龙芯 1B 系统开发指导

V1.1

2012-05

1 龙芯 1B 处理器简单介绍

龙芯 1B 一款实现 MIPS32 兼容且支持 EJTAG 调试的双发射处理器，通过采用转移预测、寄存器重命名、乱序发射、路预测的指令 CACHE、非阻塞的数据 CACHE、写合并收集等技术来提高流水线的效率，1B 芯片具有以下关键特性：

- 集成一个 GS232 双发射龙芯处理器核，主频 266MHZ,指令和数据 L1 Cache 各 8KB
- 集成一个 24 位 LCD 控制器，最大分辨率可支持到 1920*1080@60Hz/16bit
- 集成 2 个 10M/100M/1000M 自适应 MAC
- 集成 1 个 32 位 133MHz DDR2 控制器，兼容 16 位 DDR2
- 集成 1 个 USB 2.0 接口，兼容 EHCI 和 OHCI
- 集成 1 个 8 位 NAND FLASH 控制器,最大支持 32GB
- 集成中断控制器，支持灵活的中断设置
- 集成 2 个 SPI 控制器，支持主从模式，支持系统启动
- 集成 AC97 控制器
- 集成 2 个全功能串口，4 个两线串口（最高 12 个双线串口）
- 集成 3 路 I2C 控制器，兼容 SMBUS
- 集成 2 个 CAN 总线控制器
- 集成 62 个 GPIO 端口
- 集成 1 个 RTC 接口
- 集成 4 个 PWM 控制器
- 集成看门狗电路

2 调试开发环境搭建

2.1 开发环境搭建

在嵌入式开发过程中，通常由于目标板（开发板）没有足够的资源来运行开发和调试工具，所以需要借助建立了交叉编译环境的宿主机。开发环境的搭建主要有以下几个方案：

a,直接安装 linux 操作系统。

b,在 WINDOWS 下安装虚拟机后,再在虚拟机中安装 LINXUX 操作系统;

c,两台电脑，一台 linux 服务器，一台 windows 客户端。

我们这里对这几种方案都做一些简单的介绍：

a,直接安装 linux 操作系统。

这种方式适合对 linux 系统常用操作有一定了解，能在 linux 下进行程序编写，交叉编译的开发人员。

b，在 windows 中安装 linux 操作系统虚拟机方案。

这种方式适合对 linux 不是太了解，在 windows 下完成程序的编写工作，只在 linux 环境下进行交叉编译的开发人员。

c，一台 linux 服务器，多个 windows 客户端方案。

这种方式主要是方便多人同时开发，Linux 服务器用来进行交叉编译程序使用。Windows 客户端主要是通过 SSH 远程登录到 linux 服务器进行操作。

不管使用的哪种方式，最终交叉编译程序所使用的 linux 环境基本是一样的。对于所使用的 linux 操作系统（我们没有特殊要求，只要 linux 系统能够正常工作即可）安装好网络服务，TFTP 服务，NFS 服务就可以（这里不在讲述各种服务的搭建）。

2.2 交叉编译工具安装

交叉编译环境的安装配置：

a，将交叉编译工具(gcc-4.3-ls232.tar.gz)拷贝到 linux 下，进行解压缩。

命令：*tar xzvf gcc-4.3-ls232.tar.gz －C /*

这样会在/opt/安装交叉编译工具。

b，为交叉编译工具的路径添加环境变量

命令：*export PATH=/opt/gcc-4.3-ls232/bin:$PATH*

输入命令：mipsel-linux-gcc -v 查看交叉编译工具版本信息，来验证交叉编译工具是否正常安装（提示版本信息即认为交叉编译工具已经正常安装）。

注：

这条命令的作用只在当前终端有效，即交叉编译环境只在当前终端起作用。

需要在整个系统建立交叉编译环境，可以把

export PATH=/opt/gcc-4.3-ls232/bin:$PATH 添加到 /root/.bashrc 文件最后一行。

也可以运行命令:

echo "export PATH=/opt/gcc-3.4.6-2f/bin:$PATH" >>/root/.bashrc。

然后打开新终端切换到超级用户权限（sudo su），运行 echo $PATH 查看验证。

如果需要交叉编译程序只要指定交叉编译工具即可（一般的交叉编译程序会有 CROSS_COMPILE 和 ARCH

两个变量，只要指定 CROSS_COMPILE=mipsel-linux- ARCH=mips 就可以了；或者只有 CC 变量 这时只要指定 CC=mipsel-linux-gcc 就可以了）。

2.3 常用调试环境搭建

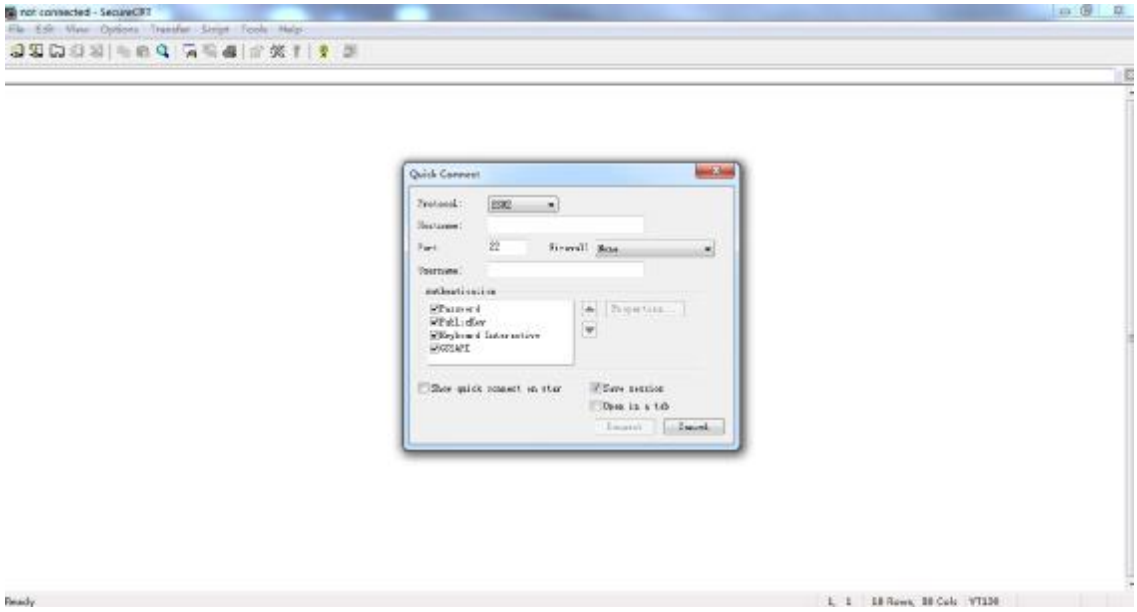
为了通过串口连接开发板，需要在 PC 机上使用一个终端仿真程序。

如果 PC 机是 Windows 操作系统，可以使用 SecureCRT 或者超级终端。Windows XP 上自带有超级终端软件，位于“开始-->程序-->附件-->通讯”，而 Windows 7 上需要额外安装，超级终端的设置请参考**“附录 Windows 超级终端使用说明”**。

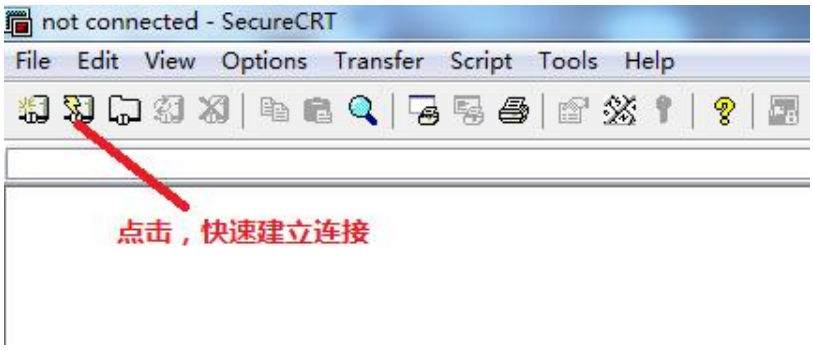
如果 PC 机是 Linux 操作系统，可以使用 minicom（详细请参考**“附录 Minicom 使用指南”**）。

