بسمه تعالی

گزارش کارآموزی

نام کارآموز : امین اله زکی زاده نام استاد کارآموزی : دکتر محمد کاظمی

موضوع : راه اندازی تراشه ی زینک

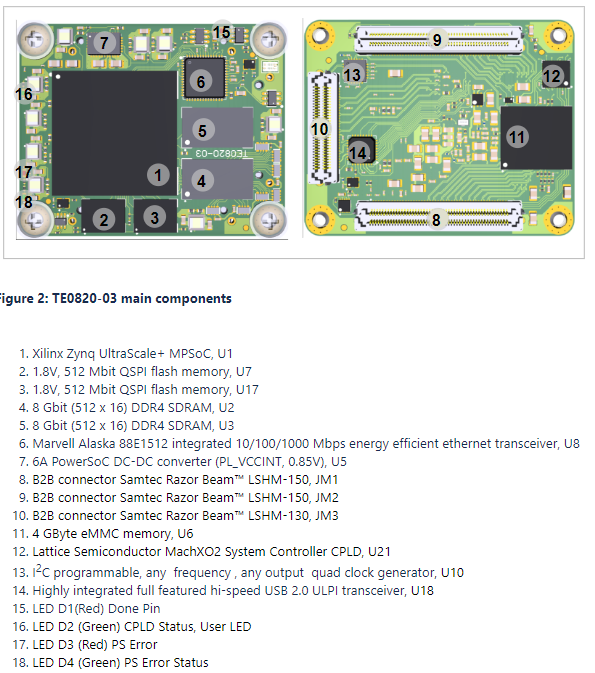
قسمت اول : تراشه ی مورد بررسی و برد ماژول

در داکیومنت های شرکت trenz به برد اصلی که حاوی قطعات زیر است ماژول گفته می شود که بر روی برد حامل یا career قرار می گیرد.

تراشه ی zynq ultrascale mpsoc با مدل XCZU3EG و با speed grade به شماره ی -1 مورد بررسی است این تراشه ی ساخت شرکت xilinx برروی برد توسعه te820 ساخته ی شرکت trenz قرار گرفته . اجزای دیگر این برد:

1. 2 گیگابایت رم از نوع ddr4
2. 4 گیگابایت emmc
3. دو عدد qspi flash مجموعا به سایز 128 مگابایت و به صورت موازی متصل به ps
4. یک چیپ pll با چهار خروجی متصل به pl به شماره ی Si5338
5. یک چیپ اترنت به شماره ی Marvell Alaska 88E1512
6. یک Hi-speed USB2 به شماره ی Microchip USB3320C
7. یک cpld کنترل کننده ی نحوه ی پروگرام شدن چیپ به هنگام راه اندازی
8. ic های مدیریت توان
9. سه عدد کانکتور razor beam

به شکل زیر :



شماره ی 15 یک led قرمز است که به محض اتمام پروگرام شدن قسمت fpga ،که به آن pl که مخفف programable logic است هم می گویند ، خاموش می شود و هنگام startup و یا هنگامی که روی منبع بوت هیچ برنامه ای نیست روشن است .

شماره ی 17 مربوط به ps\_error است که طبق trm مربوط به zynq ultrascale+ به آدرس <https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf>

به علل زیر ممکن است اتفاق بیفتد :

Asserted for accidental loss of power, a hardware error, or an exception in the PMU.

در زیر موضوع PMU Error Handling and Propagation Logic این موضوع نوشته شده.

شماره 17 یک led سبز چشمک زن است که به یکی از پایه های cpld شماره ی 12 متصل است و نحوه ی بوت را تعداد چشمک ها مشخص می کند .

در صورتی که این led روشن باشد بوت صورت نمی گیرد .

از مشکلات این پروژه خاموش نشدن این led بود که نحوه ی حل در این فایل توضیح داده شده .

برای راه اندازی si5338 باید کد i2c داشته باشیم . طبق اسناد برد روی سایت trenz قبلا محل OTP این چیپ پروگرام نشده و می توان هر بار فرکانس جدیدی تولید کرد .

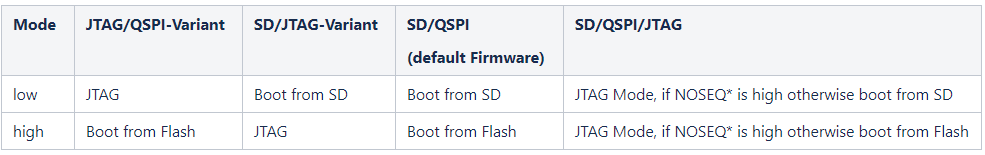
برای ایجاد کد c برای اضافه کردن به پروژه و کنترل از طریق i2c متعلق به PS یک ژنراتور کد وجود دارد که از طریق آدرس زیر می توان به آن دست پیدا کرد .

<https://wiki.trenz-electronic.de/display/PD/Si5338>

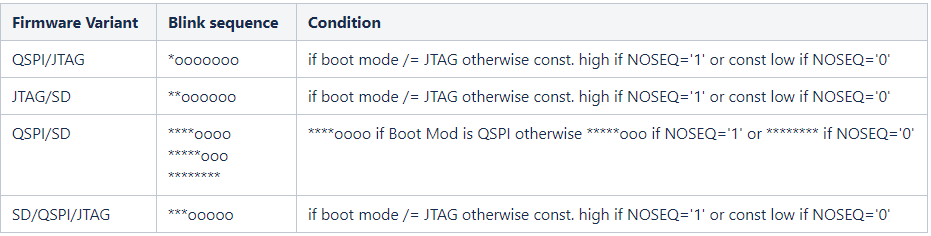
اگر به آدرس <https://wiki.trenz-electronic.de/display/PD/TE0820+CPLD>

بروید راهنمای برد ماژول را خواهید یافت.

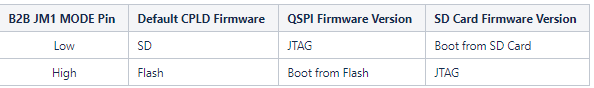
چهار نوع firmware می توان روی این cpld می توان پروگرام کرد که طبق آدرس قبلی جدول زیر را می یابید که نوشته شده sd/qspi یک default firmware است که یعنی کارخانه در هنگام تحویل این firmware را ریخته است.



پس بروید جدول زیر را خواهید یافت :



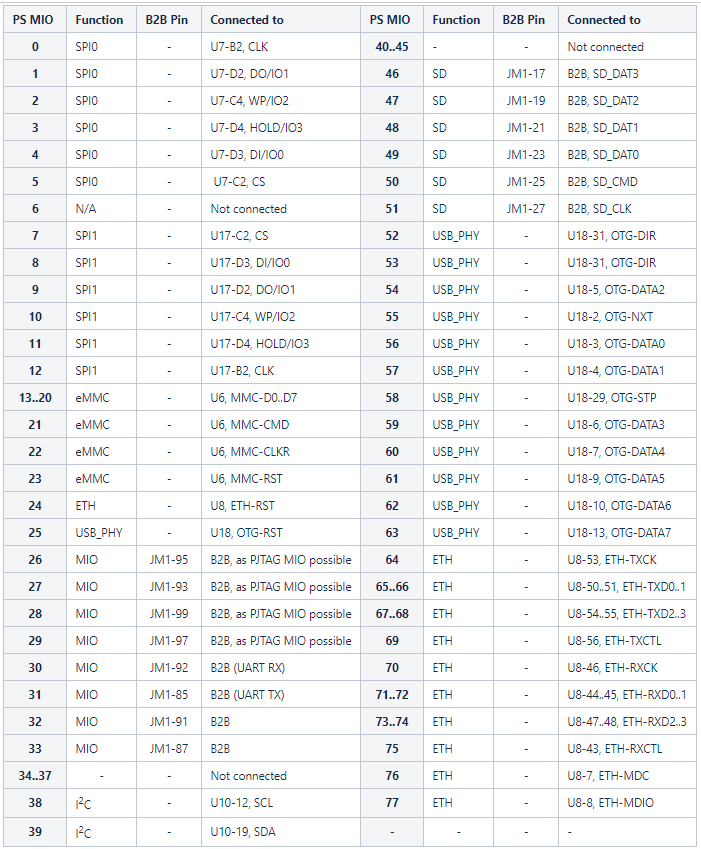
پس طبق ردیف سوم باید چشمک ها را کنترل کنیم . اگر از روی qspi در صورت روشن و خاوش شن برد بوت کند از 8 سکانس ، 4 سکانس را چشمک میزند (هر سکانس با فاصله ی 1 ثانیه) و در صورت بوت کردن از روی sd کارت 5 سکانس را چشمک می زند.



با توجه به بالا حالت بوت با تغییر پایه ی boot متصل به cpld روی برد ماژول انجام می شود. بعدا این پایه به cpld برد حامل متصل می شود و توسط یک دیپ سوییچ روی برد حامل توسط cpld کنترل خواهد شد

تغییر حالت بوت به وسیله ی cpld بر روی برد حامل carrier انجام می شود که چگونگی آن بعدا توضیح داده خواهد شد .

تصویر زیر پایه های mio نشان داده شده که هر کدام برای منظور خاصی استفاده شده



Mio ها پایه های بلا فصل متصل به ps تراشه ی zynq هستند

پایه های spi0 و spi1 به فلش های qspi متصلند و علت گرفتن فضای زیا آ« است که اتصال به صورت موازی انجام گرفته .

