Zedboard 开发例程--点亮 Led 流水灯

——基于 PS+PL+SmallRTOS

1、目标规划

硬件平台：Zedboard；

软件平台：Vivado、SDK；

实时操作系统：SmallRTOS；

实现功能：采用 PL 添加 IP 核，PS 控制 GPIO 的方式，控制 LED 流水灯，同时熟悉“小嵌”操作系统 SmallRTOS 的多任务功能；

实现流程：建立工程 添加 ARM 内核、GPIO IP 综合、实现、烧写 板级测试；

2、整体设计

整体框图为：

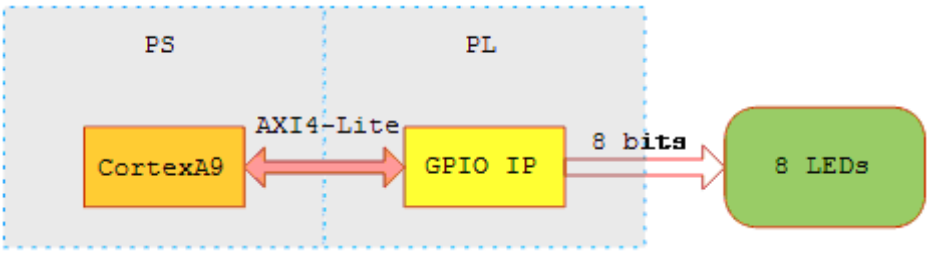

图 1、设计框图

核心功能由 ARM 软件实现。添加 PL 部分的 GPIO IP 主要目的是熟悉 IP 添加流程，如何和 ARM 内核连接，从而组建嵌入式系统。IP 功能是从总线上取出数据，送至 8 个 LED。

3、PL 部分实现

3.1. 建立工程

打开开发平台 Vivado（若没有安装，请自行到网络上搜索下载），主界面如下图所示，欢迎界面看起来和之前的 PlanAhead、XPS 的欢迎界面差别比较大了，可以说是焕然一新吧。

