APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL Edition) Project IX Place the program on the TF card and perform a TF card startup demonstration

ZYNQ HAS A VARIETY OF BOOT OPTIONS, IN ADDITION TO THE PREVIOUS INTRODUCTION OF BOOTING FROM QSPI FLASH, YOU CAN ALSO LET THE SYSTEM BOOT FROM TF CARD, THIS ARTICLE WILL DEMONSTRATE THE BOOT PROCESS OF TF CARD BY AN EMIO POINT LED LIGHT.

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 ZYNQ initiation process

1. First understand the boot process of ZYNQ, the boot process of ZYNQ is mainly based on ARM core, after power-on, the hardware first reads the IO of the PS end of the ARM core to determine whether the system boots the system from NAND, QSPI, SD Card or JTAG.

The specific boot method is shown in the figure below, our minimum system board only has QSPI and TF hardware, so we only need to pay attention to the three boot methods of QUAD-SPI, SD, JTAG. That is, look at the level status of the two pins MIO5 and MIO4 at

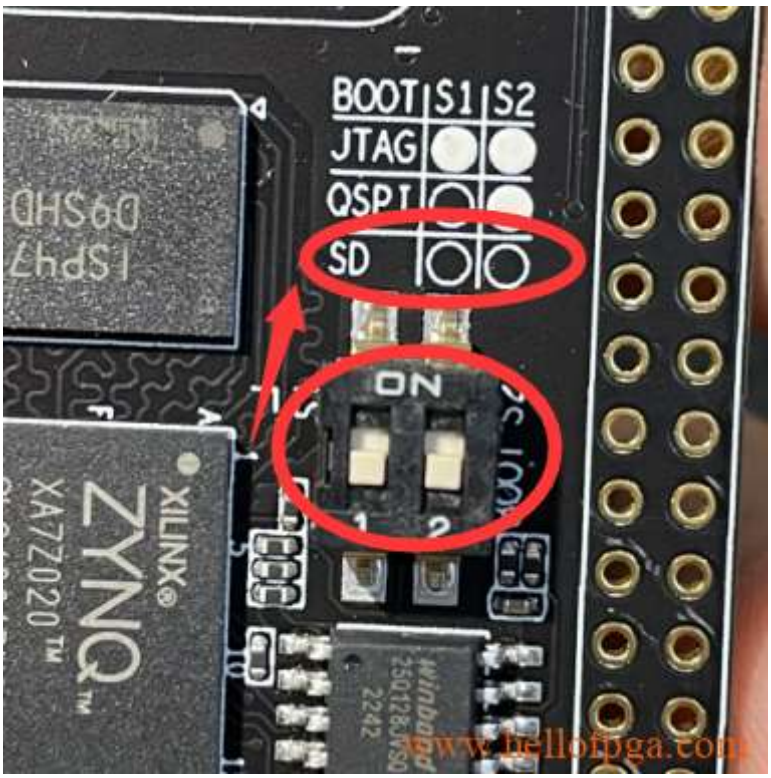
the moment the system is powered on, which is used to distinguish different power-on startup methods.

Table 6-4: **Boot Mode MIO Strapping Pins**

Pin-signal / Mode	MIO[8]	MIO[7]	MIO[6]	MIO[5]	MIO[4]	MIO[3]	MIO[2]
	VMODE[1]	VMODE[0]	BOOT_MODE[4]	BOOT_MODE[0]	BOOT_MODE[2]	BOOT_MODE[1]	BOOT_MODE[3]
Boot Devices							
JTAG Boot Mode; cascaded is most common(1)				0	0	0	**JTAG Chain Routing(2)** 0: Cascade mode 1: Independent mode
NOR Boot(3)				0	0	1	
NAND				0	1	0	
Quad-SPI(3)				1	0	0	
SD Card				1	1	0	
Mode for all 3 PLLs							
PLL Enabled			0	Hardware waits for PLL to lock, then executes BootROM.			
PLL Bypassed			1	Allows for a wide PS_CLK frequency range.			
MIO Bank Voltage(4)							
	Bank 1	**Bank 0**	Voltage Bank 0 includes MIO pins 0 thru 15. Voltage Bank 1 includes MIO pins 16 thru 53.				
2.5 V, 3.3 V	0	0					
1.8 V	1	1					

In the process of circuit design, MIO5 and MIO4 have been connected to the toggle switch respectively, that is, according to the state of the toggle switch at the moment of power-on can control the system of start-up, please refer to the silk screen mark on the board

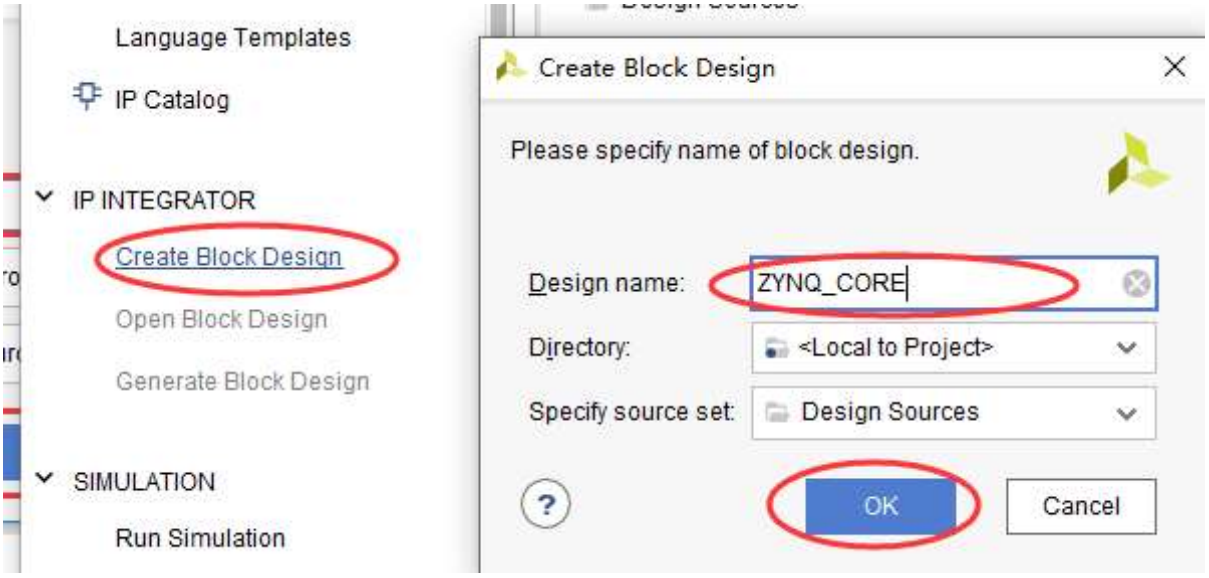
Adjust the DIP switch to the state shown in the figure, at this time, if there is an executable program on the TF card, the system will boot from the TF card after the power is off and restarted



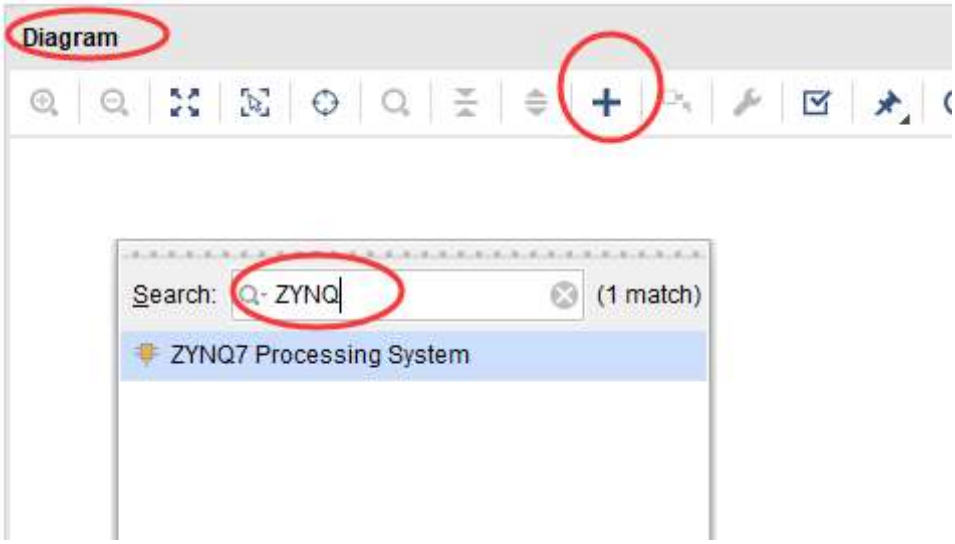
2 Software partial configuration

2.1 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

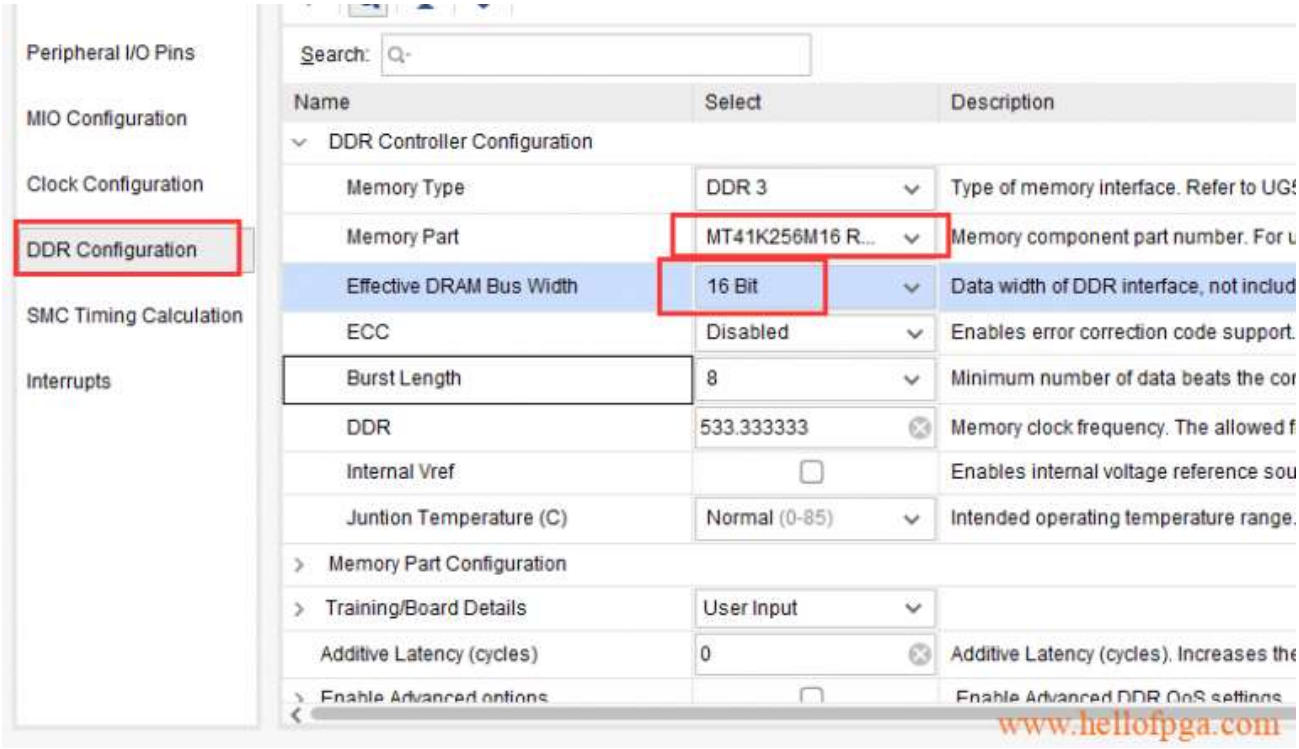


2) IN THE WINDOW ON THE RIGHT, CLICK THE PLUS SIGN, SEARCH FOR ZYNQ IN THE SELECTION BOX, AND FIND ZYNQ7 PROCESSING SYSTEM, DOUBLE-CLICK AND OPEN

