

# Mocor5 编译系统介绍

---

Version: 1.0

Date: 2018-07-03

# 声明

本文件所含数据和信息都属于紫光展锐机密及紫光展锐财产，紫光展锐保留所有相关权利。当您接受这份文件时，即表示您同意此份文件内含机密信息，且同意在未获得紫光展锐同意前，不使用或复制、整个或部分文件。紫光展锐有权在未经事先通知的情况下，对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与文件相关的直接或间接的、任何伤害或损失。

# 前言

## 文档说明

本文档简要介绍了紫光展锐公司 SC9820E Mocor5 平台芯片的编译系统及其使用方法。

## 阅读对象

本文适用于所有进行紫光展锐公司 SC9820E Mocor5 平台产品开发的客户软件工程师。

## 内容介绍


本文档包括三个章节，分别为：

- 第一章：编译方法介绍
- 第二章：新建工程和配置介绍
- 第三章：其他相关配置

## 文档约定

本文档采用下面醒目标志来表示在操作过程中应该特别注意的地方。

---

 注意：

提醒操作中应注意的事项。

---

## 相关文档

# 目 录

<b>第 1 章 代码编译的方法.....</b>	<b>3</b>
1.1 适用范围 .....	3
1.2 代码和编译环境的准备.....	3
1.3 完成一次全新的编译.....	4
1.4 单项编译和其它编译命名.....	6
1.5 编译的成果 .....	7
<b>第 2 章 新建和配置一个项目 .....</b>	<b>10</b>
2.1 新建项目编译配置文件.....	10
2.2 配置新项目的 kernel 部分.....	13
2.3 配置新项目的 u-boot 部分.....	14
2.4 配置 chipram 部分 .....	16
2.5 完成配置准备编译.....	16
<b>第 3 章 其它编译相关的内容 .....</b>	<b>17</b>
3.1 OTA 包的编译 .....	17
3.2 如何解决 Jack server 端口冲突导致编译失败的问题.....	17

# 第1章 代码编译的方法

## 1.1 适用范围

该文档适用于展讯 9820e 芯片在 Mocor5 (android4.4) 编译和配置。

## 1.2 代码和编译环境的准备

首先，客户需要解压完整的平台代码包，代码包由 CPM 向客户进行发布，其中包含代码，bin 文件和开发调试工具等。其中 AP 侧代码由开源代码包和非开源库文件两部分组成

开源代码包部分一般命名为 idh.code，以 rar 或者 tgz 压缩格式提供

非开源库文件一般包括：

- 非开源程序和库：proprieties-<平台名>-xxx.zip
- 芯片配置文件 conf-<平台名>.tar.gz

客户解压 proprieties-<平台名>-xxx.zip 之后，需要将产生的 out 目录移动到

一个路径下，如/home10/xxx/project/sp9820e\_1h10 下，然后在该项目的项目产品配置文件 device/sprd/sharkle/sp9820e\_1h10/sp9820e\_1h10\_native.mk 中 添 加 SPRD\_IDH\_PROP:= /home10/xxx/project/sp9820e\_1h10/out，即可。

芯片配置文件则需要拷贝到 device/sprd/sharkle/下面。

客户需要检查自己的编译环境，google 推荐使用 64 位 ubuntu 的系统。展讯推荐 14.04 的版本。Ubuntu 10.04 - 12.04 版本也可以。可以使用下面命令来查看 ubuntu 的具体版本号：

```
lsb_release -a
```

- 需要安装 jdk 1.6，可以使用下面命令来查看 jdk 的版本：

```
java -version
```

自行上网搜索版本安装方法

- Google 推荐的 python 版本是 2.6 或者 2.7，可以在 python.org 获得，可以使用下面命令来查看 python 的版本：

```
python --version
```

- 根据 ubuntu 版本的不同，可能还需要一些其它的编译支持工具，完整的工具包在下面的网

址可以找到:

<http://source.android.com/source/initializing.html>

例如 ubuntu14.04, 可使用如下命令, 进行初始化所需工具包:

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \  
zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 \  
lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache \  
libgl1-mesa-dev libxml2-utils xsltproc unzip
```

在完成的代码和编译环境的准备之后, 就可以开始进行代码的编译工作了。

### 1.3 完成一次全新的编译

在完成了代码环境的准备后就可以进行一个完整的编译了, 当然, 客户也可以选择在完成自定义项目配置之后再开始编译, 但是我们还是建议不熟悉展讯环境的客户在准备好代码之后先进行一次默认项目的编译

通过 ubuntu 终端命令行工具进入代码的根目录, 默认的代码根目录是 /home10/xxx/project/sp9820e\_1h10/idh.code

首先执行

```
source build/envsetup.sh
```

这一步将读取各个项目的编译配置文件, 然后执行

```
lunch
```

此时终端会显示出所有被配置过的项目的列表, 如下图所示:

```
1. aosp_arm-eng  
2. aosp_x86-eng  
3. aosp_mips-eng  
4. vbox_x86-eng  
5. sp9820e_1h10_native-userdebug  
6. sp9820e_1h10_multilanguage-userdebug  
7. sp9820e_1h10_oversea-userdebug  
8. sp9820e_2h10_native-userdebug  
9. sp9820e_2h10_oversea-userdebug
```

输入对应的数字选择需要编译的项目，如果想使用 user 版本，则直接输入对应的项目名并去除 debug 关键字，例如使用对应的 user 版本：

sp9820e\_1h10\_native-user

```
=====
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.4.4
TARGET_PRODUCT=sp9820e_1h10_native
TARGET_BUILD_VARIANT=user
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a-neon
TARGET_CPU_VARIANT=generic
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-3.19.0-25-generic-x86_64-with-Ubuntu-14.04-trusty
HOST_BUILD_TYPE=release
BUILD_ID=KTU84P
OUT_DIR=out
=====
```

PLATFORM\_VERSION: 为平台对应的版本，我们使用的是 4.4 版本

TARGET\_PRODUCT: 要编译的工程名

TARGET\_BUILD\_VARIANT: 显示要编译的工程是 user 版本还是 userdebug 版本

TARGET\_ARCH: 为对应的硬件信息

TARGET\_CPU\_VARIANT: cpu 信息

HOST\_ARCH: 编译平台对应的硬件信息

OUT\_DIR: 编译生成的目标文件目录

其中 base 或 plus 关键字分别代表单卡或者双卡方案。这里建议客户选择最接近自己项目形态的参考项目。

kheader

在选择完编译项目后，先执行 kheader, 完成安装 kernel 提供给用户态程序使用的头文件，然后使用

make

命令来进行编译，如果编译使用的机器是支持多线程编译的，则可以使用-j 选项来加快编译的速度，比如

```
make -j24
```

-j 之后的数值表示多线程并行编译，主要在于编译器是否支持多线程并行编译，同时跟 cpu 有很大关系。一次全新的编译根据编译服务器的性能大约需要几十分钟到几个小时不等。

## 1.4 单项编译和其它编译命名

在完成一次全编之后，在不改变当前编译项目的前提下，修改代码后可以使用单项的编译来编译对应的部分，加快开发的速度。

---

### 注意：

如若打开新的终端并进行重新编译，则需执行 source 重新配置编译环境，并使用 lunch 的操作选择对应的编译项目

---

- 各个部分的编译命令如下

1. 单独编译 u-boot

```
make bootloader
```

主要生成目标文件：fdl2-sign.bin u-boot-sign.bin u-boot\_autopoweron-sign.bin

2. 单独编译 fd11 和 uboot-16k

```
make chipram
```

主要生成目标文件：fd11-sign.bin u-boot-spl-16k-sign.bin

3. 单独编译 boot image

```
make bootimage
```

主要生成目标文件：boot.img dt.img kernel ramdisk.img

4. 单独编译 system image