در صورت غیر موازی تمام پایه ها بین دو فلش یکسان بود و تنها پایه های فعال متمایز (CS) داشتند ولی اکنون تمام پای ها متمایزند.

تعدادی به emmc روی برد ماژول متصل هستند که با emmc مشخص شده اند .

عد ای هم به صورت b2b یعنی board to board به اسلات sd card متصلند. یعنی از برد ماژول توسط کانکتور های razor beam به برد حامل رفته.

تعدادی به usb phy متصلند .

تعدادی به چیپ اترنت متصلند .

تعدادی اصلا به جایی متصل نیستند .

تعدادی به صورت b2b برای ارتباط jtag به برد حامل رفته اند .

تعدادی برای ارتباط سریال به یک چیپ ft2223 روی برد حامل به صورت b2b متصلند از usart 1 برای ارتباط سریال با این چیپ استفاده می شود . همانطور که مستحضرید این چیپ مبدل انواع ارتباطات به usb می باشد . مانند jtag به usb و یا uart به usb و یا ...

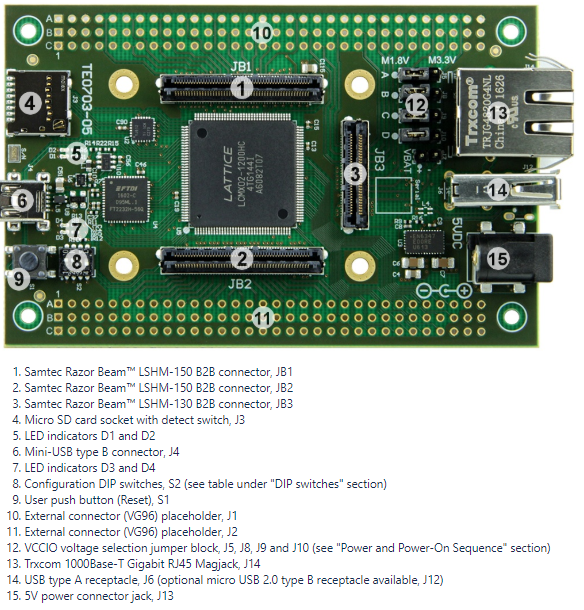
تراشه ی ft2223 دارای دو کانال usb می باشد و همین این چیپ را قابل ساخته تا همزمان هم ارتباط jtag و هم ارتباط uart را به pc بفرستد.

دو mio هم برای اتصال به سوراخ ها کناری برد حامل استفاده شده اند که نحوه ی اتصال آن ها به برد حامل و مکان آن ها توضیح داده خواهد شد.

دو عدد از آن ها هم برای ارتباط i2c برای پروگرام کردن pll و دسترسی به آدرس mac از روی یک eeprom استفاده می شود.

قسمت دوم : برد حامل

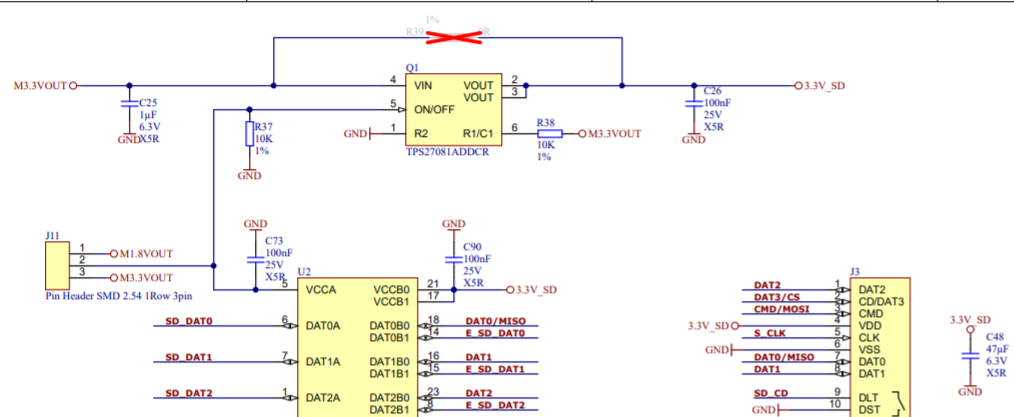
بردی که به واسطه ی کانکتور های مادگی razor beam به برد ماژول وصل می شود به شماره ها ی 1 و 2 و 3



شماره 6 درگاه usb برای ارسال اطلاعات سریال و ارتباط jtag است . شماره ی 4 محل sd card است .

برد حامل ما یک تفاوت کوچک داشت . یکی جامپر تغذیه ی sd card که با توجه به ولتاژ تغذیه ی بانک پایه های mio در برد ماژول می توانست به 1.8 ولت یا 3.3 ولت وصل شود .

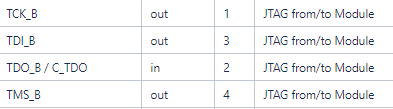
در مورد ما بایست به 1.8 وصل می شد .



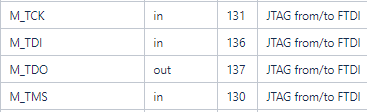
Cpld روی این برد، مسئولیت اتصال بعضی پایه های تراشه zunq به صورت غیر مستقیم به تراشه های روی برد حامل و یا به سوراخ هایی که به شماره ی 10 و 11 را دارد .

به طور مثال تصویر زیر از آدرس https://wiki.trenz-electronic.de/display/PD/TE0703+CPLD+-+CC703S

گرفته شده :



در بالا نام نت های jtag که از ماژول به cpld متصل شده را می بینید.



در بالا نیز پایه ها ی cpld به ftdi 2223 هستند . یعنی cpld به صورت واسط قرارا گرفته احتمالا برای راحت تر شدن ترسیم pcb .

چند نت واسط مهم دیگر در ادامه می آید .

  
در بالا دو led را که به cpld وصل است نشان می دهد .



در بالا نیز گفته شده توسط jb2-99 و jb2-90 کنترل می شوند که اگر در شماتیک حامل و ماژول جستوجو کنیم led3 را متصل به AH6 و led4 را متصل به AD6 می یابیم که پایه های قسمت pl هستند.

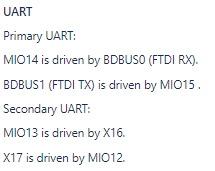


در بالا نیز دو led های 1 و 2 وضعیت uart را نشان می دهند.



در بالا هم می گوید چراغ داخل کانکتور اترنت نشانگر نحوه ی بوت است .

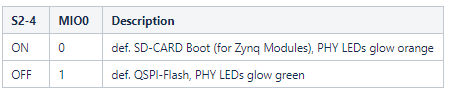
در صورت بوت از qspi رنگ سبز دارد و در صورت بوت از sd رنگ نارنجی و در حالات دیگر کار های دیگر انجام می دهد.



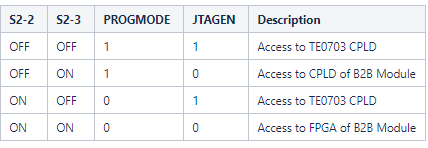
با توجه به بالا و تطبق بین دو شماتیک ftdi rx , tx به پایه های mio30 و mio31 متصلند که هر دو پایه های uart1 مربوط به arm هستند.

همچنین می توان یافت که از سوراخ های کناری j2A30 و J2A31 که همان X16 و X17 هستند به mio29 و mio28 به صورت غیر مستقیم ( cpld واسط) متصلند . میتوان از آن ها برای usart0 استفاده کرد یا یک mio معمولی

نحوه ی بوت به صورت زیر خواهد بود :



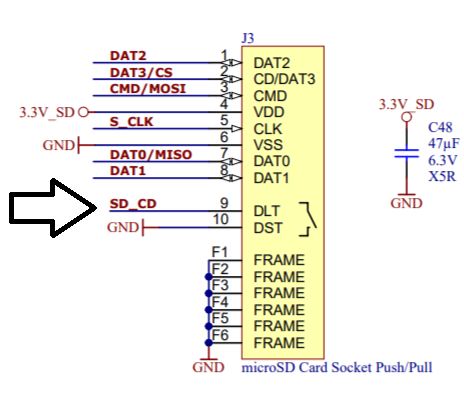
دیپ سوییچ شماره چهار نحوه ی بوت را نشان می دهد .



دیپ سوییچ 2 و 3 دسترسی jtag را معین می کند . در حالت کار با zynq باید هر دو روی on باشند .

همچنین جمله ی زیر را می یابیم :

MIO9 is SD\_CD and CM1(S2-1).Is S2-1 is ON, MIO9 is GND else status depends on SD\_CD .



Mio9 که می شود همان mio32 به وقفه ی sd\_cd که cd مخفف Card Detect است متصل است .

در صورتی که s2-1 on باشد به زمین متصل است و وقفه غیرفعال اما اگر on نباشد ، وقفه ی اتصال sd کارت به zynq می رود . به عکس بالا که دقت کنید در صورت ورود sd سوییچ بسته می شود و sd\_cd به زمین وصل می شود . با توجه به s2-1 ، cpld تصمیم می گیرد وقفه را به zynq منتقل کند یا خیر.

برای اطلاعات بیشتر به آدرس <https://wiki.trenz-electronic.de/display/PD/SD+Card+Interface>

بروید .

