

Spring, 2020



1090140071

FPGA设计及应用

Xilinx嵌入式处理器ZYNQ-7000
On PYNQ-Z1/2 by Vivado/Vitis

大连理工大学 电信学部
夏书峰

1. Xilinx嵌入式处理器概况

- 嵌入式处理器种类

- 8位软核：PicoBlaze™
- 32位软核：MicroBlaze™
- 32位硬核：PowerPC®
(PPC405、PPC440)



- 硬核：多核ARM处理器为中心可扩展平台



ZYNQ-7000 入门级ARM Cortex-A9, 32位

ZYNQ UltraScale+ 中高端系列Cortex-A53 & Cortex-R5
32/64位

- 嵌入式系统开发工具

- ISE Design Suite中EDK和SDK分别是硬、软件开发环境
- Vivado和Vitis分别是7系列的硬、软件开发环境

关于MicroBlaze处理器



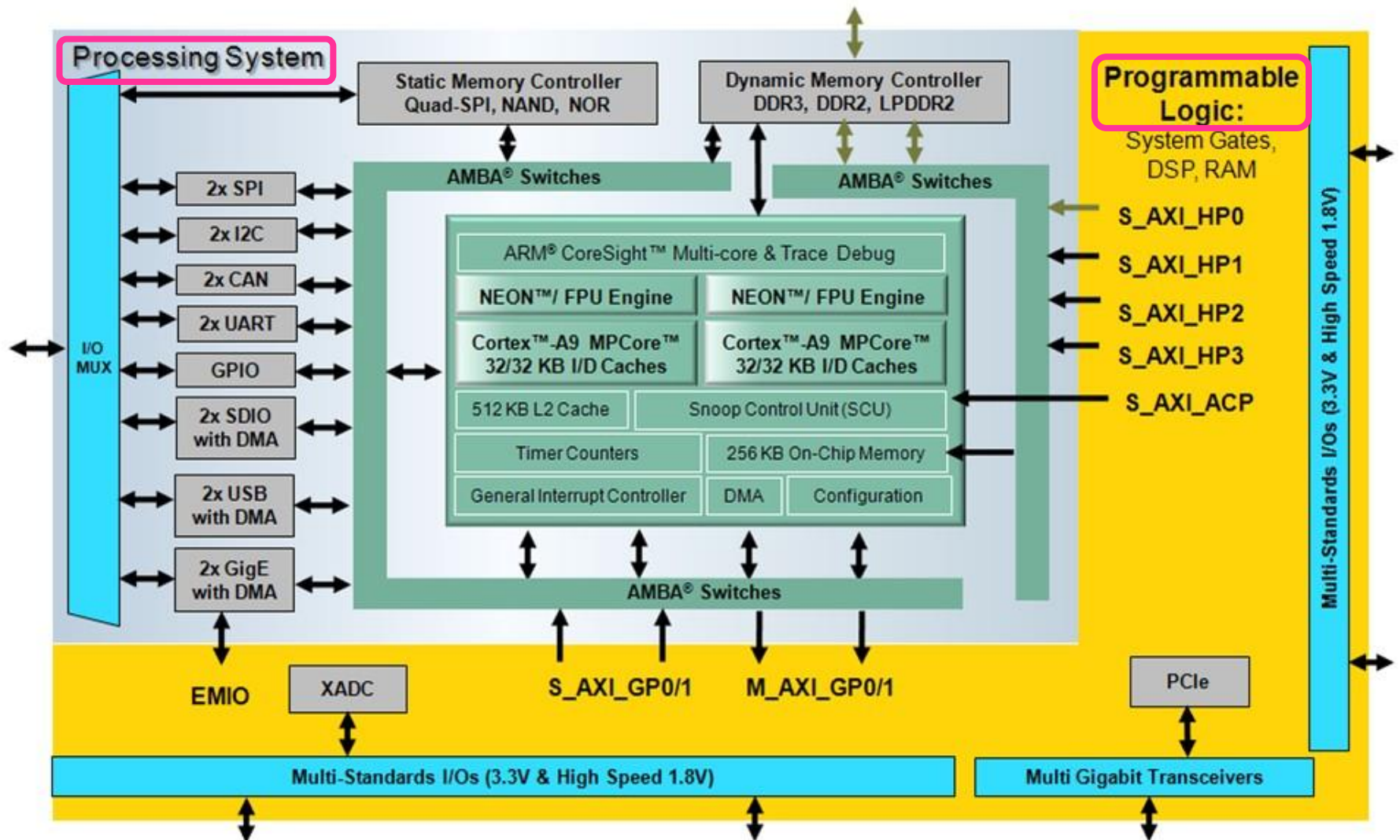
- 32-bit RISC微处理器
- **哈佛结构**—指令和数据管线分离（地址分离）
- 32位地址、32位数据总线
- 32个32位GPR(General Purpose Register)
- 3级流水线—取指、译码、执行
- 字宽—32位全字、16位半字、8位字节
- 算法逻辑单元—ALU(Arithmetic Logic Unit)
- 移位寄存器、2周期桶形移位器
- 3周期乘法器、34周期除法器
- 浮点单元FPU—6周期加减乘、30周期除，1000LUT
- 指令Cache、数据Cache
- 软IP核，对速度优化，Spartan3到100MHz，92DMIPs
- 资源需求—Spartan3速度-5级FPGA，983~1318LUTs
- 通过FSL(Fast Simplex Link)，MB与IP之间点对点高速互联，对具体应用进行硬件加速

关于ZYNQ7000的Cortex-A9处理器

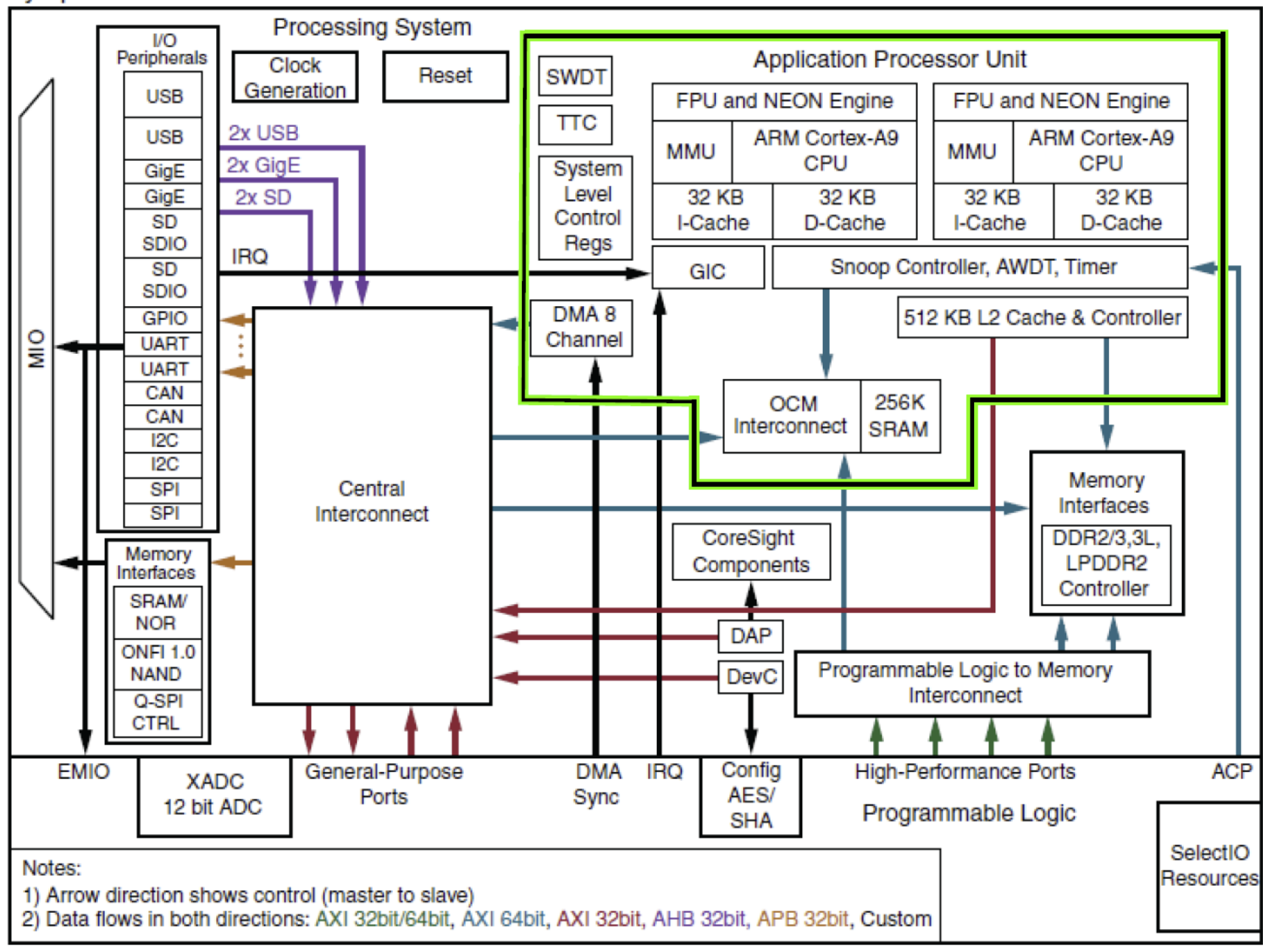


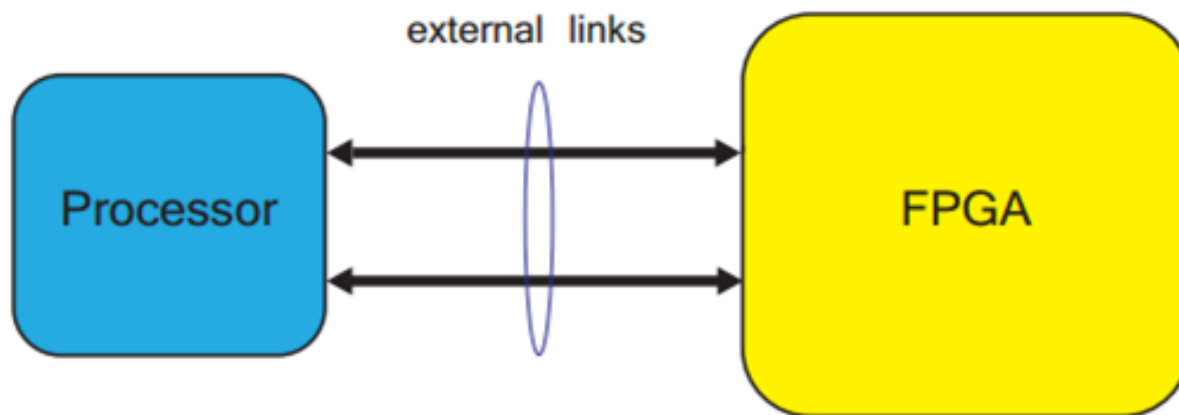
- 硬核32-bit Cortex-A9 **MPCore**, ARMv7A指令集
- 支持Jazelle RCT和Jazelle DBX Java加速
- 冯·诺依曼体系结构—指令和数据管线共用
- 32KB指令、32KB数据 L1缓存, 双核共享512KB L2缓存
- NEON 128位 SIMD协处理器, 多媒体加速
- 私有定时器和看门狗
- 运行时选项允许单处理器, 对称、非对称多处理配置
- 最高运行时钟866MHz (Z-7020)

ZYNQ-7000的PS+PL架构



Zynq-7000 AP SoC

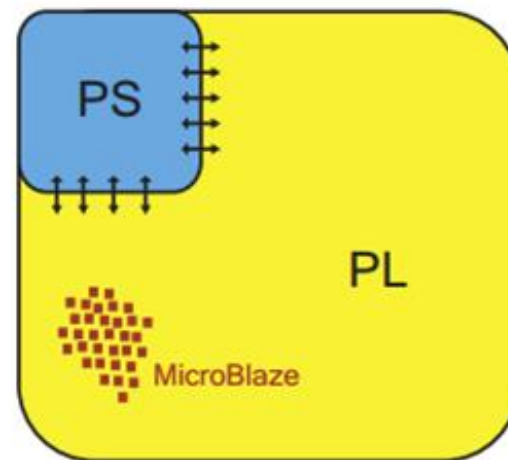




Discrete Component Architecture

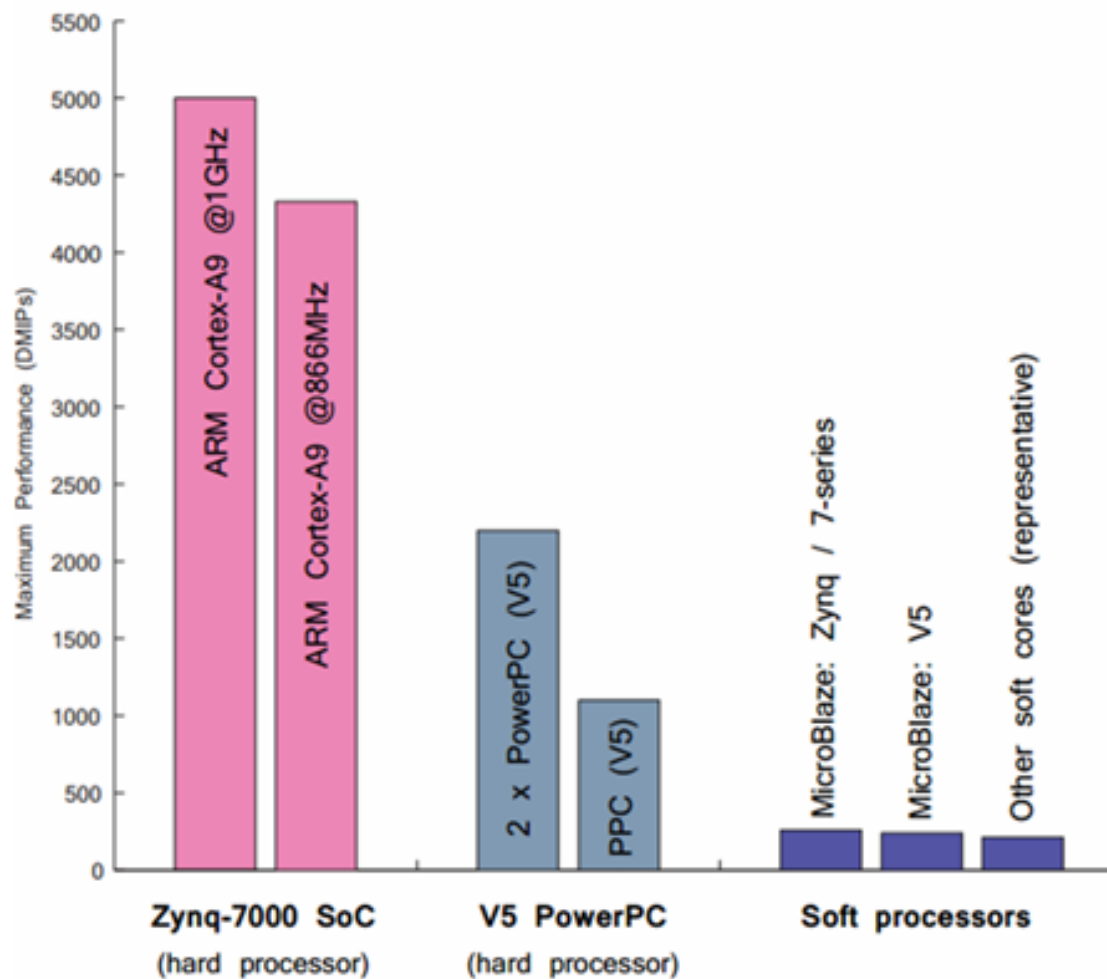


FPGA with soft processor
(e.g. MicroBlaze)



Zynq Architecture
(optional MicroBlaze)

Xilinx嵌入式处理器的性能



软核MB和硬核ARM性能

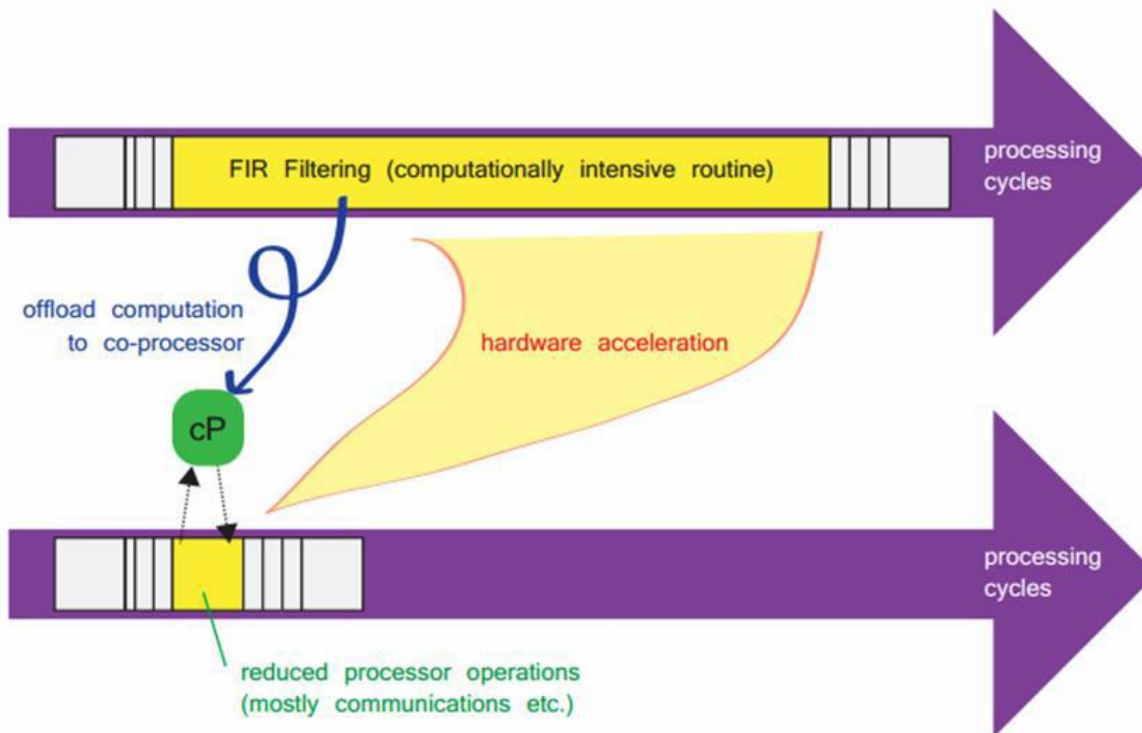
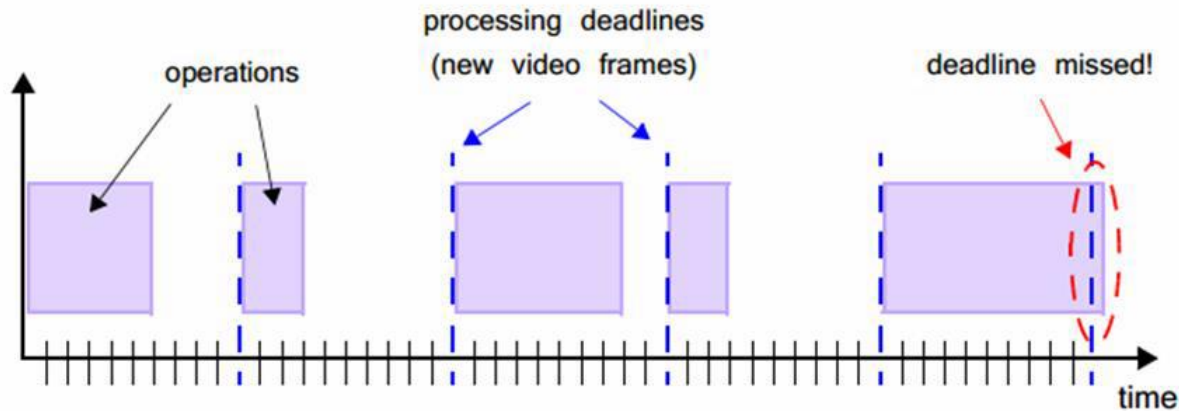
处理器类型	配置	处理器性能 (CoreMark) ^a
MicroBlaze	125MHz; 5 级流水线 (Virtex-5)	238
ARM Cortex-A9	1GHz; 两个核的总和	5927
ARM Cortex-A9	800MHz ^b ; 两个核的总和	4737

处理器类型	配置	处理器性能 (DMIPS) ^a
MicroBlaze	面积优化 (3 级流水线)	196 ^b
	带跳转优化的性能优化 (5 级流水线)	228 ^b
	不带跳转优化的性能优化 (5 级流水线)	259 ^b
ARM Cortex-A9	1GHz; 两个核的总和 ; 每个核 2500	5000 ^c

a. 所遇的配置都是基于速度级别 -3 (最快的级别), 应该被认为是最好情况下的

- 软核MB用PL部分实现, 硬核ARM集成在PS部分

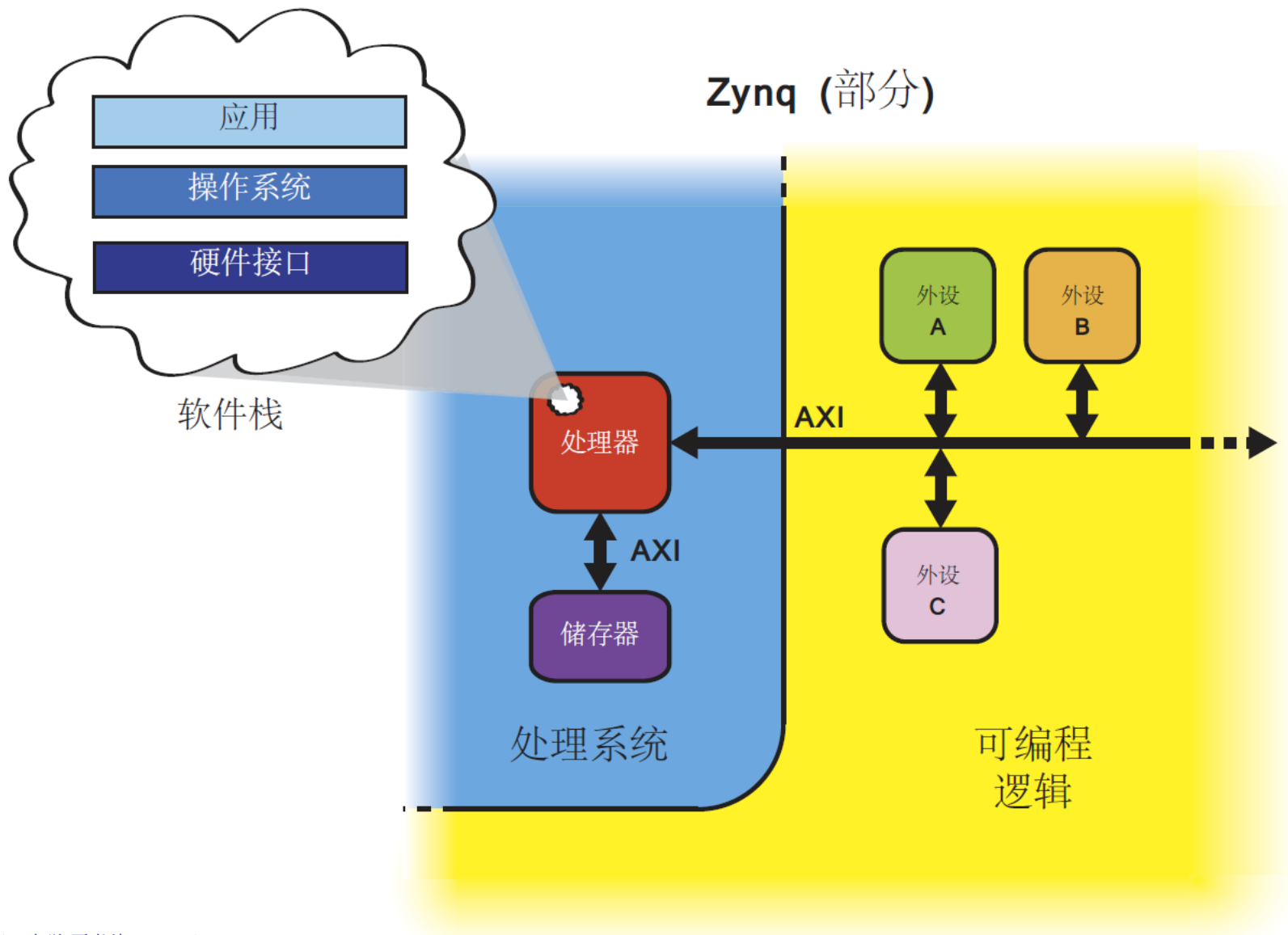
软硬件协同设计加速应用运行



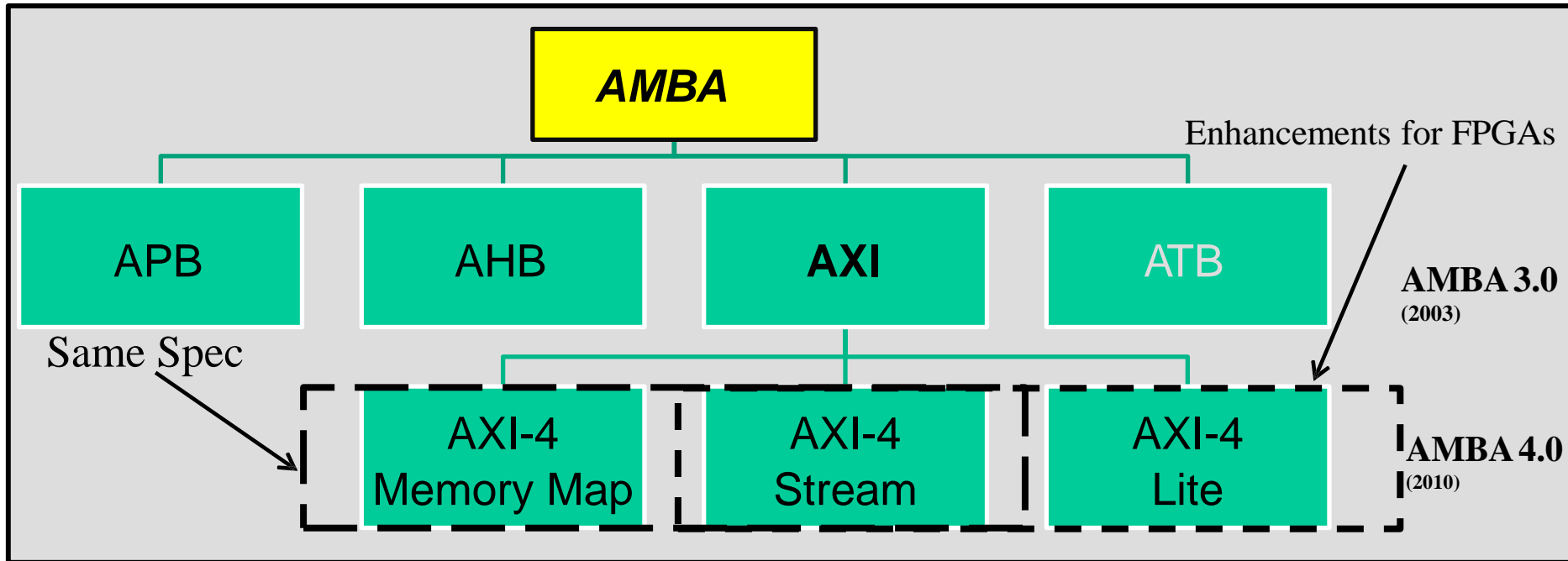
建立IP方法:

- HDL
- MathWorks HDL Coder
- Vivado HLS

ZYNQ的外设与处理器核通过AXI总线连接

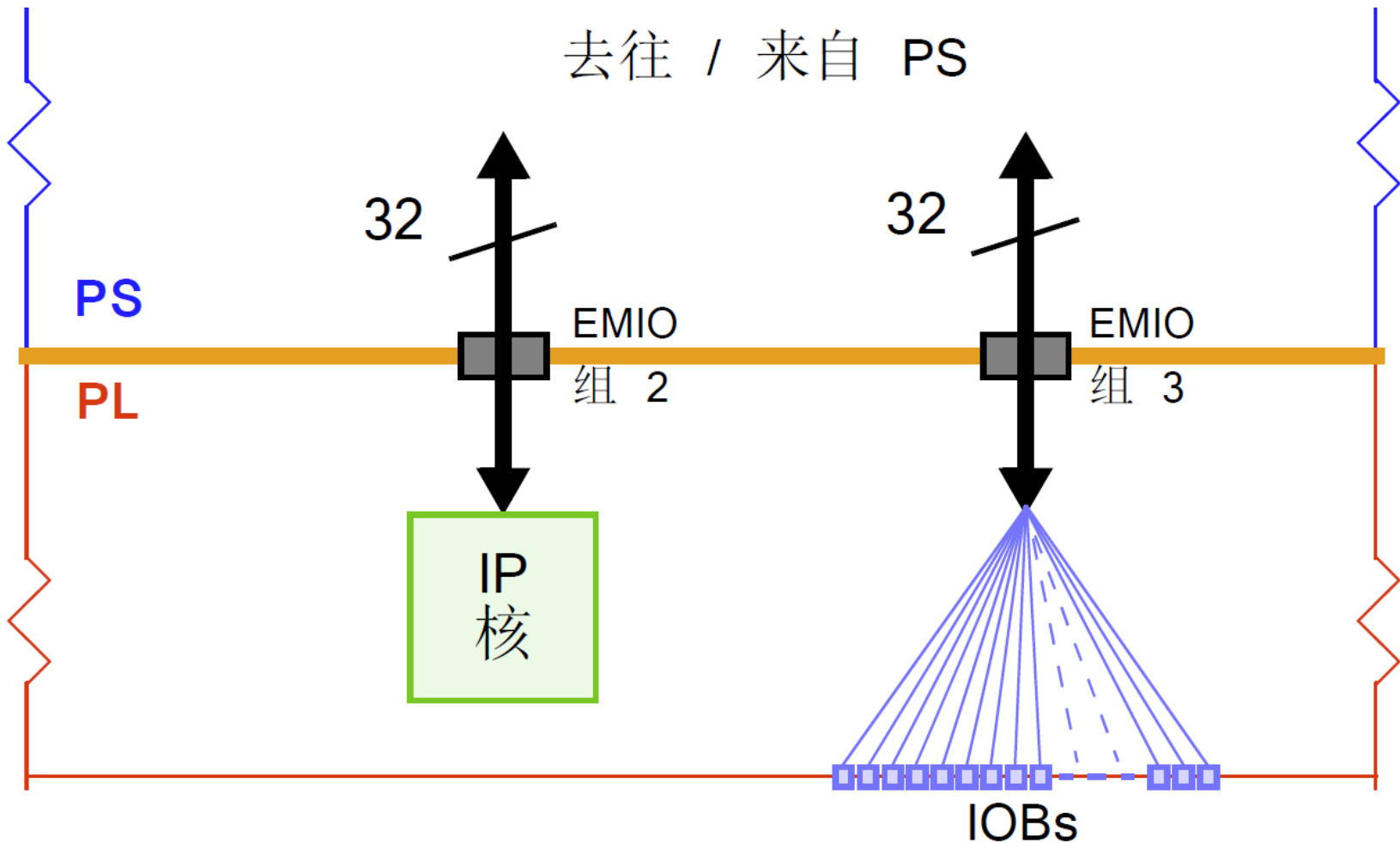


AXI: Advanced eXtensible Interface, 是AMBA协议的一部分 高带宽、低延迟的连接
 AMBA: Advanced Microcontroller Bus Architecture



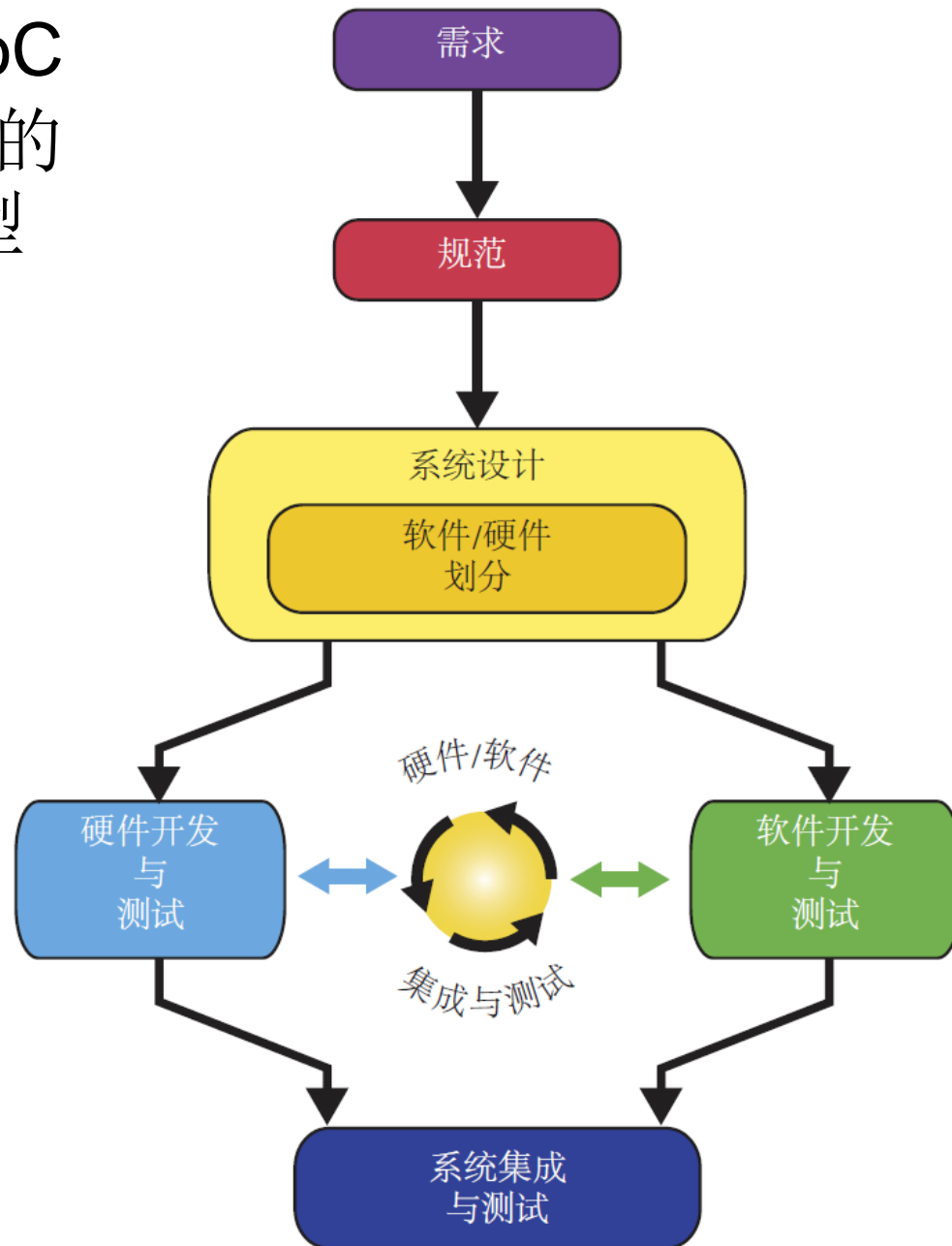
Interface	Features	Similar to
Memory Map / Full (AXI4)	传统地址数据Burst (单个地址, 多个数据)	PLBv46, PCI
Streaming (AXI4-Stream)	仅数据, Burst	Local Link / DSP Interfaces / FIFO / FSL
Lite (AXI4-Lite)	传统的地址/数据 - 没有Burst (单地址, 单数据)	PLBv46-single OPB

MIO和EMIO

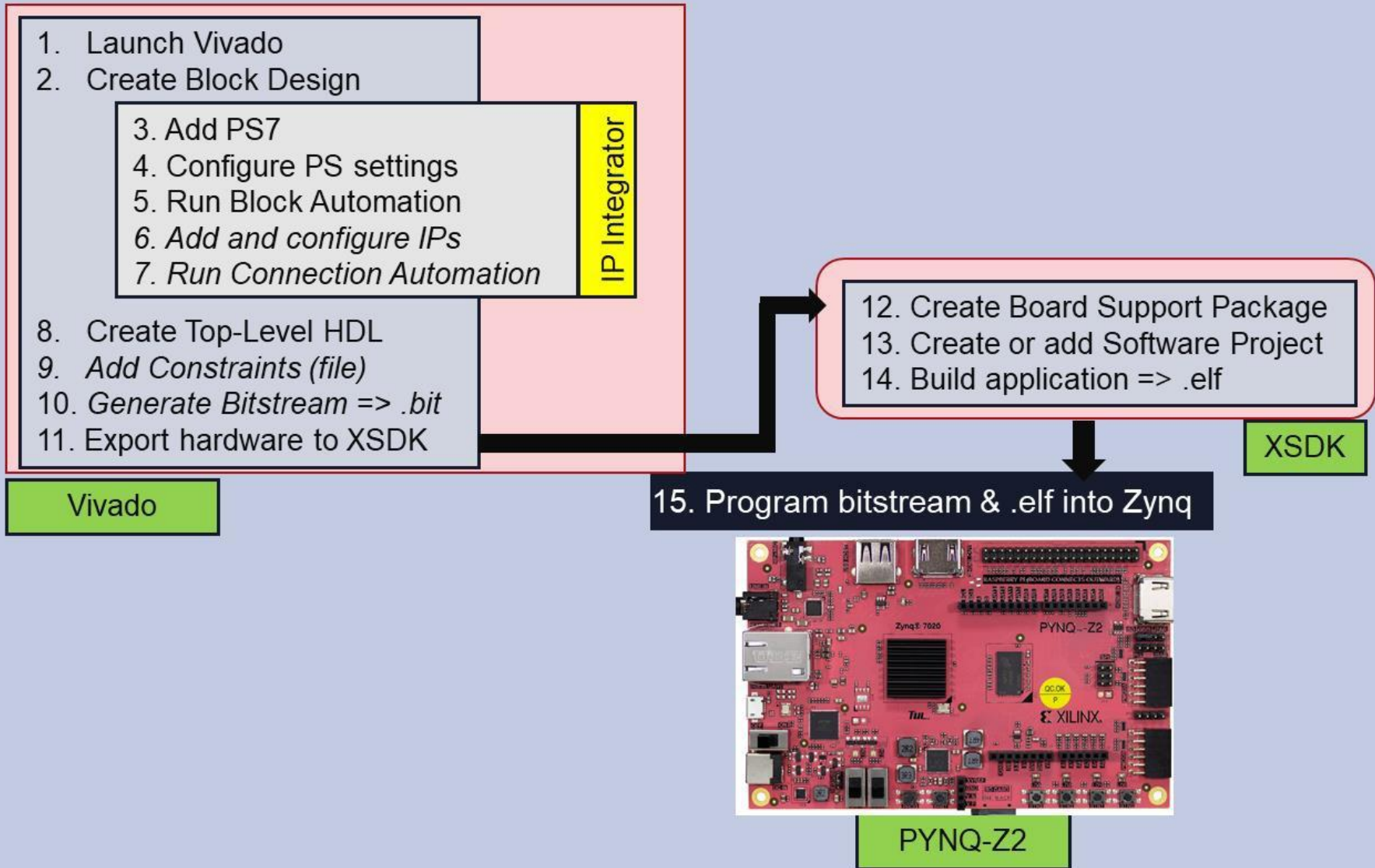


- MIO (Multiplexed IO) ; MIO端口数54, 有限, 可用EMIO扩展
- EMIO (extended MIO) ; 64输入+64输出

ZYNQ SoC 设计流程的 基本模型

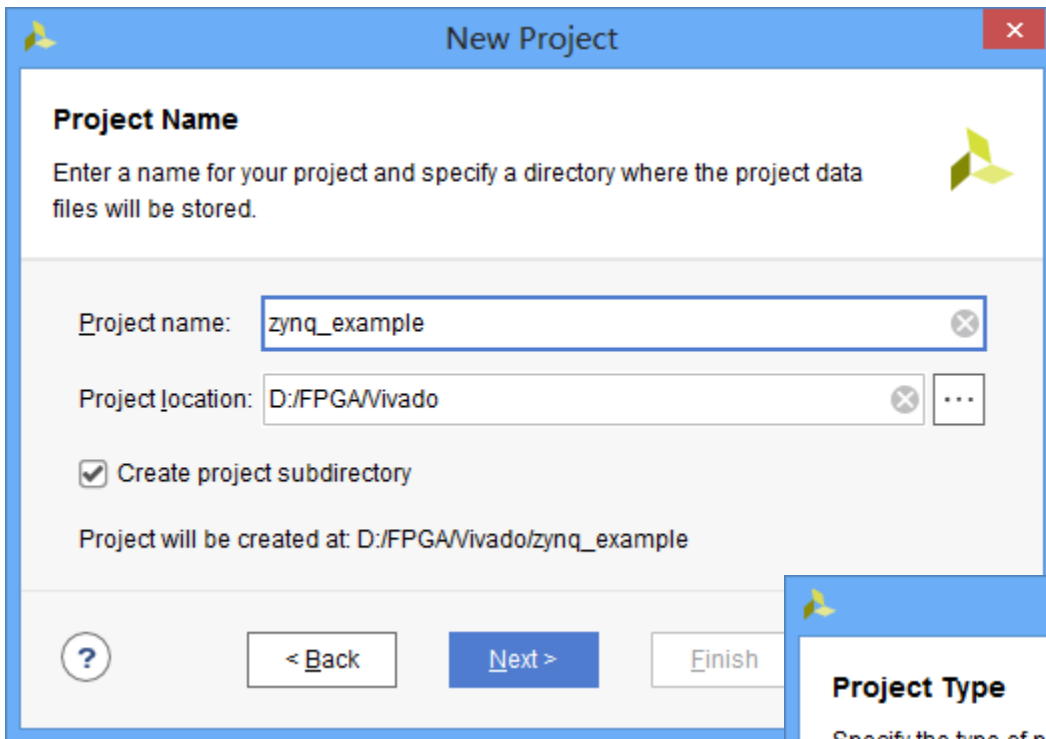


ZYNQ-7000开发流程



2. Vivado/Vitis平台下的操作步骤





New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location: ...