```
make systemimage
```

主要生成目标文件：system.img

5. 单独编译 userdata image



```
make userdataimage
```

主要生成目标文件: userdata.img

```
6:make vendorimage
```

主要生成目标文件: vendor.img(注意 4.4 上面 selinux 相关的内容修改的话要编译到 vendorimage)


我们还可以单独编译 android 的每一个模块, 比如单独编译一个 apk, 一个 java 或者本地库或者本地程序, 这时我们需要进入到对应模块的 Android.mk 所在的目录, 执行 mm 指令, 比如需要重新编译”设置”这个 apk, 我们就需要这样做

```
cd packages/apps/Settings/
```

```
mm
```

这样被单独编译出来的模块可以通过 adb push 的方式推入调试手机进行使用, 是调试阶段被经常使用到的方式。

---

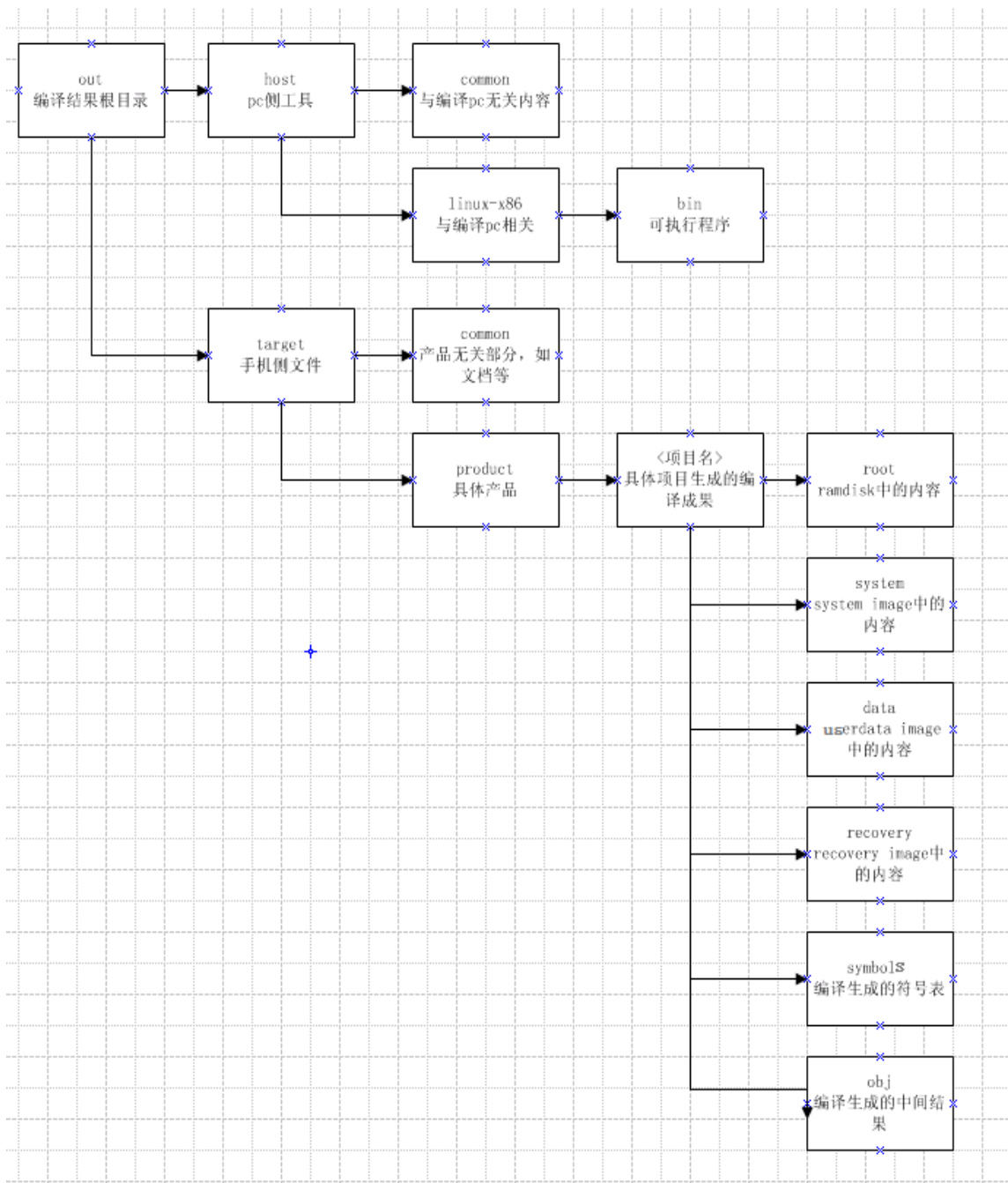
 注意:

ramdisk (手机根目录或者/bin 目录) 中的文件不建议使用 adb push, 需要重新下载 bootimage

---

## 1.5 编译的成果

Android 的编译输出路径为 out, 编译成果如下图



其中最重要的目录就是 out/target/product/<项目名>, 这里存放着用于下载的所有 bin 和 image 文件, 包括 fd11-sign.bin fd12-sign.bin u-boot-spl-16k-sign.bin u-boot-sign.bin boot.img system.img userdata.img recovery.img cache.img vendor.img。

#### 注意:

并非所有的下载用文件都是编译生成的, 比如 cp 侧的 bin 就是在版本发布中直接提供

