Wi-Fi & Bluetooth Porting Guide for S5P6442

By: Johnny Wang
Email: johnny.kj.wang@gmail.com

本文将介绍如何将 Broadcom 的 BCM4329 芯片（含 Wi-Fi 和 Bluetooth 功能）移植到三星 s3c6410/s5p6442 平台的 BSP 中。

目录

1. Wi-Fi 及 Bluetooth 简介

1.1 Wi-Fi 简介

Wi-Fi 这个术语是指无线保真（Wireless Fidelity），Wi-Fi 是一种可以将个人电脑、手持设备（如 PDA、手机）等终端以无线方式互相连接的技术。Wi-Fi 是一个无线网路通信技术的品牌，由 Wi-Fi 联盟(Wi-Fi Alliance)所持有。目的是改善基于 IEEE 802.11 标准的无线网路产品之间的互通性。现时一般人会把 Wi-Fi 及 IEEE 802.11 混为一谈。甚至把 Wi-Fi 等同于无线网际网路。

Wi-Fi 技术创建在 IEEE 802.11 标准上，目前 Wi-Fi 联盟所公布的认证种类有：

- WEP: Wired Equivalent Privacy, 即有线等效保密（WEP）协议是对在两台设备间无线传输的数据进行加密的方式，用以防止非法用户窃听或侵入无线网络。
- WPA/WPA2: WPA/WPA2 是基于 IEEE 802.11a、802.11b、802.11g 的单模、双模或双频的产品所创建的测试程序。内容包含通信协议的验证、无线网络安全性机制的验证，以及网络传输表现与兼容性测试。
- WAPI 是 WLAN Authentication and Privacy Infrastructure 的英文缩写，是无线局域网(WLAN)中的一种传输协议。WAPI 同时也是中国无线局域网强制性标准中的安全机制，也是我国首个在计算机宽带无线网络通信领域自主创新并拥有知识产权的安全接入技术标准。
- WMM（Wi-Fi MultiMedia）:当影音多媒体通过无线网络的传递时，要如何验证其带宽保证的机制是否正常运作在不同的无线网络设备及不同的安全性设置上是 WMM 测试的目的。
- WMM Power Save:在影音多媒体通过无线网络的传递时，如何通过管理无线网络设备的待命时间来延长电池寿命，并且不影响其功能性，可以通过 WMM Power Save 的测试来验证。
- WPS（Wi-Fi Protected Setup）:这是一个 2007 年年初才发布的认证，目的是让消费者可以通过更简单的方式来设置无线网络设备，并且保证有一定的安全性。目前 WPS 允许通过 Pin Input Config（PIN）、Push Button Config（PBC）、USB Flash Drive Config（UFD）以及 Near Field Communication Contactless Token Config（NFC）的方式来设置无线网络设备。
- ASD（Application Specific Device）:这是针对除了无线网络访问点（Access Point）及站台（Station）之外其他有特殊应用的无线网络设备，例如 DVD 播放器、投影机、打印机等等。
- CWG（Converged Wireless Group）:主要是针对 Wi-Fi mobile converged devices 的 RF 部分测量的测试程序。
- Wi-Fi Direct
- 还有一种加密方式为 802.1x 用户认证机制，但目前兼容性不高(注意 x 在此为小写)

1.2 Bluetooth 简介

Bluetooth 是一种支持设备短距离通信（一般 10m 内）的无线电技术。能在包括移动电话、PDA、无线耳机、笔记本电脑、相关外设等众多设备之间进行无线信息交换。利用“蓝牙”技术，能够有效地简化移动通信终端设备之间的通信，也能够成功地简化设备与因特网 Internet 之间的通信，从而数据传输变得更加迅速高效，为无线通信拓宽道路。
蓝牙工作在 2.4GHZ 的 ISM 频段，采用了蓝牙接口的设备能够提供高达 720kbit/s 的数据交换速率。

蓝牙支持电路交换和分组交换两种技术，分别定义了两种链路类型，即面向连接的同步链路(SCO)和面向无连接的异步链路(ACL)。

为了在很低的功率状态下也能使蓝牙设备处于连接状态，蓝牙规定了三种节能状态，即停等(Park)状态、保持(Hold)状态和呼吸(Sniff)状态。这几种工作模式按照节能效率以升序排依次是：Sniff 模式、Hold 模式、Park 模式。

HCI：Host Controller Interface，用来沟通 Host 和 Module。Host 通常就是 PC， Module 则是以各种物理连接形式（USB,serial,pc-card 等）连接到 PC 上的 bluetooth Dongle。

LMP：**Link Manage Protocol**，负责连接的建立和拆除以及链路的安全和控制，它们为上层软件模块提供了不同的访问人口，但是两个模块接口之间的消息和数据传递必须通过蓝牙主机控制器接口的解释才能进行。也就是说，中间协议层包括逻辑链路控制与适配协议(L2CAP)、服务发现协议(SDP)、串口仿真协议(RFCOMM)和电话控制协议规范(TCS)。

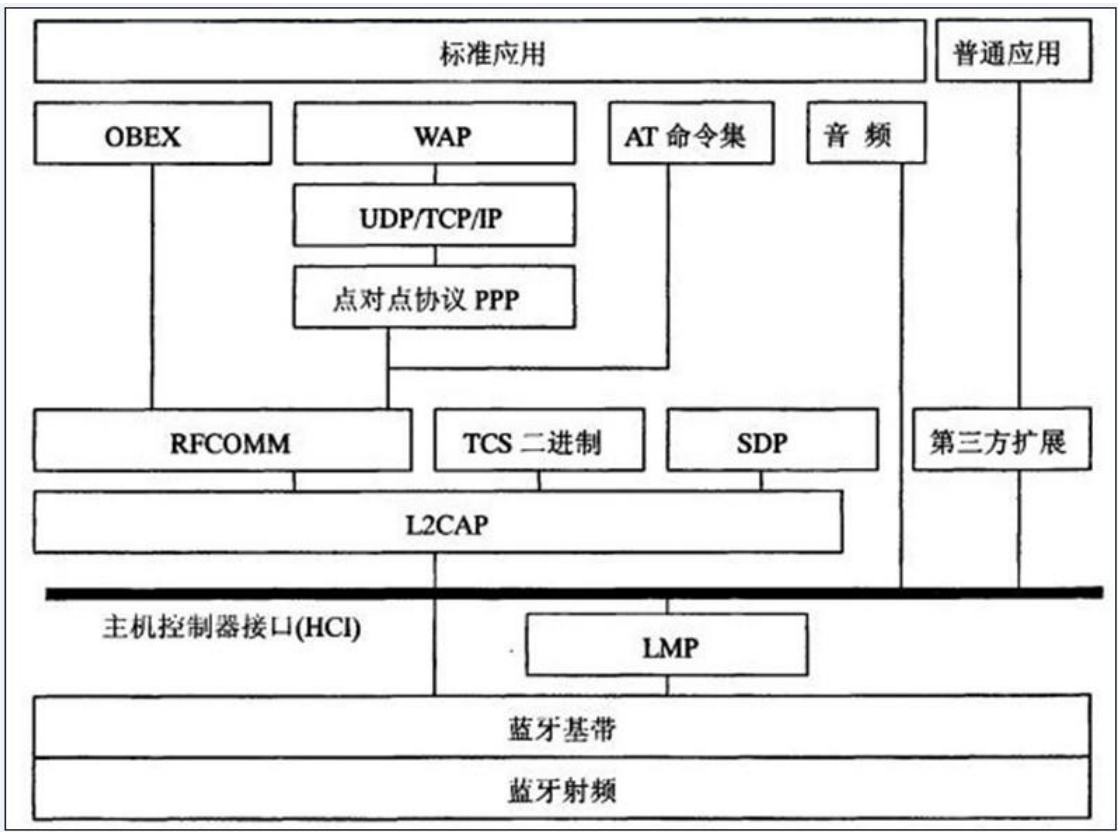
L2CAP：Logical Link Control and Adaptation Protocol，完成数据拆装、服务质量控制、协议复用和组提取等功能，是其他上层协议实现的基础，因此也是蓝牙协议栈的核心部分。所有 L2CAP 数据均通过 HCI 传输到 Remote Device。且上层协议的数据，大都也通过 L2CAP 来传送。L2CAP 可以发送 Command。例如连接，断连等等。

SDP：Service Discovery Protocol，为上层应用程序提供一种机制来发现网络中可用的服务及其特性。SDP 定义了 bluetooth client 发现可用 bluetooth server 服务和它们的特征的方法。SDP 只提供侦测 Service 的办法，但如何用，SDP 不管。每个 Bluetooth Device 最多只能拥有一个 SDP Server。如果一个 Bluetooth Device 只担任 Client，那它不需要 SDP Server。但一个 Bluetooth Device 可以同时担当 SDP Server 和 SDP client.

RFCOMM：即射频通信协议，它可以仿真串行电缆接口协议，符合 ETSI0710 串口仿真协议。通过 RFCOMM，蓝牙可以在无线环境下实现对高层协议，如 PPP、TCP/IP、WAP 等的支持。另外，RFCOMM 可以支持 AT 命令集，从而可以实现移动电话机和传真机及调制解调器之间的无线连接。

蓝牙协议堆栈依照其功能可分四层：

- 核心协议层（HCI、LMP、L2CAP、SDP）
- 即射频通信协议层（RFCOMM）
- 电话控制协议层（TCS-BIN）
- 选用协议层（PPP、TCP、IP、UDP、OBEX、IrMC、WAP、WAE）



蓝牙规范（Bluetooth profile），是指蓝牙通信在某种用途下应该使用的通信协议和相关的规范。蓝牙技术联盟定义了许多 Profile。Profile 目的是要确保 Bluetooth 设备间的互通性（interoperability），但 Bluetooth 产品无须实现所有的 Bluetooth Profile.

