全 志 科 技
Allwinner Technology

A20 LinuxBSP 使用说明

V2.1

2014-06-19

Revision History

Version	Date	Changes compared to previous issue
V1.0	2013-03-15	初建版本
V2.0	2013-12-27	增加 boot2.0 去掉 boot1.0 部分
v2.1	2014-06-19	更正源码目录结构，修正固件打包的描述。

目录

1. 概述

本文档用于介绍全志科技 A20 芯片的 Linux BSP 的组织结构、内部机制以及简单用法。该文档的目的用于指导用户如何定制和使用该 BSP。

2. 开发环境准备

2.1. 硬件资源

- A20 IC-Based SOC；
- 运行 LINUX 的电脑一台（用于编译和烧写）；
- 串口线，适配的电源和适配的 USB 线。

2.2. 软件资源

- 编译载体建议安装 Red Hat Enterprise Linux Server release 6.0 (64 bit)或者 Ubuntu 10.04/12.04(64 bit)。要求至少安装 gcc, ncurse, bison, autoconf, wget, patch, texinfo, zlib, dos2unix 软件包；
- 我们使用的交互编译工具为 arm-linux-gnueabi-gcc-4.6.3，内置于 BSP 中。

3. 目录结构介绍

BSP 主要由 Buildroot, Linux kernel(版本 3.4)和 Bootloader（basic_boot 和 u-boot）组成。其中 Buildroot 主要用于制作 Linux 根文件系统，Linux Kernel 是 BSP 的核心，Bootloader 用于引导 Linux Kernel 和量产。

3.1. buildroot

它的主要作用是

- 管理包之间的依赖关系；
- 安装 ARM 交叉工具链；
- 制作根文件系统，可以包含 strace, directfb, oprofile 等非常丰富的应用软件和测试软件；
- 管理 BSP 的编译。

它位于 lichee/buildroot 目录下，目录结构如下

```
buildroot
├── board
├── boot
├── build.sh
├── CHANGES
├── Config.in
├── configs
├── COPYING
├── dl
├── docs
├── external-packages
├── fs
├── linux
├── Makefile
├── output
├── package
├── README
├── scripts
├── target
└── toolchain
```

3.2. linux-3.4

Linux-3.4 位于 lichee/linux-3.4 目录结构如下：

.

```
linux-3.4
├── arch
├── bImage
├── block
├── build.sh
├── crypto
├── Documentation
├── drivers
├── firmware
├── fs
├── include
├── init
├── ipc
├── Kbuild
├── Kconfig
├── kernel
├── lib
├── Makefile
├── mm
├── modules
├── net
├── output
├── rootfs
├── samples
├── scripts
├── security
├── sound
├── tools
├── usr
├── version
├── virt
```


以上目录结构跟标准的 Linux 内核是一致的，除了多一个 modules 目录。modules 目录是我们扩展用来存放没有跟内核的 menuconfig 集成的外部模块

的地方。目前放了 example,nand，mali 这 3 个外部模块，其中 example 是示例用的，mali 是 GPU 驱动,nand 目录用来存放 nand 驱动。

```
modules
├─── example
├─── nand
└─── mali
```

3.3. Bootloader

Bootloader 位于 lichee/brandy 目录中目录结构如下。

.

```
brandy
├─── build.sh
├─── gcc-linaro
├─── u-boot-2011.09
```

build.sh：编译脚本

u-boot-2011.09：是目前使用的 u-boot 的位置

3.4. tools

目录结构如下：

```
tools
├─── boot-v1.0
├─── brandy
├─── buildroot
├─── build.sh
├─── linux-3.4
├─── out
├─── README
└─── tools
```

