APRIL 2023, 4 BY ACKYE

Smart ZYNQ (SP&SL version) Project VIII solidifies the program to QSPI FLASH

ZYNQ and many FPGAs, the binary files downloaded from JTAG, are lost after power failure, in order to make the program still work after power off and restart, the program needs to be solidified, this article will demonstrate how to solidify the code to the QSPI FLASH chip of the board, so that the system boots from the flash chip loader every time it boots

(Note: Generally, debugging does not require a curing program, and when the program needs to be stored permanently, you can consider solidifying the program)

This project is modified from Chapter 7 and can be viewed optionally

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

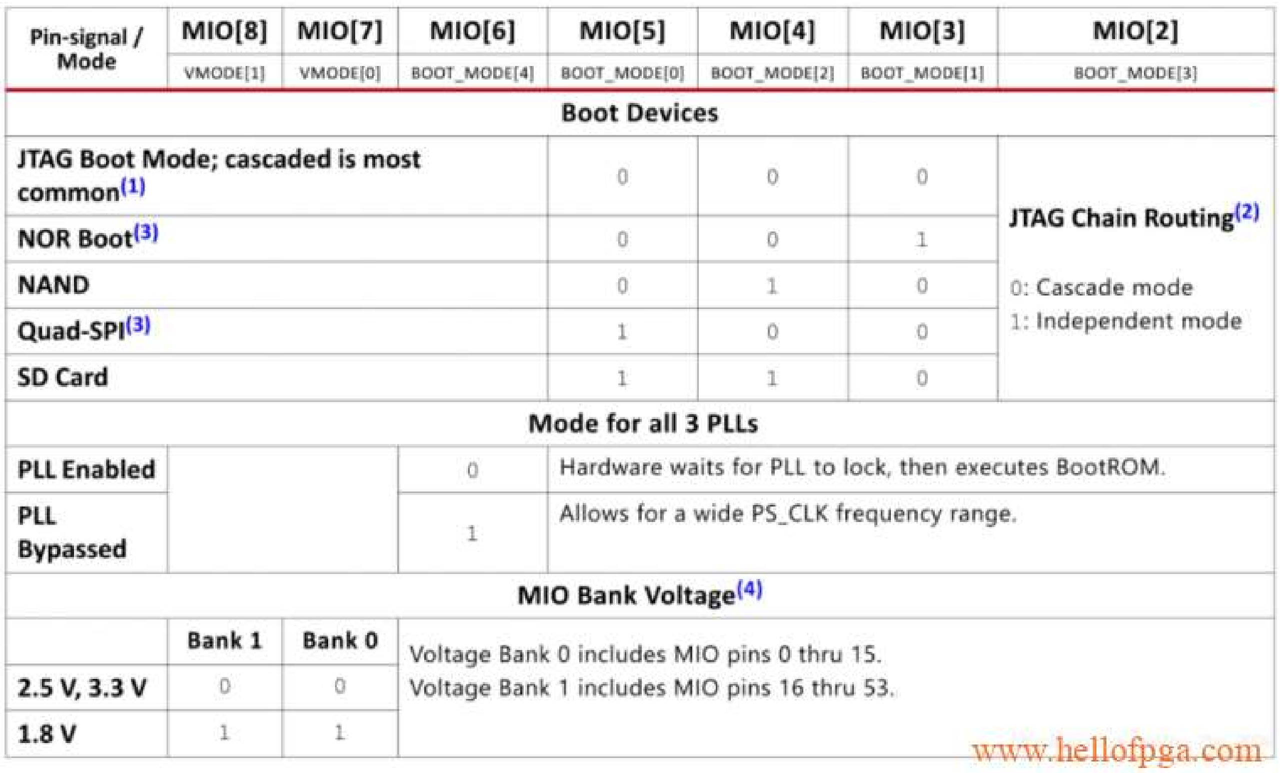
1. First understand the boot process of ZYNQ, the boot process of ZYNQ is mainly based on the ARM core, after power-on, the hardware first reads the level status of the specific pin of the PS end of the ARM core to determine whether the system starts the system from NAND, QSPI, SD Card or JTAG**. The way of starting can be adjusted by the paddle switch of the board**

2. ARM's ordinary FPGA (non-ZYNQ) curing is to directly write the BIT file to FLASH through QSPI, **but for ZYNQ, there must be the cooperation of the PS end to solidify the program**

2. Hands-on demonstration

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 of power-on on the system, which is used to distinguish different power-up methods.

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



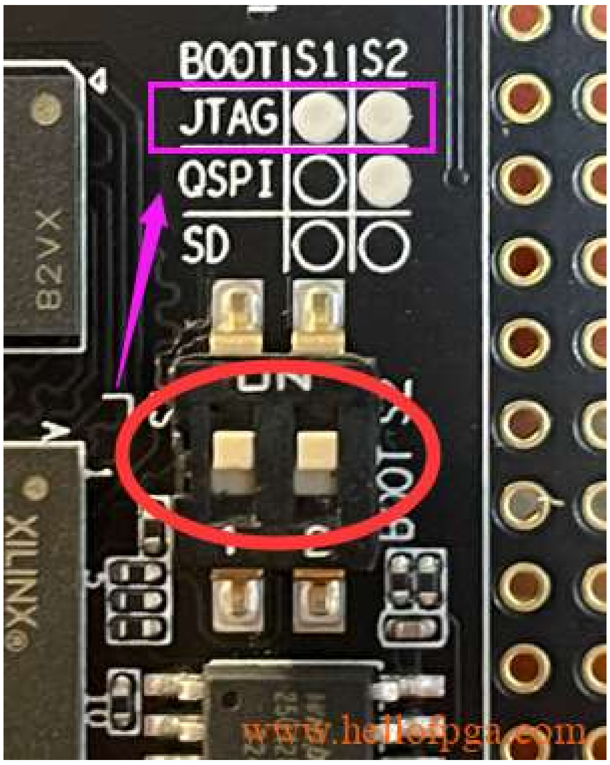
Pin-signal / Mode	MIO[8]	MIO[7]	MIO[6]	MIO[5]	MIO[4]	MIO[3]	MIO[2]
	VMODE[1]	VMODE[0]	BOOT_MODE[4]	BOOT_MODE[0]	BOOT_MODE[2]	BOOT_MODE[1]	BOOT_MODE[3]
Boot Devices							
JTAG Boot Mode; cascaded is most common[1]				0	0	0	**JTAG Chain Routing**[2] 0: Cascade mode 1: Independent mode
NOR Boot[3]				0	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

to start the system from JTAG mode to enter debug mode

In order to solidify the FLASH chip, we need to make the board temporarily start from JTAG debug mode (temporary), and only need to keep the board DIP switch in

the state where S1 and S2 are ON before powering on before performing the power-on operation. As shown in the following figure

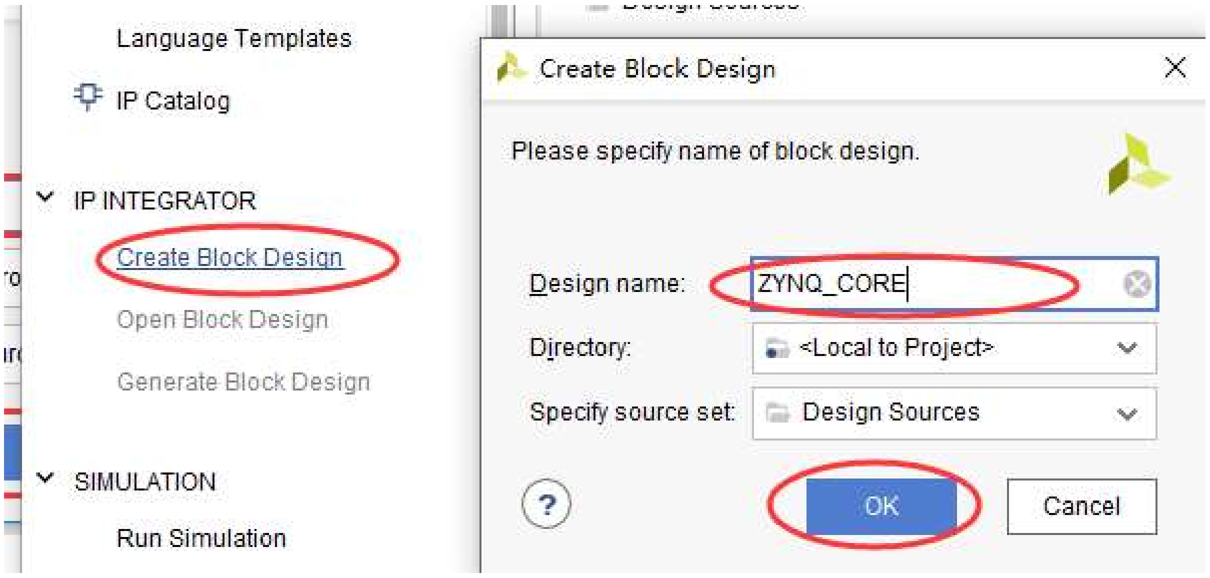


2.1 Software part configuration

This project demonstrates the curing function on the basis of Project 7, and most of the content is repeated for selective viewing

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


Language Templates

IP Catalog

IP INTEGRATOR

Create Block Design

Open Block Design

Generate Block Design

SIMULATION

Run Simulation

Create Block Design

Please specify name of block design.

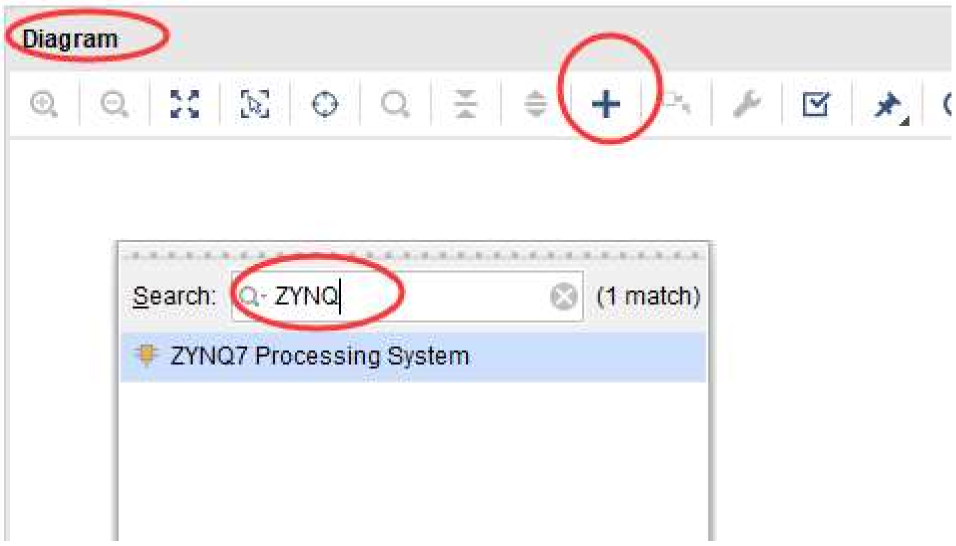
Design name: ZYNQ_CORE

Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

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

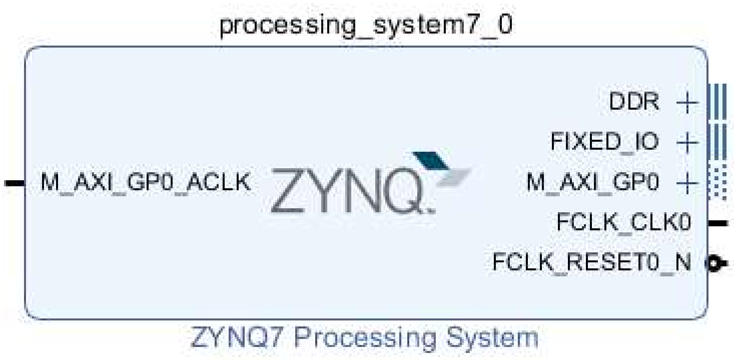


Diagram

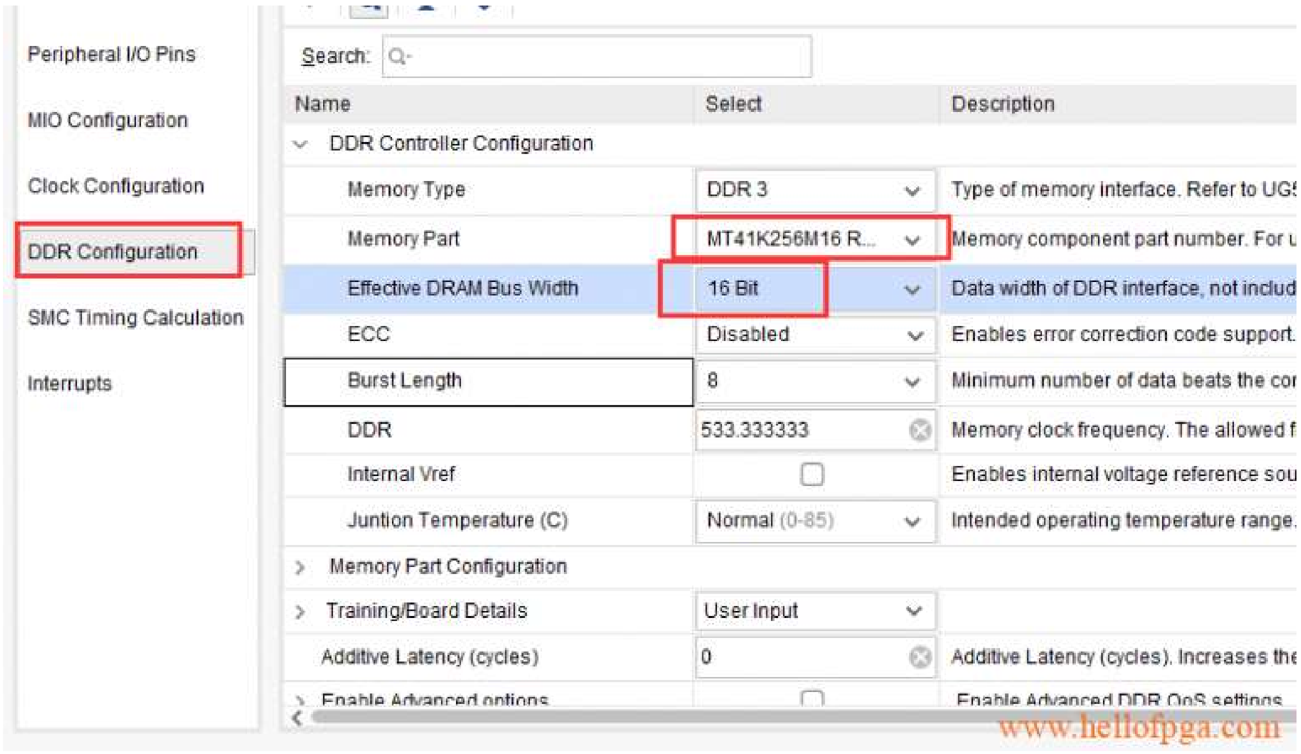
Search: ZYNQ (1 match)

