在 fpga 上的运行裁剪好的 yolov2 模型的实验步骤

配置信息

开发板型号：alinx 7020,处理器型号：xc7z020CLG400，开发板系列：zynq-7000
petalinux 版本：2017.4
Ubuntu 版本：16.0.4
vivado 和 vivado HLS 和 vivado SDK 均为 2017.4
yolo 版本：v2(非 tiny)
裁剪好的模型源码：https://github.com/dhm2013724/yolov2_xilinx_fpga

一 ：SDK 的生成

这一步的目的是为了生成配置好的硬件信息，供后面的 elf 开发使用。
Q：为什么不能直接在开发板上运行 FPGA 程序？
A：可以，前提是必须在这一步中提供 PL-PS 的通信时钟设置，很不幸，开发板原有的并没有开启该设置，所以不行。
Q：那如果开启了该设置，是不是就可以了？
A：不行，自定义的 IP 核配置的引脚必须要明确给出，即使预留了固定数量的端口，还是不能明确引脚的设置，所以必须要在这一步中生成。

安装 Ubuntu 的过程不再赘述，vivado 系列的步骤不再赘述：参考 https://blog.csdn.net/wmyan/article/details/78926324 即可

1.1 生成 IP core：

需求：vivado hls，IPcore 的源码

首先，打开 hls：打开终端，然后 source <PATH-TO-XILINX>/setting64.sh 使配置文件生效，然后输入 vivado_hls,就可以运行 hls 了。

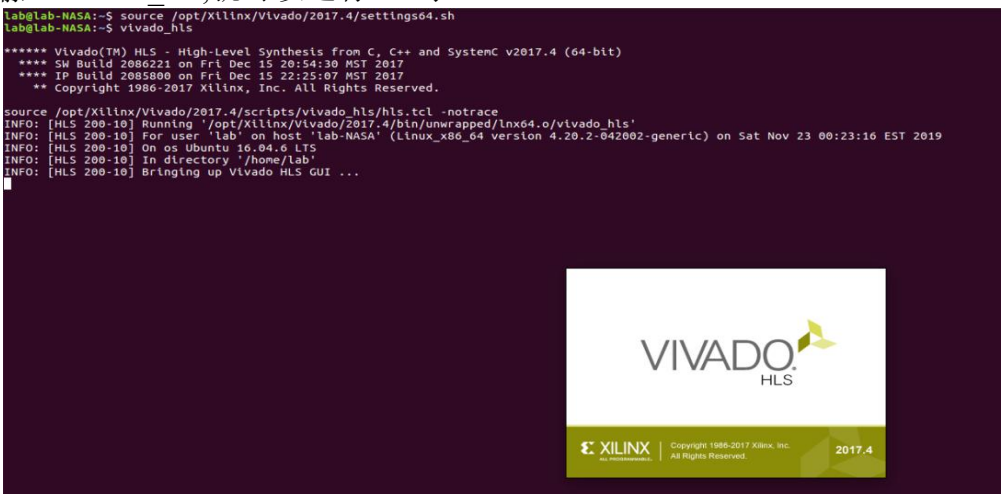

图 1.1 HLS 的运行

其次，需要新建一个工程，点击 create new project，然后输入工程名字等一系列设置，不再赘述，只有一条，**在最后要设置开发板型号和处理器型号。**

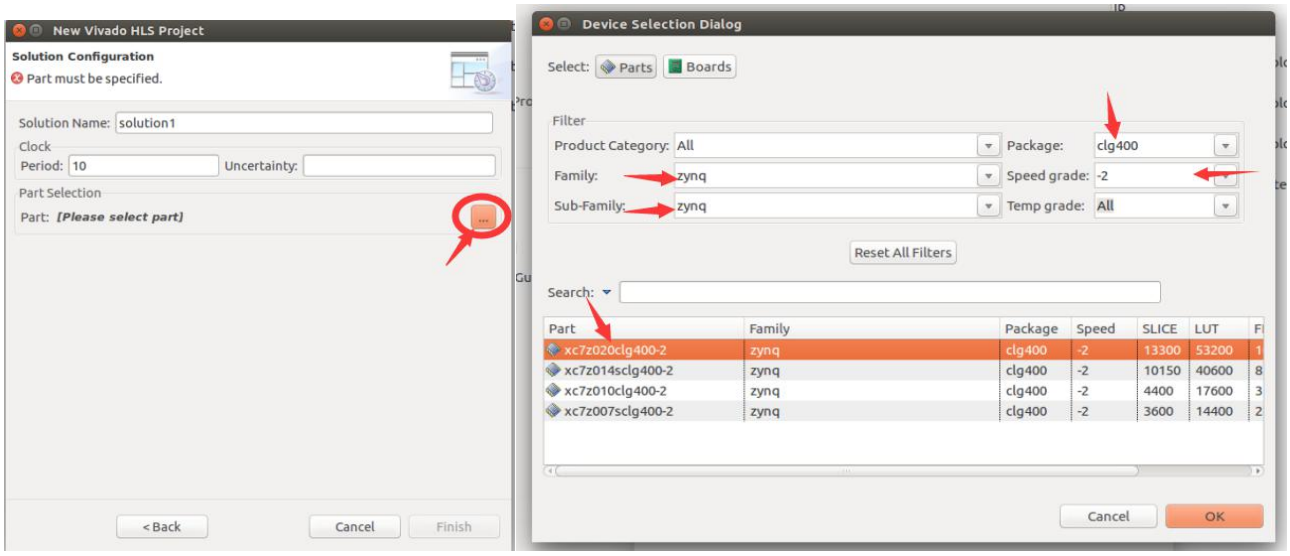

图 1.2 设置开发板类型

完成后点击 finish 即可创建工程。

然后在 hls 工程下的 source 中右击，add/new file 均可，

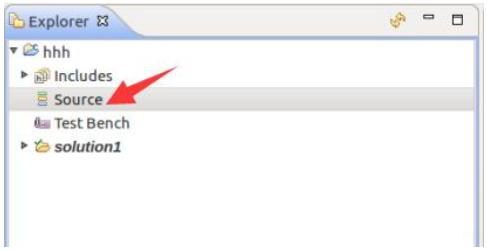

图 1.3 添加文件

但是**注意**，addfile 必须保证文件已经复制进工程目录内，hls 不会主动复制文件(这一点和 vivado 完全不同)，在非工程目录下编译会报错(file “xxx.h” cannot found) 其中要添加的文件在<PATH-TO-MODEL>/yolov2_xilinx_fpga/hls/src 下面，添加好后，点击上面的 run c synthesis，

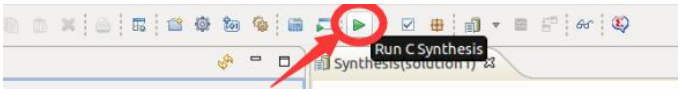

图 1.4 HLS 综合按钮

然后等待一个相当长的时间后(前提是不报错，我只是因为 addfile 出了错，这一步并没有)，会生成一份报告，里面有这个程序对板子资源的利用情况，

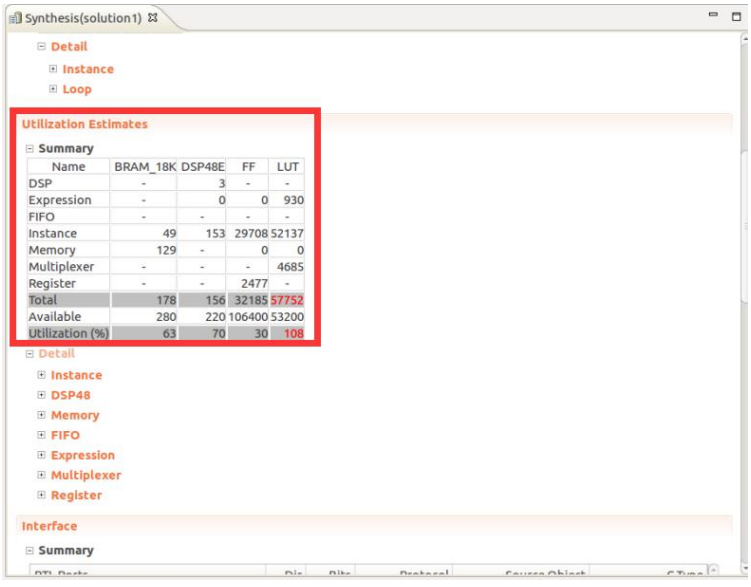

图 1.5 HLS 综合结果

无错误后证明可以生成 IPcore。最后点击上面的小方块一点的东西(export RTL),然后选择相应的设置，等待一个更长的时间，最后生成的文件就是我们需求的 IPcore(这里也没有报错)。

