دانشکده برق، کامپیوتر و فناوری های پیشرفته

گروه مهندسی برق الکترونیک

دستورکار آزمایشگاه FPGA

تهیه کنندگان:

آقای دکتر مرتضی موسی زاده

آقای مهندس سید کیان موسوی کیا

تاریخ تنظیم:

مهرماه ۱۴۰۳

آزمایش اول: معرفی بردها و آموزش نصب نرم افزار VIVADO و ISE

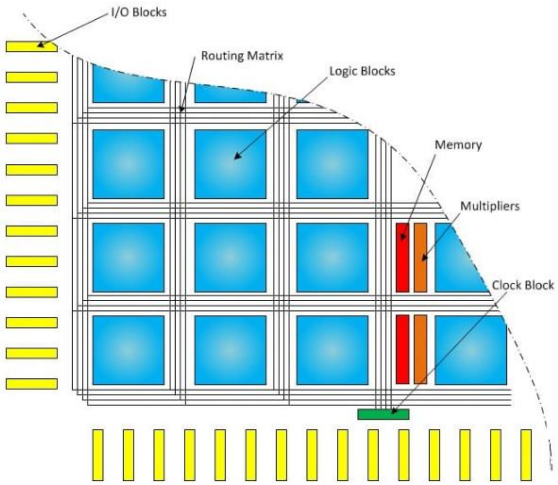
مقدمه:

FPGA چیست؟

۱-۱-

Fpga این عبارت مخفف field programmable gate array، بوده و یک تراشه نیمه رسانای قابل برنامه ریزی است که از اجزای الکترونیکی کوچکی به نام بلوک منطقی (logic block)یا سلول منطقی (logic cell)تشکیل شده است. به Fpga یا همان آرایه درگاه قابل برنامه ریزی، تراشه fpga هم گفته می شود چرا که امکان برنامه ریزی متناسب با نیاز کاربران و امکان تغییر در روابط منطقی بین داده های ورودی و خروجی را دارد. با استفاده از اف پی جی ای، کاربران می توانند مدار های خاص خود را طراحی کرده و بسازند. در واقع خود FPGA تنها یک برد خالی است که قابلیت ویژه ای ندارد بلکه هنگامی که شما مطابق نیازهایتان آن را پیکر بندی می کنید، کارایی پیدا می کند.

شکل ۱-۱ دیاگرامی از آنچه به طور معمول درون تراشه‌های FPGA وجود دارد را نشان می‌دهد. همانطور که در این شکل دیده می‌شود، درون FPGA آرایه‌ای از منابع دیجیتالی وجود دارد که به طور منظمی چیده شده‌اند. این منابع در ابتدا هیچ تابع یا مفهوم دیجیتالی را ایجاد نمی‌کنند، بلکه به عنوان بلوک‌های سازنده یک مدار دیجیتالی توسط طراح دیجیتال مورد استفاده قرار می‌گیرند تا هر مدار دلخواهی را ایجاد کند.



شکل ۱-۱ : دیاگرامی از آنچه به طور معمول درون تراشه‌های FPGA وجود دارد.

معرفی نرم افزار های مورد نیاز

۱-۲-

برای طراحی مدار های FPGA ونیز شبیه سازی و پروگرم کردن کد ها روی برد FPGA می توان از دو برنامه ی Xilinx Vivado Design Suite و ISE استفاده کرد.که در ادامه به طریقه ی نصب هر کدام می پردازیم.

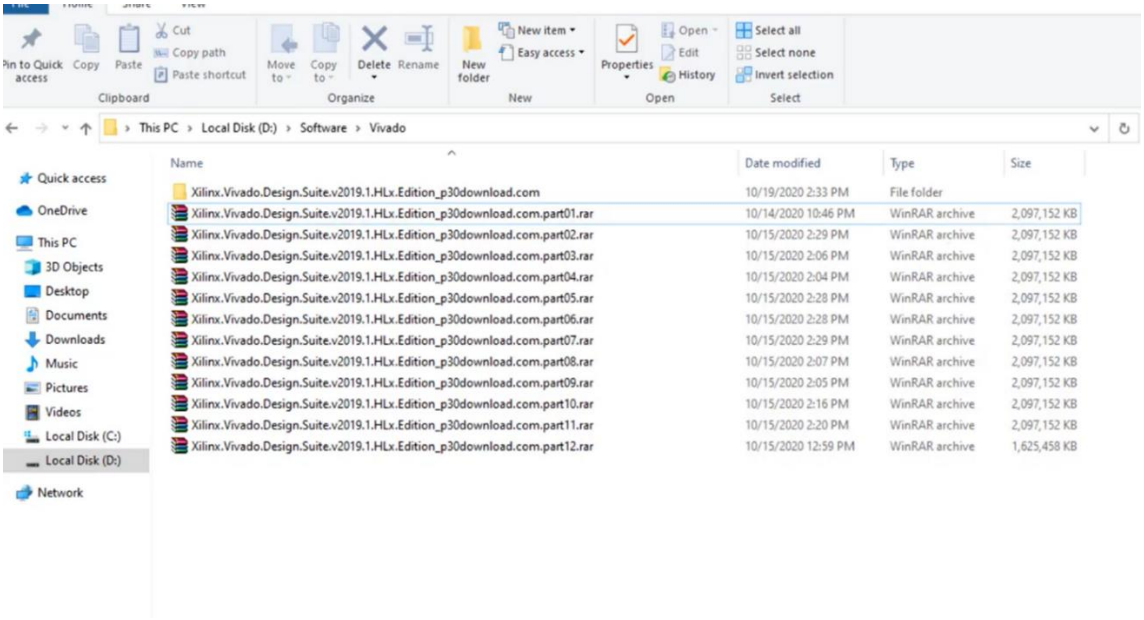
۱-۲-۱- آشنایی و نصب نرم افزار XILINX VIVADO DESIGN SUITE

Xilinx Vivado Design Suiteیک برنامه طراحی بردهای FPGA است،که جدیدترین نرم‌افزار شرکت Xilinx در حوزه پیاده‌سازی مدارات دیجیتال است. به کمک این نرم‌افزار می‌توانید تمام مراحل پیاده‌سازی مدار در FPGAهای این شرکت را انجام دهید.

در گام اول ، باید نرم افزار را دانلود کنید.

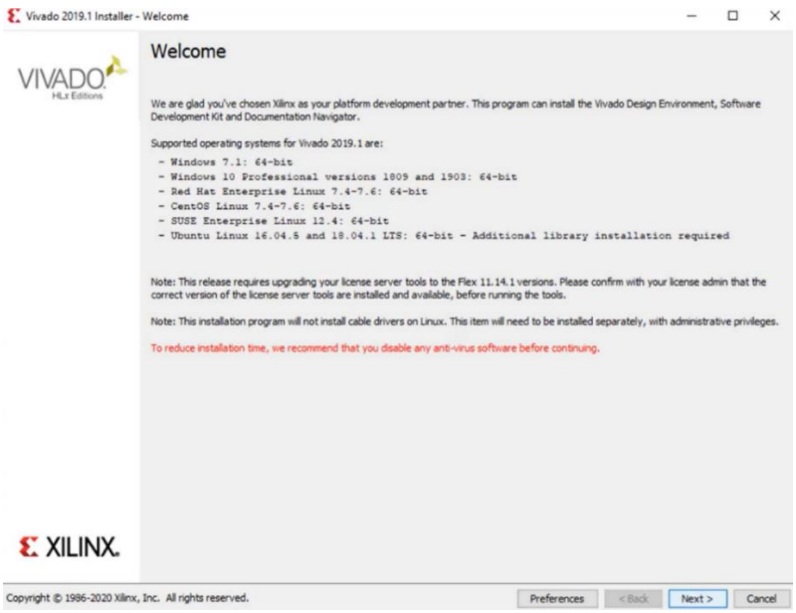
بعد از دانلود، فایل دانلود شده شامل ۱۲ فایل فشرده می باشد(بهتر است همه ی فایل ها داخل یک پوشه باشند).

روی اولین فایل فشرده راست کلیک کرده و با گزینه ی Extract Here فایل ها از حالت فشرده خارج می شوند (شکل ۲-۱).



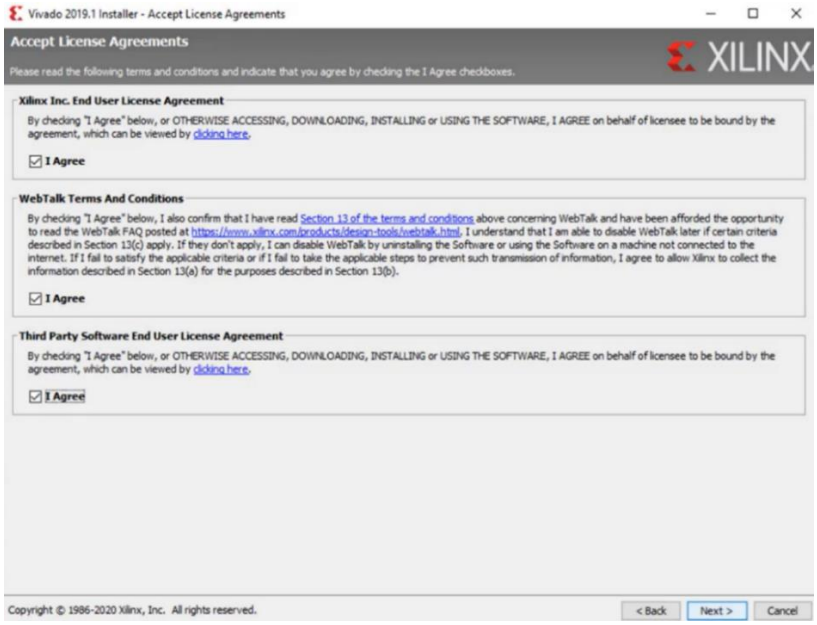
شکل ۲-۱ : فایل های دانلود شده

حال برای نصب نرم افزار روی پوشه ی ایجاد شده کلیک کرده و گزینه ی xstup.exe (آخرین فایل) را انتخاب می کنیم تا مراحل نصب شروع شوند. در اولین پنجره ی باز شده (شکل ۳-۱) گزینه ی next را انتخاب می کنیم.



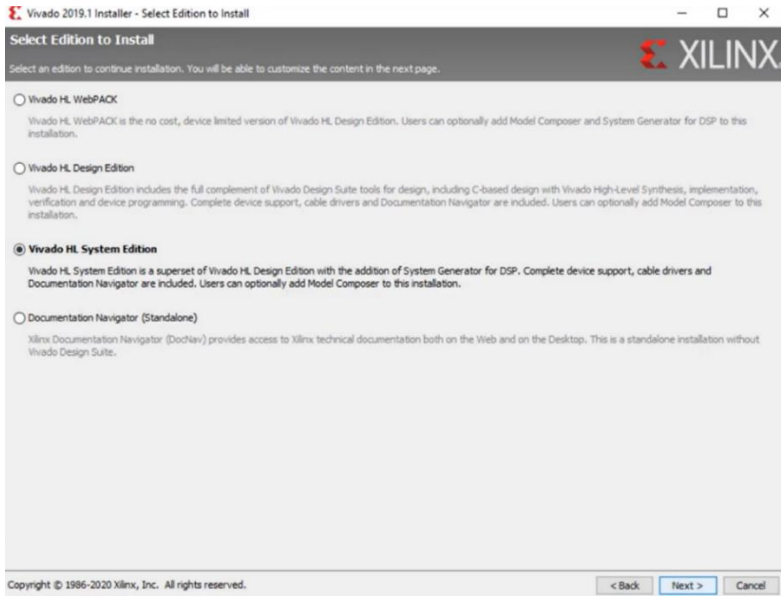
شکل ۳-۱ : مراحل نصب

در مرحله ی بعد برنامه مجوز های لازم را درخواست می کند؛که با انتخاب گزینه های مقابل I Agree تاییده های لازم داده می شود، ونهایتا گزینه ی next انتخاب می شود (شکل ٤-۱).



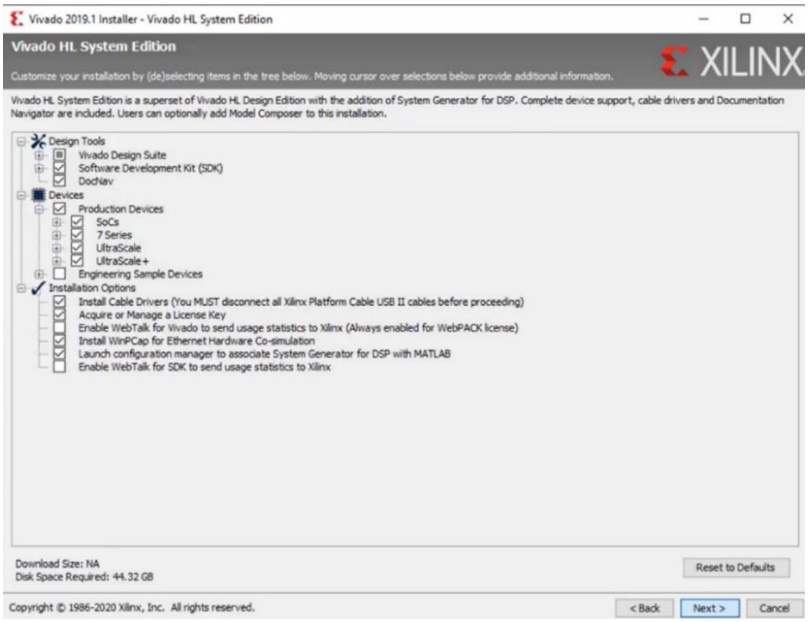
شکل ٤-١ : مراحل نصب

در صفحه ی بعدی که باز می شود، نسخه ی مورد نظر برای نصب را درخواست می کند؛ گزینه ی سوم کامل ترین نسخه می باشد که پیشنهاد می شود (شکل ٥-١).



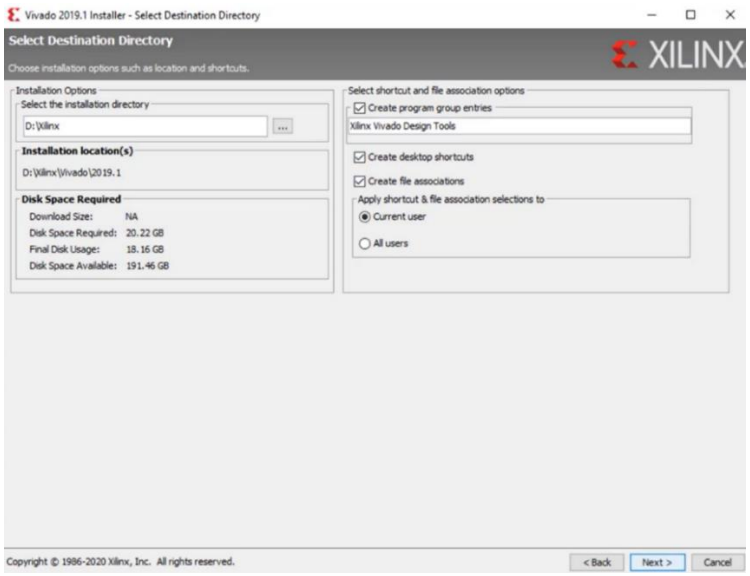
شکل ٥-١ : مراحل نصب

در پنجره ی بعدی که باز می شود، می توان تعدادی از امکانات نرم افزار را کم یا زیاد کرد؛ ترجیحا بدون تغییر گزینه ی next زد شود، ویا همانند شکل ٦-١ گزینه ها به حالت انتخاب در آید.



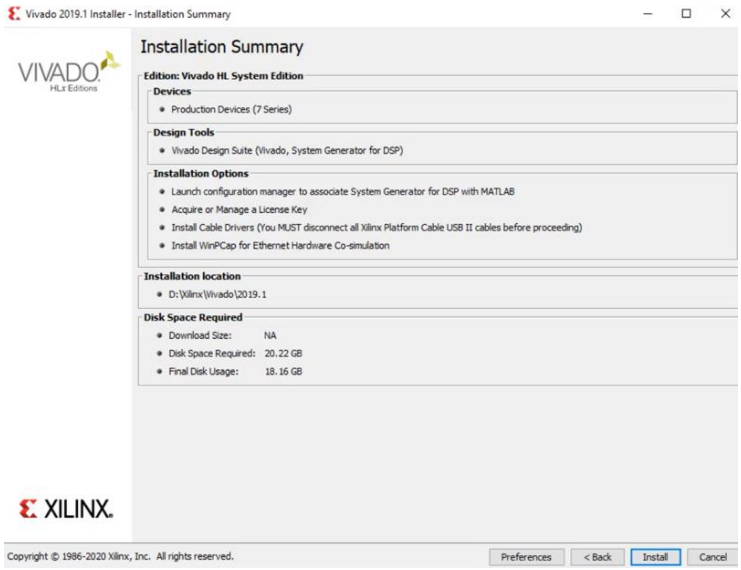
شکل ٦-١: مراحل نصب

در مرحله ی بعد؛ محل ذخیره سازی نرم افزار را می توان مشخص کرد، ونهایتا گزینه ی next را انتخاب کرد (شکل ٧-١).



شکل ٧-١ : مراحل نصب

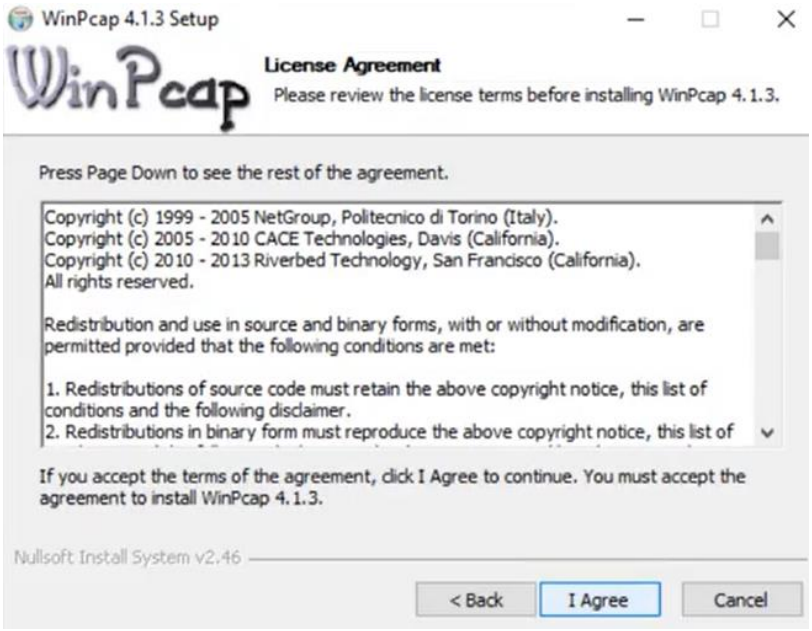
نهایتا در آخرین صفحه خلاصه ای از تنظیمات انجام گرفته آورده می شود. با انتخاب گزینه ی inistall نرم افزار شروع به نصب می کند (فرایند نصب با توجه به سیستم های مختلف زمان نسبتا زیادی را صرف می کند). (شکل ۸-۱).



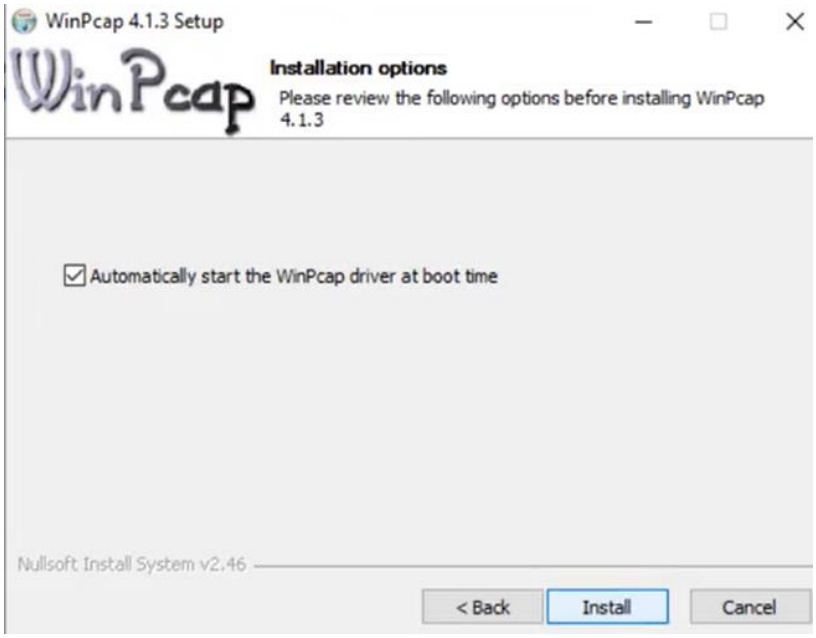
شکل ۸-۱ : مراحل نصب

در آخرین لحظات نصب نرم افزار، یک پیامی مبنی بر نصب نرم افزار دیگر آورده می شود؛ سه پنجره باز می شود که به ترتیب next وIAgree و نهایتا install را می زنیم. در نهایت بعد از نصب برنامه finish زده می شود. (شکل ۹-۱وشکل ۱۰-۱وشکل ۱۱-۱وشکل ۱۲-۱).

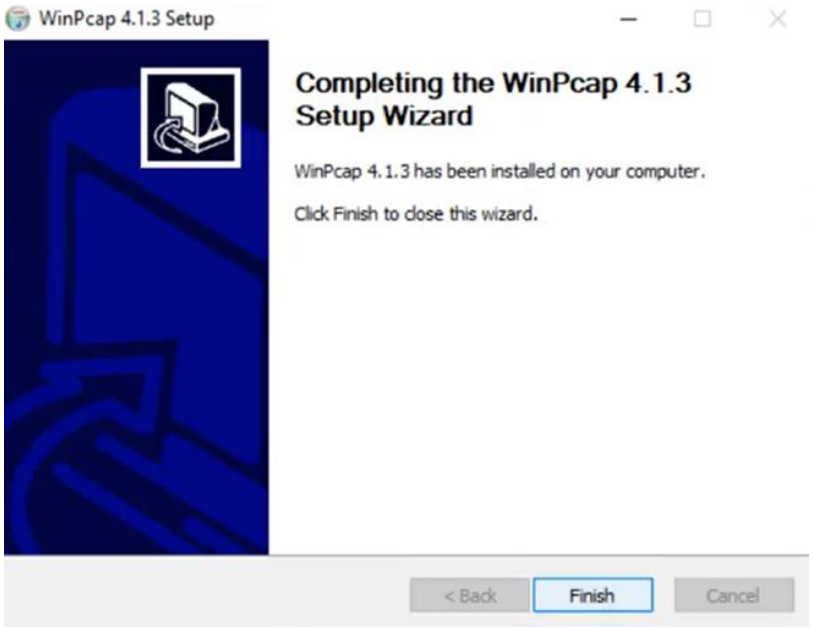
شکل ۹-۱ : مراحل نصب



شکل ۱۰-۱ : مراحل نصب

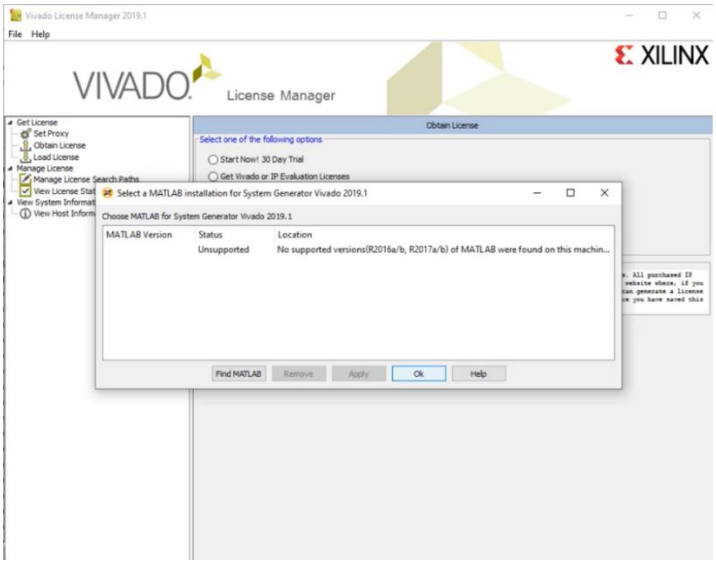


شکل ۱۱-۱ : مراحل نصب



شکل ۱۲-۱ : مراحل نصب

در ادامه ی فرایند نصب، پیامی همانند شکل ۱۳-۱ مشاهده می شود؛ که می توان در صورت وجود نرم افزار متلب بر روی سیستم آنرا به این نرم افزار متصل کرد، که ترجیحا از این موضوع صرف نظر می شود و گزینه ی ok (دو مرتبه) انتخاب می گردد.



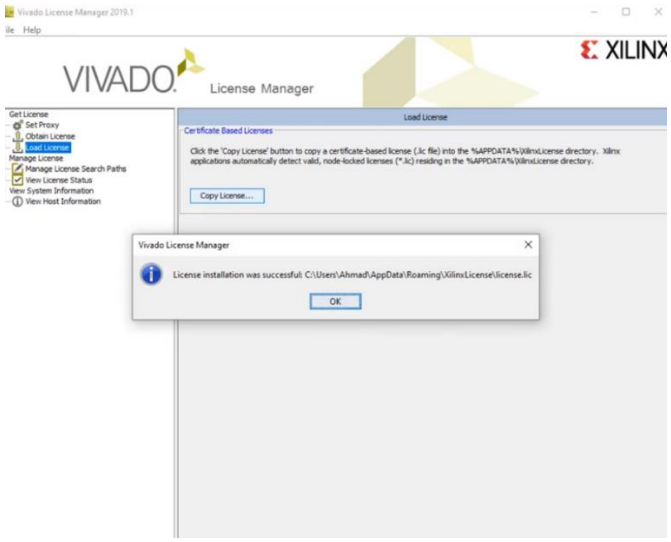
شکل ۱۳-۱ : پیام مربوط به ارتباط با متلب

در ادامه پنجره ای همانند شکل ۱۴-۱ باز می شود؛ که مربوط به لایسنس نرم افزار می باشد که باید به آن معرفی شود.

شکل ۱۴-۱ : معرفی لایسنس

برای معرفی لایسنس ابتدا گزینه load license را زده و سپس copy license را انتخاب می کنیم. از این قسمت و از پوشه ی license که در اولین مرحله به همراه فایل های نصبی از حالت فشرده، آزاد شده اند؛ فایل license را

انتخاب می کنیم. در صورت نصب درست لایسنس، پیامی همانند شکل ۱۵-۱ ظاهر می شود که گزینه ی ok را انتخاب می کنیم.



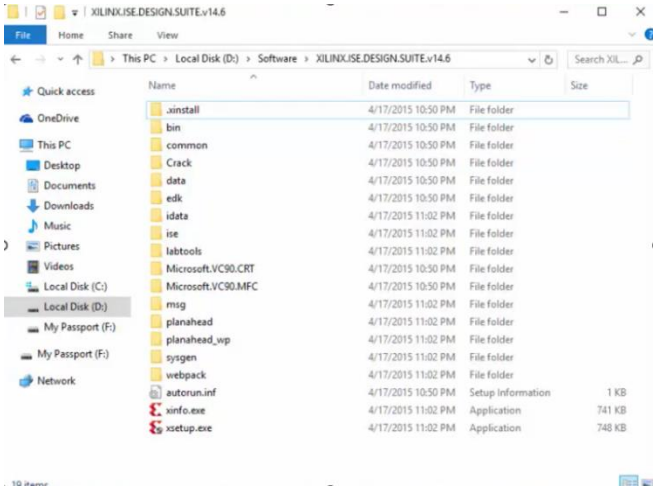
شکل ۱۵-۱ : معرفی لایسنس(اتمام)

در صورت انجام دقیق مراحل، نرم افزار vivado به درستی نصب خواهد شد.

۲-۲-۱- آشنایی و نصب نرم افزار ISE

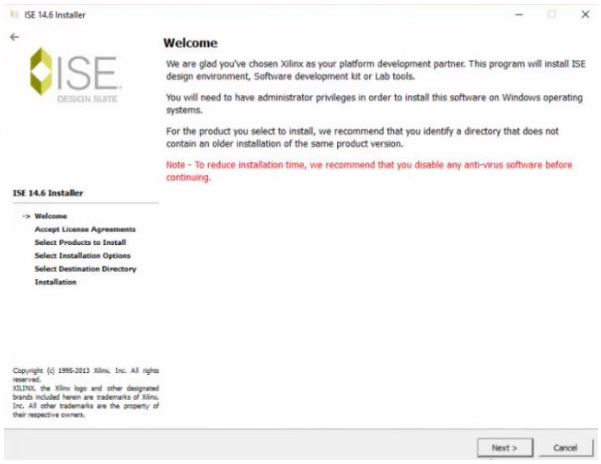
نرم‌افزار ISE Design Suite نرم‌افزار جامع شرکت XILINX است که به کمک آن می‌توانید تمام مراحل طراحی، پیاده‌سازی و تست مدار را انجام دهید. نرم افزار ISE برای بهینه سازی نیرو و هزینه، از طریق بهره وری طراحی بیشتر، تولید شده است است.

برای نصب نرم افزار، کافیست فایل های دانلود شده از حالت فشرده در آیند و گزینه ی xsetup.exe انتخاب شود(شکل ۱۶-۱).



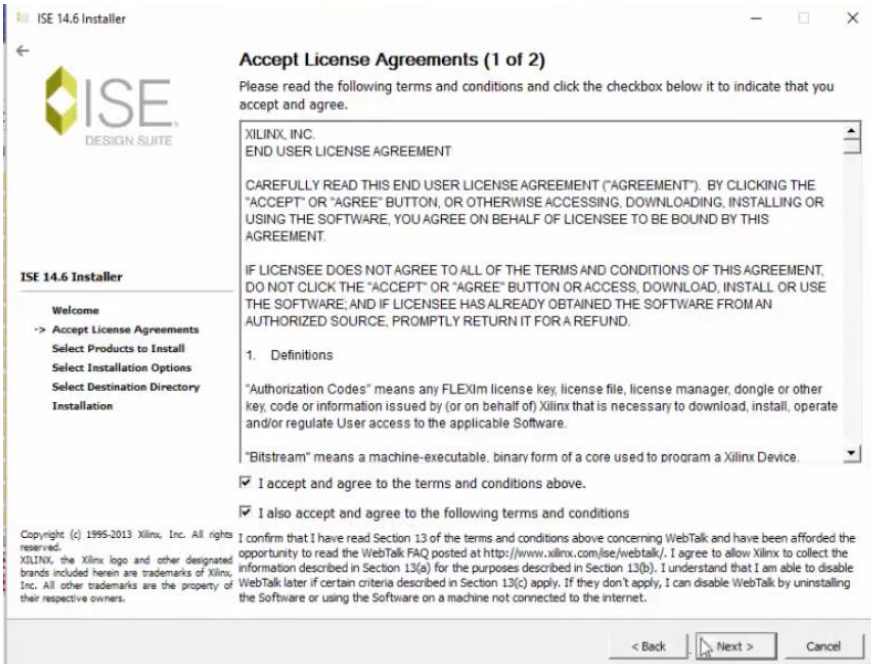
شکل ۱۶-۱ : فایل های نصبی

در پنجره ی باز شده گزینه ی next زده می شود(شکل ۱۷-۱).



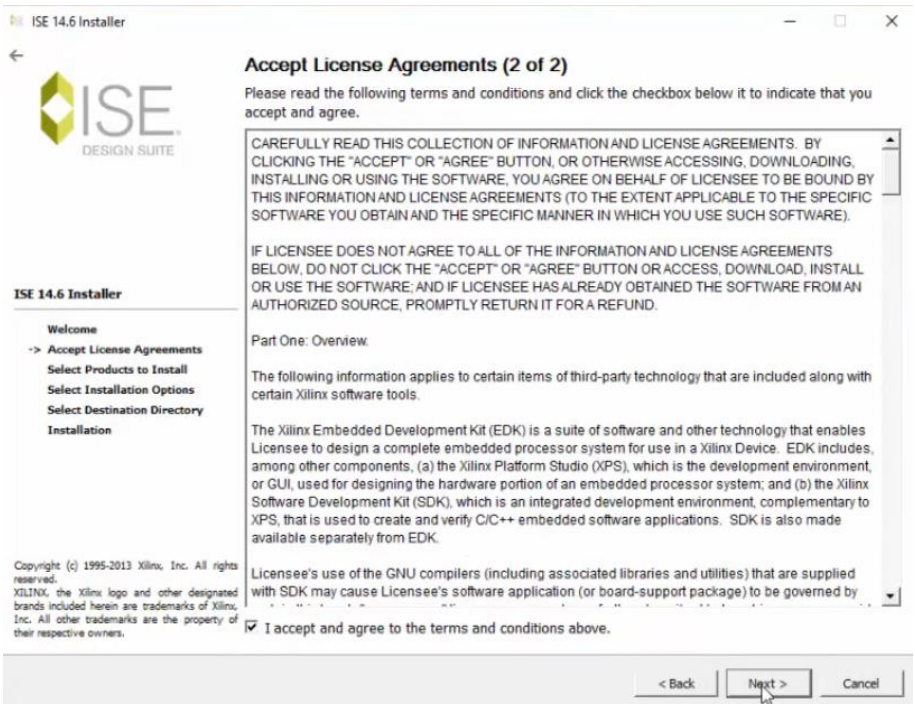
شکل ۱۷-۱ : مراحل نصب ISE

در پنجره ی باز شده دو گزینه ی مربوط به تایید برخی تعهدات می باشد(شکل ۱۸-۱ و شکل ۱۸-۱) با تیک زدن گزینه های مربوطه گزینه ی next را می زنیم.



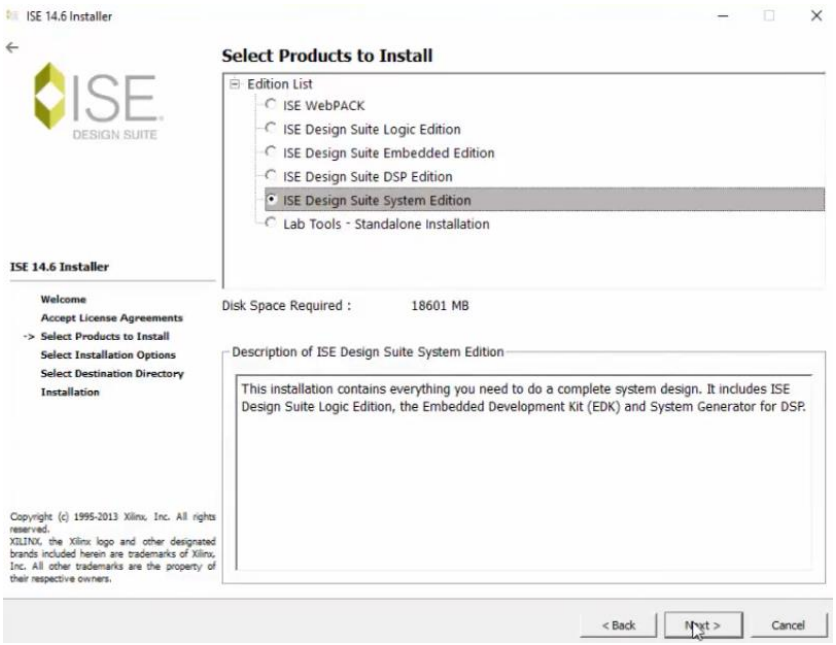
شکل ۱۸-۱ : مراحل نصب ISE

دوباره در پنجره ی جدید تیک مربوطه را زده و گزینه ی next را می زنیم(شکل ۱۹-۱).



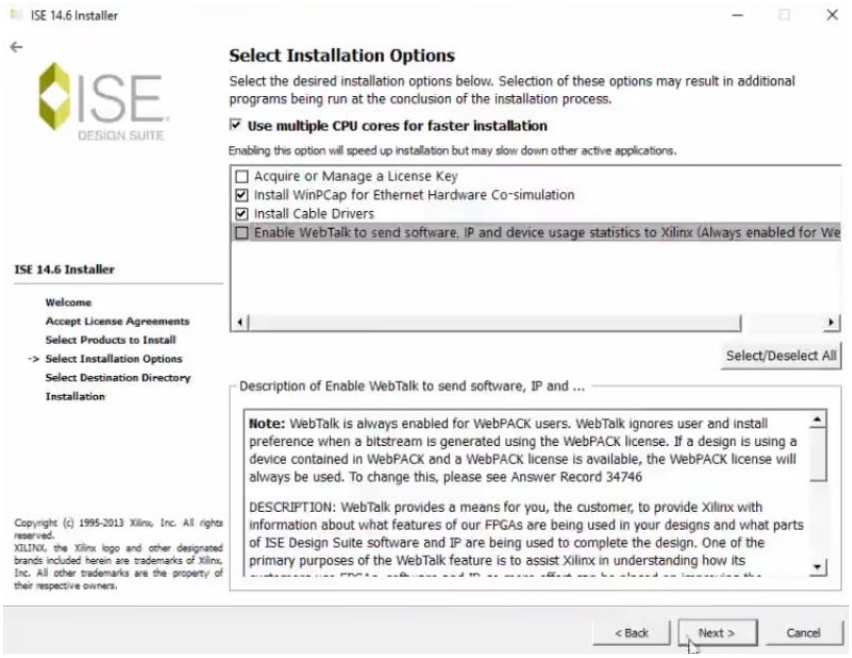
شکل ۱۹-۱ : مراحل نصب ISE

در پنجره ی جدید باز شده، امکان انتخاب نسخه های مختلف نرم افزار دیده می شود؛ که پیشنهاد می شود همانند شکل ۲۰-۱ گزینه ی پنجم را انتخاب کنید و next را انتخاب کنید.



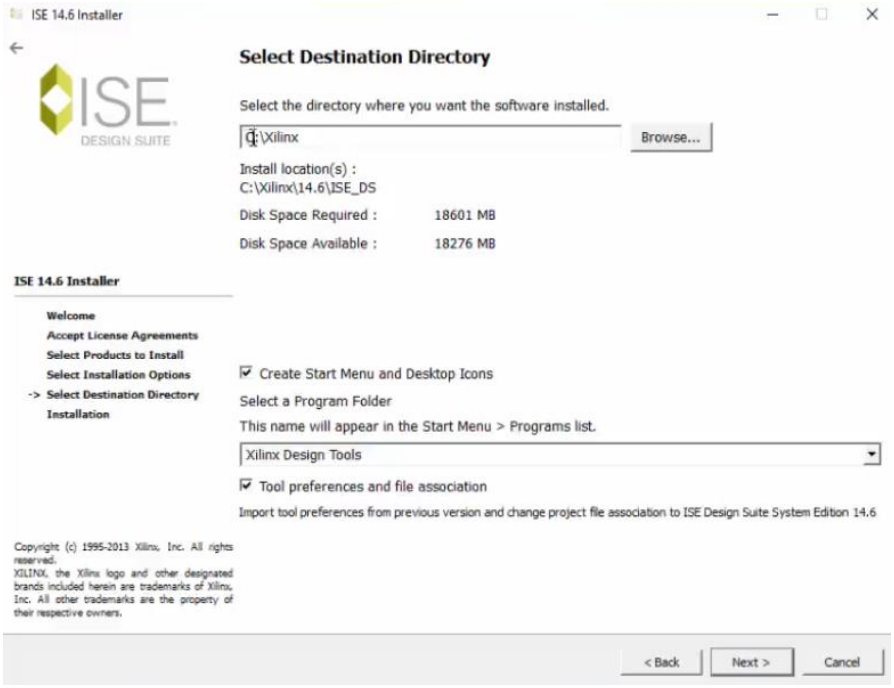
شکل ۲۰-۱ : مراحل نصب ISE (نسخه ی مناسب نصب)

درپنجره ی جدید باز شده همانند شکل ۲۱-۱ گزینه ها را انتخاب می کنیم و next را می زنیم.



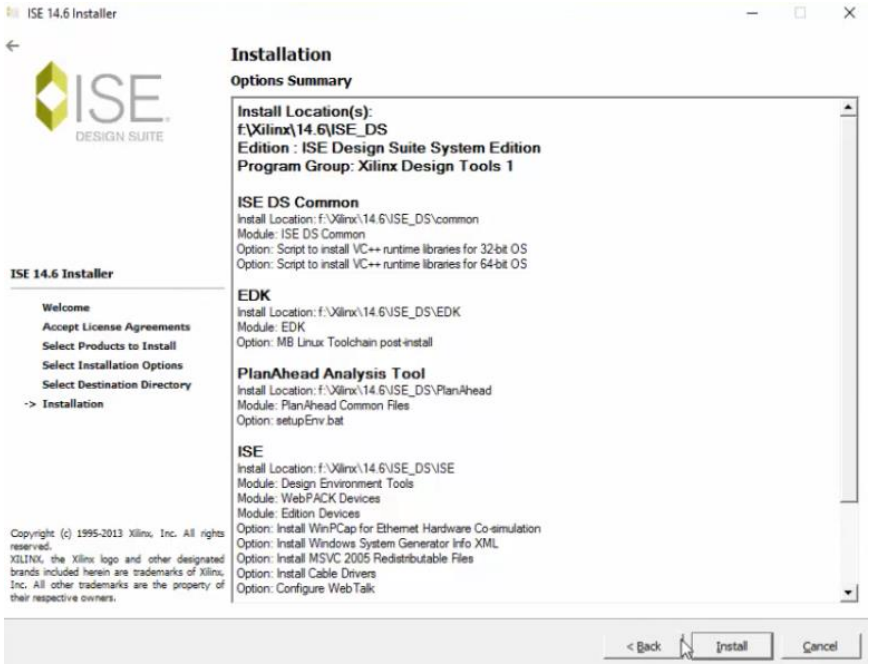
شکل ۲۱-۱ : مراحل نصب ISE

در پنجره ی جدید می توانید مسیر نصب نرم افزار را تغییر دهید و گزینه ی next را انتخاب کنید(شکل ۲۲-۱).



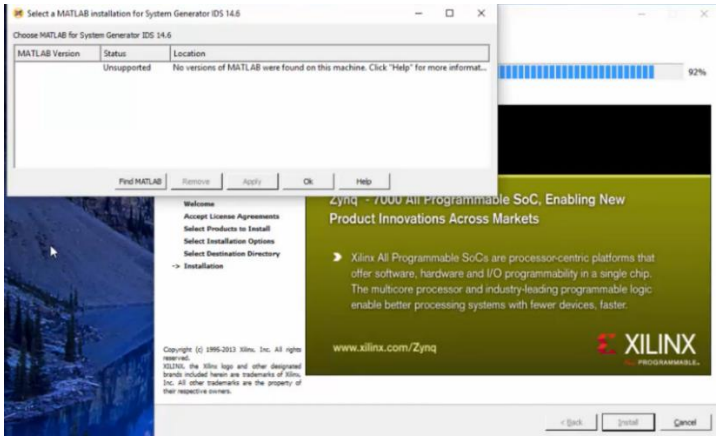
شکل ۲۲-۱ : مراحل نصب ISE (مسیر نصب)

و نهایتا همانند شکل ۲۳-۱ گزینه ی install را زده و فرایند نصب آغاز می شود.



