版本历史

文档更新记录	文档名:	A10_SoC_up 使用环境说明
	版本号	V0.1
	创建人:	计算机体系结构研讨课教学组
	创建日期 :	2017-12-04

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2017/12/05	邢金璋	V0.1	初版。

文档信息反馈: xingjinzhang@loongson.cn

1 SoC_up 使用环境说明

在学习并尝试本章节前，你需要具有以下环境和能力：

(1) 较为熟练使用 Vivado 工具。

(2) 一定的 CPU 设计与实现能力。

通过本章节的学习，你将获得：

(1) AXI 协议的知识。

(2) CPU 总线接口设计的知识。

在本章节的学习过程中，你可能需要查阅：

(1) 参考资料“AMBA 总线协议”。

(2) 课本上总线知识。

在开展本次实验前，请确认自己知道以下知识：

(1) 总线协议的握手的概念。

(2) AXI 协议中握手信号 ready 和 valid 的概念。

(3) AXI 协议 5 个通道的概念。

(4) AXI 协议各控制信号的意思。

本次实验有以下几个坑，请在碰到问题时注意查看：

(1) 对 AXI 接口的 ar、aw、w 通道，如果 master 先置上了 valid，但没有收到对应的 ready，也就是握手失败了，但这时不能更改该通道的其他信号。比如 arvalid 置上时，未看到 arready，master 不能更改 araddr。**也就是 AXI 一旦发起请求，就不能撤销或更改请求，直至请求握手完成**。

(2) 但是第一阶段我们设定的类 SRAM 接口，却只保证 req&addr_ok 有效时，也就是握手成功时，addr 和 wdata 是有效的。其他时候，**即使 req 有效，但 addr_ok 为 0，addr 和 wdata 也是 x**。这样的设定虽然对第一阶段实验而言比较麻烦，却方便了第二阶段的实现。

(3) Lab6 上板运行时，可以根据拨码开关更改随机种子。所以有可能上板运行时，更改了随机种子，测试就跑不过了。这一情况在第一阶段可能比较少碰到，但在第二阶段应该会普遍碰到。一旦碰到，就需要仿真设定相同随机种子，来仿真复现该错误进行 debug。

以下几点，第一阶段可能碰不到，但第二阶段可能会碰到，最好可以提前考虑下：

(1) AXI 读写分类，对先写后读，可能读先完成的。也就是先写后读，且写读同一地址，如果在写为收到 b 通道时就发出了读的 ar 通道，那就有可能读出旧值，这是不正确的。类似的，先读后写，且读写同一地址，读未完成，写就发出了，那有可能读到写后的值，这也是不正确的。

1.1 基于 GS232 搭建的 SoC_up 说明

1.1.1 GS232 开源版本简介

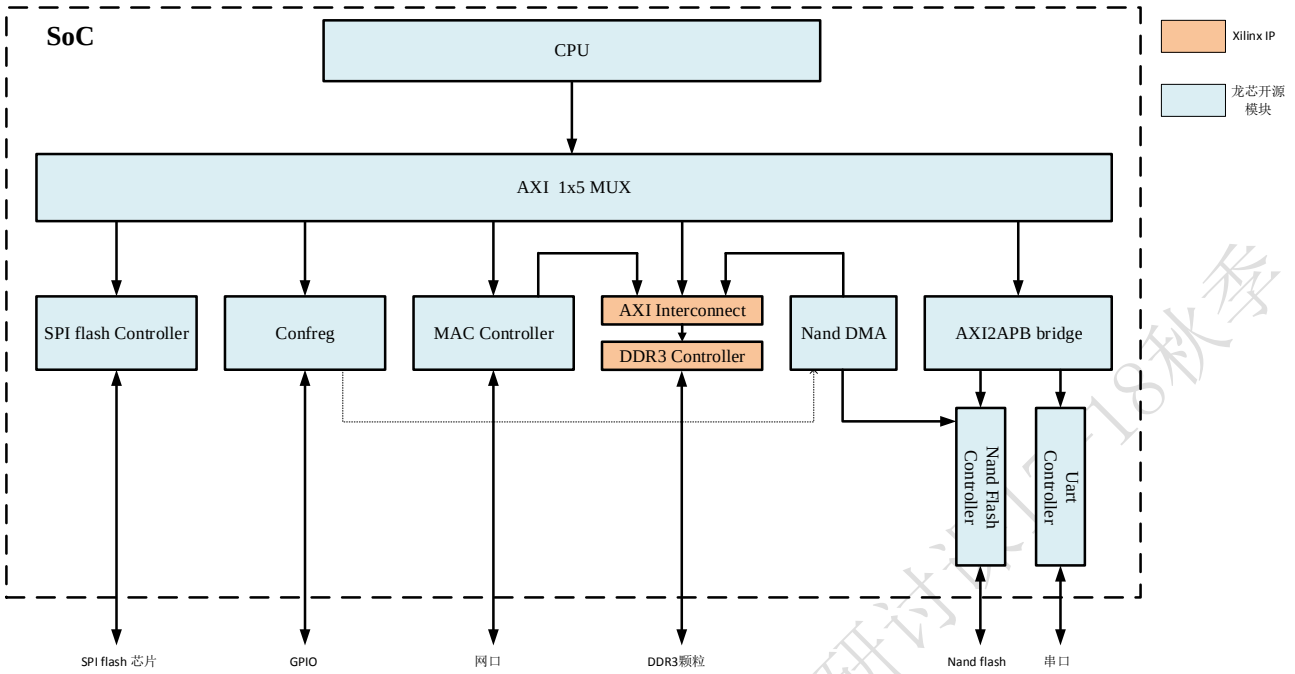
GS232 开源版本不包含 DSP、浮点部件等。

TLB 大小为 32 项。

指令和数据 Cache 为 4 路组相连，每路大小为 4KB，Cache 行大小为 32 bytes。

对外接口为 32 位 AXI 接口。

1.1.2 SoC_up 结构



SoC_up 如上图所示。开源 LS232 对外有一个 AXI 接口，连接到 AXI 互联网络上与外设相连。

SoC_up 对外连接的设备共有 6 个：SPI flash、GPIO(数码管、LED 灯、开关灯)、网口、DDR3 颗粒、Nand flash 和串口。这些外设在教学实验板上均已集成。

1.1.3 地址空间分配

各外设地址空间分配如下：

各控制器模块名	分配的虚拟地址段	地址空间大小
SPI flash	0xbfc0_0000_0xbfcf_ffff 和 0xbfe4_0000~0xbfe4_ffff	1MB(flash 存储空间) 64KB(控制器寄存器空间)
GPIO	0xbfd0_0000~0xbfd0_ffff	64KB
MAC	0xbff0_0000~0xbff0_ffff	64KB
DDR3	4GB 空间中的剩余地址[1]	128MB
Nand flash	0xbfe7_8000~0xbfe7_bfff	16KB
Uart	0xbfe4_0000~0xbfe4_3fff	16KB
地址空洞 (软件可以不关注)	0xbfe7_0000~0xbfe7_7fff 0xbfe7_c000~0xbfe7_ffff 0xbfe4_4000~0xbfe4_ffff	分配给 APB 设备的，但由于 APB 设备只有 uart 和 nand，故存在较多的地址空洞，留作后续新增 APB 设备使用。

1.1.4 NAND DMA 控制器

其一端通过 64 位 AXI 接口接到 DDR3 内存上，一端通过 APB 接口接到 APB 设备上（可认为接到 NAND 控制器

[1] 剩余地址：指 4GB 地址空间中，除了其他外设分配的地址外的地址。由于实验板上 DDR3 颗粒大小为 1Gb，即 128MB，故软件访问时注意不要越界，建议软件使用虚拟地址 0x8000_0000~0x87ff_ffff 对其进行访问。

上）。

该 DMA 只用于 nand flash 与内存交换数据。

该 DMA 的配置寄存器 ORDER_ADDR_IN 位于 CONFREG 模块，地址还是为 0xbfd0_1160。

1.1.5 NAND FLASH 控制器

通过 APB 接口接在 APB 桥上。

该 NAND FLASH 控制器不支持上电从 flash 启动和校验纠错。

FPGA 板上 NAND FLASH 颗粒 K9F1G08U0C-PCB0 的 main 区容量为：1K blocks * 64 pages/block * 2K Bytes/page =128M bytes。也就是共有 64k 页，一页为 2k bytes。每页的 spare 区为 64bytes。

1.1.6 CONFREG 模块

包含 8 个 32 位内存映射读写寄存器和一个 dma 的 order_addr_in 寄存器（0xbfd0_1160）。

1.1.7 MAC 控制器

一个从端 32 位 AXI 接口，接到 AXI 互联网络上供 CPU 访问。一个主端 32 位 AXI 接口，接到 DDR3 内存上，自带 DMA 功能。

1.1.8 DDR3 控制器

为 Xilinx IP。

不需要软件配置。一上电，整个 SoC 会先等 DDR3 完成复位，才会撤掉 CPU 的复位信号。

其实分为，一个 3x1 的 AXI 仲裁器，和一个 DDR3 控制器，均为 XilinxIP。

其中 3x1 的 AXI 仲裁器为 2 个 32 位 AXI 接口：一个接 AXI 互联网络供 CPU 访问，一个接 MAC 控制器供网口访问；还有一个 64 位 AXI 接口供 NAND DMA 访问。

1.1.9 SPI FLASH 控制器

通过一个 32 位 AXI 接口接到 AXI 互联网络上。

1.1.10 UART 控制器

仅支持一个串口设备。

1.1.11 中断连接

由于教学 SoC 比较小，故没有集成中断控制器，也就只实现 1 级中断。

其中 6 个硬件中断（$Cause_{IP7-2}$）高位未连接外部中断，其余 5 位硬件中断和外设连接关系如下。

硬件中断名	来源
$Cause_{IP6}$	dma_int：NAND DMA 的中断请求
$Cause_{IP5}$	nand_int：NAND FLASH 的中断请求
$Cause_{IP4}$	spi_int：SPI FLASH 的中断请求
$Cause_{IP3}$	uart0_int：串口的中断请求
$Cause_{IP2}$	mac_int：网口的中断请求