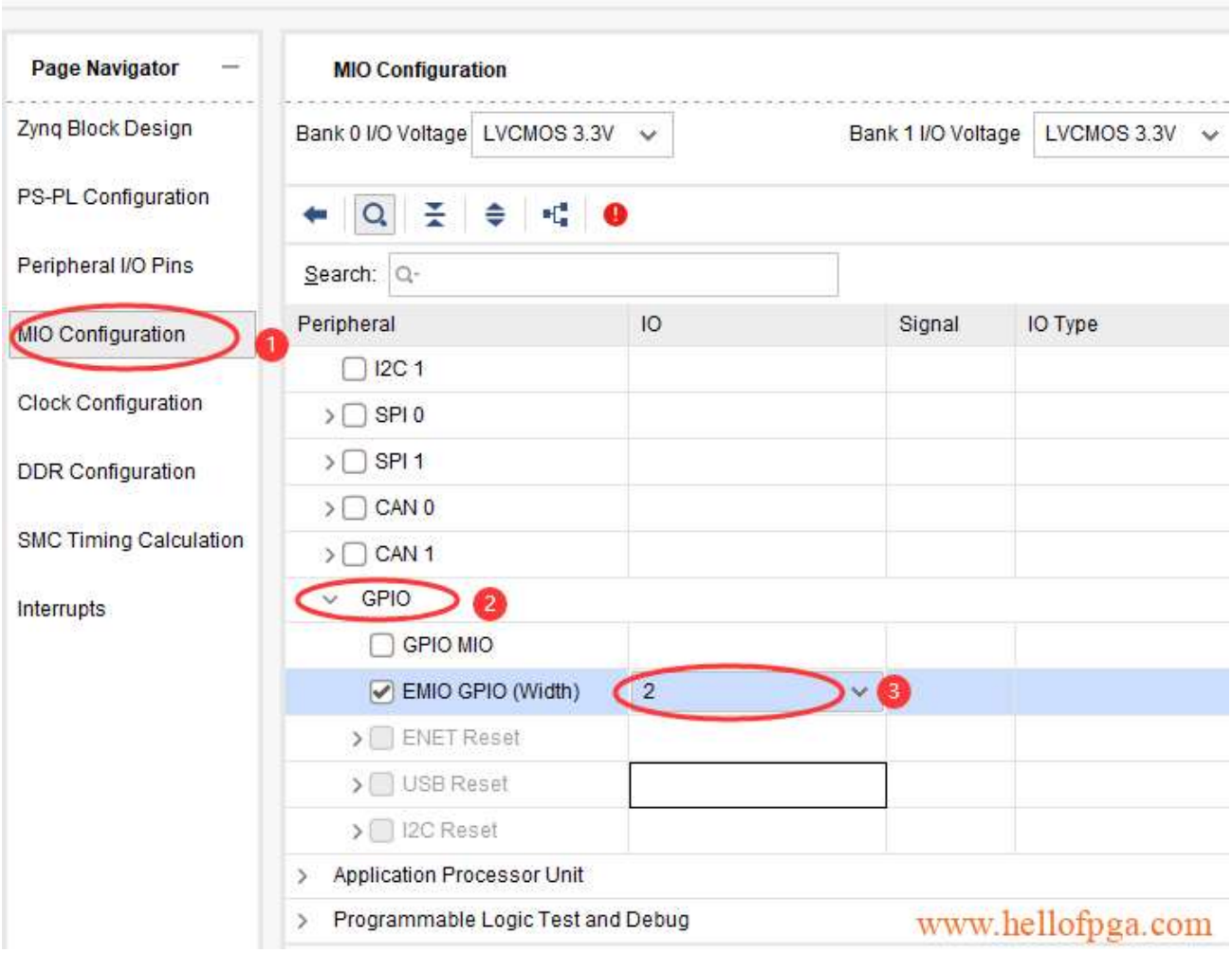


3) The software automatically generates a zynq block as shown in the figure below, next to do some corresponding settings, double-click the ZYNQ core in the figure below

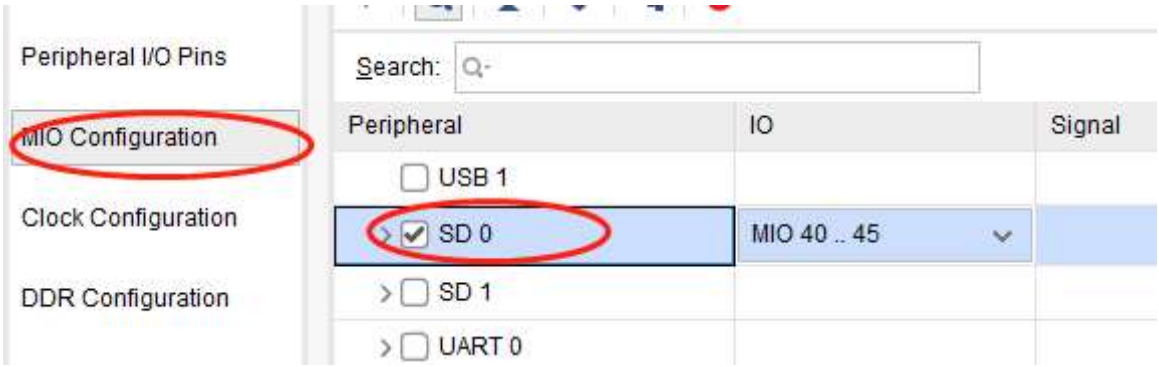
4) Find DDR Configuration →DDR Controller Configuration →DDR3 in the pop-up window in turn, select the corresponding DDR3 according to the DDR on your board in the Memory Part drop-down menu, the model used in this experiment: MT41K256M16RE-125, select 16bit of data width and finally click "OK", as shown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



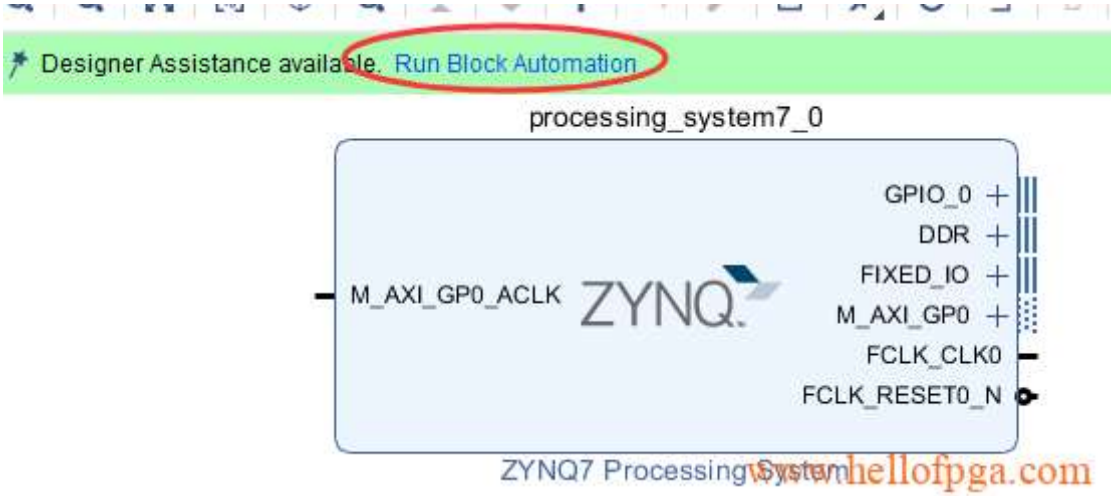
5) In the GPIO column of the MIO configuration option of PS, add two EMIOs (because this test is two, if you need to add buttons or other IOs, you can adjust them accordingly here (here the two GPIOs are used to light up the LED lights to see the effect demonstration, non-TF startup is required)



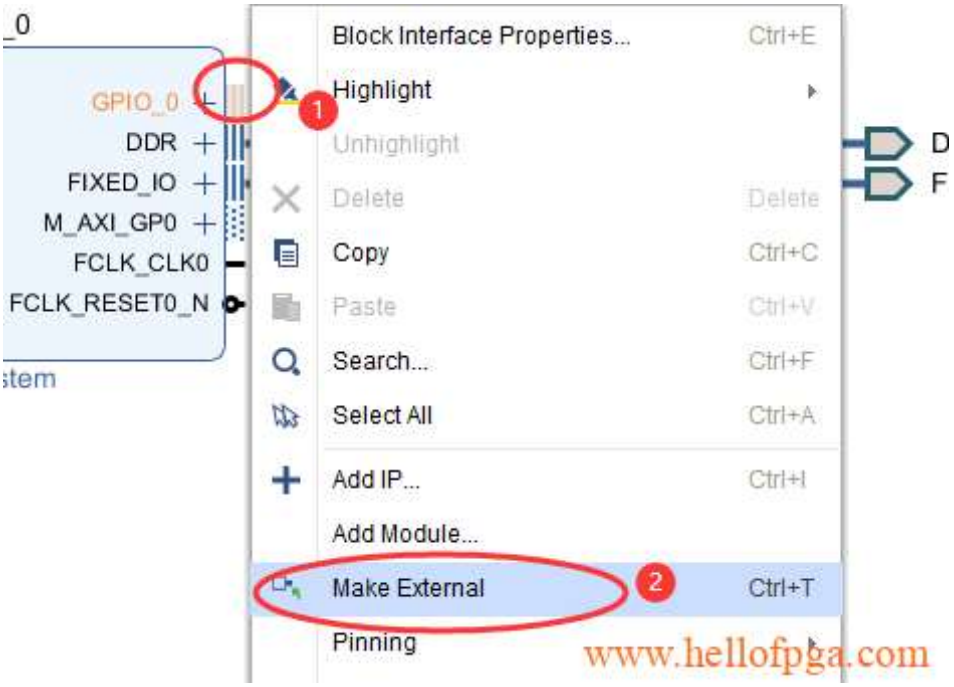
6). An important step **in MIO Configuration enables the SD 0 function, this step is the key setting for TF card startup**