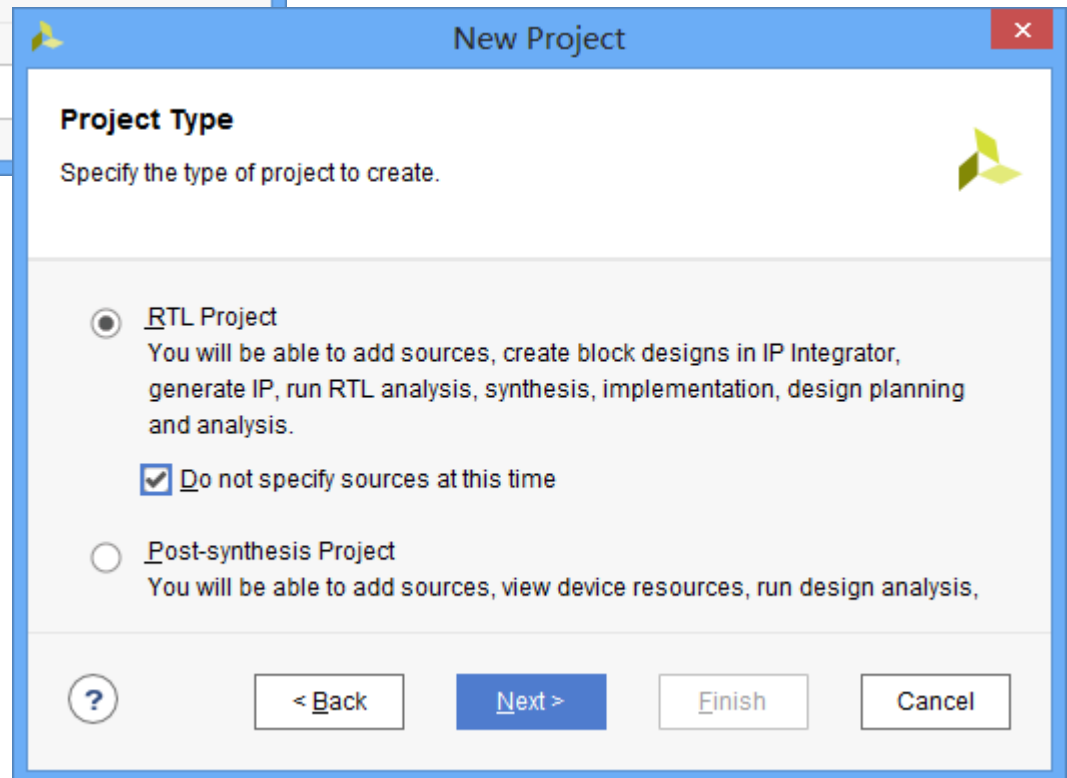
☒ Create project subdirectory

Project will be created at: D:/FPGA/Vivado/zynq_example

? < Back Next > Finish

指定工程名和目录

工程类型：RTL工程。
暂不添加源码。



New Project

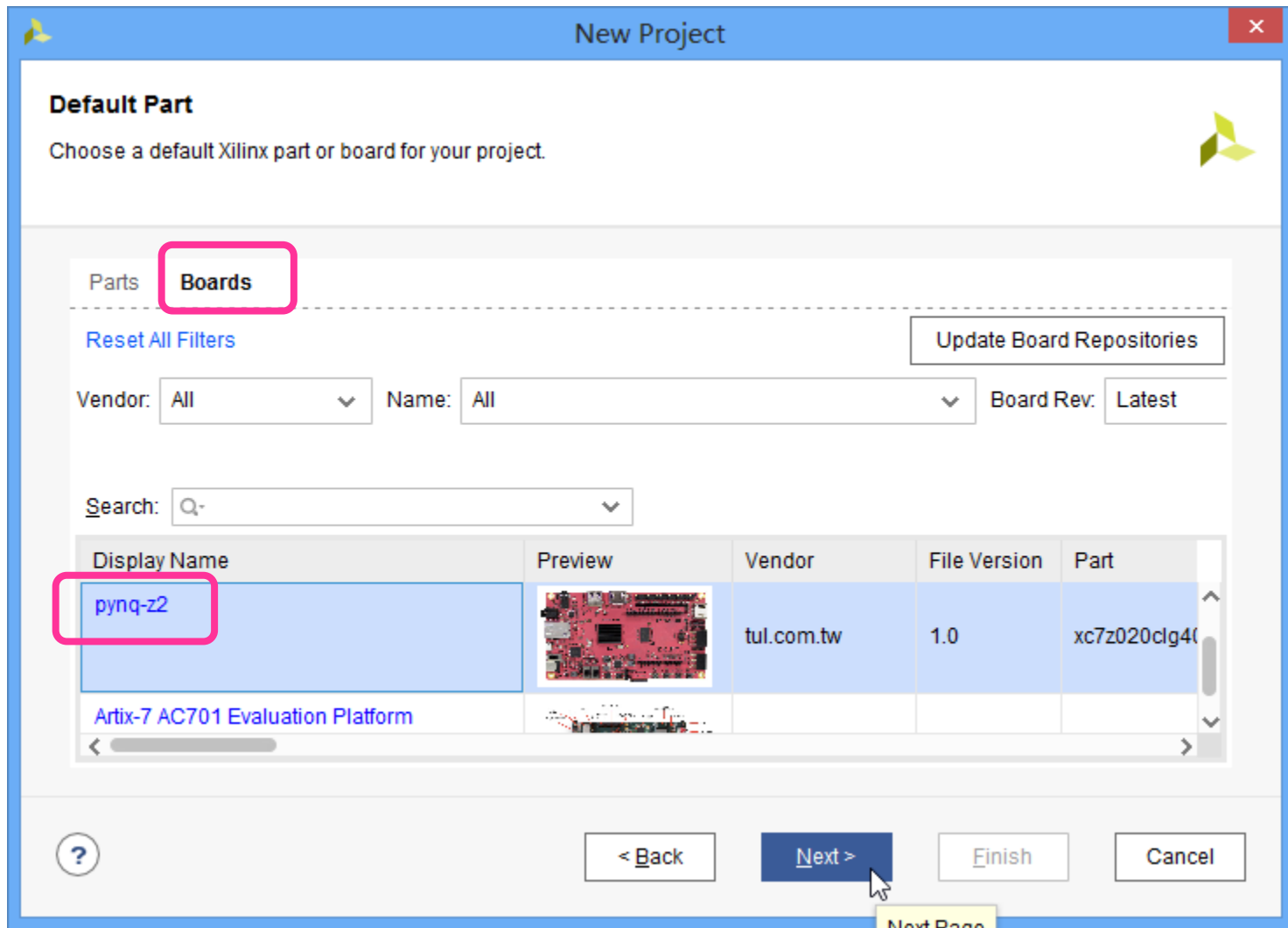
Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☒ Do not specify sources at this time

☐ **Post-synthesis Project**
You will be able to add sources, view device resources, run design analysis,

? < Back Next > Finish Cancel

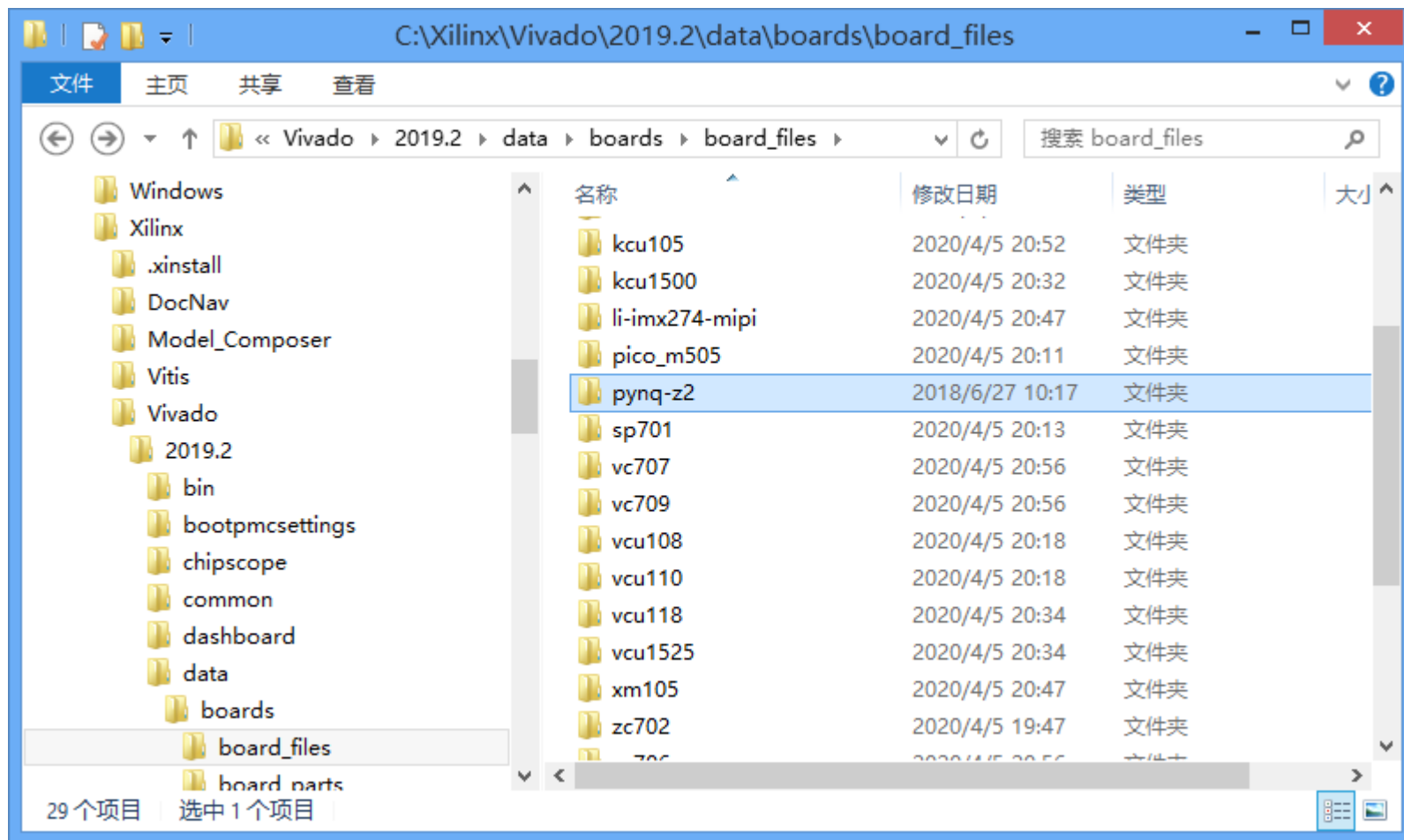
在Boards标签里选择板卡：pnyq-z2



Vivado2019.2里默认没有pnyq板，依下页下载板卡文件手工添加

手工添加板卡文件

先去下载板卡文件 <http://www.tul.com.tw/download/pynq-z2.zip>



解压进vivado目录下 `C:\Xilinx\Vivado\2019.2\data\boards\board_files`
板卡文件很重要，可以自动生成外部端口的约束，包括DDR控制器的配置



工程信息汇总

zynq-example - [D:/FPGA/Vivado/zynq-example/zynq-example.xpr] - Vivado 2019.2.1

File Edit Flow Tools Reports Window Layout View Help Q- Quick Access Ready

Flow Navigator

- PROJECT MANAGER
 - Settings
 - Add Sources
 - Language Templates
 - IP Catalog
- IP INTEGRATOR
 - Create Block Design**
 - Open Block Design
 - Generate Block Design
- SIMULATION
 - Run Simulation
- RTL ANALYSIS
 - Open Elaborated Design
- SYNTHESIS
 - Run Synthesis
 - Open Synthesized Design
- IMPLEMENTATION

创建并添加一个IP子系统

PROJECT MANAGER - zynq-example

Sources

- Design Sources
- Constraints
- Simulation Sources

Hierarchy Libraries Compile Order

Select an object to see properties

Project Summary

Overview | Dashboard

Settings Edit

Project name: zynq-example

Project location: D:/FPGA/Vivado/zynq-example

Product family: Zynq-7000

Project part: pynq-z2 (xc7z020clg400-1)

Top module name: Not defined

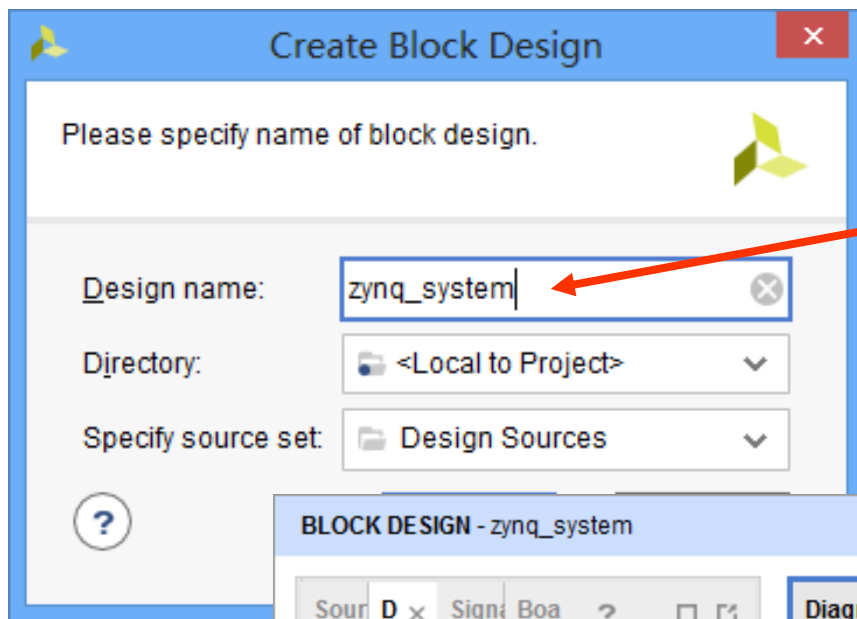
Target language: Verilog

Simulator language: Mixed

Tcl Console Messages Log Reports Design Runs

Name	Constraints	Status	WNS	TNS	WHIS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1	constrs_1	Not started									
impl_1	constrs_1	Not started									