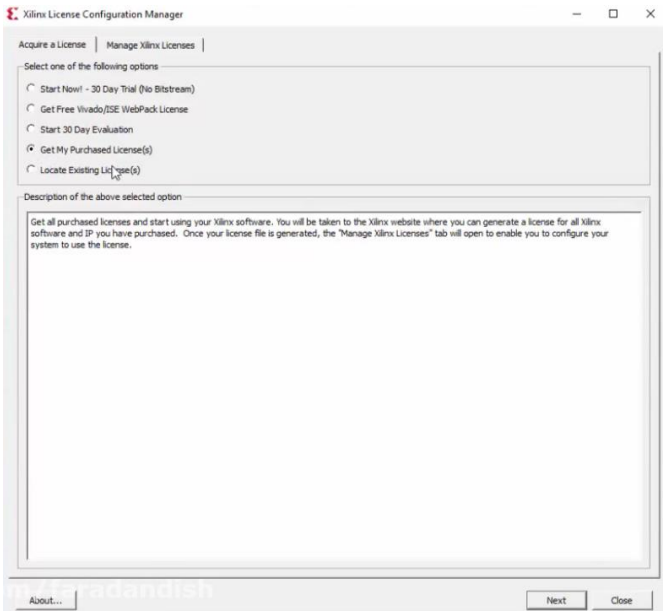
شکل ۲۳-۱ : مراحل نصب ISE

در اواخر نصب برنامه از شما درمورد ارتباط نرم افزار با متلب سوال می پرسد که فعلا نیازی به آن نیست و گزینه ی ok را می زنیم (شکل ۲۴-۱).

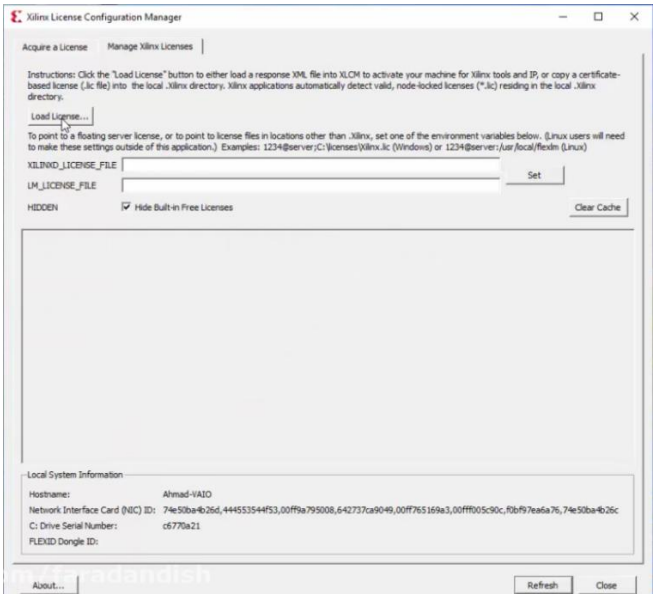


شکل ۲۴-۱ : مراحل نصب ISE (ارتباط با متلب)

بعد از اتمام نصب برنامه؛ پنجره ای باز می شود که گزینه ی finish را می زنیم. در ادامه همانند شکل ۲۵-۱ پنجره ای باز می شود که مربوط به لایسنس برنامه می باشد. ابتدا همانند شکل ۲۵-۱ گزینه ی Get My Purchased License(s) را انتخاب کرده و سپس روی تب manage Xilinx Licenses زده و همانند شکل ۲۶-۱ از گزینه ی load licenses فایل لایسنس که در همان مسیر دانلود نرم افزار می باشد را انتخاب می کنیم.



شکل ۲۵-۱ : مراحل نصب ISE (مراحل لایسنس)



شکل ۲۶-۱ : مراحل نصب ISE (انتخاب لایسنس)

نهایتا close را زده ونصب و کرک نرم افزار تمام می شود. و می توان از آن استفاده کرد.

ممکن هست که بعد از نصب یا در هنگام کرک کردن نرم افزار، برنامه خود به خود بسته شود؛ می توانید همانند زیر عمل کنید (شکل ۲۷-۱) .

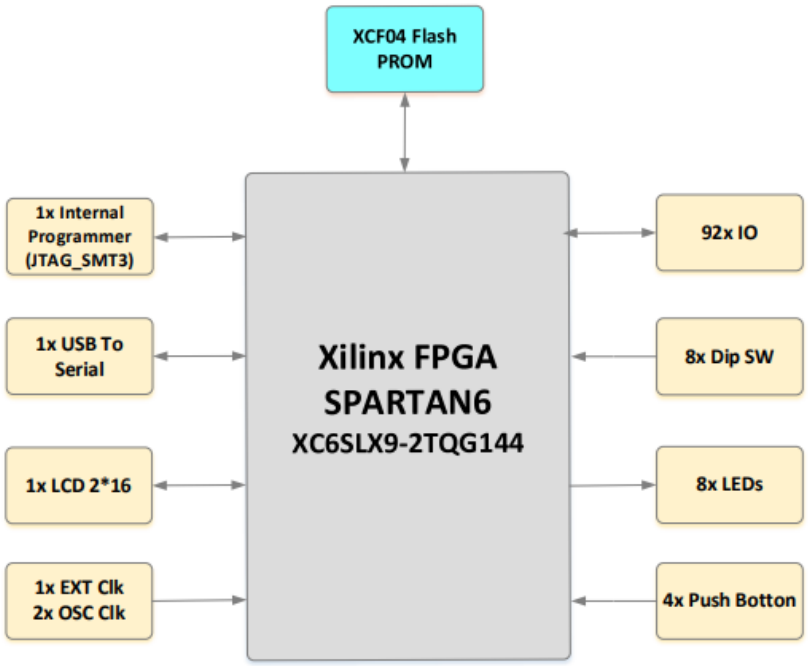
1. ابتدا نرم افزار ISE Suit رو نصب کنید.
2. در محلی که نرم افزار رو نصب کردین به آدرس Xilinx\14.7\ISE_DS\ISE\lib\nt64 برید.
3. رو پیدا کنید و اسمشو به libPortability.dll تغییر بدهید libPortability.dll.orig.
4. یک کپی از libPortabilityNOSH.dll بگیرید و اسم آن را به libPortability.dll تغییر دهید و در همان پوشه paste کنین.
5. مجدداً یک کپی از libPortabilityNOSH.dll بگیرید ولی این بار در آدرس Xilinx\14.7\ISE_DS\common\lib\nt64 آن را Paste کنید.
6. به آدرس Xilinx\14.7\ISE_DS\common\lib\nt64 بروید و libPortability.dll را پیدا کرده و به libPortability.dll.orig تغییر نامش دهید.
7. در همین آدرس libPortabilityNOSH.dll را به libPortability.dll تغییر نام دهید.

شکل ۲۷-۱ : راه حل بسته شدن خودکار ISE

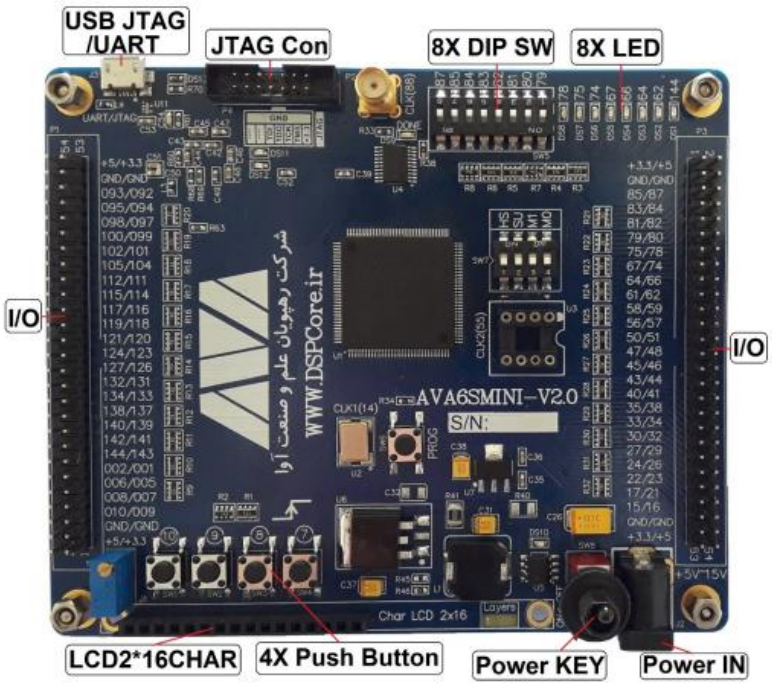
۱-۳-۱- آشنایی با برد آموزشی AVA٦SMINI

برد AVA٦SMINI به منظور شروع به کار با FPGA و یا انجام پروژه های کوچک با قیمت کم ارائه شده است.
-۳-۱
این برد بر اساس تراشه ی XC٦SLX٩ از خانواده Spartan٦ شرکت XILINX است. به منظور عملکرد خوب در سرعت های بالا برد به صورت ٤ لایه طراحی شده است و تمامی پایه های IO آن در دسترس است.

بلوک دیاگرام و نمای کلی برد در شکل ۲۸-۱ و شکل ۲۹-۱ آورده شده است.



شکل ۲۸-۱ : بلوک دیاگرام AVA٦SMINI



شکل ۲۹-۱ : نمای کلی برد AVA٦SMINI

برخی از مشخصات فنی برد AVA٦SMINI در ادامه آمده است:

- دارای تراشه ی XC٦SLX٩ از خانواده SPARTAN٦
- دارای حافظه XCF٠٤S برای برنامه ریزی FPGA
- دارای پروگرامر پرسرعت JTAG SMT٣ بصورت پورت USB است که نیاز به پروگرامر خارجی را برطرف می کند.
- دو عدد کانکتور ٢x٢٧ ،جهت در دسترس بودن پایه های FPGA و اتصال ماژول
- در دسترس بودن ولتاژهای ٣.٣V و ٥V بر روی کانکتورهای ٢x٢٧ برای تغذیه ی ماژول.
- دارای کانکتور SMA برای کلاک خارجی
- دارای پورت USB به سریال که امکان استفاده به صورت پورت COM برای وسایل مانند لپ تاپ که

پورت سریال ندارند را فراهم می کند.

• پورت LCD ۲*۱۶

• یک عدد DIP SWITCH هشت تایی جهت اعمال DATA به تراشه

• چهار عدد کلید فشاری جهت اعمال DATA (پالس) به تراشه

• هشت عدد LED جهت نمایش داده های خروجی

• اسیلاتور ۵۰Mhz روی برد

• کانکتور اسیلاتور اضافی برای نصب اسیلاتور دوم برای استفاده خاص کاربر

•**ابعاد** : ۱۰cm x ۱۲cm

در استفاده از این برد نکاتی وجود دارد که عدم توجه به آنها می تواند باعث آسیب به برد شود؛ و یا نتایج مطلوبی از آن به دست نیاید.

• ولتاژ اعمالی به پورتها نباید از ۳.۳ ولت بیشتر باشد، در غیر اینصورت FPGA معیوب خواهد شد.

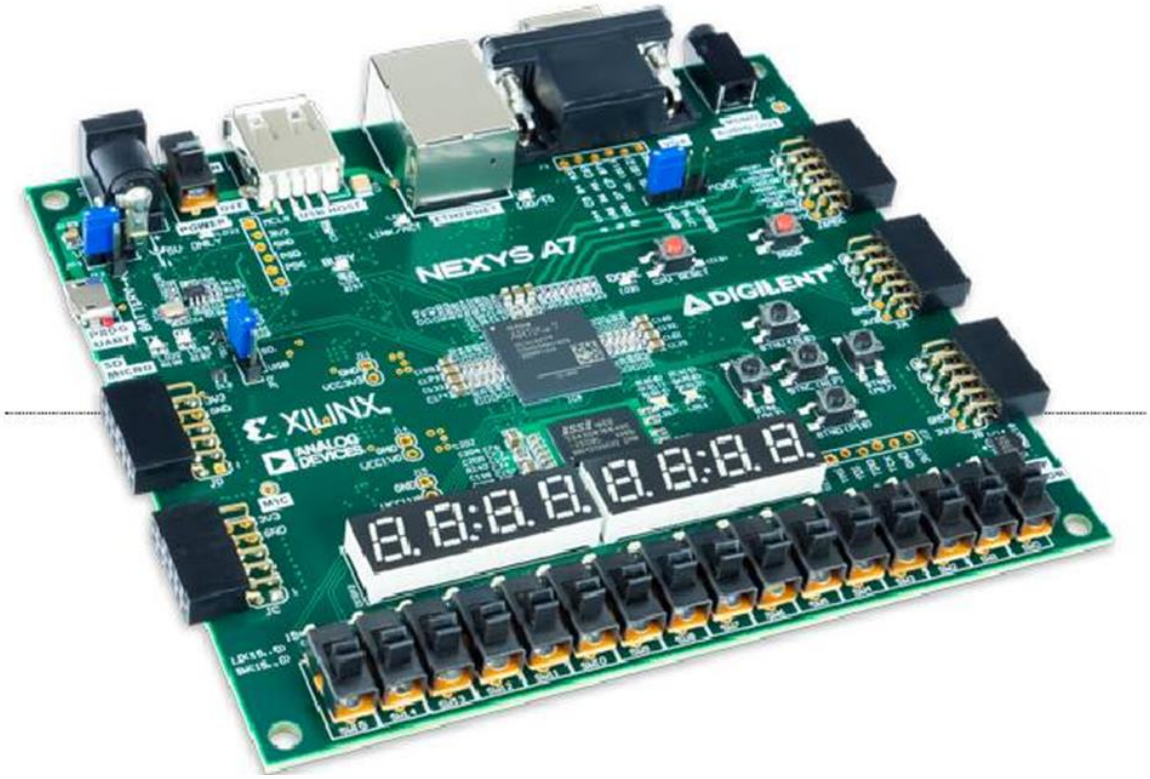
• برای تغذیه برد حتی الامکان از آداپتور همراه با برد استفاده شود، در صورت استفاده از تغذیه ی دیگر از یک منبع تغذیه DC در محدوده ۷ تا ۱۲ ولت استفاده نمایید.

• حتی الامکان از پروگرامر داخلی برد استفاده نمایید و در صورتی که از پروگرامرهای متفرقه استفاده می کنید، ابتدا پایه های Jtag را بررسی و از استاندارد بودن آن اطمینان حاصل نمایید و در صورت نیاز پایه ها را به صورت دستی تطبیق دهید.

۱-۳-۲- آشنایی با برد NEXYS A۷™ FPGA BOARD REFERENCE MANUAL

برد Nexys A۷ یک پلت فرم توسعه مدار دیجیتال کامل و آماده برای استفاده است. برد Nexys A۷ دارای یک FPGA بزرگ با ظرفیت بالا، حافظه ی خارجی(external memories) و مجموعه پورت‌های USB، Ethernet و سایر پورت‌ها می باشد که می تواند پکیج کاملی برای طراحی و پیاده سازی مدار های ترکیبی از ساده تا پیچیده باشد.

این برد همچنین دارای چندین دستگاه جانبی داخلی، از جمله شتاب‌سنج، سنسور دما، میکروفون دیجیتال MEM، تقویت‌کننده بلندگو و چندین دستگاه ورودی/ خروجی می باشد که به Nexys A۷ اجازه می‌دهند تا برای طیف وسیعی از طراحی‌ها بدون نیاز به اجزای دیگر استفاده شود.

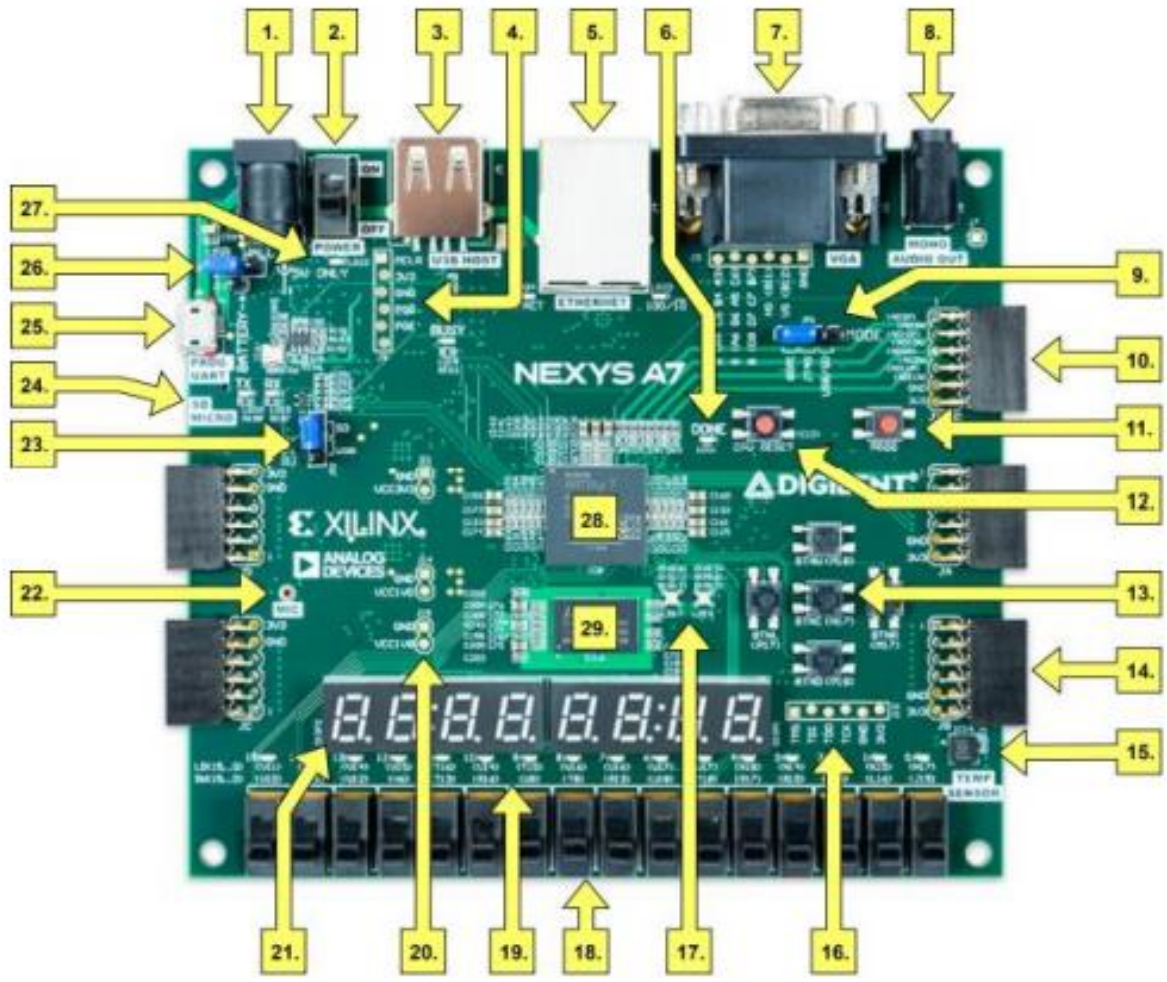


شکل ۳۰-۱ : **برد** Nexys A۷ FPGA

جدول ۱-۱ : مشخصات فنی Artux-۷ FPGA

Product Variant	Nexys A۷-۱۰۰T
FPGA Part Number	XC۷A۱۰۰T-۱CSG۳۲٤C
Look-up Tables (LUTs)	٦۳,٤۰۰
Flip-Flops	۱۲٦,۸۰۰
Block RAM	۱,۱۸۸ Kb
DSP Slices	۲٤۰
Clock Management Tiles	٦

بخش های مختلف Artux-۷ FPGA به صورت شماتیک در شکل ۳۰-۱ آورده شده است.



شکل ۳۱-۱ : بخش های مختلف Artux-۷ FPGA

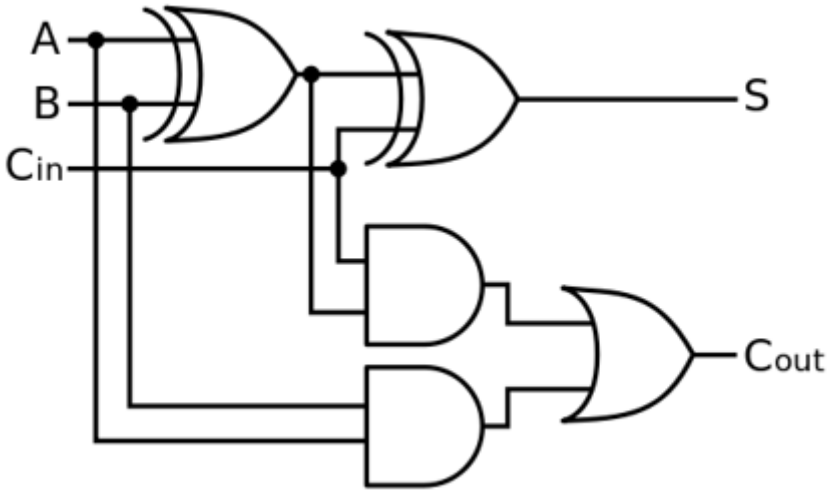
جدول ۱-۲ : مشخصات Artux-۷ FPGA

بخش	شرح قطعه	بخش	شرح قطعه
۱	(Power jack)	۱٦	JTAG port for (optional) external cable
۲	(Power switch)	۱۷	Tri-color (RGB) LEDs
۳	USB host connector	۱۸	Slide switches (۱٦)
٤	PIC۲٤ programming port (factory use)	۱۹	LEDs (۱٦)
٥	Ethernet connector	۲۰	Power supply test point(s)
٦	FPGA programming done LED	۲۱	Eight digit ۷-seg display
۷	VGA connector	۲۲	Microphone
۸	Audio connector	۲۳	External configuration jumper (SD / USB)
۹	Programming mode jumper	۲٤	MicroSD card slot
۱۰	Analog signal Pmod port (XADC)	۲٥	Shared UART/ JTAG USB port
۱۱	FPGA configuration reset button	۲٦	Power select jumper and battery header
۱۲	CPU reset button (for soft cores)	۲۷	Power-good LED
۱۳	Five pushbuttons	۲۸	Xilinx Artix-۷ FPGA
۱٤	Pmod port(s)	۲۹	DDR۲ memory
۱٥	Temperature sensor		

آزمایش دوم: مدارات ترکیبی

در این آزمایش سعی داریم یکی از مدارات ترکیبی را با استفاده از نرم افزار ISE به صورت سخت افزاری برنامه ریزی کنیم و با پروگرم کردن کد ها روی برد موجود عملکرد مدار را برسی کنیم.

می خواهیم مدار ترکیبی Full adder را روی برد پروگرم کنیم. مدار full adder همانند شکل ۱-۲ می باشد.

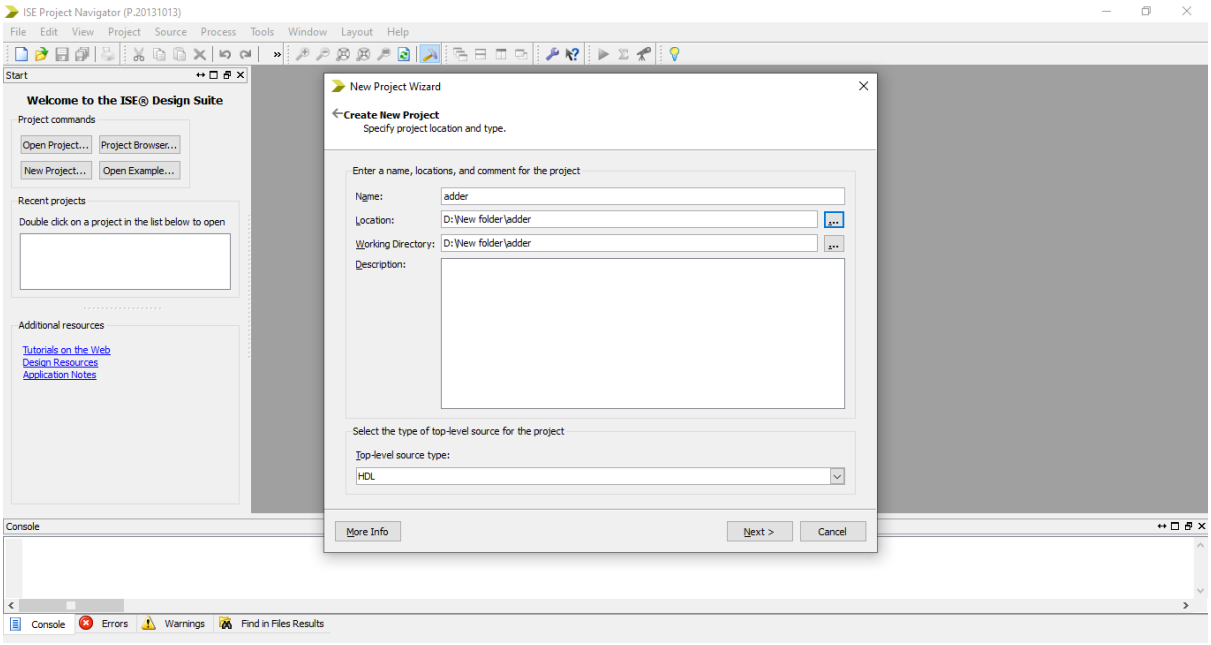


شکل ۱-۲ : مدار ترکیبی FULL ADDER

نرم افزار ISE را اجرا می کنیم.

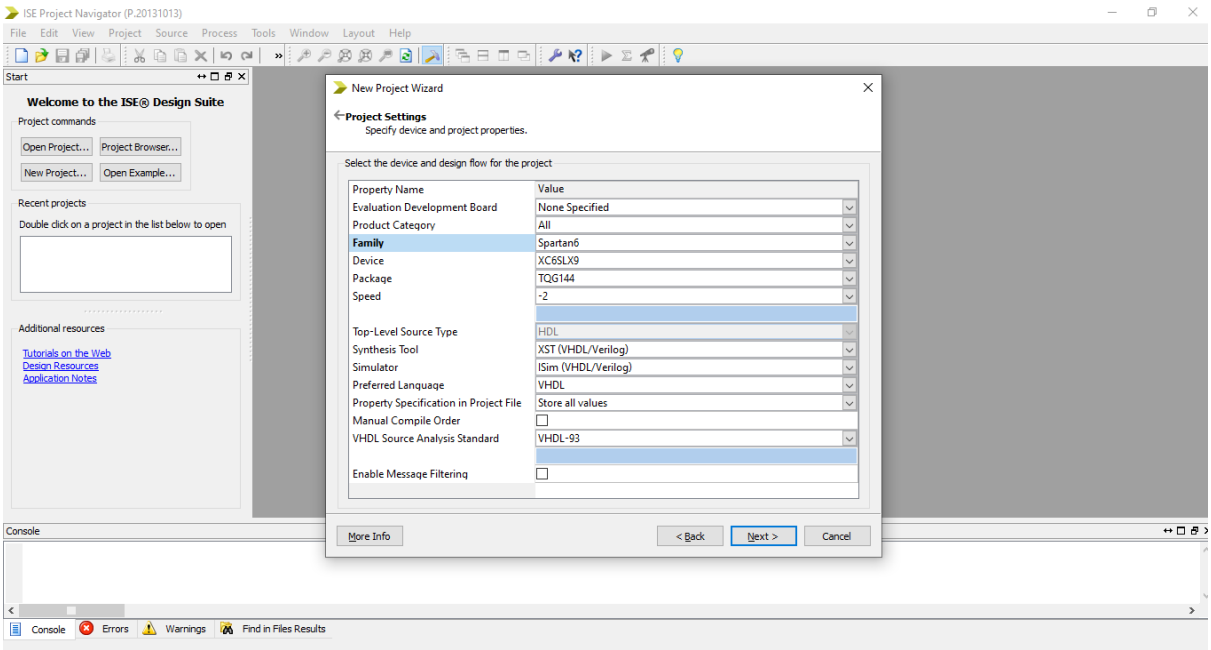
از تب file گزینه ی new project را می زنیم. و همانند شکل ۲-۲ اسم پروژه ومسیر نصب را مشخص می کنیم.

توجه داشته باشید که گزینه ی آخر رویHDL باشد.



شکل ۲-۲ : تنظیمات ایجاد پروژه

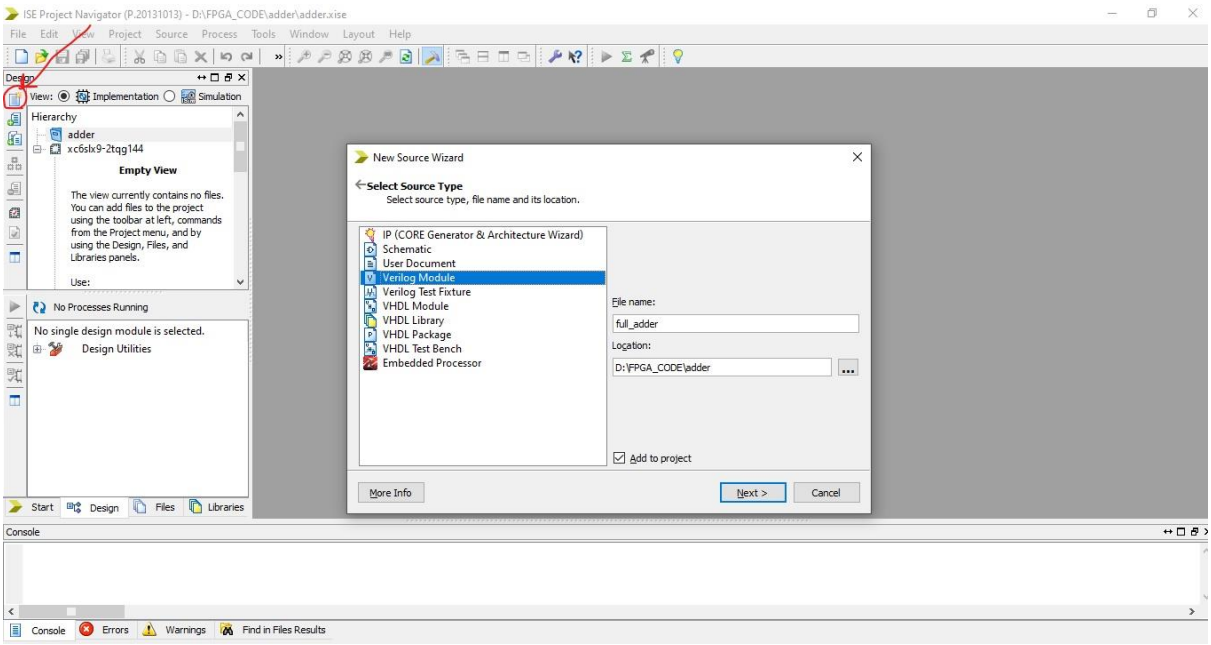
در صفحه ی جدیده باز شده همانندشکل ۳-۲ مشخصات برد خود را وارد کرده و next را می زنیم.



شکل ۳-۲ : تنظیمات برد

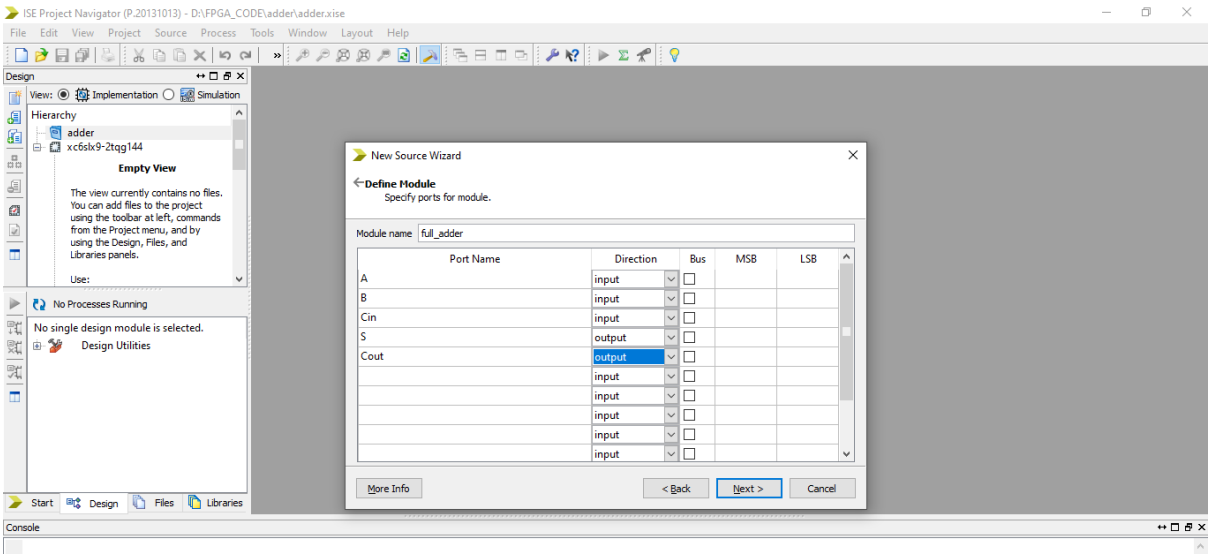
در پنجره ی بعدی خلاصه ای از تنظیماتی را که انجام داده اید را نشان می دهد.گزینه ی finish را می زنیم.

حال همانند شکل ٤-٢ source جدید ایجاد می کنیم. و همانند سورس تنظیمات مربوطه و اسم سورس را وارد می کنیم. سعی کنید از اسم های با معنی استفاده کنید.



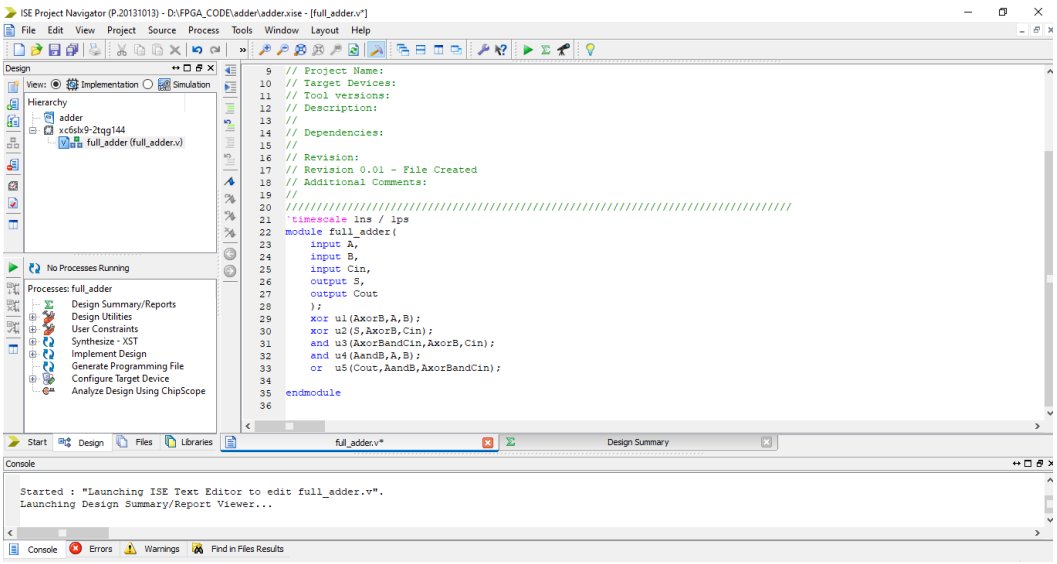
شکل ٤-٢ : ایجاد SOURCE

در پنجره ی بعد می توانید ورودی و خروجی ها را همانند شکل ٥-٢ مشخص کنید.



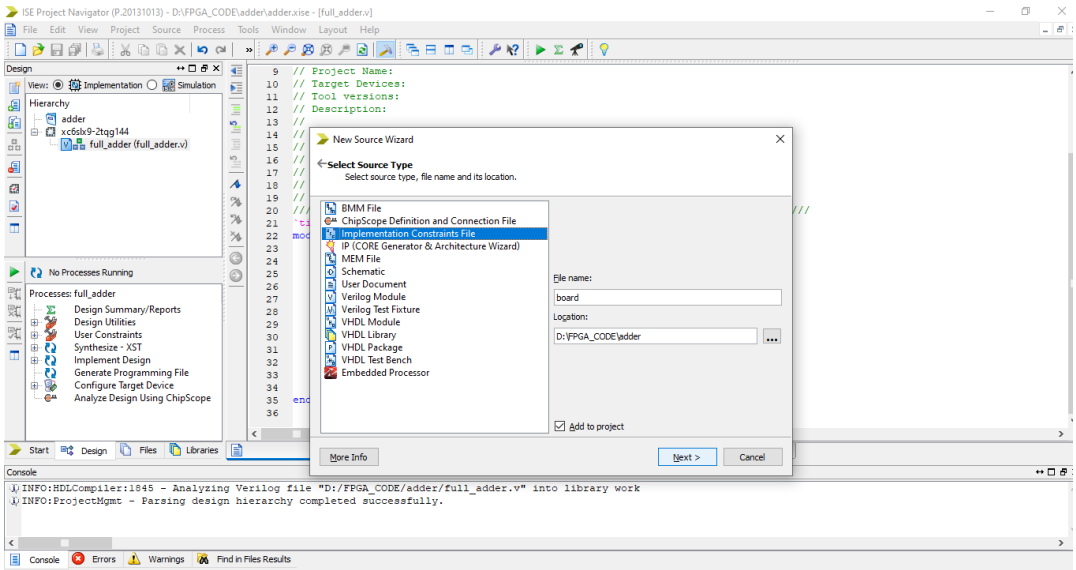
شکل ٥-٢ : تنظیمات ورودی خروجی

در ادامه خلاصه ای از تنظیمات انجام شده آورده می شود که finish را می زنیم؛ و وارد محیط برنامه نویسی می شویم. همانند شکل ٦-٢ کد های مربوطه را می نویسیم. در نوشتن کد ها از گذاشتن فاصله بپردازید.



شکل ٦-٢ : محیط برنامه نویسی

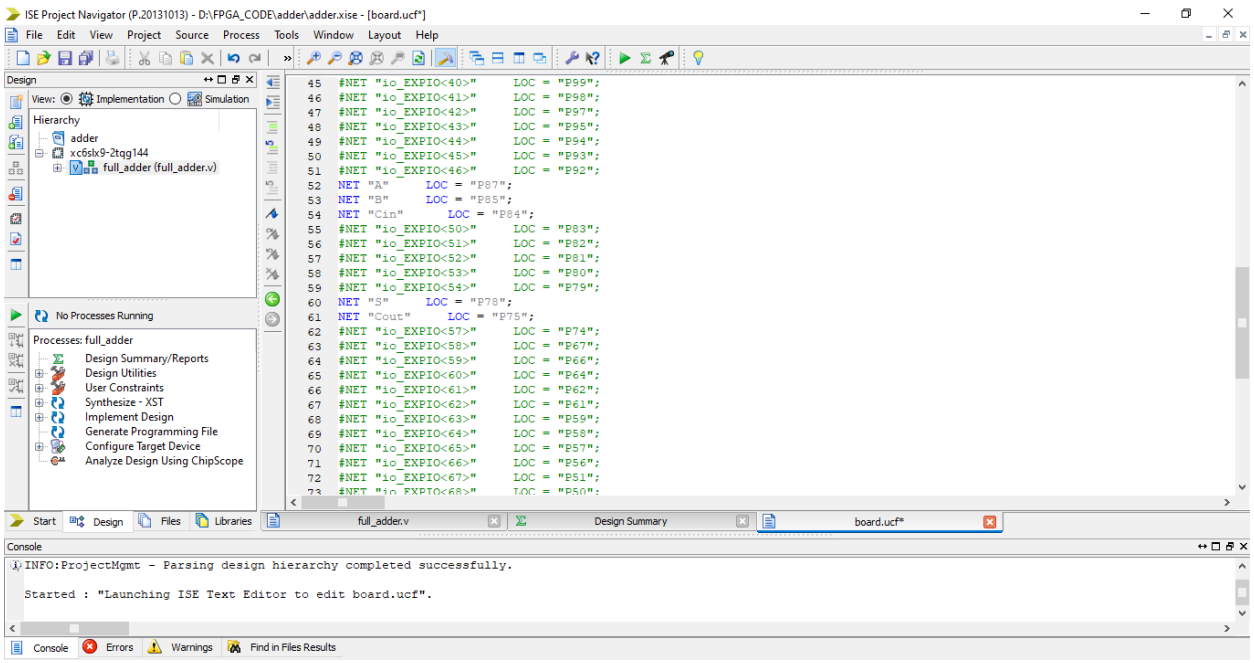
در مرحله ی بعد برای مشخص کردن ورودی خروجی های روی برد از فایل AVA٦SMINI.ucf مربوط به برد استفاده می کنید. (می توانید ازDVD های همراه برد پیدا کنید و یا از استاد خود فایل مربوطه را بخواهید). حال یک source جدید همانند شکل ٧-٢ ایجاد می کنیم.



شکل ٧-٢ : ایجاد SOURCE

با تنظیمات فوق next و سپس finish را می زنیم. محتویات داخل AVA٦SMINI.ucf را همانند شکل ۸–۲ در

پنجره ی باز شده کپی می کنیم؛ و پایه های مورد نظر را همانند شکل تعریف می کنیم.



شکل ۸–۲ : تنظیمات پایه ها

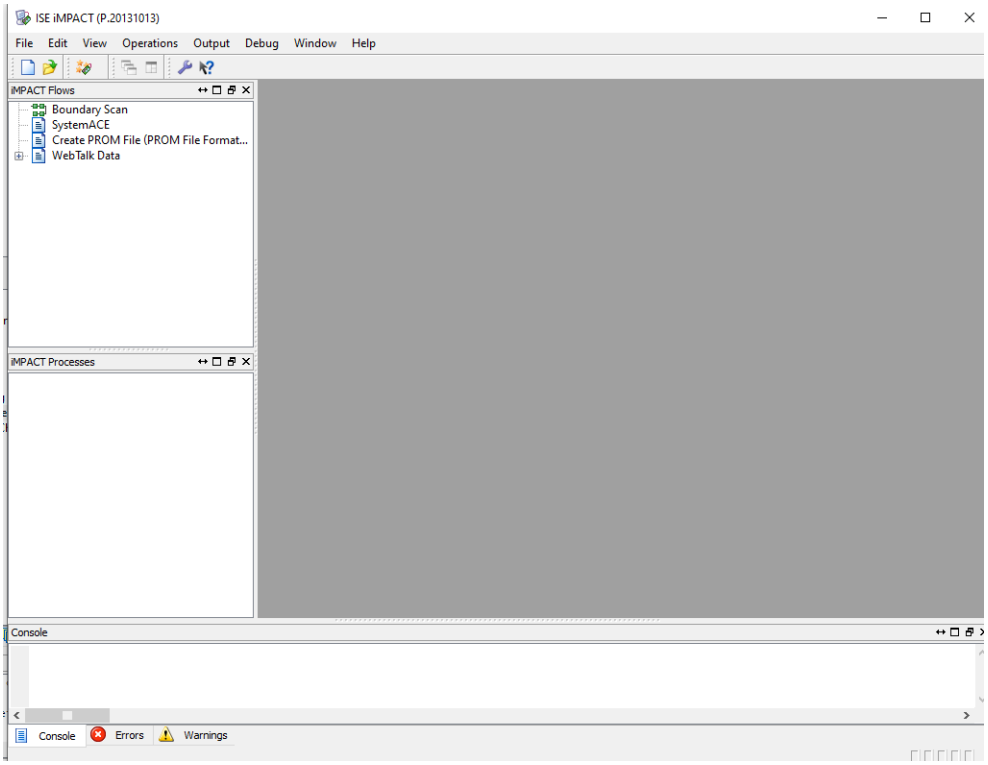
در ادامه برای پروگرم کردن روی برد؛ برد را به کامپیوتر متصل می کنیم. و روی مثلث سبز سمت چپ کلید می

کنیم.

