APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL Edition) Project 6 Use ZYNQ's PS to light up the LED light connected to the PL end AXI_GPIO_IP method (full picture and text)

The LED lights lit before are lit through PL resources, that is, through the logic of the FPGA to light the LED lights, this project will try to call ZYNQ's PS resources for the first time to achieve a point LED light operation, because the LED resources on the board are connected to the PL end, so here is a linkage, with PS resources to light the PL end LED lights

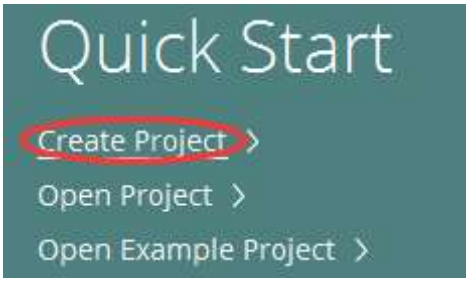
(This method will occupy PL resources, generally recommended to use EMIO method, will save resources)

This article is demonstrated on vivado2018.3, please research for other versions

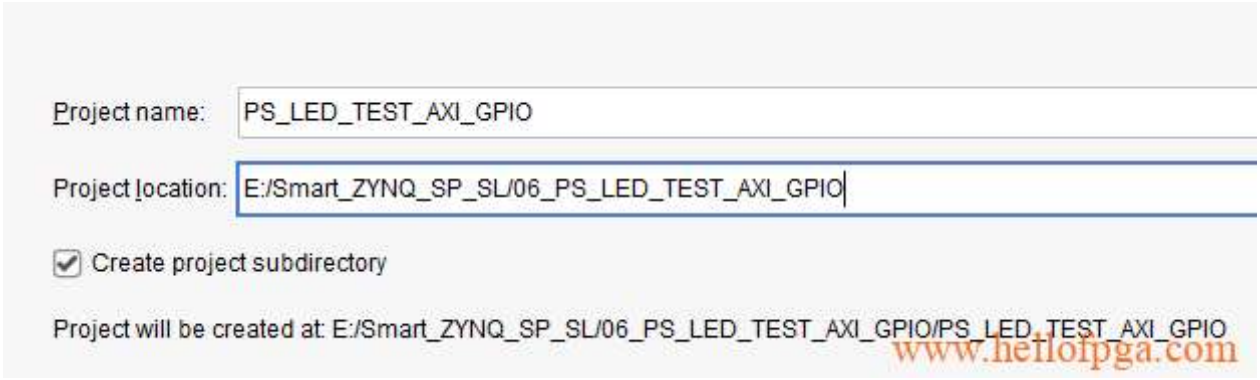
(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

1. Create a Vivado project

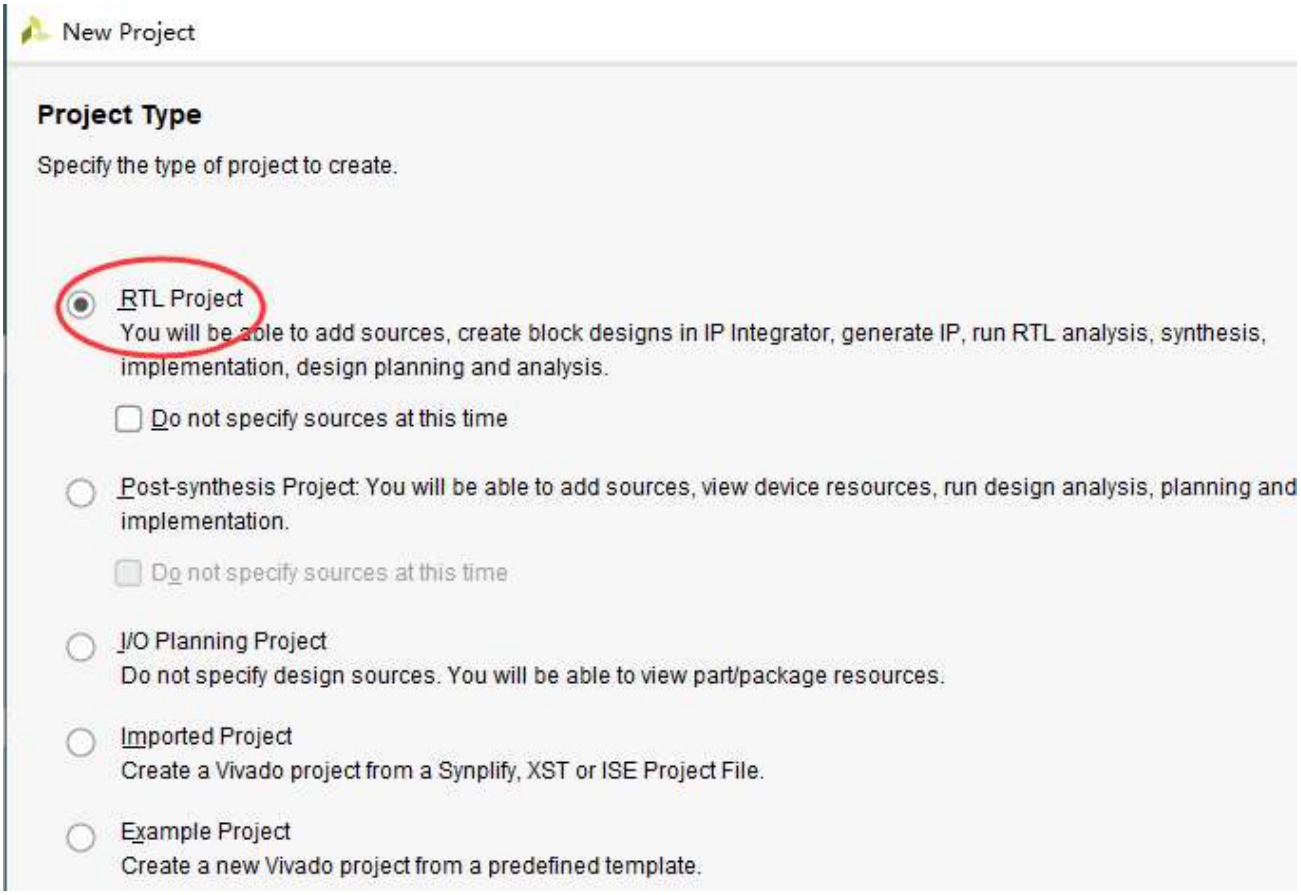
1) Specific steps to create a VIVADO project, open the software and select Create Project, as shown in the following figure



2) Click NEXT and enter the project name in the second dialog box "Project name" that appears; Select the save path in Project location; Check "Create project subdirectory" and click "Next" **Note, all paths cannot appear Chinese name**



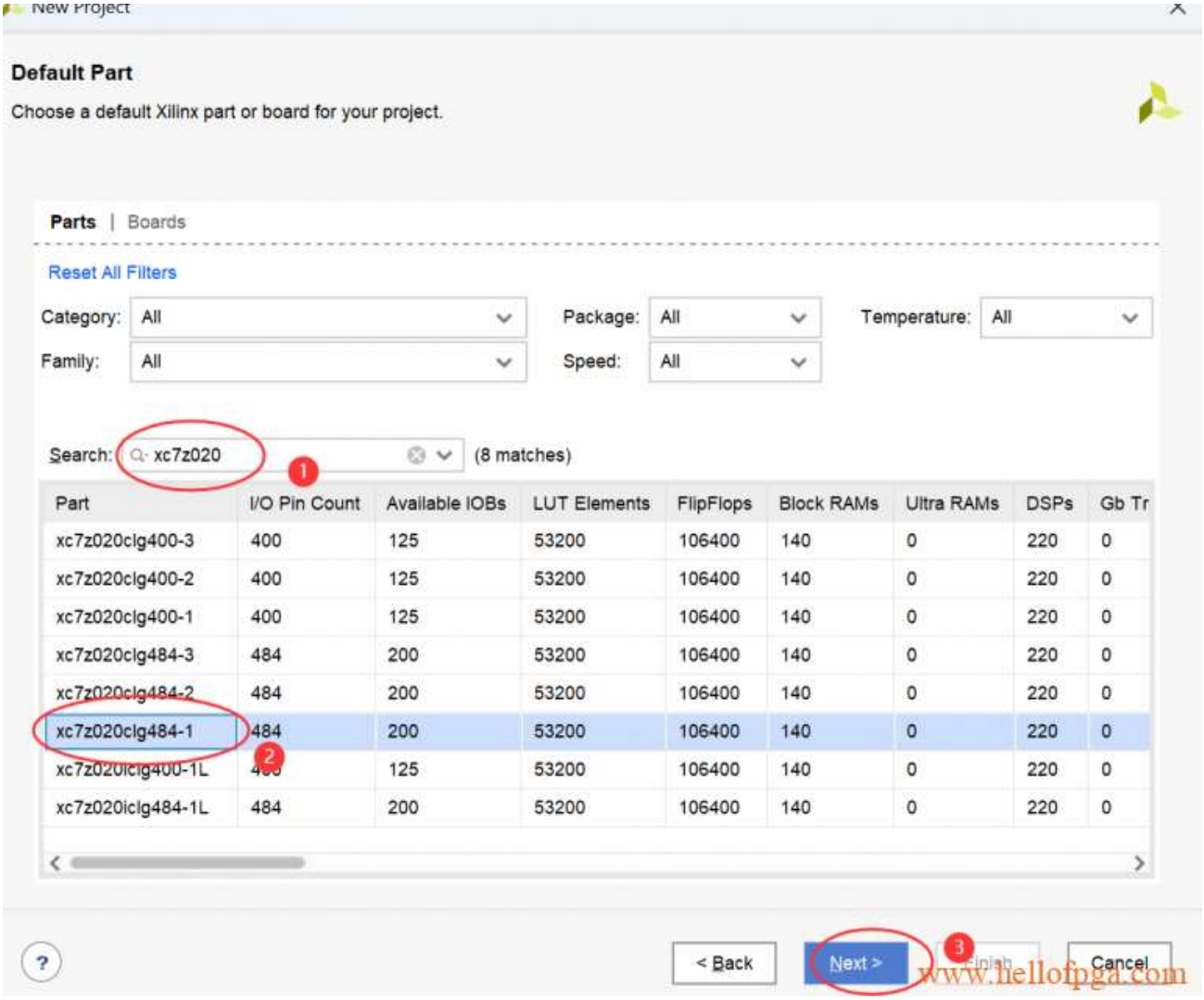
3) Click the RTL PROJECT option and click NEXT