建立Block Design

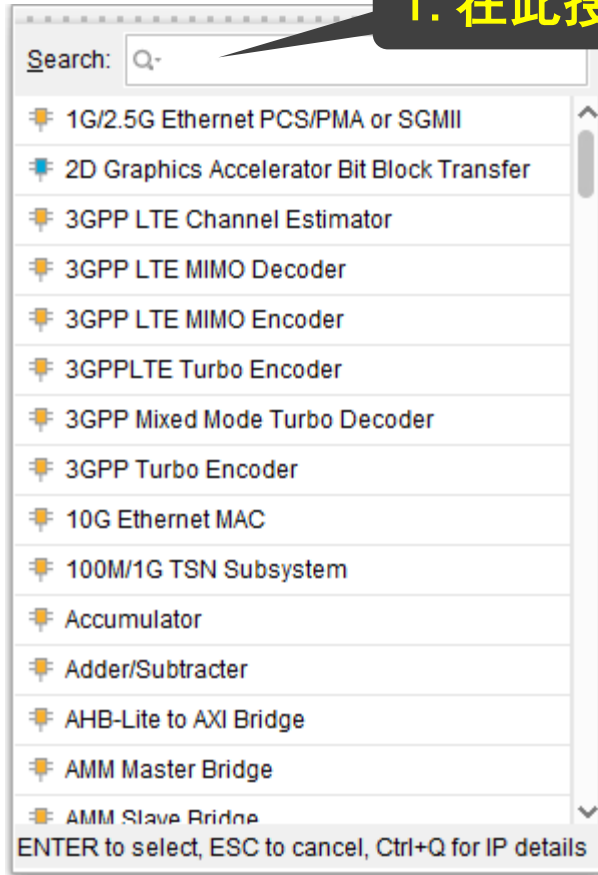


给block design命名

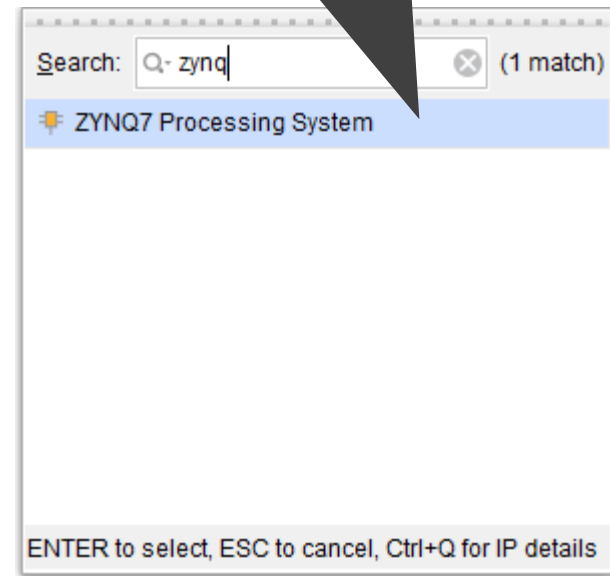


添加ZYNQ7 PS IP

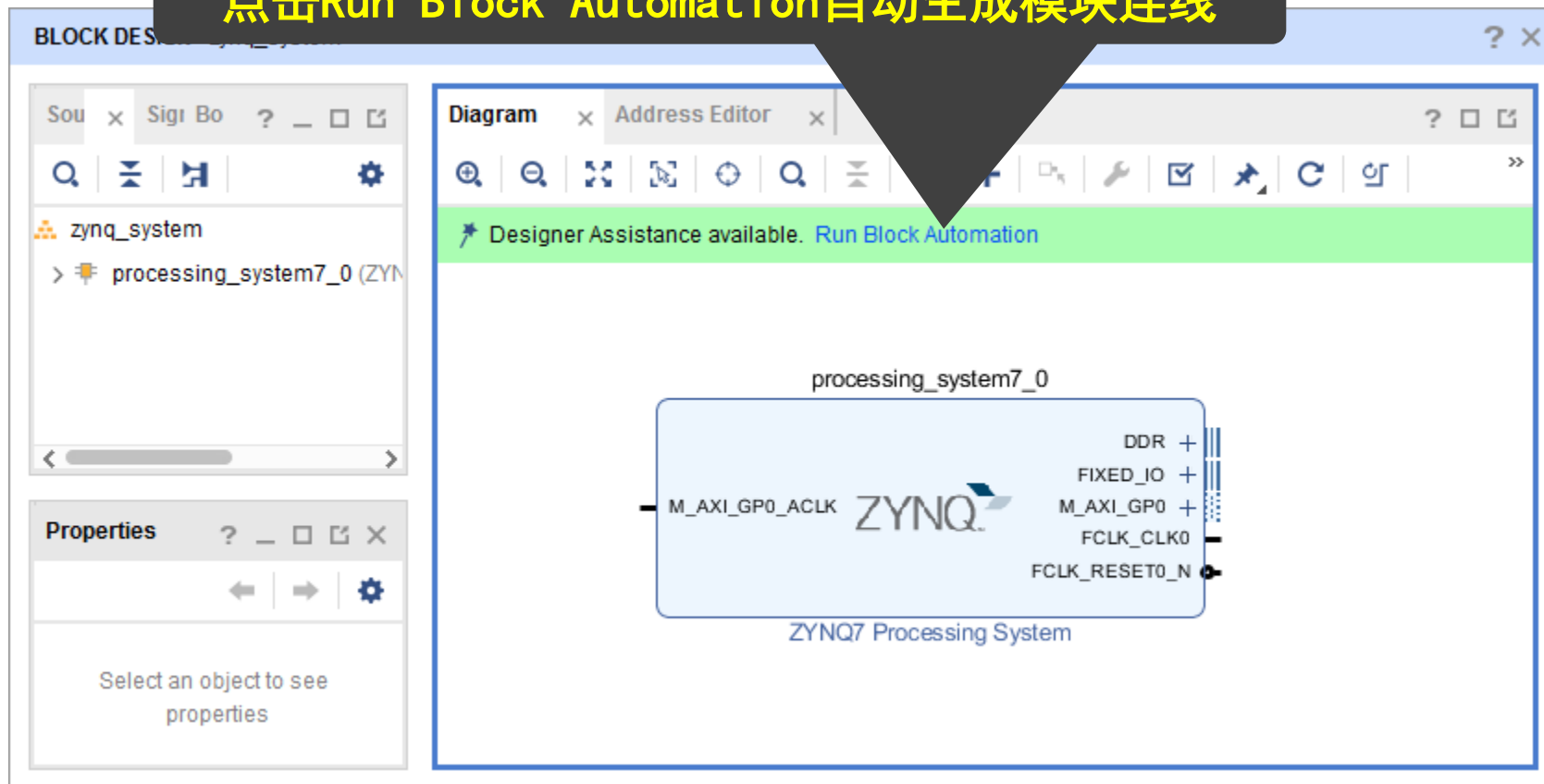
1. 在此搜索zynq



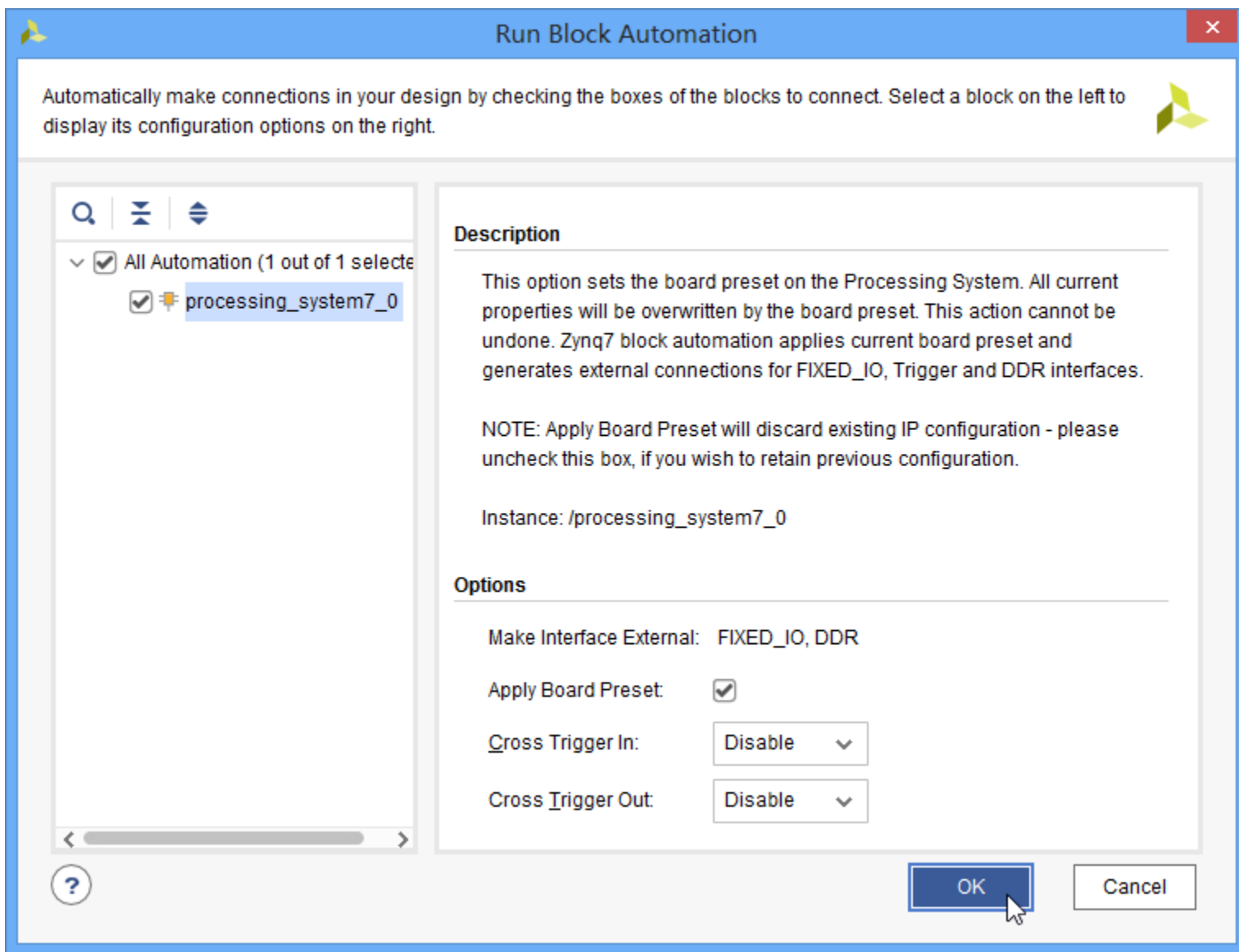
2. 双击添加ZYNQ7 IP



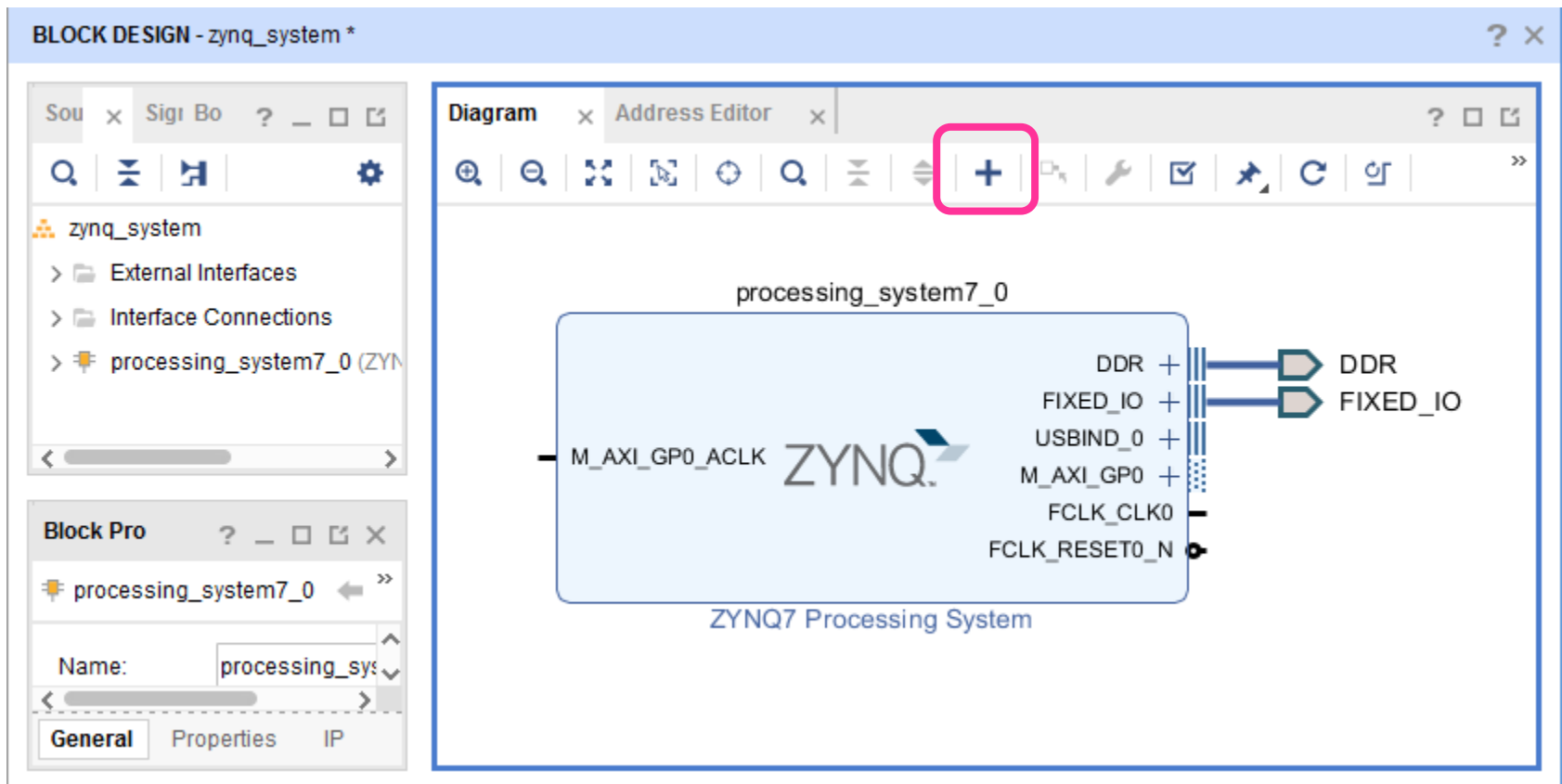
点击Run Block Automation自动生成模块连线



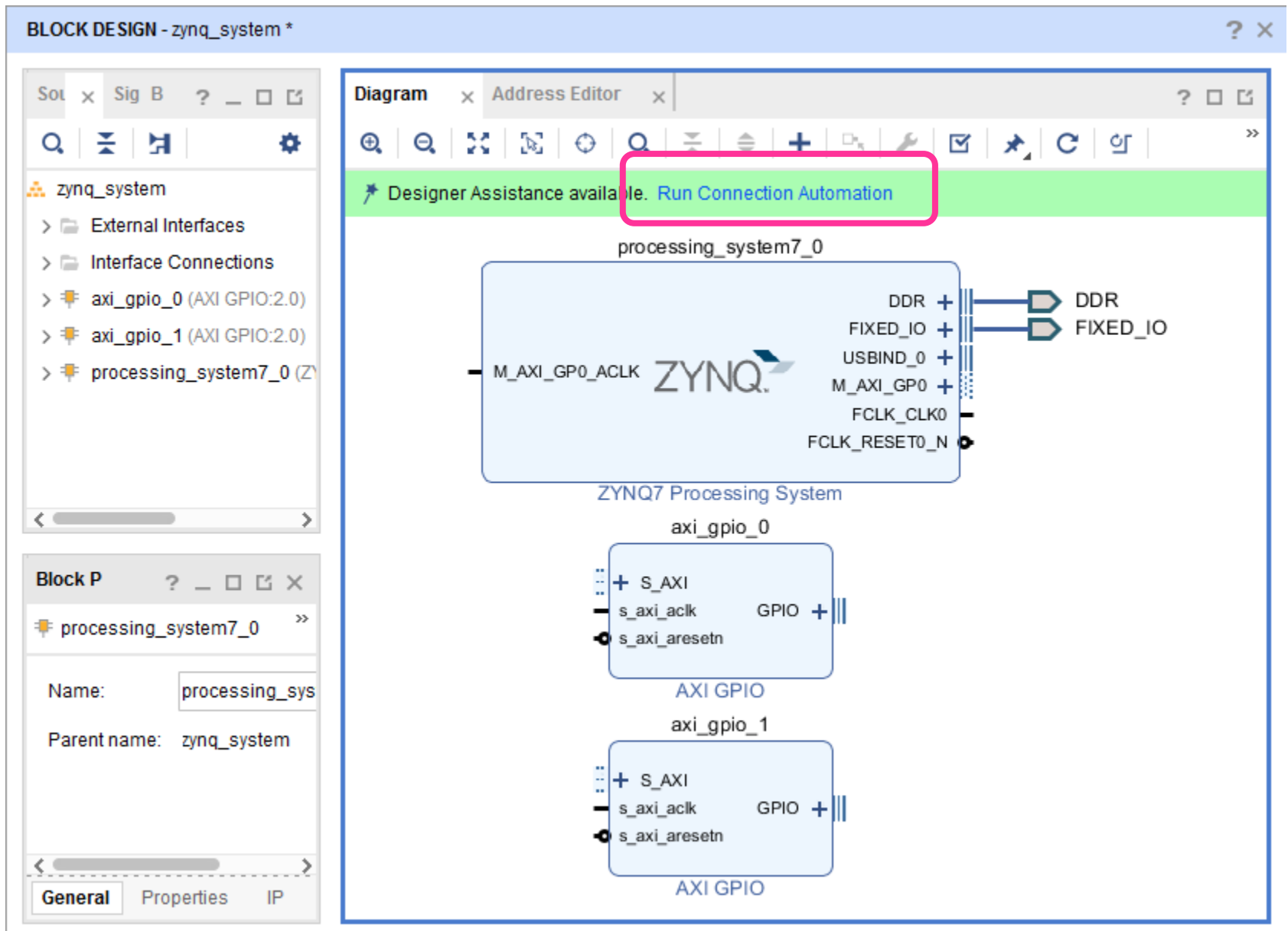
- Block Automation完成模块配置和外部连接生成



ZYNQ-7000模块用默认设置即可

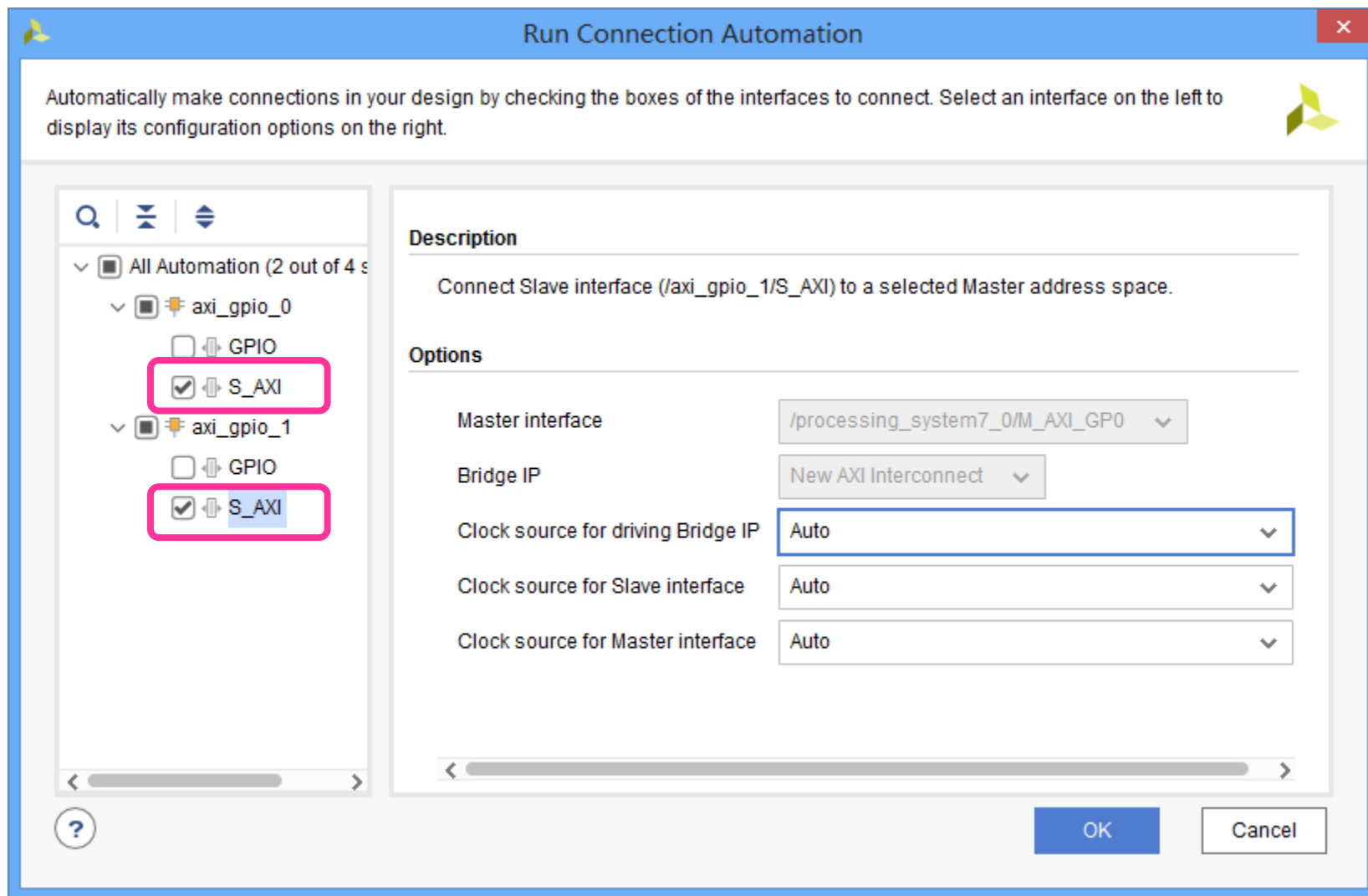


- 自动添加了DDR存储器和FIXED_IO的外部接口
- 点+号继续搜索GPIO添加2个AXI GPIO接口，用来连接LED和Button

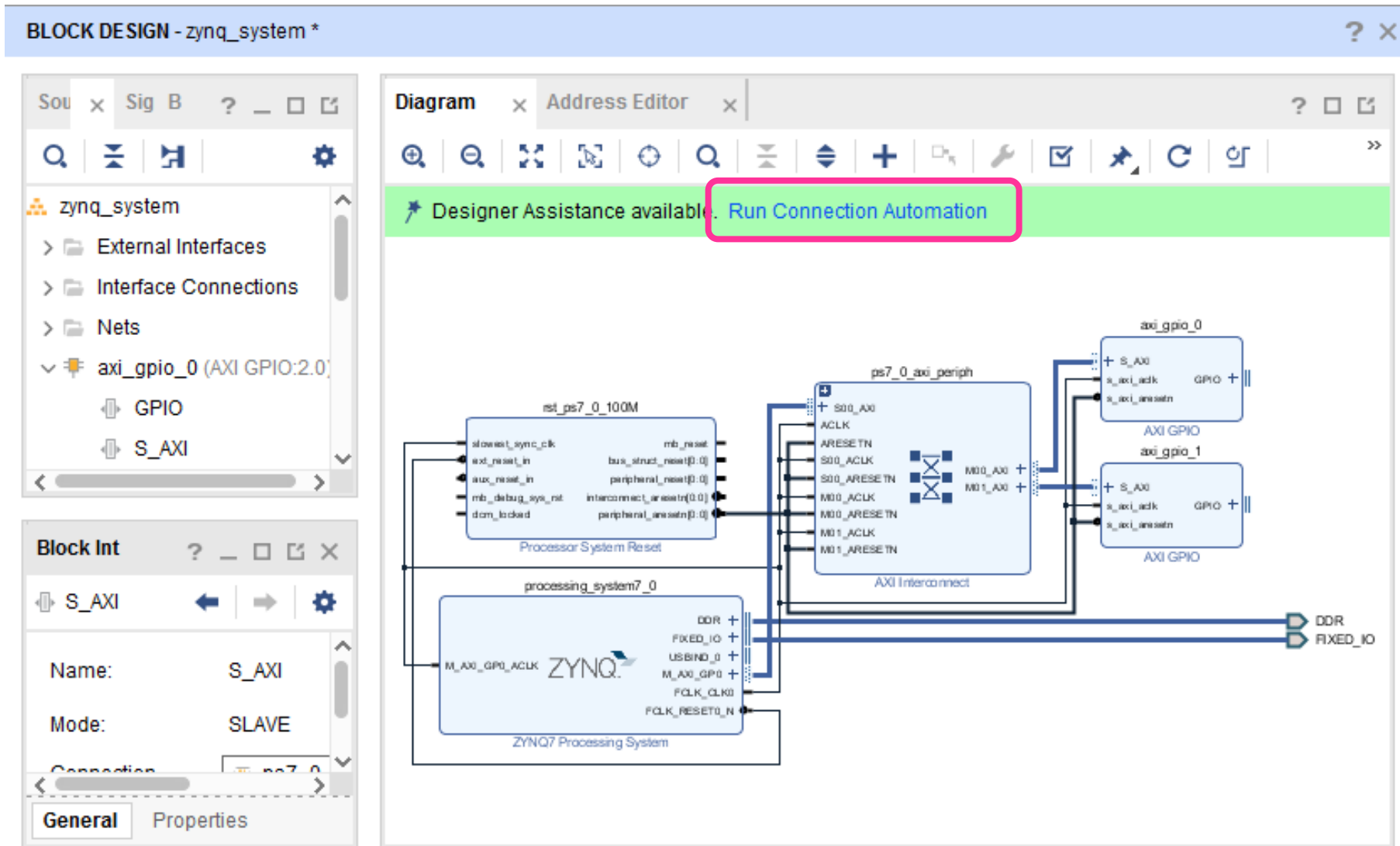


点Run Connection Automation，自动建立总线连接、中间模块添加和外部IO端口连接

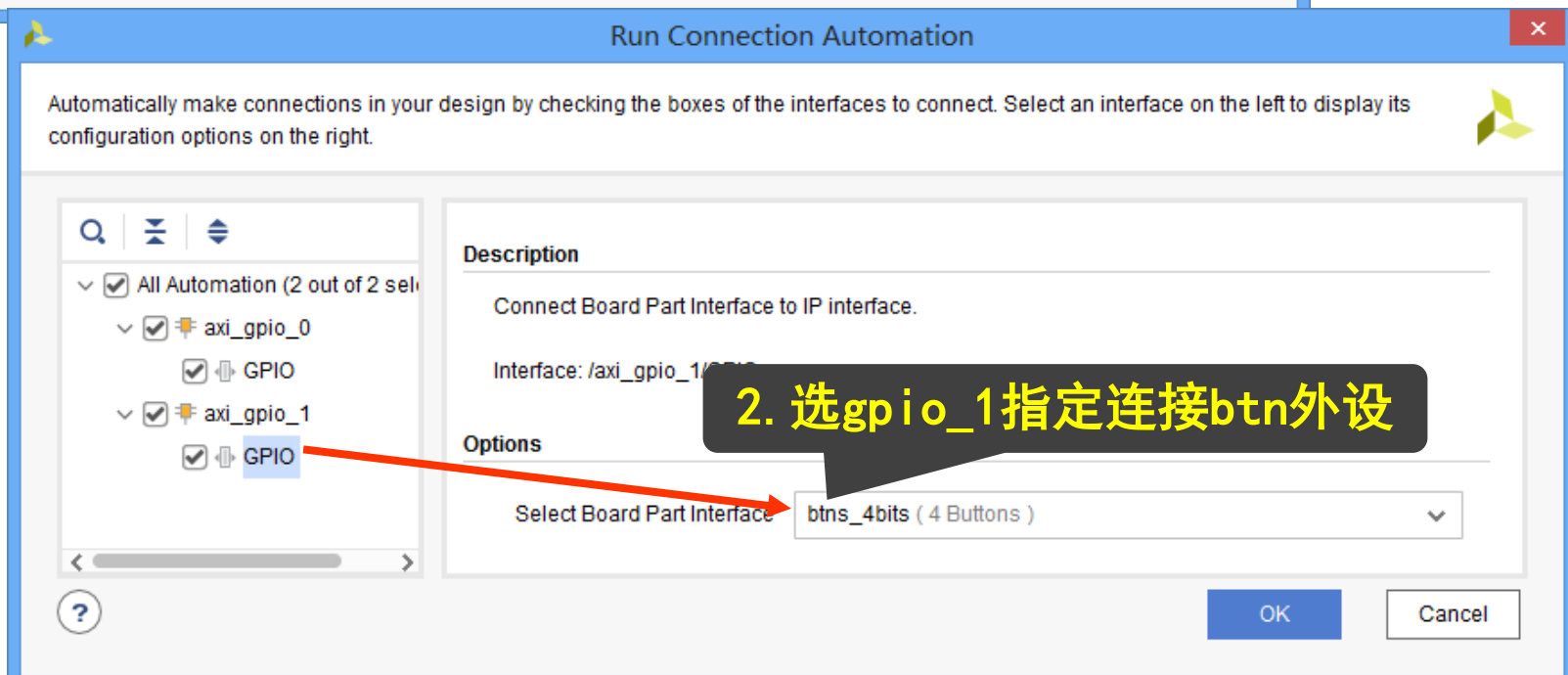
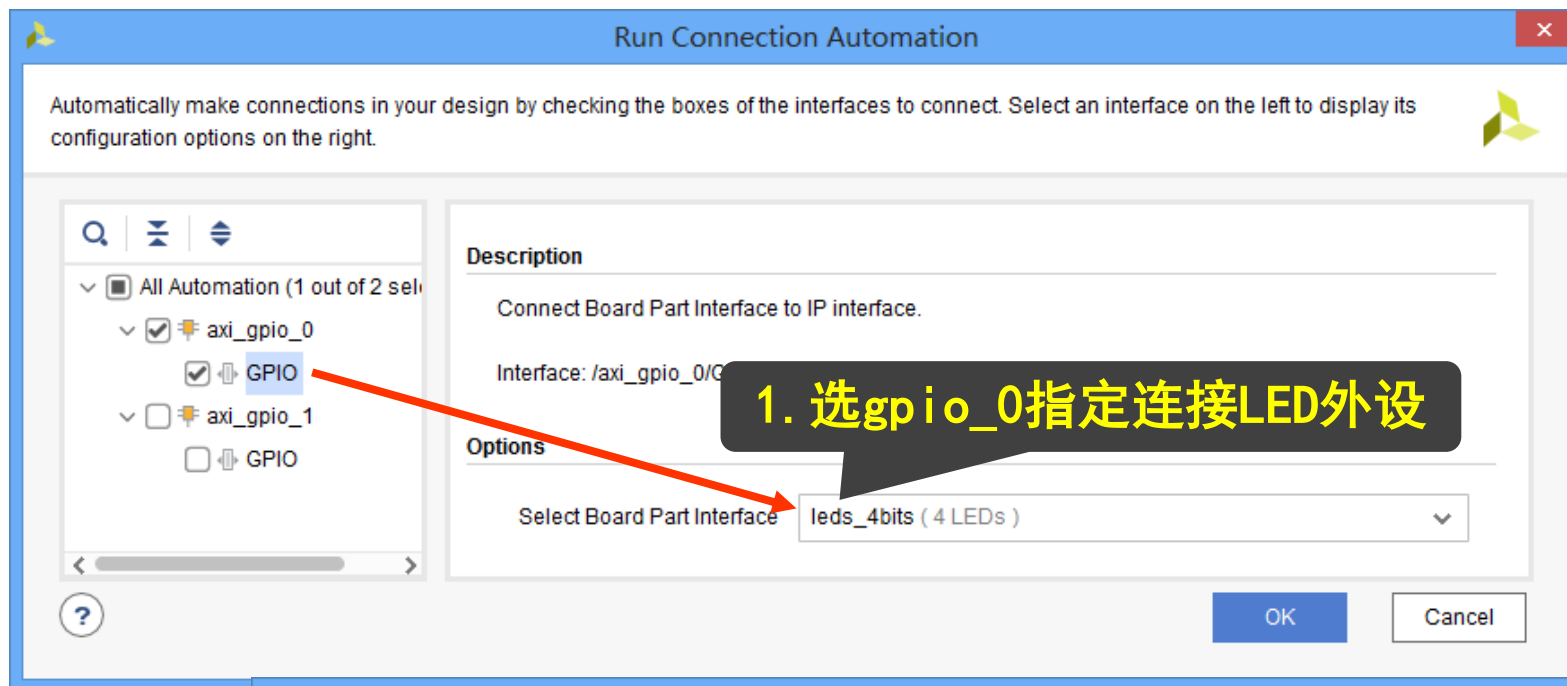
先选S_AXI连接总线接口

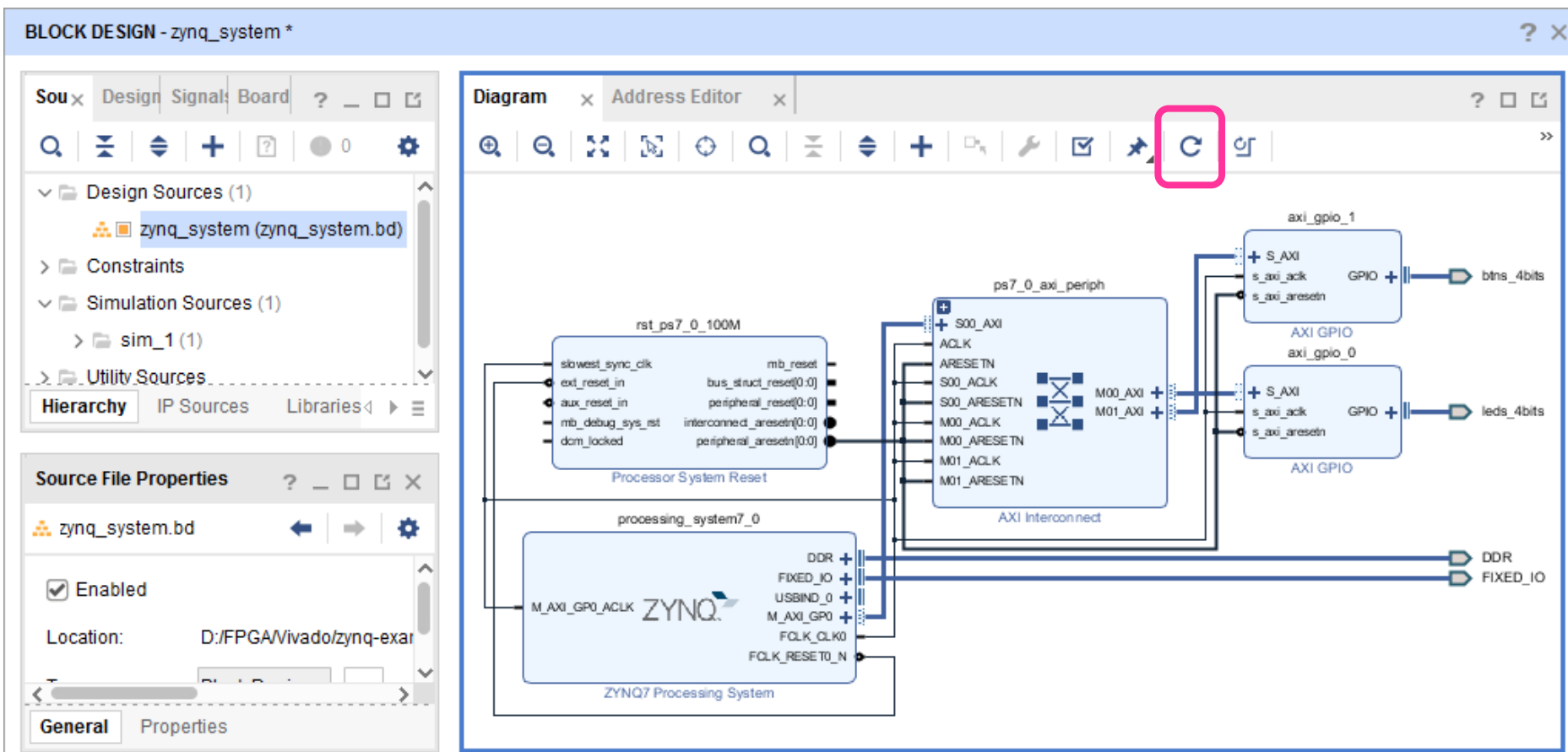


上一步自动添加了AXI从设备与PS相连需要的辅助模块



再点Run Connection Automation，建立外部IO端口连接





- 完成连接的电路图。布局可能不同，连接应该是相同的，按红框按钮可以重绘电路图

BLOCK DESIGN - zynq_system

Sources | **Design** | Signals | Board

zynq_system

- External Interfaces
- Interface Connections
- Nets
- axi_gpio_0 (AXI GPIO:2.0)
- axi_gpio_1 (AXI GPIO:2.0)
- processing_system7_0 (ZYNQ7 Processing Sys)

Source File Properties

zynq_system.bd

☒ Enabled

Location: D:/FPGA/Vivado/zynq-example/zynq

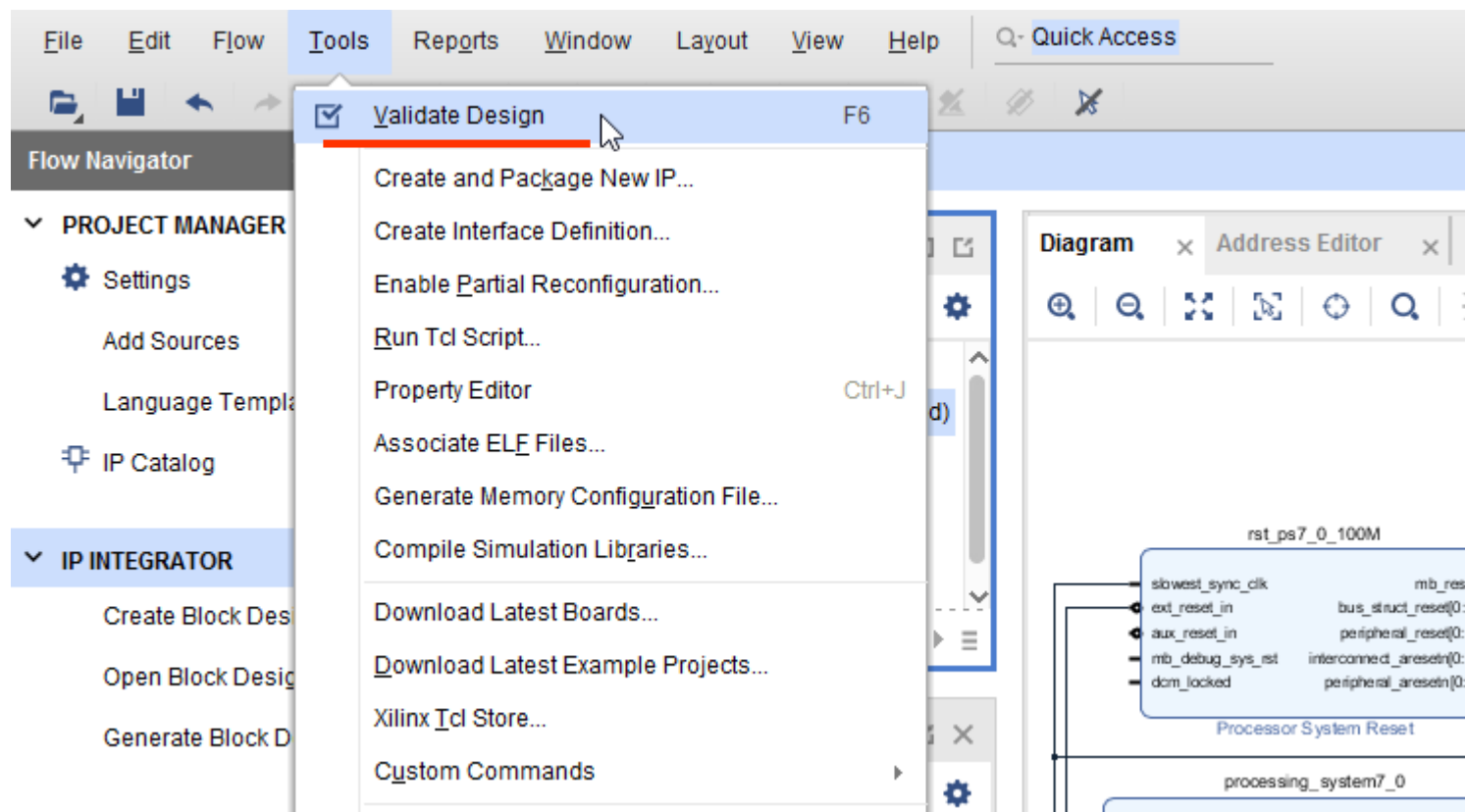
General | Properties

Diagram | Address Editor

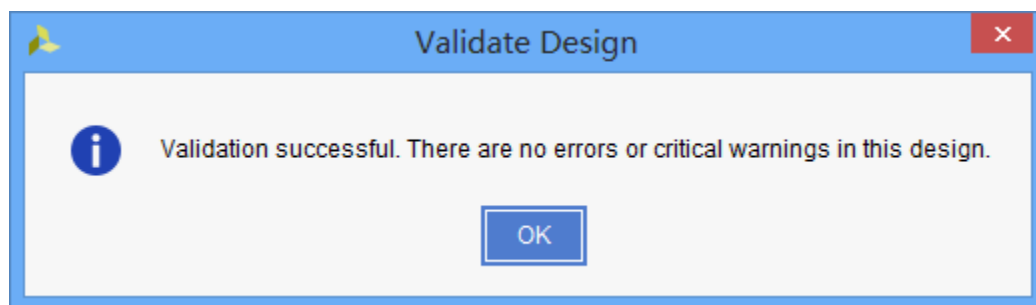
Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
Data (32 address bits : 0x40000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

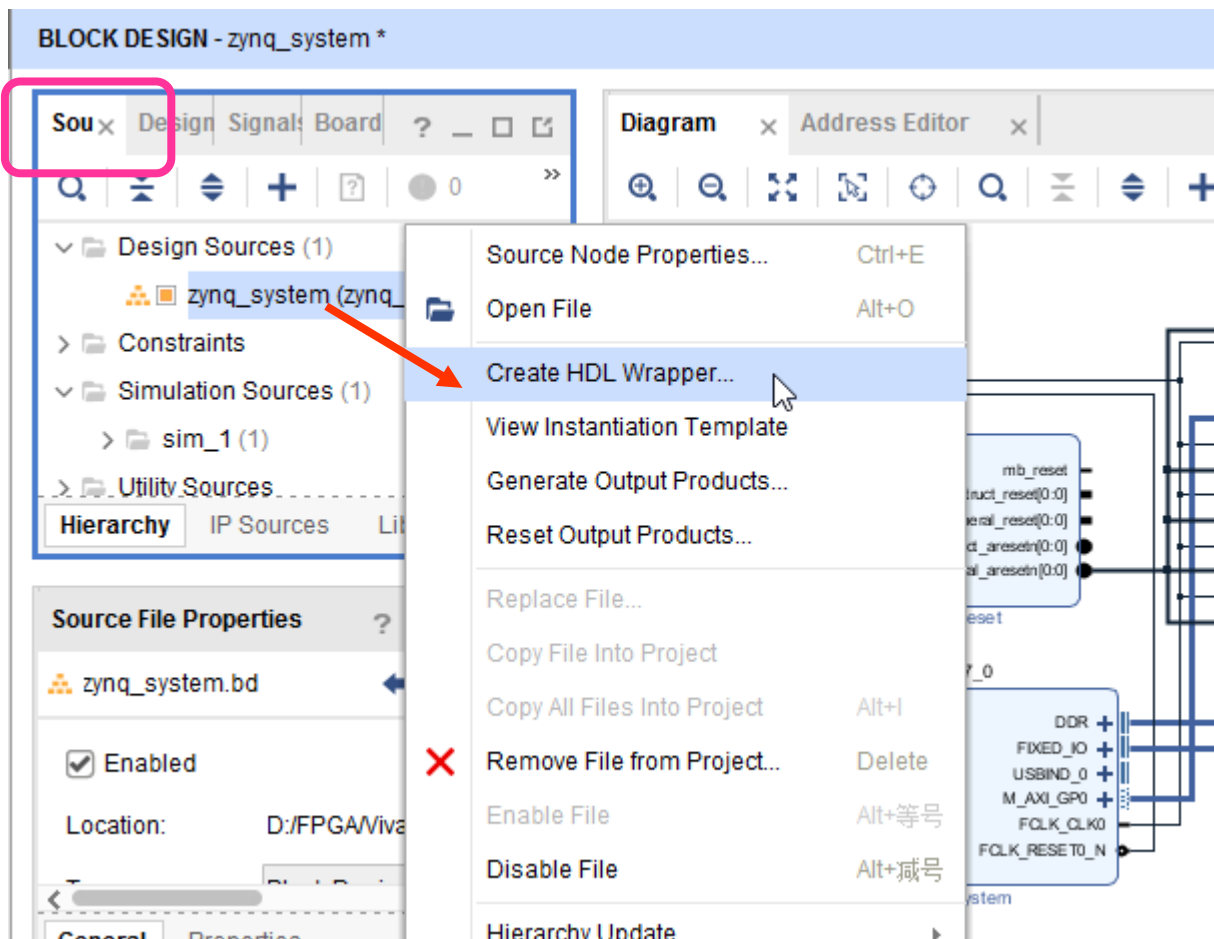
这是vitis里会用到的外设基地址BASEADDR

Vivado给添加的外设自动分配了地址

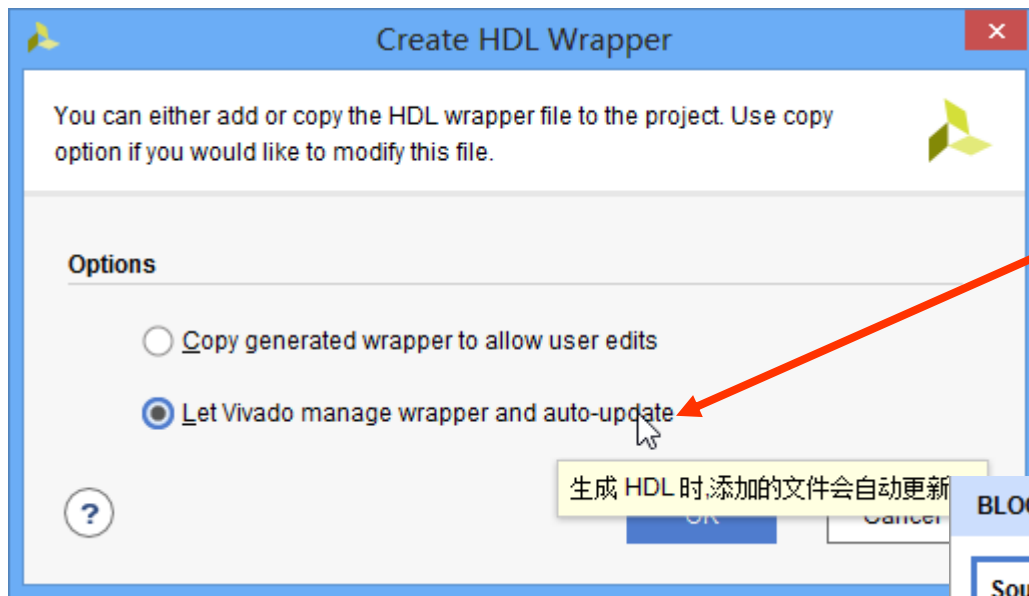


点击主菜单Tools -> Validate Design，验证设计的正确性。若无问题显示右侧信息框。





切换到Source标签，鼠标右键点击Design Source下面的块图文件名zynq_system，在右键菜单里选Create HDL Wrapper，封装顶层Verilog HDL文件。

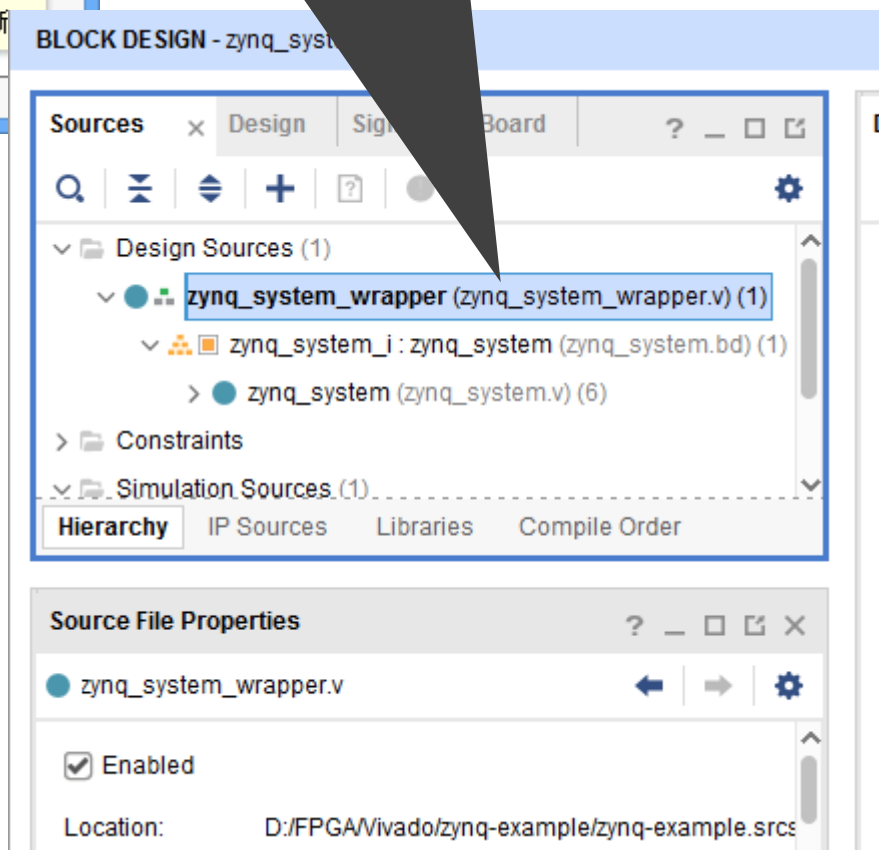


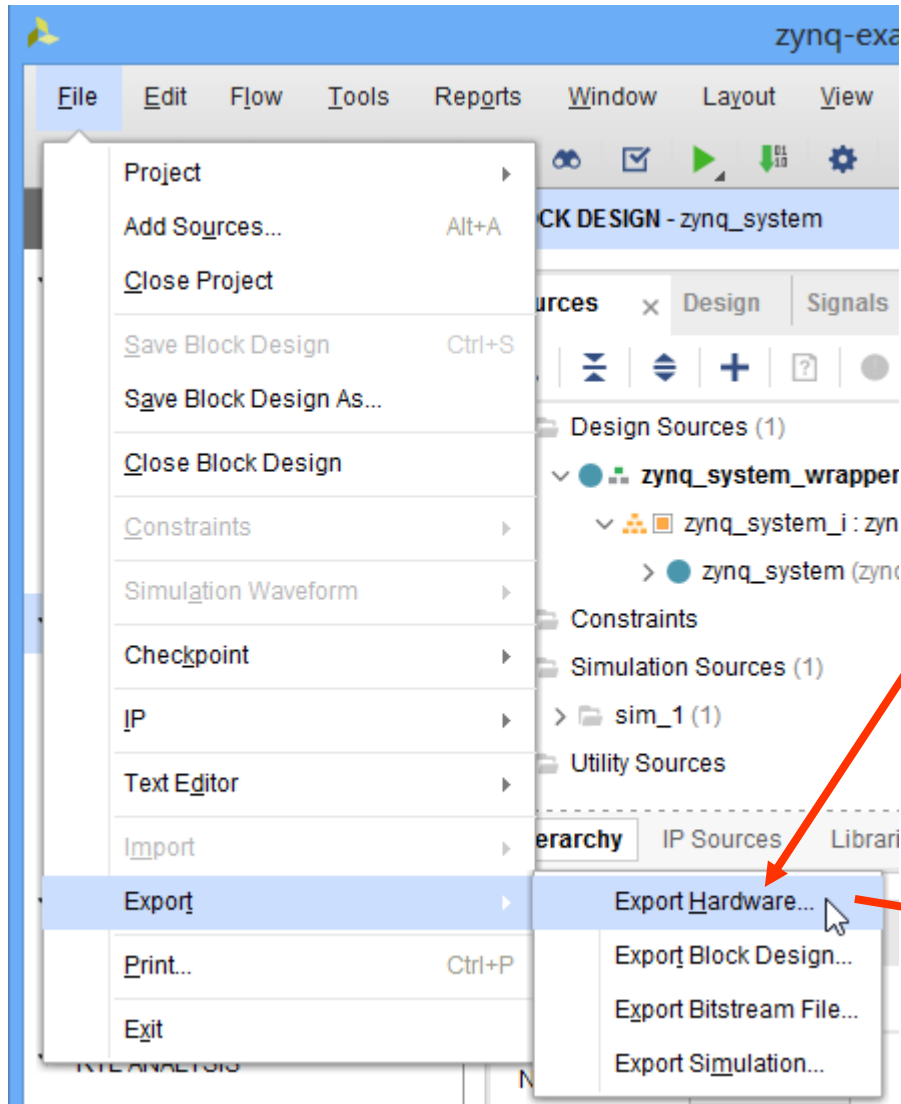
1. 在弹出的对话框里选择让Vivado管理封装文件、自动更新

封装后的顶层v文件

2. 设计完成，去综合、实现：
到左侧Flow Navigator里选最后的
Generate Bitstream，将自动完成前
面的Synthesis和Implementation。

如果因错误中止，也可能不是真的有设计错误，只是计算机内存不够用。如果计算机只有8GB内存，可能需要关掉其它程序才能顺利运行完流程。

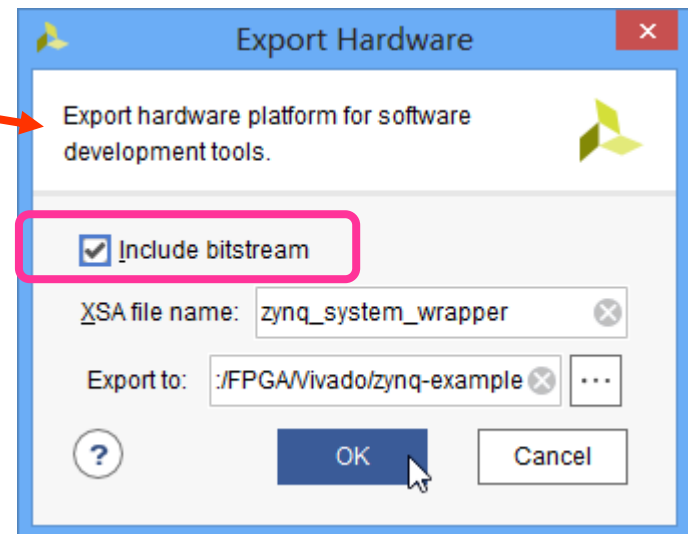




1. 点击系统菜单File -> Export -> Export Hardware...

