1 ZYNQ 程序固化

章节导读

一般在基于开发板进行程序设计时，我们都是通过 JTAG 接口进行下载验证。虽然大多数情况下非常方便，但是每当系统断电后，程序就需要重新进行下载。对于需要频繁使用该程序或者进行出厂调试来说非常不方便，因而本章将会从理论入手再到实践一步步给大家介绍 zynq 中程序固化的方法与流程。

1.1 ZYNQ 程序固化理论

1.1.1 什么是固化

程序固化用我们通俗的话来说，就是将程序烧写到芯片上去。在将一段特定的程序烧写到芯片的非易失性存储器（Quad SPI Flash，NAND Flash，NOR Flash 或 SD 卡）中后，用户如果不使用特定的烧写工具，将无法再对这一段程序进行任何修改，像是把程序固定住了一样，所以我们称之为程序固化。

Zynq 中有五种可能的启动源：NAND、NOR、SD 卡、Quad-SPI 和 JTAG。前四个启动源用于主启动方法，在主启动方法中，CPU 将外部启动映像从非易失性内存加载到 PS 中。JTAG 是从引导模式，仅支持非安全引导。外部主机作为主机，通过 JTAG 连接将引导映像加载到 OCM 中。

ZYNQ7000 SOC 芯片可以从 FLASH 启动，也可以从 SD 卡启动，芯片上电后，最先运行的是 ARM 端系统(PS)。然后再通过 ARM 系统软件部分加载 FPGA 的比特流文件.bit 至 FPGA(PL)，配置 FPGA PL 端的逻辑功能。ZYNQ 系统的启动流程如下：

- 阶段 0： BOOT ROM（ZYNQ 厂家固化代码）
- 阶段 1： FSBL (First Stage Bootloader)，通过 SDK 工具来生成。
- 阶段 2： 完成 Linux 系统启动过程（u-boot）

（1）阶段 0 (BOOTROM)

上电后，Zynq7000 SOC 会首先执行片内 BootROM 代码，BootROM 中的代码会对除 SD 卡以外的所有引导设备进行头文件搜索，该头文件里定义了一些启动信息，用于配置 BootROM 的运行。不同的启动设备搜索的地址空间不同，具

体如下：

- NAND：前 128M 地址空间内搜索
- NOR：前 32M 地址空间内搜索
- QSPI（signal/dual SS with 4-bit I/O）：前 16M 地址空间内搜索
- QSPI（dual SS with 8-bit Parallel I/O）：前 32M 地址空间内搜索
- SD 卡：不进行头文件搜索

BootROM 代码读取 Boot mode 寄存器来判断是哪一种启动方式(SD card/QSPI Flash/JTAG)。确定启动方式后，BootROM 从相应的启动设备(SD Card/QSPI Flash)加载 First Stage Bootloader (FSBL)或用户代码到 On Chip Memory(OCM)的 RAM 上，并且将执行权交付给 FSBL，这个被拷贝到片上 RAM 执行的程序就来自于我们要制作的文件——BOOT.bin。

（2）阶段 1 FSBL (First Stage Bootloader)

第一阶段引导程序(First Stage Boot Loader，FSBL)启动，BOOT.bin 开始执行：首先继续配置 PS，PS 初始化好后，再配置 PL，最后还可以加载阶段 2 的代码。BOOT.bin 其实就是由 FSBL.elf + 原工程.bit + 原工程.elf 构成(其中原工程.bit 和原工程.elf 在程序编译完成后便已产生)。这一阶段所需的 FSBL 代码和可执行文件需要用户使用 SDK 工具一步步生成，在这一阶段，FSBL 主要做了如下工作：

- 初始化 CPU，初始化串口；
- Processor System (PS) 一些控制器的初始化，如 MIO, PLL, CLK，DDR；
- 使用比特流文件对 PL 进行配置；
- 将应用程序加载到 DDR 或者加载 Second-Stage Boot Loader（SSBL）；

（3）阶段 2 SSBL

这一阶段是可选的，通常 SSBL 就是裸机程序，对于 linux 启动来说就是 u-boot 之类的 bootloader。这一阶段也完全由用户控制，本章暂不详述。

1.1.2 zynq 启动流程

Zynq 系统启动过程如图 1-1 所示：

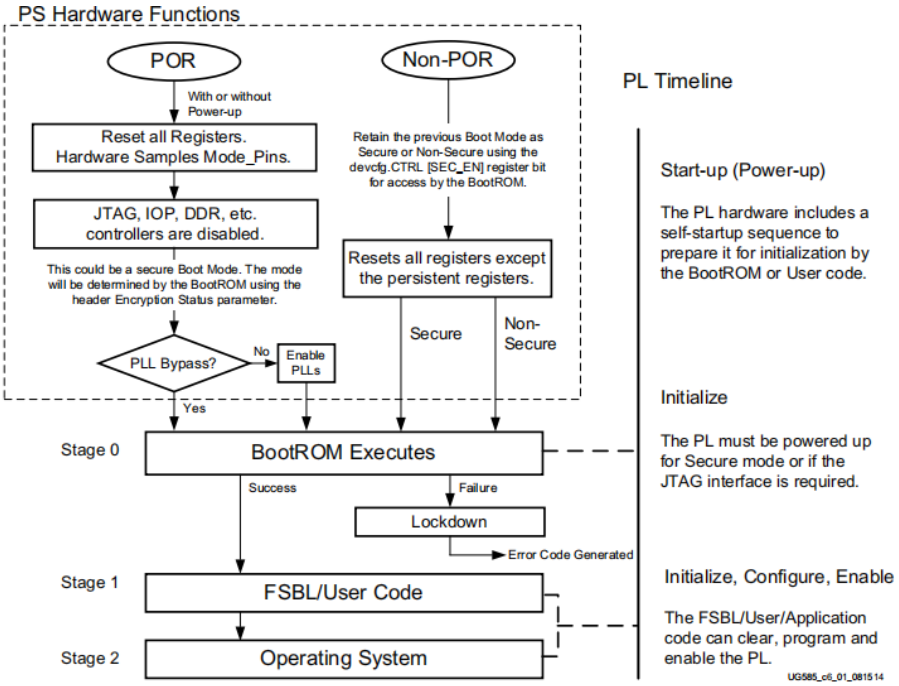

图 1-1 zynq 系统启动过程

Zynq 系统在进行步骤 0 之前，会先进行一次复位，以清除相关寄存器的配置。复位又分为 POR（Power On Reset）复位和 Non-POR 复位。POR 复位会复位整个芯片，包括调试的配置信息，PL 需要重新编程，该复位信号可以通过 PS_POR_B 管脚产生。而如果是在 Non-POR 复位的情况下，只会复位除调试的配置信息外的整个芯片，PL 同样需要重新编程。该信号可以由 PS_SRST_B 管脚或系统内部复位产生。复位完成后，zynq 开始启动。zynq 在进行 FLASH 启动（NAND、NOR、Quad-SPI）和 SD 卡启动时，启动流程如下:

Flash 启动方式：

（1）初始化对应 Flash 控制器；

（2）从 Flash 拷贝 system.bit 到 FPGA (如果 Flash 中存有 system.bit)；

（3）从 Flash 拷贝应用程序的代码到 DDR3；

（4）跳转到应用程序执行；

SD Card 启动方式：

（1）初始化 SD 控制器；

（2）从 SD Card 拷贝 system.bit 到 FPGA (如果 BOOT.BIN 中存有 system.bit)；

（3）从 SD Card 拷贝应用程序的代码到 DDR3；

（4）跳转到应用程序执行；

zynq 启动首先运行的是 BootROM 代码，然后是 FSBL 用户代码和系统代码。值得注意的是，zynq 的启动方式有五种，但 zynq 却能识别出具体是使用的哪种启动方式，依靠的就是识别其启动模式引脚（Boot Mode Pin）。zynq 总共有 7 个模式引脚 MIO[8:2]，其中前四个引脚 MIO[5:2]控制 zynq 的启动模式，MIO[6]控制是否启用锁相环 PLL，MIO[8:7]则分别用来控制两个 MIO bank 电压信号电平。其具体关系如图 1-2 所示：

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					

图 1-2 模式引脚与启动模式关系

BX71，支持 JTAG、QSPI、SDcard 3 种启动方式。启动方式的选择由 MIO4 和 MIO5 两个引脚进行控制，此外在 BX71 开发板上 MIO4 与 PS_QSPI_DQ2、MIO5 与 PS_QSPI_DQ3 引脚复用。