- General Access Profile（GAP）
- Advanced Audio Distribution Profile (A2DP)
- Audio Video Remote Control Profile (AVRCP)
- Basic Imaging Profile (BIP)
- Basic Printing Profile (BPP)
- Blood Pressure Profile and Service (BPP)
- Current Time Service Profile(CTSP)
- Device ID Profile (DIP)
- Dial-Up Networking Profile (DUN)
- Find Me Profile (FMF)
- File Transfer Profile (FTP)
- Generic Audio Video Distribution Profile (GAVDP)
- Generic Object Exchange Profile (GOEP)
- Hardcopy Cable Replacement Profile (HCRP)
- Health Device Profile (HDP)
- Hands-Free Profile (HFP)
- Headset Profile (HSP)
- Human Interface Device Profile (HID)
- Immediate Alert Service Profile (IASP)
- Link Lost Service Profile (LLSP)

- Message Access Profile (MAP)
- Next DST Change Service Profile (NDCSP)
- Object Push Profile (OPP)
- Personal Area Networking Profile (PAN)
- Phone Alert Status Profile and Service (PASP)
- Phone Book Access Profile (PBAP)
- Proximity Profile (PP)
- Reference Time Update Service Profile (RTUSP)
- SIM Access Profile (SAP)
- Service Discovery Application Profile（SDAP）
- Serial Port Profile（SPP）
- Synchronization Profile (SYNCH)
- Time Profile (TP)
- Tx Power Service Profile(TPSP)
- Video Distribution Profile (VDP)
- Object Push Profile (OPP)

版本	规范发布日期	增强功能
0.7	1998 年 10 月 19 日	Baseband、LMP
0.8	1999 年 1 月 21 日	HCI、L2CAP、RFCOMM
0.9	1999 年 4 月 30 日	OBEX 与 IrDA 的互通性
1.0 Draft	1999 年 7 月 5 日	SDP、TCS
1.0 A	1999 年 7 月 26 日	/
1.0 B	2000 年 10 月 1 日	WAP 应用上更具互通性
1.1	2001 年 2 月 22 日	IEEE 802.15.1
1.2	2003 年 11 月 5 日	列入 IEEE 802.15.1a
2.0 + EDR	2004 年 11 月 4 日	EDR 传输率提升至 2-3Mbps
2.1 + EDR	2007 年 7 月 26 日	简易安全配对、暂停与继续加密、Sniff 省电
3.0 + HS	2009 年 4 月 21 日	High Speed, Enhanced Power Control
4.0 + HS	2010 年 6 月 30 日	High Speed, Low Power Consumption

具体协议及 Profile 定义及规范请参考以下网址:

https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/Overview.htm

https://www.bluetooth.org/Technical/Specifications/adopted.htm

UUID (Universally Unique Identifier)，即通用唯一识别码。
UUID 是针对某项协议或服务的,不是针对设备的,你知道对方开了哪个服务,就可以查询到对应的 UUID.

UUID_Name	UUID
BASE_UUID	00000000-0000-1000-8000-00805F9B34FB

PDF-XCHANGE Click to buy NOW! www.docu-track.com

不同协议及 Profile 之 UUID 由 UUID 的 5-8 位识别，如蓝色部分
具体协议及 Profile 之 UUID 定义请参考以下网址:
http://www.bluetooth.org/Technical/AssignedNumbers/service_discovery.htm

本案所使用的 Wi-Fi 和 Bluetooth 设备为 CyberTAN 公司的 NC023 产品，其核心芯片为 Broadcom 公司的 BCM4329. 该芯片整合了 IEEE802.11a/b/g/n,Bluetooth2.1+EDR(Enhanced Data Rate),FM RDS Tx/Rx,WLAN SPI/SDIO interface, BT UART interface,具体设备及芯片信息请参 NC023 及 BCM4329 的官方 Data sheet。具体功能设计请参考相关线路图。

2. Kernel 中相关选项的配置

2.1 Wi-Fi 部分

Networking support → Networking options → 选中 Packet socket

该选项用于支持 wpa_supplicant（可以理解为 Wi-Fi 服务的守护进程，详见 wpa_supplicant 的 man page）与无线网卡直接通信，而无需使用 kernel 中已经实现的中间网络协议。若此选项没有选中，则在 Android 中运行 wpa_supplicant 程序时会报类似于以下的错误导致无法开启进程：socket(PF_PACKET): Address family not supported by protocol

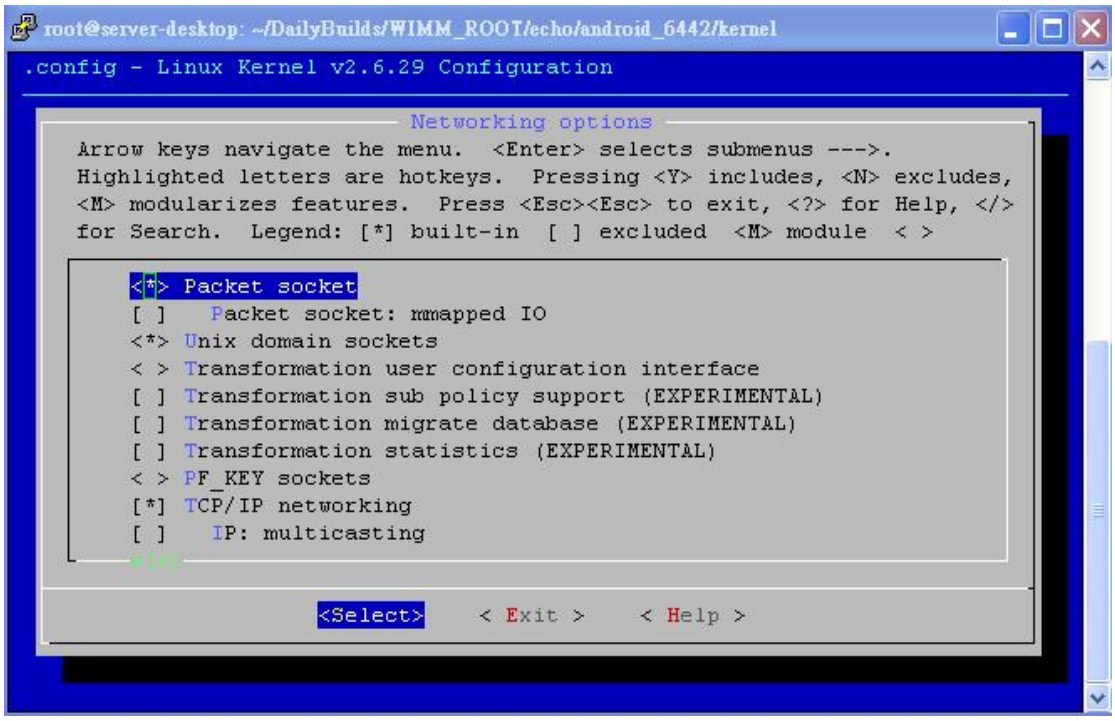


图 2.1.1

PDF-XCHANGE Click to buy NOW! www.docu-track.com

2.2 Bluetooth 部分

Bluetooth 需要和 CPU 交换数据以取得信息，所以需要一个接口，通常 bluetooth 会采用 UART，USB 等接口方式，本案中 bluetooth 采用 UART 接口，因此本文仅对 UART 的接口方式进行说明，USB 接口部分在本文不做讨论。需要说明的是，蓝牙模块在红外功能的应用时使用的是 PCM 接口，由于本案在蓝牙模块中没有语音部分的应用，在此亦不做说明。从蓝牙协议层的角度来说，radio, SDP, RFCOMM, L2CAP, HCI 等层基本都不用我们操心，需要我们操心的就是蓝牙模块和 CPU 之间的接口，以及在这个接口上传递的数据的格式，或者说是数据包的格式，以及数据包的处理，此部分在 Linux 驱动层中已经有完整实现，因此我们在 kernel 对蓝牙驱动进行配置时，只需选择相关协议即可。

Kernel 下具体协议定义及初始化流程请参考本人另一篇文章之 Bluetooth_Protocol_Flow.pdf.

BCM4329 的 Bluetooth 使用 kernel built-in driver，menuconfig 中具体位置是在 Networking support → Bluetooth subsystem support，配置如图 2.2.1 和 2.2.2：

- BT_L2CAP [=y] //L2CAP protocol support
- BT_SCO [=y] //SCO links support
- BT_RFCOMM [=y] //RFCOMM protocol support
- BT_BNEP [=y] //BNEP protocol support
- BT_HIDP [=y] //HIDP protocol support

L2CAP: Logical Link Control and Adaptation Protocol, 即逻辑链接控制及适配协议，提供链路(连接向导)及无连接的数据传输,大多数蓝牙应用需要通过 L2CAP 提供支持。

SCO: SCO 链接通过蓝牙提供语音传输, 一般的语音应用如 Headset, Audio 等需要 SCO 支持。

RFCOMM: RFCOMM 提供流传输的连接向导，拨号网络，OBEX 及其他的蓝牙应用需要 RFCOMM 提供支持。

BNEP: Bluetooth Network Encapsulation Protocol, 即蓝牙网络封装协议, 属于蓝牙最上层的 Ethernet 仿真层, 蓝牙的 PAN 规范需要 BNEP 支持。