该目录为打包目录，与打包相关的脚本和工具，以及 BSP 的配置和都放在该

目录中。

4. 内部工作机制

以 sun7i 为例子

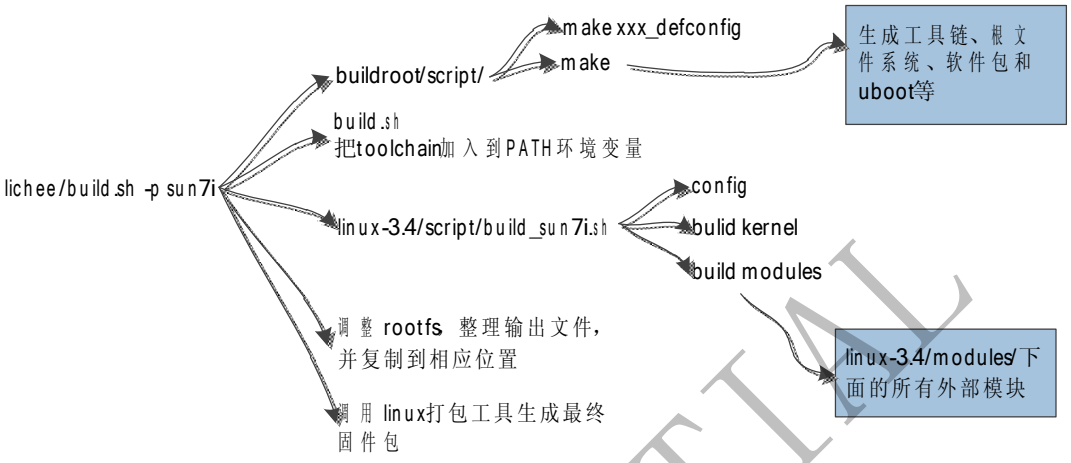

图 3.1 自动化编译流程图

注意：**在执行 build.sh 脚本时需要指定参数，具体可以参考./build.sh -h 输出**

5. 编译代码

在 lichee 目录下，键入指令

$ ./build.sh -h

可以得到如下的编译帮助信息：

```
USAGE: buildroot/scripts/mkcommon.sh [flags] args

flags:

   -p,--platform:   platform to build, e.g. sun7i (default: 'sun7i')

   -k,--kernel:   kernel to build, e.g. 3.3 (default: '3.4')

   -b,--board:   board to build, e.g. evb (default: '')

   -m,--module:   module to build, e.g. buildroot, kernel, uboot,

         clean (default: '')

   -v,--boot:   boot to build. e.g. boot_v1.0 (default: 'boot_v2.0')

   -i,--[no]independent:   output build to independent directory

         (default: false)

   -h,--[no]help:   show this help (default: false)
```


帮助信息解释：
-h　获取帮助信息
-p　编译平台选择，sun7i -- linux 编译，sun7i_android -- android 编译
-m　指定编译目录，可选 kernel，buildroot，uboot。缺省为 3 个一起编译
-i　编译输出到单独的目录

如果是编译 linux 固件：

$ ./build.sh -p sun7i_linux

编译成功输出：

```
INFO: build rootfs OK.

INFO: build lichee OK.
```

6. 打包固件

打包是指将我们编译出来的 bootloader，内核，和根文件系统一起写到一个镜像文件中，这个镜像文件也叫固件。然后可以将这个镜像写到 nand flash 或是 sd 卡上，从而启动我们的系统。

6.1. 自动打包

编译完成后便可打包（打包 android 具体参见 android 的相关文档），在 Lichee 目录下键入：

```
$ ./build.sh pack
```

随后会出现 3 次选择，屏幕上会出现如下输出：

```
Start packing for Lichee system


####################

#     use boot2.0     #

####################

All valid chips:

0. sun7i

Please select a chip:0

All valid platforms:

0. android

1. dragonboard

2. linux

Please select a platform:2

All valid boards:
```

```
0. evb-v10

1. k70

Please select a board:0

sun7i linux evb-v10

INFO: Packing for linux
```

假如添加了自己的方案板，最后一个选项中就会出现新方案板的名称。
生成的 image 文件在 lichee/tools/pack_brandy 目录.
例如：lichee/tools/pack_brandy/sun7i_linux_evb-v10.img

6.2. 定制 Nand 分区

(1) 规划磁盘分区

分区，是指存储设备(通常是 nandflash 或者 sdcard)上，根据逻辑关系划分的空间。习惯上，分区的编号从 0 开始，代表第一个分区，1 代表第二个分区，以此类推。这项规则类似于 PC 上的硬盘分区，如图 X 所示。

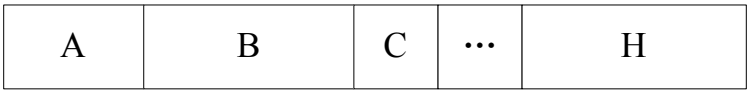

图 6.1 分区示意图

图 6.1 表示，存储设备上一共有 A-H 共 8 个分区，其中，分区 A 的起始位置从存储设备的头部开始，是第一个分区，分区 H 占用了尾部，是最后一个分区。规划分区，是指在固件包中指明存储设备上的分区个数，并由用户自己定义分区属性。当烧写固件包后，存储设备上就会存在这样由用户定义的分区。用户可以通过这样的规划，修改图 6.1 的分区，成为如下的情况：