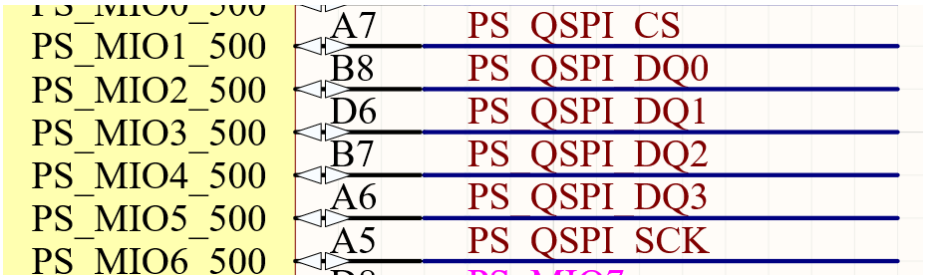

图 1-3 BX71 启动模式引脚

BX71 开发板启动方式与 MIO4、MIO5 的对应关系为：

表 1-1 BX71 启动方式与引脚对应关系

JTAG	MIO4 低电平、MIO5 低电平

QSPI	MIO4 低电平、MIO5 高电平
SD card	MIO4 高电平、MIO5 高电平
NAND	MIO4 高电平、MIO5 低电平（未使用）

在芯路恒 BX71 开发板上，我们对 MIO4 和 MIO5 两个引脚的电平控制方式如下：

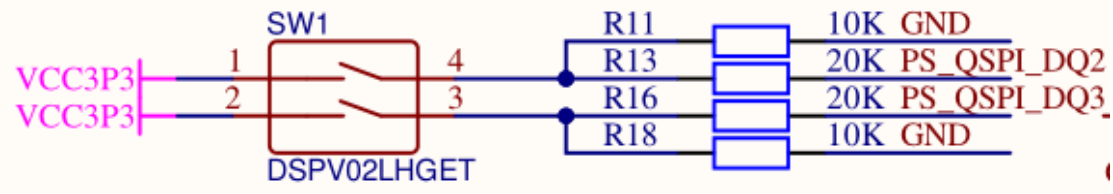

图 1-4 控制引脚电平方案

BX71 通过拨码开关对引脚电平进行控制，具体的使用方法如下：

表 1-2 BX71 开发板启动模式

模式	拨码开关	
	1	2
JTAG	↑	↑
QSPI	↑	↓
SD card	↓	↓

了解了 zynq-7000 系列器件的启动原理以及 BX71 开发板启动模式切换后，接下来将以一个具体工程为实例，手把手教大家如何一步步实现程序的固化。

1.2 程序固化设计实例

接下来我们将以前面章节创建的“key_ctrl_led”工程为范例，进行 SD 卡和 QSPI 启动的 zynq 程序固化实验。二者的固化流程具体步骤如下：

1.2.1 生成固化文件

1.打开“key_ctrl_led”工程，点击工程→save as，将工程另存为一个新工程，如图 1-5 所示。

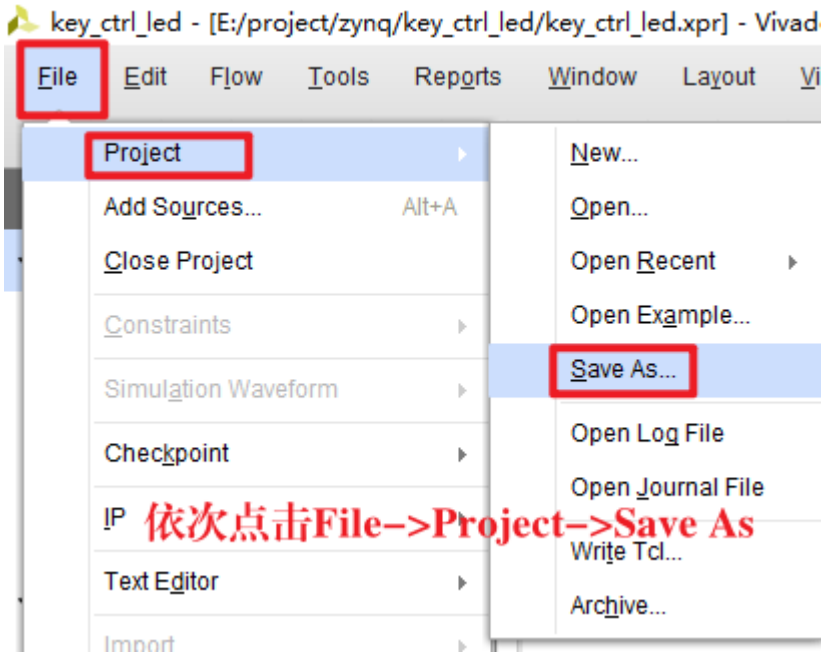

图 1-5 工程另存为

另存为新工程后，点击 File->Launch SDK，运行 SDK。打开 SDK 后可以看到此时有两个硬件资源描述文件，如图 1-6所示。通常当文件路径发生变化或对硬件改动较大时，软件会重新生成一个硬件资源描述文件。一般情况下后缀数字最大的即为最新文件，用户在对硬件修改后，软件会将信息更新到最新的硬件资源描述文件中，而 SDK 工作时基于后缀为 0 的文件。所以，我们需要更改硬件硬件资源描述文件路径，具体方法如下：

1. 右键单击后缀为 1 的硬件资源描述文件，点击更改硬件规范
2. 复制后缀为 1 的硬件规范路径，随后点击取消
3. 右键单击后缀为 0 的硬件资源描述文件，点击更改硬件规范
4. 将后缀为 0 的硬件规范文件路径修改为复制的后缀为 1 的硬件规范文件路径
5. 删除除后缀为 0 以外的所有硬件资源描述文件，删除时勾选删除磁盘文件

