8, 10, 12, 13)$$

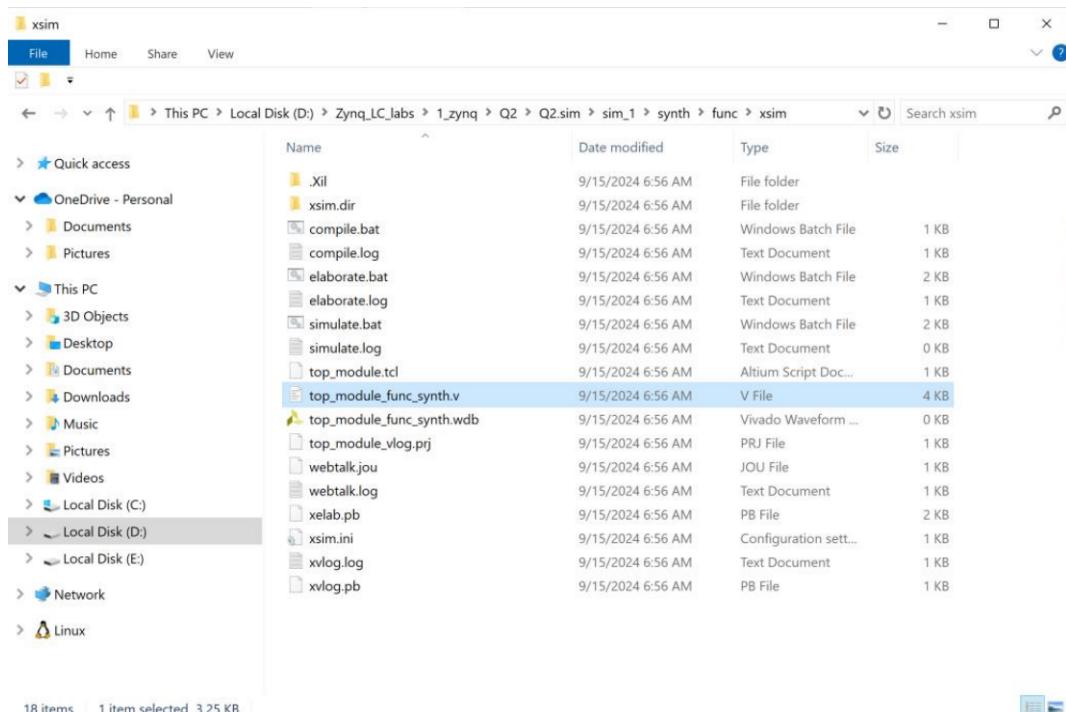
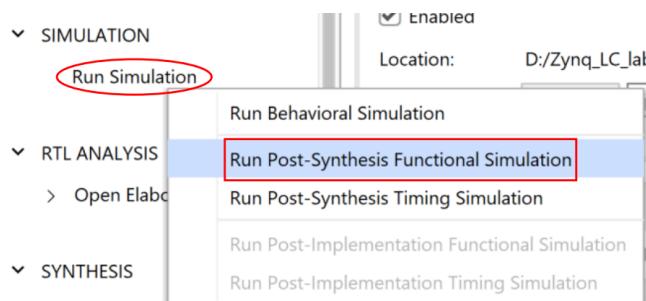
توجه کنید که لازم نیست قبل از توصیف، مدار را بهینه کنید چون این فرایند را نرمافزار به صورت خودکار انجام خواهد داد. با استفاده از کلیدهای فشاری به عنوان ورودی و LEDها به عنوان خروجی، مدارهایتان را راستی آزمایی نمایید.

۱. شماتیک مدار توصیف شده (RTL Schematic) و پس از سنتز، شماتیک مداری که روی پیاده شده FPGA (Technology Schematic – Synthesized Design Schematic) را مشاهده کنید و نتیجه را در گزارش بنویسید. (RTL Schematic را با طراحی خود در آزمایش اول مقایسه کنید). برای مشاهده RTL Schematic و Technology Schematic از بخش Analysis و Synthesis گزینه Schematic را انتخاب نمایید.