قسمت سوم : نصب کردن برد بر روی vivado

در صورتی که به آدرس <https://wiki.trenz-electronic.de/display/PD/Installation>

بروید ، سه راهکار نصب برد بر روی vivado مطرح می شود . منظور از نصب برد آن است که به جای انتخاب چیپ برای شروع پروژه ، برد را انتخاب کنیم . مزیت انتخاب برد در راه اندازی سریع تنظیمات ps است . به عنوان مثال تنظیمات خودکار ram و یا راه اندازی تعدادی از پریفرال ها ی متصل به قطعات روی برد .

پیشنهاد می شود این عملیات را بر روی یک vivado ی تازه نصب شده انجام دهید .

برای این کار سه راه ارائه شده

اول دانلود رفرنس دیزاین بایست برای تمام این سه روش انجام شود . برای این کار باید به آدرس <https://shop.trenz-electronic.de/Download/?path=Trenz_Electronic/Modules_and_Module_Carriers/4x5/TE0820> بروید و فایل reference design مربوط به ورژن vivado را دانلود کنید .

توجه : حتما ورژن مطابق خود را دانلود کنید. در مشاهدات میدانی دیده شده در غیر این صورت با خطا در هنگام سنتز مواجه می شوید .

راه حل اول گفته از پروژه ی از قبل ساخته شده استفاده کنید .

برای راه دوم و سوم به پوشه ی board\_files بروید سپس فایل چیپ zynq مورد استفاده را بردارید . در مورد برد ما 3eg می باشد .

برای راه دوم یک فولدر جدید در مکانی دلخواه ایجاد می کنید سپس فایل انتخابی را در آن محا ذخیره می کنید . درمسیر C:/Users/<user>/AppData/Roaming/Xilinx/Vivado/

یا مسیر <*installdir*>/vivado/<version>/scripts/ فایلی به نام و پسوند "init.tcl"  برای قبل از ورژن 2017.1 و با نام  "Vivado\_init.tcl" برای بعد از آن با notepad تولید می کنیم.

سپس نوشتار زیر را در آن وارد می کنیم .

if { [catch {

  # #############################

  # insert settings here:

  set tepath "C:/TE"

  if {[file exists ${tepath}/]} {

    set\_param board.repoPaths $tepath

    puts "Set Board Part RepoPath: [get\_param board.repoPaths]"

  }

  # #############################

}] } {

  puts "Info: (TE) init.tcl failed"

}

در مقابل set tepath فولدر مسیری که فایل برد در آن است را قرار می دهید . در صورت صحیح انجام شدن کار پیام زیر را در هنگام باز کردن vivado مشاهده می کنید :



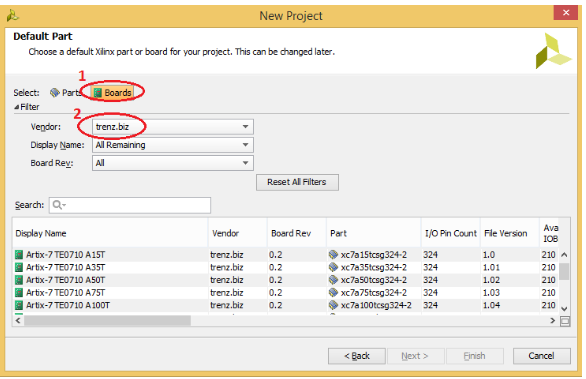
در مشاهدات میدانی گاها دیده شده در صورت تازه نصب نبودن vivado این روش کارساز نخواهد بود.

روش سوم بسیار سریع تر و راحت تر است اما شرکت trenz آن را پیشنهاد نمی کند . در عین حال دیده شده که این روش هم جواب داده .

فایل برد کپی شده را در مسیر <*installdir*>/vivado/<version>/data/boards/board\_files/ قرار می دهیم .

قابل توجه است که برای اضافه شدن برد به vivado باید vivado را در صورت باز بودن بست .

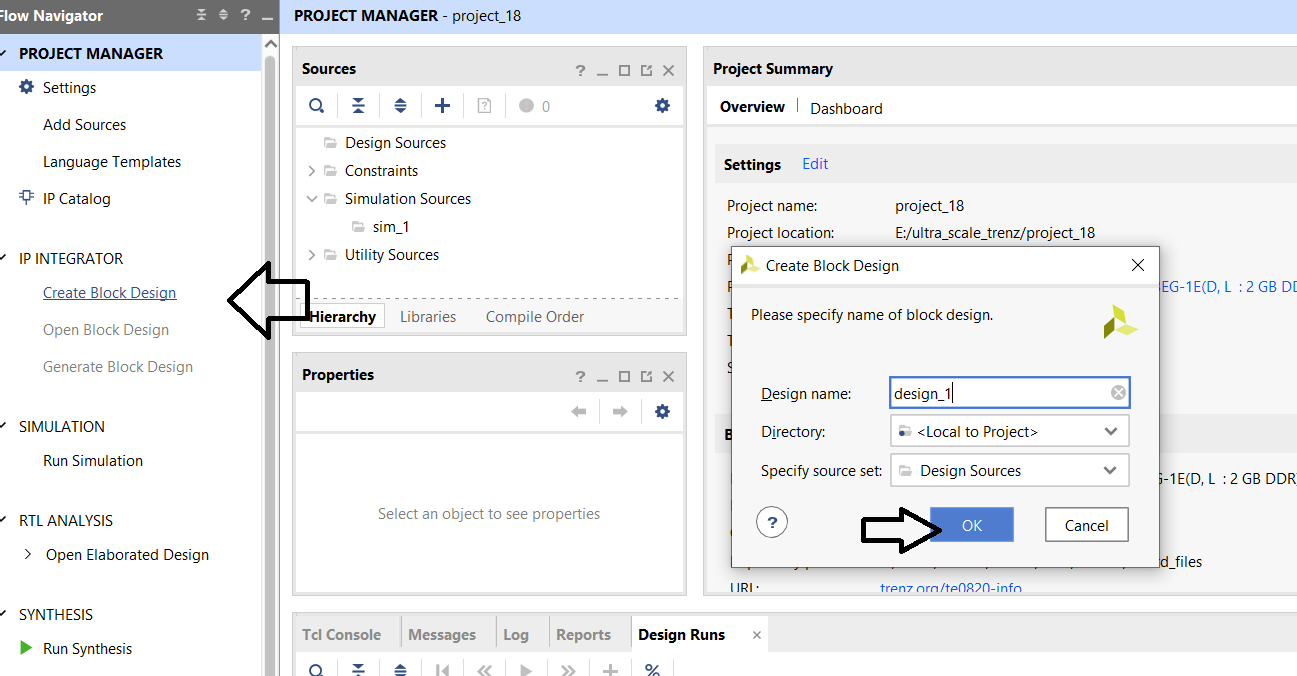
روش امتخاب برد را در تصویر زیر می بینید : که در مرحله ی انتخاب چیپ در بالا ی پنجره board را به جای part بایست انتخاب کرد . سپس در محل board rev بعد از انتخاب برد te820، revision3 باید انتخاب شود چون revision2 دارای 1 گیگابایت رم است ولی revision3 که خصوصیات برد ما مطابق با آن است دارای 2 گیگابایت رم است



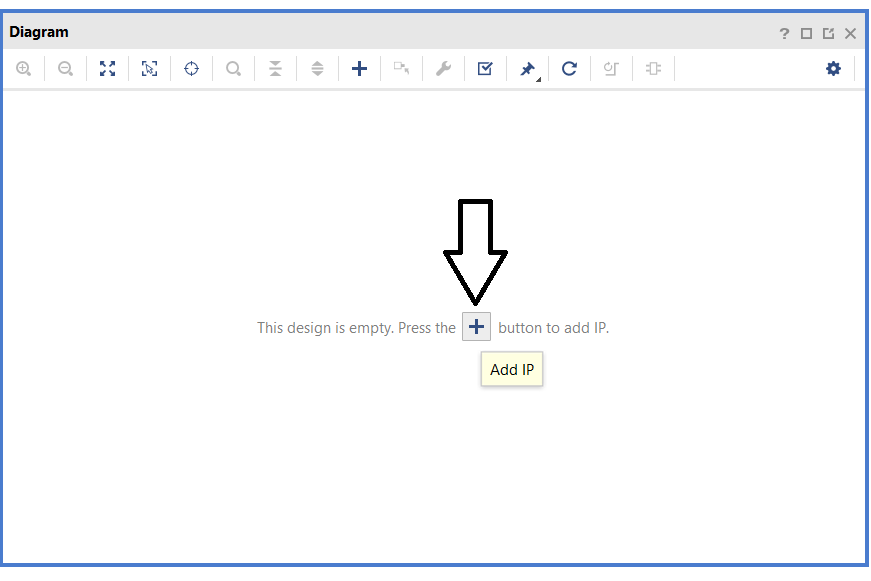
قسمت چهارم : اجرای برنامه ی hello world بر روی PS

ابتدا مانند بالا برد را انتخاب می کنیم و پروژه را باز می کنیم .