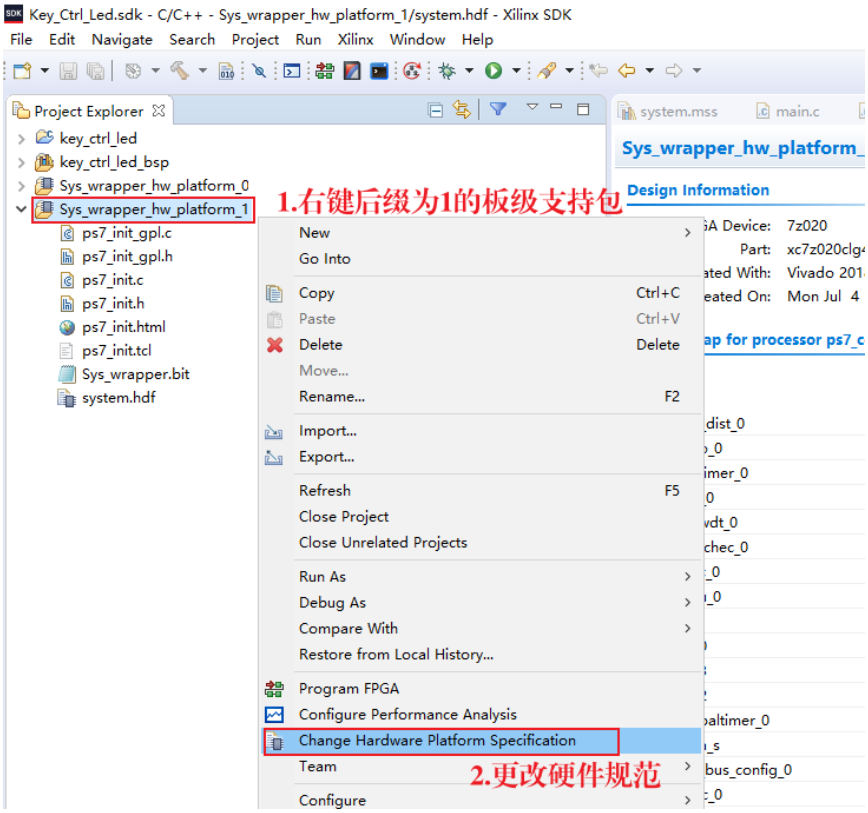

图 1-6 步骤 1

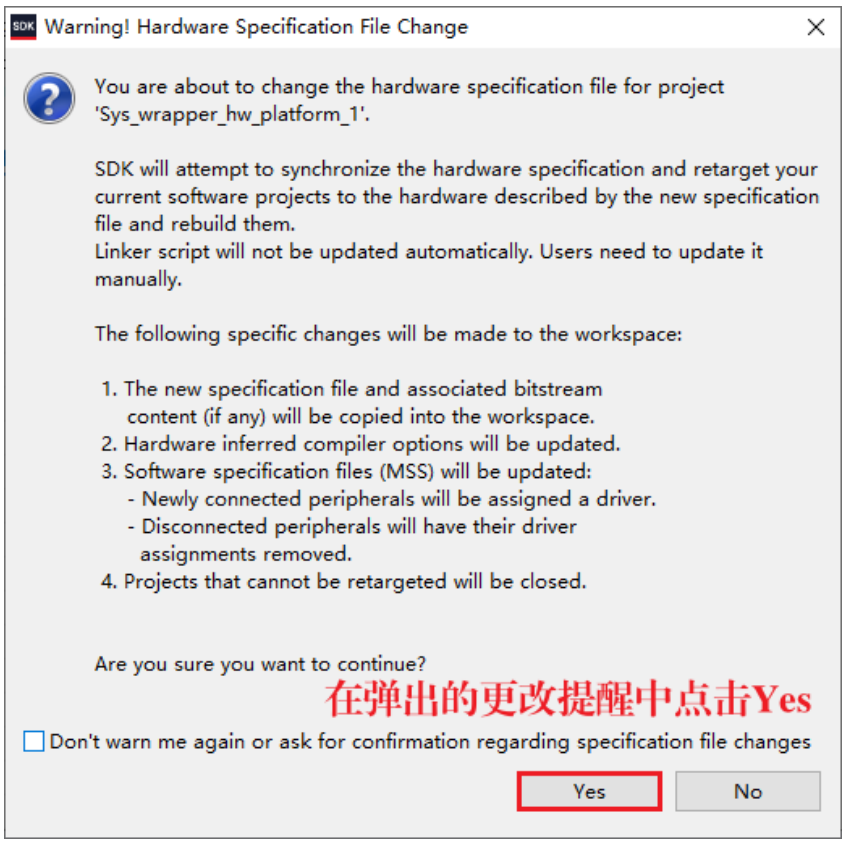

图 1-7 步骤 2

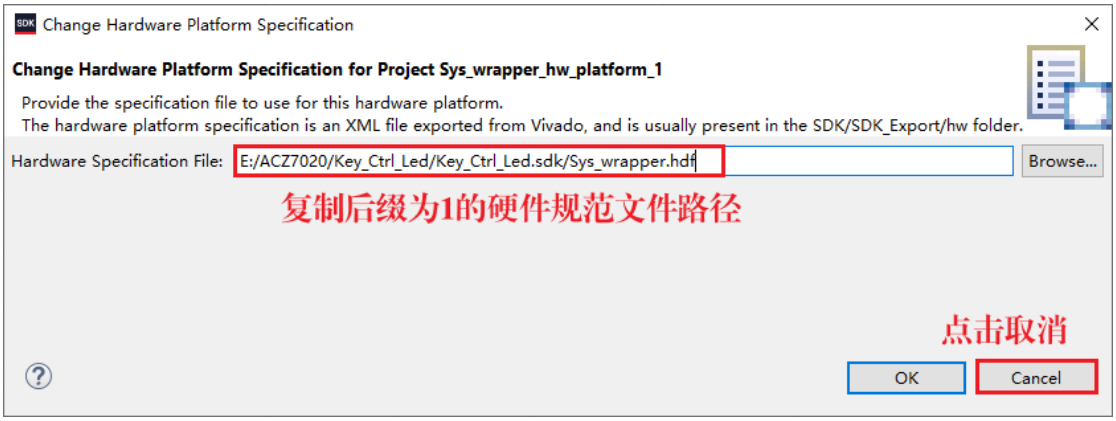

图 1-8 步骤 3

图 1-9 步骤 4

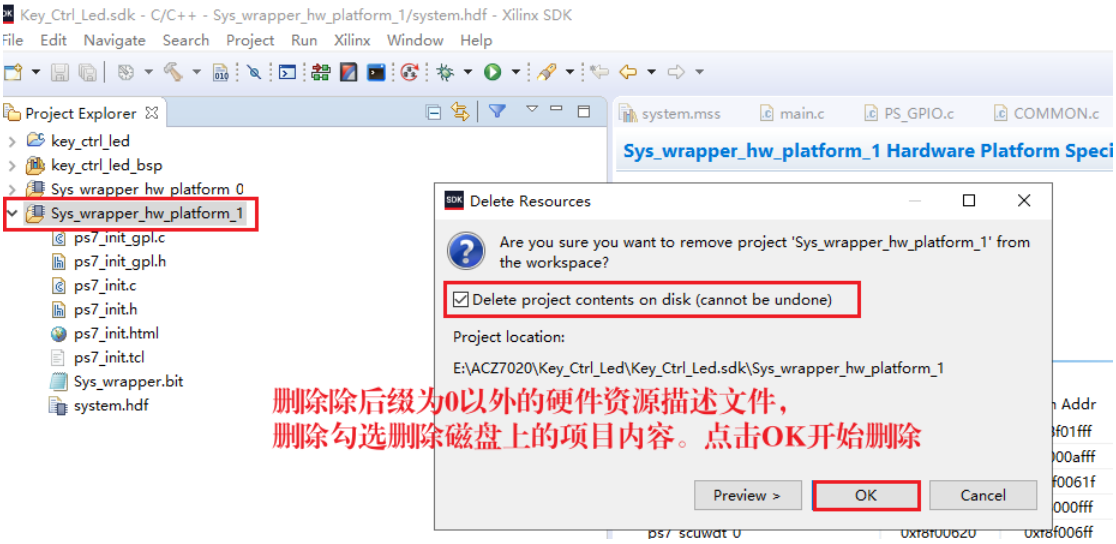

图 1-10 步骤 5

删除时，软件会弹出如图 1-11 所示提示，这里我们直接点击“Continue”即可。

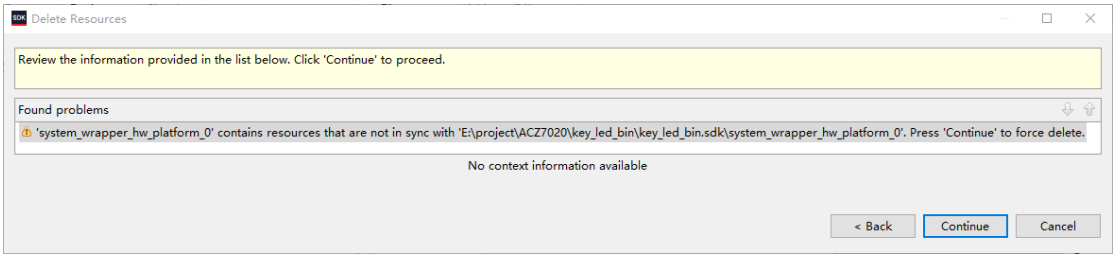

图 1-11 删除提示

删除后的工程可能会出现图 1-12 所示的报错：

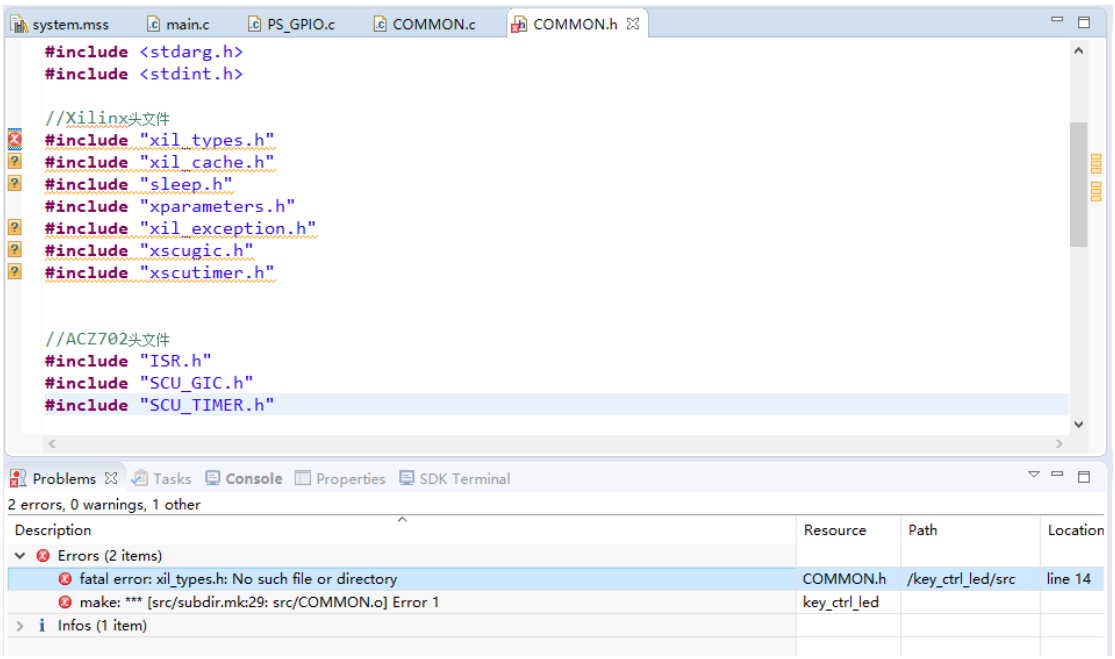

图 1-12 工程报错

这是因为此时工程的板级支持包（BSP）是基于被删除的硬件资源描述文件生成，我们删除了描述文件后没有对其更新。右键单击 key_ctrl_led_bsp 文件夹，在弹出的选项中选择重新生成 BSP，如图 1-13 所示：

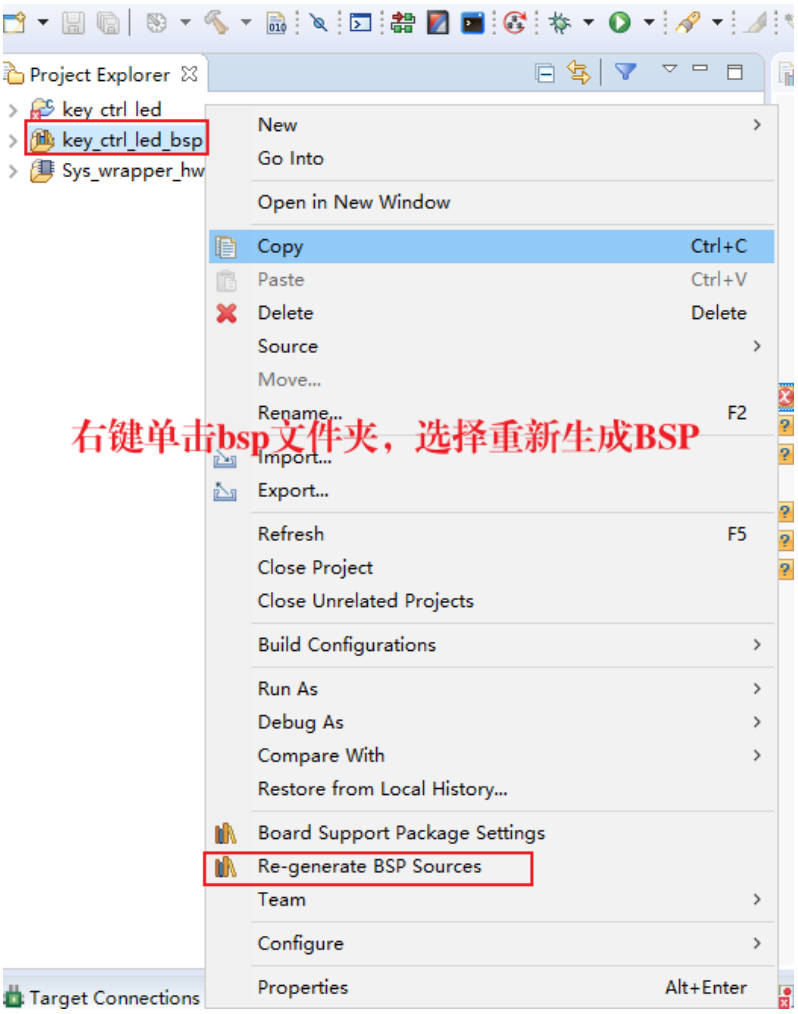

图 1-13 重新生成 BSP

2.回到 Vivado 中，点击“Open Block Design”准备对设计进行修改。

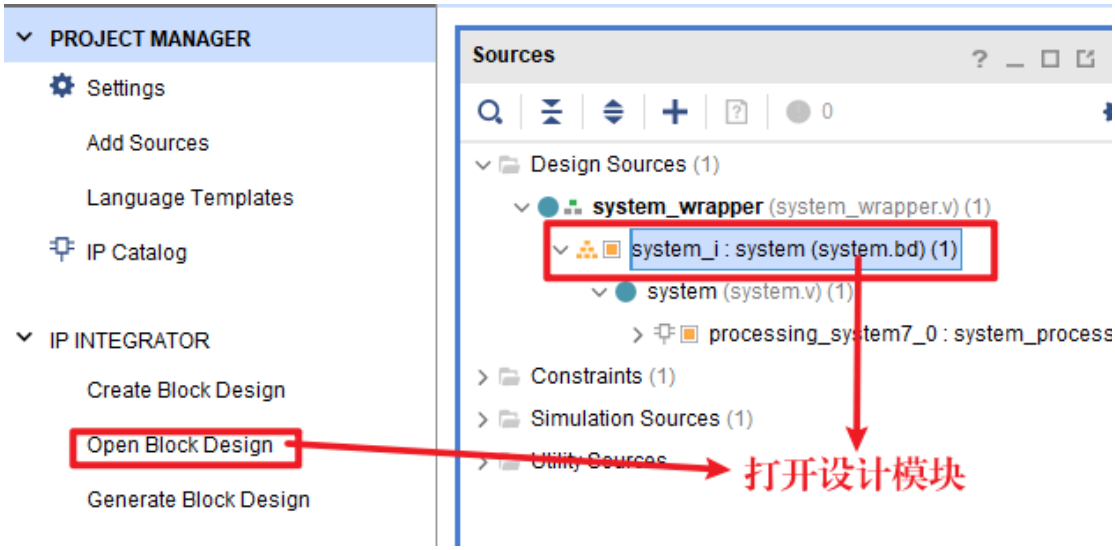

图 1-14 打开模块设计

对 zynq IP 核进行配置，由于本次设计我们的目的是要将程序固化到 SD 卡中通过 SD 卡启动以及将程序固化到 QSPI Flash 中，通过 QSPI 启动。所以我们需要使能 SD 接口和 QSPI 接口。

BX71 开发板的 QSPI 接口和 SD 接口硬件引脚分配如图 1-15 所示：

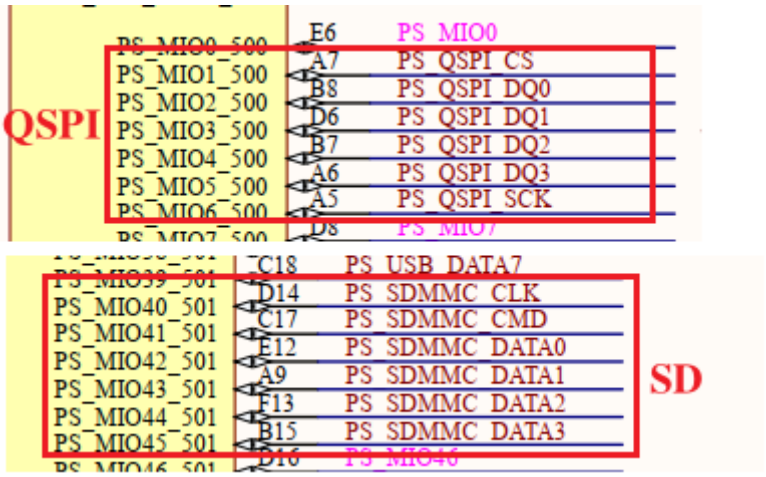

图 1-15 硬件引脚分配

可以看到 QSPI 接口对应 PS 侧的 MIO1 到 MIO6 引脚，SD 接口对应的引脚为 PS 侧的 MIO40 到 MIO45。Zynq 上有两个 SD 控制器，MIO40 到 MIO45 属于 SD0，对 Zynq IP 核的配置如图 1-16 和图 1-17 所示：