۲. با بررسی فایل وریلاغ پس از سنتز، حاصل بهینه‌سازی لاجیک مدار را مشاهده و ارزیابی کنید و در گزارش بنویسید.

پس از سنتز Run Post-Synthesis Functional Simulation را انتخاب کنید و پس از تولید ماژول Post-Synthesis مطابق تصاویر زیر این فایل را از فolder پروژه پیدا کنید:

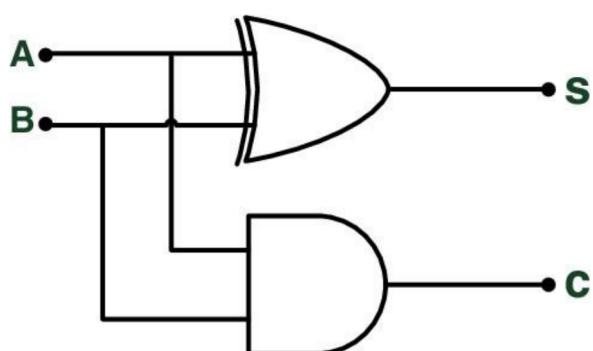


آزمایش شما Post-Synthesis Functional Simulation را انجام دادید که صرفاً بررسی صحت عملکردی کد وریلاغ Behavioral Simulation بود. اما این بار، پس از سنتز که اطلاعات بیشتری از مدار نهایی (که قرار است درون FPGA پیاده شود) بدست آمده، می‌توان با داشتن جزئیات بیشتر از مدار، شبیه‌سازی دقیق‌تری از آن داشت.

۴-۳-بخش سوم (زمان تقریبی: ۴۰ دقیقه)

هدف از این بخش طراحی یک Full Adder تکبیتی است.

۱. ابتدا کد وریلاغ مازول یک Half Adder تکبیتی را بزنید:



۲. سپس یک top module طراحی کنید و با instantiate کردن دو Half Adder تکبیتی درون آن، یک Full Adder تکبیتی بسازید. (در نحوه instantiate کردن و تایپ مناسب برای ورودی خروجی مازولها در اتصالات میان آنها دقت کنید)

۳. ابتدا پروژه خود را طبق مراحل simulation بخش اول، در Vivado شبیه‌سازی کرده و از صحت عملکرد آن اطمینان حاصل کنید.

۴. نتیجه را پس از انجام مراحل پیاده‌سازی روی بورد، به کمک کلیدها و LED ها تست کنید.

۵. این بار به جای instantiate کردن از Half Adder ها درون یک top module یک Full Adder (Block Design) Vivado اضافه کنید و با برقراری اتصالات لازم میان دو Half Adder، یک Full Adder بسازید. نهایتاً با انجام مراحل مربوط به دیاگرام بلاک دیزاین و مراحل پیاده‌سازی روی بورد، نتیجه را به کمک کلیدها و LED ها تست کنید.

۳-۵-۳- بخش چهارم (زمان تقریبی: ۴۰ دقیقه)

هدف از این بخش آشنایی با پیاده‌سازی مدارات با استفاده از multiplexer بر روی FPGA است.

در این بخش نیز فقط مجاز به استفاده از عبارات Continuous Assignment به عنوان مثال، برای پیاده‌سازی با multiplexer باید از عبارت زیر استفاده کنید:

```
1   wire x;  
2   assign x = ... ? ... : ...;
```

در مدار این بخش، پس از پیاده‌سازی، شماتیک مدار خود را (RTL Schematic) در Vivado به ترتیبی که در بخش اول همین آزمایش توضیح داده شد، مشاهده کنید. همچنین برای تست مدار این بخش بر روی FPGA از کلیدها و LED های بورد استفاده کنید.