图 6.2 修改后的分区示意图

通过规划分区可以看出，B 分区的容量减小，C 分区容量增大，同时减少了 H 分区。

要在存储设备上规划分区，需要按照如下的步骤做（以 ref-001 方案为例）：

打开 lichee/tools/pack_brandy/chips/sun7i/configs\android(linux)\ref-001 目录下的 sys_partition.fex 文件；分区配置存放在 sys_partition.fex 脚本中，它里面描述了分区信息。

分区起始以[partition_start]为标志，后面连续存放每个分区的信息。当遇到非分区信息或者结束，认为分区的配置结束。

每个分区的完整配置如下：

```
[partition]
name            = bootloader
size            = 32768
downloadfile = "bootloader.fex"
user_type       = 0x8000
```

- partition：表示这是一个分区，每个分区配置都以它为开始；
- name：分区名称，最大 16 个字符；
- size：分区大小，以扇区为单位；当为 0 时，此分区无法操作；
- downloadfile：下载文件名称，可以带相对路径或者绝对路径，可以有后缀或者无后缀；当分区不需要烧录文件时，此项留空或者直接删除此项；
- user_type：默认值为 0x8000，如果为 0xc000 打开掉电保护；

分区个数根据配置的 patition 项为标准。比如下面配置了 4 项 patition，那么代表 4 个分区信息，每个分区信息由 parttition 进行标志。
下面给出一个完整的分区表示例：

```
[partition_start]
[partition]
    name         = bootloader
    size         = 32768
    downloadfile = "bootloader.fex"
[partition]
    name         = env
    size         = 16384
[partition]
    name         = boot
    size         = 16384
[partition]
    name         = rootfs
    size         = 524288
```

6.3. 固件烧写

请参考 PhoenixSuit 软件自带的《PhoenixSuit 烧写使用说明文档.pdf》

7. 定制根文件系统

7.1. 修改 Nand Flash 的 rootfs

1) 复制一份现有的配置文件
   $cd lichee/buildroot
   $cp configs/sun7i_defconfig .config
2) 进入 buildroot 界面进行配置
   $make ARCH＝arm menuconfig
   上述命令后，显示下面的界面

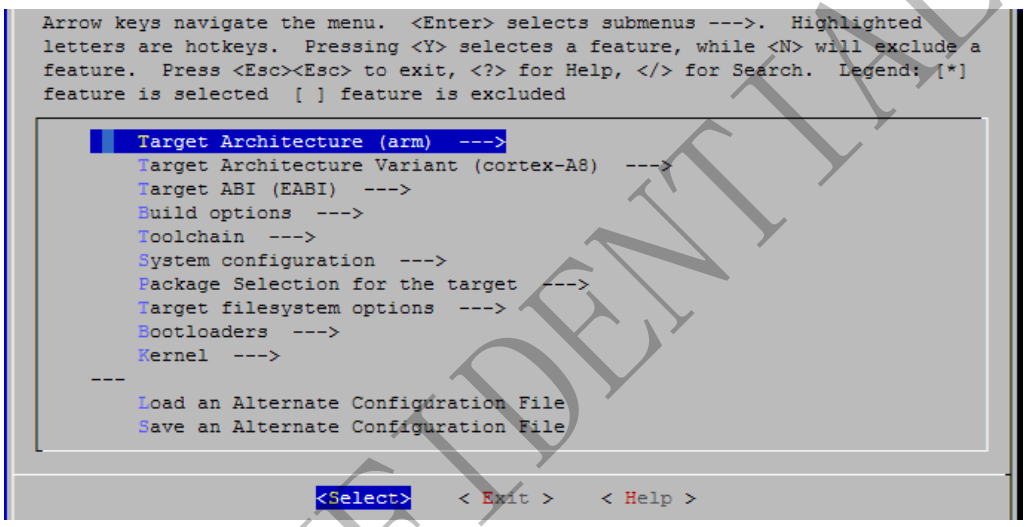

图 7.1 linux 内核 make menuconfig 界面

配置完后保存，然后到 lichee 目录下重新运行 build.sh 脚本。

编译过程中，如果有软件包缺失，则 buildroot 会自动从网上下载，而此时如果编译机器无法连接网络，则需要从网上下载相应版本的软件包，把软件包复制到 buildroot/dl 目录下面。

8. 集成软件包

8.1. 源代码包

对于用户态的应用程序、动态库和静态库应该集成到 buildroot 中，在 buildroot/packages 下面每个目录对应一个包。关于如何在 buildroot 中集成软件包的说明，请参考 http://buildroot.uclibc.org/docs.html。

