

基于 Zynq7045 的 ZC706 平台开发指南

（注意：本指南适用于有 Xilinx SoC 平台一定开发基础的设计人员）

作者：Washington X.R

日期：2018 年 1 月 29 日星期一

版本：v1.0

邮件：washingtonxr@gmail.com

CONFIDENTIAL by x

目录

1	系统环境搭建	1
1.1	Linux 版本	1
1.1.1	环境依赖库	1
1.1.2	Petalinux 核心环境分布	1
1.2	Windows 版本	2
1.3	Modelsim 版本	2
2	开发软件	2
2.1	Petalinux 工具集	2
2.1.1	组件位置	2
2.1.2	配置要求	3
2.1.3	依赖包	3
2.1.4	安装 Petalinux	3
2.1.5	Petalinux 应用开发	4
2.1.6	打包 BSP 文件	19
2.1.7	配置固件版本	19
2.1.8	Qemu 调试	20
2.1.9	生成镜像	21
2.1.10	版本控制	24
2.1.11	GDB 调试	24
2.1.12	限制条款	25
2.1.13	FIT Image 镜像日志	28
2.2	Vivado	29
2.2.1	安装 Vivado	29
2.2.2	创建工程	29
2.2.3	创建最小系统	32
2.2.4	管脚分配和约束	34
2.2.5	ILA 仿真相关	35
2.2.6	仿真	36
2.2.7	导出 “*.hdf” 文件	38

2.2.8	设计 IP.....	39
2.2.9	Synthesis.....	41
2.2.10	Implementation.....	42
2.2.11	Project Summay.....	43
2.2.12	Program & Debug.....	43
3	参考文件.....	44

CONFIDENTIAL by X

本文主要介绍 Zynq7045 SoC 芯片平台的应用，通过 Xilinx 官方公布的 FPGA 开发集成环境（Vivado）、嵌入式开发软件平台（Petalinux）和开源 SDK（ZYNQ BSP）进行搭建和开发，其中的系统环境主要分布在 Ubuntu 的桌面发布版与 Windows 7 专业版之上。

1 系统环境搭建

1.1 Linux 版本

Linux tub 4.13.0-26-generic #29~16.04.2-Ubuntu SMP Tue Jan 9 22:00:44 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux。

如上版本可以用于安装 Petalinux、Vivado 和 Modelsim 系列软件。

表 1.1 Linux 系统分区

序号	分区	大小
1	/	20GB
2	/boot	200MB
3	/tmp	5GB
4	/Swap	2GB
5	/home	>30GB

1.1.1 环境依赖库

环境依赖库是根据 Linux 环境下需求的依赖库。

Ubuntu 系统通过管理员权限安装（sudo apt-get install）如下安装包：

```
sudo apt-get install gcc-multilib build-essential libsdl1.2-dev  
libglib2.0-dev screen pax gzip
```

1.1.2 Petalinux 核心环境分布

```
test@tub:~/Petalinux$ du -lh --max-depth=1  
643M  ./etc  
7.2G  ./tools  
7.2G  ./components  
15G   .
```

图 1.1.2-1 环境分布

```
declare -x PETALINUX="/home/test/Petalinux"  
declare -x PETALINUX_VER="2017.3"  
declare -x PWD="/home/test/Petalinux"
```

图 1.1.2-2 环境配置

```

~ .bashrc (/home/test) |115  . /etc/bash_completion
~ |116  fi
~ |117 fi
~ |118
~ |119 # Assign source of Petalinux.
~ |120 source ~/Petalinux/settings.sh
~ |121 █
~ |122
~ |123
~ |124
Tag_List_ 3,1 Bot .bashrc 121,0-1 Bot

```

图 1.1.2-3 系统环境变量

1.2 Windows 版本

Windows 7 Pro 64 bit

如上版本可以用于安装 Vivado 和 Modelsim 系列软件。

1.3 Modelsim 版本

Modelsim SE-64 10.1

2 开发软件

开发软件主要是 Xilinx 官方提供的 FPGA SoC 集成开发环境 Vivado、Petalinux 嵌入式开发平台工具集和开发板的板级支持包 BSP 等。

表 2 资源地址列表

资源内容	下载地址
通用资源	https://www.xilinx.com/support/download.html
Vivado	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html
Petalinux	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html
BSP	http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/petalinux.html

2.1 Petalinux 工具集

Petalinux 是嵌入式 Linux 开发工具集，可以用于 Xilinx 的 FPGA-based 系列的 SoC 嵌入式软件的开发。

2.1.1 组件位置

表 2.1 开发包 SDK 组件

组建位置	芯片架构
\$PETALINUX/components/yocto/source/aarch64	Zynq® UltraScale+™ MPSoC

\$PETALINUX/components/yocto/source/arm	Zynq® SoC
---	-----------

2.1.2 配置要求

最低工作站配置要求，在应用环境推荐：

- 1) 8GB 双通道外存资源，推荐 16GB；
- 2) 2GHz CPU 多核架构，推荐 12 核；
- 3) 100GB 的硬盘空间，推荐固态硬盘；
- 4) 系统支持(64 位系统)，推荐 Ubuntu 发行版：
 - a. RHEL 7.2/7.3；
 - b. CentOS 7.2/7.3；
 - c. Ubuntu 16.04.1；
- 5) 需要系统 root 权限，后期获取 root 用户权限；
- 6) 需要安装相关 Linux 系统和 Petalinux 支持的相关依赖包。

2.1.3 依赖包

Ubuntu 系统通过管理员权限安装（sudo apt-get install）如下安装包：

```
sudo apt-get install python3 tofrodos iproute2 gawk xvfb gcc git make net-
tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1
gnupg wget diffstat chrpath socat xterm autoconf libtool tar unzip texinfo
zlib1g-dev gcc-multilib build-essential libsdl1.2-dev libglib2.0-dev screen
pax gzip zlib1g:i386
```

2.1.4 安装Petalinux

将下载的 Petalinux 安装包通过共享硬盘等方法挂载到 Linux 之上安装，无需直接考入到 Linux 系统。安装步骤如下：

- 1) \$ mkdir -p /home/test/petalinux
- 2) \$./petalinux-v2017.3-final-installer.run /home/test/petalinux
- 3) \$ chmod 755 -R petalinux
- 4) Reading and agreeing to the PetaLinux End User License Agreement (EULA)
is a required
- 5) 继续安装 PetaLinux Tools installation process
- 6) 应用 Petalinux 之前需要在在启动脚本中添加（如图 1.1.2-3）系统环境变量
- 7) \$ petalinux-util --webtalk off

Petalinux Tools 将被安装到 “/home/test/petalinux” 目录之下, 并给予整个文件夹 “755” 继承权限。

2.1.5 Petalinux 应用开发

2.1.5.1 配置环境变量

- 1) 配置系统环境变量

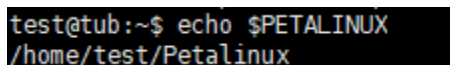
```
$ source <path-to-installed-PetaLinux>/settings.sh
```

- 2) 确认环境变量

```
$ echo $PETALINUX
```

```
/opt/pkg/petalinux
```

“\$PETALINUX”应该要指向的是 Petalinux 安装包所在的位置, 例如:



```
test@tub:~$ echo $PETALINUX
/home/test/Petalinux
```

图 2.1.5.1 Petalinux 位置

2.1.5.2 Petalinux BSP 安装

下载对应开发板或平台的官方 BSP 文件, (表 2 资源地址列表) 中下载。
安装 BSP 步骤:

- 1) \$ cd /home/test/Documents/Petalinux_prj
- 2) \$ petalinux-create -t project -s <BSP 文件位置>
- 3) 配置成果 BSP 获得如下提示:

```
INFO: Create project:
INFO: Projects:
INFO: * xilinx-zc706-v2017.1-final.bsp
INFO: has been successfully installed to /home/test/Documents/Petalinux_prj
INFO: New project successfully created in
/home/test/Documents/Petalinux_prj
```

创建一个 zc706 平台的基础 Petalinux 工程

创建一个 zc706 基础工程需要完成如下几个步骤:

- 1) 建立工程 (基于模板的建立);
- 2) 导入从 Vivado 导出的硬件描述文件 (*.hdf), 需要匹配版本。

调试条件:

- 1) TTC 模块 (必须) #如果多个 TTC 都使用了的话, Linux
内核将会使用第一个 TTC 模块;

- 2) 外部 Flash 存储空间或 SD 卡（必须） #用于防止 BOOT.BIN 文件和 image.ub 文件；
- 3) UART 模块（可选，控制台打印信息用） #如果用 UART IP 模块的话，如 AXI UART，需要确保中断信号有连接到 PS；
- 4) 非易失存储器（可选） #如 QSPI Flash，SD/MMC；
- 5) 以太网接口（可选） #若果用 Ethernet IP 模块或外部 PHY 的话，需要确保中断信号连接道 PS；
- 6) 交换机（可选） #可用于和 host 主机之间应用 TFTP 等模式启动调试方法。

2.1.5.2.1 建立工程

命令行如下（可以在非 root 用户下执行）：

```
$ petalinux-create --type project --template <CPU_TYPE> --name
<PROJECT_NAME>
```

petalinux-create 的参数说明：

- 1) --type: 生成类型，project 为工程，app 为应用，module 为模块；
- 2) --template: 应用的模板类型，可以是 zynq 等。若选择的是相应的 BSP 文件，则应该通过“petalinux-config”选择相对应的模板“board configs”，可能的值包括：The possible values are: ac701-full, kc705-lite, zc1751-dc1, zc706, zcu102-revb, zedboard, ac701-lite, kcu105, zc1751-dc2, zcu102-rev1.0, kc705-full, zc702, zcu102-reva, and zcu106-reva.;
- 3) --name: 工程名。

例如：\$ petalinux-create --type project --template zynq --name zc706_dut

```
test@tub:~/Documents/Petalinux_prj$ petalinux-create --type project --template zynq --name zc706_dut
INFO: Create project: zc706_dut
INFO: New project successfully created in /home/test/Documents/Petalinux_prj/zc706_dut
```

图 2.1.5.2.1-1 建立 template 工程

得到项目工程创建成功的指令答复之后，在相应文件交中应该得到创建的目的文件夹和相应的内容，如下图所示：

```
test@tub:~/Documents/Petalinux_prj/zc706_dut$ ls
config.project  project-spec
```

图 2.1.5.2.1-2 已经创立的 zc706_dut 工程

2.1.5.2.2 添加硬件描述

硬件描述文件位置：输出的文件都在“/<vivado project>/<project name>.sdk/<top module name>_hw_platform_0”文件夹下的“*hdf”文件；添加硬件文件，步骤如下：

- 1) hdf 文件位置：\$ petalinux-config --get-hw-description=<path-to-directory-which-contains-hardware-description-file>。例如：
~/Documents/Petalinux_prj/HW/zc706_ptp/system.hdf;
- 2) 导入 hdf 文件到工程：\$ petalinux-config --get-hw-description =
~/Documents/Petalinux_prj/HW/zc706_ptp
- 3) 得到导入答复：INFO: Getting hardware description...
- 4) [INFO] generating Kconfig for project
- 5) 自动进入系统配置目录：
/home/test/Documents/Petalinux_prj/zc706_dut/project-spec/configs/config - misc/config Sy;
- 6) 配置“misc/config System Configuration”

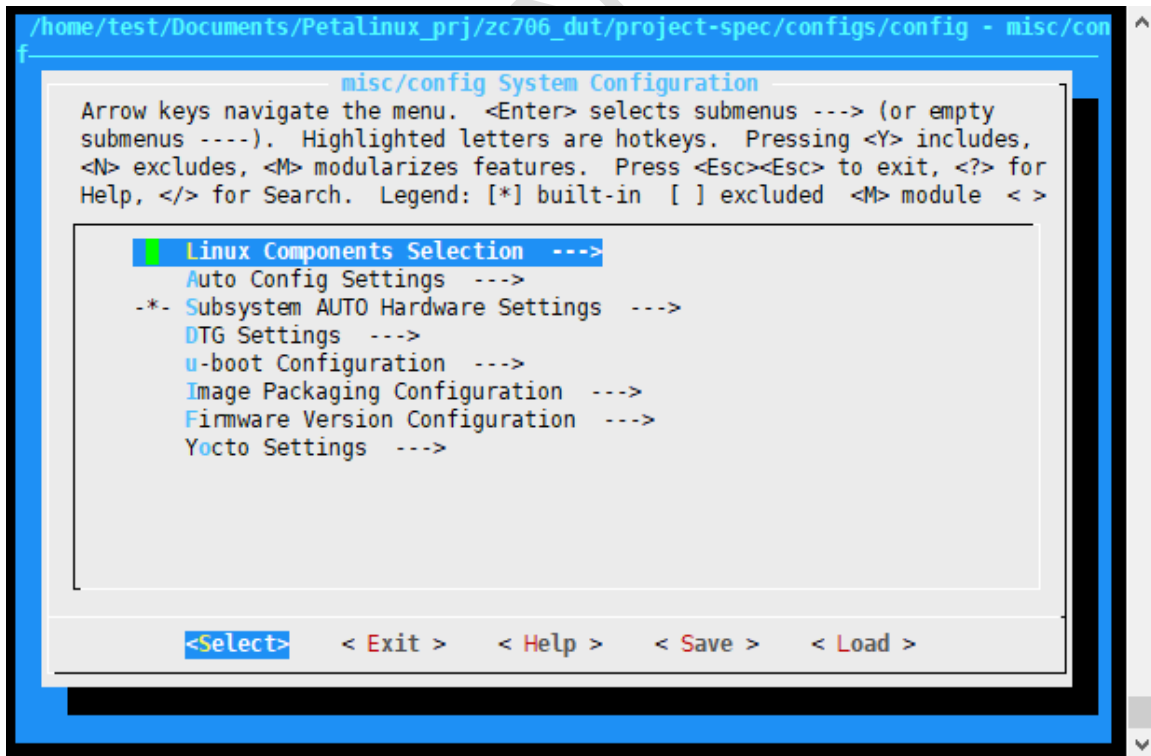


图 2.1.5.3.2 系统配置菜单

- 7) 配置完成退出并保存 “.config” 文件到工程中。需要配置的项目包括：Linux 组件、硬件启动、u-boot 配置、网络配置和输出镜像配置等。

2.1.5.2.3 系统配置

Linux 和 u-boot 配置主要包括：FSBL 启动、ps7_init、u-boot 源文件和 Linux kernel 源文件位置的配置，具体如下图所示：

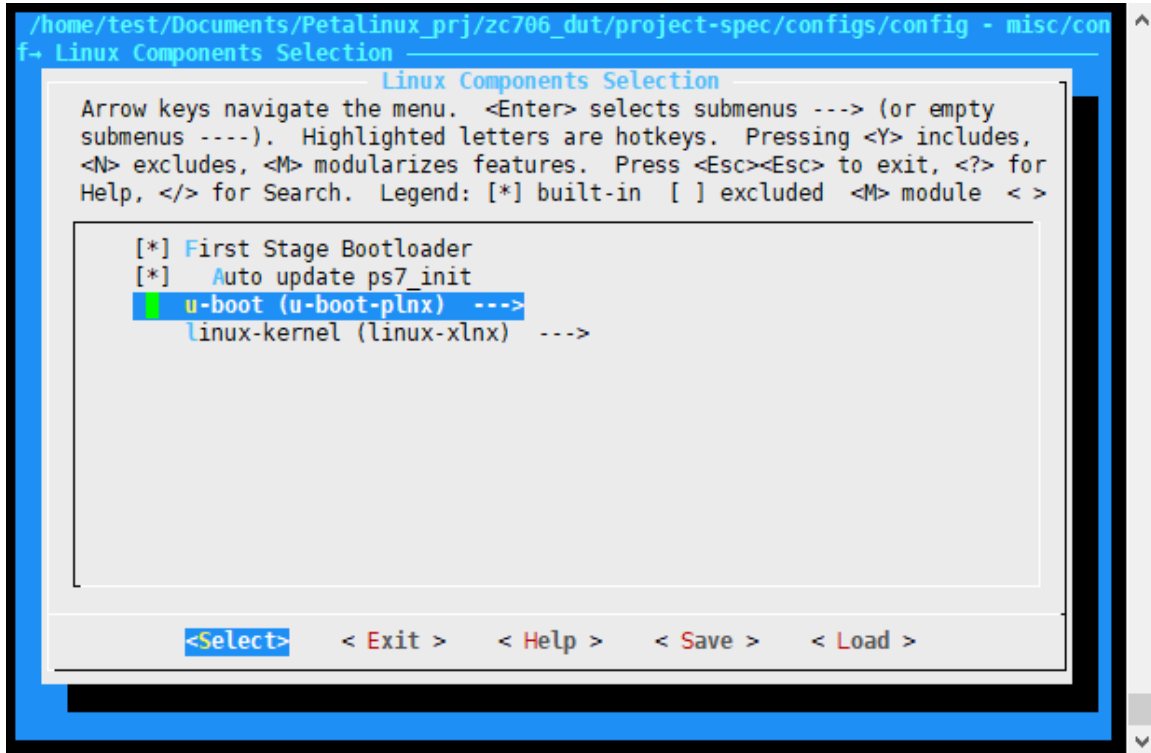


图 2.1.5.2.3-1 Linux 和 u-boot 位置

Zynq 可以选择通过 SDK 方式生成 FSBL (First Stage Bootloader) 和添加 AUP7 (Auto update ps7_init) 文件，而在 Petalinux 的工程环境中也可以用于自动生成配置，最终生成的文件 (fsbl) 等，将自动添加到 BOOT.BIN 文件中。

其中，u-boot 和 linux-kernel 可选项包括：

- 1) u-boot[Linux kernel]-plnx: xilinx 发布 u-boot-plnx 或 linux-xlnx 版本；
- 2) remote: 远程版本，可以是 xilixi 在官方 git 发布的 tag 版本或者公司私有版本，如果是应用远程版本，需要在 linux kernel 的上一级的配置 “Remote linux-kernel settings --->” 或 u-boot 的上一级配置 “Remote u-boot settings --->” 目录中配置相应的远程源文件的相应地址；
- 3) ext-local-src: 本地版本，用本地的镜像文件，用于自定义版本加入。需要在 linux kernel 的上一级的配置 “External linux-kernel local source settings

--->”或 u-boot 的上一级配置 “External u-boot local source settings --->” 目录中配置相应的本地源文件的相应地址。