جبر بول زیر را با استفاده از multiplexer بر روی FPGA پیاده‌سازی کنید.

$$F = CAB + \overline{C} (\overline{A} + B)$$

۳-۶- بخش پنجم (زمان تقریبی: ۴۰ دقیقه)

در این بخش یک Priority Encoder طراحی کنید. مدار شما چهار ورودی دارد با شماره‌های فرضی ۱ تا ۴ که همان چهار LED روی بورد هستند و در خروجی، بزرگترین شماره کلید فشرده شده را روی LED‌ها نمایش می‌دهد. (طبعتاً به سه LED نیاز دارید زیرا بیشترین شماره $3'b100 = 4$ است)

برای مثال اگر کلیدهای ۱ و ۳ همزمان فشرده باشند، بر روی LED‌ها $3'b011 = 3$ نمایش داده شود یعنی

۴- گزارش

در بخشی از قسمت های آزمایش بیان شد که نتیجه را در گزارش قید کنید. ابتدا موارد مربوطه را ذکر کرده و سپس در خصوص آنچه از این آزمایش آموختید گزارش ۱ الی ۲ صفحه‌ای (مطابق تمپلیت ارجاع شده در ابتدای این داکیومنت) بنویسید.
دیدگاه خود نسبت به بخش‌های مختلف آزمایش را به انضمام پیشنهادهایتان بنویسید.

اطلاعات تکمیلی آزمایش سوم

شما تاکنون با مدارهای دیجیتال کار کردید و در آزمایش‌های گذشته، مدارهای شامل چند گیت منطقی را با استفاده از گیت‌های منفصله‌ی خانواده $74xx$ ، با برقراری اتصالات لازم بر روی یک بردبورد، پیاده‌سازی نمودید و عملکرد آنها را بررسی کردید. در آن آزمایش‌ها، بردبورد بکار گرفته شده و اتصالات آنها همگی بصورت فیزیکی بودند. اما تصور کنید مدار دیجیتالی که قصد طراحی آن را دارد تعداد بسیار زیادتری گیت منطقی دارد، در واقع برای رسیدن به عملکرد مدنظرتان لازم است تعداد زیادی گیت داشته باشد (هزاران هزار گیت). برای مثال پردازنده‌ای که درون تلفن همراه شماست، یک مدار دیجیتال و منطقی است و برای اینکه بتواند تمام محاسبات و عملیات‌های مدنظر را انجام دهد لازم است دهها هزار گیت درونش باشد (و طبعاً تعداد بسیار بیشتری ترانزیستور که گیت‌های منطقی از آنها ساخته شده). همانطور که متوجه شده اید برای طراحی چنین مداری دیگر نمی‌توان آن را در سطح Gate Level در پروتئوس طراحی کرد و بر روی برد بورد بست.
پس راه حل چیست؟!

این وسیله و با این ادبیات مدار مدنظرمان را توصیف می‌کنیم. توجه کنید که در زبان‌های HDL درست است که کد می‌زنیم اما قرار نیست کد ما بر روی چیزی مثل پردازنده لپتاپ اجرا شود (مانند درس برنامه نویسی ترم اول) بلکه HDL راهی است برای توصیف مدار دیجیتالی که در ذهنمان است (تصور کنید یک متن می‌نویسید تا وضعیت یک اتاق را توصیف کنید، در آن متن قرار نیست دستوری به کسی بدھید تا اجرا کند، بلکه صرفاً در حال توصیف هستید پس به طور کلی داشتن ترتیب بین خطوط کد HDL بی‌معنی است، قرار نیست دستوراتی پشت سر هم و با ترتیب اجرا شوند، مانند کدها و زبان‌های نرم‌افزاری).