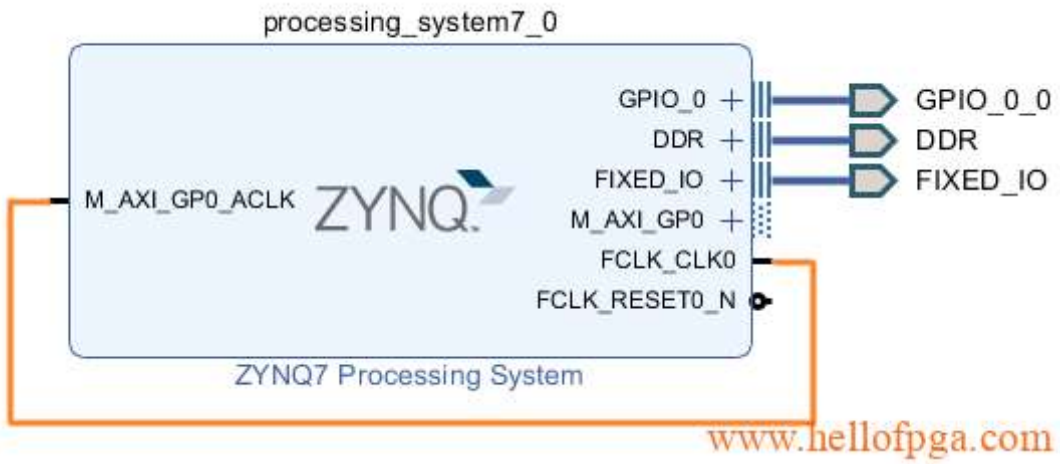
7) Finally, click on "Run Block Automation" as shown in the image below. Keep the default in the pop-up options and click "OK" to complete the configuration of the ZYNQ7 Processing System



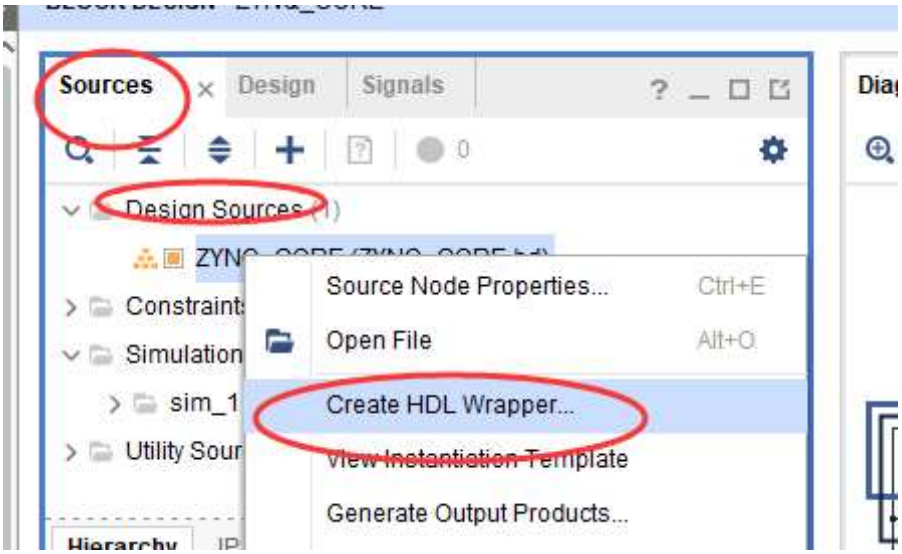
8) Right-click the GPIO_0—>Make External



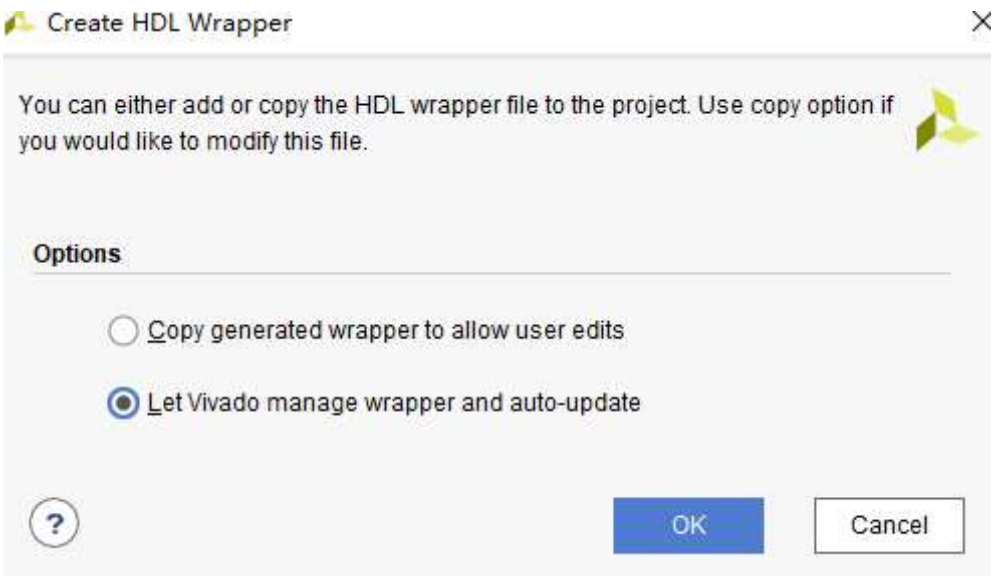
9) Connect the M_AXI_GP0_ACLK with the FCLK_CLK0 with a line



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



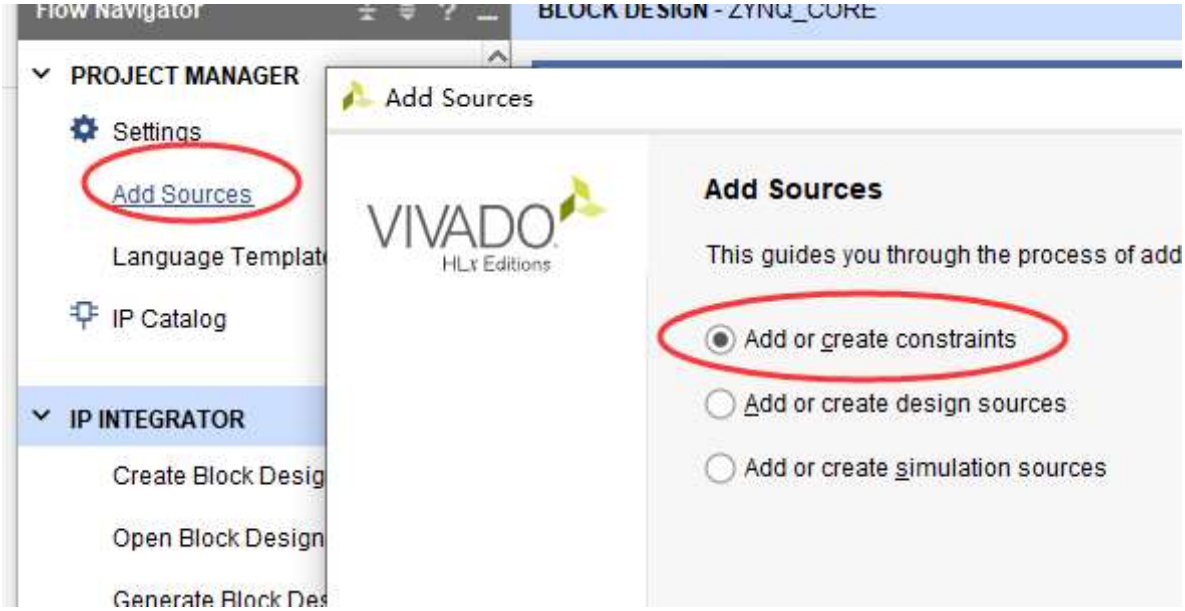
11) Keep the default in the pop-up dialog



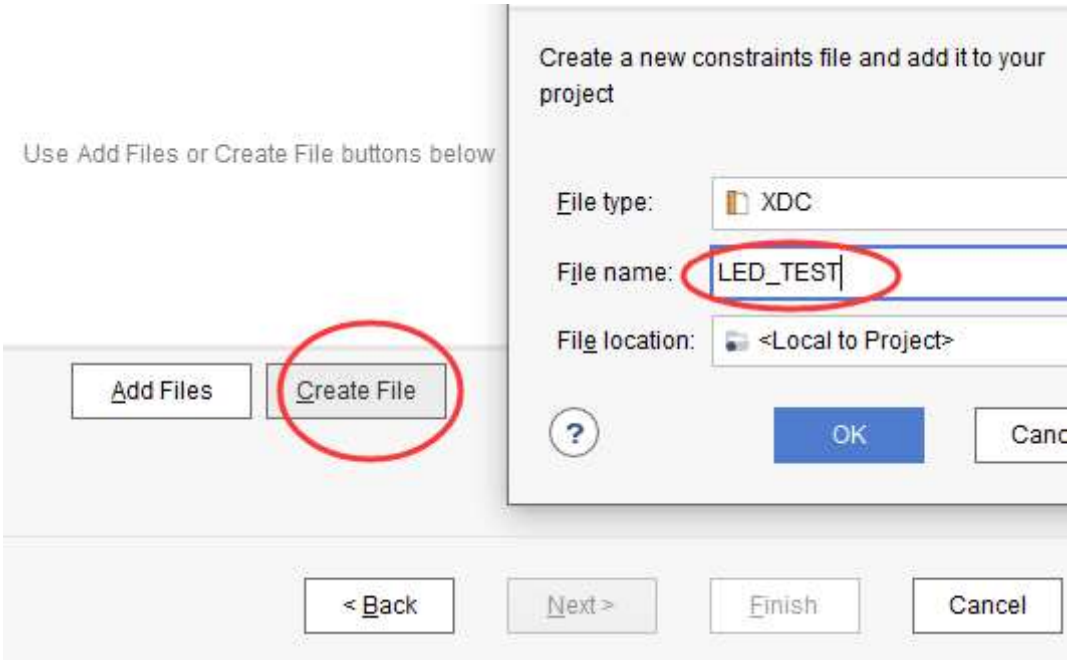
12) The software automatically generates HDL files for us

2.2 Create a constraint file and define the pins

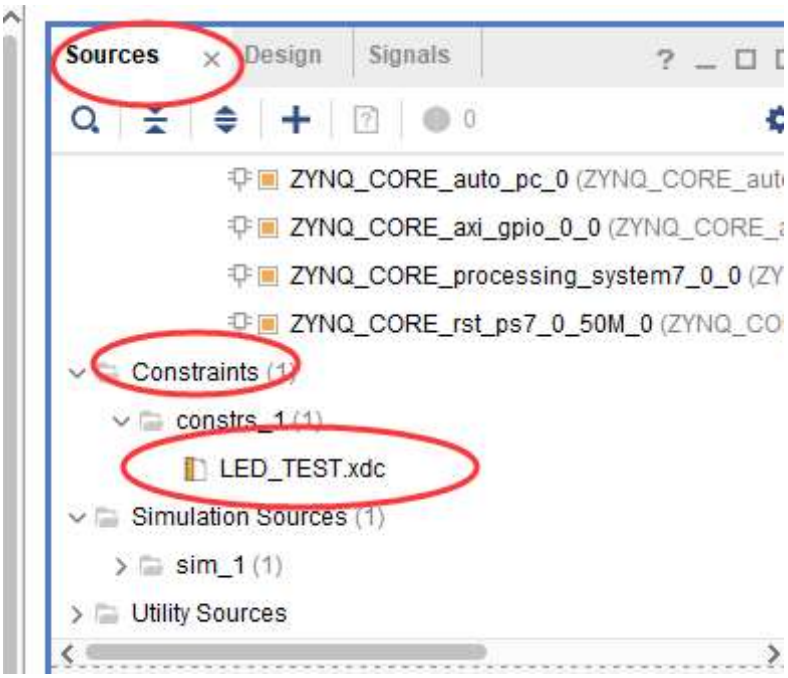
1）Add Source → Add or create constraints 点Next