HIDP: Human Interface Device Protocol, 即智能设备接口协议, 蓝牙的智能设备接口规范需要 HIDP 支持。

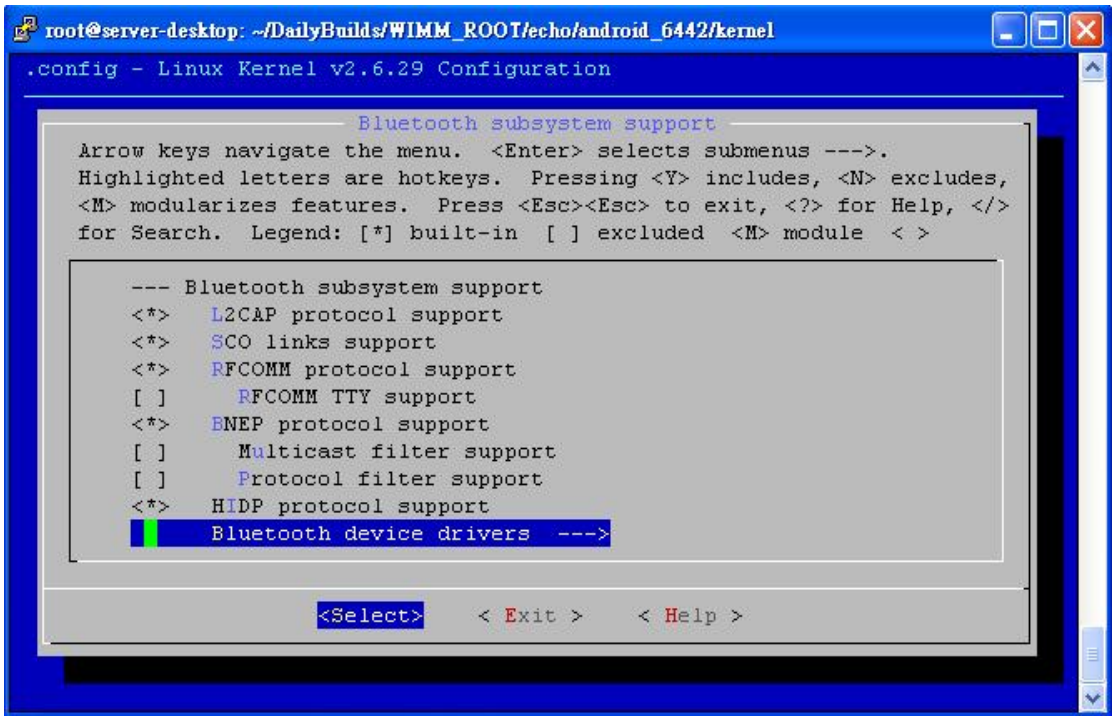


图 2.2.1

- BT_HCIUART [=y] //HCI UART driver
- BT_HCIUART_H4 [=y] // UART (H4) protocol support
- BT_HCIUART_BCSP [=y] //BCSP protocol support
- 注:由于 BCM4329 的 BT 模块所支持的规格为 Bluetooth 2.1 UART HCI (H4)，因此这里选中 UART (H4) protocol support（见 NC023 Data Sheet 的第 22 页）。

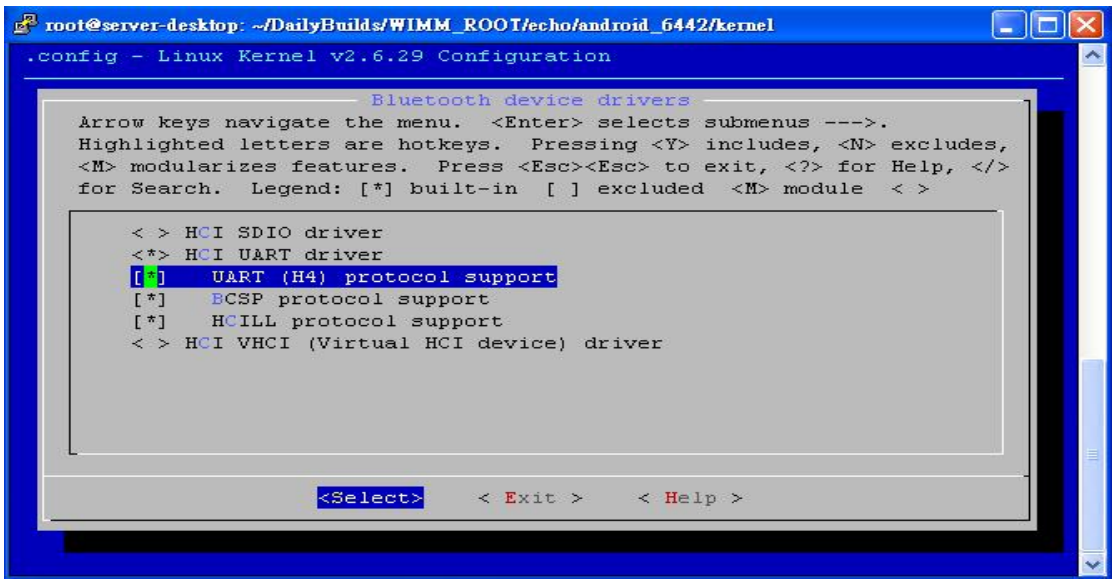


图 2.2.2

3. Driver 及 Firmware

3.1 Wi-Fi 部分

Wi-Fi 模块到目前为止所使用的驱动版本为 CyberTAN 提供的 4.218.243.0。CyberTAN 在 Broadcom 的原版驱动基础上作了一些改动，其中一部分改动需要修改我们 Kernel 的相关代码来得以支持，对 Kernel 的修改将在稍后的章节统一讲解。

3.1.1 编译驱动（需要在 Linux 环境下进行）

- 进入目录 <driver>/BCM43X9-SDIO-Android-4.218.243/open-src/src/dhd/linux
- 打开并修改 set_env.sh，

  a. 在 PATH 变量中加入你系统中交叉编译器所在的路径，如 export PATH=/usr/local/arm/4.3.1-eabi-armv6/usr/bin:$PATH；

  b. 指定变量 CROSS_COMPILE 为你交叉编译器的前缀，如 WIMM 专案所使用的编译器前缀为 export CROSS_COMPILE=arm-linux-；

  c. 将 LINUXDIR 变量指定为板子上所使用的 Kernel 的源码路径（需要已经编译过的）；

  d. 将 LINUXVER 变量指定为 LINUXDIR 中所使用的 Kernel 源码的版本号，如 export LINUXVER=2.6.29。
- 打开 Makefile，查找 “-Werror”，找到后将该行注释掉（否则编译的时候会因为不必要的 Warning 而终止编译）。
- 打開并修改 dhd_linux.c,根據需要編譯的 driver 選擇對應的 firmware。

```
/* Definitions to provide path to the firmware and nvram
 * example nvram_path[MOD_PARAM_PATHLEN]="/projects/wlan/nvram.txt"
 */
char firmware_path[MOD_PARAM_PATHLEN]="/system/etc/wifi/rtecdc_mfg.bin";
char nvram_path[MOD_PARAM_PATHLEN]="/system/etc/wifi/nvram_mfg.txt";
//char firmware_path[MOD_PARAM_PATHLEN]="/system/etc/wifi/rtecdc.bin";
//char nvram_path[MOD_PARAM_PATHLEN]="/system/etc/wifi/nvram.txt";
```

  由以上代码可知，Wifi 的 driver 在编译时已决定与其 firmware 存在一一对应关系，即:
  rtecdc_mfg.bin à dhd_mfg.ko
  rtecdc.bin à dhd.ko
- 在当前目录打开终端，并在终端中输入 . set_env.sh（注意不要漏了前面的点及空格符），然后再输入 ./buildko.sh（需要 buildko.sh 有可执行权限）。
- 编译完成后所生成的 dhd-cdc-sdmmc-gpl-xxx 目录下的 dhd.ko 及 dhd.ko.stripped 即为编译好的 Wi-Fi 驱动（推荐使用 dhd.ko.stripped，该文件为 dhd.ko 的去符号版本，因此会比 dhd.ko 小很多）。

3.1.2 定制驱动

BCM4329 驱动加载时需要将与该驱动版本对应的 firmware 传入其设备内存中以实现它的特定功能，这个 firmware 放在驱动目录下的 BCM43X9-SDIO-Android-4.218.243/firmware/ 下，用户所使用的 firmware 为 for end user/ 4329b1/ 下的 sdio-g-cdc-full11n-reclaim-roml-wme.bin，工厂 RF 测试所使用的 firmware 为 for mfg test/4329b1/ 下的 sdio-roml-cdc-g-11n-mfgtest-seqcmds.bin。用于配置 firmware 功能的配置文件为驱动目录下的 BCM43X9-SDIO-Android- 4.218.243/nvram/4329b1/nvram.txt。

PDF-XCHANGE Click to buy NOW! www.docu-track.com

按照目前的定义，Wi-Fi 的 firmware 被放到 Android 系统中的 /system/etc/ wifi/ 目录下，并命名为 rtecdc.bin；nvram.txt 也置于同一目录下。若要更改此目录，则需要需要驱动目录下的 BCM43X9-SDIO-Android-4.218.243/open-src/src/dhd/sys/dhd_linux.c，找到 char firmware_path[MOD_PARAM_PATHLEN] 一行，将其后的路径更改为将要使用的路径，下面的 nvram_path 也是如此。指定好这里的路径后，在 Android 中用 insmod 命令加载该驱动时，它会默认去指定的路径寻找并加载相应的 firmware。而 Wi-Fi 驱动(dhd.ko 或 dhd.ko. stripped)应该被放置于 Android 系统中的 /system/lib/modules 目录下(Android 中外挂驱动的默认加载路径)。

```
//In AndroidBoard.mk
    PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/wpa_supplicant.conf:system/etc/wifi/wpa_supplicant.conf   \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/dhd.ko:system/lib/modules/dhd.ko       \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/rtecdc.bin:system/etc/wifi/rtecdc.bin    \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/nvram.txt:system/etc/wifi/nvram.txt     \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/dhcpcd.conf:system/etc/dhcpcd/dhcpcd.conf
```

3.2 Bluetooth 部分

如 2.2 节中所讲，Bluetooth 使用 kernel built-in driver，所以本案中无须为其单独准备驱动，只要将 kernel 中相关选项正确配置即可。与 Wi-Fi 类似，Bluetooth 也有它自己的 firmware，这个 firmware 是厂商提供的一个.hcd 文件，我们暂且将它命名为 BCM4329_BT.hcd。这个文件被放置于 Android 系统中的/system/etc/firmware/下，开启 Bluetooth 功能时必须先将这个 firmware 加载入 Bluetooth 的设备内存。通过 hciattach 初始化串口及建立串口和蓝牙协议层之间的数据连接通道。