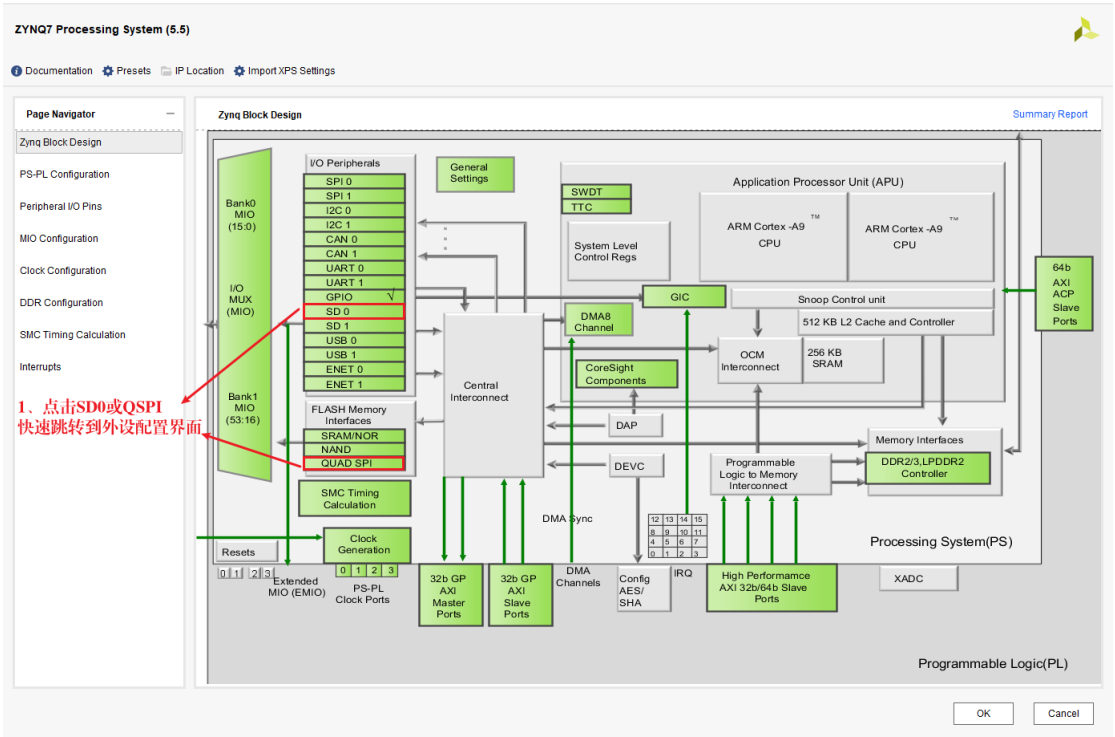

图 1-16 配置 Zynq IP 核

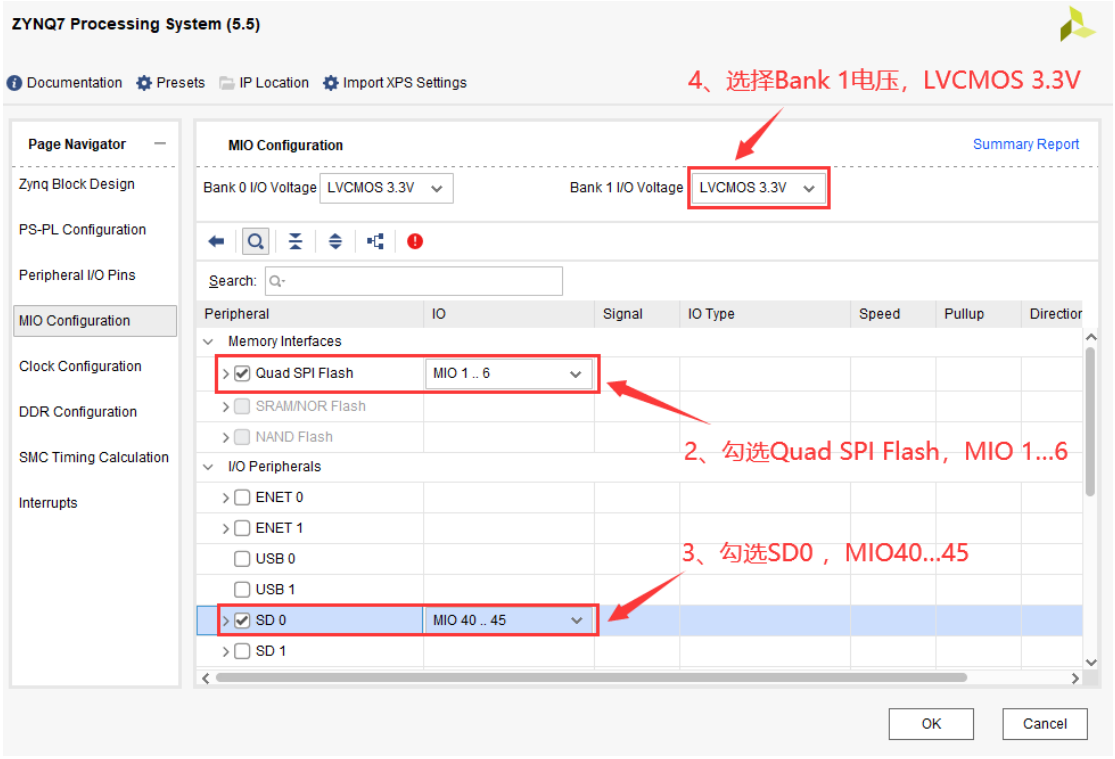

图 1-17 配置 zynq IP 核

配置完成后，Ctrl+S 保存设计，点击“Generate Bitstream”重新生成比特流，由于我们没有生成输出，软件询问我们是否生成输出，并在生成后自动开始生成比特流，这里我们点击 Yes。