Since no constraint file has been created for this project, create a constraint file here, set the name of the constraint file in the file name, and click FINISH to complete the creation of the constraint file



2) Sources → Constraints to find the constraint file you just created Double-click and open the XDC constraint file

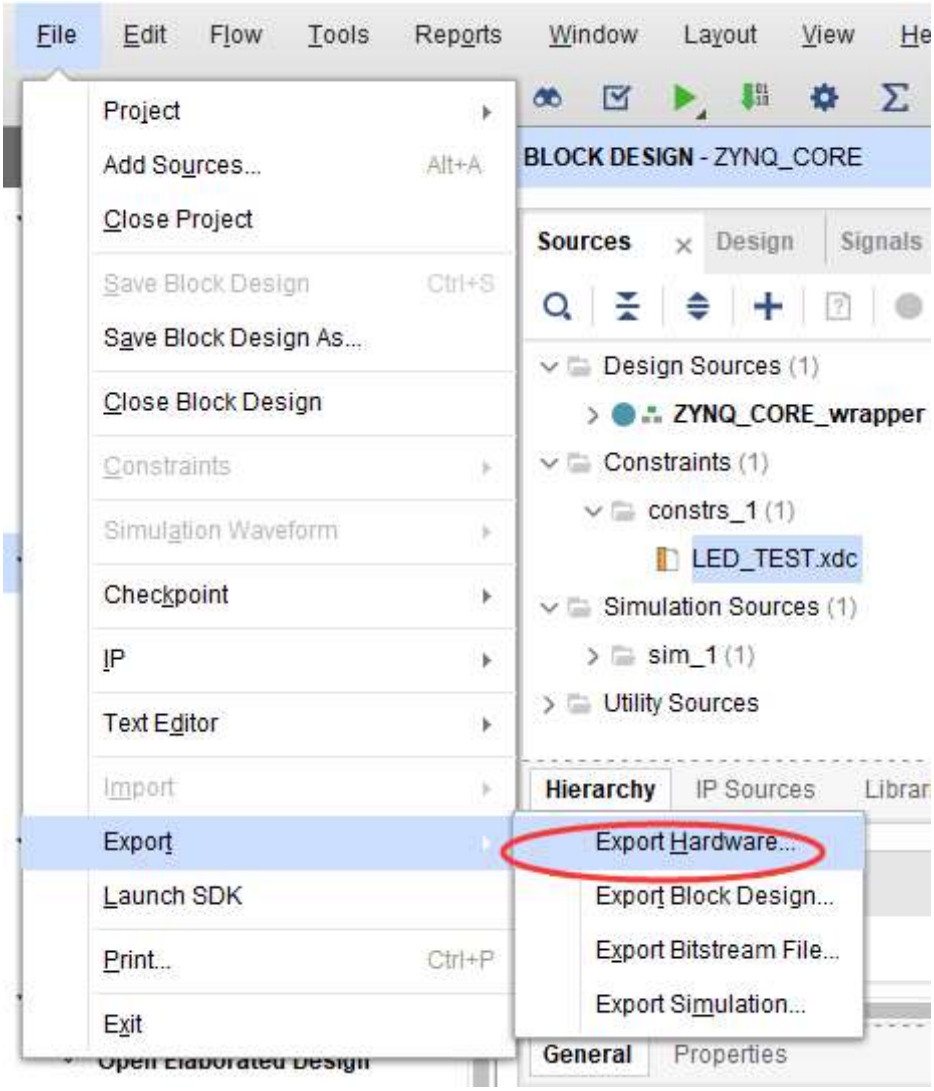


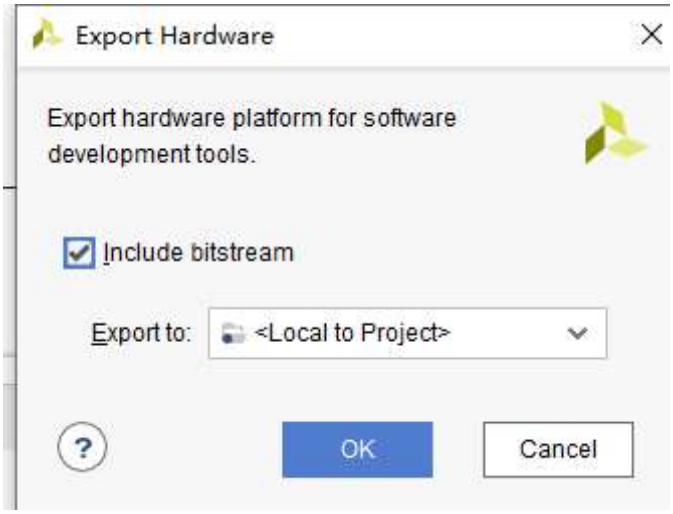
Copy the following code in the constraint file to pin the output GPIO (all pin adapter boards have actual annotations corresponding IO)

```
set_property IOSTANDARD LVCMOS33 [get_ports GPIO_0_0_tri_io[0]]
set_property IOSTANDARD LVCMOS33 [get_ports GPIO_0_0_tri_io[1]]

set_property PACKAGE_PIN P20 [get_ports GPIO_0_0_tri_io[0]]
set_property PACKAGE_PIN P21 [get_ports GPIO_0_0_tri_io[1]]
```

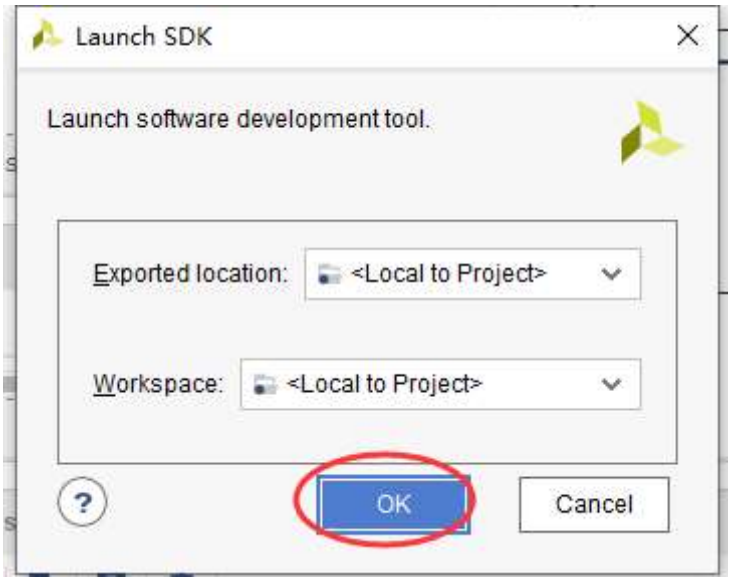
3. SDK program writing

1) File→Export→Export hardware..., check "include bitstream" in the pop-up dialog box, and click "OK" to confirm, as shown in the figure below.