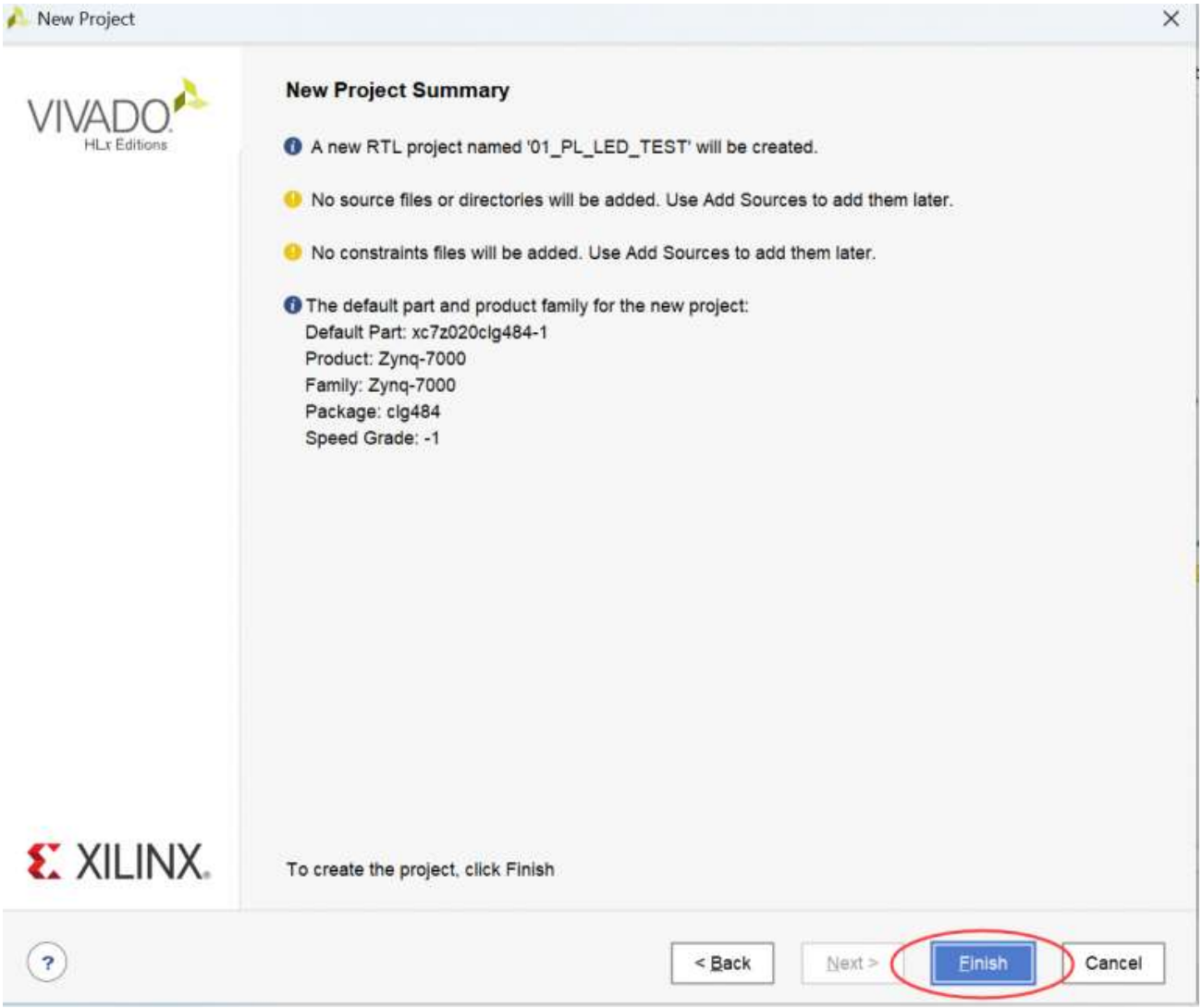
4) Step <>: The Add Sources option is left blank, NEXT

5) Step <>: The Add Constraints option is left blank, NEXT

6) Select the chip model XC7Z020CLG484-1

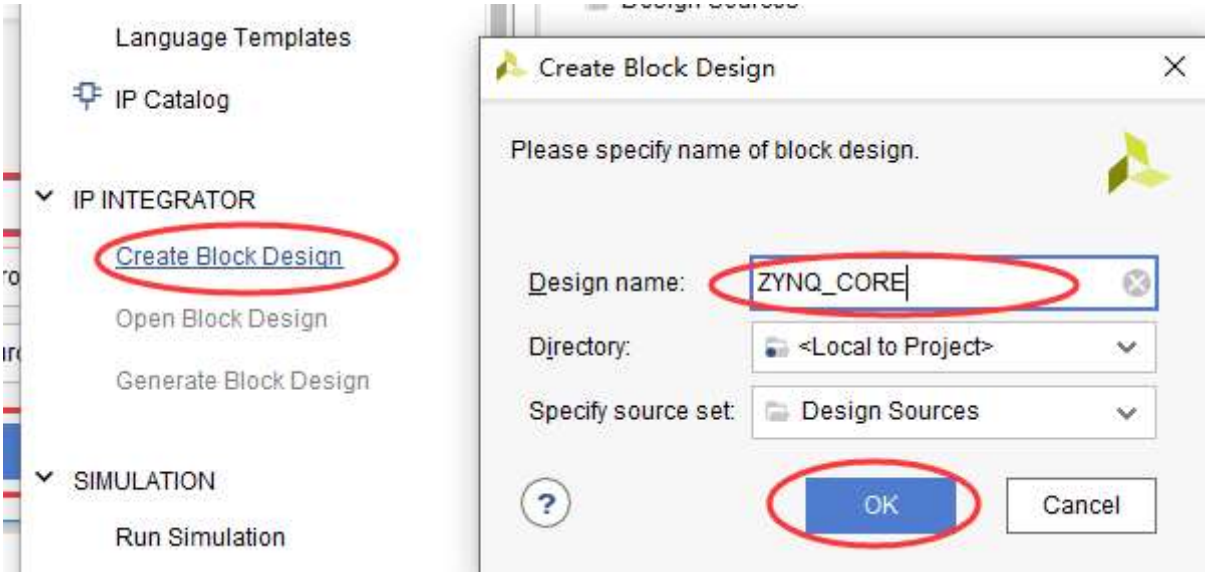


7) Confirm the selected information Click "Finish" to complete the project creation of vivado

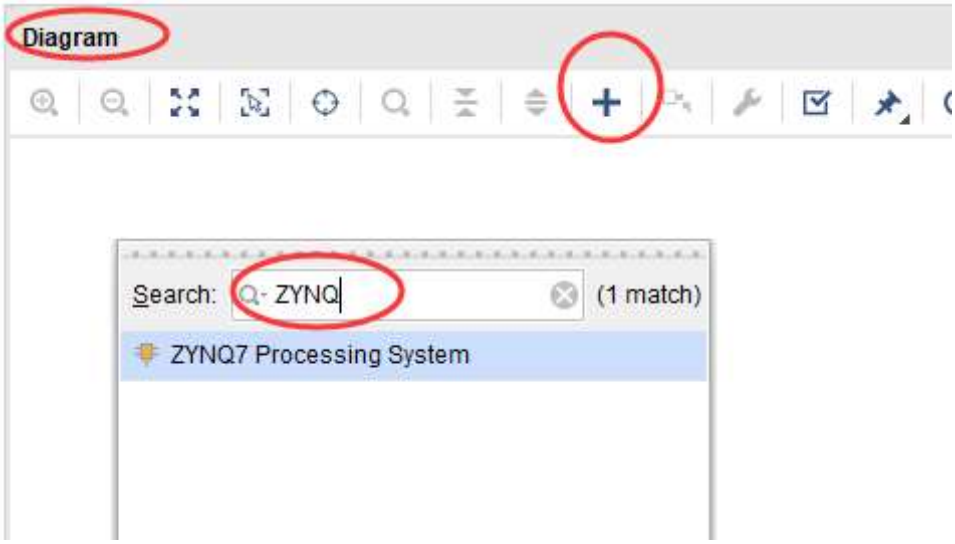


2 Create a BLOCK design

1) IP INTEGRATOR→Create Block Design, enter the design name in the pop-up dialog box, and finally click "OK", as shown in the figure below

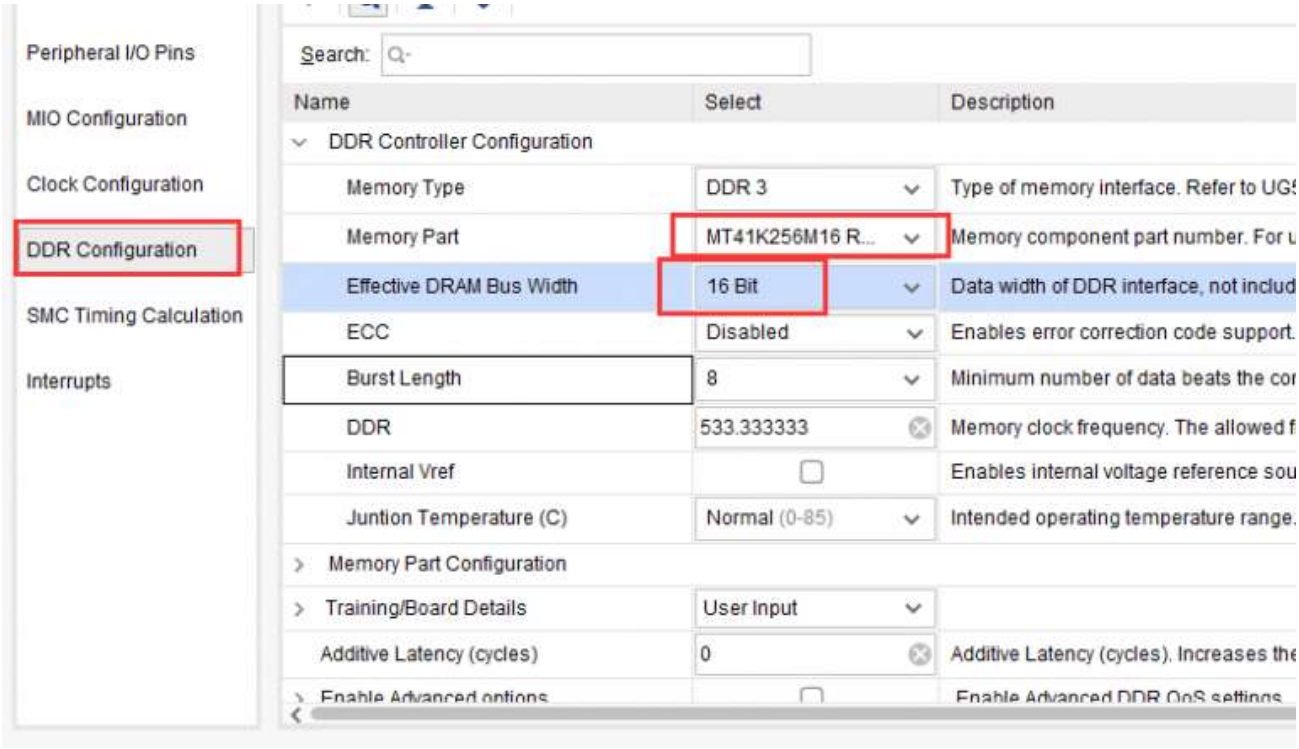


2) IN THE WINDOW ON THE RIGHT, CLICK THE PLUS SIGN, SEARCH FOR ZYNQ IN THE SELECTION BOX, AND FIND ZYNQ7 PROCESSING SYSTEM, DOUBLE-CLICK AND OPEN