蓝牙驱动的三个步骤：

1. 串口驱动，数据的传递都是通过串口，这是实现蓝牙驱动的根本。Kernel 对串口驱动的实现已较为成熟，我们只需做好相关配置即可。
2. 模块初始化，即模块上电。此部分将在第四章节进行讲解。
3. 通过 hciattach 初始化串口及建立串口和蓝牙协议层之间的数据通道。详情请参见 init.rc 中的使用说明，感兴趣的朋友可以参见 init_uart.c

```
//In AndroidBoard.mk
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/../../sec_proprietary/bluetooth/BCM4329_BT.hcd:system/etc/firmware/BCM4329_BT.hcd

//In init.rc
    mkdir /system/etc/firmware 0770 bluetooth bluetooth
    chmod 0771 /system/etc/firmware
    service hciattach /system/bin/brcm_patchram_plus --enable_hci --baudrate 460800 –patchram
    /system/etc/firmware/BCM4329_BT.hcd /dev/s3c2410_serial0
    group bluetooth net_bt_admin
    disabled
```

4. Kernel 源码修改

若无特别说明，以下文件路径均代表 Kernel source 根目录下的路径。

4.1 Wi-Fi 部分

4.1.1 GPIO 配置

WIMM 专案中大多数 I/O 接口都是使用 GPIO 来模拟的，Wi-Fi 的 SDIO 和 Bluetooth 的 UART 也不例外，因此我们需要根据 HW 提供的 GPIO List 在 Kernel source 中配置我们相应的 GPIO。对于没有连接及已有连接但目前尚未使用的 GPIO pin, 基于功耗的考量，因将其初始状态配置为 Output.

- arch/arm/plat-s5p64xx/include/plat/wimm-gpio.h

首先要在这个头文件中将所要用到的 GPIO pin 作一个定义，以提高代码的可阅读性。

```
#define GPIO_WLAN_WAKEUP                S5P64XX_GPH0(4)
#define GPIO_WLAN_WAKEUPHOST            S5P64XX_GPJ0(2)
#define GPIO_WLAN_RST_N                 S5P64XX_GPH3(3)
#define GPIO_WLAN_VBAT_POWER_EN         S5P64XX_GPJ0(3)
#define GPIO_WLAN_IO_POWER_EN           S5P64XX_GPH3(4)
#define GPIO_VDD_SD_POWER_EN            S5P64XX_GPH1(5)
```

- arch/arm/mach-s5p6442/setup-sdhci.c

在这个文件中对 SDIO 的控制 pin 和数据传输 pin 进行设置。由于 Wi-Fi 所使用的是 SDIO 的 channel 2，所以这里只需对 sdhci2 进行设置。需要注意的是，这里不要对 S5P64XX_GPG2(2) 进行设置，因为 GPG2(2) 为 Bluetooth 的 Reset pin，在这里设置会影响到 Bluetooth 的功能。

```
void s3c6410_setup_sdhci2_cfg_gpio(struct platform_device *dev, int width)
{
        unsigned int gpio;

        /* Channel 2 supports 1 and 4-bit bus width */
        switch(width) {
        case 0:
        case 1:
        case 4:
                /* Set all the necessary GPIO function and pull up/down */
                for (gpio = S5P64XX_GPG2(0); gpio <= S5P64XX_GPG2(1); gpio++) {
                        s3c_gpio_cfgpin(gpio, S3C_GPIO_SFN(2));
                        s3c_gpio_setpull(gpio, S3C_GPIO_PULL_NONE);
                }
                for (gpio = S5P64XX_GPG2(3); gpio <= S5P64XX_GPG2(6); gpio++) {
                        s3c_gpio_cfgpin(gpio, S3C_GPIO_SFN(2));
                        s3c_gpio_setpull(gpio, S3C_GPIO_PULL_NONE);
                }
```

```
                        writel(0x3fcf, S5P64XX_GPG2DRV);
                        break;
#endif
                default:
                        printk("Wrong SD/MMC bus width : %d\n", width);
                }
}
```

- arch/arm/mach-s5p6442/bcm4329-wlan.c

此文件为 Wi-Fi PM Driver，主要通过设置相应的 GPIO pin 来达到对 Wi-Fi 模块进行电源开关控制。

```
#include <linux/kobject.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/types.h>
#include <linux/interrupt.h>
#include <linux/platform_device.h>
#include <linux/io.h>
#include <linux/mmc/card.h>
#include <linux/mmc/host.h>
#include <mach/gpio.h>
#include <plat/gpio-cfg.h>
#include <mach/gpio.h>
#include <mach/gpio-core.h>
#include <mach/map.h>
#include <plat/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/mmc/host.h>
#include <plat/wimm-gpio.h>
#include <plat/regs-rtc.h>

static char power_state[5] ="off";
extern struct mmc_host    *mmc_wifi;

extern int bcm4329_bt_power_state( void );
extern void bcm_wifi_clock_enable(int enable);

int    wlan_power_state( void )
{
        int wlan_power_state;
        if ( !strncmp( power_state, "on",2 ))
                wlan_power_state = 1;
        else
```

```
            wlan_power_state = 0;
        return wlan_power_state;
}
EXPORT_SYMBOL( wlan_power_state );

void wlan_power_enable( int enable )
{
        printk("%s\n", __func__);
        if ( enable ) {
                gpio_set_value( GPIO_WLAN_VBAT_POWER_EN, 1 );
                s3c_gpio_setpull( GPIO_WLAN_VBAT_POWER_EN, S3C_GPIO_PULL_NONE );

                gpio_set_value( GPIO_WLAN_IO_POWER_EN, 1 );
                s3c_gpio_setpull( GPIO_WLAN_IO_POWER_EN, S3C_GPIO_PULL_NONE );

                gpio_set_value( GPIO_VDD_SD_POWER_EN, 1 );
                s3c_gpio_setpull( GPIO_VDD_SD_POWER_EN, S3C_GPIO_PULL_NONE );
        }
        else {
                gpio_set_value( GPIO_WLAN_VBAT_POWER_EN, 0) ;
                s3c_gpio_setpull( GPIO_WLAN_VBAT_POWER_EN, S3C_GPIO_PULL_NONE );

                gpio_set_value( GPIO_WLAN_IO_POWER_EN, 0 );
                s3c_gpio_setpull( GPIO_WLAN_IO_POWER_EN, S3C_GPIO_PULL_NONE );

                gpio_set_value( GPIO_VDD_SD_POWER_EN, 0 );
                s3c_gpio_setpull( GPIO_VDD_SD_POWER_EN, S3C_GPIO_PULL_NONE );
        }
}
EXPORT_SYMBOL( wlan_power_enable );

void wlan_reset(int enable)
{
        if (enable) {
                gpio_set_value( GPIO_WLAN_RST_N, 0 );
                mdelay( 100 );
                gpio_set_value( GPIO_WLAN_RST_N, 1 );
                strcpy( power_state, "on" );
        } else {
                gpio_set_value( GPIO_WLAN_RST_N, 0 );
                mdelay( 100 );
                strcpy( power_state, "off" );
        }
}
```

```
EXPORT_SYMBOL(wlan_reset);

void wlan_power(int enable)
{
	printk("[%s] \n", __func__);

	if (enable) {
		if (!bcm4329_bt_power_state()) {
			wlan_power_enable(1);
		}
	} else {
		if (!bcm4329_bt_power_state()) {
			wlan_power_enable(0);
		}
	}
}
EXPORT_SYMBOL(wlan_power);

static int bcm4329_pm_probe( struct platform_device *pdev )
{
	printk("%s\n",__func__);

	s3c_gpio_cfgpin( GPIO_WLAN_VBAT_POWER_EN, S3C_GPIO_SFN(1) );
	s3c_gpio_setpull( GPIO_WLAN_VBAT_POWER_EN, S3C_GPIO_PULL_NONE );
	gpio_set_value(GPIO_WLAN_VBAT_POWER_EN, 0);

	s3c_gpio_cfgpin( GPIO_WLAN_IO_POWER_EN, S3C_GPIO_SFN(1) );
	s3c_gpio_setpull( GPIO_WLAN_IO_POWER_EN, S3C_GPIO_PULL_NONE );
	gpio_set_value(GPIO_WLAN_IO_POWER_EN, 0);

	s3c_gpio_cfgpin( GPIO_VDD_SD_POWER_EN, S3C_GPIO_SFN(1) );
	s3c_gpio_setpull( GPIO_VDD_SD_POWER_EN, S3C_GPIO_PULL_NONE );
	gpio_set_value(GPIO_VDD_SD_POWER_EN, 0);

	/*wlan_host_wakeup*/
	s3c_gpio_cfgpin(GPIO_WLAN_WAKEUP, S5P64XX_GPH0_4_EXT_INT0_4);
	s3c_gpio_setpull( GPIO_WLAN_WAKEUP, S3C_GPIO_PULL_DOWN );
	/*gpio_direction_output(GPIO_WLAN_WAKEUP, 1);*/

	/*wakeup wlan*/
	s3c_gpio_cfgpin( GPIO_WLAN_WAKEUPHOST, S3C_GPIO_SFN(1) );
	s3c_gpio_setpull( GPIO_WLAN_WAKEUPHOST, S3C_GPIO_PULL_NONE );
	gpio_set_value(GPIO_WLAN_WAKEUPHOST, 0);
```

```
	s3c_gpio_cfgpin( GPIO_WLAN_RST_N, S3C_GPIO_SFN(1) );
	s3c_gpio_setpull( GPIO_WLAN_RST_N, S3C_GPIO_PULL_NONE );
	gpio_set_value(GPIO_WLAN_RST_N, 0);

	return 0;
}

static int bcm4329_pm_remove(struct platform_device *pdev)
{
	return 0;
}

static struct platform_driver bcm4329_pm_driver = {
	.probe = bcm4329_pm_probe,
	.remove = bcm4329_pm_remove,
	.driver = {
		.name = "bcm4329-pm-driver",
		.owner = THIS_MODULE,
	}
};

static int __init wlan_pm_init(void)
{
	int ret;
	ret = platform_driver_register(&bcm4329_pm_driver);
	if (ret < 0)
	{
		printk("platform_driver_register wifi returned %d\n", ret);
		return ret;
	}
	return 0;
}

static void __exit wlan_pm_exit(void)
{
	platform_driver_unregister(&bcm4329_pm_driver);
}

module_init(wlan_pm_init);
module_exit(wlan_pm_exit);

MODULE_AUTHOR("archermind");
MODULE_DESCRIPTION("NC023 BCM4329    WIFI MODULE PM DRIVER");
MODULE_LICENSE("GPL");
```

PDF-XCHANGE Click to buy NOW! www.docu-track.com

4.1.2 Wi-Fi 驱动支持

这部分是针对 CyberTAN 提供的 Wi-Fi 4.218.243.0 驱动的要求所作出的修改，详见 Wi-Fi Driver 根目录中的 README.pdf.