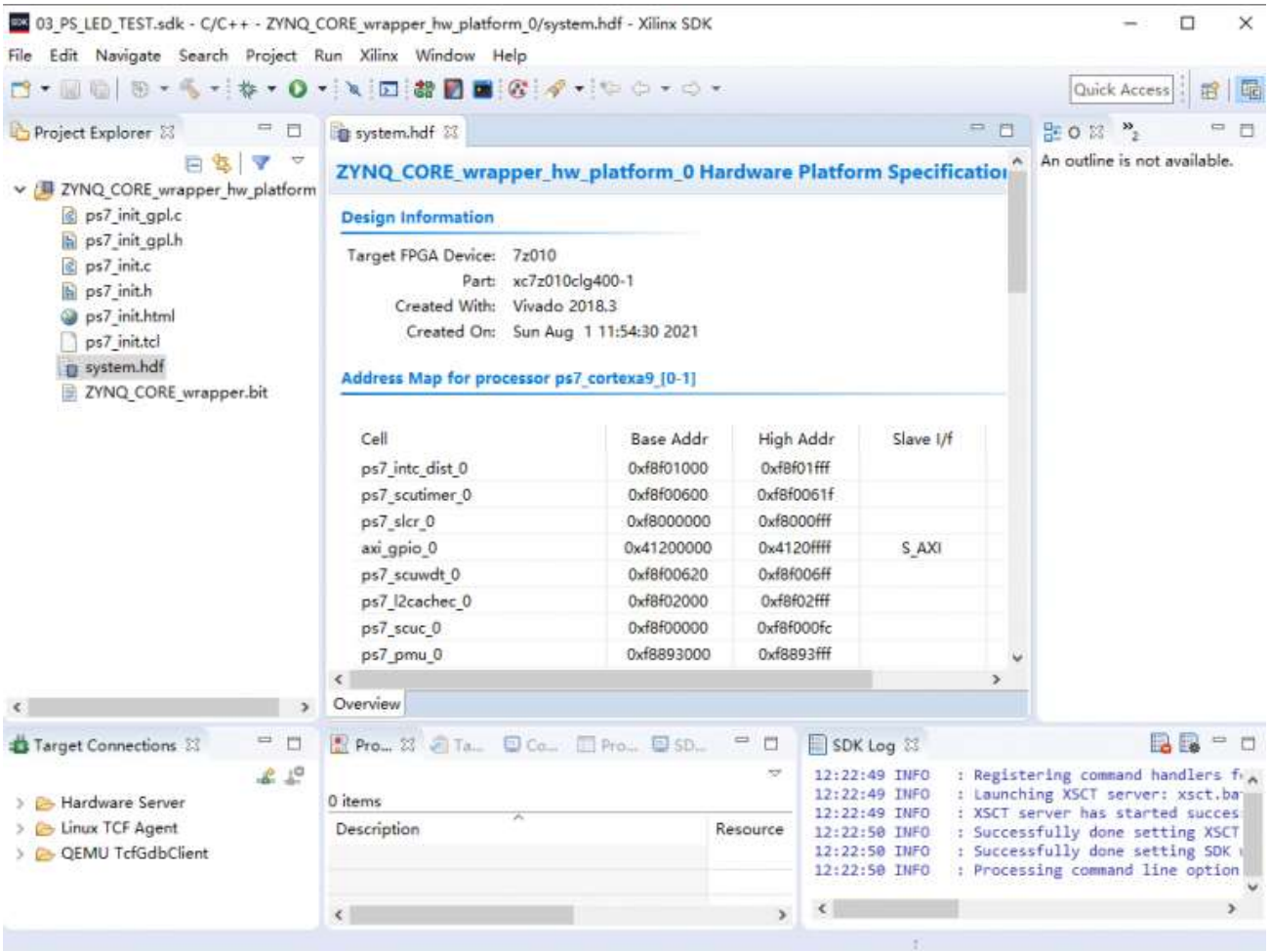




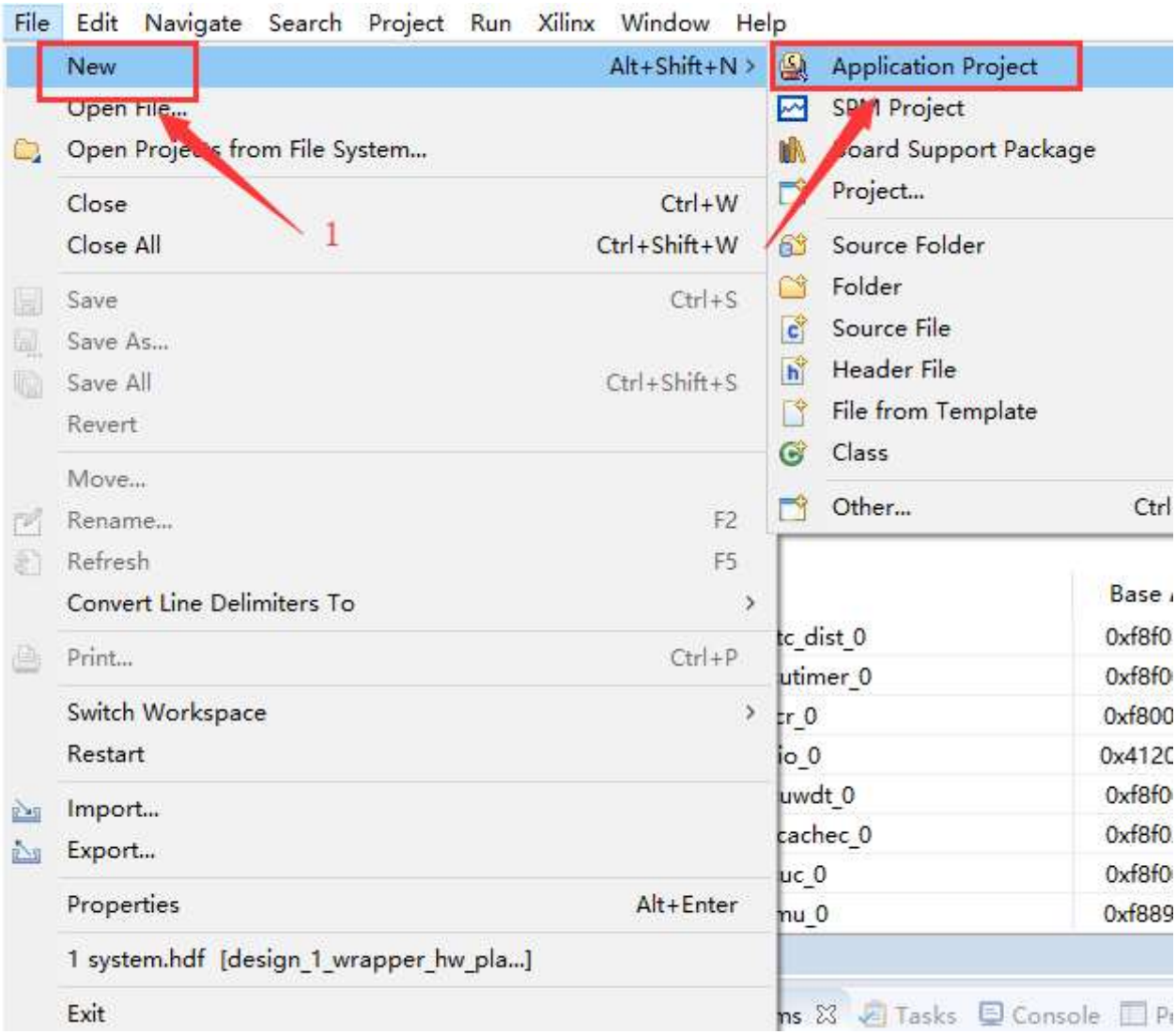
2) File→ Lauch SDK, in the pop-up dialog box, save the default, click "OK", as shown in the figure below.



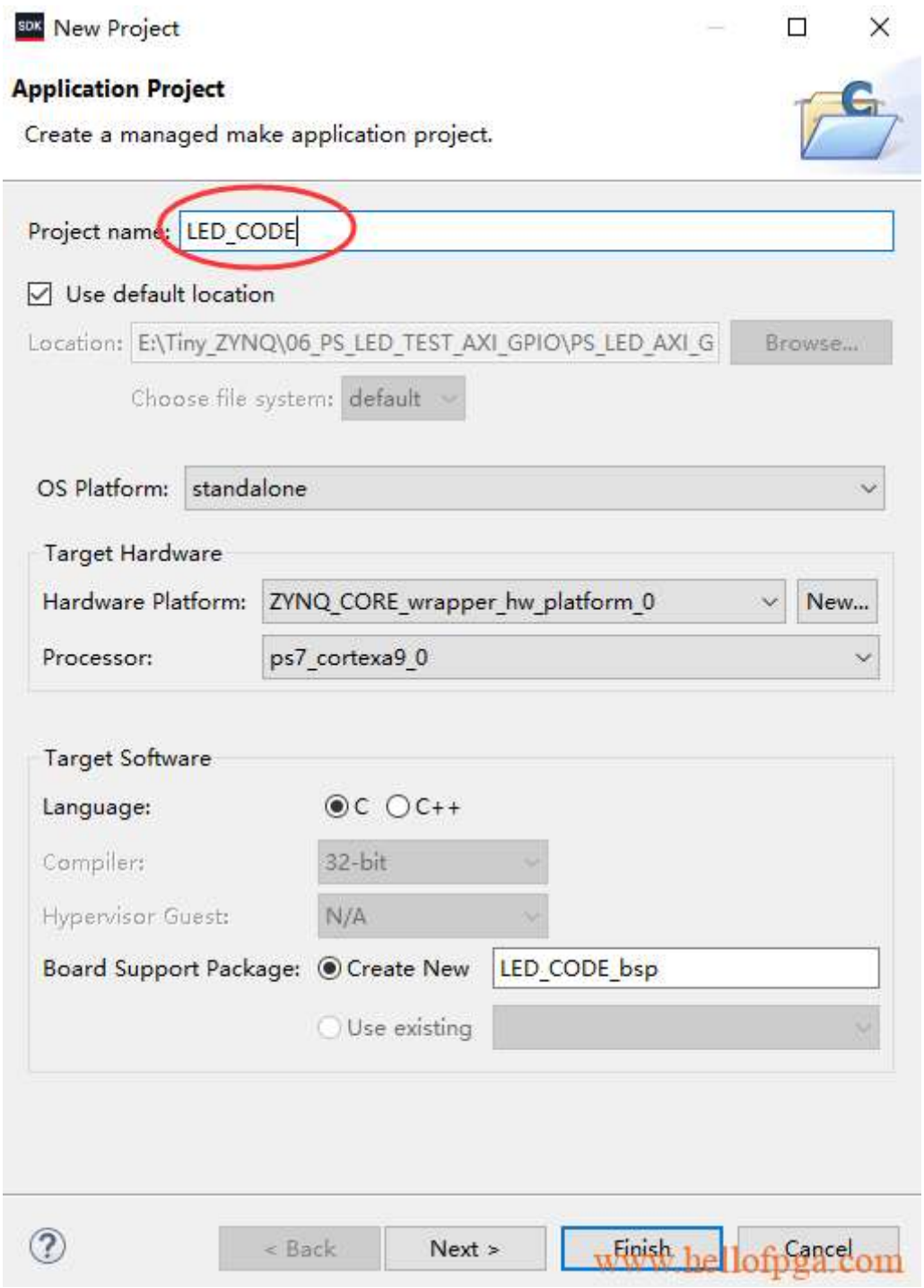
The SDK development environment will automatically open



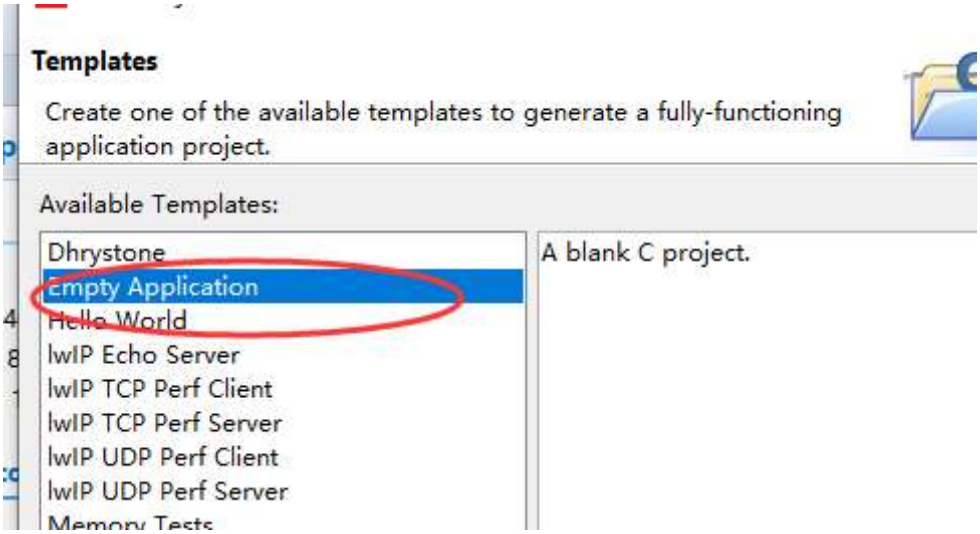
3) Create a new project file→new→Application Project to create an "Application Project", as shown in the figure below.



