**東軟機密**

环境构筑手册

**Ver 0.5**

**2011年2月12日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ページ数 | 14 | 本文 | 14 | 付録 |  | 発行日付 | 2011/2/12 |
| 作成者 | 吕凯 | | | 承認者 |  | | |

**変更履歴**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版数** | **年/月/日** | **改版内容** | **担当者** | **承認者** |
| 0.1 | 2010/11/16 | 新規作成 | 吕凯 |  |
| 0.2 | 2010/11/30 | 内容更新与精简 | 吕凯 |  |
| 0.3 | 2010/12/15 | 内容更新与精简 | 吕凯 |  |
| 0.4 | 2011/1/12 | 追加 | 吕凯 |  |
| 0.5 | 2011/2/12 | 追加交叉编译工具链获取 | 吕凯 |  |
|  |  |  |  |  |

※変更種別

**A:追加**

B:削除

**C:変更**

**目 次**

[1 编译环境和编译 4](#_Toc285265822)

[1.1 环境搭建 4](#_Toc285265823)

[1.1.1 centeros 安装.（省略） 4](#_Toc285265824)

[1.1.2 安装交叉编译工具链(SONY-CE-Linux-5.0.4.1-ARM.iso) 4](#_Toc285265825)

[1.1.3 安装内核头文件 4](#_Toc285265826)

[1.2 代码编译 4](#_Toc285265827)

[1.2.1 release时 4](#_Toc285265828)

[1.2.2 开发时 5](#_Toc285265829)

[2 运行环境和运行 6](#_Toc285265830)

[2.1 pc机端配置 6](#_Toc285265831)

[2.1.1 配置nfs服务 6](#_Toc285265832)

[2.1.2 配置tftp服务 6](#_Toc285265833)

[2.1.3 配置minicom 7](#_Toc285265834)

[2.1.4 准备内核 7](#_Toc285265835)

[2.1.5 准备文件系统 7](#_Toc285265836)

[2.2 开发板设置 7](#_Toc285265837)

[2.2.1 release时 7](#_Toc285265838)

[2.2.2 开发时 8](#_Toc285265839)

[2.3 程序运行 9](#_Toc285265840)

[3 其他补充 10](#_Toc285265841)

[3.1 三种格式的内核 10](#_Toc285265842)

[3.2 NBL编译过程 10](#_Toc285265843)

[3.3 NBL烧写 10](#_Toc285265844)

[3.4 烧写内核时常见的问题 11](#_Toc285265845)

[3.5 交叉编译工具链的获取 12](#_Toc285265846)

# 编译环境和编译

## 环境搭建

### centeros 安装.（省略）

### 安装交叉编译工具链(SONY-CE-Linux-5.0.4.1-ARM.iso)

#cd SONY-CE-Linux-5.0.4.1-ARM/Tools\_Userland/arm

#rpm -e $(rpm -qa "arm-sony-linux-gnueabi-\*")

#rm -rf /usr/local/arm-sony-linux-gnueabi/

#rpm -Uvh --ignorearch \*.rpm

其中SONY-CE-Linux-5.0.4.1-ARM/doc/install\_en.txt里面有相关的文档。

### 安装内核头文件

#cd <latest kernel path>

#export PATH=/usr/local/arm-sony-linux-gnueabi/devel/bin:${PATH//\/usr\/local\/arm-sony-linux-gnueabi\/devel\/bin:/}

#make avp13\_defconfig

#make

#make kernel\_headers\_install

## 代码编译

### release时

#### 编译内核

$export PATH=/usr/local/arm-sony-linux-gnueabi/devel/bin:${PATH//\/usr\/local\/arm-sony-linux-gnueabi\/devel\/bin:/}

$cd <latest kernel path>

$ make avp13\_rel\_defconfig

$ make

生成的内核avp13Build/vmlinux

#### 编译app以及生成文件系统镜像

#cd <latest rootfs path>

#cd makerootfs

#./makerootfs.sh clean

#./makerootfs.sh all

注意，

生成的镜像名称为target/\*.img

对于该脚本具体信息，最好参考下makerootfs/readme

### 开发时

#### 编译内核

$export PATH=/usr/local/arm-sony-linux-gnueabi/devel/bin:${PATH//\/usr\/local\/arm-sony-linux-gnueabi\/devel\/bin:/}

$cd <latest kernel path>

$ make avp13\_defconfig

$ make

生成的内核avp13Build/vmlinux

#### 编译app以及生成文件系统镜像

#cd <latest rootfs path>

#cd makerootfs

#./makerootfs.sh all

这样会根据时间戳编译最新的内容。

# 运行环境和运行

## pc机端配置

### 配置nfs服务

配置nfs服务的目的是让板通过网络把nfs的目录当做板子环境中的根目录。

(1)编辑/etc/exports

添加类似如下一行：

<your nfs directory> \*(rw,sync,no\_root\_squash)

在配置NFS之前先查看“rpm -q nfs-utils portmap“两个软件包是否安装，默认都是安装的。

这里，'<your nfs directory>'是你本地的nfs目录，可以随意设置。具体含义参见"info exports".