یکی از های HDL که در دروس دانشگاه می‌آموزید Verilog است. این زبان اولین بار در سال ۱۹۸۴ ابداع شد و چند سال بعد به استاندارد IEEE درآمد. (در این متن برای سادگی از این پس Verilog را به فارسی خواهیم نوشت یعنی وریلاغ)

در این مرحله باید انتظار داشته باشید ابزاری وجود داشته باشد تا کد وریلاغ شما را به مدار دیجیتال مذکور تبدیل کند؛ به این ابزار اصطلاحاً Synthesis Tool گفته می‌شود یعنی ابزار سنتز مدار. بعد از سنتز شدن مدار و طی کردن یک فرایند و flow مشخص در نرم‌افزارهای مخصوص

این کار، به layout مدار مجتمع مدنظر خواهیم رسید که آماده ساخته شدن (اصطلاحاً fabricate شدن) در شرکت‌هایی مثل TSMC است که کارشان تولید IC است. به این کار طراحی ASIC (Application Specific Integrated Circuit) می‌گویند؛ یعنی طراحی یک IC دیجیتال خاص که دقیقاً و تنها همان عملکرد و کاربردی را دارد که ما در کد وریلاغ توصیف کردیم. اما هدف ما در این درس طراحی ASIC نیست! پس قرار است با وریلاغ چه کنیم؟! فقط در حد شبیه‌سازی کردن مدار توصیف شده با وریلاغ و بررسی صحت عملکرد آن؟! خیر، گرچه این کار را هم مختصراً انجام خواهیم داد.

مدار افزاری است که مدار ثابت با عملکرد FPGA (Field Programmable Gate Array) مشخصی درون آن شکل نگرفته، بلکه طوری طراحی شده تا بتواند مدارات دیجیتال دلخواه ما را درون خود بسازد. دنیای FPGA بسیار وسیع بوده و نحوه‌ی کارکرد آن نسبتاً پیچیده و از موضوع درس و این آزمایش و دستور کار خارج است. برای کاربرد جاری و تسهیل موضوع و با یک ساده‌سازی نسبتاً عمیق، FPAG را می‌توان یک برد الکترونیکی دانست.

این بدان معناست که در یک FPGA شما می‌توانید چند صد گیت را قرار داده و با کمترین زحمت، اتصالات میان آنها را برقرار کنید.

در ساده‌ترین حالت تصور کنید یکسری گیت منطقی در بلوک‌هایی مشخص درون FPGA وجود دارد و بین آنها سیم‌ها و سوئیچ‌هایی قرار گرفته است؛ حال این سیم‌ها طوری گیت‌ها را به هم متصل می‌کنند تا مداری که ما (با وریلاغ) توصیف کرده‌ایم درون بخشی از FPGA ساخته شود؛ درواقع خانواده‌های FPGA (همانطور که از اسمشان مشخص است) سخت‌افزاری قابل برنامه‌ریزی (programmable) هستند. طبیعتاً انتظار دارید FPGA‌ها هم Synthesis tool داشته باشند تا کد وریلاغ ما را به مدار دیجیتال و سپس به ادبیاتی درآورد که پروگرام FPGA آن را بفهمد و درون FPGA بسازد.

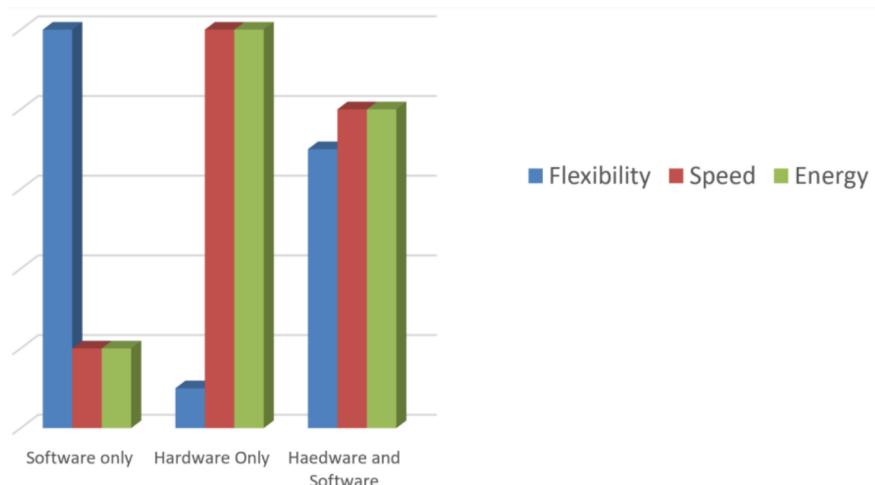
Xilinx یکی از شرکت‌هایی است که FPGA می‌سازد و نرم‌افزار Vivado را به عنوان ابزاری برای این فرایند در جدیدترین تراشه‌های خود (از نسل ۷ به بعد) معرفی کرده است. شما برای آزمایش‌های این درس لازم است **نرم‌افزار Vivado (نسخه ۲۰۱۹.۱)** را نصب نمایید.