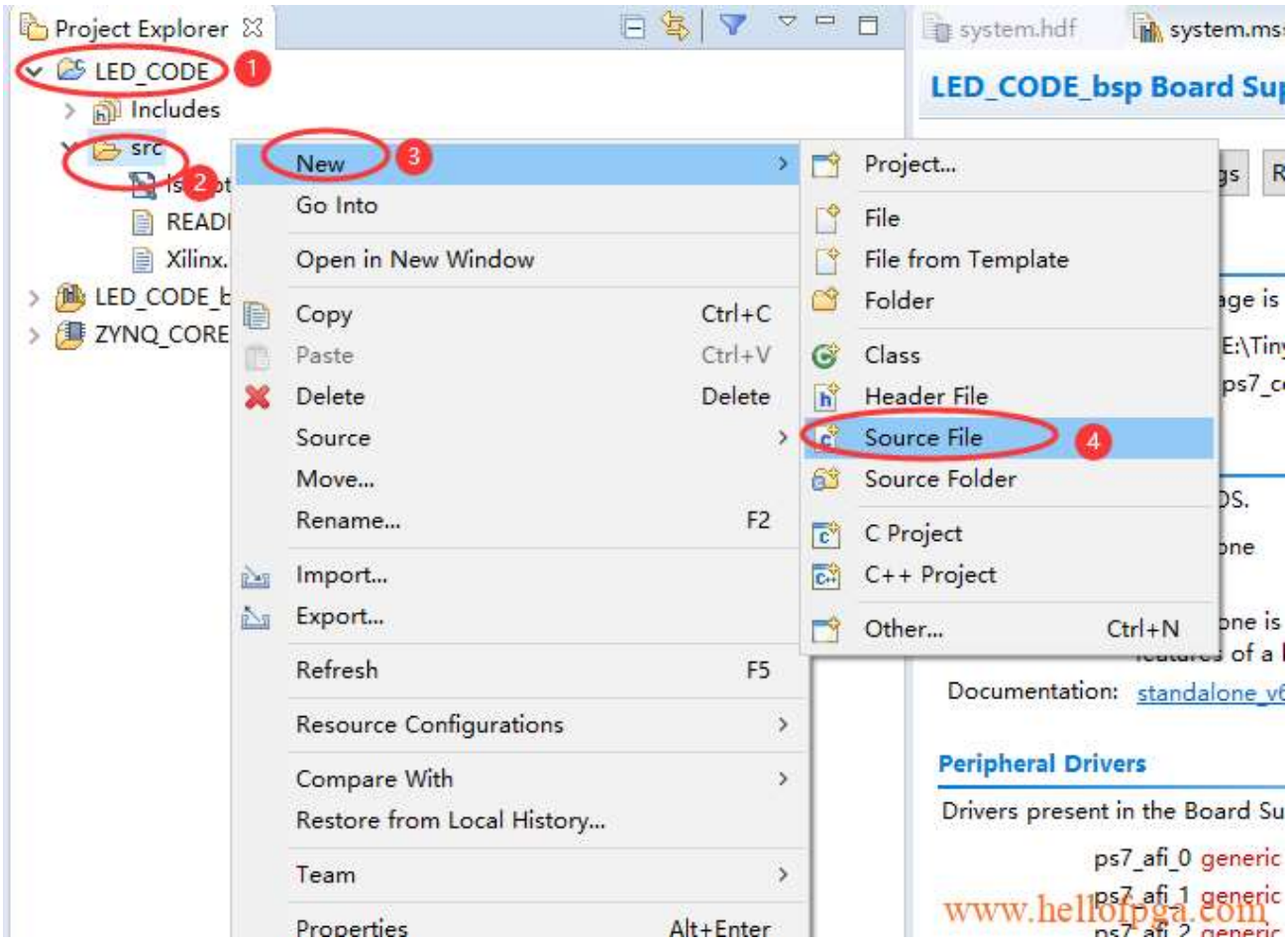
4) Enter your own project name in New Project Name and click NEXT



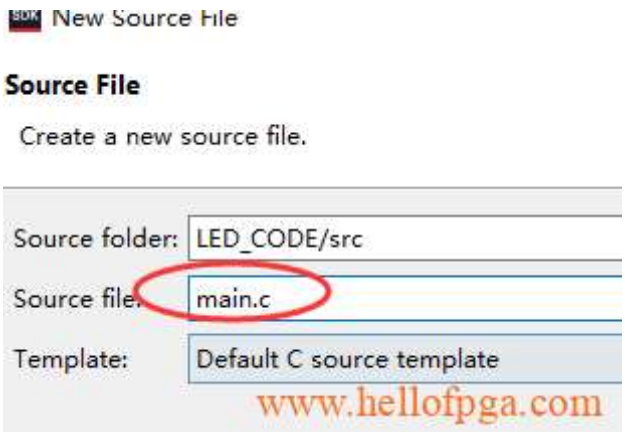
5) Select the empty project and click Finish



6) Add main.c file to the project src—>New—>Source File as shown in the figure below



7) Fill in main.c in the pop-up window and save



8). Open the main.c you just created

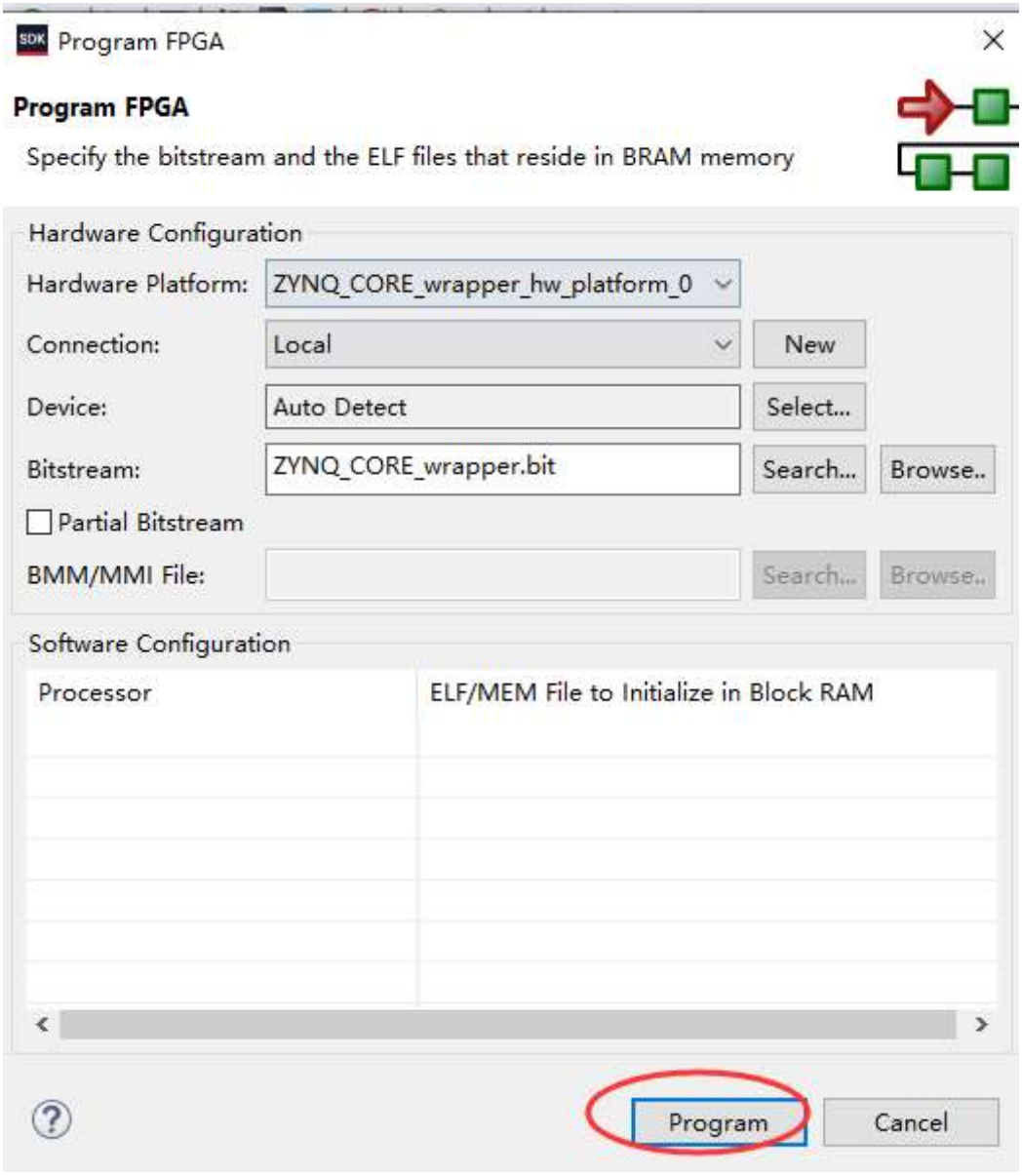
Then write the following code (the code is condensed on the basis of routines) There is a place worth noting that the IO port number of EMIO starts from 54, that is, the EMIO port created under my VIVADO, and the PS side is sorted from 54-55-56 (tips, less than 54 is MIO, that is, the hardware IO port of chip PS)

```
#include "xparameters.h"
#include "xgpiops.h"
#include "xstatus.h"
#include "xplatform_info.h"

#define LED1            54
#define LED2            55

#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
XGpioPs Gpio;


void Gpio_Init(void){
        XGpioPs_Config *ConfigPtr;

        ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
        XGpioPs_CfgInitialize(&Gpio, ConfigPtr,ConfigPtr-
>BaseAddr);

        XGpioPs_SetDirectionPin(&Gpio, LED1, 1);
        XGpioPs_SetOutputEnablePin(&Gpio, LED1, 1);

        XGpioPs_SetDirectionPin(&Gpio, LED2, 1);
        XGpioPs_SetOutputEnablePin(&Gpio, LED2, 1);
```

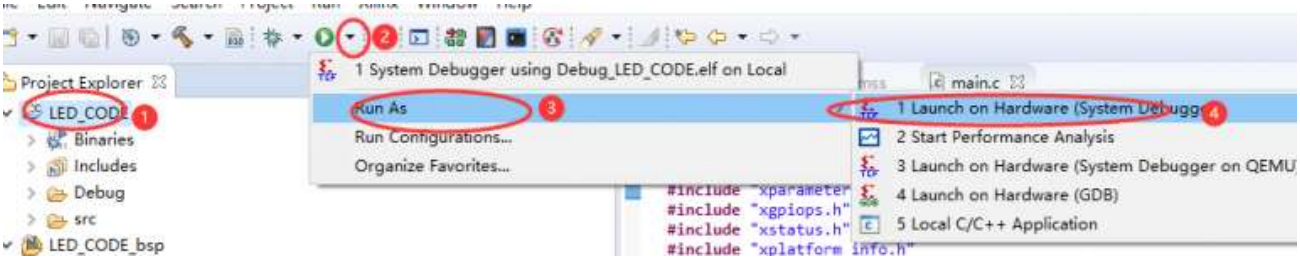
```
        XGpioPs_WritePin(&Gpio, LED1, 0);
        XGpioPs_WritePin(&Gpio, LED2, 0);
}


#define LED_DELAY     10000000
volatile int Delay;

int main(void)
{
        Gpio_Init();

        while(1){

                XGpioPs_WritePin(&Gpio, LED1, 0);
                XGpioPs_WritePin(&Gpio, LED2, 1);

                for (Delay = 0; Delay < LED_DELAY; Delay++);

                XGpioPs_WritePin(&Gpio, LED1, 1);
                XGpioPs_WritePin(&Gpio, LED2, 0);

                for (Delay = 0; Delay < LED_DELAY; Delay++);

        };

        return 0;
}
```