下面以 Windows 操作系统上的 SecureCRT 终端仿真软件为例。

1）打开 SecureCRT，如下图：

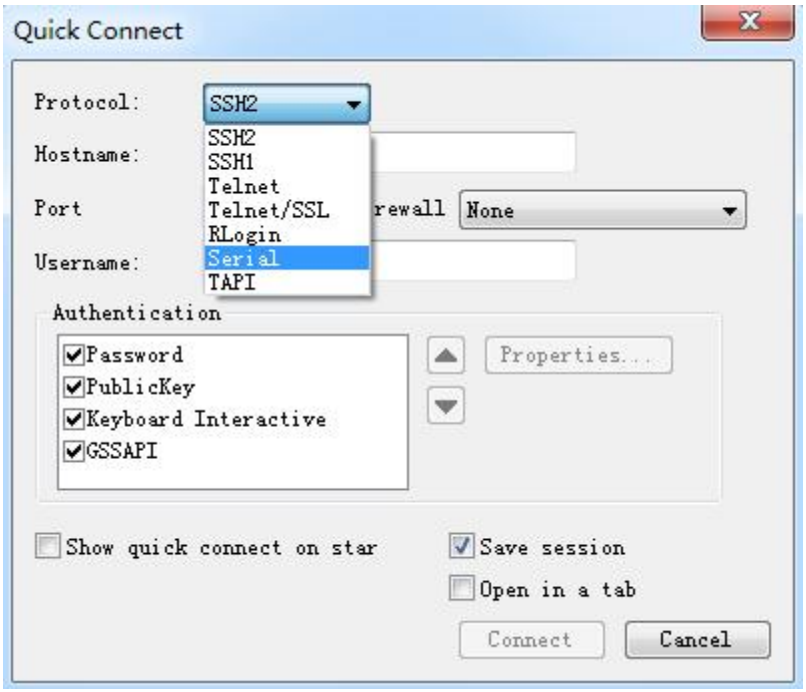


2）点击工具栏的带闪电图标，快速建立连接，如下图：

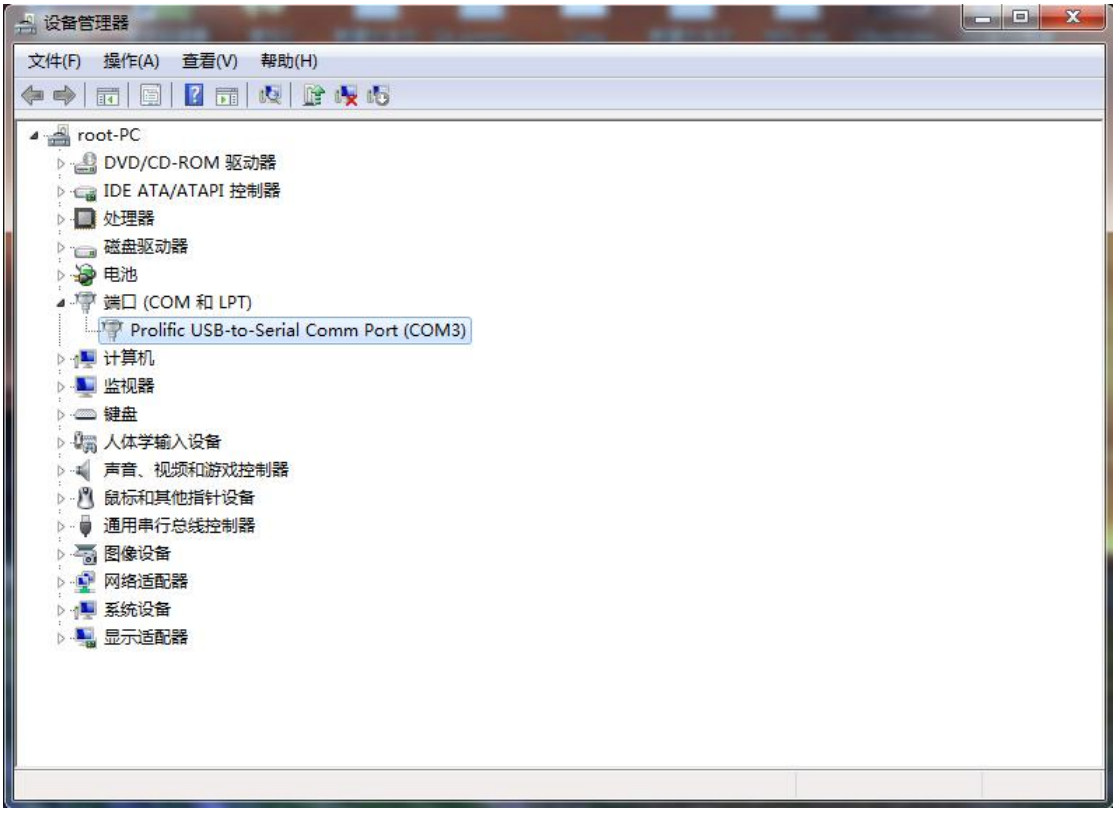


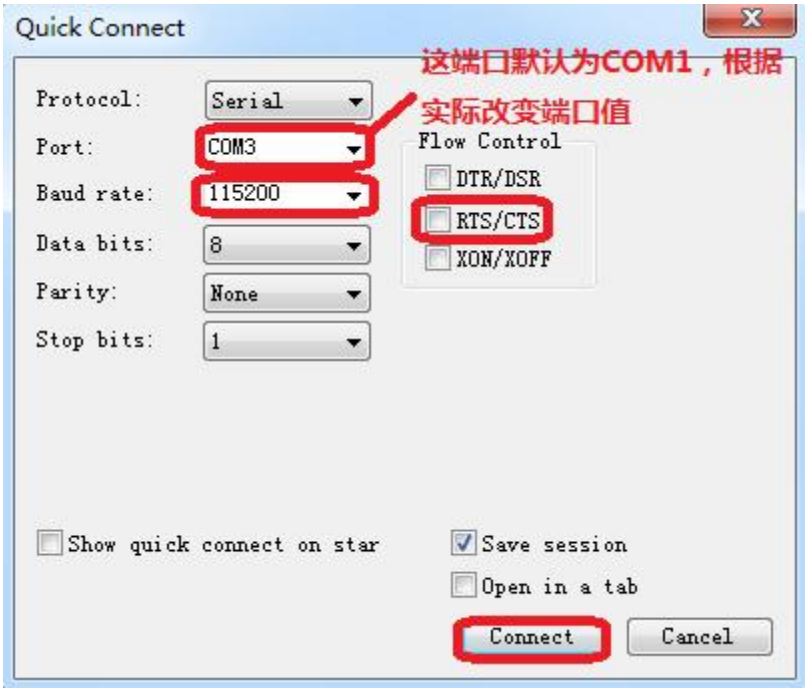
3）在 Quick Connect 对话框中，Protocol 的下拉菜单选择 Serial 协议选项，

选择串口协议，如下图：

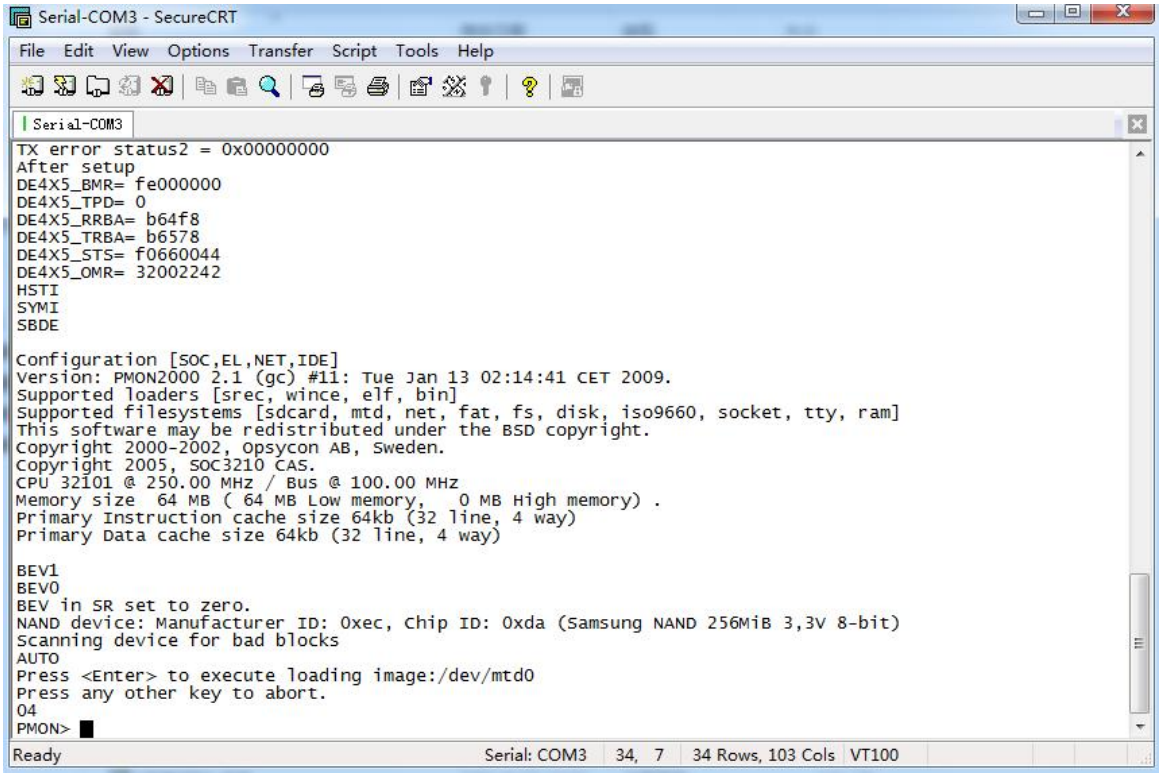


4）在 Quick Connect 对话框中，设置端口 Port 为 COM3，在“设备管理器”中可以看到串口的端口信息（注意：应根据具体信息设置正确的端口）；设置波特率 Baud rate 为 115200；去掉 Flow Control 中所有的勾选；最后 Connect，如下图：





5）开发板上电，按空格键，进入 BIOS（PMON），如下图：



3 基本开发

3.1 PMON

PMON 是一个兼有 BIOS 和 bootloader 部分功能的开放源码软件，多用于嵌入式系统。基于龙芯的系统采用 PMON 作为类 BIOS 兼 bootloader，并在其基础上做了很多完善工作，支持 BIOS 启动配置，内核加载，程序调试，内存寄存器显示、设置以及内存反汇编等等。

3.1.1 Pmon 编译

工具与依赖库安装

工具与依赖库的安装有两种方式：有互联网时使用 apt-get（ubuntu 系统）命令来安装；无互联网时使用源码包来安装。对于我们这里**例子**使用的 ubuntu 系统使用 apt-get 命令安装，要求 PC 机连接有互联网。

1）因为编译 PMON 过程需要使用到工具 pmoncfg，该工具 1b-pmon/tools/pmoncfg 目录下，编译该工具又需要依赖下面的工具：

```
#apt-get install bison
#apt-get install flex
```

2）解压 1b-pmon.tar.gz，编译生成 pmoncfg 工具：

```
#tar xzvf 1b-pmon.tar.gz
#cd 1b-pmon
#cd 1b-pmon/tools/pmoncfg
#make
```

3）编译完成后会在当前目录下生成 pmoncfg，拷贝该工具至用户工具目录或交叉编译工具链的 bin 目录（参看建立交叉编译环境）下。（推荐拷贝至交叉编译工具链目录中）

```
#cp pmoncfg /opt/gcc-4.3-ls232/bin
```

4）pmon 编译还依赖于工具 makedepend：

```
#apt-get install xutils-dev
```

配置与编译 pmon

1）建立交叉编译环境（第二章建立交叉编译环境）。

2）解决库与工具依赖以后，开始编译 pmon：

```
#cd 1b-pmon/zloader.ls1b
```

3）编译 bin 格式的 pmon

```
#make cfg ;make tgt=rom
```

执行后就在当前目录下生成了 gzrom.bin。

3.1.2 Pmon 更新

将新编译好的 pmon 文件（gzrom.bin）拷贝到 tftp 目录（确认能够正常访问 tftp 服务），设置好板子的 IP 地址，使用命令来更新应用新的 pmon。

1） 设置板子 IP 地址（参见 PMON 常用命令）

2）更新 PMON 命令：load –r –f 0xbfc00000 tftp://tftp-server-ip/gzrom.bin

该命令在执行完毕后会自动提示新 PMON 更新完成。重新启动即可应用新的 PMON。

注： a，在这里我们只介绍在现有的 PMON 之上更新新的 PMON。

b,在整个更新 Pmon 的过程中，注意不要断电。

3.1.3 Pmon 常用命令

参见附录

3.2 Kernel

Linux 内核很庞大，linux 初学者以及致力于 linux 应用软件开发的技术人员，熟悉内核的好的开始就是对内核进行配置，得到符合自己需求的经过裁剪的内核，并将编译后的内核下载到开发板中运行使用。本篇内容是为想要对内核进行个性配置的人员准备的，不涉及到代码编写，学习 linux 不必一切从“零”开始，一切可从学会配置、编译、下载运行开始。Linux 内核的编译分为两个步骤，一、内核配置；二、内核编译。开发包默认提供一个配置文件，用户可以根据此配置文件对内核进行裁剪或者增加新的功能。