اما کاربرد FPGA چیست؟ دو نوع الگوریتم داریم، یکسری الگوریتم‌ها ثابت یا static هستند یعنی عملیات مشخصی روی داده‌های ورودی انجام می‌شود (مثلاً وارون یک ماتریس ورودی باید محاسبه شود). عمدۀ الگوریتم‌های مربوط به پردازش سیگنال و پردازش تصویر از این دسته‌اند. از

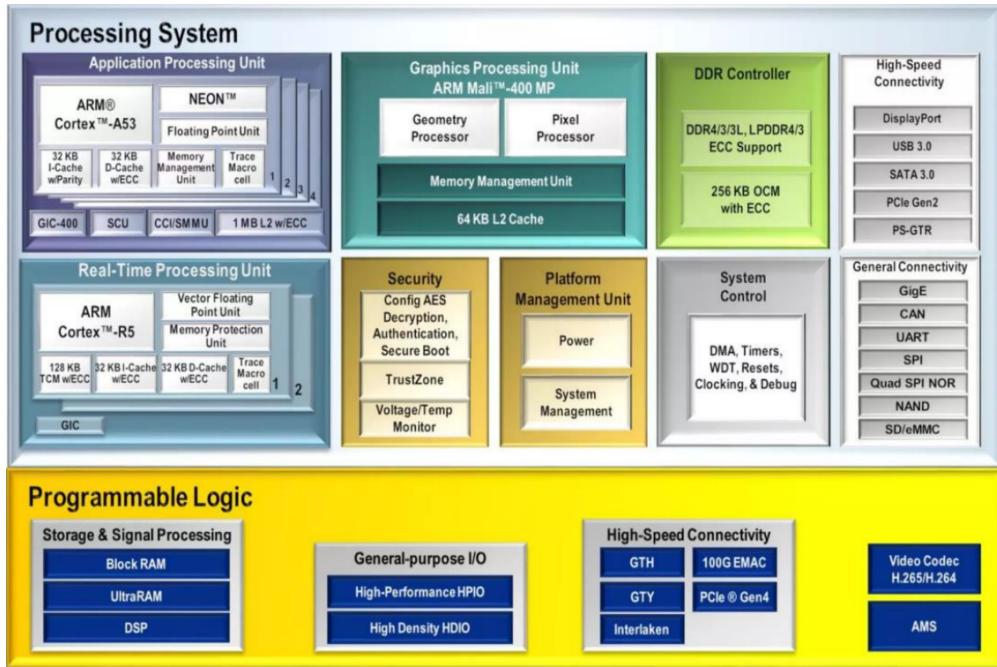
طرفی این پردازش‌ها باید با بیشترین سرعت و کمترین تأخیر انجام شود و به جهت static بودن الگوریتم از نظر منابع سخت‌افزاری مصرفی امکان پیاده‌سازی مداری که خاصّ انجام این الگوریتم باشد، وجود دارد؛ بنابراین مدار آنها را در FPGA پیاده می‌کنیم.

دسته دوم الگوریتم‌های adaptive هستند یعنی الگوریتم‌هایی که پر از شرایط گوناگون و نامشخص اند (به زبان ساده پر از if و else اند). به جهت منابع و محدودیت‌های دیگر امکان پیاده کردن مدار خاصّ این الگوریتم‌ها وجود ندارد. لذا برای اجرای آنها از پردازنده‌ها و میکروکنترلرها استفاده می‌کنیم.