```
out/target/product/<项目名>/root  
out/target/product/<项目名>/system  
out/target/product/<项目名>/data  
out/target/product/<项目名>/recovery
```

这四个目录分别是 boot system userdata 和 recovery image 中的直接内容，其中的文件和手机运行后各个对应分区中的内容是一一对应的，当我们通过 mm 指令来编译某个特定的 Android 模块时，更新的也是这些目录中的文件。

另外编译的符号表在很多调试和 bug 解决中是非常重要的，所有符号表可以在 out/target/product/<项目名>/symbols 目录下找到，我们也可以在 out/target/product/<项目名>/obj 目录下找到同样的内容，不同的是 obj 目录更加具体，不仅仅有符号表，而且有所有 c/c++ java 文件的中间编译结果。同时，kernel 的符号表 vmlinux 也可以在 out/target/product/<项目名>/obj/KERNEL 目录下找到。

在 out/target/host/linux-x86/bin 目录下有一些常用的 pc 侧工具，包括 fastboot mkbooting adb 等。

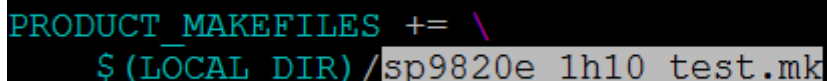
## 第2章 新建和配置一个项目

### 2.1 新建项目编译配置文件

项目的编译配置文件所在目录为 device/sprd/<项目名>，由于项目的编译配置内容较多，我们建议客户选择一个已有的项目作为参考，通过拷贝的方式新建自己的项目。例如，可以 sp9820e\_1h10 为模版新建一个 sp9820e\_1h10\_test 项目（在 device/sprd/sharkle 目录下）：

```
cp -avr sp9820e_1h10 sp9820e_1h10_test
```

- 在完成拷贝后，首先修改在 sp9820e\_1h10\_test/AndroidProducts.mk 中包含在该项目下的各工程：



```
PRODUCT_MAKEFILES += \  
$(LOCAL_DIR)/sp9820e_1h10_test.mk
```

- 接着将 sp9820e\_1h10\_test 的 board 中对应的各 mk 修改成和 AndroidProducts.mk 中的各工程同名，以 sp9820e\_1h10\_native.mk 为例(客户可以根据自己的需求，可以在此项目上派生出更多或仅配置需要的产品)：

```
mv sp9820e_1h10_native.mk sp9820e_1h10_test.mk
```

- 修改 root 等路径下的 init.xx.rc 文件：

```
device/sprd/sharkle/sp9820e_1h10_test/rootdir/root$
```

```
mv init.sp9820e_1h10.rc init.sp9820e_1h10_test.rc
```

```
device/sprd/sharkle/sp9820e_test_1h10/recovery$
```

```
mv init.recovery.sp9820e_1h10.rc init.recovery.sp9820e_1h10_test.rc
```

- 修改 SPRD\_IDH\_PROP 变量：

在 sp9820e\_1h10\_test.mk 中确认配置了 SPRD\_IDH\_PROP，即