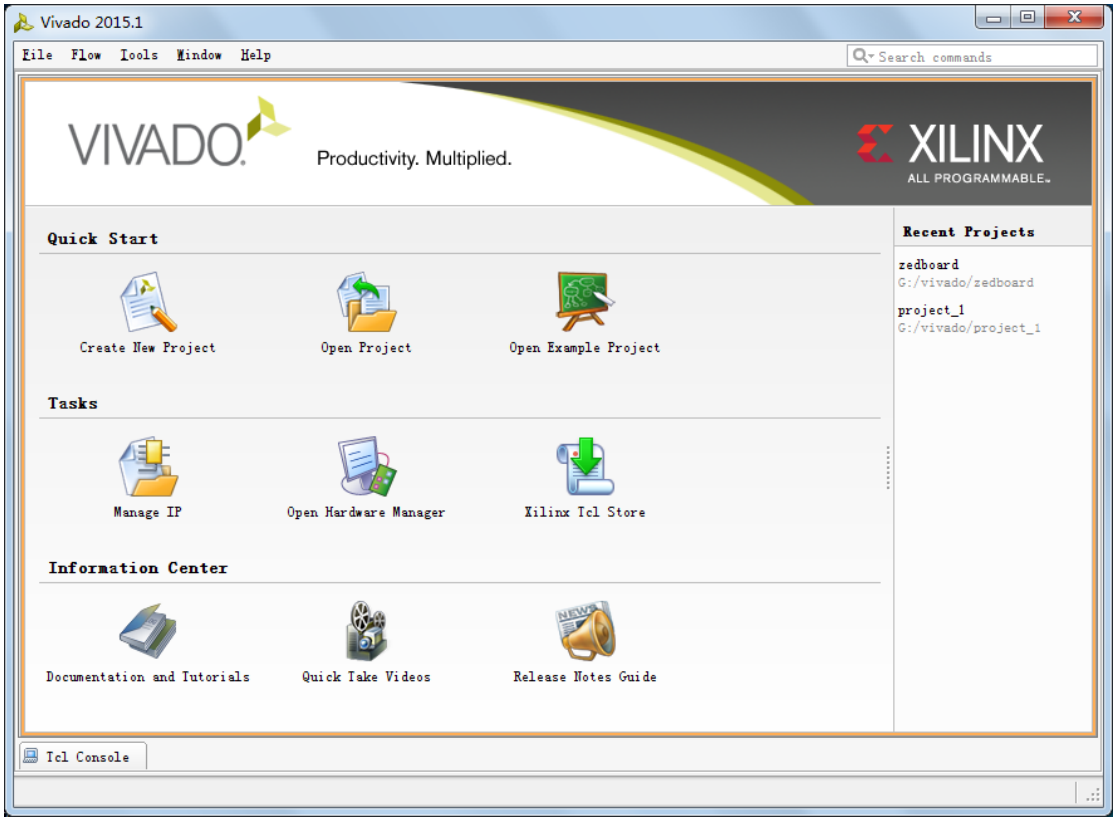

图 2、 欢迎界面

点击 Create New Project，进入新建工程的向导，逐步 next 即可。

图 3、 新建工程向导 step1

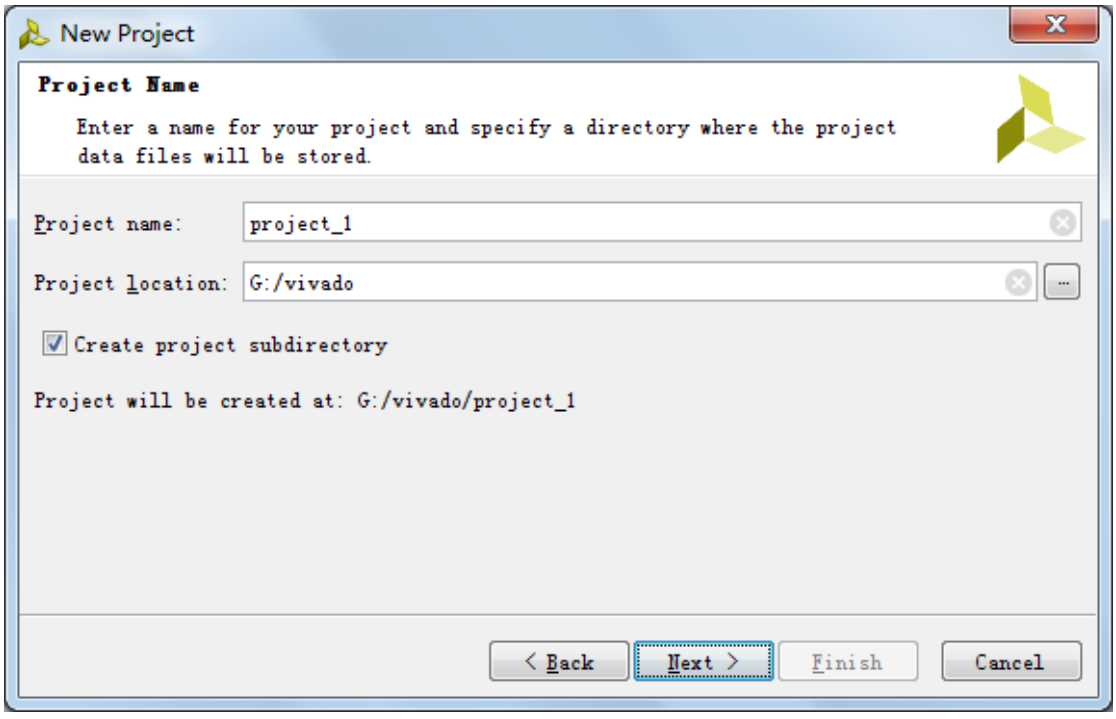

图 4、 新建工程向导 step2 指定工程名和存储路径

勾选图 5 复选框内的选项，表明不在此时指定源文件。

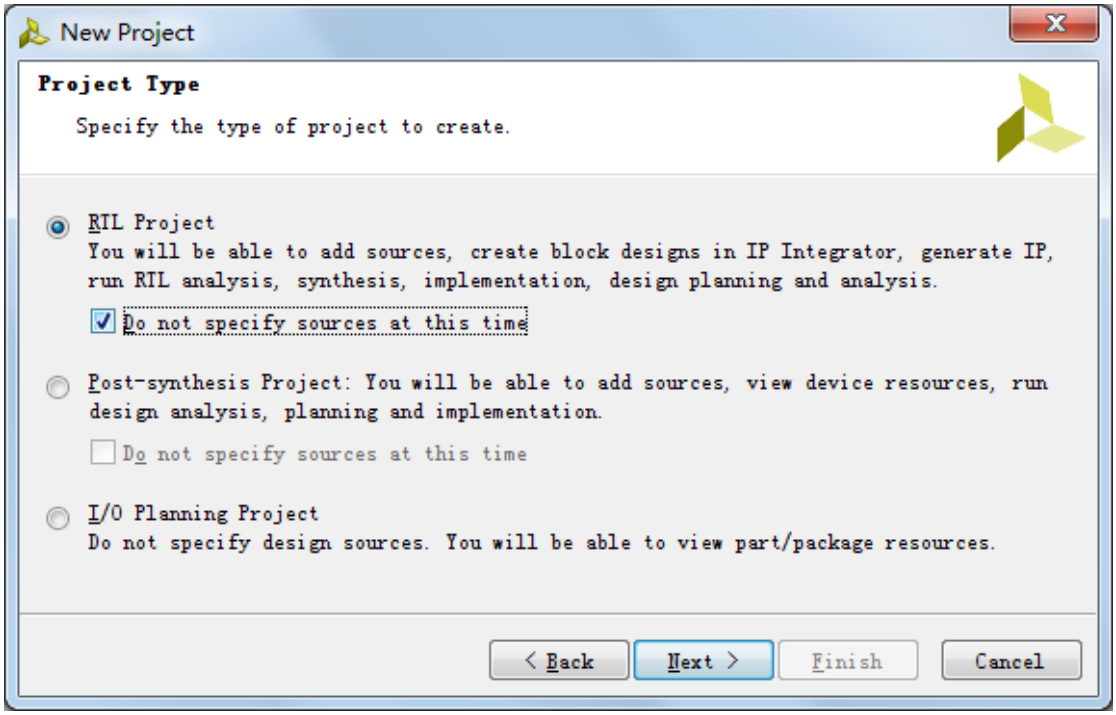

图 5、 新建工程向导 step3

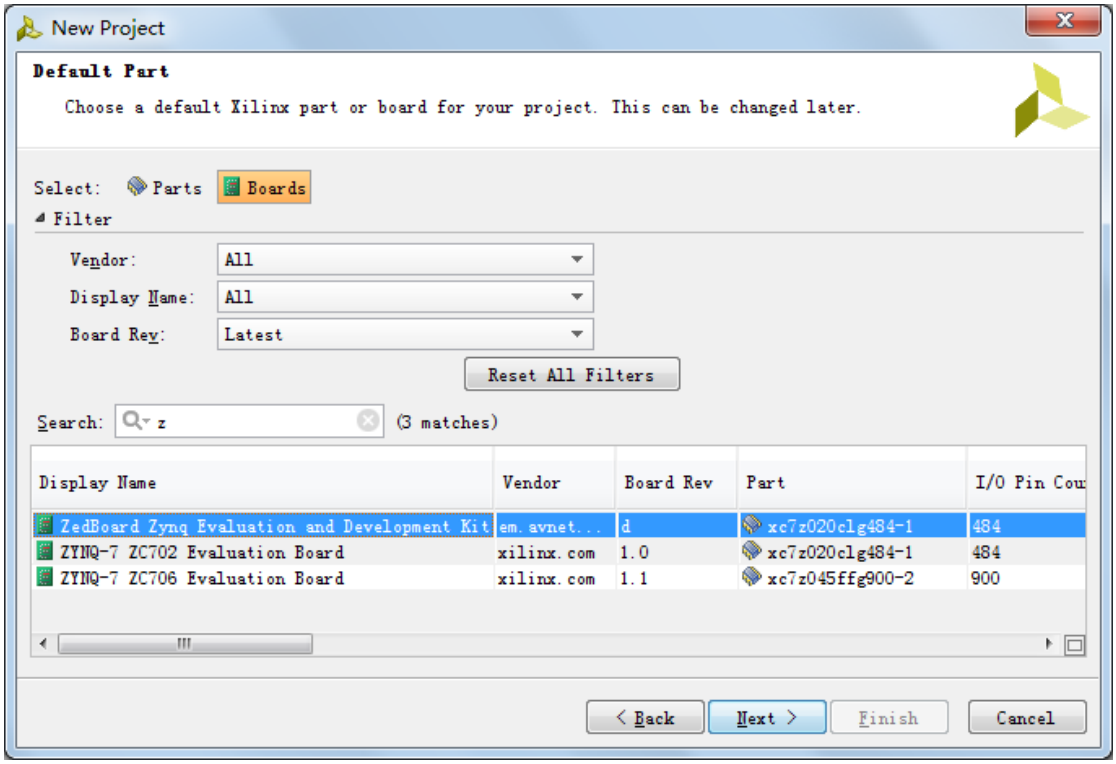

图 6、 选择器件/开发板

最后一步：所建立工程的概要信息。

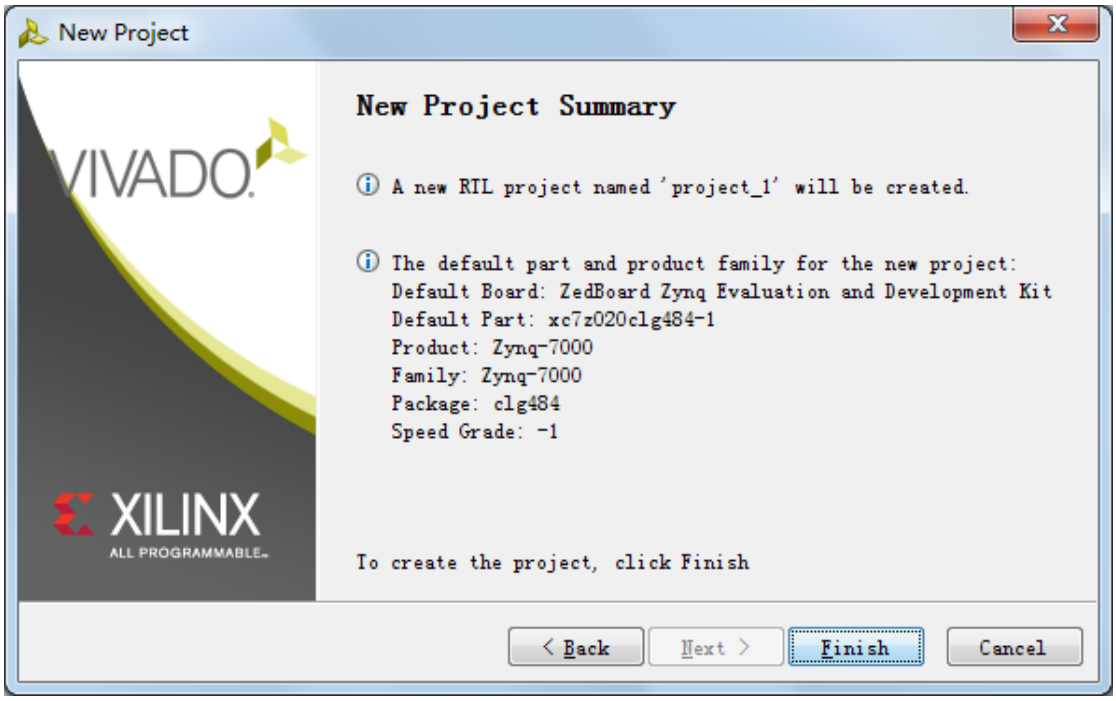

图 7、 新建工程信息总览

点击 Finish 之后，就进入 Vivado 主界面了，如图 8 所示。

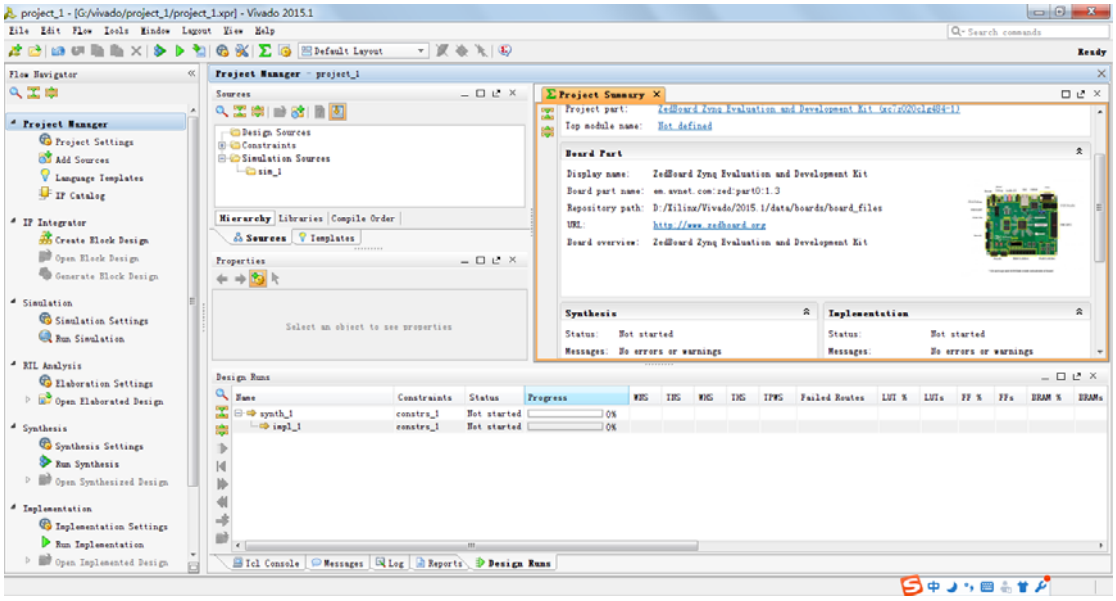

图 8、 Vivado 主界面

主界面主要包括：

Flow Navigator： 在此可以找到整个设计过程中涉及到的所有流程

Sources： 工程包含的设计源文件，源码、约束等

Properties： 所选中对象的属性信息

Project Summary： 在此可以查看工程信息

Design Runs： 在此可以查看提示信息、警告、错误、也可以输入 tcl 命令

3.2. 流程控制子窗口

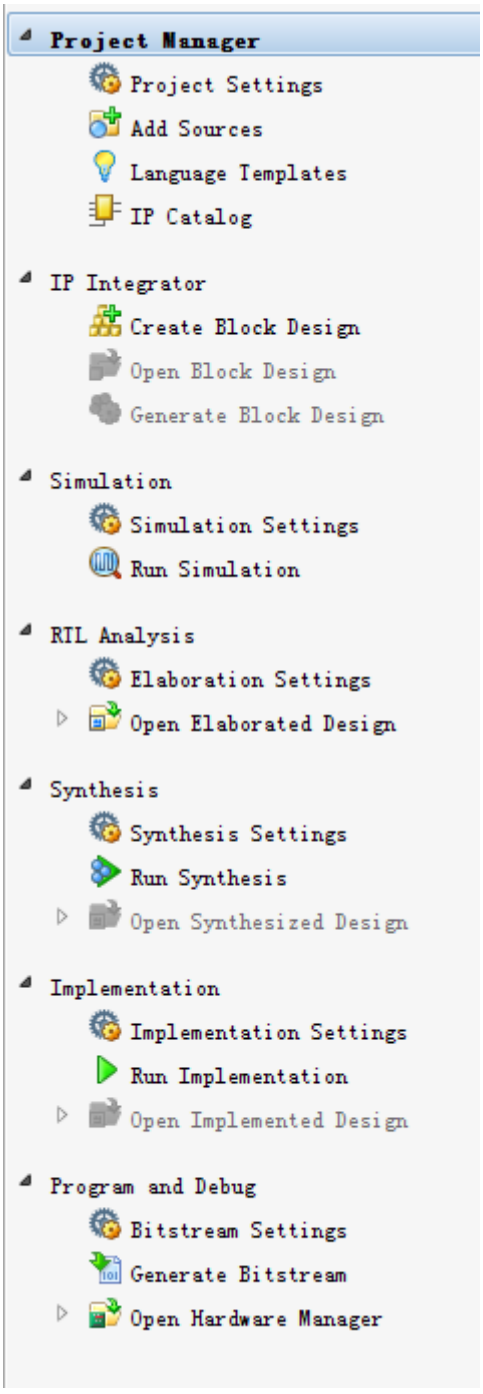

图 9、 流程子窗口

后续的“Add Sources”、“Simulation”、“Synthesis”、“Implementation”以及“Generate Bitstream”，都可以在该窗口找到执行菜单。