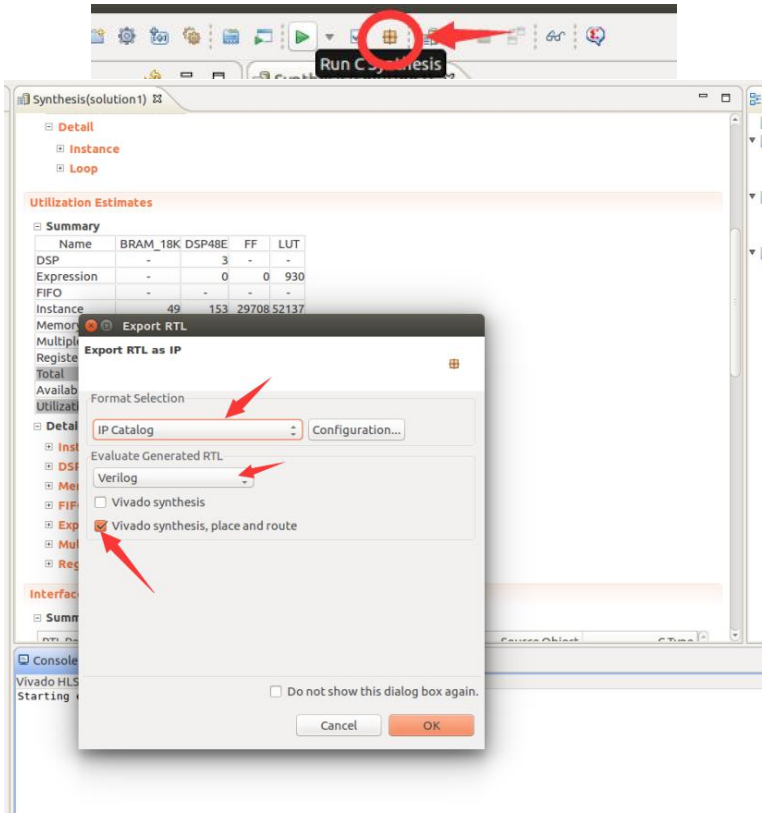

图 1.6 导出 IP 核

然后就可以关闭 hls，同时**记住**你的 ipcore 保存的位置。

注意：运行时需要设置 Topfunction，本工程的设置为 yolov2_fpga

1.2 生成 blockdesign：

需求：上一步生成的 IPcore，别人连接好的 blockdesign 样本，给出的 blockdesign 配置使配置生效后运行 vivado 与建立工程不再介绍，同 1.1 需要进行 board 配置

图 1.7 vivado 工程设置

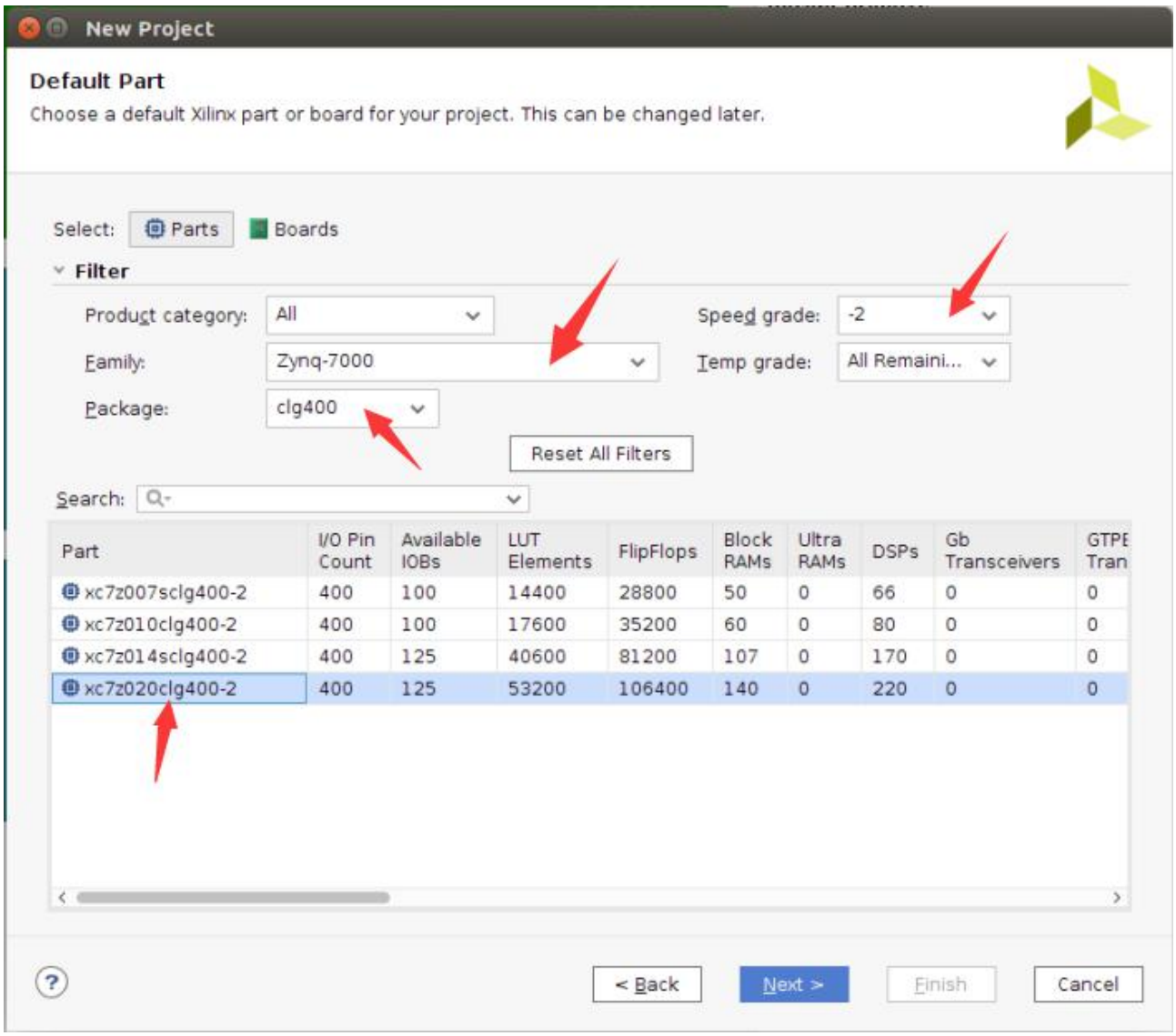

图 1.8 vivado 工程的开发板型号设置

工程设置完后，等待一小会，进入工程主界面。

点击 ip catalog 在右边的 IPcore 内，右击任意一个 IPcore，点击添加 ip 后，找到在上一步保存好 IPcore 的文件夹，vivado 会提示你导入了几个 ipcore，本项目只有一个自定义 IPcore，其余是系统自带，成功导入后进行下一步。