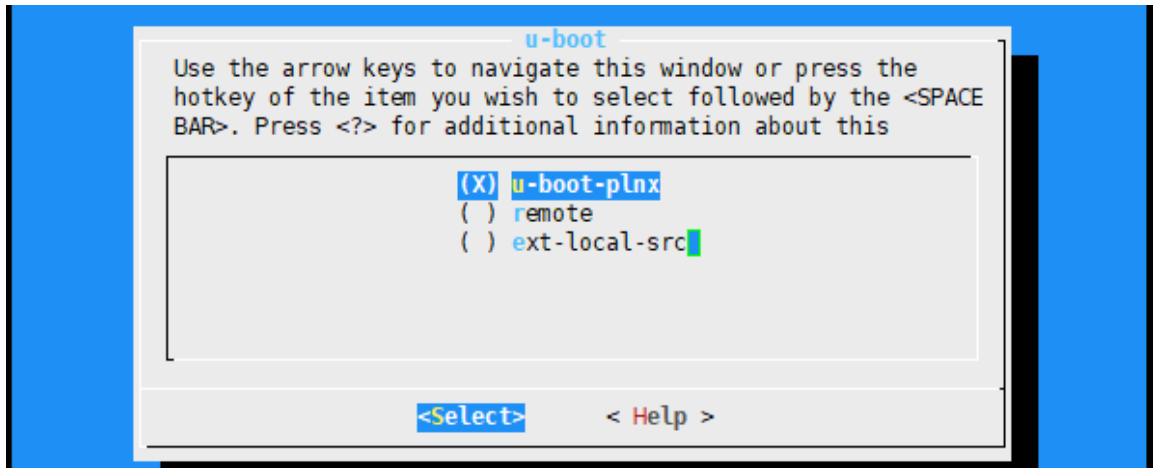


图 2.1.5.2.3-2 u-boot-plnx 默认配置

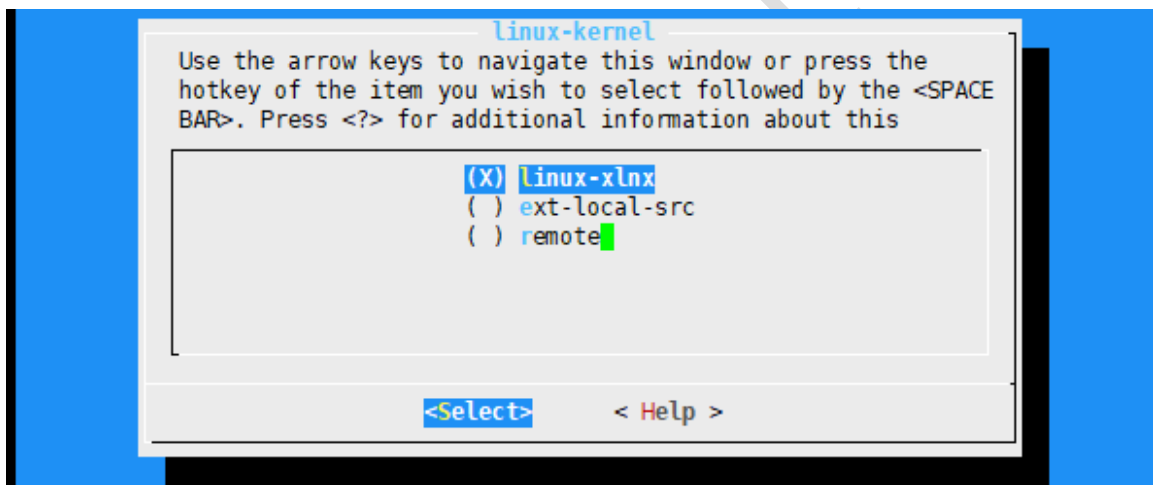


图 2.1.5.2.3-3 linux-xlnx 默认配置

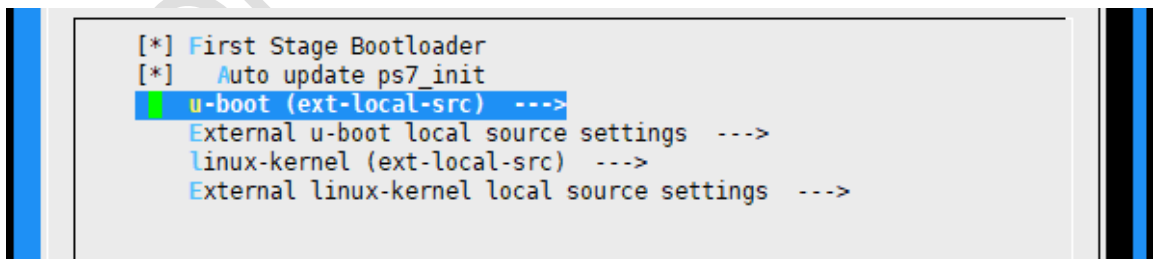


图 2.1.5.2.3-4 本地源配置

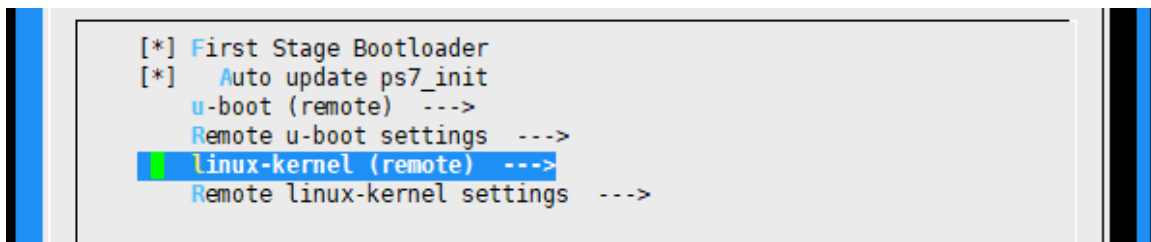


图 2.1.5.2.3-5 远程服务器源配置

以本地 Linux kernel 源码 GIT 仓库为例（参考所在地址为“/home/test/Documents/Git_sources/linux-xmlns”），配置 linux-kernel 为本地源码地址，在选择框内填写如源码的绝对地址并确认，具体简易流程，如图（图 2.1.5.2.3-6 到图 2.1.5.2.3-8）各步流程所示：

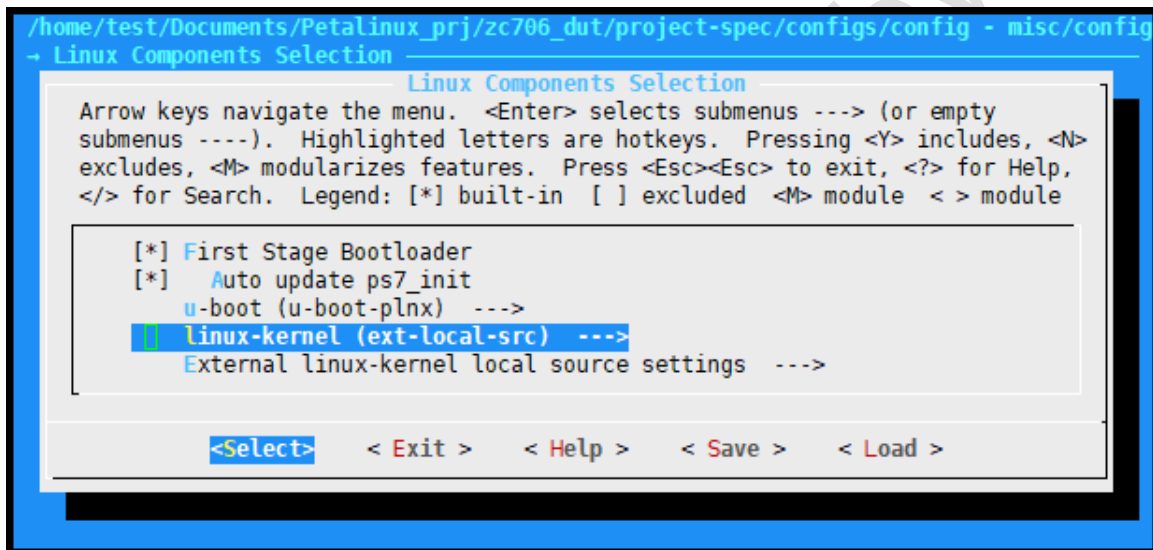


图 2.1.5.2.3-6 选择本地源

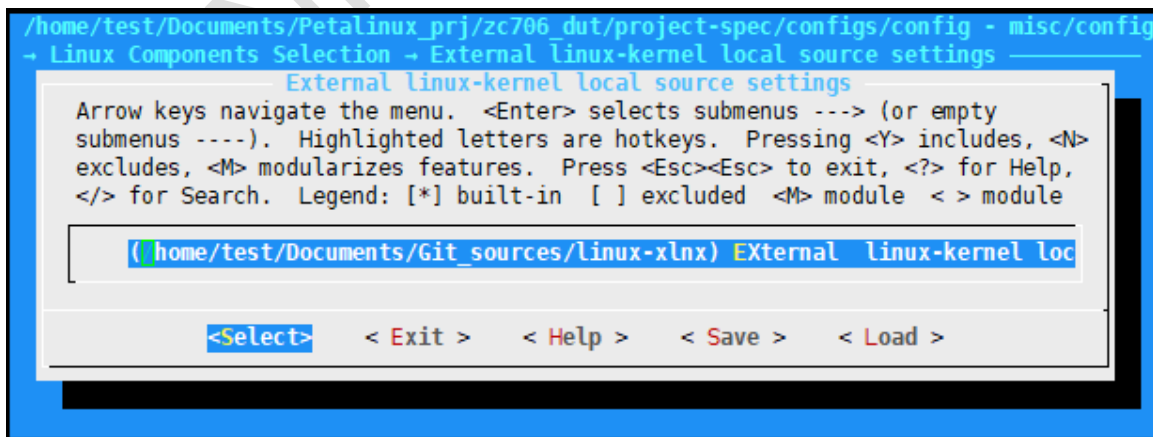


图 2.1.5.2.3-7 输入配置

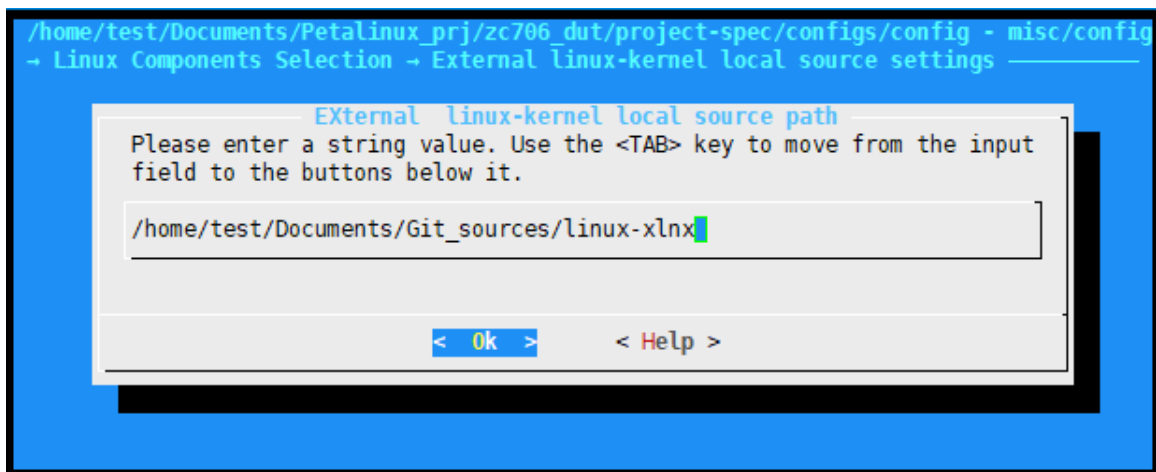


图 2.1.5.2.3-8 填写本地源

以远程 Xilinx 的 Linux kernel 源码 GIT 仓库为例（参考所在地址为“<https://github.com/Xilinx/linux-xlnx.git>”），配置 linux-kernel 为源码地址，在选择框内填写如源码的 GIT 地址并确认，具体简易流程，如图（图 2.1.5.2.3-9 到图 2.1.5.2.3-11）各步流程所示：

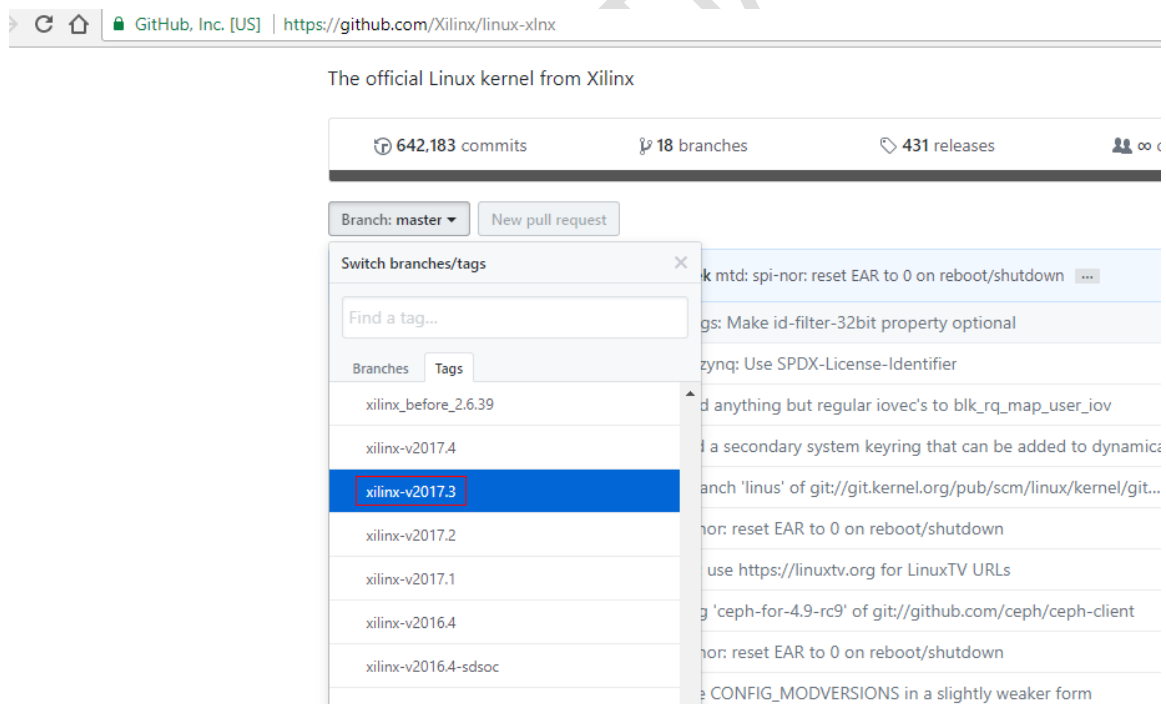


图 2.1.5.2.3-9 确认目标分支

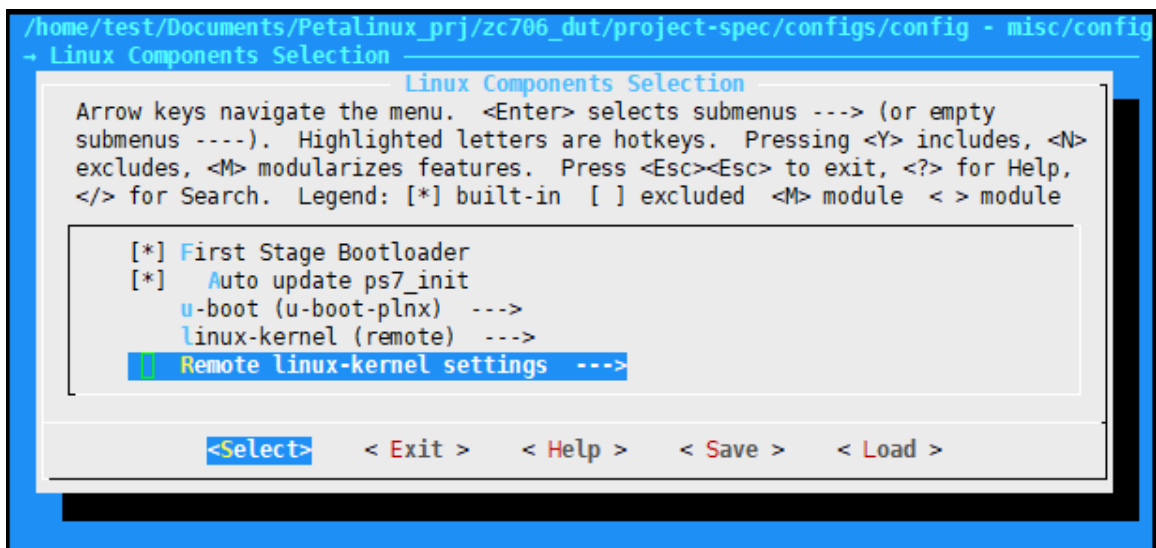


图 2.1.5.2.3-10 选择远程配置

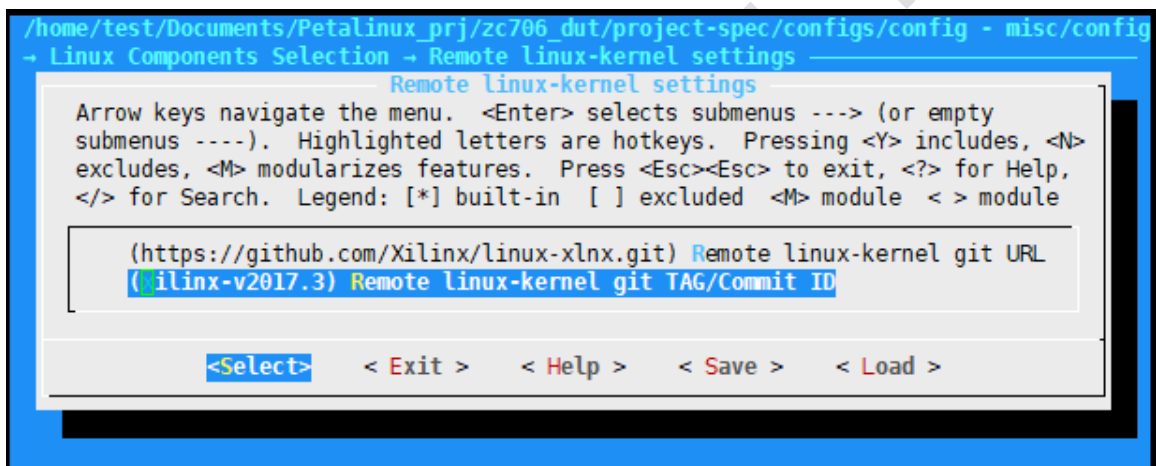


图 2.1.5.2.3-11 填写 git 地址和对应分支

2.1.5.2.4 启动方式

实验中可采用 3 种启动方式，分别是 QSPI Flash 启动、SD 卡启动和 NFS 启动方式。其中，开发板 zc706 上启动方式配置通过 SW11 开关进行配置，有如下图表（图