目前 SoC 的各控制器的中断连接为上表的方案。

1.2 串口软件

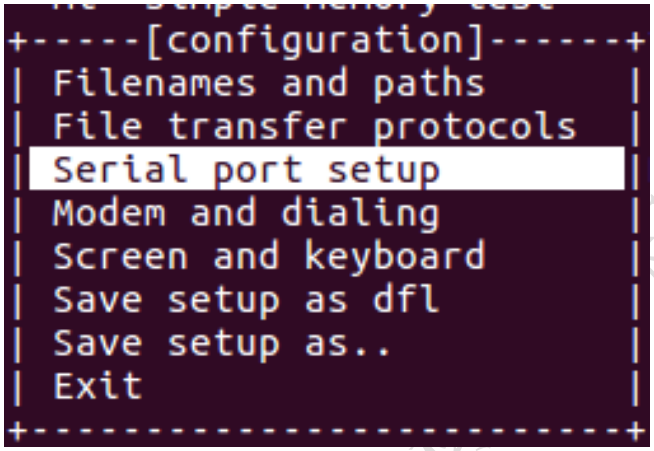
1.2.1 Linux 下

(1) 配置

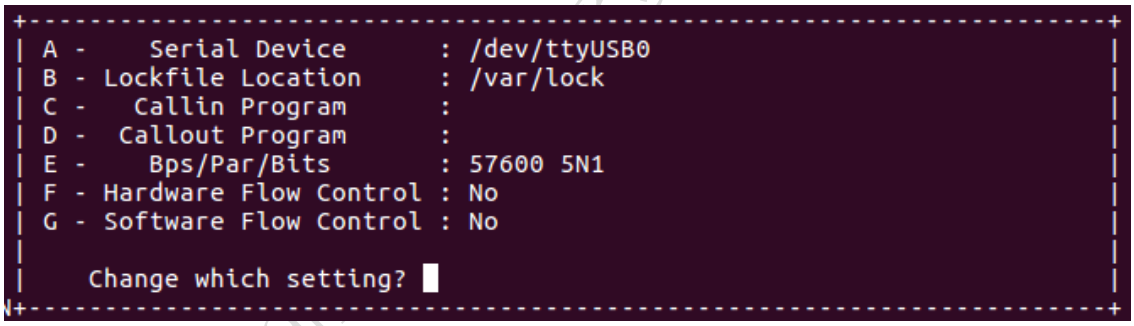
在终端下运行：

```
[abc@www ~]$ minicom -s
```

选择 Serial port setup：



进入配置界面：



其中 E 行依据开发板上串口控制器的初始化代码中设置的波特率进行选择，F 和 G 行选择 NO。

配置完成后按 Enter 返回，选择 Save setup as dfl 保存为默认设置。

(2) 运行

将 USB 转串口一端连到电脑上，一端连到串口线上，串口线另一端连接到开发板上串口接口上。在 Linux 终端运行如下命令，开启电脑上的串口界面，开发板即可与电脑进行交换了。

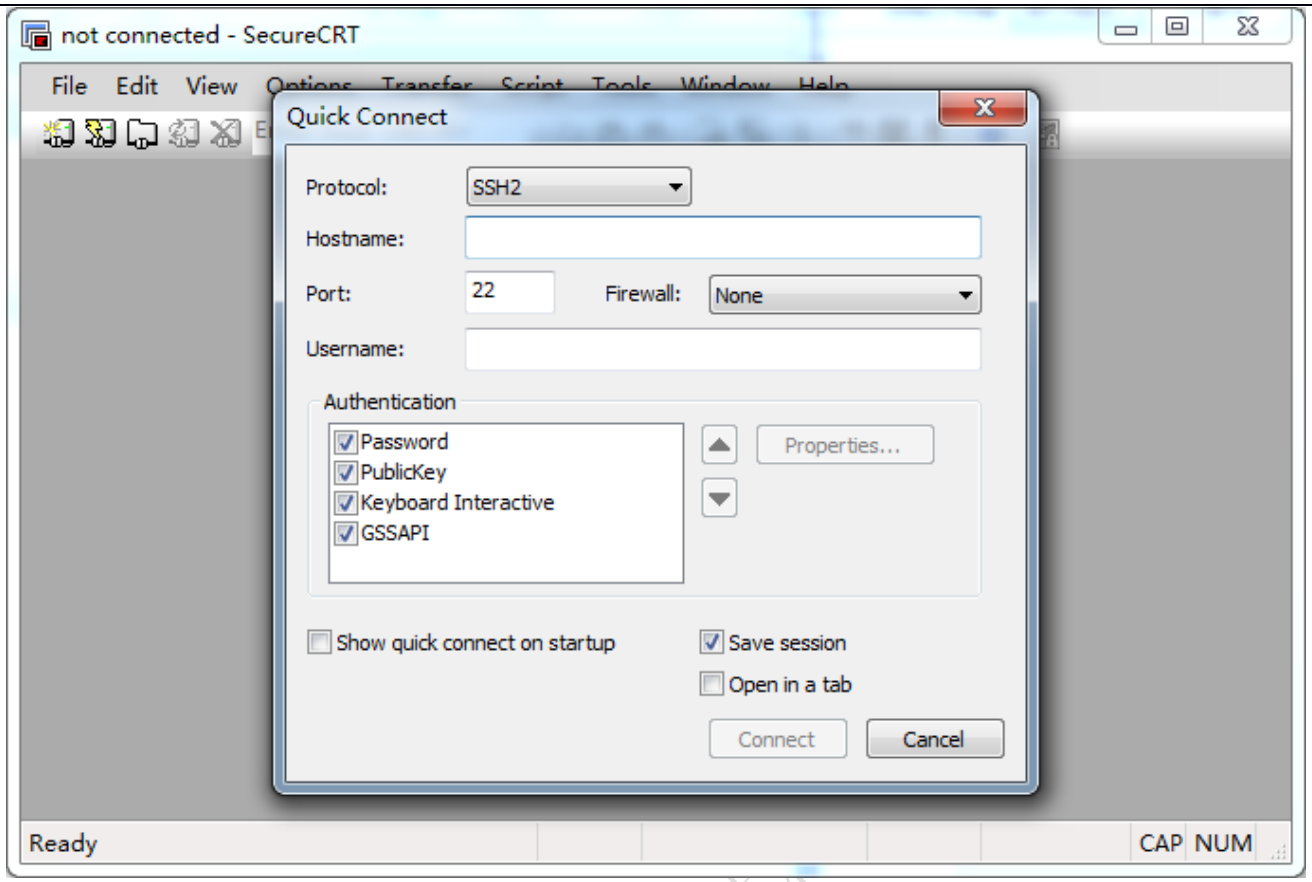
```
[abc@www ~]$ sudo minicom
```

1.2.2 Windows 下

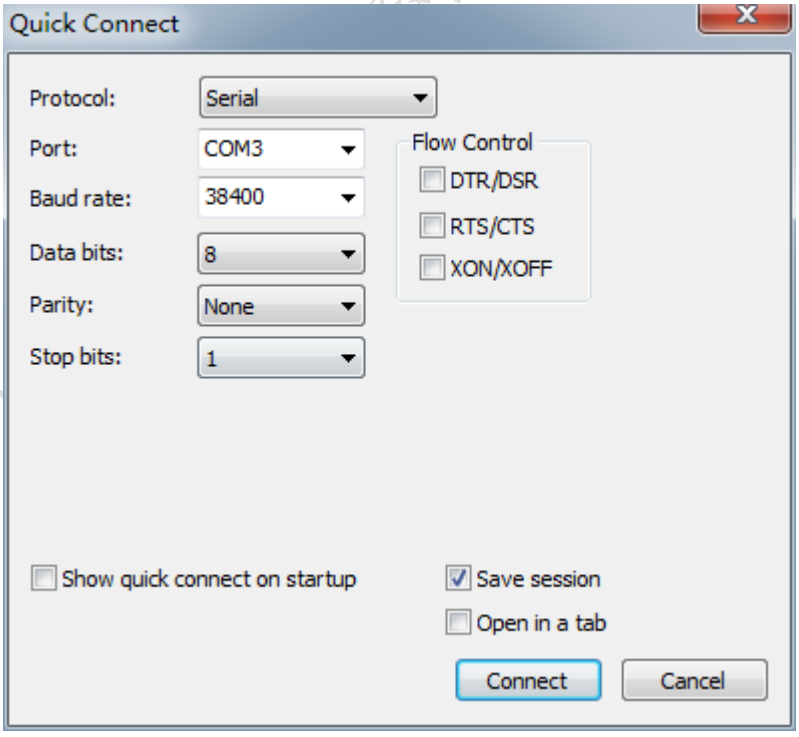
Windows 下可以使用免安装的 SecureCRTPortable 串口软件(见 lab_environment/uart_soft)。