3.3、创建 Block

在左侧的 Flow Navigator 子窗口找到 IP Integrator（默认展开），找到该目录下的 Create Block Design 命令并单击。

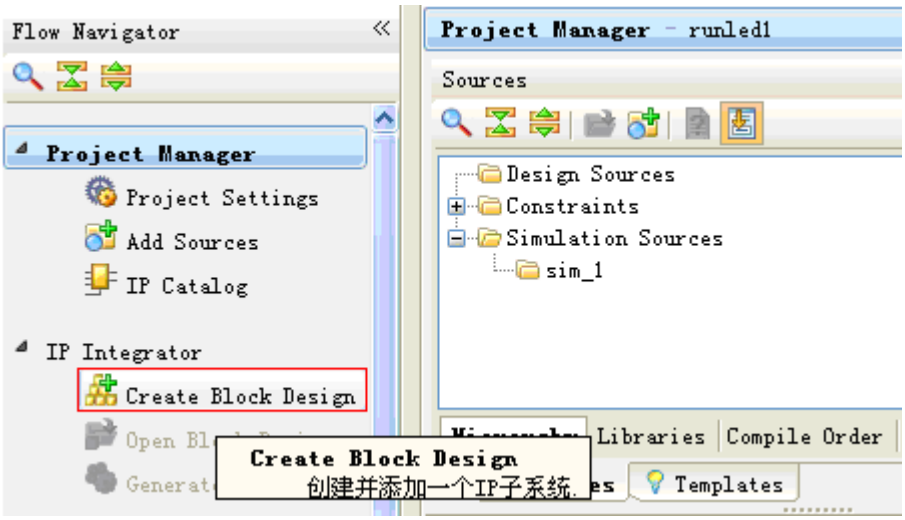

图 10

命名为 zedboard，如图 11 所示。

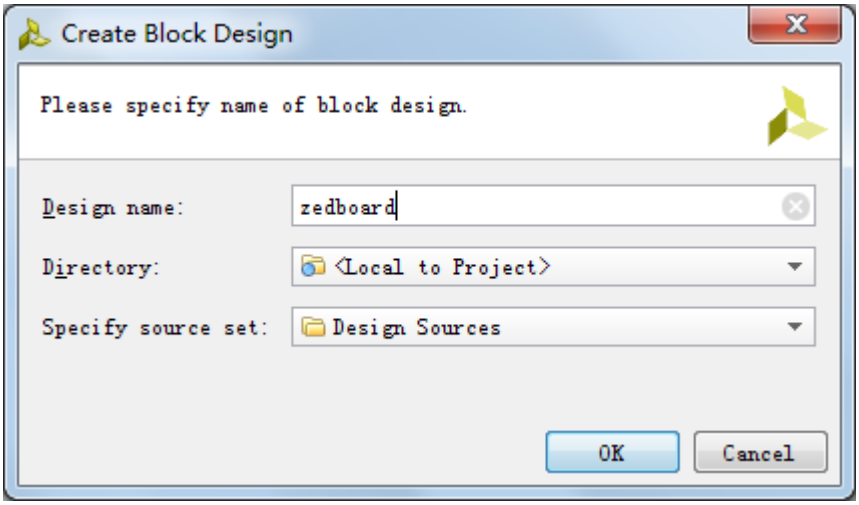

图 11

进度条跑完之后界面发生变化，出现了 Diagram 子窗口，Sources 自选项卡右边多出了 3 个选项卡，如图 12 所示

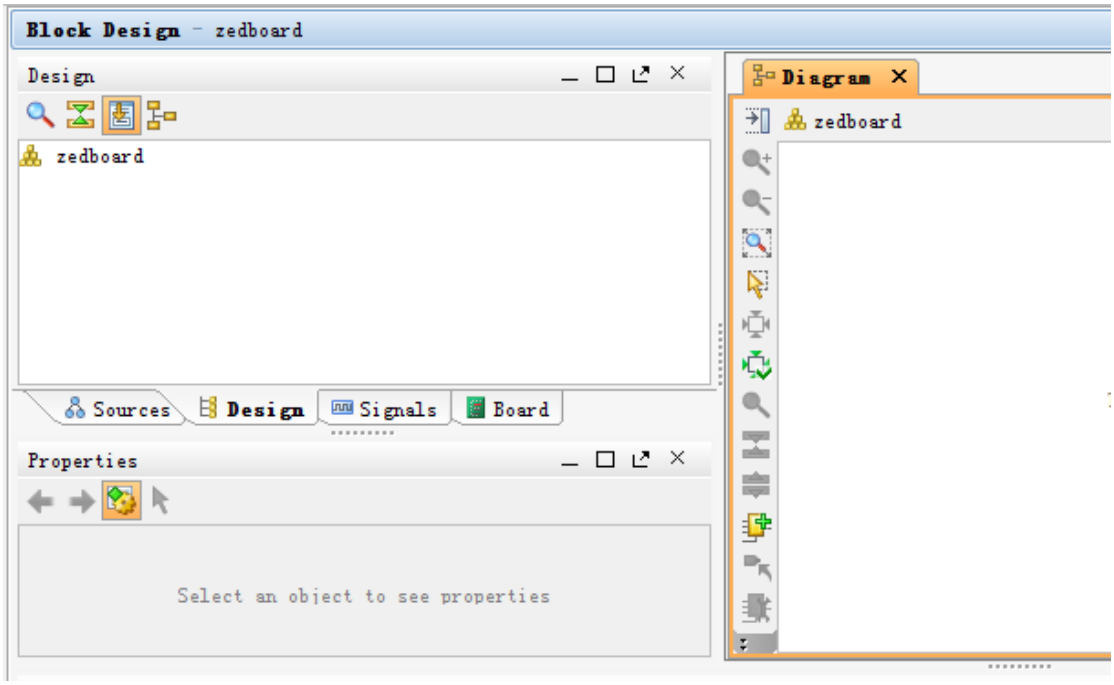

图 12

3.4、添加 cpu

在 Diagram 子窗体中找到 Add IP 按钮，位置如图 13 所示的提示框左上方。

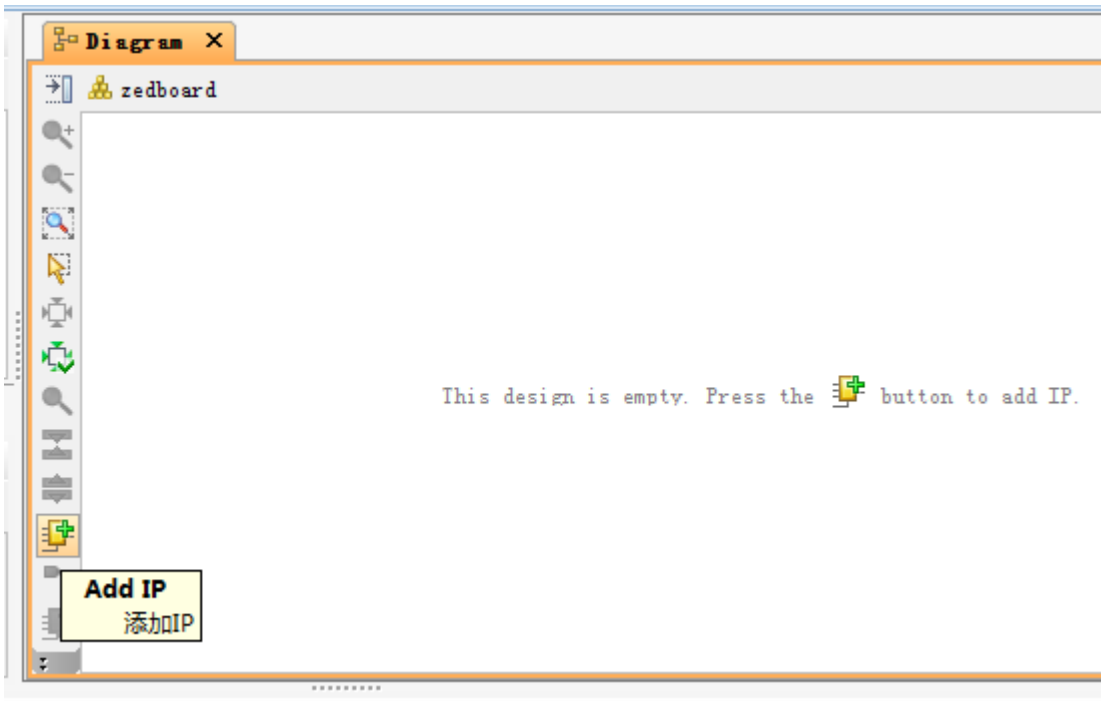

图 13

点击按钮后，弹出 IP 搜索对话框，如图 14 所示。

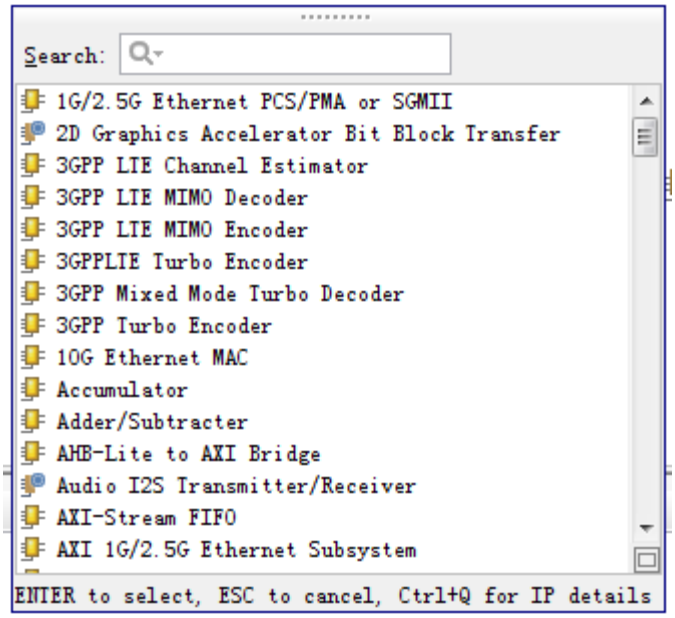

图 14

输入 zynq，对 IP 进行过滤，结果如图 15 所示。

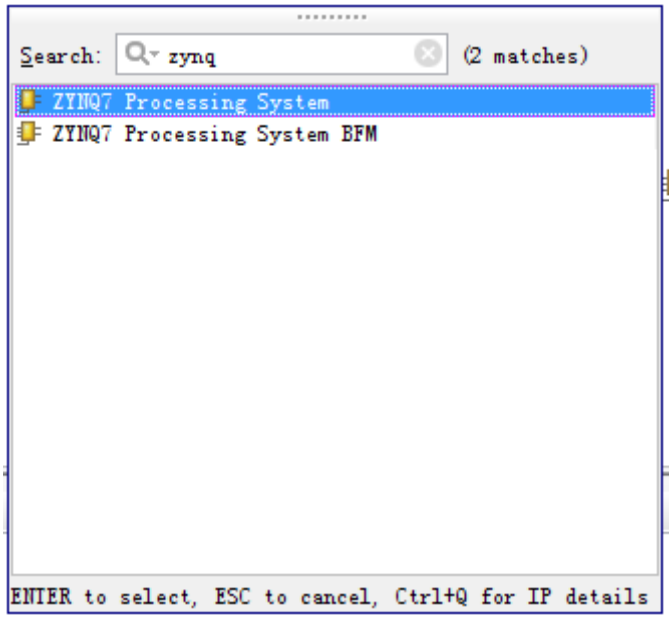

图 15

双击 ZYNQ7 Processing System，会自动添加该 IP，结果如图 16 所示。

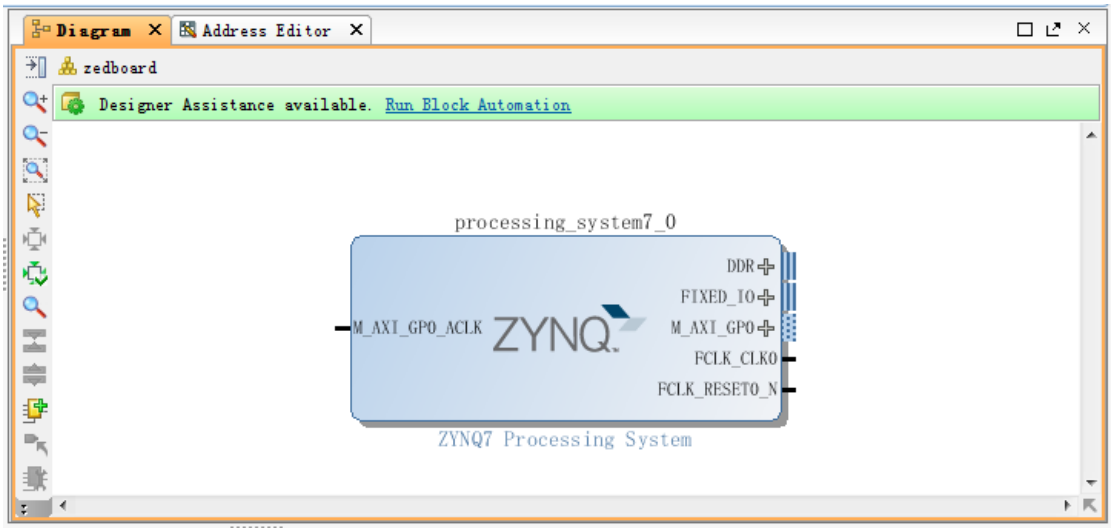

图 16

除了双击之外，还可以选中后回车；还可以选中后拖拽至 Diagram 窗口中释放。

3.5、添加其他 IP

用同样的方法查找并添加 GPIO IP，如图 17 所示。

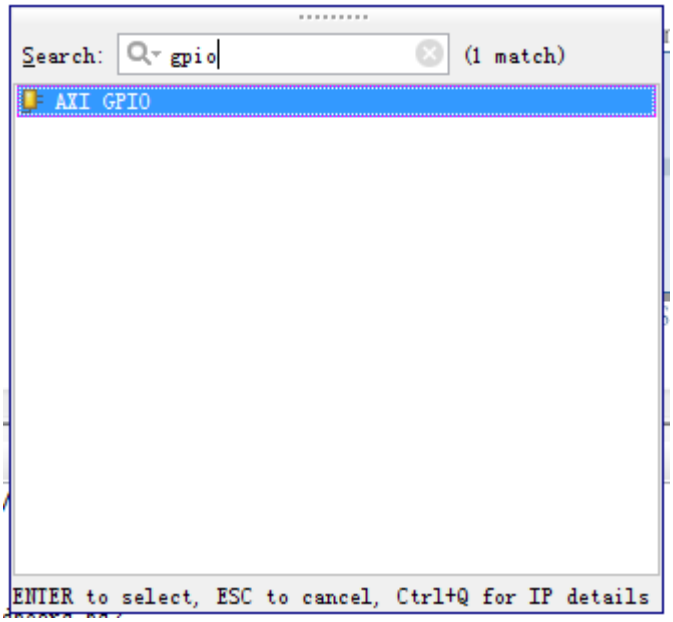

图 17

添加完成后的情形如图 18 所示。

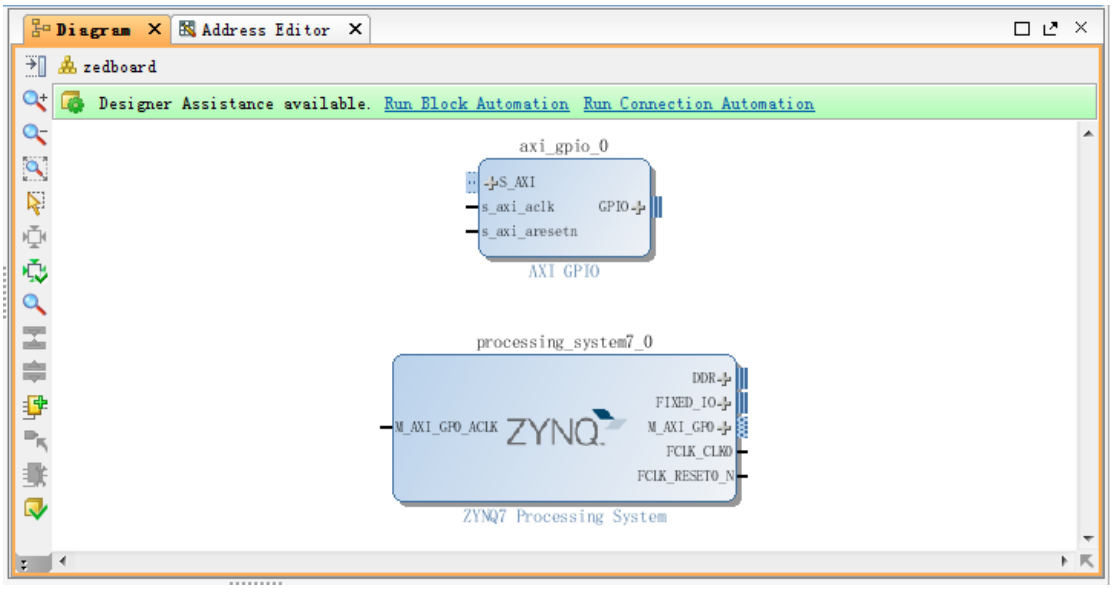

图 18

在 XPS 中添加 AXI IP 时，XPS 会自动完成总线连接，创建相关端口，Vivado 默认不做任何工作。还可以看到总线的时钟和复位信号被单独拿了出来，如果没记错的话，之前是和其他总线信号放在一起的，封装在 BUS_INTERFACE 中的。

3.6、端口连接和 IP 配置

图 18 中可以看到 Run Block Automation 和 Run Connection Automation 选项，Run Block Automation 可以为模块创建端口，Run Connection Automation 可以完成端口的自动连线。

点击 Run Block Automation，进入图 19 所示的界面。

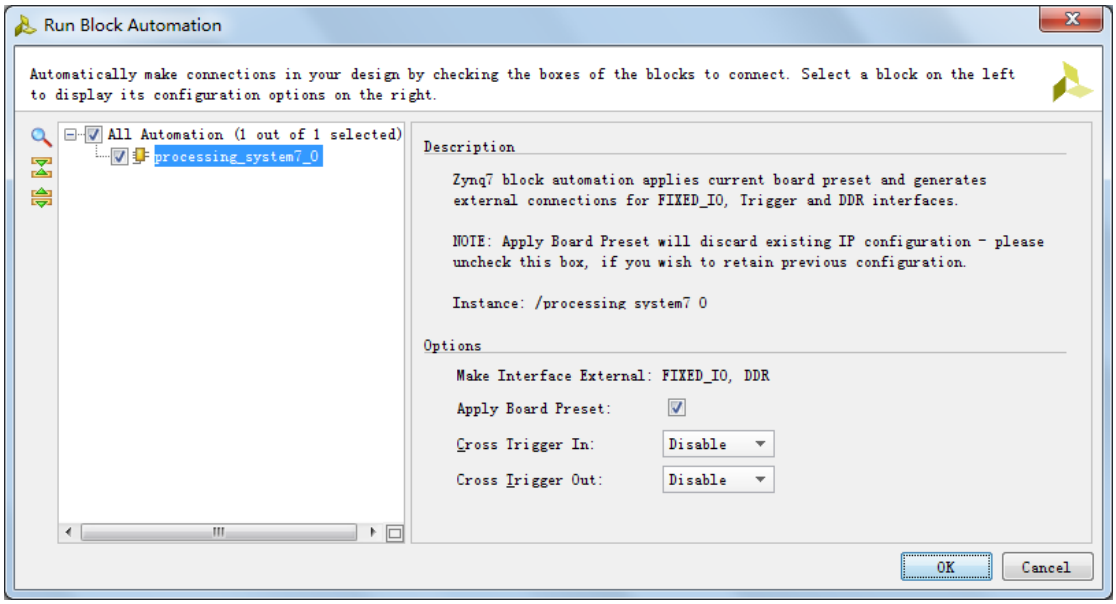

图 19

看一下注释，该过程是将开发板的预设置导入进来，并且将 FIXED_IO(MIO)、DDR 接口生成外部连接端口。OK，进度条跑完之后，界面发生变化，如图 20 所示。

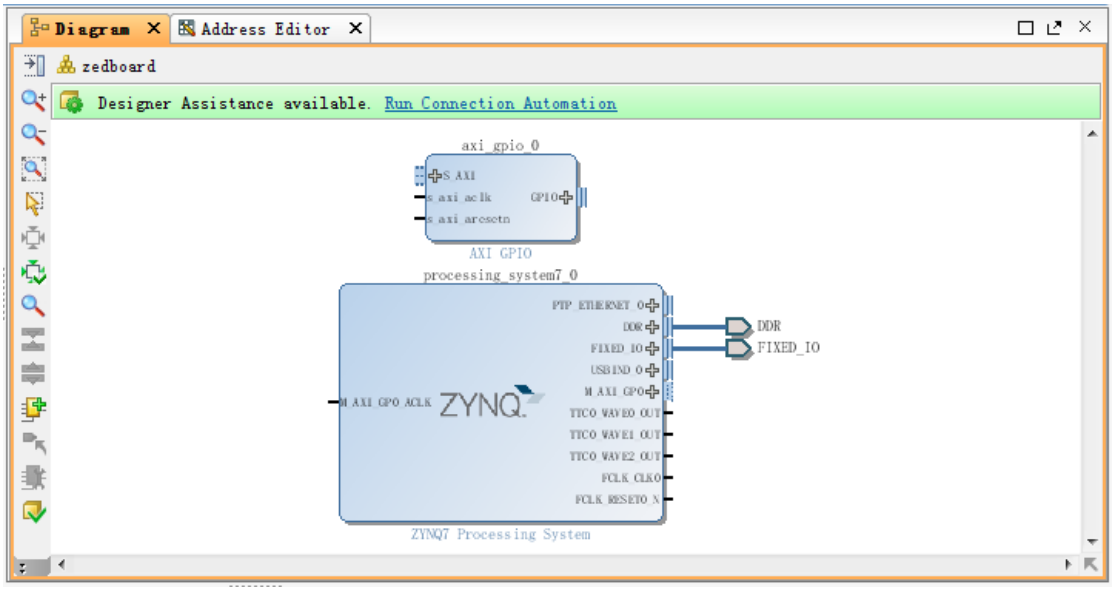

图 20

DDR，FIXED_IO 已经连接到端口，还多出了 TTC0_WAVE0_OUT 等选项，应该是对 PS7 进行了参数配置造成的，双击 PS7 可以对之进行配置，界面如图 21 所示，和 XPS 中大同小异，配置过程不多说了，这里没有改动。