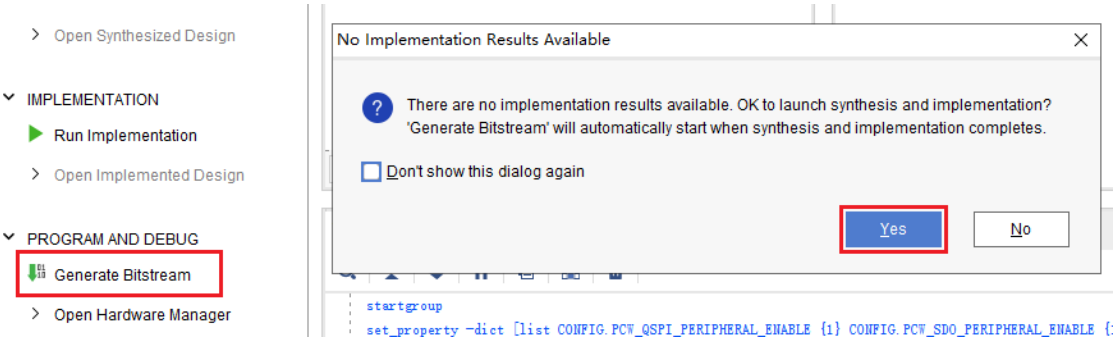

图 1-18 重新生成比特流

接下来软件会弹出如图 1-19 所示窗口，这里我们将 jobs 的数目设置为最大值，该值会影响软件运行的速度，随后点击 OK 即可。

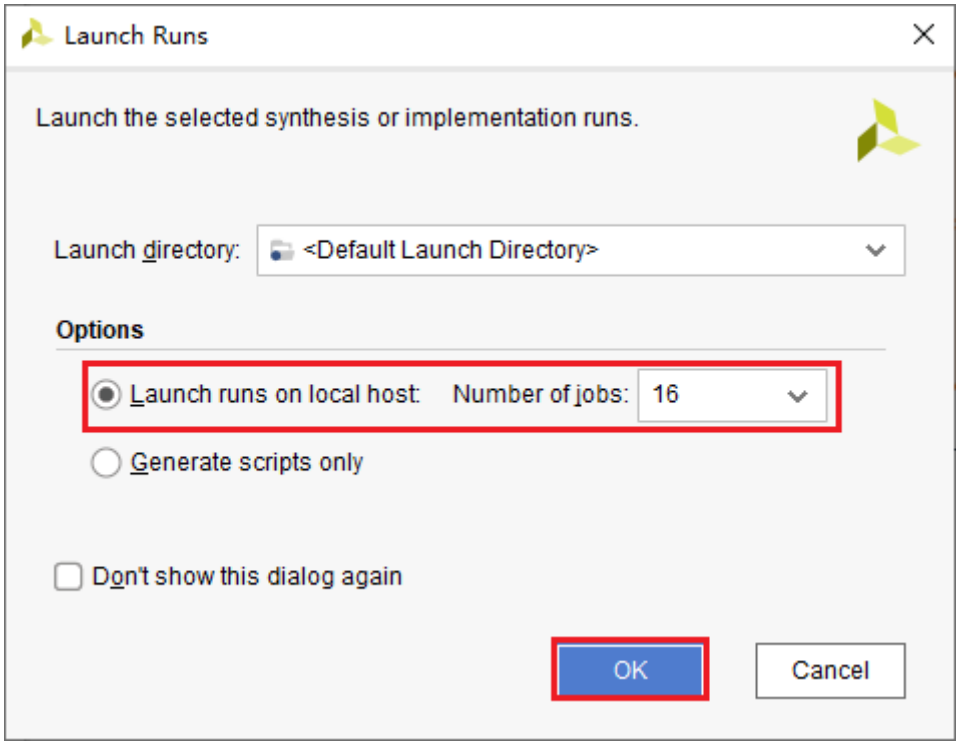

图 1-19 启动运行

当比特流生成完成后会弹出如图 1-20 所示

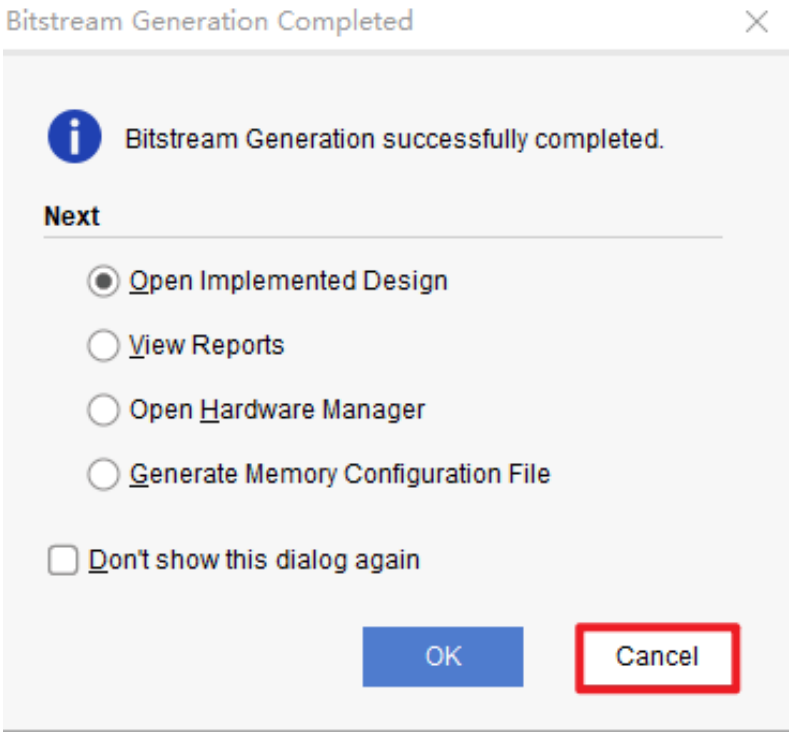

图 1-20 比特流生成完成

在重新生成比特流后我们将比特流和硬件资源描述文件一起导出到 SDK 中，如图 1-21 所示：

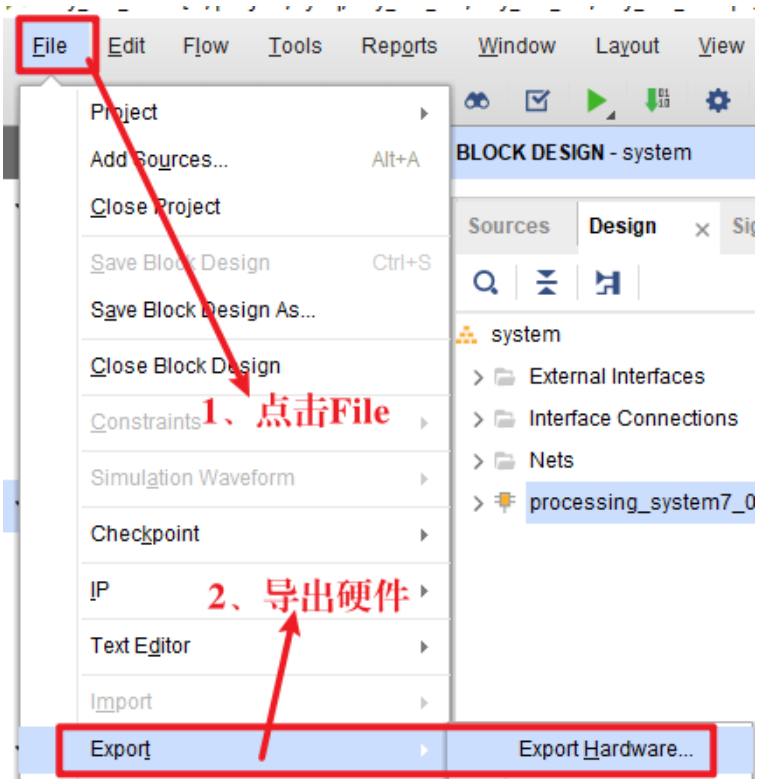

图 1-21 导出硬件资源描述文件

回到 SDK，此时软件提示我们硬件资源描述文件发生了更改，是否更新，点击 Yes 进行更新，如图 1-22 所示：

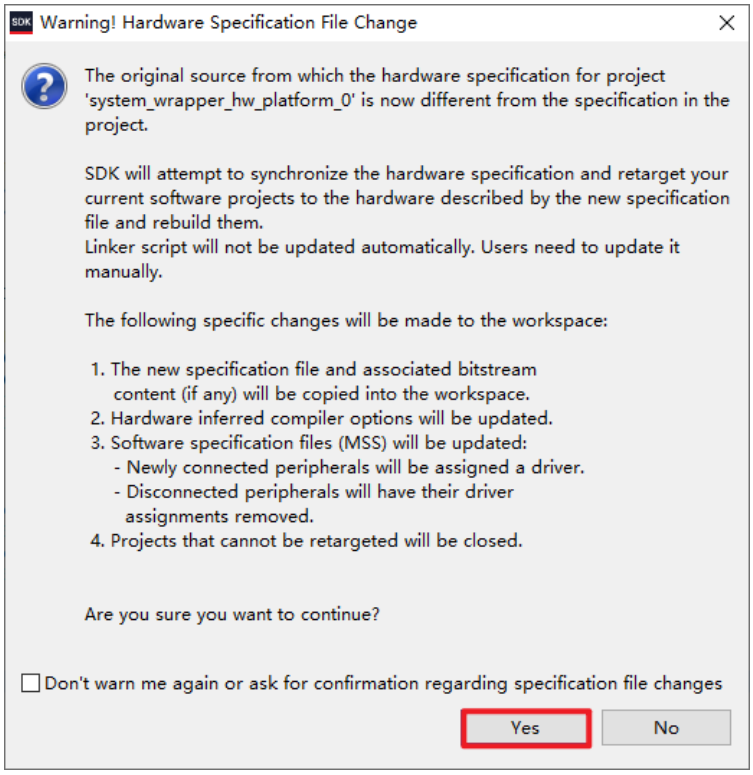

图 1-22 硬件资源描述文件更改提醒

右键单击“key_ctrl_led_bsp”，选择“Board Support Package Settings”对板级支持包进行设置。

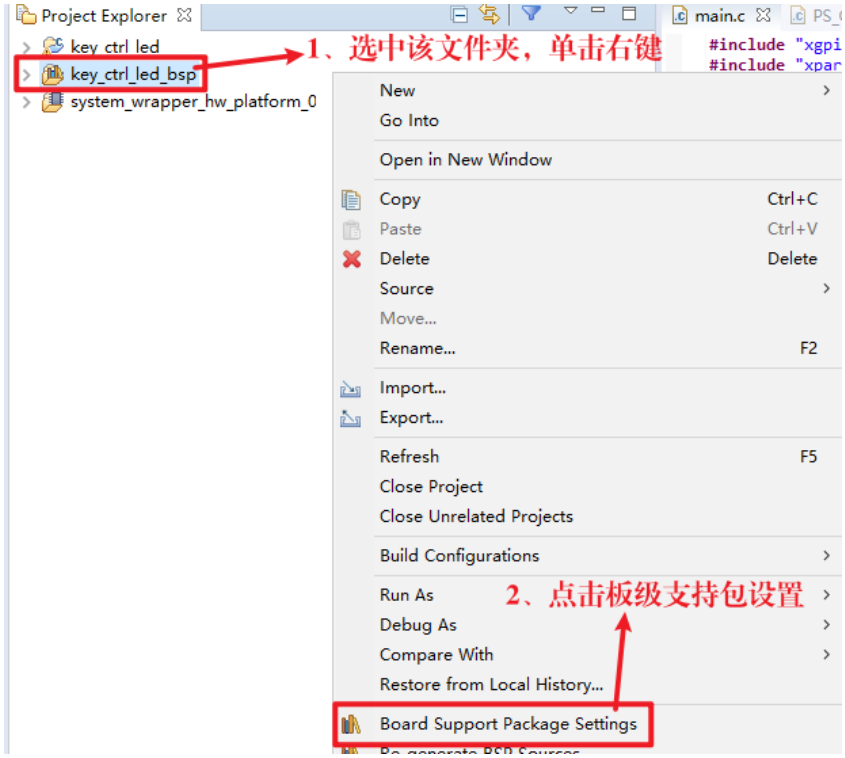

图 1-23 板级支持包设置

在弹出的窗口中勾选 xilffs，启用 FAT 系统，用于 SD 卡固化。选中后点击 ok，完成板级支持包设置。

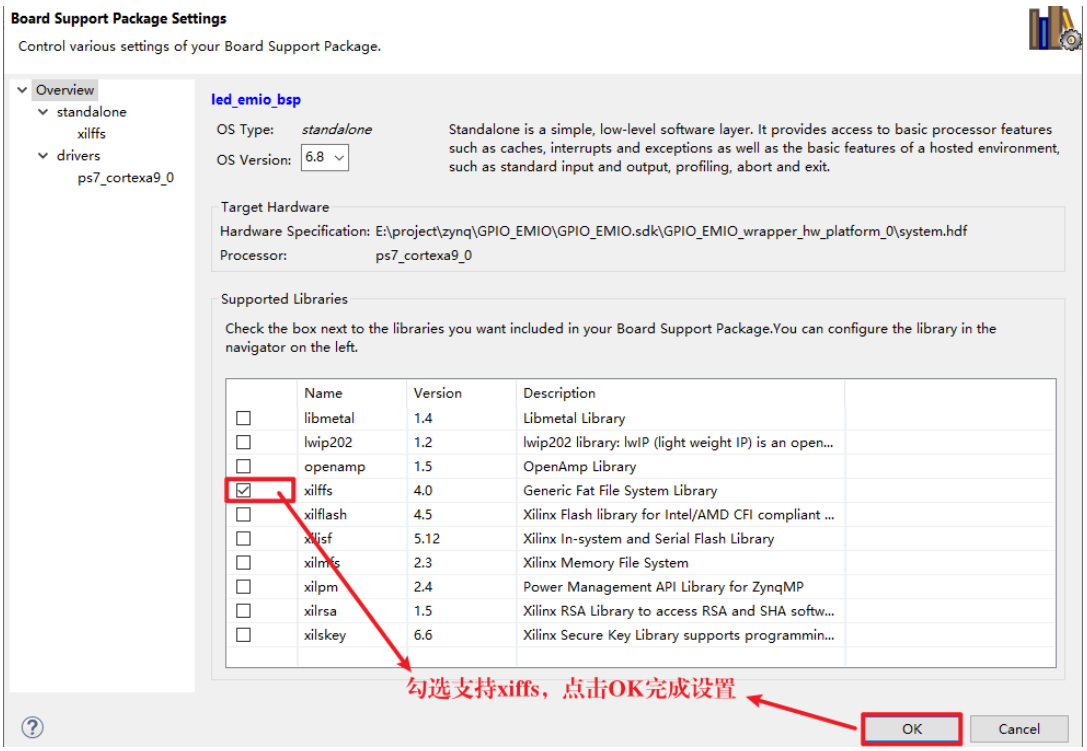

图 1-24 板级支持包设置

接下来新建一个 FSBL 工程，依次点击 File->new->Application Project，这里建议命名为 FSBL，方便区分。当然用户可以自行命名，只要与原工程不重复即可。在板级支持包项选择已有文件，点击 Next 进行下一步，如图 1-26 所示：

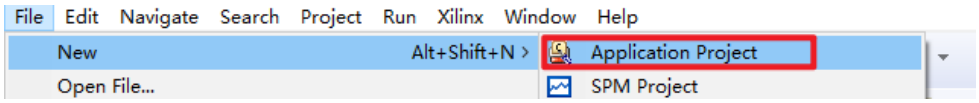

图 1-25 新建工程

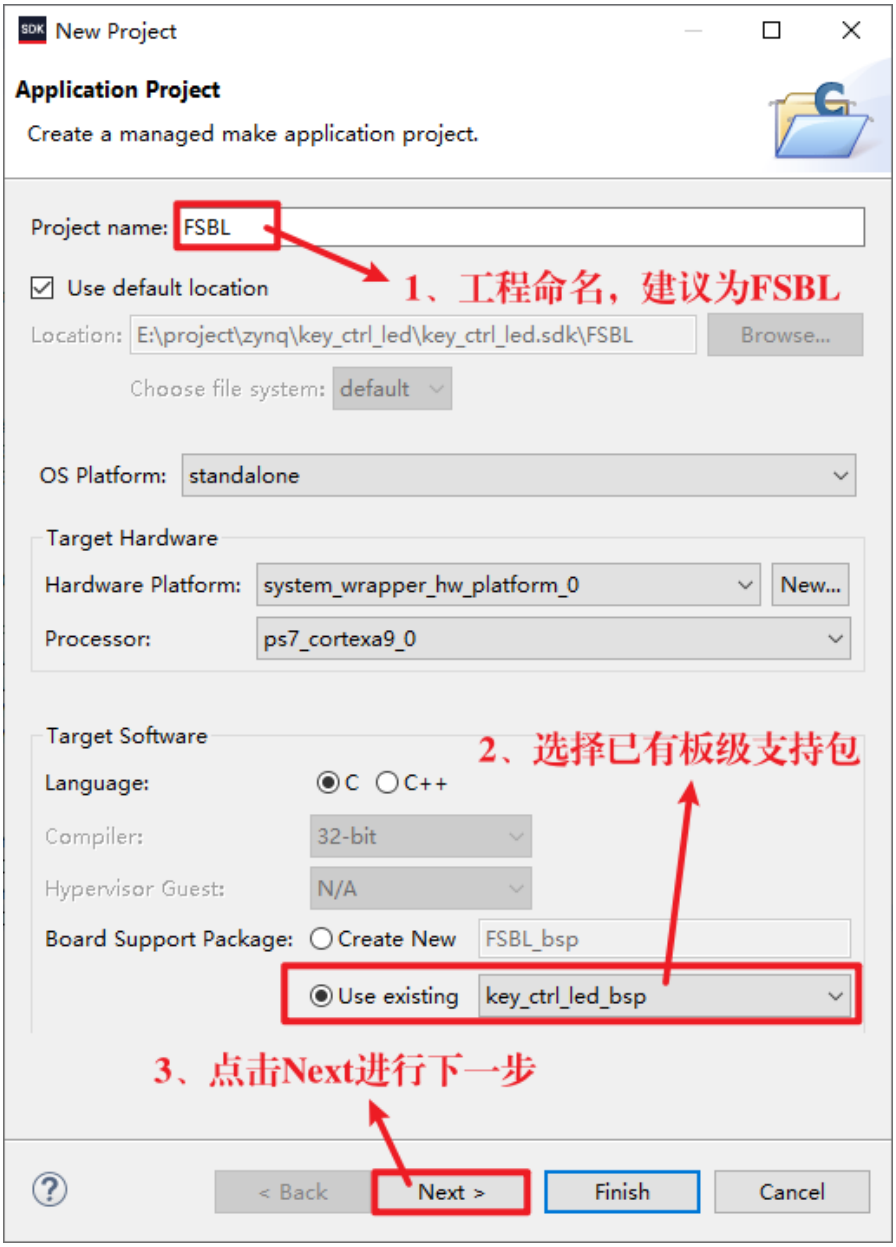

图 1-26 新建工程

此时系统会弹出工程模板，如图 1-27 所示，这里我们选择为 Zynq FSBL，可以看到右边对该模板进行了解释。该模板的作用是产生 FSBL，即第一阶段引导程序。点击 Finish 完成工程创建。