先使用 USB 转串口和串口连接线将电脑和开发板相连。

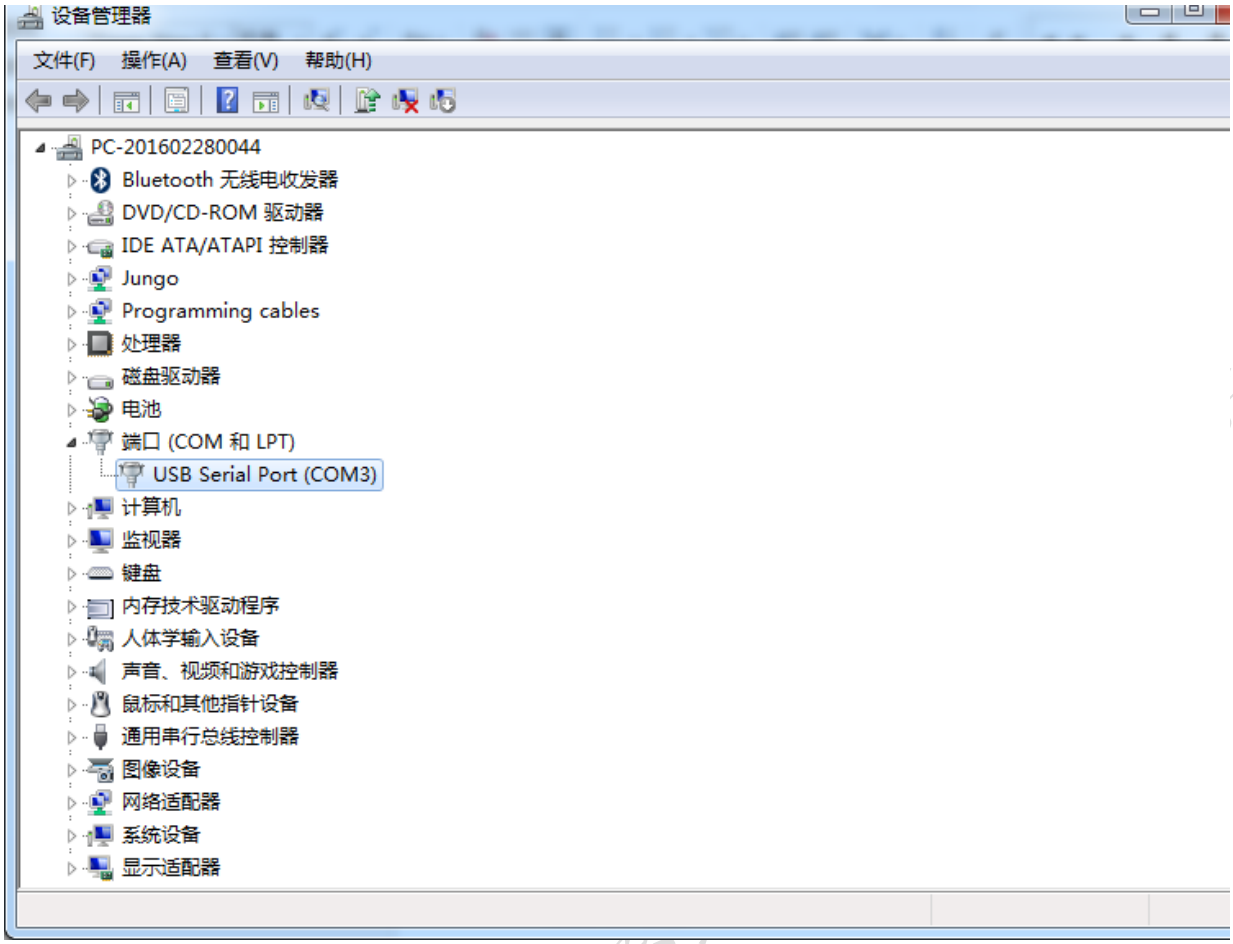
双击程序打开，第一次启动界面如下：



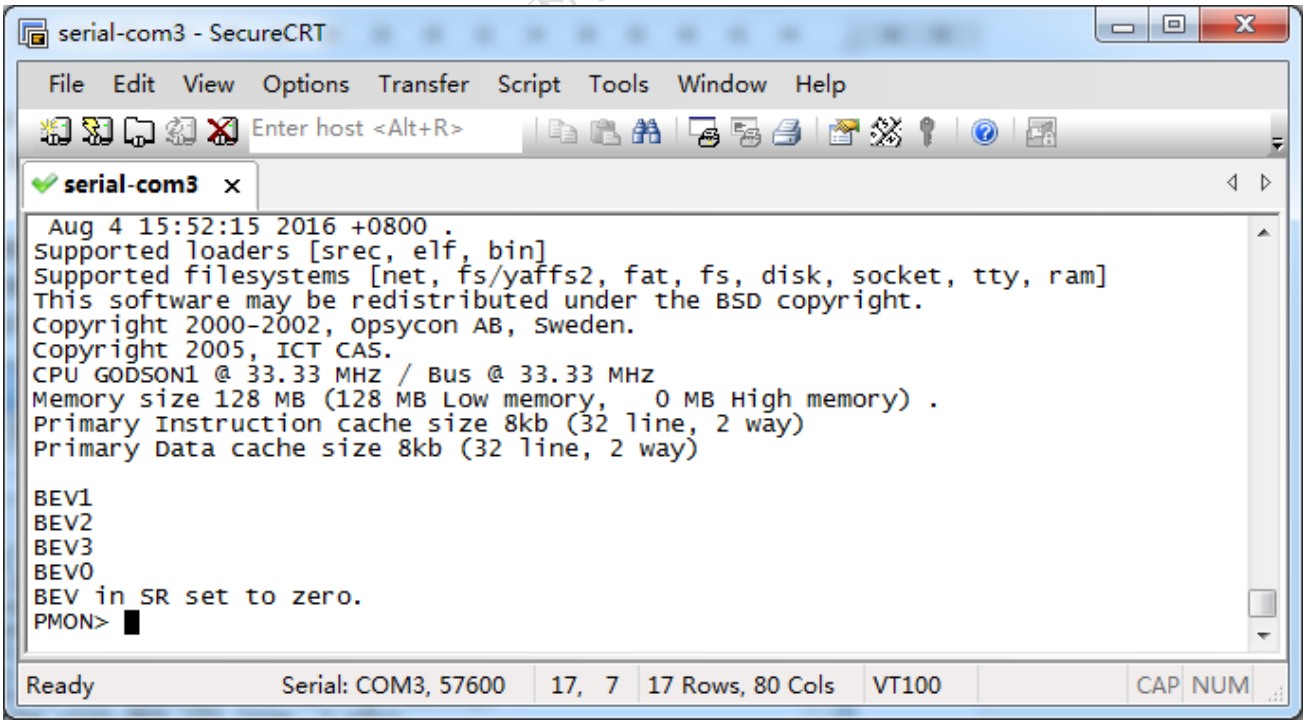
第一行 **Protocol** 下拉选择 **Serial**，如下：



其中 Baud rate 为选择波特率，需根据开发板上串口控制器的初始化代码中设置的波特率进行选择。右侧 **Flow Control** 全不选。Port 的选择需根据 Windows 电脑上的端口进行选择，可以右键电脑选择设备管理器进入**设备管理器**查看：