4. 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

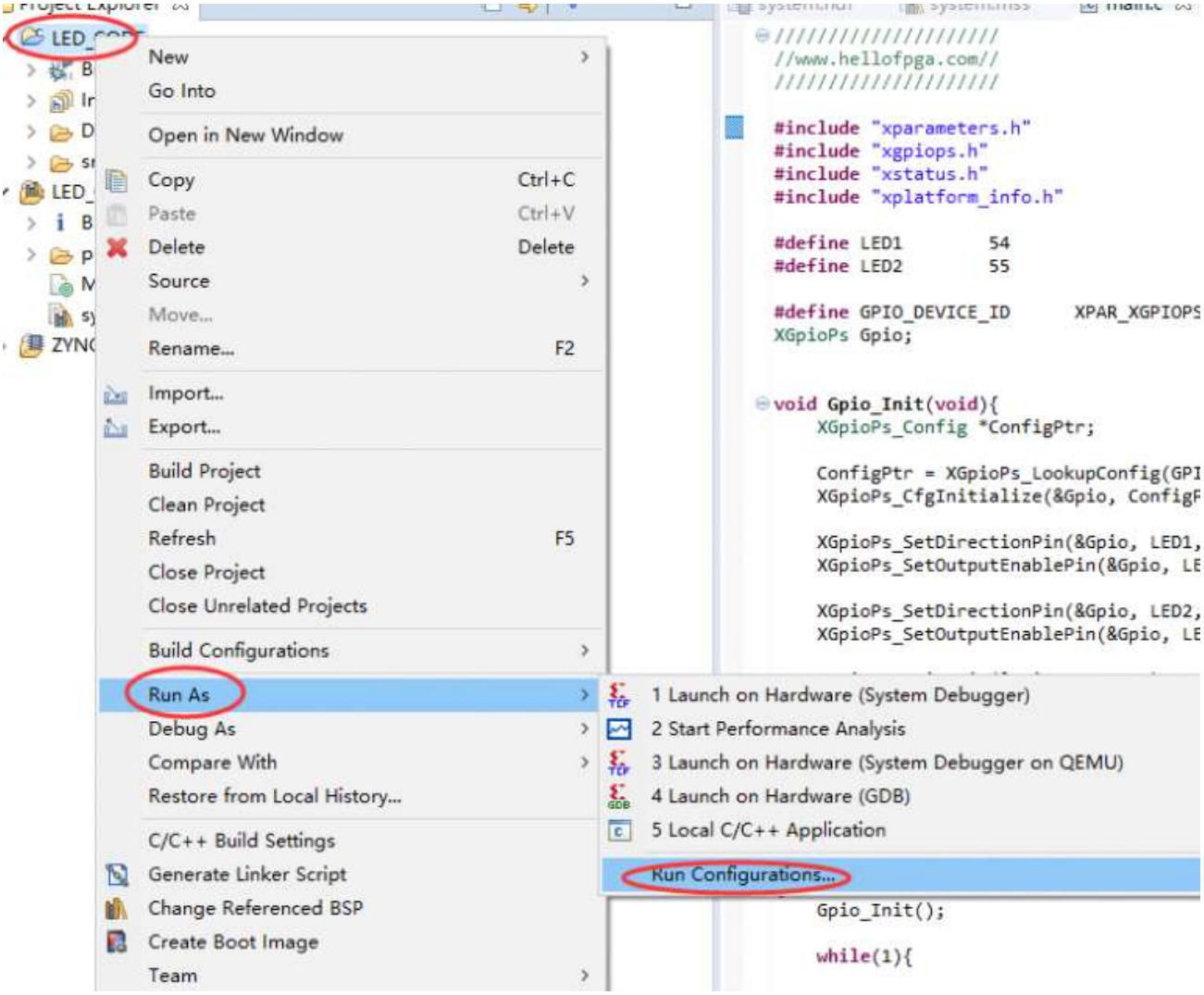


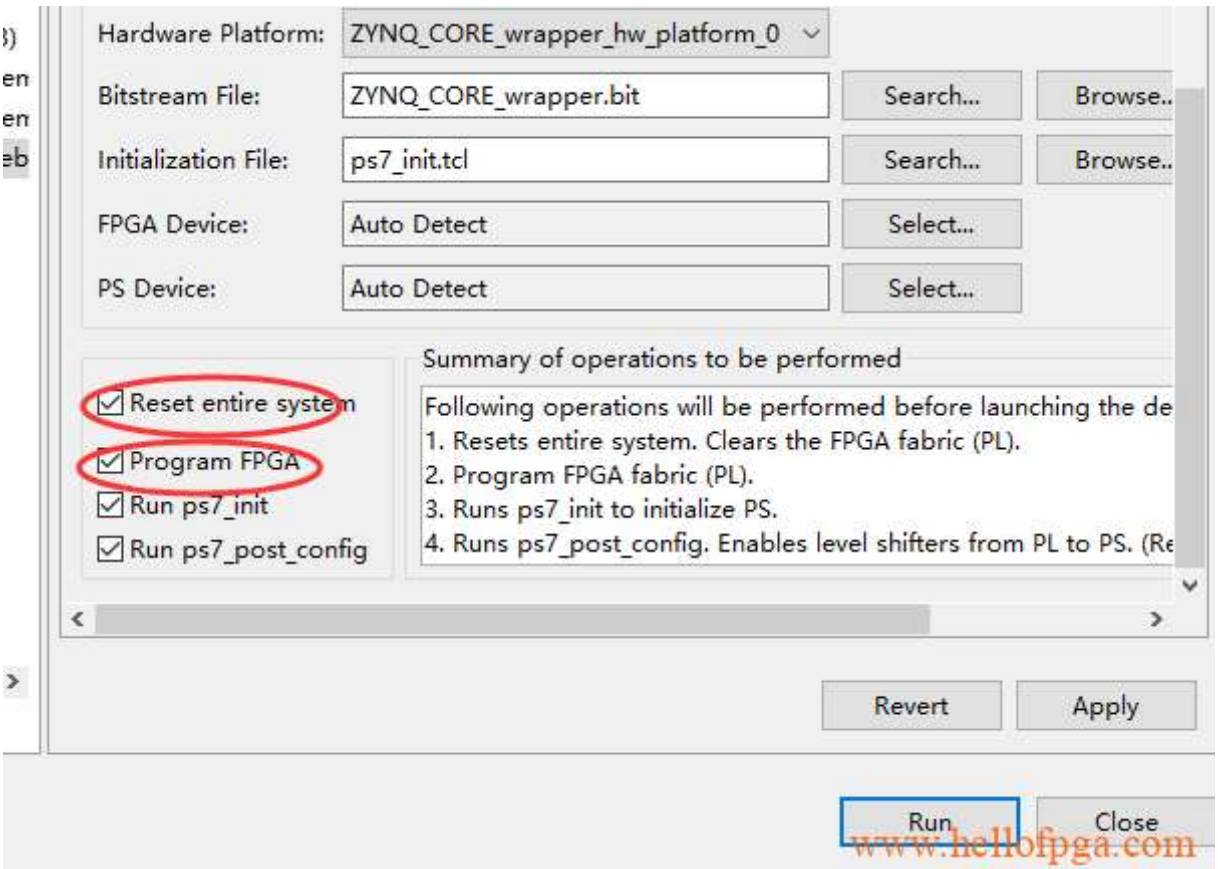
2) Select the GPIO project we generated, expand the icon to the right of the green arrow (RUN), and select Run As→1 Launch on Hardware (System Debugger)



You can see the two LED lights on the board blinking alternately

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


New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Build Configurations
Run As
Debug As
Compare With
Restore from Local History...
C/C++ Build Settings
Generate Linker Script
Change Referenced BSP
Create Boot Image
Team
1 Launch on Hardware (System Debugger)
2 Start Performance Analysis
3 Launch on Hardware (System Debugger on QEMU)
4 Launch on Hardware (GDB)
5 Local C/C++ Application
Run Configurations...
//www.hellofpga.com//
#include "xparameters.h"
#include "xgpiops.h"
#include "xstatus.h"
#include "xplatform_info.h"
#define LED1 54
#define LED2 55
#define GPIO_DEVICE_ID XPAR_XGPIOPS
XGpioPs Gpio;
void Gpio_Init(void){
XGpioPs_Config *ConfigPtr;
Gpio_Init();
while(1){


Hardware Platform: ZYNQ_CORE_wrapper_hw_platform_0
Bitstream File: ZYNQ_CORE_wrapper.bit
Search...
Browse...
Initialization File: ps7_init.tcl
Search...
Browse...
FPGA Device: Auto Detect
Select...
PS Device: Auto Detect
Select...
Summary of operations to be performed
Reset entire system
Program FPGA
Run ps7_init
Run ps7_post_config
Following operations will be performed before launching the de
1. Resets entire system. Clears the FPGA fabric (PL).
2. Program FPGA fabric (PL).
3. Runs ps7_init to initialize PS.
4. Runs ps7_post_config. Enables level shifters from PL to PS. (Re
Revert
Apply
Run
Close
www.hellofpga.com

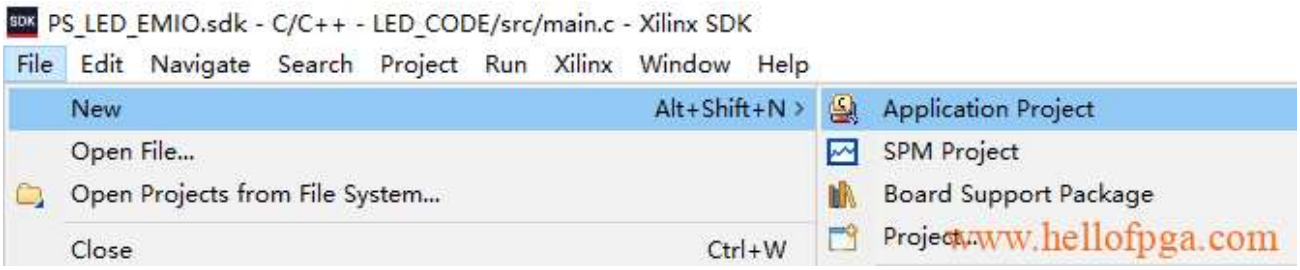
Then click APPLY and then select Run As→1 Launch on Hardware (System Debugger) to see if the download is successful (if it still doesn't work, please power off the board and try again, generally this problem occurs because of conflicts with previously run programs when debugging)

5. Create an image file of the TF card

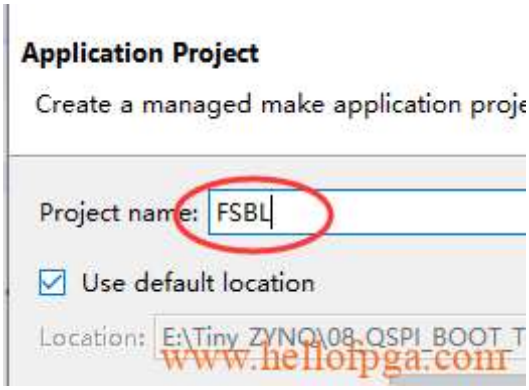
Create an image file in two steps: 1. Create the FSBL module, 2. Create the image(boot.bin) file