ZYNQ7 Processing System

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

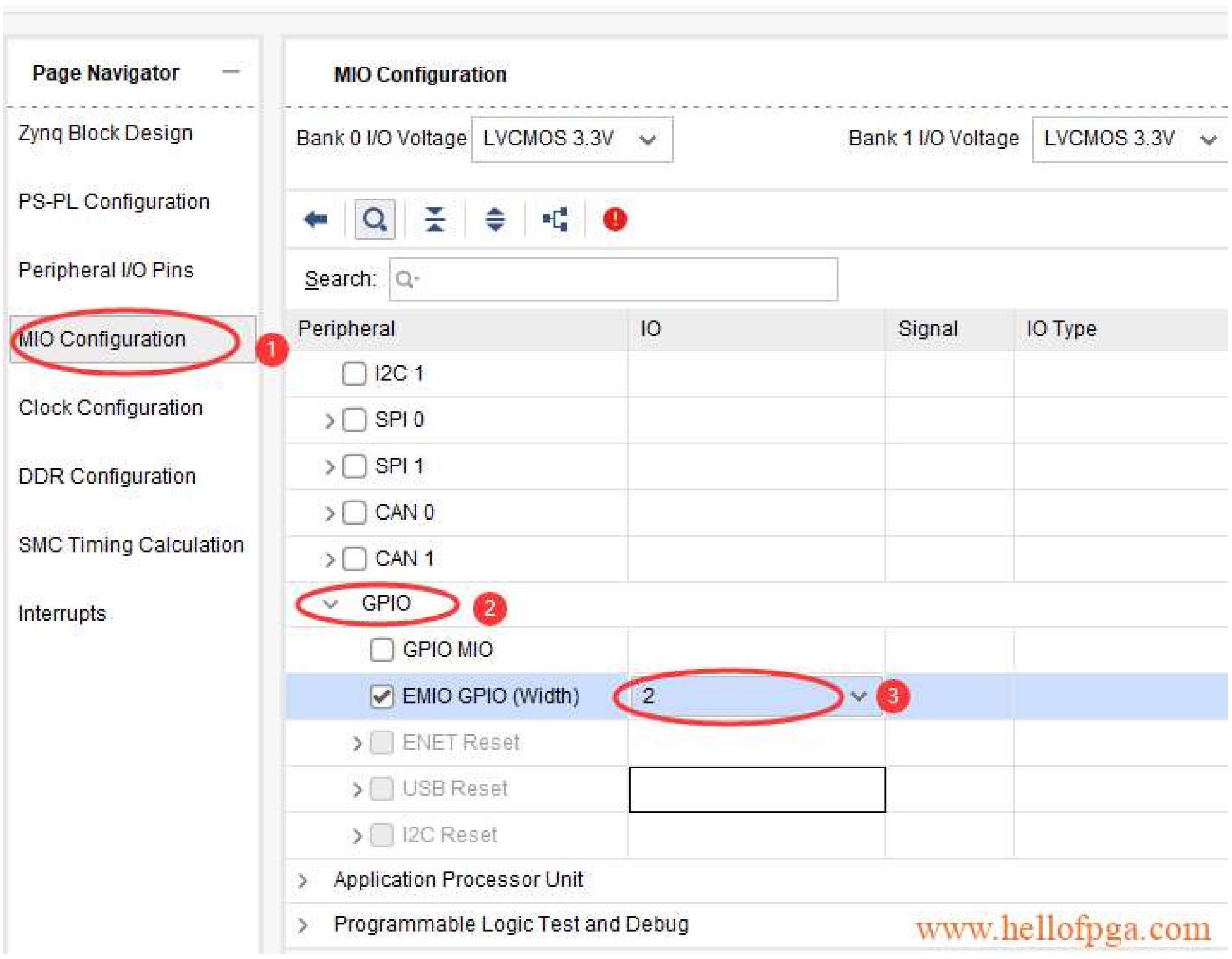


4) Find DDR Configuration →DDR Controller Configuration →DDR3 in the pop-up window in turn, select the corresponding DDR3 according to the DDR on your board in the Memory Part drop-down menu, the model used in this experiment: MT41K256M16RE-125, select 16bit of data width and finally click "OK", as shown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



Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Type of memory interface. Refer to UG
Memory Part	MT41K256M16 R...	Memory component part number. For u
Effective DRAM Bus Width	16 Bit	Data width of DDR interface, not includ
ECC	Disabled	Enables error correction code support.
Burst Length	8	Minimum number of data beats the co
DDR	533.333333	Memory clock frequency. The allowed f
Internal Vref	☐	Enables internal voltage reference sou
Juntion Temperature (C)	Normal (0-85)	Intended operating temperature range.
Memory Part Configuration		
Training/Board Details	User Input	
Additive Latency (cycles)	0	Additive Latency (cycles). Increases the
Enable Advanced options	☐	Enable Advanced DDR QoS settings

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-curing is necessary)



Page Navigator

Zynq Block Design

PS-PL Configuration

Peripheral I/O Pins

MIO Configuration

Clock Configuration

DDR Configuration

SMC Timing Calculation

Interrupts

MIO Configuration

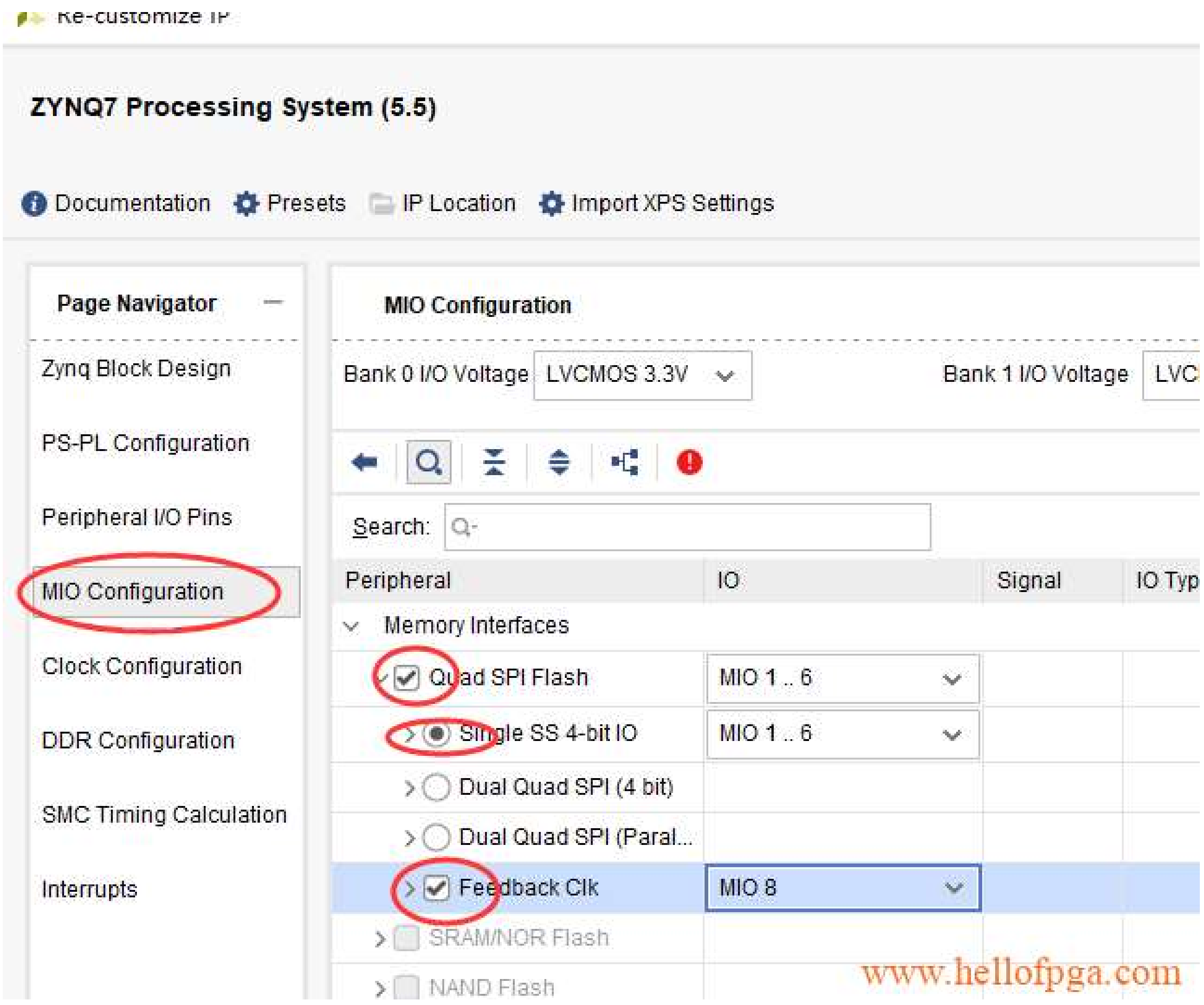
Bank 0 I/O Voltage LVCMOS 3.3V

Bank 1 I/O Voltage LVCMOS 3.3V

Search:

Peripheral	IO	Signal	IO Type
I2C 1			
SPI 0			
SPI 1			
CAN 0			
CAN 1			
GPIO			
GPIO MIO			
✓ EMIO GPIO (Width)	2		
ENET Reset			
USB Reset			
I2C Reset			
Application Processor Unit			
Programmable Logic Test and Debug			

5). 重要的一步在block design 的设置界面 使能QSPI的功能 如下图所示（当QSPI 时钟大于40MHZ的时候 就需要勾选Feedback clk）


Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

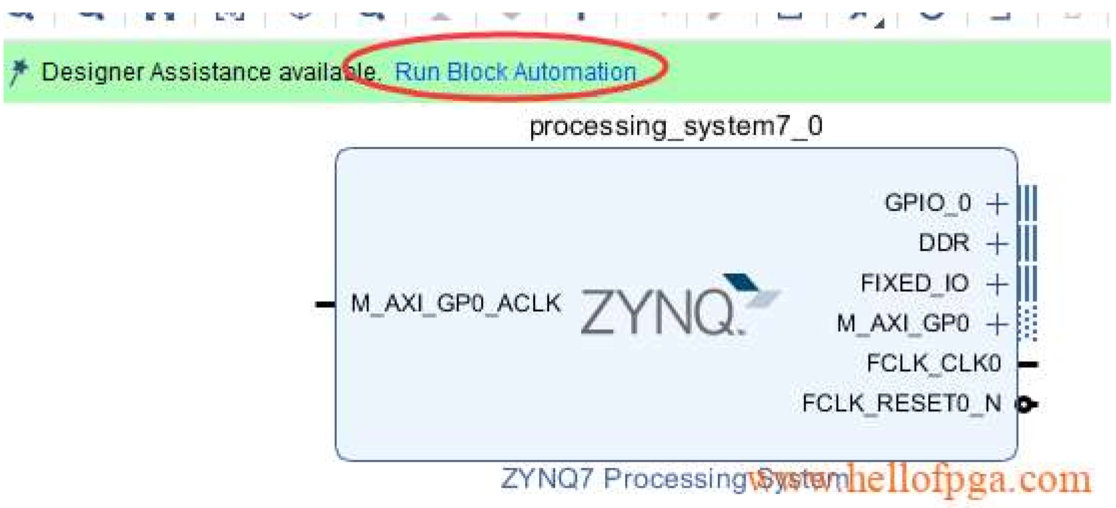
MIO Configuration

Bank 0 I/O Voltage: LVCMOS 3.3V   Bank 1 I/O Voltage: LVC

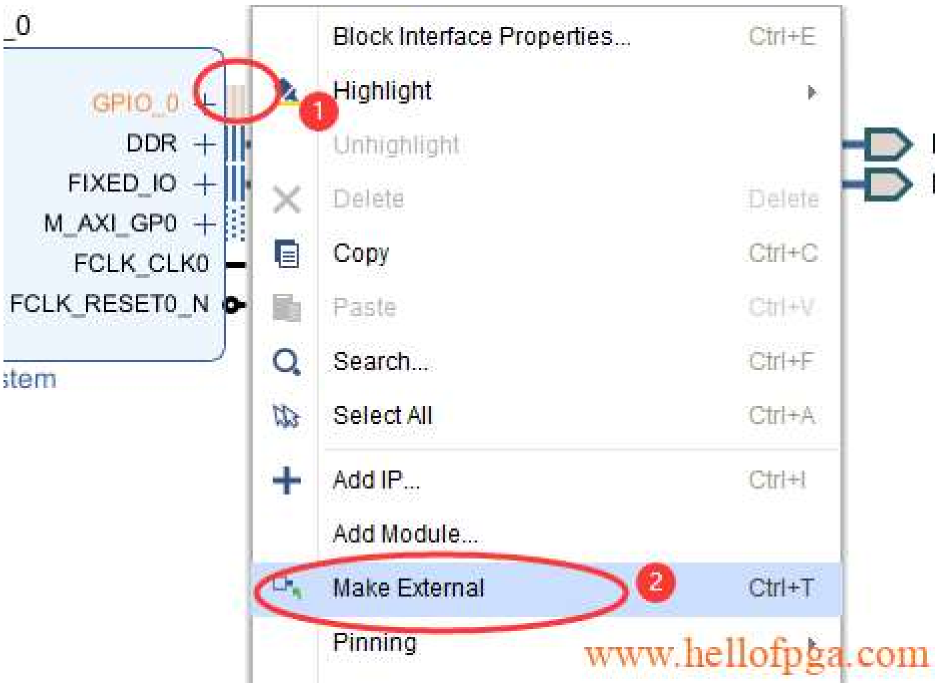
Search:

Peripheral	IO	Signal	IO Typ
Memory Interfaces			
✓ Quad SPI Flash	MIO 1 .. 6		
● Single SS 4-bit IO	MIO 1 .. 6		
○ Dual Quad SPI (4 bit)			
○ Dual Quad SPI (Paral...			
✓ Feedback Clk	MIO 8		
SRAM/NOR Flash			
NAND Flash			

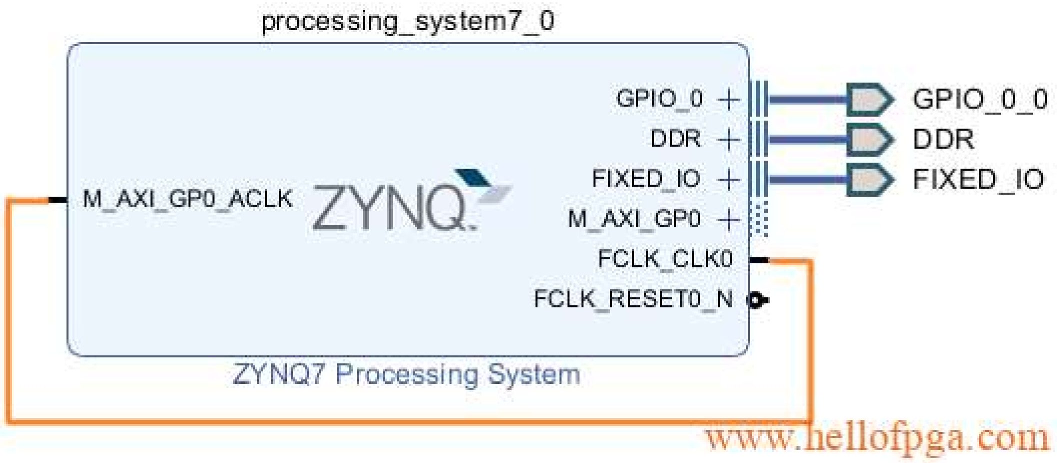
6）最后 点击“Run Block Automation”如下图所示。在弹出的选项中保持默认，点击“OK”，即可完成对ZYNQ7 Processing System的配置


Designer Assistance available. Run Block Automation

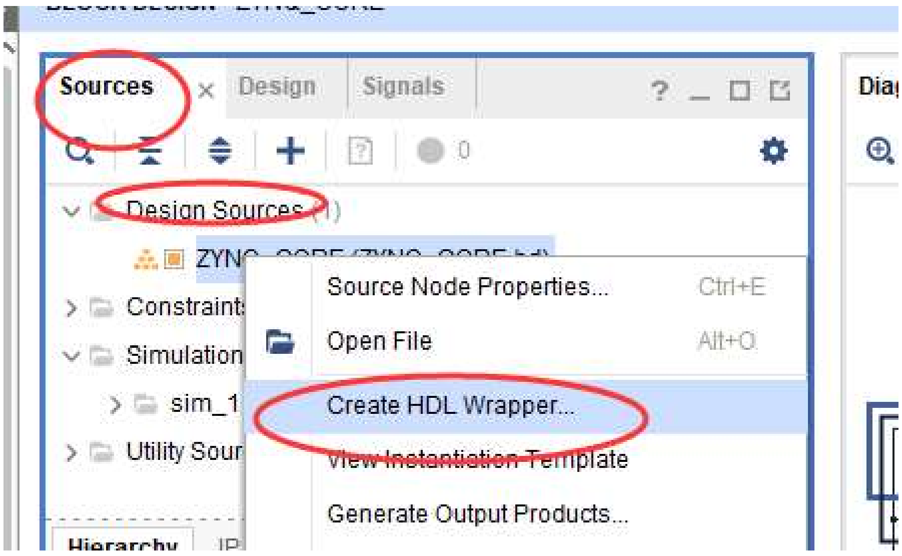
7）将刚才添加EMIO GPIO 引出 右键GPIO_0—->Make External



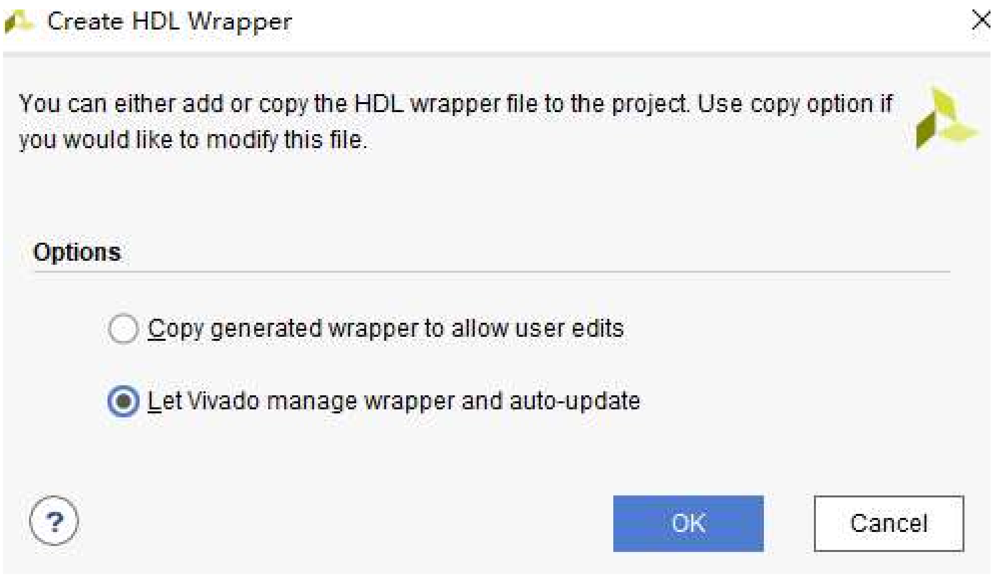
8）用线将M_AXI_GP0_ACLK与 FCLK_CLK0连接起来



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



10) Keep the default in the pop-up dialog



Create HDL Wrapper

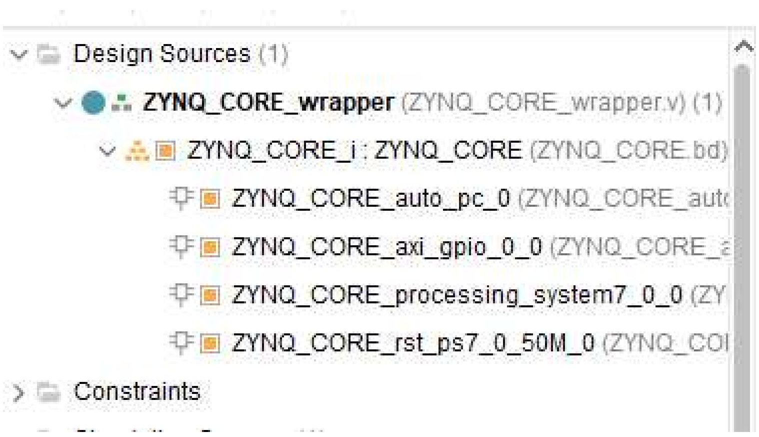
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

( ) Copy generated wrapper to allow user edits

(●) Let Vivado manage wrapper and auto-update

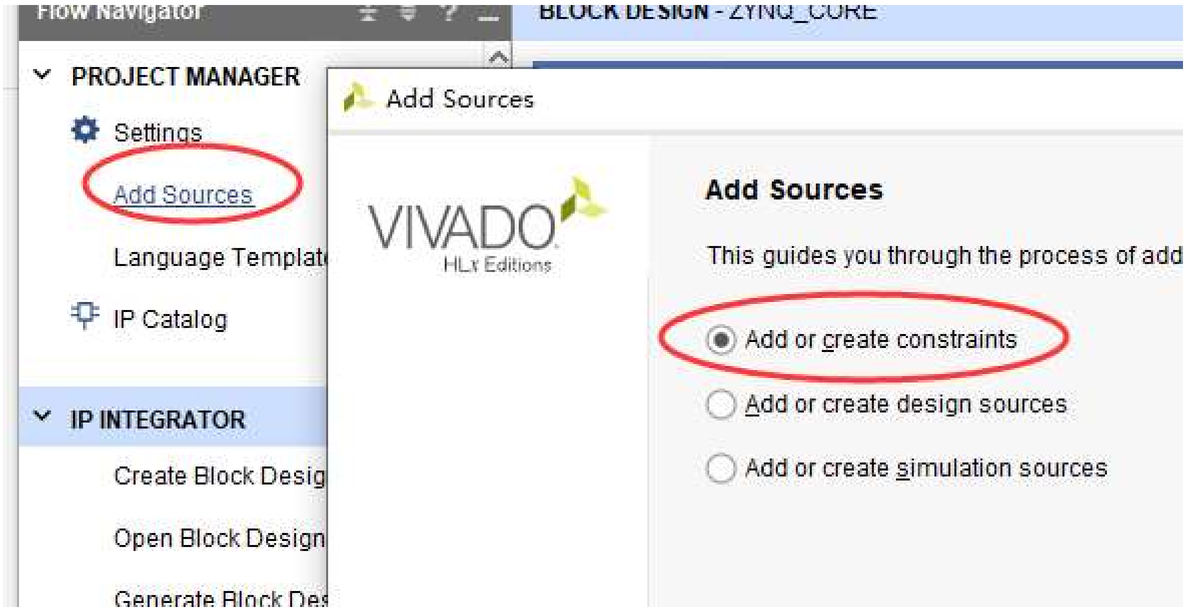
11) The software automatically generates HDL files for us



```
Design Sources (1)
  ZYNQ_CORE_wrapper (ZYNQ_CORE_wrapper.v) (1)
    ZYNQ_CORE_i : ZYNQ_CORE (ZYNQ_CORE.bd)
      ZYNQ_CORE_auto_pc_0 (ZYNQ_CORE_aut
      ZYNQ_CORE_axi_gpio_0_0 (ZYNQ_CORE_a
      ZYNQ_CORE_processing_system7_0_0 (ZY
      ZYNQ_CORE_rst_ps7_0_50M_0 (ZYNQ_COI
Constraints
```

2.2 Create a constraint file and define the pins

1) Add Source → Add or create constraints Click Next



Add Sources

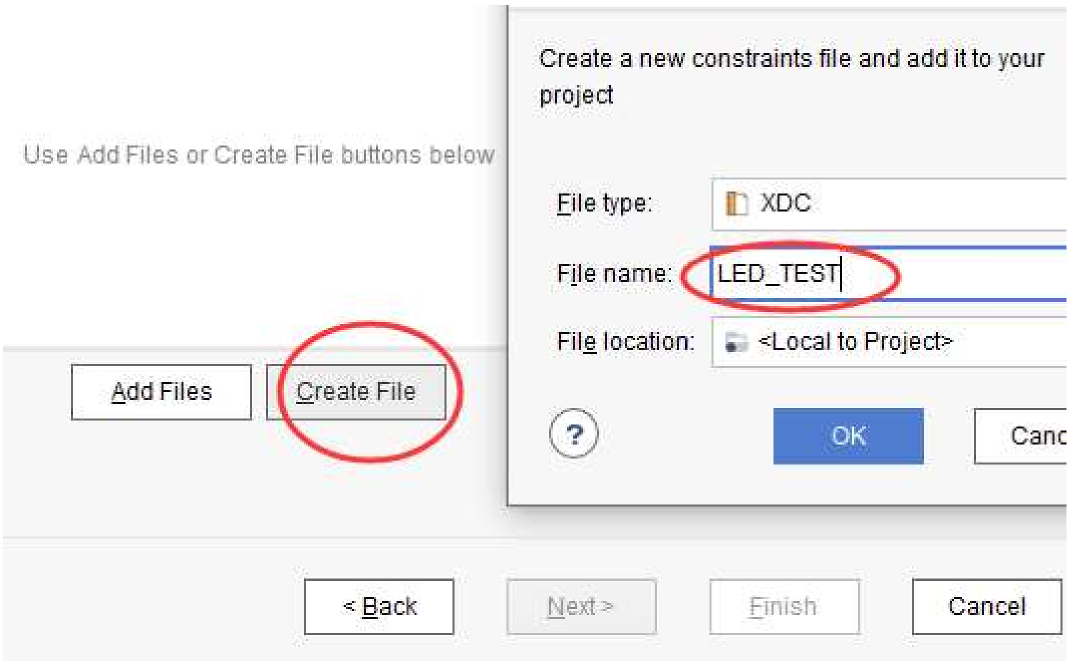
This guides you through the process of add