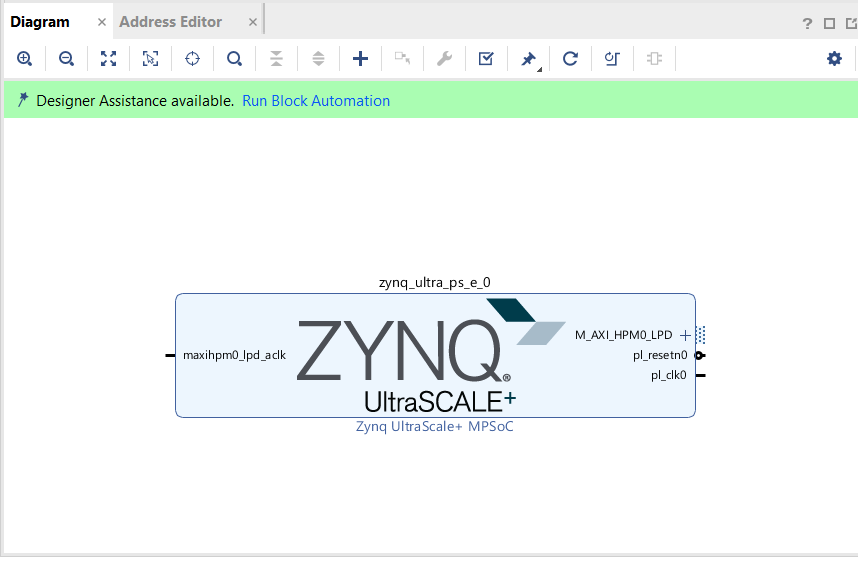
بر روی create block design کلیک کرده سپس ok را میزنیم



در زیر بر + کیک کرده سپس تایپ می کنیم zynq سپس enter می کنیم تا zynq اضافه شود .



سپس بر روی run block automation کلیک می کنیم تا تنظیمات رم و فعال سازی پریفرال ها انجام شود .



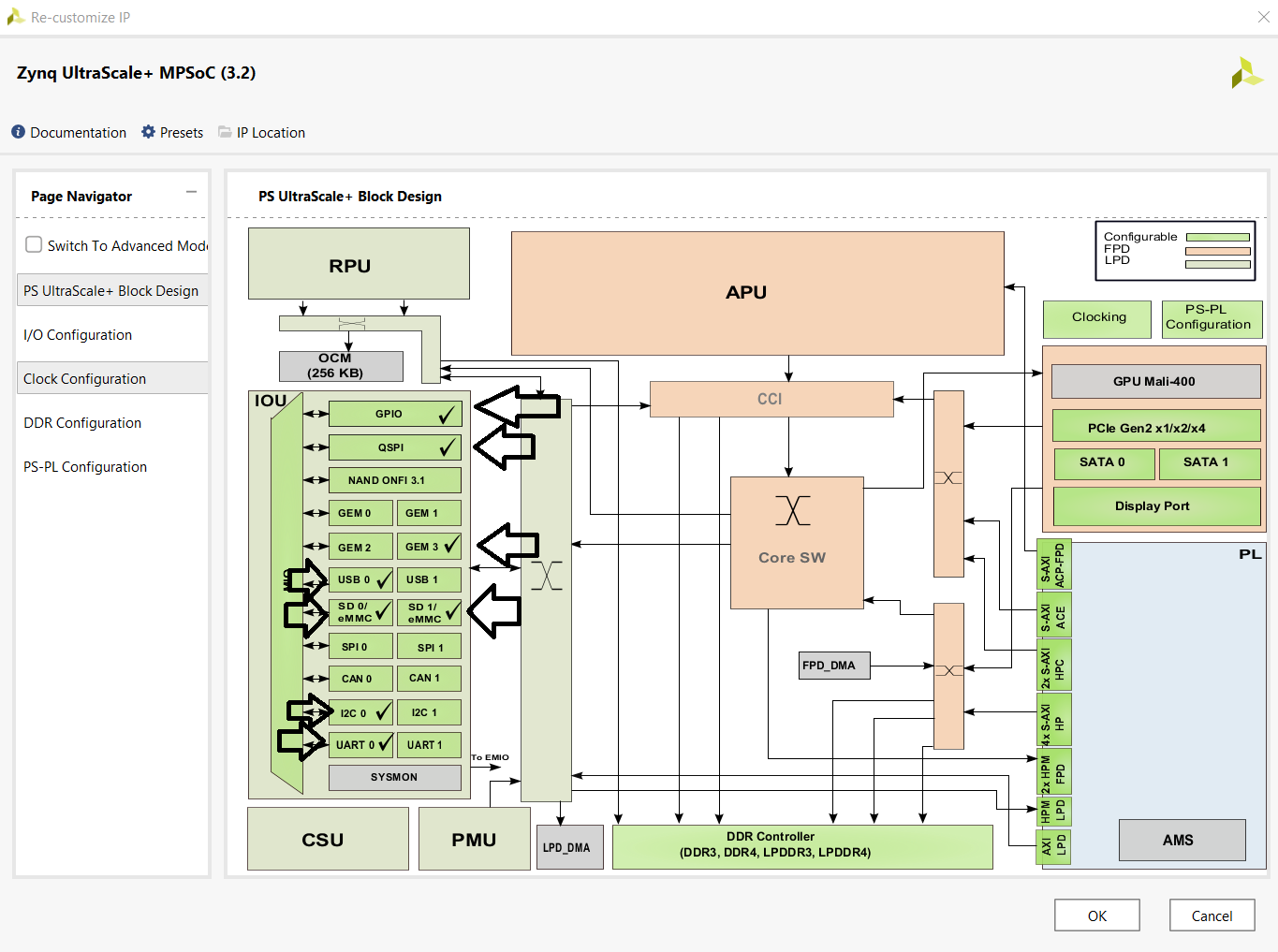
دوبار بر روی zynq کیلک می کنیم

در تصویر زیر qspi برای پروگرام کردن فلش ها استفاده می شود . در zynq باید توجه کرد که پروگرام mmc یا qspi باید در محیط sdk انجام شود و در محیط vivado امکانپذیر نیست .

Usb0 برای ارتباط با چیپ usb phy sd1 و sd2 برای ارتباط با emmc و sd card فعال شده

Gem3 برای ارتباط با چیپ اترنت و i2c0 برای ارتباط با چیپ pll و uart0 برای ارتباط سریال با pc استفاده شده .

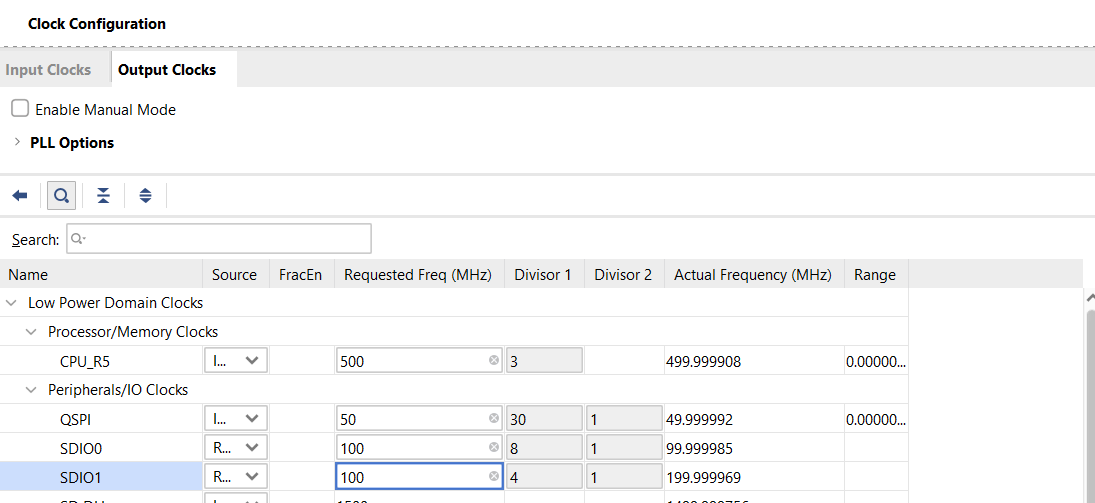
برای کار با qspi و sd باید فرکانس آن ها را کاهش داد . معمولا فرکانس qspi ها زیر 100 مگاهرتز است .



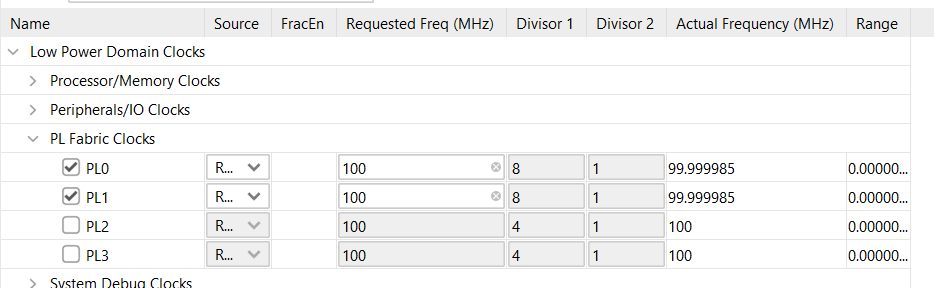
برای تغییر فرکانس به clock configuration میرویم

به لبه ی output clocks و به low power domain clocks میرویم . به قسمت peripherals/io clocks رفته و فرکانس sdio1 و sdio0 را روی 100 یا 50 مگ می گذاریم و فرکانس qspi را 50 مگ می گذاریم .

مانند زیر



اگر نمی خواهید از چیپ pll روی برد استفاده کنید یک راه فرار برای این کار دارید .