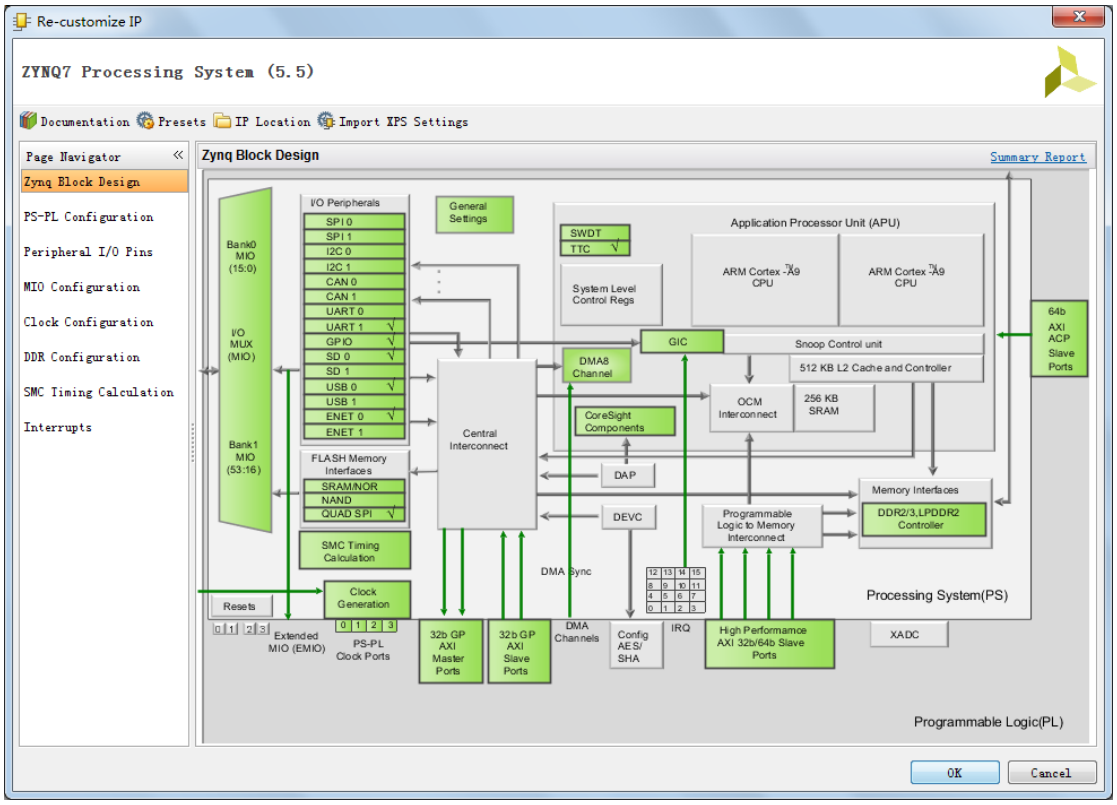

图 21

双击 GPIO IP 对之进行配置，主要是将位宽改为 8bits，如图 22 所示。

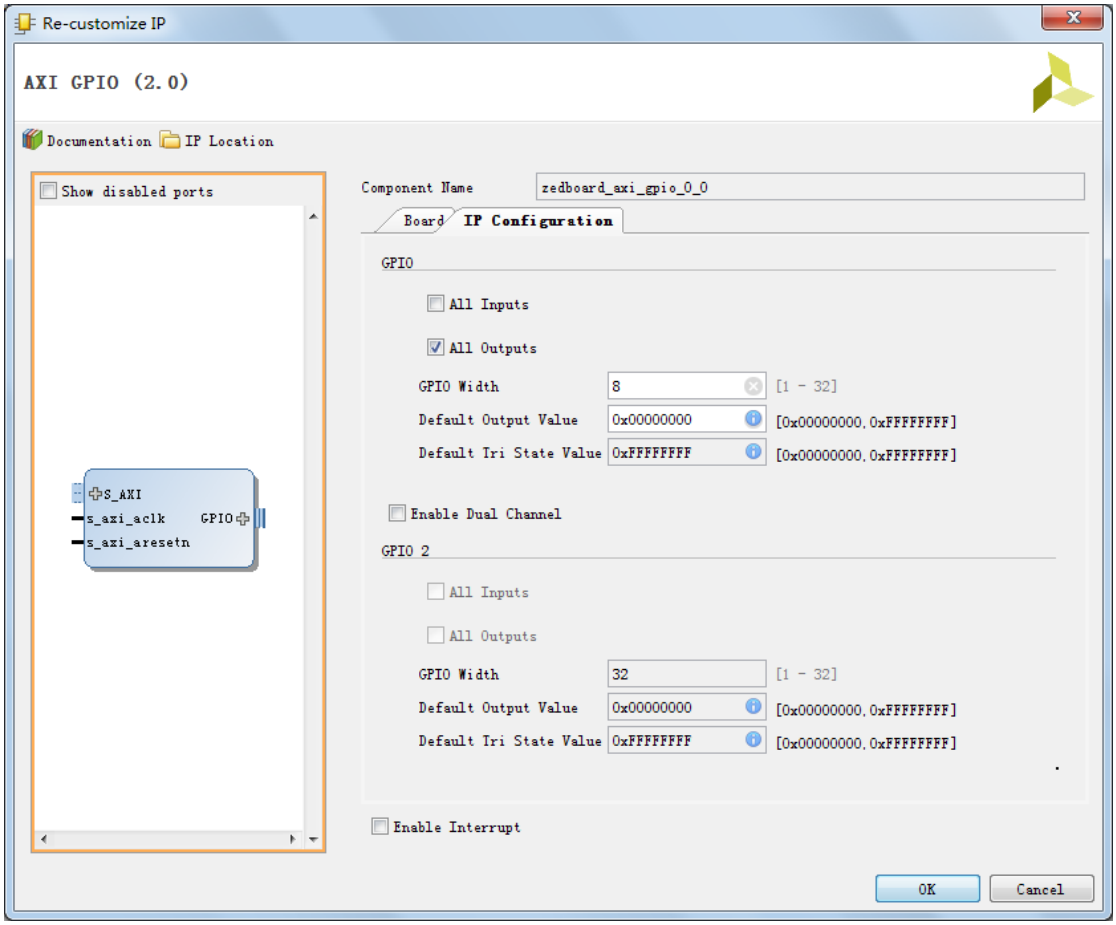

图 22

点击 Run Connection Automation，选择，如图 23 所示。

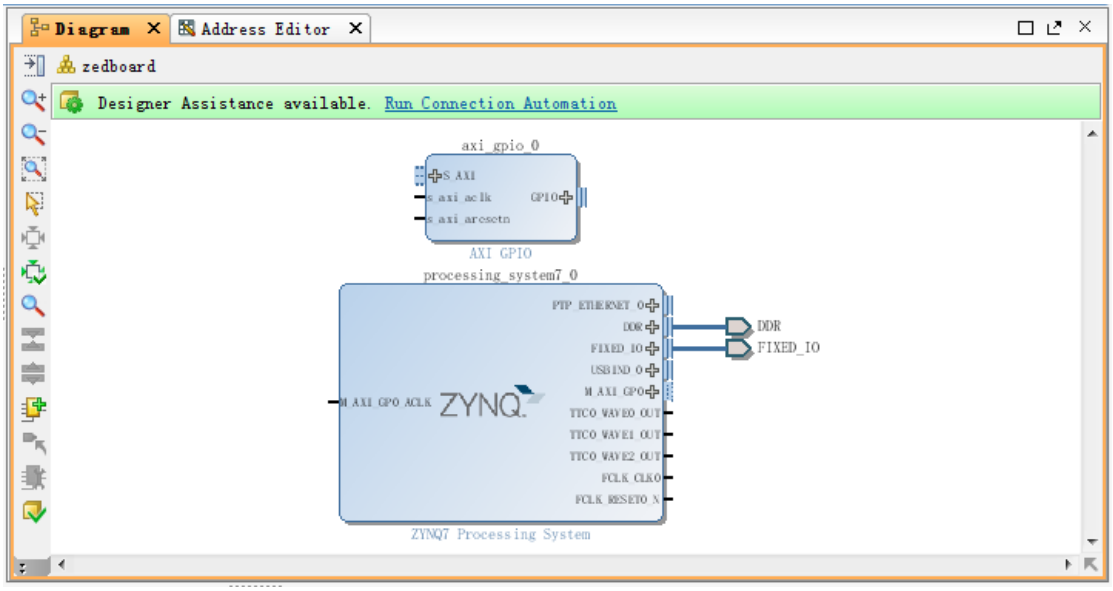

图 23

弹出对话框如图 24 所示，提示将 gpio 的 AXI 接口映射到主端的地址空间。还询问是否自动连接时钟，这里不管它，直接 OK。

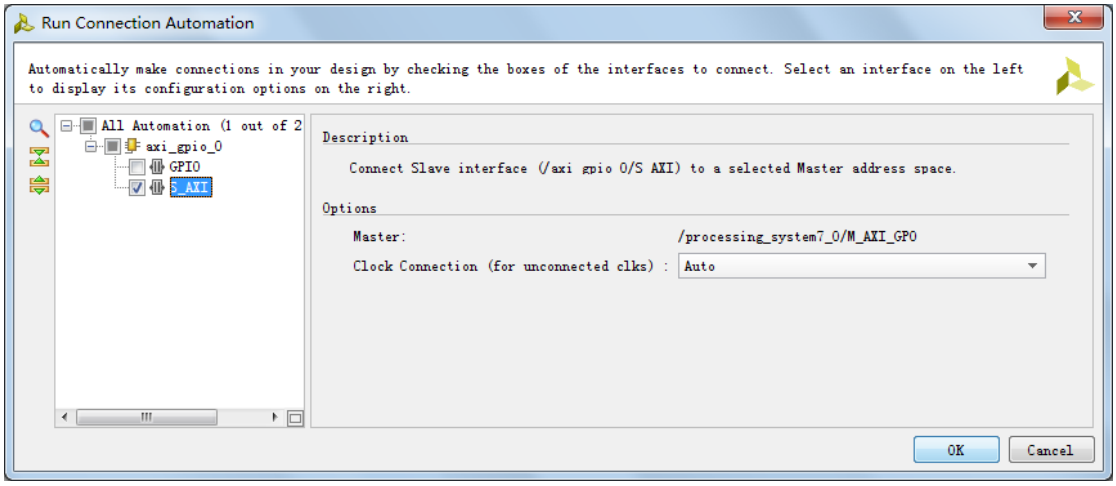

图 24

连线结果如图 25 所示。

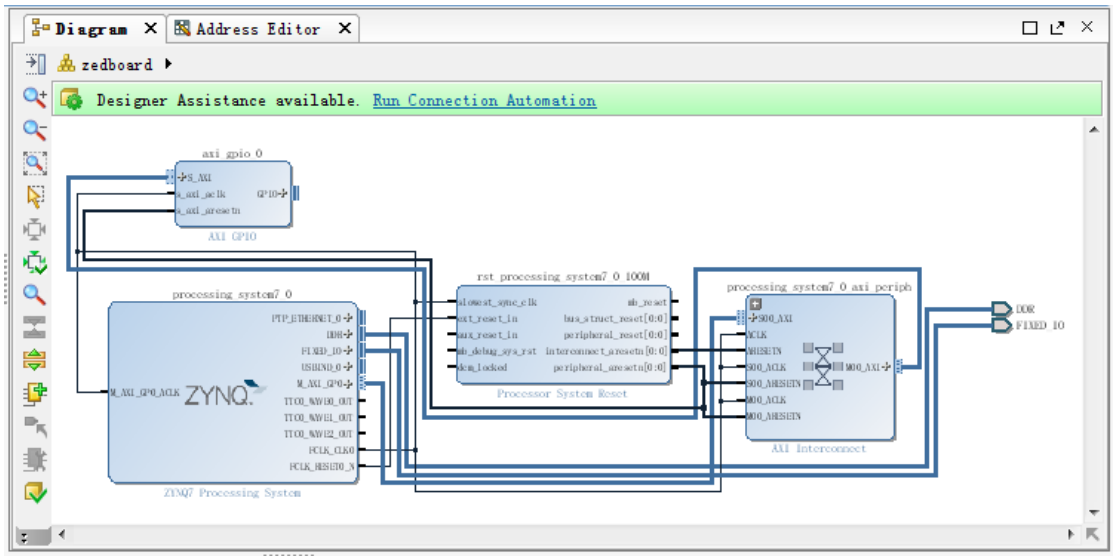

图 25

系统自动添加了互联模块（AXI Interconnect，和 XPS 相同）和一个时钟、复位控制模块。

再次 Run Connection Automation，将 GPIO 连接到输出端口，在图 26 所示对话框中选择开发板端口为 leds_8bits。

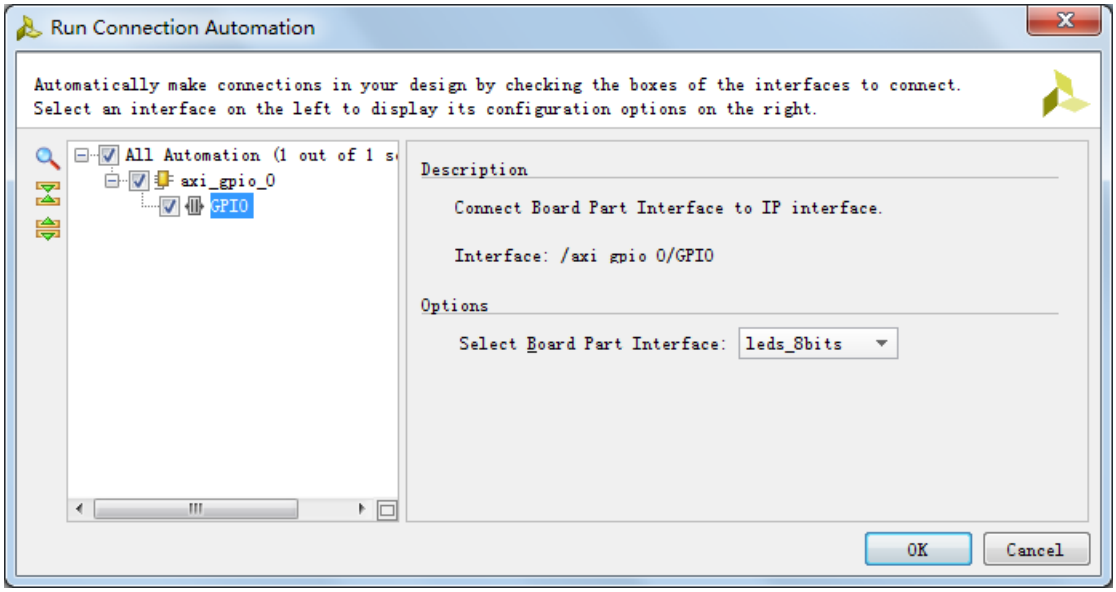

图 26

终于完成了，不容易啊，最终结果如图 27 所示。

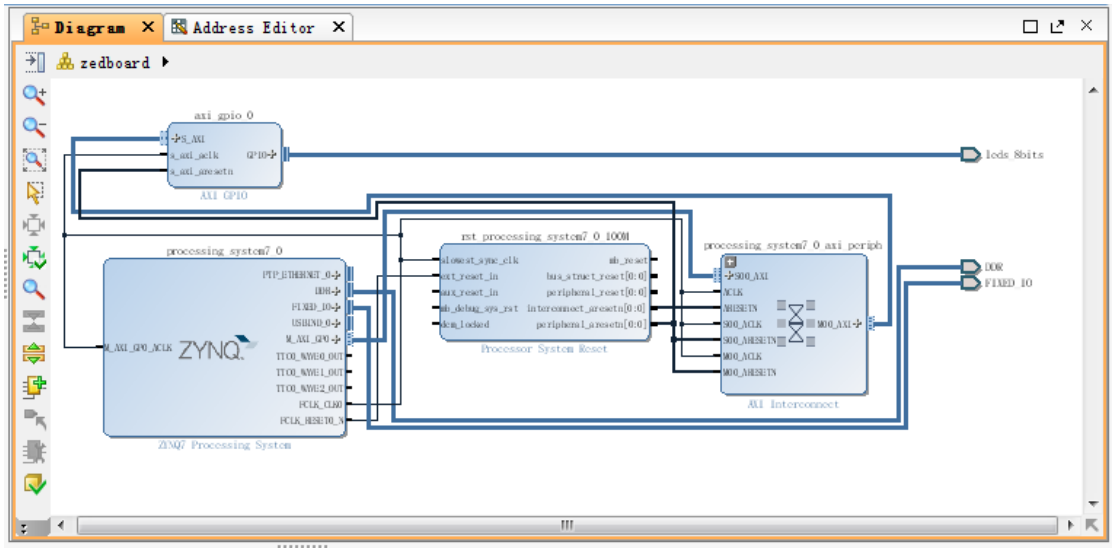

图 27

3.7、为 Block Design 创建 wrapper 文件

点击 Sources 自选项卡，按照图 28 的指示为该 Block Design 创建 wrapper 文件，bd 文件是不可以作为设计顶层的，为了方便顶层调用，需要为之生成一个 wrapper 文件。

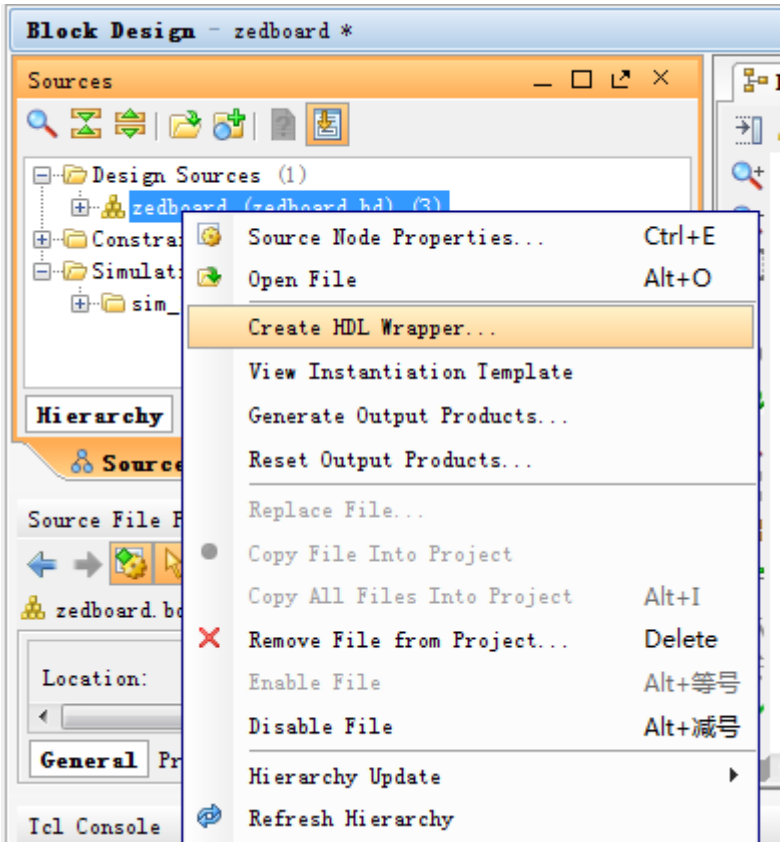

图 28

选择让 Vivado 帮助我们管理 wrapper 文件，如图 29 所示。

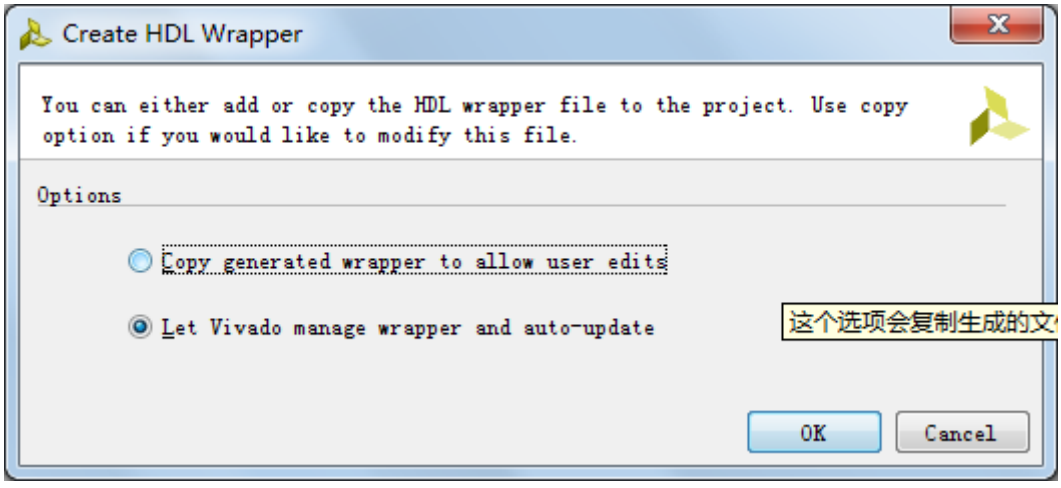

图 29

3.8、添加约束文件

这里我也不确定是否需要添加 DDR 和 MIO 相关引脚的约束，暂时不管他们。由于所有连接都是自动完成的，有理由相信 Vivado 为这些引脚做了约束，所以，这里我忐忑的偷个懒，不写约束代码，直接 Generate Bitstream。

果然直接过了，估计没问题了，转战 SDK。

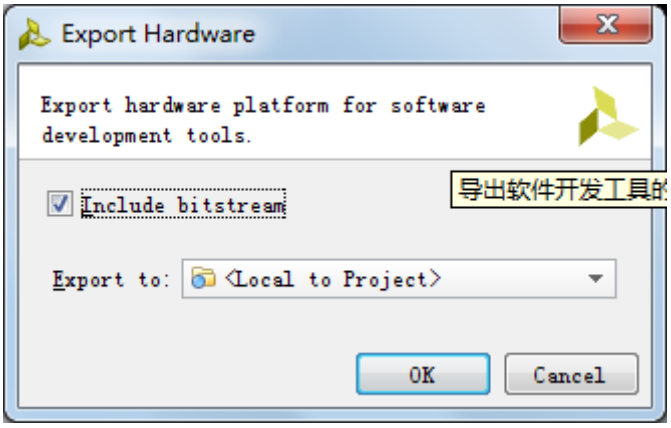

图 30、导出硬件设计

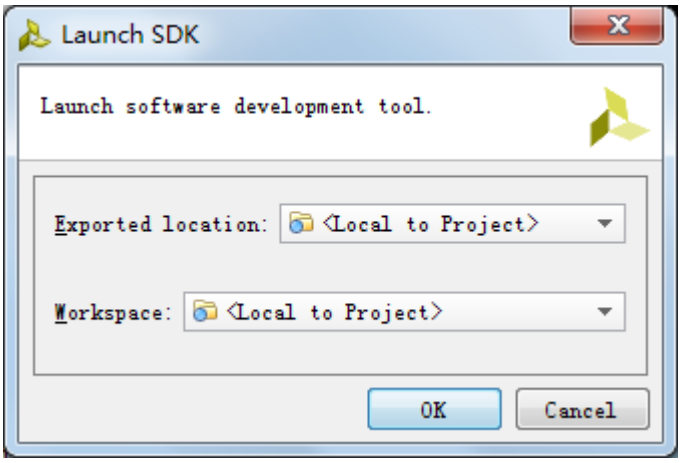

图 31 启动 sdk

将硬件导出(1、File→Export Hardware 2、File→Launch SDK，相当于 PlanAhead 和 XPS 的 Export Hardware & Launch SDK)。进入 SDK 欢迎界面后，SDK 会自动导入硬件信息，如图 32 所示。