注： Linux-2.6.21 内核为例，描述整个 Linux 系统的配置编译更新运行过程。

3.2.1 Kernel 编译

1）建立交叉编译环境（参见交叉编译工具安装和环境搭建）。

2）解压缩并进入 linux 源代码树根目录。

命令： ***tar xzvf 1b-linux-2.6.21.tar.gz***
cd 1b-linux-2.6.21

3）确认安装图形化配置“make menuconfig”依赖的工具 Ncurses（如果安装过跳过此步骤）：

命令：***#apt-get install libncurses5-dev***

4）图形化配置(调整终端窗口到合适的大小)：

命令：***make menuconfig***

注： a，在内核的源代码中我们会提供一个默认的配置文件（名字为 config.def）。
b，参见附录中内核常用配置说明。

3.2.2 内核的编译

最终在内核源码树跟目录中，生成内核映像文件 vmlinux。

命令： ***make vmlinux***

mipsel-linux-strip vmlinux (去除无用的调试和符号信息来减小内核的大小)

注： 如果在环境变量中没有指定 CROSS_COMPILE 和 ARCH，需要指定这两个环境变量的值（参见交叉编译工具安装），CROSS_COMPILE=mipsel-linux- ARCH=mips

3.2.3 Kernel 更新

将新编译好的内核文件（vmlinux）拷贝到 tftp 目录（确认能够正常访问 tftp 服务），设置好板子的 IP 地址，使用命令来更新应用新的内核（此操作在 PMON 中进行）。

1）设置板子 IP 地址（参见 PMON 常用命令）

2）使用命令：*devcp tftp://tftp-server-ip/vmlinux /dev/mtd0* 将内核烧写到 nandflash 中。

3）设置启动参数：*set al /dev/mtd0*

set append “console=ttyS0,115200 rdinit=/sbin/init”

注： 这里设置的参数请根据实际情况修改。但一定要设置上 al 和 append 两个环境变量。

4 文件系统

4.1 文件系统制作

说明：这里只介绍在提供的 ramdisk 基础上如何修改编辑文件系统内容,如要定制自己全新的文件系统,请参考网上资料。

首先将 ramdisk-fs.tar.gz 文件系统拷贝到一个目录下（假设：是 path 目录），解压,然后挂载到 /mnt/ramdisk-fs 目录

命令： ***tar xzvf ramdisk-fs.tar.gz -C path***
mkdir -p /mnt/ramdisk-fs
mount -o loop -t ext2 path/ramdisk-fs /mnt/ramdisk-fs

修改自启动文件 /mnt/ramdisk-fs/etc/init.d/rcS

命令 ： ***vim /mnt/ramdisk-fs/etc/init.d/rcS***
添加需要的命令后，保存退出。

卸载制作好的文件系统：

命令：*umount /mnt/ramdisk-fs*

这样文件系统就制作完成。

4.2 文件系统烧写

挂载制作好的文件系统到 /mnt/ramdisk-fs

命令：*mount -o loop -t ext2 ramdisk-fs /mnt/ramdisk-fs*

4.2.1 yaffs2

使用 yaffs2 文件系统格式，并将文件系统烧写到 nandflash 上面;

1）使用 yaffs2 镜像制作工具制作 yaffs2 的镜像文件 rootfs-yaffs2.img。

#mkyaffs2image /mnt/ramdisk-fs rootfs-yaffs2.img
#chmod 777 rootfs-yaffs2.img **//修改文件系统权限，防止出现无法烧写的情况**

2）将制作好的 yaffs2 镜像文件(rootfs-yaffs2.img)烧写到 nandflash 上面（这里提供通过网络烧写的命令）。

命令：devcp tftp://tftp-server-ip/rootfs-yaffs2.img /dev/mtdxy

注： 这里的/dev/mtdx 指代是 nandflash 分区，参见附录 nandflash 分区。

4.2.2 jffs2

使用 jffs2 文件系统格式，并将文件系统烧写到 nandflash 上面;

1）使用 jffs2 镜像制作工具制作 jffs2 的镜像文件 rootfs-jffs2.img。

```
#mkfs.jffs2 -r /mnt/ramdisk-fs -o rootfs-jffs2.img -e 0x20000    --pad=0x2000000 -n
#chmod 777 rootfs-jffs2.img          //修改文件系统权限，防止出现无法烧写的情况
```

mkfs.jffs2 各参数的意义：

-r：指定要生成 image 的目录名。

-o：指定输出 image 的文件名。

-e：每一块要擦除的 block size，不同的 flash，其 block size 会不一样。这里为 128KB。

--pad：用 16 进制来表示所要输出文件的大小，也就是 rootfs-jffs2.img 的大小，如果实际大小不足此设定的大小，则用 0xFF 补足。

-n，-no-cleanmarkers：指明不添加清楚标记（nandflash 有自己的校检块，存放相关的信息）。

2） 将制作好的 jffs2 镜像文件(rootfs-jffs2.img)烧写到 nandflash 上面（这里提供通过网络烧写的命令）。

```
命令：devcp tftp://tftp-server-ip/rootfs-jffs2.img   /dev/mtdx
```

注：这里的/dev/mtdx 指代是 nandflash 分区，参见附录 nandflash 分区。

4.2.3 cramfs

使用 cramfs 文件系统格式，并将文件系统烧写到 nandflash 上面;

1） 使用 cramfs 镜像制作工具制作 jffs2 的镜像文件 rootfs-cramfs.img

```
#mkfs.cramfs   /mnt/ramdisk-fs     rootfs-cramfs.img
#chmod 777 rootfs-cramfs.img        //修改文件系统权限，防止出现无法烧写的情况
```

2） 将制作好的 cramfs 镜像文件(rootfs-cramfs.img)烧写到 nandflash 上面（这里提供通过网络烧写的命令）。

```
命令：devcp tftp://tftp-server-ip/rootfs-cramfs.img   /dev/mtdx
```

注：这里的/dev/mtdx 指代是 nandflash 分区，参见附录 nandflash 分区。

附录

1 Pmon 常用命令

1.1 常用命令表

类型	命令	说明	例子	例子含义
帮助	h	查看帮助信息	h	列出所有可以使用命令
			h ping	查看 ping 命令的用法
调试	d1	读某个地址的值(读一个 byte)	d1 0x80300000	查看地址 0x80300000 处的值
	d2	读某个地址的值(读一个 halt word)	d1 0x80300000	查看地址 0x80300000 处的值
	d4	读某个地址的值(读一个 word)	d4 0x80300000	查看地址 0x80300000 处的值
	m1	在某个地址处写入一个值(写入一个 byte 大小)	m1 0x80300000 0x12	在地址 0x80300000 处写入 0x12
	m2	在某个地址处写入一个值(写入一个 halt word 大小的值)	m2 0x80300000 0x1234	在地址 0x80300000 处写入 0x1234
	m4	在某个地址处写入一个值(写入一个 word 大小的值)	m4 0x80300000 0x12345678	在地址 0x80300000 处写入 0x12345678
				读出此地址值为 0x12345678
内存	mt	内存测试命令	mt	测试板的内存是否正常
	load	下载 linux 内核到内存	load tftp://192.168.3.18/vmlinux	从网络 IP 为 192.168.3.18 的主机下载内核 vmlinx 到内存

网络	ifaddr	设置板的 ip 地址(只当次有效，断电后会丢失)	ifaddr syn0 192.168.3.25	设置板的 ip 地址为 192.168.3.25
	ifconfig	查看设置的 ip 地址信息	Ifconfig syn0	查看网络口 syn0 的信息
	ping	测试网络	ping 192.168.3.1	测试与 192.168.3.1 网口是否连通
环境管理	set	设置环境变量；设置的参数会保存到 norflash 高位地址，在 pmon 一开始运行时就会自动去调用	set	列出所有已经设置好的环境变量
	ping	测试网络	set ifconfig syn0:192.168.3.88	设置开发板的 IP 地址，重启开发板后 IP 地址固定存在
flash 管理	set env devcp	设置环境变量；设置的参数会保存到 nandflash 高位地址，在 pmon 一开始运行时就会自动去调用 查看板上已经设好的环境变量 PMON 上的拷贝下载，通常拷贝从网络下载的文件到 Nand Flash 中	set al /dev/mtd0	自动从 nandflash 的 mtd0 分区 load 内存，设置板在一上电时自动执行 load 内 核到内存操作
			set append 'console=ttyS0'	设置板的运行的启动参数
			set	列出所有环境变量
			devcp tftp://192.168.3.18/vmlinux /dev/mtd0	从网络下载 vmlinux 到内存中并拷贝到 nandflash 中
	mtd_erase	擦除 Nand Flash 某分区的数据	mtd_erase /dev/mtd1	擦除 Nand Flash 分区 1 的数据
其他	reboot	重启 PMON	reboot	重启 PMON
	devls	查看设备列表	devls -n	查看网络设备
	mtdparts	查看 nandflash 分区	mtdparts	查看 nandflash 分区

1.2 常用命令解释

这里给出的命令解释只是基本的命令格式解释，关于详细的解释和使用方法可以使用（命令 –h ）的方式查看，如： ***devls –h***

d1

命令： **d1**
格式： **d1 <addr> <num>**
解释： display 简写，按无符号字符型（unsigned char）访问显示 num 个从地址 addr 开始的内容值。如：
d1 0x85000000 0x10
显示从内存地址 0x85000000 地址处的 0x10 个字节(8 位)，0x85000000-0x8500000f

d2

命令： **d2**
格式： **d2 <addr> <num>**
解释： display 简写，按（unsigned short）访问显示 num 个从地址 addr 开始的内容值。如：
d2 0x85000000 0x10
显示从内存地址 0x85000000 地址处的 0x10 个 unsigned short 类型(16 位)的数值，0x85000000-0x8500001e

d4

命令： **d4**
格式： **d4 <addr> <num>**
解释： display 简写，按（unsigned long）访问显示 num 个从地址 addr 开始的内容值。如：
d4 0x85000000 0x10
显示从内存地址 0x85000000 地址处的 0x10 个 unsigned long 类型(32 位)的数值，0x85000000-0x8500003c

d8

命令： **d8**

格式： **d8 <addr> <num>**

解释： display 简写，按（unsigned long long）访问显示 num 个从地址 addr 开始的内容值。如：

d8 0x85000000 0x10

显示从内存地址 0x85000000 地址处的 0x10 个 unsigned longlong(64 位)类型的数值，0x85000000-0x85000078

devcp

命令： **devcp**

格式： **devcp <src-device> <desc-device>**

解释： devcp 命令应该是 PMON 中比较重要的命令。其实现是：打开 src-device 和 desc-device 两个设备，从 src-device 读取一定数量的字节数据，写到 desc-device 设备中去，完成后关闭这两个设备。如：