(● Add or create constraints)

○ Add or create design sources

○ Add or create simulation sources

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



Use Add Files or Create File buttons below

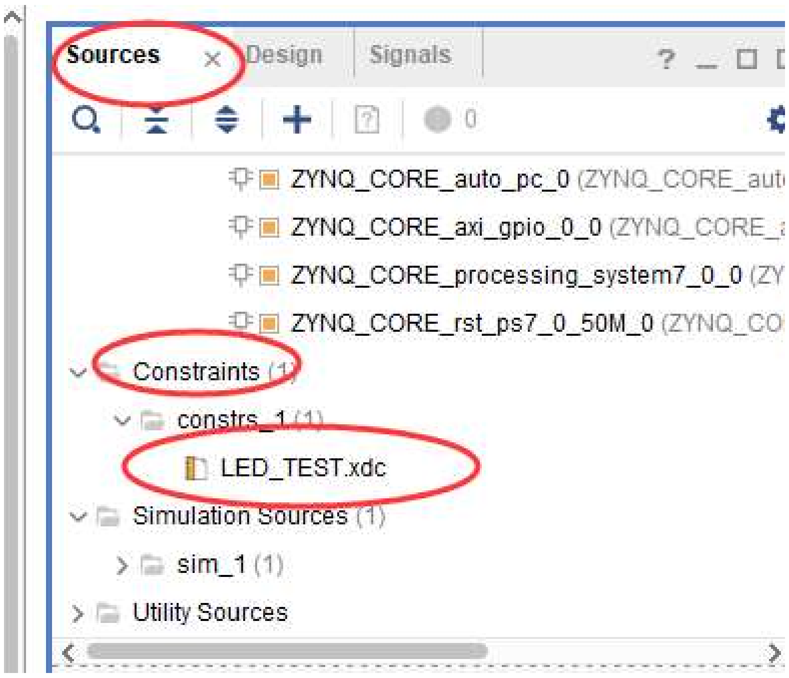
Create a new constraints file and add it to your project

File type: XDC

File name: LED_TEST

File location: <Local to Project>

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

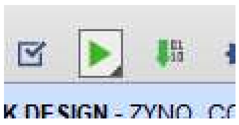

Sources
Design
Signals
ZYNQ_CORE_auto_pc_0 (ZYNQ_CORE_aut
ZYNQ_CORE_axi_gpio_0_0 (ZYNQ_CORE_
ZYNQ_CORE_processing_system7_0_0 (ZY
ZYNQ_CORE_rst_ps7_0_50M_0 (ZYNQ_CO
Constraints (1)
constrs_1 (1)
LED_TEST.xdc
Simulation Sources (1)
sim_1 (1)
Utility Sources

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）

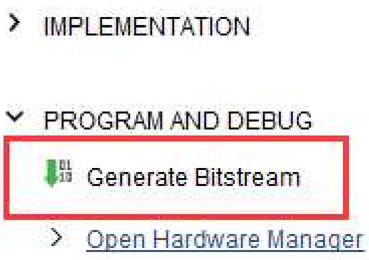
```
set_property IOSTANDARD LVCMOS33 [get_ports GPIO_0_0_tri_io[0]]
set_property IOSTANDARD LVCMOS33 [get_ports GPIO_0_0_tri_io[1]]

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]]
```