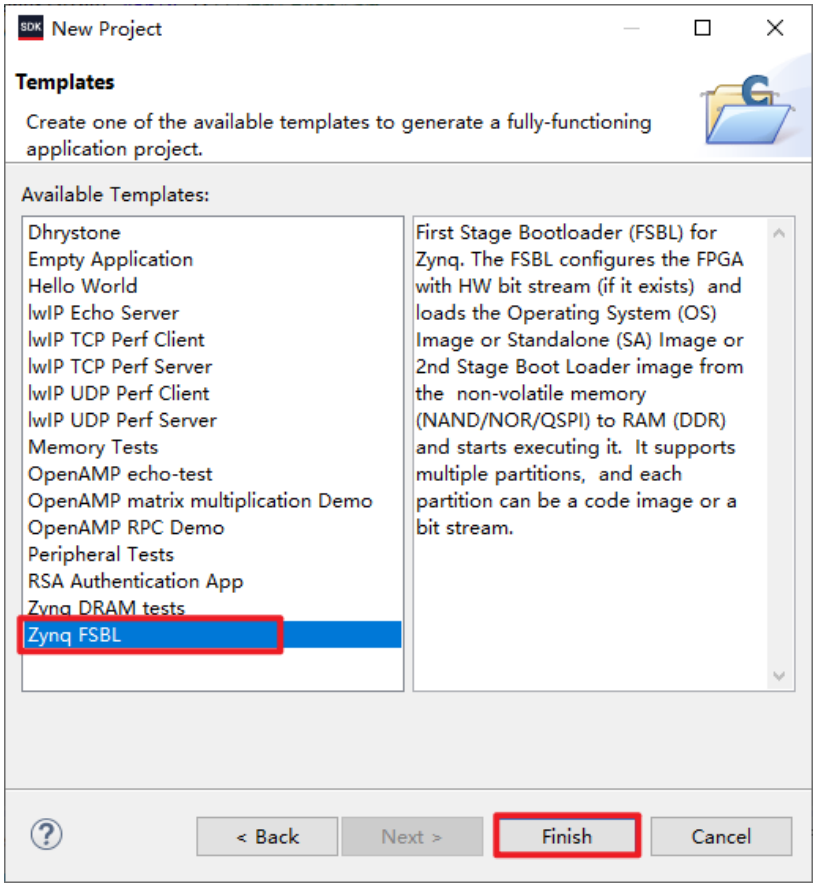

图 1-27 选择模板

接下来生成 bin 文件，点击 FSBL 工程文件，随后点击工具栏的 Xilinx，在展开的菜单栏中，选择“Create Boot Image”。这一步目的是快捷指定 bootimage 的路径，用户选中哪个文件夹，软件就会自动在对应文件夹下创建路径。

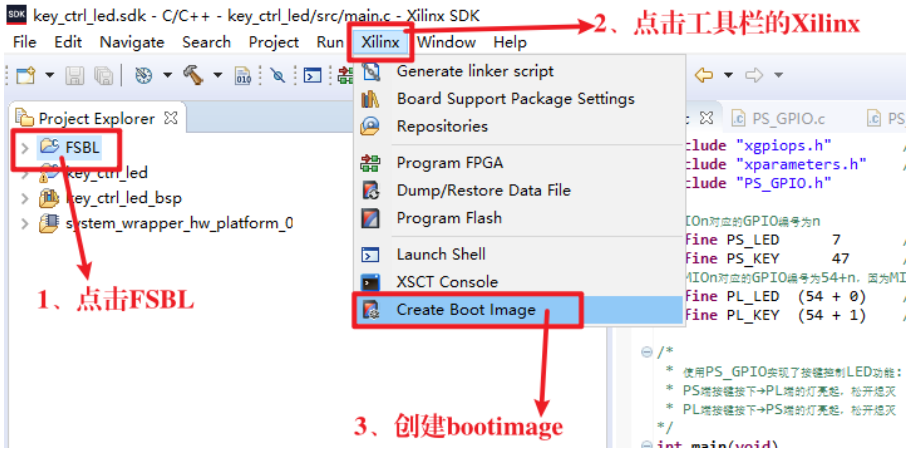

图 1-28 创建引导映像

打开后的界面如图 1-29 所示，有时软件会因为找不到路径而需要用户手动寻找。从图 1-29 可以看到该操作会生成一个 BOOT.bin 文件和 FSBL.bif 文件。FSBL.bif 文件由于本次设计不会用到所以暂不介绍，BOOT.bin 文件便是第一阶段的引导程序 FSBL 所需要用到的代码，其路径如图 1-29 所示。在前面固化理论中我们有说到，BOOT.bin 文件的组成为 FSBL.elf+原工程.bit+原工程.elf。从图 1-29 的 File path 中我们可以清楚的看到此时软件已经帮我们添加好了其中两个文件，我们还需要手动添加工程.elf 文件。

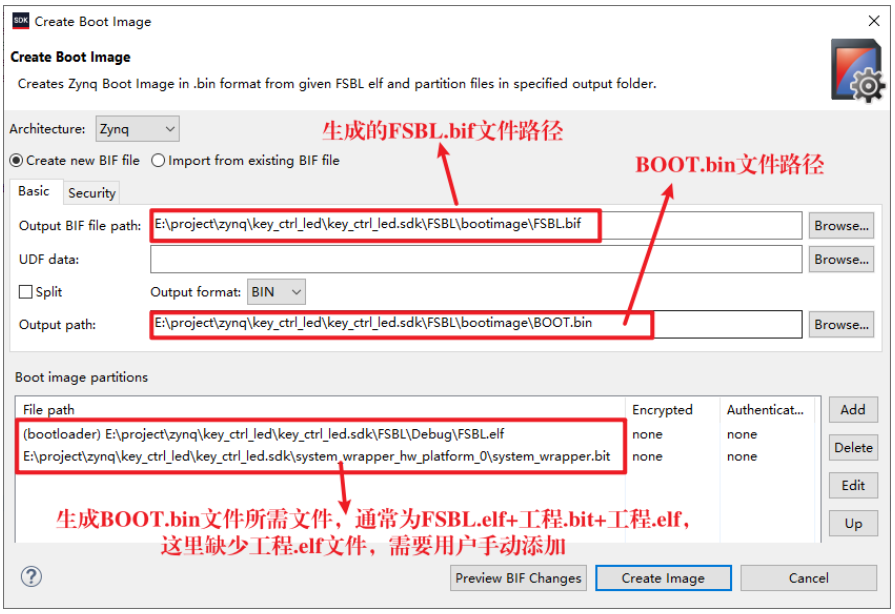

图 1-29 生成 bin 文件

接下来开始添加缺少的工程.elf 文件，点击 Add 找到对应路径下的文件，添加完成后如图 1-30 所示，工程.elf 文件文件路径如框中所示。添加完成后点击 Create image 开始生成 BOOT.bin 文件。