配置好串口后，点击 **connect**，即可进入串口界面，在波特率设置正确的情况下，可以通过串口与开发板进行交互，如下：



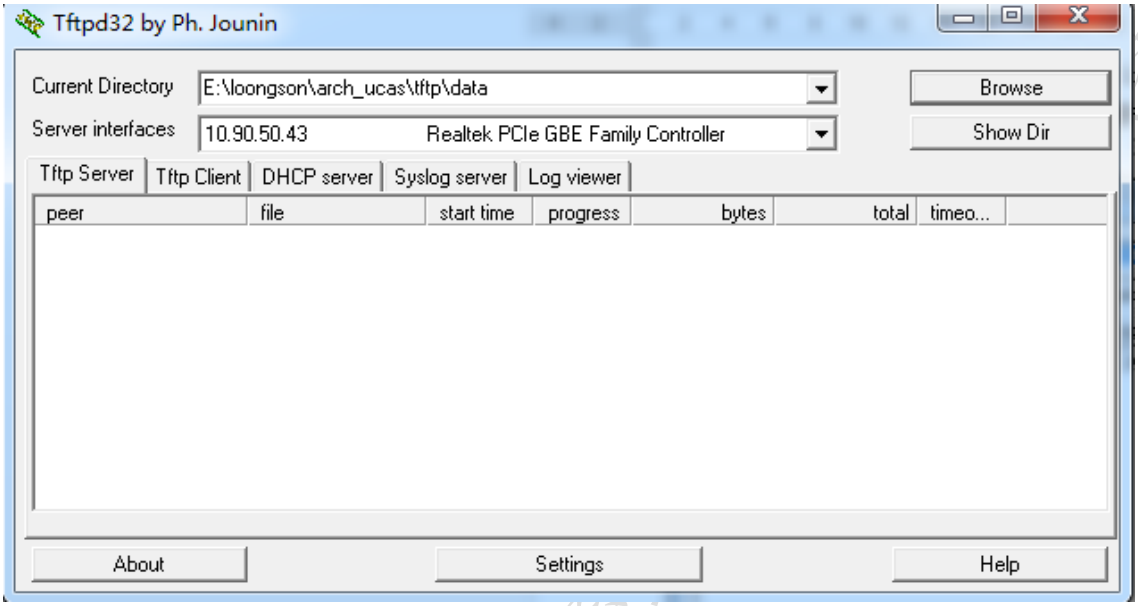
1.3 Tftp 服务器搭建

由于运行 linux 时，最初的内核需要使用网口 load 进入内存执行，因而需要先搭建 Tftp 服务器。

此处只给出 Windows 搭建 tftp 服务器的方法，Linux 下的搭建方法也很简单，请自行学习。

Windows 下可以使用 tftpd32 软件(lab3 实验包里有该软件)。

双击打开应用程序 tftpd32：

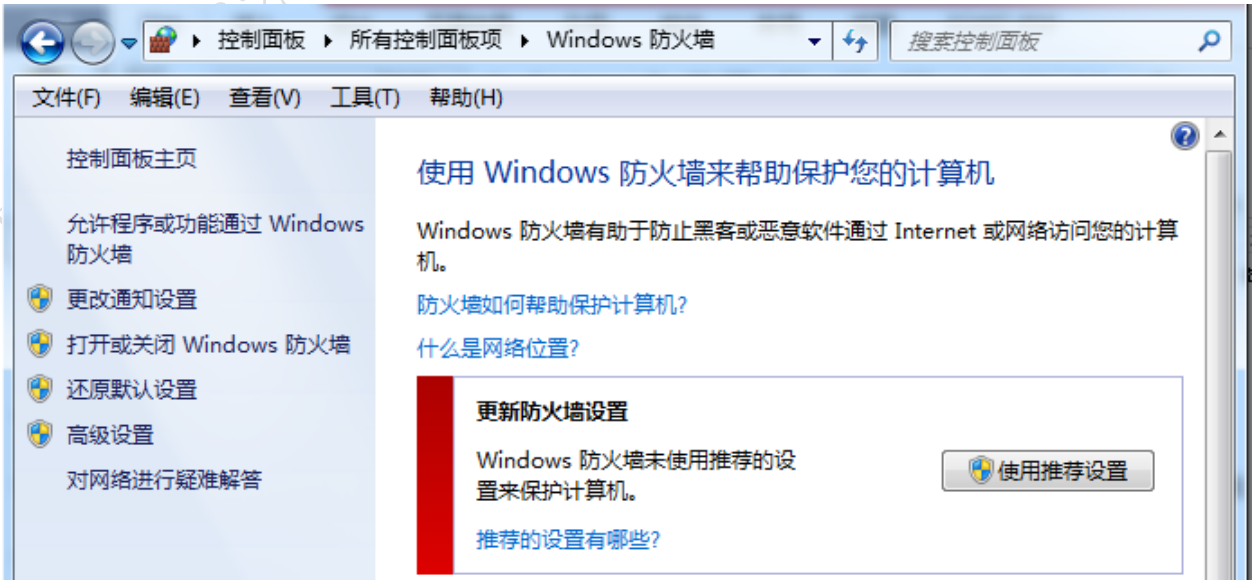


其中 **Current Directory** 为 tftp 服务器的根目录，可以点击 **Browse** 进行更改。点击 **Show Dir** 可以查看该根目录下的文件。

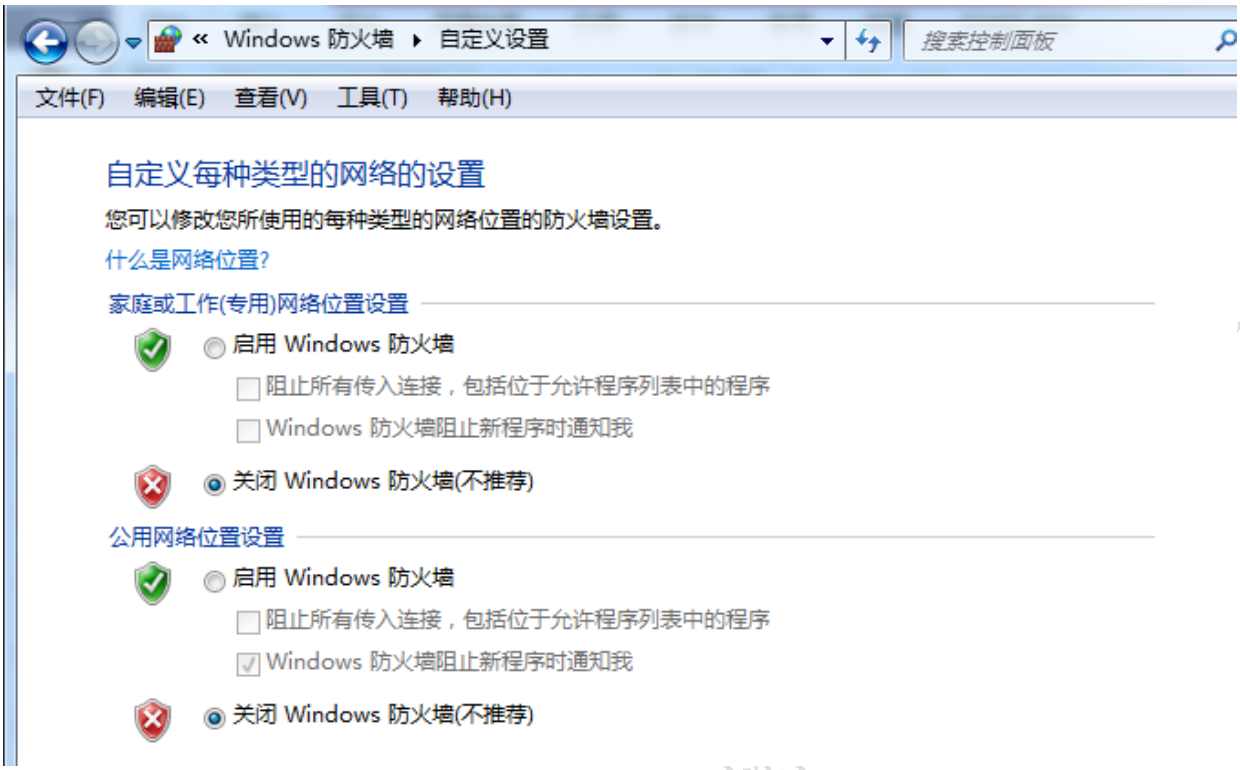
Server interfaces 为选择网卡作为 tftp 服务器的网络入口，可以下拉进行选择，示例中选择了有线网卡接入的 IP：**10.90.50.43**。

至此，Windows 上的 tftp 服务器已正常开启了，虚拟机里的 ubuntu 可以正常访问，但局域网里的其他设备还无法访问，需要关闭电脑上的防火墙。

在控制面板中找到 Windows 防火墙，选择“**打开或关闭 Windows 防火墙**”:



选择关闭 Windows 防火墙即可。



这样同一局域网上的设备就可通过 tftp://10.90.50.43 访问电脑上搭建的 tftp 服务器了，可以从根目录下载文件，或上传文件到根目录下。

同样虚拟机里的 ubuntu 可以再终端下输入如下命令，可登录到搭建的 tftp 服务器上：

```
[abc@www ~]$ tftp 10.90.50.43
tftp>
tftp> binary
tftp> put gzrom.bin
Sent 299024 bytes in 0.7 seconds
tftp> get gzrom.bin
Received 299024 bytes in 0.7 seconds
tftp>
```

使用 **put** 命令可以上传一个文件到 tftp 服务器的根目录下，使用 **get** 命令可以下载一个文件到本地。当传输 bin 文件时，需要先输入 **binary** 命令更改传输模式为二进制模式，否则 bin 文件传输后会出错。

1.4 Flash 烧写

目前提供两种可插拔 flash 芯片烧写方法：使用 flash 编程器和使用 FPGA 上编程 bit 流。

推荐大家使用方法二，因为这样不需要插拔 flash 芯片，防止 flash 芯片引脚损坏。

1.4.1 使用 EZP2010 编程器：

需要具有该编程器设备。

使用该编程器，需手动安装驱动。

烧写时，需拔下 Flash 芯片，之后烧写完成，再插入到 FPGA 板上。

1.4.2 使用 FPGA 编程 bit 流

该方法是使用龙芯开源的 gs132 核搭建了一个小的 soc，该 soc 外设有串口、flash 芯片和指令数据 ram。该 soc 在 FPGA 上生成的 bit 流文件可实现通过串口在线编程 flash 芯片。

编程过程中，不需要拔下 flash 芯片，且速率达到 6KB/sec。具体使用方法如下：

(1) 现有一个使用 gcc 编译好的 binary 文件(后缀名为.bin)，准备使用串口将其下载至 FPGA 上的 flash 芯片中。

(2) 准备工具：

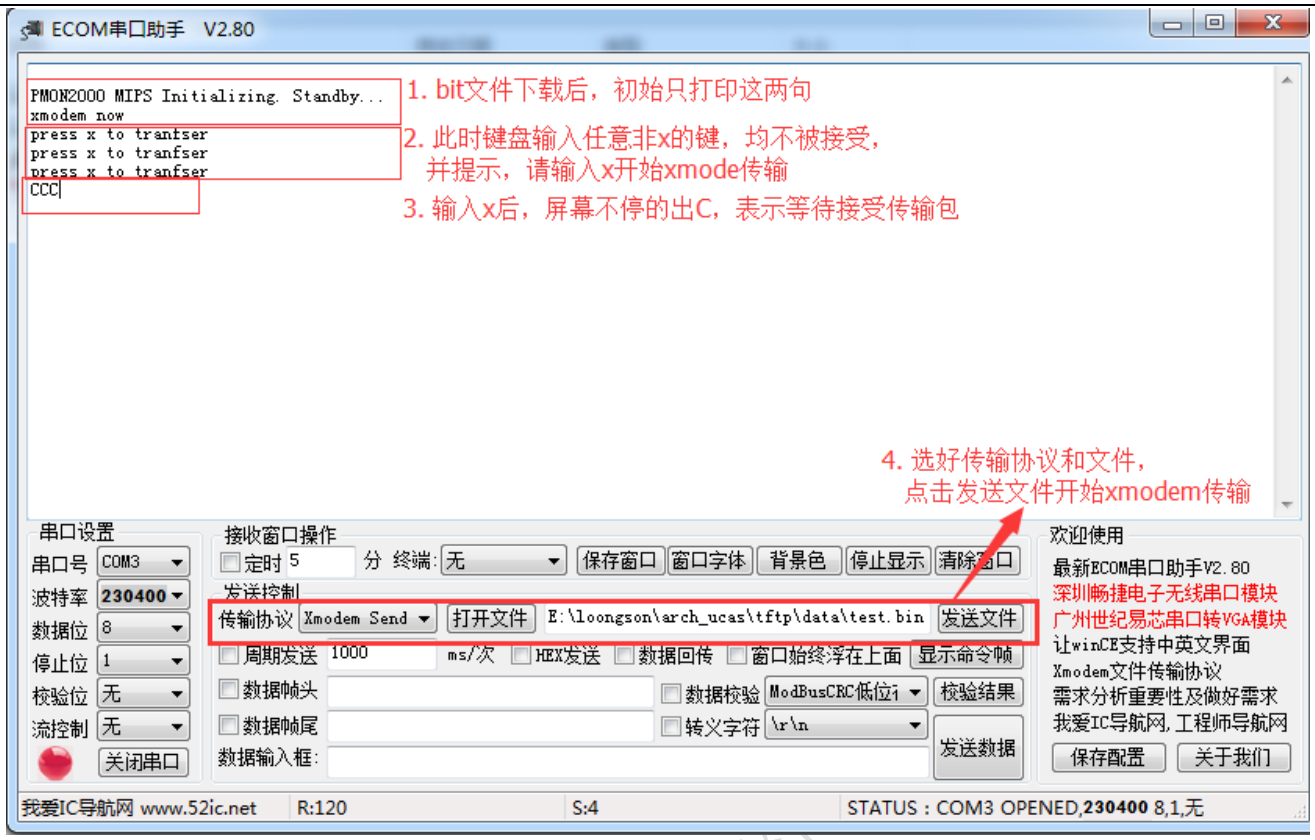
- FPGA 开发板。
- FPGA 电源线。
- FPGA 下载线。
- flash 芯片。
- 串口线（可能还需要 usb 转串口线）。
- xilinx 下载工具。
- 串口软件（ECOM 或 SecureCRT）