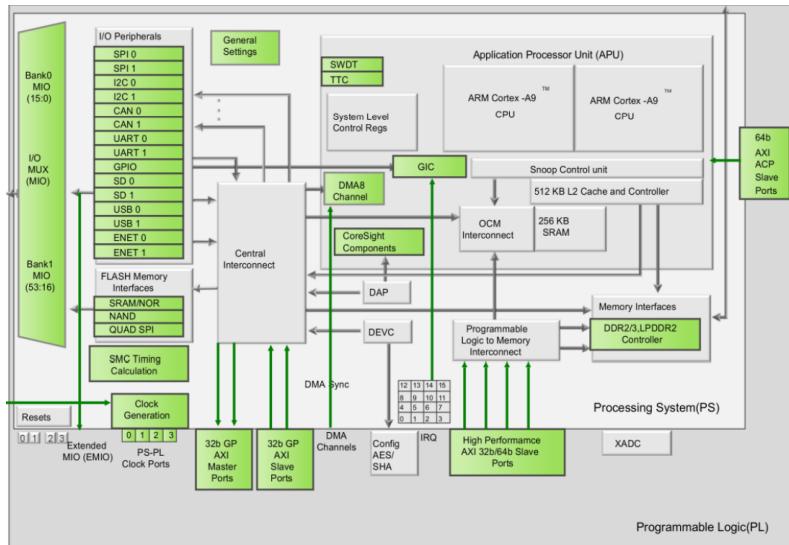
پروژه‌های ما دارای بخش‌هایی با الگوریتم‌های static و بخش‌هایی با الگوریتم‌های adaptive اند. برای برآورده کردن این نیاز شرکت Xilinx در سال ۲۰۱۱ تراشه‌ای را معرفی کرد که یک (System on Chip) SoC است یعنی یک سیستم داخل یک تراشه جا گرفته است. این IC به طور کلی شامل یک بخش FPGA است که به آن (Programmable Logic) PL می‌گوییم و یک بخش PS (Processing System) معروف است. علاوه بر پردازنده دو هسته‌ای ARM که دارد، شامل موارد گوناگون دیگری همچون حافظه، پورت‌ها، پروتکل‌ها و پریفرال‌های پیاده شده می‌باشد.



شکل ۳-۳: مقایسه سیستم‌های دارای بخش‌های سخت‌افزاری (FPGA) و نرم‌افزاری (میکرو) و تلفیق آنها از نظر انعطاف، سرعت و توان مصرفی



شکل ۴: بخش‌های مختلف درونی تراشه Zynq UltraScale+



شکل ۵: نمایی از درون تراشه Zynq 7000

برای کسب اطلاعات بیشتر در مورد تراشه Zynq می‌توانید به [Zynq 7000 reference manual](#) مراجعه کنید.



	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Block RAM (# 36 Kb Blocks)	1.8 Mb (50)	2.5 Mb (72)	3.8 Mb (107)	2.1 Mb (60)	3.3 Mb (95)	4.9 Mb (140)	9.3 Mb (265)	17.6 Mb (500)	19.2 Mb (545)	26.5 Mb (755)
	DSP Slices (18x25 MACCs)	66	120	170	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	73 GMACs	131 GMACs	187 GMACs	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express (Root Complex or Endpoint) ⁽³⁾		Gen2 x4			Gen2 x4		Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security ⁽²⁾	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication									