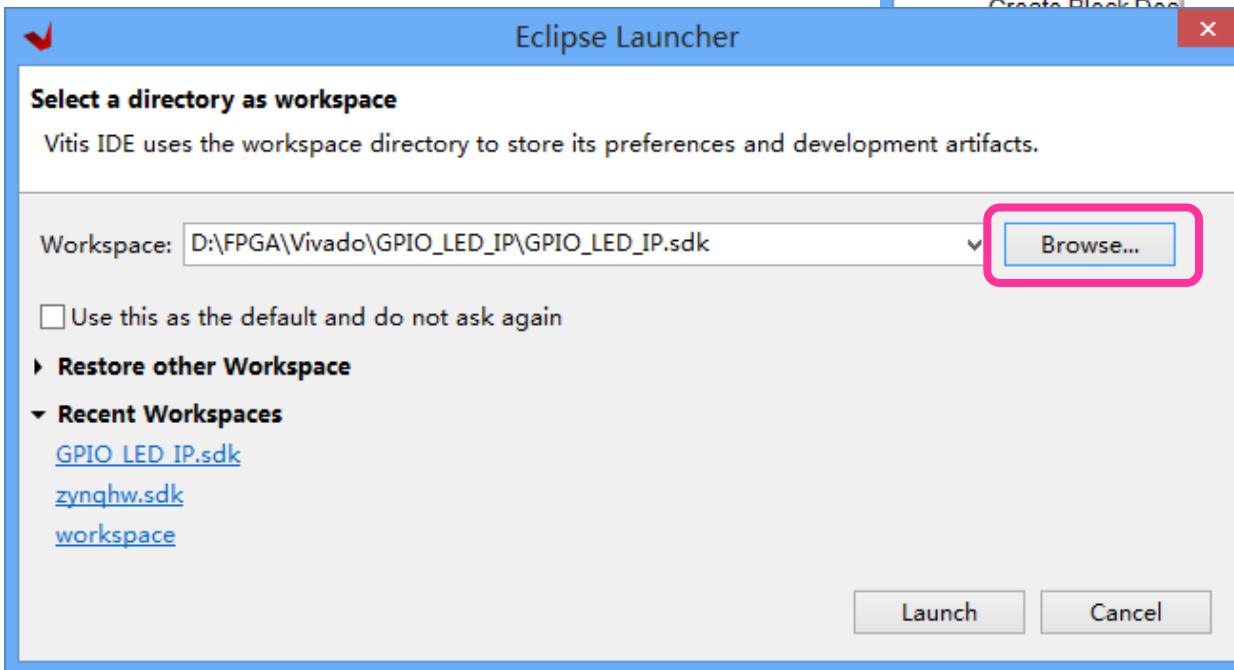
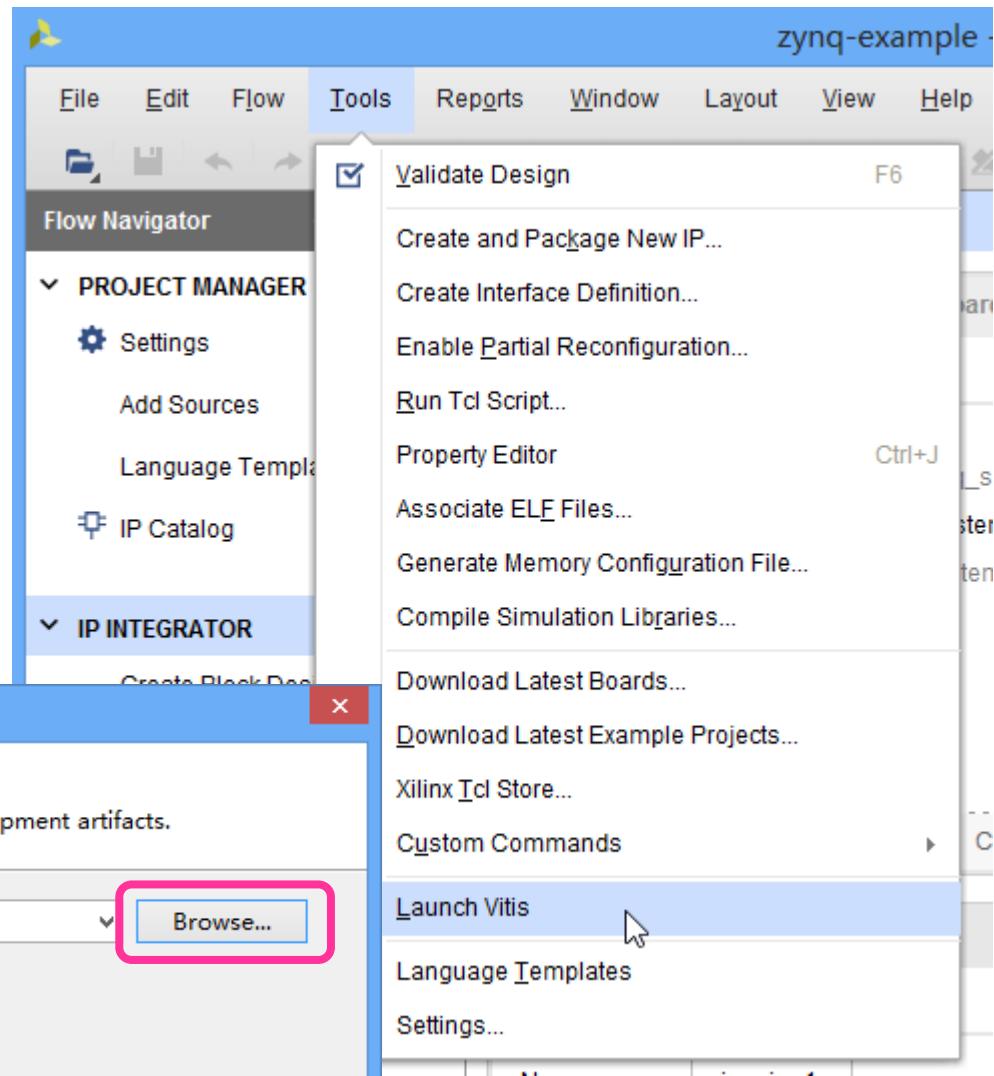
导出xsa文件到本工程的目录

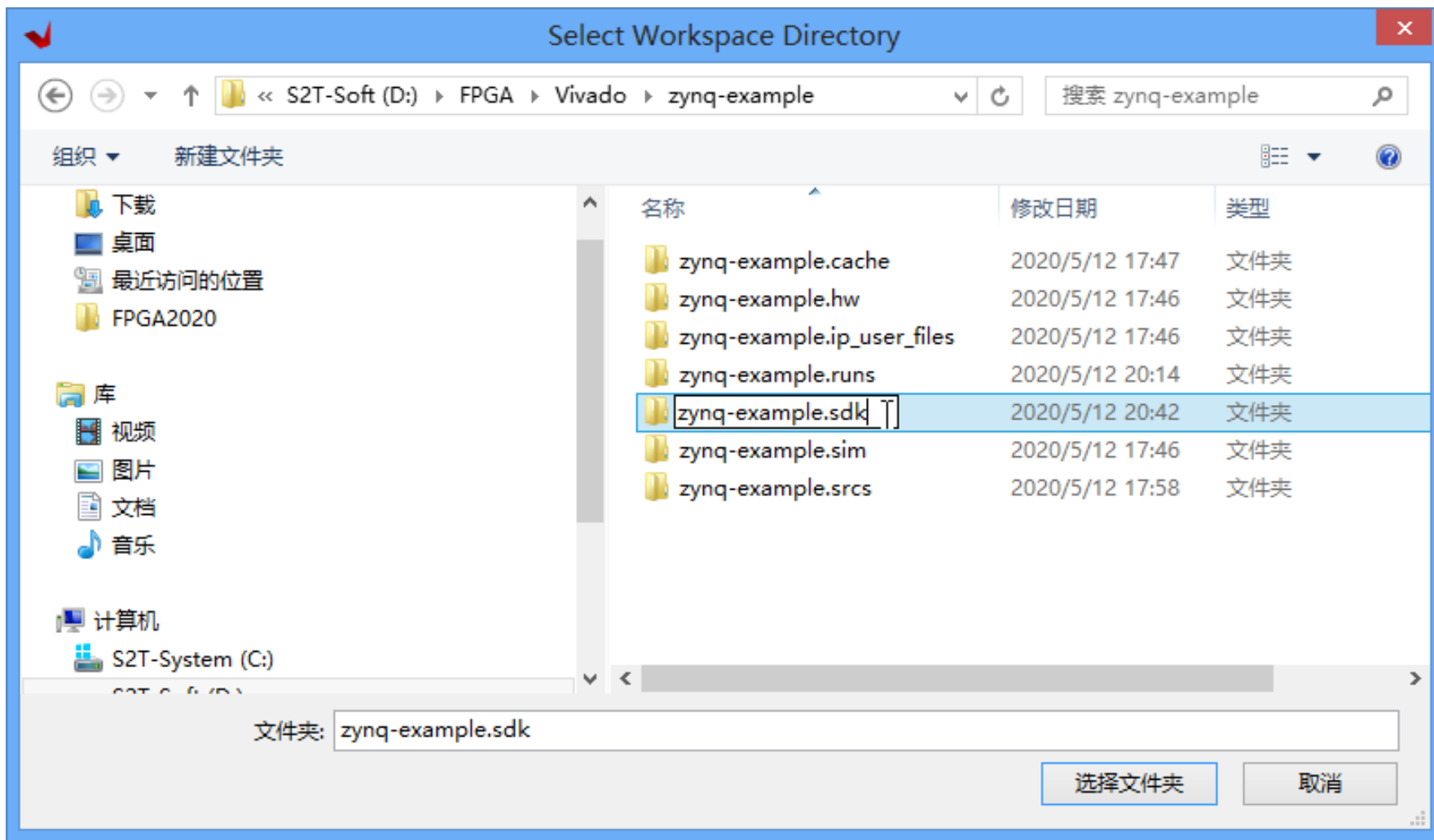
2. Export Hardware对话框里确保 Include Bitstream选项是选中的



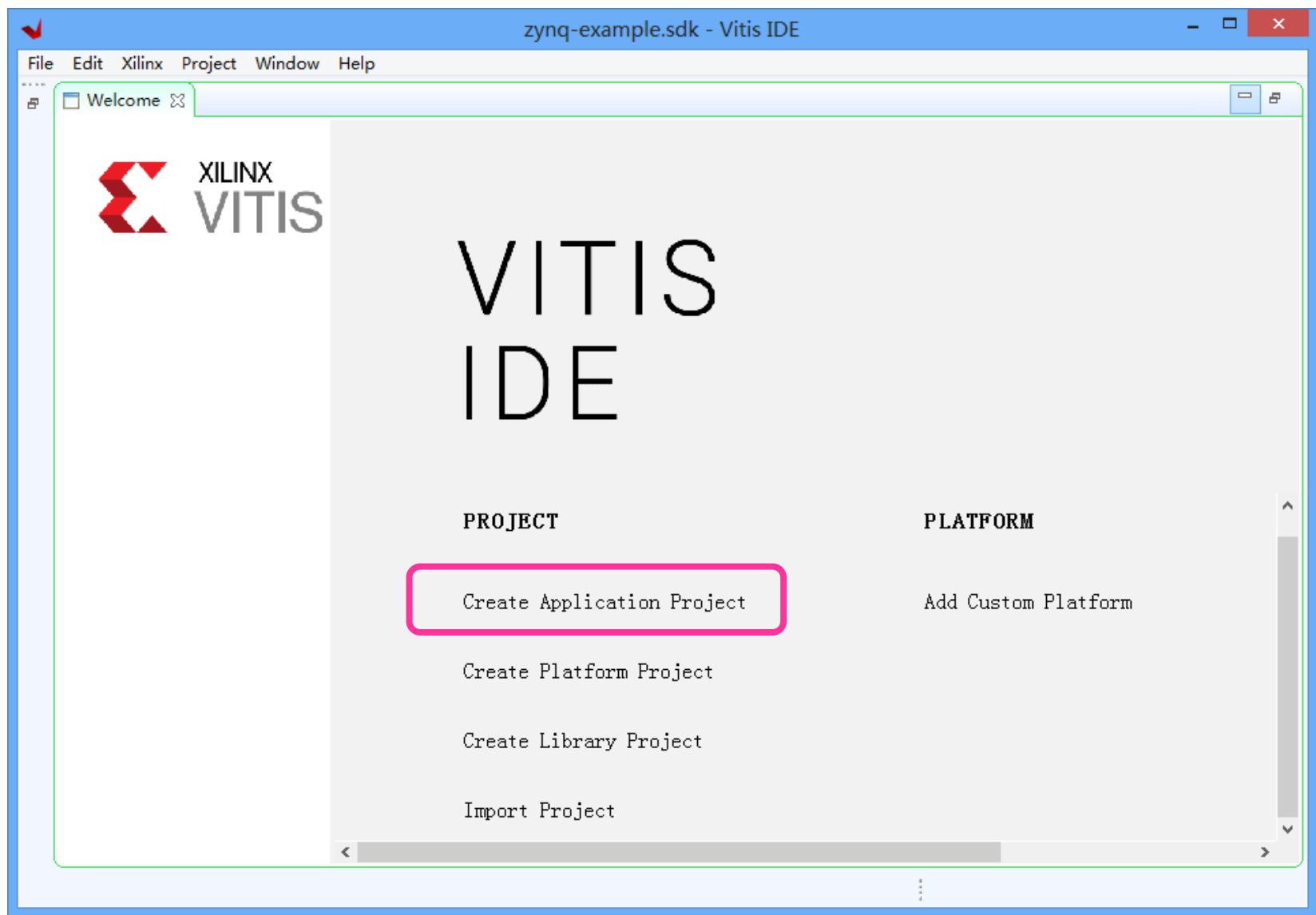
以上vivado里的硬件组装工作完成。之后去vitis里完成软件应用程序开发工作。

去主菜单Tools -> Launch Vitis里启动Vitis。弹出的对话框里是上次用过的工程所在目录，要点击Browse按钮，选择本工程的软件工作目录





- 注意把不同硬件工程的软件工作目录分开，以免Vitis里的工程混乱。比如在各自己的硬件工程目录里建sdk目录。本例在本硬件工程的目录里建立zynq_example.sdk目录，选中或进入它，点击“选择文件夹”。



之后就来到了Vitis IDE界面里，选Create Application Project

New Application Project

Create a New Application Project

Enter a name for your Application project.

Project name:

☒ Use default location

Location:

Choose file system:

System project:

System Project 1

Baremetal App

System Project 2

Linux App 1

Linux App 2

AI Engine App

A72_0 Baremetal Domain

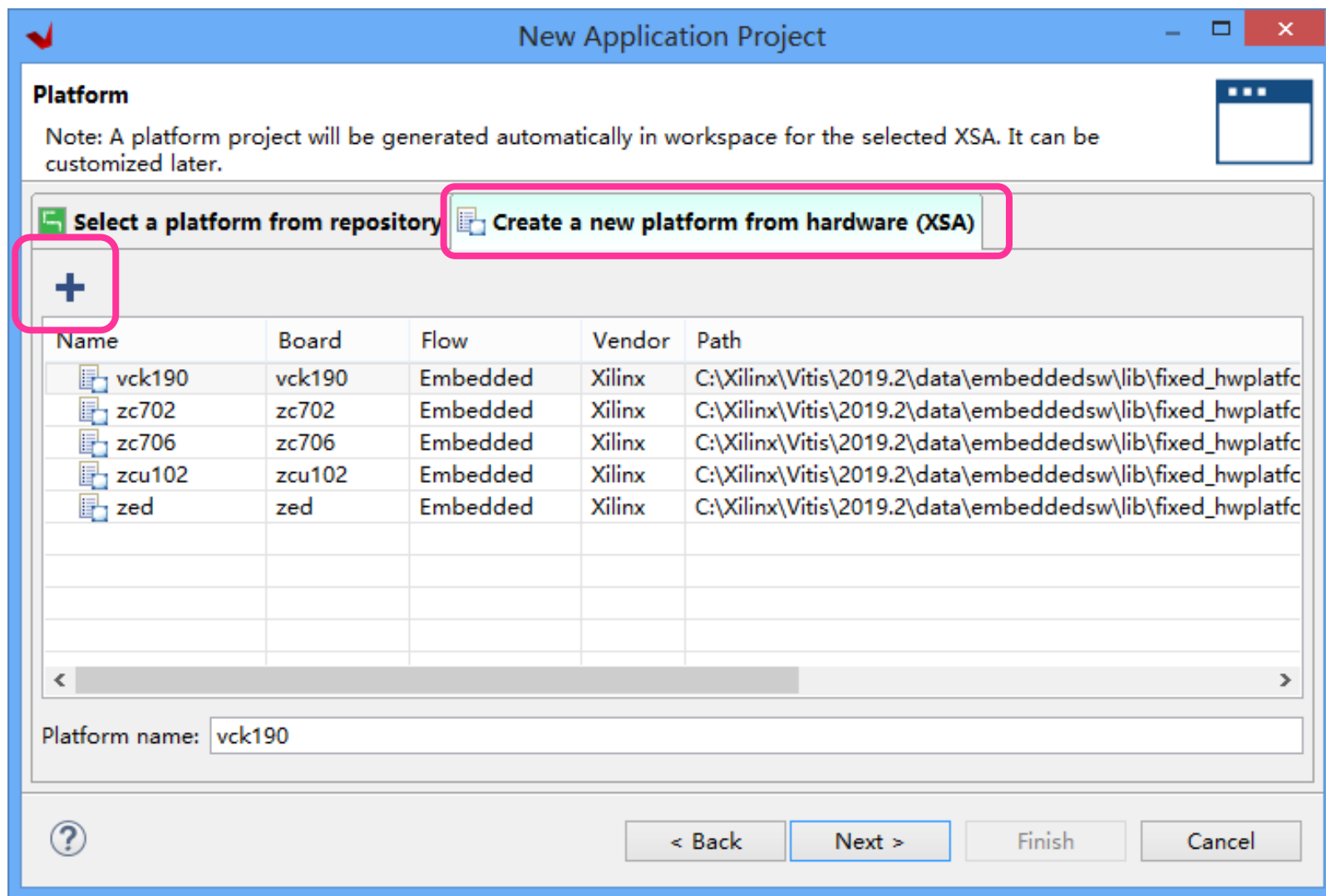
A72 Linux Domain

AI Engine Domain

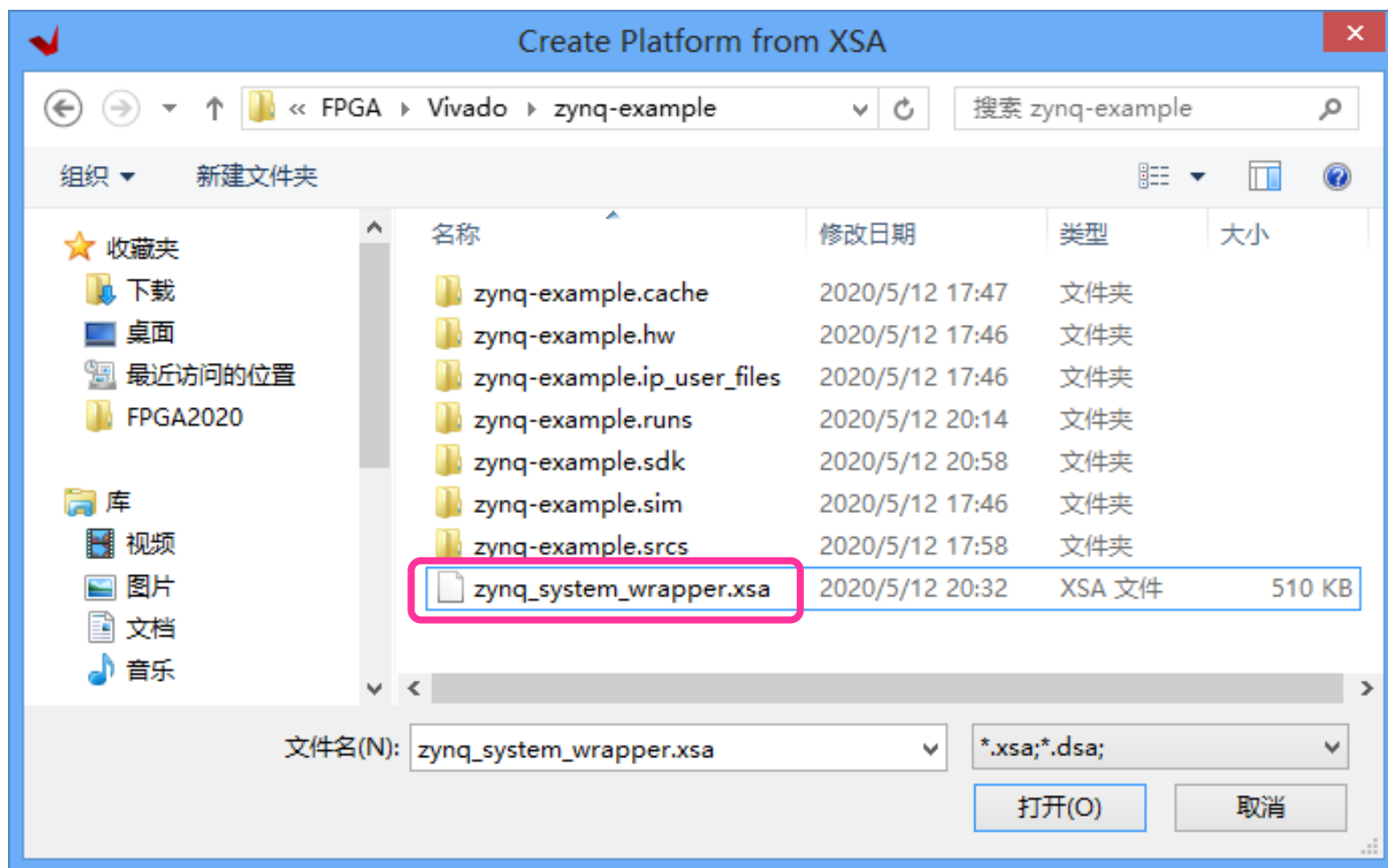
Platform

- A system project is a container for multiple applications that would run on different domains of a platform at the same time.
- A domain is the BSP/OS that controls one or more isomorphic processors.
- A platform contains one or more domains.
- A workspace can contain unlimited platforms and unlimited system projects

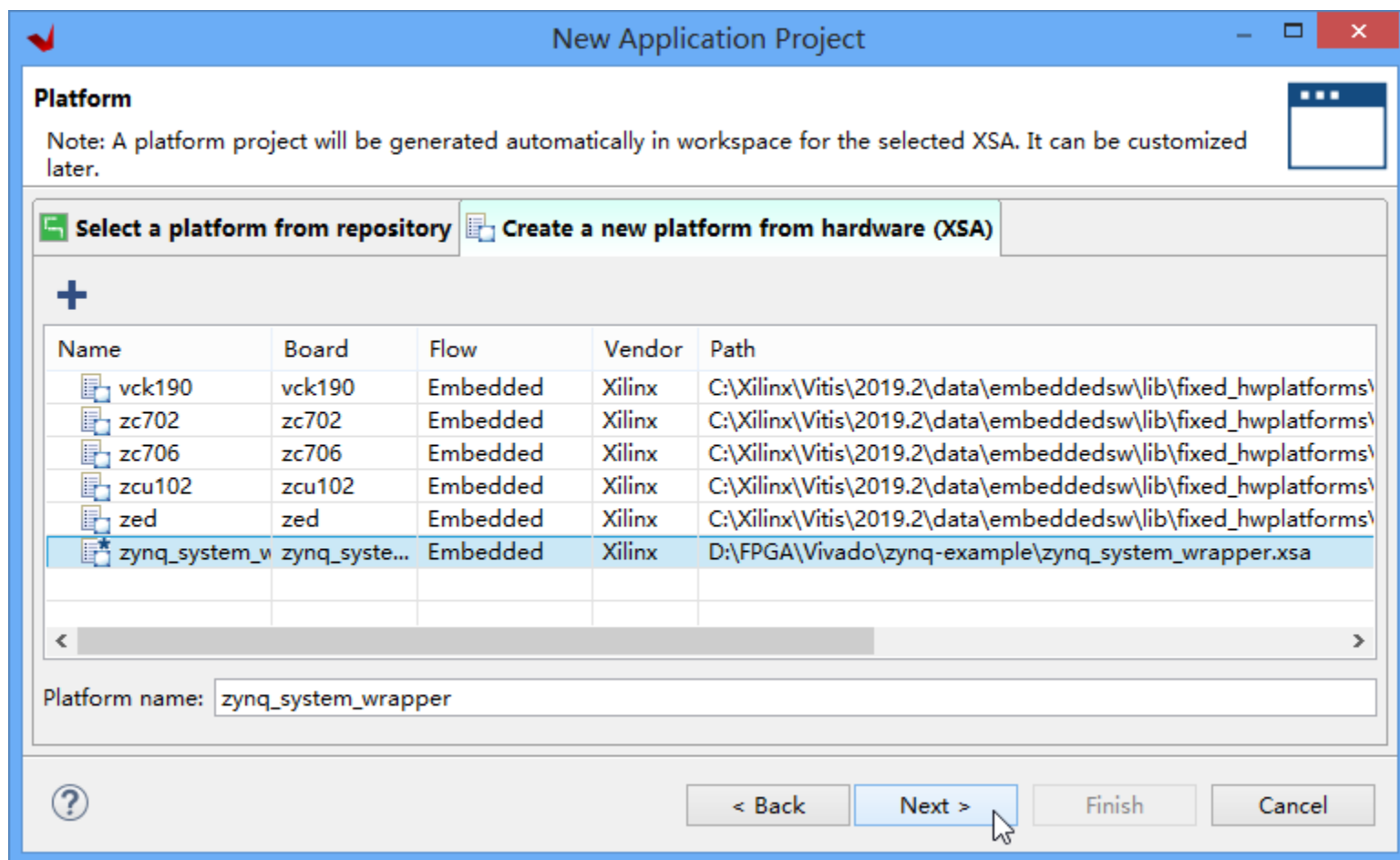
建立一个新的软件项目，输入工程名称，下面会同时建立一个 system project（BSP工程）。BSP是板级支持包Board Support Package（软硬件信息库及驱动程序库等）。



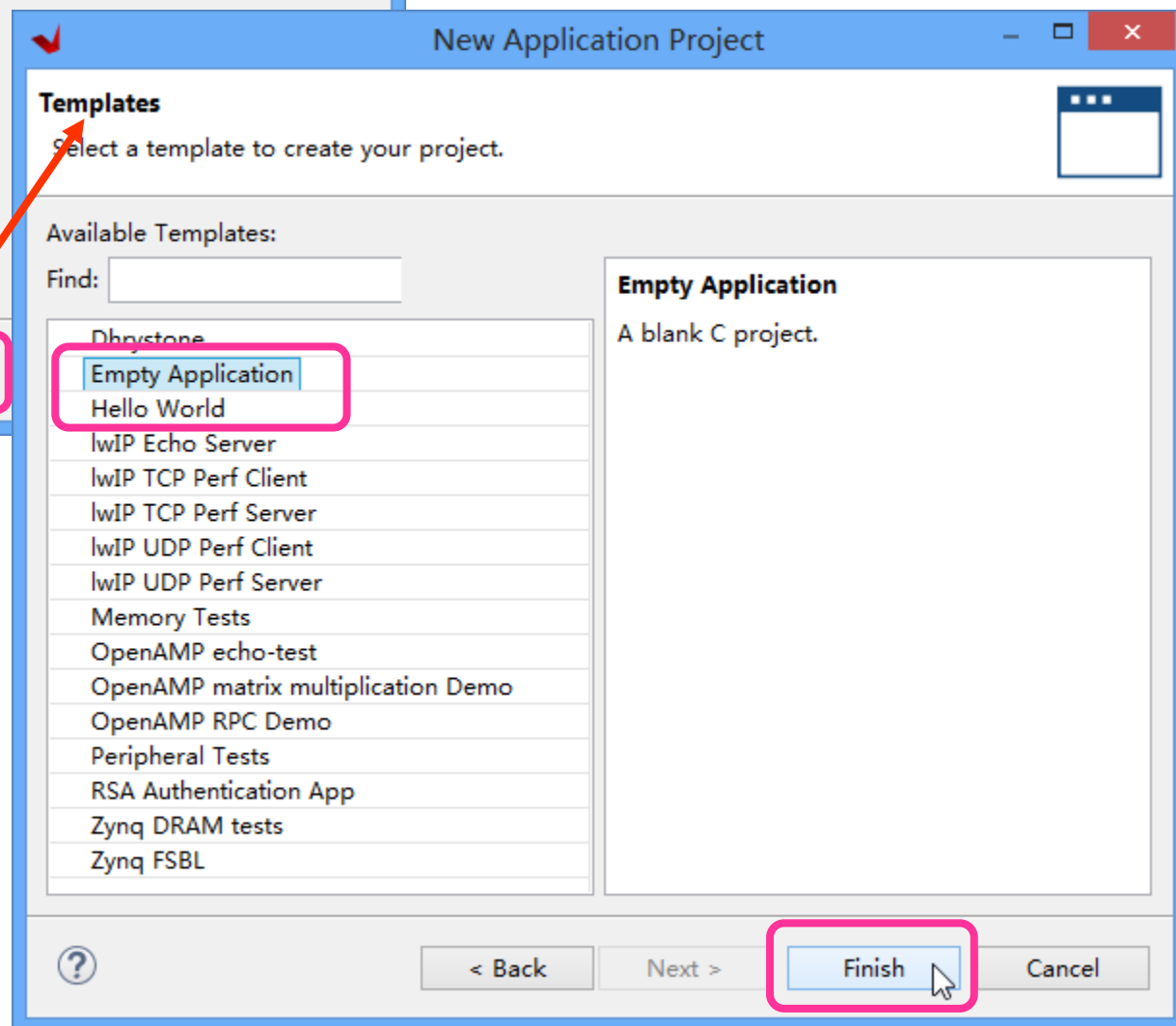
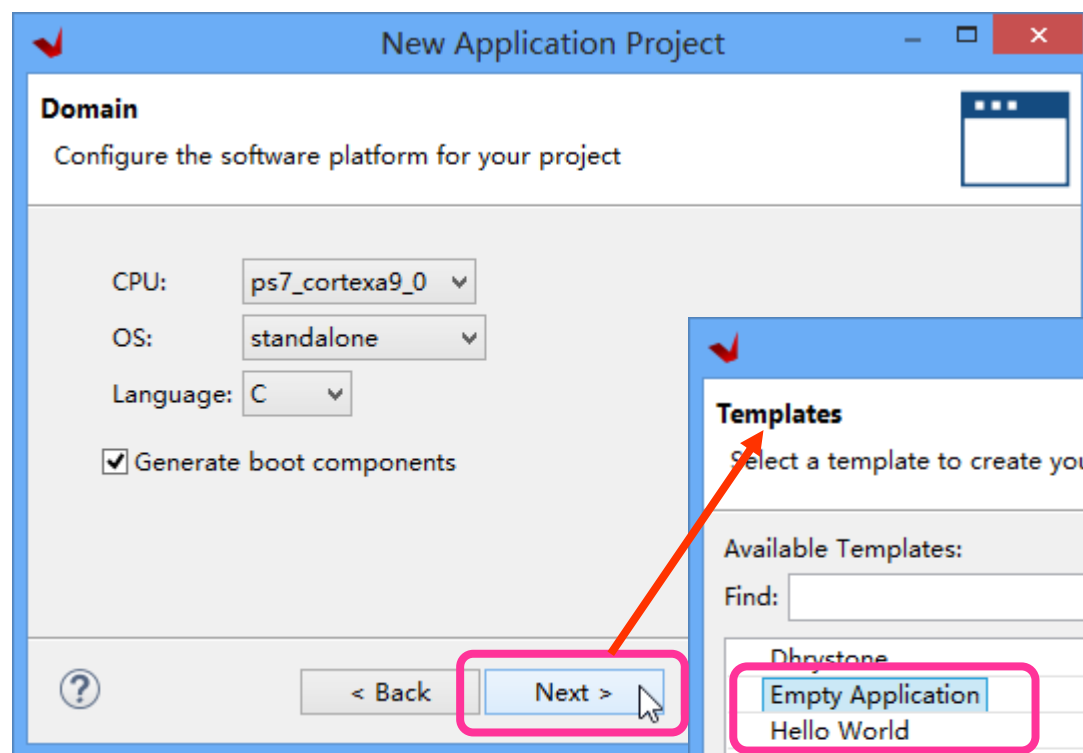
- 點選Create a new platform from hardware (XSA)标签，点击+号