(3) 烧写步骤：

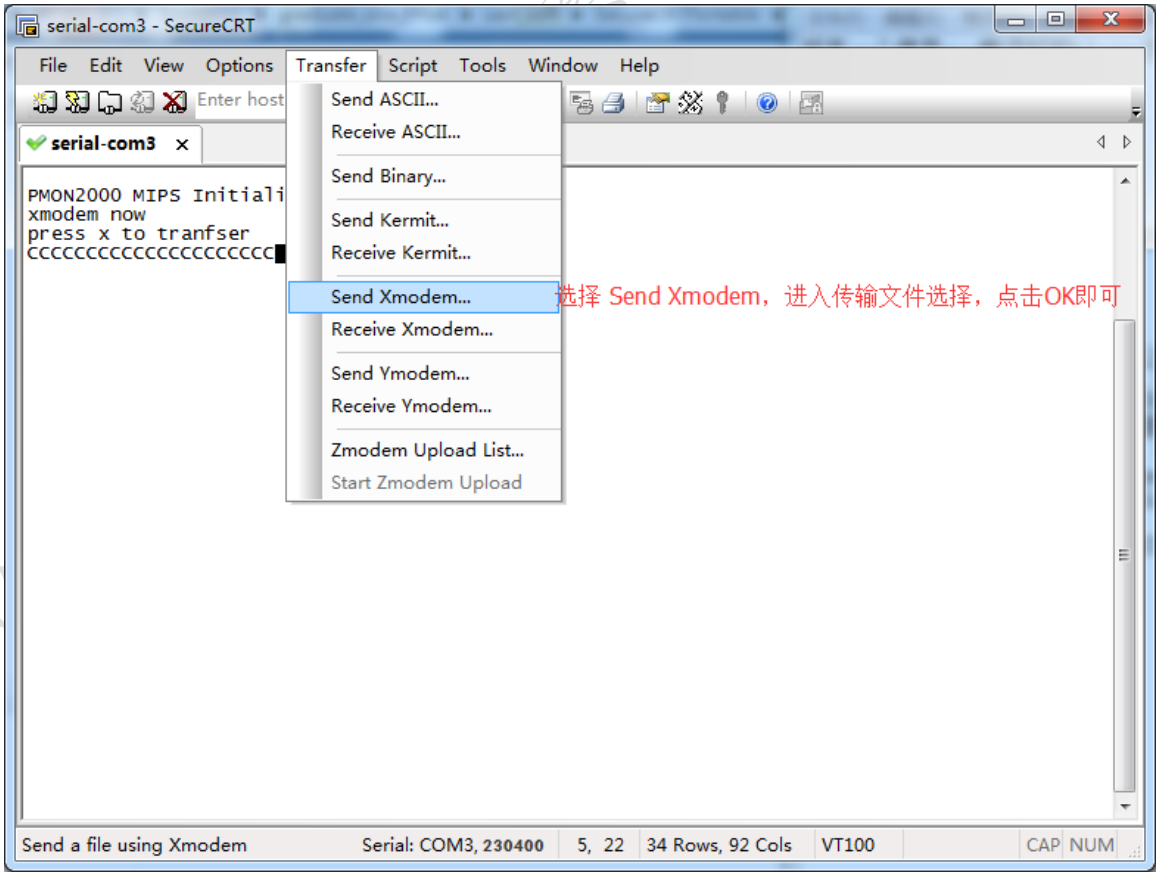
- flash 芯片正确放置 FPGA 开发板上。
- FPGA 开发板与电脑连接下载线、串口线。
- 电脑上打开 Vivado 工具中的 Open Hardware Manager，打开串口软件。
- FPGA 板上电，如正常下载 bit 流文件一样下载 programmer_by_uart.bit 至 FPGA 上。
- 打开串口软件，波特率选为 230400，连接正常后 根据提示，键盘输入 x 表示开始 xmodem 传输
- 串口软件使用 xmodem 模式传输 binary 文件。等待传输完成。

Linux 环境下可以使用串口进行 xmodem 传输完成 flash 烧写的。

Windows 下，如果串口软件使用的是 ECOM 软件，烧写步骤如下。ECOM 传输 xmodem 的进度会弹出一个界面进行显示，可以看到每个传输包的情况。



如果是串口软件使用的是 SecureCRT 软件，烧写步骤如下。SecureCRT 会在串口界面显示传输百分百。



注意： 每次需要烧写 flash 时都需要下载一次 programmer_by_uart.bit 文件，不是很方便。可以考虑将 bit 文件

转换为 mcs 文件，固化到 FPGA 开发板上。固化方法，参见本章 1.6 节。

1.5 FPGA 板上 SoC_up 下载

使用 FPGA 开发板运行教学 SoC 时，需要下载对应的 bit 流文件。

目前教学 SoC 在 Artix-7 实验板上运行的频率为 33MHz，DDR3 控制器的工作频率为 400MHz，上升下降沿均采集数据。

下载工具可选：

(1) ISE 工具套装里的 ChipScope。

ise13.2 的 ChipScope 不支持下载 Artix-7 的 bit 流文件，建议使用更高版本的 ChipScope，目前实验了 14.3 版本的是可以用的。

(2) vivado 工具里的 Open Hardware Manager。

1.6 Artix-7 教学实验板固化方法

本节给出基于 Artix-7 的教学实验板上固化一个 FPGA 设计的方法。

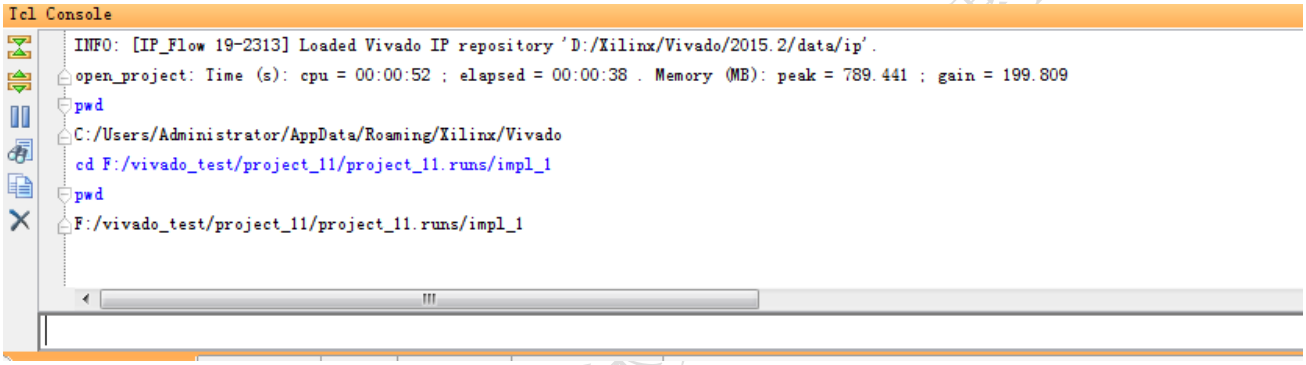
固化后，每次上电时，实验板会自动加载设计到 FPGA 芯片上。因而断电重新上电后不需要重新下载 bit 流文件，极大方便了实验板上软件编程。