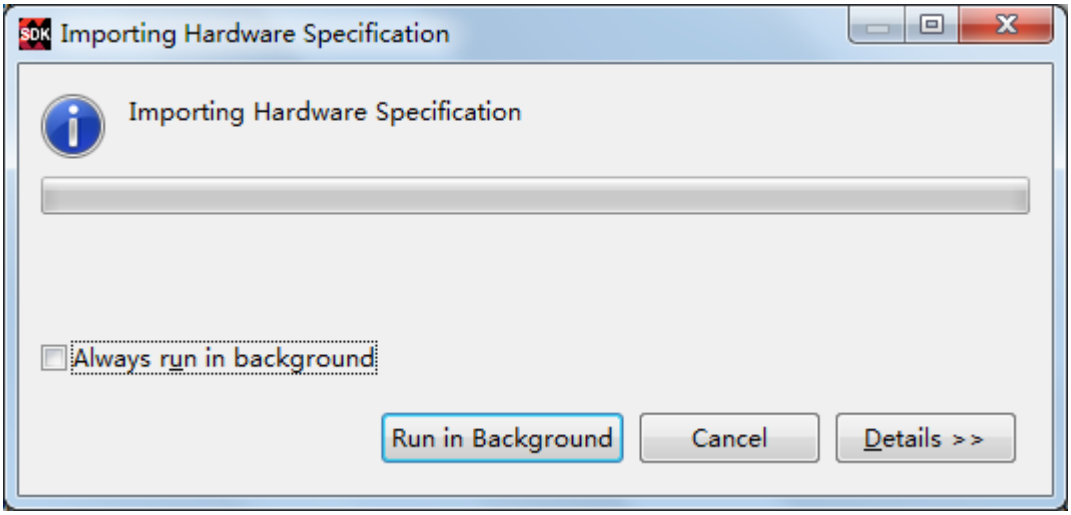

图 32

4、PS 部分实现

SDK 导入后，如下图所示。

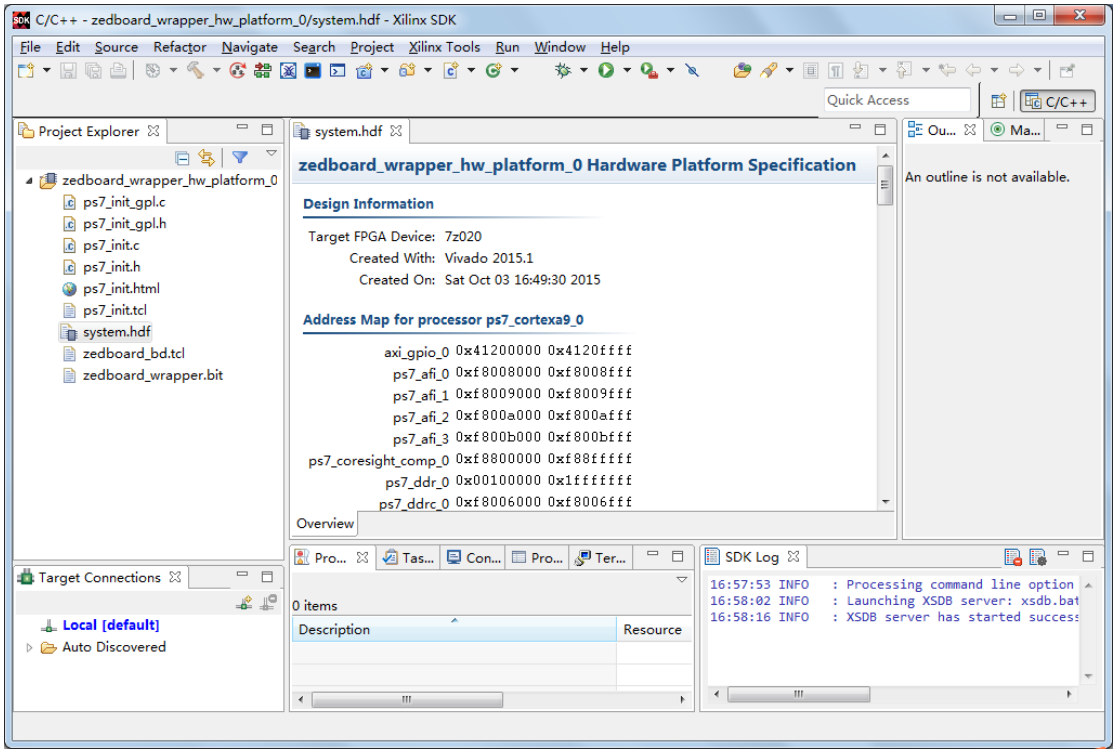

图 33

在此基础上，我们创建板级支持包 BSP 工程，点击 SDK 菜单下的 File->New->Board Support Package

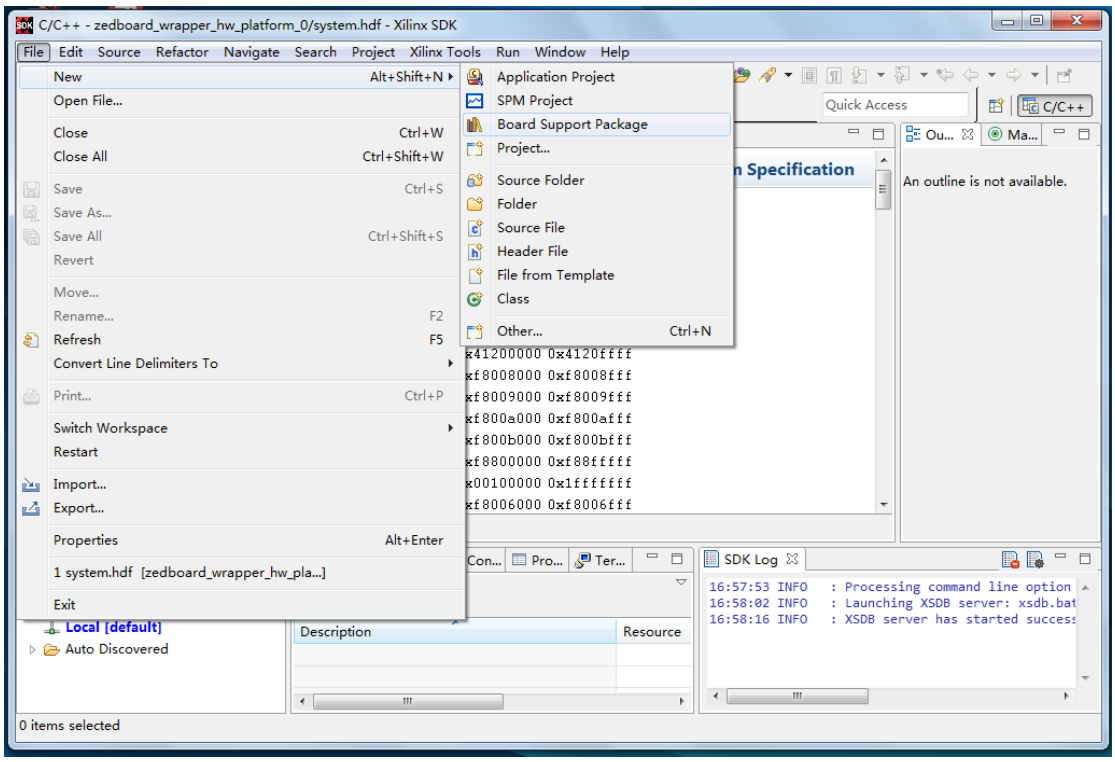

图 34

SDK 弹出如下配置对话框，我们把 Project name 重命名为 SmallRTOS_bsp，其它采用默认配置即可，直接点击 Finish 按钮。

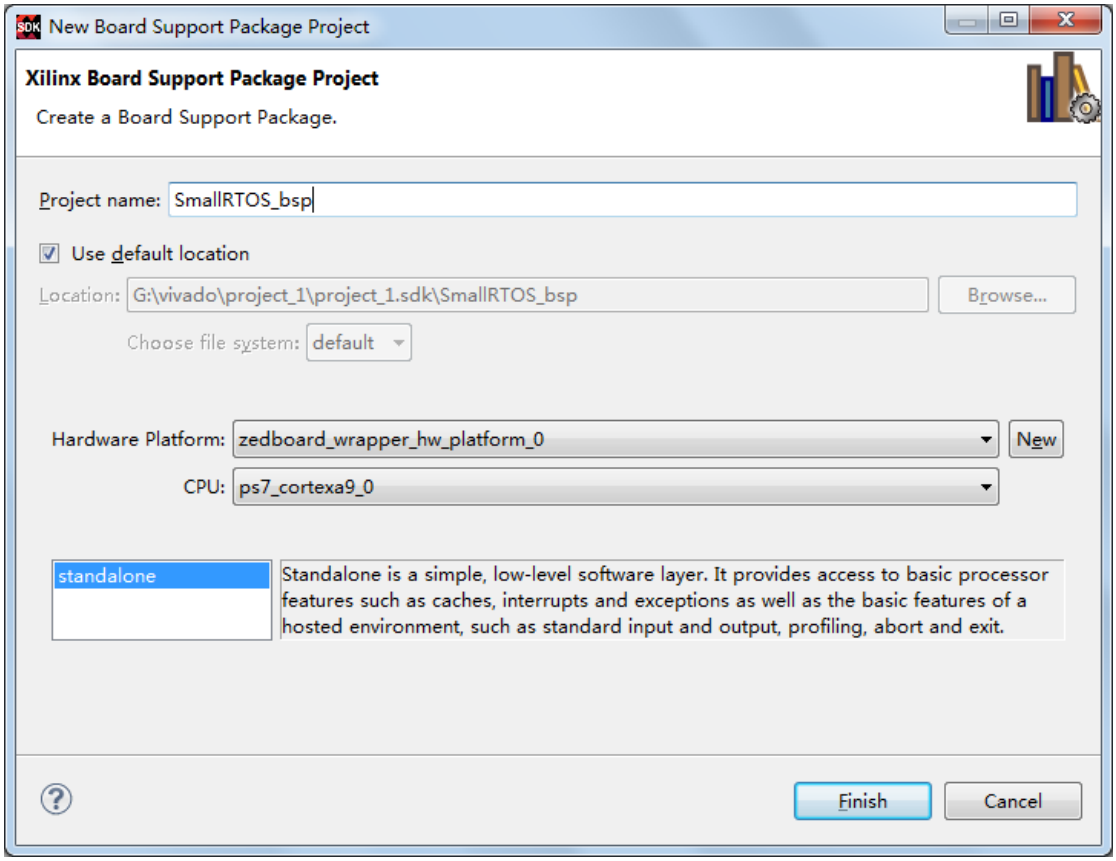

图 35

接着弹出如下对话框，选择根据需要选择板级支持包的功能组件，我们选择如下两个组件：xilffs，xilrsa 等

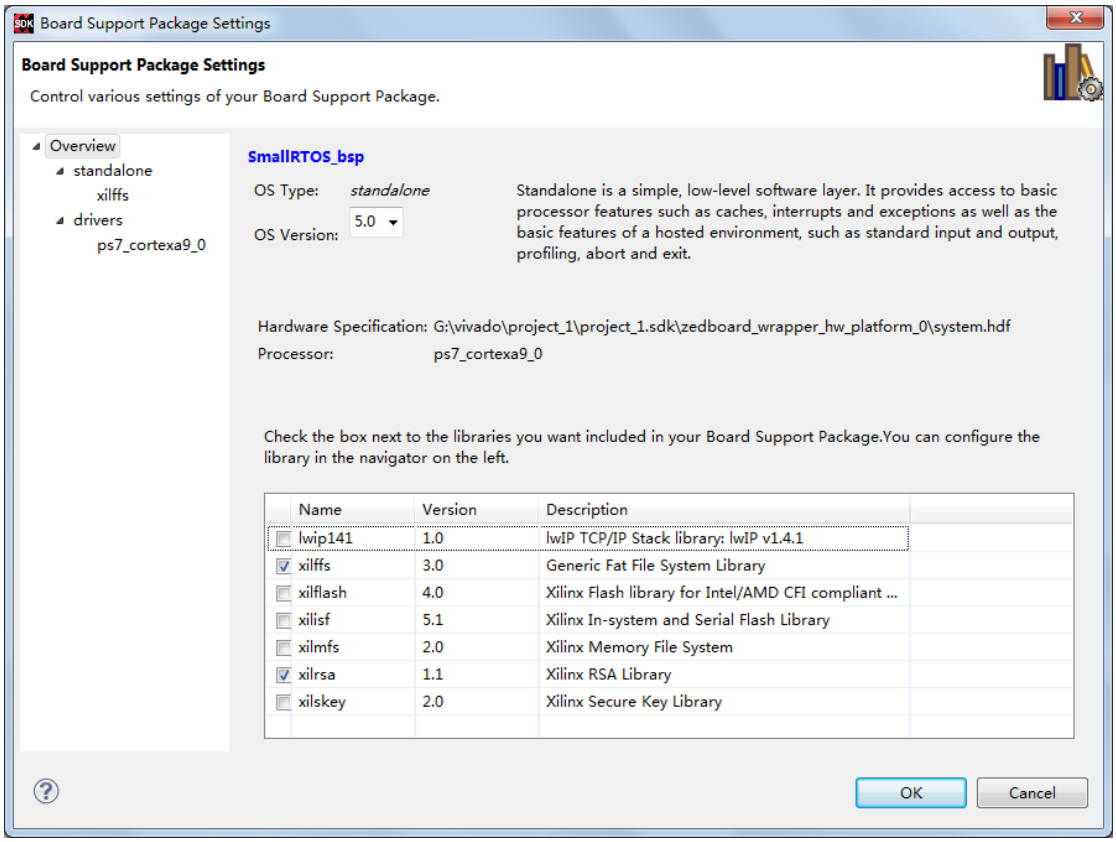

图 36

点击 OK 按钮后，弹出如下进度提示框，表示正在生产 BSP 板级支持包。

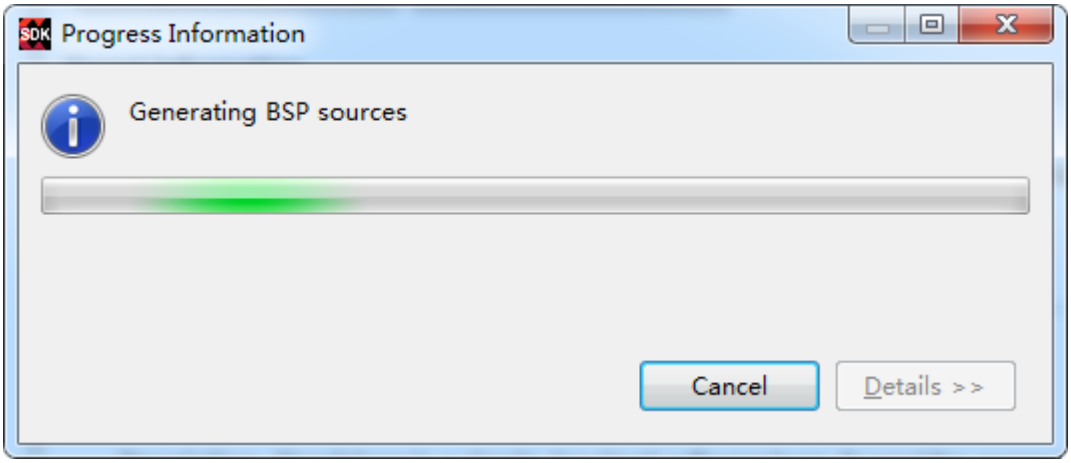

图 37

生成完毕后，在左侧栏会出现如下所示的两个工程：

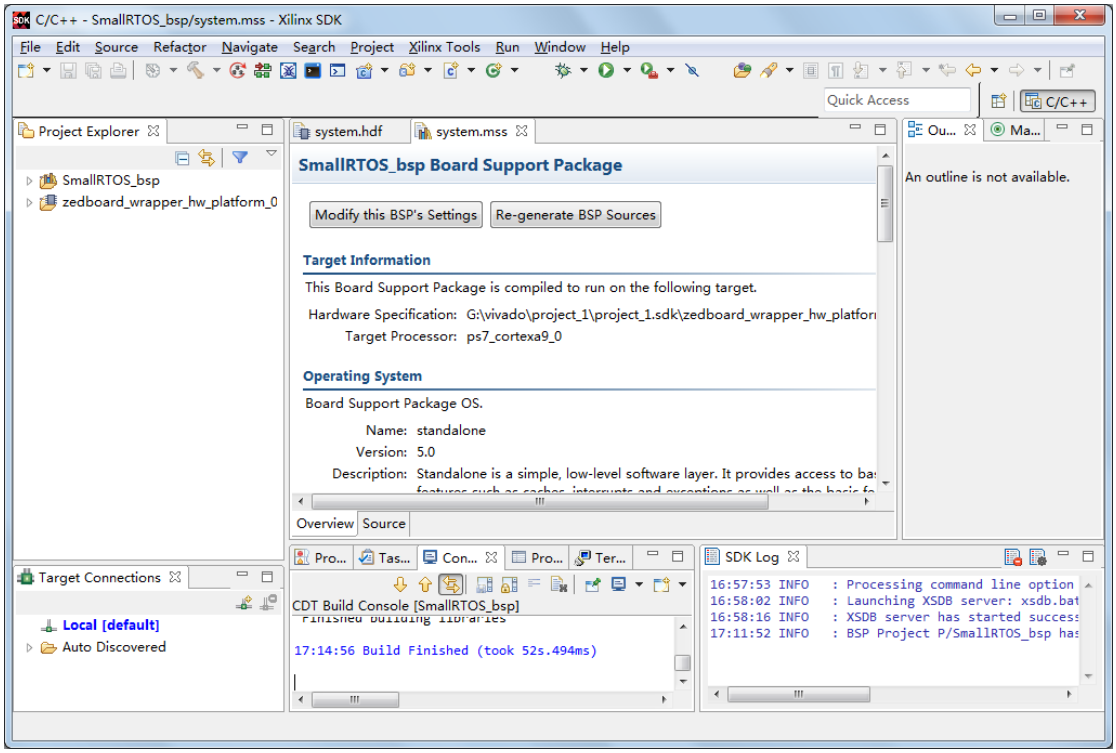

图 38

接着，就可以创建应用程序了。

接着，就可以创建应用程序了。