举一个简单的例子：

要在 buildroot 下添加一个源码包，首先要在 buildroot/package 目录下新建一个目录，目录名称为软件包的名称，目录中，再在目录中添加一个 config.in 文件和一个 xxxx.mk 文件（xxxx 为软件包的名称）。这 2 个文件的具体写法，参见 buildroot/package 目录下的其他的软件包，或者官方网站（软件源码包分为网上的官方软件包和自己编写的源码包，这 2 类包的 config.in 文件形式是一致的，但是.mk 文件的书写会有较大区别，假如是后者，请参见 fsck-msdos 包中的.mk，前者请参见 argus 包中的.mk）。做完以上操作以后，还需要在 buildroot/package 目录下的 config.in 文件中添加：

```
source "package/panlong/Config.in"
```

注意：**假设要添加的软件包的名称为 panlong 的话。至于段代码添加的位置由具体情况而定，添加位置影响执行 make menuconfig 是软件包对应选项的位置。**

示例：

```
menu "Package Selection for the target"


source "package/busybox/Config.in"

source "package/customize/Config.in"



#source "package/lcd-test/Config.in"

#source "package/tp-test/Config.in"

#source "package/kernel-header/Config.in"
```

```
#source "package/sw-tools/Config.in"

#source "package/ext4-utils/Config.in"

#source "package/tiobench/Config.in"

#source "package/fsck_msdos/Config.in"

#source "package/mali-3d/Config.in"

#source "package/cedar/Config.in"

source "package/panlong/Config.in"

# Audio and video applications

source "package/multimedia/Config.in"
```

这里“#”开头的行在执行 make menuconfig 时是看不到的。我们将 source "package/panlong/Config.in"添加到了 menu "Package Selection for the target" 菜单下，所以在我们执行 make menuconfig 后：