- include/linux/mmc/host.h

  在这里定义 BCM_CARD_DETECT 宏供其它地方使用。

```
#define BCM_CARD_DETECT 1
#ifdef BCM_CARD_DETECT
extern struct mmc_host *mmc_get_sdio_host(void);
#endif    // BCM_CARD_DETECT
```

- drivers/mmc/core/host.c

  根据 Wi-Fi Driver 中 README.pdf 的要求修改。

全局:

```
#ifdef BCM_CARD_DETECT
#define MMC_SDIO_SLOT 2
static struct mmc_host *sdio_host = NULL;

static int select_sdio_host(struct mmc_host *host, int add)
{
        if(!add){
                if(host == sdio_host){
                        sdio_host = NULL;
                        printk("%s: sdio_host cleaned.\n", __FUNCTION__);
                }
                return 0;
        }

        if(host->index == MMC_SDIO_SLOT){
                sdio_host = host;
                printk("%s: sdio_host assigned.(%p)\n", __FUNCTION__, sdio_host);
        }
        return 0;
}

struct mmc_host *mmc_get_sdio_host(void)
{
        return sdio_host;
}

EXPORT_SYMBOL(mmc_get_sdio_host);
#endif // BCM_CARD_DETECT
```

int mmc_add_host(struct mmc_host *host) 函数末尾 return 0 之前:

```
#ifdef BCM_CARD_DETECT
            select_sdio_host(host,1);
#endif    // BCM_CARD_DETECT
```

void mmc_remove_host(struct mmc_host *host) 函数起始处:

```
#ifdef BCM_CARD_DETECT
            select_sdio_host(host,0);
#endif    // BCM_CARD_DETECT
```

4.1.3 其他修改

- arch/arm/mach-s5p6442/mach-smdk6442.c

这个文件为板级的初始化文件，Kernel 初始化时会按照这个文件来初始化板子上在这里注册过的设备。我们在这里对上面的 Wi-Fi PM Driver (bcm4329_wlan.c) 注册。

全局:

```
static struct platform_device bcm_wlan_device = {
            .name      = "bcm4329-pm-driver",
            .id          = -1,
};
```

static struct platform_device *smdk6442_devices[] __initdata 初始化变量中：

```
#ifdef CONFIG_BCM4329_WLAN_PM
            &bcm_wlan_device,
#endif
```

- drivers/mmc/host/sdhci-s3c.c

定义并设置 Wi-Fi PM Driver 中所使用的变量 mmc_wifi

全局:

```
struct mmc_host *mmc_wifi;
```

sdhci_s3c_probe() 函数中 sdhci_add_host(host) 語句之前:

```
mmc_wifi = host->mmc;
```

sdhci_s3c_probe() 函数之后:

```
EXPORT_SYMBOL(mmc_wifi);
```

- arch/arm/mach-s5p6442/Kconfig

在 Kernel 配置选项中加入对 Wi-Fi PM Driver 的配置

```
config BCM4329_WLAN_PM
            tristate "BCM4329 WLAN PM Driver"
            help
              WLAN power on & off
```

- arch/arm/mach-s5p6442/Makefile

Makefile 中加入 Wi-Fi PM Driver 的编译选项

```
obj-$(CONFIG_BCM4329_WLAN_PM) += bcm4329-wlan.o
```

4.2 Bluetooth 部分

4.2.1 GPIO 配置

- arch/arm/plat-s5p64xx/include/plat/wimm-gpio.h

对 Bluetooth 的 GPIO pin 进行定义，以便于代码阅读。

```
#define GPIO_BT_UART_RXD            S5P64XX_GPA0(0)
#define GPIO_BT_TXD                 S5P64XX_GPA0(1)
#define GPIO_BT_CTS                 S5P64XX_GPA0(2)
#define GPIO_BT_RTS                 S5P64XX_GPA0(3)
#define GPIO_BT_WAKEUP              S5P64XX_GPH1(0)
#define GPIO_BT_WAKEUPHOST          S5P64XX_GPJ0(1)
#define GPIO_BT_RESET               S5P64XX_GPG2(2)
#define GPIO_BT_PCM_CLK             S5P64XX_GPC1(0)
#define GPIO_BT_PCM_SYNC            S5P64XX_GPC1(2)
#define GPIO_BT_PCM_IN              S5P64XX_GPC1(3)
#define GPIO_BT_PCM_OUT             S5P64XX_GPC1(4)
```

- arch/arm/mach-s5p6442/bcm4329-bt.c

此文件为 Bluetooth PM Driver，这里主要对 Bluetooth 的 GPIO 数据 pin 及 power pin 作了设置，加入了对 Bluetooth power 的控制，并在 sysfs 中注册了 rfkill 以便于在操作系统中对 Bluetooth power 进行操作。

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/gpio.h>
#include <linux/delay.h>
#include <linux/rfkill.h>
#include <mach/gpio-core.h>
#include <plat/gpio-cfg.h>
#include <plat/regs-gpio.h>
#include <mach/gpio.h>
#include <plat/wimm-gpio.h>

static volatile int bt_state = 0;
static char bt_power_before_suspend = 0;

extern int wlan_power_state(void);
extern void wlan_power_enable(int);

static void bcm4329_bt_on(void)
{
    s3c_gpio_cfgpin(GPIO_BT_UART_RXD, S3C_GPIO_SFN(2));//set to UART_0_RXD
    s3c_gpio_cfgpin(GPIO_BT_TXD, S3C_GPIO_SFN(2));      //set to UART_0_TXD
    s3c_gpio_cfgpin(GPIO_BT_CTS, S3C_GPIO_SFN(2));      //set to UART_0_CTSn
    s3c_gpio_cfgpin(GPIO_BT_RTS, S3C_GPIO_SFN(2));      //set to UART_0_RTSn
```

```
	if (!wlan_power_state()) {
		wlan_power_enable(1);
	}

	gpio_set_value(GPIO_BT_RESET, 0);
	mdelay(100);
	gpio_set_value(GPIO_BT_RESET, 1);

	bt_state = 1;
}

static void bcm4329_bt_off(void)
{
	if (!wlan_power_state()) {
		wlan_power_enable(0);
	}

	gpio_set_value(GPIO_BT_RESET, 0);
	mdelay(20);

	s3c_gpio_cfgpin(GPIO_BT_UART_RXD, S3C_GPIO_SFN(1));//set to output
	s3c_gpio_cfgpin(GPIO_BT_TXD, S3C_GPIO_SFN(1));		//set to output
	s3c_gpio_cfgpin(GPIO_BT_CTS, S3C_GPIO_SFN(1));		//set to output
	s3c_gpio_cfgpin(GPIO_BT_RTS, S3C_GPIO_SFN(1));		//set to output

	bt_state = 0;
}

static int bcm4329_bt_toggle_radio(void *data, enum rfkill_state state)
{
	if (bt_state == state) {
		return 0;
	} else {
		if (state == RFKILL_STATE_ON) {
			pr_info("BCM4329_BT: going ON.\n");
			bcm4329_bt_on();
		} else {
			pr_info("BCM4329_BT: going OFF.\n");
			bcm4329_bt_off();
		}
	}

	return 0;
```

```
}

static int bcm4329_bt_get_state(void *data, enum rfkill_state *state)
{
	if (bt_state) {
		*state = RFKILL_STATE_UNBLOCKED;
	} else {
		*state = RFKILL_STATE_SOFT_BLOCKED;
	}

	return bt_state;
}

int bcm4329_bt_power_state(void)
{
	return bt_state;
}
EXPORT_SYMBOL(bcm4329_bt_power_state);

static int bcm4329_bt_probe(struct platform_device *dev)
{
	printk("%s\n", __func__);
	int rt;
	struct rfkill *rfk;

	s3c_gpio_cfgpin(GPIO_BT_UART_RXD, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_TXD, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_CTS, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_RTS, S3C_GPIO_SFN(1));

	s3c_gpio_cfgpin(GPIO_BT_PCM_CLK, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_PCM_SYNC, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_PCM_IN, S3C_GPIO_SFN(1));
	s3c_gpio_cfgpin(GPIO_BT_PCM_OUT, S3C_GPIO_SFN(1));

	/*bt_host_wakeup*/
	s3c_gpio_cfgpin(GPIO_BT_WAKEUP, S3C_GPIO_SFN(1));
	s3c_gpio_setpull(GPIO_BT_WAKEUP, S3C_GPIO_PULL_DOWN);