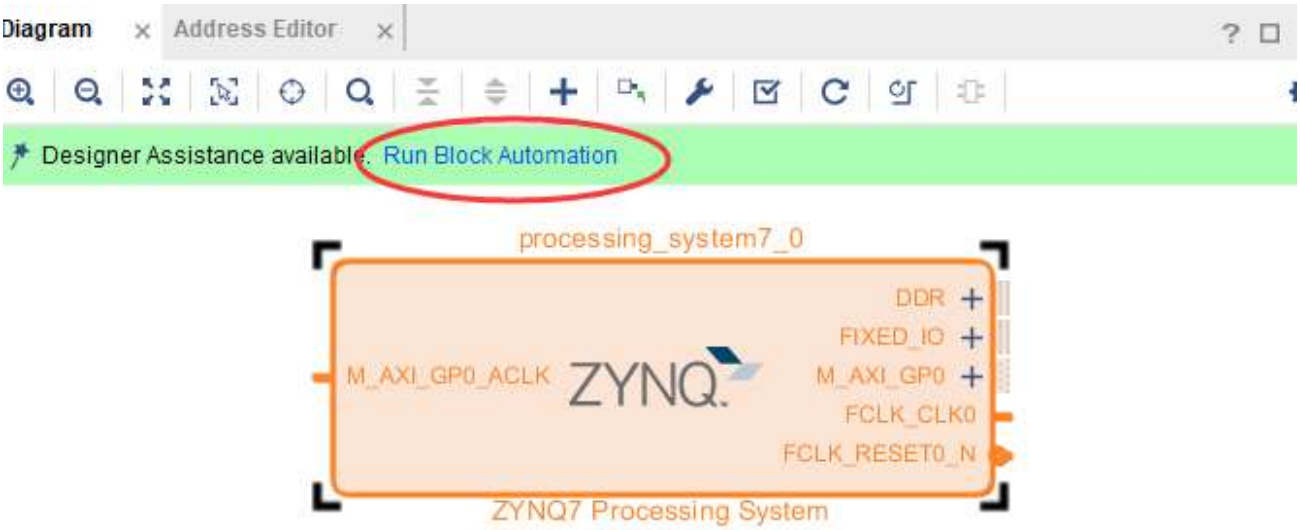


3) The software automatically generates a zynq block as shown in the figure below, next to do some corresponding settings, double-click the ZYNQ core in the figure below

4) Find DDR Configuration →DDR Controller Configuration →DDR3 in the pop-up window in turn, select the corresponding DDR3 according to the DDR on your board in the Memory Part drop-down menu, the model used in this experiment: MT41K256M16RE-125, select 16bit of data width and finally click "OK", as shown in the figure below.

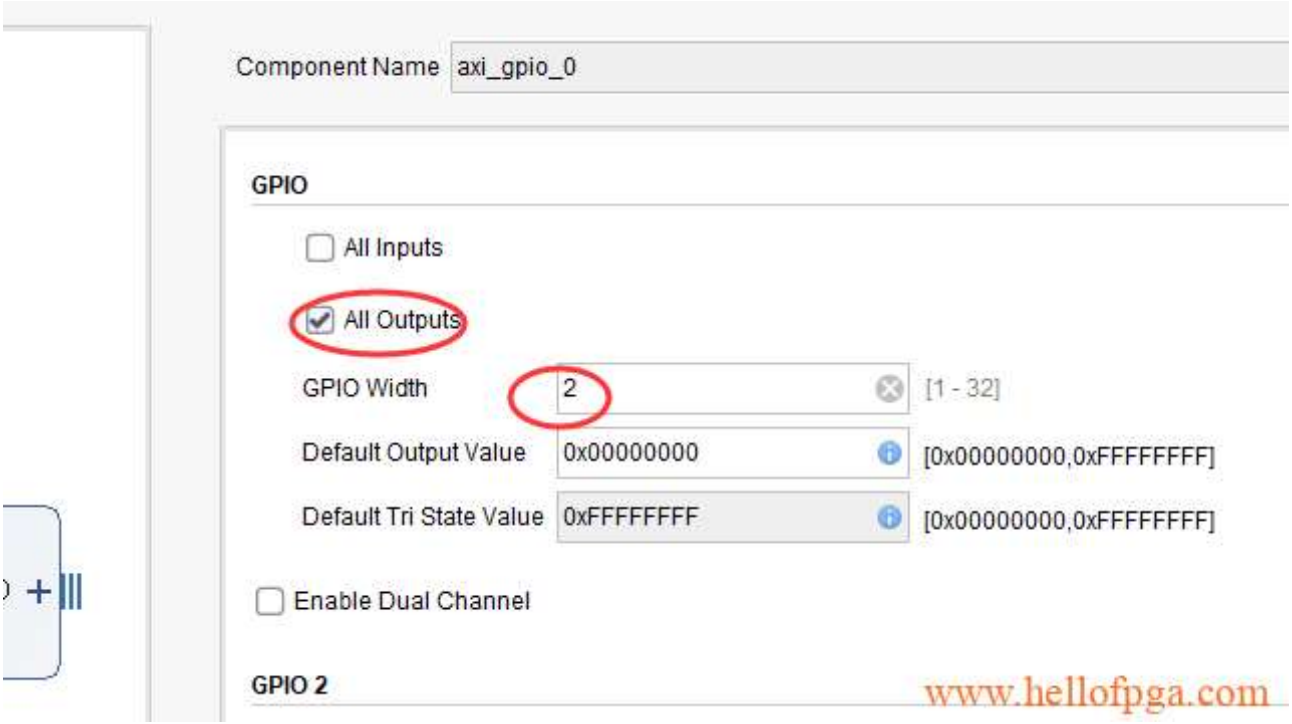


Finally, click on "Run Block Automation" as shown in the image below. Keep the default in the pop-up options and click "OK" to complete the configuration of the ZYNQ7 Processing System

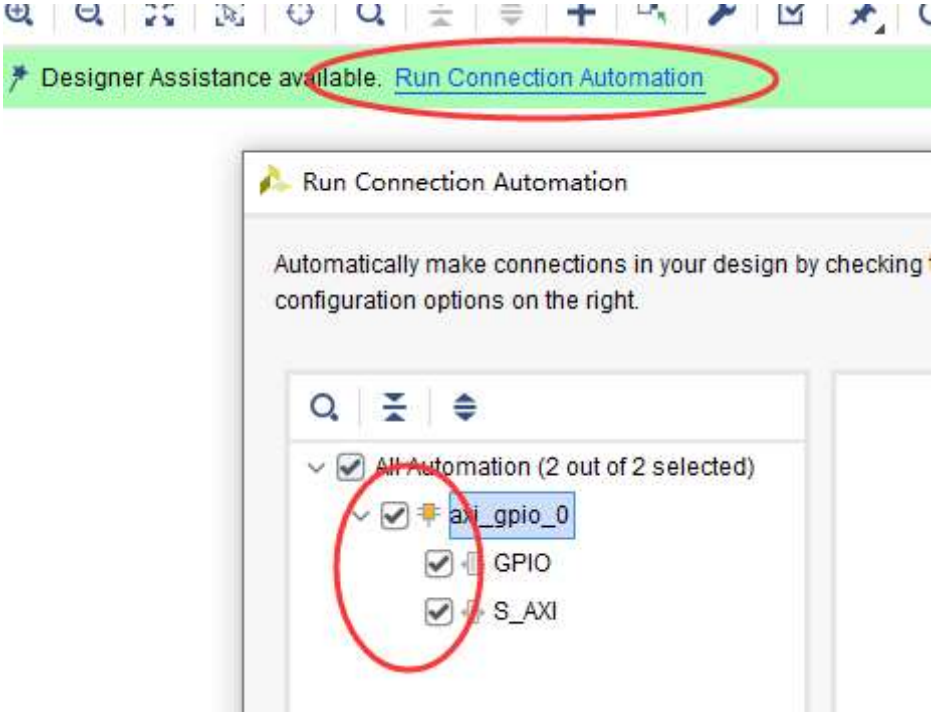


5) Use the same method to increase GPIO resources Click the plus sign, search for GPIO in the settings and add, then double-click and open the generated GPIO resources for settings

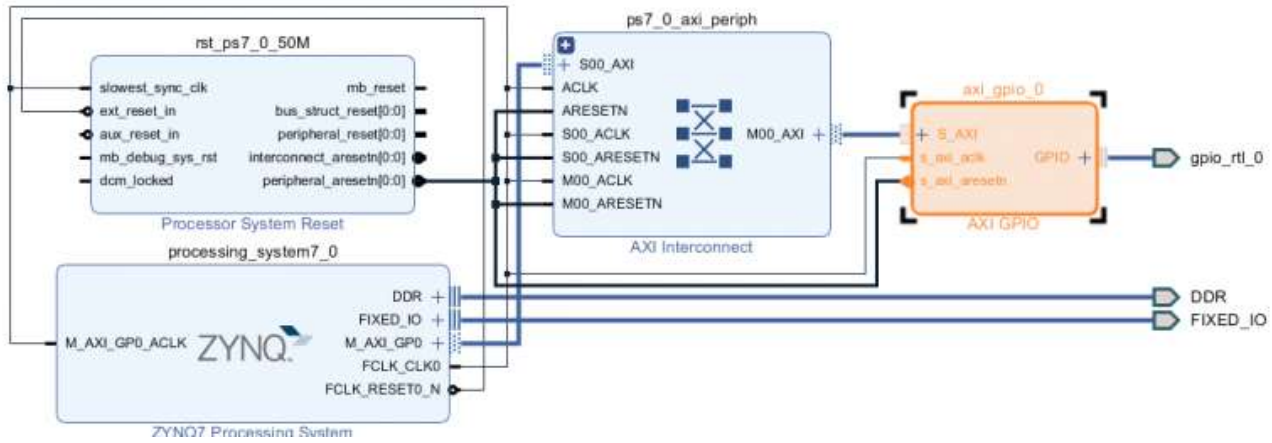
6) In the GPIO settings bar, change the input and output properties to All Output ALL Outputs, because there are only 2 LED lights on our board, so set the GPIO Width width to 2 bits, and click OK