2.1.5.2.4-1 SW11 配置和启动方式定义）：

Boot Mode	SW11.1	SW11.2	SW11.3	SW11.4	SW11.5
JTAG mode ⁽¹⁾	0	0	0	0	0
Independent JTAG mode	1	0	0	0	0
QSPI mode	0	0	0	1	0
SD mode	0	0	1	1	0
MIO configuration pin	MIO2	MIO3	MIO4	MIO5	MIO6

图 2.1.5.2.4-1 SW11 配置和启动方式定义

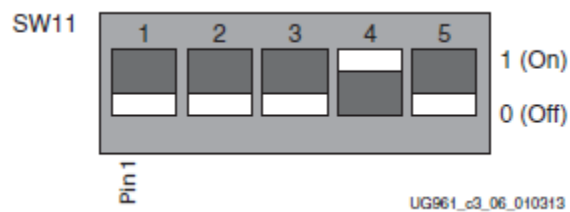


图 2.1.5.2.4-2 SW11 开关示意图

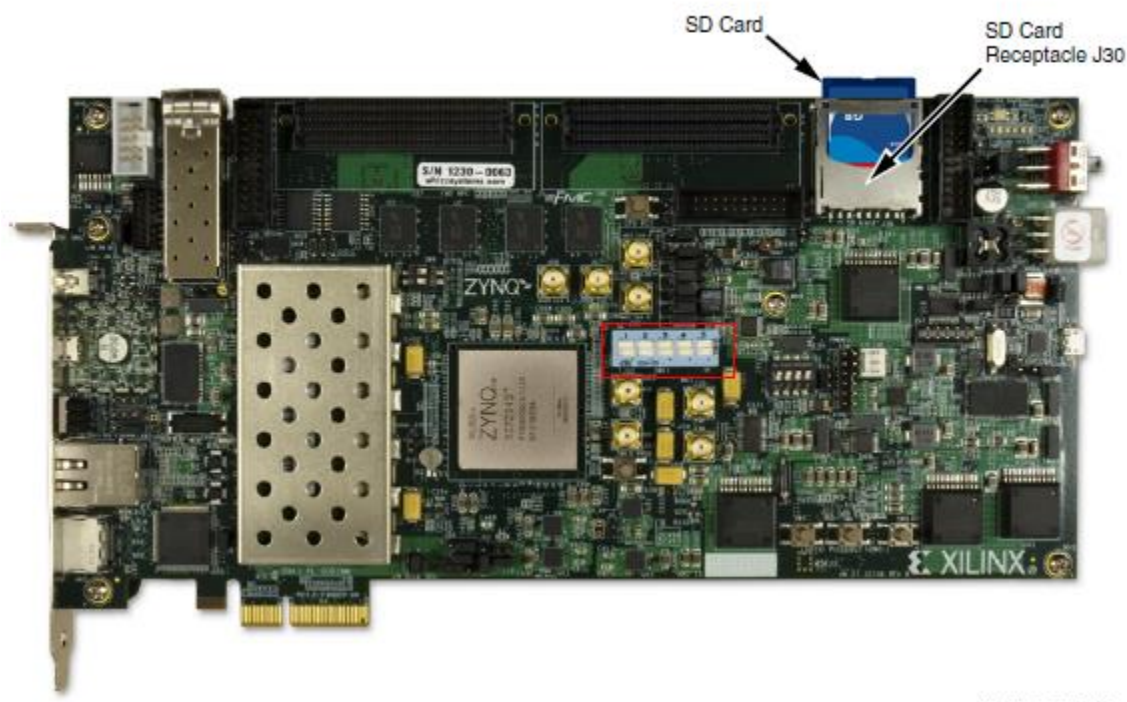


图 2.1.5.2.4-3 红色框内为 SW11 开关

2.1.5.2.4.1 采用 QSPI Flash 芯片启动

Flash 分区配置（若应用 qsip 的 flash 模块启动），文件系统分布如表（表 2.1.5.2.4-1）所示，需要配置对应的分区大小。

表 2.1.5.2.4.1-1 嵌入式文件系统

类别	内部结构和包含的文件类型		
1	BOOT		
	BOOT.BIN	Devicetree.dtb	uImage
	FSBL.elf		
	System_wrapper.bit		

	u-boot.elf		
2	ROOTFS	Linux_filesystem	

表 2.1.5.2.4.1-2 QSPI FLASH (128Mbit) 区间规划

空间名称	物理起始地址偏移	空间大小
boot.bin	0x00000000	0x1000000
env	0x1000000	0x0020000
image.ub	0x1020000	0x0A80000
其他	0x1AA0000	0x0560000

注意：

- 1) 目前调试阶段在 SD 卡上启动，SD 卡推荐大于 1Gb；
- 2) 用户空间可以随着 app 容量增大进行修正，但是需要重新压制 image.ub 文件。

QSPI Falsh 的系统配置（图 2.1.5.2.4.1-3 Flash 上分区分配置）需要根据表（表 2.1.5.2.4.1-2 QSPI FLASH (128Mbit) 区间规划）进行规划和配置，若选择的是 SD Flash 启动模式，则无需配置本处。但需要确认是否所有配置均从 SD 中启动，大多数情况下系统的 bootloader 的环境变量依然需要存储在 QSPI 的 Flash 中，配置选择如下图：

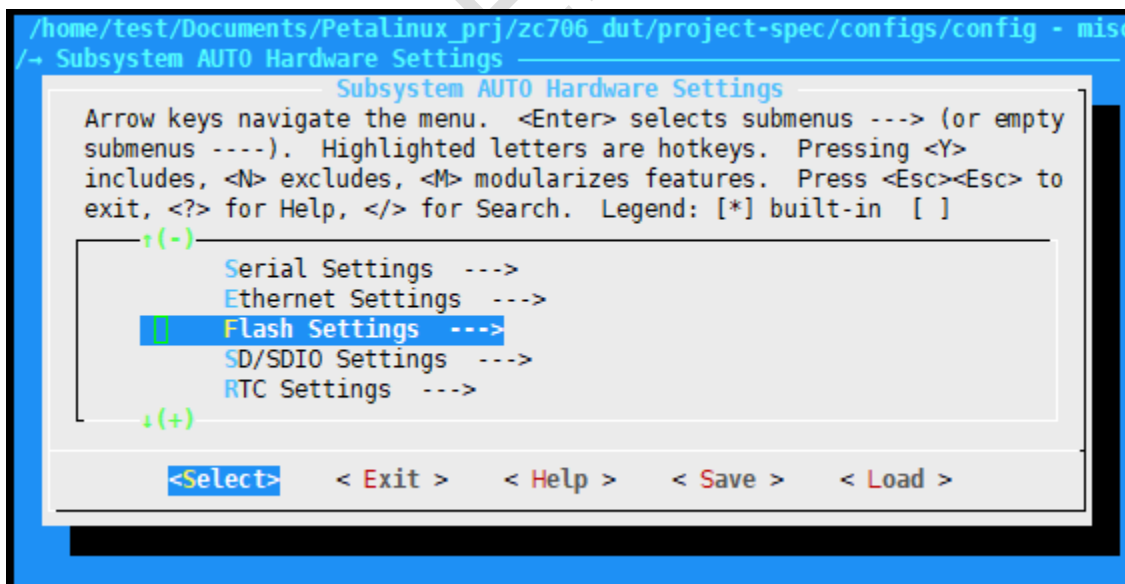


图 2.1.5.2.4.1-1 Flash 配置节点

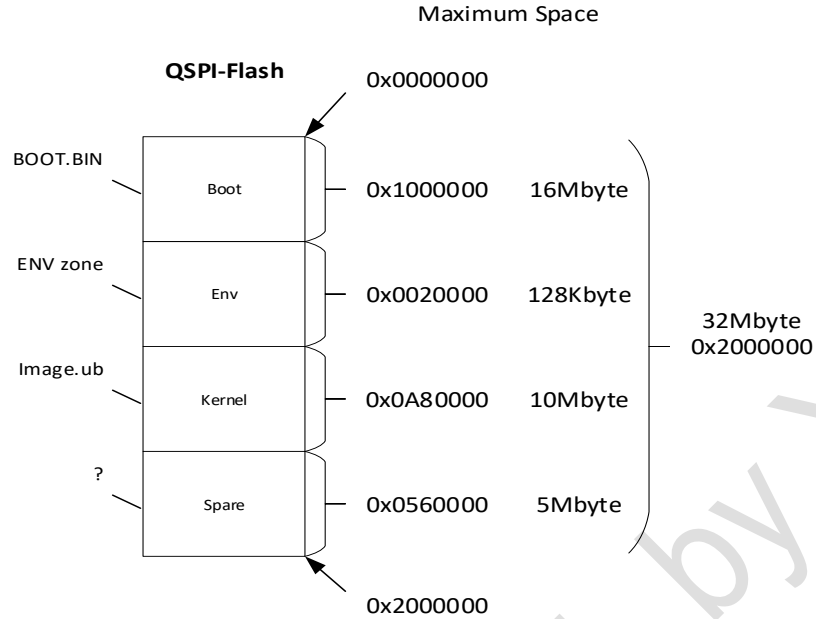


图 2.1.5.2.4.1-2 Flash 上分区分配图

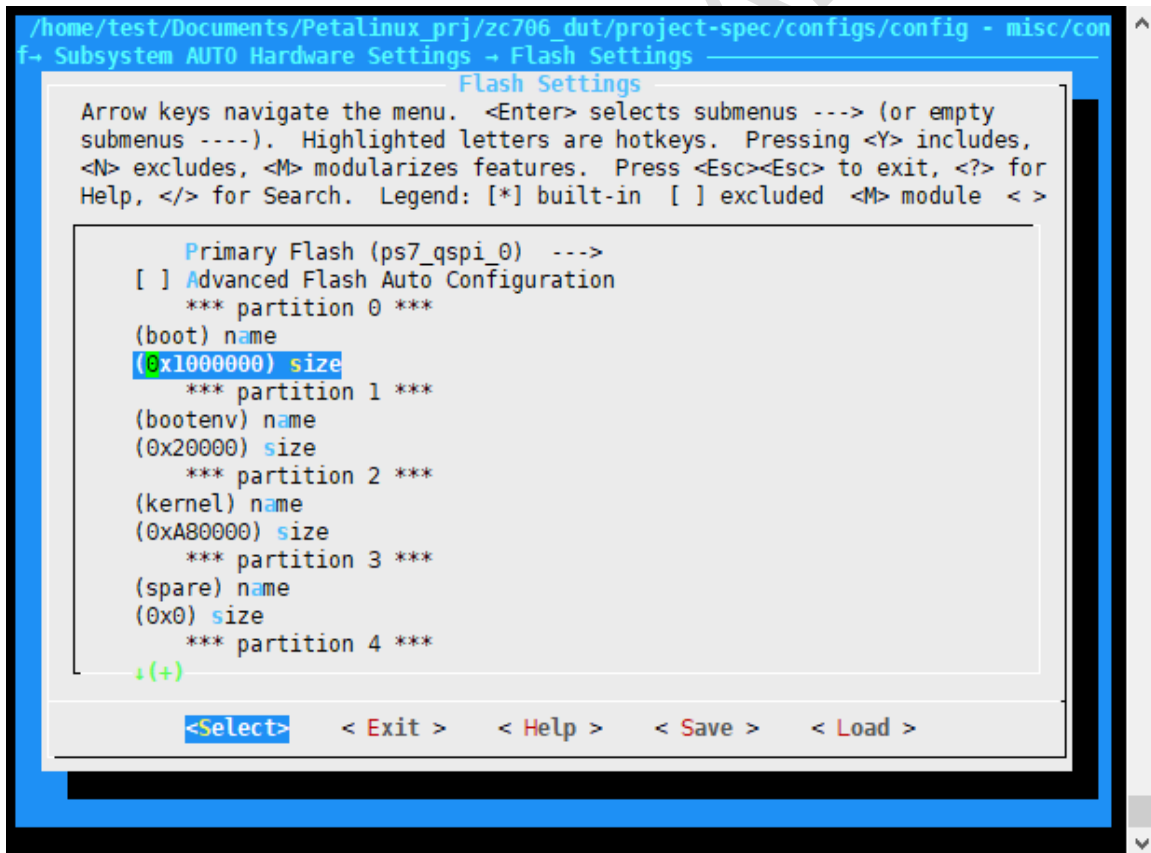


图 2.1.5.2.4.1-3 Flash 上分区分配配置

2.1.5.2.4.2 采用 SD 卡启动

只需要在默认大小的 SD Falsh 中写入相应的 BOOT.BIN 和 image.ub 文件则，通过启动硬件启动为 SD 卡启动则可以正常启动 kernel。配置启动存储器位置如：

/home/test/Documents/Petalinux_prj/zc706_dut/project-spec/configs/config - misc/conf→ Subsystem AUTO Hardware Settings → Advanced bootable images storage Settings

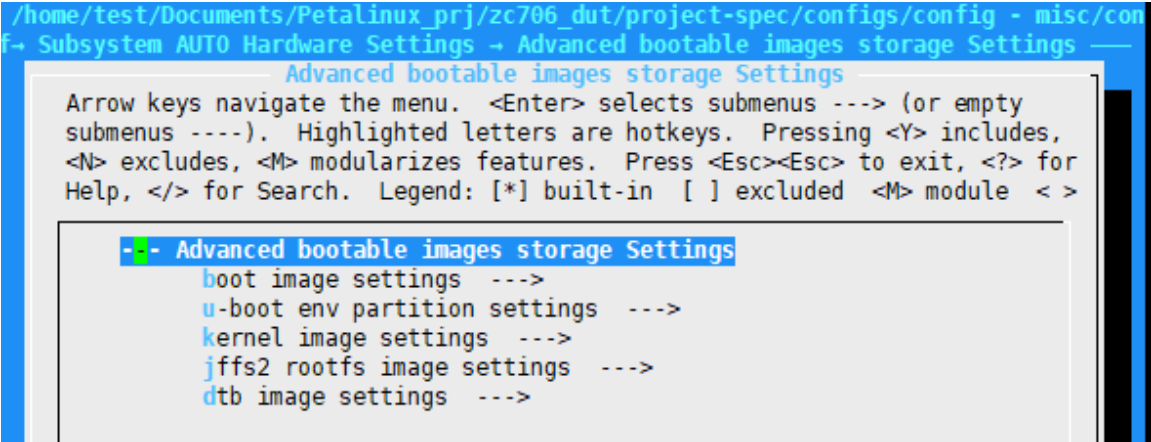


图 2.1.5.2.4.2-1 Flash 上分区分配置

SD 卡的 Flash 内容包括：

- 1) 第一区域：fsbl 启动文件、fpga bitstream、u-boot 启动文件；
- 2) 第二区域：image.ub（Kernel、Flattened Device Tree blob、conf@1）。

这部分为当前区间配置安排，配置位置文件位置：project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs。

其中，载入的 SDRAM 空间规划如下：

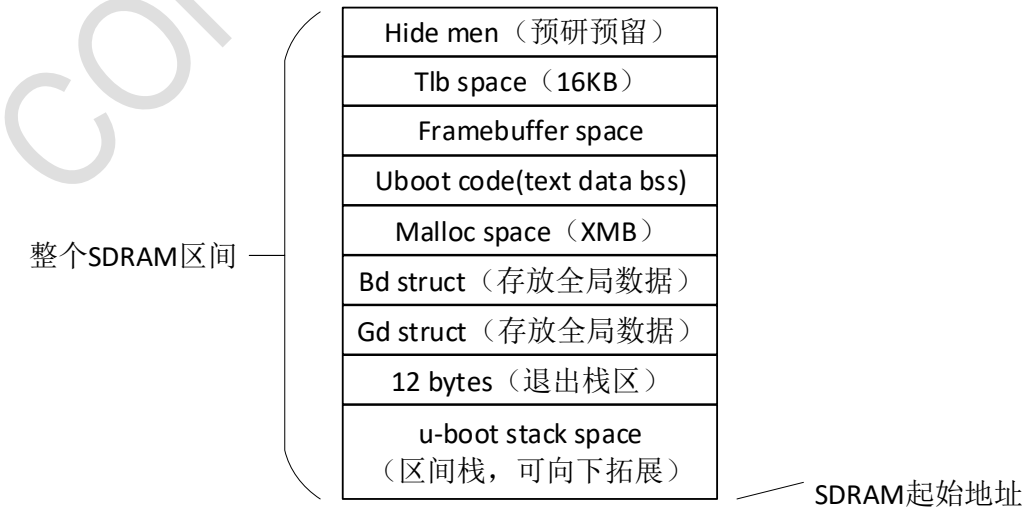


图 2.1.5.2.4.2-2 外装载入的 SDRAM 空间规划分布

配置成功后提示:

```
Your configuration changes were saved.
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
~/Documents/Petalinux_prj/zc706_dut/build/misc/plnx-generated
~/Documents/Petalinux_prj/zc706_dut
~/Documents/Petalinux_prj/zc706_dut
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
~/Documents/Petalinux_prj/zc706_dut/build/misc/plnx-generated
~/Documents/Petalinux_prj/zc706_dut
~/Documents/Petalinux_prj/zc706_dut
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
```

2.1.5.2.5 添加用户应用程序

通过 Petalinux 的模板中添加用户程序并加入系统编译。

- 1) \$ cd <plnx-proj-root>
- 2) \$ petalinux-create -t apps [--template TYPE] --name <user-application-name> --enable

例如, 添加 “helloworld” 例程:

- 1) 普通添加并加入编译:
\$ petalinux-create -t apps --name helloworld -enable
- 2) 模板添加并加入编译:
\$ petalinux-create -t apps --template c --name helloworld -enable
- 3) 添加 c++ 程序:
\$ petalinux-create -t apps --template c++ --name helloworld -enable
- 4) 添加 “helloworld” 为 autoconf application 程序:
\$ petalinux-create -t apps --template autoconf --name helloworld --enable
- 5) 添加之后的应用程序存放在目录:

<plnx-proj-root>/project-spec/meta-user/recipes-apps/helloworld/

例如:

```
test@tub:~/Documents/Petalinux_prj/zc706_dut$ petalinux-create -t apps --template c --
name helloworld --enable
INFO: Create apps: helloworld
INFO: New apps successfully created in
/home/test/Documents/Petalinux_prj/zc706_dut/project-spec/meta-user/recipes-
apps/helloworld
INFO: Enabling created component...
INFO: sourcing bitbake
INFO: oldconfig rootfs
INFO: helloworld has been enabled
test@tub:~/Documents/Petalinux_prj/zc706_dut/project-spec/meta-user/recipes-apps$ ls
gpio-demo helloworld myapp peekpoke
```

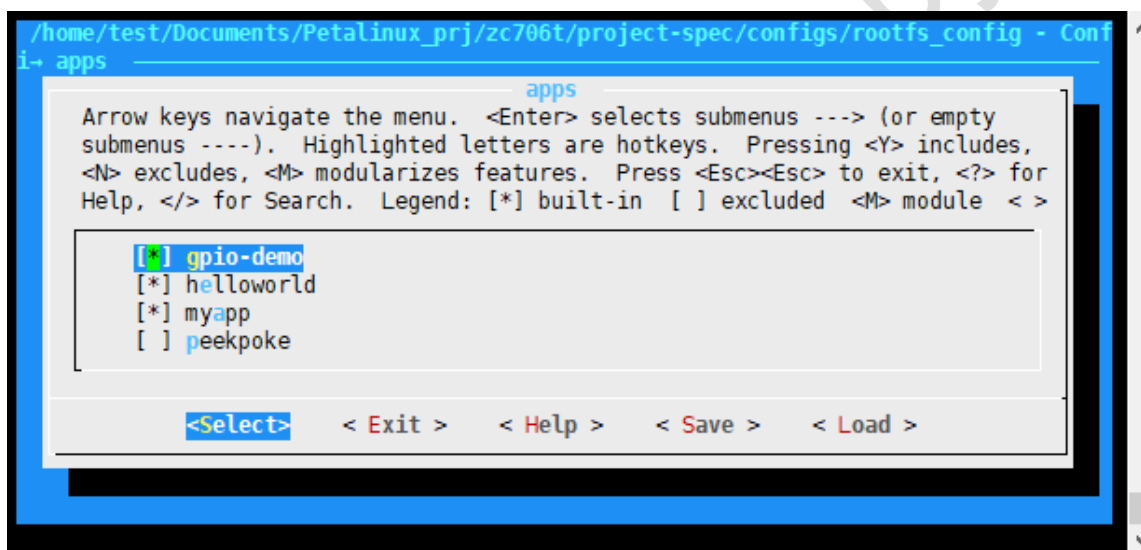


图 2.1.5.2.5 rootfs 中新添加并启用的 helloworld 用户程序

用户可以在目录“~/Documents/Petalinux_prj/zc706_dut/project-spec/meta-user/recipes-apps/helloworld”下对程序进行进一步的修改。

若添加新的头文件，需要在上级 Makefile 中添加相应的文件内容和参与编译的内容。

参与调试需要在“project-spec/meta-user/conf/petalinuxbsp.conf”文件中添加调试模块开关：RM_WORK_EXCLUDE += “helloworld”

其中，“artifacts”所在目录为：\${TMPDIR}/work/aarch64-xilinx-linux/helloworld/1.0-r0/。

2.1.5.2.6 添加用户模块

通过 Petalinux 的模板中添加用户模块并加入系统编译，用户可以通过添加模块模板或仅仅添加模块两种方式进行添加。

- 1) 进入相应的工程目录: `cd <plnx-proj-root>`
- 2) 添加模块: `$ petalinux-create -t modules --name <user-module-name> --enable`
- 3) 模块建立路径: `<plnx-proj-root>/project-spec/meta-user/recipes-modules/mymodule`

例如，添加 helloworld 模块，用户可以将模块内容和 Makefile 修改为特定项目功能，可以之间替换文件和写入 makefile:

- 1) 进入工程目录: `cd ~/Documents/Petalinux_prj/zc706_ptp`
- 2) 创建模块: `$ petalinux-create -t modules --name helloworld --enable`
- 3) 模块路径: `/home/test/Documents/Petalinux_prj/zc706_ptp/project-spec/meta-user/recipes-modules`

例如:

```
test@tub:~/Documents/Petalinux_prj/zc706_dut$ petalinux-create -t modules --name helloworld --enable
INFO: Create modules: helloworld
INFO: New modules successfully created in
/home/test/Documents/Petalinux_prj/zc706_dut/project-spec/meta-user/recipes-modules/helloworld
INFO: Enabling created component...
INFO: sourcing bitbake
INFO: oldconfig rootfs
INFO: helloworld has been enabled
```

```
test@tub:~/Documents/Petalinux_prj/zc706_ptp/project-spec/meta-user/recipes-modules$ ls
demo helloworld virtnet
```

图 2.1.5.2.6-1 添加 helloworld 模块

```
test@tub:~/Documents/Petalinux_prj/zc706_ptp/project-spec/meta-user/recipes-modules/helloworld$ tree
.
├── files
│   ├── COPYING
│   ├── helloworld.c
│   ├── Makefile
│   ├── helloworld.bb
│   └── README
└──
1 directory, 5 files
```

图 2.1.5.2.6-2 模块模板文件内容

表 2.1.5.2.6 模块文件

模板	描述
Makefile	编译文件模板
README	模块简易描述
Helloworld.c	Helloworld.c 模块
<plnx-proj-root>/projectspec/ meta-user/recipes-co re/images/petalinux-image .bbappened	模块配置文件所在位置和文件。

参与调试需要在“project-spec/meta-user/conf/petalinuxbsp.conf”文件中添加调试模块开关：RM_WORK_EXCLUDE += “helloworld”

其中，“artifacts”所在目录为：\${TMPDIR}/work/aarch64-xilinx-linux/helloworld/1.0-r0/。

2.1.6 打包 BSP 文件

打包 BSP 文件目的是基于项目合作的支持文件共享。在工程已经完成完整的“petalinux-build”基础上，进行“petalinux-package”操作，具体步骤如下：

- 1) 退出到工程目录
- 2) \$ petalinux-package --bsp -p <plnx-proj-root> --output xxx.BSP
- 3) 打包如下组件：
 - <plnx-proj-root>/project-spec/
 - <plnx-proj-root>/config.project
 - <plnx-proj-root>/.petalinux/
 - <plnx-proj-root>/pre-built/
 - All selected components

- 4) 添加硬件文件

```
$ petalinux-package --bsp -p <plnx-proj-root> --hwsources <hw-project-root>
--output xxx.BSP
```

- 5) 添加外源组件到目录“components/ext_sources.”下。

2.1.7 配置固件版本

配置固件版本主要用于项目开发过程中固件软件的版本确定和审核之用，具体步骤如下：

- 1) 退出到退出到工程目录
- 2) 进入系统配置目录：\$ petalinux-config
- 3) 选择 “Firmware Version Configuration”
- 4) 选择 “Host Name, Product Name, Firmware Version” 等加以配置；
- 5) 退出配置目录，并选择 “yes” 保存。

2.1.8 Qemu 调试

必须条件为：完整编译生成 BOOT.BIN、uImage.ub 镜像。

2.1.8.1 Qemu 调试 u-boot

调试 u-boot 启动命令：\$ petalinux-boot --qemu --u-boot

Zynq-7000 系列启动 QEMU 包含如下文件：

- <plnx-proj-root>/images/linux/u-boot.elf

Zynq UltraScale+ MPSoC 启动 QEMU 包含如下文件：

- <plnx-proj-root>/images/linux/u-boot.elf
- <plnx-proj-root>/images/linux/bl31.elf

2.1.8.2 Qemu 调试 Linux Kernel

调试 linux kernel 启动命令：\$ petalinux-boot --qemu - kernel

Zynq-7000 系列启动 QEMU 包含如下文件：

- <plnx-proj-root>/images/linux/zImage

Zynq UltraScale+ MPSoC 系列启动 QEMU 包含如下文件：

- <plnx-proj-root>/images/linux/Image
- <plnx-proj-root>/images/linux/bl31.elf

2.1.8.3 Qemu 退出方式

Qemu 的中断退出方式为命令退出步骤：

- 1) 同时按住：Ctrl+A
- 2) 释放上步再按：X

2.1.9 生成镜像

利用“petalinux-build”生成系统镜像文件，文件生成位于“images/linux/”子目录。生成镜像命令为：`$ petalinux-build`，其中日志信息统计在“build.log”中。

生成成功后提示：

```
test@tub:~/Documents/Petalinux_prj/zc706_dut$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
INFO: bitbake petalinux-user-image
Parsing of 2468 .bb files complete (2436 cached, 32 parsed). 3261 targets, 226 skipped, 0
masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100%
|#####| Time: 0:00:12
Initialising tasks: 100%
|#####| Time: 0:00:18
Initialising tasks: 100%
|#####| Time: 0:01:54
Checking sstate mirror object availability: 100%
|#####| Time:
0:00:04
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
fsbl-2017.3+gitAUTOINC+3c9f0cfde9-r0 do_compile: NOTE: fsbl: compiling from external
source tree /home/test/Petalinux/tools/hsm/data/embeddedsw
NOTE: Tasks Summary: Attempted 2458 tasks of which 2443 didn't need to be rerun and all
succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
NOTE: Successfully copied built images to tftp dir: /tftpboot
[INFO] successfully built project
```

2.1.9.1 生成 uImage 镜像

若重新配置 u-boot 启动 uImage，需要“PetaLinux u-boot config”中作为 u-boot 的配置目标，修位置为“<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h”，并重 CONFIG_EXTRA_ENV_SETTING。

其中生成 uImage 的命令如下：

```
$ petalinux-package --image -c kernel --format uImage
```

例如：

```

test@tub:~/Documents/Petalinux_prj/zc706_dut/images/linux$ petalinux-package --image -
c kernel --format
ulmage
SDK environment now set up; additionally you may now run devtool to perform development
tasks.
Run devtool --help for further details.
### Shell environment set up for builds. ###
You can now run 'bitbake <target>'
Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support
You can also run generated qemu images with a command like 'runqemu qemux86'
INFO: Adding user layer: /home/test/Documents/Petalinux_prj/zc706_dut/project-
spec/meta-user
INFO: generating ulmage
INFO: bitbake -R
/home/test/Documents/Petalinux_prj/zc706_dut/build/conf/kerneltype.conf virtual/kernel
Parsing recipes: 100%
|#####| Time:
0:01:26
Parsing of 2468 .bb files complete (0 cached, 2468 parsed). 3261 targets, 226 skipped, 0
masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100%
|#####| Time:
0:00:11
Checking sstate mirror object availability: 100%
|#####| Time: 0:00:05
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2458 tasks of which 2408 didn't need to be rerun and all
succeeded.
test@tub:~/Documents/Petalinux_prj/zc706_dut/images/linux$ ls
BOOT.BIN  rootfs.cpio.gz  rootfs.ext4.gz  system.dtb  ulmage
image.ub  rootfs.cpio.gz.u-boot  rootfs.jffs2  System.map.linux  vmlinux
PtP_Top.bit  rootfs.ext3  rootfs.manifest  u-boot.bin  zlmage
rootfs.cpio  rootfs.ext4  rootfs.tar.gz  u-boot.elf  zynq_fsbl.elf
test@tub:~/Documents/Petalinux_prj/zc706_dut/images/linux$

```

2.1.9.2 生成 Zynq UltraScale+ MPSoC 启动镜像

在确认硬件 “*.hdf” 文件对应平台无误情况下，导入 “FSBL” 镜像、FPGA
“bitstream” 文件，并执行如下命令生成镜像：

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --pmufw
```

<PATH_TO_PMU_FW_ELF> --u-boot

2.1.9.3 生成 Zynq 系列启动 BOOT.BIN 镜像

生成文件为“BOOT.BIN”，确认硬件“*.hdf”文件对应平台无误情况下，导入“FSBL”镜像、FPGA“bitstream”文件，并执行如下命令生成镜像：

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

例如：

```
test@tub:~/Documents/Petalinux_prj/zc706_dut/images/linux$ petalinux-package --boot --fsbl zynq_fsbl.elf --fpga PtP_Top.bit --u-boot
INFO: File in BOOT BIN:
"/home/test/Documents/Petalinux_prj/zc706_dut/images/linux/zynq_fsbl.elf"
INFO: File in BOOT BIN:
"/home/test/Documents/Petalinux_prj/zc706_dut/images/linux/PtP_Top.bit"
INFO: File in BOOT BIN: "/home/test/Documents/Petalinux_prj/zc706_dut/images/linux/u-boot.elf"
INFO: Generating zynq binary package BOOT.BIN...
INFO: Binary is ready.
test@tub:~/Documents/Petalinux_prj/zc706_dut/images/linux$ ls
BOOT.BIN  rootfs.cpio.gz  rootfs.ext4.gz  system.dtb  image.ub  rootfs.cpio.gz.u-boot
rootfs.jffs2  System.map.linux  vmlinux
PtP_Top.bit  rootfs.ext3      rootfs.manifest  u-boot.bin   zImage
rootfs.cpio  rootfs.ext4      rootfs.tar.gz    u-boot.elf   zynq_fsbl.elf
```

2.1.9.4 编译用户应用程序

生成应用程序流程在原有镜像基础上添加用户程序，例如在“petalinux-build”基础上建立 helloworld 的用户程序，需要通过如下命令在原有生成镜像中加入已经修改的用户程序到 rootfs 中。编译之后文件所在位置为“<TMPDIR>/work/aarch64-xilinx-linux/helloworld/1.0-r0/”

具体步骤如下：

- 1) 进入工程目录：\$ cd <plnx-proj-root>
- 2) 添加 sysroot：\$ petalinux-build -x do_populate_sysroot
- 3) 重建 rootfs 文件系统：\$ petalinux-build -c rootfs
- 4) 打包新用户程序到镜像：\$ petalinux-build -x package
- 5) 清除用户程序命令如下：\$ petalinux-build -c helloworld -x do_clean
- 6) 安装用户程序：\$ petalinux-build -c helloworld -x do_install
- 7) 重建用户程序命令如下：\$ petalinux-build -c helloworld

8) 重新打包 rootfs: \$ petalinux-build -c rootfs

2.1.9.5 编译用户模块

编译模块在“petalinux-build”基础上建立 helloworld 模块，启用特定模块的编译开关，编译以后模块所在位置为“<TMPDIR>/work/{ARCH}-xilinx-linux-gnueabi/helloworld/1.0-r0/”，编译模块步骤：

编译整个完整镜像: \$ petalinux-build

在完整镜像中添加模块：

- 1) 进入工程目录: \$ cd <plnx-proj-root>
- 2) 建立用户 rootfs 目录: \$ petalinux-build -c rootfs
- 3) 重建镜像文件: \$ petalinux-build -x package
- 4) 清除用户模块 helloworld: \$ petalinux-build -c helloworld -x do_cleansstate
- 5) 重编译用户模块 helloworld: \$ petalinux-build -c helloworld
- 6) 安装用户模块: \$ petalinux-build -c helloworld -x do_install, 本命令添加用户命令到 rootfs 文件“<TMPDIR>/work/plnx_aarch64-xilinx-linux/petalinux-user-image/1.0-r0/rootfs/.”中。

2.1.10 版本控制

控制版本需要控制如下文件内容，可以利用 SVN 或者 GIT 进行版本控制，其中“<plnx-proj-root>”代表的是当前工程目录，具体子目录如下：

- 1) <plnx-proj-root>/petalinux
- 2) <plnx-proj-root>/build/
- 3) <plnx-proj-root>/images/
- 4) <plnx-proj-root>/pre-built/
- 5) <plnx-proj-root>/project-spec/meta-plnx-generated/
- 6) <plnx-proj-root>/components/plnx-workspace/

默认配置情况下，在创建工程过程中，如上文件均已被添加到“.gitignore”中。在共享工程时，建议打包当前工程为 BSP 包提供。

2.1.11 GDB 调试

利用 GDB 调试需要在 Qemu 中进行，例如在完成工程编译基础上，需要调试 kernel 镜像和对应的用户 app，则有如下步骤：

- 1) 调试 kernel 情况下, 启动 Qemu 调试: `$ petalinux-boot --qemu - kernel`
- 2) 在 Qemu 的窗口中获取 GDB 的 TCP 端口, 如 `-gdb tcp:<TCP_PORT>`

INFO: qemu-system-arm ... -gdb tcp::9001 ...

- 3) 开启另一个命令窗口, 并进入项目镜像目录 `"$ cd "<plnx-proj-root>/images/linux"`
- 4) 在 vmlinux kernel 镜像上开启, GDB: `$ petalinux-util --gdb vmlinux`
- 5) 进入 GDB 调试模式, 并用标准 GDB 进行调试

```
GNU gdb (Linaro GDB 2017.03) 7.12.1.20170130-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu
--target=aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vmlinux...done.
```

- 6) 在 GDB 中添加 Qemu 目标: `(gdb) target remote :9000`
- 7) 继续 Qemu 执行程序: `(gdb) continue`
- 8) 退出方法为: `control+c`
- 9) 设置断电方式为: `b`
- 10) 进步方式为: `s`

2.1.12 限制条款

2.1.12.1 Petalinux 使用限制

2.1.12.1.1 环境适配

Petalinux 必须和相应的 Vivado 适配, 不适配会出现 dhf 导出文件无法正常通过 `get_hw_description` 导入到相应的项目中或报错等。适配平台, 如:

- 1) Petalinux v2017.3 必须和 Vivado 2017.3 配对搭建和应用;
- 2) Petalinux v2016.4 必须和 Vivado 2016.4 配对搭建和应用;
- 3) Petalinux v2015.4 必须和 Vivado 2015.4 配对搭建和应用。

2.1.12.1.2 文件夹保护

安装完 Petalinux 之后不能再移动 Petalinux 的文件夹位置，否则将导致工具链无法启用。

2.1.12.2 Shell 环境

Petalinux 工具集需要系统的 bash shell 环境。从 Ubuntu 6.10 开始，默认使用 dash(the Debian Almquist Shell)而不是 bash(the GNUBourne-Again Shell)。Login Shell 还是 bash，原因是 dash 更快、更高效，而且它符合 POSIX 规范。Ubuntu 在启动的时候会运行很多 shell 脚本，使用 dash 可以加快启动速度。

通过如下命令切换默认 shell 到 bash: `sudo dpkg-reconfigure dash`

将环境通过提示窗口配置到 bash 环境，如图 2.1.12.2-1/2 中所示。



图 2.1.12.2-1 切换 shell 环境

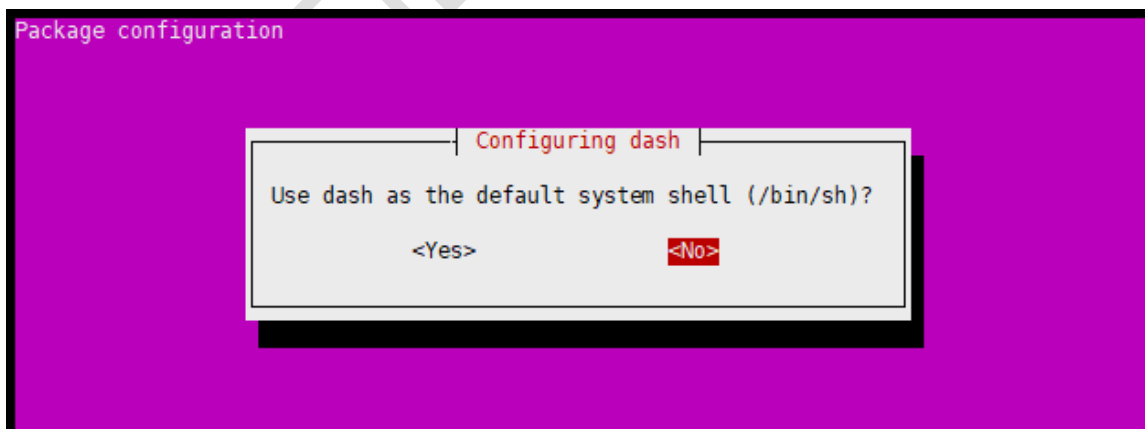


图 2.1.12.2-2 切换 shell 切换窗口

2.1.12.3 NFS 启动方式

若 Petalinux 工程应用的是在板 NFS 启动，“petalinux-create”将自动切换 TMPDIR 到 “/tmp/<projname_timestamp>” 的位置启动。