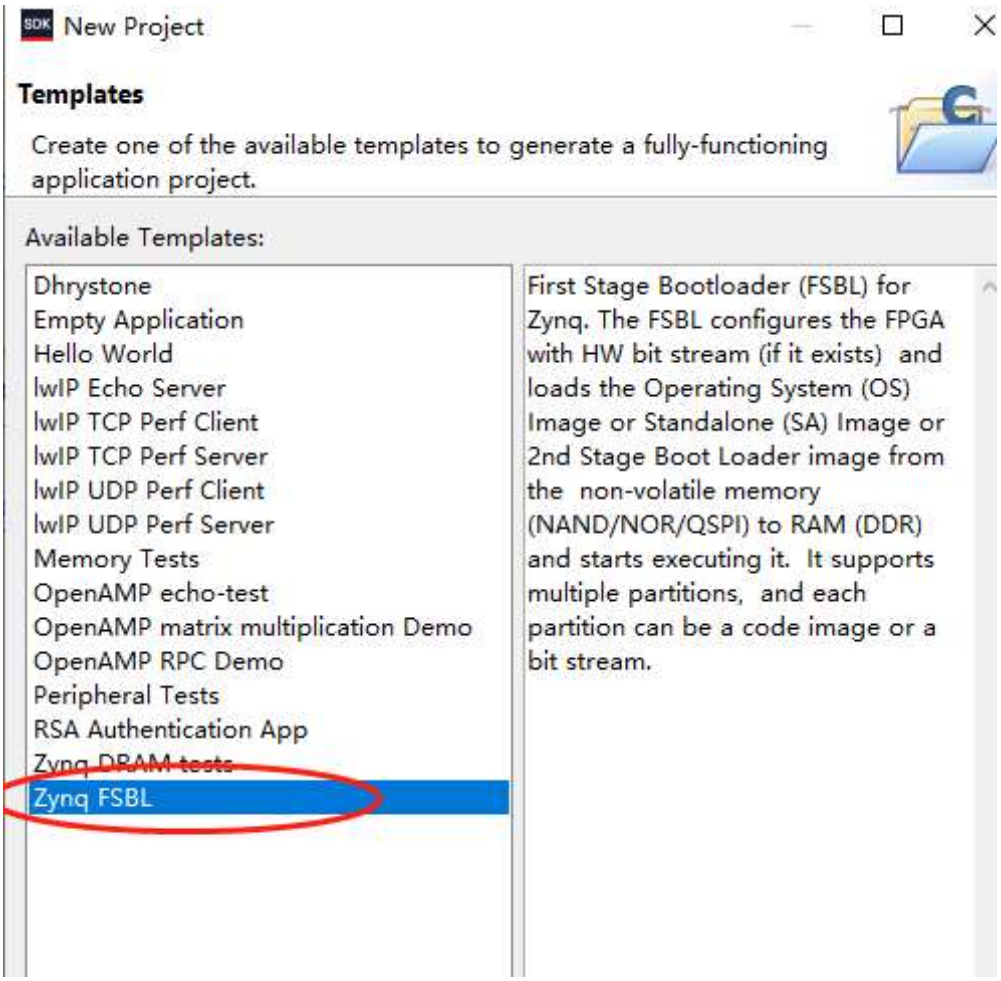
When running a program on zynq, a file must be used in the loading process, that is, fsbl, which is the loader of the first stage of zynq startup, and after the fsbl stage, the system can run the naked runner or the u-boot that boots the operating system

1) Create an FSBL: Create a new project in the SDK as follows



Set the project name fsbl and click Next to select the FSBL template

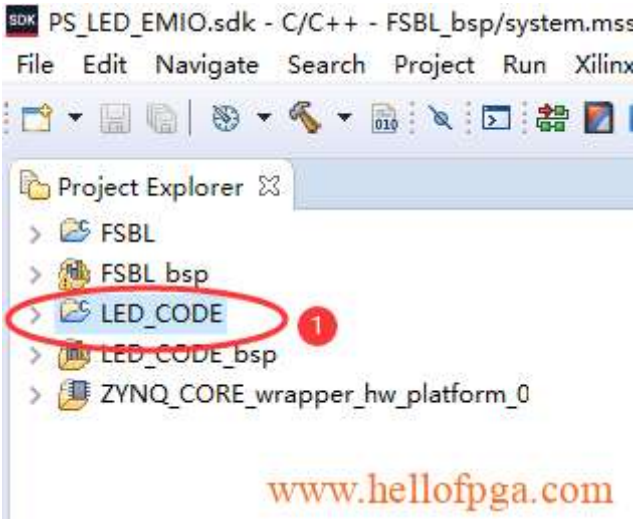




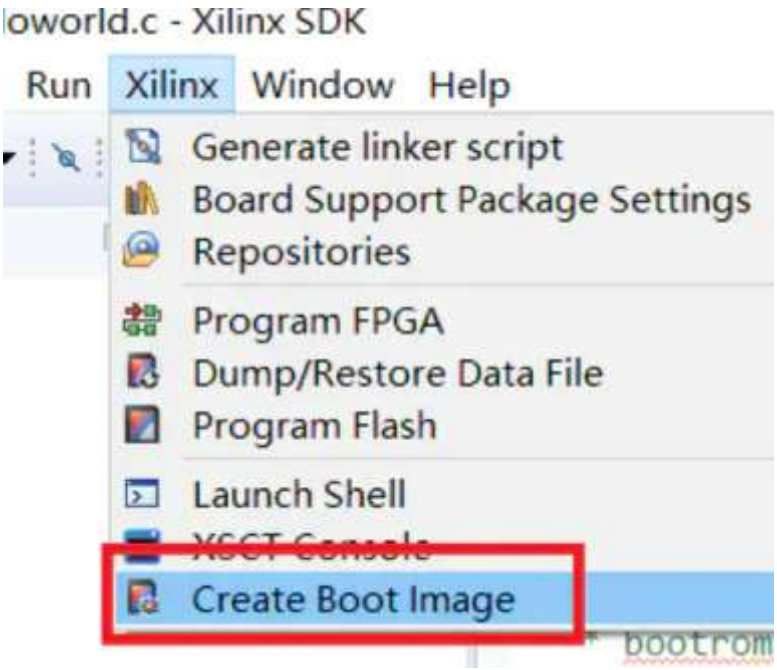
2) Create a TF card image file

BOOT.bin consists of three files: fsbl.elf file, the bit file of the FPGA generated by vivado, and the elf file of the app project (here the elf of the led project). When generating an image, you have to fill in the paths of three files separately, and there is also a lazy way here, that is, select our APP project before clicking Create boot image, so that the system will automatically add the path for us. As shown below

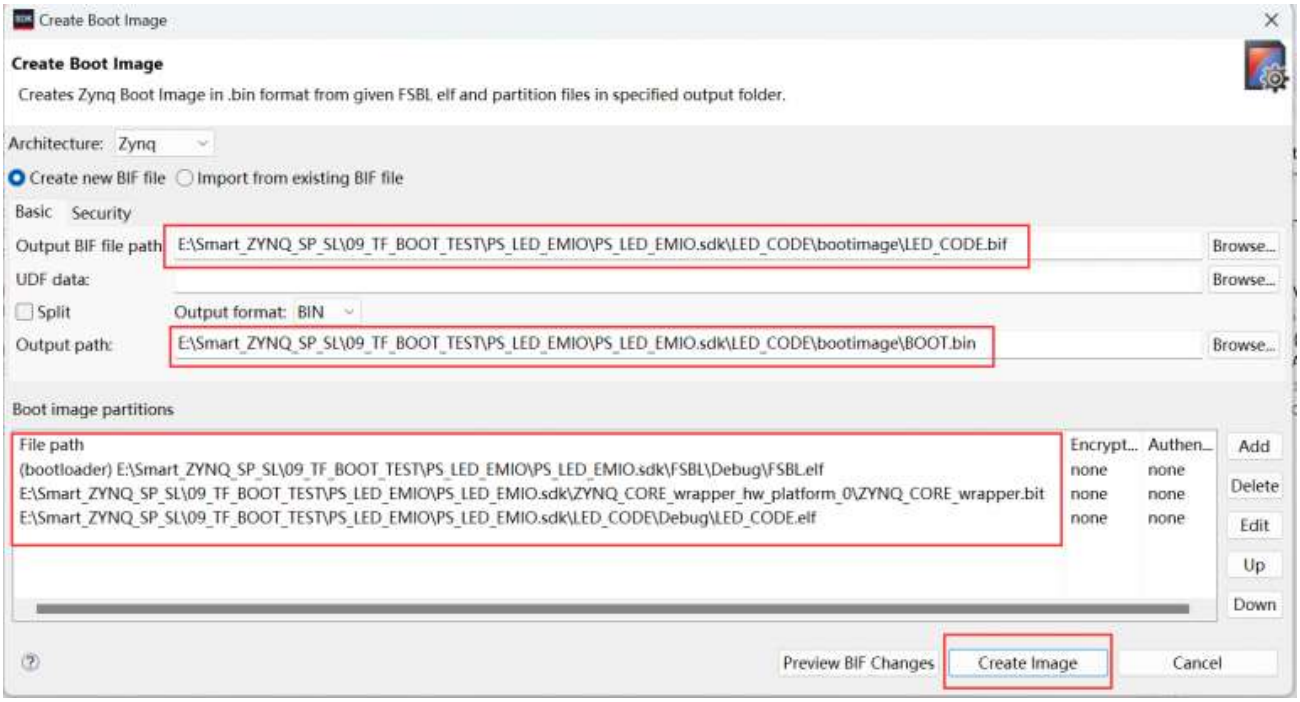
a) First select our APP project directory, here is the LED_CODE (not FSBL nor BSP and platform) **If it has been selected at the beginning, please select any directory above and below with the mouse, and then select back to the APP directory, otherwise the path may not be automatically loaded**



b) 点xilinx –>create boot image

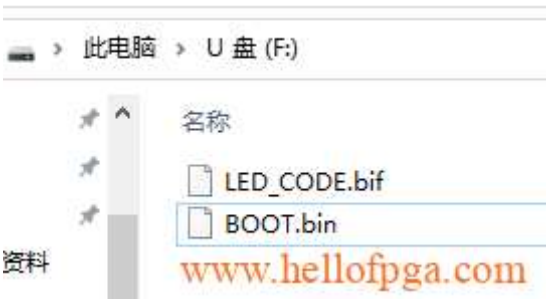


Set the output path of the boot .bin in the interface, and the path of the three files needed to generate the boot .bin (here is a lazy way, that is, select our APP project before clicking Create Boot Image), **as shown in the following figure, if the path of the three files does not appear, please try the previous step. If there are three file paths, click create image directly to generate the image**



6. Verify that the SD TF card boots

- 1) Copy these two files into the empty TF card in the ouput bif and boot .bin file in the export folder of the boot image just now, (remember to format in FAT32 format)



2. Insert the TF card into the card slot of the motherboard

3. Set the board to TF card boot mode as shown in the image below

4. Power off again (**or press the POR_RST button**), under normal circumstances, the two indicators of the system begin to flash alternately, indicating that the TF card has started successfully

If the startup is unsuccessful, there are several possible conditions

a. The format of the TF card must be FAT32 format

b. TF card has multiple partitions, which need to be unified into one and then formatted, (this will happen if you have made a boot disk with tools such as old wool peach)

c. TF card is not supported, try to replace different brands of TF cards with different capacity sizes, large capacity, the higher the speed TF card compatibility, the worse the compatibility (PS My side test 128g SanDisk high-speed disk can also support)

The complete project is as follows

09_TF_BOOT_TEST_XC7Z020 **Download**

SMART ZYNQ SP & SL