- 浏览到本硬件工程所在目录，点选、打开刚才生成的xxx_wrapper.xsa文件

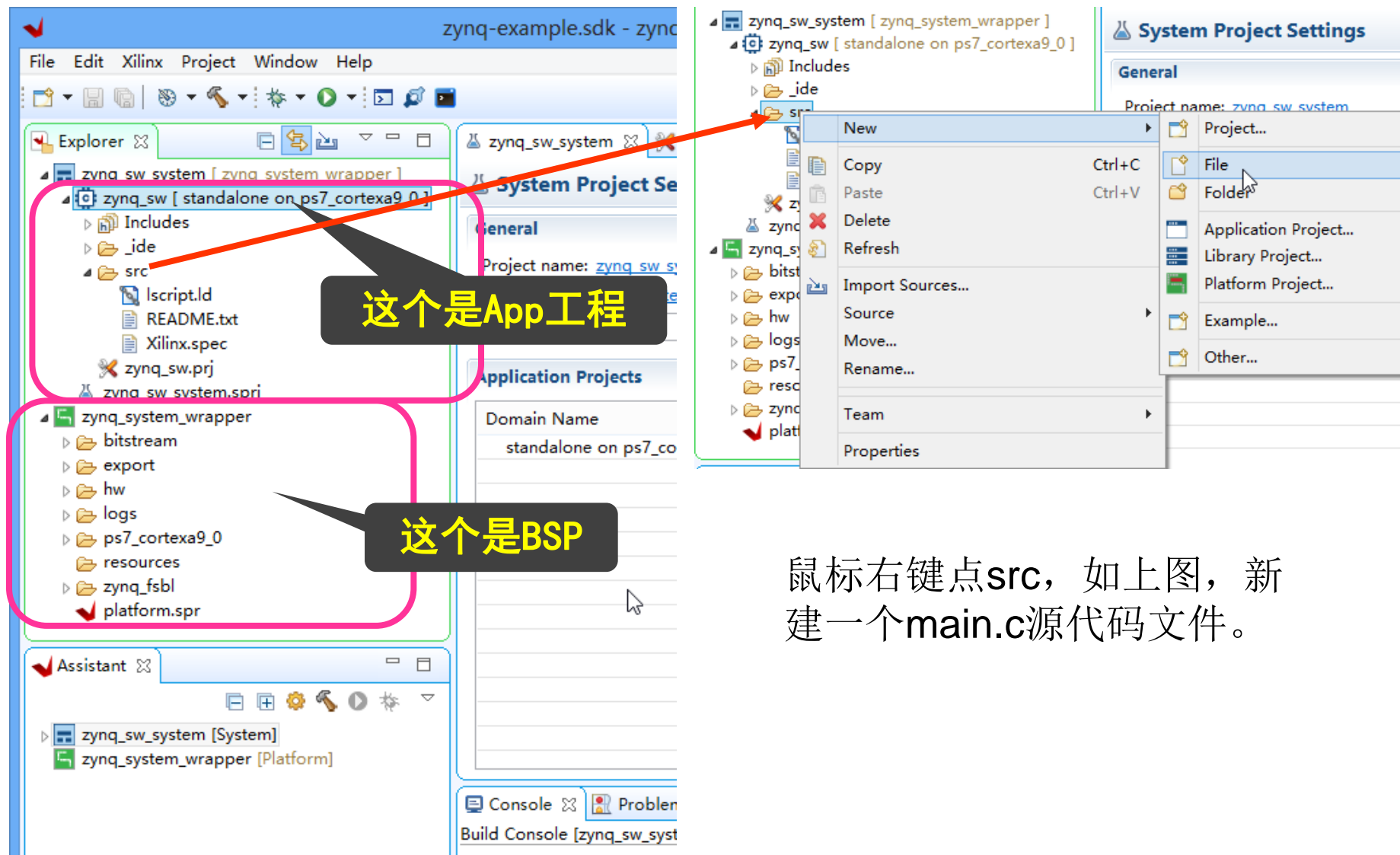


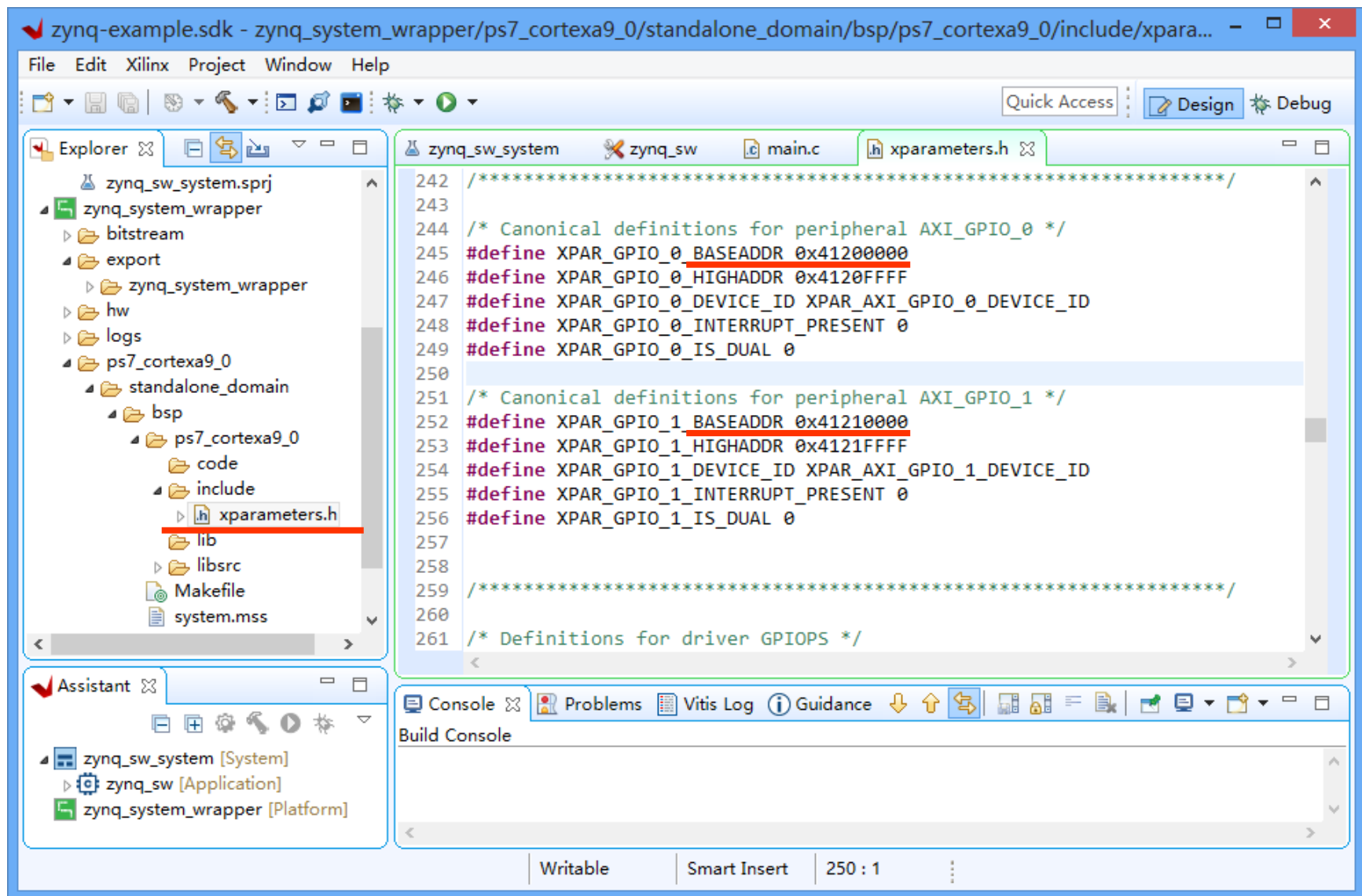
点选刚才导入的xsa文件，按Next按钮



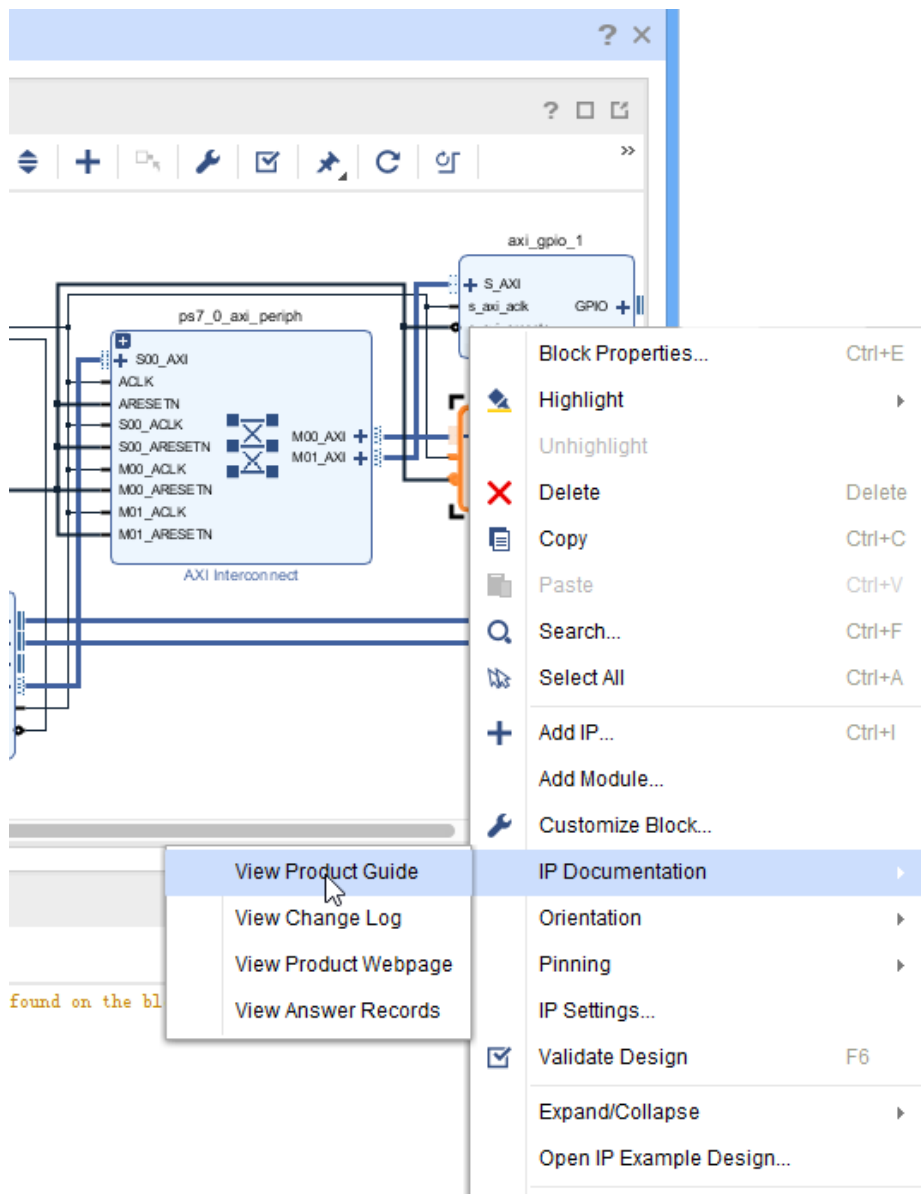
可以选新建一个空的C语言工程，更快捷的方法是选HelloWorld等例程，在此基础上修改。不同的例程包含不同的文件。

如果从空白工程过来，src源码目录里无C文件，则要新建主程序





在BSP xxx_wrapper -> ps7_cortexa9_0 -> standalone_domain -> bsp -> ps7_cortexa9_0 -> include里有个xparameters.h文件，里面有各个外设BASEADDR。总线型的IP通过读写寄存器操作，而寄存器分布在从基地址开始的一段地址空间内。这和Vivado的Address Editor里的设定一致。



去Vivado的块图编辑器，鼠标右键点击gpio块，菜单里有IP Documentation -> View Product Guides，可以在Documentation Navigator工具里下载和打开该IP的手册

其中第10页开始有Table2-4、2-6、2-7三个表格，在编程时得到。

Table 2-4: Registers

Address Space Offset ⁽³⁾	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER ⁽¹⁾	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER ⁽¹⁾	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR ⁽¹⁾	R/TOW ⁽²⁾	0x0	IP Interrupt Status Register.

DATA寄存器是GPIO的数据。TRI寄存器用来控制IO口方向，默认0x00

Table 2-7: AXI GPIO Three-State Register Description

Bits	Field Name	Access Type	Reset Value	Description
[GPIOx_Width-1 :0]	GPIOx_TRI	Read/Write	GPIO: Default Tri State Value GPIO2: Default Tri State Value	AXI GPIO 3-State Control Register. Each I/O pin of the AXI GPIO is individually programmable as an input or output. For each of the bits: 0 = I/O pin configured as output. 1 = I/O pin configured as input.

TRI寄存器的功能描述，复位默认0x00对应输出状态，然后去看表2-6

Table 2-6: AXI GPIO Data Register Description

Bits	Field Name	Access Type	Reset Value	Description
[GPIOx_Width-1 :0]	GPIOx_DATA	Read/Write	GPIO: Default Output Value GPIO2: Default Output Value	<p>AXI GPIO Data Register.</p> <p>For each I/O bit programmed as input:</p> <ul style="list-style-type: none"> • R: Reads value on the input pin. • W: No effect. <p>For each I/O bit programmed as output:</p> <ul style="list-style-type: none"> • R: Reads to these bits <u>always return zeros</u> • W: Writes value to the corresponding AXI GPIO data register bit and output pin.

表2-6里提到：

如果IO口被置为输出状态，读DATA寄存器将得到0（而不端口是实际逻辑值），这和现在许多微处理器的GPIO外设的行为是有悖的。

因为IO口默认为输出状态（TRI=0），如果作为输入口（比如接按键），需要先设置为输入状态，否则读回来的数据不对。

```
zynq_sw_system  main.c  xparameters.h  xgpio.c  xgpio_l.h
1 #include "xgpio_l.h"
2
3 #define LED_BASE  XPAR_GPIO_0_BASEADDR
4 #define BTN_BASE  XPAR_GPIO_1_BASEADDR
5
6 int main(void)
7 {
8     unsigned int uBtnValue;
9     //XGpio_WriteReg(BaseAddress, RegOffset, Data)
10    XGpio_WriteReg(BTN_BASE, 0x04, 0xff); //set as input
11    XGpio_WriteReg(LED_BASE, 0x04, 0x00); //set as output
12
13    while (1)
14    {
15        //XGpio_ReadReg(BaseAddress, RegOffset)
16        uBtnValue = XGpio_ReadReg(BTN_BASE, 0);
17        XGpio_WriteReg(LED_BASE, 0, uBtnValue);
18    }
19
20    return 0;
21 }
```

定义助记符。

初始化IO口方向。

读按键状态，显示在指示灯上。

Console Problems Vitis Log Guidance

Build Console [zynq_sw, Debug]

'Invoking: ARM v7 Print Size'

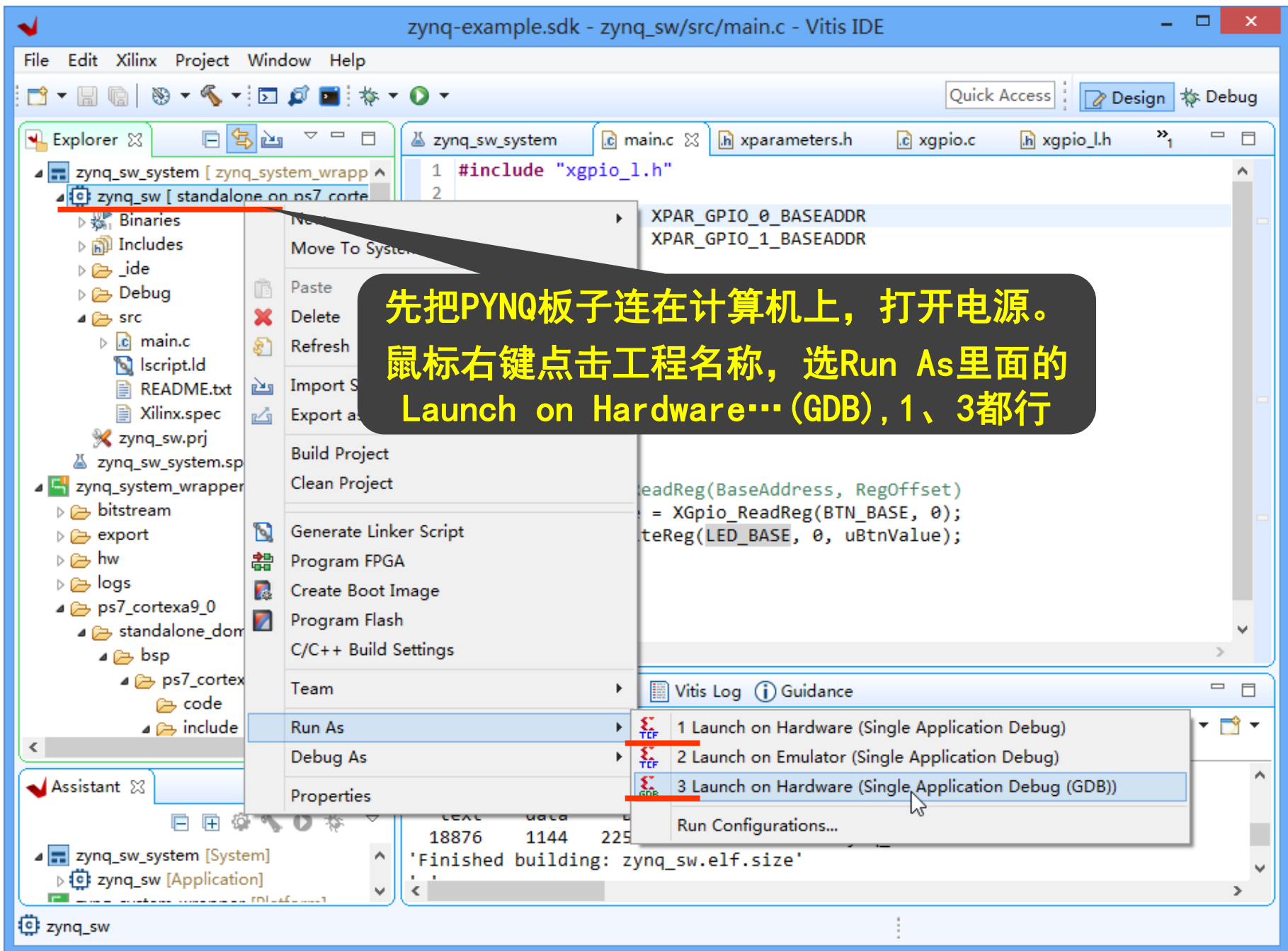
arm-none-eabi-size zynq_sw.elf |tee "zynq_sw.elf.size"

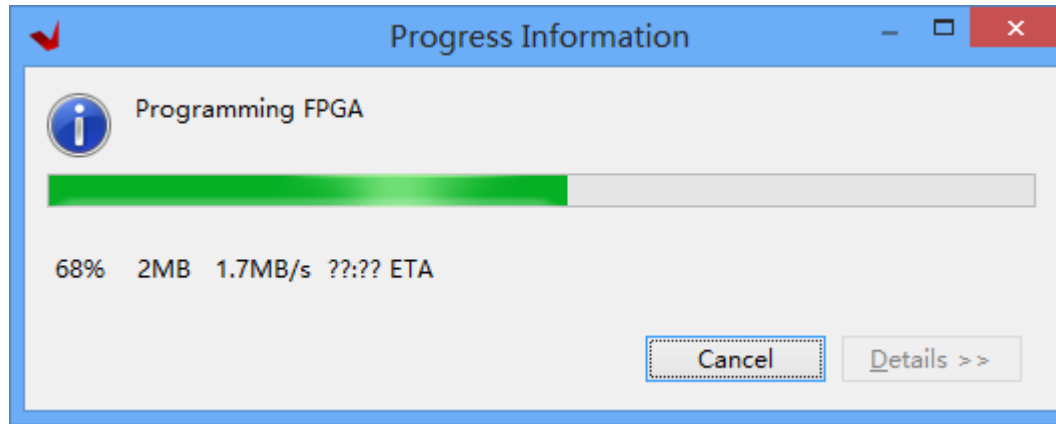
text	data	bss	dec	hex	filename
------	------	-----	-----	-----	----------

18876	1144	22568	42588	a65c	zynq_sw.elf
-------	------	-------	-------	------	-------------

'Finished building: zynq_sw.elf.size'

写完程序，去主菜单 Project
→ Build Project

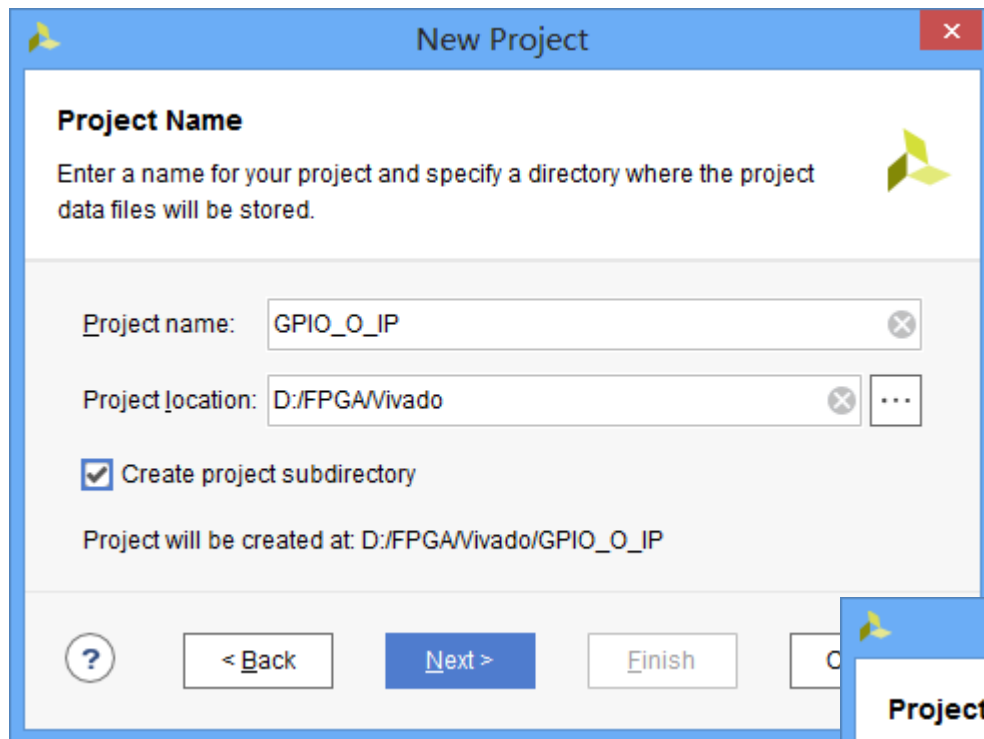




- Vitis IDE会自动下载bitstream配置FPGA、下载App配置PS、在系统里运行程序。
- 按板子上的按键，对应的LED应该会亮。
- 注意到：在工程实现过程中，未曾编写过xdc文件，但是工程正常实现。因为建立工程时使用了板卡文件，用户未指定的io约束，都默认使用板卡文件里的设置。

3. 自定义IP及使用





New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location: ...

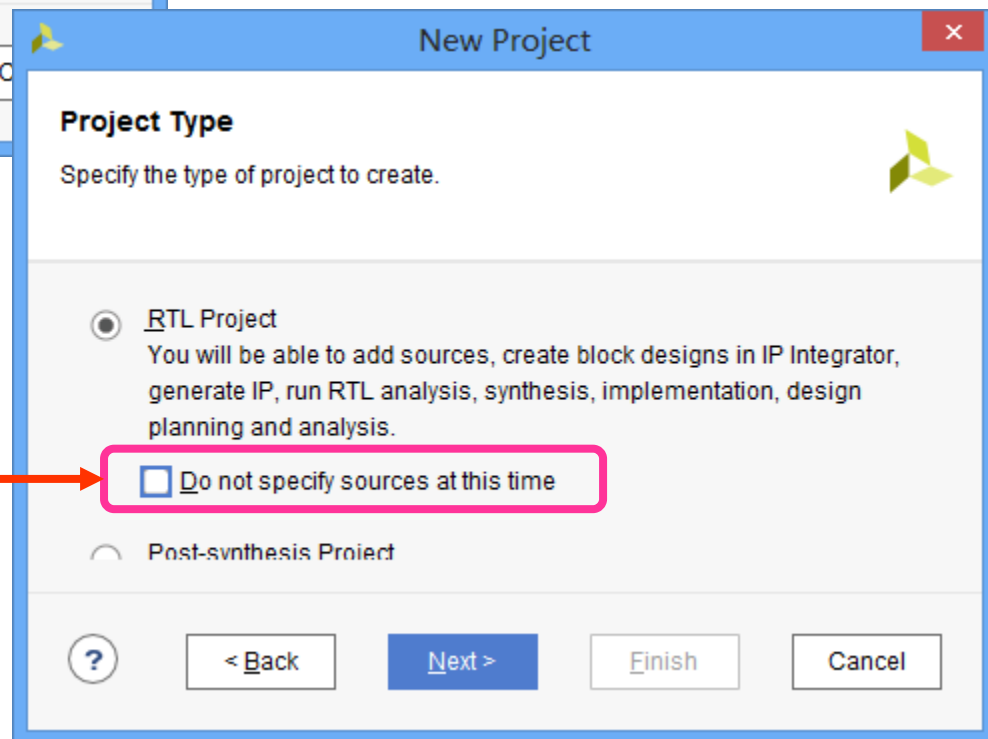
☒ Create project subdirectory

Project will be created at: D:/FPGA/Vivado/GPIO_O_IP

? < Back Next > Finish

指定工程名和目录

工程类型：RTL工程。
此项不选。



New Project

Project Type

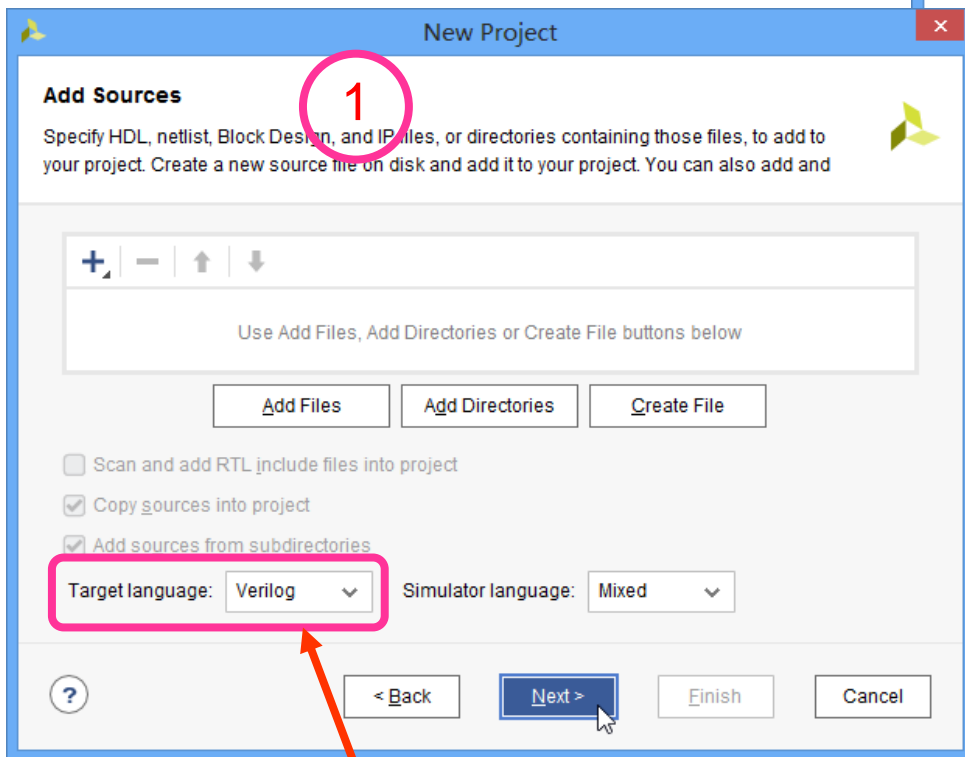
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ **Do not specify sources at this time**

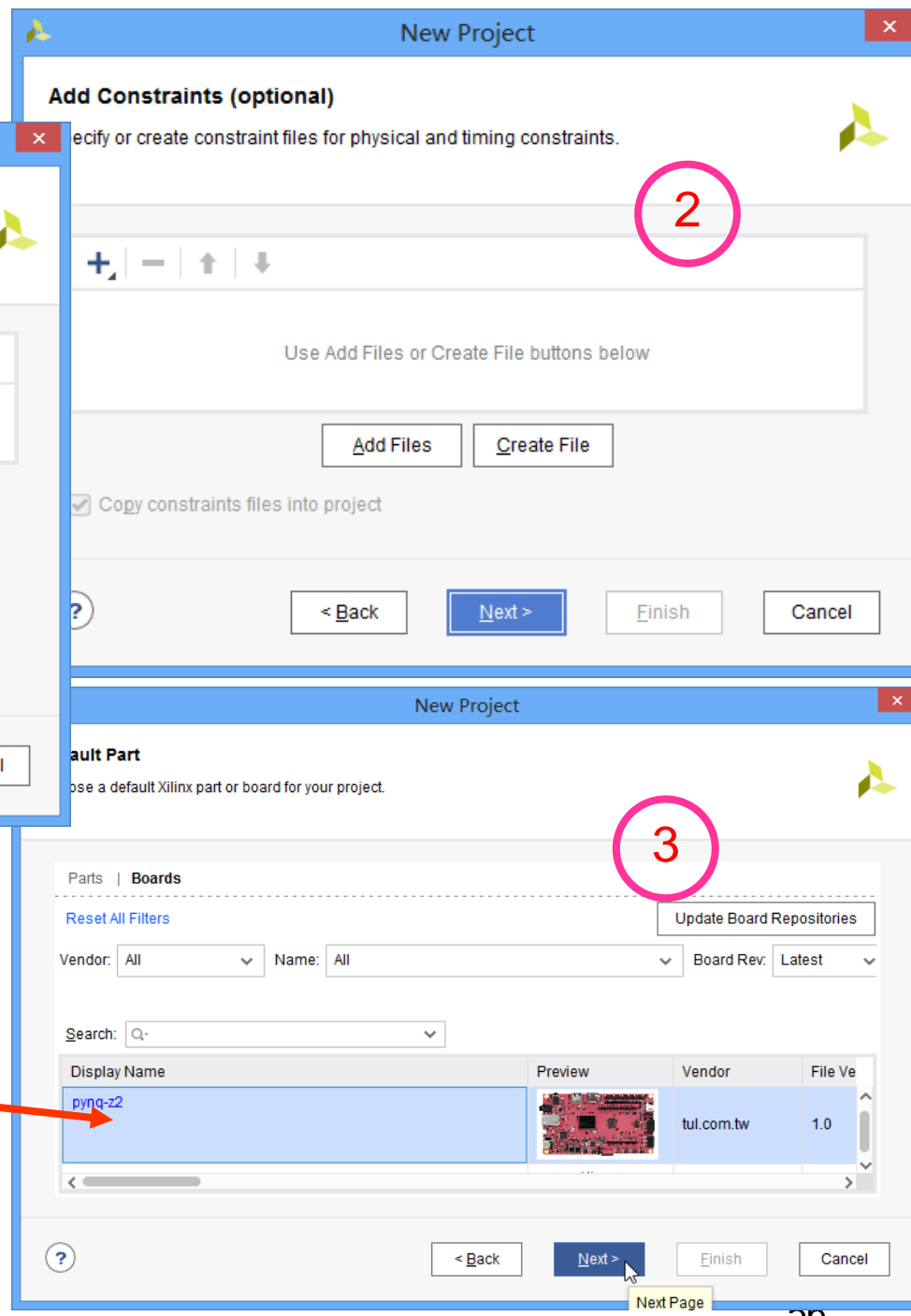
☐ **Post-synthesis Project**

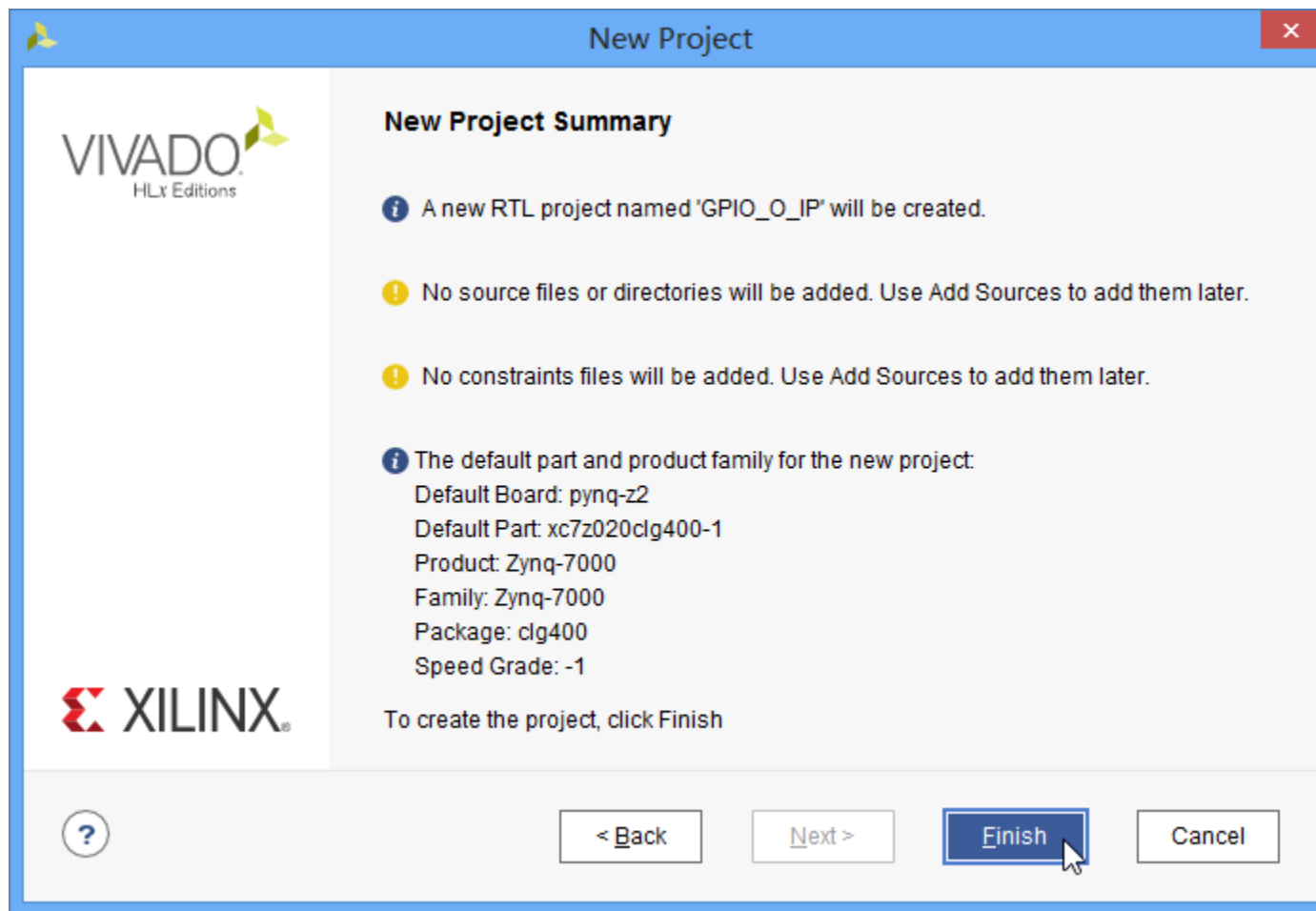
? < Back Next > Finish Cancel



目标语言选Verilog HDL

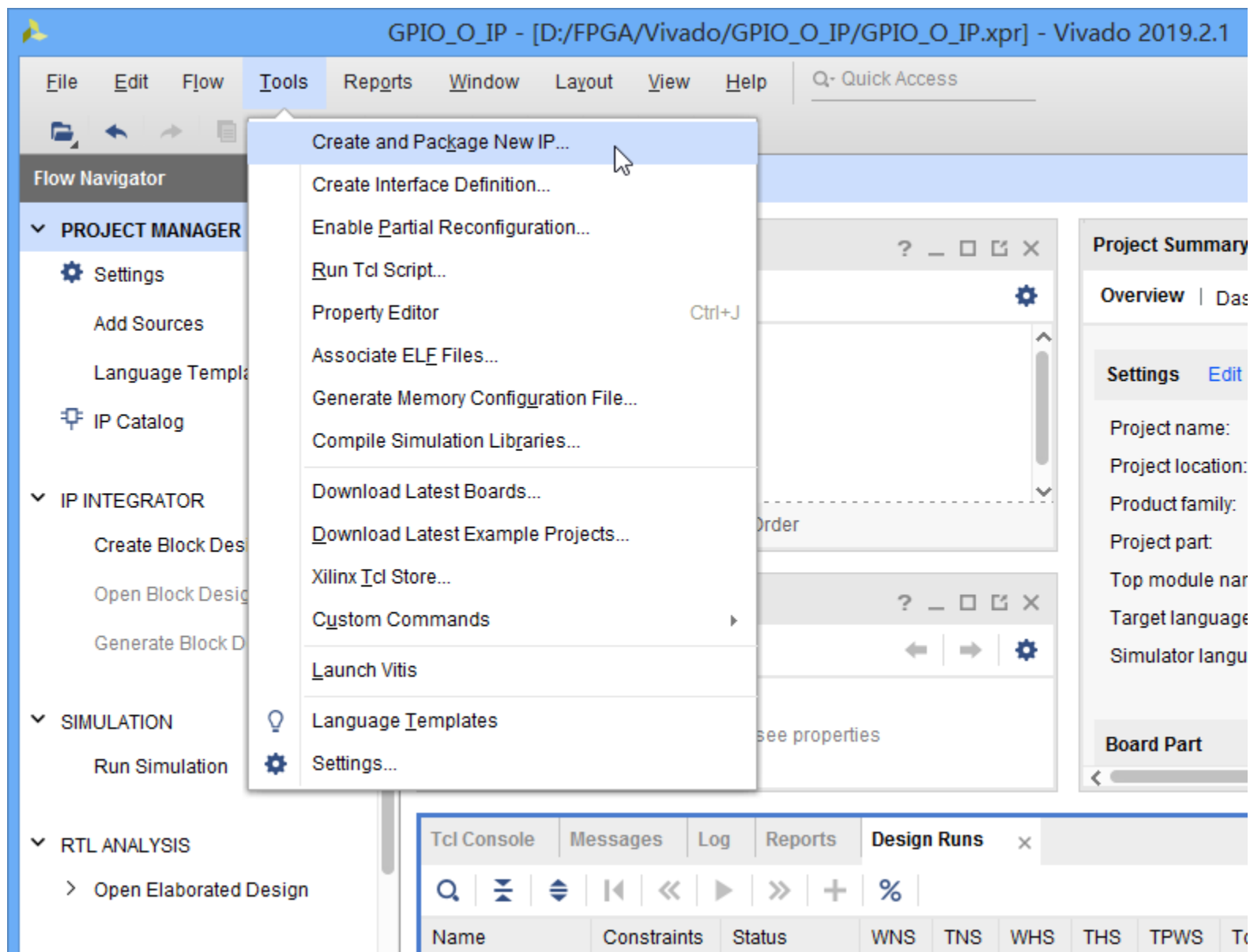
板子还选TUL的PYNQ-Z2



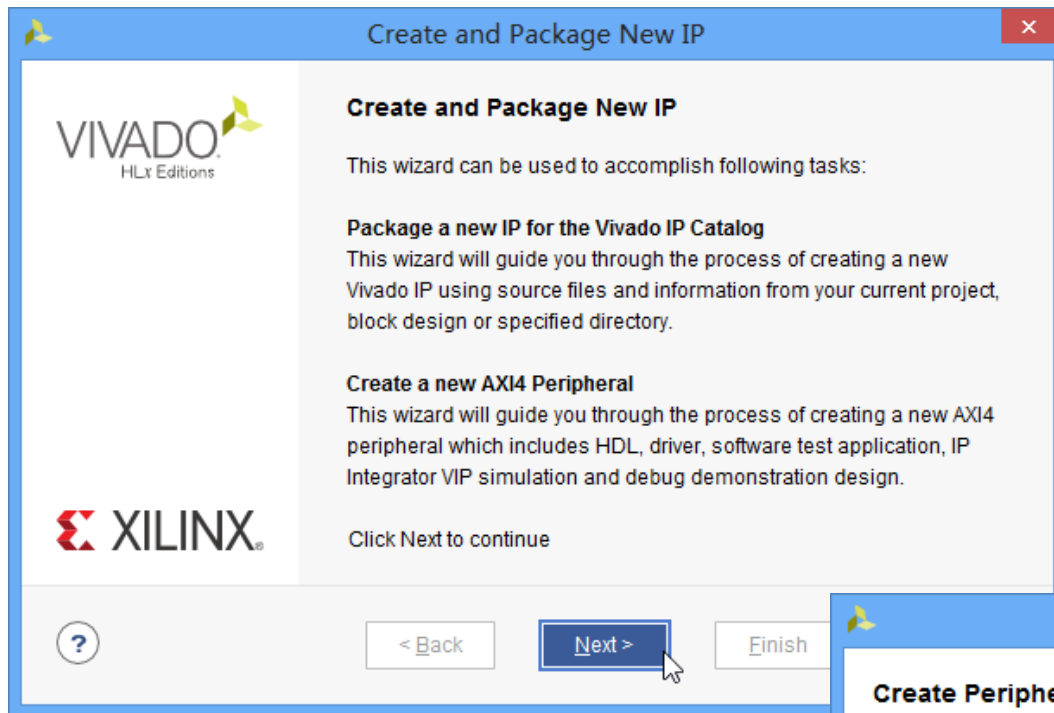


工程信息汇总

以下步骤要新建一个GPIO的IP，用来连接LED

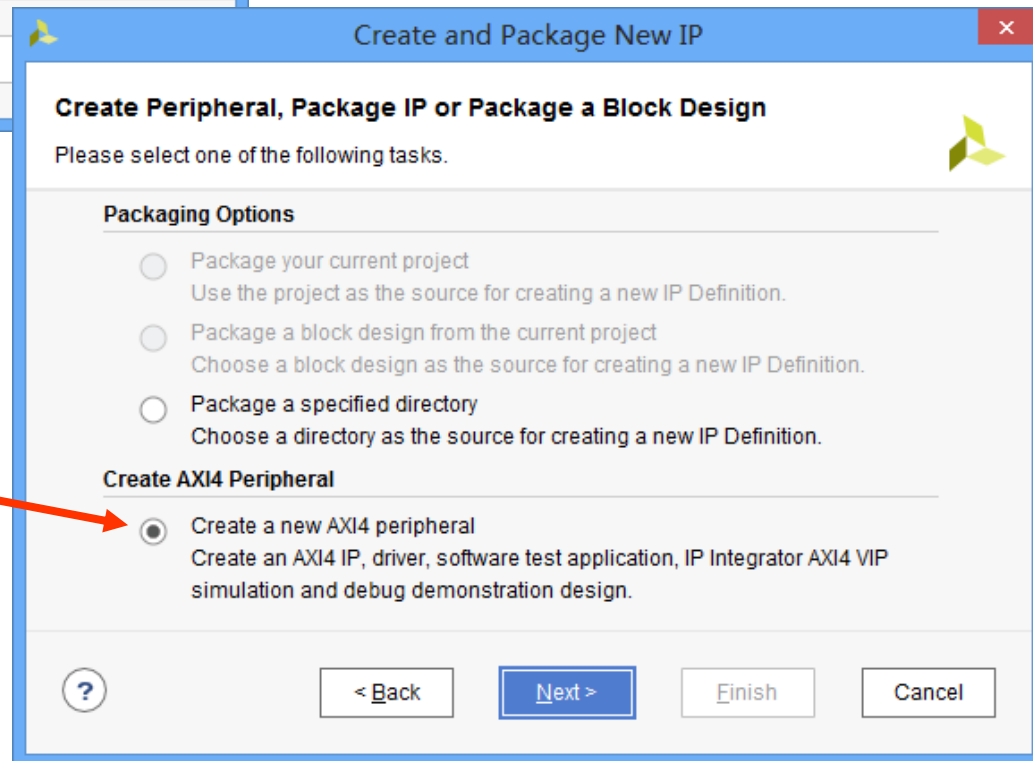


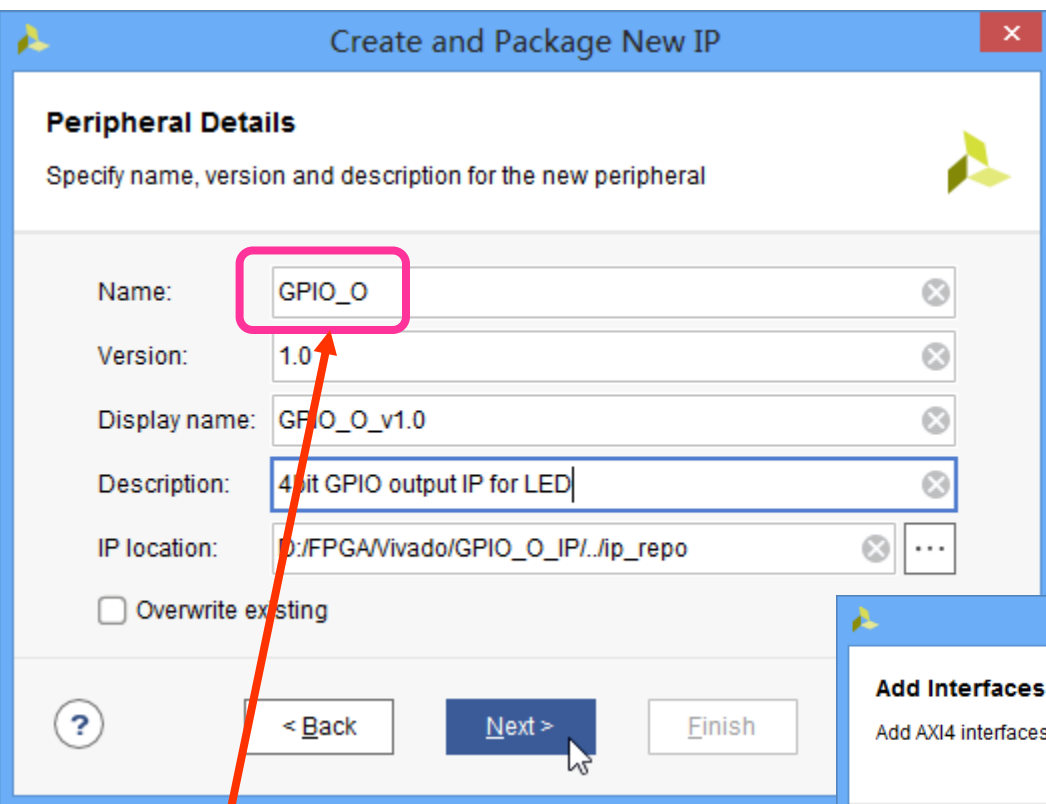
在主菜单选Tools -> Create and Package New IP...