2.3 综合布线和生成bit文件

按下绿色箭头对工程进行编译

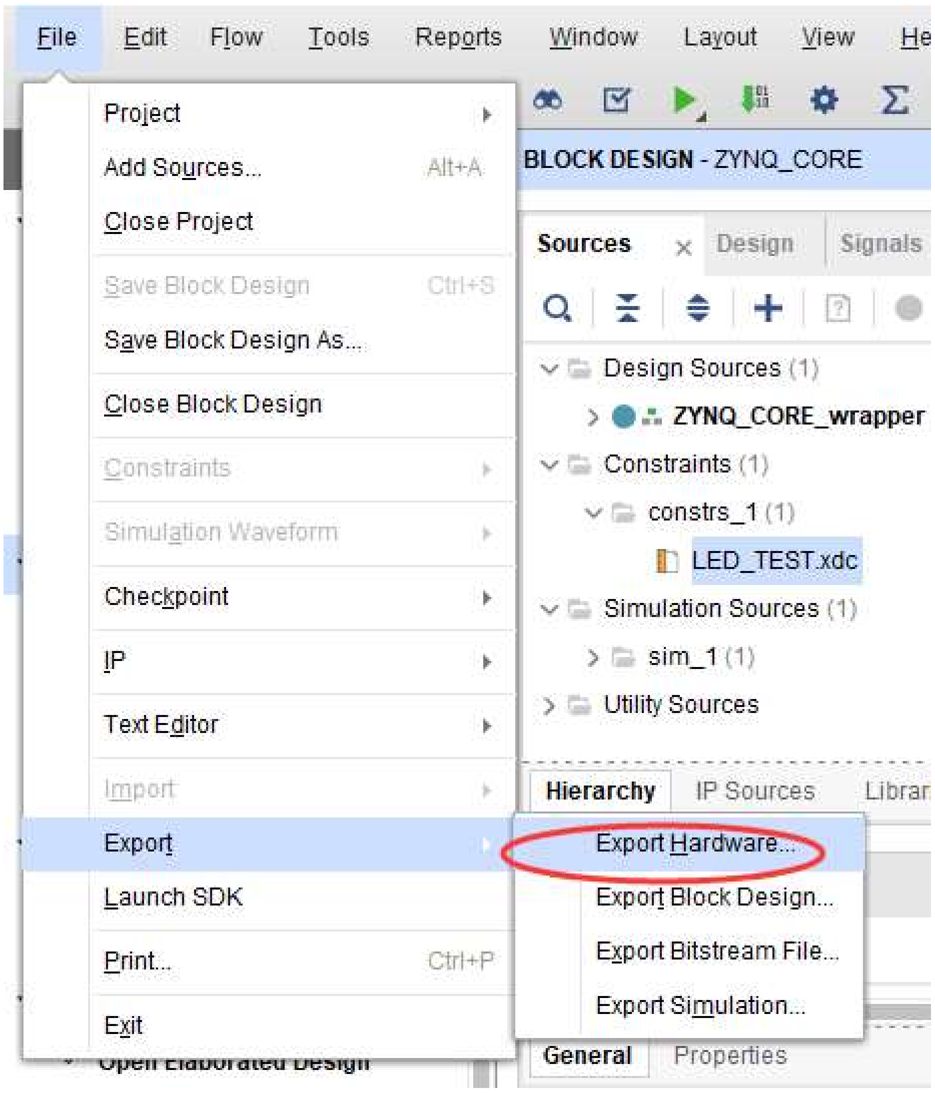


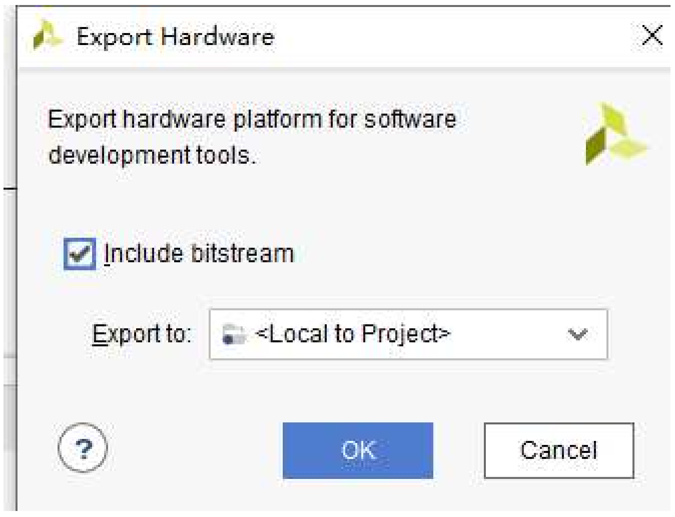
按下Generate Bitstream 完成综合以及生成bit文件



2.4 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。



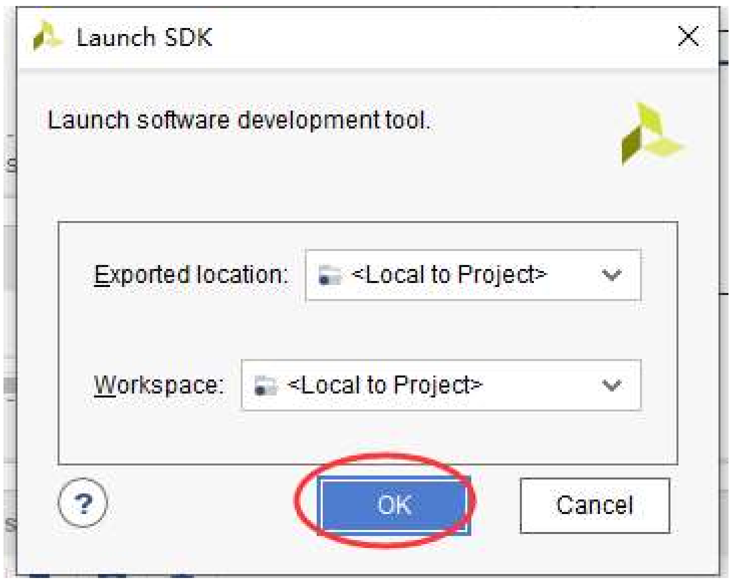

Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

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

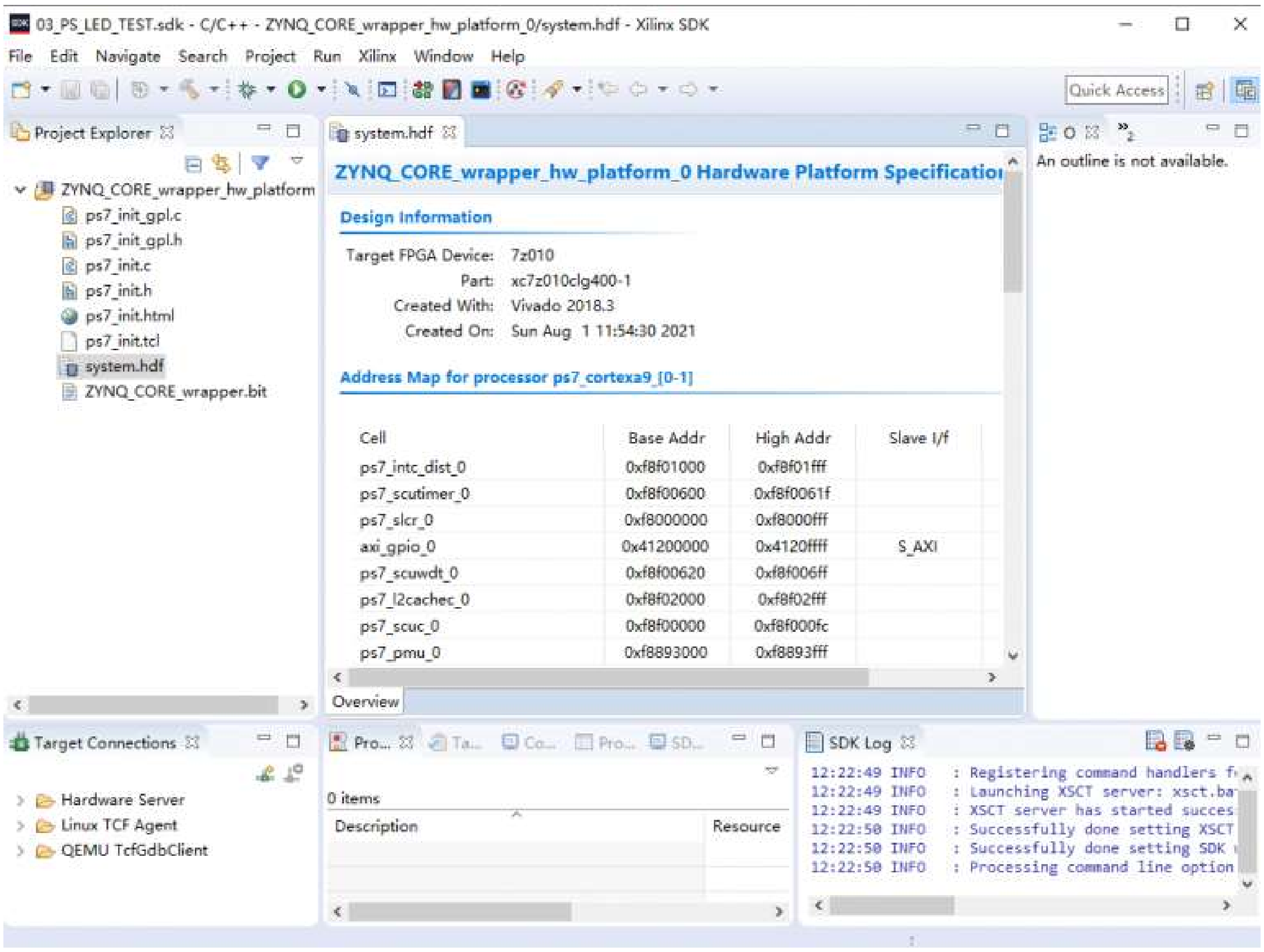

Launch SDK

Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

系统将自动打开SDK开发环境



ZYNQ_CORE_wrapper_hw_platform_0 Hardware Platform Specificatio

Design Information

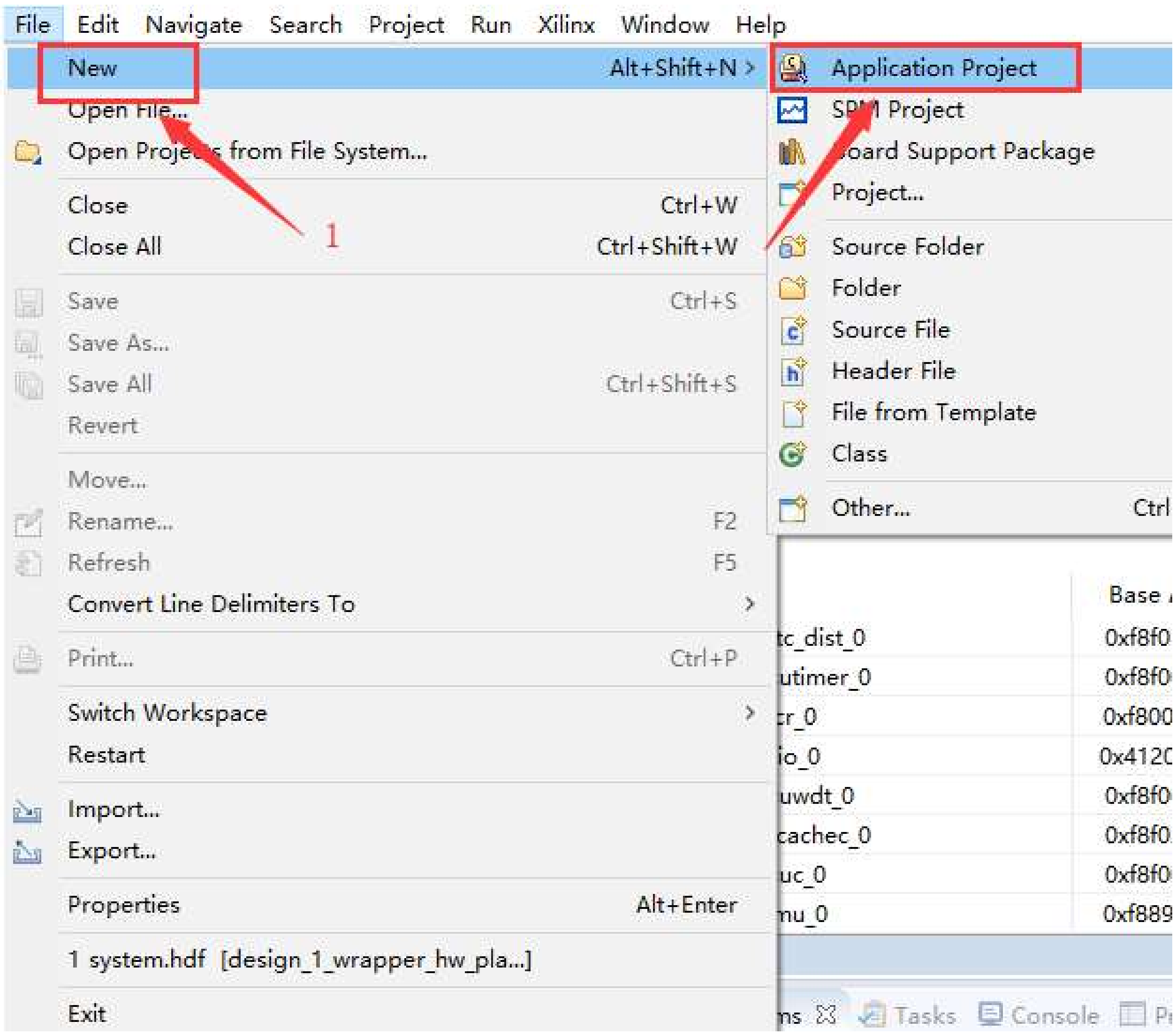
Target FPGA Device:	7z010
Part:	xc7z010clg400-1
Created With:	Vivado 2018.3
Created On:	Sun Aug 1 11:54:30 2021

Address Map for processor ps7_cortexa9_[0-1]

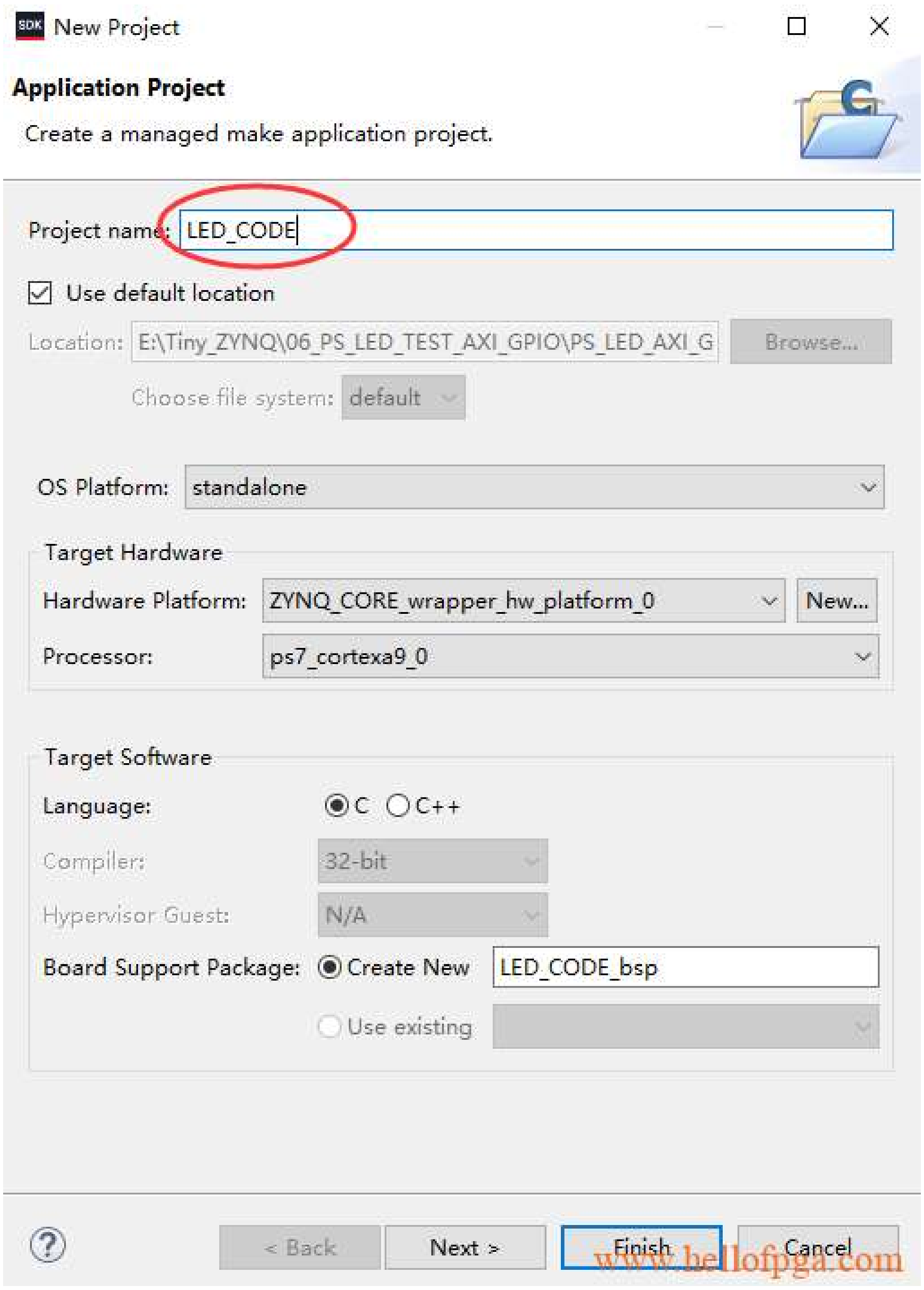
Cell	Base Addr	High Addr	Slave I/f
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	
ps7_scutimer_0	0xf8f00600	0xf8f0061f	
ps7_slcr_0	0xf8000000	0xf8000fff	
axi_gpio_0	0x41200000	0x4120ffff	S_AXI
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	
ps7_scuc_0	0xf8f00000	0xf8f000fc	
ps7_pmu_0	0xf8893000	0xf8893fff	

```
12:22:49 INFO  : Registering command handlers f
12:22:49 INFO  : Launching XSCT server: xsct.ba
12:22:49 INFO  : XSCT server has started succes
12:22:50 INFO  : Successfully done setting XSCT
12:22:50 INFO  : Successfully done setting SDK 
12:22:50 INFO  : Processing command line option
```

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


New Project

Application Project

Create a managed make application project.

Project name: LED_CODE

[✓] Use default location

Location: E:\Tiny_ZYNQ\06_PS_LED_TEST_AXI_GPIO\PS_LED_AXI_G

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: ZYNQ_CORE_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

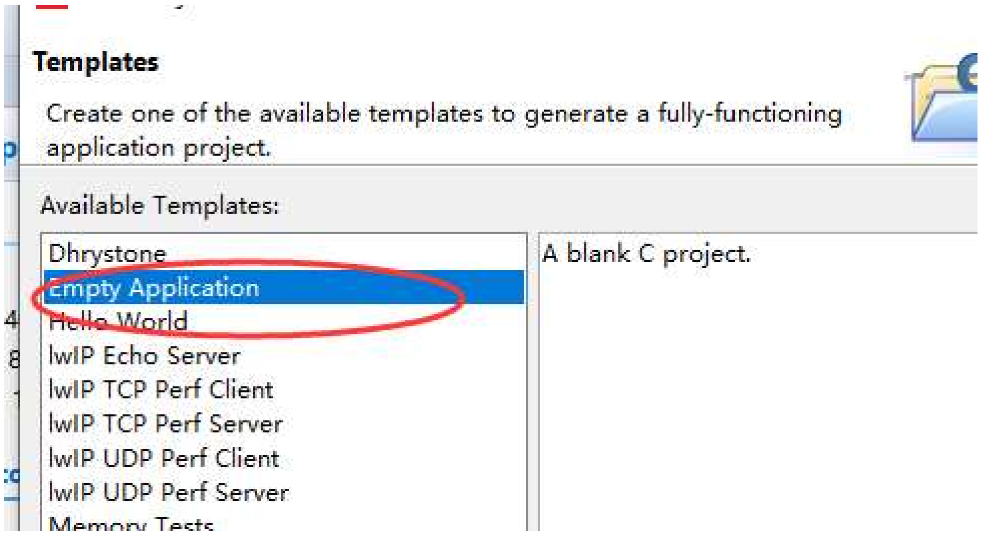
Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] LED_CODE_bsp

Use existing

5）选择空工程，点击完成FINISH


Templates

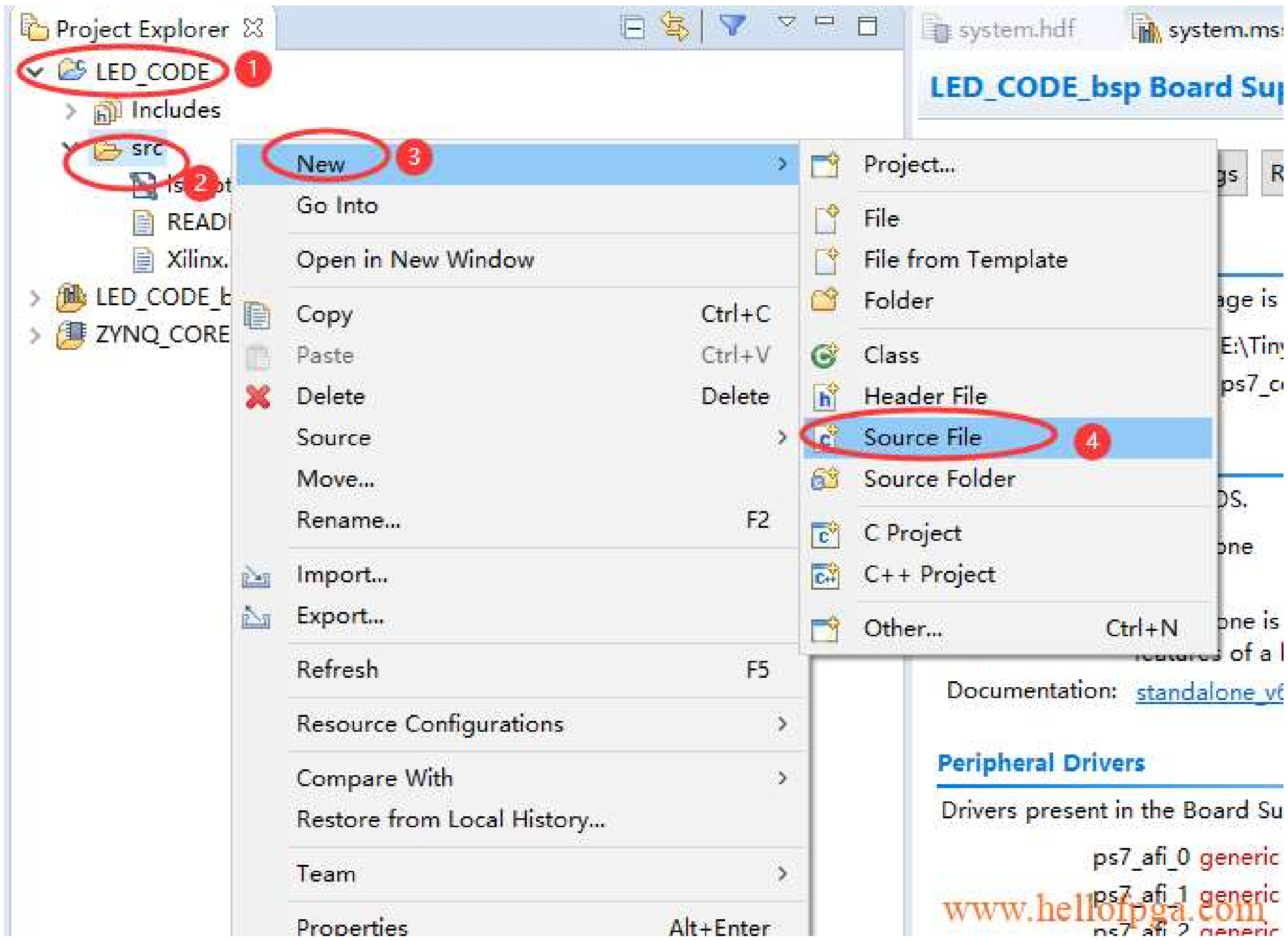
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

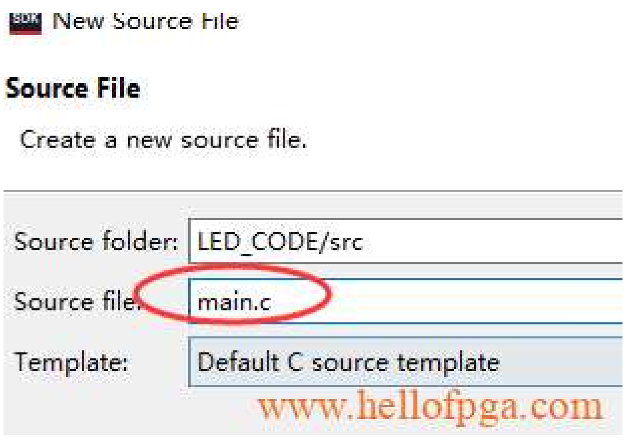
- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- Memory Tests

A blank C project.