	/*bt_wakeup*/
	s3c_gpio_cfgpin(GPIO_BT_WAKEUPHOST, S3C_GPIO_SFN(1));
	s3c_gpio_setpull(GPIO_BT_WAKEUPHOST, S3C_GPIO_PULL_NONE);
	gpio_set_value(GPIO_BT_WAKEUPHOST, 0);
```

PDF-XCHANGE Click to buy NOW! www.docu-track.com


```
	s3c_gpio_cfgpin(GPIO_BT_RESET, S3C_GPIO_SFN(1));
	s3c_gpio_setpull(GPIO_BT_RESET, S3C_GPIO_PULL_NONE);
	gpio_set_value(GPIO_BT_RESET, 0);

	rfk = rfkill_allocate(&dev->dev, RFKILL_TYPE_BLUETOOTH);
	if (!rfk) {
		rt = -ENOMEM;
		goto err_rfk_alloc;
	}

	rfk->name = "bcm4329-bt";
	rfk->toggle_radio = bcm4329_bt_toggle_radio;
	rfk->get_state = bcm4329_bt_get_state;
	rfk->state = RFKILL_STATE_SOFT_BLOCKED;

	rt = rfkill_register(rfk);
	if (rt) {
		goto err_rfkill;
	}

	platform_set_drvdata(dev, rfk);

	return 0;

err_rfkill:
	if (rfk) {
		rfkill_free(rfk);
	}
	rfk = NULL;
err_rfk_alloc:
	bcm4329_bt_off();

	return rt;
}

static int __devexit bcm4329_bt_remove(struct platform_device *dev)
{
	struct rfkill *rfk = platform_get_drvdata(dev);

	platform_set_drvdata(dev, NULL);

	if (rfk) {
		rfkill_unregister(rfk);
	}
```

```
	rfk = NULL;

	bcm4329_bt_off();
	gpio_free(GPIO_BT_UART_RXD);
	gpio_free(GPIO_BT_TXD);
	gpio_free(GPIO_BT_CTS);
	gpio_free(GPIO_BT_RTS);
	gpio_free(GPIO_BT_WAKEUP);
	gpio_free(GPIO_BT_WAKEUPHOST);
	gpio_free(GPIO_BT_RESET);

	return 0;
}

static int bcm4329_bt_suspend(struct platform_device *dev, pm_message_t state)
{
	printk("%s: Enter\n", __func__);

	return 0;
}

static int bcm4329_bt_resume(struct platform_device *dev)
{
	printk("%s: Enter\n", __func__);

	return 0;
}


static struct platform_driver bcm4329_bt_driver = {
	.probe = bcm4329_bt_probe,
	.remove = __devexit_p(bcm4329_bt_remove),

	.suspend = bcm4329_bt_suspend,
	.resume = bcm4329_bt_resume,

	.driver = {
		.name = "bcm4329-bt",
		.owner = THIS_MODULE,
	},
};

static int __init bcm4329_bt_init(void)
{
```

```
	int ret;
	ret = platform_driver_register(&bcm4329_bt_driver);
	if (ret < 0) {
		printk(KERN_ALERT "bt driver register failed. %d\n", ret);
		return ret;
	}

	return 0;
}

static void __exit bcm4329_bt_exit(void)
{
	platform_driver_unregister(&bcm4329_bt_driver);
}

module_init(bcm4329_bt_init);
module_exit(bcm4329_bt_exit);

MODULE_DESCRIPTION("bcm4329 bluetooth driver");
MODULE_LICENSE("GPL");
```

4.2.2 其它修改

- arch/arm/mach-s5p6442/Kconfig

  在 Kernel 配置选项中加入对 Bluetooth PM Driver 的配置

```
config BCM4329_BT
		bool "BCM4329 BT Driver"
		help
		BCM4329 Bluetooth driver
```

- net/bluetooth/Kconfig

```
config BT_L2CAP
	tristate "L2CAP protocol support"
	depends on BT
config BT_SCO
	tristate "SCO links support"
	depends on BT
```

- net/bluetooth/rfcomm/Kconfig

```
config BT_RFCOMM
	tristate "RFCOMM protocol support"
	depends on BT && BT_L2CAP
```

- net/bluetooth/bnep/Kconfig

```
config BT_BNEP
        tristate "BNEP protocol support"
        depends on BT && BT_L2CAP
        select CRC32
```

- net/bluetooth/hidp/Kconfig

```
config BT_HIDP
        tristate "HIDP protocol support"
        depends on BT && BT_L2CAP && INPUT
        select HID
```

- arch/arm/mach-s5p6442/Makefile

  Makefile 中加入 Bluetooth PM Driver 的编译选项

```
obj-$(CONFIG_BCM4329_BT) += bcm4329-bt.o
```

- net/bluetooth/Makefile

```
obj-$(CONFIG_BT)      += bluetooth.o
obj-$(CONFIG_BT_L2CAP) += l2cap.o
obj-$(CONFIG_BT_SCO)    += sco.o
obj-$(CONFIG_BT_RFCOMM)   += rfcomm/
obj-$(CONFIG_BT_BNEP)   += bnep/
obj-$(CONFIG_BT_CMTP)  += cmtp/
obj-$(CONFIG_BT_HIDP)   += hidp/

bluetooth-objs := af_bluetooth.o hci_core.o hci_conn.o hci_event.o hci_sock.o hci_sysfs.o lib.o
```

- net/bluetooth/rfcomm/Makefile

```
obj-$(CONFIG_BT_RFCOMM) += rfcomm.o

rfcomm-y := core.o sock.o
rfcomm-$(CONFIG_BT_RFCOMM_TTY)+= tty.o
```

- net/bluetooth/bnep/Makefile

```
obj-$(CONFIG_BT_BNEP) += bnep.o

bnep-objs := core.o sock.o netdev.o
```

- net/bluetooth/hidp/Makefile

```
obj-$(CONFIG_BT_HIDP) += hidp.o

hidp-objs := core.o sock.o
```

5. Android 中命令行下的调试方法

5.1 Wi-Fi 模块在命令行下的启用/调试方法

依次键入以下命令:

```
# insmod /system/lib/modules/dhd.ko        // 加载 Wi-Fi Driver
# ifconfig eth0 up                        //节点上电
# wl up                                   //Open Radio
# wl scan                                 //Scan
# wl scanresults                          // wl scan 后等待两到三秒再运行这条命令
# wl join “SSID”                          // SSID 可以在 wl scanresults 命令的输出中找到
# dhcpcd eth0                             // 通过 DHCP 获取 IP 地址
# ping -c 4 xx.xx.xx.xx                   // ping 网络中的 IP，如可以 ping 通说明 Wi-Fi 已连上
```

注：wl join “SSID” 后可以使用# wl status 来查看已连接的 AP 信息。目前只能以 open 的方式连接 AP, 加密方式尚未做深入研究。

亦可通过 wpa_cli 查看相关网络信息

```
# wpa_cli interface                       //查看当前 WiFi 使用的接口
# wpa_cli scan                            //扫描当前网络
# wpa_cli scan_results                    //列出上一次的扫描结果
# wpa_cli list_networks                   //列出当前网络
```

5.2 BT 模块在命令行下的启用/调试方法

首先要将 Bluetooth 的电源打开，由于我们的 Kernel 在 sysfs 里注册了蓝牙电源的接口，因此这里可以使用 sysfs 下的接口方便地去打开/关闭电源:

```
# echo 1 > /sys/class/rfkill/rfkill0/state
```

其次要使用 brcm_patchram_plus 工具将厂商提供的 firmware 加载到 BT 模块的设备内存中:

```
# brcm_patchram_plus --patchram /system/etc/firmware/BCM4329_BT.hcd /dev/ s3c2410_serial0
```

注: brcm_patchram_plus 为 Broadcom 提供的工具，在 Android source tree 中包含有这个工具的 source，要将这个工具编译进 Android 需要在 BoardConfig.mk 文件 (对于 p2.0 板的 Android source，这个文件位于 <android_source_root>/ vender/sec/smdk6442/ 目录下) 中加入 BOARD_HAVE_BLUETOOTH_BCM := true; BCM4329_BT.hcd 为上面提到的 BT 模块的 firmware；/dev/s3c2410_serial0 为 p2 板上 BT 模块所使用的 UART 节点设备。

接下来要使用 hciattach 命令将 BT driver 与 BT 设备进行绑定:

```
# hciattach -n /dev/s3c2410_serial0 any &
```

命令成功执行后会在终端中显示 "Device setup complete"。

然后再用 hciconfig 命令将 BT 设备接口 hci0 启用:

```
# hciconfig hci0 up
```

此时再使用 hciconfig 命令会打印出类似于下面的信息:

```
# hciconfig
hci0:    Type: UART
         BD Address: 43:29:B0:00:95:99 ACL MTU: 1021:7 SCO MTU: 64:1
         UP RUNNING
         RX bytes:352 acl:0 sco:0 events:10 errors:0
         TX bytes:45 acl:0 sco:0 commands:10 errors:0
```

至此 BT 已成功开启，现在使用 hcitool 去 scan 周围的蓝牙设备已可扫描到:

```
# hcitool scan
Scanning ...
         C4:46:19:29:A5:2E          YSD-AIO-CSD
```

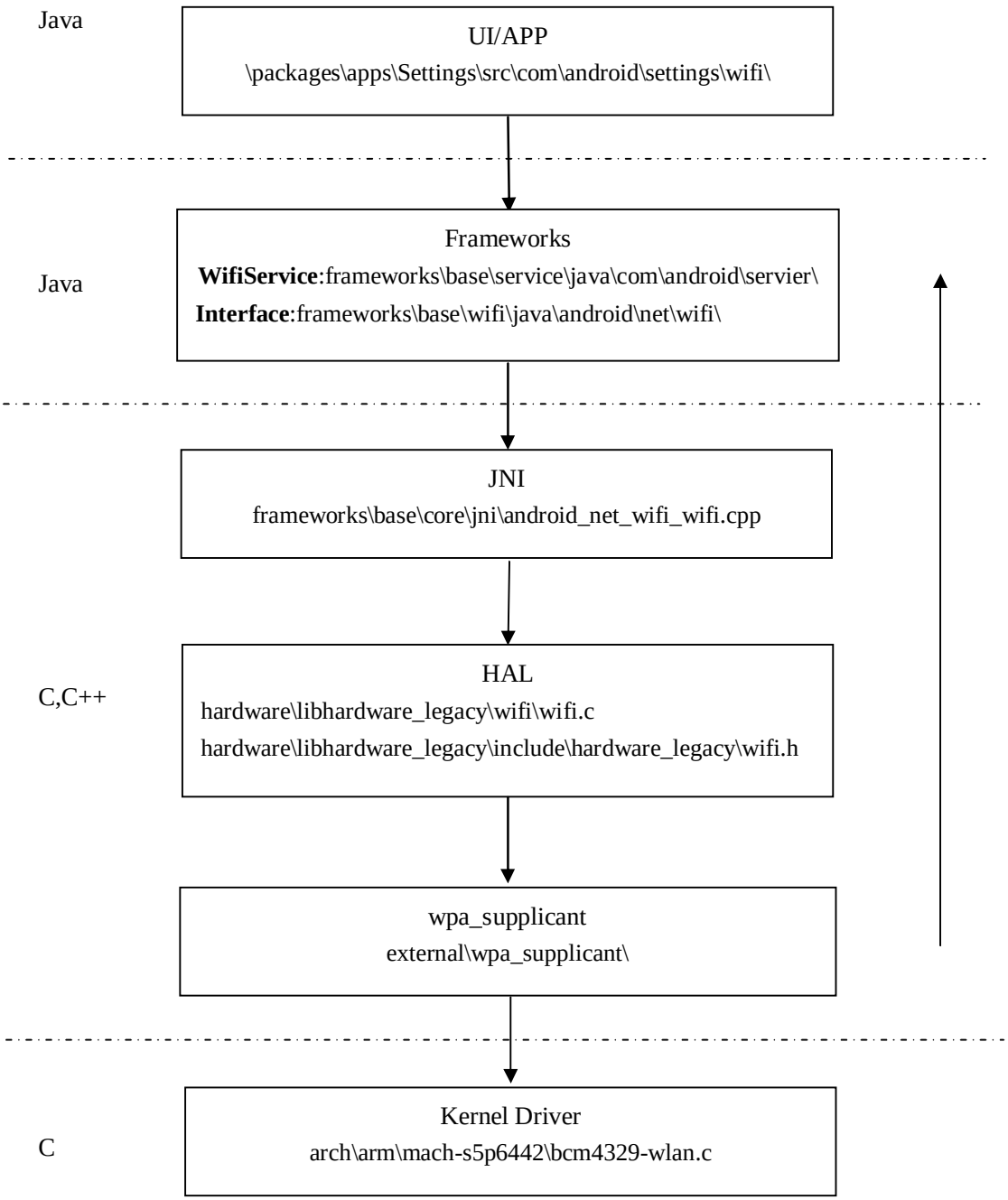
注:

1. hciattach, hciconfig, hcitool 都是 bluez 提供的工具，要将 android source 中的 bluez 编译 system imag 中需要在 BoardConfig.mk 文件中加入 BOARD_ HAVE_BLUETOOTH := true。
2. hcitool –cc dev_mac 的作用是建立两个蓝牙设备的连接，但从 BlueZ_4.0 之后此命令不再起作用。

6. Wi-Fi/Bluetooth 架构

以下所提到的路径，若无特殊说明，均指 Android source 的根目录下。

6.1 Wi-Fi 部分



6.1.1 相关代码分布

Android source 里与 Wi-Fi 有关的代码主要分布在以下几个位置:

- packages/apps/Settings/src/com/android/settings/wifi/

  Wi-Fi UI 层代码，用于控制 Wi-Fi 的开启/关闭和相关的设置。

  Settings 中的 WiFi 部分是用戶可見的設定界面，它提供了 Wifi 的開啟，關閉，掃描，連線等基本功能。通過 WifiLayer.Callback 提供了一組回調函數，用以回應用戶所關心的 wifi 狀態的變化。

  WifiEnabler 提供了開啟和關閉 WIFI 的功能，設定中的外層 WIFI 開啟、關閉菜單就是通過它實現的。

  WifiLayer 提供了 AP 選擇等功能，以供使用者自訂。

- frameworks/base/wifi/java/android/net/wifi/

  Wi-Fi Framework 层代码，为 UI 层代码提供相关的函数调用。

  WifiNative 為 WifiService,WifiStateTracker,WifiMonitor 提供底層操作支援。

  Service:WifiService:負責處理驅動載入，掃描，連線，斷開等命令以及底層所報告的事件。針對來自用戶端的控制命令，呼叫響應 WifiNative 的底層實做。在接收到用戶端的命令后，將其轉換成對應的自身訊息并存入信息隊列中，以便用戶端呼叫時可以及時傳回，然後在 WifiHandler 的 handleMessage 中處理對應的信息。對於底層說報告的事件，WifiService 則通過 WifiStateTracker 來處理。

  WifiStateTracker 的核心是 WifiMonitor 的事件循環機制，以及信息處理函數 handleMessage.

  WifiMonitor 通過開啟一個 MonitorThread 來實做事件的循環（循環的關鍵函數為 WifiNative.waitForEvent），獲取事件后，WifiMonitor 通過一系列的 Handler 通知給 WifiStateTracker。

  WifiMonitor 的通知機制是將底層時間轉換成 WifiStateTracker 能夠識別的信息，應加入 WifiStateTracker 的信息循環中，最後在 handleMessage 中由 WifiStateTracker 來完成對應的處理。

  WifiWatchdogService 是 ConnectivityService 所啟動的服務，它的作用是監控同一網絡內的接入點，如果當前接入點的 DNS 無法 PING 通，就自動切換到下一個接入點。WifiWatchdogService 通過 WifiManager 和 WifiStateTracker 來輔助完成具體的控制動作。通過更改 Settings.Secure.WIFI_WATCHDOG_On 來控制開啟和關閉 WifiWatchdogService。

  Client:WifiManager: wifi 部分與外界的接口，可以通過它來存取 Wifi 的核心功能。

- frameworks/base/core/jni/

  JNI 部分的函數通過調用 wpa_supplicant HAL 層的接口來實做，通過引入 HAL 層的頭文件 wifi.h 來獲取 HAL 層所定義的界面。

- hardware/libhardware_legacy/wifi/

  Wi-Fi HAL 层代码，用于加载 Wi-Fi 驱动以及运行相关的守护进程。

  主要用於與 wpa_supplicant 常駐程序的通訊，以提供給 Frameworks 使用，它實做了載入，控制，信息監控等功能。WIFI 的 HAL 層是 libhardware_legacy.so 的一部分,它通過引入 wpa_supplicant 的頭文件 wpa_ctrl.h 連接到動態函數庫 libwpa_client.so à 即

PDF-XCHANGE Click to buy NOW! www.docu-track.com

wpa_supplicant。透過該層的頭文件 wifi.h 為 JNI 提供程序調用界面。
HAL 層還提供了一些與 DHCP(Dynamic Host Configuration Protocol)有關的操作。

- external/wpa_supplicant/

Wi-Fi 的守护进程，用于直接与 Wi-Fi Driver 进行交互。
一個獨立執行的常駐程序，其核心是一個信息循環，在信息循環中處理 WPA 狀態機，控制命令，驅動事件，設定信息等。其控制方式有命令行與圖形界面。Android 與 wpa_supplicant 的通訊通過 socket 完成。
編譯此目錄文件后產生動態函數庫 libwpa_client.so，可執行程序 wpa_supplicant，以及用于调试的工具 wpa_cli.

- arch\arm\mach-s5p6442\bcm4329-wlan.c

在以上文件中进行 bcm4329 wlan 部分相關 GPIO 的配置，實現 wlan 的上電，下電及電源管理等事件的處理。
WiFi 协议部分的主要接口是 include/net/目录中的 wireless.h
WiFi 协议部分的源文件在 net/wireless/目录中。
对应内核配置菜单，WiFi 协议的内容在 Networking support à Wireless 中选定，其中 Wireless extensions 和 Wireless extensions sysfs files 选项表示使用 ioctl 或 sysfs 对无线网络进行附加的控制。

WiFi 的驱动程序在 drivers/net/wireless/目录中。
对应内核配置菜单，WiFi 驱动程序配置选项为 Device Drivers à Network device support à Wireless LAN.

6.1.2 相关配置与代码修改

- vender/sec/smdk6442/BoardConfig.mk

开启 wpa_supplicant 的编译选项:

```
BOARD_WPA_SUPPLICANT_DRIVER:=WEXT
```

让 HAL 层自动加载驱动:

```
WIFI_DRIVER_MODULE_PATH:=/system/lib/modules/dhd.ko
WIFI_DRIVER_MODULE_NAME:=dhd
```

注: 以上几个变量也可在 hardware/libhardware_legacy/wifi/wifi.c 中指定。Wi-Fi 启动时，wifi.c 中会调用 insmod 函数加载上面指定的 driver。

- external/wpa_supplicant/wpa_supplicant.conf

wpa_supplicant 的配置文件，主要配置以下两个参数:

```
ctrl_interface=/data/wpa_supplicant
update_config=1
```

其中上面的参数指定 wpa_supplicant 可被其他系统服务访问的管道；下面的参数表明 AP 的配置可被最近一次的修改覆盖。

- vendor/sec_proprietary/wifi/res/wpa_supplicant.conf //wpa_supplicant 的配置文件