建立和封装新IP

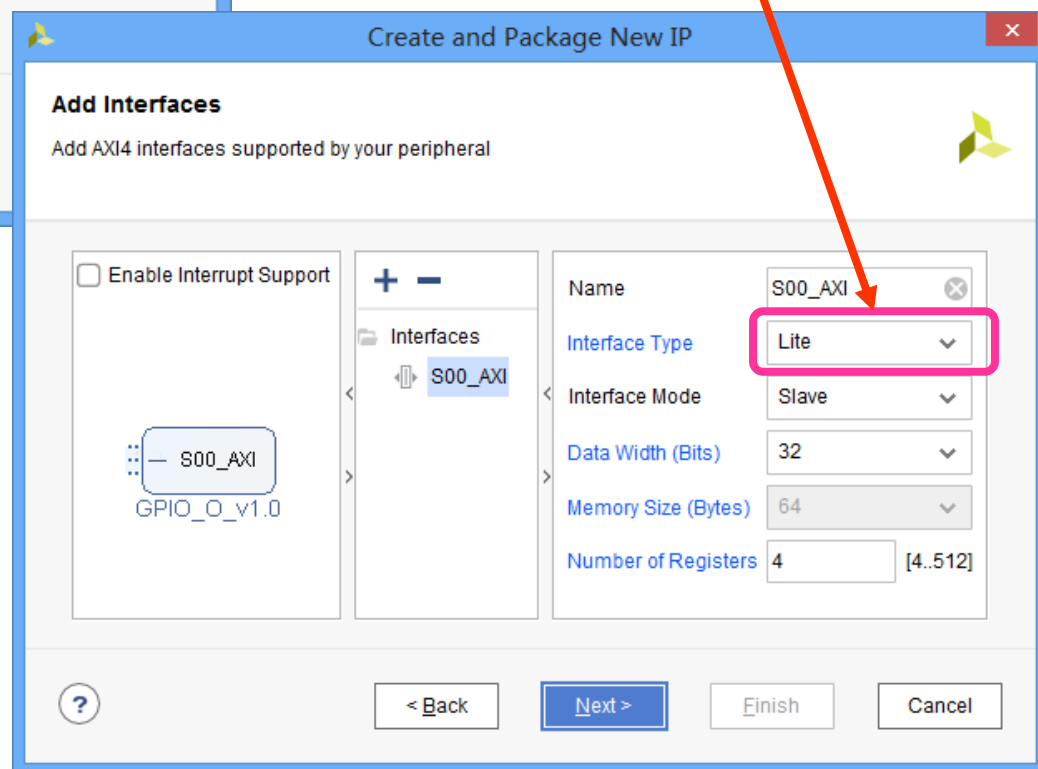
选这项：新建AXI4外设

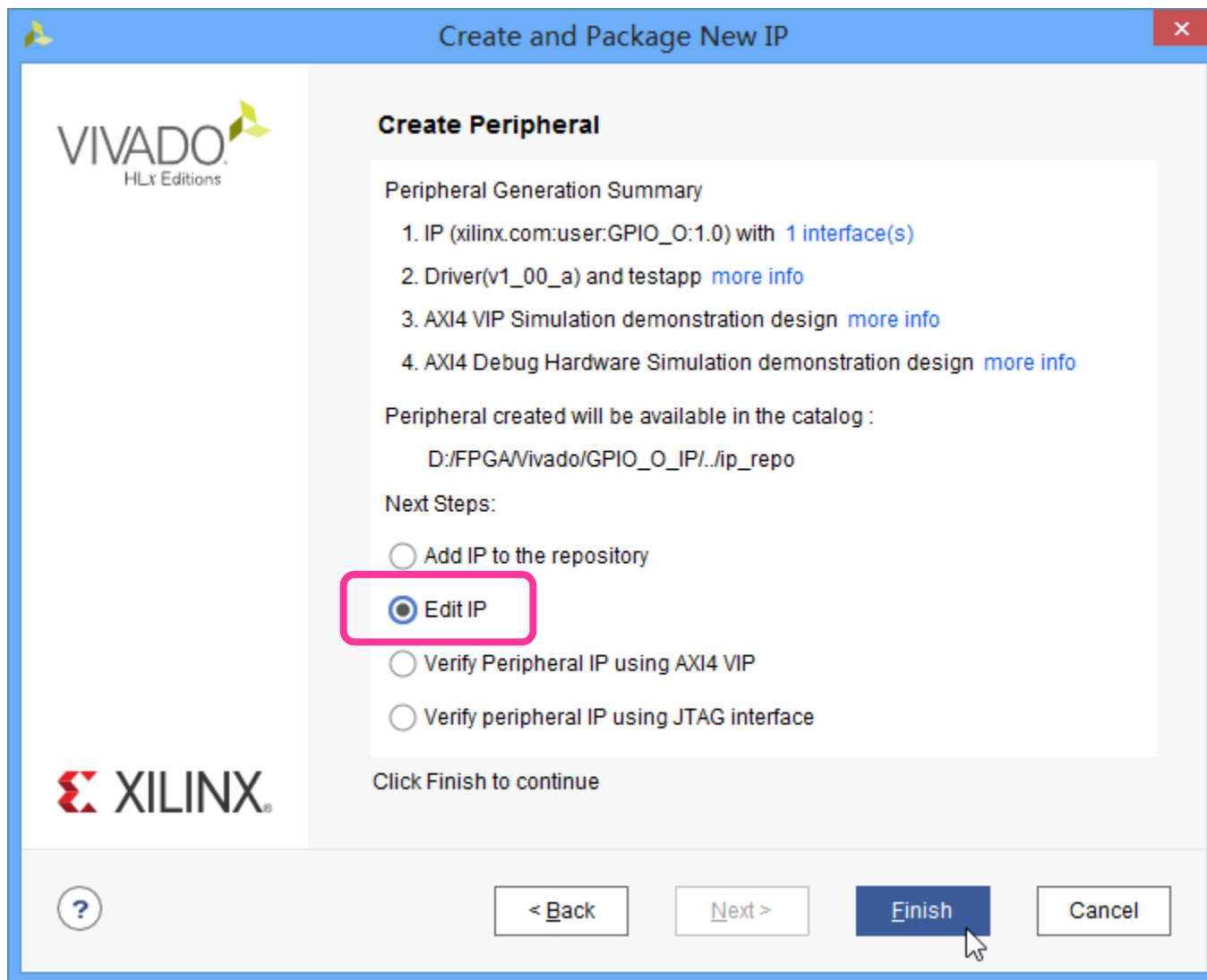




给新建的IP命名、
填写Description

简单的GPIO IP接口类型
选Lite，寄存器最少是4个





选Edit IP这项，要去编辑IP源码，设计自己的逻辑

编辑IP文件

PROJECT MANAGER - edit_GPIO_O_v1_0

Sources

- Design Sources (2)
 - GPIO_O_v1_0 (GPIO_O_v1_0.v) (1)
 - GPIO_O_v1_0_S00_AXI_inst: GPIO_O_v1_0
- IP-XACT (1)
- Constraints
- Simulation Sources
- Utility

Properties

Select an object to see properties

Project Summary | **Package IP - GPIO_O**

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- ✓ Review and Package

Identification

Vendor: xilinx.com

Library: user

Name: GPIO_O

Version: 1.0

Display name: GPIO_O_v1.0

Description: 4bit GPIO output IP for LED

Vendor display name:

Company url:

Root directory: d:/FPGA/Vivado/ip_repo/GPIO_O_1.0

Xml file name: d:/FPGA/Vivado/ip_repo/GPIO_O_1.0/component.xml

Categories

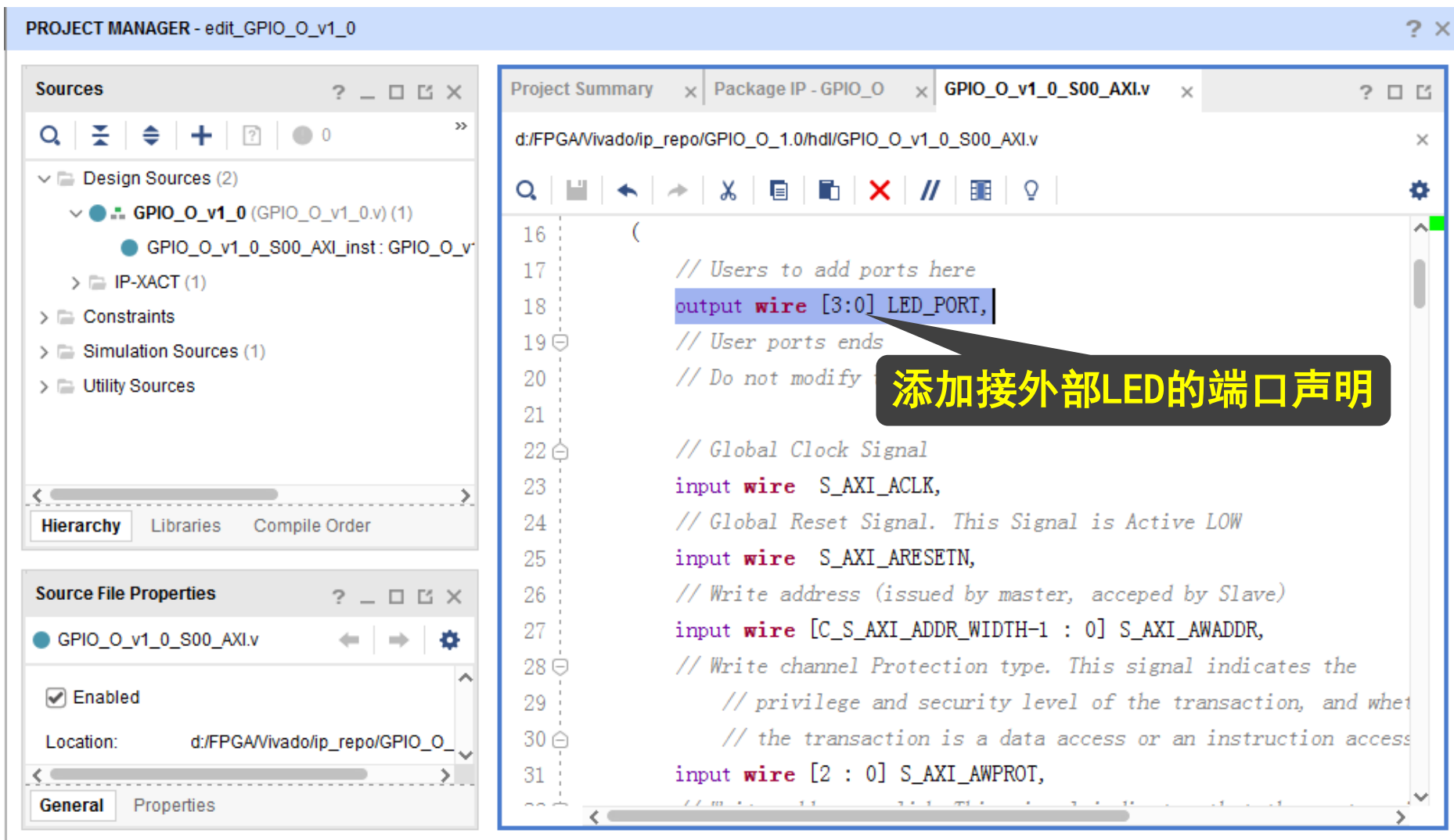
+ - ↑ ↓

AXI_Peripheral

封装IP步骤

双击打开底层v文件

添加外部端口和逻辑



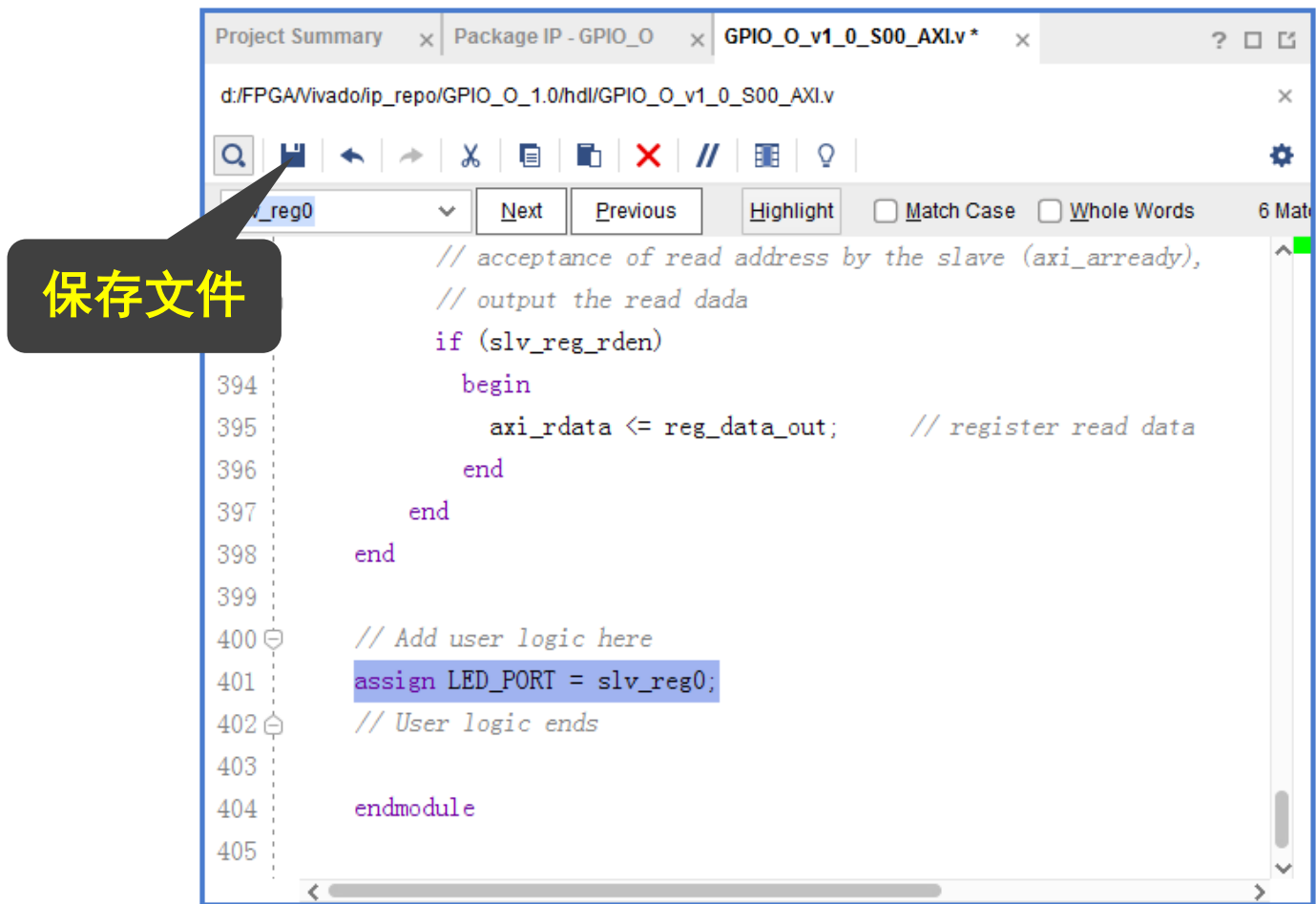
IP模板里只有AXI总线接口相关的逻辑，需要在底层功能模块里加入端口和逻辑。首先找到“//Users to add ports here”，添加外部端口声明。

```
Project Summary x Package IP - GPIO_O x GPIO_O_v1_0_S00_AXI.v x ? □ □
d:/FPGA/Vivado/ip_repo/GPIO_O_1.0/hdl/GPIO_O_v1_0_S00_AXI.v
slv_reg0 Next Previous Highlight Match Case Whole Words
103 //-----
104 //-- Signals for user logic register space example
105 //-----
106 //-- Number of Slave Registers 4
107 reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg0;
108 reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg1;
109 reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg2;
110 reg [C_S_AXI_DATA_WIDTH-1:0] slv_reg3;
111 wire slv_reg_rden;
112 wire slv_reg_wren;
113 reg [C_S_AXI_DATA_WIDTH-1:0] reg_c
114 integer byte_index;
115 reg aw_en;
```

```
231 case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
232     2'h0:
233         for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/
234             if ( S_AXI_WSTRB[byte_index] == 1 ) begin
235             // Respective byte enables are asserted as per write
236             // Slave register 0
237             slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_i
238         end
239     2'h1:
240         for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/
241             if ( S_AXI_WSTRB[byte_index] == 1 ) begin
242             // Respective byte enables are asserted as per write
243             // Slave register 1
244             slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_i
245         end
```

V文件里定义了4个reg，之后在230行附近，用case语句分出了地址。

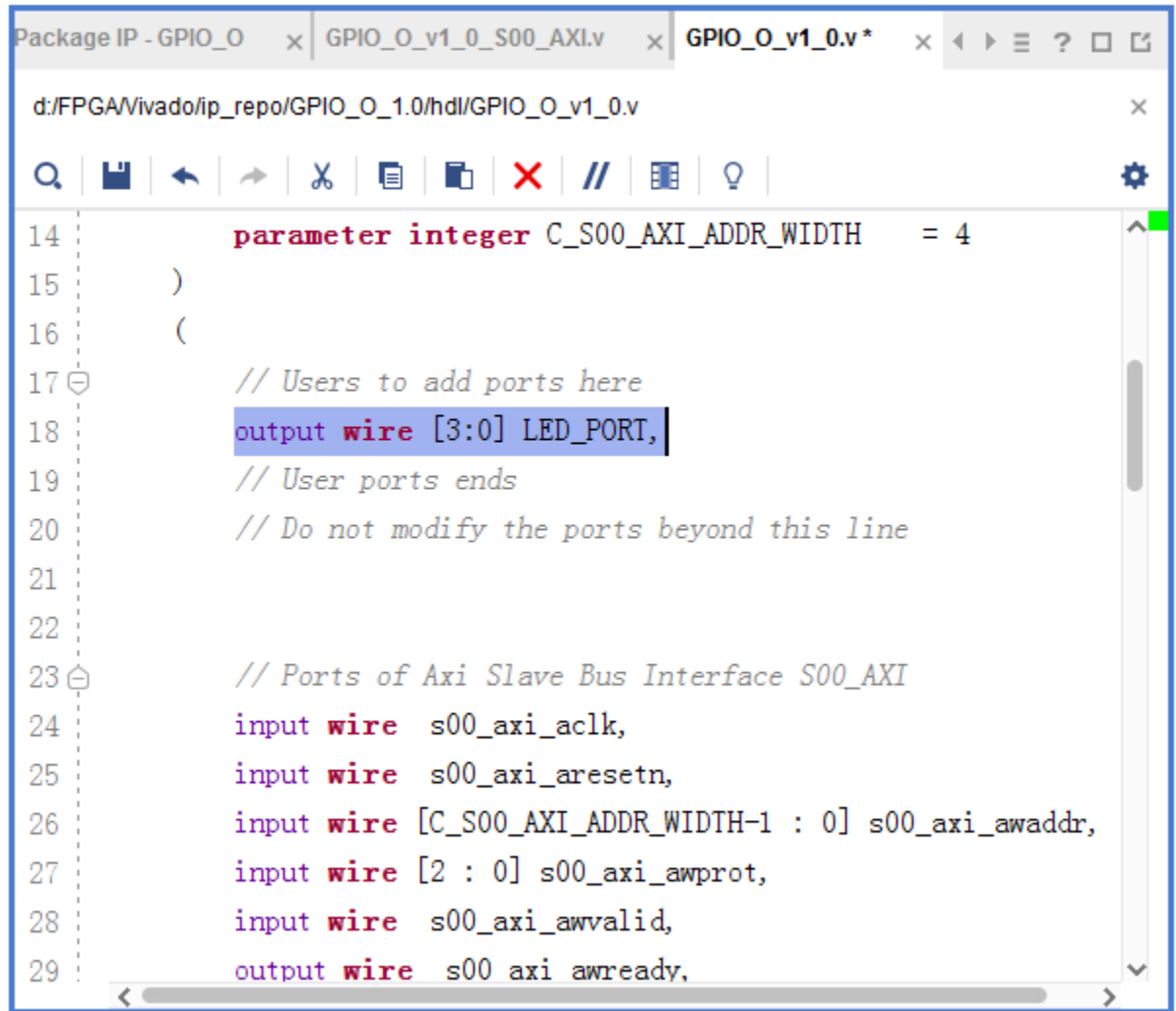
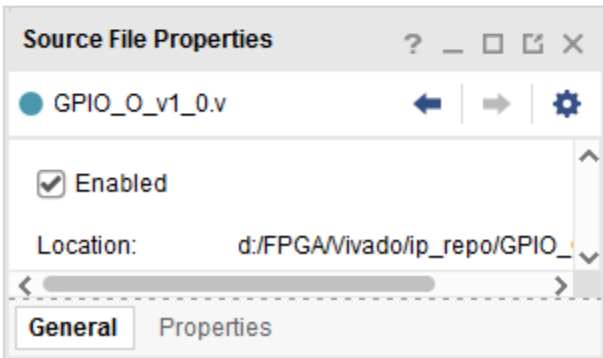
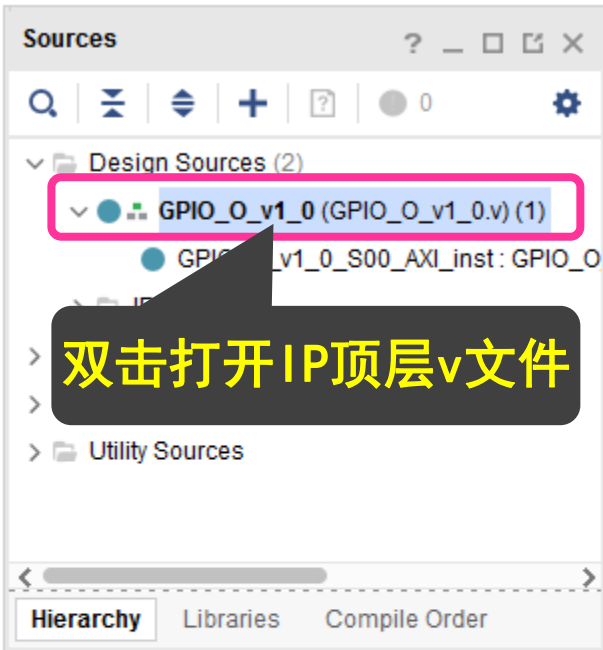
这里使用slv_reg0传递数据



转到v文件最后，在 //Add user logic here 这里添加用户逻辑，因为仅需要把寄存器的值输出给LED端口，所以只需添加一句：

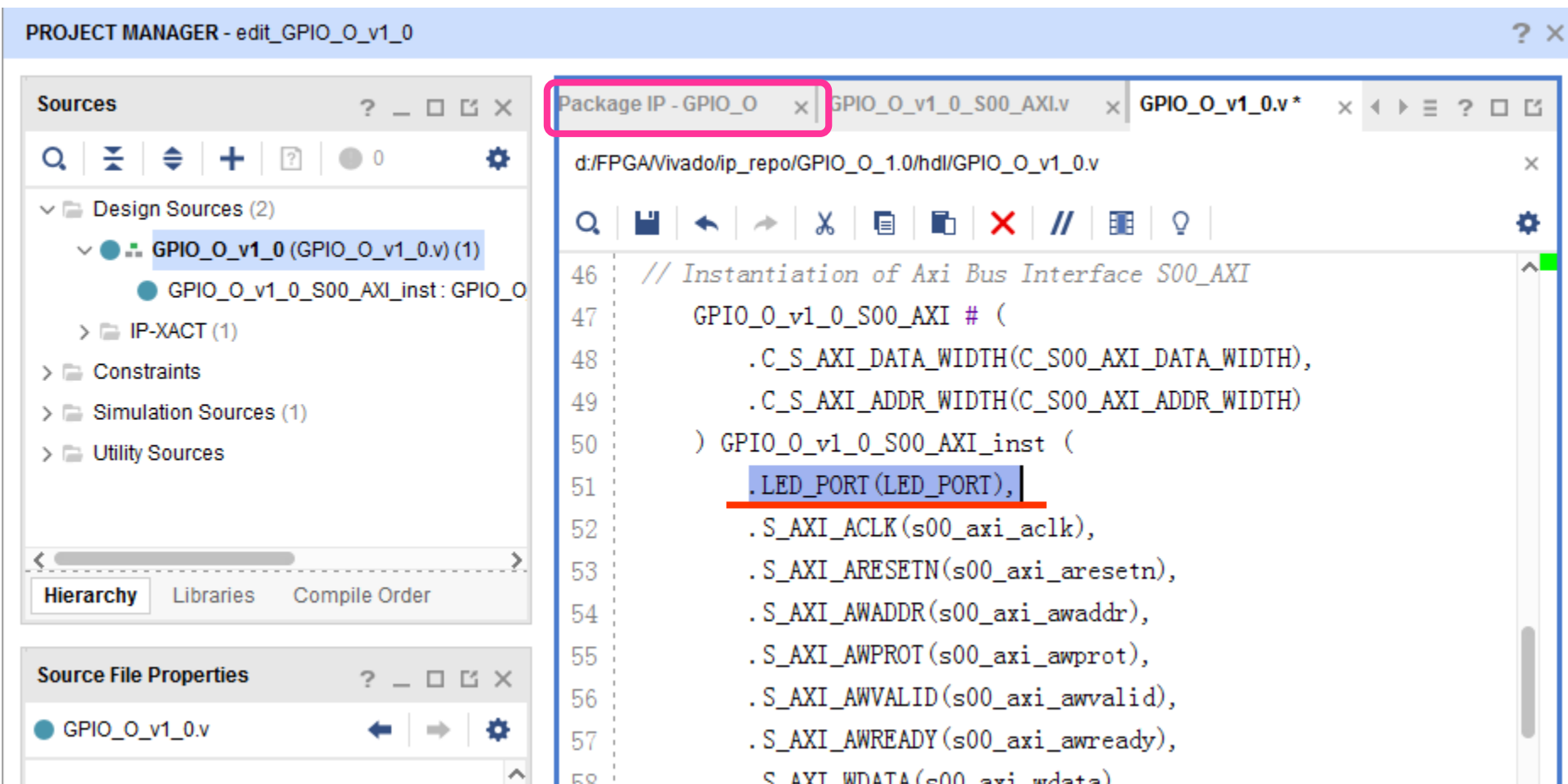
assign LED_PORT = slv_reg0;

程序通过AXI总线把数据写给reg0，上述语句把reg0的值输出给外部端口。



需要把底层模块新加的外部端口引出到顶层模块外部，在Source窗口双击IP的顶层文件，在// Users to add ports here之后加一句外部端口声明：

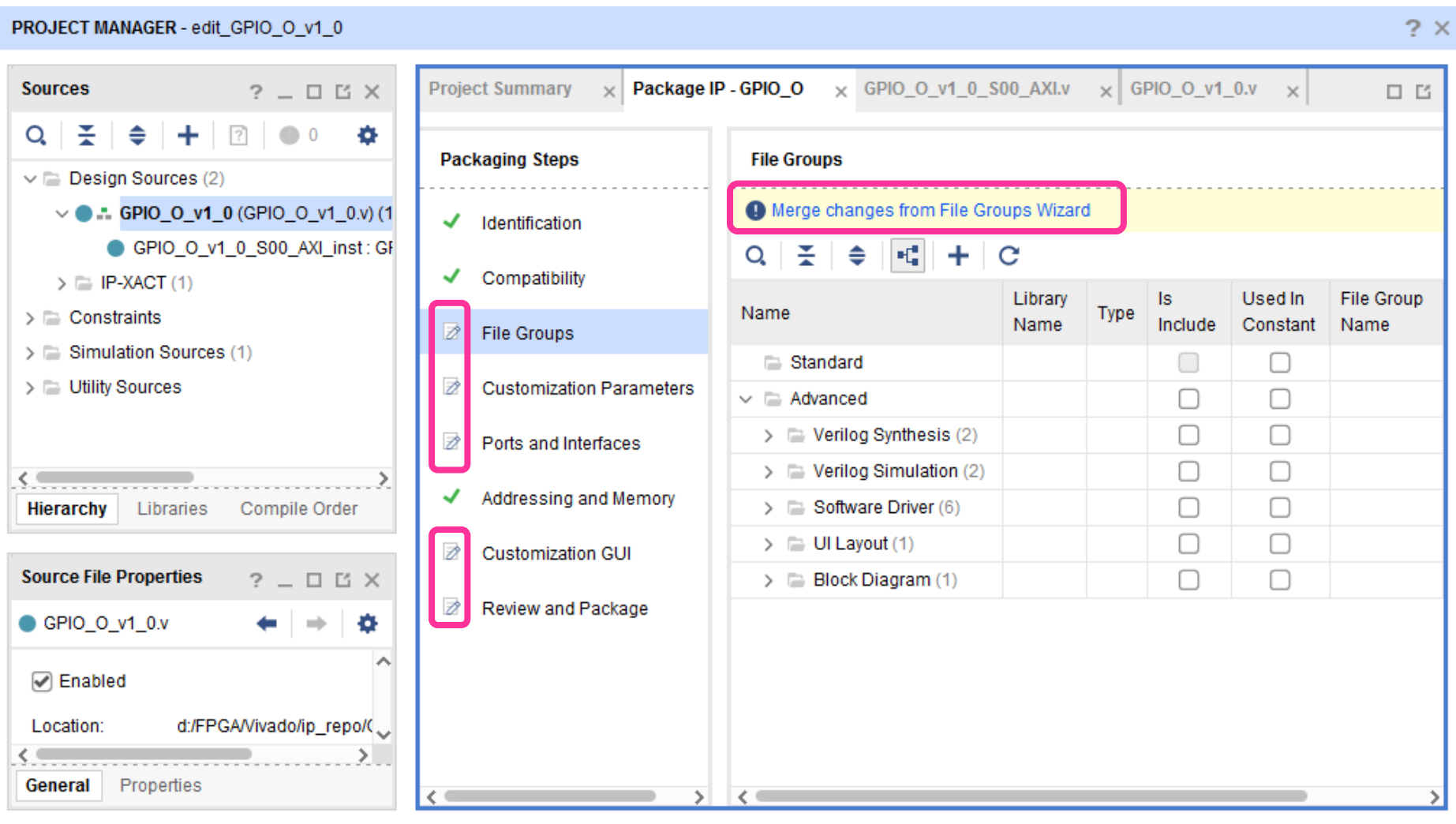
output wire [3:0] LED_PORT,



因为之前在底层总线接口模块v文件里添加了新端口，因此顶层模块里，底层模块实例化部分也需要体现出新加的端口，如图添加一句：

.LED_PORT(LED_PORT),

修改完毕，保存文件，回到Package IP标签。

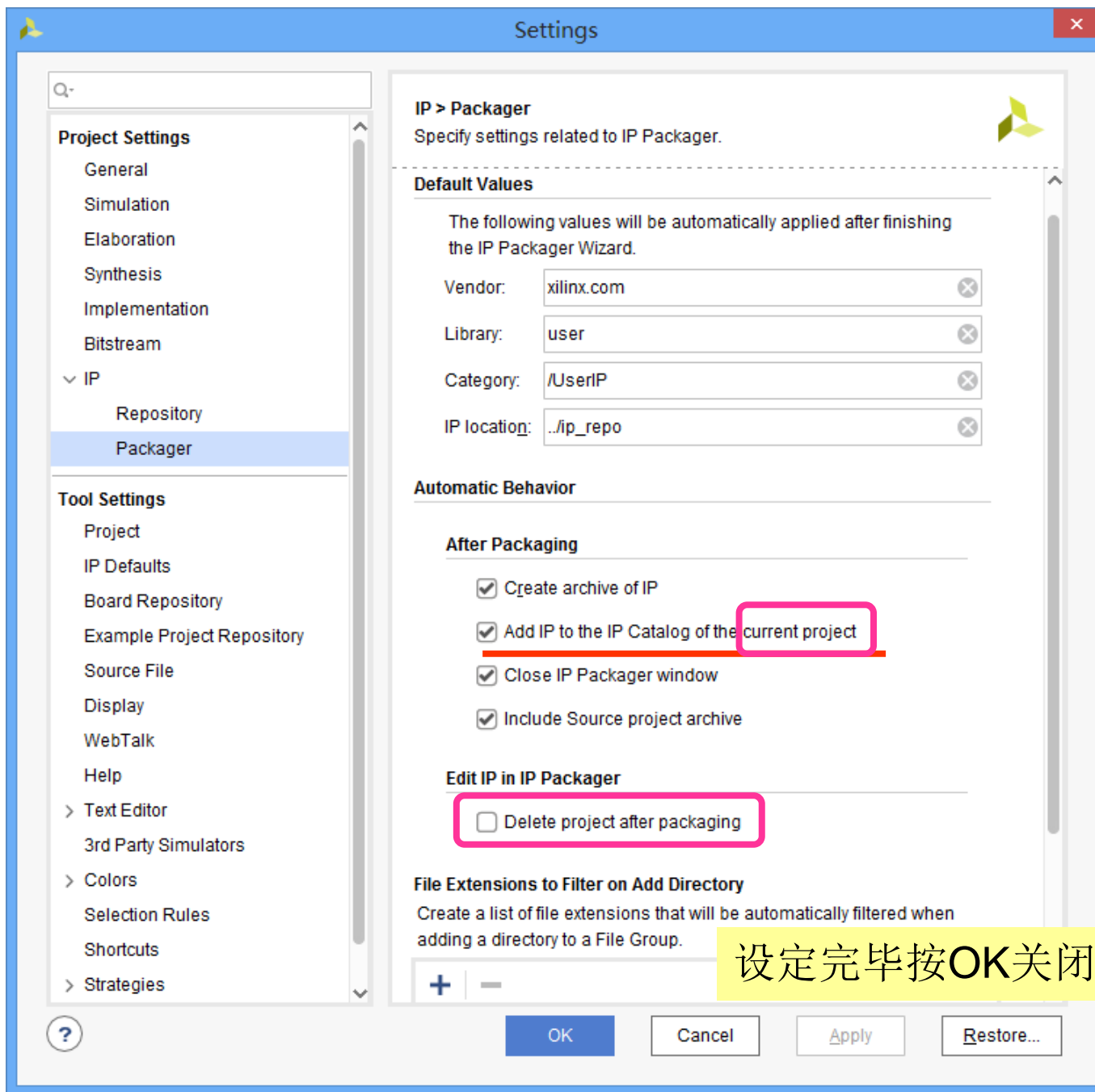


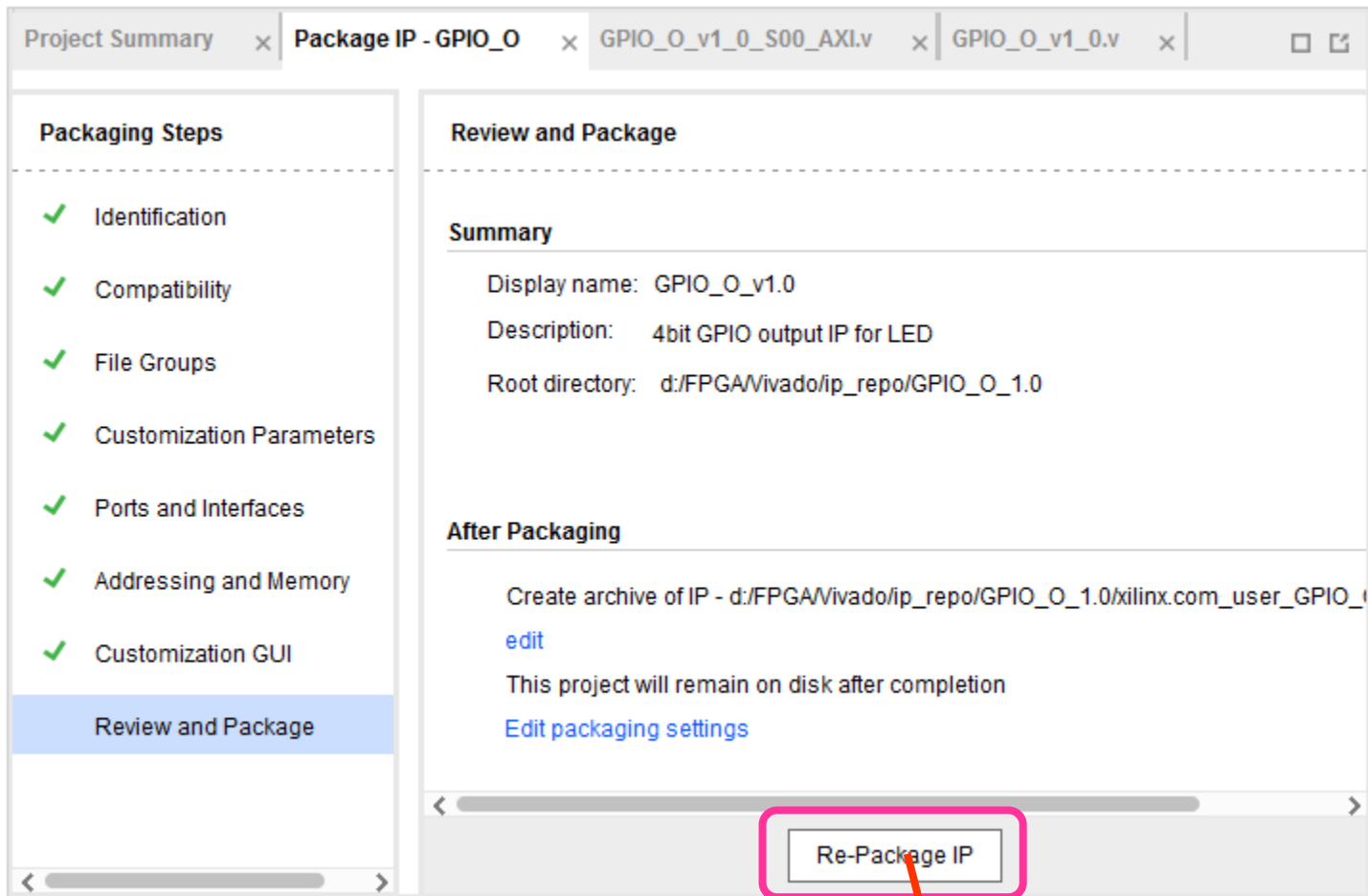
没有对号这些项是改动过的，点击每一项，右侧会出现类似Merge changes from File Groups Wizard的提醒，点击这些项，刷新信息。