```
SPRD_IDH_PROP:=/home10/xxx/project/sp9820e_1h10_test/out
```

其中，/home10/xxx/project/sp9820e\_1h10\_test/out 是 proprietaries-<平台名>-xxx.zip 解压后的 out 目录所在的路径，这个路径可以自由设定，只要将 proprietaries-<平台名>-xxx.zip 解压后的 out 目录放置于其中即可。

- 修改 root 路径下的 fstab.xxx 文件:

```
device/sprd/sharkle/sp9832e_1h10_test/rootdir/root$
```

```
mv fstab.sp9820e_1h10 fstab.sp9820e_1h10_test
```

```
mv fstab.sp9820e_1h10.f2fs fstab.sp9820e_1h10_test.f2fs
```

```
mv fstab.sp9820e_1h10.f2fs.verify fstab.sp9820e_1h10_test.f2fs.verify
```

```
mv fstab.sp9820e_1h10.verify fstab.sp9820e_1h10_test.verify
```

- 修改项目名 TARGET\_BOARD。也是 out 目录后面输出的项目名称

```
device/sprd/sharkle/sp9820e_1h10_test/sp9820e_1h10_test.mk:
```

```
PLATDIR := device/sprd/sharkle
TARGET_BOARD := sp9820e_1h10
BOARDDIR := $(PLATDIR)/$(TARGET_BOARD)
PLATCOMM := $(PLATDIR)/common
ROOTDIR := $(BOARDDIR)/rootdir
```

```
PLATDIR := device/sprd/sharkle
TARGET_BOARD := sp9820e_1h10_test
BOARDDIR := $(PLATDIR)/$(TARGET_BOARD)
PLATCOMM := $(PLATDIR)/common
ROOTDIR := $(BOARDDIR)/rootdir
```

- 修改各个产品的 mk 文件中的产品名。

以 sp9820e\_1h10\_test.mk 为例:

```
PRODUCT_NAME := sp9820e_1h10_native
PRODUCT_DEVICE := sp9820e_1h10
PRODUCT_BRAND := SPRD
PRODUCT_MODEL := sp9820e_1h10_native
```

```
PRODUCT_NAME := sp9820e_1h10_test
PRODUCT_DEVICE := sp9820e_1h10_test
PRODUCT_BRAND := SPRD
PRODUCT_MODEL := sp9820e_1h10_test
```

PRODUCT\_NAME //产品名, 必须与 product makefile(sp9820e\_1h10\_test.mk)一致

PRODUCT\_MODEL //一般同项目名

- 在 vendorsetup.sh 将产品添加到 lunch 编译选项

需要修改的是 add\_lunch\_combo, 如

```
add_lunch_combo <产品名>-userdebug
```

至此, 在 android 根目录下, 重新执行 envsetup.sh 和 lunch 之后就会看到新增加的项目产品了。

```
1. aosp_arm-eng
2. aosp_x86-eng
3. aosp_mips-eng
4. vbox_x86-eng
5. sp9820e_1h10_native-userdebug
6. sp9820e_1h10_multilanguage-userdebug
7. sp9820e_1h10_oversea-userdebug
8. sp9820e_2h10_native-userdebug
9. sp9820e_2h10_oversea-userdebug
10. sp9820e_1h10_test-userdebug
```

- 在对应项目中(sp9820e\_1h10\_test.mk)中添加 u-boot 的支持等

需要修改的项包括

TARGET\_BOOTLOADER\_BOARD\_NAME //u-boot 所使用的板级配置名称

CHIPRAM\_DEFCONFIG //chipram 所使用的配置文件名

UBOOT\_DEFCONFIG //u-boot 所使用的配置文件名

UBOOT\_TARGET\_DTB //u-boot dtb 文件名, u-boot15/AndroidUBoot.mk 中用到

TARGET\_DTB //kernel 目录下的 dtb 文件名,  
device/sprd/sharkle/common/generate\_dt\_image.mk 中用到

这些项的配置是用于配置 u-boot、chipram 和 kernel 的,在 u-boot 配置、chipram 配置和 kernel 配置的部分会具体解释它们的使用方法。例如修改如下:

```
TARGET_BOOTLOADER_BOARD_NAME := sp9820e_1h10
CHIPRAM_DEFCONFIG := sp9820e_1h10
UBOOT_DEFCONFIG := sp9820e_1h10
UBOOT_TARGET_DTB := sp9820e_1h10
```

```
TARGET_DTB := sp9820e-1h10-native
```

另外如果当前的工程 mk 文件包含了其他一个 mk 文件,还需要修改以下路径

```
include device/sprd/sharkle/sp9820e_1h10/sp9820e_1h10_base.mk
```

并且相应修改 include 进来的 sp9820e\_1h10\_base.mk 文件内容,如下

```
TARGET_BOARD := sp9820e_1h10_test
```

- Kernel defconfig 配置文件定义说明

9820e 的 Kernel defconfig 公共配置文件是在 device/sprd/sharkle/common/BoardCommon.mk 中定义的, KERNEL\_DEFCONFIG := sprd\_sharkle\_fp\_defconfig

另外不同系统、不同 board 等差异配置文件在 device/sprd/sharkle/common/AndroidKernel.mk

中定义，如 `$(KERNEL_DIFF_CONFIG_ARCH)/$(TARGET_BOARD)_mocor5_diff_config`

## 2.2 配置新项目的 kernel 部分

在完成了项目的编译配置后，我们需要配置项目的 kernel 部分

- 添加对应的 kernel defconfig

所有项目对应的 kernel config 公共配置文件都位于 `kernel/arch/arm/configs` 目录下，以 `_defconfig` 结尾。需要注意的是，kernel config 的文件名需要与编译配置中 `BoardCommon.mk` 文件中 `KERNEL_DEFCONFIG` 项的配置一致。

9820e 不同系统、不同 board 等差异配置文件位于 `kernel\sprd-diffconfig\sharkle\arm` 目录下，如 `sp9820e_1h10_mocor5_diff_config`，不同 board 的差异化配置可在之内定义，这里需创建新建 board 的 config 文件，文件名需与编译配置中 `AndroidKernel.mk` 文件内的 `$(TARGET_BOARD)_mocor5_diff_config` 的定义一致

```
cp sp9820e_1h10_mocor5_diff_config sp9820e_1h10_test_mocor5_diff_config
```

- 添加对应的 dts 文件

1. 在 `kernel/arch/arm/boot/dts` 中, 创建 `sp9820e-1h10-test.dts`

```
cp sp9820e-1h10-native.dts sp9820e-1h10-test.dts
```

```
#include "sc9820e.dtsi"
#include "sp9820e-common.dtsi"

/ {
    model = "Spreadtrum SC9820e Board";

    compatible = "sprd,sp9820e-1h10-test", "sprd,sc9820e";

    sprd,sc-id = <9820 1 0x20000>;

    aliases {
        serial1 = &uart1;
    };
};
```

2. 修改 `kernel/arch/arm/boot/dts/Makefile`，如下图增加 `sp9820e-1h10-test.dtb \`

```
sp9820e-1h10-native.dtb \
sp9820e-1h10-test.dtb \
sp9820e-13c10-native.dtb \
sp9820e-5c10-native.dtb \
sp9820e-2c10-native.dtb \
sp9820e-1h10-kaios-native.dtb \
```

3. 修改 androidboot.hardware(跟 board 同名) 为 sp9820e\_1h10\_test

```
console=ttyS1,115200n8 loglevel=1 init=/init root=/dev/ram0 rw androidboot.hardware=sp9820e_1h10_test";
```

- 通过 dts 配置自己项目板子相关的硬件信息

kernel/arch/arm/boot/dts/sp9820e-1h10-test.dts

这些信息包括使用的设备信息, gpio, i2c, lcd 等等.

## 2.3 配置新项目的 u-boot 部分

- 添加 Kconfig 支持

在 u-boot15/board/spreadtrum/Kconfig 中, 增加