```
U-Boot 2017.03 (Jan 28 2017 - 14:38:56 +0800)
Board: Xilinx Zynq
I2C: ready
DRAM: ECC disabled 1 GiB
MMC: sdhci@e0100000: 0 (SD)
SF: Detected s25fl128s_64k with page size 512 Bytes, erase size 128 KiB, total 32 MiB
In: serial
Out: serial
Err: serial
Net: ZYNQ GEM: e000b000, phyaddr ffffffff, interface rgmii-id
Warning: ethernet@e000b000 (eth1) using random MAC address - da:6d:cc:f4:fb:3c
eth1: ethernet@e000b000
Hit any key to stop autoboot: 0
SF: Detected s25fl128s_64k with page size 512 Bytes, erase size 128 KiB, total 32 MiB
device 0 offset 0x1020000, size 0xa80000
SF: 11010048 bytes @ 0x1020000 Read: OK
Wrong Image Format for bootm command
ERROR: can't get kernel image!
Zynq> tftpboot image.ub
Using ethernet@e000b000 device
TFTP from server 192.168.206.200; our IP address is 192.168.206.234
Filename 'image.ub'.
Load address: 0x1000000
Loading: #####
#####
#####
#####
#####
147.5 KiB/s
done
Bytes transferred = 9495336 (90e328 hex)
Zynq> bootm
.....
Starting kernel ...
4 ofpart partitions found on MTD device spi0.0
Creating 4 MTD partitions on "spi0.0":
Uncompressing Linux... done, booting the kernel.

PetaLinux 2017.3 zc706_dut /dev/ttyPS0

zc706_dut login:
zc706_dut login: root
Password:
root@zc706_dut:~#
```

2. 1. 13 FIT Image 镜像日志

此处记录 u-boot 启动 kernel 镜像日志：

```
## Loading kernel from FIT Image at 01000000 ...
  Using 'conf@1' configuration
  Verifying Hash Integrity ... OK
  Trying 'kernel@0' kernel subimage
    Description: Linux Kernel
    Type:       Kernel Image
    Compression: uncompressed
    Data Start: 0x010000d4
    Data Size:  3746560 Bytes = 3.6 MiB
    Architecture: ARM
    OS:         Linux
    Load Address: 0x00008000
    Entry Point: 0x00008000
    Hash algo:  sha1
    Hash value:  38ae503cf9148ca05d8753419123649dbb1e084f
  Verifying Hash Integrity ... sha1+ OK
## Loading ramdisk from FIT Image at 01000000 ...
  Using 'conf@1' configuration
  Trying 'ramdisk@0' ramdisk subimage
    Description: ramdisk
    Type:       RAMDisk Image
    Compression: uncompressed
    Data Start: 0x01396ce4
    Data Size:  5730898 Bytes = 5.5 MiB
    Architecture: ARM
    OS:         Linux
    Load Address: unavailable
    Entry Point: unavailable
    Hash algo:  sha1
    Hash value:  cfd7f088db14665681de5827835fb14d04dc90e
  Verifying Hash Integrity ... sha1+ OK
## Loading fdt from FIT Image at 01000000 ...
  Using 'conf@1' configuration
  Trying 'fdt@0' fdt subimage
    Description: Flattened Device Tree blob
    Type:       Flat Device Tree
    Compression: uncompressed
    Data Start: 0x01392cc8
    Data Size:  16235 Bytes = 15.9 KiB
    Architecture: ARM
    Hash algo:  sha1
    Hash value:  f46aa709f6268f9c562a90ebb29690339d39a3ad
  Verifying Hash Integrity ... sha1+ OK
```


2.2 Vivado

2.2.1 安装 Vivado

Vivado 系列软件下载、安装、注册和授权，请详细参照《Vivado Design Suite Release Notes, Installation and Licensing Guide》，ug973，官方用户手册。在公司未提供正版 License 情况下，部分 IP 需要自行通过 Xilinx 官方网站申请试用版 License。

2.2.2 创建工程

打开 Vivado 软件，直接在欢迎界面点击 Create Project，或在开始菜单中选择 File - New Project 即可新建工程。

之后配置项目名称、项目路径和开发板或平台选择匹配硬件原理架构之后，即可开展 FPGA 硬件编码、IP 核调用、IP 核设计和相关设计。

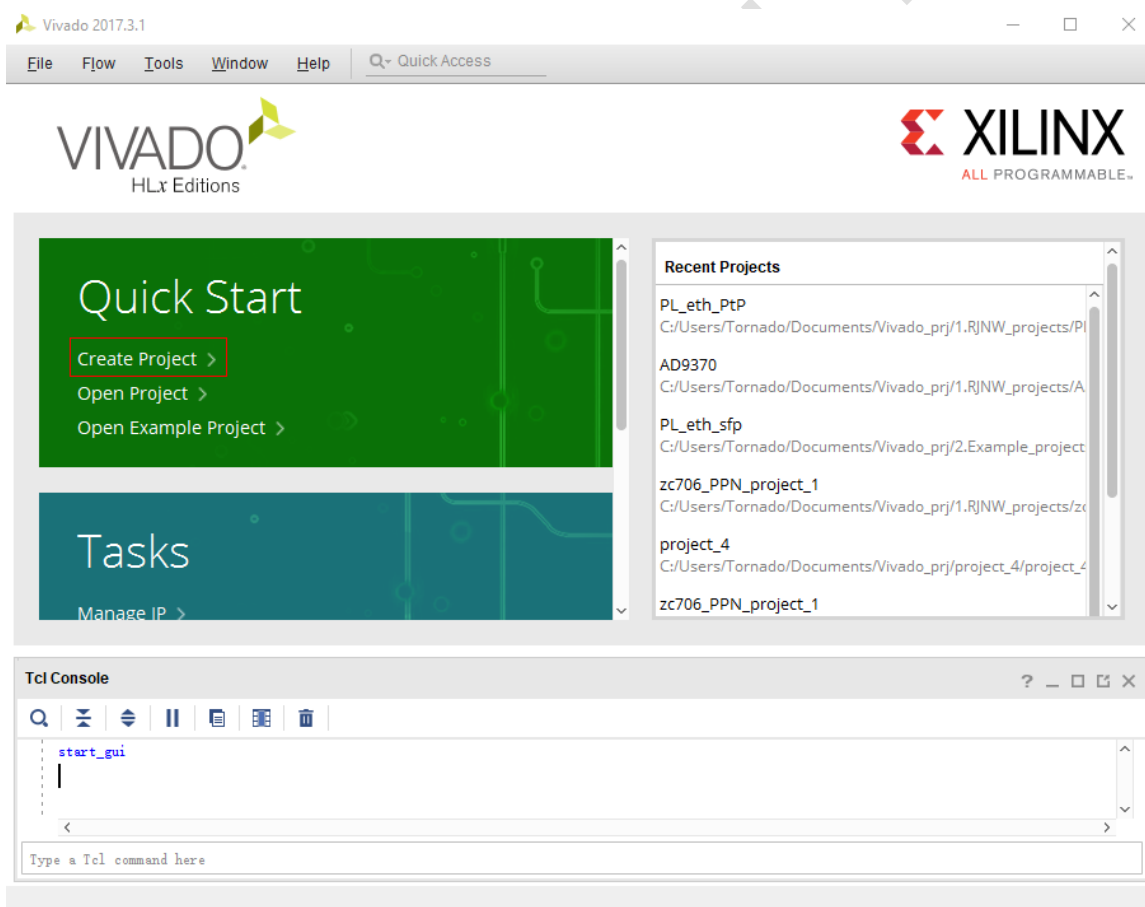


图 2.2.2-1 打开 Vivado 工具集

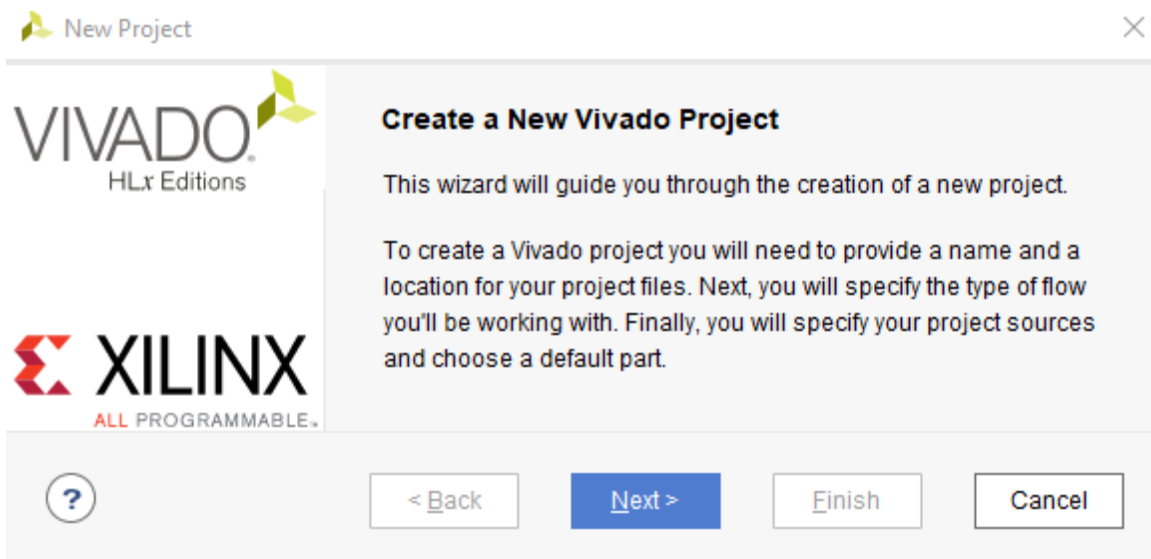


图 2.2.2-2 创建新工程

设置项目名称和项目文件位置路径，如下图（图 2.2.2-3 创建 zc706_dut 工程）所示。

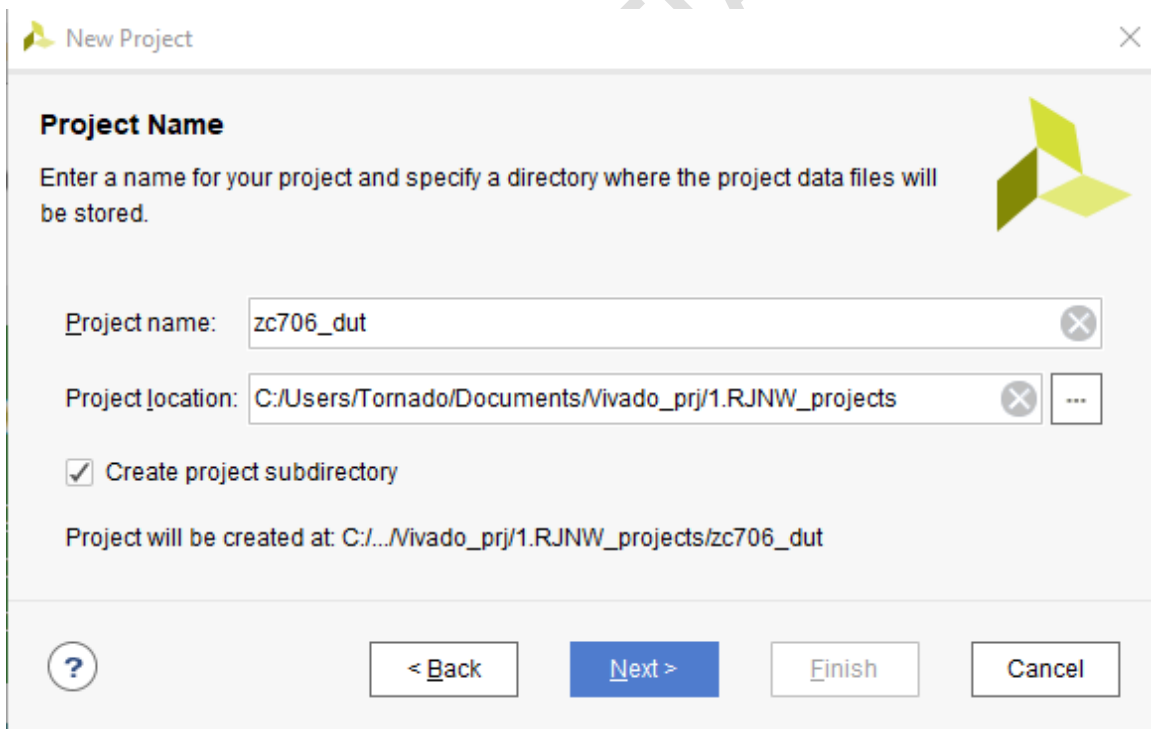


图 2.2.2-3 创建 zc706_dut 工程

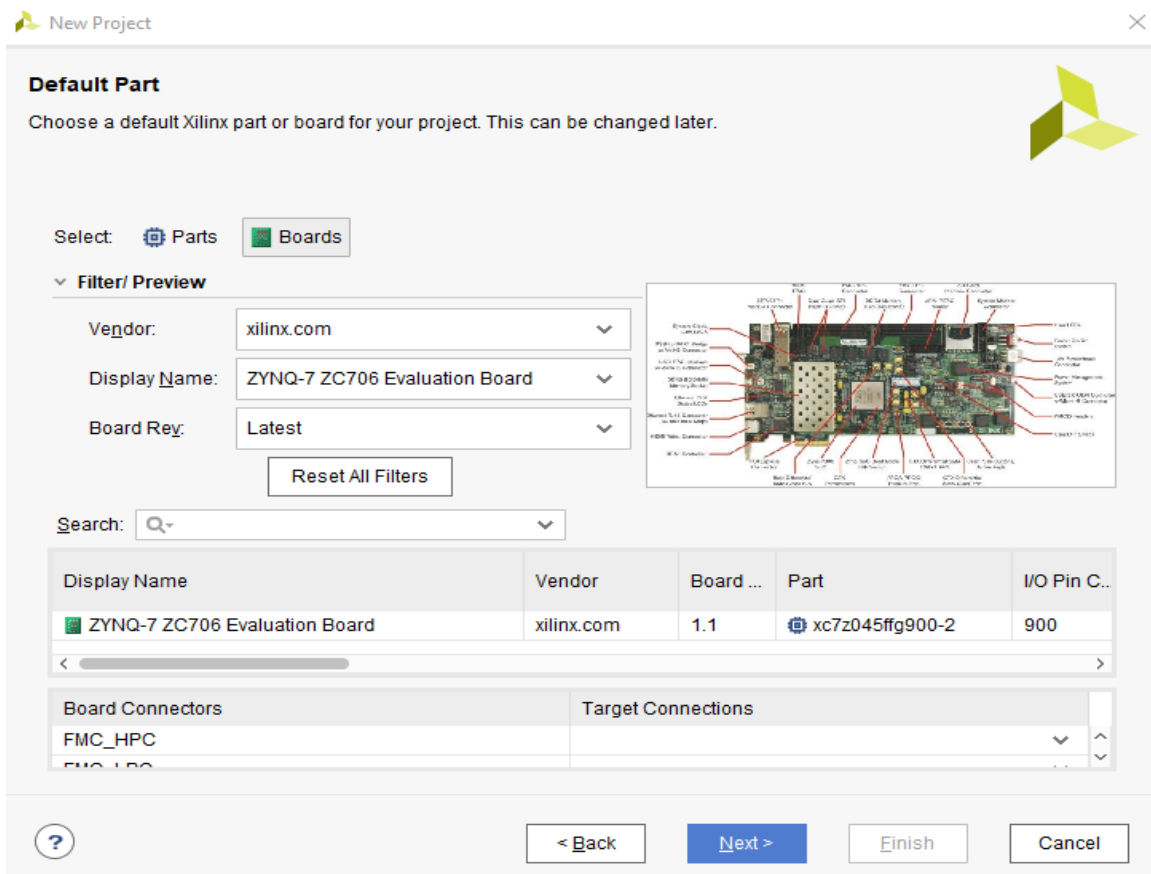


图 2.2.2-4 选择硬件平台

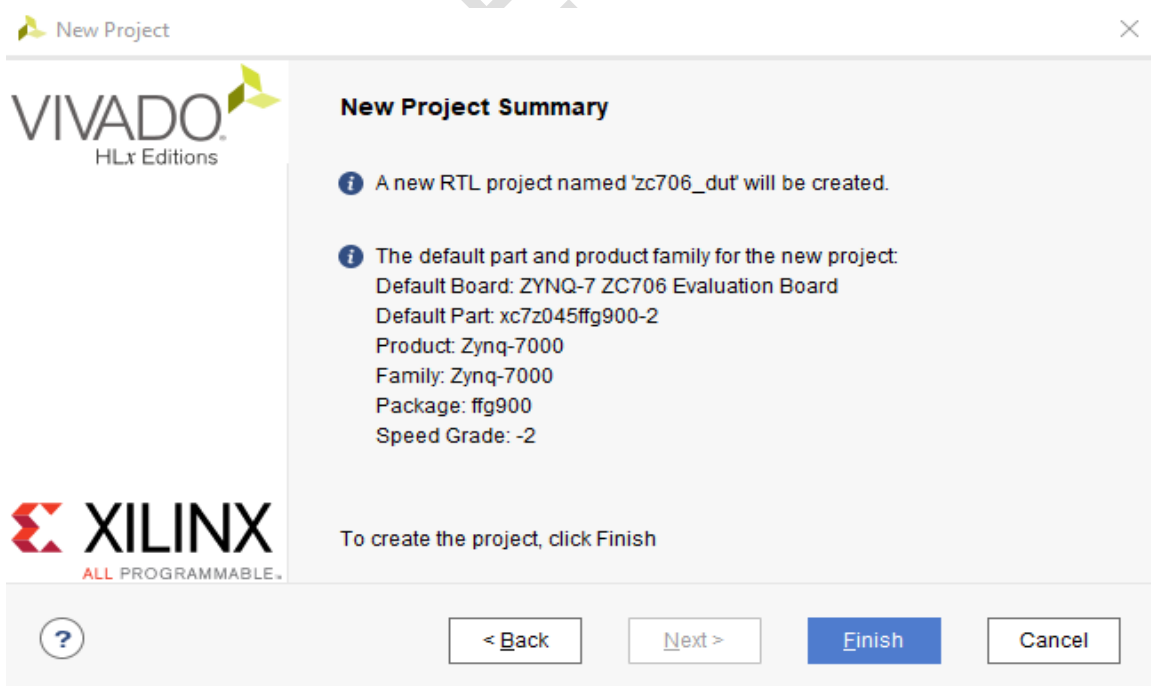


图 2.2.2-5 创建工程

2.2.3 创建最小系统

创建最小系统，目的是搭建适合于特定工程项目的 PS-PL 交互开发平台，可以通过源码形式添加 IP 或者局部通过 Block Diagram 方式添加 IP，基于 BD 方式设计步骤如下：

- 1) 最小系统包括创建顶层文件（top.v），添加用户部分设计文件；
- 2) 创建内核（通过 Block Design 方式），通过 IP 方式添加 Zynq ARM 核心功能，如图（图 2.2.3-1 通过 Block Design 方式设计），添加必要的 Zynq PS 功能选项和配置参数；
- 3) 参数配置部分包括：PS-PL 配置、外围 IO 配置、MIO 配置、时钟配置、DDR 配置、SMC 时许配置和中断配置等几个部分，具体可以通过添加 ZYNQ7 Processing System 的官方 IP 核进行调用，在 Block Design 图中双击“ZYNQ”，在弹出的 Re-customize IP 用户对话框中对相应参数进行配置、启用等；
- 4) 添加用户设计或其他官方 IP，不同类型 IP 需要的许可证或 License 会有所不同；
- 5) 针对 Block 设计选择自动生产代码；
- 6) 如果有需要，需添加 Block 设计到顶层 top.v 文件中。

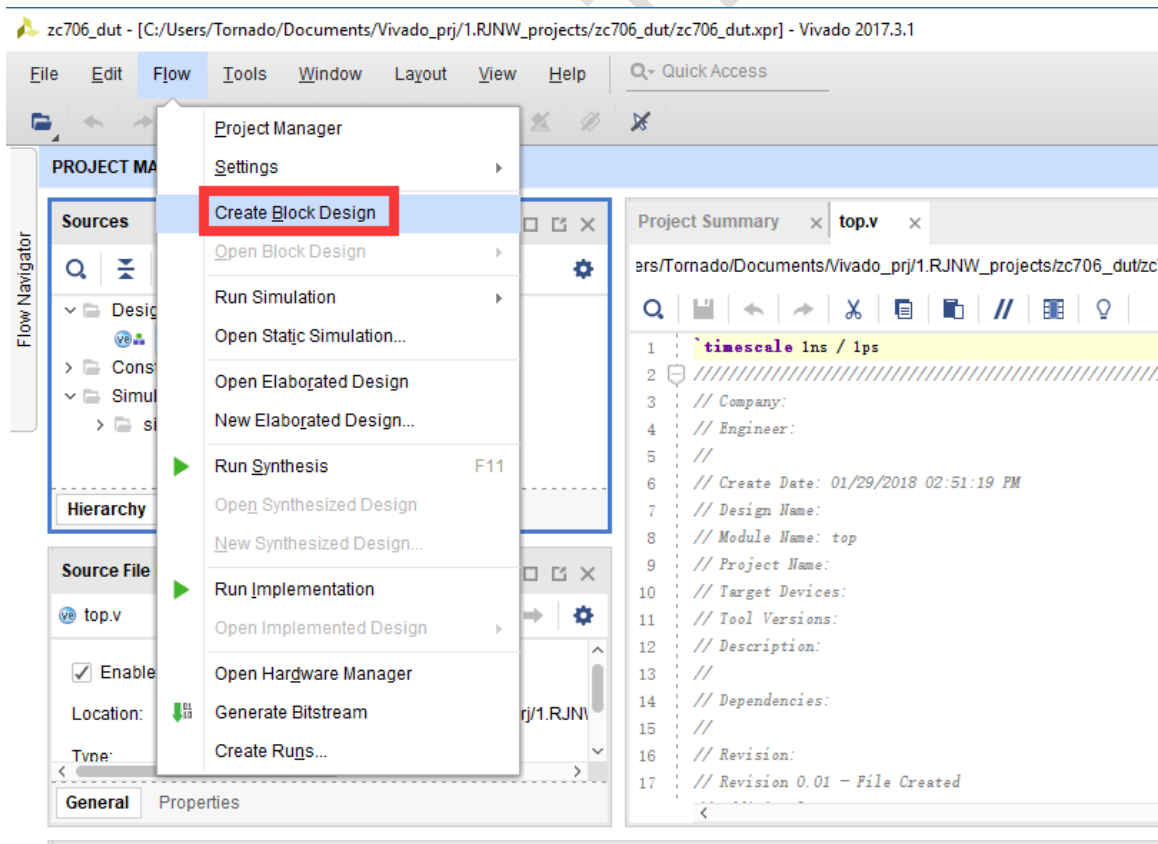


图 2.2.3-1 通过 Block Diagram Design 方式设计

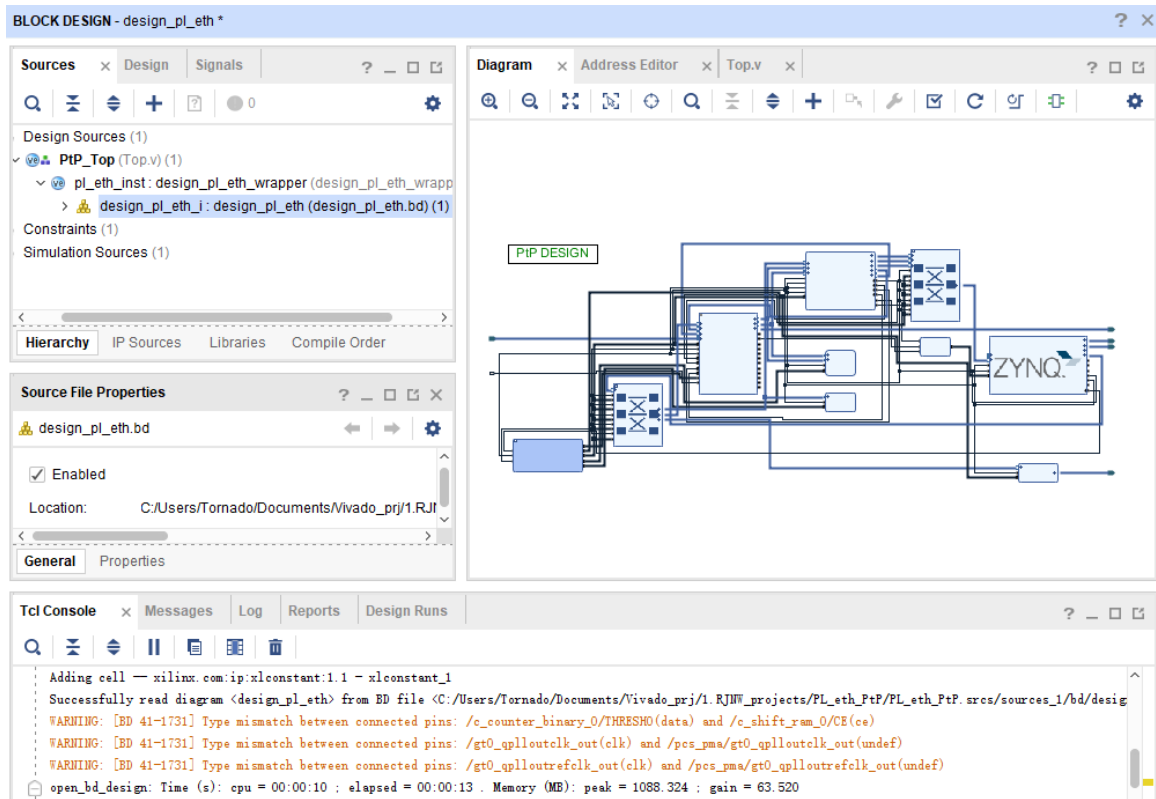


图 2.2.3-2 配置 PS 内核

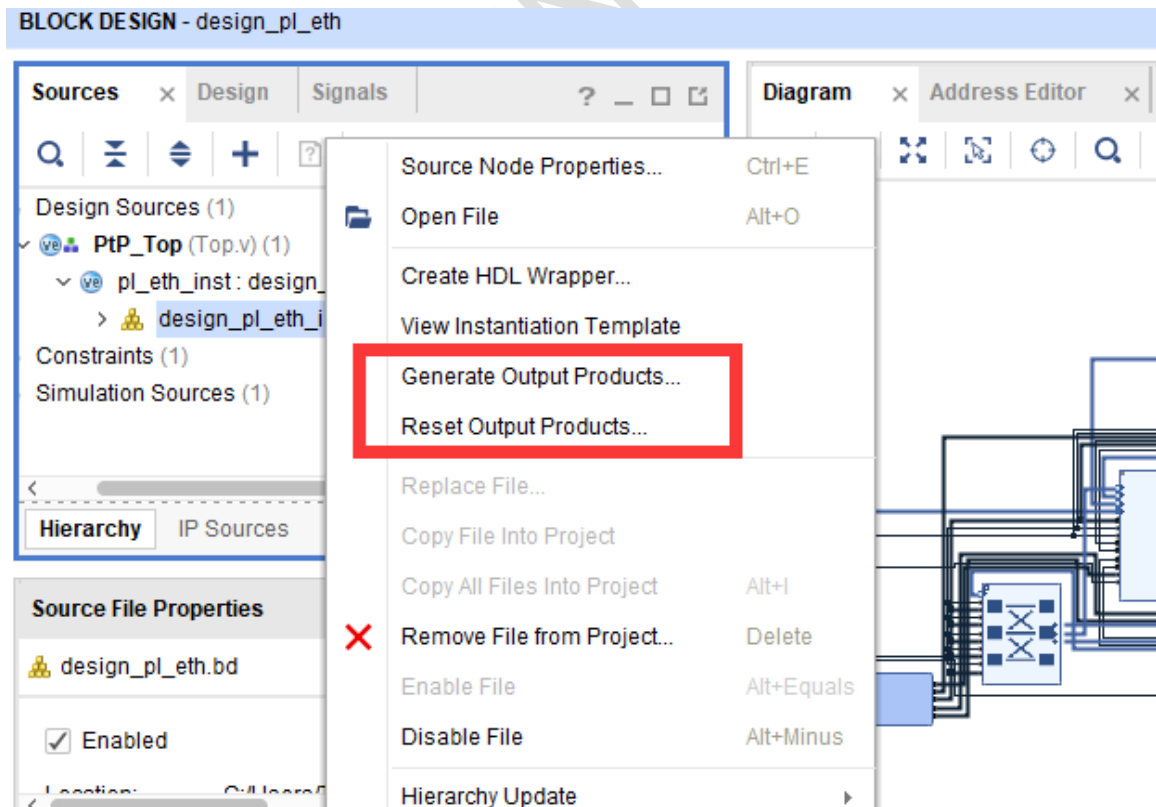


图 2.2.3-3 生成设计

2.2.4 管脚分配和约束

针对管脚分配需要对非 MIO 部分的其他用户 IO 进行相关映射，约束文件服从官方对“*.xdc”文件的规范。“实现”步骤后，系统可用“IO 分配”部署，并归一到相关的“*.xdc”文件。例如，添加外部 LED 的 IO、添加 SFP 的 IO 和外部时钟 IO，设计如下：