اگر بدون ارور تمام شود؛ لازم است فایل bit. مخصوص برای پروگرم را ایجاد کنیم. برای این کار ابتدا Generate

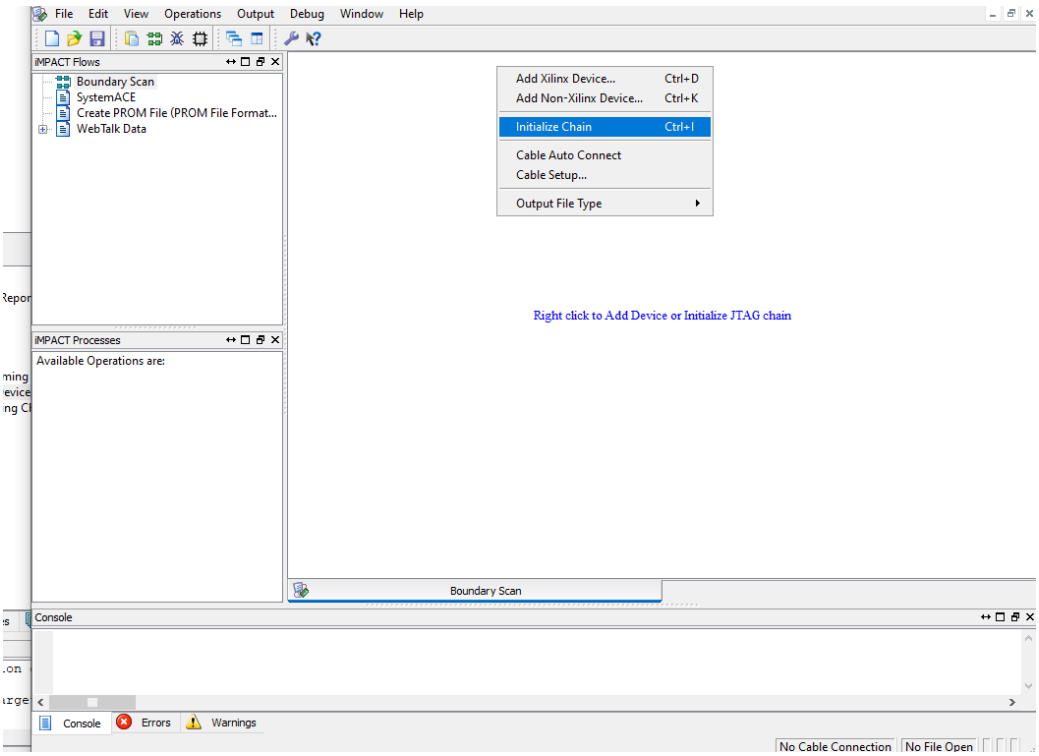
Programming را می زنیم. بعد از اتمام آن Configure Target Device و سپس ok را می زنیم. درپنجره ی باز

شده همانند شکل ۹–۲ گزینه ی اول (Boundry Scan) را انتخاب می کنیم.



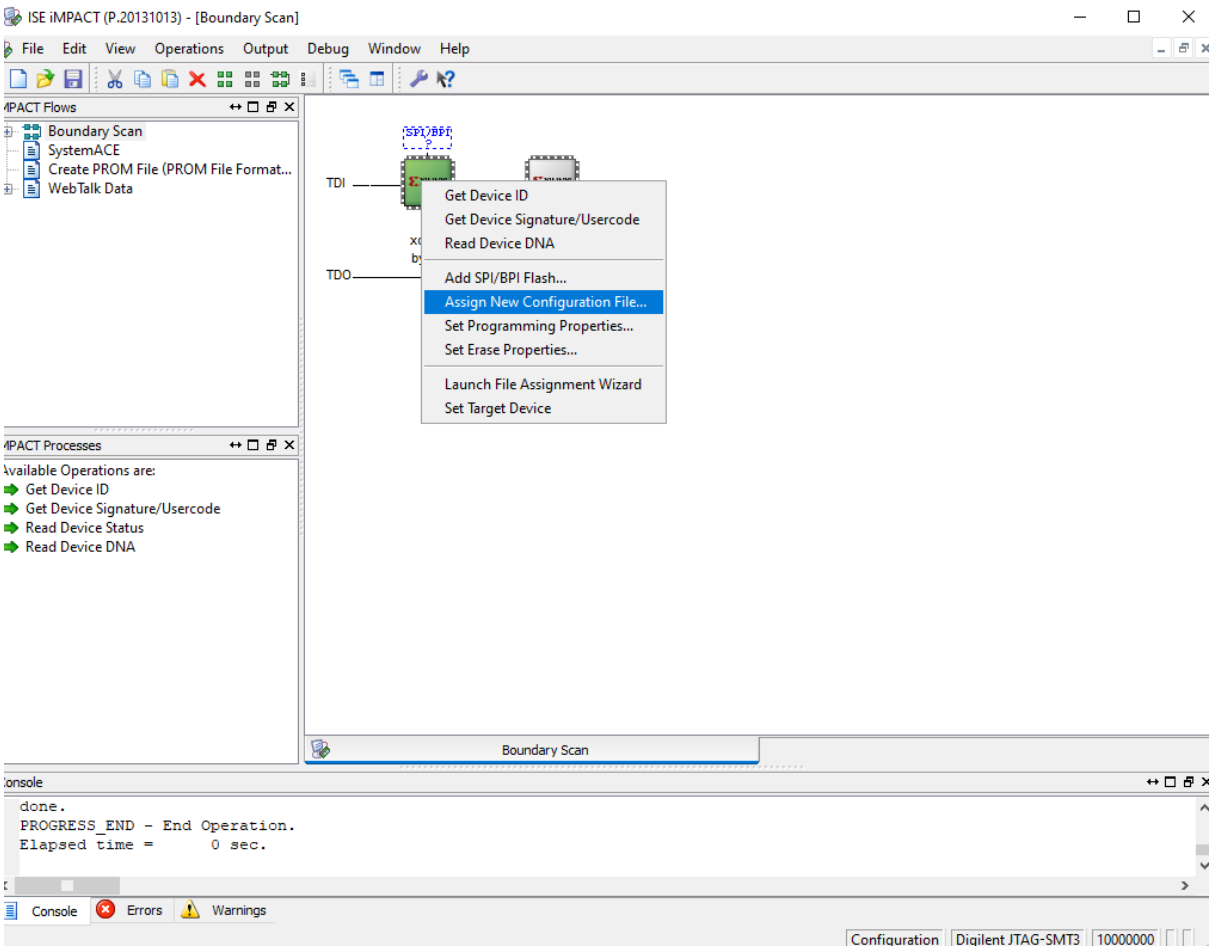
شکل ۹-۲ : ایجاد فایل BIT.

در پنجره ی باز شده کلیک چپ کرده وinitialize chain را می زنیم.(شکل ۱۰-۲)



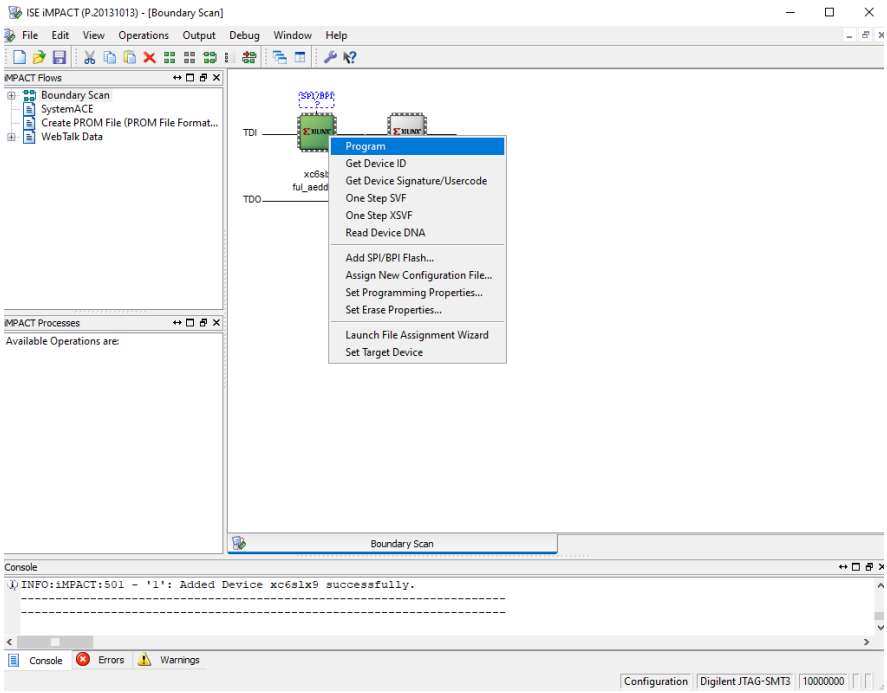
شکل ۱۰-۲

در ادامه no و سپس cancel را می زنیم. روی هسته ی اصلی راست کلیک کرده و گزینه ی Assign New Configuration File … را می زنیم(شکل ۱۱-۲). در مسیر ذخیره ی پروژه فایلbit. که ایجاد شده را انتخاب می کنیم.



شکل ۱۱-۲ : انتخاب فایل BIT.

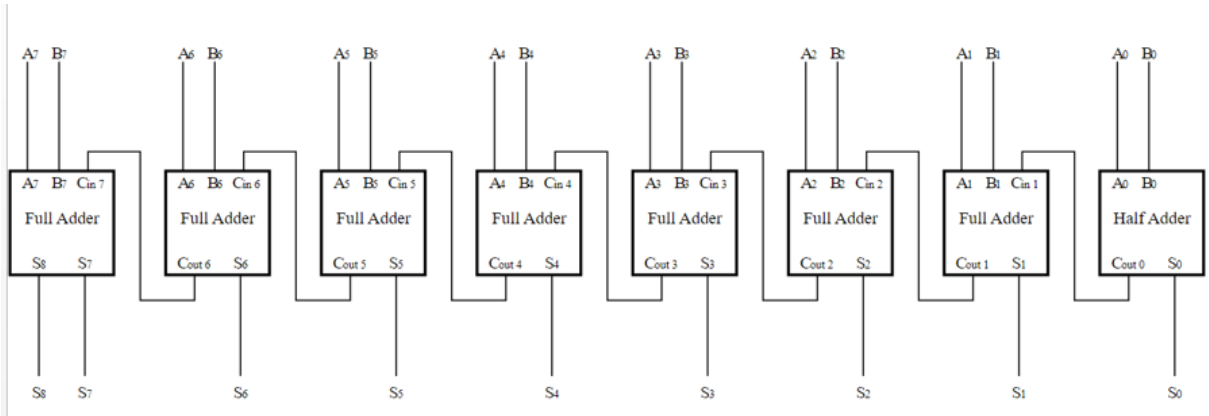
بعد انتخاب bit. در ادامه پنجره ی باز شده no را می زنیم. دوباره روی هسته ی اصلی راست کلیک کرده و همانند شکل ۱۲-۲ Program را انتخاب می کنیم. و نهایتا ok را می زنیم.



شکل ۱۲-۲ : مراحل پروگرم

اگر همه ی مراحل به درستی انجام داده شده باشد؛ برنامه روی برد پروگرم می شود و می توان با اعمال ورودی ها خروجی ها را مشاهده کرد.

تمرین: با استفاده از برنامه ی نوشته شده یک جمع کننده ی ۸ بیتی همانند شکل ۱۳-۲ را روی برد FPGA پیاده سازی کنید.

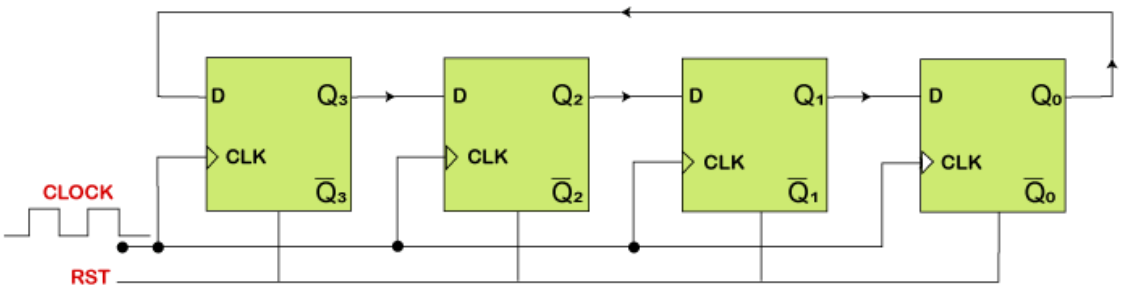


شکل ۱۳-۲ : جمع کننده ی ۸ بیتی

آزمایش سوم: مدارات ترتیبی

RING COUNTER (رینگ کانتر)

ring counter (شمارنده ی حلقه بسته) یک مدار دیجیتالی با یک سری فلیپ فلاپ است که به صورت سری به هم متصل شده اند. Ring Counter از Shift Registers تشکیل شده است. تا زمانی که پالس های ساعت اعمال شوند، الگوی داده دوباره تکرار می شود.



شکل ۱-۳ : مدار رینگ کانتر با فیلیپ فلاپ های D

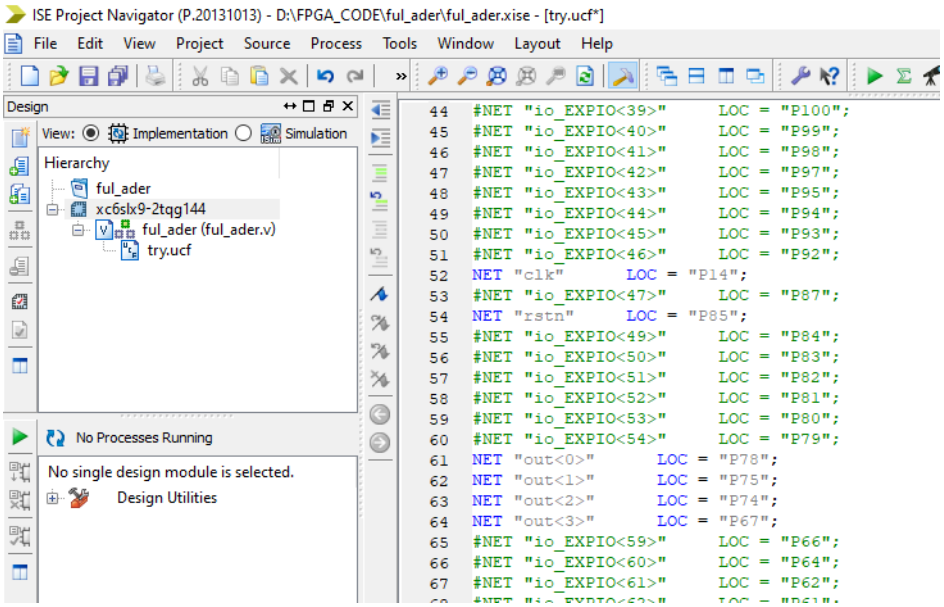
همان طوری که در شکل ۱-۳ مشاهده می کنید مدار نوع خاصی از شیفت رجیستر است که در آن خروجی آخرین فلیپ فلاپ به ورودی اولین فلیپ فلاپ باز می گردد. وقتی مدار ریست می شود، به جز یکی از خروجی های فلیپ فلاپ، بقیه مدارها صفر می شوند. برای مثال، اگر یک شمارنده حلقه ٤ بیتی بگیریم، الگوی داده هر چهار پالس ساعت تکرار می شود. اگر الگوی ۱۰۰۰ باشد، ۰۱۰۰، ۰۰۱۰، ۰۰۰۱، ۱۰۰۰ و غیره تولید می کند.

حال برای پیاده سازی این شمارنده روی FPGA می توانید از کد های vrilog زیر استفاده کنید (شکل ۲-۳) .

```
module ring_ctr  #(parameter WIDTH=4)
 (
   input clk,
   input rstn,
   output reg [WIDTH-1:0] out
 );

 always @ (posedge clk) begin
    if (!rstn)
      out <= 1;
    else begin
      out[WIDTH-1] <= out[0];
      for (int i = 0; i < WIDTH-1; i=i+1) begin
        out[i] <= out[i+1];
      end
    end
 end
endmodule
```

شکل ۲-۳ : کد های VRILOG رینگ کانتر

با استفاده از کد های بالا و تنظیم خروجی ها و پایه ی ریست و کلاک، برد را پروگرم می کنیم. شکل ۳-۳ نمونه ای از تنظیم ورودی _ خروجی مدار می باشد.



شکل ۳-۳ : ورودی- خروجی ها و کلاک

اگر همه ی مراحل به درستی انجام شود؛ LED هایی که به عنوان خروجی انتخاب شده اند، به صورت کم رنگ روشن می شوند و حالت شمارشی بین آنها ایجاد نمی شود. در واقع انتظاری که ما داشتیم این بود که LED ها یکی یکی روشن و خاموش شوند؛ در حالی که همه ی LED ها به صورت هم زمان روشن هستند و این به خاطر فرکانس بالای کلاک می باشد و درنتیجه LED ها یا فرکانس بالایی روشن و خاموش می شوند که برای انسان قابل رویت نمی باشد.

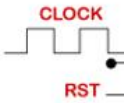
برای حل این مشکل باید فرکانس کلاک را در حدود ۱ هرتز پایین بیاوریم تا شمارش بین LED ها در حدود ۱ ثانیه باشد. برای این کار می توانیم از تقسیم کننده ی فرکانسی (clock_divider) استفاده کنیم. همانند شکل ٤-۳ یک ماژول تقسیم کننده ی فرکانسی ایجاد می کنیم.

```
1   `timescale 1ns / 1ps
2
3   module clock_divider #(parameter dive_value = 24999999) (
4       input clk_50,
5       output reg divided_clk = 0
6   );
7
8   //division value = (50MHz/(2*desired clock frequency)) -1
9   integer counter_value = 0;
10
11  always @(posedge clk_50) begin
12      if(counter_value == dive_value)
13          counter_value<=0;
14      else
15          counter_value<=counter_value+1;
16  end
17
18  always @(posedge clk_50) begin
19      if(counter_value == dive_value)
20          divided_clk <= ~divided_clk;
21  end
```

شکل ٤-۳ : تقسیم کننده ی فرکانسی

حال باید ماژول ایجاد شده را در کد های اصلی خود فراخوانی کنیم که شمارش با فرکانس جدید انجام شود(شکل ۵-۳).

```verilog
module ring_ctr  #(parameter WIDTH=4)
  (
    input clk,
    input rstn,
    output reg [WIDTH-1:0] out
  );
  integer i;
  clock_divider dut(
    .clk_50(clk),
    .divided_clk(clk_out)
  );
  always @ (posedge clk_out) begin
      if (!rstn)
         out <= 1;
      else begin
        out[WIDTH-1] <= out[0];
        for (i = 0; i < WIDTH-1; i=i+1) begin
          out[i] <= out[i+1];
        end
      end
  end
```

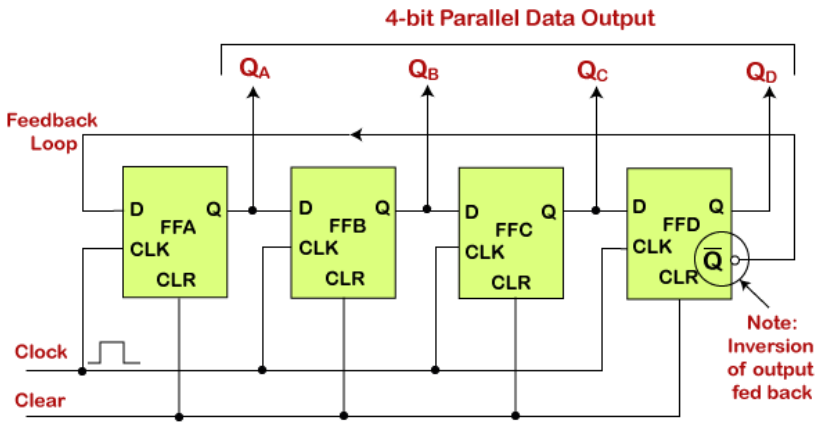


شکل ۵-۳ : شمارنده ی رینگ کانتر با فرکانس ۱هرتز

۳-۲) JOHNSON COUNTER (جانسون کانتر)

شمارنده جانسون یک مدار دیجیتالی با فلیپ فلاپ نوع D است که به صورت سری به هم متصل شده اند.

شمارنده جانسون Verilog شمارنده ای است که اگر تعداد بیت ها N باشد، ۲ N حالت را می شمارد.



شکل ٦-۳ : جانسون کانتر ٤ بیتی

شکل ۶–۳ مدار جانسون کانتر ٤ بیتی است که در آن خروجی فلیپ فلاپ ها به ورودی فلیپ فلاپ بعدی متصل است. و خروجی معکوس Q آخرین فلیپ فلاپ به ورودی D اولین فلیپ فلاپ وصل می شود.

خروجی Q به ورودی D متصل می شود، و این الگوی ۸ بیتی به طور مداوم تکرار می شود. برای مثال، تکرار به ترتیب «۱۰۰۰»، «۱۱۰۰»، «۱۱۱۰»، «۱۱۱۱»، «۰۱۱۱»، «۰۰۱۱»، «۰۰۰۱»، «۰۰۰۰» می باشد که در شکل ۷–۳ هم نشان داده شده است.

Clock Pulse No	FFA	FFB	FFC	FFD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

شکل ۷–۳ :: الگوی شمارش JOHNSON COUNTER

کد های vrilog مربوط به این شمارنده در شکل ۸–۳ آورده شده است.

```
module johnson_ctr #(parameter WIDTH=4)
 (
  input clk,
  input rstn,
  output reg [WIDTH-1:0] out
 );

 always @ (posedge clk) begin
    if (!rstn)
      out <= 1;
    else begin
      out[WIDTH-1] <= ~out[0];
      for (int i = 0; i < WIDTH-1; i=i+1) begin
        out[i] <= out[i+1];
      end
    end
 end
endmodule
```

شکل ۸–۳ :کدهای JOHNSON COUNTER

همانند روش قبل؛ اگر این کد ها هم روی برد پروگرم شود مشاهده می شود که در این جا هم فرکانس کلاک زیاد است و شمارش در خروجی دیده نمی شود. باز هم می توان از تقسیم کننده ی فرکانس همانند شکل ۹-۳ استفادکرد.

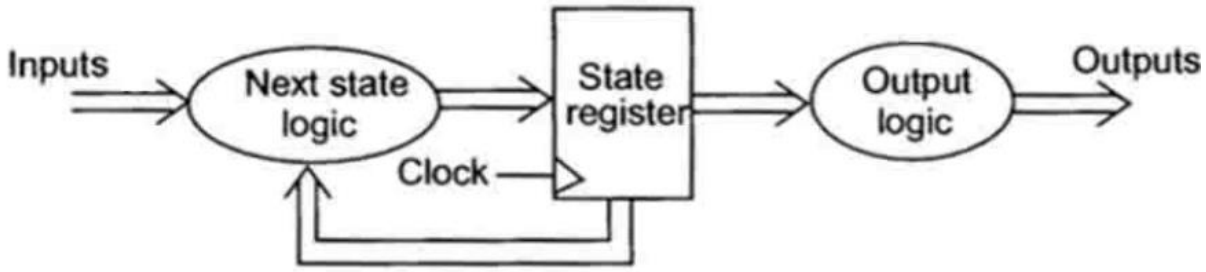
```
module ring_ctr  #(parameter WIDTH=4)
  (
    input clk,
    input rstn,
    output reg [WIDTH-1:0] out
  );
  integer i;
  clock_divider dut(
    .clk_50(clk),
    .divided_clk(clk_out)
  );
  always @ (posedge clk_out) begin
      if (!rstn)
         out <= 1;
      else begin
        out[WIDTH-1] <= out[0];
        for (i = 0; i < WIDTH-1; i=i+1) begin
          out[i] <= out[i+1];
        end
      end
  end
endmodule
```

شکل ۹-۳ : کدهای JOHNSON COUNTER با تقسیم کننده ی فرکانسی

آزمایش چهارم: بررسی FINITE STATE MACHINES (FSM)

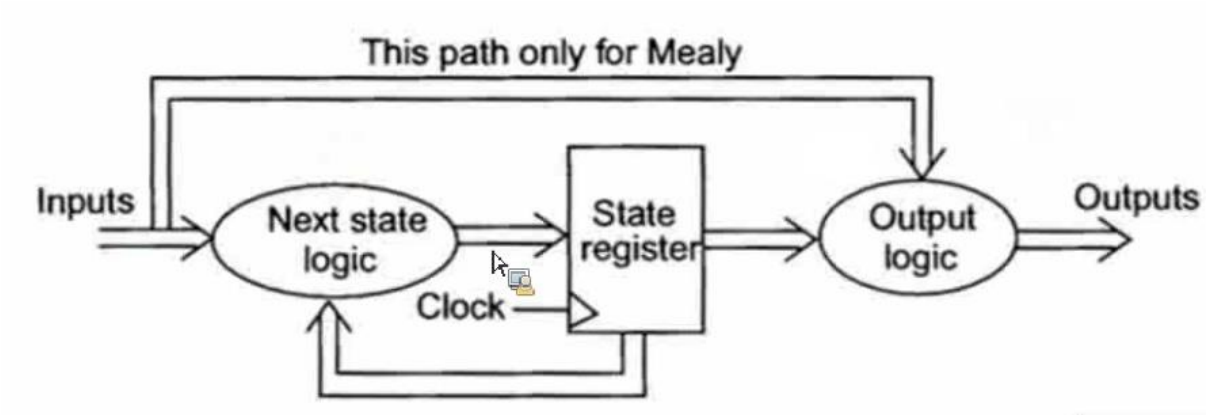
دو نوع ماشین حالت محدود وجود دارد mealy و Moore .

در تئوری محاسبات، ماشین Moore یک ماشین حالت محدود است که مقادیر خروجی فعلی آن، تنها با وضعیت فعلی آن تعیین می شود.



شکل ۱-٤ : دیاگرام ماشین مور

در تئوری محاسبات، ماشین Mealy یک ماشین حالت محدود است که مقادیر خروجی فعلی آن هم با وضعیت فعلی و هم با ورودی های فعلی تعیین می شود.



شکل ۲-٤ : دیاگرام ماشین میلی

نمونه ای از مثال و کاربرد های FSM ها :

۱- دستگاه فروش خودکار

۲- چراغ های راهنمایی

۳- سیستم گرمایشی

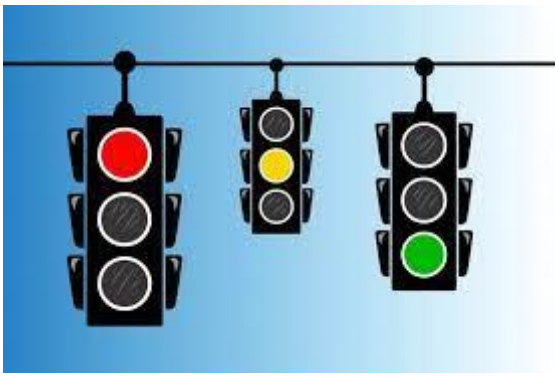
٤- آسانسور

٥- ماشین های خودران

٦- سیستم متروی خودکار

بعضی از مثال های بالا همانند چراغ های راهنمایی دارای تعداد حالت های محدود، مثلا ۳ حالت می باشد؛ در حالی که مثال هایی مانند ماشین خودران و یا دستگاه فروش دارای حالت های مختلف و نسبتا پیچیده ای می باشند.

برای آشنایی بیشتر سیستم چراغ راهنمایی را پیاده سازی می کنیم.



شکل ۳-٤ : چراغ راهنمایی

می توانید از لینک زیر کد های مربوط به این سیستم را دریافت کنید :

https://github.com/FPGADude/Digital-Design/tree/main/FPGA%۲۰Projects/Traffic_Controller_Basys

ماژول مربوط به کنترل چراغ راهنمایی به صورت زیر می باشد.

```
@@ -0,0 +1,17 @@
+ `timescale 1ns / 1ps
+
+ module traffic_controller(
+     input reset,            // button
+     input clk_100MHz,
+     output [2:0] main_st,   // LEDs
+     output [2:0] cross_st   // LEDs
+     );
+
+     wire w_1Hz, w_reset;
+
+     state_machine sm(.reset(w_reset), .clk_1Hz(w_1Hz),
+                      .main_st(main_st), .cross_st(cross_st));
+     oneHz_gen uno(.clk_100MHz(clk_100MHz), .reset(w_reset), .clk_1Hz(w_1Hz));
+     sw_debounce db(.clk(clk_100MHz), .btn_in(reset), .btn_out(w_reset) );
+
+ endmodule
```

شکل ٤–٤ : ماژول TRAFFIC-CONTROLLER

Create oneHz_gen.v مربوط به ماژولی هست که یک کلاک ۱ هرتزی یا همان ۱ ثانیه را تولید می کند. در واقع کلاک ۱۰۰مگاهرتز را به یک هرتز تبدیل می کند.

```
@@ -0,0 +1,26 @@
+ `timescale 1ns / 1ps
+
+ module oneHz_gen(
+     input clk_100MHz,   // from BASYS 3
+     input reset,        // btnC on BASYS 3
+     output clk_1Hz
+     );
+
+     reg clk_1Hz_reg = 0;
+     reg [25:0] counter_reg;
+
+     always @(posedge clk_100MHz or posedge reset) begin
+         if(reset)
+             counter_reg <= 0;
+         else
+             if(counter_reg == 49_999_999) begin
+                 counter_reg <= 0;
+                 clk_1Hz_reg <= ~clk_1Hz_reg;
+             end
+             else
+                 counter_reg <= counter_reg + 1;
+     end
+
+     assign clk_1Hz = clk_1Hz_reg;
+
+ endmodule
```

شکل ٤–٥ : ماژول CREATE ONEHZ

sw_debounce یکی دیگر از ماژول های استفاده شده می باشد که همانطور که از اسمش پیداس برای Debouncing یا نویز گیری کلید ها استفاده می شود. که یکی از راه حل های آن ایجاد تاخیر می باشد.

19 lines (14 sloc) | 328 Bytes

```
1   `timescale 1ns / 1ps
2
3   module sw_debounce(
4       input clk,
5       input btn_in,    // using a button, not switch, works the same
6       output btn_out
7       );
8
9       reg t0, t1, t2;
10
11      always @(posedge clk) begin
12          t0 <= btn_in;
13          t1 <= t0;
14          t2 <= t1;
15      end
16
17      assign btn_out = t2;
18
19  endmodule
```

شکل ٦-٤ : ماژول DEBOUNCING

کد های اصلی هم که مربوط به ماشین حالت می شود به اسم state_machine.v به صورت زیر می باشد(شکل ٤-٧ و شکل ٤-٨):

```
`timescale 1ns / 1ps

module state_machine(
    input reset,    // btnC on BASYS 3
    input clk_1Hz,
    output reg [2:0] main_st,   // 3-bits out PMOD_B
    output reg [2:0] cross_st   // 3-bits out PMOD_C
    );

    // Define states
    parameter main_green_cross_red  = 2'b00;
    parameter main_yellow_cross_red = 2'b01;
    parameter main_red_cross_green  = 2'b10;
    parameter main_red_cross_yellow = 2'b11;

    // The state register
    reg [1:0] state_reg;    // 4 states = 2 bits

    // Timer for light changes
    reg [4:0] light_counter = 0;    // main green = 15 seconds
                                    // main yellow = 3 seconds
                                    // cross green = 10 seconds
                                    // cross yellow = 3 seconds
                                    // total seconds = 31 = 5 bits

    // Next state logic
    always @(posedge clk_1Hz or posedge reset) begin
        if(reset)
            state_reg <= main_green_cross_red;  // reset state
        else
            case(state_reg)
```

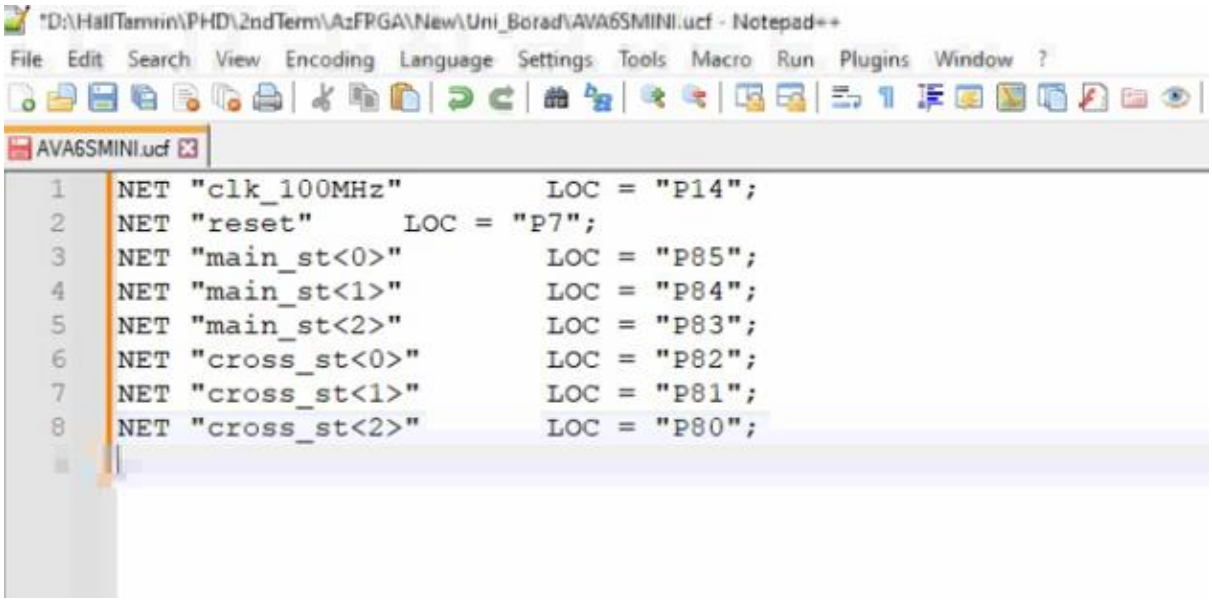
شكل ٤–٧: STATE_MACHINE

```
            case(state_reg)
                main_green_cross_red  : if(light_counter == 15)    state_reg <= main_yellow_cross_red;
                main_yellow_cross_red : if(light_counter == 18)    state_reg <= main_red_cross_green;
                main_red_cross_green  : if(light_counter == 28)    state_reg <= main_red_cross_yellow;
                main_red_cross_yellow : if(light_counter == 31)    state_reg <= main_green_cross_red;
                default : state_reg <= main_green_cross_red;
            endcase
    end

    // Light Counter control
    always @(posedge clk_1Hz or posedge reset) begin
        if(reset)
            light_counter <= 0;
        else
            if(light_counter == 31)
                light_counter <= 0;
            else
                light_counter <= light_counter + 1;
    end

    always @(posedge clk_1Hz) begin
        case(state_reg)
            main_green_cross_red  : begin
                                        main_st  = 3'b001;  // green
                                        cross_st = 3'b100;  // red
                                    end
            main_yellow_cross_red : begin
                                        main_st  = 3'b010;  // yellow
                                        cross_st = 3'b100;  // red
                                    end
            main_red_cross_green  : begin
                                        main_st  = 3'b100;  // red
                                        cross_st = 3'b001;  // green
                                    end
            main_red_cross_yellow : begin
                                        main_st  = 3'b100;  // red
                                        cross_st = 3'b010;  // yellow
                                    end
        endcase
    end

endmodule
```

شكل ٤–٨: STATE_MACHINE

ماشین حالت استفاده شده در بالا مور (moore) می باشد؛ چون خروجی ها با حالت ماشین مشخص می شود و همچنین ورودی مستقلی ندارد و فقط یک حالت اولیه برای آن تعرف شده است.

پین ها را هم در فایل usf. می توانید همانند شکل ۹-٤ مرتب کنید. البته باید فرکانس کلاک را با توجه به فرکانس کاری برد خود تنظیم کنید.



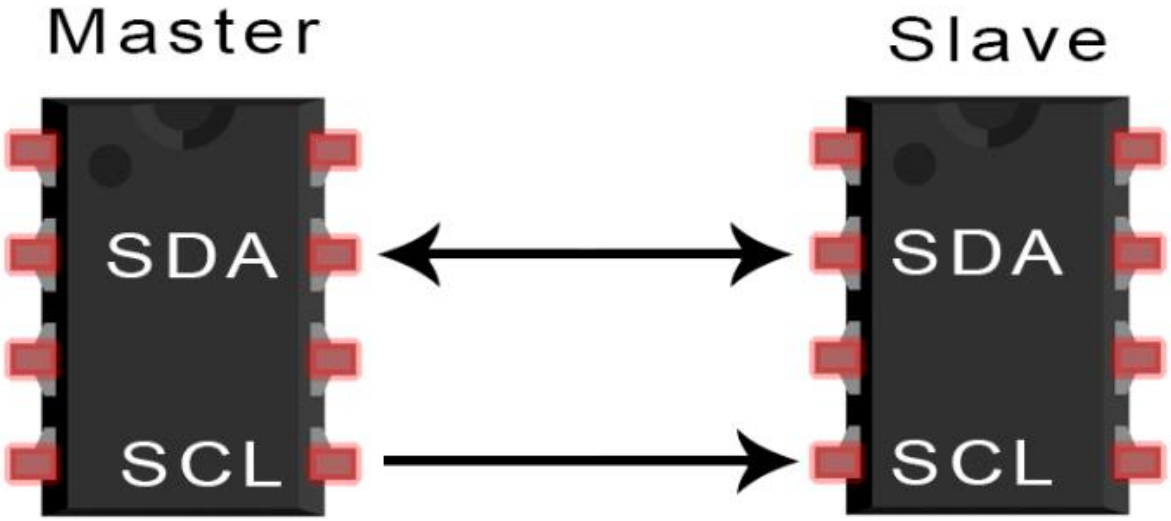
شکل ۹-٤: تنظیمات پین ها

آزمایش پنجم: آشنایی با پروتکل I۲C و راه اندازی دوربین OV۷۶۷۰

آشنایی با پروتکل I۲C

ارتباط I۲C توسط Phillips Semiconductor ساخته شد و سالها بعد اینتل پروتکل SMBus را به عنوان I۲C تعریف کرد. I۲C بهترین ویژگی های SPI و UART را با هم ترکیب می کند. با استفاده از I۲C می توانید چندین Slave به یک Master واحد متصل کنید (مانند SPI) و همچنین می توانید چندین Master را کنترل کنید که یک یا چند Slave را کنترل کنند. این امر درصورتی مفید است که می خواهید بیش از یک میکروکنترلر در پروژه خود استفاده کنید.

مانند ارتباطات UART، ارتباط I۲C نیز فقط از دو سیم برای انتقال داده بین دستگاه ها استفاده می کند:



شکل ۱-۵ : ارتباط I۲C

SDA (Serial Data): خطی برای Master و Slave برای ارسال و دریافت داده ها.

SCL (Serial Clock): خطی که سیگنال ساعت را حمل می کند.

I۲C یک پروتکل ارتباطی سریال است، بنابراین داده ها نوبتی در امتداد یک سیم(خط SDA) منتقل می شوند.

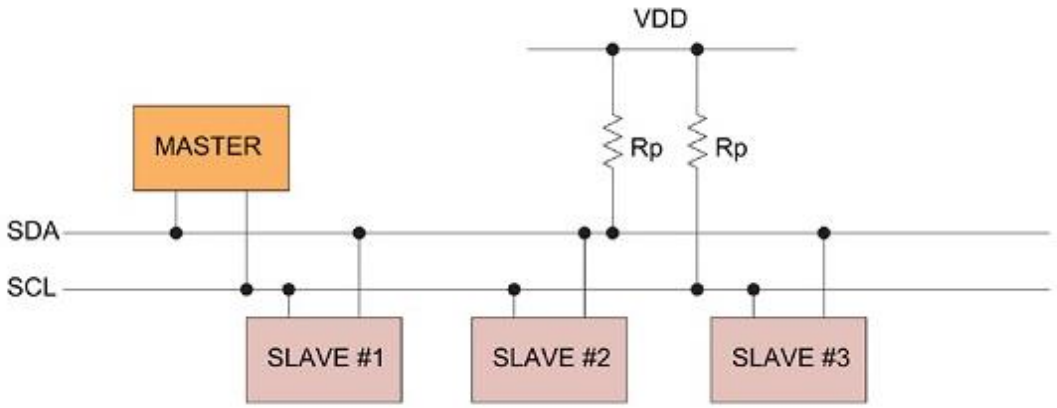
مزایای I۲C:

فقط از دو سیم استفاده می کند.

از چندین master و چندین slave پشتیبانی می کند.

بیت ACK/NACK تأیید می کند که هر فریم با موفقیت منتقل شده است.

پروتکل شناخته شده و پرکاربردی است.