```
source "board/spreadtrum/sp9820e_1h10_test/Kconfig"
```

同时, 在 u-boot15/board/spreadtrum/Kconfig 中, 增加 board 的 kconfig 支持, 如下图

```
config TARGET_SP9820E_1H10_TEST
    bool "Spreadtrum sharkle"
    select CPU_V7
    help
        This is the Spreadtrum sharkle Board
```

- 添加板级代码目录

```
u-boot15/board/spreadtrum$
```

```
cp -avr sp9820e_1h10 sp9820e_1h10_test
```

另外修改 sp9820e\_1h10\_test 目录下的文件名以及 Makefile 文件中的编译文件名, 如下

```
mv sp9820e_1h10.c sp9820e_1h10_test.c
```

```
obj-y := sp9820e_1h10_test.o sdio_cfg.o sprd_kp.o modem_entry.o pinmap-sp9820e.o ldo_sleep.o sprd_b1.o regulator_init.o lcd_printf.o
```

- 修改配置头文件



1. 在 u-boot15/board/spreadtrum/sp9820e\_1h10\_test/Kconfig 中, 修改 SYS\_CONFIG\_NAME 为 "sp9820e\_1h10\_test", 如下图

```
config SYS_CONFIG_NAME
    default "sp9820e_1h10_test"
```

同时, 修改 TARGET\_SP9820E\_1H10 为 TARGET\_SP9820E\_1H10\_TEST 以及 SYS\_BOARD 为 "sp9820e\_1h10\_test"(必须与 board 目录下的项目名一致, Makefile 中的 BOARDDIR 变量会用到), 如下图

```
if TARGET_SP9820E_1H10_TEST

config SYS_BOARD
    default "sp9820e_1h10_test"
```

2. 在 u-boot15/include/configs 添加 .h 文件, 文件名必须与编译配置中 u-boot15/board/spreadtrum/sp9820e\_1h10\_test/Kconfig 文件中 SYS\_CONFIG\_NAME 的值一致, 同样建议拷贝添加。u-boot15/include/configs\$

```
cp sp9820e_1h10.h sp9820e_1h10_test.h
```

3. 在 u-boot15/configs 中, 添加 sp9820e\_1h10\_test\_defconfig, 文件名必须与 UBOOT\_DEFCONFIG 的定义一致, 如下

```
cp sp9820e_1h10_defconfig sp9820e_1h10_test_defconfig
```

并修改 sp9820e\_1h10\_test\_defconfig 中 CONFIG\_TARGET\_SP9820E\_1H10=y 为

CONFIG\_TARGET\_SP9820E\_1H10\_TEST=y

### ● 添加 u-boot 的 dts 配置

1. 在 u-boot15/arch/arm/dts/Makefile 中, 增加 sp9820e\_1h10\_test.dtb \, 如下图

```
sp9820e_64b.dtb \
sp9820e_1h10.dtb \
sp9820e_1h10_test.dtb \
sp9820e_13c10.dtb \
sp9820e_5c10.dtb \
sp9820e_2c10.dtb \
sp9820e_1h10_kaios.dtb \
```

2. 添加 uboot dts 文件, 文件名必须与 UBOOT\_TARGET\_DTB 的定义一致,

```
u-boot15/arch/arm/dts$
```

```
cp sp9820e_1h10.dts    sp9820e_1h10_test.dts
```

注意:

obj/u-boot15/u-boot.cfg 是生成的, 见 u-boot15/Makefile 中:

```
u-boot.cfg: include/config.h  
$(call if_changed, cpp_cfg)
```

## 2.4 配置 chipram 部分

### 添加配置头文件

在 chipram/include/configs 添加 .h 文件, 文件名必须与编译配置中 sp9820e\_1h10\_test.mk 文件中 CHIPRAM\_DEFCONFIG 的值一致, 同样建议拷贝添加:

```
cp sp9820e_1h10.h    sp9820e_1h10_test.h
```

- 在 board.def 中添加配置, 如下图, 目标名字必须是 \$(CHIPRAM\_DEFCONFIG)\_config