devcp tftp://server-ip/vmlinux /dev/mtd0b

如上命令，从 tftp 服务器中接收读取内核 vmlinux，写到 nandflash 的第一个分区中，这是常用的烧写内核到 flash 中一个常用命令

devcp tftp://server-ip/roofs-yaffs2.img /dev/mtd1y

如上命令，从 tftp 服务器中接收读取根文件系统的 yaffs2 镜像文件 rootfs-yaffs2.img，烧写到 nandflash 的第二个分区中。这是常用的烧写 yaffs2 文件系统到 nandflash 中的一个常用命令。

devcp /dev/mtd0c /dve/ram@0x85000000,2112

如上命令，从 nandflash 的第一个分区读取数据写到，内存地址 0x85000000 的地方，写 2112 个字节。

注： 关于这里的设备/dev/mtd0b /dev/mtd1y /dev/mtd0c 的说明请参看附录中 nandflash 分区说明。

devls

命令： **devls**

格式： **devls**

解释： 查看 pmon 的部分设备，这里显示出来的设备不是全部的设备。一般会有网卡，USB，CF 卡，sd 卡，硬盘等等，主要用查看设备的名字，以供其他操作的使用。

g

命令： **g**

格式： **g [-s][-b bpaddr][-e addr][--args]**

解释： g 命令是 PMON 中一个很重要的命令，直接从指定内存地址处开始执行程序，如：

g –e addr

如上命令，直接从内存地址 addr 处开始执行程序，正常操作前提是在 addr 开始处的内存中已经存放了一个可执行的程序，addr 是这个程序的入口地址，如我们知道内存地址 0x80010000 是 pmon 代码段存放开始地址，执行 ***g –e 0x80010000*** 将会从新执行 pmon 代码，相当于了重启 PMON（这时不会从 flash 中拷贝执行 pmon）。

load tftp://server-ip/vmlinux

g console=ttyS0,115200 rdinit=/sbin/init

如上命令，和 load 命令配合使用，在 load 完成后，使用 g 命令自动从加载后的内核入口地址开始执行内核，这是 g 后面在参数“console=ttyS0,115200 rdinit=/sbin/init”是内核启动是的参数。

h

命令： **h**

格式： **h**

解释： 显示 pmon 中所有命令的帮助信息，单独查看一个命令的帮助信息，可以使用 ***cmd –h*** 来查看。如： ***load –h*** 查看 load 命令的帮助信息。

ifaddr

命令： **ifaddr**

格式： **ifaddr <interface> <ipaddr>**

解释： 设置网卡的 IP 地址，如：***ifaddr syn0 192.168.0.1*** （syn0 是网卡的名字，可以使用 devls 命令查看获得，这里不用设置子网掩码，pmon 会自动设置子网掩码为 255.255.255.0）

load

命令： **load**

格式： **load <elf-file>**

解释： load 命令是 PMON 中一个很重要命令。作用是加载一个 elf 文件到内存中（这里只是存放到内存中，而没有烧写到 flash 中），加载过程是一个自动根据 elf 文件的信息处理 elf 文件重定向的等等操作的总体过程，所以这里不需要指定加载的内存地址，load 命令会自动完成。如：

load tftp://server-ip/vmlinux

如上命令，是从网络 tftp 服务器中加载内核到内存中。

load /dev/mtd0b

如上命令，是从 nandflash 的第一个分区开始处中加载一个 elf 文件到内存中，这里

一般存放着内核，这也是加载内核的另一种方法。

注：这里的 vmlinux 是一个 elf 文件，如果指定加载的文件不是 elf 文件，将提示错误。

ifconfig

命令： **ifconfig**
格式： **ifconfig <interface>**
解释： 查看网卡的配置信息，如：***ifconfig syn0*** （syn0 是网卡的名字，可以使用 devls 命令查看获得）

m1

命令： **m1**
格式： **m1 <addr> <value>**
解释： modify 简写，按无符号字符型（unsigned char）访问修改地址 addr 的内容值。如：
m1 0x85000000 0x10
修改内存地址 0x85000000 地址处的一个 unsigned char 型值
相当于：*（unsigned char*）addr = value；

m2

命令： **m2**
格式： **m2 <addr> <value>**
解释： modify 简写，按无符号字符型（unsigned short）访问修改地址 addr 的内容值。如：
m2 0x85000000 0x10
修改内存地址 0x85000000 地址处的一个 unsigned short 型值
相当于：*（unsigned short*）addr = value；

m4

命令： **m4**
格式： **m4 <addr> <value>**
解释： modify 简写，按无符号字符型（unsigned long）访问修改地址 addr 的内容值。如：
m4 0x85000000 0x10
修改内存地址 0x85000000 地址处的一个 unsigned long 型值
相当于：*（unsigned long*）addr = value；

m8

命令： **m8**
格式： **m8 <addr> <value>**
解释： modify 简写，按无符号字符型（unsigned long long）访问修改地址 addr 的内容值。
如：
m8 0x85000000 0x10
修改内存地址 0x85000000 地址处的一个 unsigned long 型值
相当于：*（unsigned long long*）addr = value；

mtdparts

命令： **mtdparts**
格式： **mtdparts**
解释： 显示 nandflash 的现在分区信息。

mtd_erase

命令： **mtd_erase**
格式： **mtd_erase <mtd-device>**
解释： 擦除 nandflash 的一个分区，根据 mtd-device 的不同相应的功能不同。如：
mtd_erase /dev/mtd0b
或者 ***mtd_erase /dev/mtd0c***
或者 ***mtd_erase /dev/mtd0y***
或者 ***mtd_erase /dev/mtd0***
(擦除 nandflash 的第一个分区，跳过已经是坏块的地方，不会尝试擦除已经是坏块的地方)；
mtd_erase /dev/mtd0r
(擦除 nandflash 的第一个分区，这时会尝试擦除所有的块，即使是坏块也会尝试执行擦除操作)。

reboot

命令： **reboot**
格式： **reboot**
解释： 重启 PMON

ping

命令：ping
格式：ping <ipaddr>
解释： 简单检测，如：***ifconfig syn0*** （syn0 是网卡的名字，可以使用 devls 命令查看获得）

set

命令：set
格式：set [name] [value]
解释： 查看 pmon 中的环境变量,设置环境变量 name 为 value 如：***set*** （查看显示 pmon 中所有的环境变量）； ***set wk kkk*** （添加或者修改环境 wk 的值为 kkk）

unset

命令：unset
格式：unset <name>
解释： 去掉名字为 name 的环境变量。关于环境变量的讲解参看附录 PMON 环境变量

2 Pmon 环境变量说明

2.1 环境变量操作说明

PMON 中有很灵活的环境变量机制，操作简便，可以通过命令 set 和 unset 这两个命令设置和取消一项环境变量（这两个命令的使用，参看常用命令解释）。

用户可自定义环境变量方法：

1，在 pmon 启动后使用 set 命令定义环境变量，这种方法定义的环境变量，存储在 spiflash 中，只要不擦除 spiflash，会一直存在。使用 unset 命令即可去除。

2，在 pmon 源代码中添加的环境变量（对于 1B 的 pmon 源代码在 ./Targets/LS1F/include/pmon_target.h 中的 TGT_DEFENV 宏，根据例子添加即可），这种方法添加是编译在 pmon 可执行文件中的，如果不修改源代码，命令无法去除，只可以修改其值，使用 unset 命令将会恢复成默认值（mtdparts 和 bootdelay 就是这样的环境变量）。

2.2 常用环境变量解释

autocmd

名　字： **autocmd**
常用值： 无
依　赖： **al**
解　释： 在 pmon 启动完成后会自动检查 autocmd 是否设置，如果设置了 autocmd，pmon 将会首先执行 autocmd。
注： 在设置了 al 环境变量后，autocmd 才能生效。

al

名　字： **al**
常用值： **/dev/mtd0b**
依　赖： 无
解　释： 在 pmon 启动完成后会自动检查 al 是否设置，如果设置了 al，pmon 将会首先根据 al 的值来用 load 命令加载内核。功能作用和 u-boot 中的 bootfile 相似。

append

名　字： **append**

常用值： **console=ttyS0,115200 rdinit=/sbin/init root=/dev/mtdblock1 rw rootfs=yaffs2**
依　赖： **al**
解　释： pmon 启动后根据 al 加载内核后，会根据 append 的值作为内核参数执行 g 命令启动内核。功能作用和 u-boot 中的 bootargs 相似。

bootdelay

名　字： **bootdelay**
常用值： **3**
依　赖： **al**
解　释： 设置在 pmon 启动后等待多少时间之后根据 al 加载 elf 文件或者执行 autocmd，依赖 al，如果没有设置 al，bootdelay 将无效。**单位为秒**。

ethaddr

名　字： **ethaddr**
常用值： **无**
依　赖： **无**
解　释： 设置网卡的物理地址

ifconfig

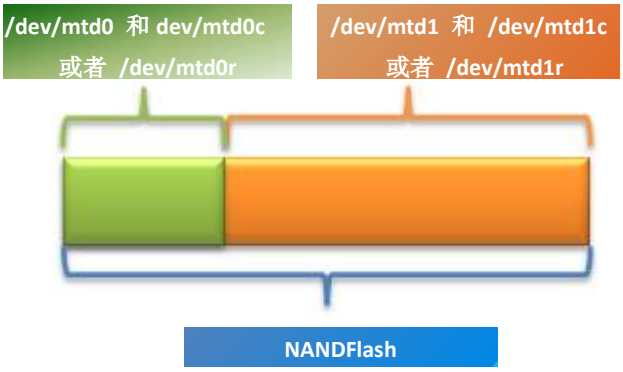
名　字： **ifconfig**
常用值： **无**
依　赖： **无**
解　释： 设置网卡的 ip 地址，命令格式：set ifconfig　syn0:xxx.xxx.xxx.xxx　syn0 是网卡名字。当设置了 ifconfig 后，启动 pmon 中会自动根据 ifconfig 的值来设置 ip。

mtdparts

名　字： **mtdparts**
常用值： **nand-flash:20m(kernel)ro,-(rootfs)**
依　赖： **无**
解　释： 设置 nandflash 分区，值的格式和 linux 内核参数中 mtdparts 参数一致，mtdparts 的值也会被内核处理，来设置 nandflash 的分区。这样保证了 pmon 中 nandflash 分区方式和内核分区的一致性。此项环境变量更新值后即时生效。
注：由于内核也会使用 mtdparts 的值设置 nandflash 分区，所以建议在设置了 mtdparts 的值后，不要随意更改 mtdparts 的值，否则会有可能损坏 flash 中存储的数据。
使用 ***unset mtdparts*** 命令即可恢复 mtdparts 的默认值。