شکل ۲-۵: ارتباط I۲C

معایب I۲C:

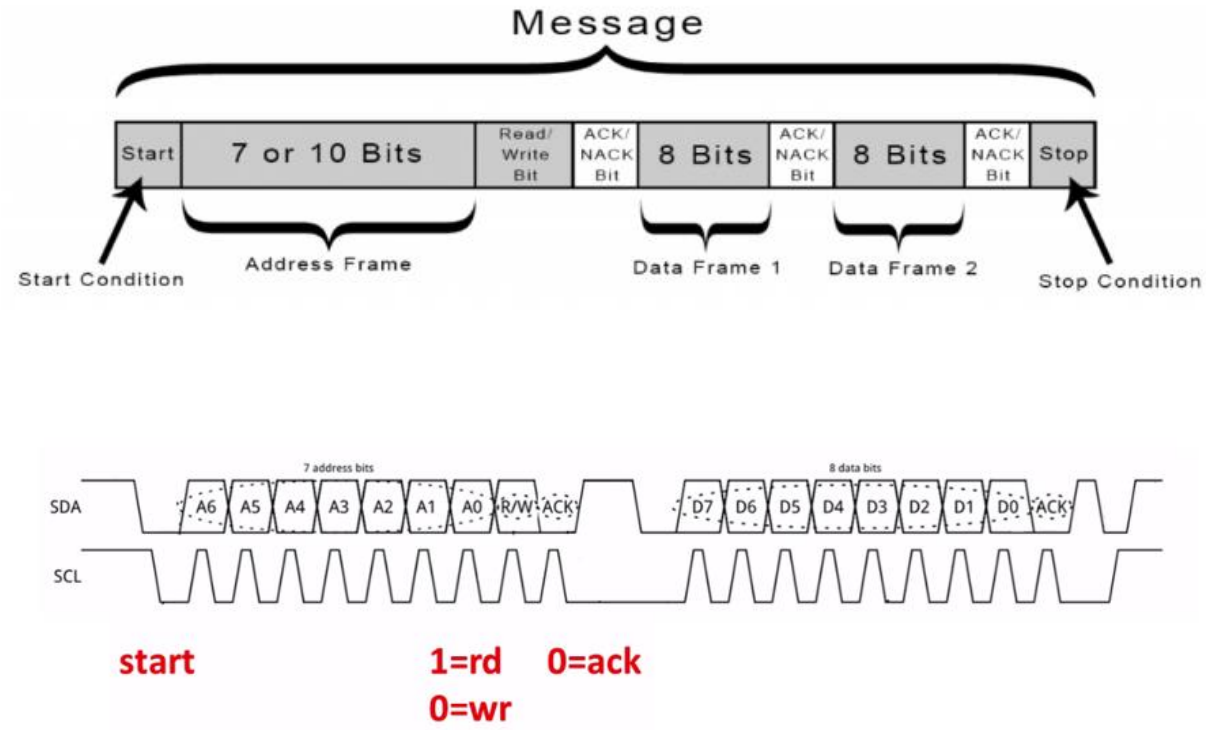
Half Dublex (نیمه دو طرفه)

سرعت انتقال داده کندتر از SPI است.

اندازه فریم داده به ۸ بیت محدود می شود.

سخت افزار پیچیده تری نسبت به SPI و UART برای اجرا لازم است.

با استفاده از I²C، داده ها در پیام ها منتقل می شوند. پیام ها به فریم های داده تقسیم می شوند. هر پیام دارای یک فریم آدرس است که شامل آدرس باینری Slave و یک یا چند فریم داده است که حاوی داده های منتقل شده است. این پیام همچنین شامل شرایط شروع و توقف، بیت های خواندن / نوشتن و بیت های ACK / NACK بین هر قاب داده است:



شکل ۳-۵: ساختار I²C

بسته های داده I²C در بایت های ۸ بیتی انجام می شود. این بایت ها شامل آدرس slave، شماره رجیستر و داده های منتقل شده می باشند. این انتقال از طریق باس به معنی یک عمل خواندن یا نوشتن است. پروتکل های خواندن و نوشتن بر اساس یک سری از پروتکل های فرعی مانند شرایط شروع و توقف، بیت های تکراری شروع، بایت آدرس، بیت های انتقال داده و بیت های تأیید و عدم تأیید ساخته می شوند.

راه اندازی دوربین OV۷۶۷۰

سنسور تصویر OV۷۶۷۰ محصولcamerachip، دارای بخشهای مختلف برای پردازش اولیه روی تصویر (میزان رنگ، کنترانس و....) میباشد. همچنین توانایی تنظیم خودکار بهره AGC ، تنظیم تراز سفیدی AWB را دارا میباشد، مدارات بایاسینگ و رگولاتور ۱.۸ ولت بر روی خود برد قرارگرفته است.
-۲-۵

شکل ٤-٥: دوربین OV۷۶۷۰

مشخصات:

حساسیت بالا در محیط های با نور کم

حداکثر ابعاد تصویر ٦٤٠ در ٤٨٠ پیکسل

قابلیت تصویر برداری با ۳۰ فریم بر ثانیه

ابعاد تصویر قابل تعریف در استاندار های مختلف (vga , qvga ,...)

نسبت سیگنال به نویز ٤٦ دسیبل

برقراری ارتباط با میکروکنترلر از طریق رابط SCCB مشابهI۲C.

توان مصرفی: ٦۰ میلی وات در ۱۵ فریم

جریان مصرفی در حالت خواب: کمتر از ۲۰ میکرو آمپر

دمای کارکرد: ۷۰ °C to ۳۰°C- سانتی گراد

سایز اپتیکال: ۱/۶اینچ

میدان دید: ۲۵ درجه

حداکثر نرخ فریم: ۳۰ فریمVGA

نسبت سیگنال به نویز: ٤٦ دسی بل

داینامیک رنج: ۵۲ دسی بل

اکسپوژر الکترونیکی: ۱ خط تا ۵۱۰ خط

ابعاد پیکسل: ۳.٦ µm x ۳.٦ µm

کاربردها:

سیستم های امنیتی

سیستم های هوشمند

کاربردهای آموزشی

پروژه های الکترونیکی

بینایی ماشین

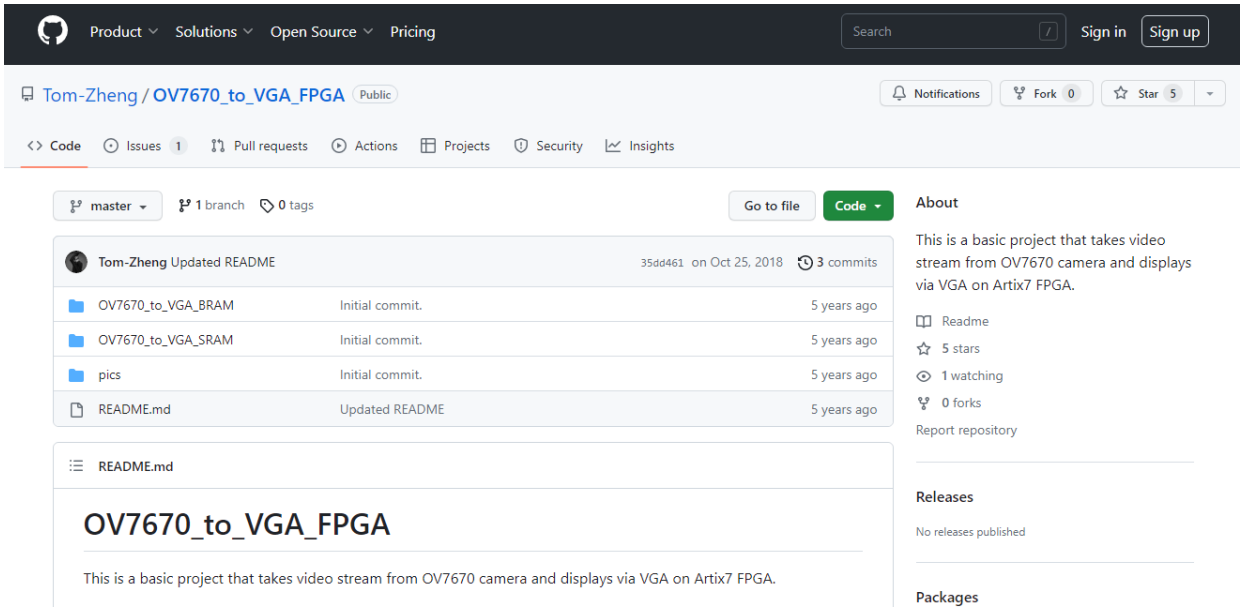
هر پروژه ای که نیاز به تصویر برداری یا عکس برداری داشته باشد

برای دریافت دیتاشیت OV۷٦۷۰ و آشنایی بیشتر با آن می توانید به آدرس زیر مراجعه کنید.

https://datasheetspdf.com/pdf-file/۵۵۵۲۲۰/OmniVisionTechnologies/OV۷٦۷۰/۱

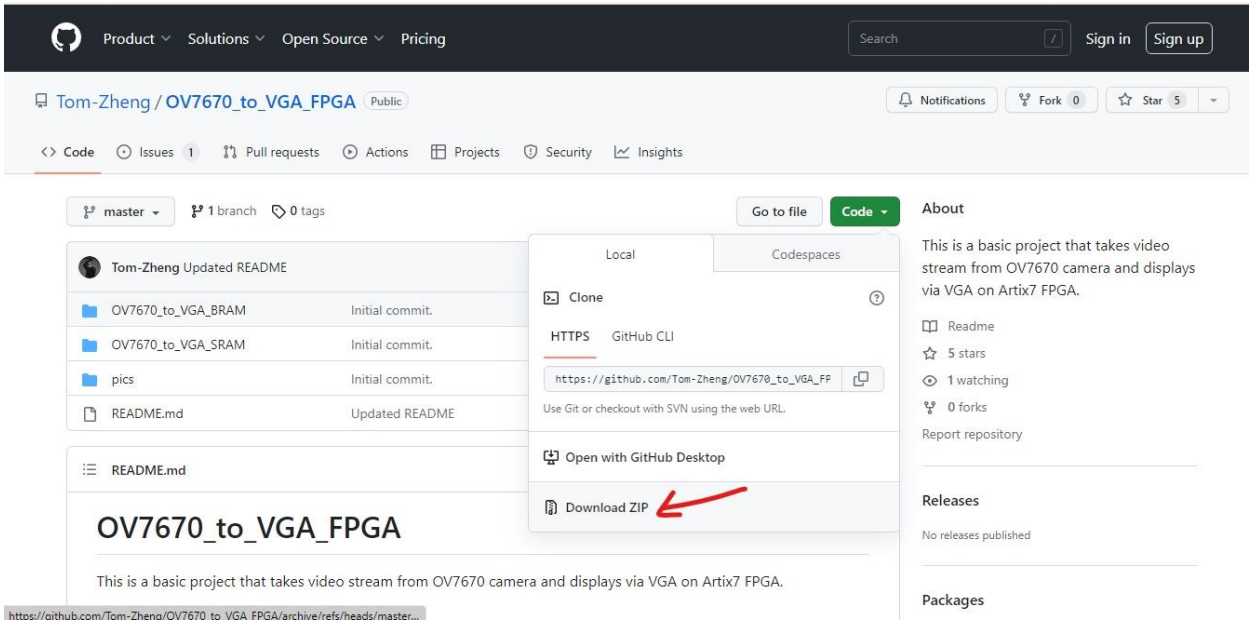
حال برای استفاده از دوربین ابتدا دوربین را به برد FPGA وصل می کنیم. از لینک زیر کد های مربوطه را دانلود می کنیم و بعد از وصل دوربین؛ کدها را روی برد پروگرم می کنیم.

https://github.com/Tom-Zheng/OV۷٦۷۰_to_VGA_FPGA



شکل ۵-۵: کد های دوربین در GitHub

بعد از ورود به سایت GitHub به آدرس فوق؛ به صفحه ای همانند شکل ۵-۵ وارد شده و از گزینه ی code آخرین گزینه یعنی Download Zip می توانید کد ها را در اختیار داشته باشید(شکل ۶-۵).



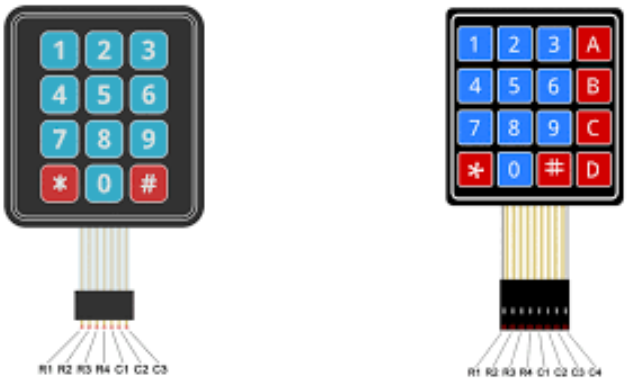
شکل ۶-۵: کد های دوربین در GitHub

با استفاده از دیتاشیت دوربین می توانید بعضی از کد ها را در فایل I۲C_OV۷۶۷۰_RGB۵۶۵_Config۲ را ویرایش کنید و دوربین را باتوجه به نیاز خودتان تنظیم کنید.

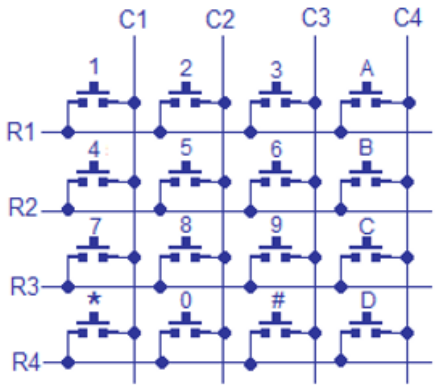
آزمایش ششم: **پیاده سازی CALCULATOR**

KEY PAD(کیپد یا صفحه کلید)

کیپد مجموعه از تاچ سوئیچ ها می‌باشند که به صورت سطری، ستونی؛ به طوری که تشکیل یک ماتریس دهند در کنار یک دیگر قرار گرفته اند. صفحه‌کلیدها (شکل ۱-٦)اغلب در دستگاه‌هایی یافت می‌شوند که عمدتاً به ورودی عددی نیاز دارند، مانند ماشین‌حساب، کنترل از راه دور تلویزیون، تلفن‌های دکمه‌ای، دستگاه‌های فروش خودکار، دستگاه‌های خودپرداز، پایانه‌های فروش، قفل‌های ترکیبی، گاوصندوق و قفل دیجیتالی درب.



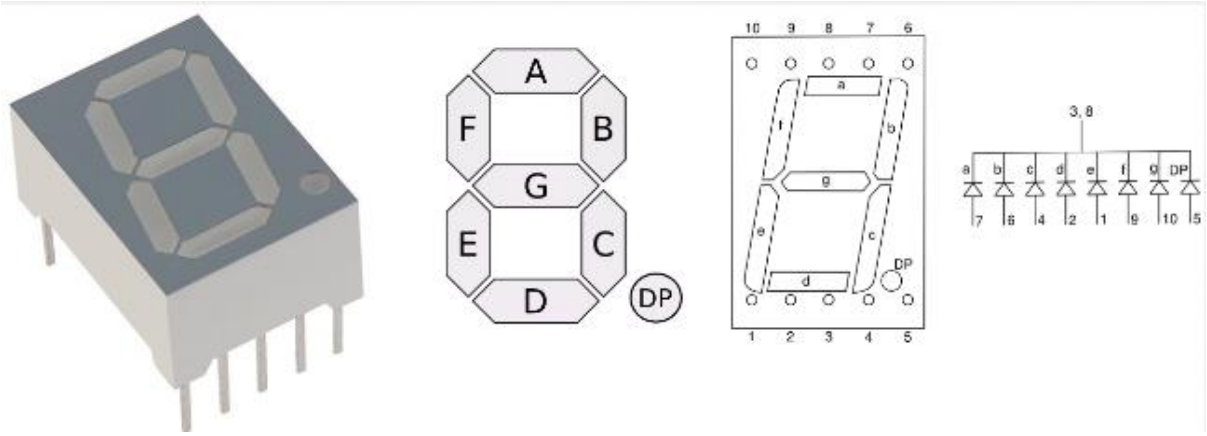
شکل ۱-٦: نمونه ای از کیپد



شکل ۲-٦: ساختار داخلی کیپد

SEVEN SEGMENT DISPLAY

سون سگمنت (Seven Segment Display) یا همان SSD ها یکی از ارزان ترین، پر کاربردترین و راحت ترین قطعات الکترونیکی هستند که به عنوان قطعات نمایشگرها مورد استفاده قرار می گیرند. علت اصلی نام گذاری این محصول این است که این قطعه از ۷ قسمت تشکیل شده است که درشکل ۳-٦-۲- قابل مشاهده است.



شکل ۳-٦: سون سیگمنت

در کل دو نوع سون سگمنت داریم :

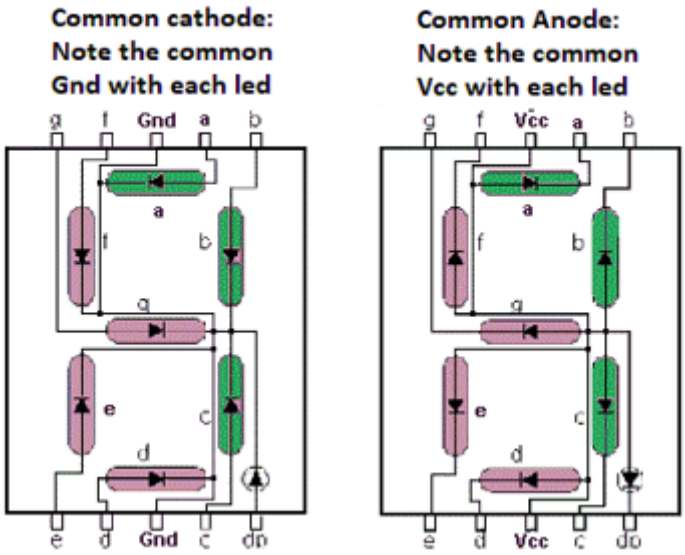
۱- سون سگمنت های کاتد مشترک

۲- سون سگمنت های آند مشترک

در صورتی که در سون سگمنت، کاتد کلیه LED ها به هم متصل شده باشد به آن سون سگمنت، سون سگمنت کاتد مشترک گفته می شود. به عبارت دیگر برای روشن کردن هر یک از LED های نام گذاری شده A تا G باید پایه مشترک که در اینجا همان پایه کاتد مشترک محسوب می شود به صفر یا زمین وصل شود و برای روشن کردن هر سگمنت به پایه ی متناظر آن سگمنت مثبت یا یک داده شود.

در صورتی که در سون سگمنت، آند کلیه LED ها به هم متصل شده باشد به آن سون سگمنت، سون سگمنت آند مشترک گفته می شود. به عبارت دیگر برای روشن کردن هر یک LED های نام گذاری شده A تا G باید پایه

مشترک که در اینجا همان پایه آند مشترک محسوب می شود به یک یا مثبت وصل شود و برای روشن کردن هر سگمنت به پایه ی متناظر آن سگمنت صفر یا زمین داده شود.



شکل ٤-٦: کاتد و آند مشترک

کنترل یک SEVEN SEGMENT با کلید های کنترلی

با فرض اینکه سون سیگمنت، آند مشترک باشد؛ برای روشن کردن آن با استفاده از کلید های کنترلی باید از یک استفاده کنیم؛ با توجه به این موضوع کد های زیر را می نویسیم:

```
1   `timescale 1ns / 1ps
2   module SevenSeg(
3       input [3:0] num,
4       output [3:0] AN,
5       output reg CA,
6       output reg CB,
7       output reg CC,
8       output reg CD,
9       output reg CE,
10      output reg CF,
11      output reg CG,
12      output reg DP
13      );
14      assign AN = 4'b0001;
15      always @(*) begin
16      case(num)
17      4'b0000: begin
18      CA <= 1;
19      CB <= 1;
20      CC <= 1;
21      CD <= 1;
22      CE <= 1;
23      CF <= 1;
24      CG <= 0;
25      DP <= 0;
26      end
```

```
27      4'b0001: begin
28      CA <= 0;
29      CB <= 1;
30      CC <= 1;
31      CD <= 0;
32      CE <= 0;
33      CF <= 0;
34      CG <= 0;
35      DP <= 0;
36      end
37      4'b0010: begin
38      CA <= 1;
39      CB <= 1;
40      CC <= 0;
41      CD <= 1;
42      CE <= 1;
43      CF <= 0;
44      CG <= 1;
45      DP <= 0;
46      end
47      4'b0011: begin
48      CA <= 1;
49      CB <= 1;
50      CC <= 1;
51      CD <= 1;
52      CE <= 0;
53      CF <= 0;
54      CG <= 1;
55      DP <= 0;
56      end
```

```
57      4'b0100: begin
58      CA <= 0;
59      CB <= 1;
60      CC <= 1;
61      CD <= 0;
62      CE <= 0;
63      CF <= 1;
64      CG <= 1;
65      DP <= 0;
66      end
67      4'b0101: begin
68      CA <= 1;
69      CB <= 0;
70      CC <= 1;
71      CD <= 1;
72      CE <= 0;
73      CF <= 1;
74      CG <= 1;
75      DP <= 0;
76      end
77      4'b0110: begin
78      CA <= 1;
79      CB <= 0;
80      CC <= 1;
81      CD <= 1;
82      CE <= 1;
83      CF <= 1;
84      CG <= 1;
85      DP <= 0;
86      end
```

```
87      4'b0111: begin
88      CA <= 1;
89      CB <= 1;
90      CC <= 1;
91      CD <= 0;
92      CE <= 0;
93      CF <= 0;
94      CG <= 0;
95      DP <= 0;
96      end
97      4'b1000: begin
98      CA <= 1;
99      CB <= 1;
100     CC <= 1;
101     CD <= 1;
102     CE <= 1;
103     CF <= 1;
104     CG <= 1;
105     DP <= 0;
106     end
107     4'b1001: begin
108     CA <= 1;
109     CB <= 1;
110     CC <= 1;
111     CD <= 1;
112     CE <= 0;
113     CF <= 1;
114     CG <= 1;
115     DP <= 0;
116     end
```

```
117     default: begin
118     CA <= 0;
119     CB <= 0;
120     CC <= 0;
121     CD <= 0;
122     CE <= 0;
123     CF <= 0;
124     CG <= 0;
125     DP <= 0;
126     end
127     endcase
128     end
129
130 endmodule
```

شکل ۵-٦: کد های OneSevenSeg

فایل usf. مربوطه را هم می توانید همانند شکل ٦-٦ مرتب کنید.

```
64  #NET "io_EXPIO<59>"     LOC = "P66";
65  #NET "io_EXPIO<60>"     LOC = "P64";
66  NET "DP"    LOC = "P27";
67  #NET "io_EXPIO<62>"     LOC = "P61";
68  #NET "io_EXPIO<63>"     LOC = "P59";
69  NET "CG"    LOC = "P30";
70  #NET "io_EXPIO<65>"     LOC = "P57";
71  #NET "io_EXPIO<66>"     LOC = "P56";
72  NET "CF"    LOC = "P33";
73  #NET "io_EXPIO<68>"     LOC = "P50";
74  #NET "io_EXPIO<69>"     LOC = "P48";
75  #NET "io_EXPIO<70>"     LOC = "P47";
76  #NET "io_EXPIO<71>"     LOC = "P46";
77  #NET "io_EXPIO<72>"     LOC = "P45";
78  #NET "io_EXPIO<73>"     LOC = "P44";
79  #NET "io_EXPIO<74>"     LOC = "P43";
80  NET "CE"    LOC = "P35";
81  #NET "io_EXPIO<76>"     LOC = "P40";
82  #NET "io_EXPIO<77>"     LOC = "P38";
83  #NET "io_EXPIO<78>"     LOC = "P35";
84  NET "CD"    LOC = "P40";
85  NET "num<3>"      LOC = "P83";
86  NET "CC"    LOC = "P43";
87  NET "num<2>"      LOC = "P84";
88  NET "CB"    LOC = "P45";
89  NET "num<1>"      LOC = "P85";
90  NET "CA"    LOC = "P47";
91  #NET "io_EXPIO<86>"     LOC = "P24";
92  NET "num<0>"      LOC = "P87";
93  NET "AN<0>"    LOC = "P24";
94  NET "AN<1>"    LOC = "P22";
95  NET "AN<2>"    LOC = "P17";
96  NET "AN<3>"    LOC = "P15";
97  #NET "io_EXPIO<92>"     LOC = "P15";
```

شکل ٦-٦: فایل usf. سون سیگمنت

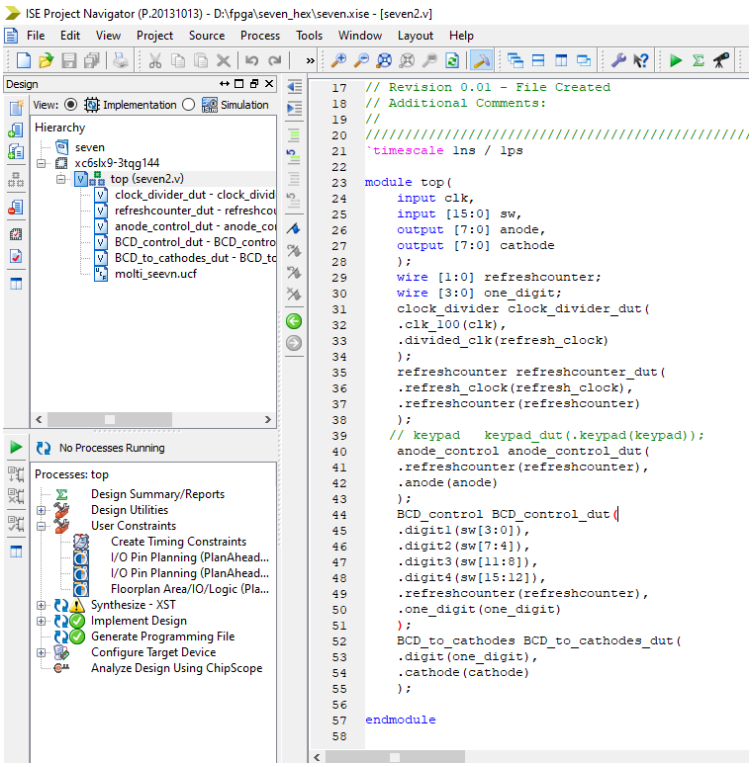
اگر کد های بالا به درستی نوشته شوند؛ می توانید با استفاده از ٤ سویچ یکی از سون سیگمنت ها را کنترل کنید.

کنترل چهار SEVEN SEGMENT با کلید های کنترلی

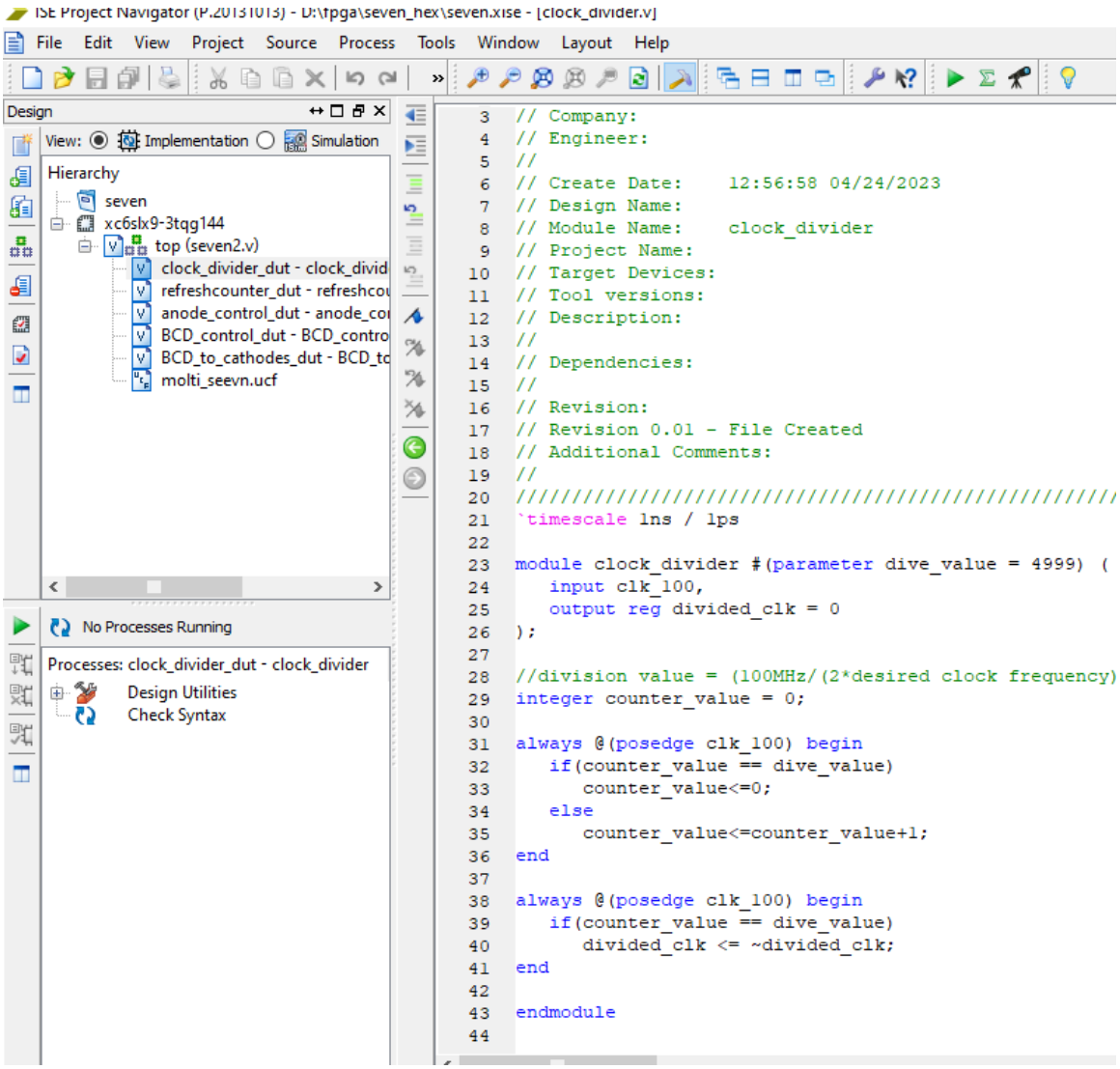
حال می خواهیم همزمان از ٤ سون سیگمت استفاده کنیم؛ برای این کار به ١٦ کلید کنترلی نیاز داریم به طوری که هر ٤ کلید برای نوشتن اعداد بین ٠ تا ٩ بر روی هر کدام از سون سیگمنت ها استفاده شود، در واقع اعداد به صورت BCD نمایش داده خواهند شد.

با توجه به این که نمی توانیم هم زمان ٤ مقدار مختلف را در ٤ سون سیگمنت با ورودی های یکسان نمایش دهیم؛ از روشی استفاده می کنیم که در آن سون سیگمنت ها به ترتیب با فرکانس کلاک روشن و خاموش می شوند و در لحظه ای که هر کدام از سون سیگمنت ها روشن می شوند، عدد متناظر با همان سون سیگمت به ورودی داده می شود؛ با توجه به این که فرکانس کلاک خیلی زیاد است؛ چشم انسان متوجه این تغییرات نمی شود.

حال با توجه به این توضیحات ماژول های زیر را تعریف می کنیم:



شکل ٧-٦: ماژول اصلی



شکل ۸-۶: CLOCK_DIVIDER

```
//
//////////////////////////////////////////////
`timescale 1ns / 1ps


module refreshcounter(
    input refresh_clock,
    output reg [1:0] refreshcounter = 0
    );

    always @(posedge refresh_clock)
        refreshcounter <= refreshcounter + 1;

endmodule
```

```
//
//////////////////////////////////////////////
`timescale 1ns / 1ps


module anode_control(
    input [1:0] refreshcounter,
    output reg [7:0] anode = 0
    );

    always @(refreshcounter) begin
        case(refreshcounter)
            2'b00: anode = 4'b0001;
            2'b01: anode = 4'b0010;
            2'b10: anode = 4'b0100;
            2'b11: anode = 4'b1000;
        endcase
    end

endmodule
```

شکل ۹-٦: دیگر ماژول ها

```
21  `timescale 1ns / 1ps
22
23
24  module BCD_control(
25      input [3:0] digit1,
26      input [3:0] digit2,
27      input [3:0] digit3,
28      input [3:0] digit4,
29      input [1:0] refreshcounter,
30      output reg [3:0] one_digit = 0
31      );
32
33      always @(refreshcounter) begin
34          case(refreshcounter)
35              2'b00: one_digit  = digit1;
36              2'b01: one_digit  = digit2;
37              2'b10: one_digit  = digit3;
38              2'b11: one_digit  = digit4;
39          endcase
40      end
41
42  endmodule
43
44
```

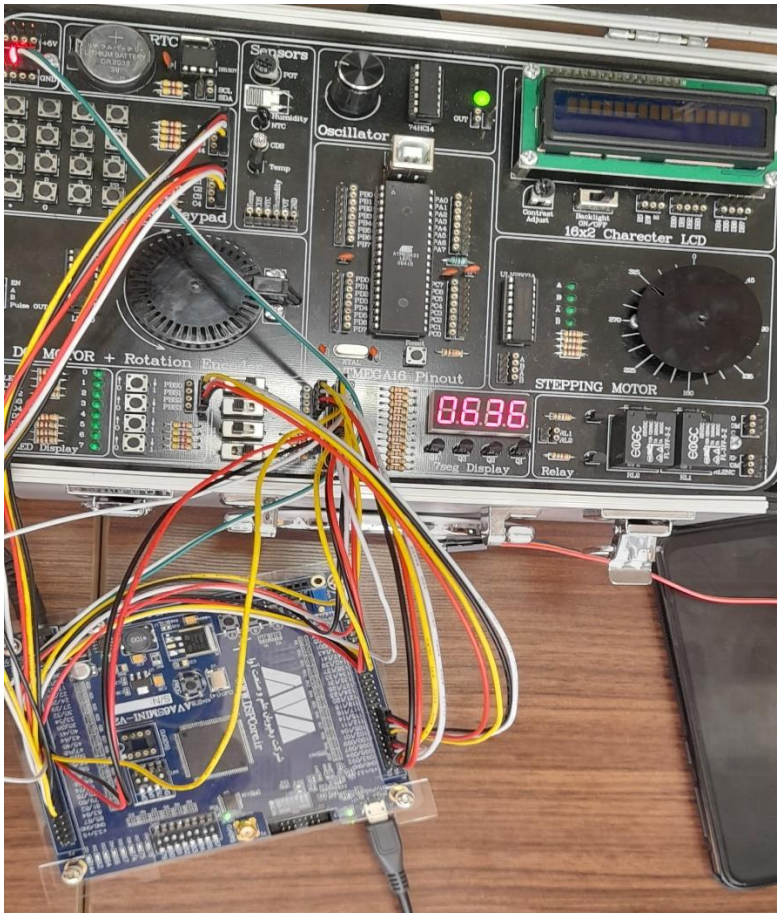
```
21  `timescale 1ns / 1ps
22
23
24  module BCD_to_cathodes(
25      input [3:0] digit,
26      output reg [7:0] cathode = 0
27      );
28
29      always @(digit) begin
30          case(digit)
31              8'd0: cathode = 8'b00111111;
32              8'd1: cathode = 8'b00000110;
33              8'd2: cathode = 8'b01011011;
34              8'd3: cathode = 8'b01001111;
35              8'd4: cathode = 8'b01100110;
36              8'd5: cathode = 8'b01101101;
37              8'd6: cathode = 8'b01111101;
38              8'd7: cathode = 8'b00000111;
39              8'd8: cathode = 8'b01111111;
40              8'd9: cathode = 8'b01101111;
41              default: cathode = 8'b00000000;
42          endcase
43      end
44
45  endmodule
46
47
```

شکل ۱۰-٦: دیگر ماژول ها

```
 85  #NET "sw<14>"      LOC = "P93";
 86  #NET "sw<13>"      LOC = "P120";
 87  #NET "sw<12>"      LOC = "P118";
 88  #NET "sw<11>"      LOC = "P7";
 89  #NET "sw<10>"      LOC = "P114";
 90  #NET "sw<9>"       LOC = "P111";
 91  #NET "sw<8>"       LOC = "P116";
 92  NET "sw<15>"       LOC = "P115";
 93  NET "sw<14>"       LOC = "P112";
 94  NET "sw<13>"       LOC = "P105";
 95  NET "sw<12>"       LOC = "P102";
 96  NET "sw<11>"       LOC = "P100";
 97  NET "sw<10>"       LOC = "P98";
 98  NET "sw<9>"     LOC = "P95";
 99  NET "sw<8>"     LOC = "P93";
100  NET "sw<7>"     LOC = "P87";
101  NET "sw<6>"     LOC = "P85";
102  NET "sw<5>"     LOC = "P84";
103  NET "sw<4>"     LOC = "P83";
104  NET "sw<3>"     LOC = "P82";
105  NET "sw<2>"     LOC = "P81";
106  NET "sw<1>"     LOC = "P80";
107  NET "sw<0>"     LOC = "P79";
108  NET "anode<7>"     LOC = "P104";
109  NET "anode<6>"     LOC = "P101";
110  NET "anode<5>"     LOC = "P99";
111  NET "anode<4>"     LOC = "P97";
112  NET "anode<3>"     LOC = "P15";
113  NET "anode<2>"     LOC = "P17";
114  NET "anode<1>"     LOC = "P22";
115  NET "anode<0>"     LOC = "P24";
116  NET "cathode<0>"      LOC = "P47";
117  NET "cathode<1>"      LOC = "P45";
118  NET "cathode<2>"      LOC = "P43";
119  NET "cathode<3>"      LOC = "P40";
120  NET "cathode<4>"      LOC = "P35";
121  NET "cathode<5>"      LOC = "P33";
122  NET "cathode<6>"      LOC = "P30";
123  NET "cathode<7>"      LOC = "P27";
124
125  #NET "io_EXPIO<86>"      LOC = "P24";
126
```

شکل ۱۱-٦: نمونه فایل USF. (علاوه بر پین های بالا کلاک هم باید تعریف شود)

اگر همه ی مراحل به درستی انجام گرفته باشد؛ می توان نتایجی همانند زیر را مشاهده کرد :

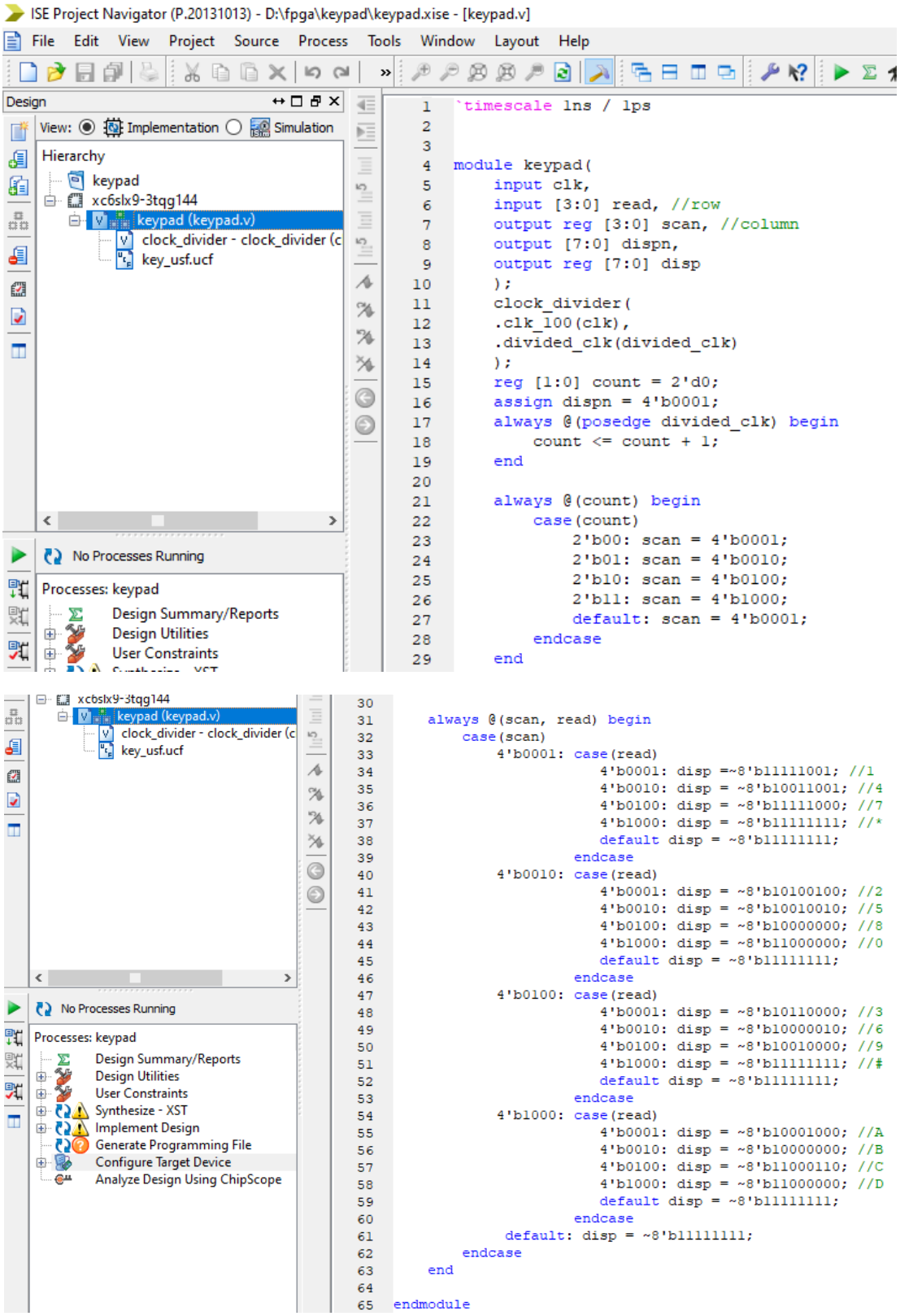


شکل ۱۲-٦: نتایج کنترل چهار SEVEN SEGMENT
٥-٦-

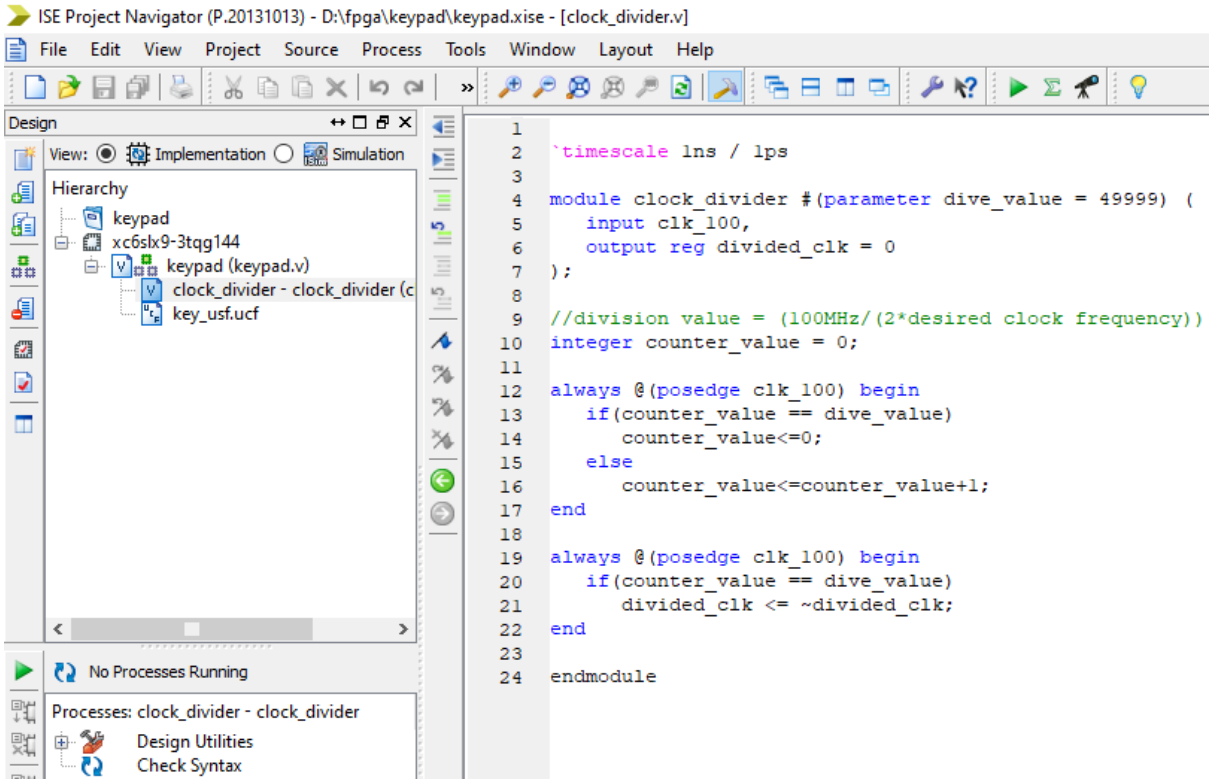
کنترل SEVEN SEGMENT با کیپد

کیپد ۸ پین دارد که ٤ پین برای خواندن و ٤ پین برای نوشتن استفاده می شود. با نوشتن ۰ در ٤ پین نوشتنی و چرخاندن آن بین این ٤ پین و با توجه به اتصال ماتریسی کلید ها؛ با خواندن ورودی ها می توان متوجه شد که کدام یک از کلید ها فشرده شده است.