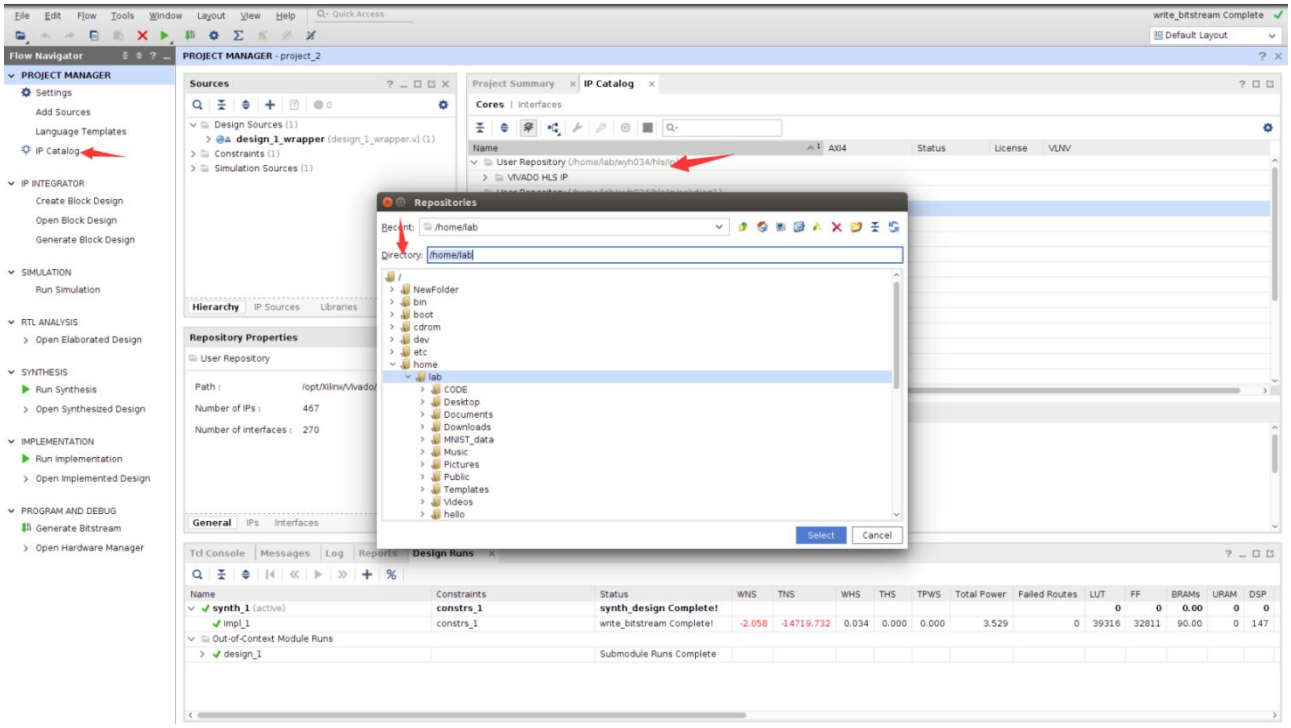

图 1.9 导入 IP 核步骤

然后点击 blockdesign 按钮，点击加号添加 IPcore，按照作者的图添加，可以按照名字搜索，这块不用赘述。

然后根据作者提供的连线图，连接自己的设置。

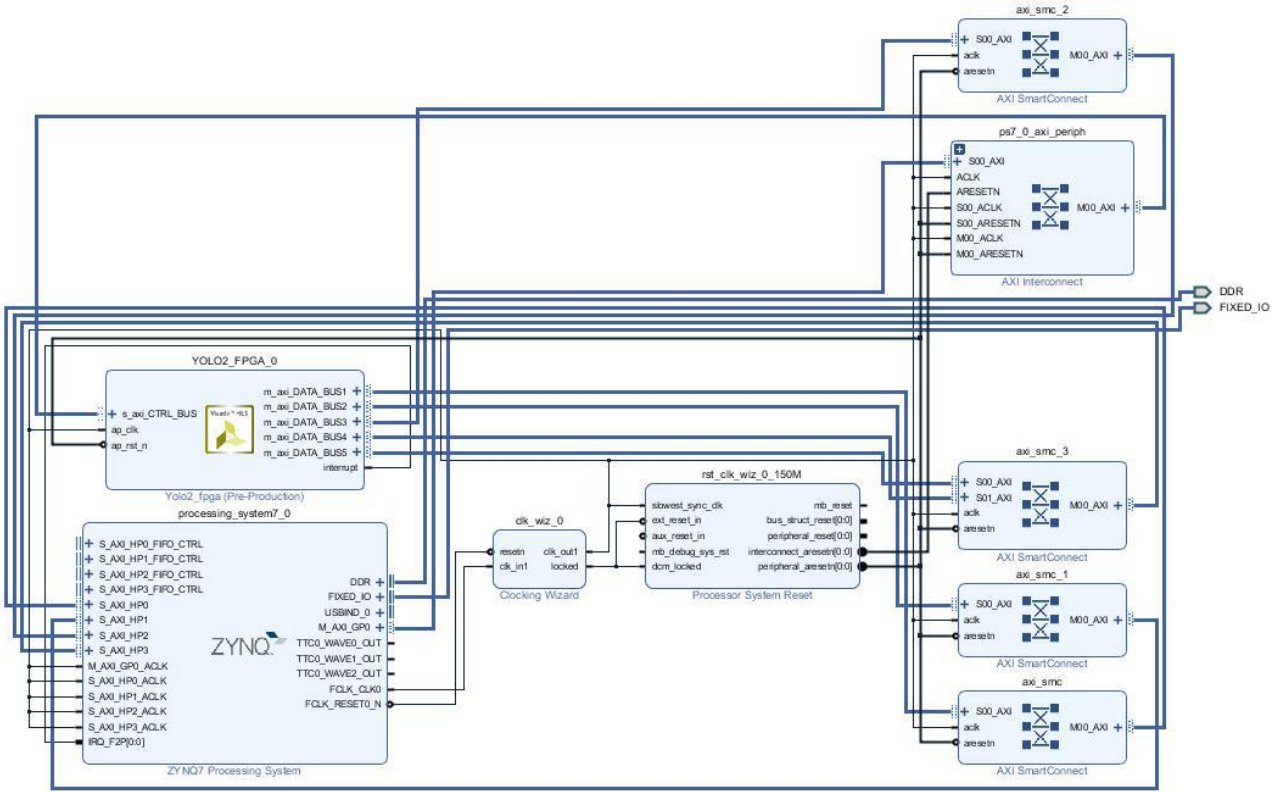

图 1.10 作者提供的连线图

注意事项 1：连线一定要仔细认真，认真核查，在这里就因为没有连接 IPcore 上的复位与整个系统其他的复位线而报错误。

注意事项 2：有关一些具体的配置一定要注意：首先是时钟设置，也就是图中的 clk-wiz 和 rst-clk-wiz 输出时钟为 150MHZ，输入为 100MHZ。

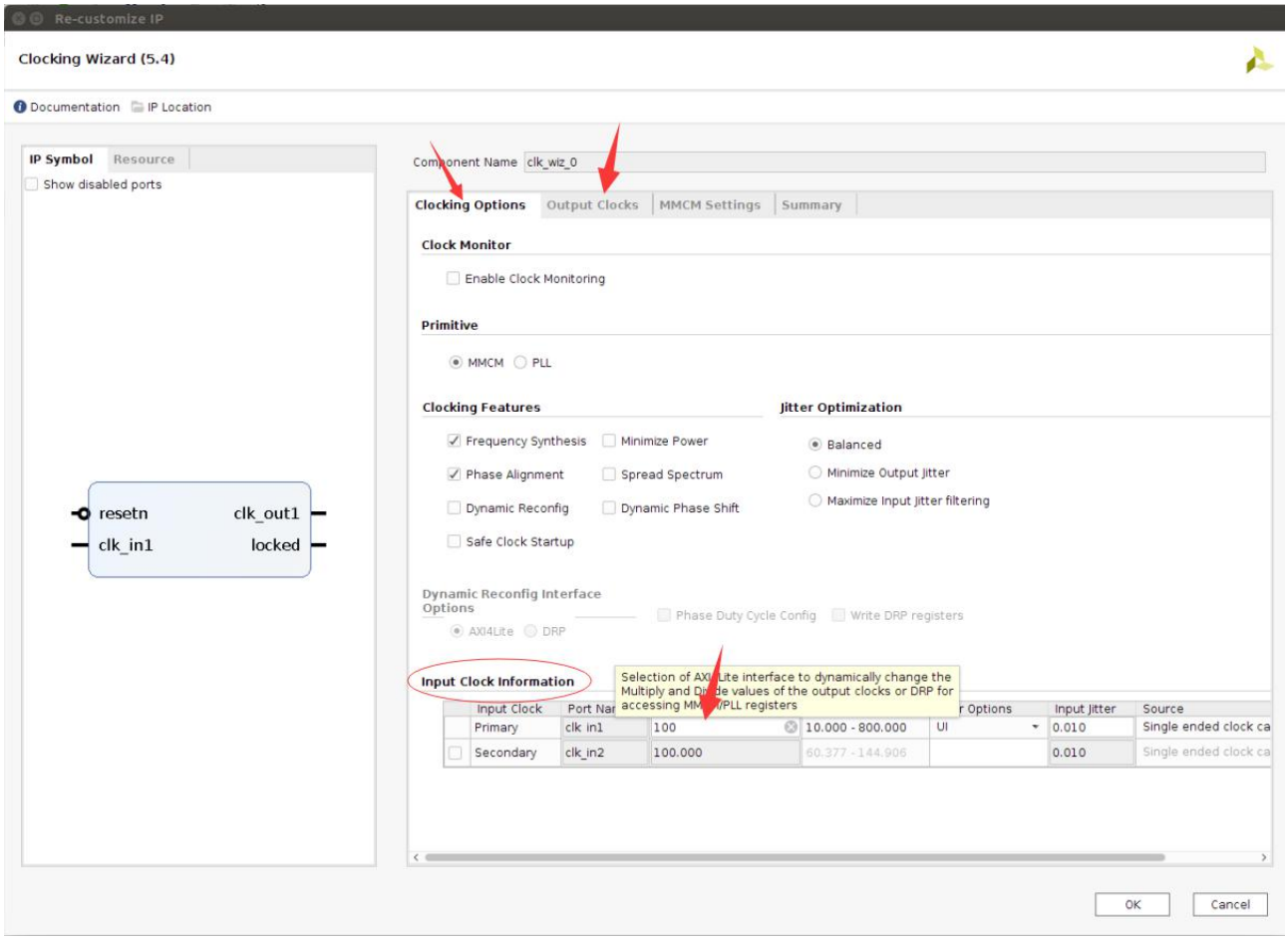

图 1.11 整体系统时钟配置

其次是 zynq-7020 板子具体内容的设置，主要有

1. 2. 1 引脚配置

点击 peripheral io pins，打开相应的引脚

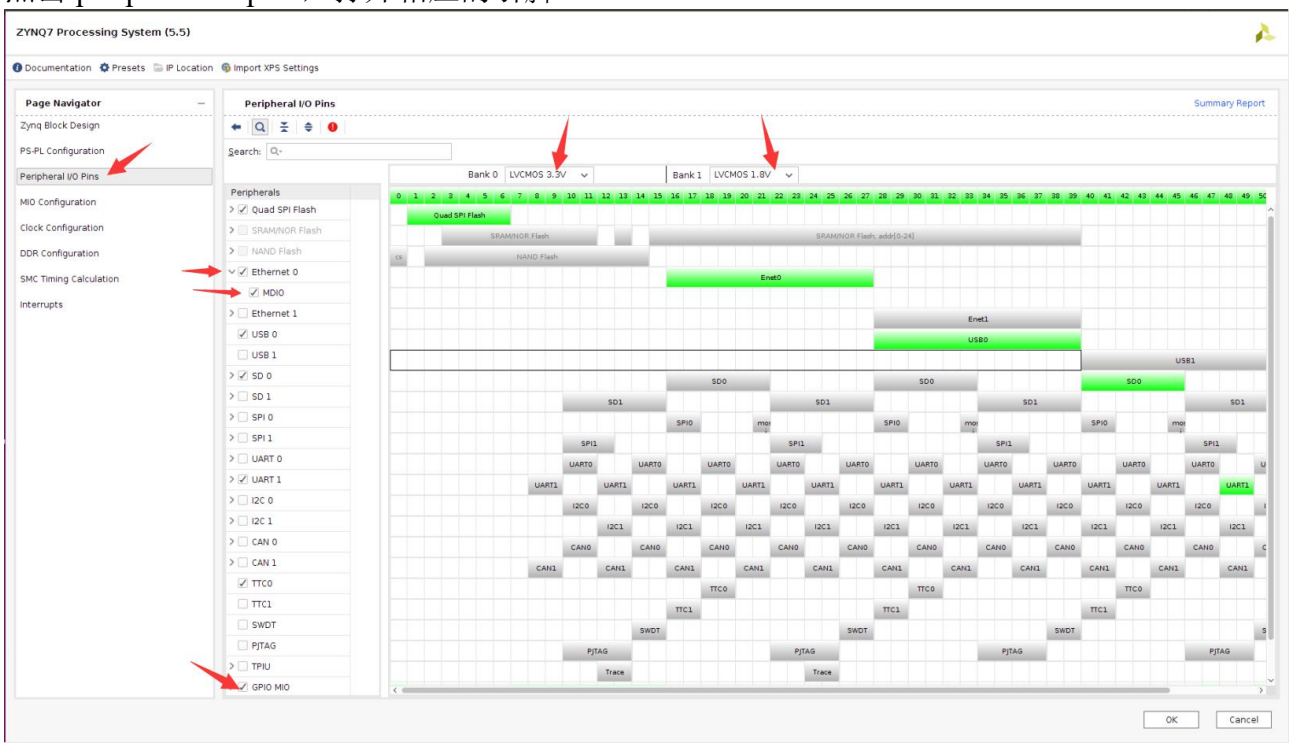

图 1.12 开发板引脚复用设置

这里注意，在导入 zynq 设置后，引脚没有作者图片的那么多，只有开启相应的该设置后才可以进行配置

1. 2. 2 时钟配置

这里别的不再赘述，保持默认或者升高频率都可以，只有一条：注意 requested frequency 和 actual frequency 的区别，过高的 requested Frequency 会导致 actual 不能达到相应的频率，FCLK 设置为 100MHZ