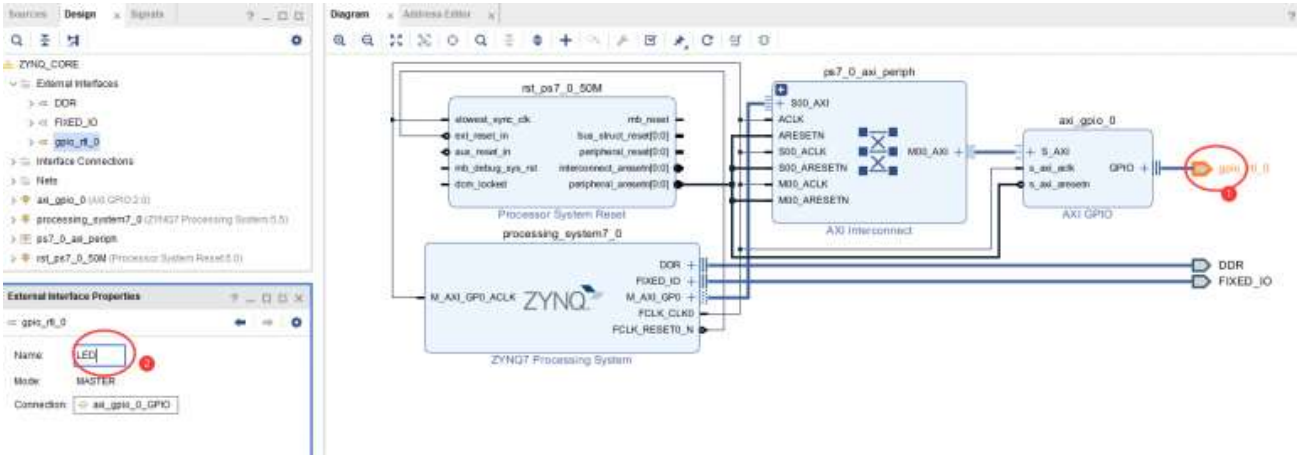
7) Click Run Connection Automation above, and check both the GPIO and the tick in front of the S_AXI, as shown in the following figure click OK



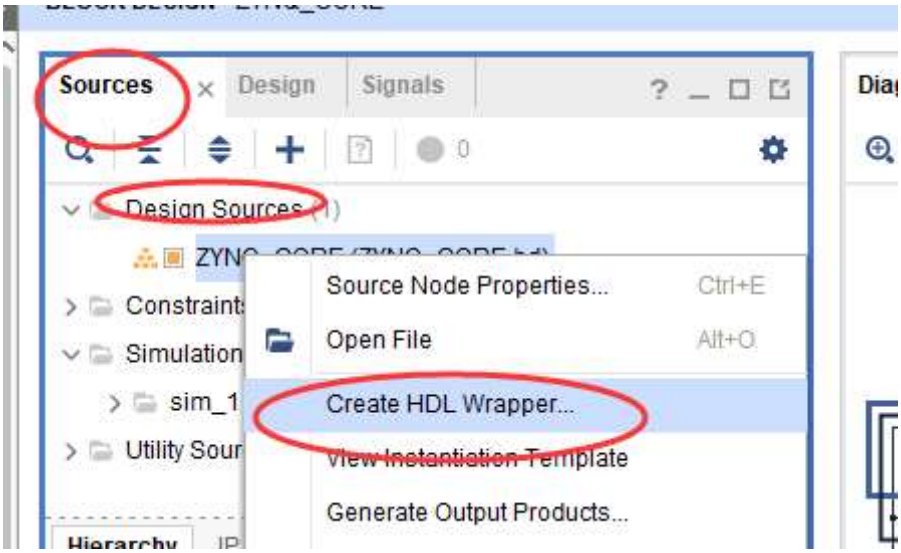
8) The software will automatically help us complete all the necessary connections and add the necessary modules



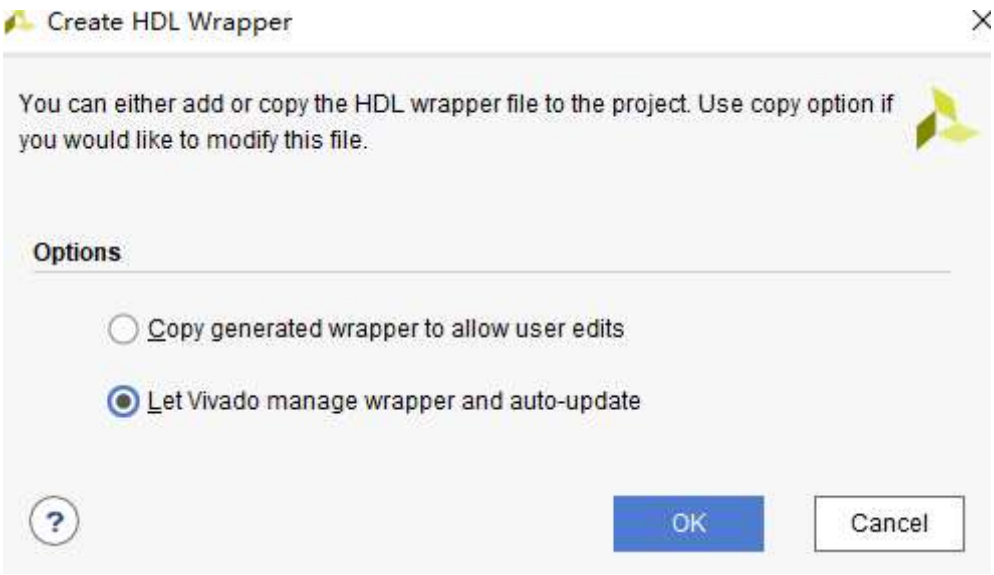
9) Double-click the pin of the GPIO output in the block diagram and rename it to LED to facilitate software programming later



10) source→Design Source, right-click on the BLOCK project we created, and click create HDL wrapper as shown in the figure below.



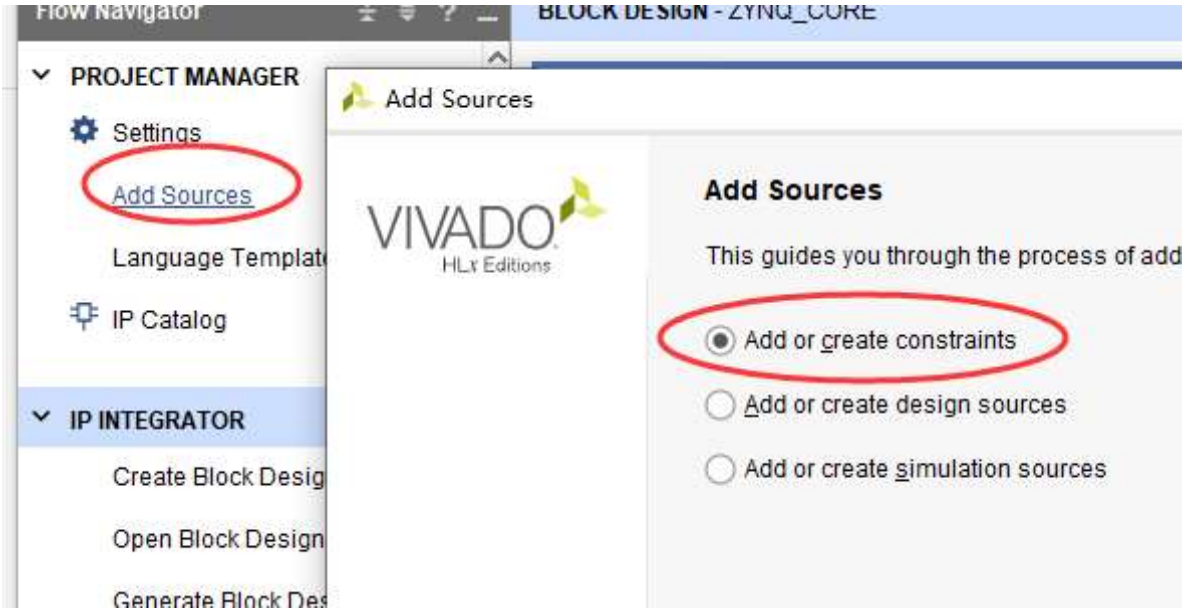
Keep the default in the pop-up dialog



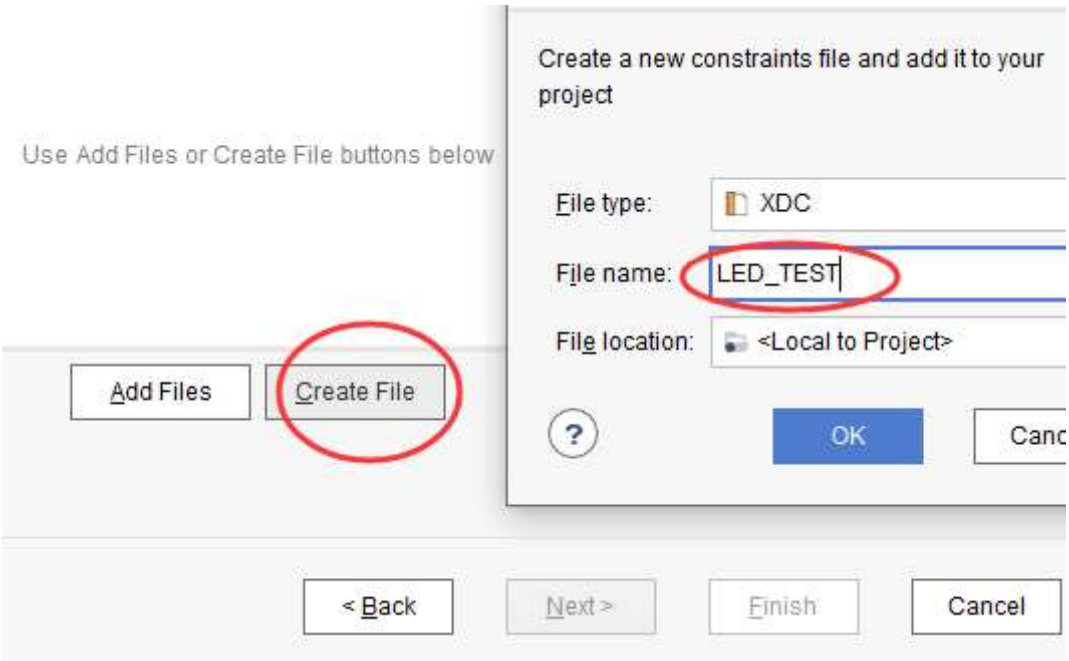
The software automatically generates HDL files for us

3.创建约束文件，并且定义管脚

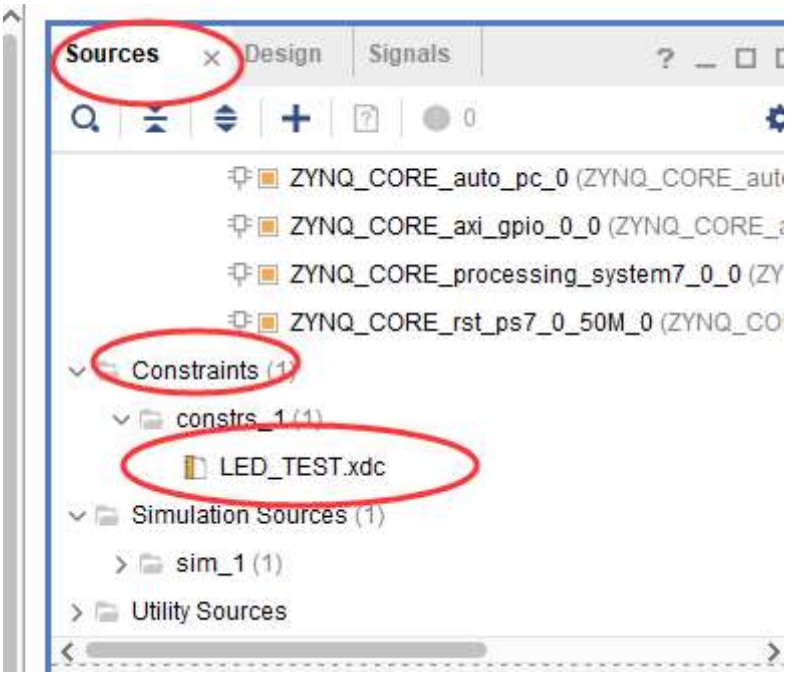
1）Add Source → Add or create constraints 点Next