با توجه به توضیحات بالا کد های وریلاگ به صورت زیر نوشته می شوند.



شکل ۱۳-۶: ماژول اصلی

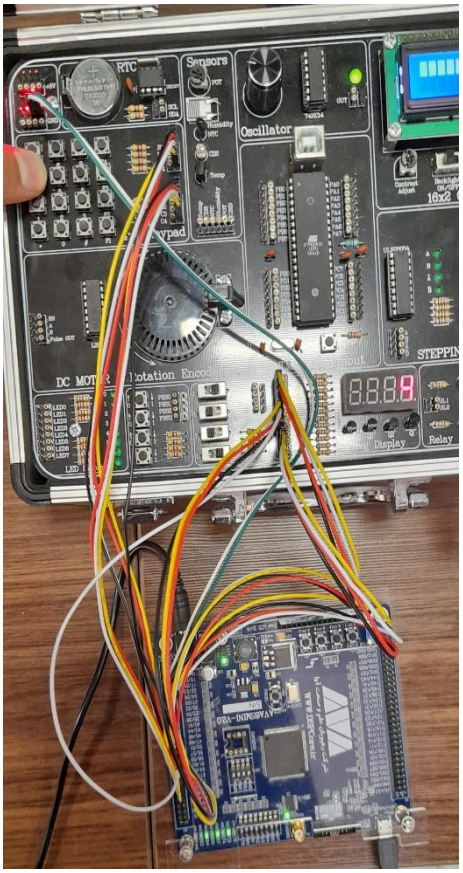


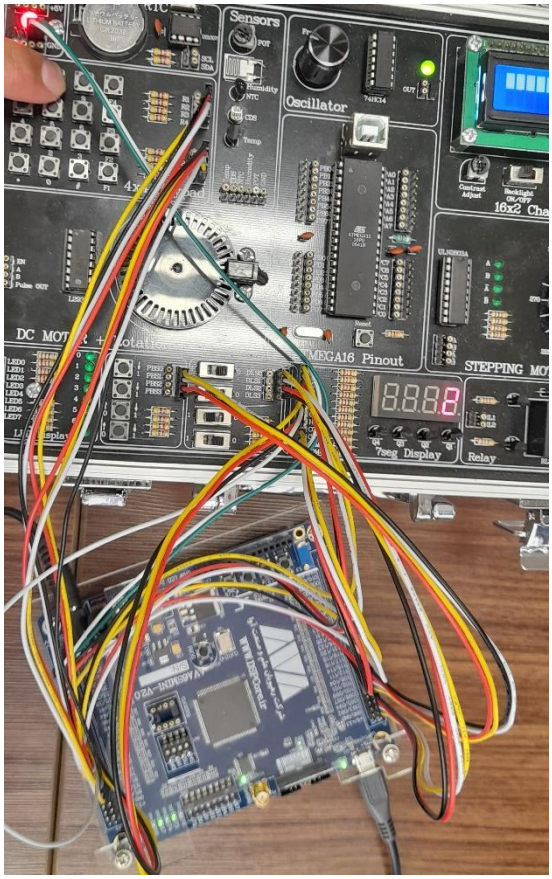
شکل ۱۴-۶: CLOCK_DIVIDER

```
122 #NET "cathode<4>"    LOC = "P35";
123 #NET "cathode<5>"    LOC = "P33";
124 #NET "cathode<6>"    LOC = "P30";
125 #NET "cathode<7>"    LOC = "P27";
126
127 NET "read<0>"     LOC = "P50";
128 NET "read<1>"     LOC = "P56";
129 NET "read<2>"     LOC = "P58";
130 NET "read<3>"     LOC = "P61";
131
132 NET "scan<0>"     LOC = "P64";
133 NET "scan<1>"     LOC = "P67";
134 NET "scan<2>"     LOC = "P75";
135 NET "scan<3>"     LOC = "P79";
136
137 NET "dispn<0>"    LOC = "P24";
138 NET "dispn<1>"    LOC = "P22";
139 NET "dispn<2>"    LOC = "P17";
140 NET "dispn<3>"    LOC = "P15";
141
142 NET "disp<0>"     LOC = "P47";
143 NET "disp<1>"     LOC = "P45";
144 NET "disp<2>"     LOC = "P43";
145 NET "disp<3>"     LOC = "P40";
146 NET "disp<4>"     LOC = "P35";
147 NET "disp<5>"     LOC = "P33";
148 NET "disp<6>"     LOC = "P30";
149 NET "disp<7>"     LOC = "P27";
150 #NET "io_EXPIO<86>"     LOC = "P24";
151
152 #NET "digit<0>"      LOC = "P24";
153 #NET "digit<1>"      LOC = "P22";
154 #NET "digit<2>"      LOC = "P17";
155 #NET "digit<3>"      LOC = "P15";
156 #NET "io_EXPIO<92>"     LOC = "P15";
```

شکل ۱۵-۶: نمونه فایل USF. (علاوه بر پین های بالا کلاک هم باید تعریف شود)

اگر کیپد را به طور صحیح به سون سیگمنت ها وصل کنید، نتایج زیر را می توانید مشاهده کنید :

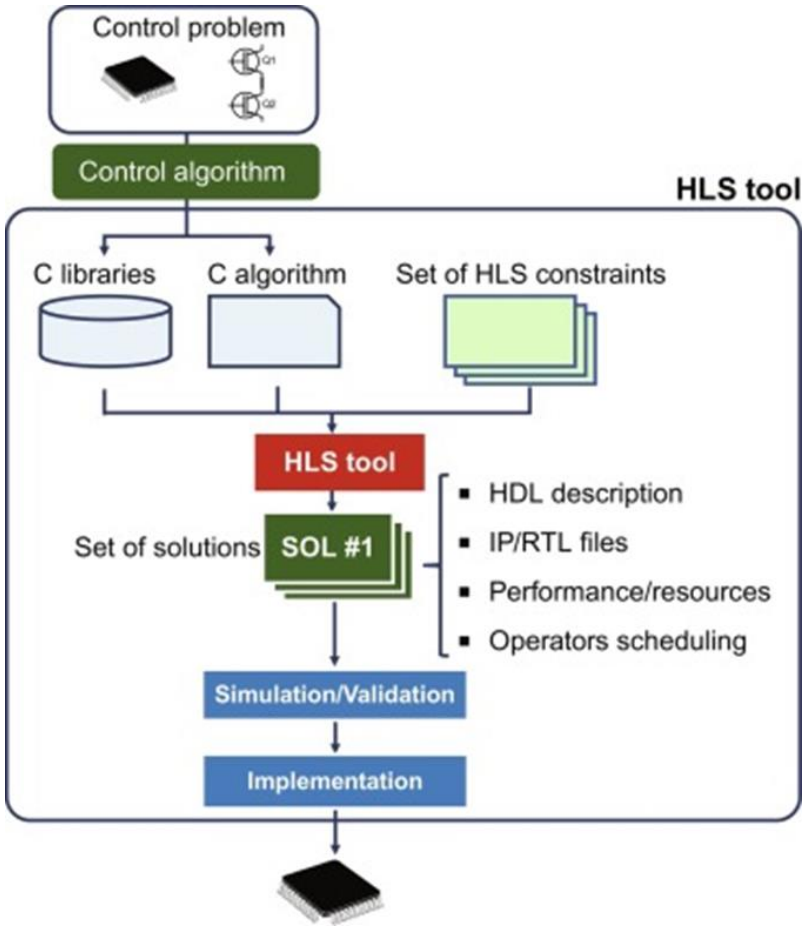




آزمایش هفتم: آشنایی با سنتز سطح بالا HIGH LEVEL SYNTHESIS (HLS)

تمامی کد هایی که با وریلاگ نوشته می شوند سنتز پذیر نیستند و همچنین پیاده سازی برخی محاسبات ریاضی همانند تقسیم، توان و توابع مثلثاتی و... بسیار پیچیده و نیازمند تسلط بسیار بالایی می باشد. اما با استفاده از سنتز سطح بالا بسیاری از این مشکل ها تا حدودی حل می شود؛ به طوری که با استفاده از زبان Cیا++C می توان به راحتی با فراخوانی کتابخانه های مورد نیاز تمامی عملیات ریاضی را در حد یک خط کد انجام داد. ولی در عوض معایبی که دارد، این نوع سنتز در مقایسه با سنتز معمولی بخش های بسیار زیادی از FPGA را اشغال می کند و همچنین با توجه به در حال توسعه بودن آن برای برخی از مباحث قابل استفاده نمی باشد.

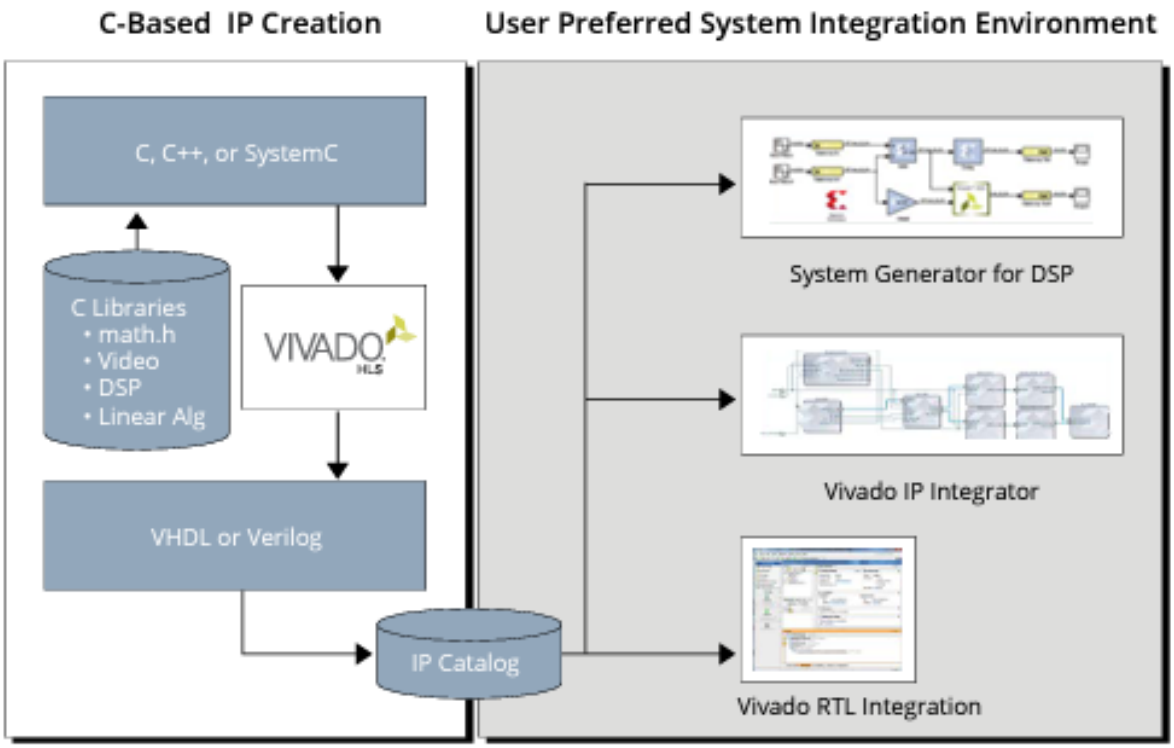
در واقع می توان با استفاده از کتابخانه های زبان C، عملیات مورد نظر را به زبان C نوشت و به کمک HLS این کد ها را به RTL تبدیل کرد و در ISE ویا VIVADO استفاده کرد.



شکل ۱-۷: دیاگرام HLS

سنتز سطح بالا را می توان با ابزار های مختلفی انجام داد، یکی از این ابزار ها vivado HLS می باشد که در کنار خود vivado نصب می شود.

شکل ۲-۷: VIVADO HLS

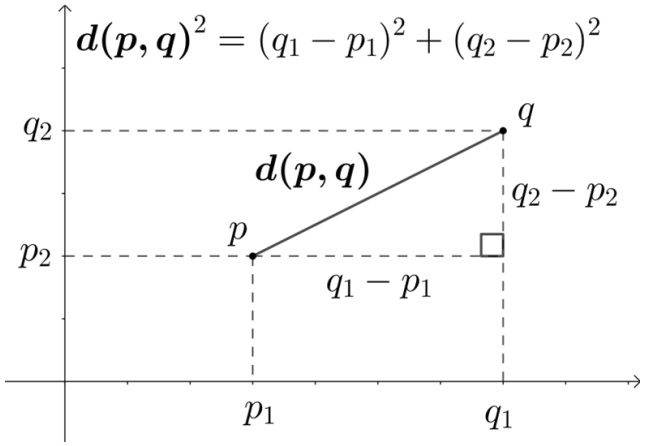


شکل ۳-۷: فرایند تبدیل کد C به IP CATALOG در VIVADO HLS

برای درک بهتر مفاهیم بالا، مثال زیر را به کمک HLS پیاده سازی می کنیم.

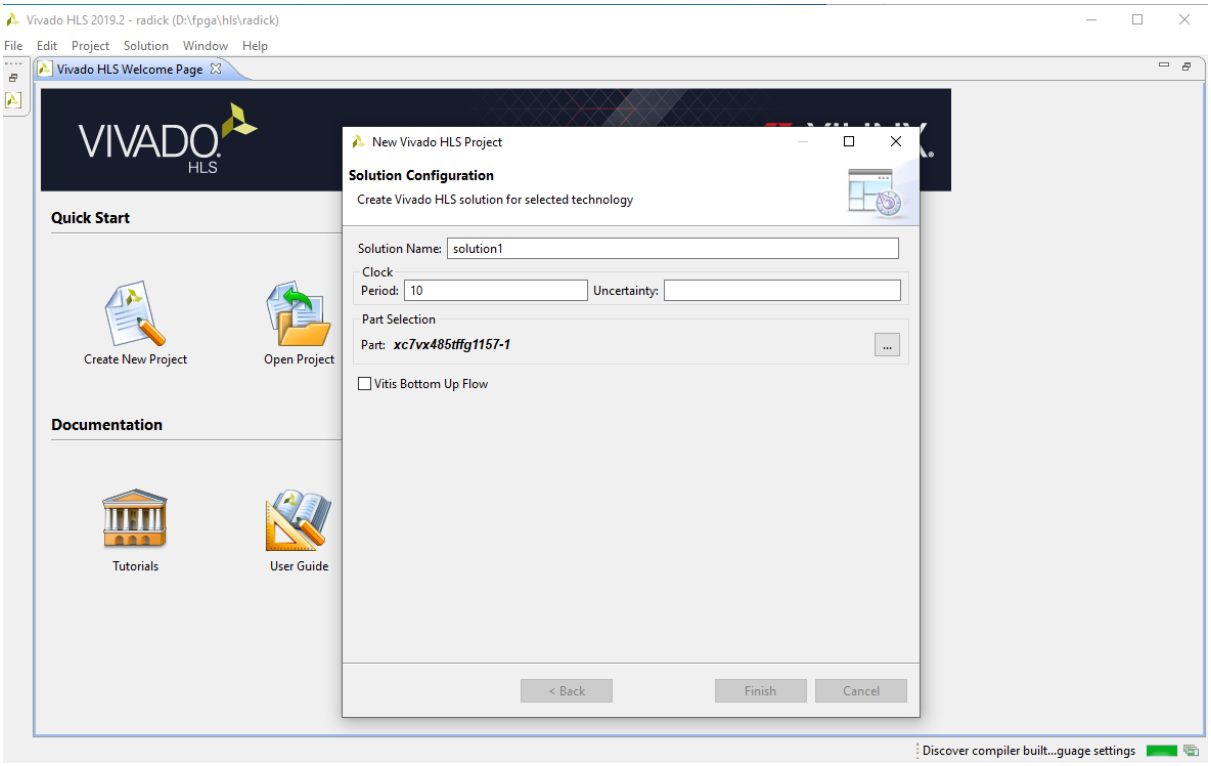
با توجه به شکل ٤-۷ فاصله ی دو نقطه از هم از رابطه ی زیر به دست می آید. هدف از این مثال پیاده سازی این رابطه ی ریاضی می باشد.

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$



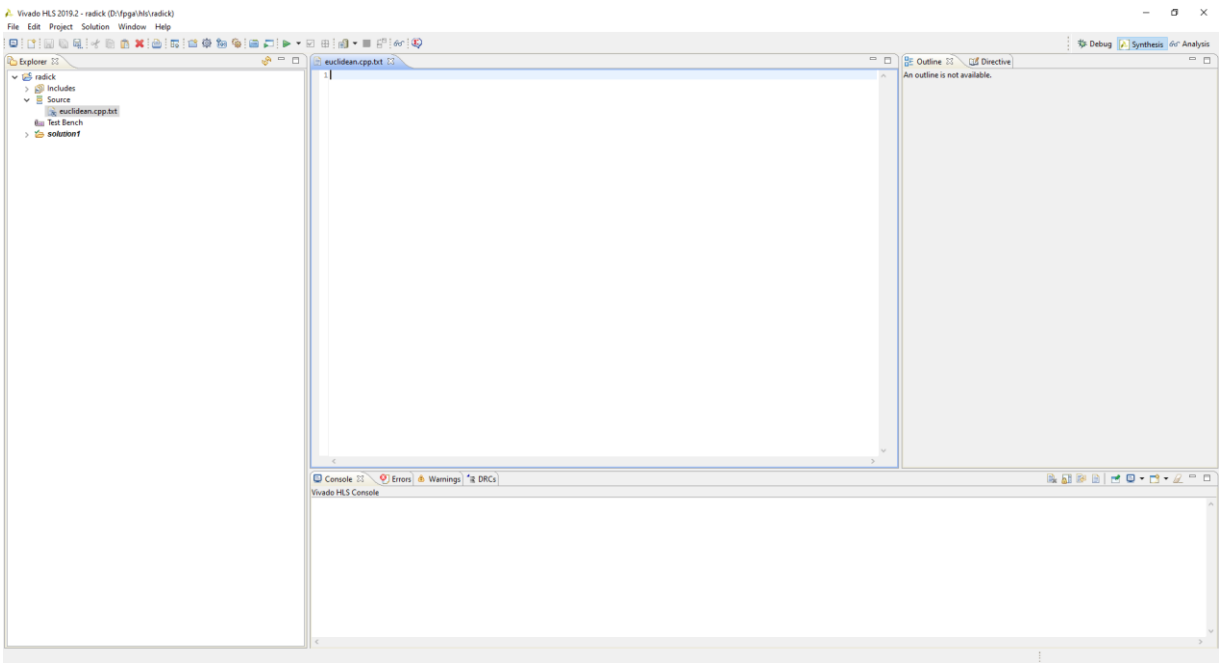
شکل ٤-٧: فاصله ی بین دو نقطه

برای این کار به صورت زیر عمل می کنیم. vivado HLS را اجرا می کنیم؛ همانند شکل ٥-٧ یک پروژه ی جدید باز می کنیم و تنظیمات مربوط به محل ذخیره سازی را انجام می دهیم؛ همچنین یک فایل text با پسوندcpp. ایجاد می کنیم و بهvivado HLS معرفی میکنیم.



شکل ٥-٧: ایجاد پروژه در VIVADO HLS

نهایتا یک محیط همانند شکل ٦-٧ ایجاد می شود که می توان کد ها را به زبان C نوشت.



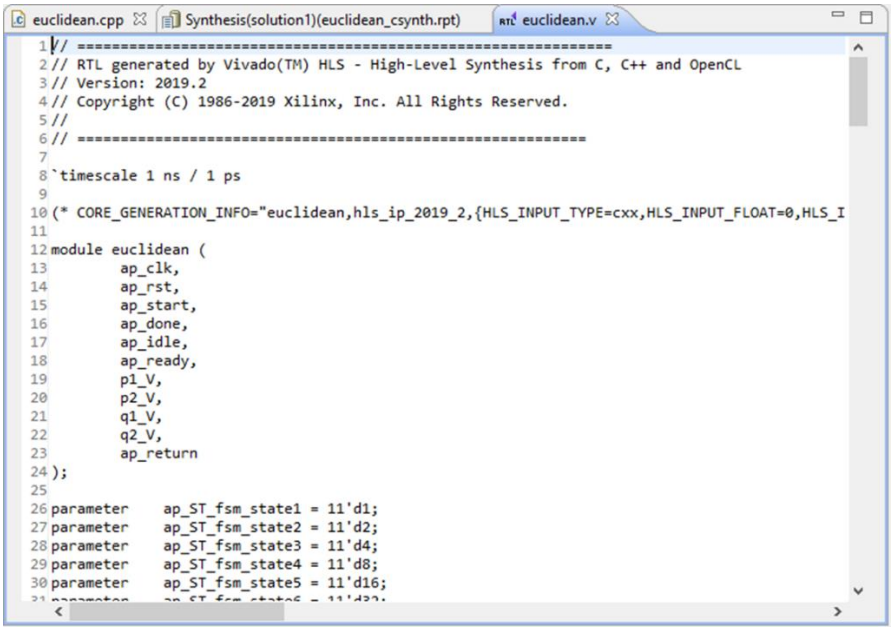
شکل ٦-٧: محیط VIVADO HLS

حال کد های مربوط به فرمول بالا را همانند شکل ٧-٧ می نوبسیم:

```
#include "hls_math.h"
#include <math.h>
#include <ap_fixed.h>

typedef ap_fixed<8, 4> fix_t;

fix_t euclidean(fix_t p1, fix_t p2, fix_t q1, fix_t q2) {
    return hls::sqrt(((p1 - q1) * (p1 - q1)) + ((p2 - q2) * (p2 - q2)));
}
```

شکل ٧-٧: کدهای اصلی

بعد از نوشتن کد ها، آنها را سنتز می کنیم و برنامه کد ها را همانند شکل ٨-٧ به RTL تبدیل می کند.



شکل ۸-۷: کد RTL

فایل Test Bnech مربوطه را هم می توان به صورت نوشت:

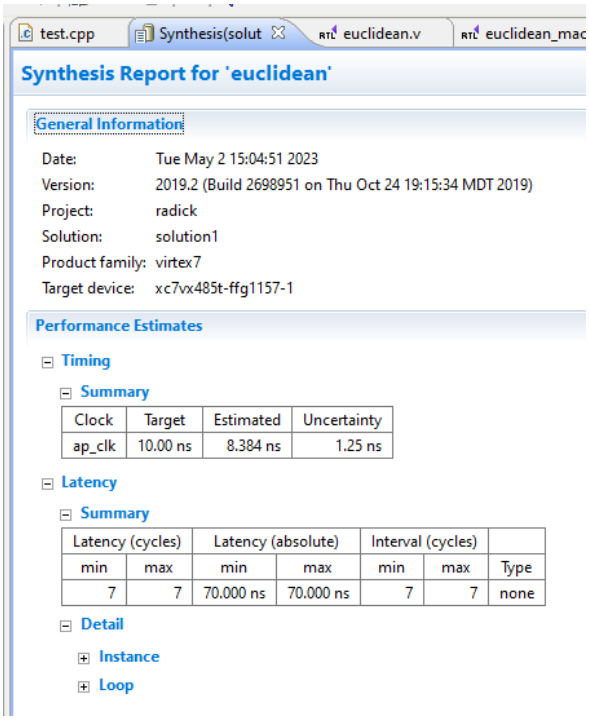
```
#include <stdio.h>
#include <ap_fixed.h>

typedef ap_fixed<8, 4> fix_t;
fix_t euclidean(fix_t p1, fix_t p2, fix_t q1, fix_t q2);

int main() {
    fix_t p1 = 1.25;
    fix_t p2 = 0.75;
    fix_t q1 = 3.5;
    fix_t q2 = 7;

    fix_t dist = euclidean(p1, p2, q1, q2);

    printf("Distance Between 2 Numbers is:%f", float(dist));

    return 0;
}
```

شکل ۹-۷: Test Bnech

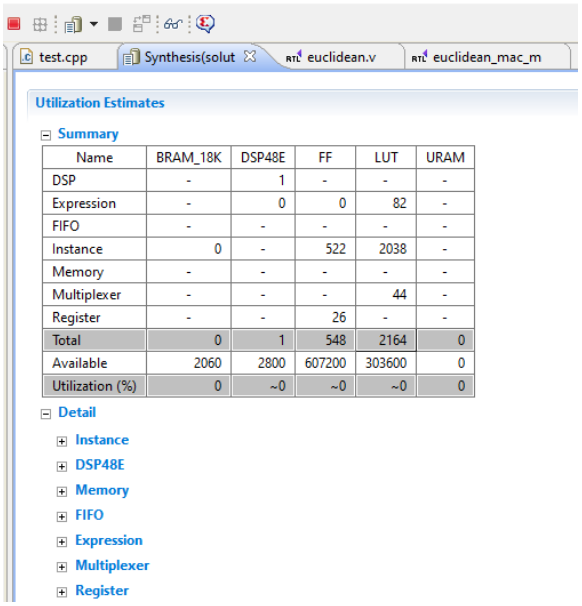
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../test.cpp in debug mode
   Generating csim.exe
Distance Between 2 Numbers is:6.625000
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

شکل ۱۰-۷: خروجی Test Bnech

بعد از سنتز کردن پنجره ای همانند شکل ۱۱-۷ باز می شود که اطلاعات مفیدی درباره ی برد، تعداد فلیپ فلاپ های استفاده شده و یا LUT و ... در آن ها توضیح داده شده است.

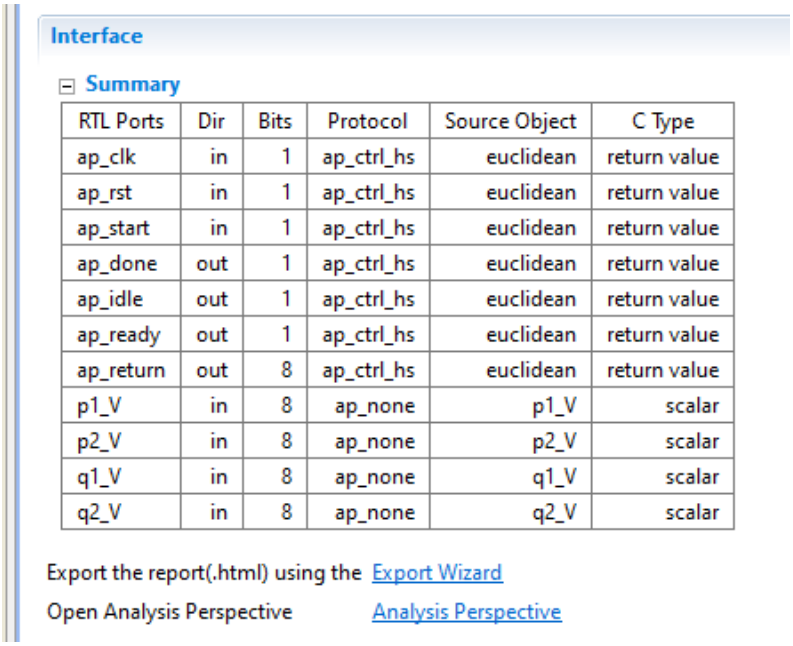


شکل ۱۱-۷: اطلاعات برد



Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	0	0	82	-
FIFO	-	-	-	-	-
Instance	0	-	522	2038	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	44	-
Register	-	-	26	-	-
Total	0	1	548	2164	0
Available	2060	2800	607200	303600	0
Utilization (%)	0	~0	~0	~0	0

شکل ۱۲-۷: موارد استفاده شده از برد



RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	euclidean	return value
ap_rst	in	1	ap_ctrl_hs	euclidean	return value
ap_start	in	1	ap_ctrl_hs	euclidean	return value
ap_done	out	1	ap_ctrl_hs	euclidean	return value
ap_idle	out	1	ap_ctrl_hs	euclidean	return value
ap_ready	out	1	ap_ctrl_hs	euclidean	return value
ap_return	out	8	ap_ctrl_hs	euclidean	return value
p1_V	in	8	ap_none	p1_V	scalar
p2_V	in	8	ap_none	p2_V	scalar
q1_V	in	8	ap_none	q1_V	scalar
q2_V	in	8	ap_none	q2_V	scalar

شکل ۱۳-۷: ورودی خروجی ها

بعد از سنتز کردن یک پنجره ی دیگری به اسم Co Simulation هم باز می شود؛ که خروجی وریلاگ را با خروجی C مقایسه می کند، اگر خروجی وریلاگ همان خروجی سنتز شده ی؛ کدC نوشته شده باشد؛ همانند شکل ۷-۱٤ نتیجه ی pass را بر می گرداند در غیر این صورت fail بر می گرداند.

Cosimulation Report for 'euclidean'

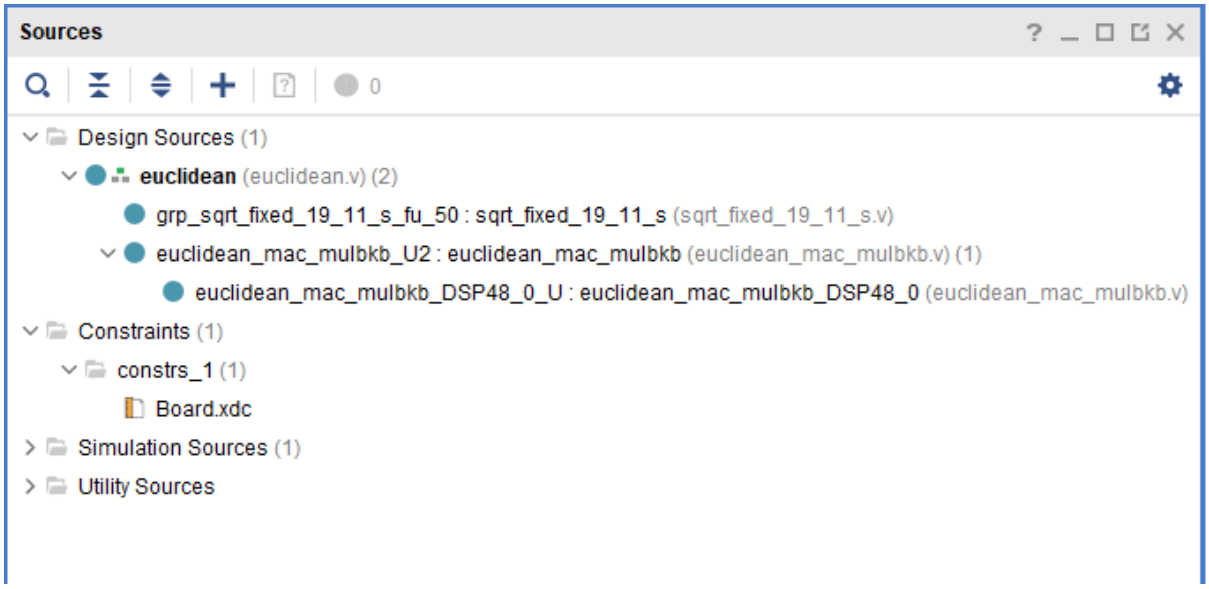
Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	10	10	10	NA	NA	NA

Export the report(.html) using the Export Wizard

شکل ۷-۱٤: Co Simulation

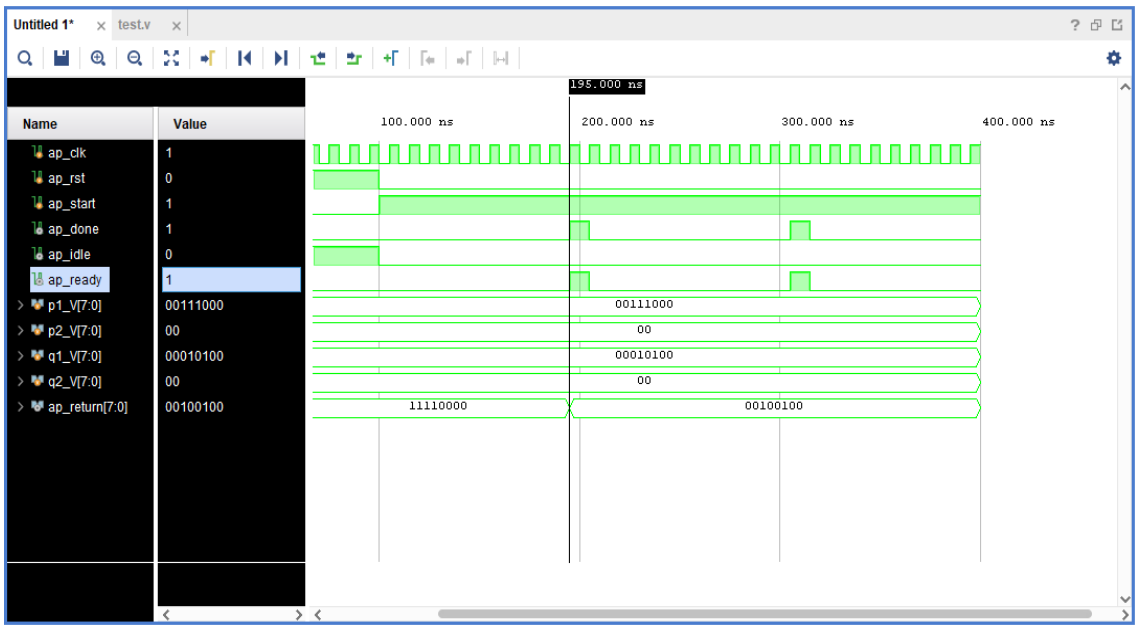
و در نهایت کد های RTL سنتز شده را با ISE ویا vivado همانند شکل ۷-۱۵ اجرا می کنیم و برای اطمینان از صحت فرایند، فایل Test Bnech را برای آن می نویسیم و با ورودی های مختلف آن را امتحان می کنیم.



شکل ۷-۱۵: کد ها در VIVADO

```
`timescale 1ns / 1ps
module test();
reg   ap_clk = 0;
reg   ap_rst = 1;
reg   ap_start = 0;
wire   ap_done;
wire   ap_idle;
wire   ap_ready;
reg  [7:0] p1_V = 8'b00111000;
reg  [7:0] p2_V = 8'd0;
reg  [7:0] q1_V = 8'b00010100;
reg  [7:0] q2_V = 8'd0;
wire  [7:0] ap_return;
euclidean   dut(
.ap_clk(ap_clk),
.ap_rst(ap_rst),
.ap_start(ap_start),
.ap_done(ap_done),
.ap_idle(ap_idle),
.ap_ready(ap_ready),
.ap_return(ap_return),
.p1_V(p1_V),
.p2_V(p2_V),
.q1_V(q1_V),
.q2_V(q2_V)
);
always #5 ap_clk = ~ap_clk;
initial begin
#100;
ap_rst = 0;
ap_start = 1;
#300;
$stop;
end
endmodule
```

شکل ۱۶-۷: کد های TEST BNECH

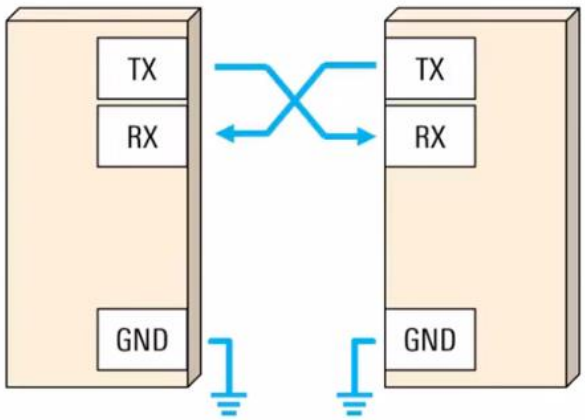


شکل ۱۷-۷: نتایج TEST BNECH

آزمایش هشتم: **آشنایی با پروتکل UART (TRANSMITTER)**

"UART"مخفف **Universal Asynchronous Receiver-Transmitter** به معنی **فرستنده و گیرنده سریال ناهمزمان جهانی** است. UART یک پروتکل سریال کامل دوطرفه است که در اصل یک تراشه طراحی شده برای انجام ارتباطات ناهمزمان می باشد. عملکرد UART این است که داده های ورودی و خروجی را به جریان دودویی سریال تبدیل می کند. داده های سری ۸ بیتی دریافت شده از دستگاه با استفاده از مبدل سریال به موازی به شکل موازی وداده های موازی دریافت شده از CPU با استفاده از مبدل موازی به سریال به شکل سریال تبدیل می شوند.

UART بسیار ساده است و فقط از دو سیم بین فرستنده و گیرنده برای ارسال و دریافت در هر دو جهت استفاده می کند.



شکل ۱-۸: ساختار UART

UART می تواند simplex یا Half Duplex یا Full Duplex باشد.

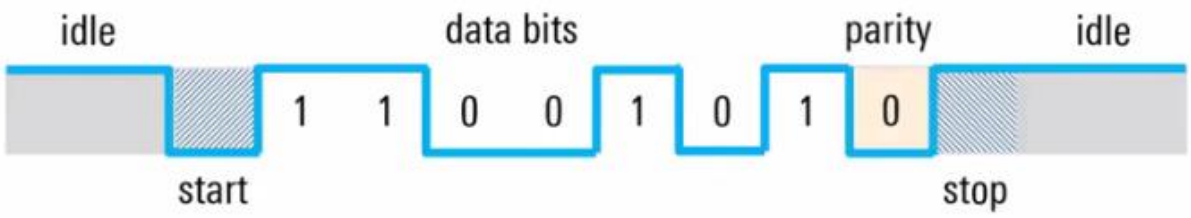
UART ناهمزمان است و فرستنده و گیرنده clock مشترکی ندارند؛ بنابراین فرستنده و گیرنده باید:

از یک ساختار و پارامتر های مشخص برای انتقال اطلاعات استفاده کنند.

با یک سرعت معلوم اطلاعات را ارسال کنند.

Common UART baud rates
٤٨٠٠
٩٦٠٠
١٩٢٠٠
٥٧٦٠٠
١١٥٢٢

ساختار UART به صورت زیر می باشد:



شکل ۲-۸: ساختار UART

بیت های شروع و پایان:

وقتی در بیت های دیتا idle برابر یک بود؛ سپس صفر شود، نشان دهنده ی شروع می باشد. و اگر idle از صفر به یک برود، نشان دهنده ی اتمام دیتا است.

بیت های اطلاعات :

بلافاصله بعد از شروع بیت های اطلاعات می آید؛ که اغلب ۸ بیت هستند. بیت دیتا می تواند کد اسکی باشد که ما در این پروژه از آنها استفاده می کنیم.

بیت Parity :

این بیت اختیاری است و برای تشخیص خطا با توجه به تعداد بیت های ۱ عمل می کند. این بیت قابلیت تشخیص خطا را دارد ولی نمی تواند آنرا اصلاح کند و فقط درخواست ارسال مجدد می کند.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

شکل ۳-۸: کد های اسکی

حال هدف این است که به کمک FPGA کد های اسکی را از FPGA و از طریق پورتکل UART به کامپیوتر بفرستیم و از طریق ترمینال آنهارا مشاهده کنیم.