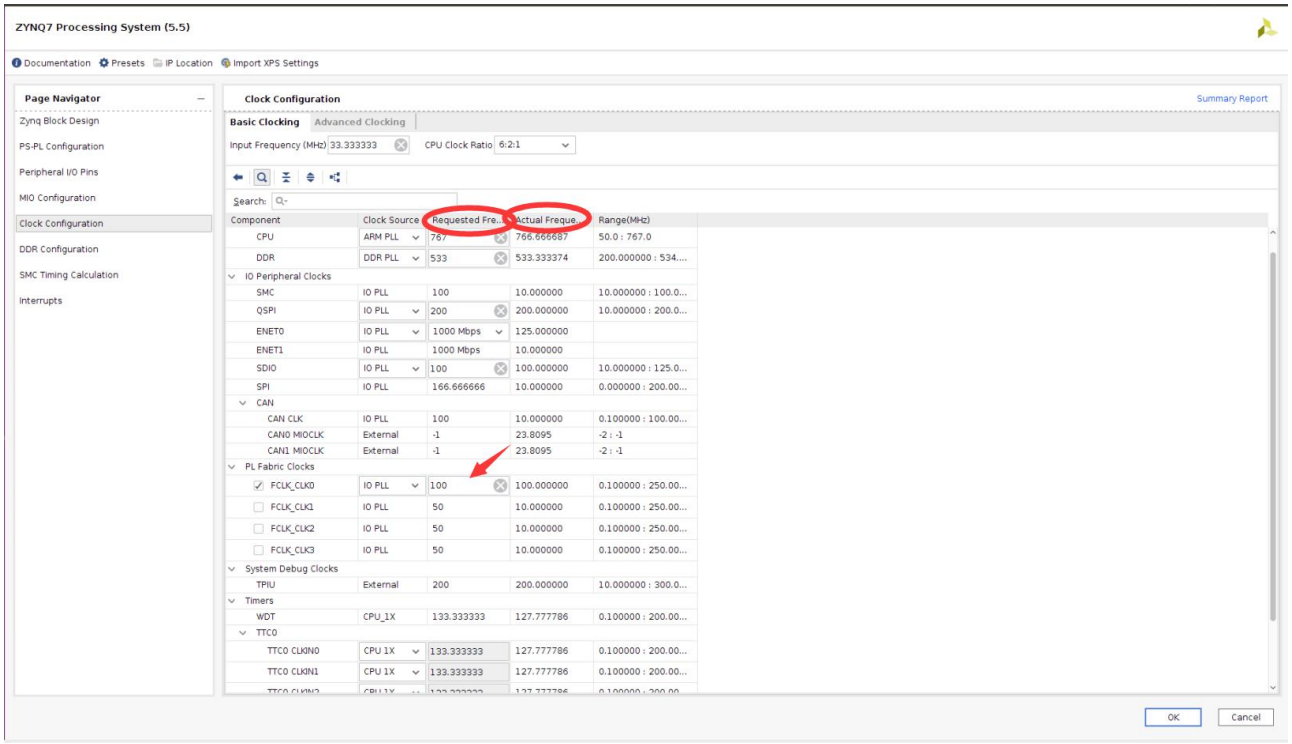

图 1.13 开发板时钟设置

1. 2. 3 内部时钟及定时器配置

这一步同 1 类似，开启相应设置即可，注意一点，数据总线位宽为 32。

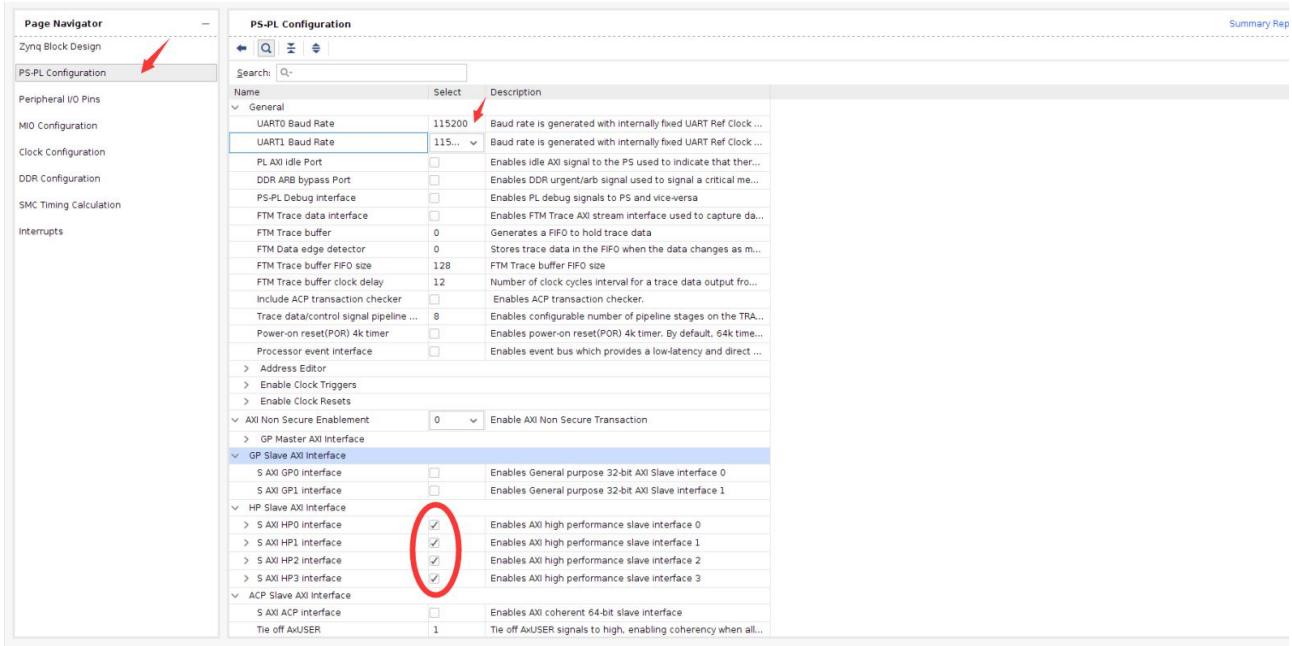

图 1.14 互联引脚设置

1. 2. 4 外设配置。

这里又分为三部分，mio 配置，ddr 配置，中断配置，后两者按照板子设置和作者图片给出的开启相应的配置即可，主要问题存在于 mio 配置。

在 mio 配置中，调整电平 bank1 为低电平 LVCMOS1.8V，只有 ethernet 的配置要**注意**，需要手动配置为 iotype 为 HSTL1.8V，speed 为 fast（别的都是 LVCMOS 和 low），**注意注意**，此问题极其严重，配置失败会导致不能开机!!!