因为这个项目没有创建过约束文件 所以这里创建一个约束文件，并在File name 里设置约束文件的名称，并且点击FINISH 完成约束文件的创建



2）Sources → Constraints 里找到刚才创建的约束文件 双击并打开该XDC约束文件



在约束文件里面复制下面代码来对输出的GPIO进行管脚（所有的管脚转接板上丝印都有实际标注对应的IO）

```
set_property IOSTANDARD LVCMOS33 [get_ports LED_tri_o[0]]
set_property IOSTANDARD LVCMOS33 [get_ports LED_tri_o[1]]

set_property PACKAGE_PIN P20 [get_ports LED_tri_o[0]]
set_property PACKAGE_PIN P21 [get_ports LED_tri_o[1]]
```

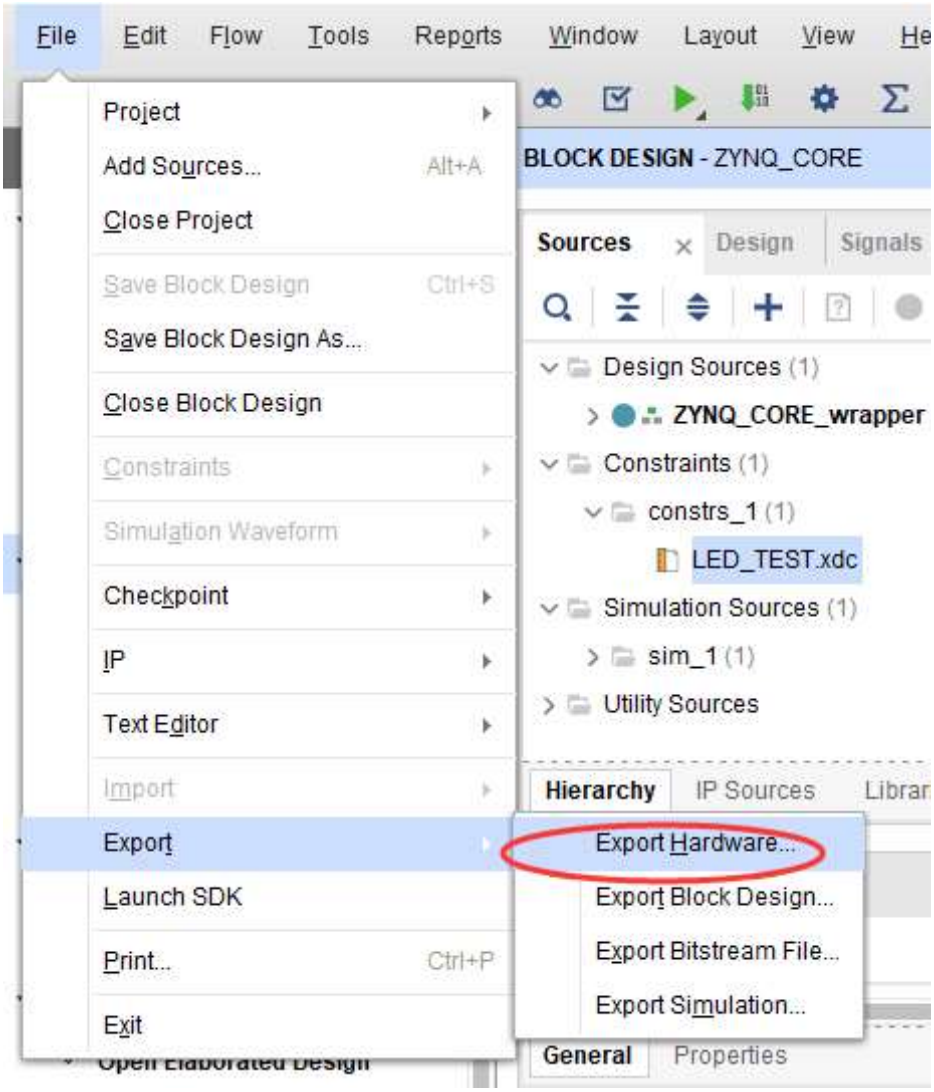
4.生成bit文件

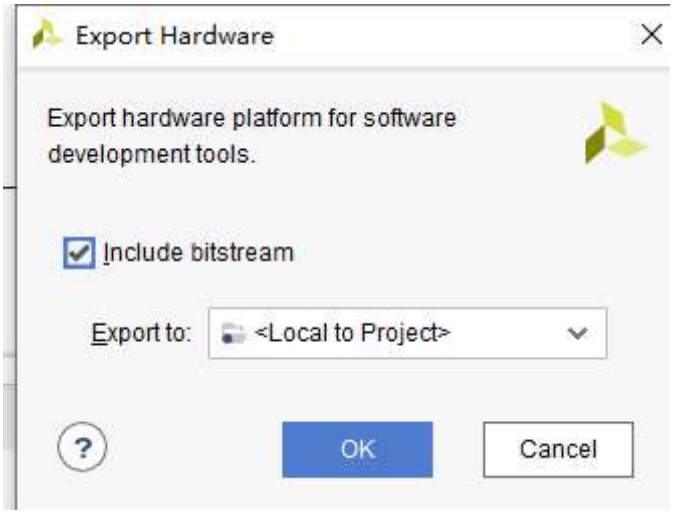