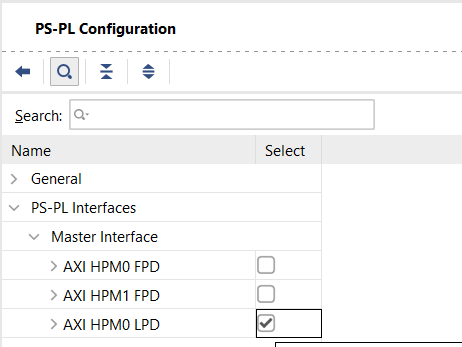
به جای تب peripheral/io clocks به pl fabric clocks میرویم و شماره یک آن را فعال می کنیم . شماره 0 آن برای axi ها استفاده خواهد شد که پیشفرض فعال است .

فرکانس آن هم در مقابل قابل تغییر خواهد بود .

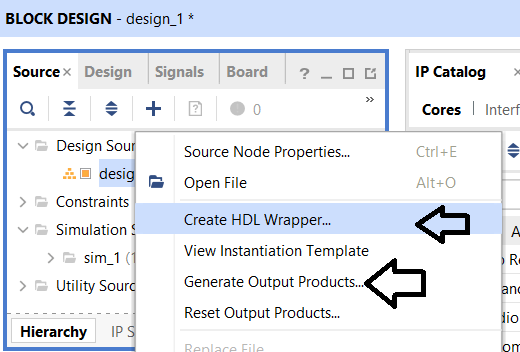
سپس بر روی pl\_clk1 کلیک راست کرده و make external می کنیم .(فعلا نیاز نیست پس این کار را نمی کنیم.

در صورت داشتن یک پروژه ی بالادست که در بخش axi\_gpio به آن میرسیم می توان آن را به صورت component به پروژه اضافه کرد .

اکنون که ارتباط axi نداریم باید maxi\_lpm\_clk را از zynq حذف کنیم تا با خطا مواجه نشویم . از قسمت ps-pl configuratuion در محل تصویر زیر تیک را برمی داریم .



سپس در تب source بر روی design کلیک راست کرده و generate output product و بعد از اتمام ، create hdl wrapper را می زنیم .

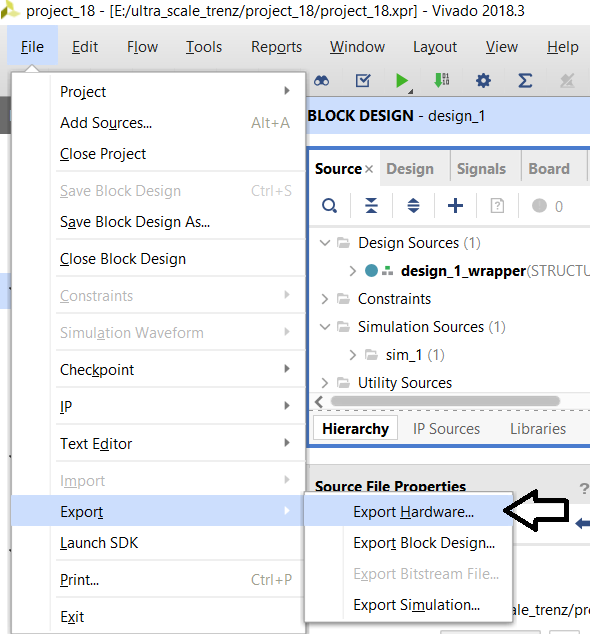


اکنون از flow navigator بایست generate bitstream را زد .البته کد hdl خاصی نداریم اما در همه ی پروژه ها این موضوع را باید رعایت کرد .

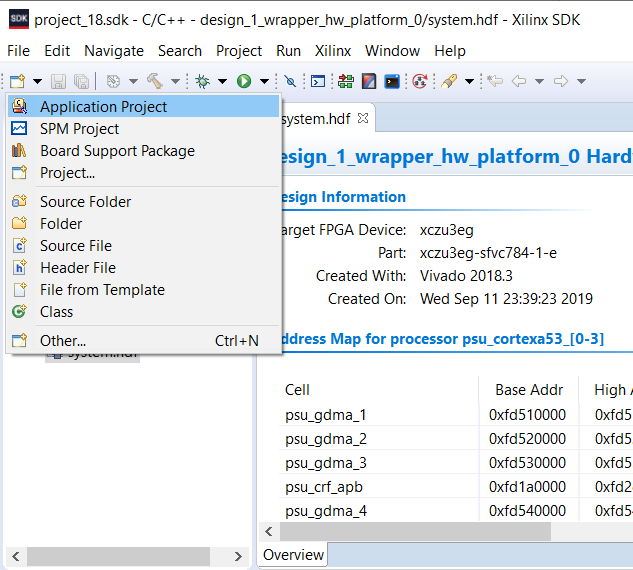
بعد از اتمام به تب file رفته از قسمت export بایست export hardware را میزنیم . این کار یک فایل hdf برای برای انجام تنظیمات بر روی sdk تولید می کند.

می توان bitstream را هم include کرد تا در محیط sdk برای پروگرام روی فلش یا sd استفاده شود.

بعد مراحل بالا دوباره به تب file رفته launch sdk را میزنیم.

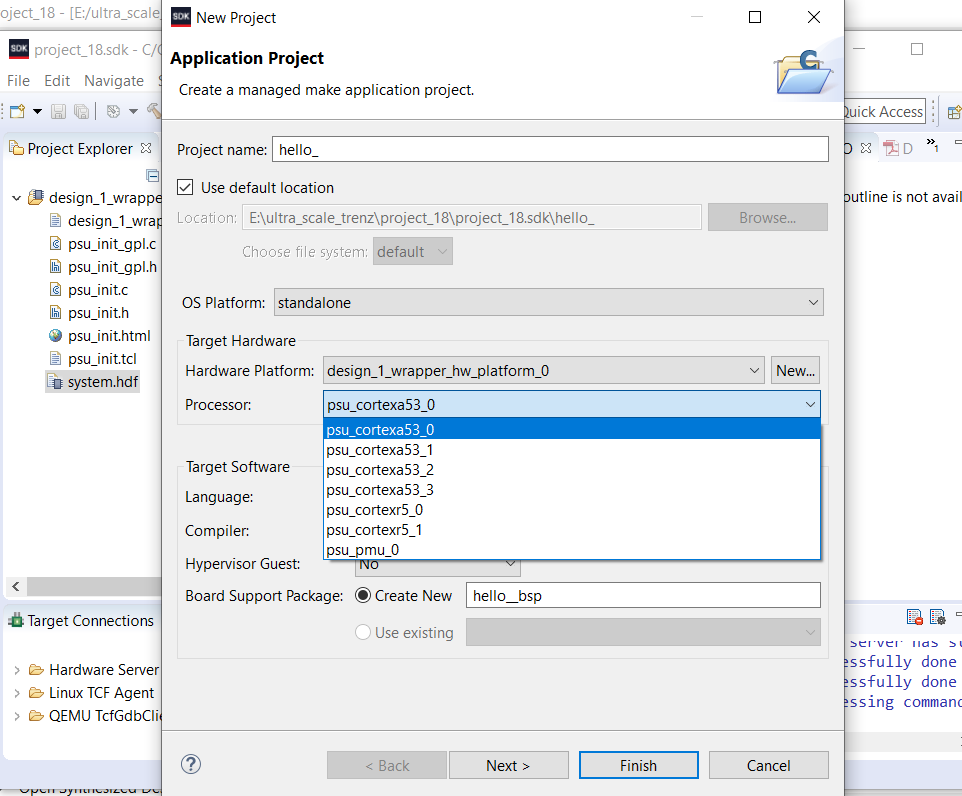


اکنون در sdk از تب file سپس new سپس application project



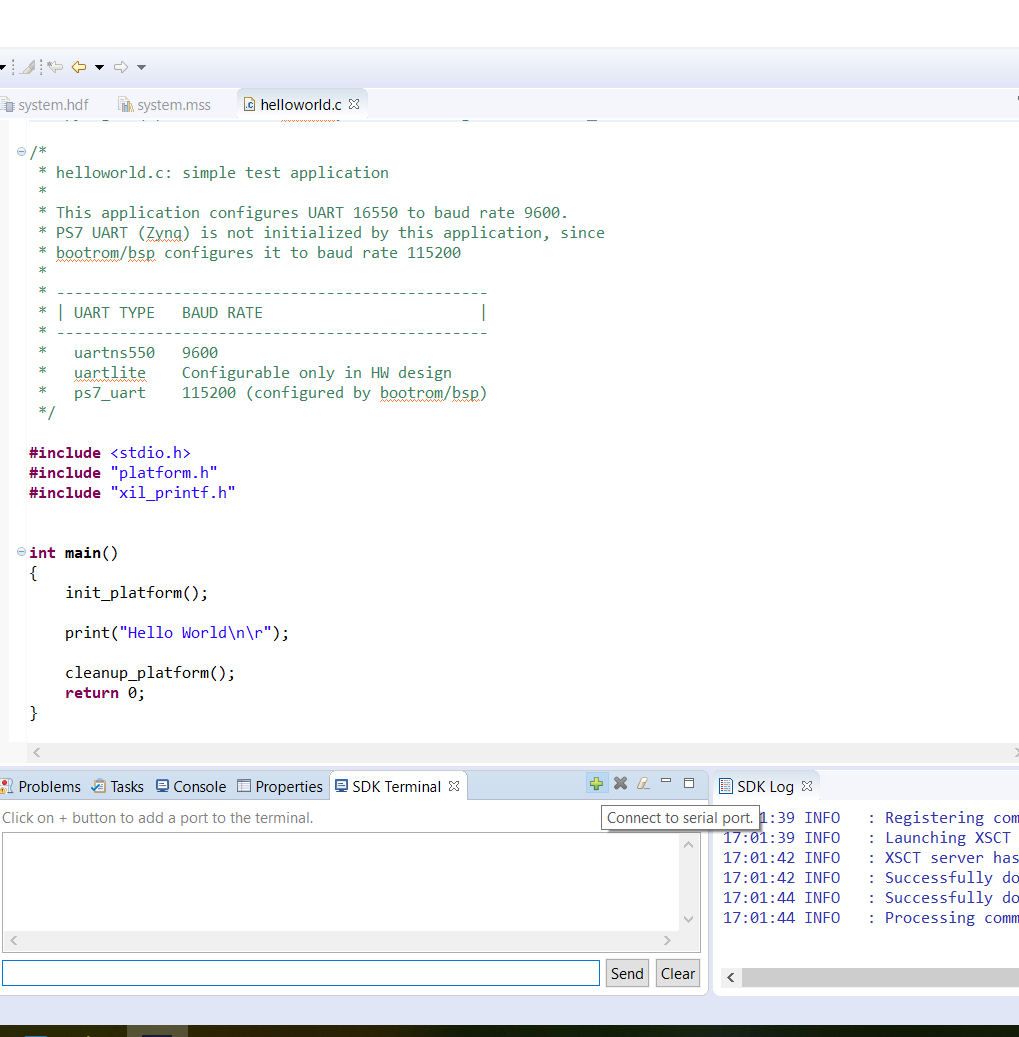
سپس پنجره ی زیر باز می شود :