3 NandFlash 分区说明

一般我们只默认添加了两个 nand 分区。在 PMON 中分别对应设备名字是/dev/mtd0 和 /dev/mtd1 来表示。



如图所示 /dev/mtd0 指的是第一个分区/dev/mtd1 指的是第二个分区；如果添加新的分区序号会以此增加（/dev/mtd0 /dev/mtd1 /dev/mtd2 不包括 spare（oob）区域）。

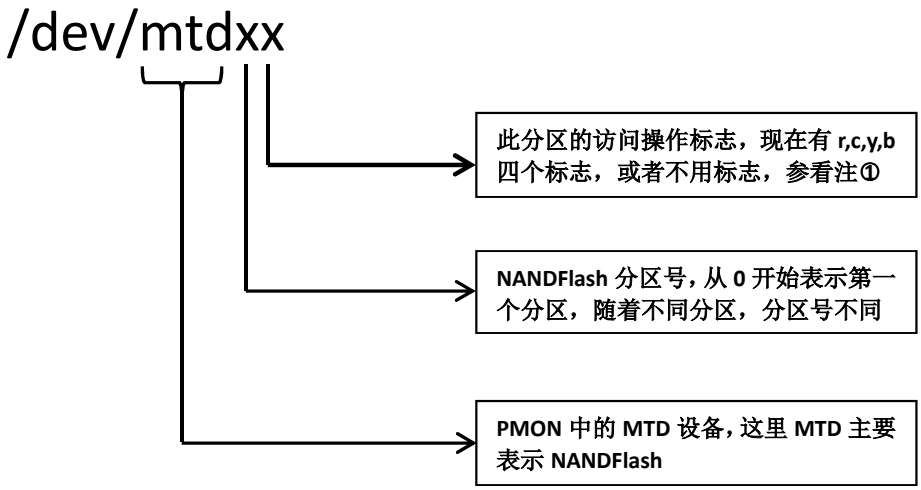
而设备文件 /dev/mtd0c /dev/mtd1c /dev/mtd2c （名字的后面多一个字符“c”），指的是包括 spare（oob）区域对应分区。

在烧写 **yaffs2** 镜像文件的时候需要读写 nandflash 的 spare（oob）区域，所以用的设备文件是/dev/mtd0y /dev/mtd1y 或者 /dev/mtd2y。而在烧写 jffs2 或者 cramfs 镜像文件的时候，不需要操作 spare（oob）区域。所以使用的是/dev/mtd0 /dev/mtd1 或者 /dev/mtd2。

设备文件名	说明
/dev/mtd0	同/dev/mtd0b 默认是带“b”标志
/dev/mtd1	同/dev/mtd1b 默认是带“b”标志
/dev/mtd0b	第一个 nandflash 分区 ，不包括 spare（oob）区域
/dev/mtd1b	第二个 nandflash 分区 ，不包括 spare（oob）区域
/dev/mtd0c	第一个 nandflash 分区 ，包括 spare（oob）区域，oob 全部是用户数据
/dev/mtd1c	第二个 nandflash 分区 ，包括 spare（oob）区域，oob 全部是用户数据
/dev/mtd0y	第一个 nandflash 分区，包括 spare（oob）区域，部分是用户数据
/dev/mtd1y	第一个 nandflash 分区，包括 spare（oob）区域，部分是用户数据
/dev/mtd0r	第一个 nandflash 分区，包括 spare（oob）区域，不处理坏块
/dev/mtd1r	第二个 nandflash 分区，包括 spare（oob）区域，不处理坏块

对于 NANDFlash 分区的操作，我们在 PMON 中提供了和 linux 内核一致命令行分区命令（mtdparts），对于 mtdparts 命令的使用可以参考 PMON 常用命令相关章节。我们在这里简单说明一下 PMON 中 NANDFlash 的其他操作。

PMON 中 NANDFlash 设备的分区，我们使用 /dev/mtdxx 的形式来表示。



例如：

/dev/mtd0b

正常访问第一个 NANDFlash 分区（**不包括**每一块的 spare 区域），坏块由 PMON 中坏块策略保证，用户不用关心坏块的问题。

/dev/mtd0

同 ***/dev/mtd0b*** 含义相同

/dev/mtd0c

正常访问第一个 NANDFlash 分区（**包括**每一块的 spare 区域），坏块由 PMON 中坏块策略保证，用户不用关心坏块的问题。Spare 所有数据全部由用户传入。

/dev/mtd0y

正常访问第一个 NANDFlash 分区（**包括**每一块的 spare 区域），坏块由 PMON 中坏块策略保证，用户不用关心坏块的问题。Spare 会根据 MTD 层的 obb 布局自动控制，主要用于 yaffs 镜像文件的烧写操作。

/dev/mtd0r

正常访问第一个 NANDFlash 分区（**包括**每一块的 spare 区域），此时会绕过 PMON 中坏块策略，用户需要考虑坏块的问题。这在擦除一个分区的时候很有用，如果错将一个好的块标记成了坏块，使用 ***mtd_erase /dev/mtd0*** 或者 ***mtd_erase /dev/mtd0c*** 都将无法擦除恢复此块（因为此时坏块策略中可能已经记录了此块，所有通过坏块策略访问的操作，都将无法访问到此块），只有使用“r”标志访问，绕过坏块策略才能成功擦除恢复，如命令： ***mtd_erase /dev/mtd0r***

注①：

所有的标志同时只有一个有效，多个标志同时使用时第一个传入的有效。

c ：char 简写，使用 PMON 中的坏块策略，能访问到 NANDFlash 每一块的 mian 和 spare 区域。Spare 区域数据全部从用户数据获取（与“**y**”标志的不同点在于对 spare 区域数据的处理）。

y ：同“**c**”标志相识，但是在处理 spare 区域的时候，坏块标志不会受用户控制，整个 spare 布局由 MTD 层控制，**此标志一般用于 yaffs 镜像文件的烧写或者读取操作中**。

b ：block 简写，使用 PMON 中的坏块策略，能访问到 NANDFlash 每一块的 mian 区域。

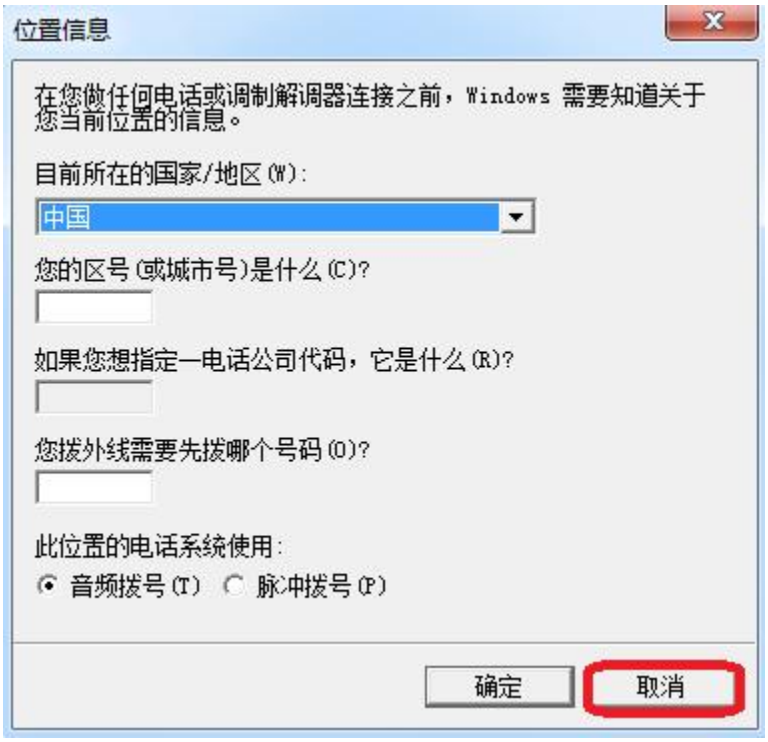
r ：raw 简写，不使用 PMON 中坏块策略，能访问到 NANDFlash 每一块的 main 和 spare 区域。

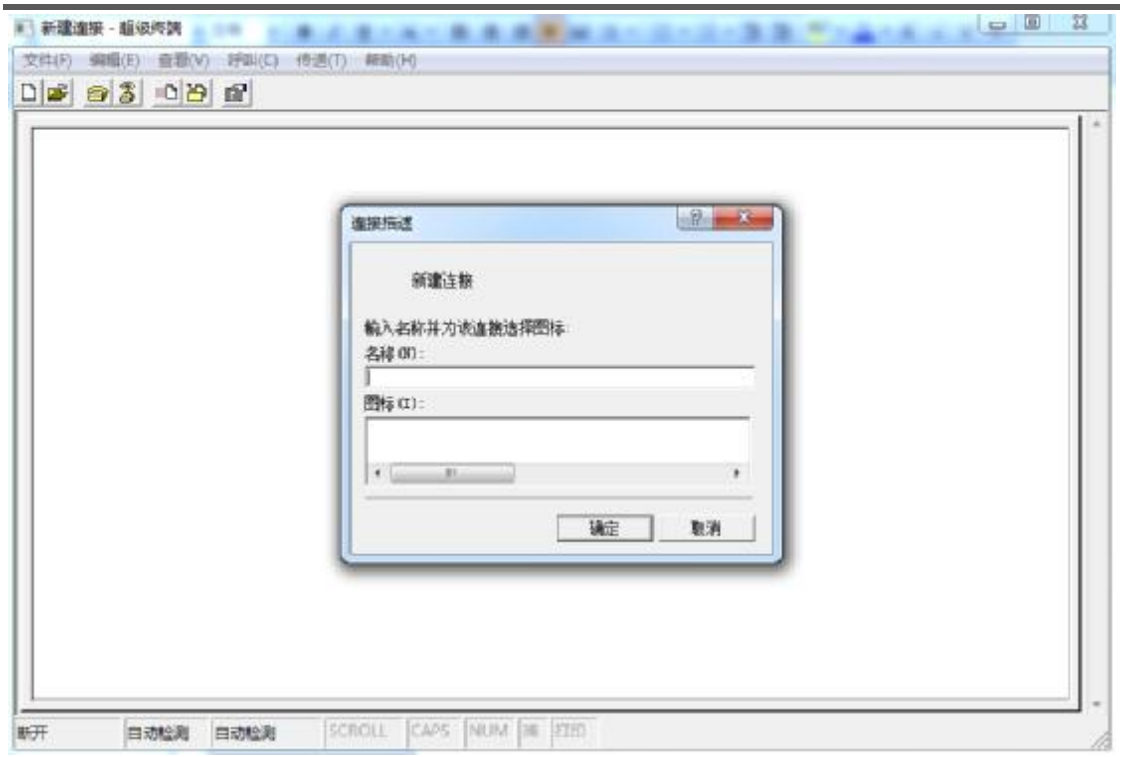
当没有标志的时候，使用 PMON 的默认标志“**b**”，访问会默认使用 PMON 中的坏块策略，但是只能访问到 NANDFlash 每一块的 main 区域。