می توانید از کد های زیر استفاده کنید :

```
`timescale 1ns / 1ps
module Uart_Tx(
    input clk,
    input [7:0] data,
    input transmit,
    input reset,
    output reg TxD
    );
    //internal variables
    reg [3:0] bit_counter; //counter to count the 10 bits (8bit data, 2bit satrt and stop)
    reg [13:0] baudrate_counter; //5,208 counter = clk/BR = 50MHZ/9600
    reg [9:0] shiftright_register; //10bits that are serialy transmitted from Uart
    reg state, next_state; //idle mode and transmitting mode
    reg shift; //shift signal to start shifting the bits in the UART
    reg load; //load signal to start loading the data into the shift right register also add the stop and start bits
    reg clear; //reset the bit counter for UART transmission
    //Uart transmission
    always @(posedge clk) begin
        if(reset) begin
            state <= 0; //state is idle
            bit_counter <= 0; //for bit transmission is reset to 0
            baudrate_counter <= 0;
        end
        else begin
            baudrate_counter <= baudrate_counter + 1;
            if(baudrate_counter == 14'd5207) begin
                state <= next_state; //state changes from idle to transmitting
                baudrate_counter <= 0; //resetting the counter
```

شکل ٤-٨: کد های پروژه

```
                if(load)
                    shiftright_register <= {1'b1, data, 1'b0}; //the data is loaded into the register 10-bits
                if(clear) begin
                    baudrate_counter <= 0; //resetting the counter
                    bit_counter <= 0;
                end
                if(shift) begin
                    shiftright_register <= shiftright_register >> 1; //start shifting the data and transmitting bit by bit
                    bit_counter <= bit_counter + 1;
                end
            end
        end
    end
    //Mealy Machine
    always @(posedge clk) begin
        load  <= 0;
        shift <= 0;
        clear <= 0;
        TxD   <= 1; //there is no transmission in progress
        case(state)
            0: begin
                if(transmit)begin //transmit button is pressed
                    next_state <= 1; //switches to transmission state
                    load <= 1;
                    shift <= 0;
                    clear <= 0;
```

شکل ٥-٨: کد های پروژه

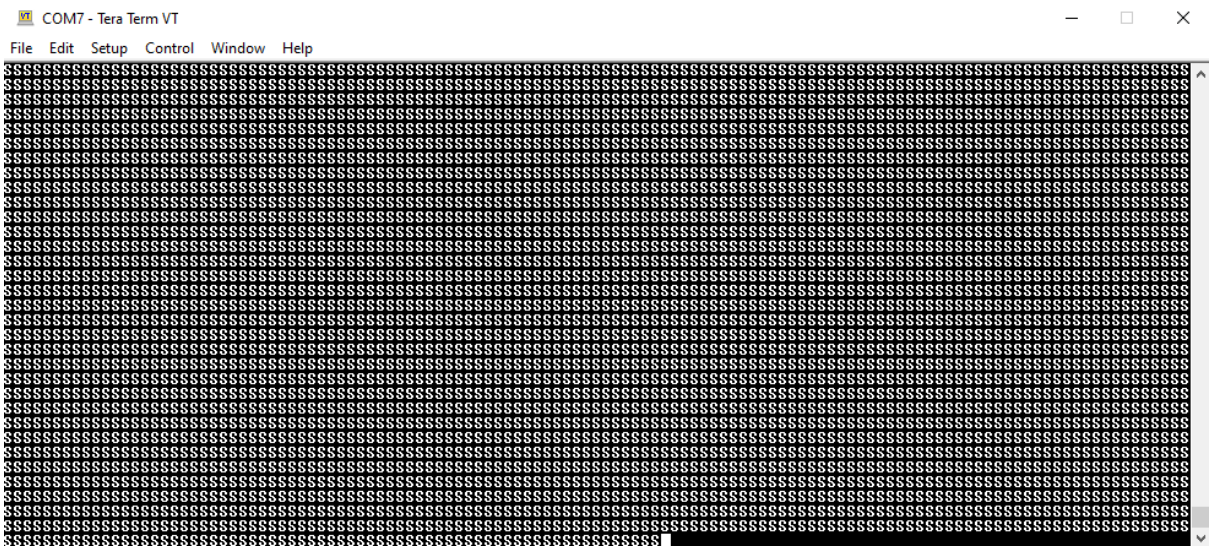
```
                    clear <= 0;
                end
                else begin
                    next_state <= 0; //stays at the idle mode
                    TxD <= 1; //no transmission
                end
            end
            1: begin //transmitting state
                if(bit_counter == 10)begin
                    next_state <= 0; //switch from transmission mode to idle mode
                    clear <= 1;
                end
                else begin
                    next_state <= 1; //stay in the trasmitt state
                    TxD <= shiftright_register[0];
                    shift <= 1;//make sure you are continuing shifting the data, new bit arrives at right most bit
                end
            end
            default: next_state <= 0;
        endcase
    end

endmodule

```

شکل ٦-٨: کد های پروژه

```
NET "clk"       LOC = "P14";
#NET "i_OSC2_OUT"    LOC = "P55";
#NET "clk"      LOC = "P88";
NET "TxD"          LOC = "P12";
#NET "o_USB_RXD"         LOC = "P12";
#NET "io_EXPIO<1>"       LOC = "P10";
#NET "io_EXPIO<2>"       LOC = "P9";
#NET "io_EXPIO<3>"       LOC = "P8";
NET "data<0>"      LOC = "P79";
NET "data<1>"      LOC = "P80";
NET "data<2>"      LOC = "P81";
NET "data<3>"      LOC = "P82";
NET "data<4>"      LOC = "P83";
NET "data<5>"      LOC = "P84";
NET "data<6>"      LOC = "P85";
NET "data<7>"      LOC = "P87";
NET "reset"     LOC = "p8";
NET "transmit"     LOC = "p7";
#NET "clo"      LOC = "P9";
#NET "io_EXPIO<8>"       LOC = "P1";
#NET "io_EXPIO<9>"       LOC = "P144";
#NET "io_EXPIO<10>"      LOC = "P143";
```

شکل ۷–۸: فایل USF.

اگر همه ی مراحل فوق به درستی انجام گرفته باشد، لازم است برنامه ی Tera Term VT در کامپیوتر نصب شود تا کد های اسکی ارسالی از FPGA توسط این برنامه نمایش داده شود. نمونه ای از نتایج در شکل ۸–۸ نشان داده شده است.



شکل ۸–۸: نتایج نهایی

در این آزمایش ارسال اطلاعات با پروتکل UART به طور کامل انجام گرفت.

تمرین: سعی کنید با استفاده از پروتکل UART اطلاعات را از کامپیوتر دریافت کنید و آنرا نمایش دهید.

آزمایش نهم: **آشنایی با پروتکل UART (RESEIVER)**

در آزمایش قبلی به طور کامل با پروتکل UART آشنا شدیم؛ و همچنین بخش ارسال آن را هم مورد برسی قرار دادیم؛ و توانستیم از برد FPGA اطلاعاتی را به کامپیوتر ارسال کنیم و در ترمینال مربوطه آنها را مشاهده کنیم.

حال در این آزمایش هدف این است که بخش دریافت را تحقق ببخشیم به این صورت که اطلاعاتی را از طریق کامپیوتر ارسال کنیم و با FPGA آن را دریافت کنیم و نمایش دهیم.

در این آزمایش هم باز از کد اسکی استفاده می کنیم و یک مقدار مشخص از کامپیوتر ارسال می کنیم تا مقدار مربوط به همان کد اسکی توسط LED های روی برد(که در فایل usf. تعریف شده است) روشن شود.

می توانید از کد های زیر استفاده کنید :

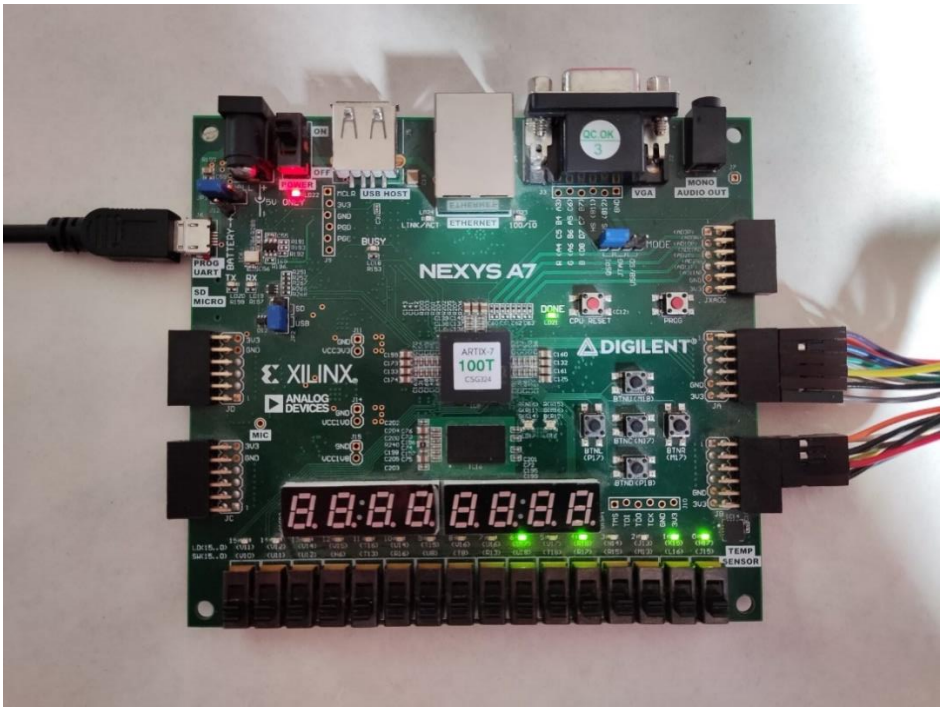
```
1  `timescale 1ns / 1ps
2  module Receiver_RxD(
3      input clk,reset,RxD,
4      output [7:0] RxData);
5      reg shift;
6      reg state;
7      reg nextstate;
8      reg [3:0] bit_counter;
9      reg [1:0] sample_counter;
10     reg [13:0] baudrate_counter;
11     reg [9:0] rxshift_reg;
12     reg clear_bitcounter;
13     reg inc_bitcounter;
14     reg inc_samplecounter;
15     reg clear_samplecounter;
16
17     parameter clk_freq = 50_000_000;
18     parameter baud_rate = 9_600;
19     parameter div_sample = 4;
20     parameter div_counter = clk_freq/(baud_rate * div_sample);
21     parameter mid_sample = (div_sample/2);
22     parameter div_bit = 10;
23
24     assign RxData = rxshift_reg[8:1];
25
26     always @(posedge clk) begin
27         if(reset) begin
28             state <= 0;
29             bit_counter <= 0;
30             baudrate_counter <= 0;
31             sample_counter <= 0;
32         end
33         else begin
34             baudrate_counter <= baudrate_counter + 1;
35             if(baudrate_counter >= div_counter - 1)begin
36                 baudrate_counter <= 0;
37                 state <= nextstate;
38                 if(shift) rxshift_reg <= {RxD,rxshift_reg[9:1]};
39                 if(clear_samplecounter) sample_counter <= 0;
40                 if(inc_samplecounter) sample_counter <= sample_counter + 1;
41                 if(clear_bitcounter) bit_counter <= 0;
42                 if(inc_bitcounter) bit_counter <= bit_counter + 1;
```

شکل ۱-۹: بخش اول کد های RECEIVER

```
40                  if(inc_samplecounter) sample_counter <= sample_counter + 1;
41                  if(clear_bitcounter) bit_counter <= 0;
42                  if(inc_bitcounter) bit_counter <= bit_counter + 1;
43              end
44          end
45      end
46
47      always @(posedge clk) begin
48          shift <= 0;
49          clear_samplecounter <= 0;
50          inc_samplecounter <= 0;
51          clear_bitcounter <= 0;
52          inc_bitcounter <= 0;
53          nextstate <= 0;
54          case(state)
55              0: begin
56                  if(RxD) begin
57                      nextstate <= 0;
58                  end
59                  else begin
60                      nextstate <= 1;
61                      clear_bitcounter <= 1;
62                      clear_samplecounter <= 1;
63                  end
64              end
65              1: begin
66                  nextstate <= 1;
67                  if(sample_counter == mid_sample - 1) shift <= 1;
68                  if(sample_counter == div_sample - 1) begin
69                      if(bit_counter == div_bit - 1) begin
70                          nextstate <= 0;
71                      end
72                      inc_bitcounter <= 1;
73                      clear_samplecounter <= 1;
74                  end
75                  else inc_samplecounter <= 1;
76              end
77              default nextstate <= 0;
78          endcase
79      end
80
81  endmodule
```

شکل ۲-۹: بخش دوم کدهای RECEIVER

اگر مراحل به درستی انجام گرفته باشند، نتایجی همانند شکل ۳-۹ مورد انتظار است.



شکل ۳-۹: نتایج نهایی (نمایش 'S' که کد اسکی آن برابر HEX۵۳)

آزمایش دهم: سلسله مراتب MEMORY

حافظه ها یکی از مهمترین منابع درون تراشه FPGA هستند و بدون آن ها جریان طراحی به شکلی که امروزه انجام می شود، امکان پذیر نبود، حافظه ها درون FPGA به دو دسته تقسیم می شوند.

•حافظه های بلوکی یا Block RAM

•حافظه های توزیع شده یا Distributed RAM

حافظه های بلوکی:

حافظه های بلوکی یکسری منابع اختصاصی سخت افزاری هستند و به صورت بلوک هایی با ظرفیت ذخیره سازی چند ۱۰ کیلوبیت روی تراشه های FPGA قابل فراخوانی هستند. ظرفیت هر کدام از بلوک های حافظه در تراشه های سری ۷ برابر با ۳۶ کیلوبیت است.

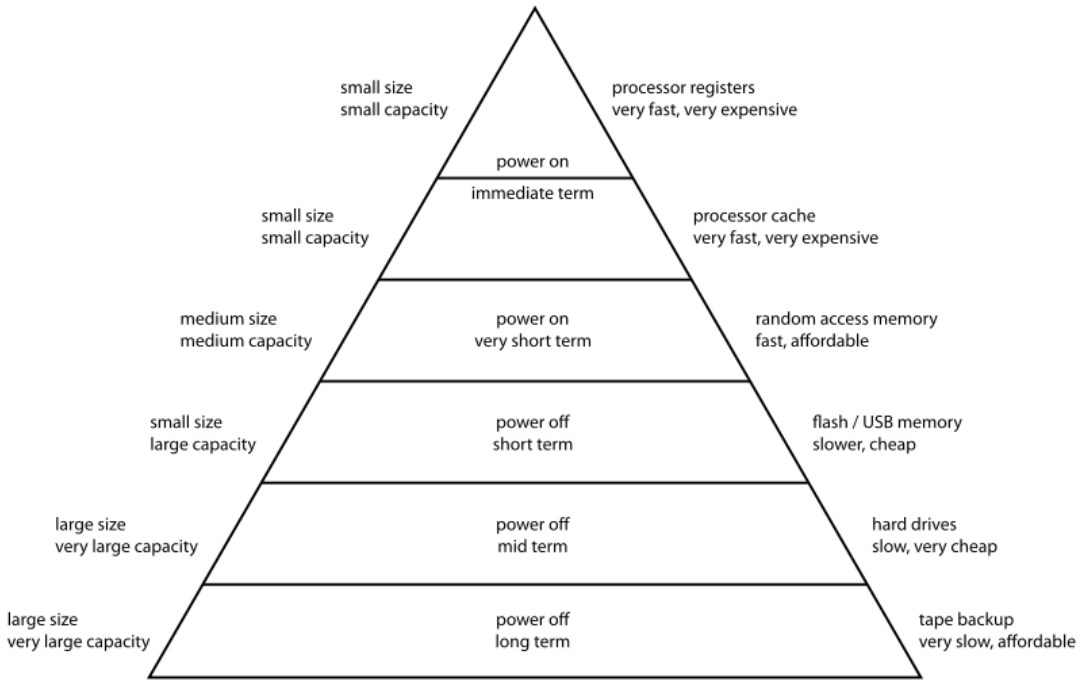
حافظه های توزیع شده:

برخلاف حافظه های بلوکی حافظه های توزیع شده، منابع اختصاصی سخت افزاری نیستند، و با استفاده از LUT ها ساخته می شوند و می توانند در هر جایی از تراشه پیاده سازی شوند. این حافظه ها قابلیت ذخیره سازی تعداد محدودی بیت دارند و در مواردی که نیاز به ذخیره سازی حجم کمی از داده ها وجود داشته باشد، مورد استفاده قرار می گیرند. ۱۰-۱-

: MEMORY HIERARCHY

در سیستم های کامپیوتری، سلسله مراتب حافظه، ذخیره سازی کامپیوتر را به سلسله مراتبی بر اساس زمان پاسخ تفکیک می کند. از آنجایی که زمان پاسخ، پیچیدگی و ظرفیت به هم مرتبط هستند، سطوح ممکن است با عملکرد و فناوری‌های کنترلی نیز متمایز شوند. هر چقدر حافظه بزرگ تر باشد؛ پیچیدگی آن بیشتر است و سرعت پاسخ دهی کمتری دارد؛ در مقابل هرچقدر حافظه کوچکتر باشد، زمان پاسخ دهی آن کم تر است و به عبارتی سریع تر است.

در شکل ۱-۱۰ سلسله مراتب حافظه FPGA از لحاظ مختلف مورد برسی قرار گرفته اند.

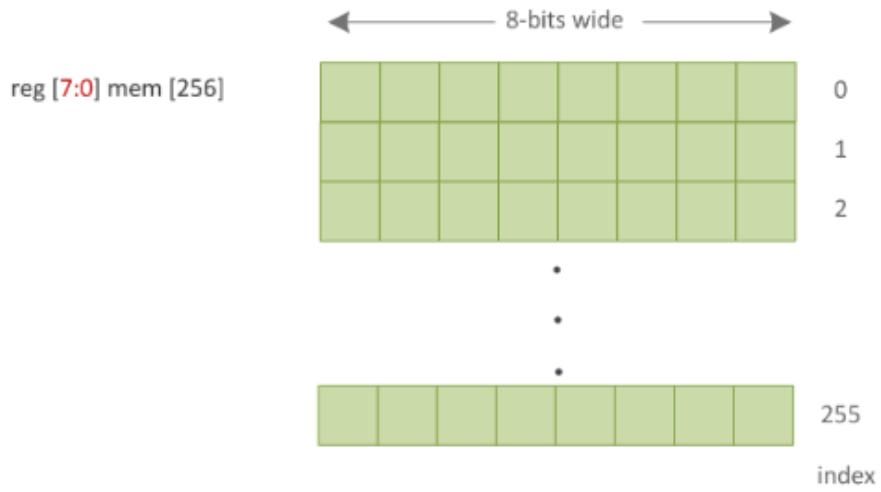


شکل ۱-۱۰: سلسله مراتب حافظه

روش های مختلفی برای تعریف حافظه در FPGA وجود دارد؛ یکی از این روش ها این است که کدی همانند شکل ۲-۱۰ نوشته شود و ریجستر را به صورت آرایه ی دوبعدی تعریف کنیم.

```
1   timescale 1ns / 1ps
2
3   module memory(
4       input clk,
5       input rstn,
6       input [7:0] addr,
7       input wr,
8       input sel,
9       input [7:0] wdata,
10      output [7:0] rdata
11      );
12
13      reg [7:0] register [0:255];
14      integer i;
15
16      always @(posedge clk) begin
17          if(!rstn) begin
18              for(i=0;i<256;i=i+1) begin
19                  register[i] <=8'd0;
20              end
21          end
22          else begin
23              if(sel&wr) begin
24                  register[addr] <= wdata;
25              end
26              else
27                  register[addr] <= register[addr];
28          end
29      end
30
31      assign rdata = (sel&~wr)? register[addr] : 0;
32  endmodule
```

شکل ۲-۱۰: تعریف ریجیستر



شکل ۳-۱۰: ریجستر تعریف شده

بعد از این که کد بالا را سنتز کردیم؛ بخش های مختلف استفاده شده از هسته در برد دانشگاه به صورت شکل ٤-۱۰ می باشد.

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	2,048	11,440	17%	
Number used as Flip Flops	2,048			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	2,856	5,720	49%	
Number used as logic	2,856	5,720	49%	
Number using O6 output only	2,848			
Number using O5 output only	0			
Number using O5 and O6	8			
Number used as ROM	0			
Number used as Memory	0	1,440	0%	
Number of occupied Slices	738	1,430	51%	
Number of MUXCYs used	0	2,860	0%	
Number of LUT Flip Flop pairs used	2,856			
Number with an unused Flip Flop	808	2,856	28%	
Number with an unused LUT	0	2,856	0%	
Number of fully used LUT-FF pairs	2,048	2,856	71%	
Number of unique control sets	1			

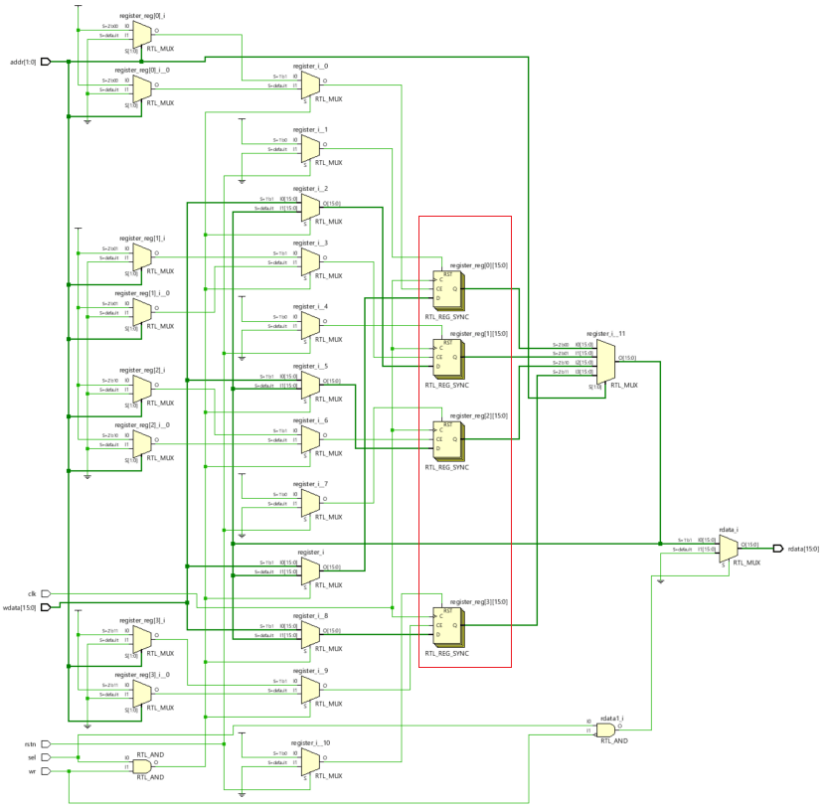
شکل ٤-۱۰: موارد مصرف شده از برد

همان طوری که در شکل ٤-١٠ دیده می شود، یک حافظه ی کوچک از فلپ فلاپ و LUT وlogic blockو... استفاده کرده است که برای مثال حدودا به اندازه ی ٤٩ درصد از LUT استفاده کرده است که حجم بسیار زیادی می باشد.

حال اگر حافظه را کوچک تر کنیم؛ یعنی اندازه ی ریجستر را کوچک کنیم می بینیم که مقدار استفاده از LUT و فلیپ فلاپ ها به شدت کاهش می یابد و این نشان می دهد که در تعریف ریجستر ها این دو بخش هستند که مورد استفاده قرار می گیرند.

```
1   timescale 1ns / 1ps
2
3   module memory(
4       input clk,
5       input rstn,
6       input [1:0] addr,
7       input wr,
8       input sel,
9       input [15:0] wdata,
10      output [15:0] rdata
11      );
12
13      reg [15:0] register [0:3];
14      integer i;
15
16      always @(posedge clk) begin
17          if(!rstn) begin
18              for(i=0;i<4;i=i+1) begin
19                  register[i] <=0;
20              end
21          end
22          else begin
23              if(sel&wr) begin
24                  register[addr] <= wdata;
25              end
26              else
27                  register[addr] <= register[addr];
28          end
29      end
30
31      assign rdata = (sel&~wr)? register[addr] : 0;
32  endmodule
```

شکل ٥-١٠: تعریف ریجستر کوچکتر



شکل ٦-١٠: شماتیک مداری (GATE LEVEL)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	64	11,440	1%	
Number used as Flip Flops	64			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	88	5,720	1%	
Number used as logic	88	5,720	1%	
Number using O6 output only	80			
Number using O5 output only	0			
Number using O5 and O6	8			
Number used as ROM	0			
Number used as Memory	0	1,440	0%	
Number of occupied Slices	24	1,430	1%	
Number of MUXCYs used	0	2,860	0%	
Number of LUT Flip Flop pairs used	88			
Number with an unused Flip Flop	24	88	27%	
Number with an unused LUT	0	88	0%	
Number of fully used LUT-FF pairs	64	88	72%	
Number of unique control sets	1			

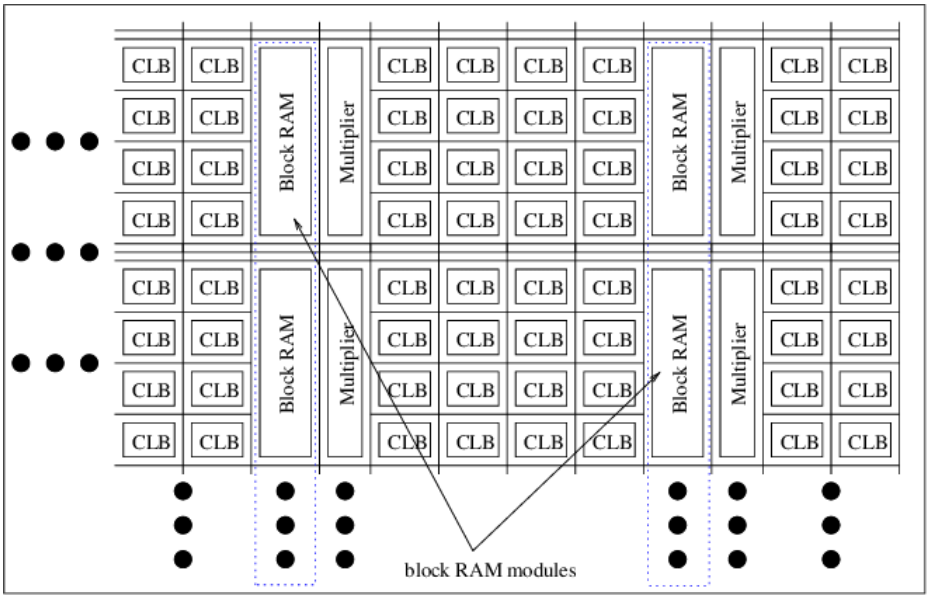
شکل ٧-١٠: بخش های مختلف استفاده شده در ریجیستر کوچکتر

همانطوری که در شکل ٧-١٠ دیده می شود؛ مقدار استفاده از LUT ها به ١ درصد کاهش پیدا کرده است.

برای استفاده در حافظه های کوچک ولی سریع روش بالا مناسب است ولی برای حافظه های بزرگ از BRAM استفاده می کنیم.

: BRAM (BLOCK RAM)

حافظه‌های بلوکی یاBlock RAM ها، منابع مستقل دیجیتالی درFPGA ها هستند که در بخش‌های خاصی از FPGA از قبل به صورت سخت‌افزاری تعبیه[۱۰]شده‌اند. این حافظه‌ها در اندازه‌های مشخصی وجود دارند و می‌توانند توسط پیاده‌ساز استفاده شوند. این حافظه در داخل هسته ی FPGA قرار دارد.



شکل ۸-۱۰: حافظه ی BRAM

برای استفاده از این حافظه یکی از روش ها استفاده از IP ها می باشد.

: IP یا Intellectual Property

IP یا به طور کامل تر Core IP، مخفف کلمه ی Core Property Intellectual است و به ماجولهایی اشاره دارد که از قبل توسط افراد یا شرکت های دیگر ی آماده و تست شده اند و میتوان از عملکرد صحیح آنها مطمئن بود.

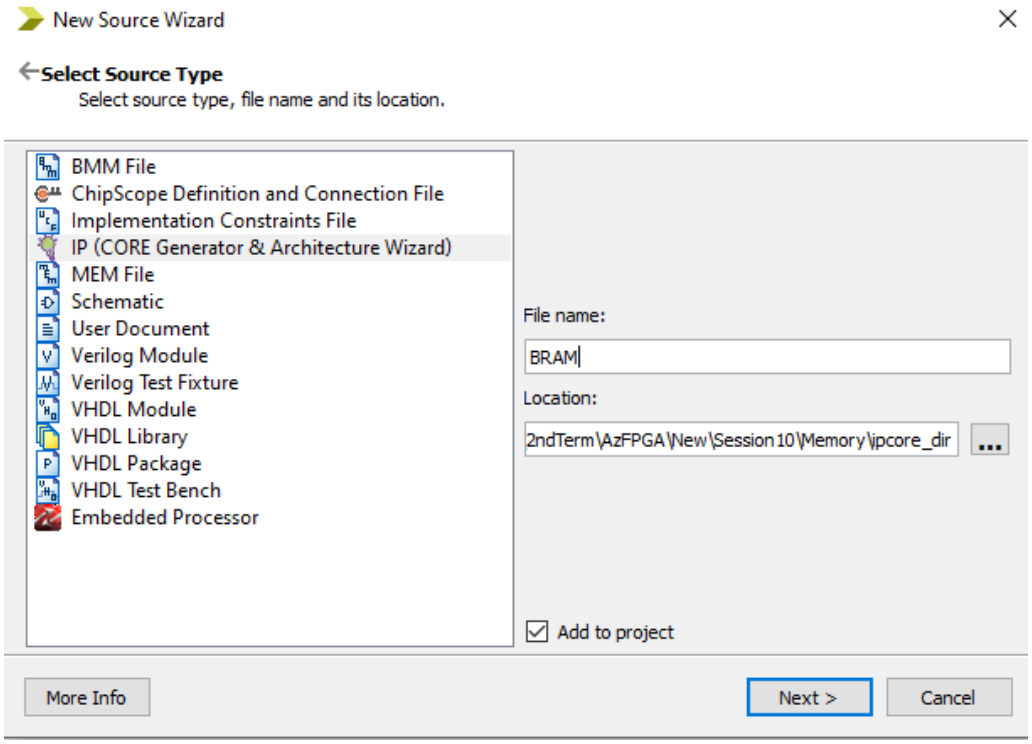
استفاده ازIP ها می تواند دو مزیت مهم زیر را به دنبال داشته باشد:

• افزایش سرعت روند پیاده سازی سیستم ها

• صرفه جویی در هزینه ی توسعه ی پروژه

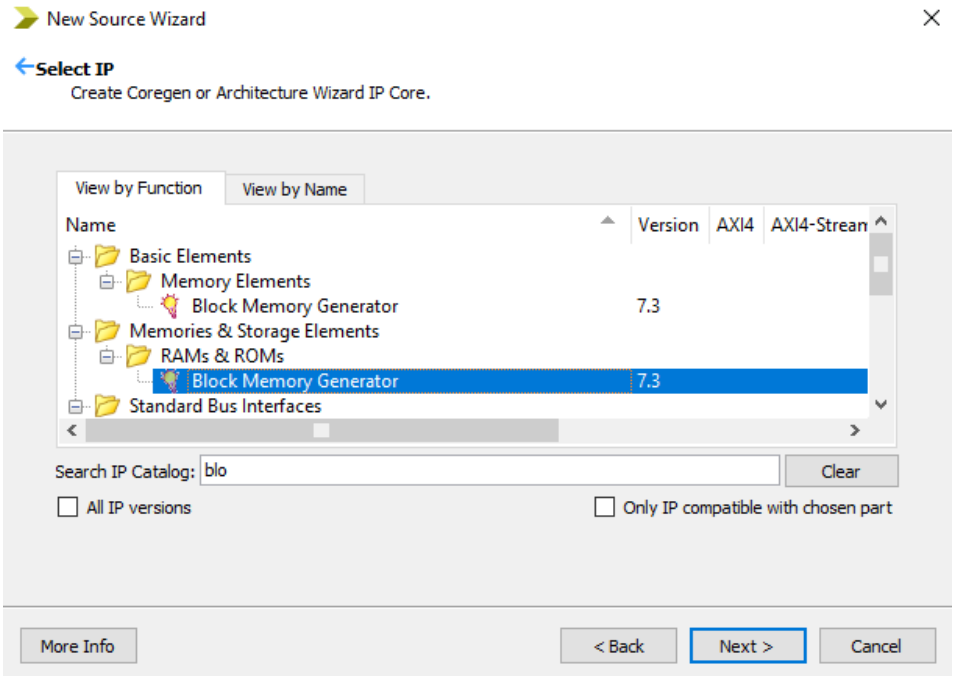
حال در نرم افزار ISE به صورت زیر از این حافظه استفاده می کنیم:

ابتدا یک پروژه باز می کنیم و به صورت شکل ۹-۱۰ ادامه می دهیم:

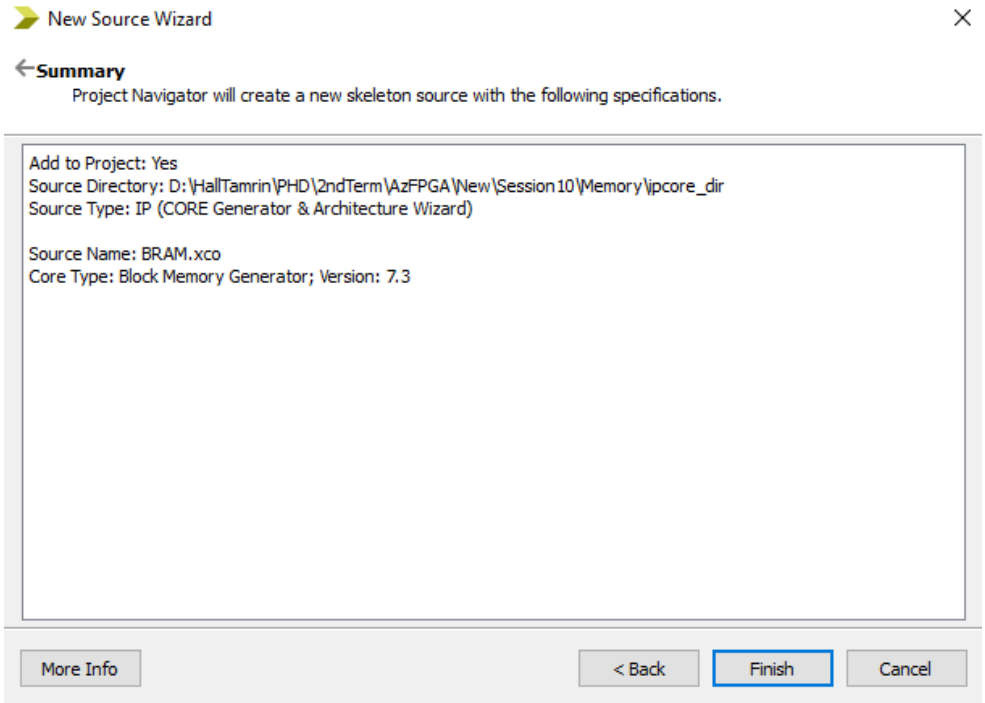


شکل ۹-۱۰: انخاب BRAM

در ادامه پنجره ای همانند شکل ۱۰-۱۰ باز می شود که next می زنیم و بعد از آن finish(شکل ۱۱-۱۰) .



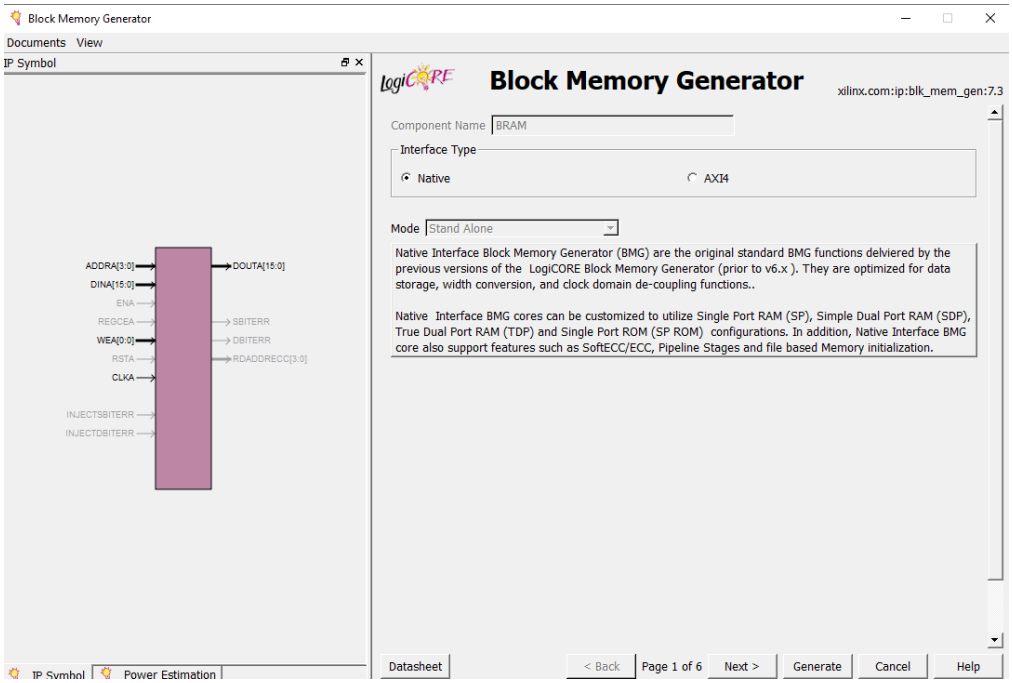
شکل ۱۰-۱۰



شکل ۱۱-۱۰

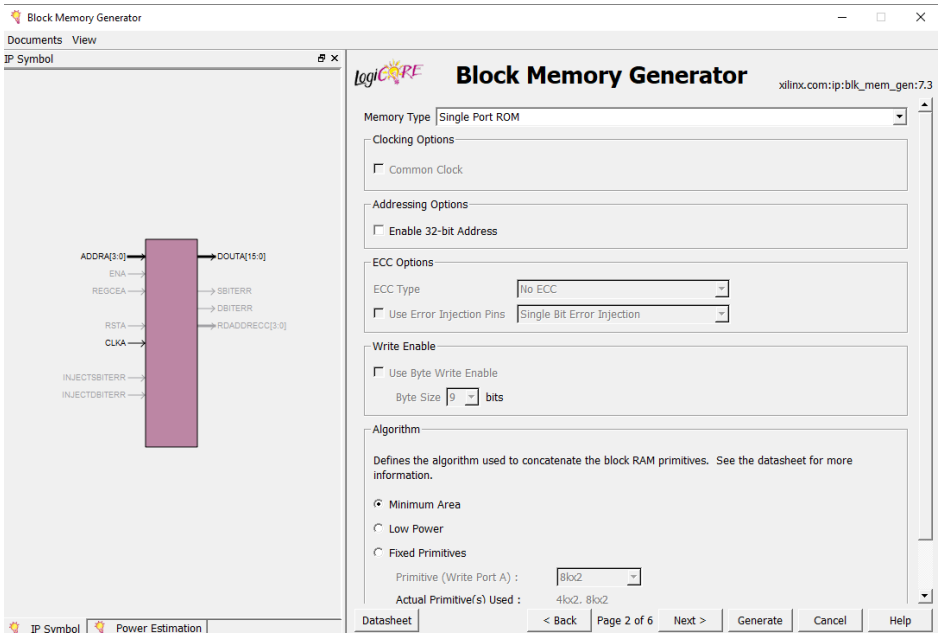
در ادامه به تعریف حافظه می پردازیم. در پنجره ی باز شده همانند شکل ۱۲-۱۰ تنظیمات انجام شده را انجام می

دهیم:



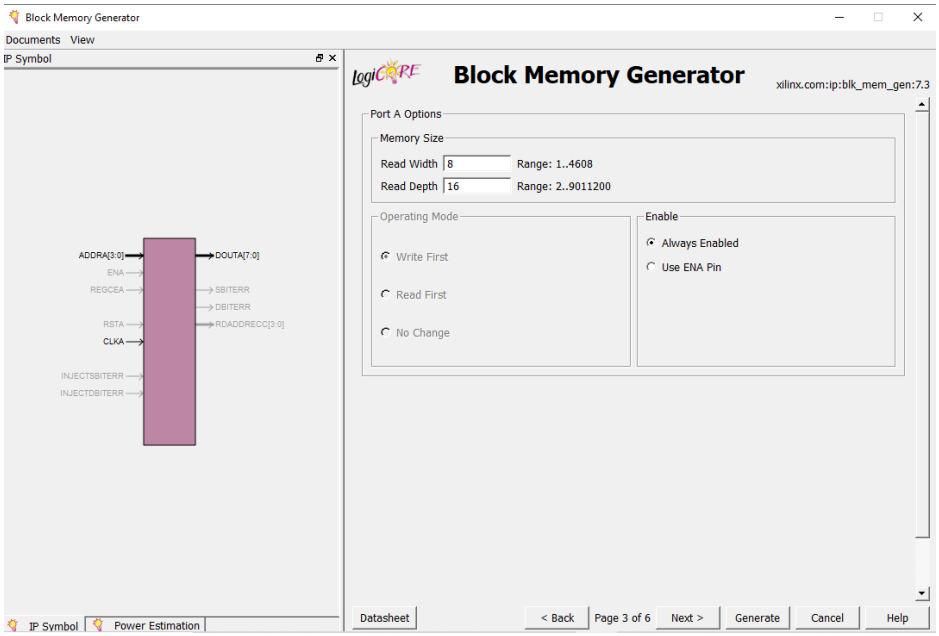
شکل ۱۲-۱۰

در پنجره ی بعدی نوع حافظه را انتخاب می کنیم که ما اینجا فعلا از ROM استفاده می کنیم؛ دیگر موارد را هم همانند شکل ۱۳-۱۰ تنظیم می کنیم:



شکل ۱۳-۱۰

در پنجره ی بعدی همانند شکل ۱۴-۱۰ می توان اندازه ی حافظه را مشخص کرد.