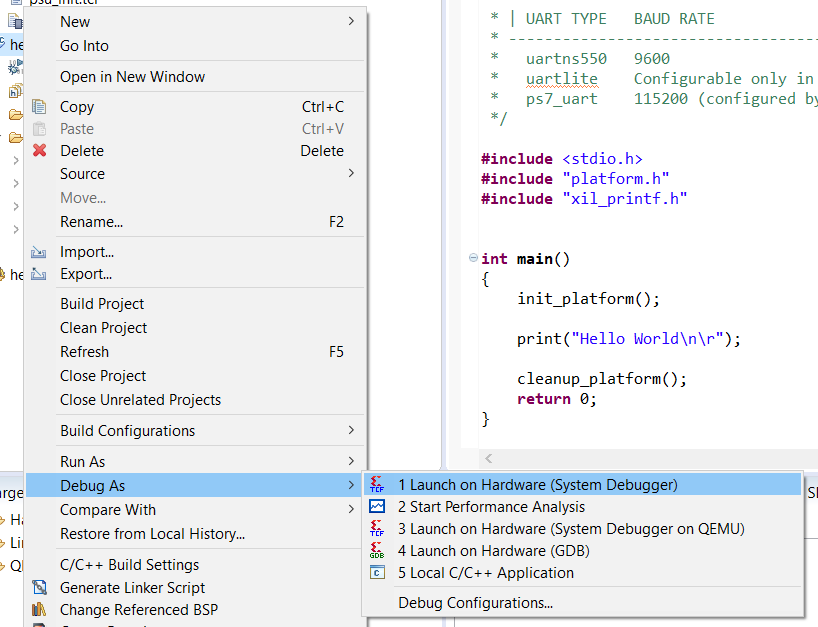
از پنجره ی processor انتخاب می کنیم که برنامه ای که می نویسیم روی کدام هسته اجرا شود .4 تای اول هسته های arm a53 و 2 تای بعدی arm r5 و آخری به نام pmu یک پروژه ی microblaze ثابت بر روی pl است که مخفف power management unit است که کار آن از نام آن مفهوم است.



می توان next را انتخاب کرد و از پروژه های از قبل نوشته شده استفاده کرد . اگر uart1 یا uart2 را فعال کرده باشید به صورت پیشفرض روی برنامه ی hello\_world قرار دارد و اگر نه بر روی empty\_project قرار دارد .

اکنون uart0 فعال است . و پروژه به صورت زیر در می آید . در پایین علامت + که همان connect to serial port است از تب SDK TERMINAL انتخاب می شود.



بعد از تنظیم پورت به صورت زیر دیباگ را انجام می دهیم که بعد از وارد شن به محیط دیباگ و زدن دکمه ی resume بایست hello world را داخل terminal نشان می دهد . .

قسمت چهارم:راه اندازی emio

پایه های ZYNQ دو قسمت است . گروهی متعلق به PL و گروه بسیار کمتری متعلق به PS است .

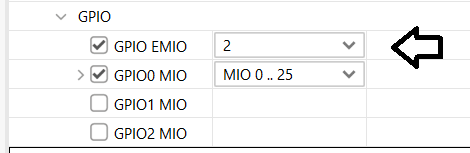
به پایه های PS که به صورت ذاتی متعلق به PS است MIO می گویند.

اگر پایه های MIO جوابگوی PERIPHERAL های PS نباشد می توان از پایه های PL برای ارتباط استفاده کرد که به این پایه ها EMIO می گویند.

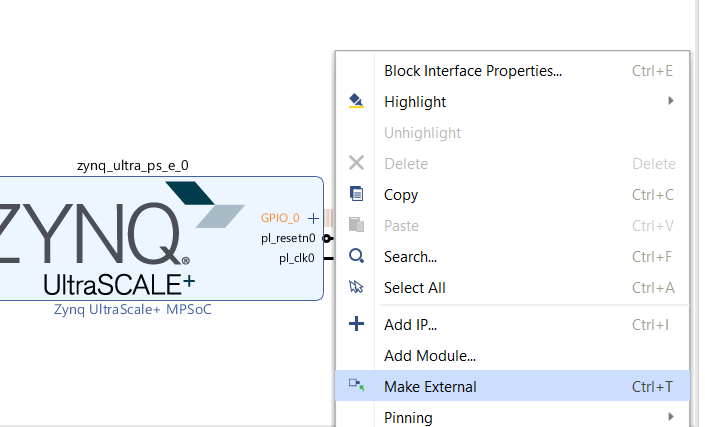
دو عدد LED به پایه های PL متصل است که می خواهیم به وسیله ی PS آن ها را کنترل کنیم.

تمام اتفاقات قسمت قبل باید بیفتد سپس طبق تصویر زیر برای GPIO باید EMIO فعال شود.

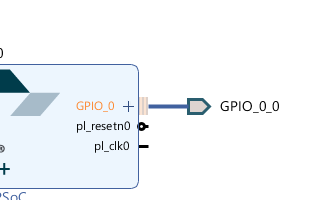
از 95 پایه ی ممکن 2 عدد را تنها می خواهیم.



اکنون MAKE external را میزنیم تا پورت برایمان ایجاد کند .

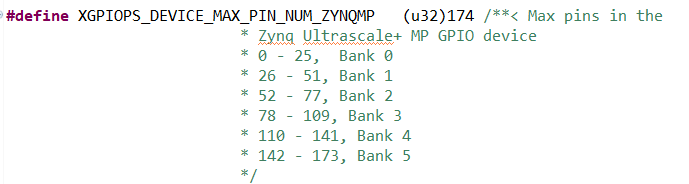


مانند زیر :



در implementation آن ها را به AH6 و AD6 متصل کنید . سپس کد زیر را در SDK وارد کنید تا به صورت فاصله ها 1 ثانیه روشنی آن ها TOGGLE می شوند.

قبل از آوردن کد برای gpio در ultrascale 6 بانک داریم . که سه بانک اول توسط mio و سه بانک دوم توسط emio کنترل می شود.



یعنی شماره ی 78 به بعد . برای همین در کد زیر دو پورت 78 و 79 موثرند.

**#include** "sleep.h"

**#include** "xgpiops.h"

**int** **main**()

{

XGpioPs gp;

XGpioPs\_Config \*gpcon;

gpcon=XGpioPs\_LookupConfig(XPAR\_PSU\_GPIO\_0\_DEVICE\_ID);

XGpioPs\_CfgInitialize(&gp,gpcon, gpcon->BaseAddr);

XGpioPs\_SetDirectionPin(&gp,78,1);

XGpioPs\_SetDirectionPin(&gp,79,1);

XGpioPs\_SetOutputEnablePin(&gp,78,1);

XGpioPs\_SetOutputEnablePin(&gp,79,1);

**while**(1)

{

XGpioPs\_WritePin(&gp,78,1);

XGpioPs\_WritePin(&gp,79,0);

sleep\_A53(1);

XGpioPs\_WritePin(&gp,79,1);

XGpioPs\_WritePin(&gp,78,0);

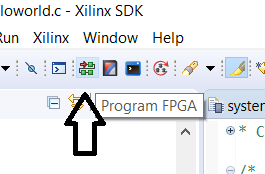
sleep\_A53(1);

}

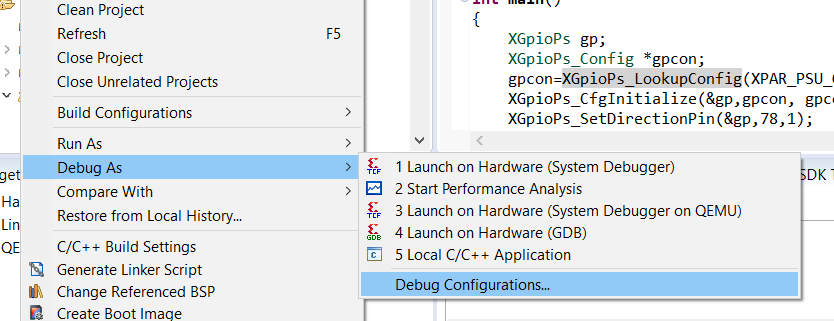
}

توجه کنید که باید برای این پروژه fpga را هم باید پروگرام کنید زیرا دارید پایه های pl را استفاده می کنید.