4 内核编译常用配置

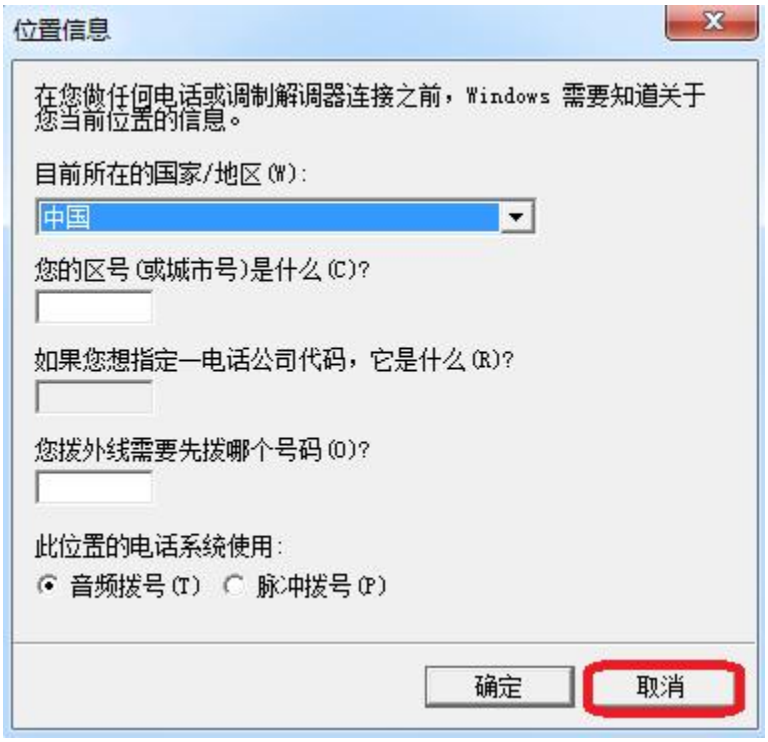
5 Windows 超级终端使用说明

（1）打开超级终端：

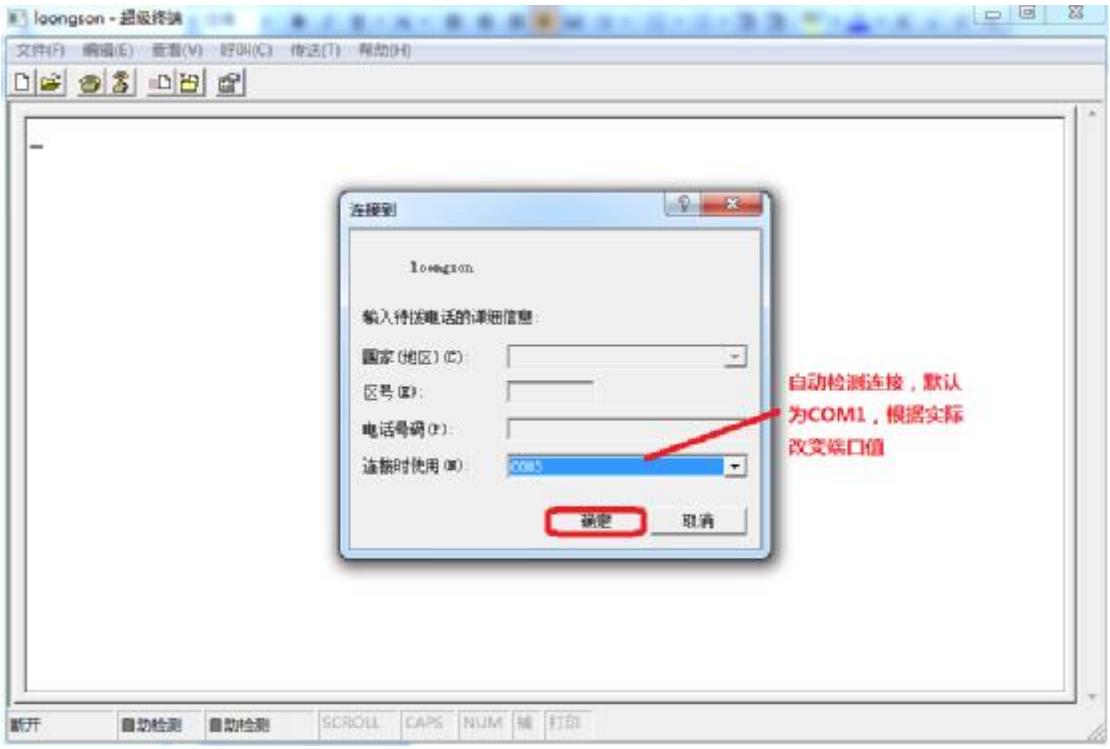




（2）新建连接，修改名称：


位置信息
在您做任何电话或调制解调器连接之前，Windows 需要知道关于您当前位置的信息。
目前所在的国家/地区(W):
中国
您的区号(或城市号)是什么(C)?
如果您想指定一电话公司代码，它是什么(R)?
您拨外线需要先拨哪个号码(O)?
此位置的电话系统使用:
音频拨号(T)
脉冲拨号(P)
确定
取消

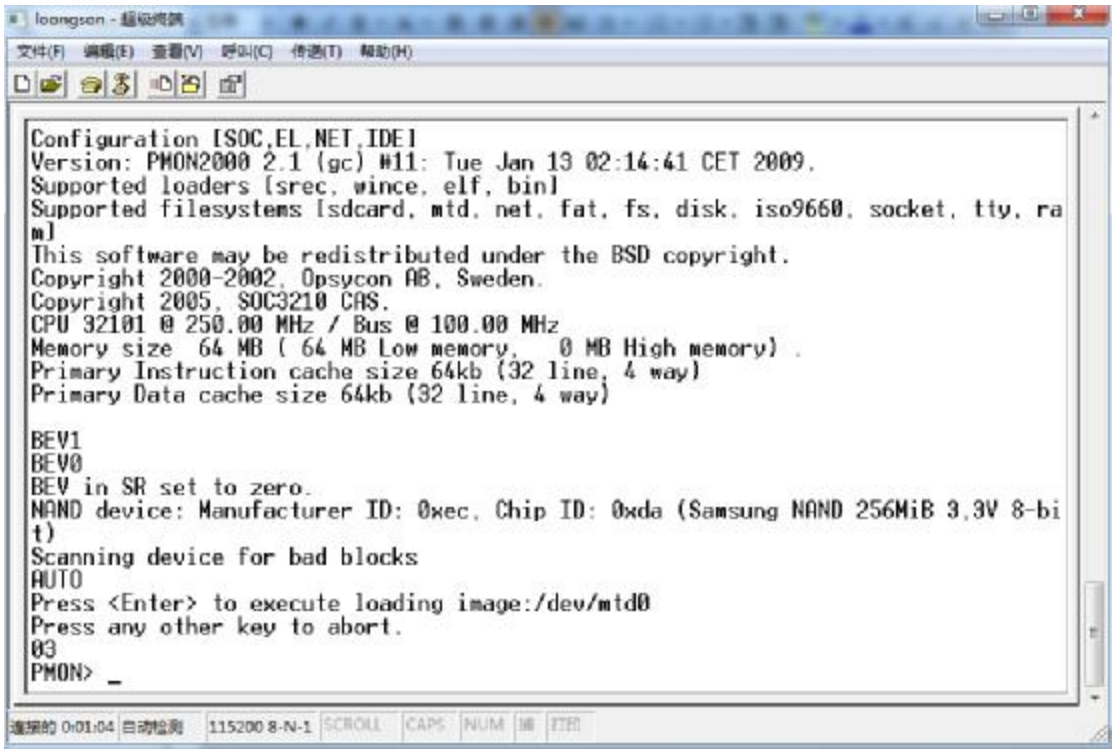
（3）设置：


loongson - 超级终端
连接到
loongson
输入待拨电话的详细信息:
国家(地区)(C):
区号(E):
电话号码(P):
连接时使用(N):
COM1
确定
取消
自动检测连接，默认为COM1，根据实际改变端口值


COM3 属性
端口设置
位/秒(B):
2400
数据位(D):
8
奇偶校验(P):
无
停止位(S):
1
数据流控制(F):
硬件
还原为默认值(R)
确定
取消
应用(A)


COM3 属性
端口设置
位/秒(B): 115200
数据位(D): 8
奇偶校验(P): 无
停止位(S): 1
数据流控制(F): 无
还原为默认值(R)
确定
取消
应用(A)

（4）开发板上电：


Configuration [SOC,EL,NET,IDE]
Version: PMON2000 2.1 (gc) #11: Tue Jan 13 02:14:41 CET 2009.
Supported loaders [srec, wince, elf, bin]
Supported filesystems [sdcard, mtd, net, fat, fs, disk, iso9660, socket, tty, ra
m]
This software may be redistributed under the BSD copyright.
Copyright 2000-2002, Opsycon AB, Sweden.
Copyright 2005, SOC3210 CAS.
CPU 32101 @ 250.00 MHz / Bus @ 100.00 MHz
Memory size 64 MB (64 MB Low memory, 0 MB High memory) .
Primary Instruction cache size 64kb (32 line, 4 way)
Primary Data cache size 64kb (32 line, 4 way)
BEV1
BEV0
BEV in SR set to zero.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB 3.3V 8-bi
t)
Scanning device for bad blocks
AUTO
Press <Enter> to execute loading image:/dev/mtd0
Press any other key to abort.
03
PMON> _

6 Minicom 使用指南

minicom 是 Linux 上最常用的终端仿真程序，它类似于 Windows 下的“超级终端”的程序，一般完全安装大部分发行版的 Linux 时都会包含它，下面介绍它的使用方法。