6) 在工程中添加main.c文件 src—>New—>Source File 如下图所示



7）在弹出的窗口中填入main.c 并且保存


New Source File

Source File

Create a new source file.

Source folder: LED_CODE/src

Source file: main.c

Template: Default C source template

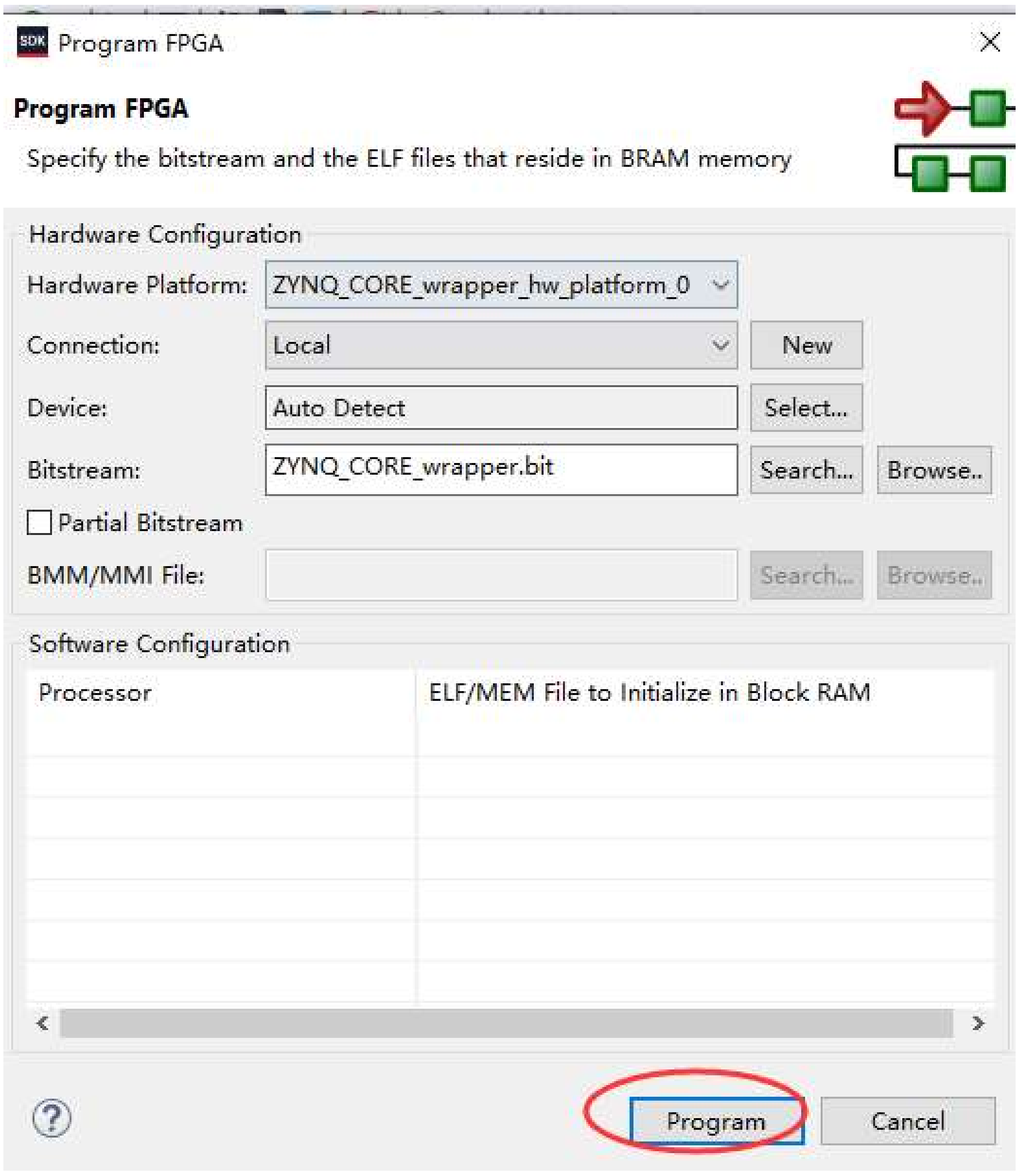
8）.打开刚才创建的main.c

然后 写入以下代码（代码是在 例程的基础上进行精简的） 有一个地方值得注意 EMIO的 IO口编号 是从54开始的，也就是我VIVADO 下创建的 EMIO端口，在PS端都是从54-55-56 依次排序的（小贴士 小于54的是MIO 也就是芯片PS的硬件IO口）

```
#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, 1);
                XGpioPs_WritePin(&Gpio, LED2, 0);

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

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

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

        };

        return 0;
}
```

3.下载到板子上进行验证

选中工程中的硬件平台，并点击右键→Program FPGA，在弹出的对话框中选择默认，点击“program”，完成FPGA PL部分的Program工作



Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

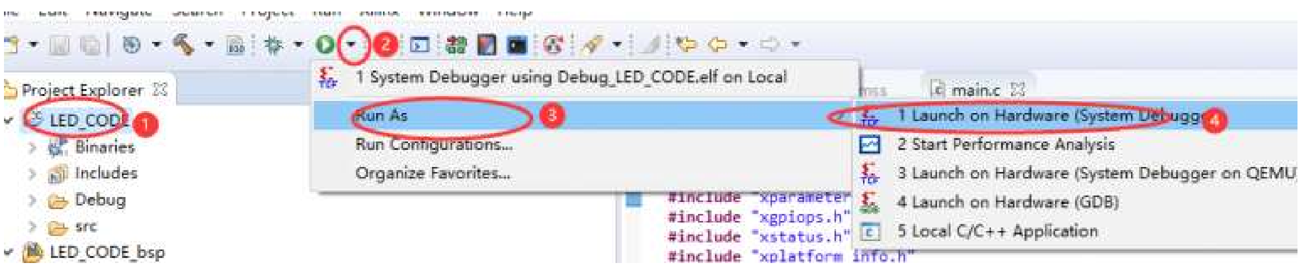
Hardware Configuration

Hardware Platform:	ZYNQ_CORE_wrapper_hw_platform_0
Connection:	Local
Device:	Auto Detect
Bitstream:	ZYNQ_CORE_wrapper.bit
☐ Partial Bitstream	
BMM/MMI File:	

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM

2）选中我们生成的GPIO工程 展开绿色箭头（RUN）右边的图标，选择Run As→1 Launch on Hardware（System Debugger）

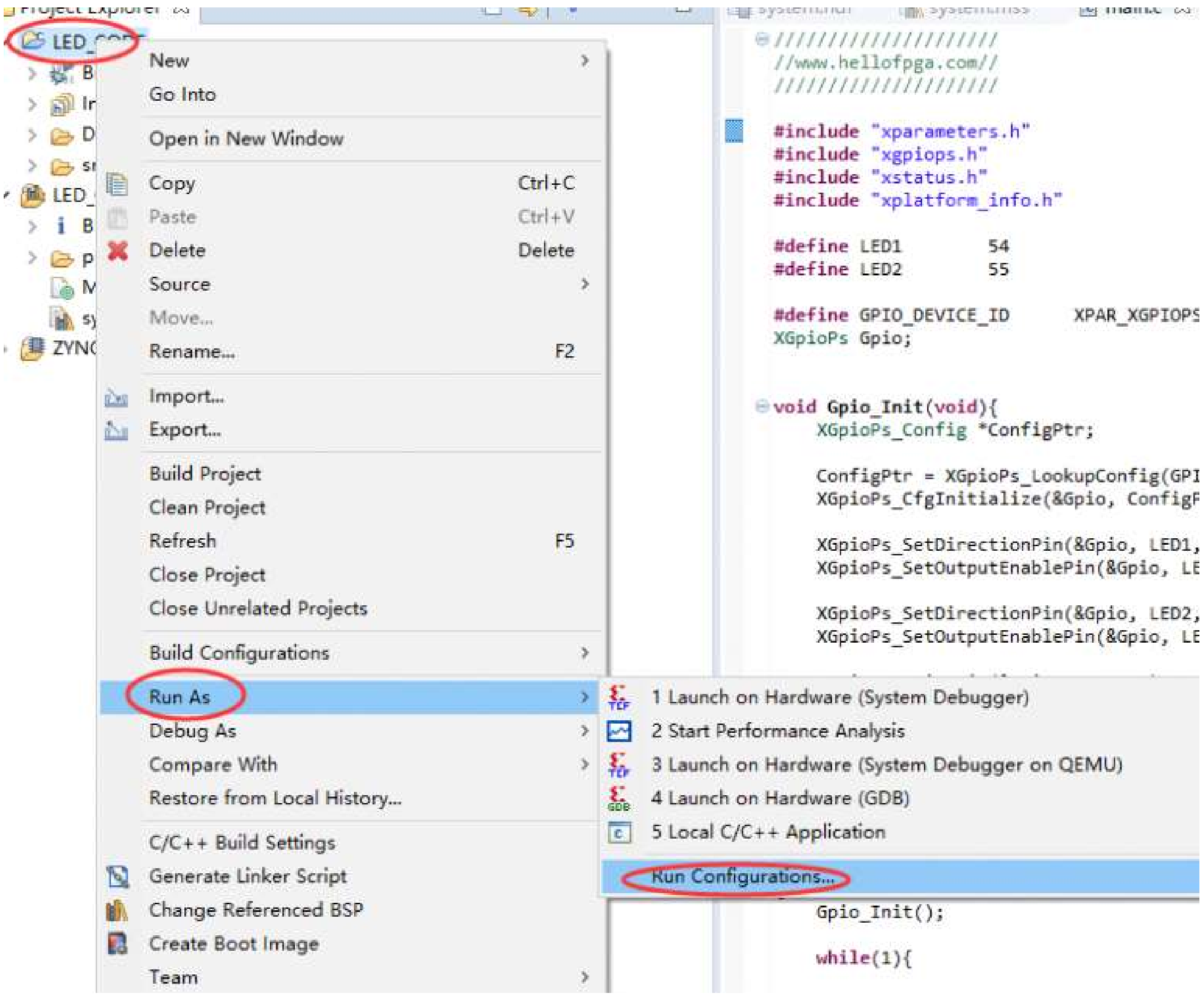


Project Explorer
LED_CODE
1
Binaries
Includes
Debug
src
LED_CODE_bsp
2
1 System Debugger using Debug_LED_CODE.elf on Local
Run As
3
Run Configurations...
Organize Favorites...
main.c
1 Launch on Hardware (System Debugger)
4
2 Start Performance Analysis
3 Launch on Hardware (System Debugger on QEMU
4 Launch on Hardware (GDB)
5 Local C/C++ Application