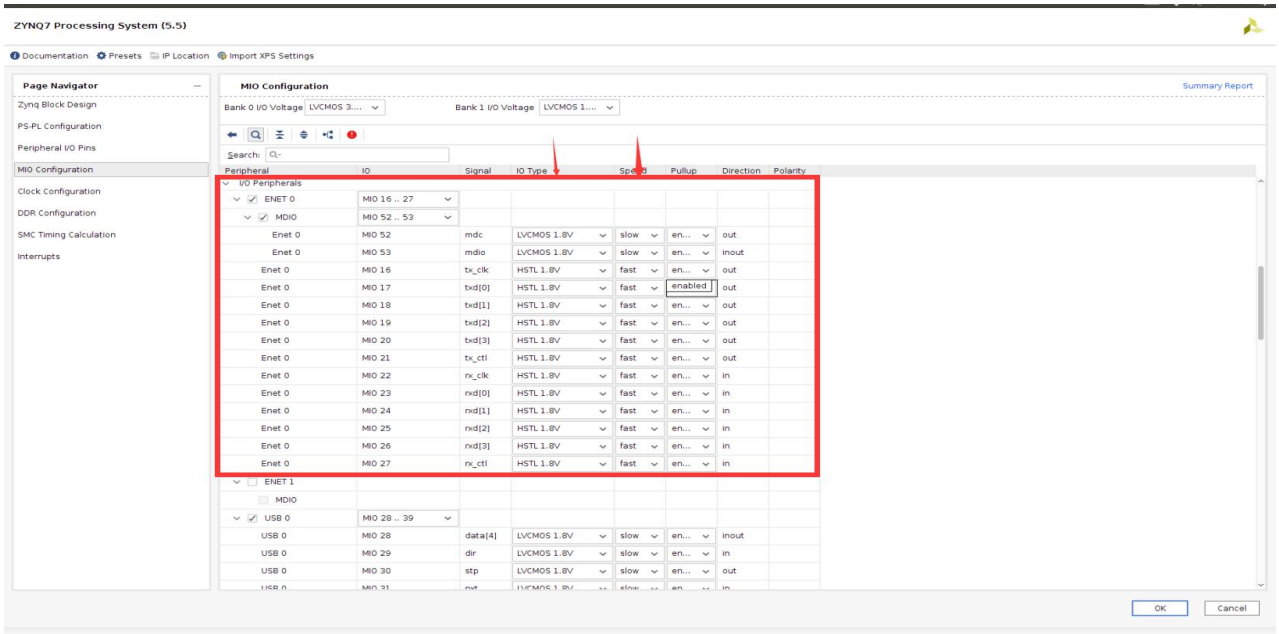

图 1.15 Ethernet 的具体设置

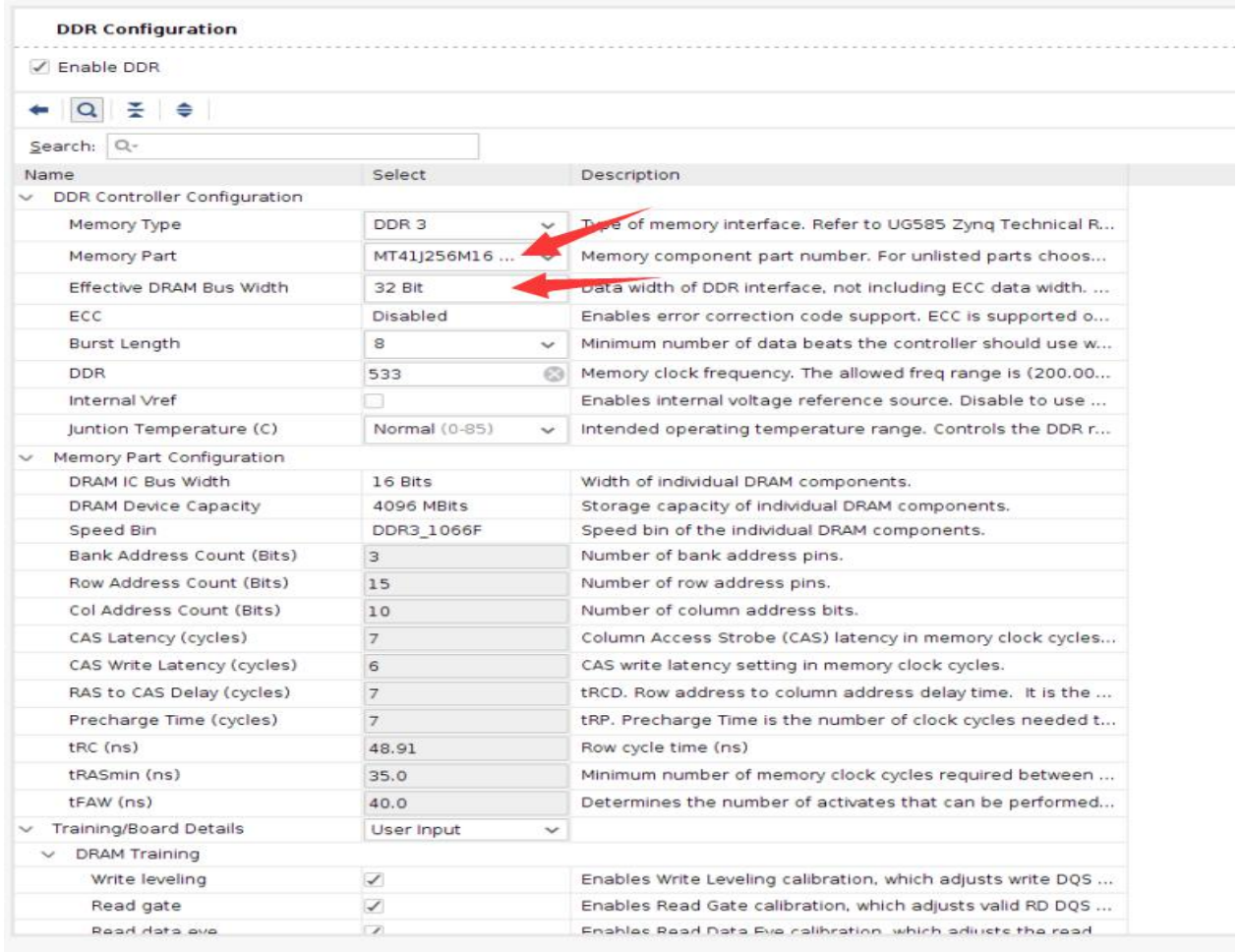

图 1.16 DDR 设置

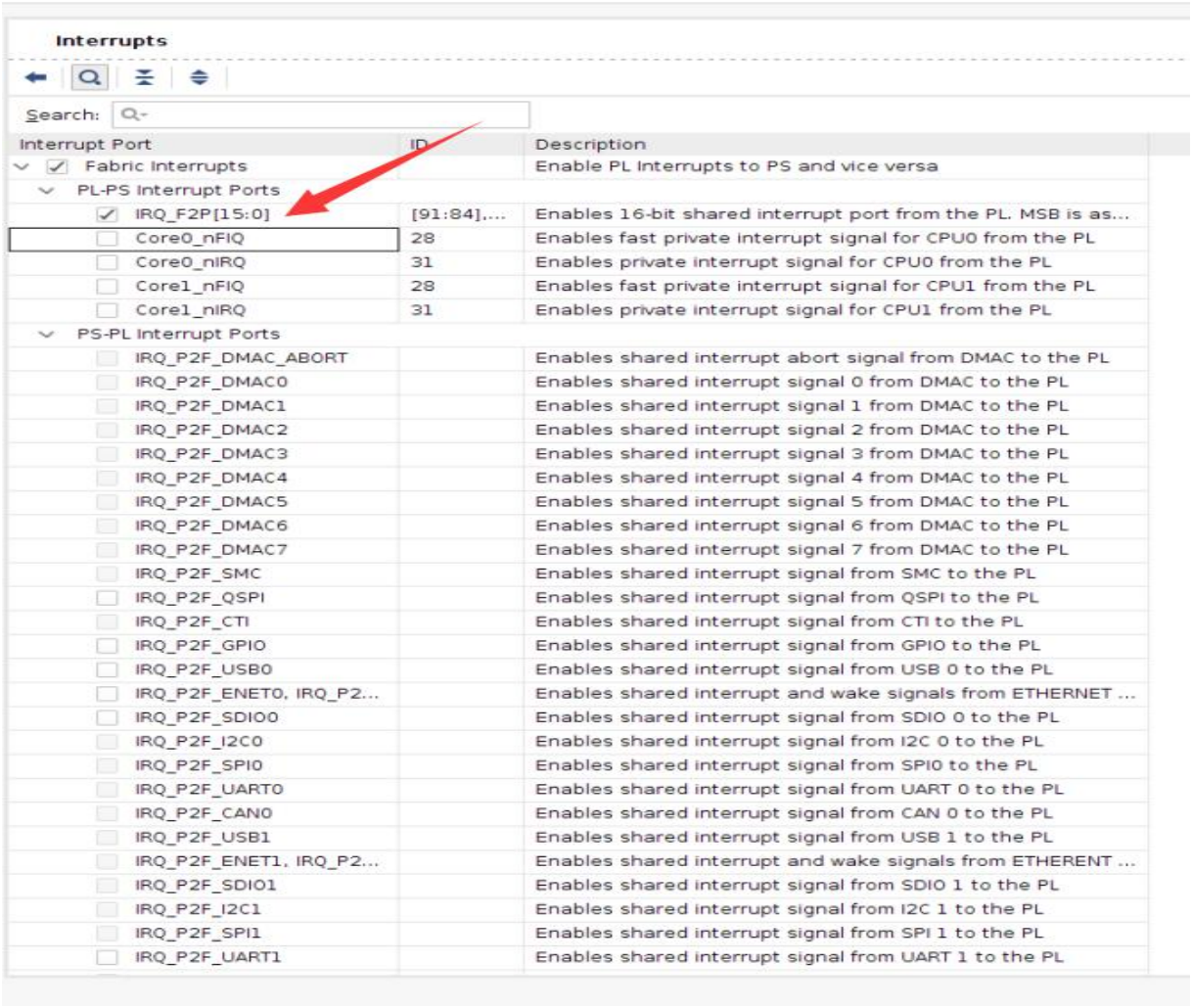

图 1.17 中断设置

在长时间的连线，配置完成后就可以点击设计图上的小对号，验证设计图是否有原则性的设计问题，注意：警告可以忽略，但是 critical warning 不能忽略，会导致一系列的问题。

一般的，只会有一些因为忘记连线的小错误，验证后这一步就结束了，结束后点击自动排序按钮，让设计图更好看一点调整一下心情。

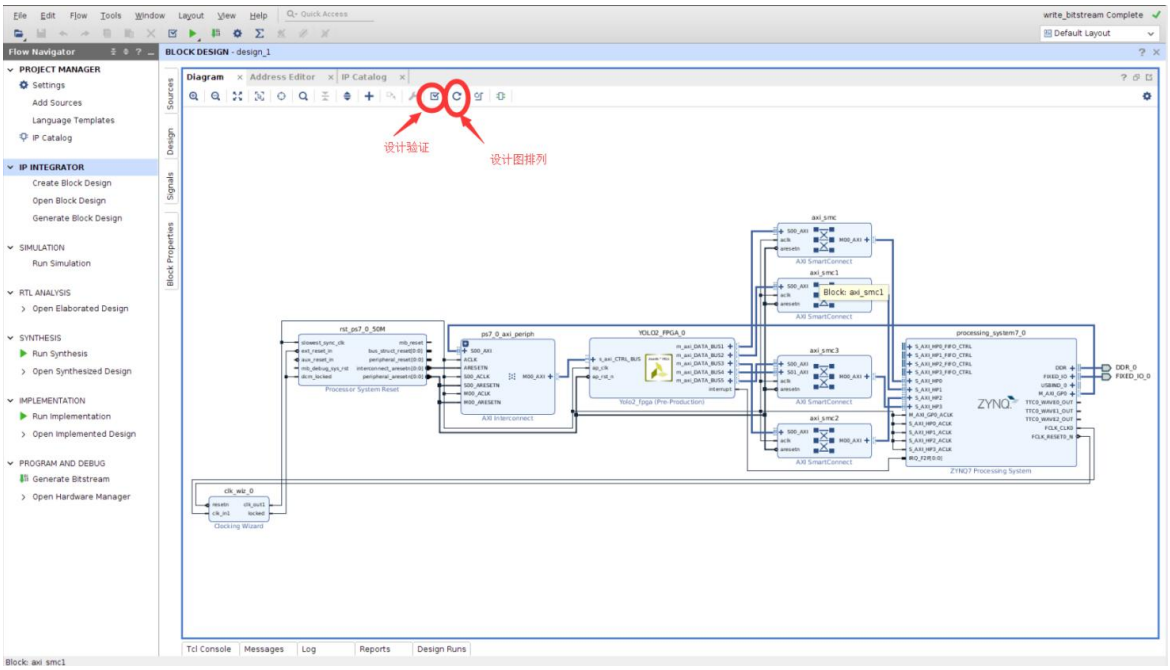

图 1.18 自己生成的设计图

并不是说验证后就没有任何错误，1.3 与 1.4 才是真正查错的开始。

1.3 vivado 仿真

1.4 vivado implement

1.5 生成 bitstream

这两步依次点击，最终无错后才算完成，在这里的步骤不用多说，**说一些自己遇到的错误。**

[BD 41-703] 设备之间读写速率不匹配

[BD 41-971] 地址分配错误

这两个错误的主要原因是因为 vivado 的自动地址分配机制引起的，这个错误是通过原作者的提示解决的，先全部取消地址映射，然后手动分配地址，通过此办法顺利解决了。