6.1 安装 minicom

如果没有安装 minicom，先安装：

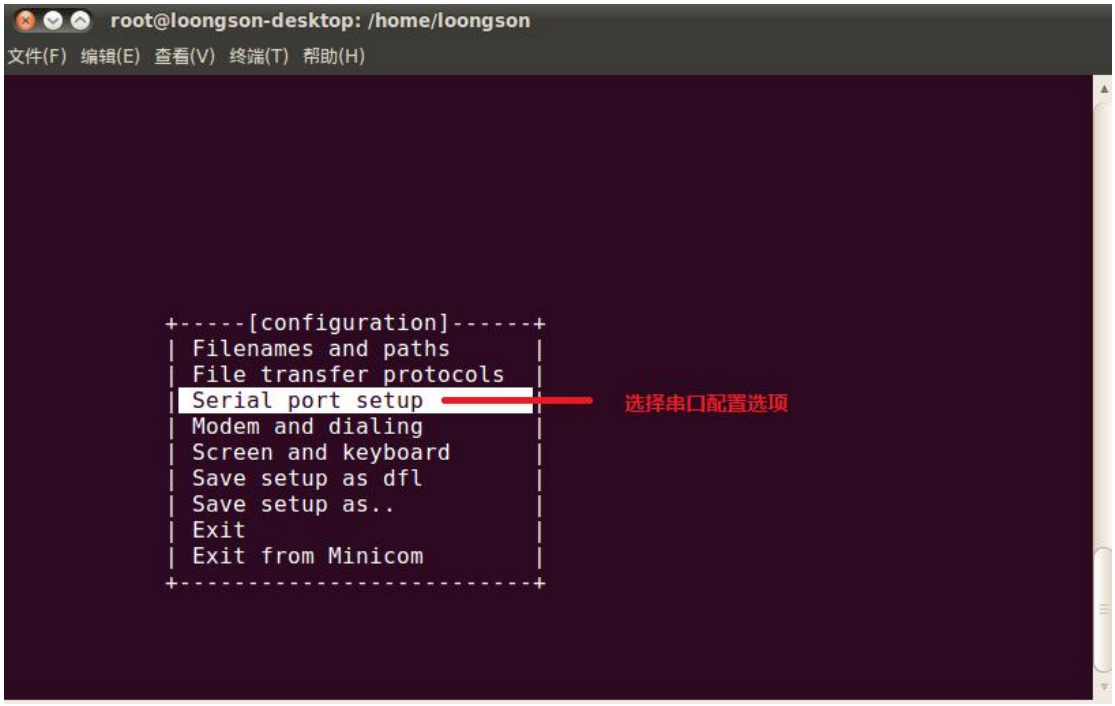
```
apt-get install minicom
```

6.2 配置 minicom

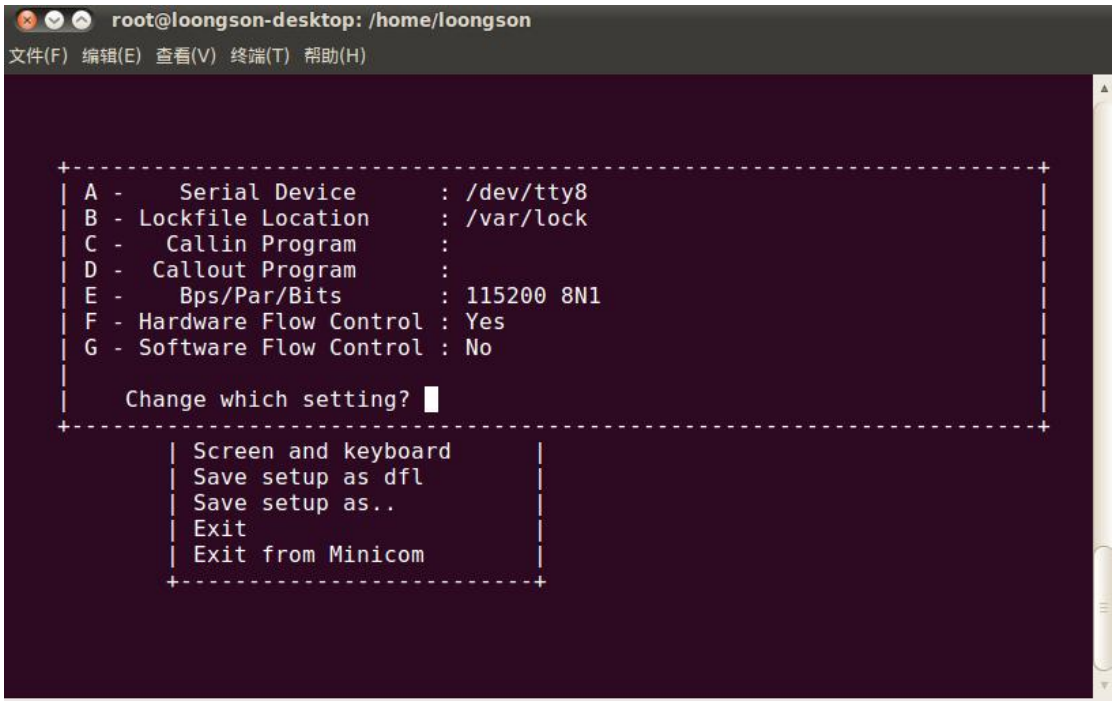
使用 minicom 之前先设置一下：

（1）Linux 终端输入

```
minicom -s
```



（2）进入“Serial port setup”：



提示：

A 对应的是键盘‘A’键，按下‘A’键配置串口设备。通常的配置是/dev/ttyS0 或/dev/tty0 或者/dev/ttyUSB0, 对应于 windows 操作系统的 COM1。