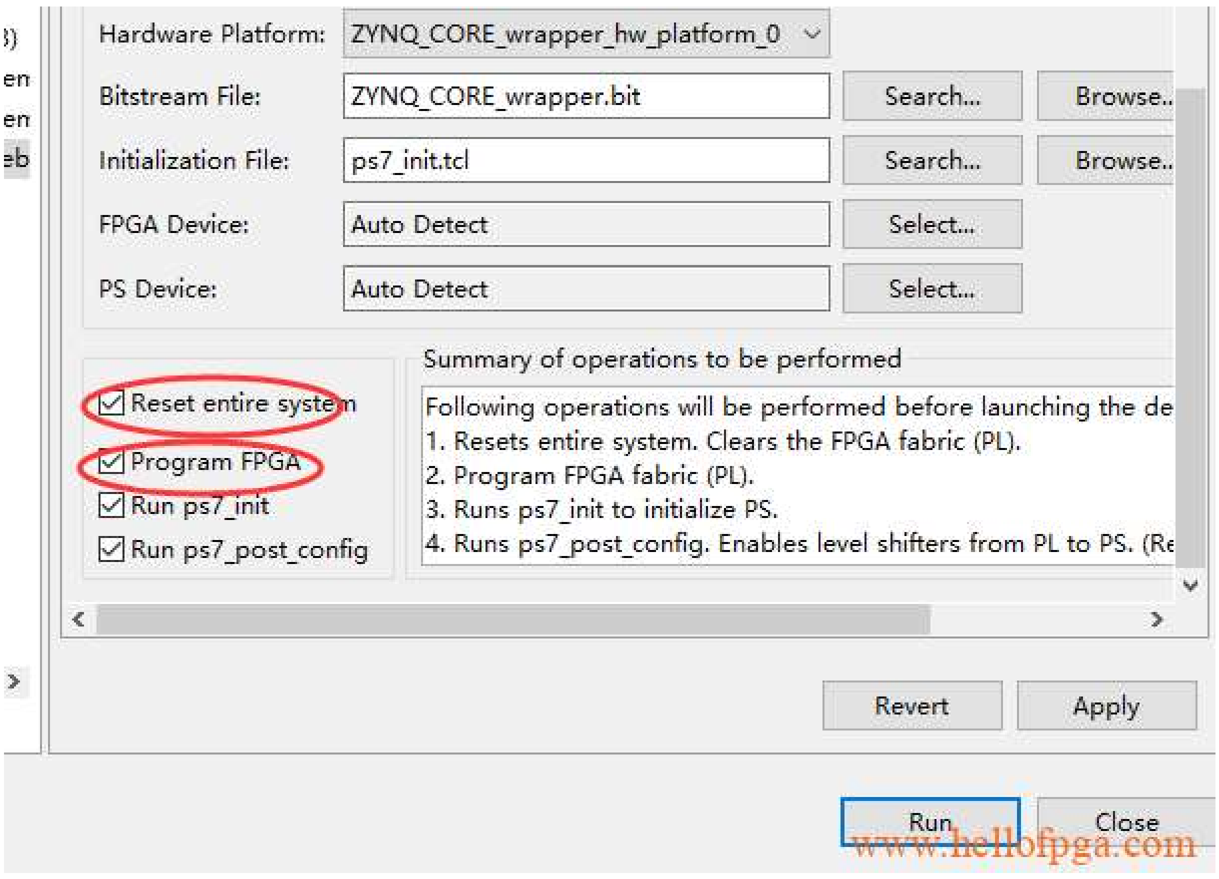
```
#include "xparameter
#include "xgpiops.h"
#include "xstatus.h"
#include "xplatform_info.h"
```

可以看到板子上的LED1灯在闪烁

备注： 如果 RUN 的时候弹出错误 可以按照下面的操作 进行设置 再进行DEBUG



```
////////////////////
//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;

    ConfigPtr = XGpioPs_LookupConfig(GPI
    XGpioPs_CfgInitialize(&Gpio, ConfigP

    XGpioPs_SetDirectionPin(&Gpio, LED1,
    XGpioPs_SetOutputEnablePin(&Gpio, LE

    XGpioPs_SetDirectionPin(&Gpio, LED2,
    XGpioPs_SetOutputEnablePin(&Gpio, LE

    Gpio_Init();

    while(1){
```



Hardware Platform: ZYNQ_CORE_wrapper_hw_platform_0

Bitstream File: ZYNQ_CORE_wrapper.bit

Initialization File: ps7_init.tcl

FPGA Device: Auto Detect

PS Device: Auto Detect

☑ Reset entire system
☑ Program FPGA
☑ Run ps7_init
☑ Run ps7_post_config

Summary of operations to be performed

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

之后点 APPLY 然后 再选择Run As→1 Launch on Hardware（System Debugger）看是否下载成功（如果仍然不行，请对板子进行断电后重试，一般发生这种问题的原因是因为debug的时候跟之前运行的程序产生冲突导致的）

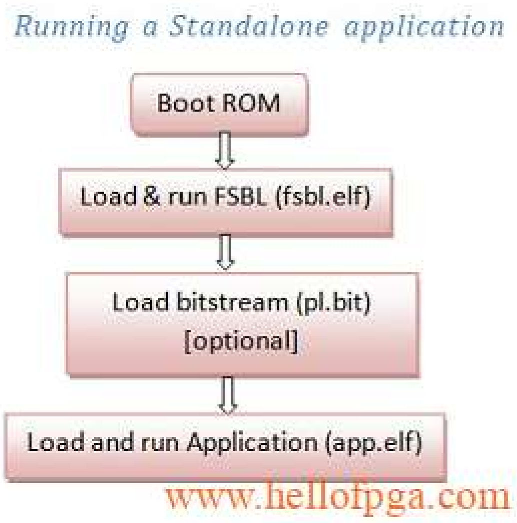
备注 这里的调试下载是没有固化的，断电后 程序会丢失。

以上内容除了BLOCK DESIGN 里设置QSPI的部分和 第七个工程不同外，其他过程几乎相同， 接下来才是 PS部分的固化设置

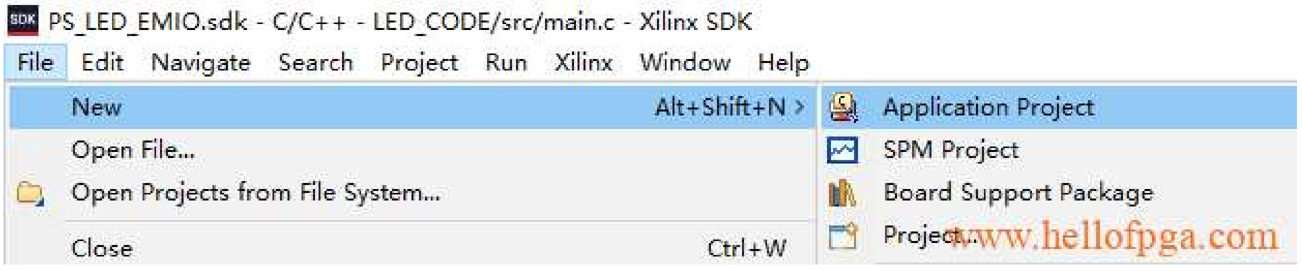
4. 进行固化的操作

程序固化分为两步1. 创建FSBL模块，2.创建image（boot.bin）文件

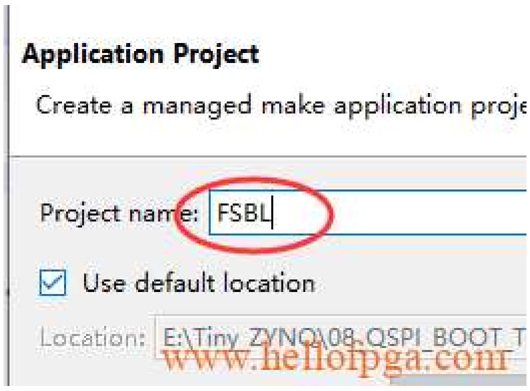
在zynq上运行程序的时候，加载过程中肯定需要用到一个文件，那就是fsbl，是zynq启动第一阶段的加载程序，经过了fsbl这一阶段，后面系统才能够运行裸奔程序或者是引导操作系统的u-boot


Running a Standalone application
Boot ROM
Load & run FSBL (fsbl.elf)
Load bitstream (pl.bit) [optional]
Load and run Application (app.elf)
www.hellofpga.com

1）创建FSBL:在SDK中新建工程如下



设置工程名fsbl，点next选择FSBL模板

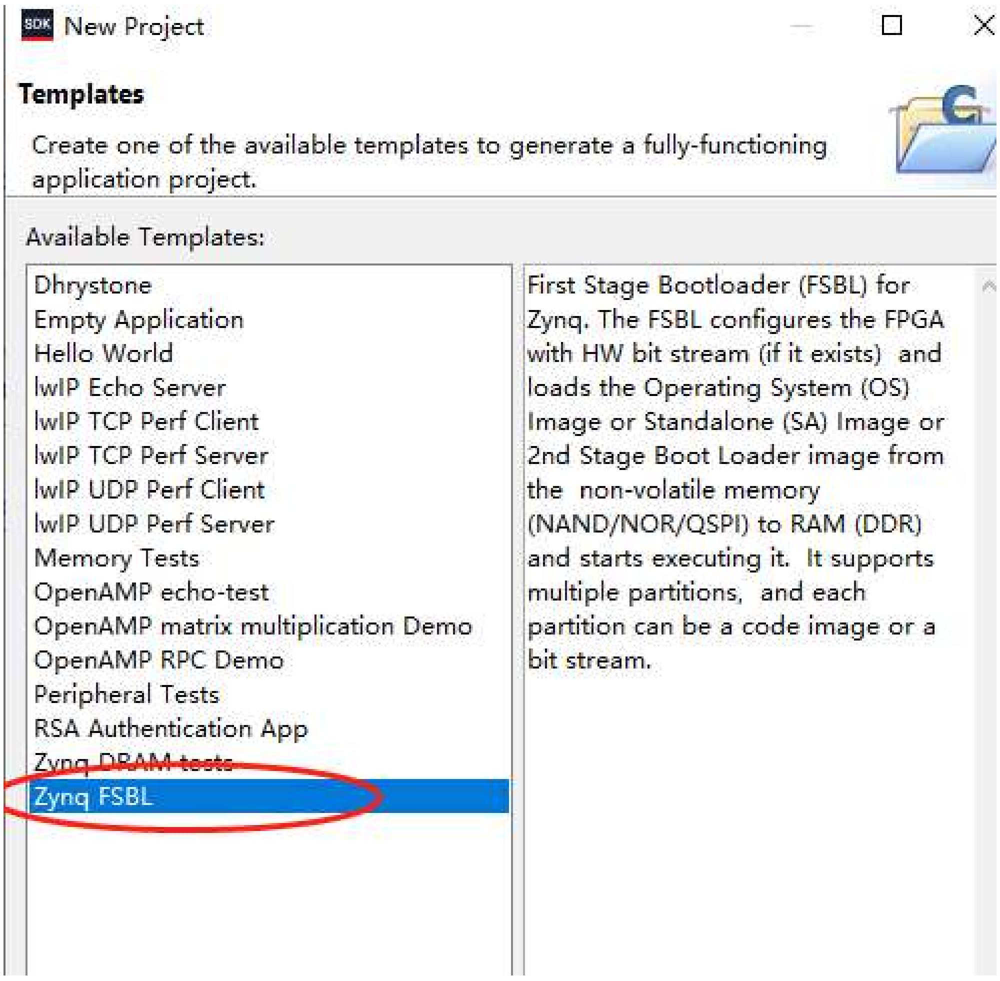

Application Project

Create a managed make application proje

Project name: FSBL

☑ Use default location

Location: E:\Tiny ZYNQ\08 QSPI BOOT T


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

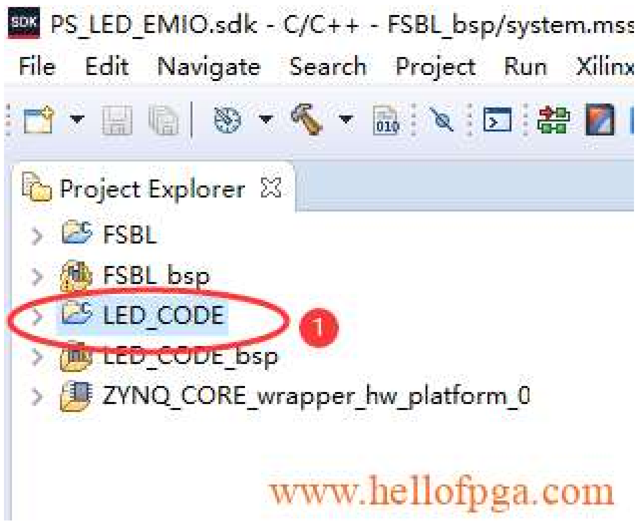
- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- lwIP TCP Perf Client
- lwIP TCP Perf Server
- lwIP UDP Perf Client
- lwIP UDP Perf Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

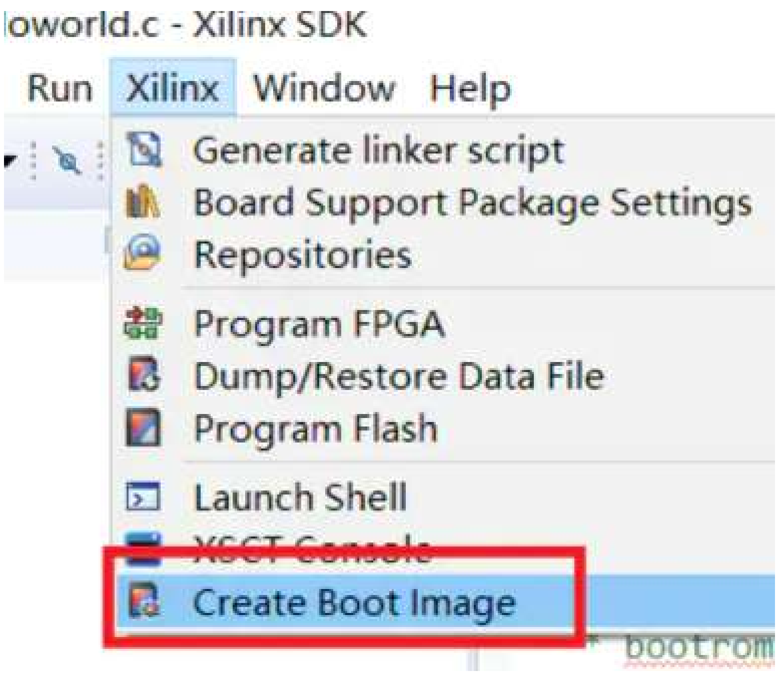
2) Make BOOT .bin for curing to NAND.