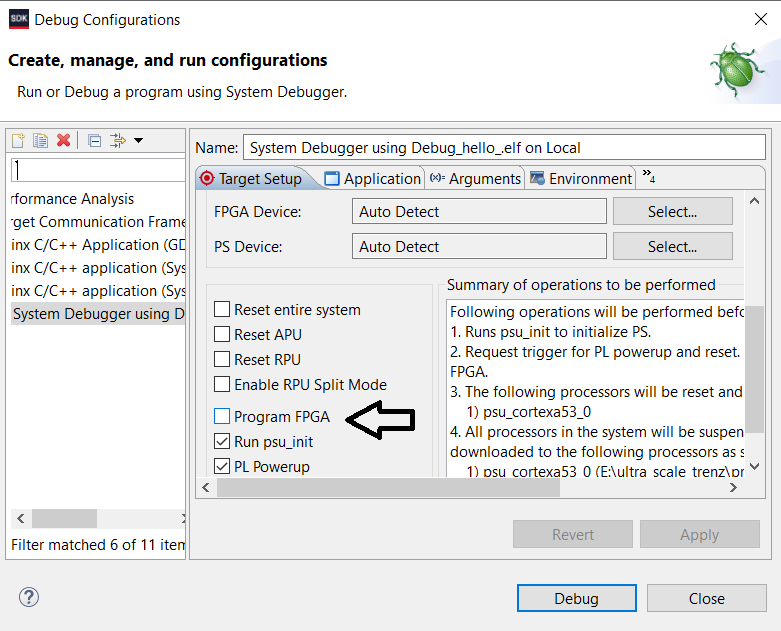
برای این کار یا باید این کار را به صورت دستی انجام داد یا یک تیک در debug configuration فعال کنیم . برای انجام دستی به صورت زیر :



و برای اتوماتیک ابتدا مرحله ی زیر :



و سپس تیک program fpga را می زنیم :



قسمت چهارم : دانلود برنامه بر روی qspi یا sd

می خواهیم همین برنامه ی emio را روی کار ت حافظه بریزیم .

برای این کار سه فایل elf ویک فایل bit لازم داریم . سه فایل elf توسط sdk و فایل بیت توسط vivado تشکیل شده .

ابتدا باید یک فایل elf مربوط به pmu و یکی برای fsbl تشکیل داد .

نحوه ی تشکیل pmu به صورت زیر است :

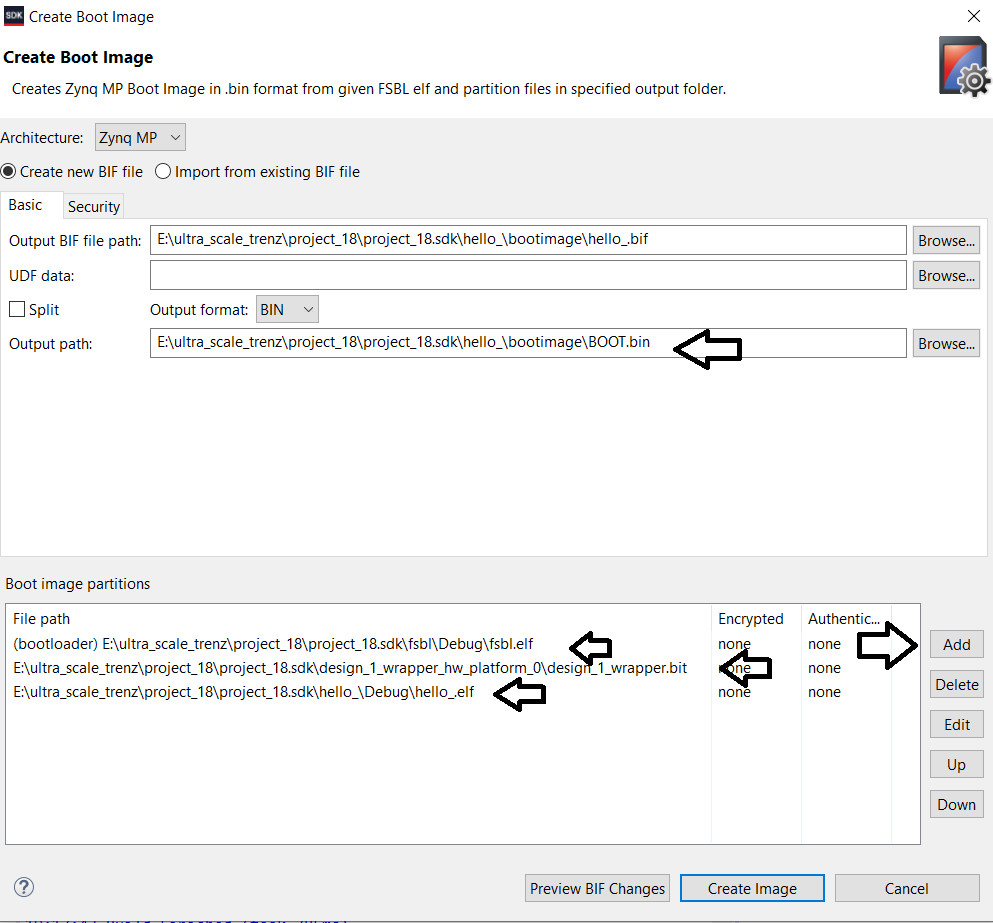
مانند قبل یک application project می سازیم سپس processor را بر روی pmu قرار می دهیم و یک نام برای آن می گذاریم . سپس next را می زنیم و مطمئن می شویم بر روی

Zynqmp pmu firmware قرار دارد سپس finish را می زنیم .

بر روی پروژه ی تشکیل شده از داخل project explorer کلیک راست می کنیم و build را میزنیم تا elf ساخته شود.

برای fsbl هم processor بر روی همان هسته ی arm پیشین قرار می گیرد و next زده می شود و این بار از بین برنامه های از پیش آماده zynqmp fsbl انتخاب می شود . این را هم مانند pmu ، build می کنیم .

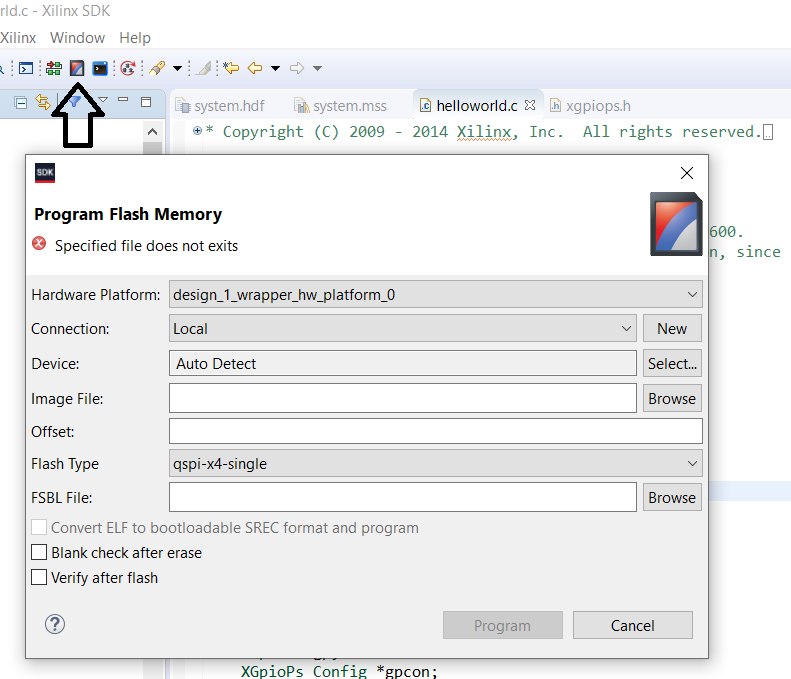
اکنون بر روی پروژه اصلی از داخل flow navigator کلیک راست کرده سپس create boot image را انتخاب می کنیم.



برای اضافه کردن pmu و fsbl اگر موجود نباشند add را باید زد . سپس pmu.elf را که قبلا تشکیل شده را اضافه می کنیم . سپس create image را می زنیم فایلی به نام boot.bin تشکیل دهد .

برای qspi یا emmc یا nand flash می توان به صورت زیر عمل کرد .

ابتدا گزینه ی program flash را می زنیم همان جایی که در عکس زیر فلش به آن اشاره می کند سپس برای برد ما flash type را qspi-dual-x8-parallel می گذاریم سپس boot.bin را در محل image file قرار می دهیم بعد از آن fsbl.elf و سپس program را فشار می دهیم . البته بهتر است قبلا حالت بوت را بر روی qspi قرار دهیم .



برای sd card هم حالت بوت را بر روی sd می گذاریم و boot.bin را روی آن قرار می دهیم .

بعد از خاموش و روشن شدن از روی sd card بوت خواهد کرد .