```
set_property PACKAGE_PIN AA18 [get_ports sfp_tx_disable]
set_property IOSTANDARD LVCMOS33 [get_ports sfp_tx_disable]

# SI5324 output to MGTREFCLK1
set_property PACKAGE_PIN AC7 [get_ports sfp_125_clk_n]
set_property PACKAGE_PIN AC8 [get_ports sfp_125_clk_p]

create_clock -period 8.000 -name sfp_125_clk_p [get_ports sfp_125_clk_p]

set_property PACKAGE_PIN Y5 [get_ports sfp_rxn]
set_property PACKAGE_PIN Y6 [get_ports sfp_rxp]
set_property PACKAGE_PIN W3 [get_ports sfp_txn]
set_property PACKAGE_PIN W4 [get_ports sfp_txp]

set_property PACKAGE_PIN G9 [get_ports clk_200_n]
set_property PACKAGE_PIN H9 [get_ports clk_200_p]

set_property IOSTANDARD DIFF_SSTL15 [get_ports clk_200_p]
set_property IOSTANDARD DIFF_SSTL15 [get_ports clk_200_n]

create_clock -period 5.000 -name clk_200_p [get_ports clk_200_p]

#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets
pl_eth_inst/design_pl_eth_i/axi_ethernet_1/axi_ethernet_clock/inst/gtrefclk_buf_i]

set_property PACKAGE_PIN A17 [get_ports {led_4bits_tri_o[0]}]
set_property PACKAGE_PIN W21 [get_ports {led_4bits_tri_o[1]}]
set_property PACKAGE_PIN G2 [get_ports {led_4bits_tri_o[2]}]
set_property PACKAGE_PIN Y21 [get_ports {led_4bits_tri_o[3]}]

set_property IOSTANDARD LVCMOS15 [get_ports {led_4bits_tri_o[0]}]
set_property IOSTANDARD LVCMOS25 [get_ports {led_4bits_tri_o[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {led_4bits_tri_o[2]}]
set_property IOSTANDARD LVCMOS25 [get_ports {led_4bits_tri_o[1]}]

set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]