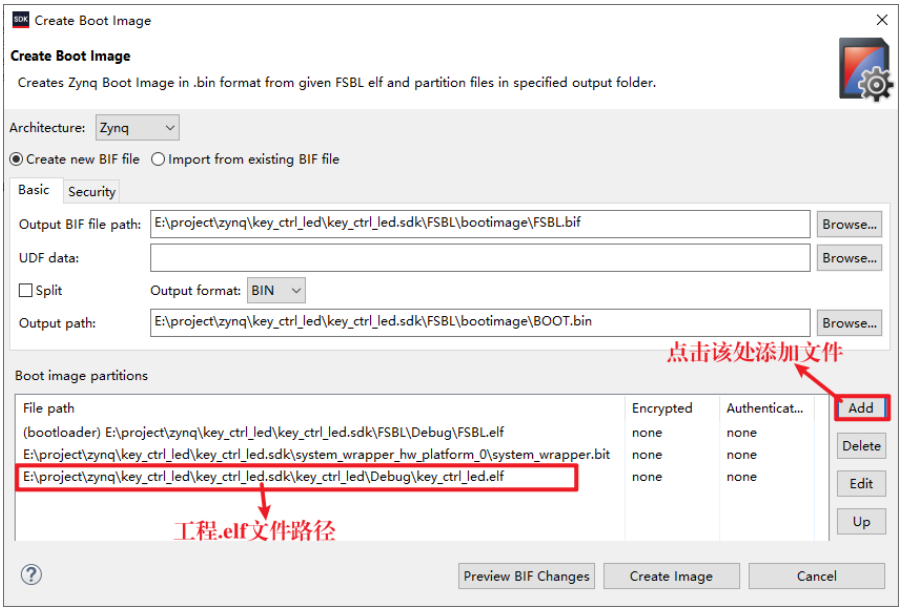

图 1-30 添加文件

当界面最下方的 Console 栏出现该提示时，代表 BOOT.bin 文件生成成功。接着打开文件夹，按照图 1-29 的生成路径，我们可以看到生成的该文件。

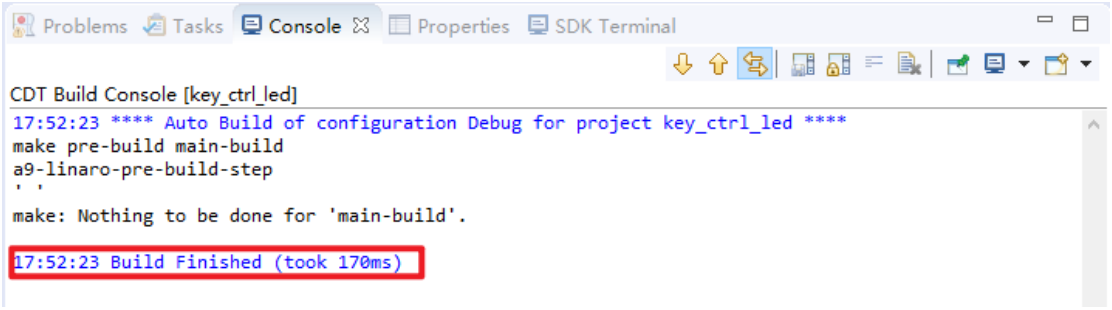

图 1-31 生成 Boot.bin 成功

图 1-32 BOOT.bin 文件路径

1.2.2 固化到 SD 卡

接下来只需将“BOOT.bin”文件拷贝到空白的 SD 卡中，再将 SD 卡插入开发板卡槽，将开发板断电重启，便可以开始程序的固化了。步骤如下：

（1）将 SD 卡插入读卡器中，再将读卡器插入电脑的 USB 口，接着将 SD 卡格式化为 FAT32 格式，如图 1-33 所示：

图 1-33 格式化 U 盘

（2）格式化完成后将我们生成的 BOOT.bin 文件拷贝到 SD 卡的根目录上，随后弹出读卡器，拔出 SD 卡。

图 1-34 拷贝文件

（3）将 SD 卡插入开发板中，注意 SD 卡插入时的朝向

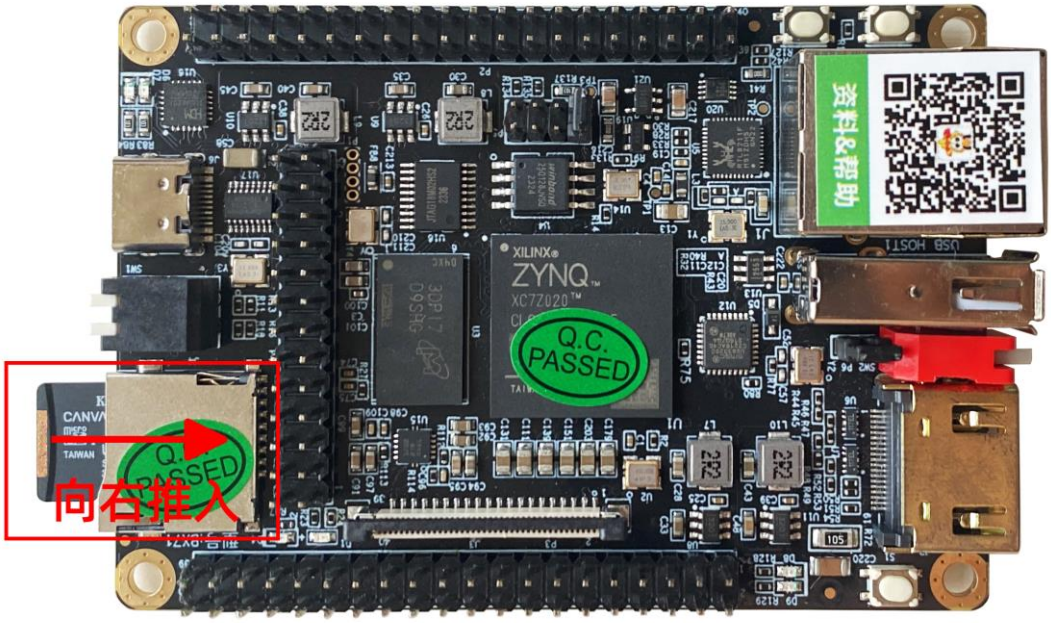

图 1-35 zynq 中插入 SD 卡

（4）手动设置 zynq 开发板启动模式，将 SD 卡卡槽下方的拨码开关全部拨到向下，如图 1-36 所示：

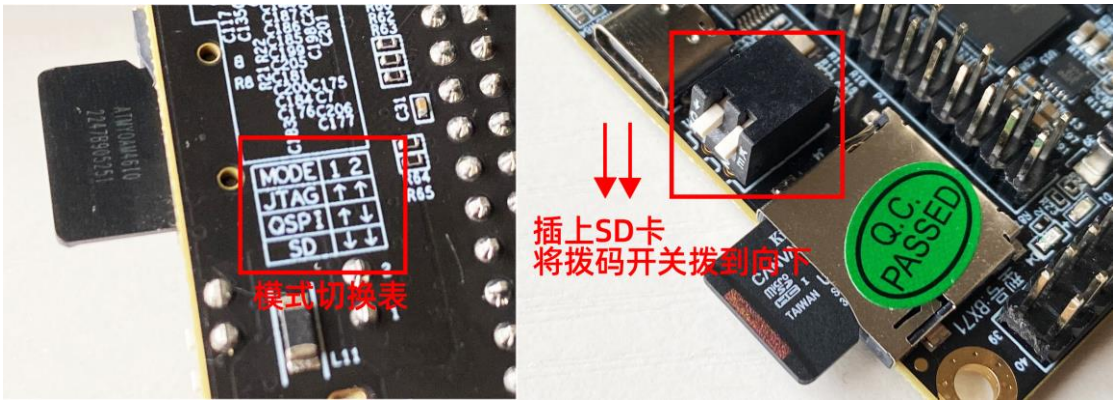

图 1-36 SD 卡模式配置

（5）将 zynq 开发板断电并重新上电，此时可以看到用来表示程序下载完成的 LED 灯在短暂延时后常亮，表明此时程序固化成功。按下按键，对应 LED 灯点亮，松开按键，对应 LED 灯熄灭，如图 1-37 和图 1-38 所示。

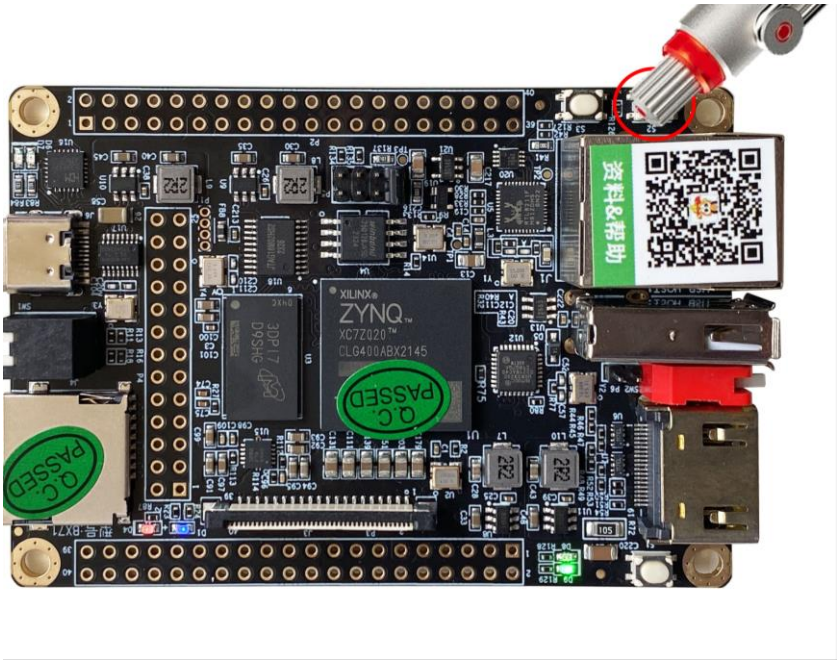

图 1-37 PS 侧按键按下

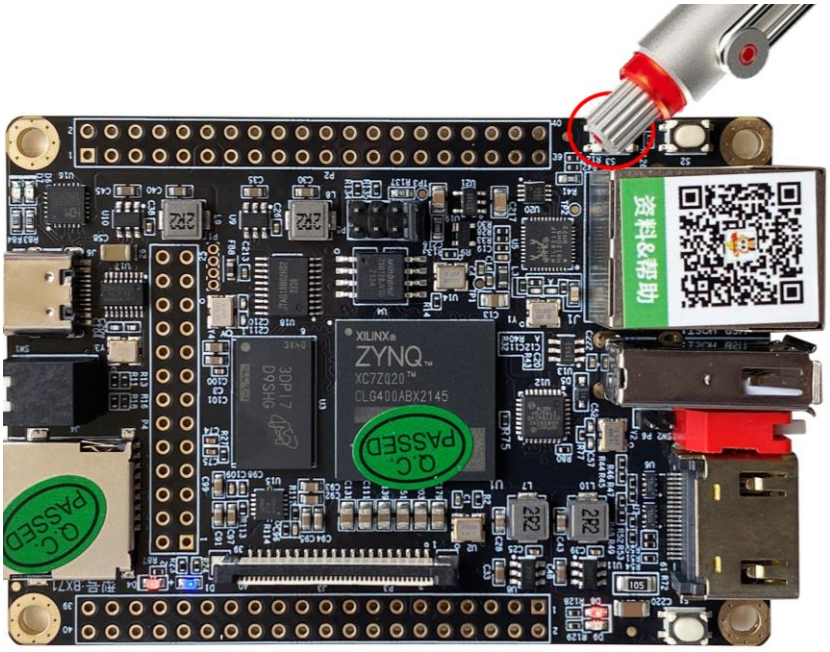

图 1-38 PL 侧按键按下

1.2.3 固化到 QSPI Flash

（1）点击 SDK 上方功能栏的 Xilinx，在展开的功能栏中选择“Program Flash”。

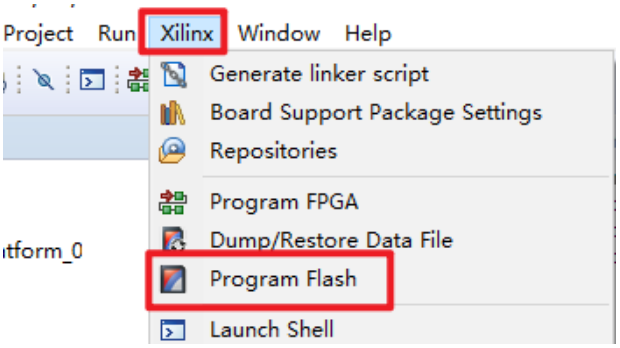

图 1-39 下载到 Flash

（2）接下来会进入如图 1-40 所示界面，我们需要在方框中分别选中生成的 BOOT.bin 文件和 FSBL.elf 文件（对应本次工程的 UART.elf 文件）。相关文件路径可以参考图 1-29。

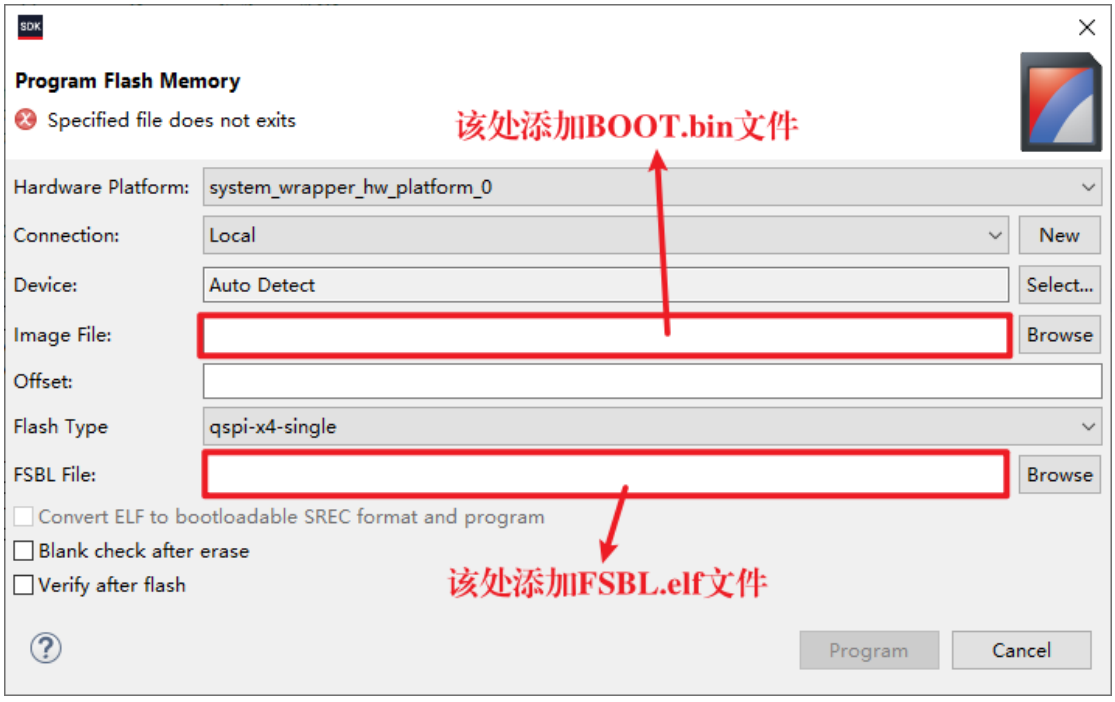

图 1-40 选择文件

（3）文件添加完成后如图 1-41 所示，点击 program 即可下载程序。下载程序时注意当前开发板启动模式应为 Jtag 模式，BX71V2.0 和 V1.0Jtag 启动模式设置如图 1-42 所示。