[Timing 38-282] 这个就是因为电平的设置引起的

[警告代码忘了] 这是提示 IPcore 过期，也就是说，导入了 ipcore 的原 ipcore 不能再使用 hls 重新综合打包，然而此错误可以忽略，不必理财（按照 vivado 提示步骤走即可解决）

上两步完成后 1.5 基本没有问题(这里我就没有遇到问题)

注：这里的错误每个人遇到的可能都不完全相同，搜索错误时建议使用错误代码，然后就会出现 xilinx 官网的一些信息，大部分都可以解决。

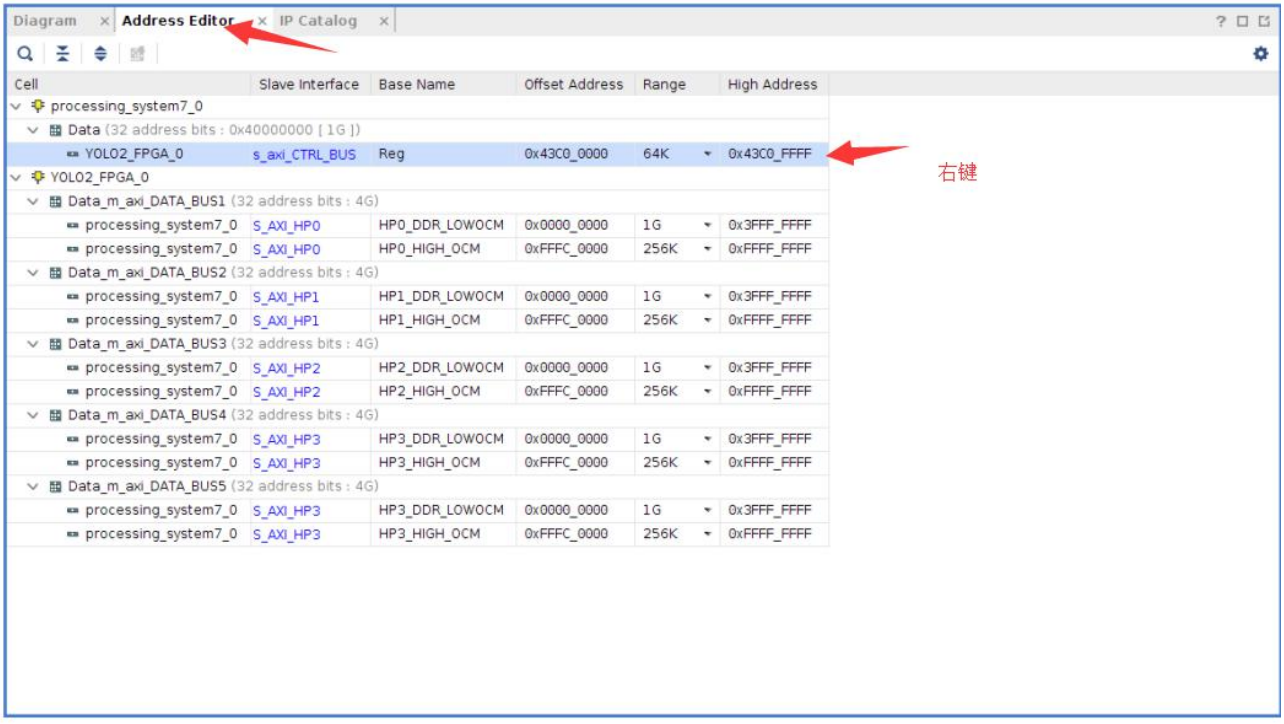

图 1.19 地址分配

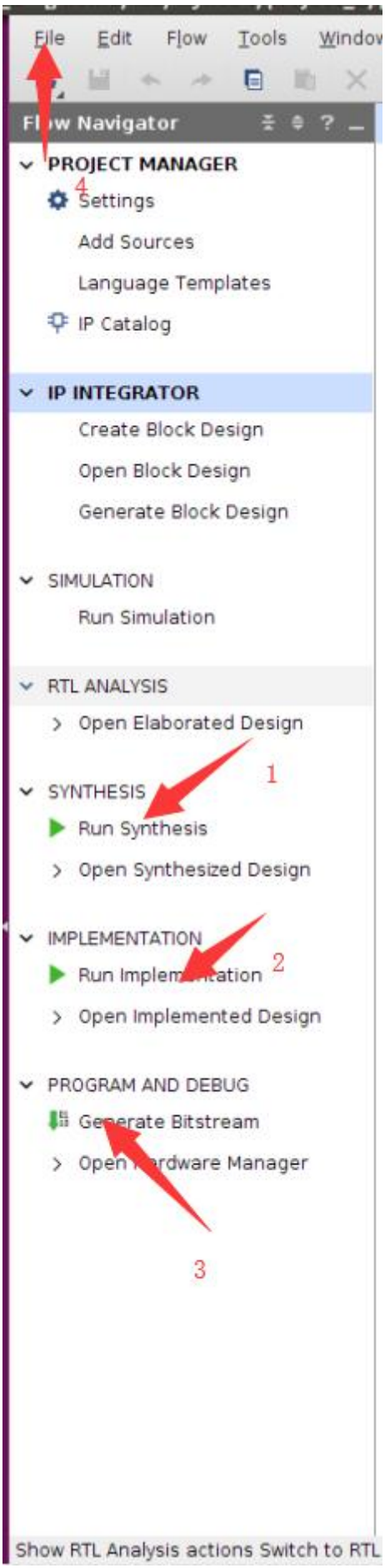
File
Edit
Flow
Tools
Window
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Open Implemented Design
PROGRAM AND DEBUG
Show RTL Analysis actions Switch to RTL
1
2
3
4

图 1.20 各个步骤顺序

在经过**很久很久很久**的等待后，以上步骤会全部完成。
注：电脑配置一定要足够好，否则会跑不动，最好是 i7+16G，本人 i5+8G 就卡死了

完成后点击 file→export→export hardware
然后点击 launch SDK
这一步就完全结束了

二：elf 的生成

在上一步的 launch SDK 后进行这一步的设置，这一步的困难不是很多，这一步生成的就是在开发板上的可执行程序，只有此步骤在后期可以单独运行，其他三个步骤是全部配套的。如果只是测试 BOOT 文件，**可以跳过这一步**。

打开 SDK 后等待 sdk 装载，成功打开后点击 file→new project->application，而不是在 hardware 上右键生成 application，后者是不行的。最明显的区别就是直接生成的不会有 bsp 文件，**如果**生成了 bsp，请核实步骤。

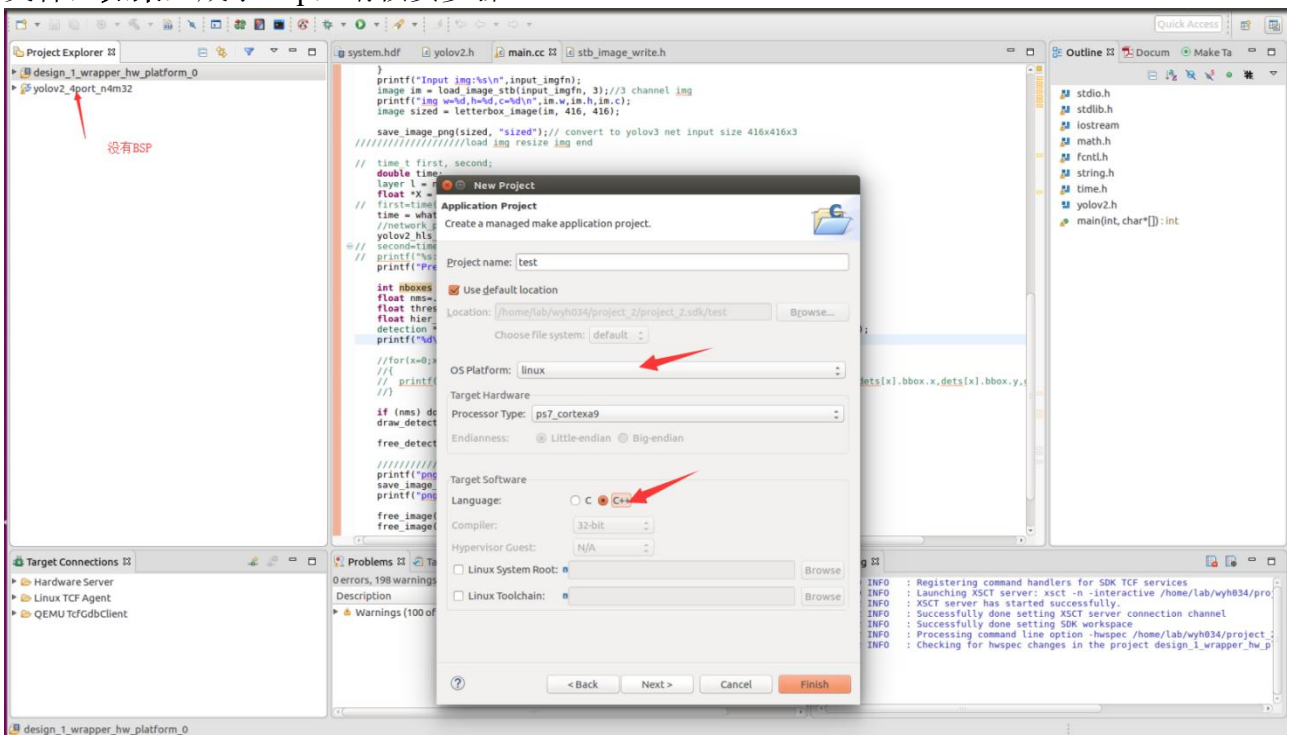

图 2.1 工程细节设置

之后在工程内添加由原作者提供的源文件，添加完成后右击工程名称→properity→settings 设置版本为 release 版本，编译器为 arm-v7-linux-g++，在命令后添加-static