(2)重启服务：

#/etc/init.d/nfs restart

#/etc/init.d/portmap restart

(3)关闭防火墙：

这一步有的人可以不用做。

具体在："系统"->"管理"->"安全级别和防火墙"

至此nfs服务搭建完毕，测试方法如下：

在其它机器上运行：

#mkdir nfstest

#mount -t nfs <你的ip>:<你的nfs目录> nfstest

如果成功则完毕。

### 配置tftp服务

配置tftp服务的目地是让板子通过tftp下载内核并启动。

(1)安装tftp客户/服务端:

#yum install tftp.i386

#yum install tftp-server.i386

这里，如果不行就先运行"#yum makecache"试试。

(2)编辑/etc/xinetd.d/tftp

service tftp

{

socket\_type = dgram

protocol = udp

wait = yes

user = root

server = /usr/sbin/in.tftpd

server\_args = -s /tftpboot

disable = no

per\_source = 11

cps = 100 2

flags = IPv4

}

这里，disable是指关闭还是打开tftp服务，取值yes/no.

server\_args指定tftp服务在本地机器上的导出目录。其它参数具体含义参见“info xinetd.conf”.

(3)重启服务：

#/etc/init.d/xinetd restart

至此tftp服务搭建完毕，测试方法如下：

在本地机器上运行：

#>>/tftpboot/mytest

在其它机器上运行：

#tftp <你的ip地址>

#get mytest

然后"[Ctrl]D"退出，如果在那个其他机器的当前目录看到mytest说明成功。

### 配置minicom

配置minicom的目的是让pc机可以通过minicom连接开发板。

(1)#minicom -s

(2)选择"Serial port setup"

配置其中的如下配置选项：

A - Serial Device : /dev/ttyS0

E - Bps/Par/Bits : 115200 8N1

F - Hardware Flow Control : No

G - Software Flow Control : No

### 准备内核

将编译好的内核放到tftp的导出目录。

导出目录按前面配置应该是:/tftpboot

#cp vmlinux /tftpboot/vmlinux

### 准备文件系统

为release使用，将文件系统镜像(ramdisk.img)拷贝到tftp导出目录(/tftpboot)。

为开发使用，将文件系统中所有内容拷贝到nfs导出目录(<your nfs directory>)。

编译时候生成的文件系统以及镜像在目录"Rootfs/makerootfs/target/"中.

## 开发板设置

pc端配置好之后，启动minicom，再启动开发板，会自动进入bootloader。

### release时

需先将文件系统和内核写到flash rom中,配置如下：

NBL> diag ethaddr 00:11:22:33:44:55

NBL> setenv myip 10.1.29.180

NBL> setenv servip 10.1.29.44

NBL> sdm init

NBL> sdm create -n -l 0x400000 kernel

NBL> sdm create -n -l 0x800000 initrd\_fat

NBL> sdm create -n -l 0x300000 nvm

NBL> cp net:tftp:vmlinux sdmd::

NBL> cp net:tftp:ramdisk.img sdme::

NBL> bootprof init

NBL> bootprof param -k sdmd:raw:vmlinux -c "mem=252MB console=ttyS0,115200n8 initrd=0x40a00000,2744425 rdinit=/sbin/init ip=10.1.29.135::10.1.29.1:255.255.255.0" -i sdme:raw:ramdisk.img -t 3 -n 3 Linux

NBL> bootprof param -k sdmd:raw:vmlinux -c "mem=252MB console=ttyS0,115200n8 initrd=0x40a00000,2744425 rdinit=/sbin/init ip=10.1.29.135::10.1.29.1:255.255.255.0" -i sdme:raw:ramdisk.img -t 3 -n 3 Linux0

这里注意：

0x40a00000是ramdisk.img的地址，在NBL中已经固定;

2744425是ramdisk.img的大小，可以用"ls -l ramdisk.img"命令来获取;

10.1.29.xx1是开发板ip.