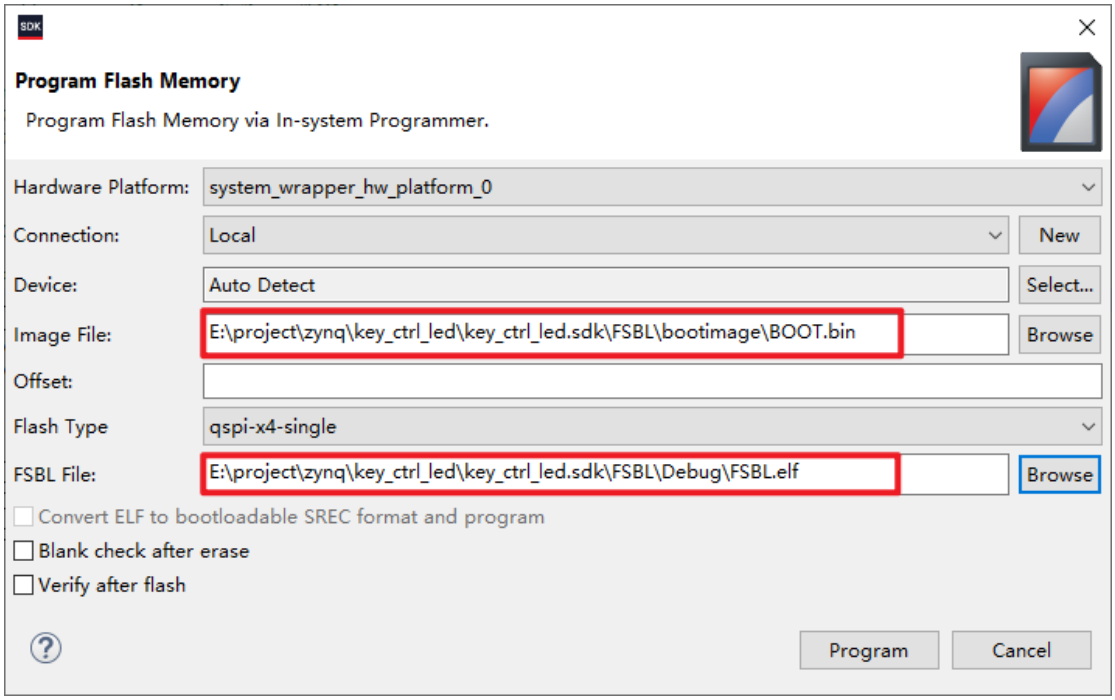

图 1-41 选择文件

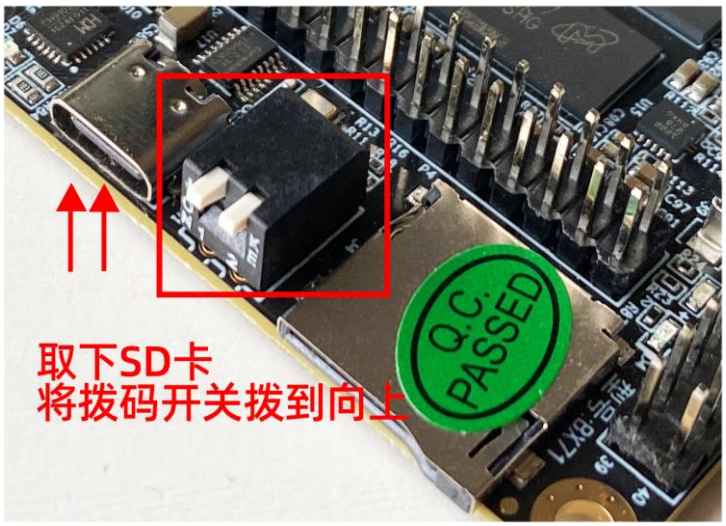

图 1-42 开发板 JTAG 启动模式

（4）点击 Program 后会出现如图 1-43 所示弹窗，提示正在执行编写操作。当界面最下方的 Console 栏出现如图 1-43 所示提示且时，代表程序已经成功固化到 QSPI Flash 中。

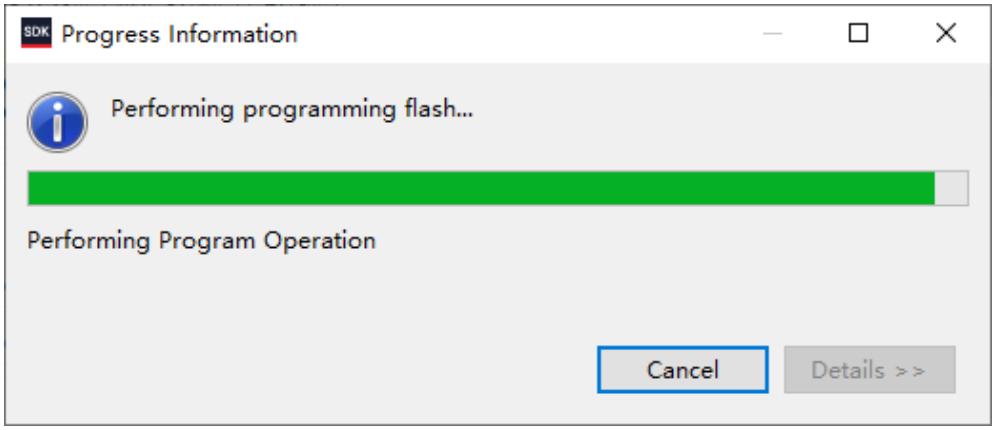

图 1-43 编写提示

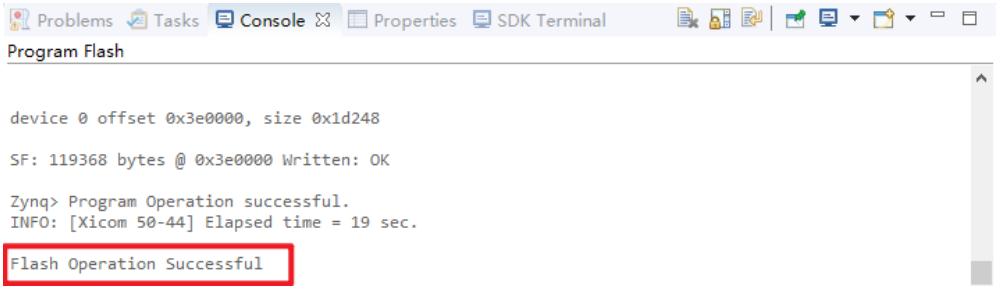

图 1-44 固化成功提示

（5）接下来将开发板切换到 QSPI 模式（↑ ↓），如图 1-45 所示：

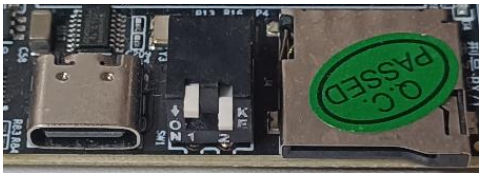

图 1-45 QSPI 启动模式

随后，将开发板断电并重新上电。按下按键，对应 LED 灯点亮，松开按键，对应 LED 灯熄灭，如图 1-46 所示。

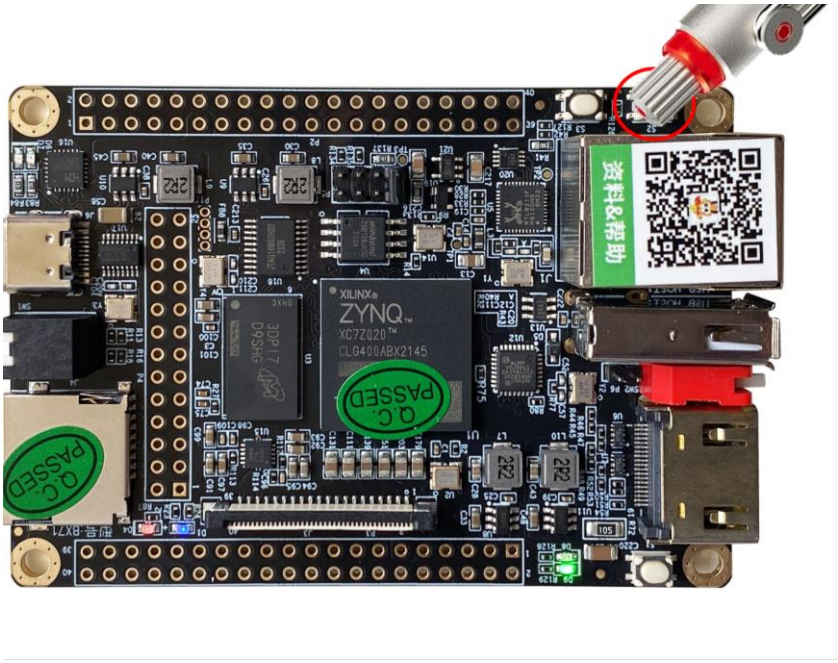

图 1-46 验证程序

1.3 章节总结

1. 本章实验以 key_ctrl_led 工程为例，一步步教大家如何创建 FSBL 工程，如何生成固化所需的 BOOT.bin 文件以及如何进行程序固化。程序在被固化后，即使掉电也不会丢失，等到重新上电后便会自动运行固化好的程序。用户在需要固化个人程序时，可以参考本章实验。在切换启动模式时，切记必须将开发板重新上电，否则启动方式将不会更改。
2. zynq 系统在启动时分为 3 个阶段，第 0 阶段用户无法进行操作， 该阶段完成对芯片的配置，而最主要的功能是确定启动模式。第一阶段完全由用户控制，该阶段主要根据用户的 BOOT.bin 文件完成 PS 的初始化和 PL 的配置。第二阶段为可选阶段，完成 Linux 等程序的启动，对于不需要使用到操作系统的情况下可以跳过该阶段。

用户通过 SDK 生成的 BOOT.bin 文件由 FSBL.elf 文件+原工程.bit 文件+原工程.elf 文件构成。bit 文件我们较为熟悉，是我们平时下载到开发板的配置文件；elf 文件为连接文件，参与程序的连接与执行。原工程.elf 文件在工程编译完成时便已经产生，FSBL.elf 文件通过我们创建 FSBL 工程产生。