connect_debug_port dbg_hub/clock [get_nets clk]
```

2.2.5 ILA 仿真相关

Zynq 系列可以沿用 Xilinx 对 FPGA 的片内在线逻辑分析仪（ILA）功能，针对于需要监控和分析的信号可以直接通过在源码上添加仿真标准或者在“Design Diagram”中添加“Debug”标志，而插入仿真。

具体步骤如下：

- 1) 添加仿真标志位；
- 2) 配置 ILA 在线分析仪的“Re-customize IP”配置参数。

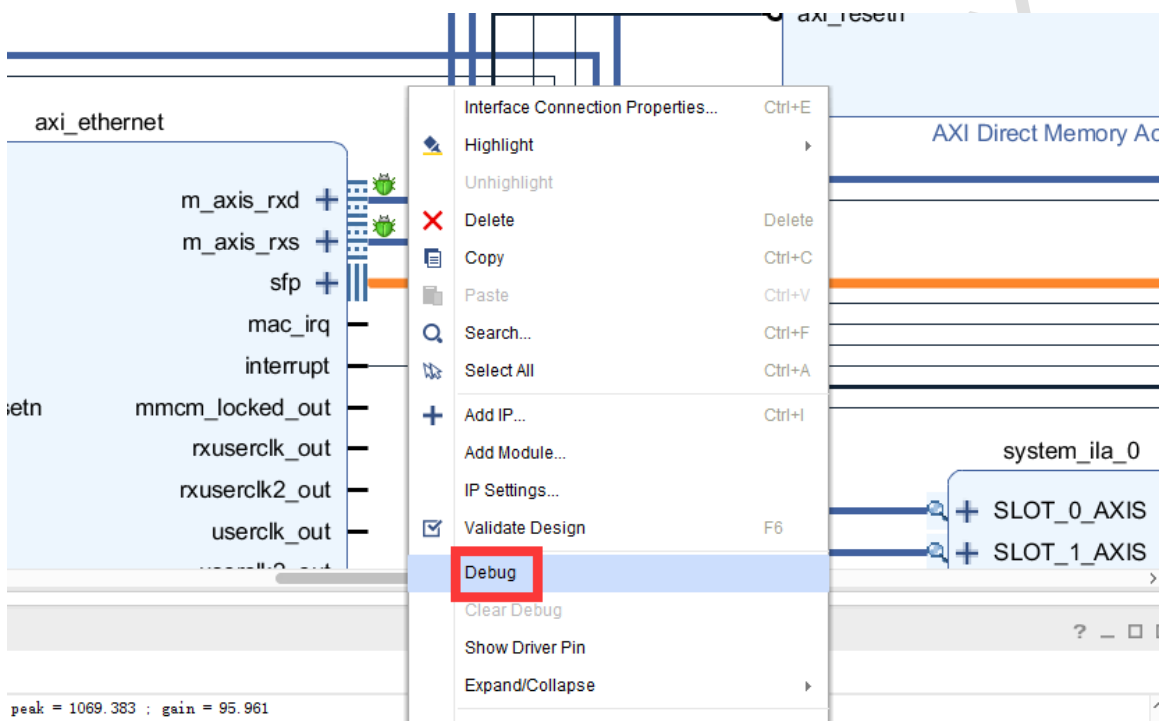


图 2.2.5-1 在 Design Diagram 信号上添加仿真标志

System ILA 即为用户仿真 IP 的调用，在 ILA “Re-customize IP” 配置框内，需要配置信号分析的“Monitor Type”、“Sample Data Depth”、“Trigger”、“Interface”等参数。具体接口类型需要根据信号线或总线类型进行选择，其中普通信号线为 Native 模式、AXI 等总线为 Interface 模式、同时需要分析普通信号线和标准信号总线则为 MIX 模式，其中采样深度和时钟需要按需分配，具体应用请参照《LogiCore IP ChipScope Pro Integrated Logic-Analyzer (ILA) 1.0.4a》手册。

在具体应用的时候要注意 ILA 是需要消耗芯片逻辑资源的，具体 BRAM Slice 使用估计，请在配置完 ILA IP 之后查看 Resource 的“Resource Usage”统计，在前综合之前资源统计具备一定的参考价值，具体请考虑具体项目约束。

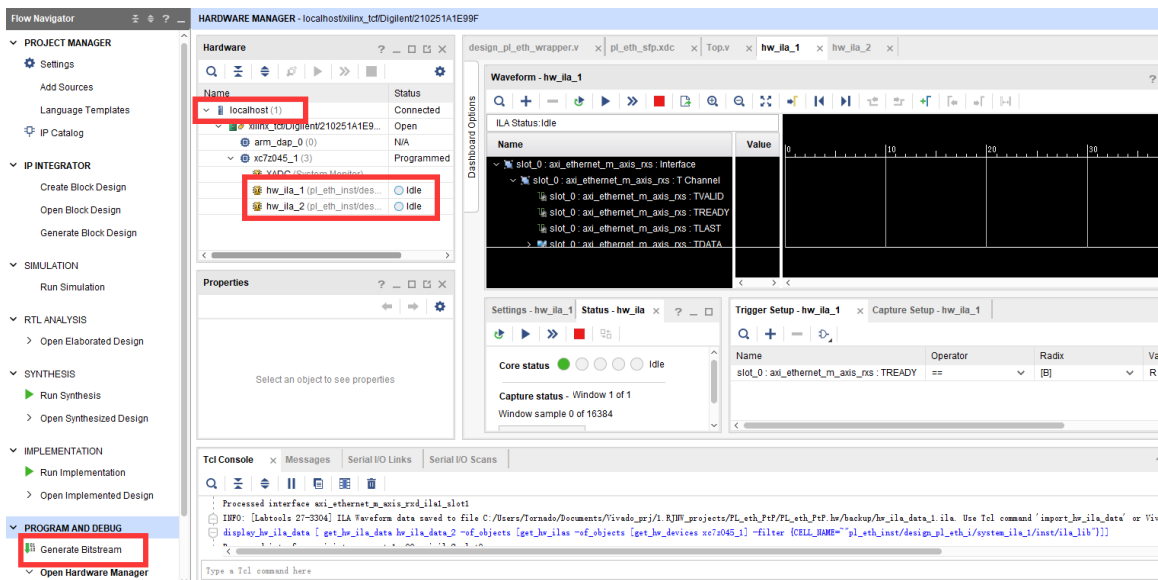


图 2.2.5-2 LIA 在线分析仪

在线逻辑分析仪在写入 FPGA bit 流文件之后，在 JTAG 在线，且写入相应的 PS 固件并启动内核等用户程序之后，可在线生效并进行相关信号的获取等。

2.2.6 仿真

Vivado 的仿真环境具备设计前仿真功能，需要用户自行添加设计的“Test Bench”测试用例框架，对具体的设计和模块进行实现前仿真。也可以导出模型到 Modelsim 软件中进行具体逻辑和模块的外源仿真和分析。

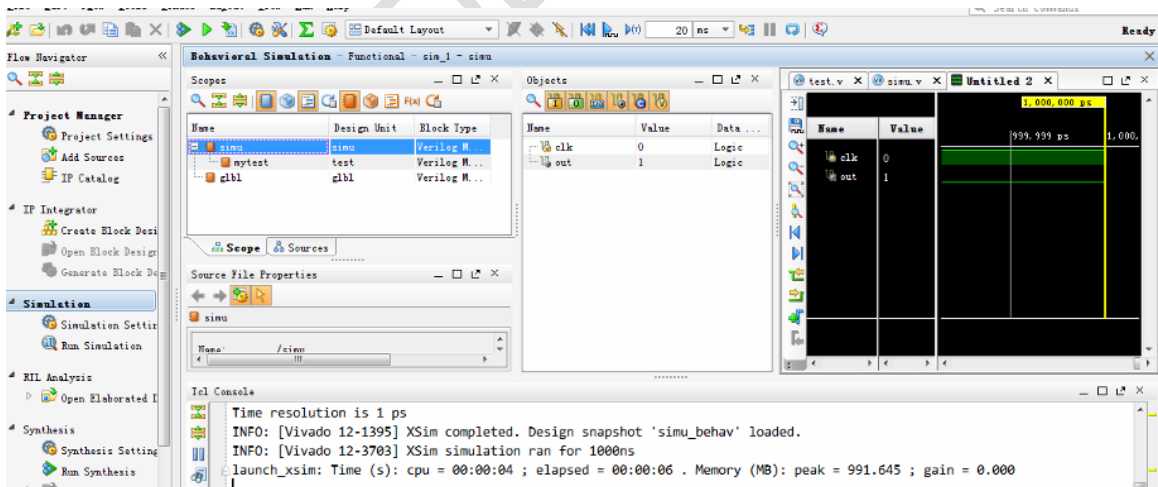


图 2.2.6-1 时序仿真

在“Behavioral Simulation”窗口中的“Scopes”子窗口，根据模块连接关系选中需要查看时序的信号，在右侧“Objects”窗口即可看到选中的所有信号（包括内部信号，即没有写到模块声明语句 module(,)括号等）。

右击信号，选择“Add To Wave Window”，可将波形添加到右侧的仿真波形窗口，保存仿真文件，再次仿真时就可以看到该信号的波形。在波形窗口按住 Ctrl 键并滚动鼠标滚轮，可以横向缩放波形；按住 Shift 并滚动鼠标滚轮，可以横向平移波形。

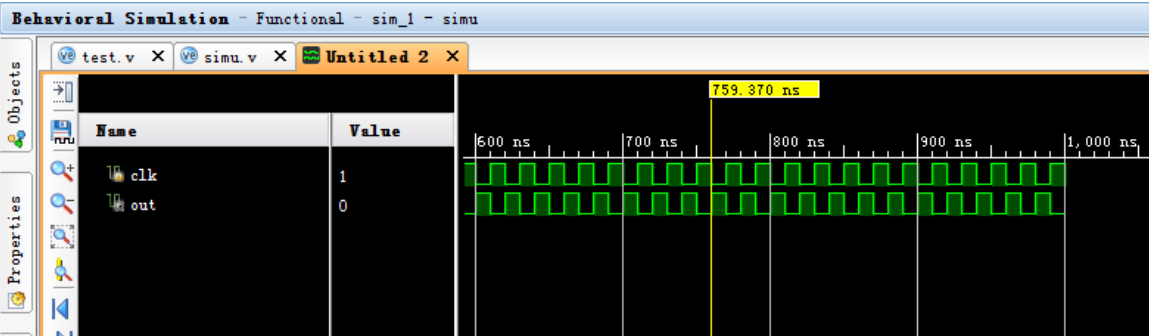


图 2.2.6-2 时序仿真

对于一些输出数字信号波形的情况，例如让 reg [7:0] sina_out 输出正弦波，仿真后右击信号，选择“Waveform Style - Analog”，即可以波形的形式查看信号。如图显示的就是正弦波信号（注意这里信号本身还是数字信号，并不是模拟信号，只是用软件显示出了其幅值随时间变化的波形）。

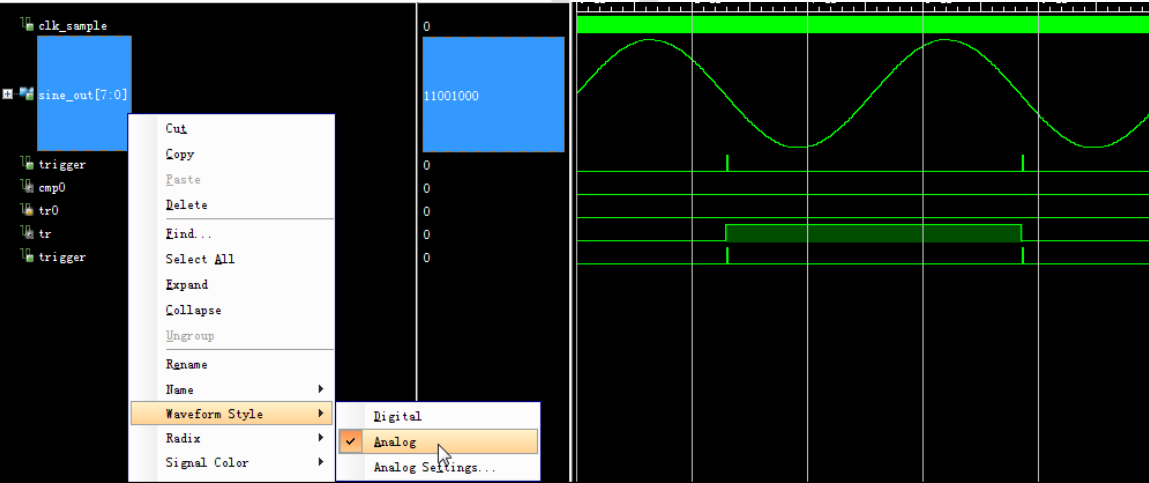


图 2.2.6-3 时序仿真

对于多位信号例如 wire [7:0]p，默认使用二进制形式显示，可以根据需要修改（十进制/十六进制/八进制等选择）。例如，右击选择“Radix - Unsigned Decimal”即可设置为无符号十进制显示。也可以设计相应信号的色彩等，如图（图 2.2.6-4 时序仿真）所示。

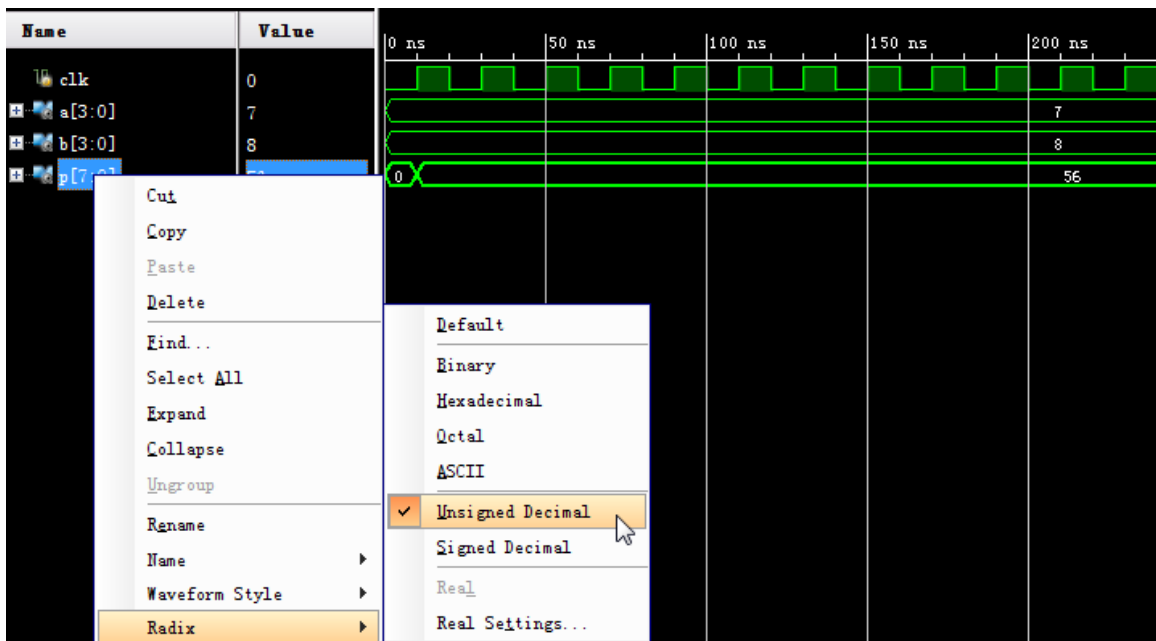


图 2.2.6-4 时序仿真

2.2.7 导出“*.hdf”文件

导出“*.hdf”文件可以用于将用户设计的硬件架构和 FPGA 的“bit stream”文件直接添加文件当中，在应用 Petalinux 设计时候可以应用相应的初始化功能直接生成相应的 FSBL 文件和映射为对应的设备树文件和寄存器列表等，导出方法见下图所示。

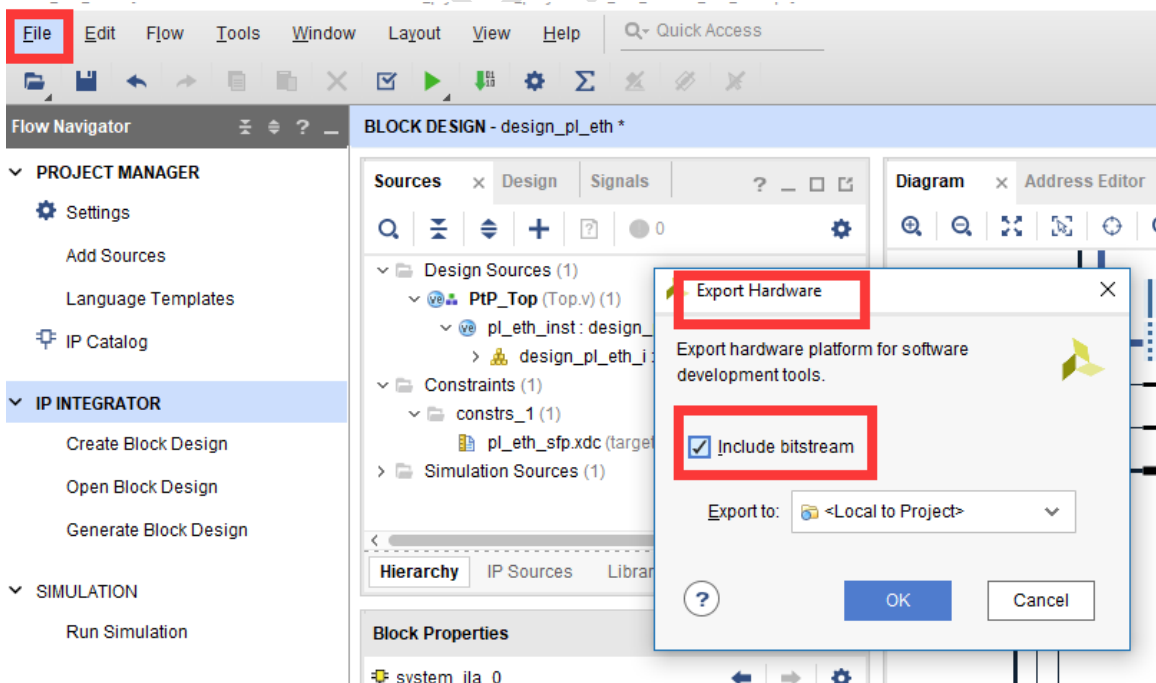


图 2.2.7 导出方法

2.2.8 设计 IP

用户可以通过如下选项将用户设计的模块或功能直接打包成一个私有 IP 共其他上层模块导入和应用，可以添加设计和相应的约束，在打包现有设计选项中选择“打包现在的工程”、“从项目中打包一个 block 设计”或者“指定地址项目”三种模式。

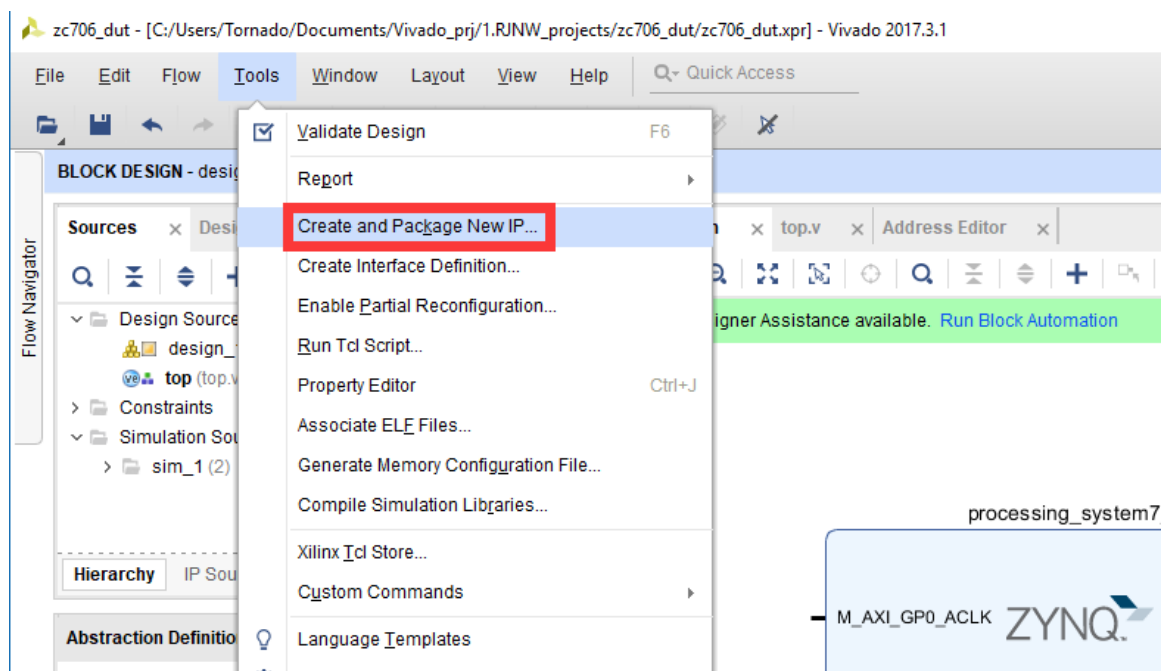


图 2.2.8-1 创建用户 IP

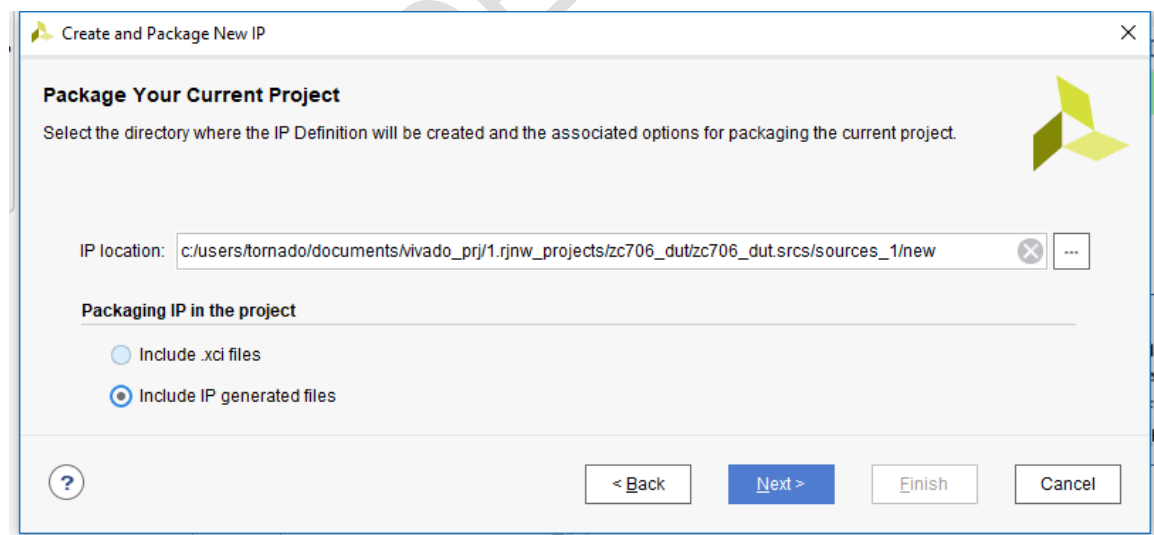


图 2.2.8-2 打包设计并创建为 IP

其中，也可创建带“AXI4”外围总线的用户私有 IP，其中创建类型包括带有 AXI4 总线接口的 AXI4IP、驱动、测试软件接口、集成 AXI4 VIP 的仿真和调试设计模板等，具体如图（图 2.2.8-3 创建 AXI4 接口 IP）所示。

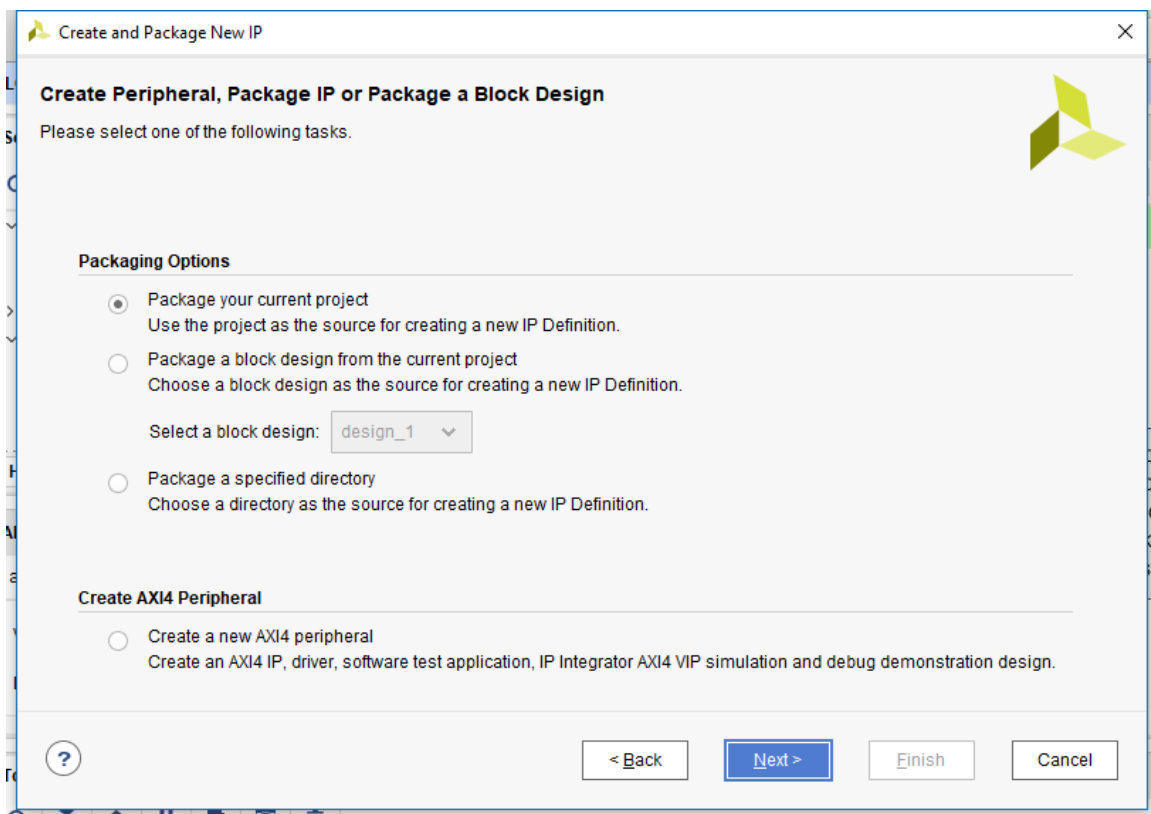


图 2.2.8-3 创建用户 IP 类型选择

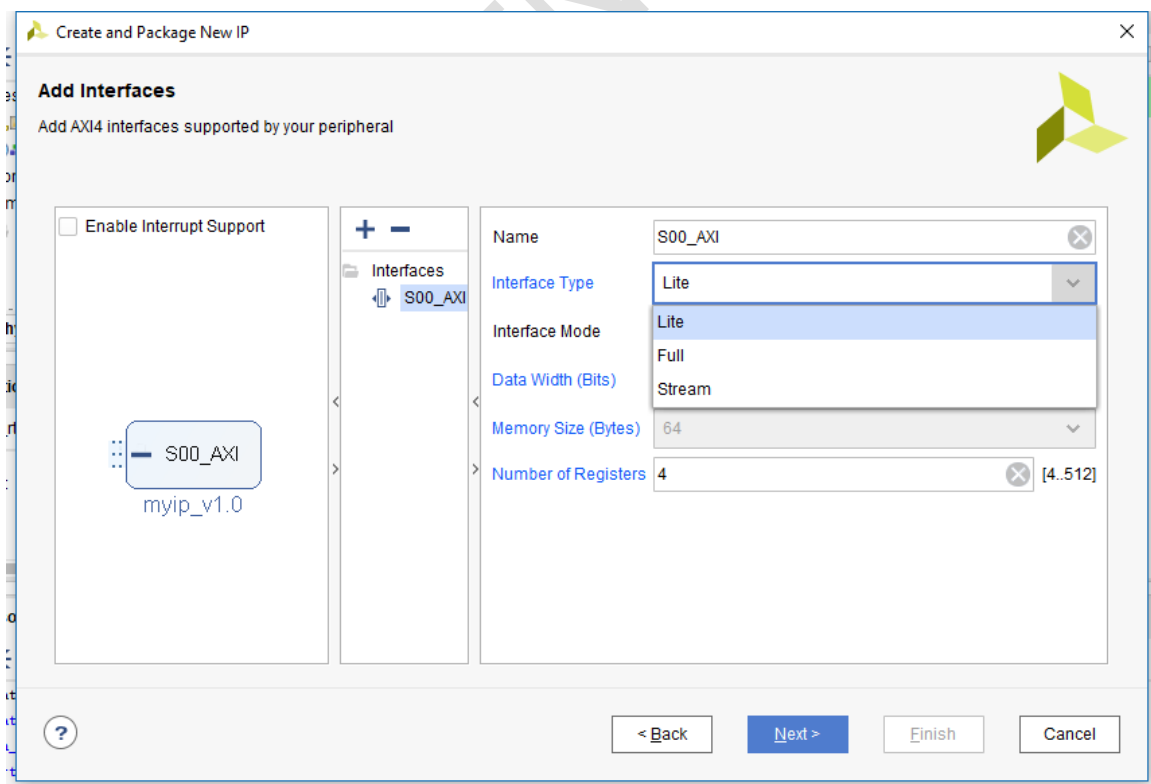


图 2.2.8-3 创建 AXI4 接口 IP

2.2.9 Synthesis