按下绿色箭头对工程进行编译

按下Generate Bitstream 完成综合以及生成bit文件

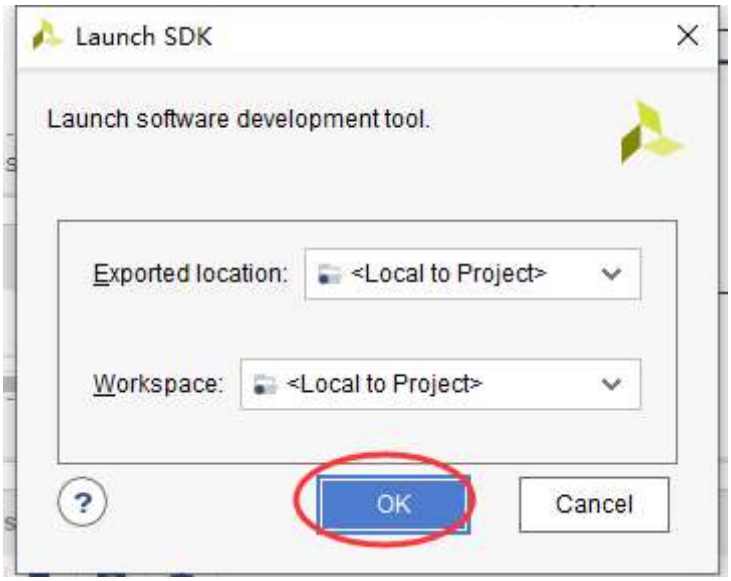
5.SDK程序编写

1）File→Export→Export hardware…，在弹出的对话框中勾选“include bitstream”，点击“OK”确认，如下图所示。

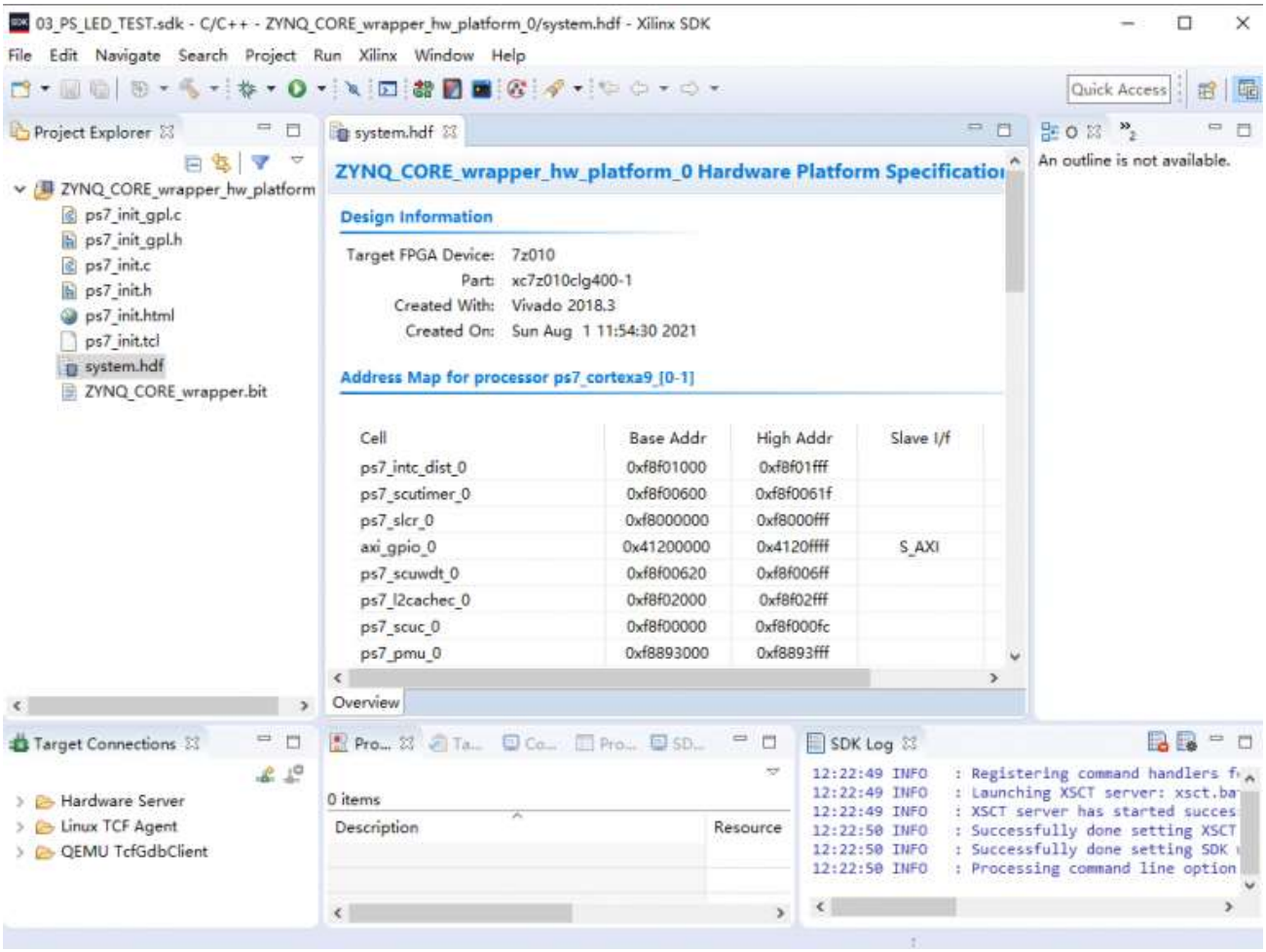




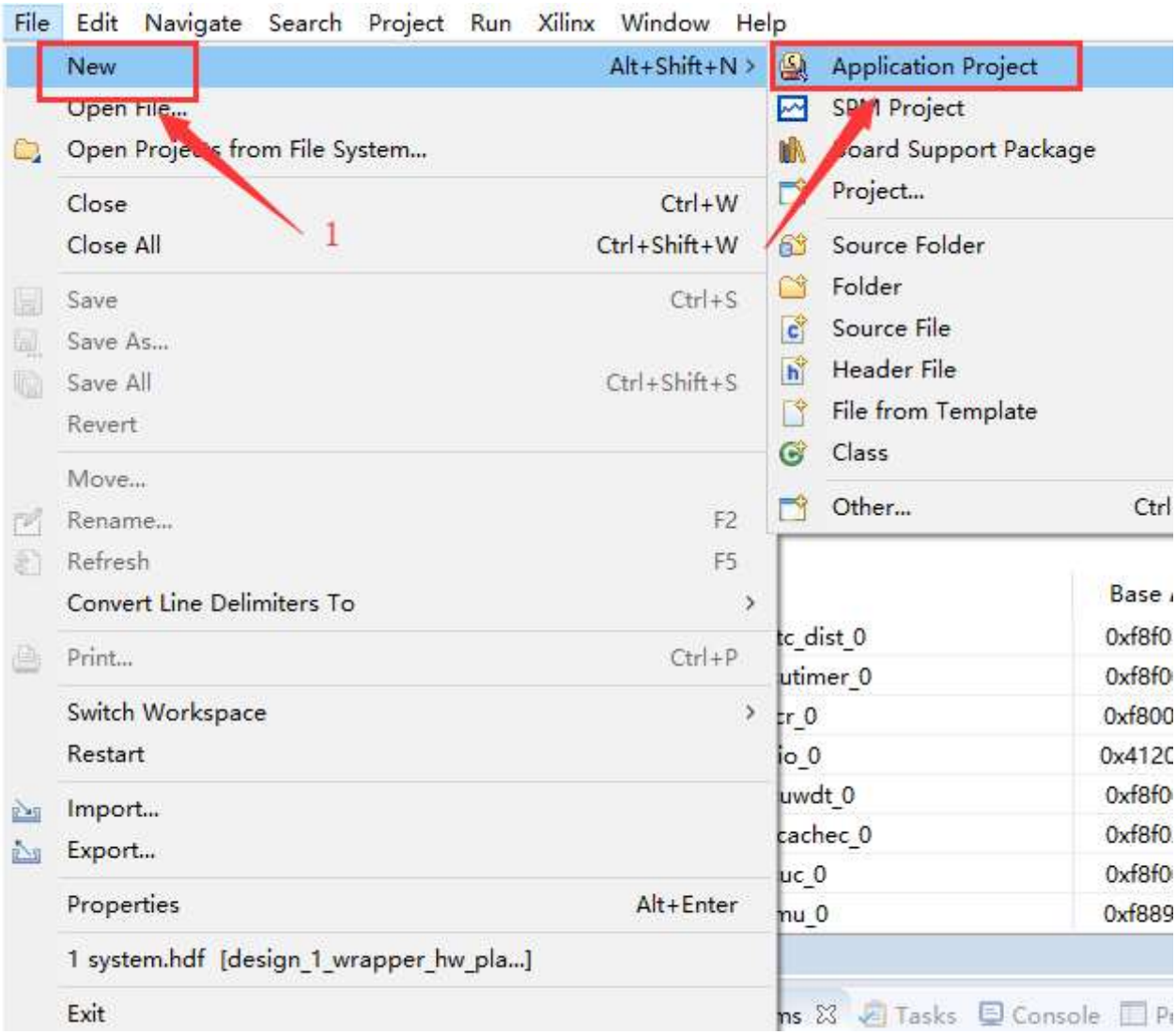
2）File→Lauch SDK，在弹出的对话框中，保存默认，点击“OK”，如下图所示。



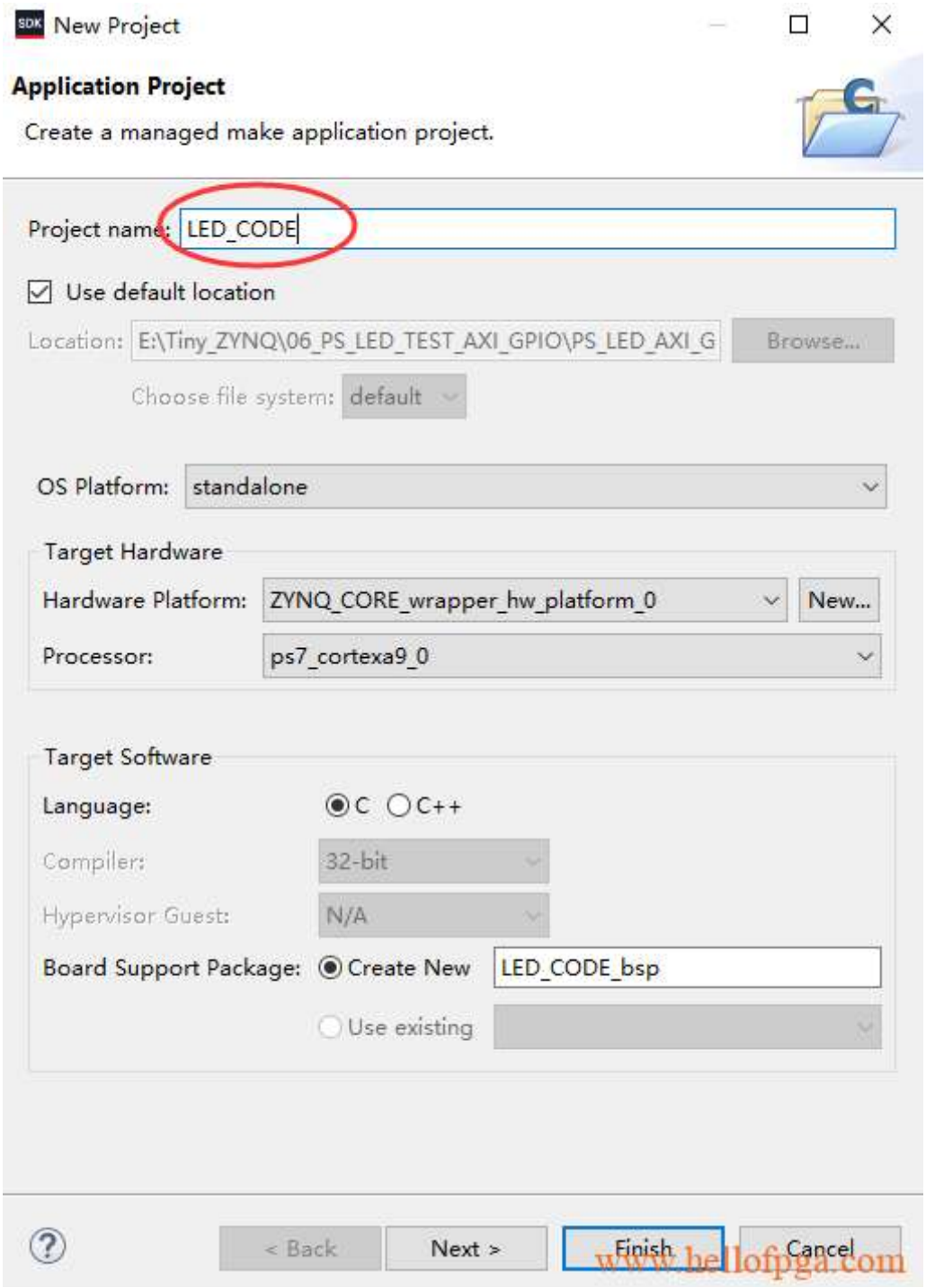
系统将自动打开SDK开发环境



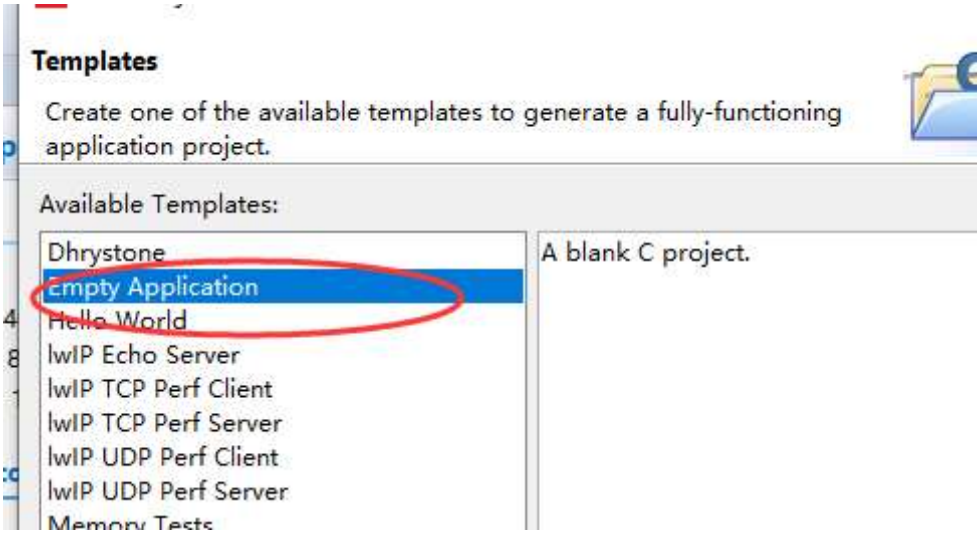
3）新建一个工程 file→new→Application Project，来新建一个“Application Project”，如下图所示。