```
sp9820e_1h10_test_config : preconfig  
    @$(MKCONFIG) $@ arm armv8 sp9820e_1h10_test spreadtrum sharkle
```

## 2.5 完成配置准备编译

在完成了上面所有的操作后重新执行

```
source build/envsetup.sh
```

```
lunch
```

```
kheader
```

就可以选择新添加项目的产品进行编译了

## 第3章 其它编译相关的内容

### 3.1 OTA 包的编译

#### 1 版本的 ota 包生成方式

首先，我们需要完整的编译对应的版本代码，然后需要确认 cp 相关的 bin 文件已经拷贝到 device/sprd/sharkle/sp9832e\_lh10\_test/modem\_bins/目录下，根据不同的芯片，是不一样的。在 sharkle 平台上，包括 ltemodem.bin, ltegdsp.bin 等等。然后，使用命令

```
make otapackage
```

来生成 ota 包，ota 包会在 out 目录下以 <产品名>-ota-<序列号>.zip 的命名以压缩文件的格式生成

#### 2 ota 差分包的制作

在获得新旧两个版本的 ota 包之后，可以制作对应的升级差分包，命令为

```
./build/tools/releasetools/ota_from_target_files -i ota_old.zip ota_new.zip  
update.zip
```

ota\_old.zip 与 ota\_new.zip 分别是升级前和要升级到版本的 ota 包。

ota 包和升级包可以通过 sd 卡在 recovery 模式下进行 ota 升级。

更多信息见《Android 7.0-OTA 升级特殊注意事项》

### 3.2 如何解决 Jack server 端口冲突导致编译失败的问题

Google 在 android7.0 里启用了 jack-server (6.0 里也有这个东西，不过 6.0 里是可禁用的，6.0 默认是禁的，也可手动开启)，7.0 里默认是开启的，并且无法禁用，且 Jack-server 有个 bug—端口是写死的；所以在单人 PC 或服务器上第 1 个人是没问题的，碰到服务器上多人使用时就会因为端口冲突导致失败。

目前我们解决这个问题方法是提供一个通用的脚本，在第一次编译 android7.0 时，先执行一下脚本配置好端口然后进行编译；

脚本具体执行如下：

1. 首先在执行脚本之前需要将 jack-server 放到 /usr/local/bin/Jack 目录下
2. 然后将附件中的脚本也放到 /usr/local/bin/Jack 目录下
3. 创建 port1 和 port2 文件，写入初始端口号；

---

#### 4. 执行 sh /usr/local/bin/Jack/jack.sh

---

jack.sh 内容如下:

```
#!/bin/bash
PS=`ps -ef | grep java | grep .jack-server | grep $HOME | wc -l`
if [ $PS = 0 ];then
cp -r /usr/local/bin/Jack/.jack-server ~/
cp /usr/local/bin/Jack/.jack-settings ~/
Port1=`cat /usr/local/bin/Jack/port1.txt`
Port2=`cat /usr/local/bin/Jack/port2.txt`
Port1=$((Port1+2))
Port2=$((Port2+2))
echo "$Port1" > /usr/local/bin/Jack/port1.txt
echo "$Port2" > /usr/local/bin/Jack/port2.txt
sed -i "s/8076/$Port1/g" ~/.jack-server/config.properties
sed -i "s/8077/$Port2/g" ~/.jack-server/config.properties
sed -i "s/8076/$Port1/g" ~/.jack-settings
sed -i "s/8077/$Port2/g" ~/.jack-settings
chmod 700 ~/.jack-server
chmod 600 ~/.jack-server/client.jks
chmod 600 ~/.jack-server/client.pem
chmod 600 ~/.jack-server/launcher.jar
chmod 600 ~/.jack-server/server-1.jar
chmod 600 ~/.jack-server/server.jks
chmod 600 ~/.jack-server/server.pem
chmod 600 ~/.jack-server/config.properties
echo "your jack-server port is $Port1 and $Port2"
fi
```