在 SDK 的菜单中点击 File->New->Application Project，如下图所示：

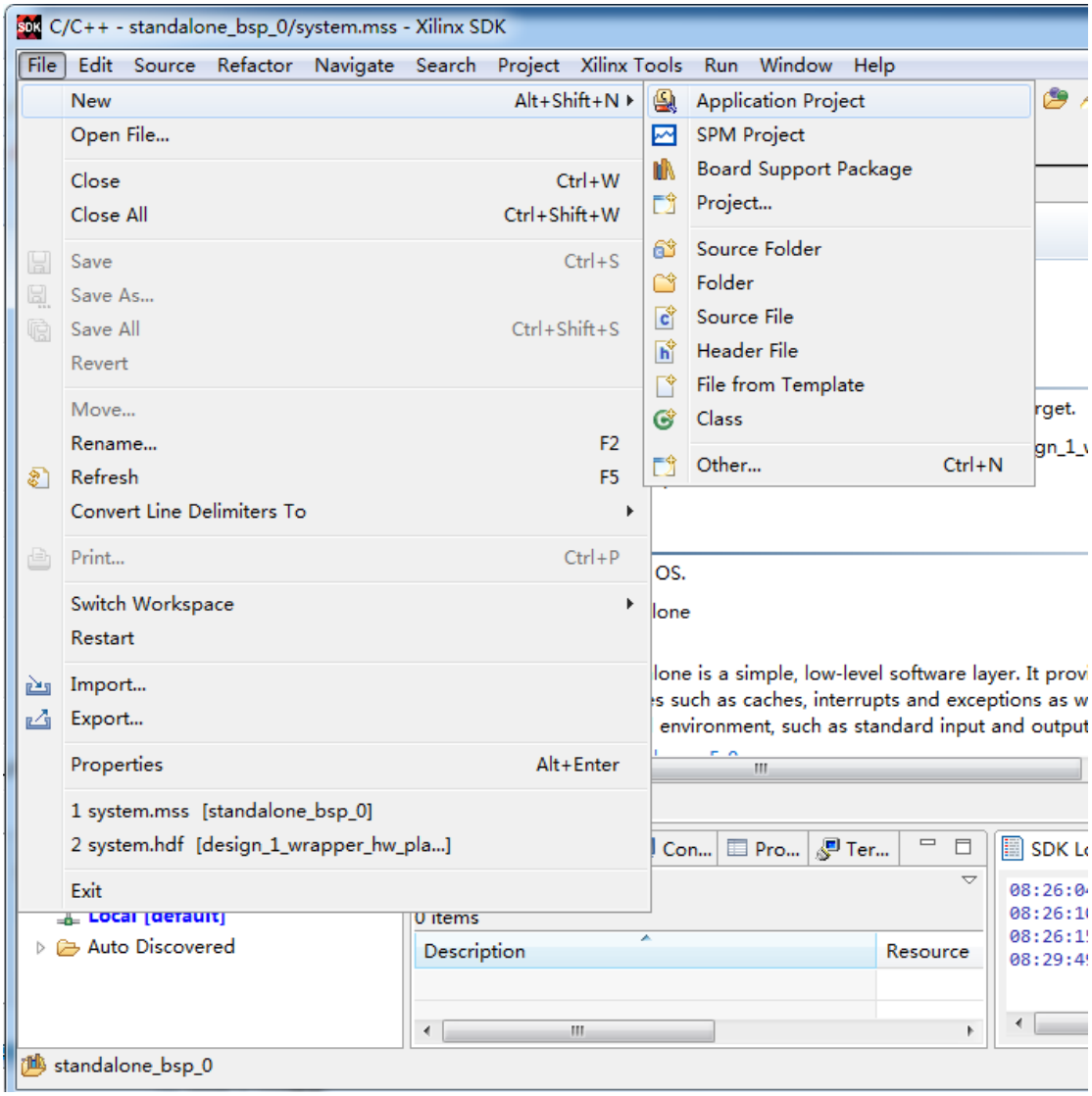

图 39

在 SDK 的弹出对话框中，填写应用程序的名字，比如 SmallRTOSDemo 等，板级支持包选择已经创建的工程 SmallRTOS_bsp。之后点击按钮 Next

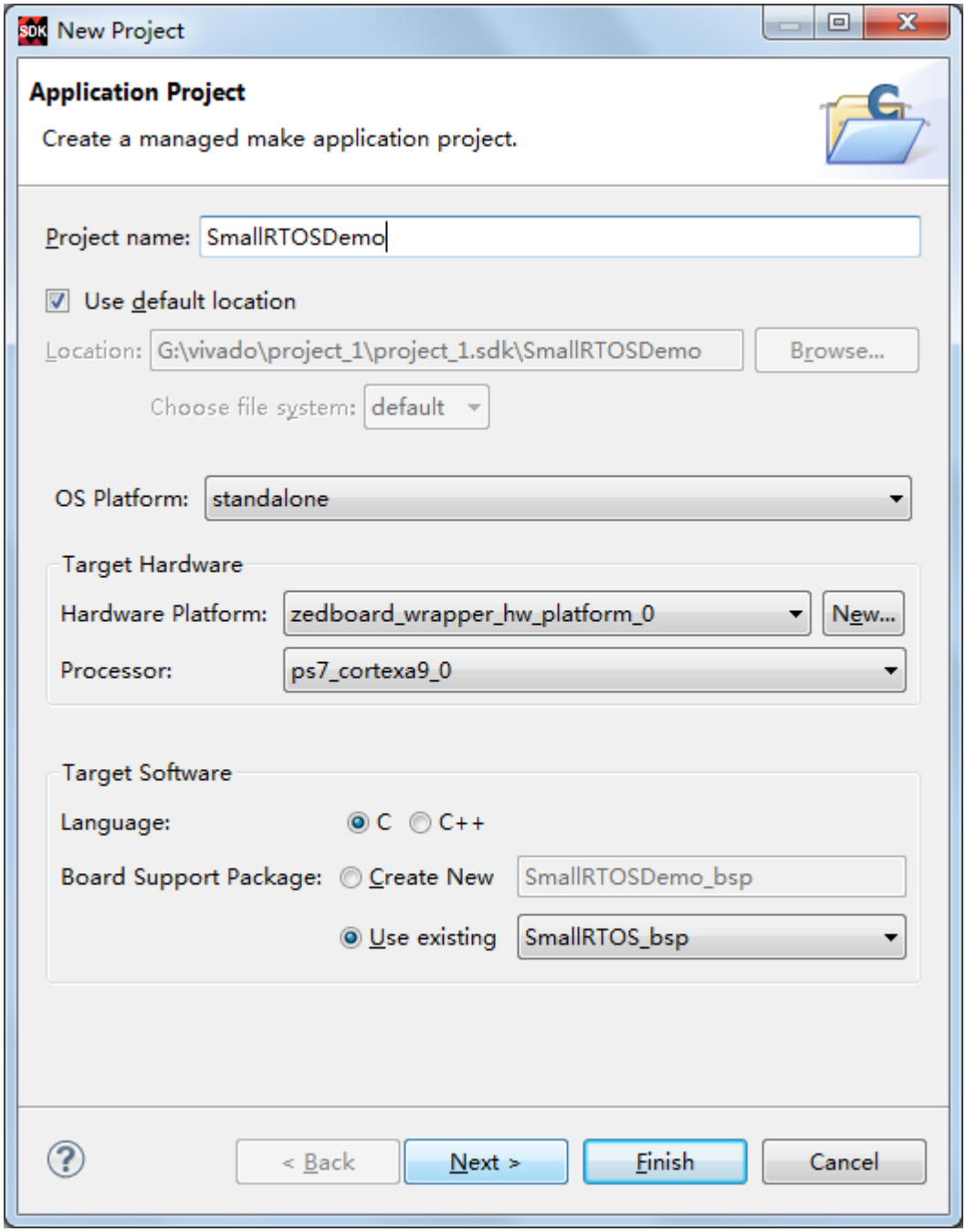

图 40

此时会出现示例工程选择对话框，如下所示，可以根据需要自行选择，我们选择 Empty Application

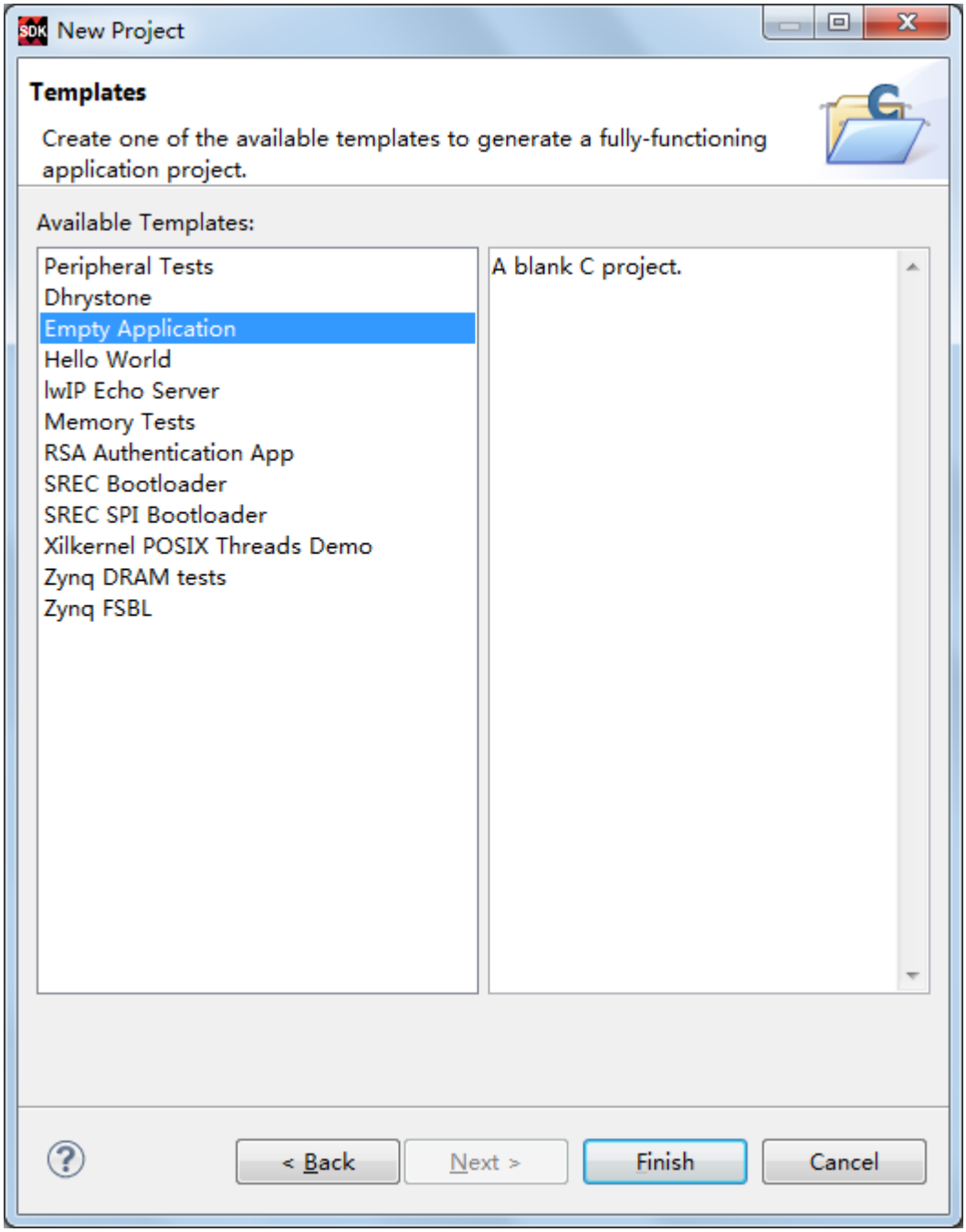

图 41

点击按钮 Finish 后，应用工程就创建完毕了，不过此时应用工程是空的，还无法使用。

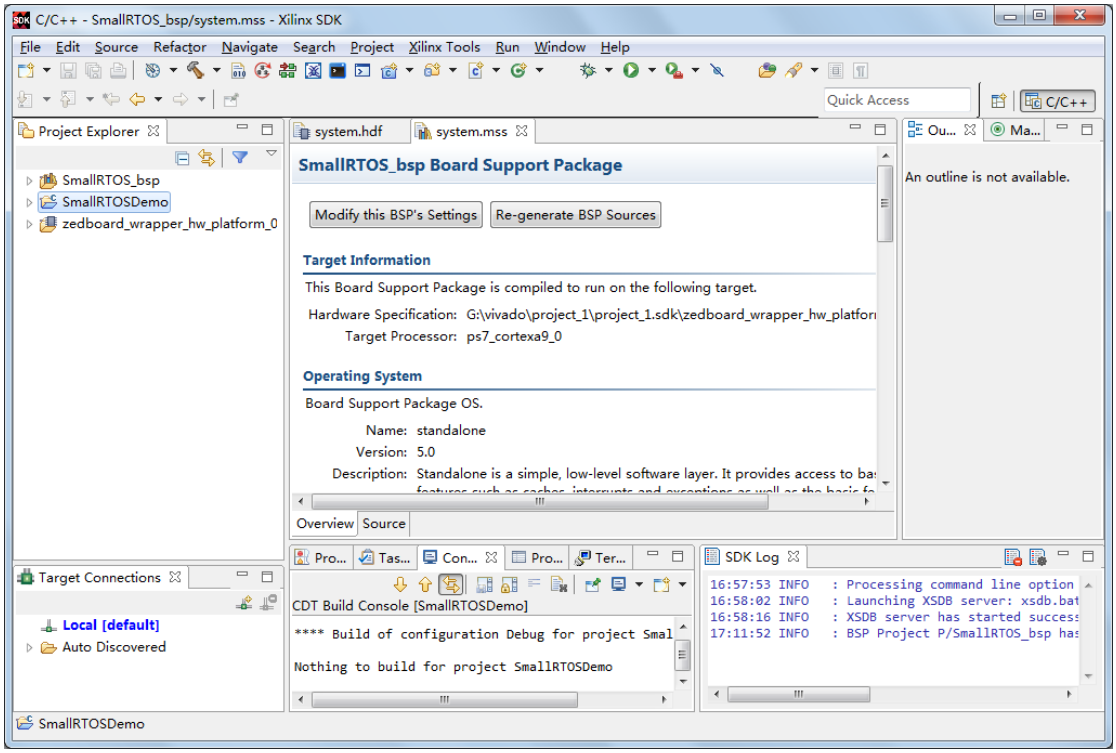

图 42

接下来我们向应用程序 SmallRTOSDemo 中添加代码文件，我们在这里偷个懒，直接采用现成的代码。

从“小嵌”操作系统官方网站下载其源代码，打开 SmallRTOS 压缩包，找到如下路径\Demo\CORTEX_A9_Zynq_Zedboard_SDK\SmallRTOSDemo\src，把该路径下的文件全部拷贝到刚刚创建的 SmallRTOSDemo\src 文件夹下，**注：把 lscript.ld 文件也拷贝过去**；