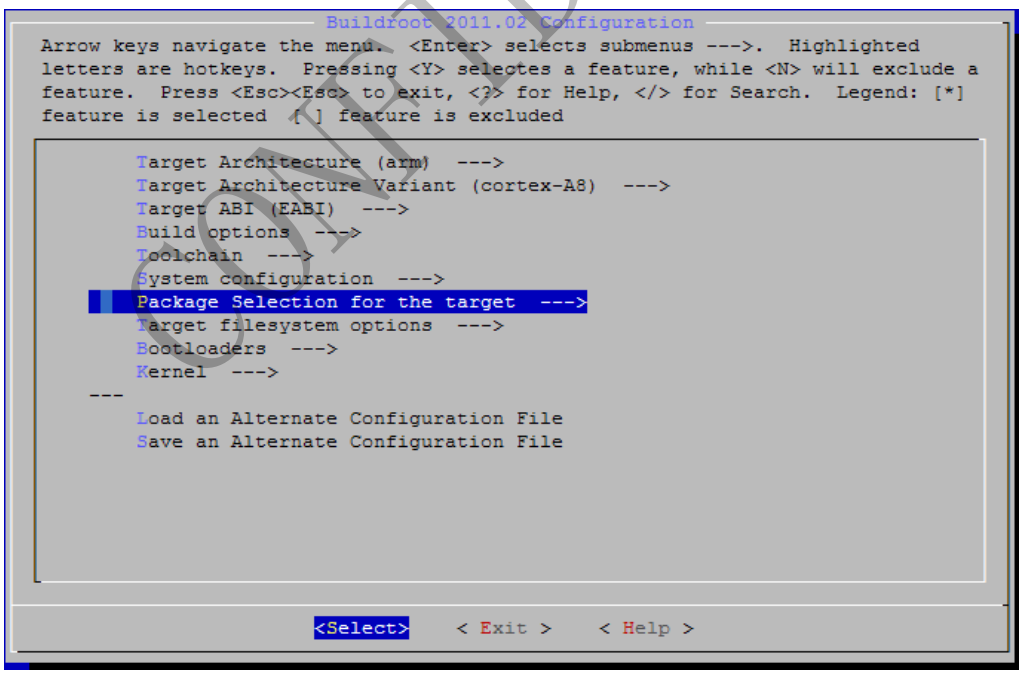

图 7.2 Buildroot make menuconfig 界面

做如图的选择，按下 enter 键：

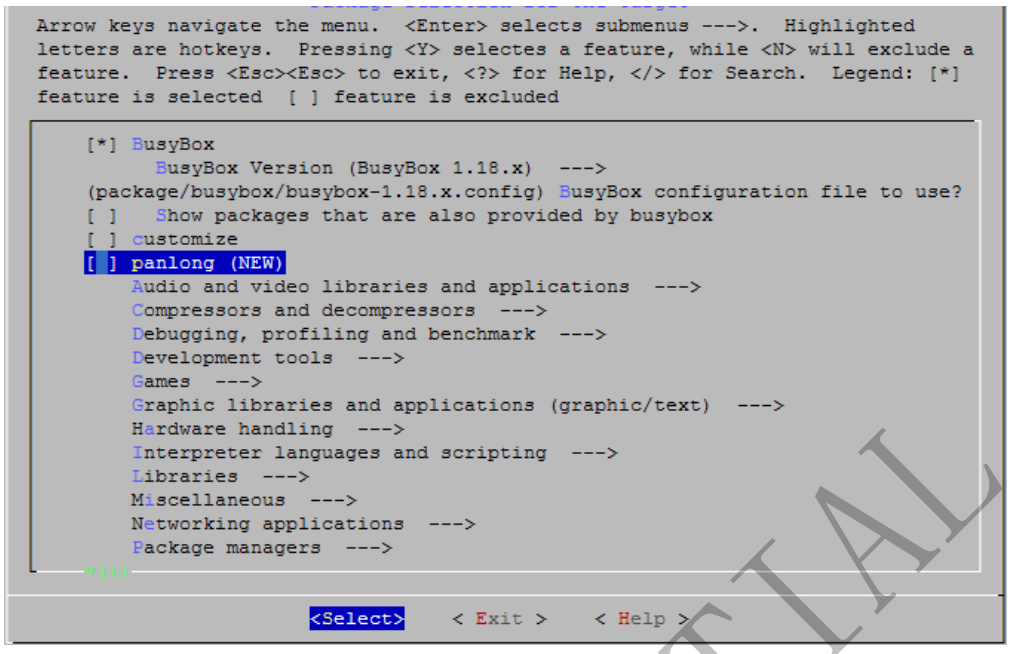

图 7.3 package selection for the target 子菜单界面

就可以看到我们添加的软件包了。

注意：**以上只是演示，实际添加时尽可能添加到子菜单中，以便于软件包的管理。**

对于内核驱动，应该尽量考虑放到 linux-3.4/drivers 下面，如果无法直接跟 kernel 的 menuconfig 集成，则应该放在 linux-3.4/modules 下面。

能够和 menuconfig 集成的软件包，其添加方法参见 kconfig 相关资料。

无法与 menuconfig 集成的软件包，用 modules 下的 mali 来进行添加举例：首先，在 modules 目录下建立 mali 包的子目录，然后为这个包编辑一个总的 makefile，这里可能会用到 4 个参数：

```
LICHEE_KDIR：就是 buildroot 和 linux-3.4 所在的那一层目录

LICHEE_MOD_DIR= =${LICHEE_KDIR}/output/lib/modules/${KERNEL_

VERSION}KERNEL_VERSION= 3.3

CROSS_COMPILE= arm-linux-gnueabi-

ARCH=arm
```

这些参数的定义都在 linux-3.4/scripts/build_XX.sh 中定义（xx 表示你编译时选择的-p 后的参数，如 sun7i 等）

完成 makefile 的编辑后，为了让系统整体编译时让其被编译进去，还需在 linux-3.4/scripts/build_XX.sh 文件的 build_modules()函数中添加对 nand，wifi，eurasia_km gpu 软件包的编译规则，以及在 clean_modules()函数中添加清除规则。（具体写法可以仿照 nand）。

假如添加的项目是默认打开的，那么就需要用编辑好的.config 文件替换掉对应的 defconfig。如 sun7i 的，我们就可以把 buildroot 下的.config 重命名为 sun7i_defconfig，然后保存到 buildroot\configs 文件夹下。

8.2. 二进制包

同上，只是忽略掉编译过程。可以参考 buildroot/packages/mali-3d。

8.3. 可执行文件

假如需要添加的是一些可执行文件或者是类似 ls/cd 等指令，则可以直接添加到 lichee\out\linux\common\buildroot\output\target 中（前提是已经完全编译过一次），指令直接添加到 bin、sbin 或者 usr 下的 bin、sbin 中，其他可执行文件可以添加在希望指定的任意文件夹下。

9. 附录

9.1. 关于驱动开发

请参考驱动开发相关文档。

9.2. 在线帮助文档

makefile 帮助文档
http://www.gnu.org/software/make/manual/make.html
buildroot 帮助文档
http://buildroot.uclibc.org/downloads/buildroot.html

10. Declaration

This is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.