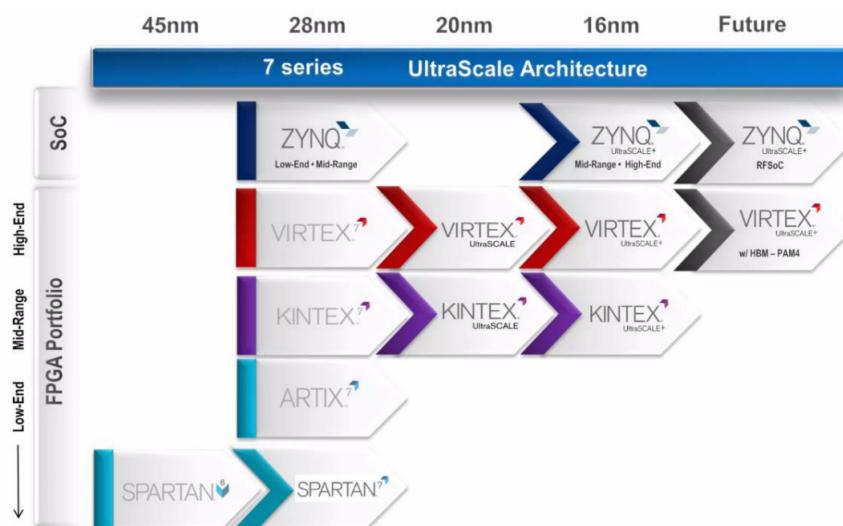
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100																			
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100																			
Processing System	Processor Core	Single-core ARM Cortex-A9 MPCore™ with CoreSight™	Dual-core ARM Cortex-A9 MPCore™ with CoreSight™																											
	Processor Extensions	NEON™ & Single / Double Precision Floating Point for each processor																												
	Maximum Frequency	667 MHz (-1); 766 MHz (-2)	667 MHz (-1); 766 MHz (-2); 866 MHz (-3)		667 MHz (-1); 800 MHz (-2); 1 GHz (-3)		667 MHz (-1) 800 MHz (-2)																							
	L1 Cache	32 KB Instruction, 32 KB data per processor																												
	L2 Cache	512 KB																												
	On-Chip Memory	256 KB																												
	External Memory Support ⁽¹⁾	DDR3, DDR3L, DDR2, LPDDR2																												
	External Static Memory Support ⁽¹⁾	2x Quad-SPI, NAND, NOR																												
	DMA Channels	8 (4 dedicated to Programmable Logic)																												
	Peripherals ⁽¹⁾	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO																												
	Peripherals w/ built-in DMA ⁽¹⁾	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO																												
	Security ⁽²⁾	RSA Authentication, and AES and SHA 256-bit Decryption and Authentication for Secure Boot																												
Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master 2x AXI 32-bit Slave																													
	4x AXI 64-bit/32-bit Memory																													
	AXI 64-bit ACP																													
	16 Interrupts																													

شکل ۳-۶: بخشی از ug585 - امکانات و مشخصات بخش PL و PS مدل های مختلف Zynq

با توجه به قابلیت‌های ویژه Zynq، این تراشه جای خود را در صنعت ایران و جهان باز کرده؛ به طوری که به دلیل تیراز بالای تولید آن، قیمت آن به صرفه‌تر از تراشه‌هایی بعض‌اً ضعیف‌تر از آن شده است. به طور مثال در جدول زیر امکانات و قیمت Zynq با سری Artix از FPGA های Xilinx مقایسه شده که نتیجه بسیار جالب است: (بخش FPGA خود این Zynq از نوع Artix است)

	Artix	Zynq	Artix
	XC7A75T	XC7z020	XC10075T
Logic Cells	75k	85k	101k
Block RAM	210	280	270
DSP48	180	220	240
GTx	8	0	8
Price	116\$	104\$	137\$

جدول ۲-۳: مقایسه امکانات و قیمت تراشه Zynq و Artix



شکل ۷-۳: تکنولوژی ساخت Zynq در مقایسه با دیگر تراشه‌های Xilinx

برای مشاهده تصاویر مفهومی بیشتر در مورد تراشه Zynq، می‌توانید به این [لينك](#) مراجعه کنید.
برای درک بهتر این مفاهیم اکیداً توصیه می‌شود این [ویدیو](#) را مشاهده بفرمایید.