The screenshot displays the Xilinx Project Manager interface for editing a project named 'GPIO_O_v1_0'. The interface is divided into several panes:

- Sources:** Shows the project hierarchy with 'Design Sources (2)', 'Constraints', 'Simulation Sources (1)', and 'Utility Sources'. The 'GPIO_O_v1_0' source is expanded, showing 'GPIO_O_v1_0_S00_AXI_inst: GPIO_O_v1_0_S00_AXI' and 'IP-XACT (1)'.
- Source File Properties:** Shows the properties for the selected source file 'GPIO_O_v1_0.v'. It is 'Enabled' and located at 'd:/FPGA/Vivado/ip_repo/'.
- Project Summary:** Shows the 'Packaging Steps' and 'Review and Package' sections. The 'Packaging Steps' list includes: Identification, Compatibility, File Groups, Customization Parameters, Ports and Interfaces, Addressing and Memory, Customization GUI, and 'Review and Package' (which is highlighted).
- Review and Package:** Contains a 'Summary' section with the following information:
 - Display name: GPIO_O_v1.0
 - Description: 4bit GPIO output IP for LED
 - Root directory: d:/FPGA/Vivado/ip_repo/GPIO_O_1.0Below the summary is an 'After Packaging' section with the text: 'Create archive of IP - d:/FPGA/Vivado/ip_repo/GPIO_O_1.0/xilinx.com_user_GPIO_O_1.0'. A button labeled 'Edit packaging settings' is highlighted with a red rectangle. At the bottom right, there is a 'Re-Package IP' button.

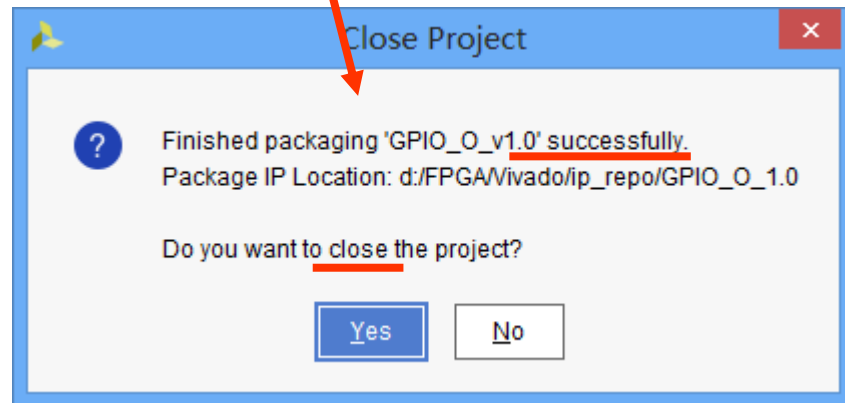
回到最后的Review and Package项里，默认是完成之后删掉工程，点击下面的Edit packaging settings，编辑选项





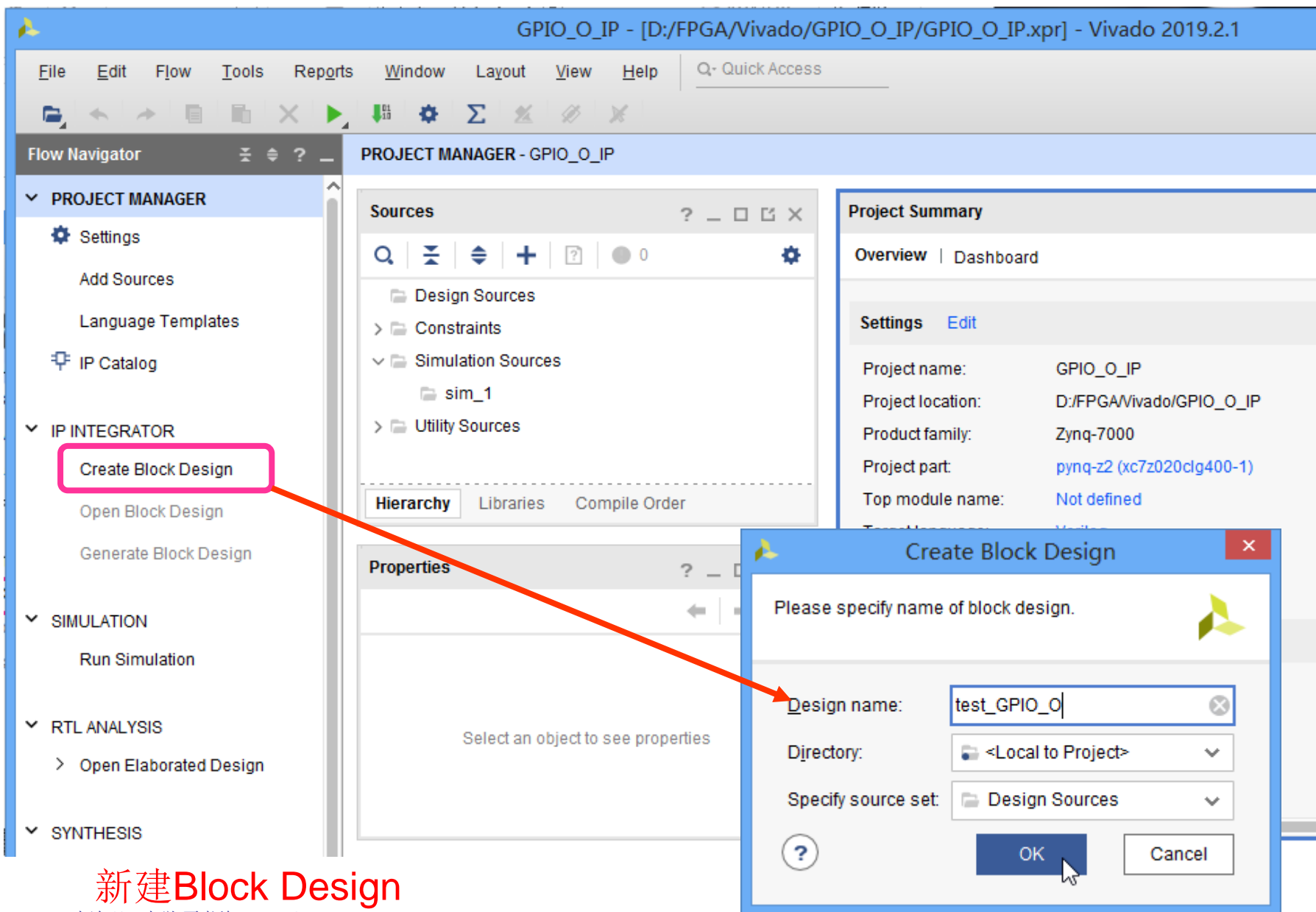
最后点击Re-Package IP，打包IP，放入当前工程的IP库。

确认后会关闭IP工程。



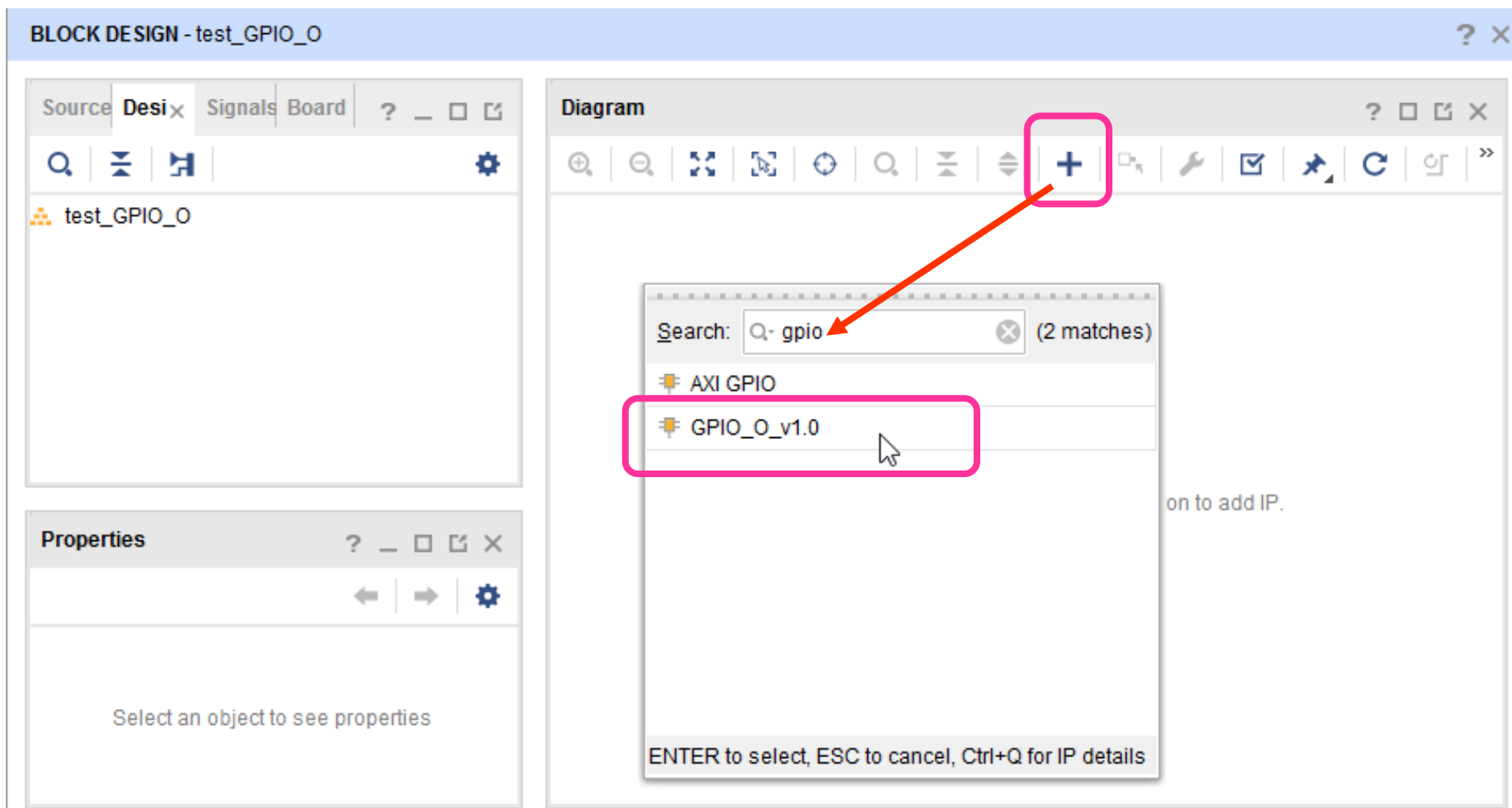
测试自制IP

关掉了修改IP代码的工程，就回到了最初新建的空白工程。



新建Block Design

大连理工大学.夏书峰.dutxia@dlut.edu.cn



在Diagram窗口点击+号，搜索关键字GPIO，会找到刚才定制的IP，双击该IP，添加进块图。

BLOCK DESIGN - test_GPIO_O

Source De x Signal Board ? _ □ □

test_GPIO_O

- > GPIO_O_0 (GPIO_O_v1.0:1.0)
- > processing_system7_0 (ZYNQ7 Process

Properties

Select an object to see properties

Diagram x Address Editor x ? □ □

Designer Assistance available Run Block Automation Run Connection Automation

processing_system7_0

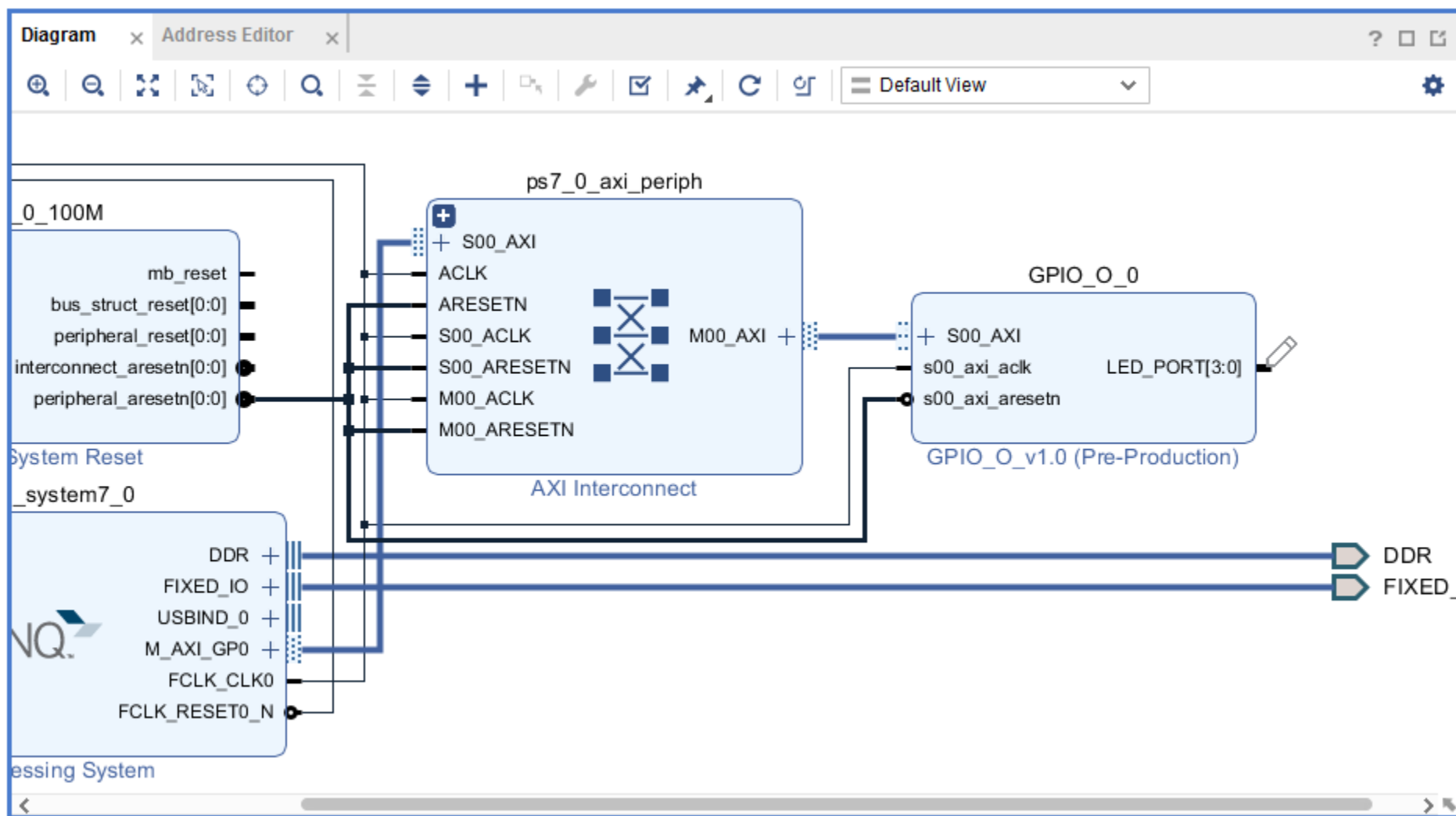
ZYNQ

ZYNQ7 Processing System

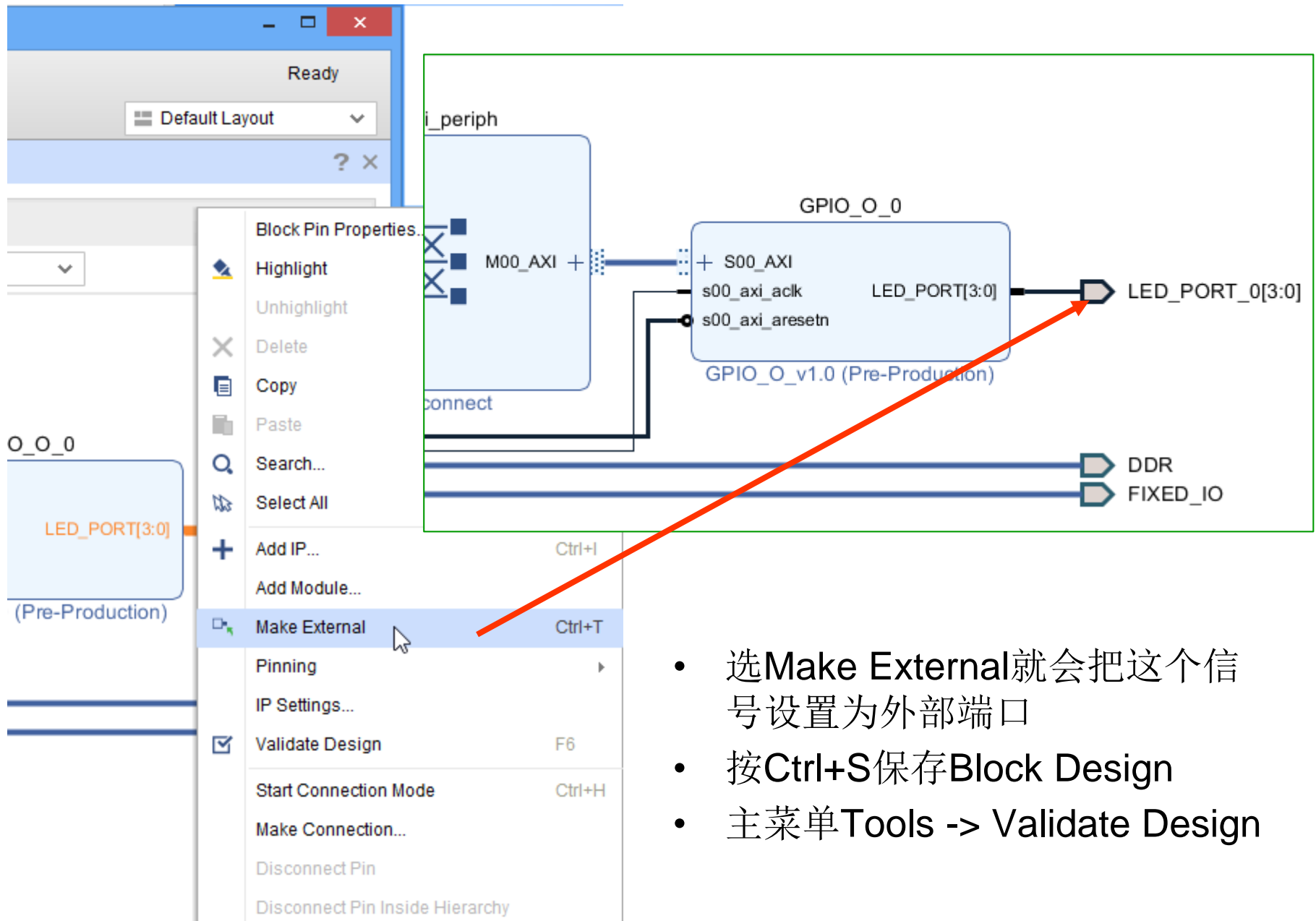
GPIO_O_0

GPIO_O_v1.0 (Pre-Production)

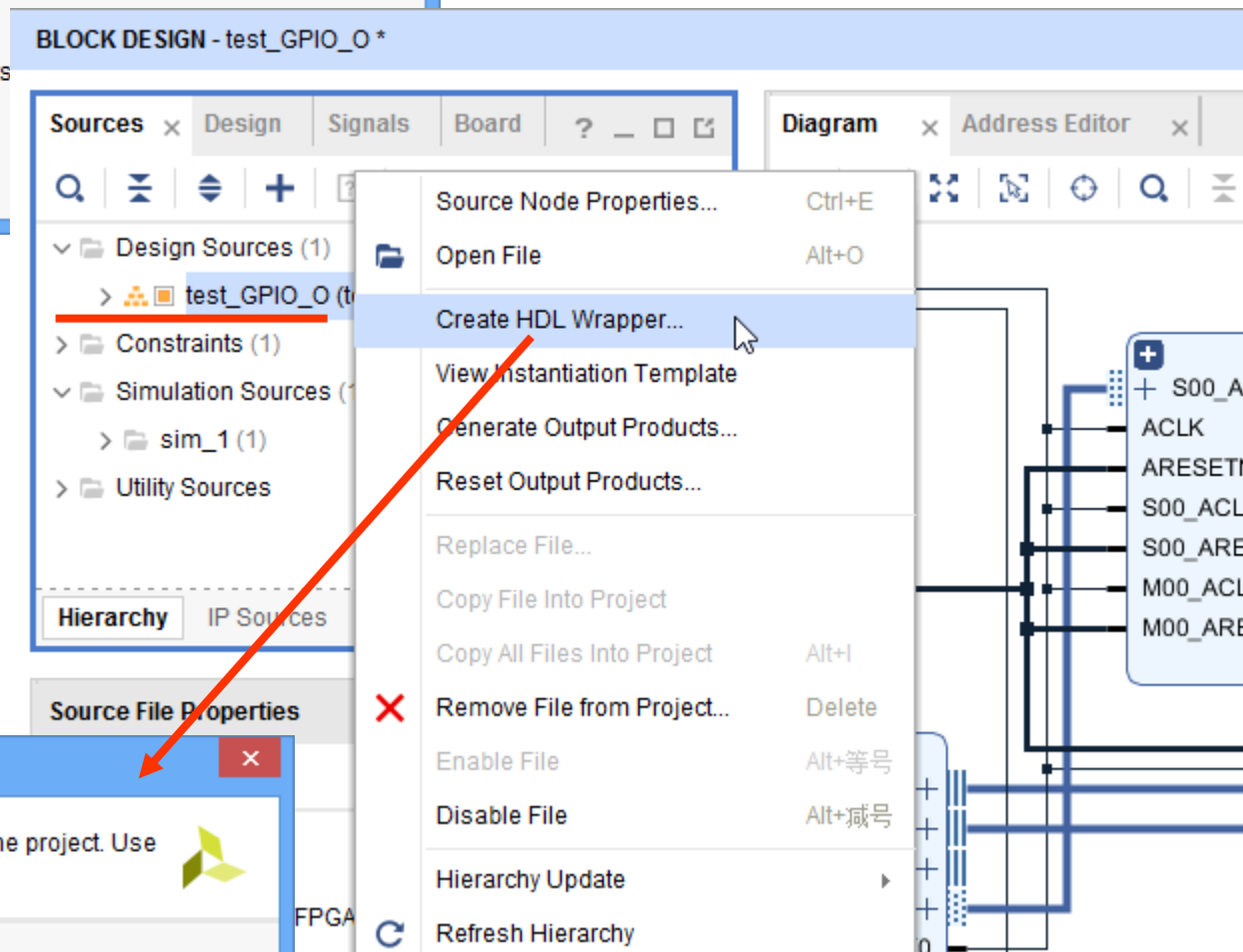
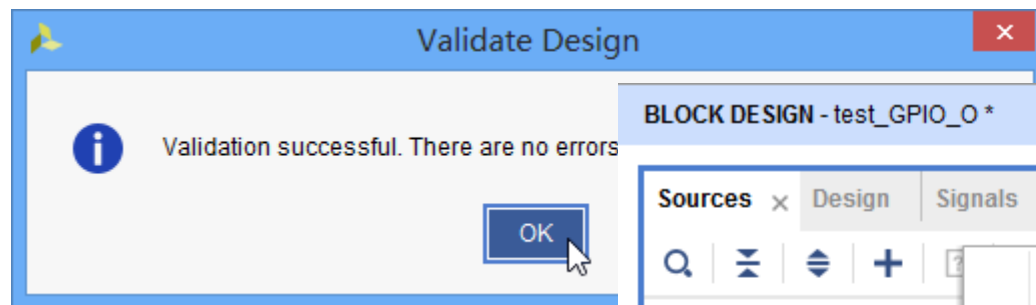
- 继续添加ZYNQ7 IP核
- 点击Run Block Automation
- 点击Run Connection Automation



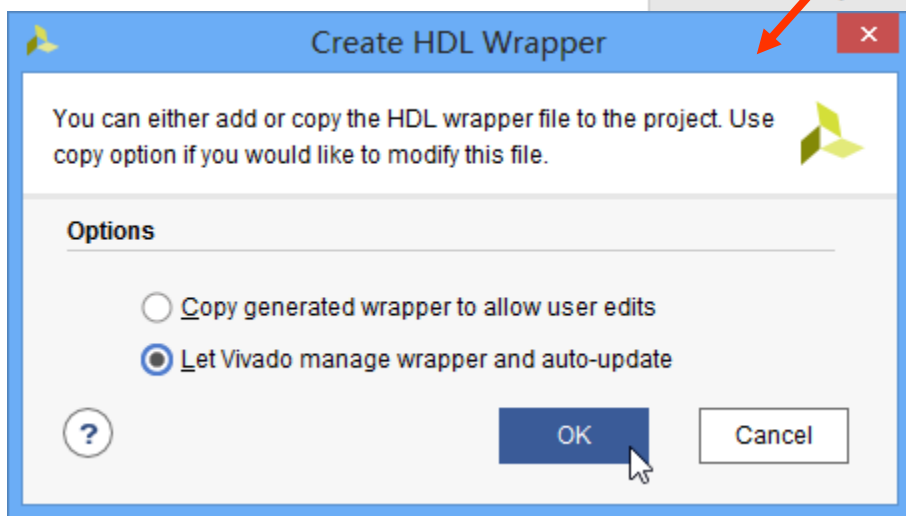
鼠标悬浮在自定义IP外部端口的线头上，光标变成笔，点鼠标右键

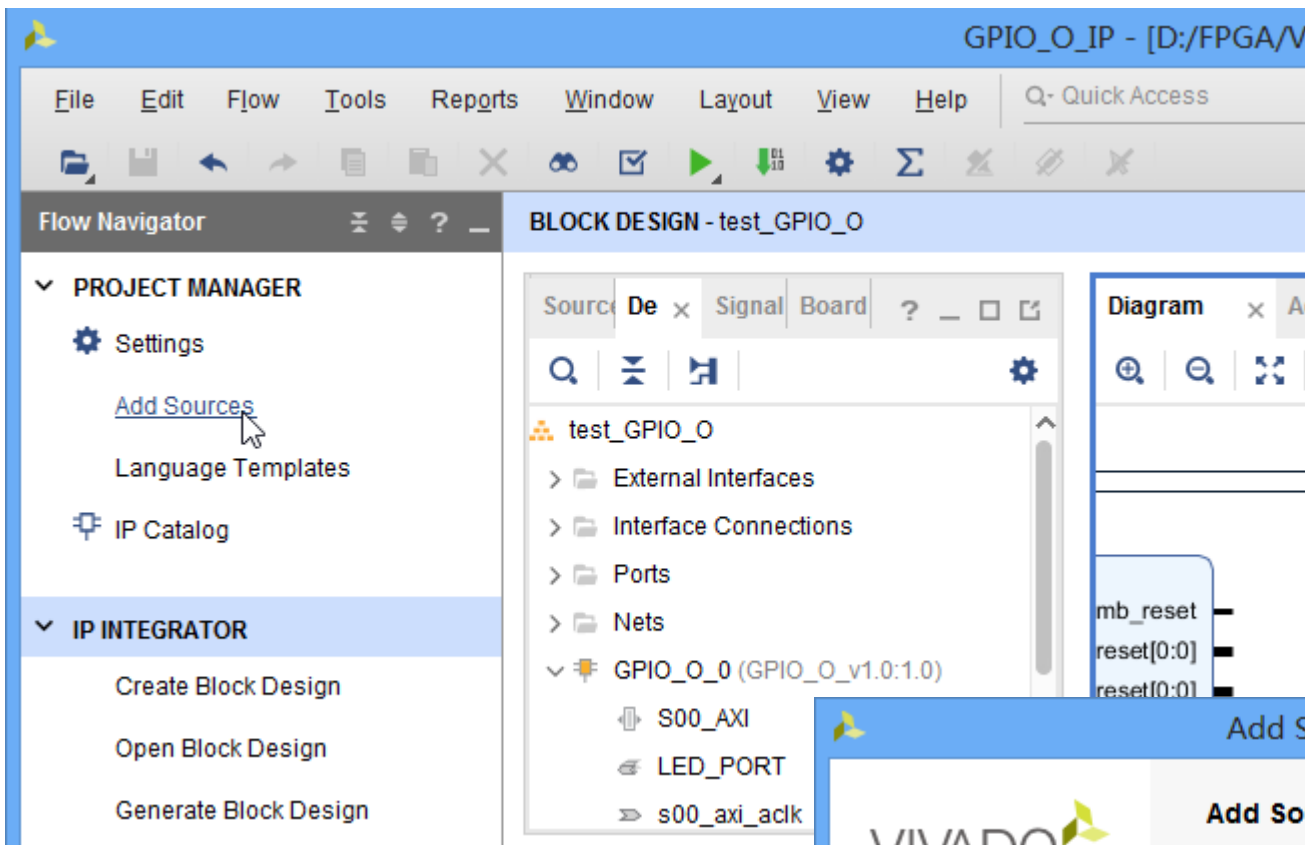


- 选Make External就会把这个信号设置为外部端口
- 按Ctrl+S保存Block Design
- 主菜单Tools -> Validate Design



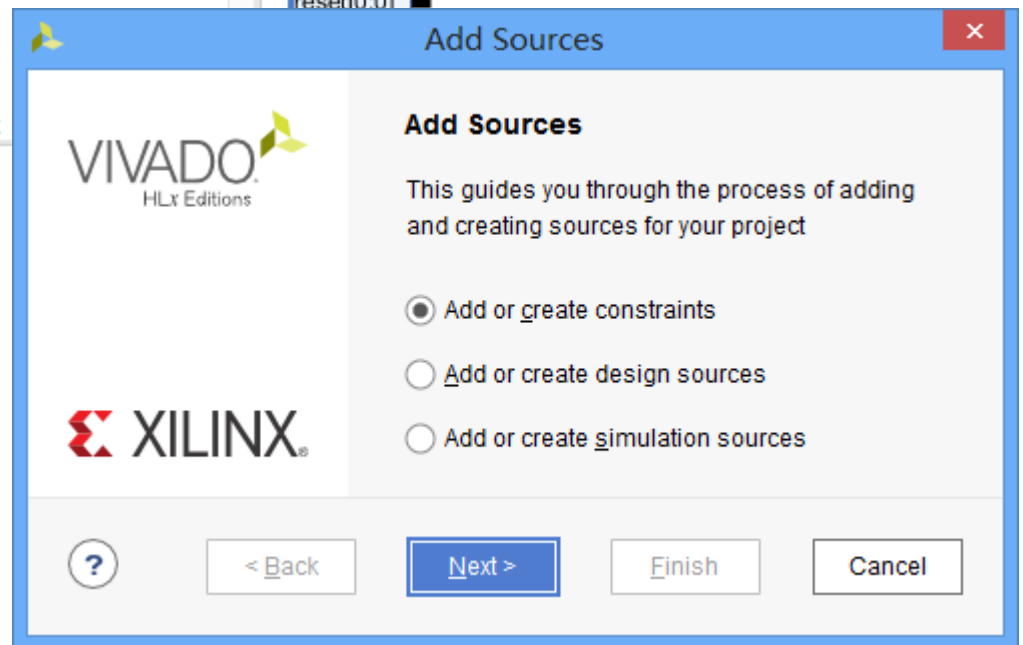
到Source标签，右键
点击块图文件，选
Create HDL Wrapper，