最后，将开发板断电并设置s1902跳线组第一号开关为"on".如上设置如果正确，上电之后即可实现自动启动。

### 开发时

基于nfs文件系统，从tftp下载内核，配置如下：

NBL> setenv myip 10.1.29.180

NBL> setenv console on

NBL> setenv servip 10.1.29.44

NBL> diag ethaddr 00:11:22:33:44:55

NBL> setenv gateip 10.1.29.1

注意，这里servip是tftp服务器地址,myip是板子ip，ethaddr是板的mac,这根据自己情况设置。gateip只为保留用，可以不设置。

最后，下电，再上电并启动开发板:

NBL> boot -t elf -c "mem=252MB console=ttyS0,115200n8 root=/dev/nfs nfsroot=10.1.29.44:<your nfs directory> ip=10.1.29.180::10.1.29.1:255.255.255.0" net:tftp:vmlinux

这里，"nfsroot=10.1.29.44:<your nfs directory>"对应你机器的nfs服务目录，“ip=10.1.29.180::10.1.29.1:255.255.255.0“对应你板子的ip地址，需要修改。

这样，就可以启动开发了。如果在开发板和pc之间通过你的nfs目录相关联，可以通过这个目录在pc和你开发板间传递数据。

## 程序运行

这样，即可通过“/diablo/diabloMainCpu/diablo.bin”运行程序了。

注意：

如果上电后没有自动启动，则断电后将s1902跳线组第一号开关为"off"，然后上电运行：

NBL> aboot -n

成功启动后将开发板断电并设置s1902跳线组第一号开关为"on"，再次上电即可。

# 其他补充

## 三种格式的内核

编译好内核之后，会生成三种格式的内核,路径分别如下：

a)bin格式：avp13Build/arch/arm/boot/Image

b)压缩格式：avp13Build/arch/arm/boot/zImage

c)elf格式：avp13Build/vmlinux

这里，我们使用elf格式启动。

注意：如果使用.bin格式或者压缩格式的内核，那么烧写的时候必须以"\*.bin"或者"\*.gz"为扩展名,以便NBL能够自动识别。

## NBL编译过程

$export PATH=/usr/local/arm-sony-linux-gnueabi/devel/bin:${PATH//\/usr\/local\/arm-sony-linux-gnueabi\/devel\/bin:/}

$ cd ${RELDIR}/loader

$ ./setup diablo avp13 rom

$ make

这里，${RELDIR}是要编译的BootLoader的源代码所在路径。

生成的文件如下：

${RELDIR}/loader/arch/arm/diablo/nbl-avp13.raw

${RELDIR}/loader/arch/arm/diablo/nbl-avp13.nui

其中，向nor写nbl用raw文件，更新nbl用nui文件。

## NBL烧写

烧写NBL有可选两种途径：

3.1)使用JTAG写NBL:

当使用JTAG向Nor Flash中写入NBL的时候，NBL的raw文件会从Nor flash的初始地址下载。

具体应该参考：ICEによるROM焼き手順/AVP-13基板とadvicePro.ppt

这里未经实践，不再叙述。

3.2)在原有NBL基础上更新：

NBL> update -f net:tftp:/nbl.nui

这个过程基于tftp传输，所以需要事先把nbl.nui文件拷贝到pc机的tftp的导出目录。

## 烧写内核时常见的问题

在烧写内核的时候，经常会遇见有时候好用，有时候不好用的情况。

原因是：

我们编译出来的内核有三个，对应三种不同格式;

bootloader在aboot启动的时候，有的类型需要用根据文件的后缀来判断其类型，以便正确加载（有的类型不需要后缀）。

这样组合起来，好用和不好用的情况比较多，很容易出错。

这里将这些好用的情况和不好用的情况总结一下，希望对烧写出现问题的时候有所帮助：

3.4.1)内核编译的时候，产生的三个文件分别是：

-------

Kernel/avp13Build/vmlinux

Kernel/avp13Build/arch/arm/boot/Image

Kernel/avp13Build/arch/arm/boot/zImage

三者的区别是：

vmlinux是elf格式的内核，启动速度最快，体积最大。（目前内核烧写分区有６Ｍ即6291456字节，而vmlinux大小是6265885字节只比整个分区少了不到30K）

Image是bin格式的内核，启动速度比较快，体积大约5M.

zImage是压缩bin格式的内核，启动速度最慢(比elf格式慢了至少２s)，体积大约2.2M.

3.4.2)常见的错误烧写命令：

-------

以下是常见的不正确的烧写命令，请注意。

3.4.2.1)名为vmlinux的内核：

NBL> cp net:tftp:vmlinux sdmd:raw:vmlinux.bin

3.4.2.2)名为Image的内核：

NBL> cp net:tftp:Image sdmd:raw:Image

或

NBL> cp net:tftp:Image sdmd::

或

NBL> cp net:tftp:Image sdmd:raw:vmlinux

3.4.2.3)名为zImage的内核：

NBL> cp net:tftp:zImage sdmd:raw:zImage

或

NBL> cp net:tftp:zImage sdmd::