固化的流程：先将一个 bit 流文件转换为 mcs 文件，将 mcs 文件下载到板上一个 SPI flash 上。

1.6.1 生成 mcs、prm 文件

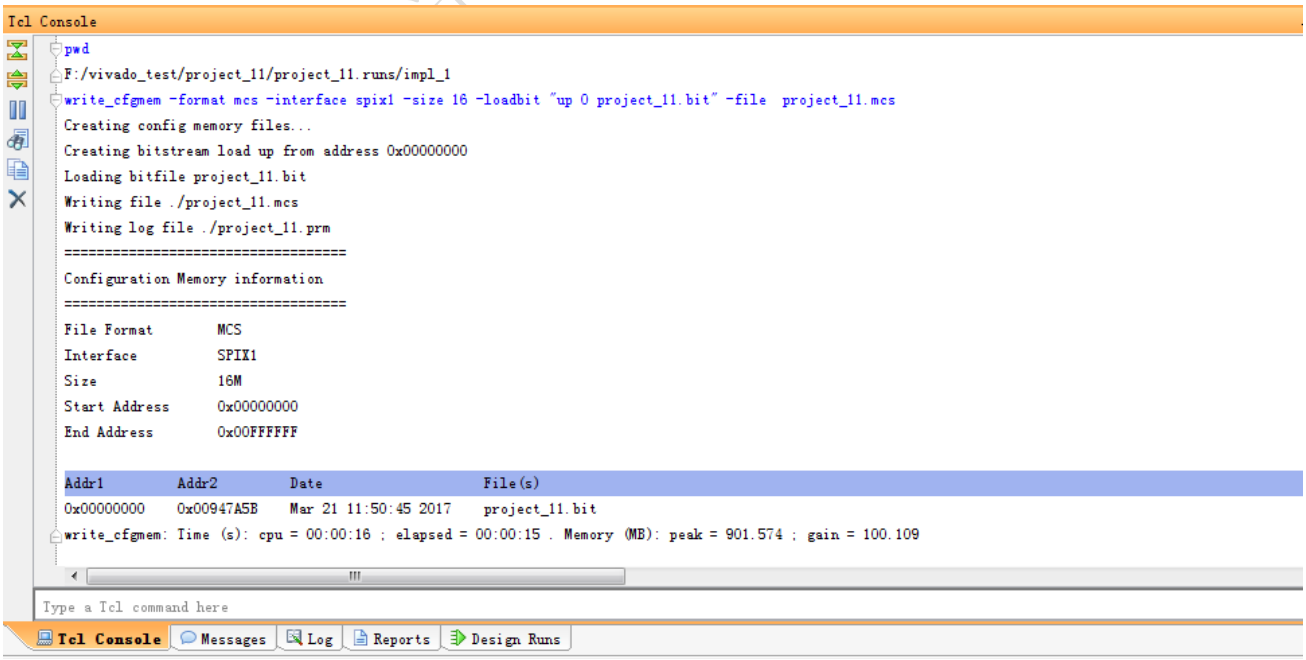
首先，需要确保 FPGA 设计的 bit 流文件已经生成。但 bit 流文件是用于直接下载到 FPGA 芯片里的文件，而不能下载到 flash 芯片里，因而需要转换为 mcs 文件。

在 ISE 工具里，可以再图形界面下点击选择就可生成 mcs 文件，但在 Vivado 工具里，生成 mcs 文件需要在命令控制器（tcl console）里输入命令。如下图：



上图中蓝色的为输入的命令，pwd 用于查看目录。随后使用 cd 命令进入 bit 流文件所在的目录。

假设生成的 bit 流文件为 project_11.bit，则输入命令串 **write_cfgmem -format mcs -interface spix1 -size 16 -loadbit "up 0 project_11.bit" -file project_11.mcs** 即可生成 mcs 文件，如下图。



其中命串里的 project_11.bit 为待转换的 FPGA 设计的 bit 文件，project_11.mcs 为生成的 mcs 文件名，可以自定义。

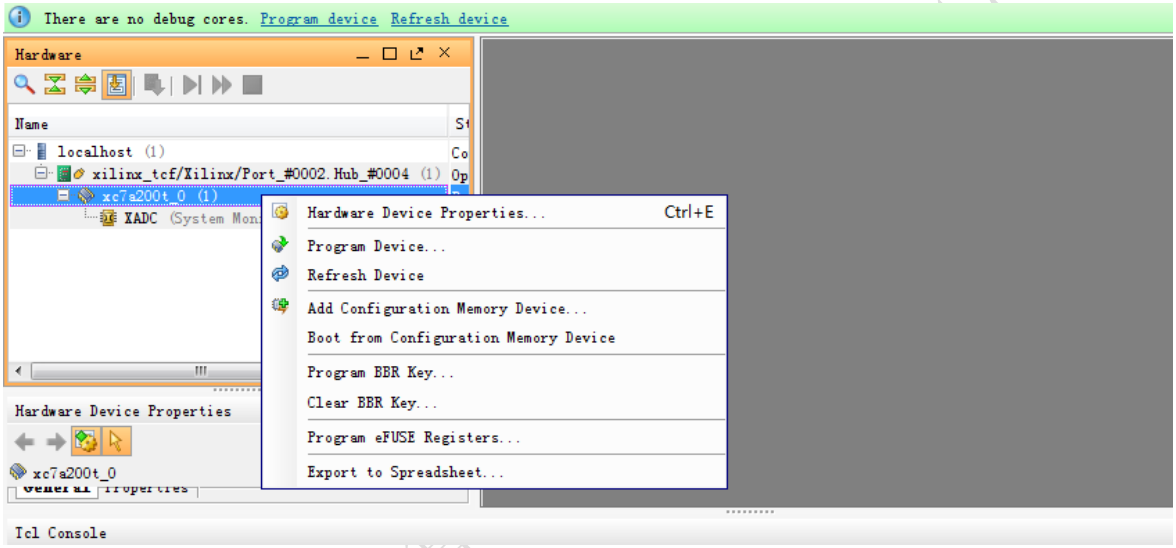
上述命令，是先输入 cd 命令将目录切换到了 bit 流文件的目录，再使用 write_cfgmem 命令将 bit 流文件转换为 mcs 文件。这两步可以使用命令串 **write_cfgmem -format mcs -interface spix1 -size 16 -loadbit "up 0 F:/vivado_test/project_11/project_11.runs/impl_1/project_11.bit" -file F:/vivado_test/project_11/project_11.runs/impl_1/project_11.mcs** 一步到位。这一命令串中明确指定了 bit 流文件的目录，和生成的 mcs 文件的保存目录，bit 流文件的目录和文件名必须正确，但 mcs 文件的目录和文件名可以自定义。