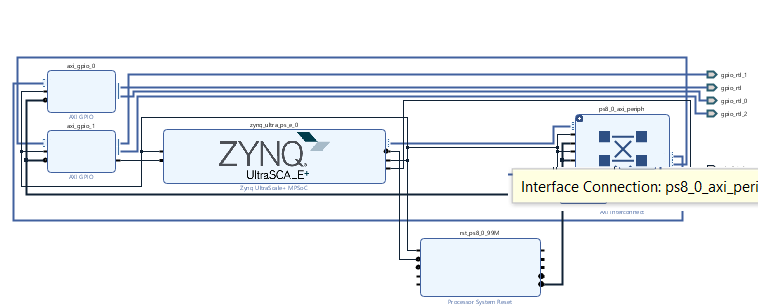
قسمت پنجم : axi\_gpio

قرار شد یک سری دیتا از روی pl به روی ps فرستاده شود و بالعکس . می دانستیم این عملیات باید به واسطه ی axi انجام شود. اما چطور انجام شد آن معلوم نبود . دیده بودیم که به وسیله ی axi\_dma انجام می شود اما به وسیله ی خط axi\_stream . ارسال stream برای دیتا های بسیار پر حجم انجام می شود برای همین دیدیم می توان این کار را به وسیله ی axi\_gpio انجام داد. به این صورت که به واسطه ی axi\_gpio می توانیم یک باس ایجاد کنیم .

Axi\_gpio\_0 دارای دو گروه 32 خطی است . می خواستیم یک باس به طول 64 بیت به صورت نیمه دو طرفه ایجاد کنیم یعنی به صورت inout اما به علت مشکلات در implementation و خطا هایی همچون اینکه چرا ibuf و obuf نگذاشته اید. به همین دلیل دو خط هر کدام 32 بیت و یکی خروجی و یکی ورودی استفاده شد.

Axi\_gpio\_1 هم دارای 2 گروه 2 خطی برای کنترل باس است . 2 بیت خروجی و 2 دو بیت ورودی. ورودی ها از سمت PL به سمت PS میرفتند و خروجی ها از PS به PL .

تصویر دیاگرام به صورت زیر است . پس از تشکیل wrapper برای دیاگرام آن را به صورت یک component به یک برنامه ی بالادست اضافه کردیم که کار برنامه ی بالادست ارسال و دریافت اطلاعات بود .



بعد از مراحل بالا در داخل sdk هم بایست عملیات هایی انجام می دادیم برای ارسال و دریافت . مثلا بعد از ارسال از ps به pl باید یک فلگ میزدیم تا pl متوجه اطلاعات جدید روی خط بشود. همچنین اگر دیتای جدید از pl هم آمد ، با یک پرچم باید pl به ps خبر بدهد که دیتای جدید روی خط فرستاد است.

همچنین بایست این را در نظر داشت که باید فرآیندی برای زدن پرچم اطلاعات بعدی و این که پرچم ها صفر شوند وجود داشته باشد . پس پرچم هایی برای تایید دریافت اطلاعات برای دریافت اطلاعات بعدی باید به وجود بیایند که در کنار دیگر پرچم ها قرار می گیرند.

کد زیر یک برنامه ی ساده برای ارسال یک دیتا از pl به ps را داخل vivado نشان می دهد.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity axi\_intr is

port(clk : in std\_logic;

make\_intr\_for\_ps : in STD\_LOGIC\_VECTOR (0 downto 0)

);

end axi\_intr;

architecture Behavioral of axi\_intr is

component design\_1\_wrapper is

port (

gpio\_rtl\_0\_tri\_o : out STD\_LOGIC\_VECTOR ( 31 downto 0 );

gpio\_rtl\_1\_tri\_i : in STD\_LOGIC\_VECTOR ( 1 downto 0 );

gpio\_rtl\_2\_tri\_o : out STD\_LOGIC\_VECTOR ( 1 downto 0 );

gpio\_rtl\_tri\_i : in STD\_LOGIC\_VECTOR ( 31 downto 0 );

pl\_clk1\_0 : out STD\_LOGIC

);

end component;

signal data\_axi\_to\_ps : STD\_LOGIC\_VECTOR ( 31 downto 0 ):=x"12345678";

signal data\_axi\_from\_ps : STD\_LOGIC\_VECTOR ( 31 downto 0 );

-- signal make\_intr\_for\_ps : STD\_LOGIC\_VECTOR ( 0 downto 0 );

signal make\_intr\_for\_pl : STD\_LOGIC\_VECTOR ( 0 downto 0 );

signal clk\_pl : STD\_LOGIC;

signal ps\_signal\_recieved : STD\_LOGIC\_VECTOR ( 0 downto 0 );

signal pl\_signal\_recieved : STD\_LOGIC\_VECTOR ( 0 downto 0 );

signal gpio\_rtl\_2\_tri\_o : STD\_LOGIC\_VECTOR ( 1 downto 0 );

signal gpio\_rtl\_1\_tri\_i : STD\_LOGIC\_VECTOR ( 1 downto 0 );

begin

u1 : component design\_1\_wrapper port map

( gpio\_rtl\_tri\_i => data\_axi\_to\_ps,

gpio\_rtl\_1\_tri\_i => gpio\_rtl\_1\_tri\_i,

gpio\_rtl\_2\_tri\_o => gpio\_rtl\_2\_tri\_o,

gpio\_rtl\_0\_tri\_o => data\_axi\_from\_ps, --gpio\_rtl\_tri\_i

pl\_clk1\_0 => clk\_pl

);

make\_intr\_for\_pl <= gpio\_rtl\_2\_tri\_o( 0 downto 0);

ps\_signal\_recieved <= gpio\_rtl\_2\_tri\_o( 1 downto 1);

gpio\_rtl\_1\_tri\_i ( 0 downto 0) <= make\_intr\_for\_ps ( 0 downto 0);

gpio\_rtl\_1\_tri\_i ( 1 downto 1) <= pl\_signal\_recieved ( 0 downto 0);

-- data\_axi\_to\_ps <= x"12345678";

process(make\_intr\_for\_pl,ps\_signal\_recieved,clk,make\_intr\_for\_ps,pl\_signal\_recieved)

variable flagrec : std\_logic;

begin

if(make\_intr\_for\_pl = "1") then

pl\_signal\_recieved <= "1";

flagrec := '1';

data\_axi\_to\_ps <= std\_logic\_vector(shift\_right(unsigned(data\_axi\_to\_ps),1));

end if;

if (flagrec = '1') then

pl\_signal\_recieved <= "0";

flagrec := '0' ;

end if;

end process;

end Behavioral;

کد sdk هم به صورت زیر است که در صورت ایجاد یک axi\_intrrupt اطلاعات از روی خط برداشت می شود.

**#include** <stdio.h>

**#include** "platform.h"

**#include** "xil\_printf.h"

**#include** "xgpio.h"

**#include** "xil\_exception.h"

**#include** "xscugic.h"

**#include** "xparameters.h"

**void** **handle\_axi**();

XGpio axigp\_1;

u32 data;

XGpio axigp\_0;

**int** **main**()

{

XGpio\_Config \*axigp\_con\_0;

axigp\_con\_0 = XGpio\_LookupConfig(XPAR\_AXI\_GPIO\_0\_DEVICE\_ID);

XGpio\_CfgInitialize(&axigp\_0,axigp\_con\_0,axigp\_con\_0->BaseAddress);

//XGpio axigp\_1;

XGpio\_Config \*axigp\_con\_1;

axigp\_con\_1 = XGpio\_LookupConfig(XPAR\_AXI\_GPIO\_1\_DEVICE\_ID);

XGpio\_CfgInitialize(&axigp\_1,axigp\_con\_1,axigp\_con\_1->BaseAddress);

XScuGic gic;

XScuGic\_Config \*gic\_conf;

gic\_conf = XScuGic\_LookupConfig(XPAR\_PSU\_ACPU\_GIC\_DEVICE\_ID);

XScuGic\_CfgInitialize(&gic,gic\_conf,gic\_conf->CpuBaseAddress);

Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_IRQ\_INT,(Xil\_ExceptionHandler)XScuGic\_InterruptHandler,&gic);

XScuGic\_SetPriorityTriggerType(&gic,XPAR\_FABRIC\_AXI\_GPIO\_1\_IP2INTC\_IRPT\_INTR,0xa0,0b11);//rising \_edge

XScuGic\_Connect(&gic,XPAR\_FABRIC\_AXI\_GPIO\_1\_IP2INTC\_IRPT\_INTR,(Xil\_InterruptHandler)handle\_axi,(**void**\*)&axigp\_1);

XGpio\_InterruptEnable(&axigp\_1,XGPIO\_IR\_CH1\_MASK);

XGpio\_InterruptGlobalEnable(&axigp\_1);

XScuGic\_Enable(&gic,XPAR\_FABRIC\_AXI\_GPIO\_1\_IP2INTC\_IRPT\_INTR);

Xil\_ExceptionEnableMask(XIL\_EXCEPTION\_IRQ);

**while**(1)

{

}

init\_platform();

print("Hello World\n\r");

cleanup\_platform();

**return** 0;

}

**void** **handle\_axi**()

{

u32 geten;

geten = XGpio\_InterruptGetEnabled(&axigp\_1);

**if**( geten == XGPIO\_IR\_CH1\_MASK )

{

XGpio\_InterruptClear(&axigp\_1,XGPIO\_IR\_CH1\_MASK);

data = XGpio\_DiscreteRead(&axigp\_0,1);

}

}