或

NBL> cp net:tftp:zImage sdmd:raw:vmlinux

或

NBL> cp net:tftp:zImage sdmd:raw:zImage.gz

3.4.3）正确的烧写命令：

-------

以下命令请烧写的时候使用。

3.4.3.1)名为vmlinux的内核：

NBL> cp net:tftp:vmlinux sdmd:raw:vmlinux.elf

或

NBL> cp net:tftp:vmlinux sdmd:raw:vmlinux

或

NBL> cp net:tftp:vmlinux sdmd::

也就是说，vmlinux可以不要后缀，或者使用".elf"的后缀进行烧写。

3.4.3.2)名为Image的内核：

NBL> cp net:tftp:Image sdmd:raw:vmlinux.bin

也就是说Image只能用".bin"格式的后缀进行烧写。

3.4.3.3)名为zImage的内核：

NBL> cp net:tftp:zImage sdmd:raw:vmlinux.bin

也就是说zImage只能用".bin"格式的后缀进行烧写。

以上为好用的烧写命令，烧写完毕后应该用"bootprof "设置参数，然后用"aboot -n" 即可启动。

## 交叉编译工具链的获取

环境中，使用的都是sony提供的制作好的交叉编译工具链，这里介绍一般arm平台下面交叉编译工具链获取方法以及实践步骤。

据目前所知，制作交叉编译工具链的方法有三种,从易到难依次为：

（1）、直接下载别人已经制做好的交叉编译工具链。

（2）、使用crosstool脚本工具半自动地制作交叉编译工具链。

（3）、自己下载源代码，手动编译并制作交叉编译工具链。

这里只给出最简单的方法。

直接到指定网站上面下载人家已经制作好的交叉编译工具链。这个是最简单的获取交叉编译工具链方法。通常下载的有rpm包或者tar包等。前者使用rpm命令可以直接将交叉编译工具链安装到指定的路径。如果是tar包，那么的具体安装位置还需要参考其中的文档，或者也可以使用包中的arm-linux-gcc -v来得到安装位置。把解压后的包拷贝到相应的位置，然后添加环境变量就行了。

(3.5.1)提供编译好的可用的交叉编译工具链的常用的网站有：

http://www.kernel.org/pub/tools/crosstool/

说明查看：

http://www.kernel.org/pub/tools/crosstool/index\_old.shtml

但是，这个网站提供的都没有glibc.

(3.5.2)另外做的很好的可以下载的交叉编译工具链的网站：

http://www.codesourcery.com

其中的:http://www.codesourcery.com/sgpp/lite/arm/portal/release1600

下面可以下载(其中的lite的是免费的)。

下载之后，直接解压，然后把其中的bin目录添加到环境量就可以直接编译内核和app了。

(3.5.3)实验工具链使用的过程：

3.5.3.1)下载工具链

$wget http://www.codesourcery.com/public/gnu\_toolchain/arm-none-linux-gnueabi/arm-2010.09-50-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

3.5.3.2)安装工具链

#tar -xjvf arm-2010.09-50-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 -C /usr/local

#export PATH=/usr/local/arm-2010.09/bin/:$PATH

3.5.3.3)使用工具链编译内核

#cd <kernel path in sony release svn>/Kernel

#make ARCH=arm CROSS\_COMPILE=arm-none-linux-gnueabi- mrproper

#make ARCH=arm CROSS\_COMPILE=arm-none-linux-gnueabi- avp13\_defconfig

#make ARCH=arm CROSS\_COMPILE=arm-none-linux-gnueabi-

这样编译的内核,使用编译生成的elf格式的二进制烧写即可启动。

3.5.3.4)制做对应的文件系统

想要测试是否能够正常运行app，首先需要制作一个简易的文件系统，如下：

#wget http://www.busybox.net/downloads/busybox-1.17.3.tar.bz2

#tar -xjvf busybox-1.17.3.tar.bz2

#mkdir {build,target}

#cd busybox-1.17.3

#make O=$(pwd)/../build defconfig

#cd $(pwd)/../build

#make menuconfig

Build Options --->

[\*] Build BusyBox as a static binary (no shared libs)

...

(arm-none-linux-gnueabi-) Cross Compiler prefix

...

#make

#make CONFIG\_PREFIX=../target install

这样，在../target中有基本的busybox工具了，在其中建立好proc,sys,以及dev等目录，并建立好init后即可成为最简单的文件系统,经过测试这样做出来的文件系统是好用的。

最后，将编译好的文件系统挂载成nfs文件系统，准备从开发板启动它。

3.5.3.5)编辑并编译一个小型的"hello world"程序：

$vi main.c

内容如下：

/\*! @file main.c \*/

#include <stdio.h>

int main(int argc, char \*argv[])

{

printf("hello!\n");

return 0;

}

$make CC=arm-none-linux-gnueabi-gcc CFLAGS+=-static main

将生成的"main"程序拷贝到nfs文件系统上面即可正常运行。