Vivado 自带综合工具可以对中小规模逻辑进行有效综合和分析，综合部分包括设计综合、时许分析报告、时许约束设置、DRC 报告、Noise 报告、功耗报告等。其中，时许相关需要关注如下几个报告，并对应于收敛情况进行相应的信号或者模块的时许约束文件和条款的创建。

No timing reports are available for display. To report timing click one of the following links:

Check Timing

Check the design for possible timing problems.

Report Timing Summary

Generate a timing summary to understand if the design met timing.

Report Timing

Generate a timing report.

Report CDC

Generate a clock domain crossings report.

Report Pulse Width

Reports the pulse width of the specified clock signals in the clock network and upon reaching the flip-flop.

图 2.2.9-1 时许工具

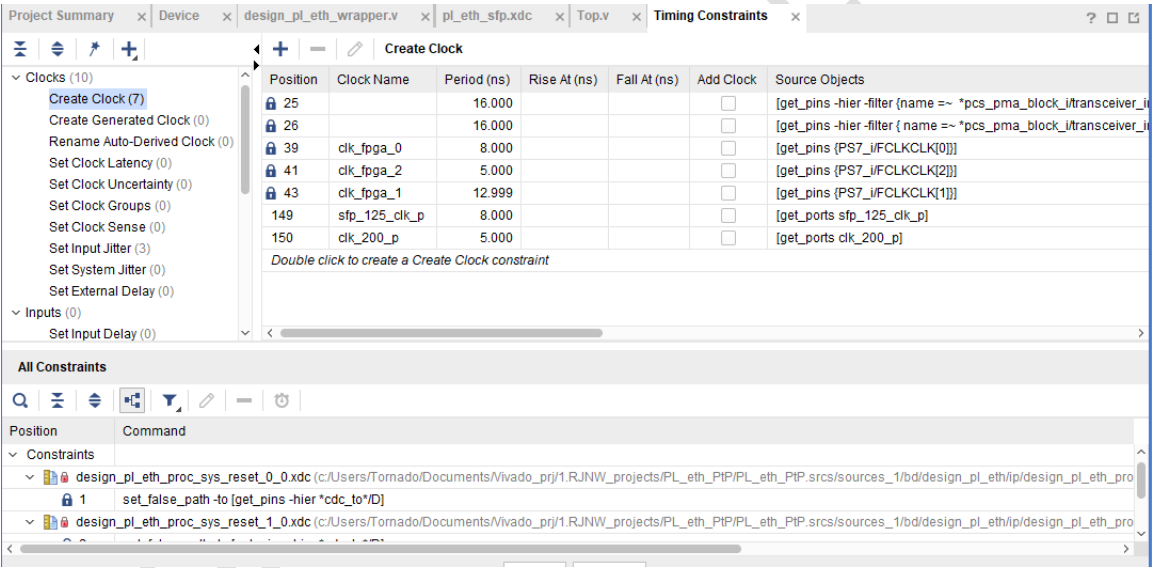


图 2.2.9-2 时许约束

约束类五种主要方式，包括：Clocks 相关、IO 相关、Assertions 相关、Exceptions 相关、其他，具体约束类型归总为下表（表 2.2.9-1）。

表 2.2.9-1 约束类型

序号	类型	子类
1	Clocks	Create Clock
		Create Generated Clock
		Rename Auto-Derived Clock

		Set Clock Latency
		Set Clock Groups
		Set Clock Sense
		Set System Jitter
		Set Externl Delay
2	Inputs/Outputs	Set Inputs/Outputs Delay
3	Assertions	Set Data Check
		Set Bus Skew
4	Exceptions	Set Case Analysis
		Set False Path
		Set Multicycle Path
		Set Maximum Delay
		Set Minimum Delay
5	Other	Group Path
		Set Disable Timing

2.2.10 Implementation

应用模块可以完成“综合之后”对相应设计进行FPGA片上映射估算后得到跟切合实际硬件平台的各类报告，包括时许、网络、时钟、DRC、噪声个、功耗等，同时可以输出逻辑模块的原理图供设计原理分析，具体约束可以应用“Constraints Wizard”约束向导进行设计。

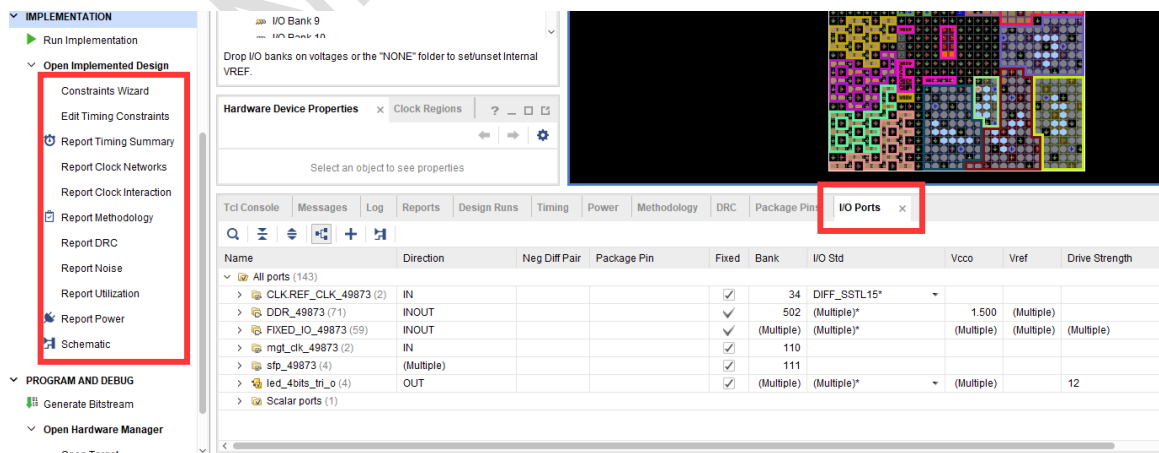


图 2.2.10 应用模块

另外，在应用设计结果可以重新对 IO 进行映射和类型确认，并最终导入到归一化的“*.xdc”文件中，如图（图 2.2.10 应用模块）所示。

2.2.11 Project Summary

这部分统计了整个项目在指定的约束策略下整体的综合数据，其中包括了“Synthesis”状态和情况、“Implementation”状态和分别在“Post Synthesis”和“Post Implementation”情况、“DRC”、“Timing”、“Utilization”资源使用估计、“Power”片上功耗估计等。

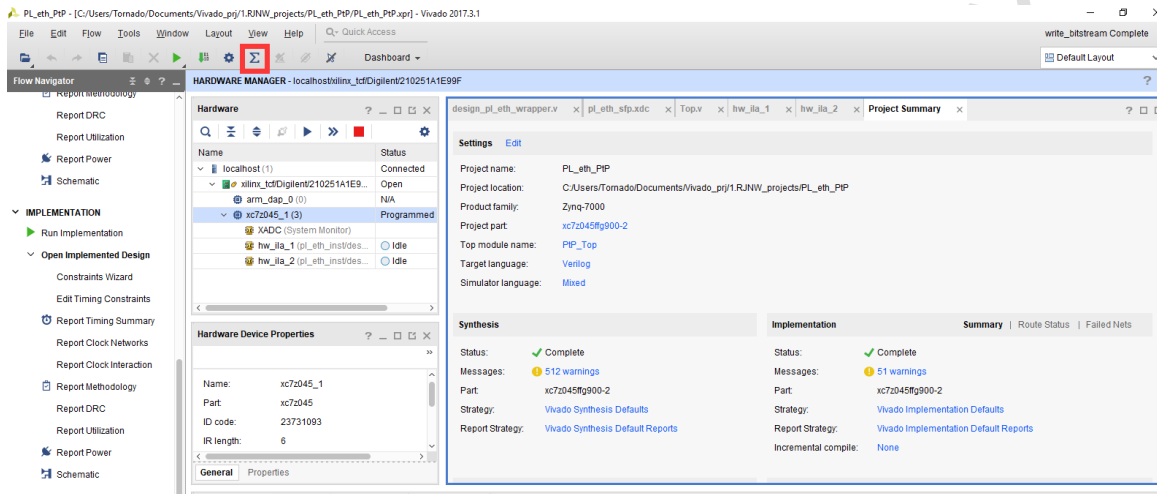


图 2.2.11 项目统计

2.2.12 Program & Debug

烧写和仿真部分较为简单，主要包括通过 JTAG 方式或网络方式烧写 BIT 文件到 FPGA 或 FLASH 之中、添加可配置 Memory 等几个环节，支持在线逻辑分析和 XADC 监控等。

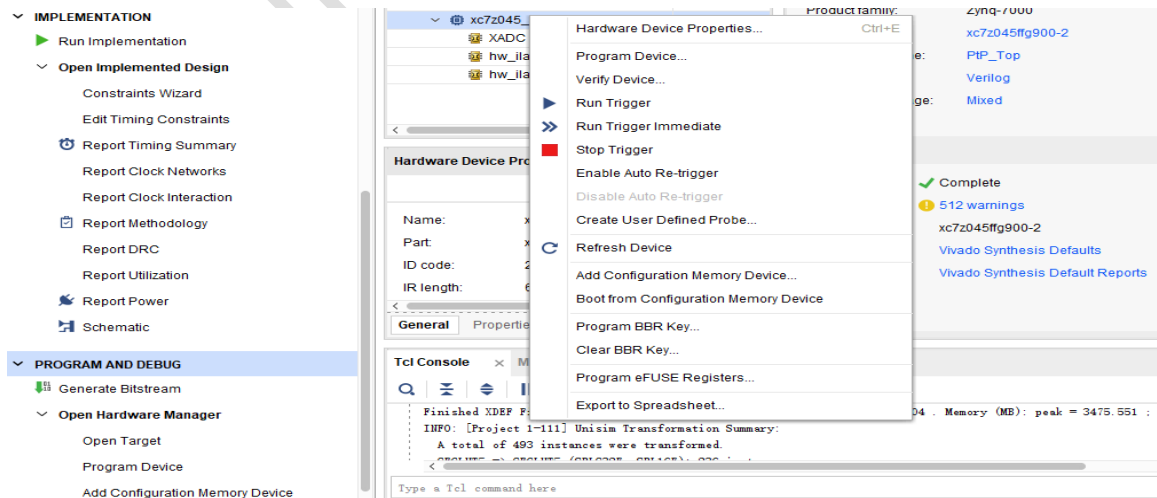


图 2.2.12 编程和仿真 FPGA

3 参考文件

- [1] Petalinux Tools Documentation: Workflow Tutorial
- [2] Petalinux Tools Documentation: Reference Guide
- [3] Petalinux Tools Documentation: Command Line Reference Guide
- [4] Xilinx Quick Emulator: User Guide
- [5] OpenAMP Framework for Zynq Devices: Getting Started Guide
- [6] Vivado Design Suite Release Notes, Installation and Licensing Guide
- [7] Vivado Design Suite User Guide: Design Flows Overview
- [8] Vivado Design Suite User Guide: Using the Vivado IDE
- [9] Vivado Design Suite User Guide: Designing with IP
- [10] Vivado Design Suite User Guide: High-Level Synthesis(HLS)
- [11] Vivado Design Suite User Guide: Implementation
- [12] Vivado Design Suite User Guide: Programming and Debugging