在生成 mcs 的同时，也会生成一个 prm 文件，后续都需要下载。

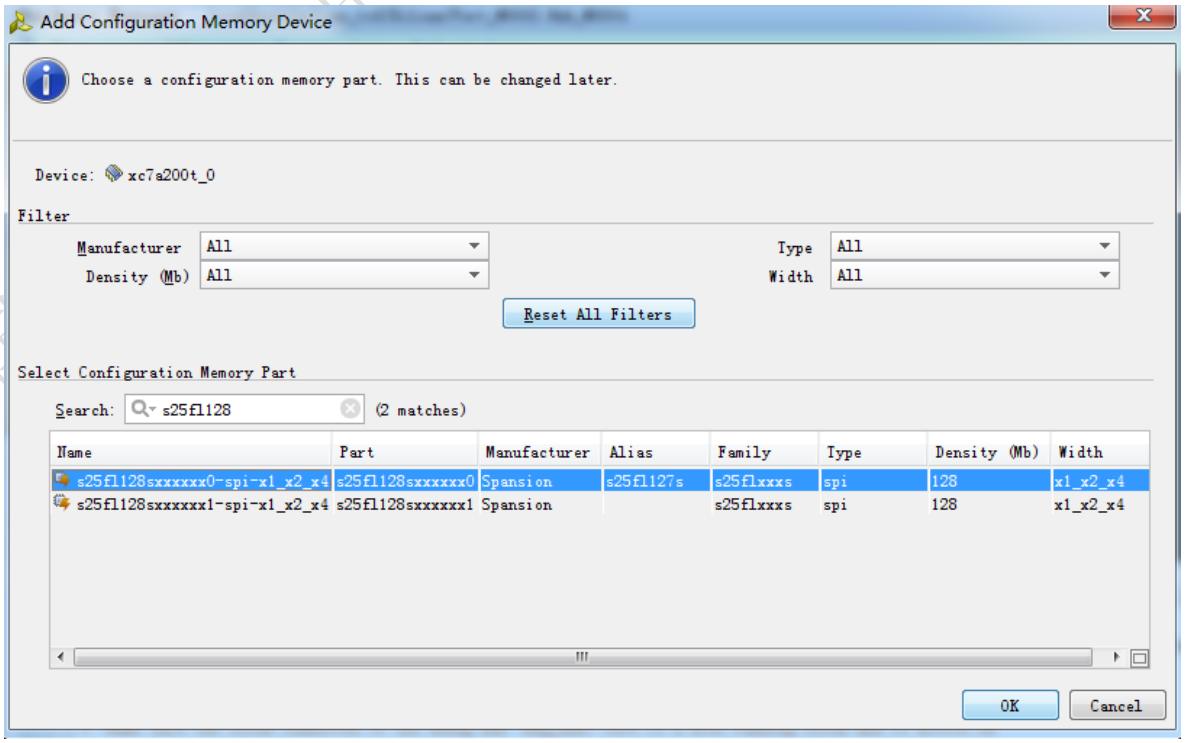
1.6.2 下载 mcs、prm 文件

生成好 mcs、prm 文件后，就需要将它们下载到实验板上 SPI flash 上。

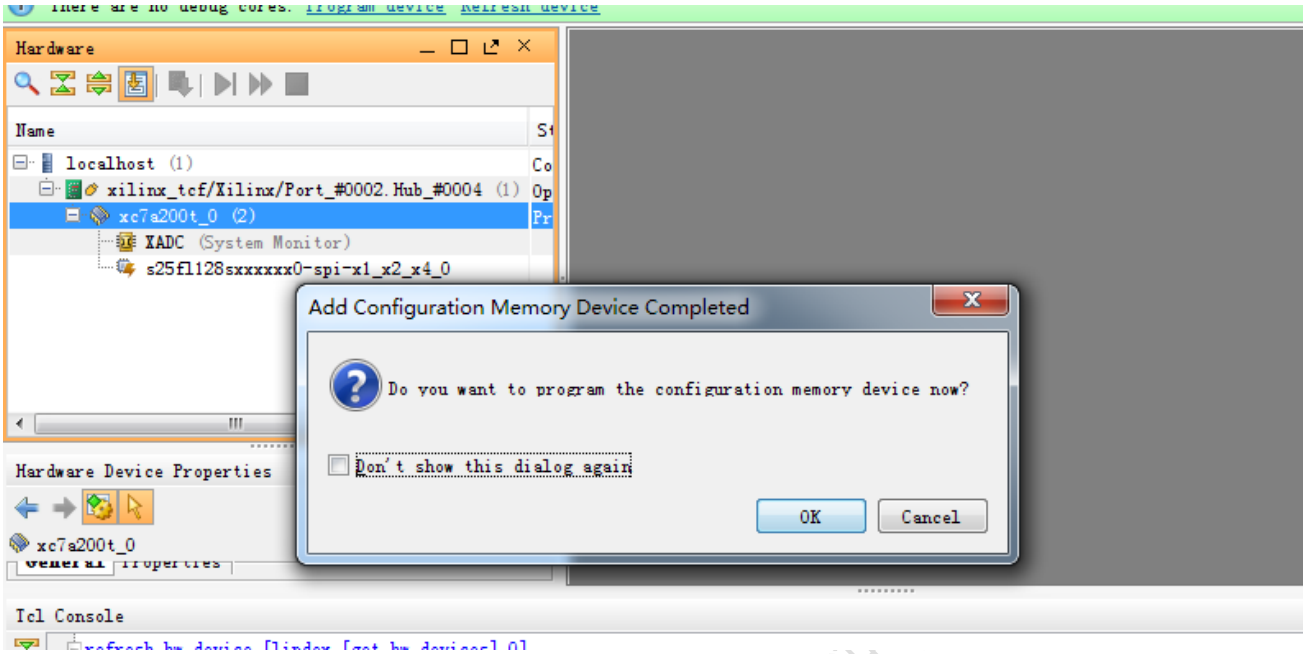
像下载 bit 流文件一样，打开 Vivado 工具里的“Open Hardwar Target”，连接设备。



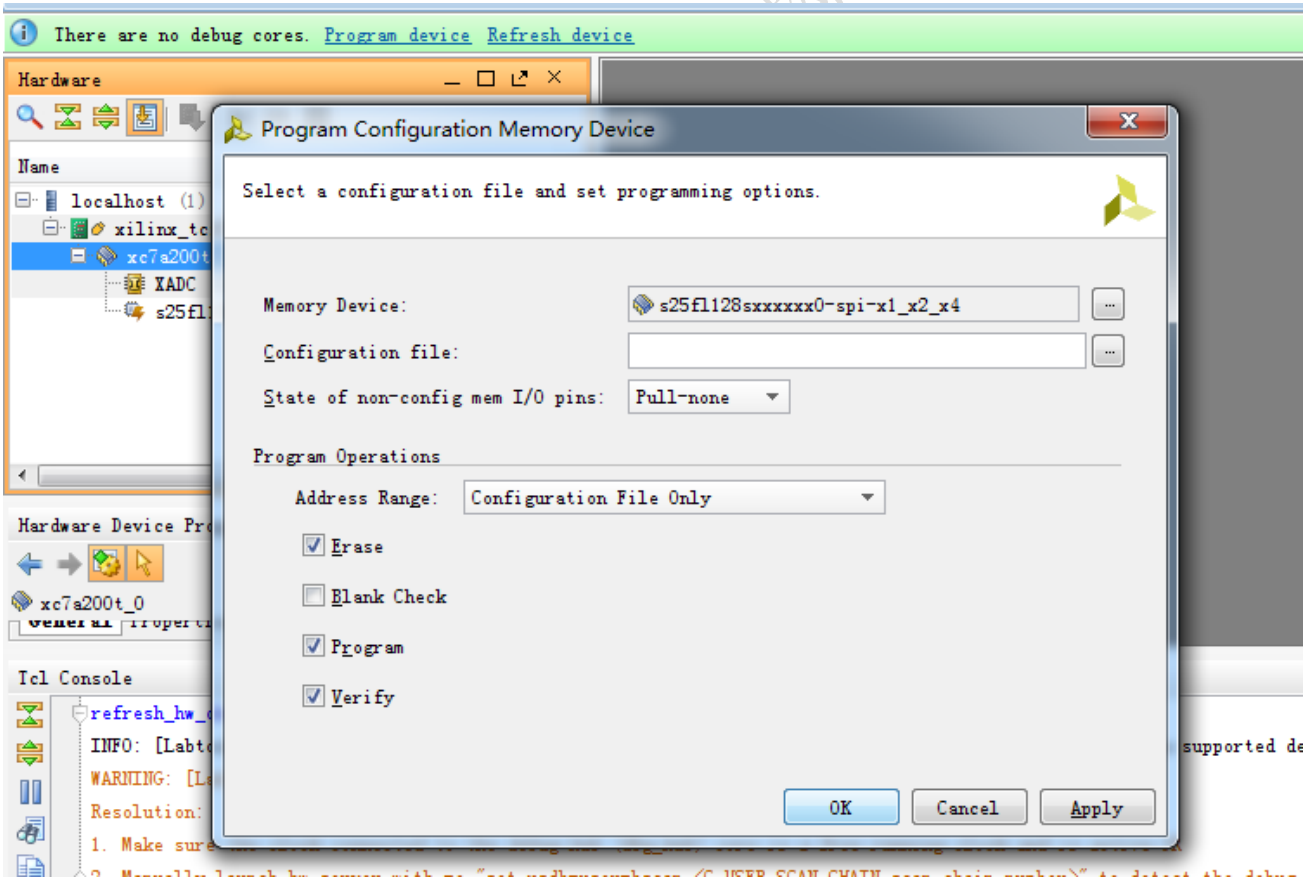
选中 xc7a200t 后，右键选择“Add Configuration Memory Device”，出现如下界面：



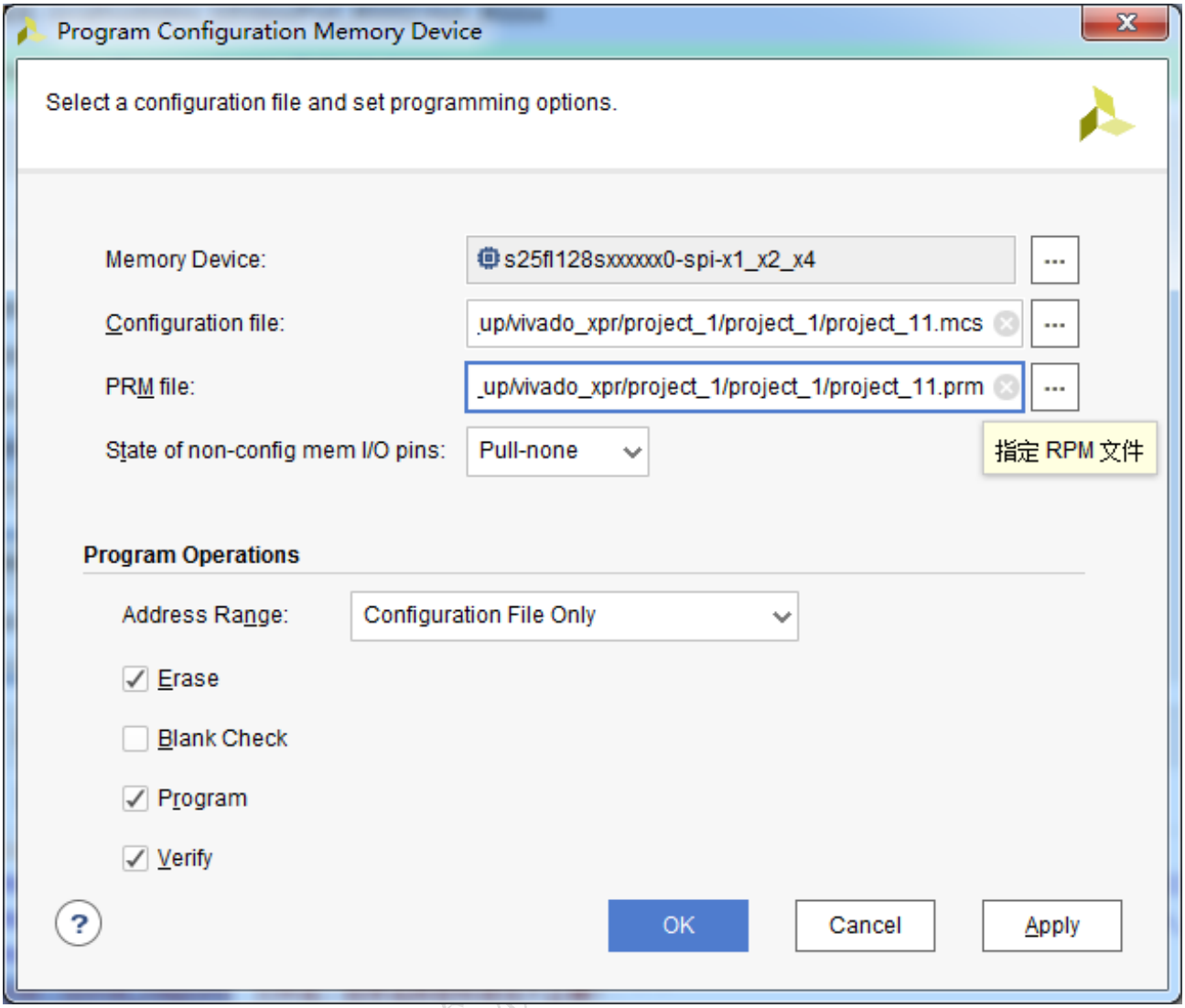
在 search 栏输入 s25fl128，选中出现的第一个 spi flash 芯片“s25fl128sxxxxxx0-spi-x1_x2_x4”(需要与板上固定的 flash 芯片型号相同，如果第一个不行，就选第二个 spi flash)，点击 OK，弹出如下窗口，询问是否现在编程 flash：