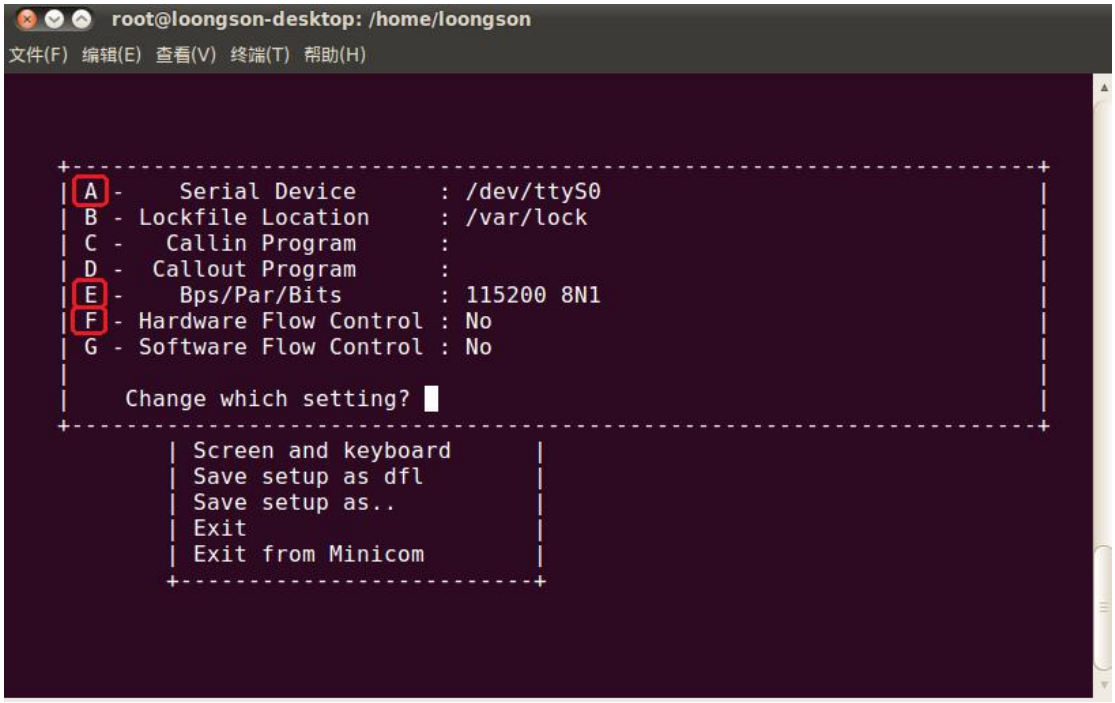
F 对应的是硬件流设置，按‘F’进行配置，通常选择 NO 。

E 对应的是键盘‘E’键，是用来配置串口波特率的，按键‘E’选择波特率 。

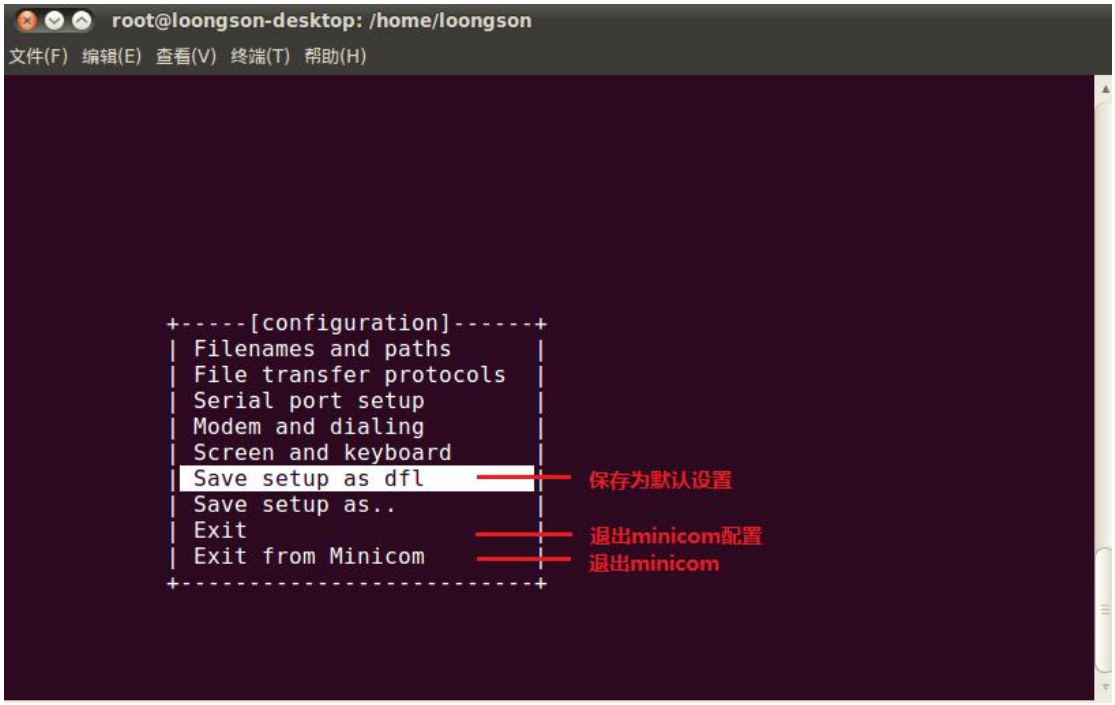
键盘‘ESC’键，退出，返回上一层。

键盘‘Enter’键，确定，保存。

（3）配置串口设备、波特率和硬件流：



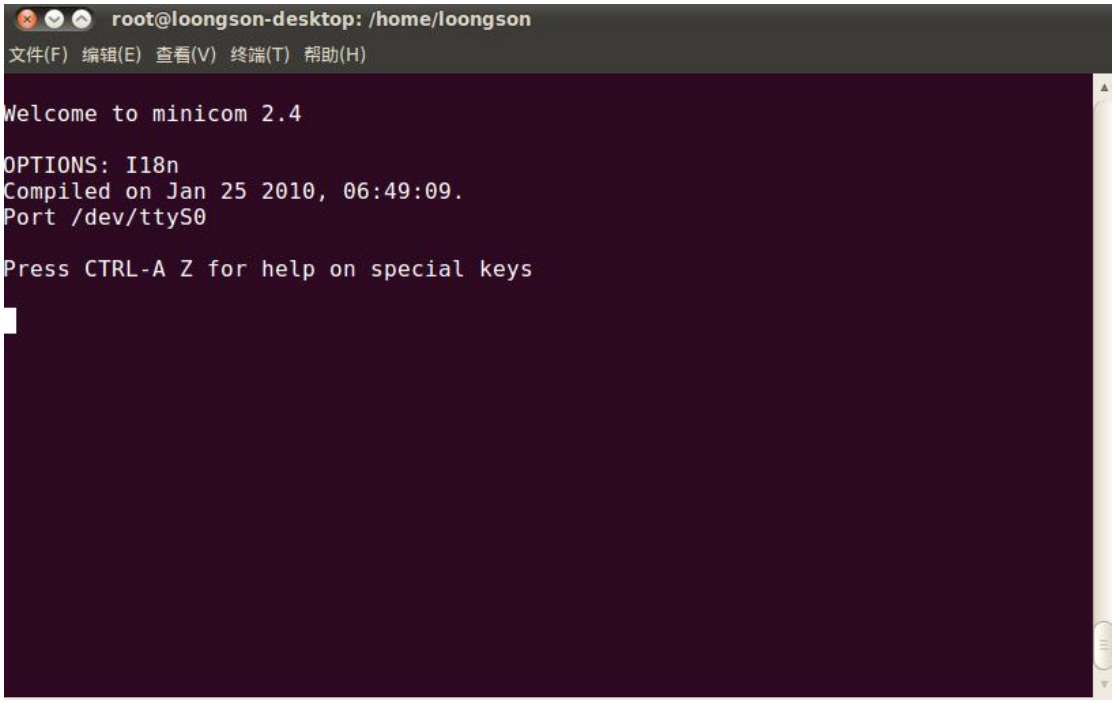
（4）保存退出到主配置界面：

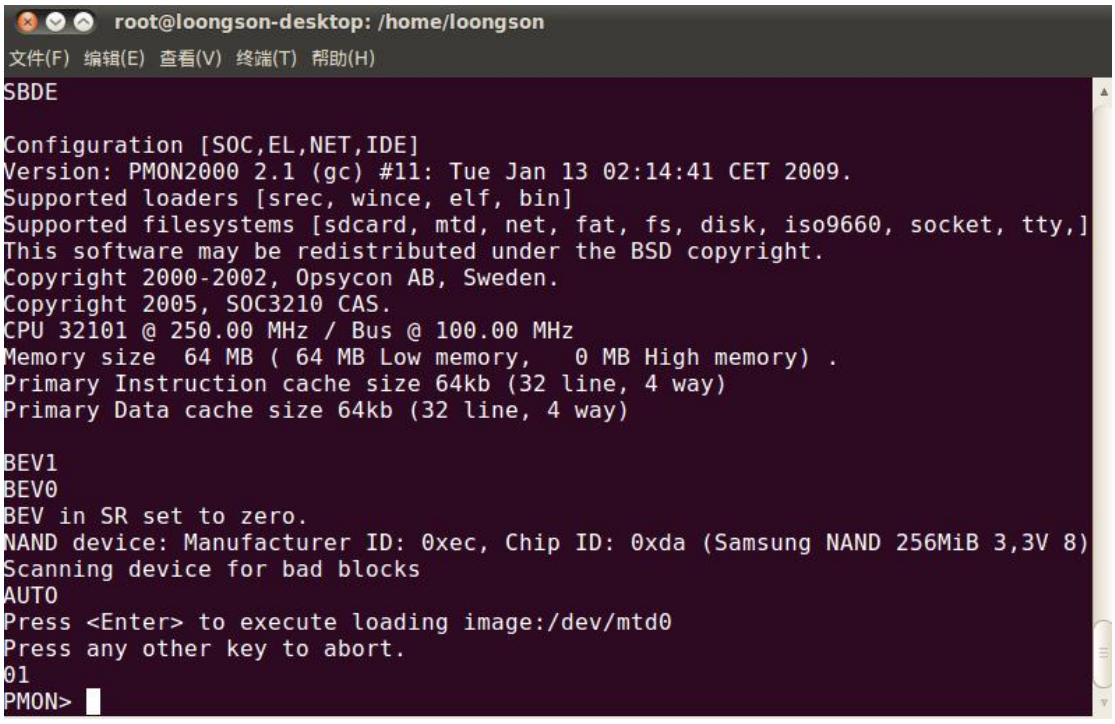


6.3 使用 minicom

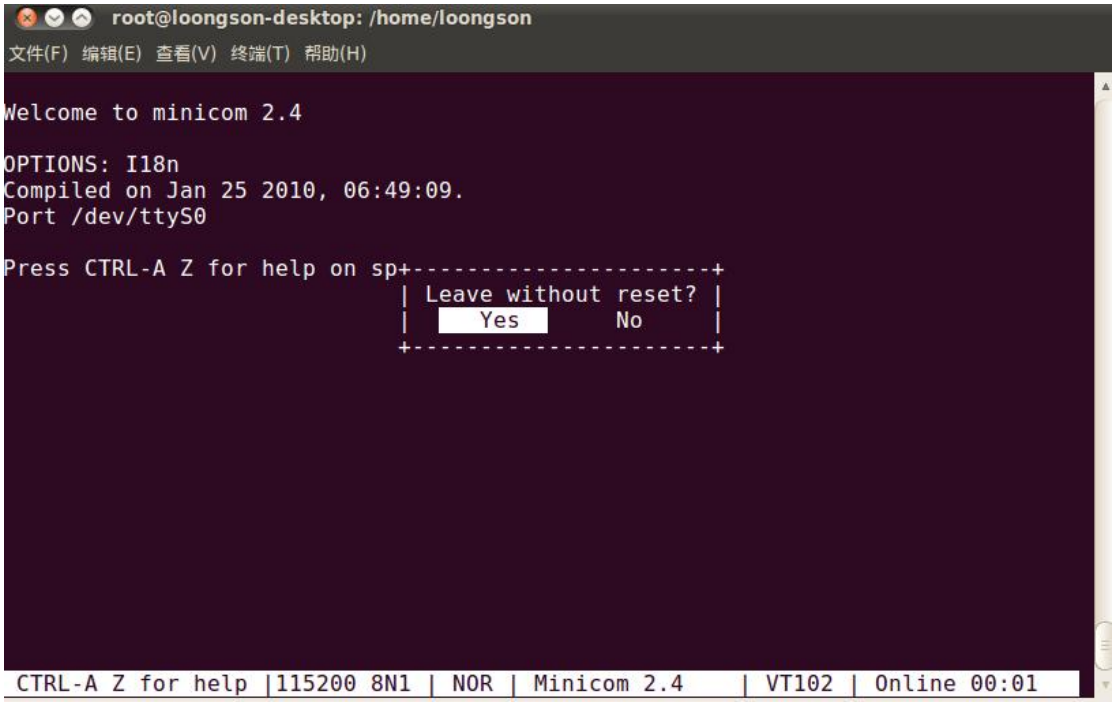
先连接好串口线或 USB 转串口。

退出 minicom 配置，进入 minicom：





在进入 minicom 后，如果想退出，则可以按下 ctrl+A，松开 ctrl+A 后再紧接着按下 Q 就可以看到退出提示了，选择 Yes 退出。



7 Pmon 下常用操作

1）查看设备

命令： ***devls***

2）配置 ip 地址

命令：*ifaddr syn0 xxx.xxx.xxx.xxx* **（syn0 为用 devls 查看得到的网卡名字）**

3）简单测试网络

命令：*ping other-ip* **（other-ip 为在同一网段其他 PC 机器 IP）**

4）pmon 下通过网络加载内核到内存

命令：*load tftp://tftp-server-ip/vmlinux*

注：此命令正常执行会打印加载信息

5）PMON 下通过网络烧写更新替换新 pmon

命令：*load –r –f 0xbfc00000 tftp://tftp-server-ip/gzrom.bin*

注：0xbfc00000 为龙芯处理器启动的第一条指令地址（映射到 Spiflash 中，即从 Spiflash 启动。

gzrom.bin 为 pmon 的二进制名字。

6）烧写内核到 nandflash（通过网络）

命令：*devcp tftp://tftp-server-ip/vmlinux /dev/mtd0*

注：这里也可以是 /dev/mtd1 或者 /dev/mtd2 ，分别烧写到不同分区，需要在内核加载参数中设置不同的内核启动位置。

7）从 nandflash 中加载内核

命令： ***load /dev/mtd0***

注：/dev/mtd0 需要根据内核的实际位置来修改（可能是/dev/mtd1 /dev/mtd2 等）。

8）内核添加参数启动

命令：*g console=ttyS0,115200 rdinit=/sbin/init*

注：使用方式是 g 后面跟内核参数，例子中的参数根据实际的使用来修改。

9）设置内核启动加载位置环境变量

命令：*set al /dev/mtd0*

注：/dev/mtd0 需要根据内核的实际位置来修改（可能是/dev/mtd1 /dev/mtd2 等）。

PMON 启动后会自动根据 al（autoload）的值来自动加载内核。

10）设置内核启动参数的环境变量

命令：*set append "console=ttyS0,115200 rdinit=/sbin/init"*

注：内核启动参数根据实际情况修改。

PMON 在启动后根据 al 的值自动加载内核，而后自动用 append 的值做为内核启动参数来启动内核系统。

11）设置延时启动时间环境变量

命令：*set bootdelay 3*

注：单位为**秒**（3：为延时 3 秒）

PMON 启动后根据 bootdelay 的值来延时加载启动系统内核。

12）重启 PMON

命令：*reboot*

13）擦除 nandflash 第一个分区（跳过坏块）

命令： ***mtd_erase /dev/mtd0***
mtd_erase /dev/mtd0b
mtd_erase /dev/mtd0c
mtd_erase /dev/mtd0y

14）擦除 nandflash 第一个分区（擦除所有块包括坏块）

命令： ***mtd_erase /dev/mtd0r***

15）设置 NANDFlash 分区（分三个分区第一个大小 10m，属性只读，名字是 kernel，第二个分区大小 40m，属性可读可写，名字是 rootfs，第三分区大小剩下的所有空间，属性可读可写，名字是 other）

命令： ***set mtdparts nand-flash:10m(kernel)ro,40m(rootfs),-(other)***
设置后需要重启生效命令：*reboot*

16）查看当前 NANDFlash 分区信息

命令： ***mtdparts***

17）恢复 NANDFlash 默认值分区

命令： ***unset mtdparts***

Change log

V1.2(2012-06-06)

1,改变了用户环境变量的源代码修改位置，./Targets/LS1F/include/pmon_target.h 中的 TGT_DEFENV 宏。P20

2，mtdparts 环境变量修改值后，现在是即时生效，不需要重启。P21