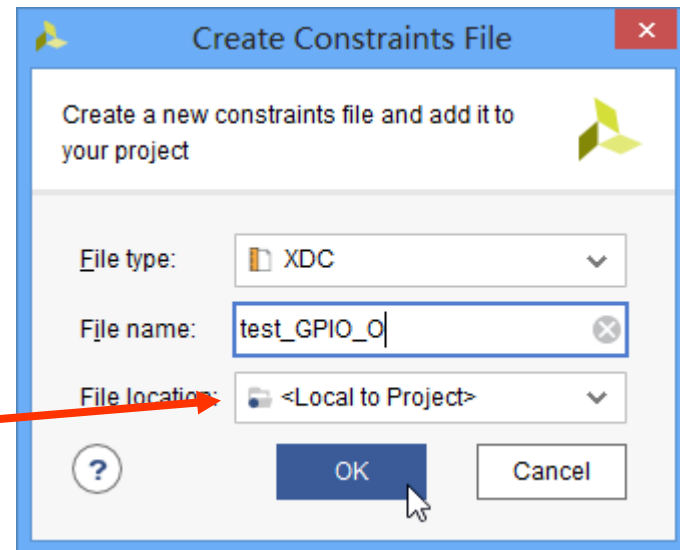
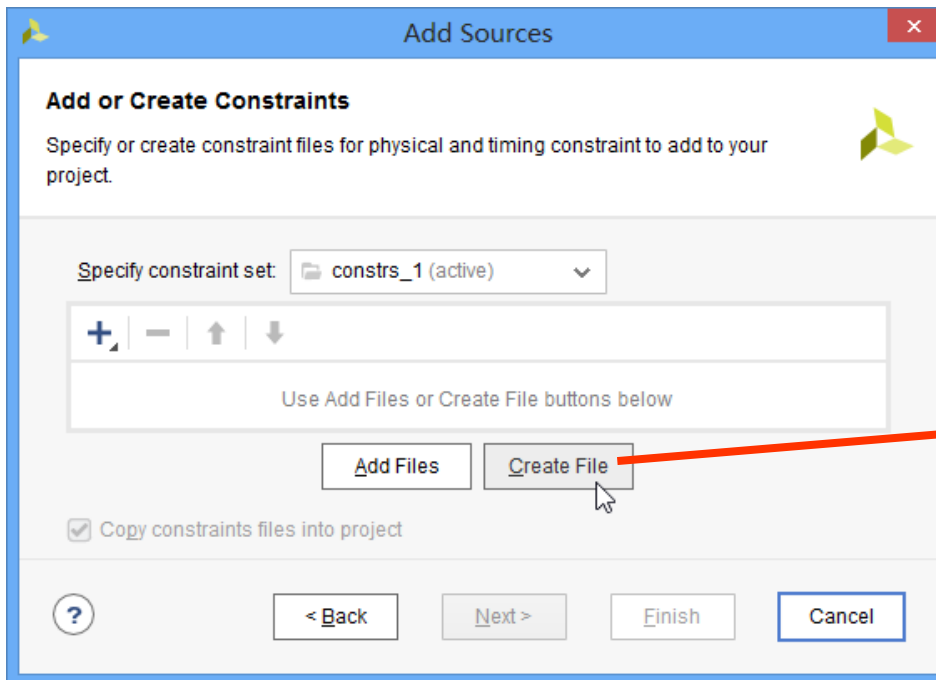




因为板卡文件里没有这个新建IP的信息，因此外部端口连接需要通过约束文件指定。

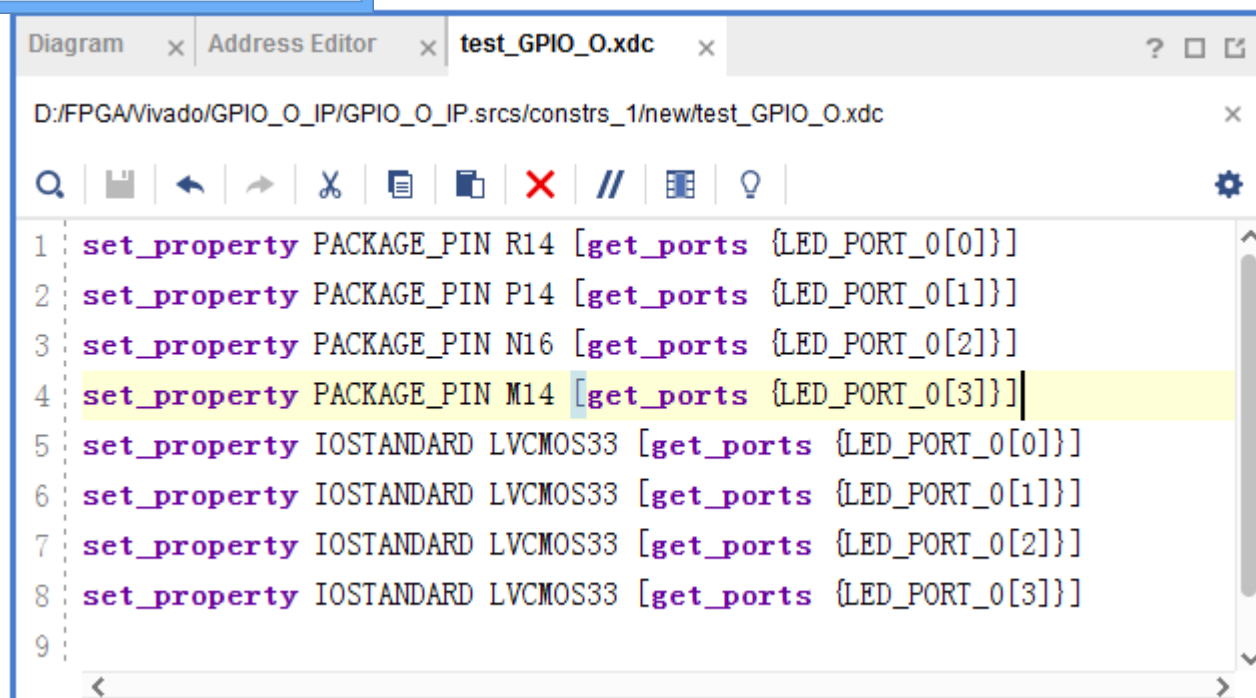
点击Add Source新建约束文件。

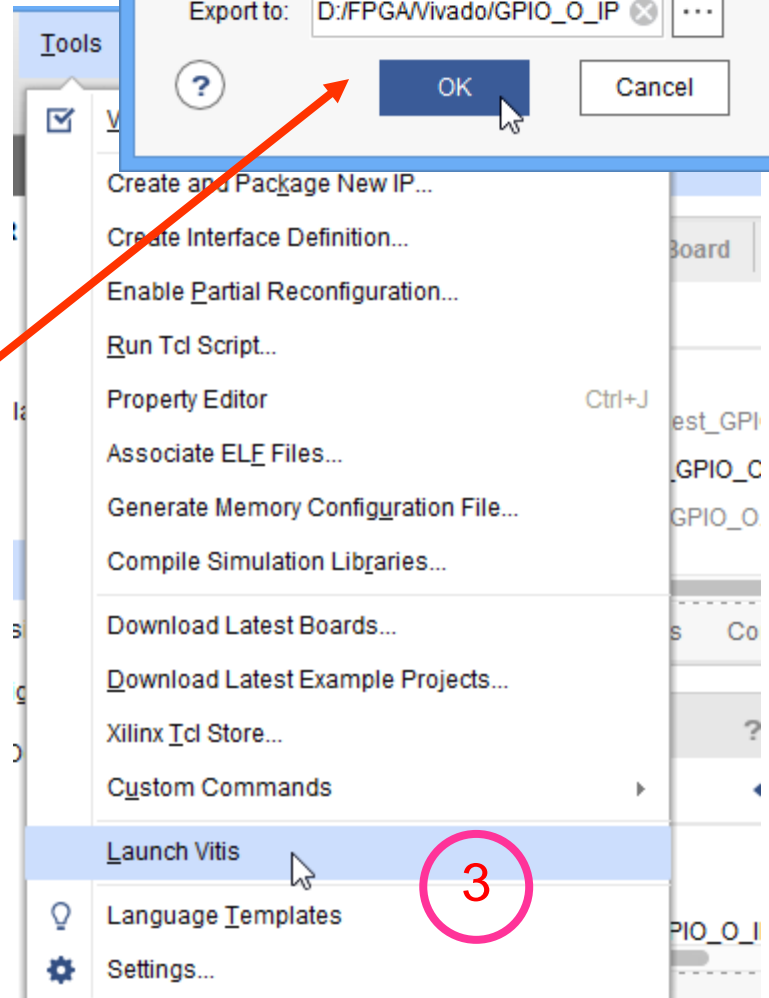
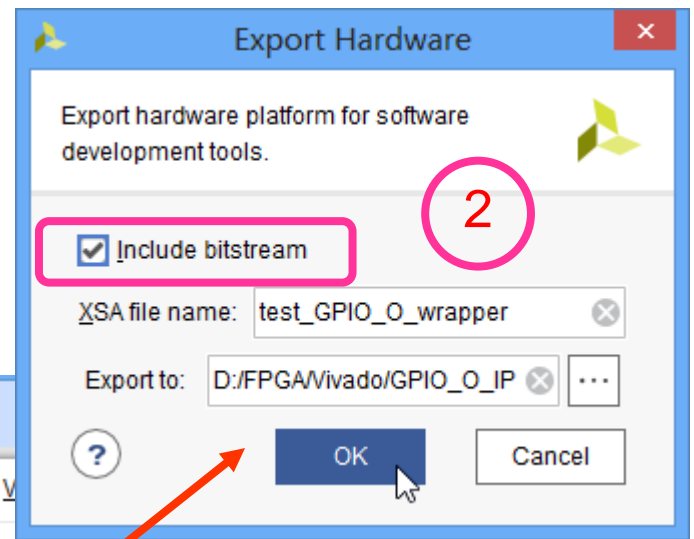
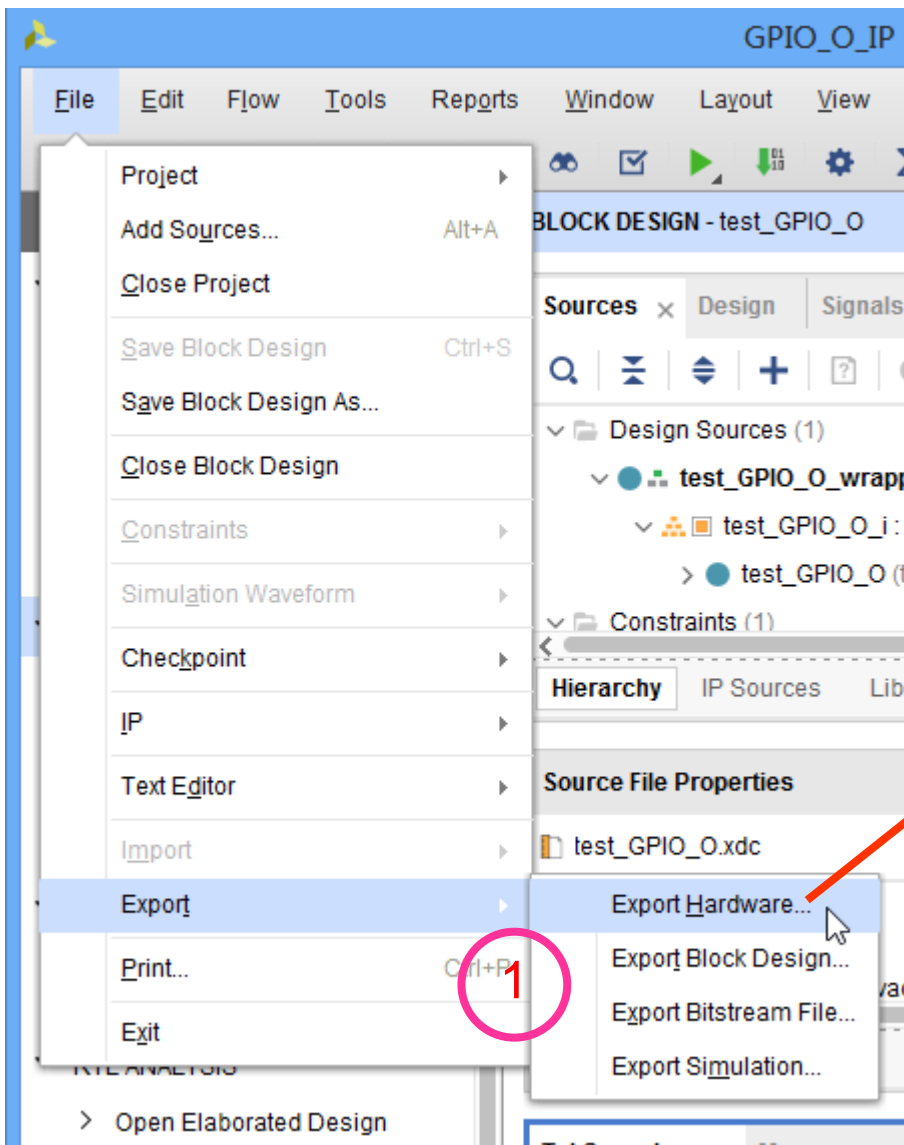




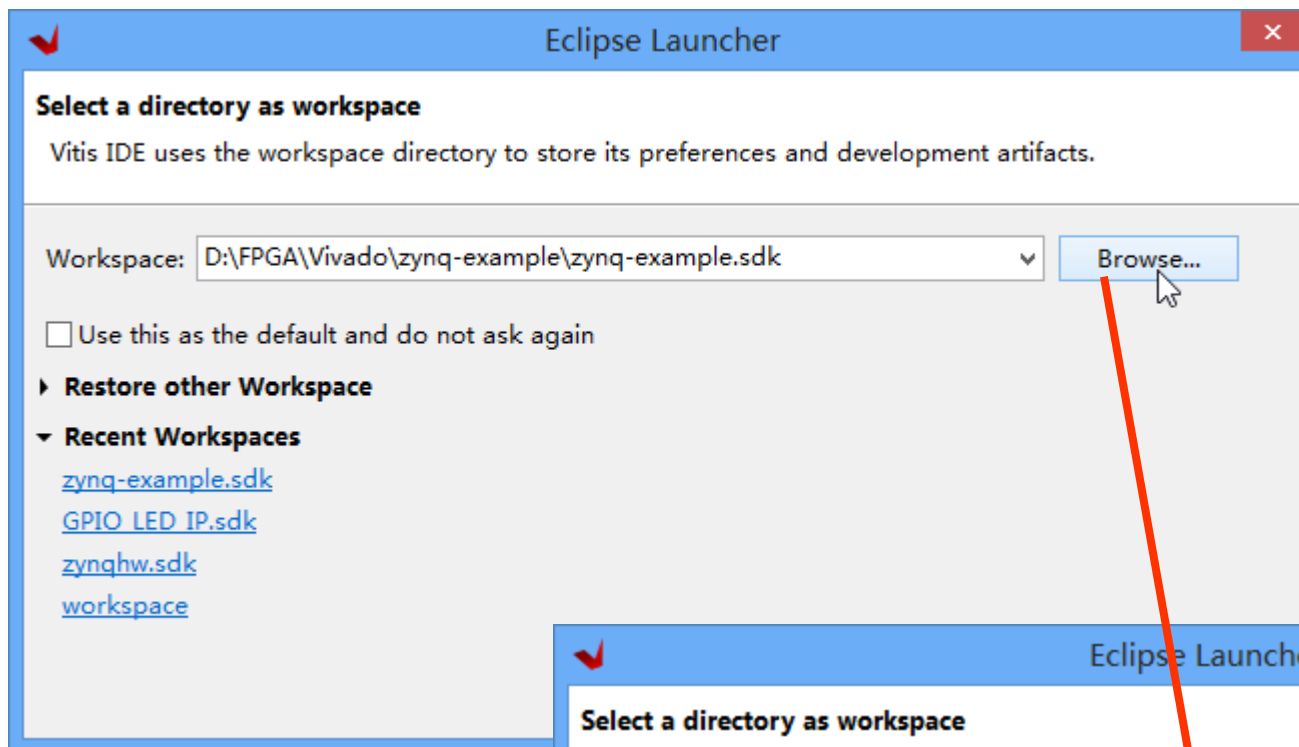
新建约束文件，编辑
内容如右图，保存。
其它约束信息在板卡
文件里都有了。

去左侧Flow栏下方点
击Generate Bitstream，
完成硬件流程。

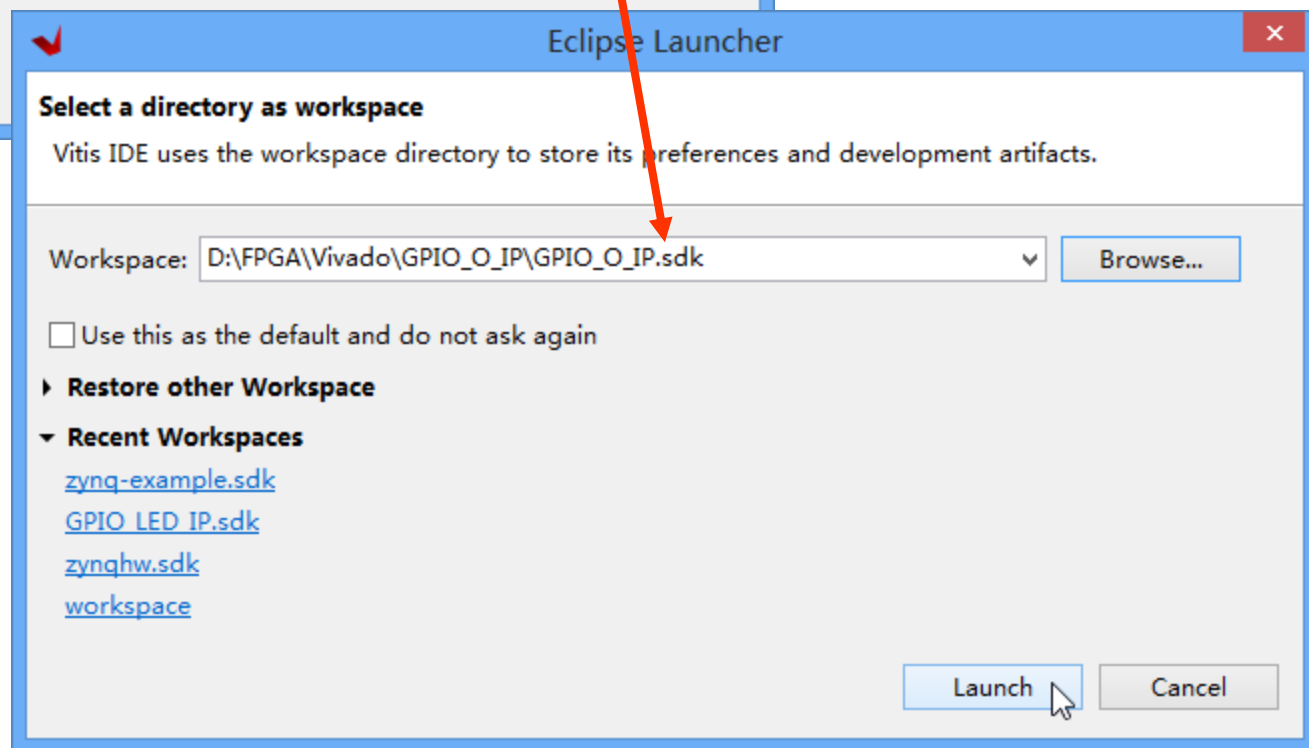


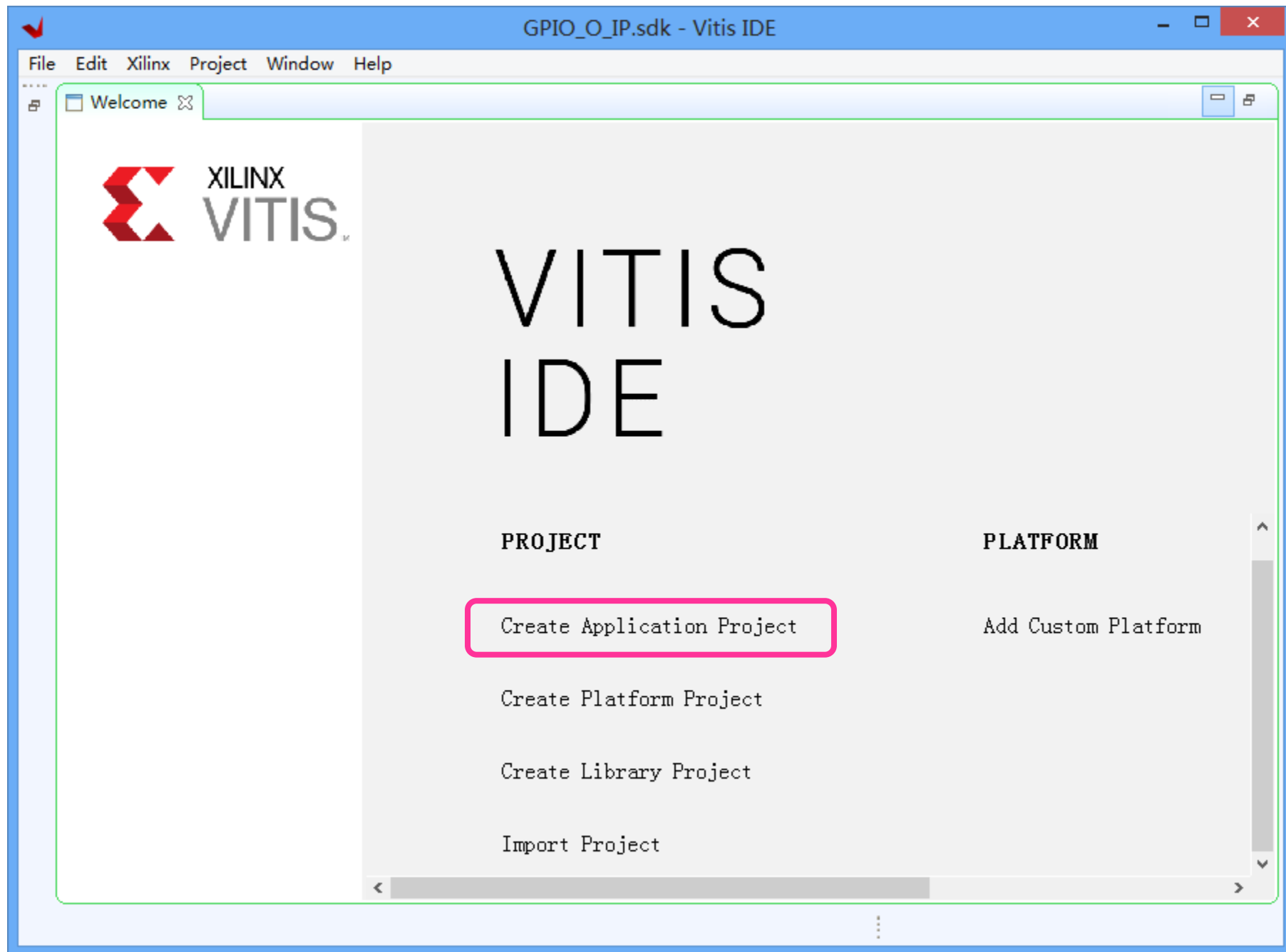


- 导出硬件系统文件XSA
- Tools -> Launch Vitis启动Vitis



要把vitis的文件放在
硬件工程的目录里，
比如单独给建个
SDK目录。





New Application Project

Create a New Application Project

Enter a name for your Application project.

Project name:

☒ Use default location

Location:

Choose file system:

System project:

System Project 1

Baremetal App

System Project 2

Linux App 1

Linux App 2

AI Engine App

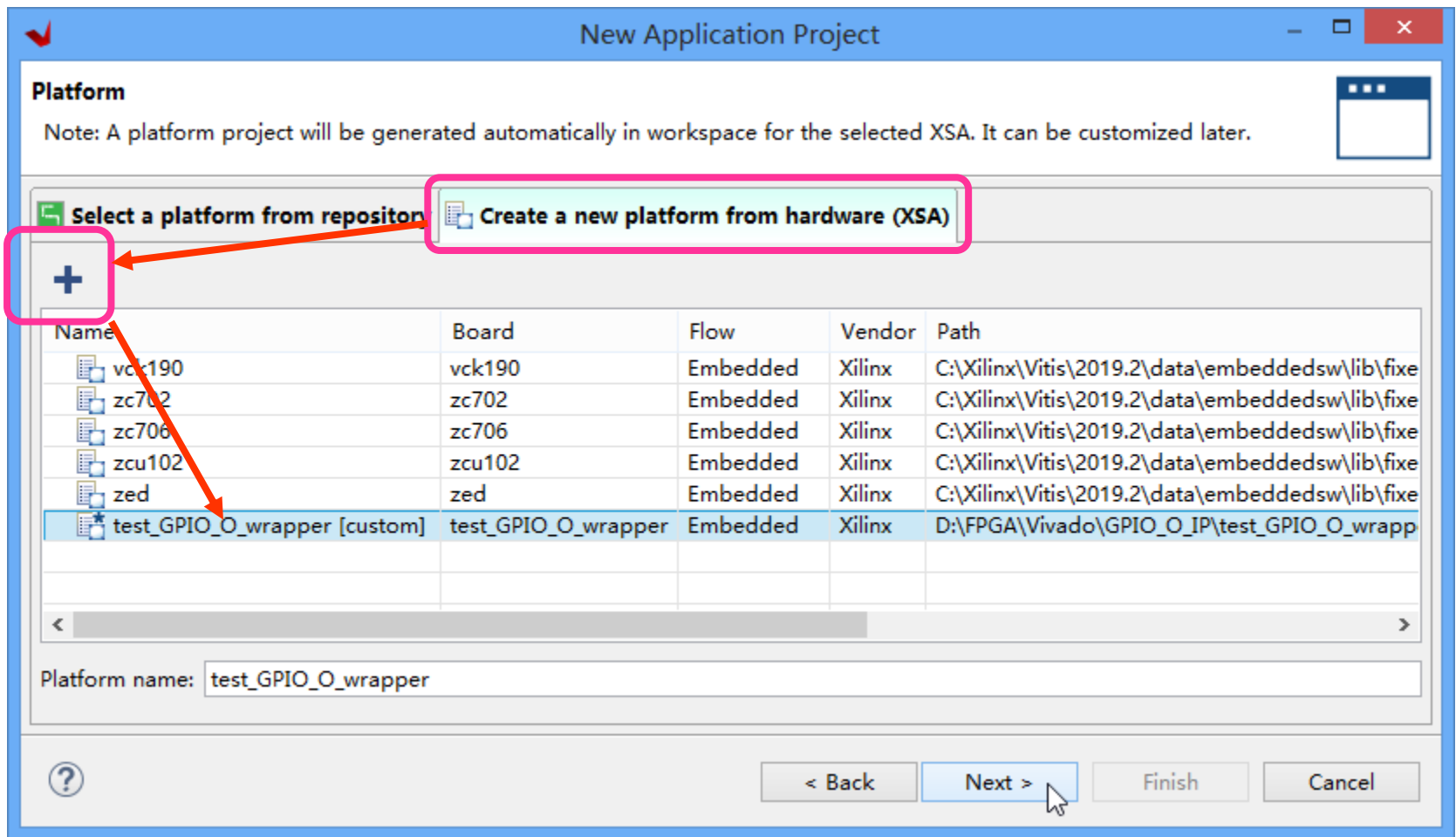
A72_0 Baremetal Domain

A72 Linux Domain

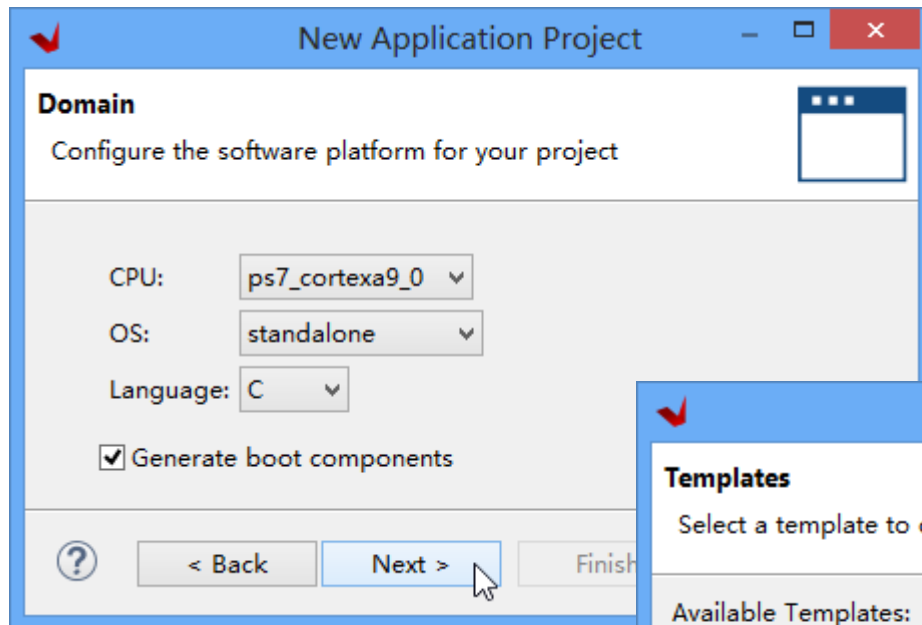
AI Engine Domain

Platform

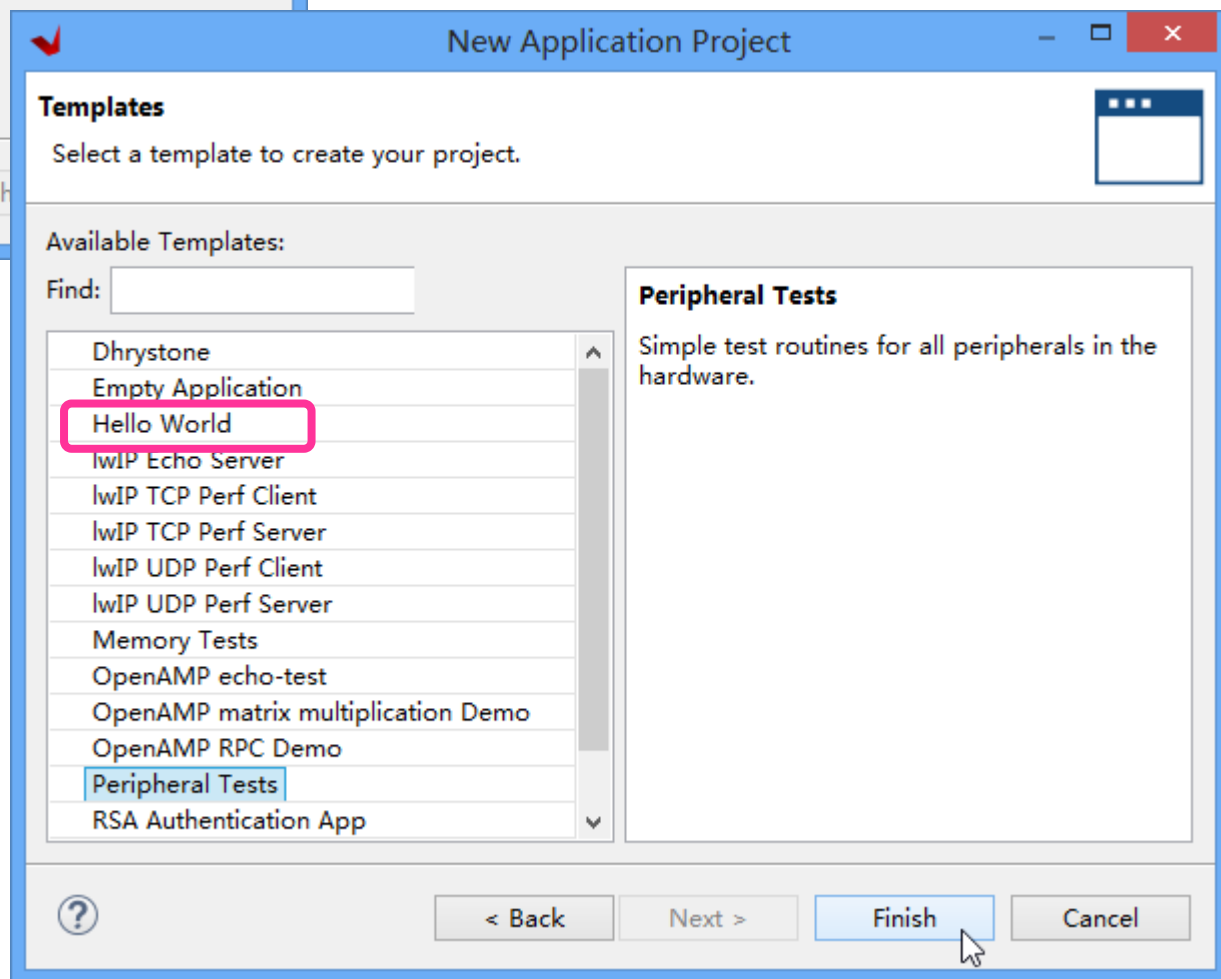
- A system project is a container for multiple applications that would run on different domains of a platform at the same time.
- A domain is the BSP/OS that controls one or more isomorphic processors.
- A platform contains one or more domains.
- A workspace can contain unlimited platforms and unlimited system projects



- 选Create a new platform from hardware (XSA)
- 点+号，浏览到之前硬件工程目录，选择之前导出的xsa文件
- 选择该条目，Next。



选择使用的处理器核，
选择一个例程，或者空白工程（之后要手工添加main.c），Finish。



在main.c里面添加如下代码，测试GPIO_LED是否正常工作

```
#include "xil_io.h"
```

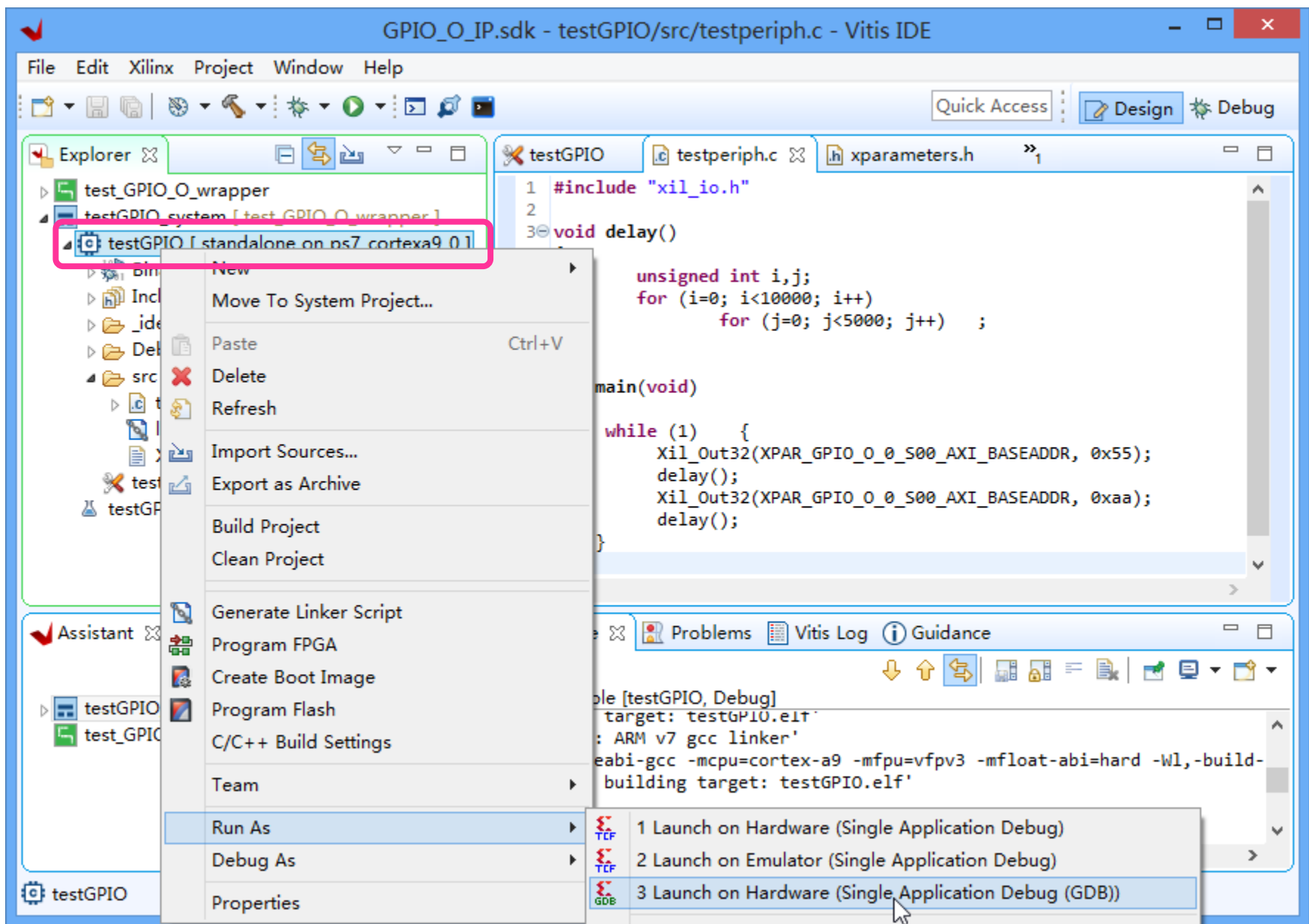
```
void delay()
```

```
{  
    unsigned int i,j;  
    for (i=0; i<10000; i++)  
        for (j=0; j<5000; j++) ;  
}
```

```
int main(void)
```

```
{  
    while (1) {  
        Xil_Out32(XPAR_GPIO_O_0_S00_AXI_BASEADDR, 0x55);  
        delay();  
        Xil_Out32(XPAR_GPIO_O_0_S00_AXI_BASEADDR, 0xaa);  
        delay();  
    }  
}
```

去主菜单 Project -> Build Project 编译工程



如图鼠标右键菜单，在FPGA上运行测试程序，看LED交替闪烁，IP工作。

参考资料

- The Zynq Book, Louise H.Crockett, Ross A. Elliot等
- The_Zynq_Book_Tutorials, Louise H.Crockett, Ross A. Elliot等
- Zynq-7000 SoC Technical Reference Manual, Xilinx
- PYNQ_Z2_User_Manual_v1.1
- “依元素科技”公司系列在线直播PPT（bilibili）
<https://www.bilibili.com/video/BV1J54y1d7AV>

DONE!

更多IP核操作及驱动程序API编程方法
有待诸位探索