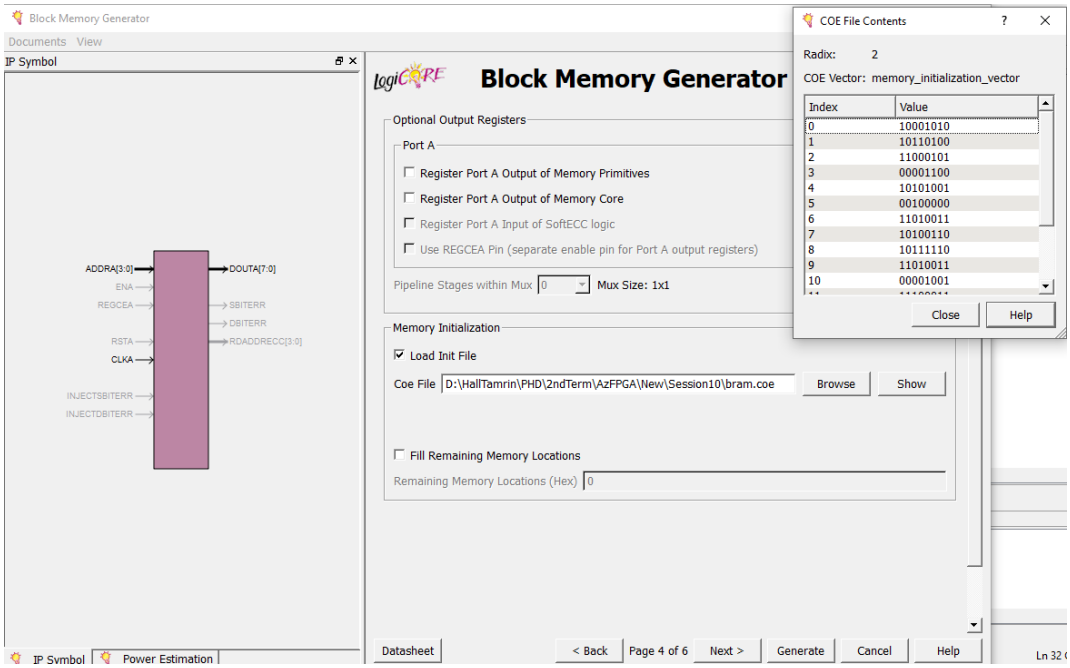
شکل ۱۴-۱۰

در ادامه باید حافظه ای که درست کرده ایم را پر کنیم. برای این کار یک فایل text درست میکنیم که پسوند آن باید coe. باشد (شکل ۱۵-۱۰).

```
1  memory_initialization_radix=2;
2  memory_initialization_vector=
3  10001010,
4  10110100,
5  11000101,
6  00001100,
7  10101001,
8  00100000,
9  11010011,
10 10100110,
11 10111110,
12 11010011,
13 00001001,
14 11100011,
15 01100011,
16 01010001,
17 11100011,
18 11100111;
```

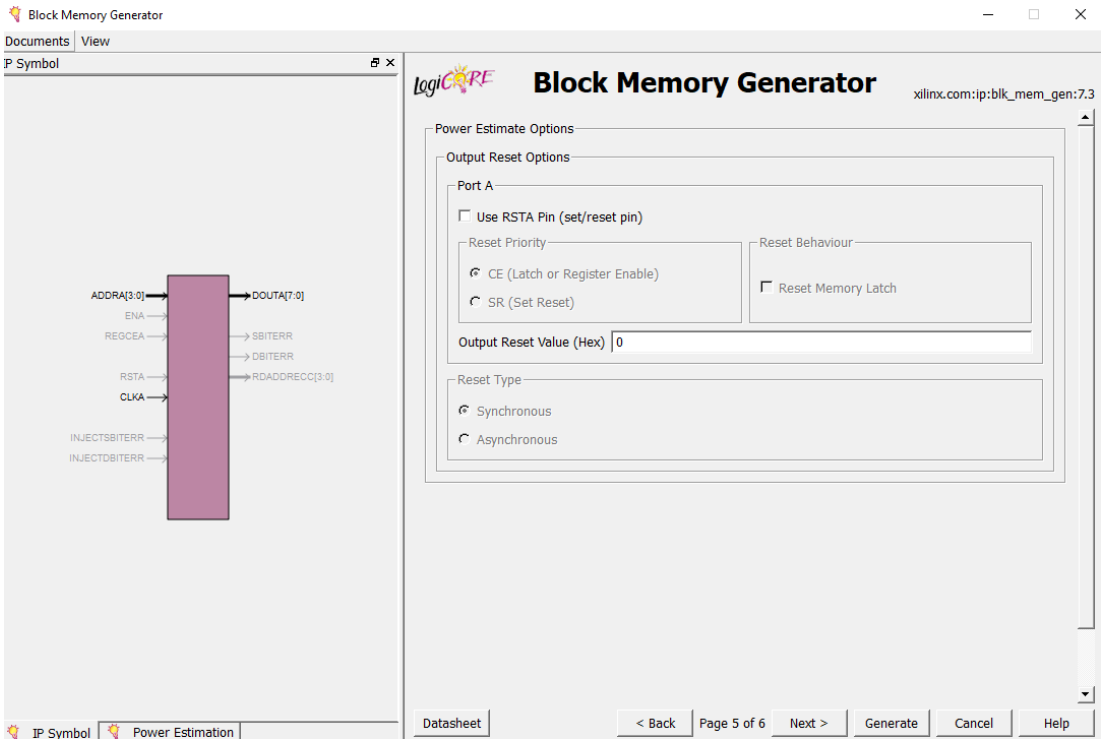
شکل ۱۵-۱۰: نمونه حافظه ی درست شده

حال در پنجره ی باز شده همانند شکل ۱۶-۱۰ آدرس فایل text را می دهیم.

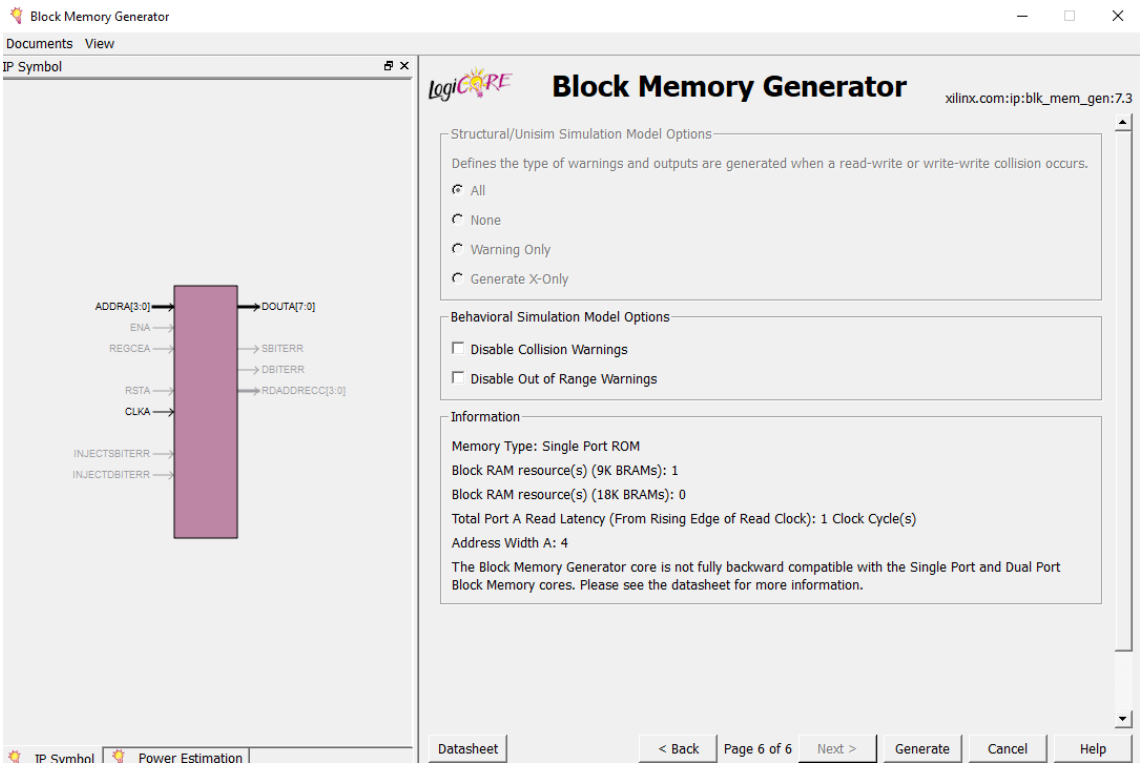


شکل ۱۶-۱۰

دو پنجره ی بعدی را به ترتیب next وGenerate می زنیم.

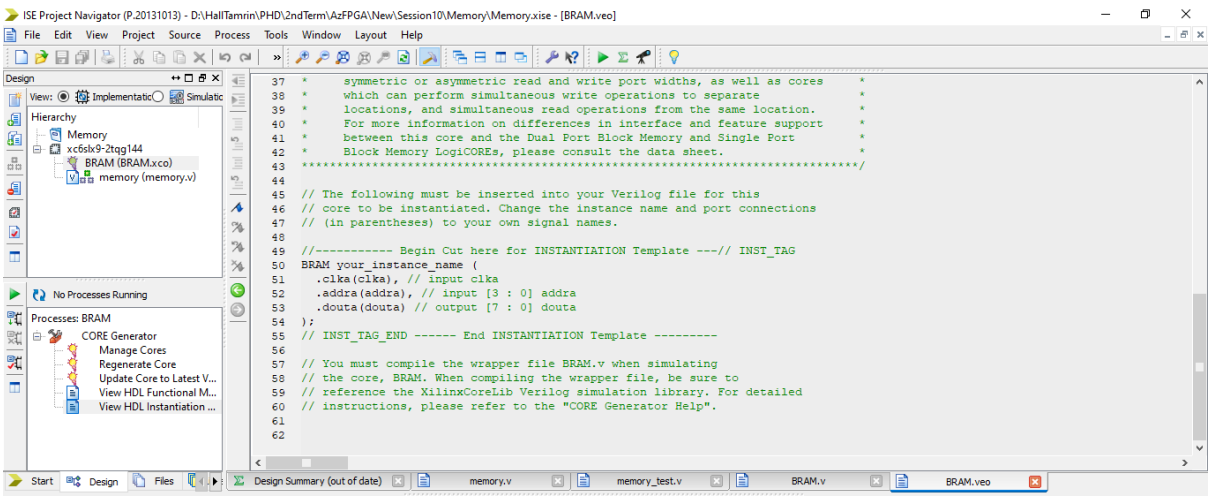


شکل ۱۷-۱۰



شکل ۱۸–۱۰

نهایتا IP آماده می شود.



شکل ۱۹–۱۰: IP CATALOG FOR BRAM IN ISE

در شکل ۱۹–۱۰ روی …view HDL instantiation می زنیم و برای خواندن دیتا کد شکل ۲۰–۱۰ را می نویسیم.

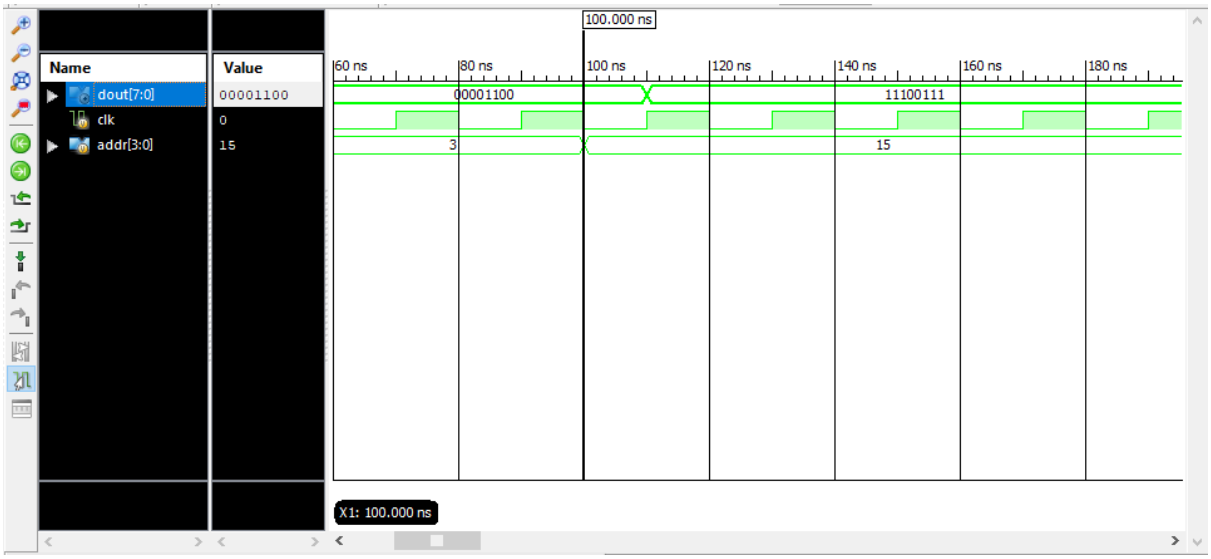
```
1  `timescale 1ns / 1ps
2
3  module BRAM_top(
4      input clk,
5      input [3:0] addr,
6      output [7:0] dout
7       );
8
9      BRAM dut (
10     .clka(clk), // input clka
11     .addra(addr), // input [3 : 0] addra
12     .douta(dout) // output [7 : 0] douta
13     );
14
15 endmodule
```

شکل ۲۰-۱۰: کد های اصلی

برای اطمینان از عملکرد کار؛ فایل تست بنچ را به صورت شکل ۲۱-۱۰ می نویسیم.

```
1  `timescale 1ns / 1ps
2
3  module BRAM_test;
4
5     // Inputs
6     reg clk = 0;
7     reg [3:0] addr;
8
9     // Outputs
10    wire [7:0] dout;
11    always #10 clk = ~clk;
12    // Instantiate the Unit Under Test (UUT)
13    BRAM_top uut (
14       .clk(clk),
15       .addr(addr),
16       .dout(dout)
17    );
18
19    initial begin
20       // Initialize Inputs
21       addr = 4'd3;
22       #100;
23       addr = 4'd15;
24       #100;
25       // Add stimulus here
26
27    end
28 endmodule
```

شکل ۲۱-۱۰: فایل تست



شکل ۲۲-۱۰: نتایج نهایی

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	11,440	0%	
Number of Slice LUTs	0	5,720	0%	
Number of occupied Slices	0	1,430	0%	
Number of MUXCYs used	0	2,860	0%	
Number of LUT Flip Flop pairs used	0			
Number of bonded IOBs	13	102	12%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	1	64	1%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	1	16	6%	
Number used as BUFGs	1			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	4	0%	
Number of ILOGIC2/ISERDES2s	0	200	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	200	0%	
Number of OLOGIC2/OSERDES2s	0	200	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	

شکل ۲۳-۱۰: بخش های مختلف استفاده شده در BRAM

همان طوری که در شکل ۲۳-۱۰ مشاهده می شود، تنها یک درصد از block RAM ها استفاده شده است. درحالی

که اگر همین حجم از دیتا را با ریجستر ها ذخیره می کردیم، حجم بسیار زیادی را می گرفت.

حافظه ی DDR :

حافظه ی DDR (Double Data Rate)، نوعی از حافظه است که فقط در برخی از برد های FPGA وجود دارد(برد دانشگاه ندارد)؛ این حافظه در خارج از هسته به صورت مستقل وجود دارد. این حافظه از طریق مسیرهایی در برد به هسته متصل شده است و با تنظیم فایل usf. این حافظه به هسته وصل می شود و قابل استفاده است. ۱۰-۳-

شکل ۲٤-۱۰: حافظه ی DDR روی برد

انواع مختلفی از حافظه ها DDR, DDR۲, DDR۳و حافظه DDR٤ با ظرفیتی در محدوده ۲~۱٦ GB موجود هستند.

ما ادامه ی آزمایش را بر مبنای برد Nexys ٤DDR انجام می دهیم که دارای حافظه ی DDR۲ می باشد.

اگر داخل سایت این برد بشویم اطلاعات زیر را می توانیم در مورد حافظه ی آن به دست بیاوریم.

3 Memory

The Nexys A7 board contains two external memories: a 1Gib (128MiB) DDR2 SDRAM and a 128Mib (16MiB) non-volatile serial Flash device. The DDR2 modules are integrated on-board and connect to the FPGA using the industry standard interface. The serial Flash is on a dedicated quad-mode (x4) SPI bus. The connections and pin assignments between the FPGA and external memories are shown below.

3.1 DDR2

The Nexys A7 includes one Micron MT47H64M16HR-25:H DDR2 memory component, creating a single rank, 16-bit wide interface. It is routed to a 1.8V-powered HR (High Range) FPGA bank with 50 ohm controlled single-ended trace impedance. 50 ohm internal terminations in the FPGA are used to match the trace characteristics. Similarly, on the memory side, on-die terminations (ODT) are used for impedance matching.

For proper operation of the memory, a memory controller and physical layer (PHY) interface needs to be included in the FPGA design. There are two recommended ways to do that, which are outlined below and differ in complexity and design flexibility.

The straightforward way is to use the Digilent-provided DDR-to-SRAM adapter module which instantiates the memory controller and uses an asynchronous SRAM bus for interfacing with user logic. This module provides backward compatibility with projects written for older Nexys-line boards featuring a CellularRAM instead of DDR2. It trades memory bandwidth for simplicity.

More advanced users or those who wish to learn more about DDR SDRAM technology may want to use the Xilinx 7-series memory interface solutions core generated by the MIG (Memory Interface Generator) Wizard. Depending on the tool used (ISE, EDK or Vivado), the MIG Wizard can generate a native FIFO-style or an AXI4 interface to connect to user logic. This workflow allows the customization of several DDR parameters optimized for the particular application. Table 3.1 below lists the MIG Wizard settings optimized for the Nexys A7.

Table 3.1.1 DDR2 settings for the Nexys A7.

Setting	Value
Memory type	DDR2 SDRAM
Max. clock period	3000ps (667Mbps data rate)

شکل ۲۵-۱۰

Setting	Value
Memory type	DDR2 SDRAM
Max. clock period	3000ps (667Mbps data rate)
Recommended clock period (for easy clock generation)	3077ps (650Mbps data rate)
Memory part	MT47H64M16HR-25E
Data width	16
Data mask	Enabled
Chip Select pin	Enabled
Rtt (nominal) – On-die termination	50ohms
Internal Vref	Enabled
Internal termination impedance	50ohms

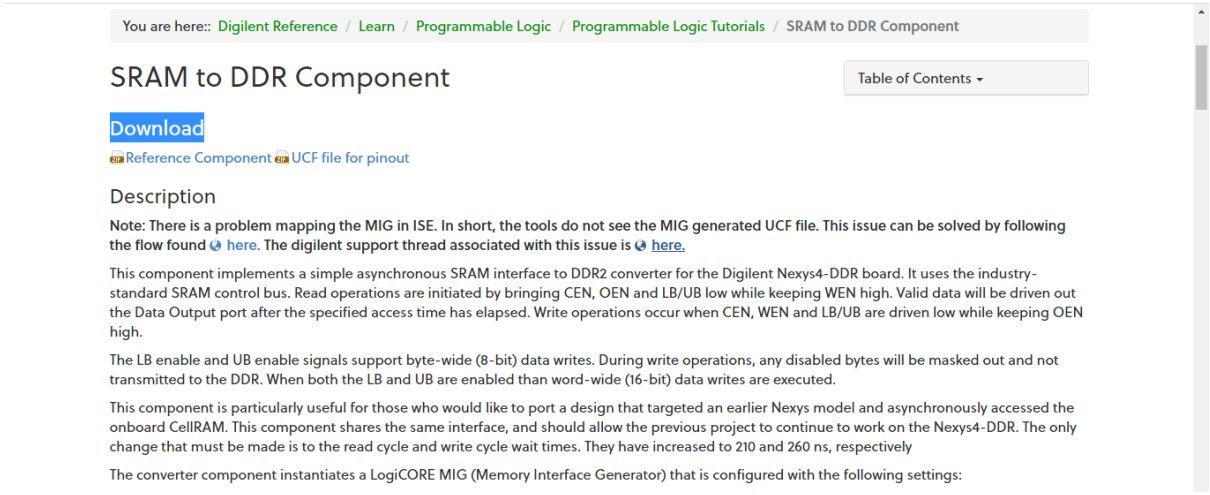
Although the FPGA, memory IC, and the board itself are capable of the maximum data rate of 667Mbps, the limitations in the clock generation primitives restrict the clock frequencies that can be generated from the 100 MHz system clock. Thus, for simplicity, the next highest data rate of 650Mbps is recommended.

The MIG Wizard will require the fixed pin-out of the memory signals to be entered and validated before generating the IP core. For your convenience, an importable UCF file is provided on the Digilent website to speed up the process.

For more details on the Xilinx memory interface solutions, refer to the 7 Series FPGAs Memory Interface Solutions User Guide (ug586)[1].

شکل ۲۶-۱۰

در ادامه یک مثال را برای آشنایی آورده شده است.



You are here:: Digilent Reference / Learn / Programmable Logic / Programmable Logic Tutorials / SRAM to DDR Component

SRAM to DDR Component

Table of Contents

Download

Reference Component UCF file for pinout

Description

Note: There is a problem mapping the MIG in ISE. In short, the tools do not see the MIG generated UCF file. This issue can be solved by following the flow found here. The digilent support thread associated with this issue is here.

This component implements a simple asynchronous SRAM interface to DDR2 converter for the Digilent Nexys4-DDR board. It uses the industry-standard SRAM control bus. Read operations are initiated by bringing CEN, OEN and LB/UB low while keeping WEN high. Valid data will be driven out the Data Output port after the specified access time has elapsed. Write operations occur when CEN, WEN and LB/UB are driven low while keeping OEN high.

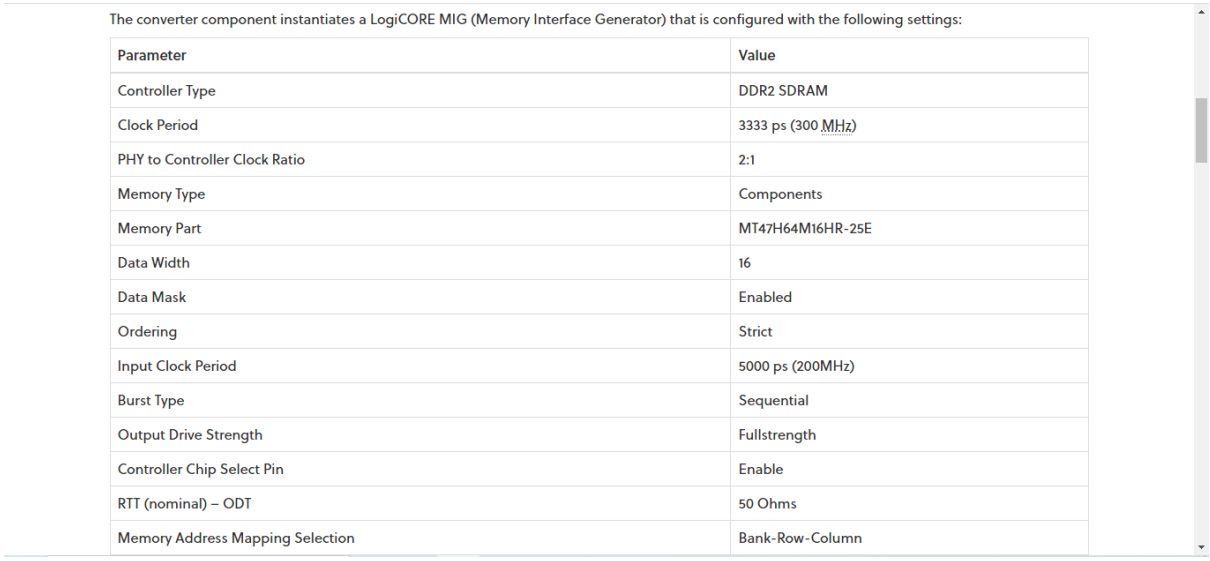
The LB enable and UB enable signals support byte-wide (8-bit) data writes. During write operations, any disabled bytes will be masked out and not transmitted to the DDR. When both the LB and UB are enabled than word-wide (16-bit) data writes are executed.

This component is particularly useful for those who would like to port a design that targeted an earlier Nexys model and asynchronously accessed the onboard CellRAM. This component shares the same interface, and should allow the previous project to continue to work on the Nexys4-DDR. The only change that must be made is to the read cycle and write cycle wait times. They have increased to 210 and 260 ns, respectively

The converter component instantiates a LogiCORE MIG (Memory Interface Generator) that is configured with the following settings:

شکل ۲۷-۱۰: مثال آورده شده در سایت برد

در ادامه، در تعریف حافظه اطلاعات آورده شده در شکل ۲۸-۱۰ و شکل ۲۹-۱۰ را وارد می کنیم:



The converter component instantiates a LogiCORE MIG (Memory Interface Generator) that is configured with the following settings:

Parameter	Value
Controller Type	DDR2 SDRAM
Clock Period	3333 ps (300 MHz)
PHY to Controller Clock Ratio	2:1
Memory Type	Components
Memory Part	MT47H64M16HR-25E
Data Width	16
Data Mask	Enabled
Ordering	Strict
Input Clock Period	5000 ps (200MHz)
Burst Type	Sequential
Output Drive Strength	Fullstrength
Controller Chip Select Pin	Enable
RTT (nominal) – ODT	50 Ohms
Memory Address Mapping Selection	Bank-Row-Column

شکل ۲۸-۱۰

Memory Address Mapping Selection	Bank-Row-Column
System Clock	No Buffer
Reference Clock	Use System Clock
System Reset Polarity	Active Low
Debug Signals for Memory Controller	Off
Internal Vref	Enabled
IO Power Reduction	On
XADC Instantiation	Enabled[1]
Internal Termination Impedance	50 Ohms

Table 1. MIG settings

[1]If the XADC is used elsewhere in the design, this should be disabled. It is disabled in the Ram2Ddr version of this core

This project contains two different components: Ram2Ddr and Ram2DdrXadx. If your design does not use the XADC core anywhere else, you should use the Ram2DdrXadc component. It automatically instantiates the XADC internally to monitor the chip temperature. If your design does use the XADC core, you should use the Ram2Ddr component. You will then have to connect the device_temp line of the Ram2Ddr component to the XADC component, as described on page 122 in version 1.9 of Xilinx's 7 Series FPGAs Memory Interface Solutions Guide (UG586).

Components can either be inserted into a project as a pre-compiled netlist (.ngc), or as sources by copying the VHDL and MIG project files into your project. Both are included with the download.

Port Descriptions

شکل ۲۹-۱۰

در شکل ۳۰-۱۰ بلوک دیاگرام حافظه ی وصل شده به هسته نمایش داده شده است.

Port Descriptions

Figure 1 shows the ram2ddr component block diagram with its ports:

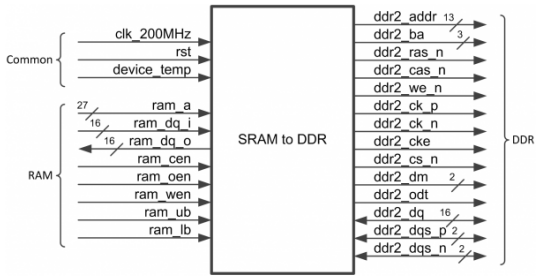


Figure 1. Block Diagram

Port	Direction	Description
clk_200MHz	Input	Single-ended, buffered 200 MHz clock input
rst	Input	Active-high global reset
device_temp	Input	This port is active only in the ram2ddr.ngc file (where the XADC module is not internally instantiated) and shall be tied to GND in the ram2ddrxadc.ngc component. See "Xilinx UG586 7 Series FPGAs Memory Interface Solutions" for more details on driving this port.

شکل ۳۰-۱۰

RAM		
ram_a (26:0)	Input	Input address
ram_dq_i (15:0)	Input	Data input
ram_dq_o (15:0)	Output	Data output
ram_cen	Input	Active-low Chip Enable
ram_oen	Input	Active-low Output Enable
ram_wen	Input	Active-low Write Enable
ram_ub	Input	Active-low Upper Byte select
ram_lb	Input	Active-low Lower Byte select
DDR		
ddr2_addr (12:0)	Output	Memory Address output
ddr2_ba (2:0)	Output	Bank Address
ddr2_ras_n	Output	Active-low Row Address Strobe
ddr2_cas_n	Output	Active-low Column Address Strobe
ddr2_we_n	Output	Active-low Write Enable
ddr2_ck_p,	Output	Differential Memory Clock output

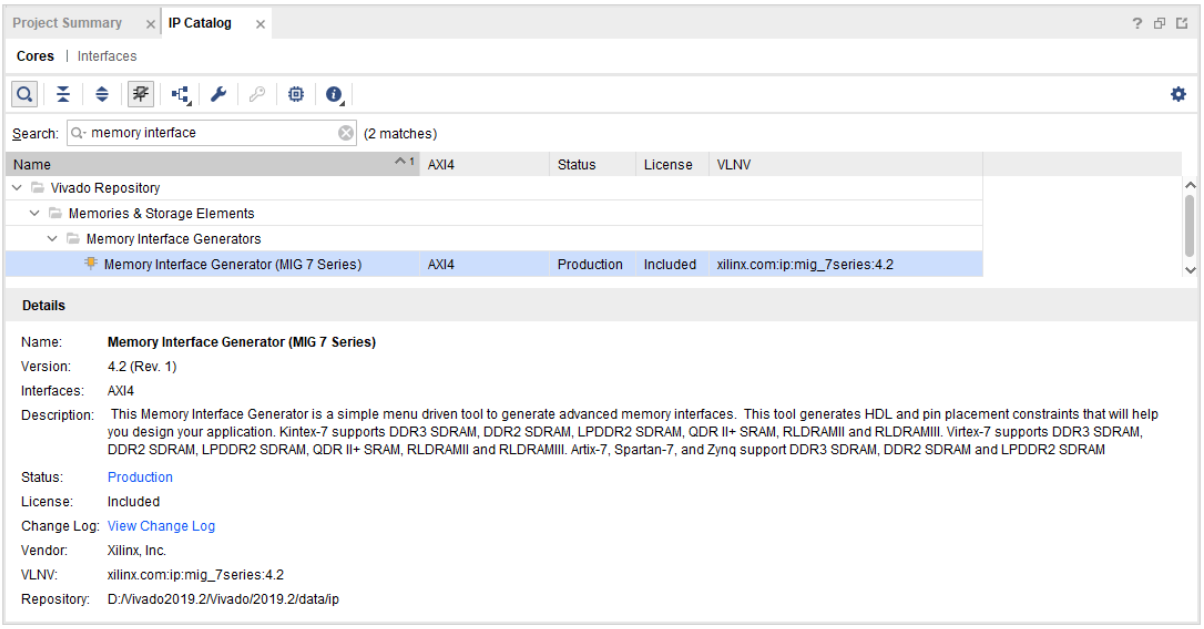
شکل ۳۱-۱۰

DDR		
ddr2_addr (12:0)	Output	Memory Address output
ddr2_ba (2:0)	Output	Bank Address
ddr2_ras_n	Output	Active-low Row Address Strobe
ddr2_cas_n	Output	Active-low Column Address Strobe
ddr2_we_n	Output	Active-low Write Enable
ddr2_ck_p, ddr2_ck_n	Output	Differential Memory Clock output
ddr2_cke	Output	Active-high Memory Clock Enable
ddr2_cs_n	Output	Active-low Chip Select
ddr2_dm (1:0)	Output	Output Data Mask
ddr2_odt	Output	On-Die Termination
ddr2_dq (15:0)	Bidirectional	Data input/output bus
ddr2_dqs_p (1:0), ddr2_dqs_n (1:0)	Bidirectional	

Table 2. Port Descriptions

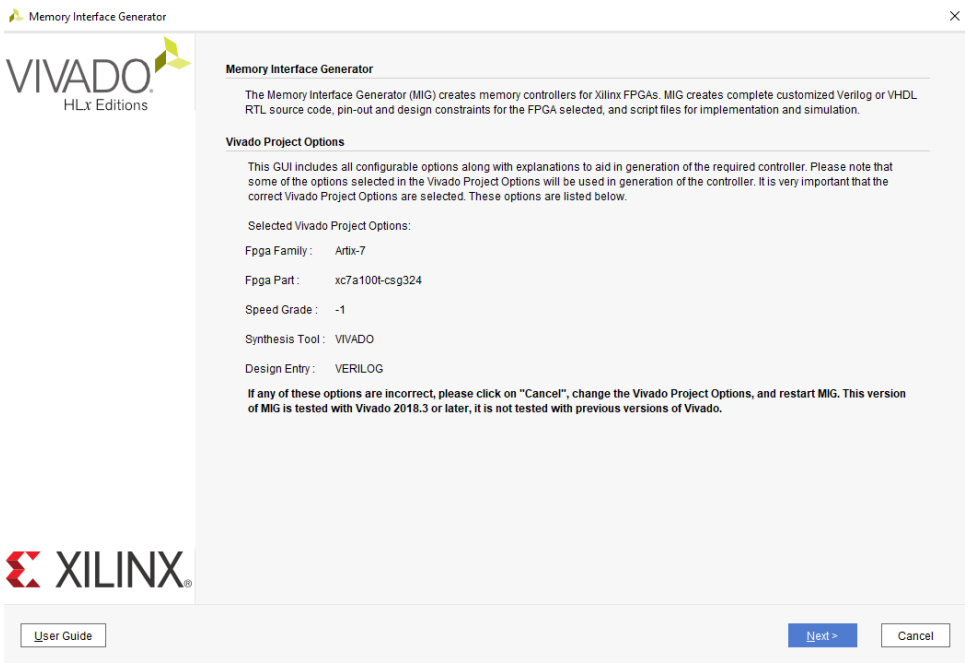
شکل ۳۲-۱۰

در vivado پروژه را باز می کنیم.

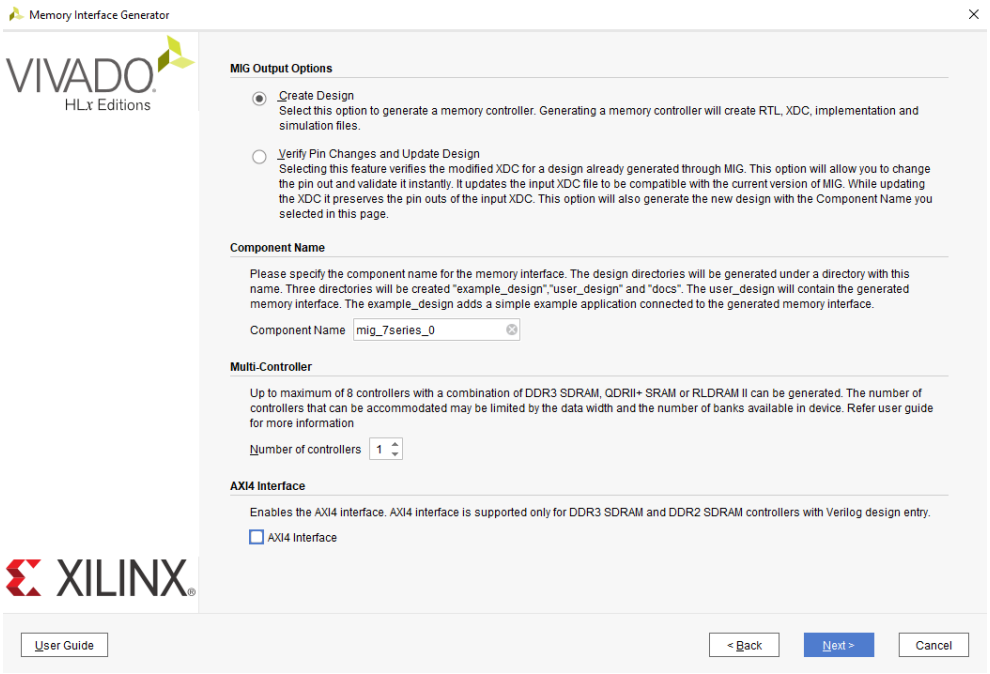


شکل ۳۳-۱۰

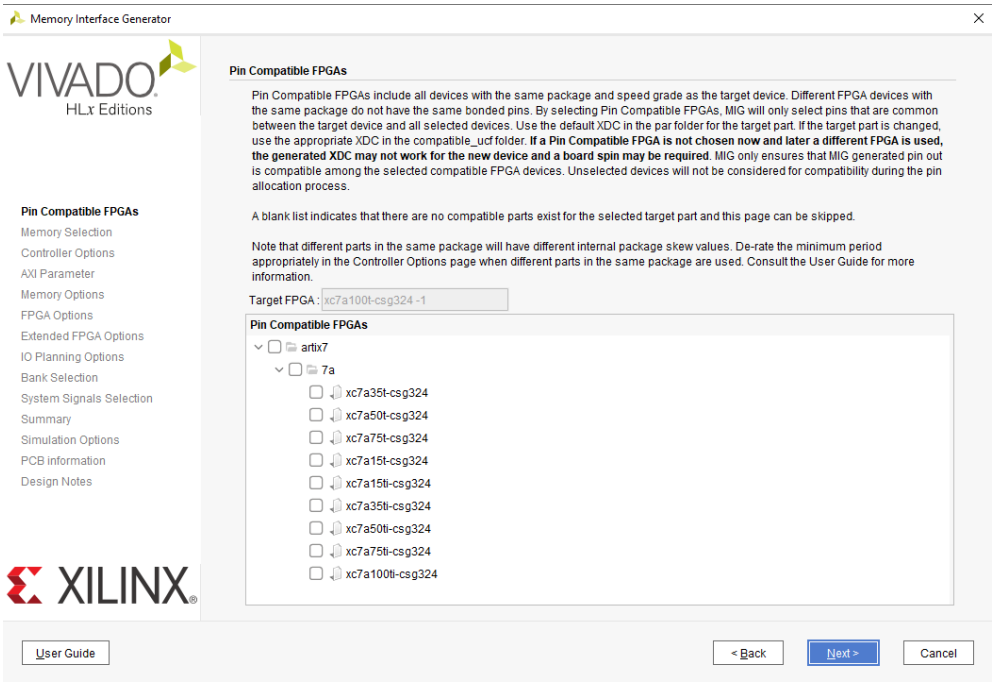
در ادامه ۳ پنجره ی بعدی اطلاعات مفیدی آورده شده است.next را می زنیم.



شکل ۳۴-۱۰

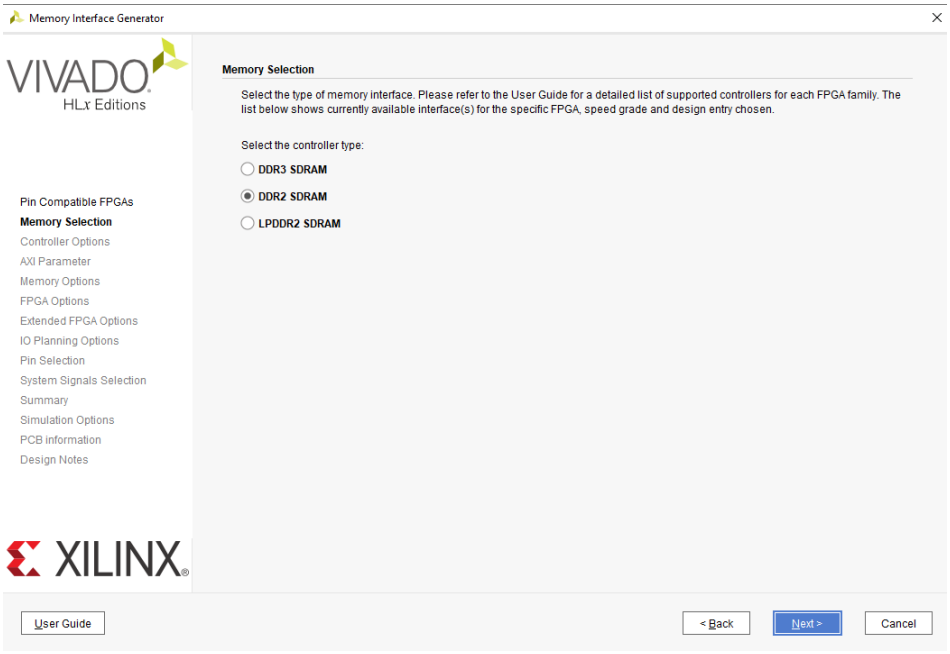


شکل ۳۵-۱۰



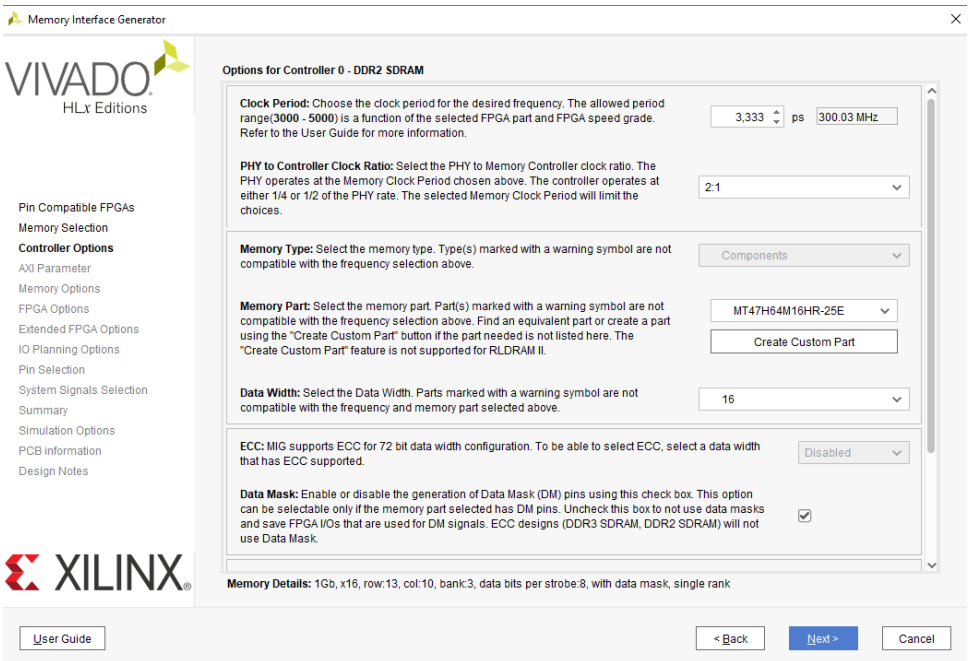
شکل ۳۶-۱۰

درپنجره ی بعدی همانند شکل ۳۷-۱۰ نوع حافظه خود؛ که برای برد ما DDR۲ می باشد را انتخاب می کنیم.

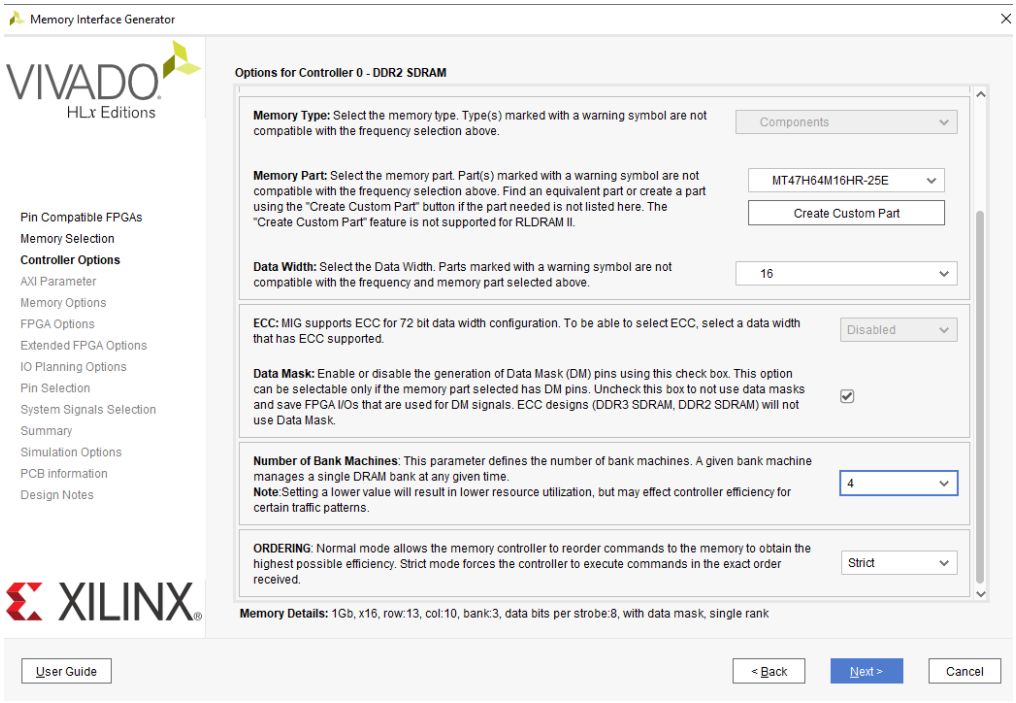


شکل ۳۷-۱۰

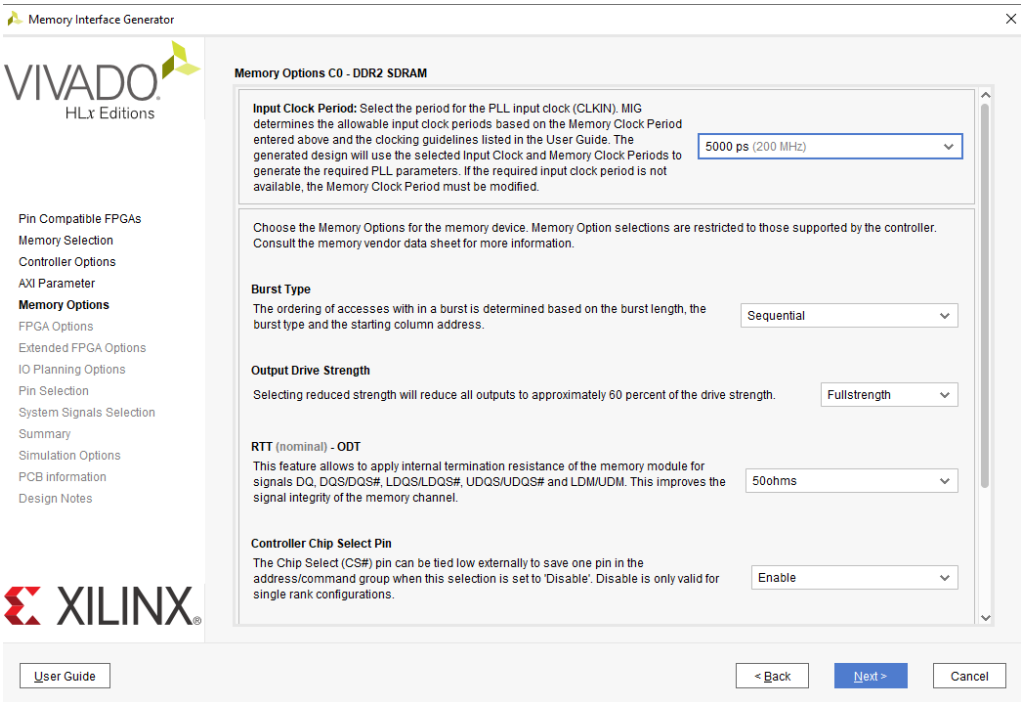
در پنجره های بعدی اطلاعاتی که در اول این بخش از آزمایش آورده شده است را وارد می کنیم و next را می زنیم.