BOOT.bin consists of three files: the 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

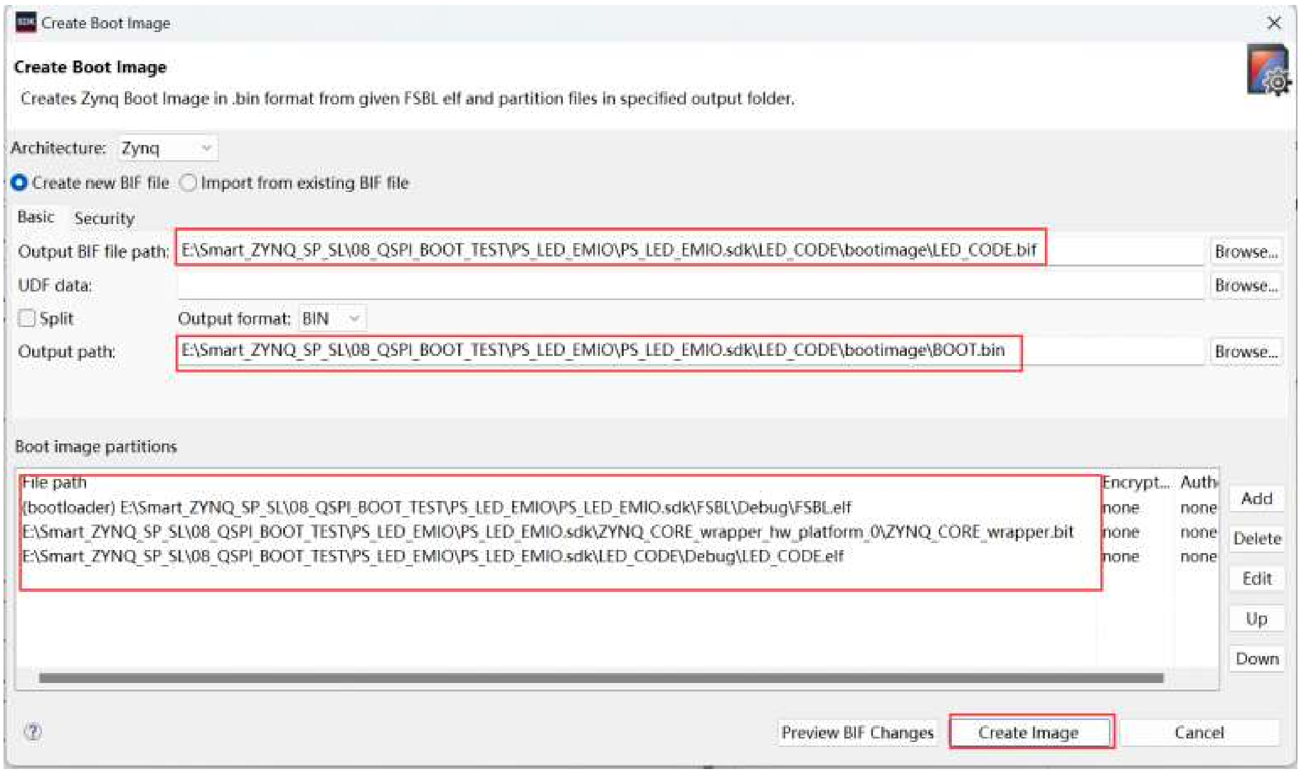
a) First select our APP project directory, here is the LED_CODE (not FSBL nor BSP and platform) **If it has been selected at the beginning, please select any directory above and below with the mouse, and then select back to the APP directory, otherwise the path may not be automatically loaded**


PS_LED_EMIO.sdk - C/C++ - FSBL_bsp/system.mss
File Edit Navigate Search Project Run Xilinx
Project Explorer
FSBL
FSBL_bsp
LED_CODE
1
LED_CODE_bsp
ZYNQ_CORE_wrapper_hw_platform_0
www.hellofpga.com

b) Point 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


Create Boot Image

Create Boot Image

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

Create new BIF file / Import from existing BIF file

Basic / Security

Output BIF file path: E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\LED_CODE\bootimage\LED_CODE.bif

UDF data:

Split / Output format: BIN

Output path: E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\LED_CODE\bootimage\BOOT.bin

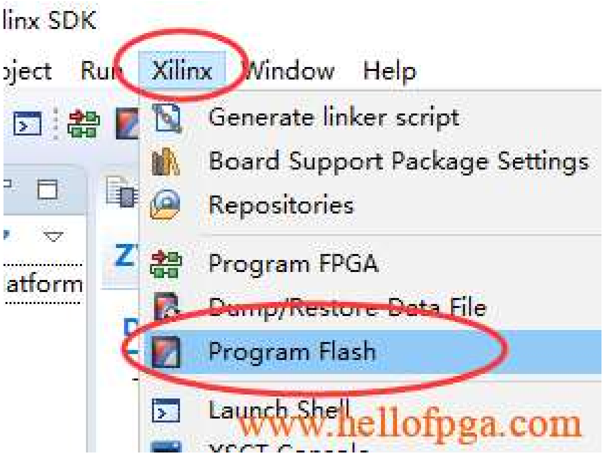
Boot image partitions

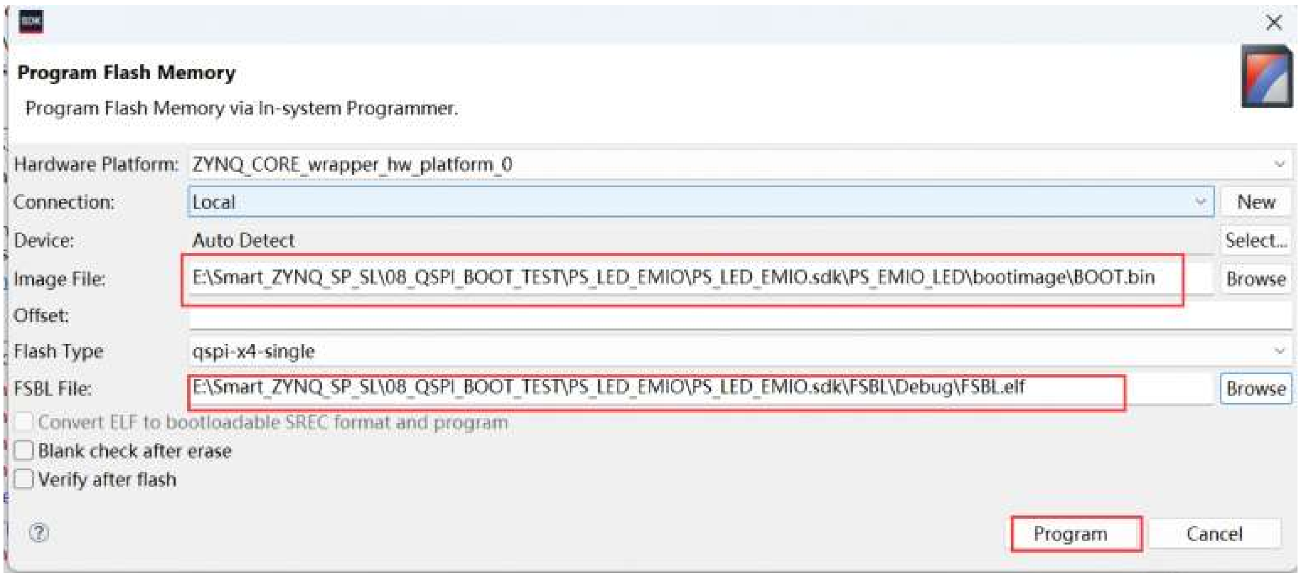
File path	Encrypt...	Auth
(bootloader) E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\FSBL\Debug\FSBL.elf	none	none
E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\ZYNQ_CORE_wrapper_hw_platform_0\ZYNQ_CORE_wrapper.bit	none	none
E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\LED_CODE\Debug\LED_CODE.elf	none	none

3） 烧录 镜像

生成完boot.bin后，下一步将boot.bin 下载到QSPI flash 中

点Program Flash





Program Flash Memory

Program Flash Memory via In-system Programmer.

Hardware Platform: ZYNQ_CORE_wrapper_hw_platform_0

Connection: Local

Device: Auto Detect

Image File: E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\PS_EMIO_LED\bootimage\BOOT.bin

Offset:

Flash Type: qspi-x4-single

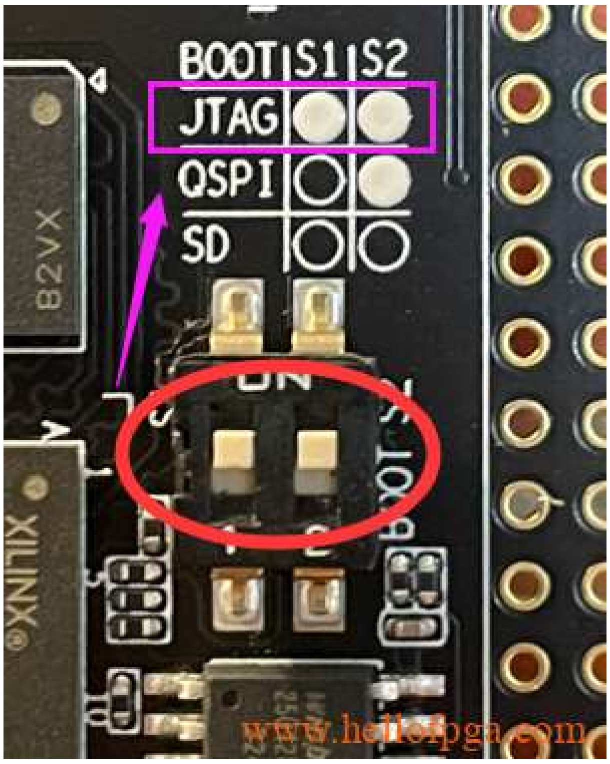
FSBL File: E:\Smart_ZYNQ_SP_SL\08_QSPI_BOOT_TEST\PS_LED_EMIO\PS_LED_EMIO.sdk\FSBL\Debug\FSBL.elf

☐ Convert ELF to bootloadable SREC format and program

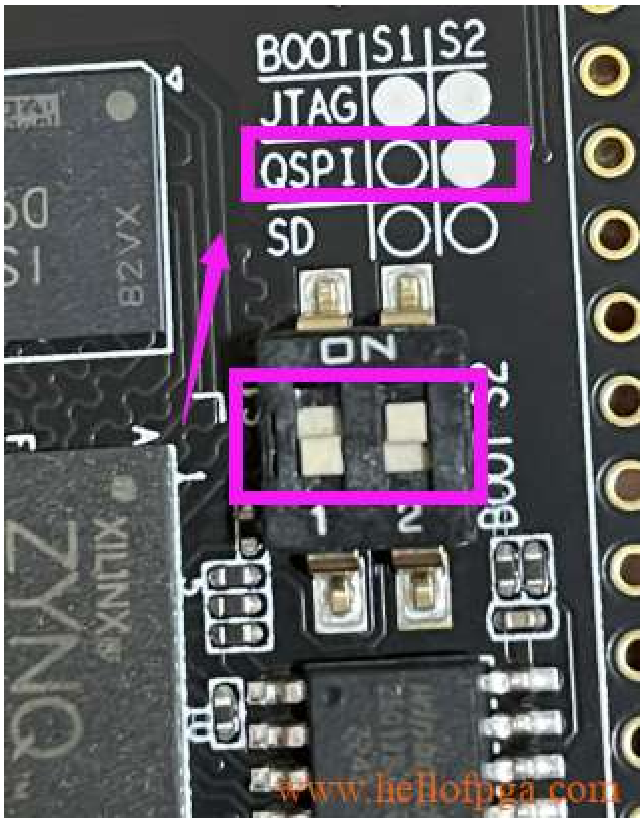
☐ Blank check after erase

☐ Verify after flash

Import the boot image file and fsbl.elf file as shown **above, set** the flash type to qspi-x4-sig, then adjust the board DIP switch to JTAG mode, and **power it back on (or press the POR RST key), and** then click PROGRAM to start downloading **(must be in JTAG mode, DIP switch to JTAG mode and restart the board or press the POR RST key).**

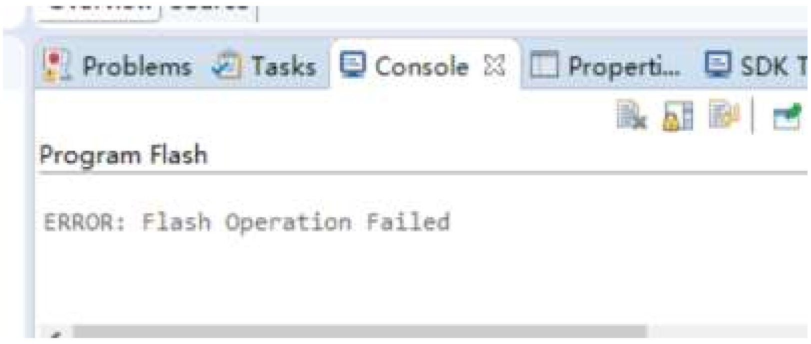


After clicking PROGRAM to download successfully, change the jumper position back to QSPI



Power on again **(or press the POR RST key),** if normal, the two LEDs will start blinking, indicating that the FPGA of the board has been initialized normally and the flash download is successful

If the following figure appears during the download process, it means that the system did not enter JTAG debugging mode normally when it was powered on, enter JTAG mode and then try to download



Program Flash

```
ERROR: Flash Operation Failed
```

Here is the complete project:

08_QSPI_BOOT_TEST_XC7Z020 **Download**