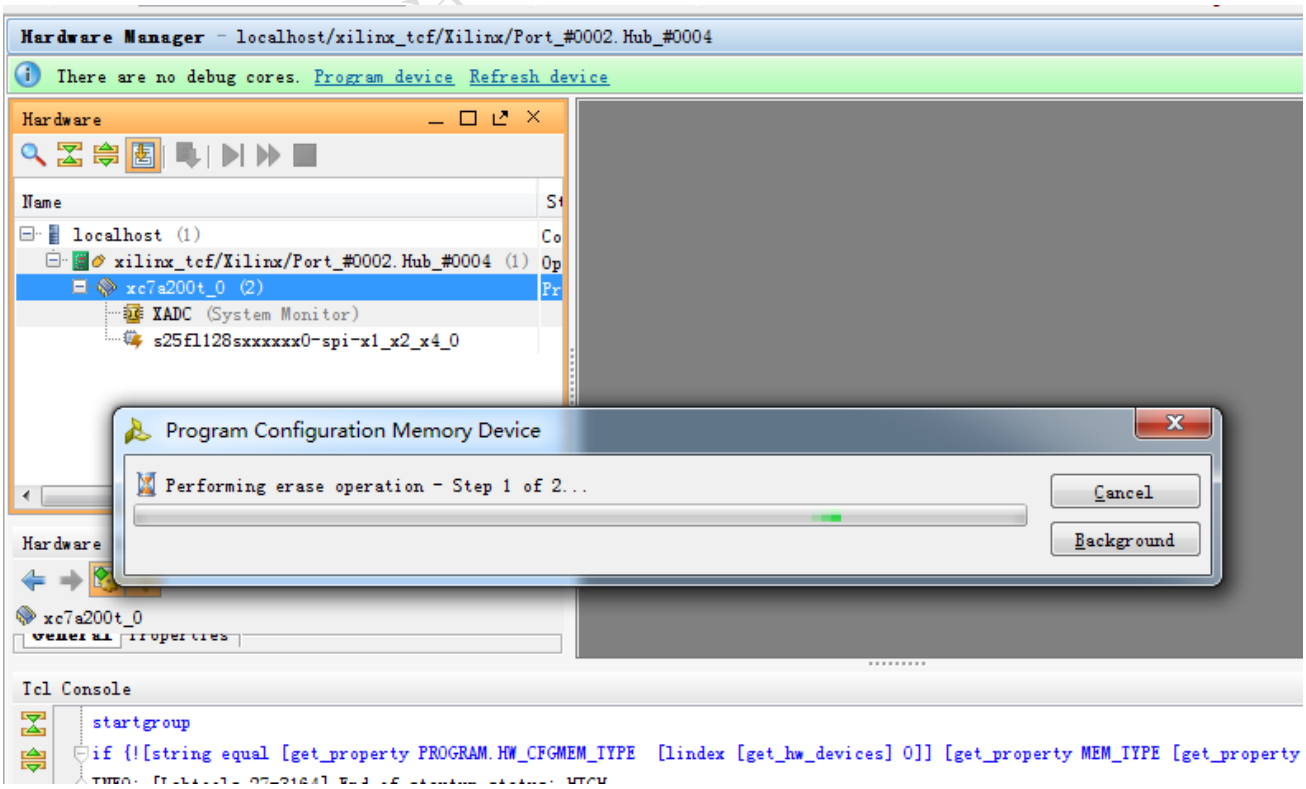
点击 OK，出现编程 flash 的界面：



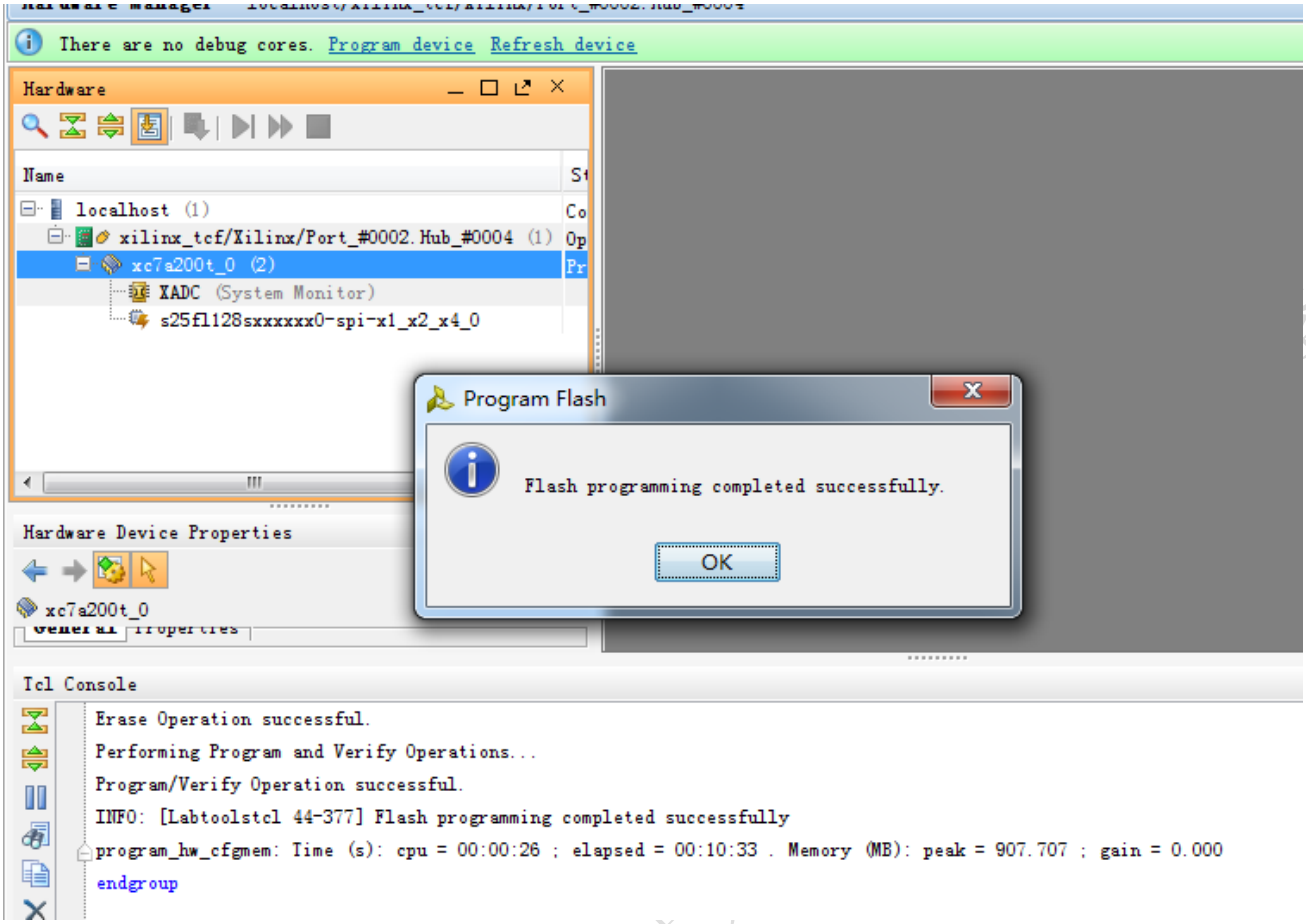
在 Configuration file 那栏选到生成的 mcs 文件，早 PRM file 那栏选到生成的 prm 文件，如下图：



点击 OK 即可。后续等待下载 mcs 到 flash 芯片完成即可，如下图：



Flash 芯片会先进行擦除，在进行编写，完成后会提示 completed Successfully，如下图：



此时烧写就完成了，需要将实验板断电重新上电，等待一段时间，就可发现固化进实验板的 FPGA 设备自动加载完成了。（**注意：断点上电后，需要关闭“Open Hardwar Target”，FPGA 才会完成自动加载固化后的设计**）

1.7 加载内存到 NandFlash

当前 SoC_up 支持 128MB 的 NandFlash 作为电脑中的硬盘功能。因而可以将 Linux 内核加载到 NandFlash 上。

如果可插拔 SPI 中 Flash 芯片的 PMON 是完善的且已配置后，且实验箱中的 SoC_up_33Mz.bit 也已固化。则下次实验箱上电后，会先自动加载 SoC_up_33M.bit 运行 SoC_up，随后自动运行 PMON 对设备进行初始化，随后 PMON 会自动加载 NandFlash 中的 Linux 内核进行启动，这就是通常电脑启动的过程。

Linux 内核加载至 NandFlash 中并配置 PMON 的方法如下：

(1) 实验箱运行至 PMON；

(2) 设置分区空间大小，PMON 命令：set mtdparts nand-flash:50M@0(kernel)ro,-(rootfs)；

(3) 设置网口 IP，PMON 命令：ifconfig dmfe0 x.x.x.x，其中 x.x.x.x 为配置的 IP；

(4) 擦除 NandFlash，PMON 命令：mtd_erase /dev/mtd0r

mtd_erase /dev/mtd1r

以上两个命令均要执行一次。

(5) 拷贝内核文件，PMON 命令：devcp tftp://x.x.x.x /vmlinux /dev/mtd0，其中 x.x.x.x 为搭建的 tftp 服务器的 IP；

(6) 设置启动分区及参数，PMON 命令：

set al /dev/mtd0；

set append "console=ttyS0,115200 rdinit=/sbin/init initcall_debug=1 loglevel=20"

(7) 重启 FPGA 实验箱，会自动完成本章开头描述的启动过程，自动运行到 Linux 内核状态：