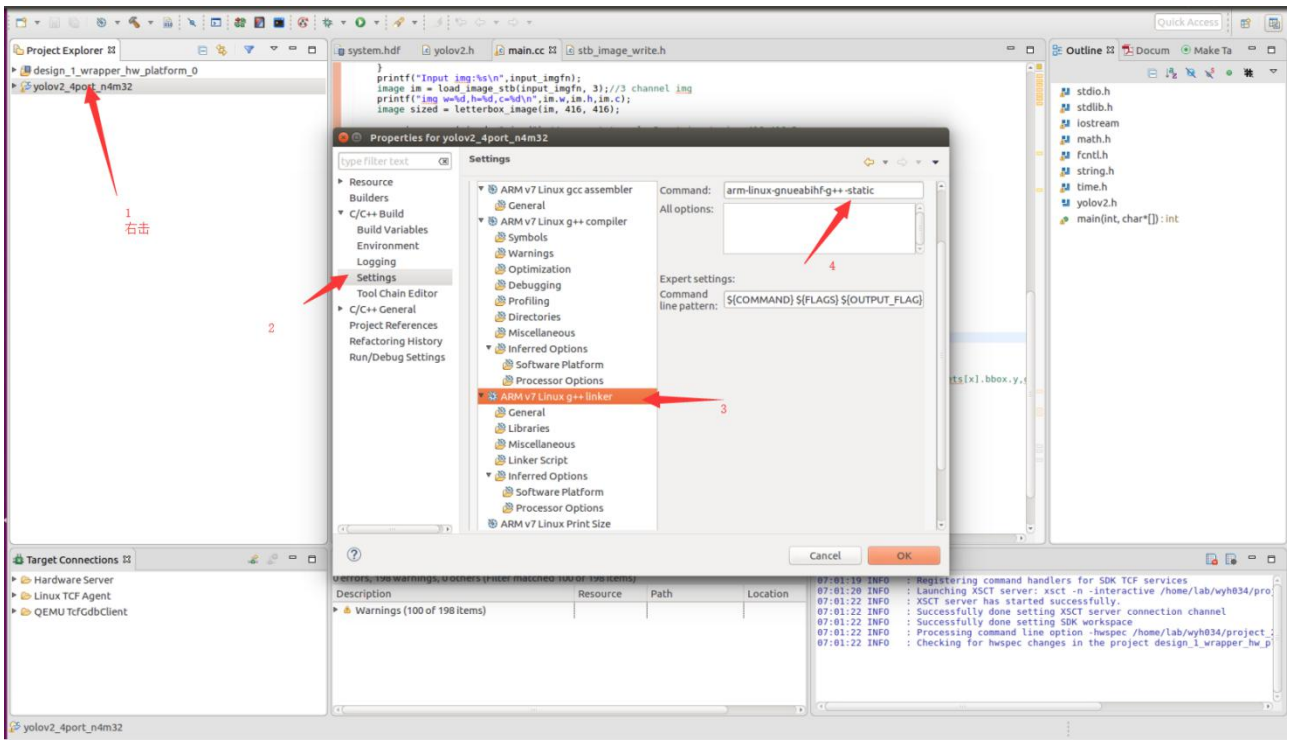

图 2.2 编译器设置

然后点击 build project，就会生成 elf，一般不会有大问题。

此后根据对代码阅读的逐步熟悉，可以改进代码重新编译，**与后面相对独立。**

三：BOOT 生成

点击关闭 SDK，找到工程文件夹下的 SDK 文件所在，并记住。

3.1 petalinux 的安装

这一步非常非常非常坑，petalinux 对版本及语言的要求极其严格，如果是非 petalinux2017.4，自寻配置信息。针对 2017.4petalinux，需求 vivado2017.4，Ubuntu16.0.4(18.0.4 会报错，无法解决)，语言必须是英文，编码格式必须为 utf-8。

系统的安装 bing 自寻（必要依赖文件以及一些其他的创建步骤），只是注意，运行 petalinux 时一定不能使用 sudo 及其他超级用户权限的命令，必须在普通用户权限下执行。

每次执行 petalinux 之前必须 source 设置文件，否则会报错。

在经历一系列的报错后，安装成功，其余错误自寻。

```
lab@lab-NASA:~$ petalinux-create
ERROR: No type is specified!
This command creates a new PetaLinux Project or component

Usage:
  petalinux-create [options] <-t|--type <TYPE> <-n|--name <COMPONENT_NAME>

Required:
  -t, --type <TYPE>                       Available type:
                                            * project : PetaLinux project
                                            * apps    : Linux user application
                                            * modules : Linux user module
  -n, --name <COMPONENT_NAME>             specify a name for the component or
                                          project. It is OPTIONAL to create a
                                          PROJECT. If you specify source BSP when
                                          you create a project, you are not
                                          required to specify the name.
Options:
  -p, --project <PROJECT>                 specify full path to a PetaLinux project
                                          this option is NOT USED for PROJECT CREATIO
                                          default is the working project.
  --force                                 force overwriting an existing component
                                          directory.
  -h, --help                              show function usage
  --enable                                this option applies to all types except
                                          project.
                                          enable the created component

Options for project:
  --template <TEMPLATE>                   zynqMP|zynq|microblaze
                                          user needs specify which template to use.
  -s|--source <SOURCE>                    specify a PetaLinux BSP as a project
                                          source.

Options for apps:
  --template <TEMPLATE>                   <c|c++|autoconf|install>
                                          c   : c user application(default)
                                          c++ : c++ user application
                                          autoconf: autoconf user application
                                          install: install data only
  -s, --source <SOURCE>                   valid source name format:
                                            *.tar.gz, *.tgz, *.tar.bz2, *.tar,
                                            *.zip, app source directory

Options for modules: (No specific options for modules)

EXAMPLES:

Create project from PetaLinux Project BSP:
  $ petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP>

Create project from template:
  $ petalinux-create -t project -n <PROJECT> --template zynq


Create an app and enable it:
  $ petalinux-create -t apps -n myapp --enable
The application "myapp" will be created with c template in:
  <PROJECT>/project-spec/meta-user/recipes-apps/myapp


Create an module and enable it:
```

图 3.1 petalinux 展示

3.2 按照作者的配置修改设备树

```
/include/ "system-conf.dtsi"
/ {
    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;

        reserved: buffer@0x10000000 {
             no-map;
             reg = <0x10000000 0x10000000>;
        };
    };

    reserved-driver@0 {
        compatible = "xlnx,reserved-memory";
        memory-region = <&reserved>;
    };

};
```

图 3.2 设备树文件

3.3 进行编译

需求：步骤 1 生成的 SDK 文件夹，3.2 生成的设备树文件。具体的使用教程自寻。

对配置保持默认即可(Image package configuration 除外，放在后面讲)，然后会进入一系列的界面，保存退出即可。

注意，这一系列的命令也很复杂，编译过程也是**很长很长很长**(35min 左右)

建议写一个脚本，如下所示

```
#!/bin/sh

oPath="./out"

if [ -d "$oPath" ]; then
  rm -rf "$oPath"
fi
echo "Input Project name:"
read filename
mkdir out
source /opt/pkg/petalinux/settings.sh                    所有的project改成自己的工程名
source /opt/Xilinx/Vivado/2017.4/settings64.sh
cd ../
petalinux-create --type project --template zynq --name $filename
cp -r ./mkBOOT/project_2.sdk ./$filename
rm -rf ./$filename/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
cp ./mkBOOT/system-user.dtsi ./$filename/project-spec/meta-user/recipes-bsp/device-tree/files/
cd ./$filename
petalinux-config --get-hw-description ./project_2.sdk/
petalinux-config -c kernel
petalinux-config -c rootfs
petalinux-build
petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga --u-boot --force
cd ../
cp ./$filename/images/linux/BOOT.BIN ./mkBOOT/out
cp ./$filename/images/linux/image.ub ./mkBOOT/out
cp ./$filename/images/linux/rootfs.cpio ./mkBOOT/out
rm -rf $filename
```

图 3.3 脚本文件

工程名自行更改，所生成的需要的三个文件会自动拷贝到 out 文件夹内

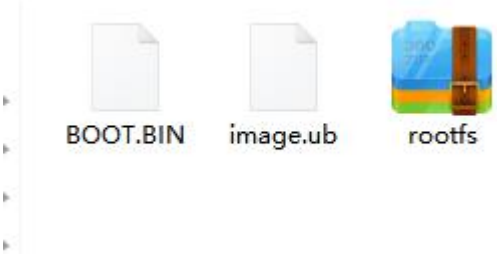

图 3.4 结果展示

3.4 拷贝到 SD 卡内

将 SD 卡分成两个区，分区教程自行百度，一个格式化成 FAT 格式，另一个格式化成 EXT4 格式。将 BOOT.BIN,image.ub 拷进 fat 格式区，将 rootfs.gpio 拷进 ext4 格式区并运行 pax -rvf {文件}。