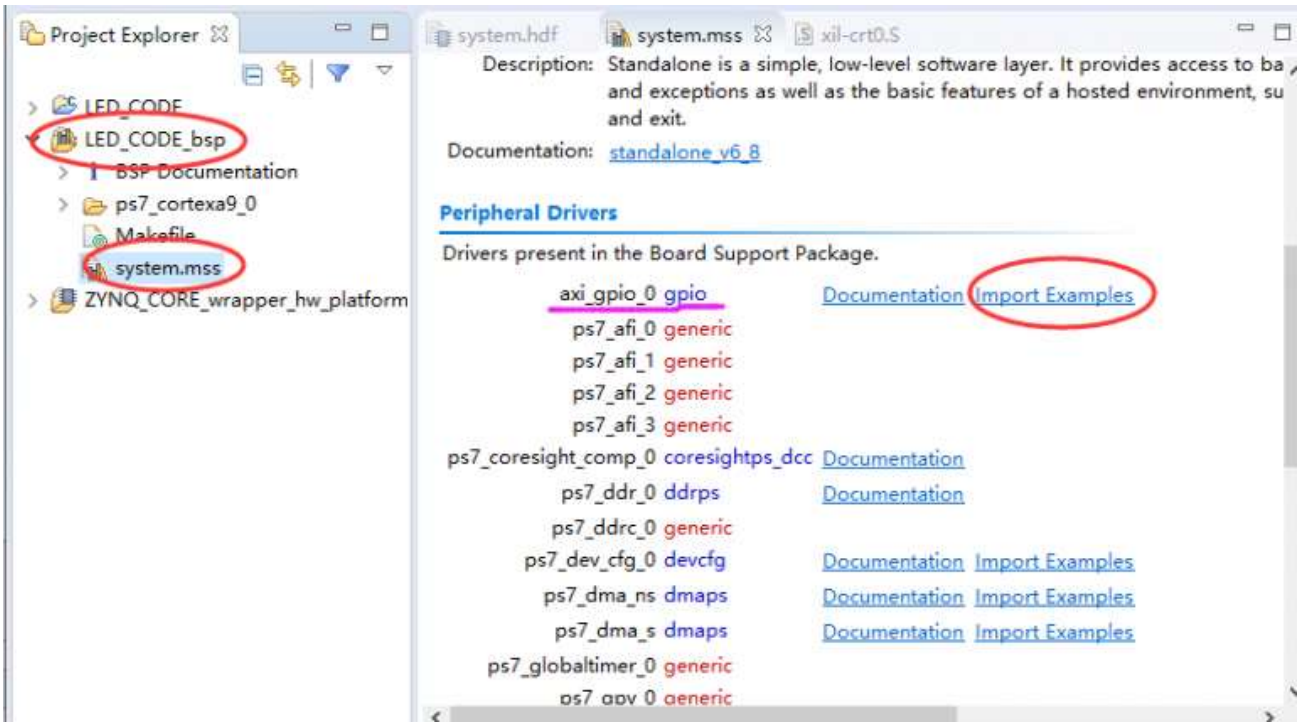
4）在新建工程名中输入自己的工程名称，点击NEXT



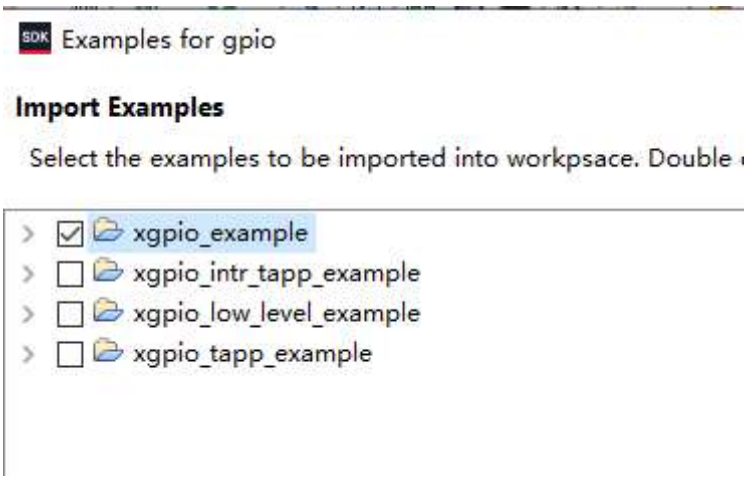
5) Select the empty project and click Finish



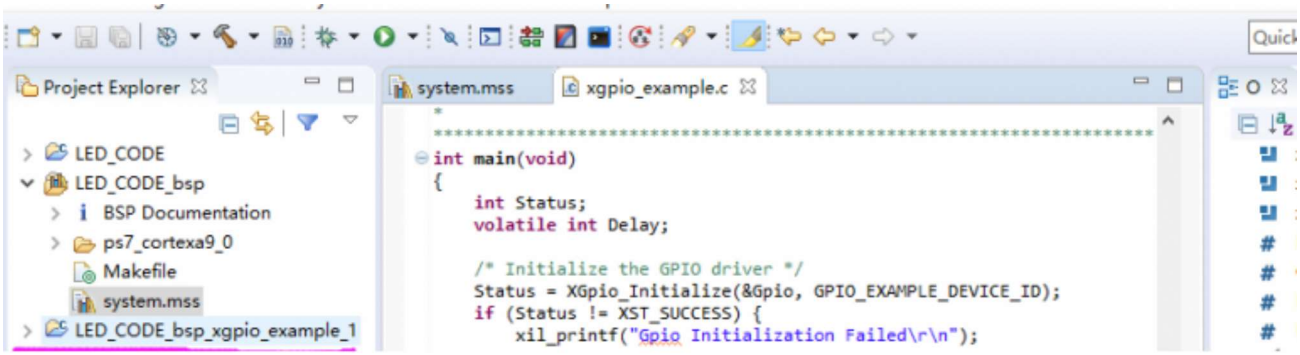
6) Select the created LED_CODE_bsp → system.mss → axi_gpio_0 gpio → import Examples



7) Select xgpio_example



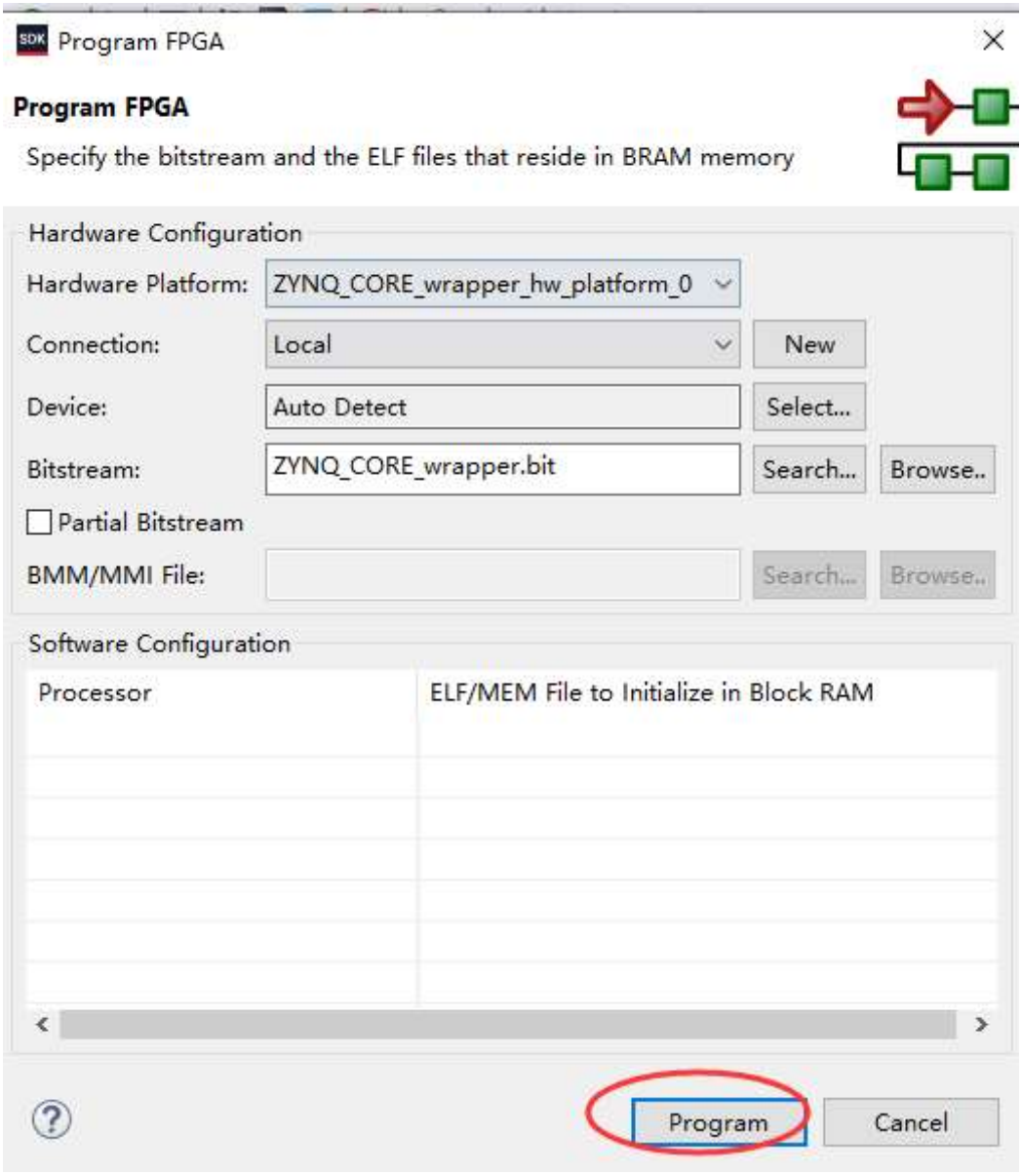
8) This automatically creates a reference routine



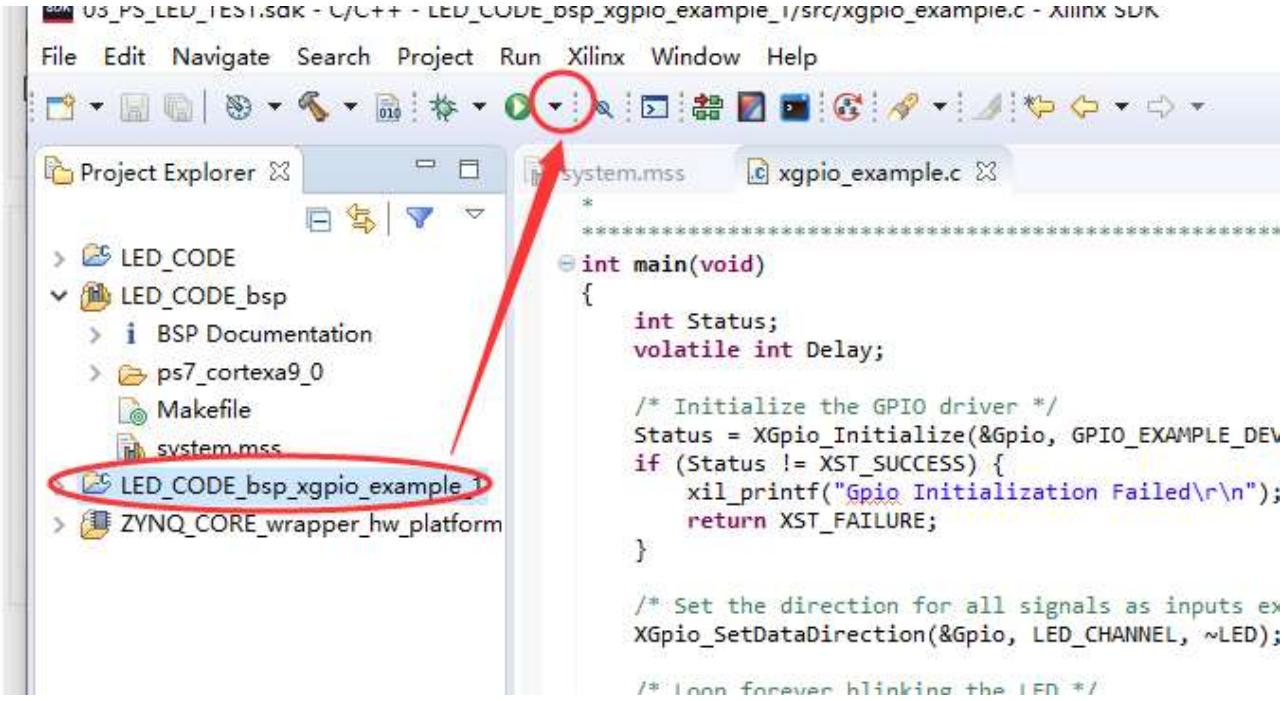
Note, you can directly download the reference design to see the effect, or you can refer to the code to add to your own main. Here we choose to download the reference code directly to see the effect