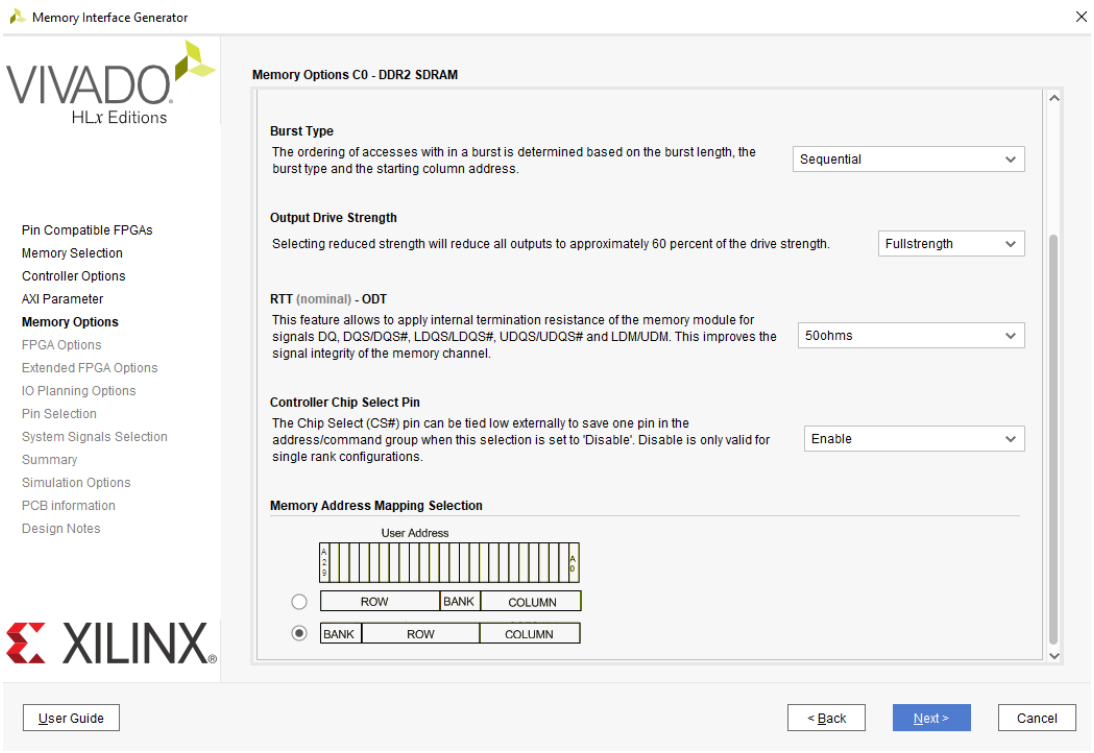
شکل ۳۸-۱۰



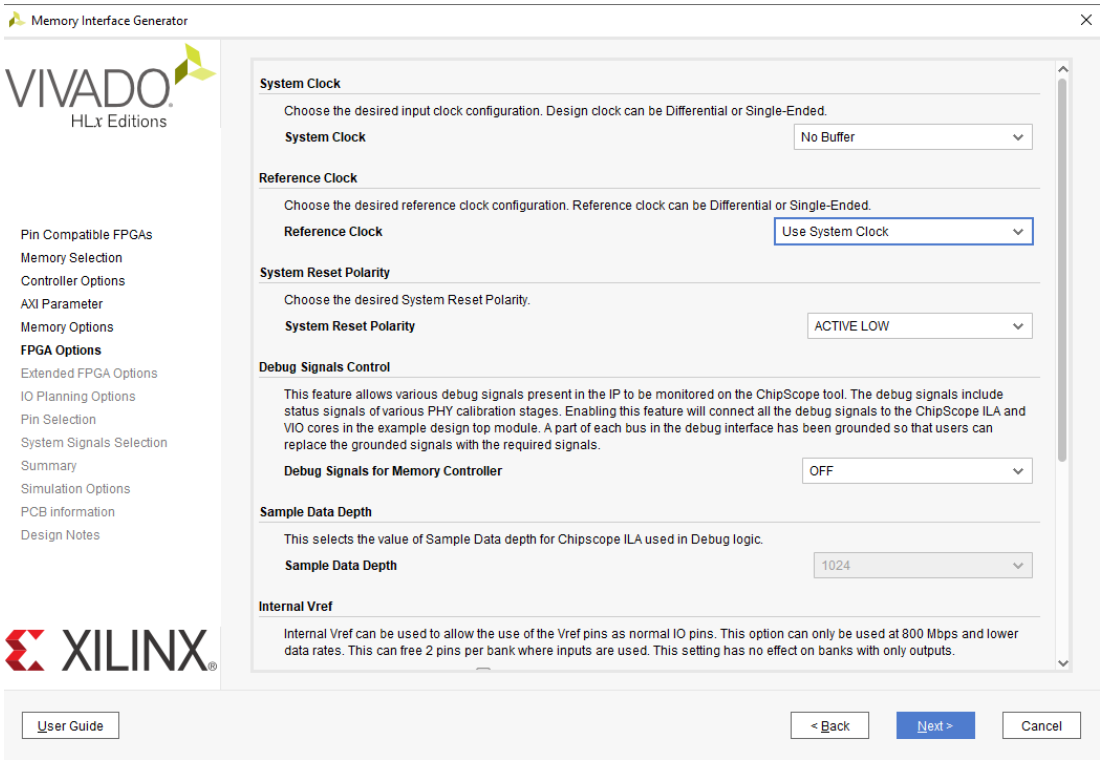
شکل ۳۹–۱۰



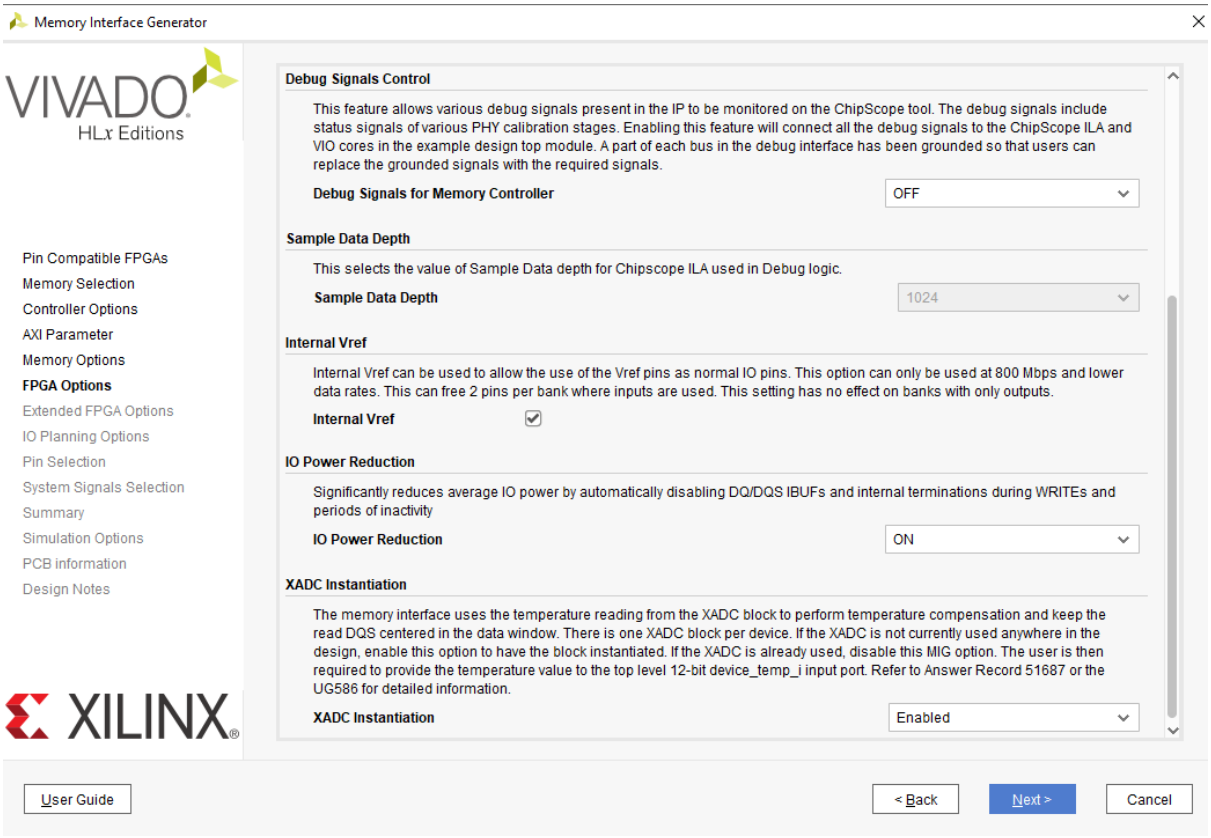
شکل ۴۰–۱۰



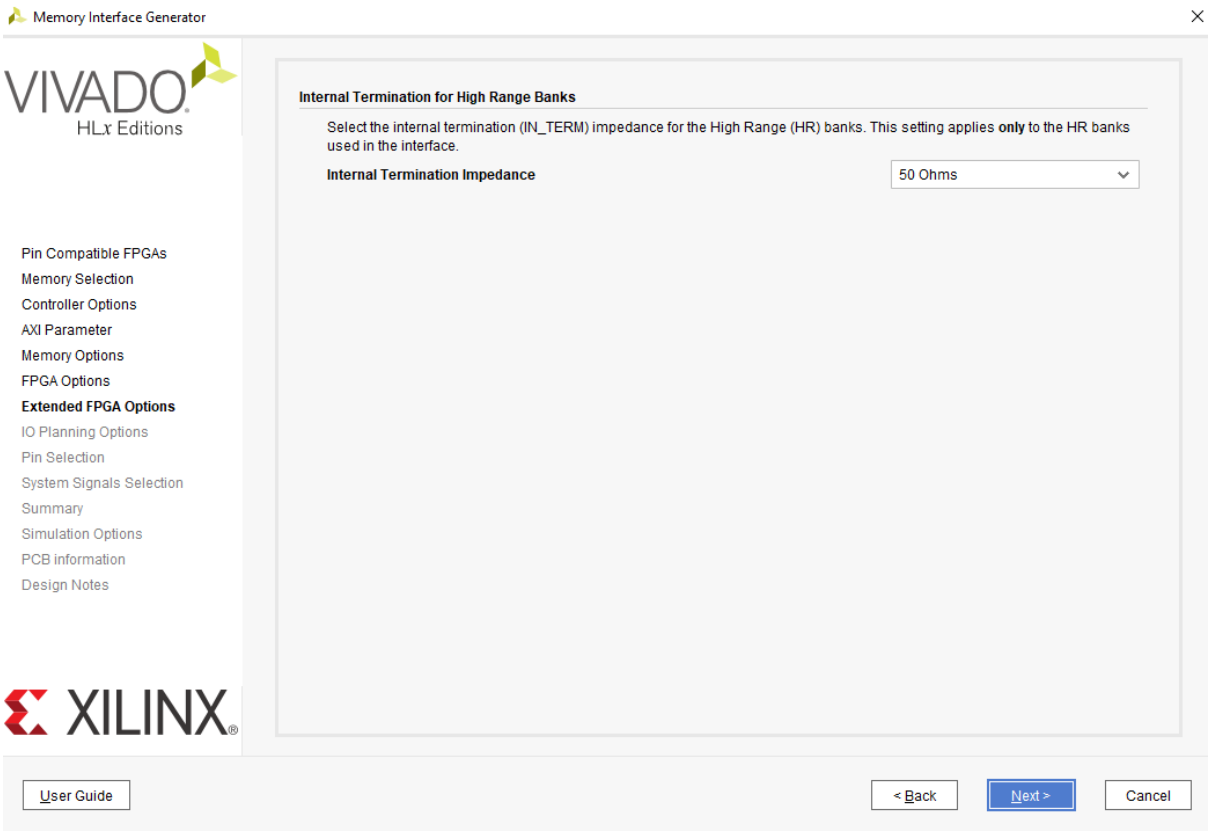
شکل ۱۰–۴۱



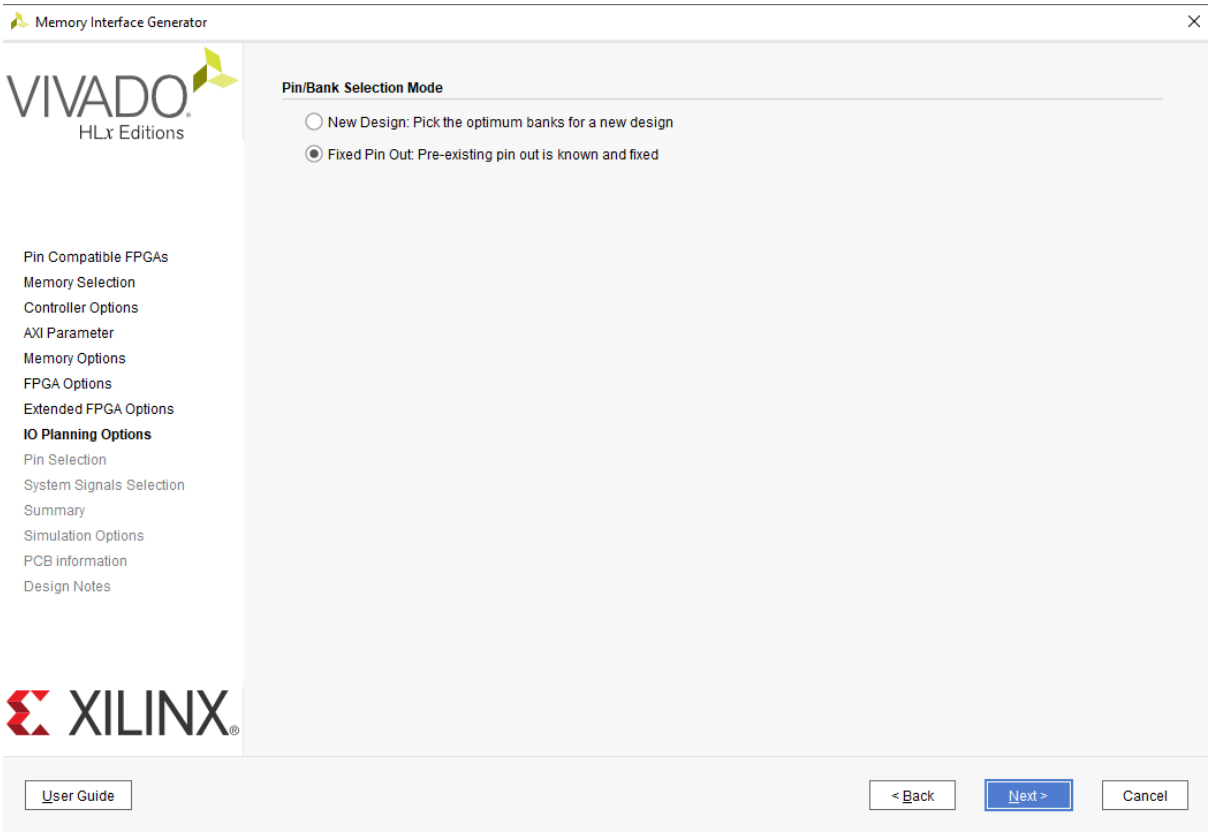
شکل ۱۰–۴۲



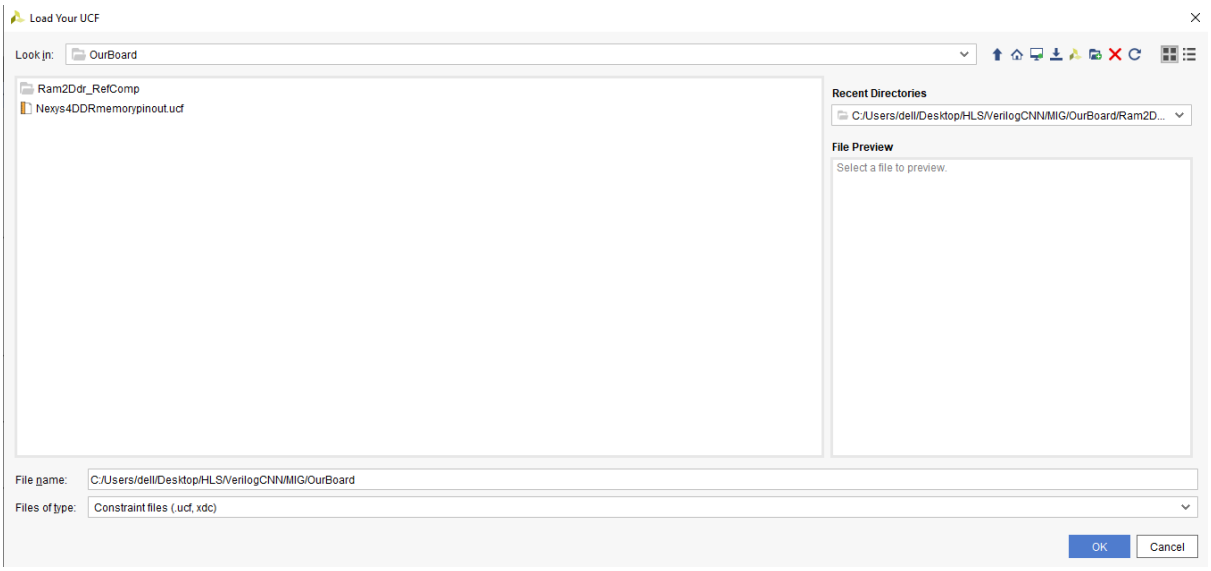
شکل ۱۰-۴۳



شکل ۱۰-۴۴

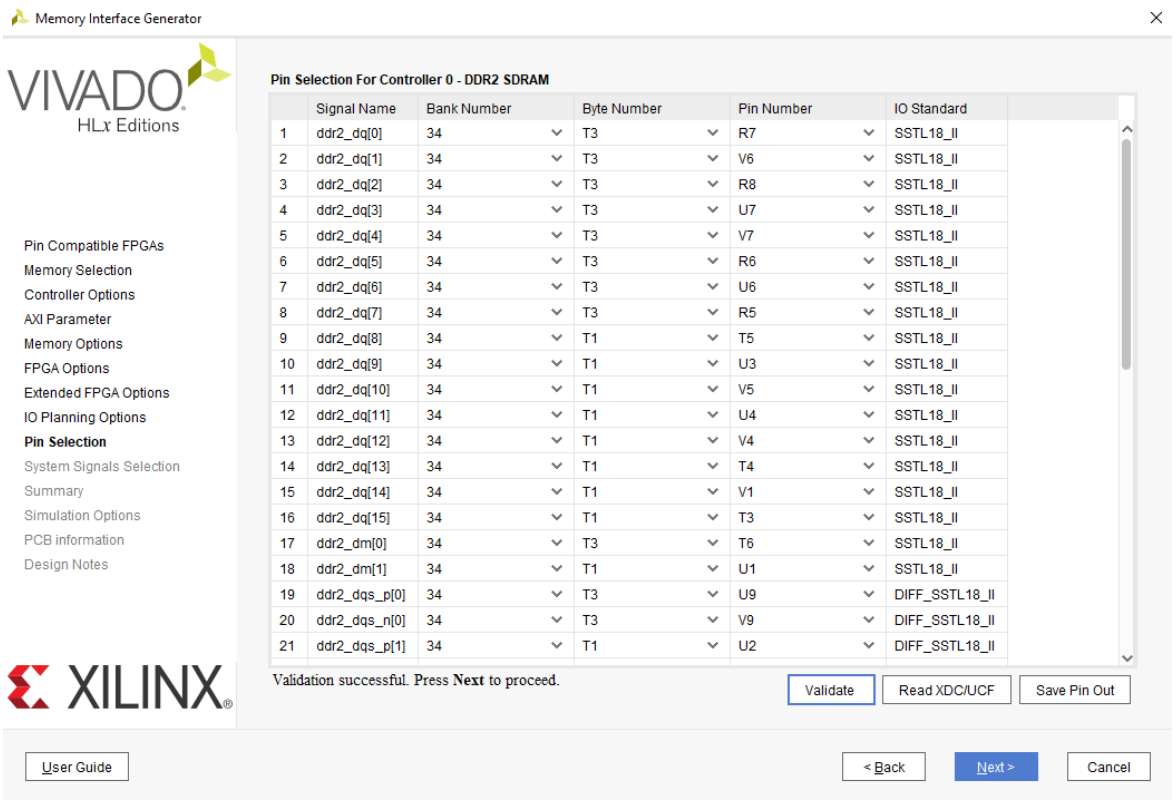


شکل ٤٥-١٠



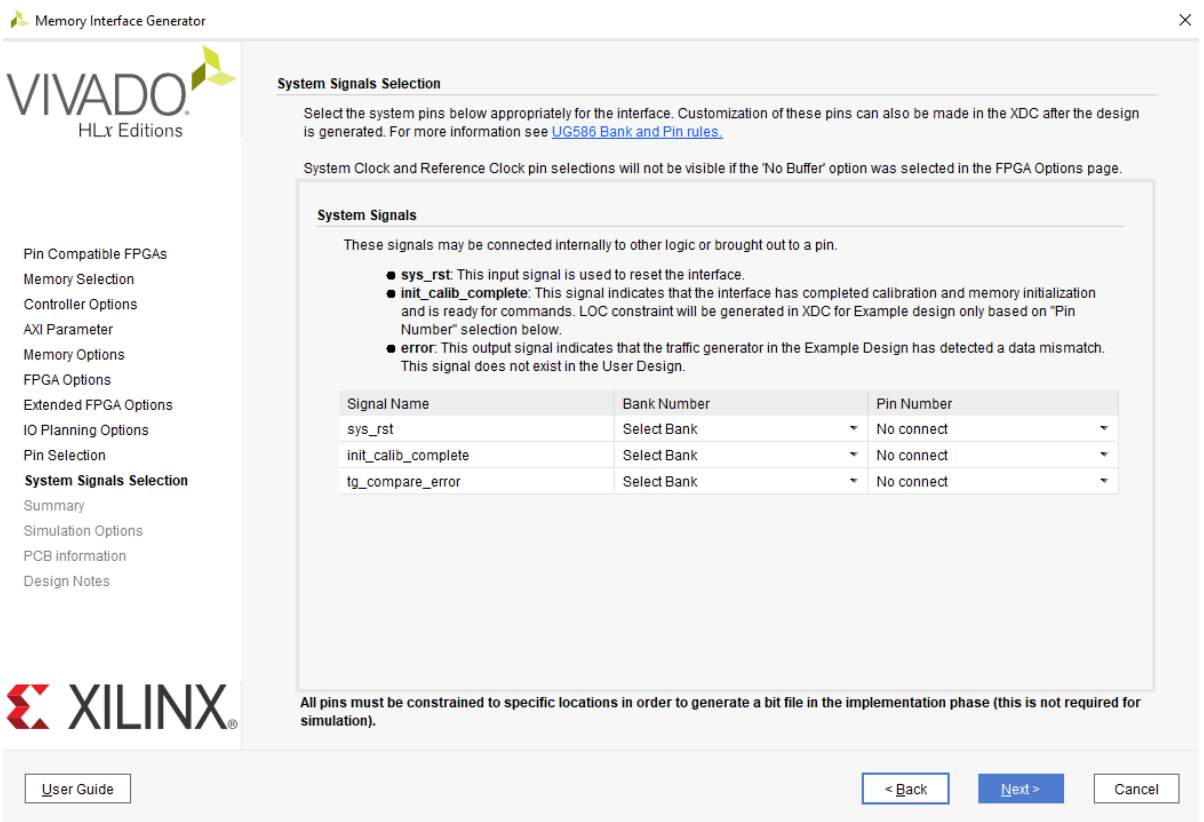
شکل ٤٦-١٠

در ادامه باید پایه های حافظه را که به هسته وصل شده است را معرفی کنیم که از فایلی که در همان سایت برد قرار داده شده است به صورت زیر معرفی می کنیم.

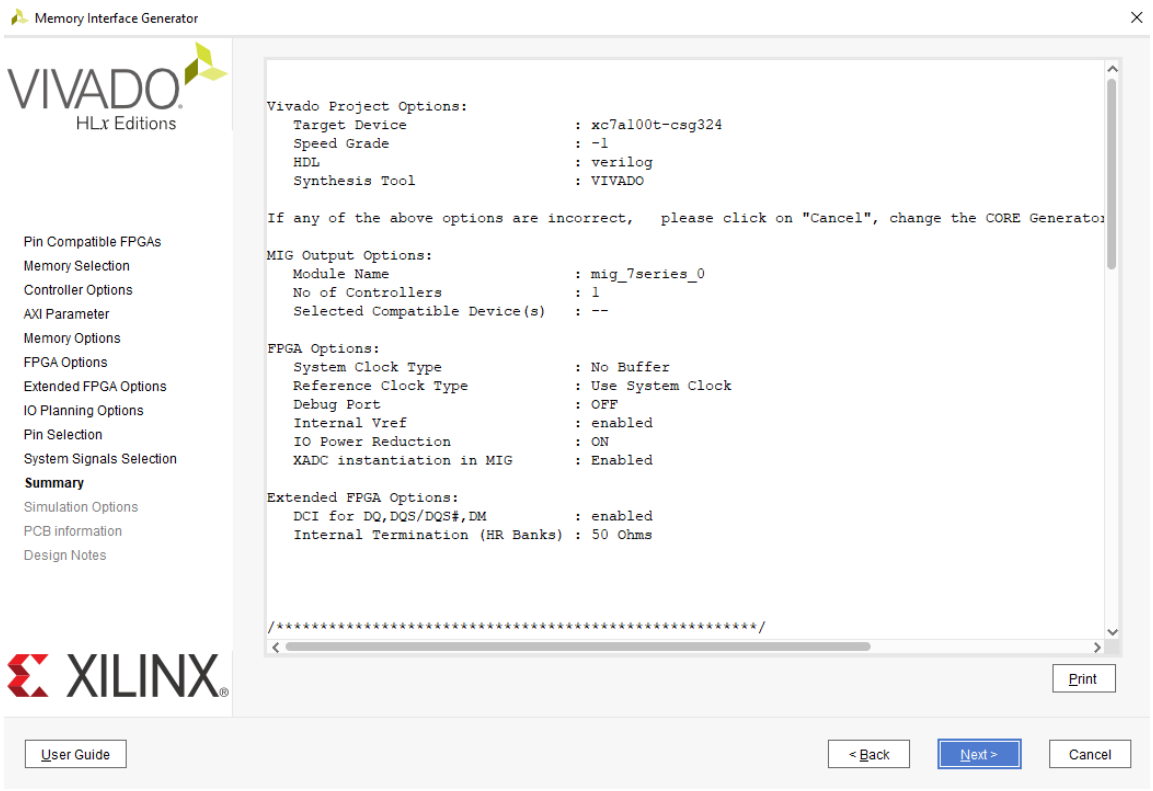


	Signal Name	Bank Number	Byte Number	Pin Number	IO Standard
1	ddr2_dq[0]	34	T3	R7	SSTL18_II
2	ddr2_dq[1]	34	T3	V6	SSTL18_II
3	ddr2_dq[2]	34	T3	R8	SSTL18_II
4	ddr2_dq[3]	34	T3	U7	SSTL18_II
5	ddr2_dq[4]	34	T3	V7	SSTL18_II
6	ddr2_dq[5]	34	T3	R6	SSTL18_II
7	ddr2_dq[6]	34	T3	U6	SSTL18_II
8	ddr2_dq[7]	34	T3	R5	SSTL18_II
9	ddr2_dq[8]	34	T1	T5	SSTL18_II
10	ddr2_dq[9]	34	T1	U3	SSTL18_II
11	ddr2_dq[10]	34	T1	V5	SSTL18_II
12	ddr2_dq[11]	34	T1	U4	SSTL18_II
13	ddr2_dq[12]	34	T1	V4	SSTL18_II
14	ddr2_dq[13]	34	T1	T4	SSTL18_II
15	ddr2_dq[14]	34	T1	V1	SSTL18_II
16	ddr2_dq[15]	34	T1	T3	SSTL18_II
17	ddr2_dm[0]	34	T3	T6	SSTL18_II
18	ddr2_dm[1]	34	T1	U1	SSTL18_II
19	ddr2_dqs_p[0]	34	T3	U9	DIFF_SSTL18_II
20	ddr2_dqs_n[0]	34	T3	V9	DIFF_SSTL18_II
21	ddr2_dqs_p[1]	34	T1	U2	DIFF_SSTL18_II

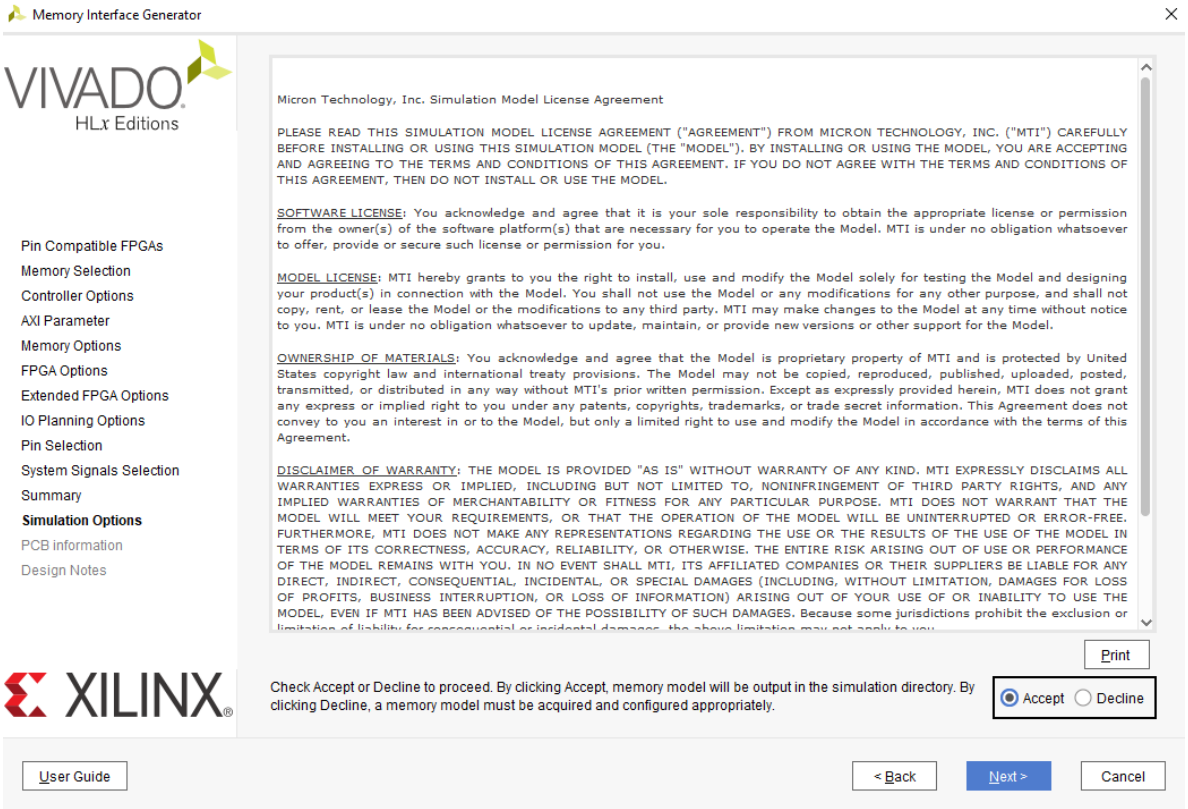
شکل ۱۰-۴۷



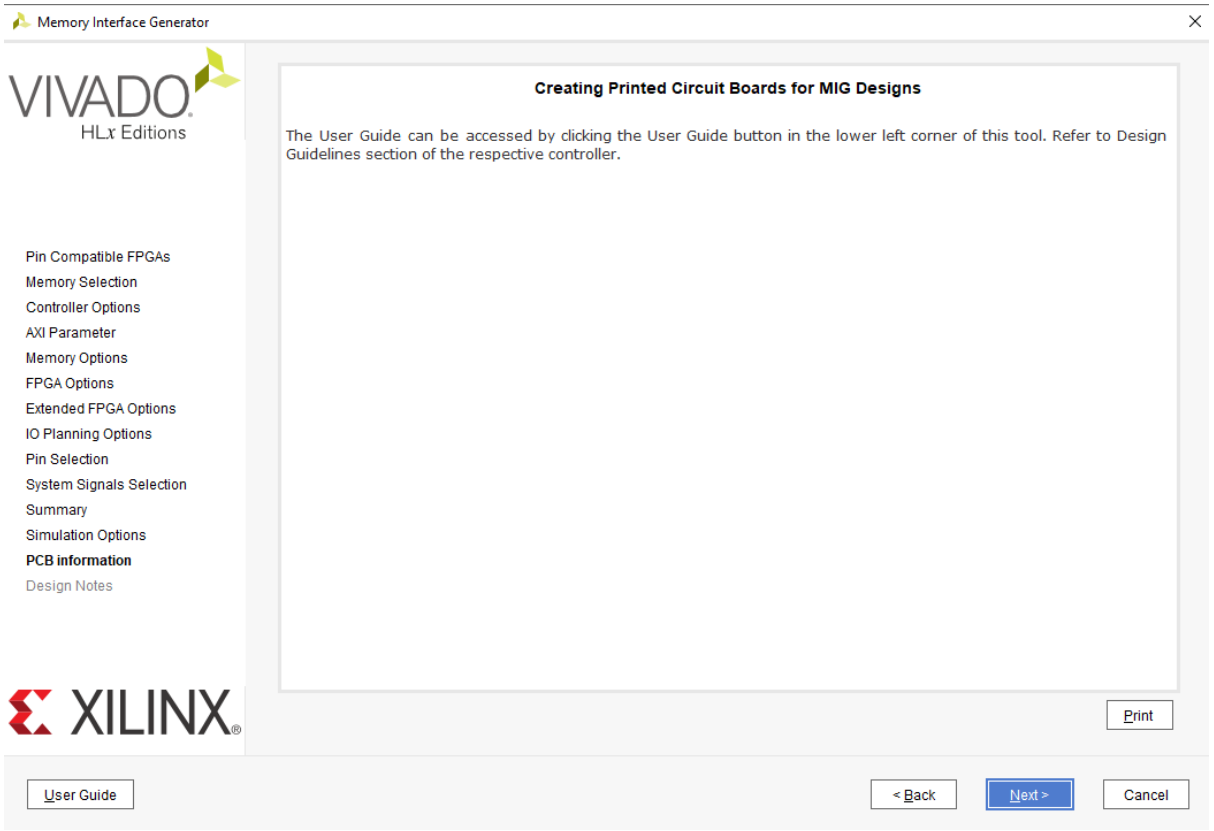
شکل ۱۰-۴۸



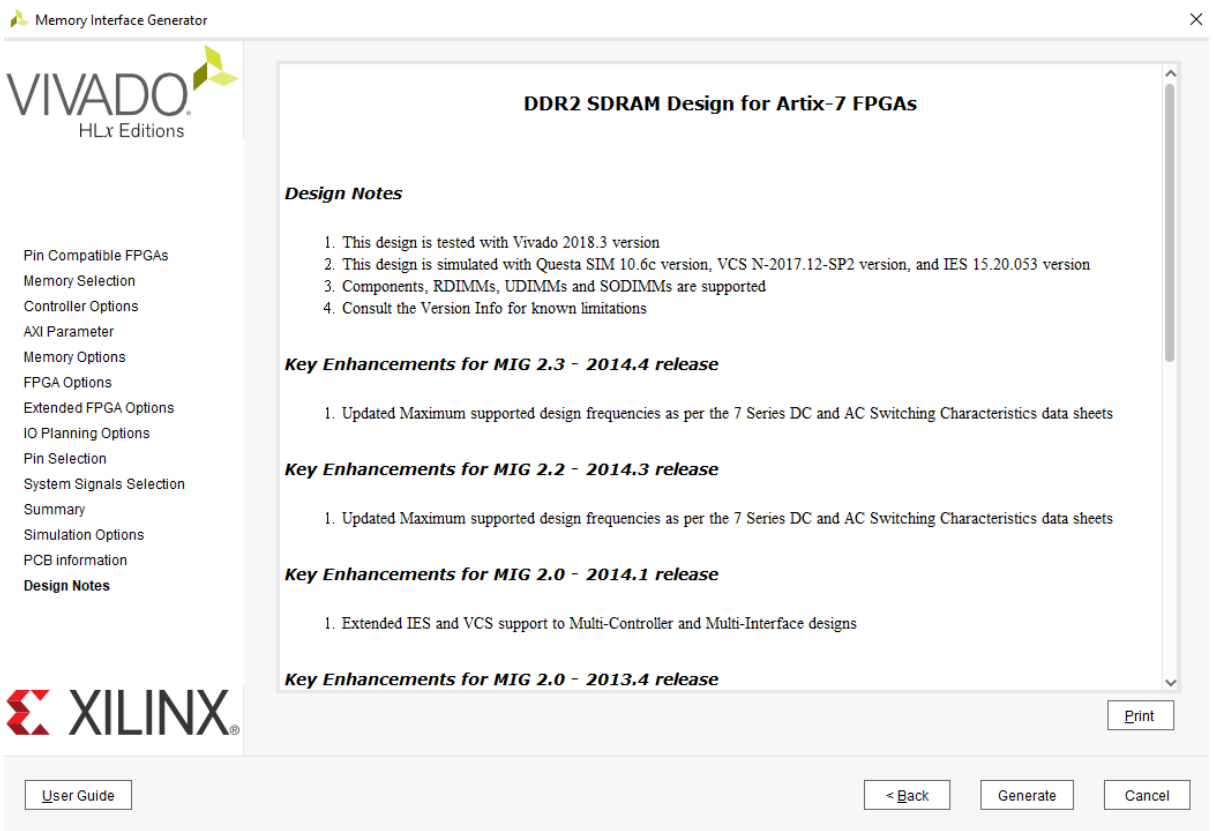
شکل ۴۹–۱۰



شکل ۵۰–۱۰



شکل ۱۰–۵۱

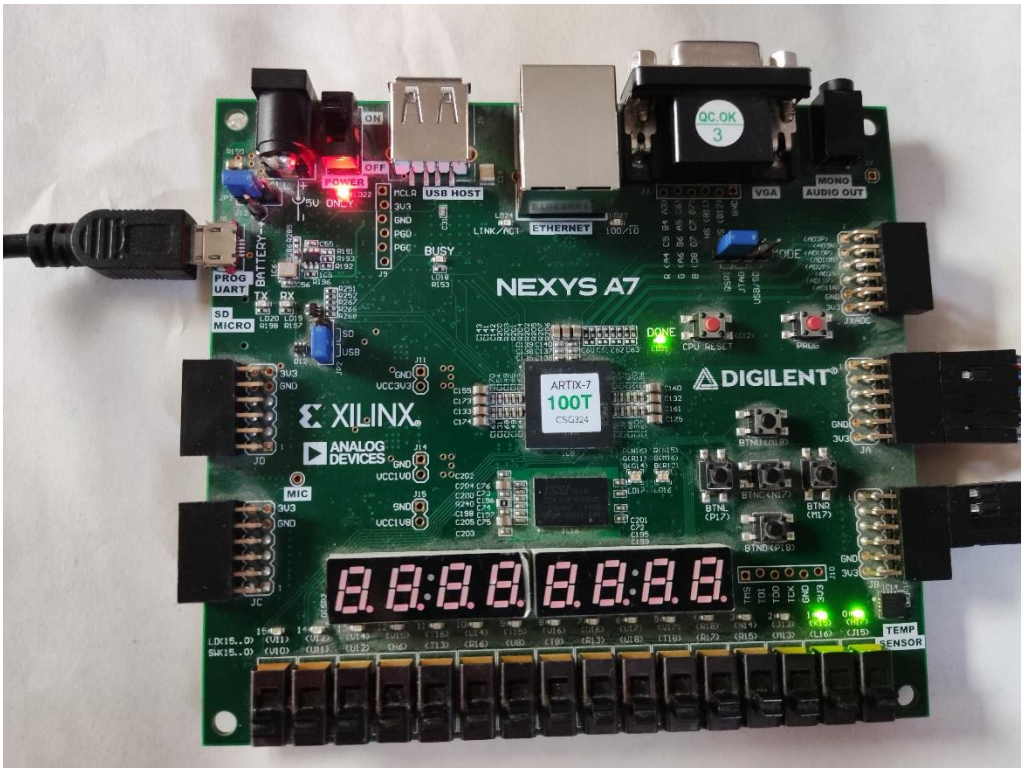


شکل ۱۰–۵۲

نهایتا کد های زیررا برای تست می زنیم و روی برد آپلود می کنیم.

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk_in }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk_in}];

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led_calib }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led_pass }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led_fail }]; #IO_L17N_T2_A25_15 Sch=led[2]
```

شکل ۵۳-۱۰: کد های مربوط به DDR



شکل ٥٤-۱۰: نتایج نهایی