3.5 Image package configuration 配置问题

这一步非常，极其特别坑爹。作者的原版配置是设置为 SD 卡模式，因为这种模式下可以正常使用 SD 卡，写入的程序掉电也不会消失。所以，作者的第四步很简单，直接运行就可以了。然而我在设置成和作者一样的格式后并没有成功运行，而是出现了一条 Unable to mount rootfs 的提示。经过多次的修改设置后编译测试，最终发现，保持默认设置即可解决。但是随之会带来另一个问题，由于默认设置的文件格式是 RAM 类型的，也就是说掉电即消失，所以需要在步骤 4 中寻求新的方法来执行测试程序(生成好的 elf)。经过多方查找资料，最后使用了 NFS 网络文件系统来作为这一步操作的补偿。注意，在与和作者的交流后发现，这一步可能是版本 bug，具体情况至今不明。

有的小机灵鬼可能会想到把 elf 在上一步的 pax 后立即拷进去这样子的操作，很遗憾，我也尝试了，这种操作并不能解决此问题，造成的结果就是，文件依然在，但是开发板并不能识别到。

四：测试运行

首先说明，由于上一步的补偿操作要使用 NFS 系统，然而 Windows 下的 nfs 怎么用自行百度，本人用的是 Ubuntu 下的 nfs。

4.1 数据准备

需求：作者提供的 software version 文件夹，下载好的 yolov2 模型

使用步骤每一步的文件夹下都有提示，按照操作就行了，唯一值得注意的是，第一步的代码是添加到 darknet/src/parser.c 中的函数下。由于这些文件特别长，要找到固定的函数太麻烦了，最好使用 grep 命令查找，因为这三步会有很多的添加删除操作。至于命令的具体用法，自行百度。

在最终的三个步骤执行完后，会生成一些文件，这些文件就是量化好的偏置矩阵，量化及重组后的权值矩阵。然后根据作者最后要用的，将需求的文件找到放在一个文件夹下备用。

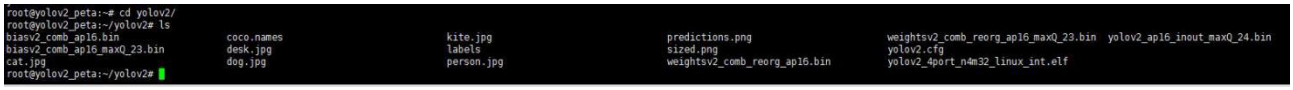

图 4.1 yolov2_xilinx_fpga/petalinux/files_that_yolov2_need.jpg

4.2 设备配置及需求软件安装

4.2.1 串口设置

一般来说 Ubuntu 会有串口驱动，将开发板的 JTag 和 Uart 连接电脑后，先安装 minicom，然后键入 dmesg | grep usb 在最下面找到最新插入的 usb 的设备号。然后 sudo minicom -s 进入设置 setup port 为刚刚的 USB 设备号，然后保存，退出。

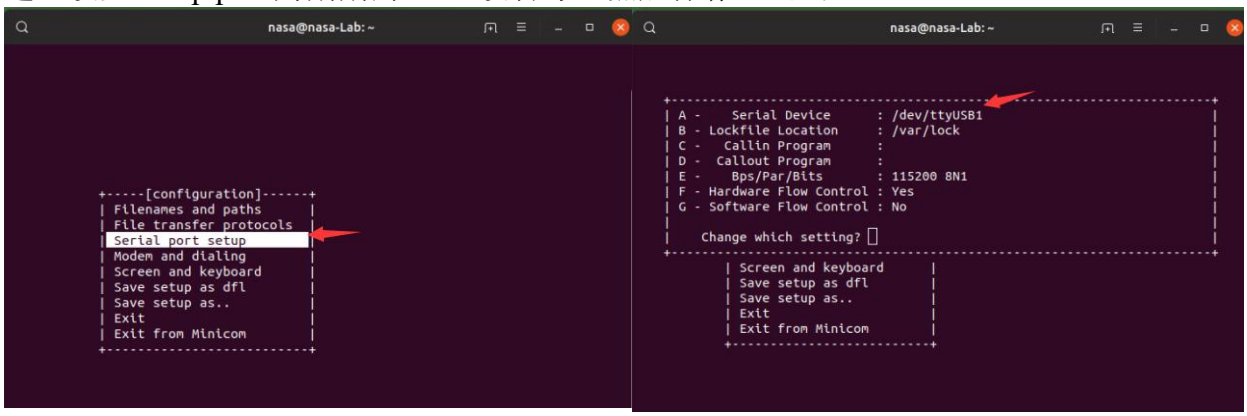

图 4.2 minicom 设置

4.2.2 安装 putty

sudo apt-get install putty 即可。

4.2.3 NFS 启用与配置

首先要使得开发板与要开启 nfs 系统的主机在一个子网内，比如 219.245.31.xxx(注意看 mask)，最简单的就是两个设备在一个交换机或者路由器上。然后需要在 PC 上打开 NFS 设置，由于 Ubuntu 的 nfs 设置变化比较大，这一步不给出具体命令，自行搜索即可，此步骤只需开启一次即可。

注意：NFS 文件夹需要通过这种特殊方式创建，轻易不要改动。

4.3 最终的运行

在经历了一系列的编译调试安装后，这一步一般不会有问题。

开发板 Jtag 和 Uart 连接电脑，开发板连接电源线和网线。启动电脑后打开 putty，选择 Seriel，波特率输入 115200(为什么是 115200，在步骤一中的 zynq 设置中可以看见)，然后输入在上一步确认的 USB 号，注意 USB 设备号是会变化的。点击连接。

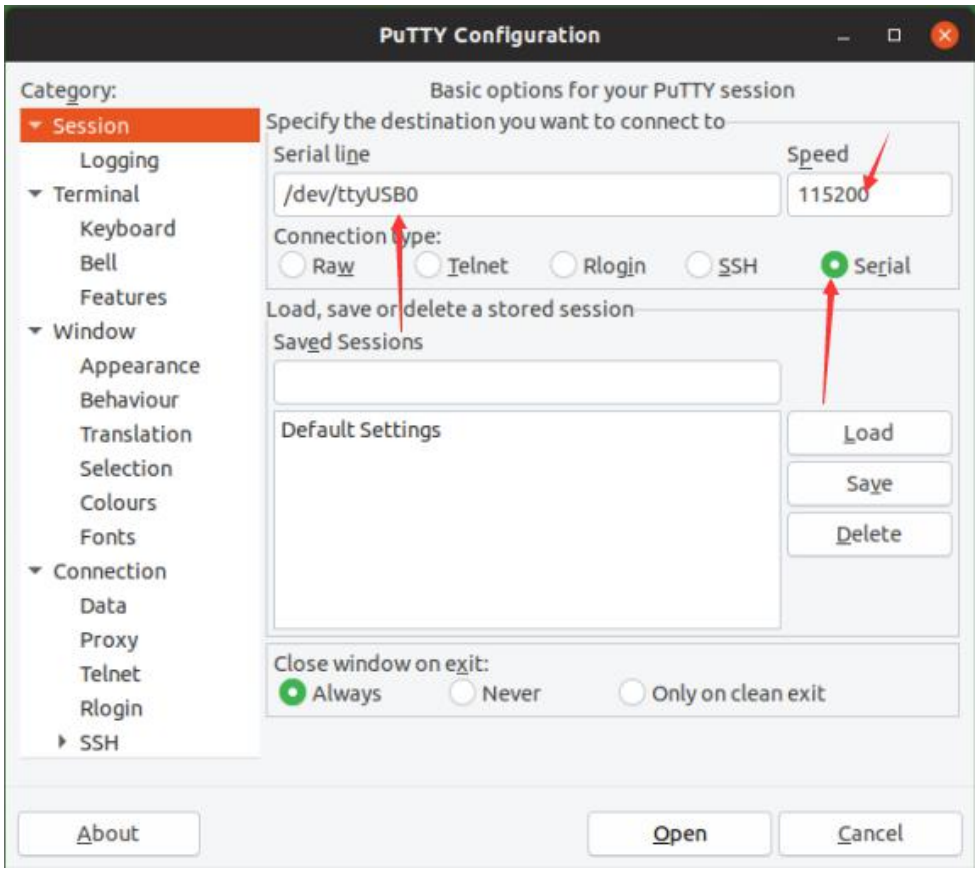

图 4.3 putty 设置

开发板上电，如果 putty 界面出现一长串的信息，证明连接成功，否则请检查 usb 设备号。

以下操作均在开发板上执行

连接成功后输入两次 root 就进入了（为什么是两次？一次是用户名，一次是密码。怎么改密码？在 BOOT 生成步骤的 petalinux 编译可以改密码。）然后确认 PC 的 ip 地址，然后输入 mount -t nfs -o nolock <PC-IPv4-ADD>:/<PATH-TO-NFS-DIR> /mnt (别问为什么是 mnt，自己看看别的文件夹权限就懂了，或者你把权限改了也行)

通过这种方式，开发板上对 mnt 文件夹的操作会保存在 PC 上，对 PC 的操作也会影响到开发板上，通过此种方式就弥补了之前设置的缺陷。

然后将 4.1 节准备好的文件夹放进 NFS 文件夹，在开发板上就可以看到了，然后的然后，cd <PATH-TO-ELF-DIR> 然后 chmod 777 .elf 然后*/.elf 看开发板表演就完事了。你说这样报错了？那肯定报错了啊，因为你没有输入需要预测的文件啊，默认使用的是 person.jpg 如果你没有这张照片，而且没输入图片名字，就会报错。最后会生成一张 prediction.png 的图片，在 PC 下打开看看，就是结果了，运行时间会输出在 putty 界面。

别问二三四为什么字比一少的多，因为实在是打不动了。