此时，右键点击工程 SmallRTOSDemo，在弹出菜单中点击 Refresh。

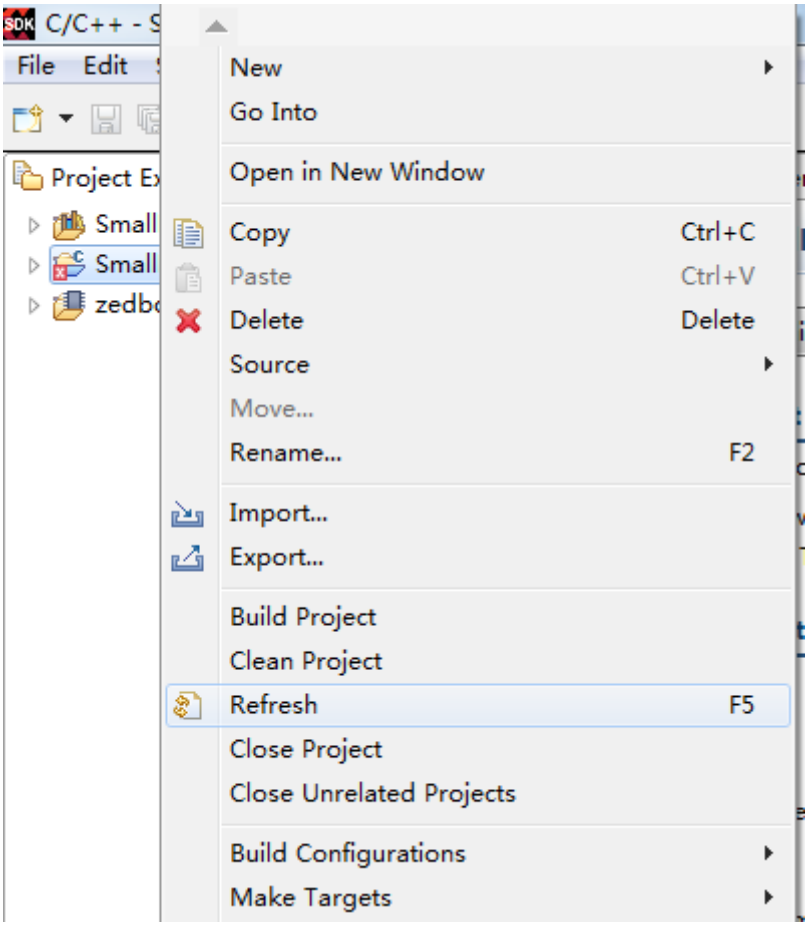

图 43

一般情况下 Refresh 完毕后，工程会自行编译，此时会提示报错信息，这是因为部分路径还没有指定，一些文件找不到。

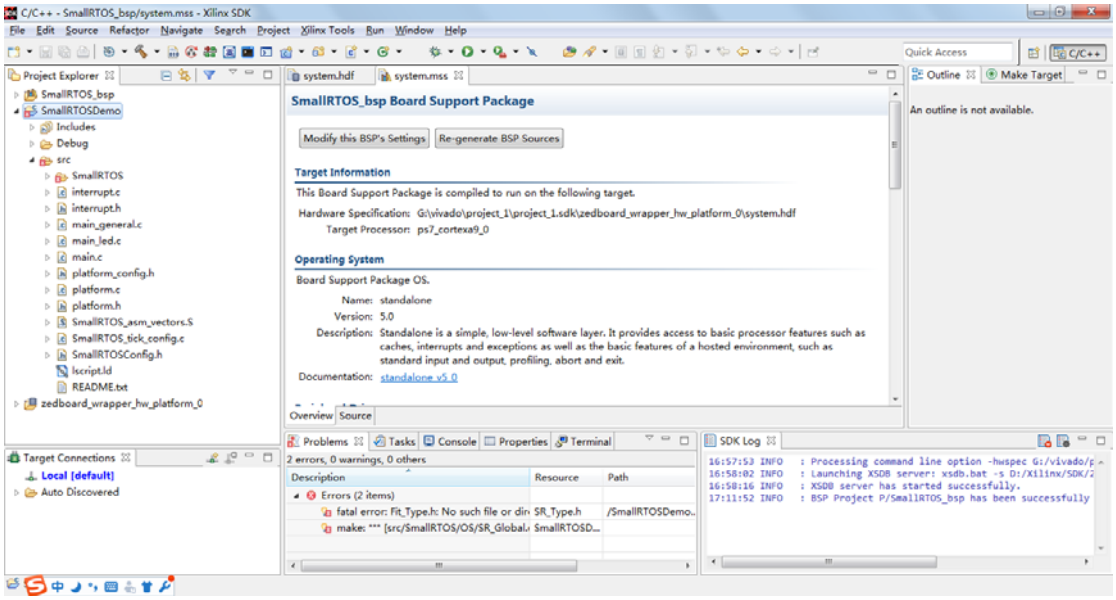

图 44

右键点击工程 SmallRTOSDemo，在弹出菜单栏中选择 Properties

此时 SDK 弹出编译环境配置对话框，选择 C/C++ General -> Paths and Symbols，打开 Include 小贴签项，点击 Add 按钮。

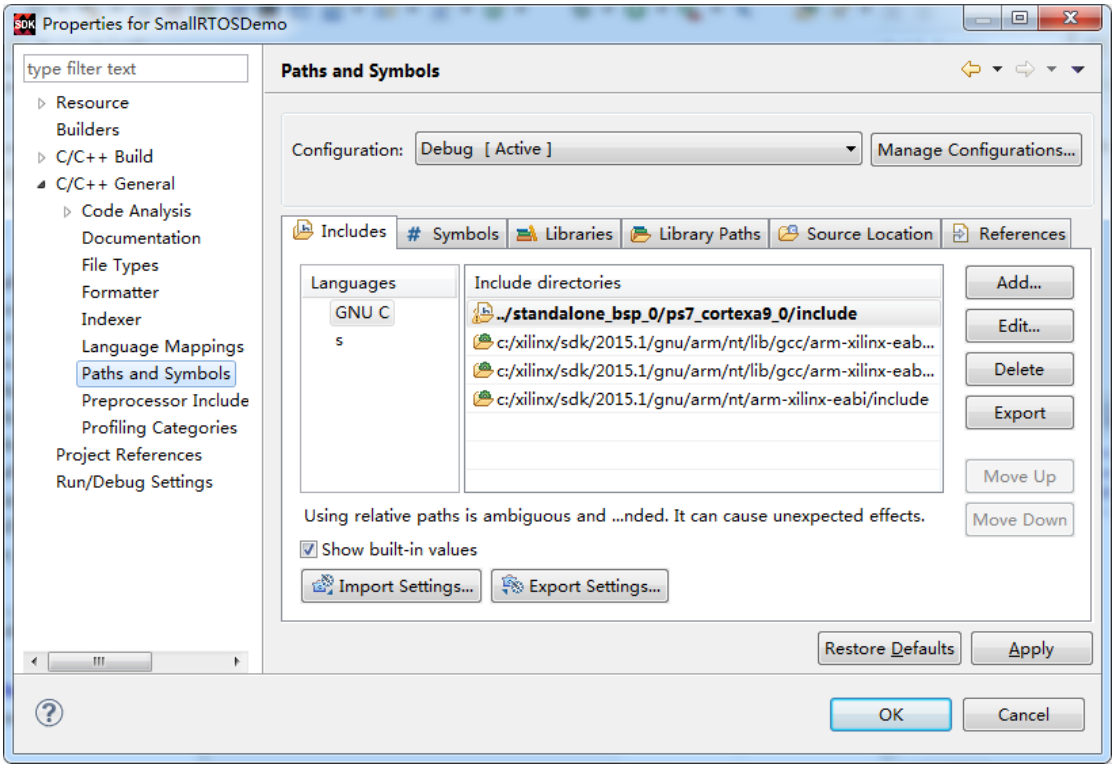

图 45

出现如下配置提示框，点击 Workspace 按钮

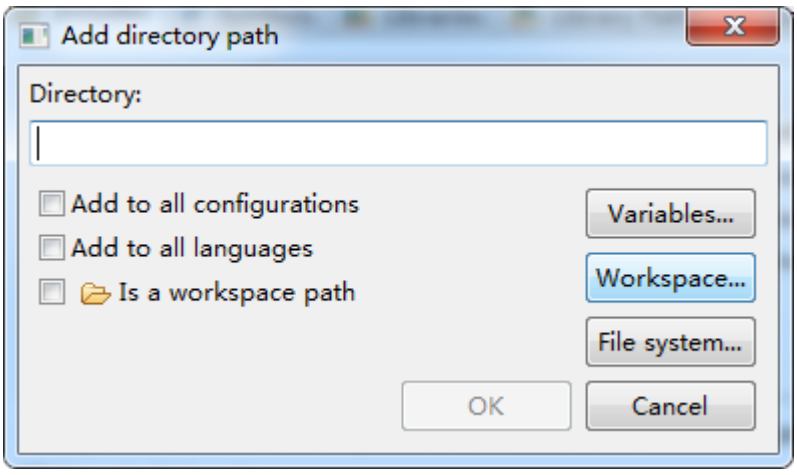

图 46

在弹出框中，选择工程 SmallRTOSDemo 下的 src 文件夹，点击 OK。

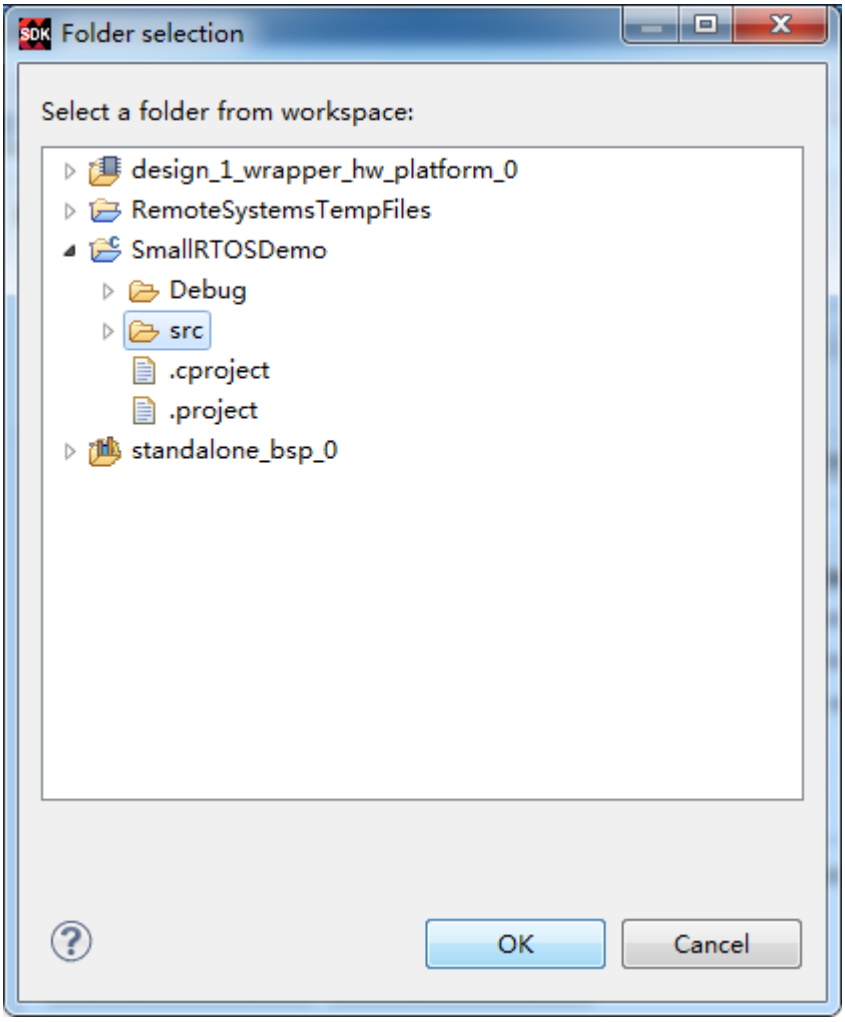

图 47

路径会显示到 Directory 的编辑框里，点击 OK 按钮即可。

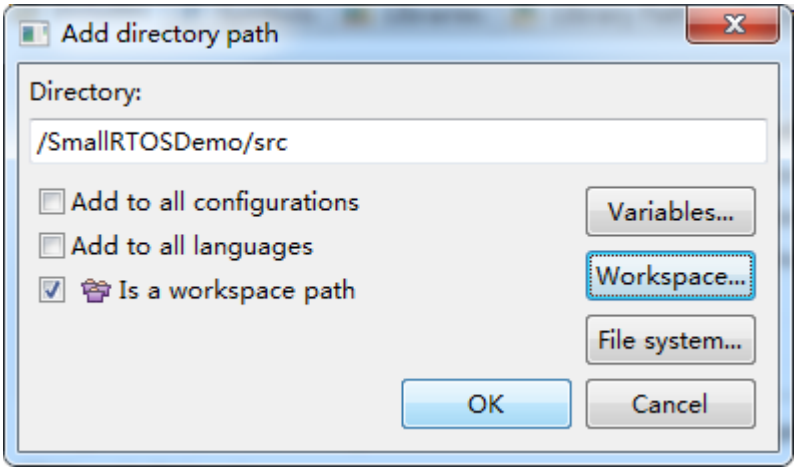

图 48

按照同样方式添加 SmallRTOS 的 OS 及 CPU 路径，如下图所示。

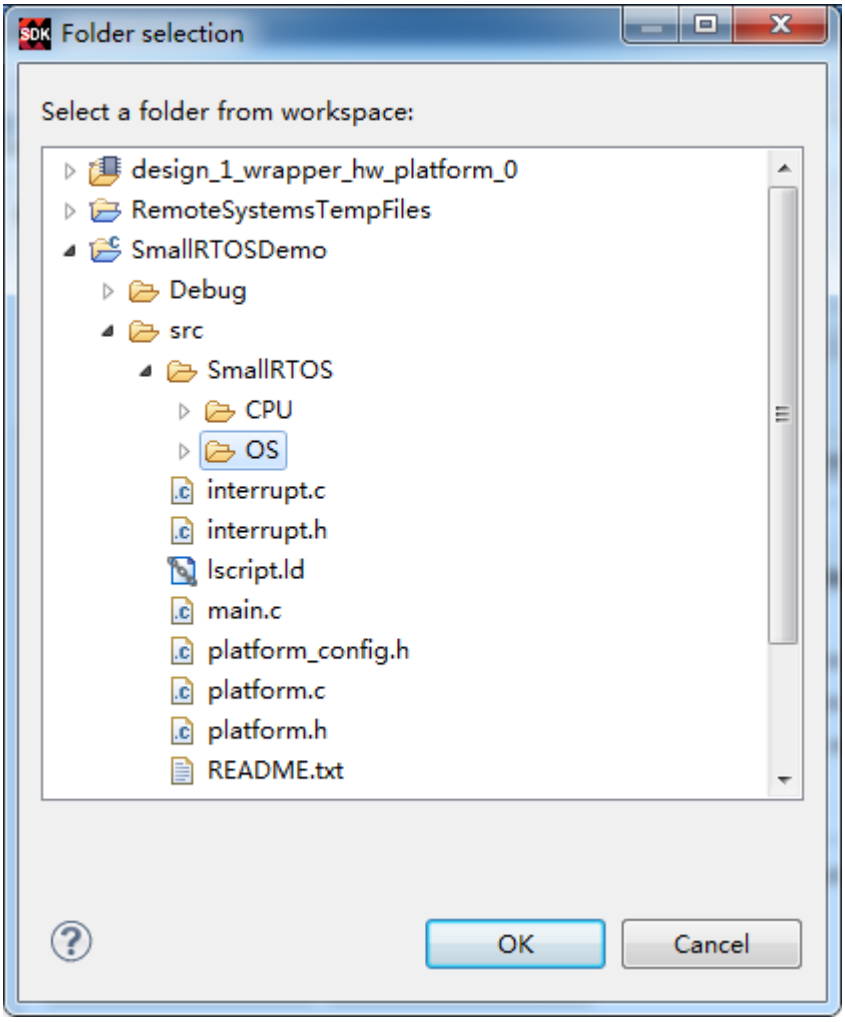
Folder selection
Select a folder from workspace:
design_1_wrapper_hw_platform_0
RemoteSystemsTempFiles
SmallRTOSDemo
Debug
src
SmallRTOS
CPU
OS
interrupt.c
interrupt.h
lscript.ld
main.c
platform_config.h
platform.c
platform.h
README.txt
OK
Cancel

图 49

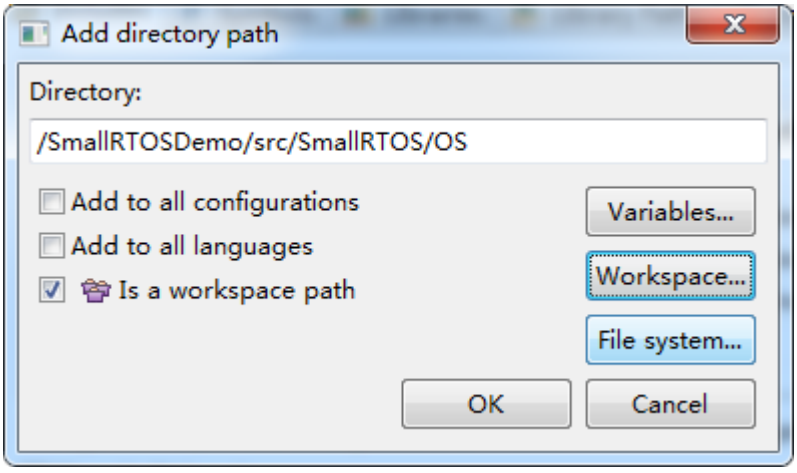
Add directory path
Directory:
/SmallRTOSDemo/src/SmallRTOS/OS
Add to all configurations
Add to all languages
Is a workspace path
Variables...
Workspace...
File system...
OK
Cancel