6. Download to the board for verification

Select the hardware platform in the project, right-click → Program FPGA, select Default in the pop-up dialog box, and click "program" to complete the Program work in the FPGA PL part

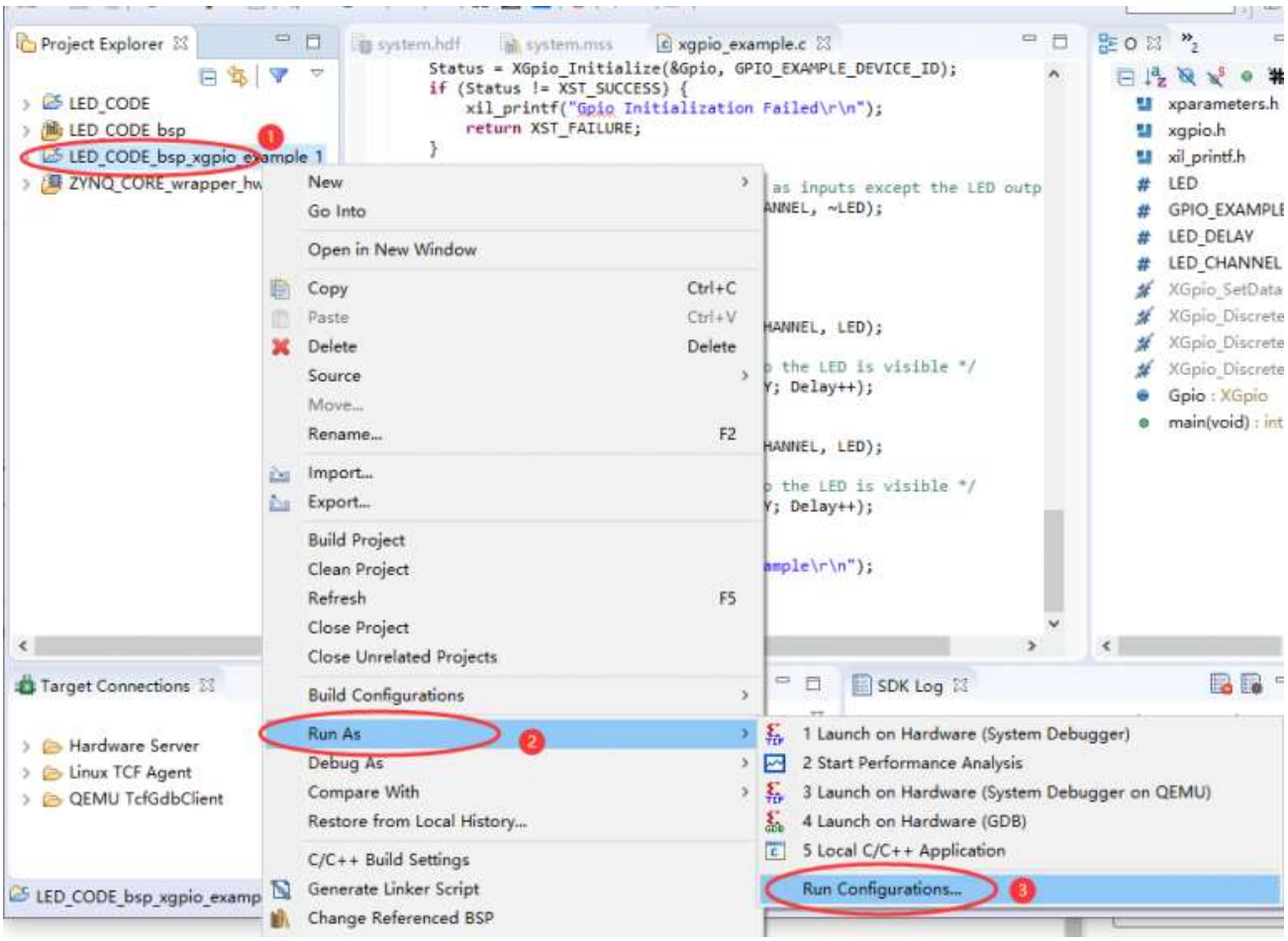


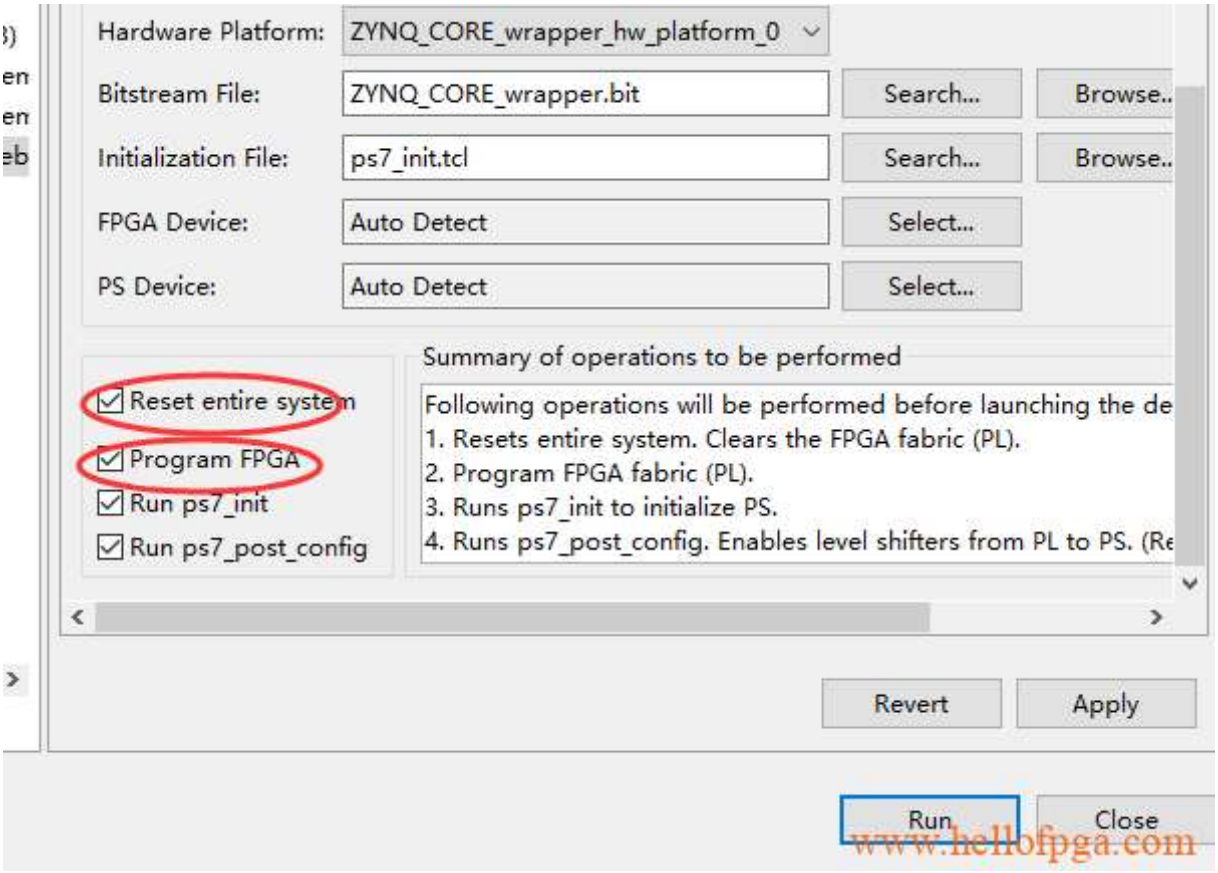
2) Select the GPIO project we generated, expand the icon to the right of the green arrow (RUN), and select Run As→1 Launch on Hardware (System Debugger)



You can see the LED1 light on the board blinking

Note: If an error pops up when RUN, you can follow the following operations to set up and then DEBUG





Then click APPLY and then select Run As→1 Launch on Hardware (System Debugger) to see if the download is successful

Code interpretation

```
XGpio_SetDataDirection 设置GPIO为输入/输出
XGpio_DiscreteWrite    置位GPIO
XGpio_DiscreteClear    拉低GPIO

#define LED 0x01
继续看代码，LED在代码中的定义为 0X01  即代表 LED 在LED_CHANNEL中代表最低位 BIT0

看懂了上面这些 就很容易看懂程序里的代码
XGpio_DiscreteClear(&Gpio, LED_CHANNEL, LED); 代表拉低GPIO LED_CHANNEL中的 第0位
XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, LED); 代表拉高GPIO LED_CHANNEL中的 第0位
XGpio_SetDataDirection(&Gpio, LED_CHANNEL, ~LED); 设置为输出模式

for (Delay = 0; Delay < LED_DELAY; Delay++); 这是一个耗费系统资源的delay函数的简写
```

所以整个程序的效果就是LED 1 不停的点亮熄灭，反复循环。

如果想要将1个灯改成2个灯　只需要将
#define LED 0x01　更改为　#define LED 0x03即可，　07代表 0000_0111即使 bit0 bit1 bit2都置1，　修改之后 3个灯就能同步进行变换了

看懂这些 可以自己创建一个main函数 然后在里面参考这个例子写驱动LED的代码（本文中介绍一下如果利用参考设计来创建工程）

完整工程如下：(板子有多个版本，请按照对应板子的芯片型号进行下载测试)

06_PS_LED_TEST_AXI_GPIO_XC7Z020　　**Download**

The following website is my own compilation of the different points of MIO EMIO AVIGPIO, if you are interested, you can take a look

http://www.hellofpga.com/index.php/2021/08/08/zynq_emio_mio_axi_gpio/

In addition, if you want to know about EMIO to drive GPIOs, you can see the next project (Engineering 7)

SMART ZYNQ SP & SL