图 50

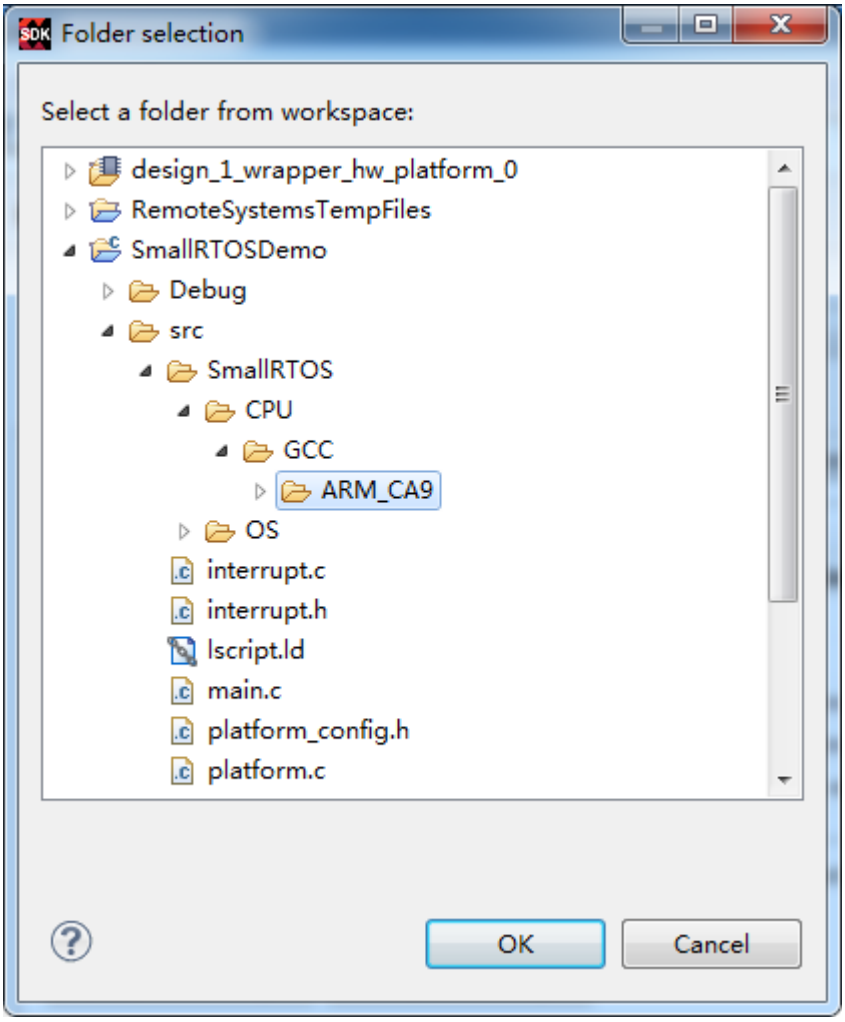

图 51

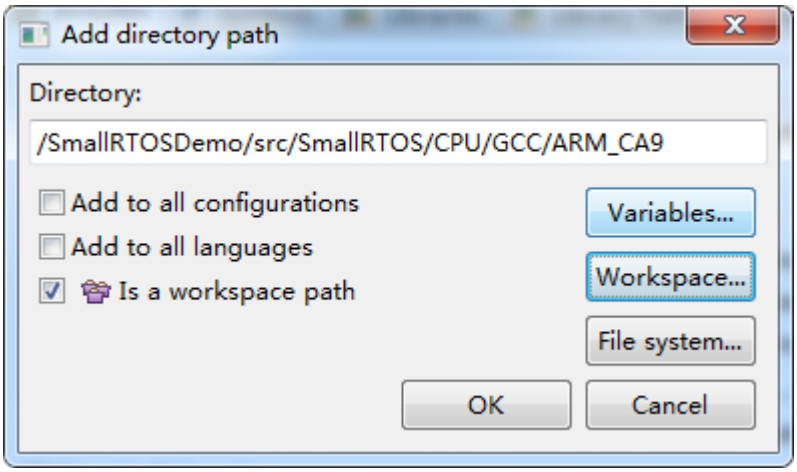

图 52

添加完毕后，路径会显示在列表中，如下图所示：

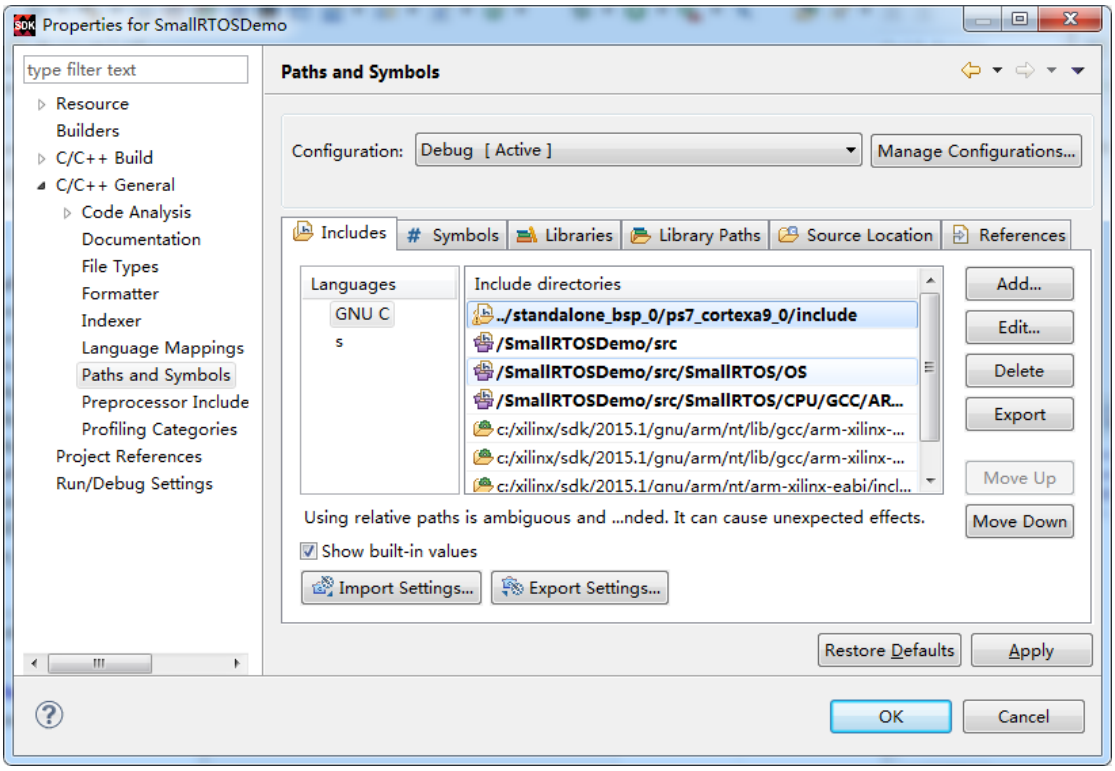

图 53

此时点击 Apply 按钮，使配置参数生效，此时弹出对话框如下所示，点击 Yes 即可。

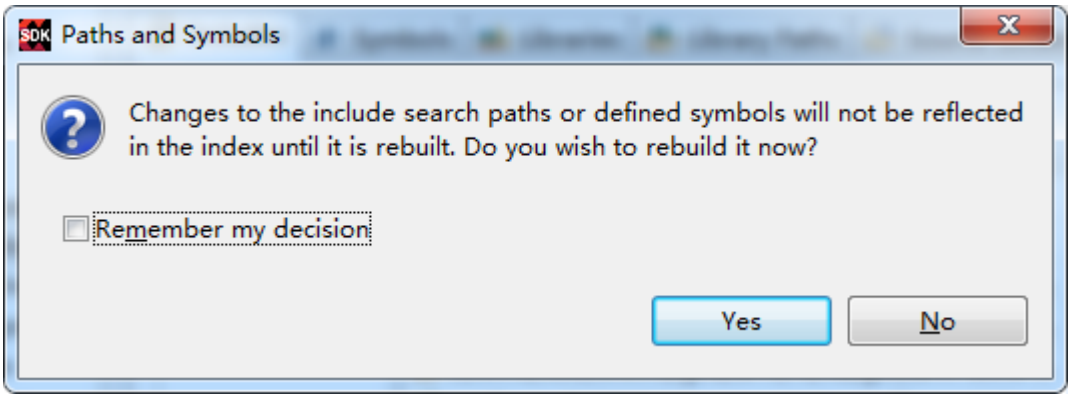

图 54

上述对话框退出后，工程在会重新编译（若无自动编译，请手动编译）。

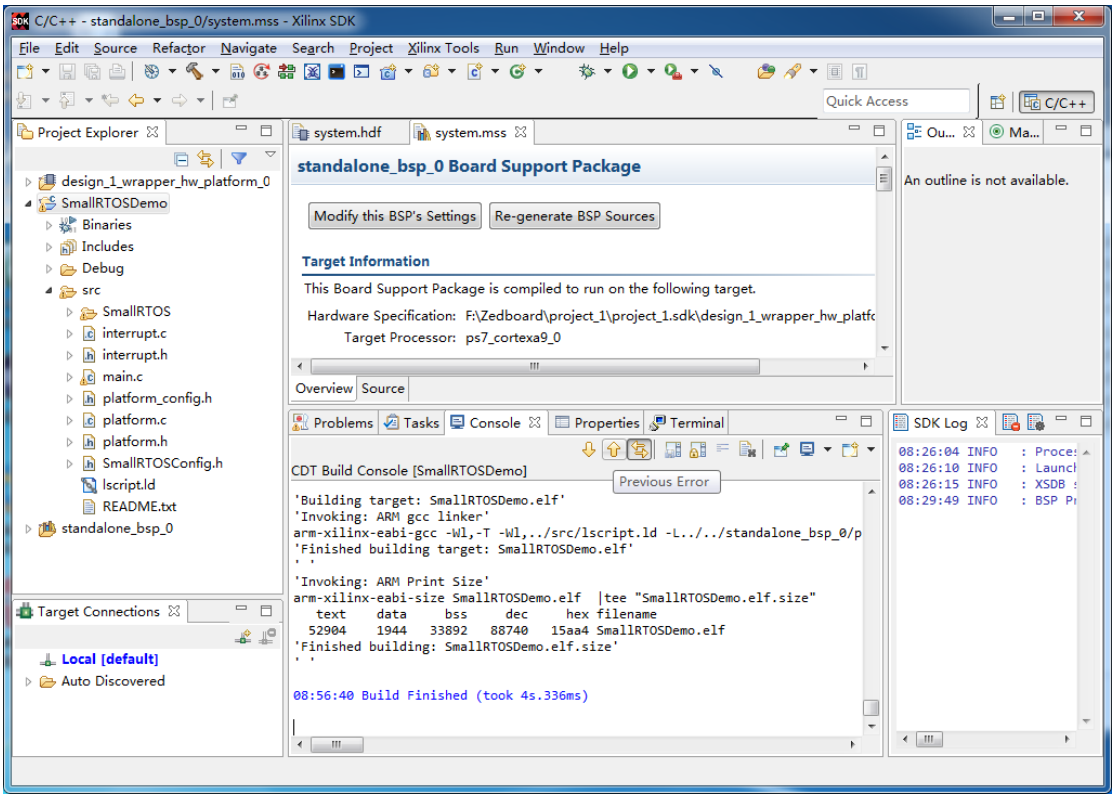

图 55

这次编译成功了，生成了 SmallRTOSDemo.elf 文件，出现如下编译后的结果：

```
arm-xilinx-eabi-size SmallRTOSDemo.elf  |tee “SmallRTOSDemo.elf.size”
   text    data     bss     dec     hex filename
  56188    1968  187220  245376   3be80 SmallRTOSDemo.elf
‘Finished building: SmallRTOSDemo.elf.size’
‘‘
17:37:46 Build Finished (took 14s.272ms)
```

到这一步，我们大部分的工作基本上已经完成了，不过默认的例程不是 LED 流水灯的控制，我们还需要进行适当的调整，来运行 LED 灯的示例工程。

找到工程中的 main.c 文件，在其中找到如下宏定义；
#define mainSELECTED_APPLICATION 0
把其更改为
#define mainSELECTED_APPLICATION 1

保存文件后，工程会重新编译，编译完毕后，即可下载到 Zedboard 上测试验证了。

注：在线下载测试时，需首先载入 bit 文件；在 SDK 中有专用的 bit 文件载入工具，如下图所示：

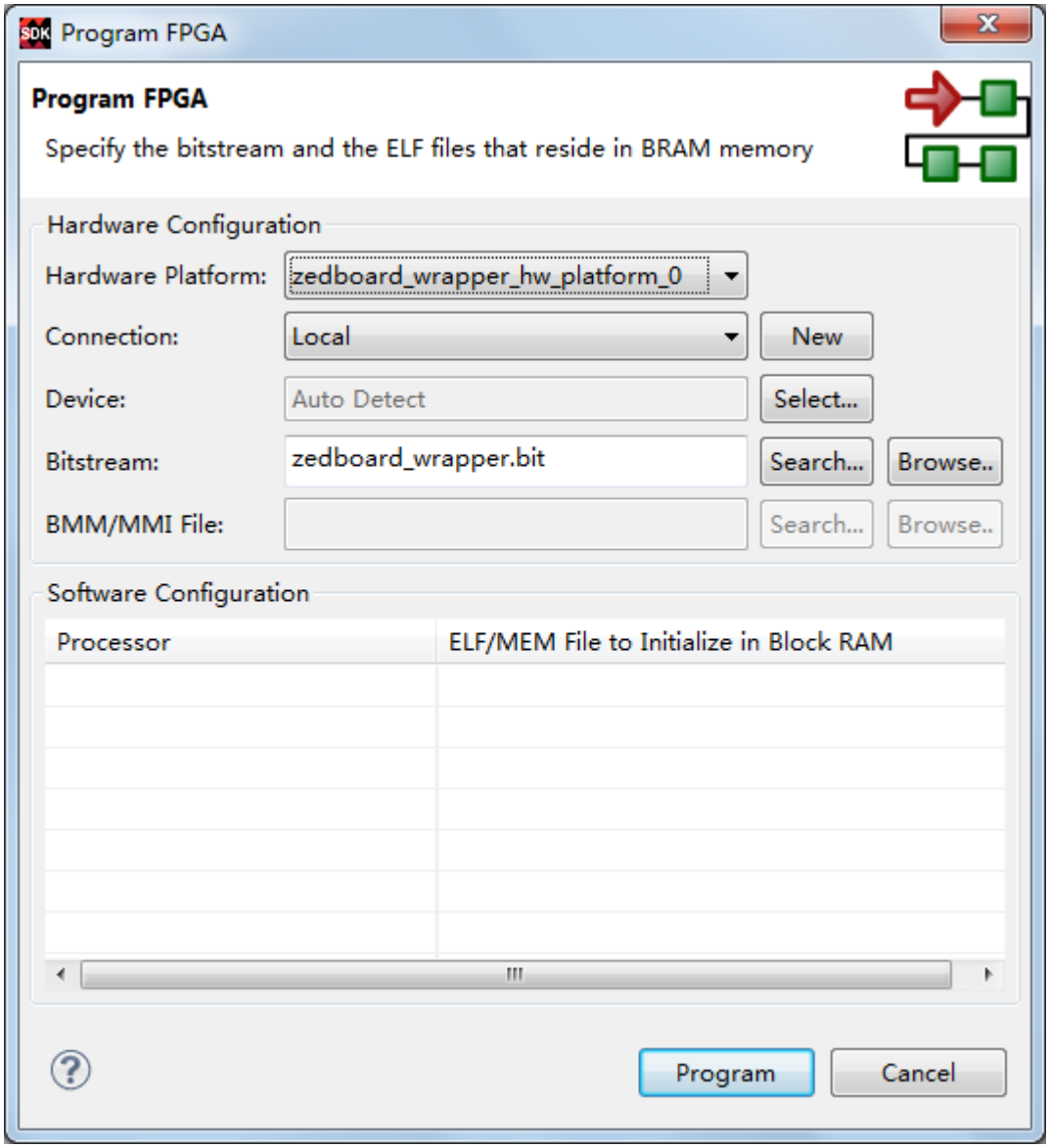

图 56

至此，全部工作已经完成，可以下载到板卡进行测试验证了。

下面把 LED 控制的关键代码贴出，供朋友们参考：

```
XGpio Gpio; /* The Instance of the GPIO Driver */

OSTaskHandle_t      CtrlTaskHandle = NULL;
OSTaskHandle_t      LedTaskHandle = NULL;
OSSemHandle_t       LedSemHandle = NULL;

voidTaskCtrl( void *pvParameters );
voidTaskLed( void *pvParameters );
```

```
staticintprvSetupGpio( void );


intmain_led()
{
    prvSetupGpio();

    LedSemHandle = OSSemCreate();

    // configure and start tasks
    CtrlTaskHandle = OSTaskCreate(TaskCtrl, NULL, OSLOWEAST_PRIORITY+1,
OSMINIMAL_STACK_SIZE, "Ctrl" );
    LedTaskHandle = OSTaskCreate(TaskLed, NULL, OSLOWEAST_PRIORITY+2,
OSMINIMAL_STACK_SIZE, "Led" );

    /* Start the tasks and timer running. */
    OSStart();

    for( ;; );
return 0;
}


staticintprvSetupGpio( void )
{
int Status;

    // Initialize the GPIO driver
    Status = XGpio_Initialize(&Gpio, XPAR_LEDS_ID);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    // Set the direction for all signals to be outputs
    XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x00);

    return XST_SUCCESS;
}

voidTaskCtrl( void *pvParameters )
{

    for( ;; )
    {
```

```
        xil_printf("Ctrl Task\r\n");
        OSSemPost(LedSemHandle);
        OSTaskSleep(500*OSTICKS_PER_MS);
    }
}

voidTaskLed( void *pvParameters )
{
    short i = 0;
    for( ;; )
    {
        xil_printf("Led Task\r\n");
        OSSemPend(LedSemHandle, OSPEND_FOREVER_VALUE);

        if(i>7)
        {
            i = 0;
        }
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, (1<<i));
        i += 1;
    }
}
```