```
ctrl_interface=/data/wpa_supplicant        //wpa_supplicant 可被其他系统服务访问的管道
ctrl_interface_group=wifi
eapol_version=1                            //Security for enterprise
ap_scan=1                                  //Connect hidden SSID AP
update_config=1                            //AP 的配置可被最近一次的修改覆盖
wps_cred_processing=1
```

- vender/sec/smdk6442/BoardConfig.mk

  开启蓝牙相关程序的编译选项:

```
BOARD_WPA_SUPPLICANT_DRIVER:=WEXT
```

- vender/sec/smdk6442/AndroidBoard.mk

  加載 Wifi 資源文件:

```
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/wpa_supplicant.conf:system/etc/wifi/wpa_supplicant.conf  \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/dhd.ko:system/lib/modules/dhd.ko    \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/rtecdc.bin:system/etc/wifi/rtecdc.bin \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/nvram.txt:system/etc/wifi/nvram.txt  \
    $(LOCAL_PATH)/../../sec_proprietary/wifi/res/dhcpcd.conf:system/etc/dhcpcd/dhcpcd.conf
```

- vender/sec/smdk6442/init.rc

  Android 系统初始化脚本，需要加入以下語句以提供对 Wi-Fi 的支持:

```
# for loading firmware
mkdir /system/etc/wifi 0770 wifi wifi
chmod 0770 /system/etc/wifi

# wpa_supplicant.conf
chmod 0660 /system/etc/wifi/wpa_supplicant.conf
chown wifi wifi /system/etc/wifi/wpa_supplicant.conf

# wpa_supplicant socket
mkdir /data/wpa_supplicant 0771 wifi wifi
chmod 0771 /data/wpa_supplicant

# wpa_supplicant control socket for android wifi.c
mkdir /data/misc/wifi 0770 wifi wifi
mkdir /data/misc/wifi/sockets 0770 wifi wifi
chmod 0770 /data/misc/wifi
chmod 0660 /data/misc/wifi/wpa_supplicant.conf

# for dhcp
mkdir /data/misc/dhcp 0770 dhcp dhcp
chown dhcp dhcp /data/misc/dhcp
```

```
# for wifi.c property_get( )
setprop wifi.interface eth0

# Services for Wi-Fi used in wifi.c (HAL)
service wpa_supplicant /system/bin/wpa_supplicant -dd -Dwext -ieth0
                                                    -c/data/misc/wifi/wpa_supplicant.conf
    socket wpa_eth0 dgram 660 wifi wifi
    user wifi system
    group system wifi inet
    disabled
    oneshot

service dhcpcd /system/bin/dhcpcd -BKL -f /system/etc/dhcpcd/dhcpcd.conf -dd eth0
    disabled
    oneshot
```

注 1: init.rc 中所使用的語句并非 bash shell 或 Android shell 中的命令，而是内置于 init 主进程中的一些命令，详见 init 主进程的源代码(system/core/init/)。

注 2: 所有 Wi-Fi 相关的目录在 init.rc 中都要重新添加权限，并重设 owner 为 wifi:wifi。因为 Android 系统每次初始化的时候会把这些目录初始化为最低权限，且 owner 和 group 都被重设为 root，这将导致 Wi-Fi 相关进程没有足够的权限来操作这些目录和文件。

注 3: service 中的 disabled 参数代表开机后不会自动启动，而是等待其他程序将其启动；oneshot 代表在服务退出后不会自动重启。

注 4: dhcpcd 服务中若没有 -dd 参数则 dhcpcd 会 fork 出一个子进程作为守护进程，然后 kill 掉主进程，导致 init 进程找不到 dhcpcd 的 PID 从而无法将其关闭，最终导致 Wi-Fi 无法再次开启。

注 5: wpa_supplicant.conf 是 wpa_supplicant 的配置文件，wpa_supplicant 中使用了几种驱动作为插件來使用，通常使用 wext (wireless extensions)作为 Wifi 的標準實現，其他幾種驅動如；用於作為 AP (Access Point)的 hostap,這些都可以在編譯 wpa_supplicant 的時候進行配置。

注 6: wpa_supplicant 本身就是 linux 中的可執行程序，可以通過以下命令運行，
wpa_supplicant -dd -Dwext -ieth0 -c/data/misc/wifi/wpa_supplicant.conf
以上內容表示使用 wext 作為驅動，输出 debug 信息，使用 eth0 為接口，使用/data/misc/wifi/目錄下的 wpa_supplicant.conf 作為配置文件。

注 7: wpa_supplicant 相关参数使用说明

drivers:
  wext = Linux wireless extensions (generic)

options:
  -b = optional bridge interface name
  -B = run daemon in the background
  -c = Configuration file
  -C = ctrl_interface parameter (only used if -c is not)
  -i = interface name

-d = increase debugging verbosity (-dd even more)
-D = driver name
-g = global ctrl_interface
-K = include keys (passwords, etc.) in debug output
-t = include timestamp in debug messages
-h = show this help text
-L = show license (GPL and BSD)
-p = driver parameters
-P = PID file
-q = decrease debugging verbosity (-qq even less)
-v = show version
-w = wait for interface to be added, if needed
-W = wait for a control interface monitor before starting
-N = start describing new interface

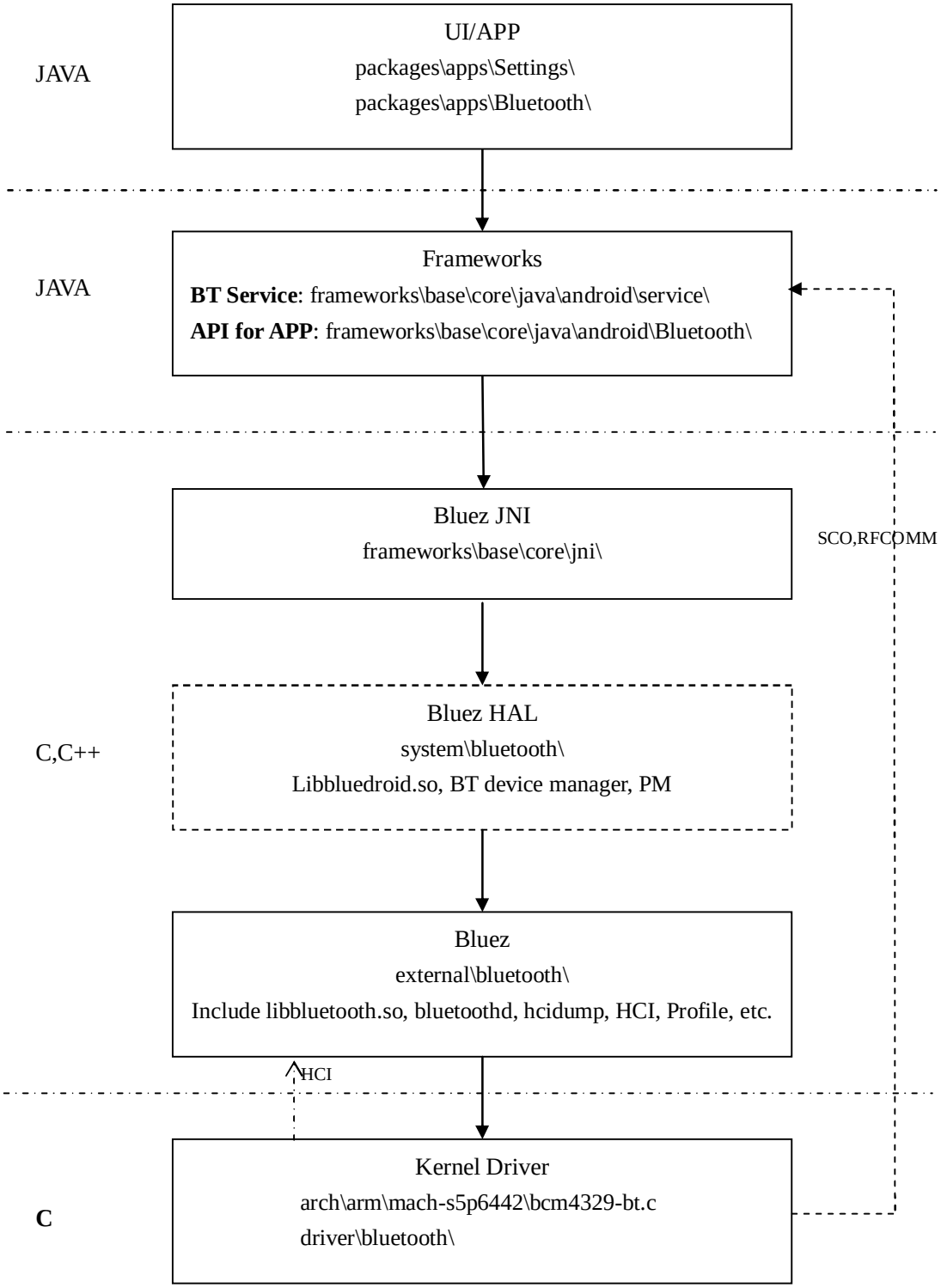
example:
wpa_supplicant -Dwext -iwlan0 -c/etc/wpa_supplicant.conf

注 8: wpa_cli 相关参数使用说明

wpa_cli [-p<path to ctrl sockets>] [-i<ifname>] [-hvB] [-a<action file>] \
[-P<pid file>] [-g<global ctrl>] [command..]
-h = help (show this usage text)
-v = shown version information
-a = run in daemon mode executing the action file based on events from wpa_supplicant
-B = run a daemon in the background

Commands:
interface [ifname] = show interfaces/select interface
level <debug level> = change debug level
reconfigure = force wpa_supplicant to re-read its configuration file
identity <network id> <identity> = configure identity for an SSID
password <network id> <password> = configure password for an SSID
new_password <network id> <password> = change password for an SSID
pin <network id> <pin> = configure pin for an SSID
list_networks = list configured networks
select_network <network id> = select a network (disable others)
enable_network <network id> = enable a network
disable_network <network id> = disable a network
add_network = add a network
remove_network <network id> = remove a network
set_network <network id> <variable> <value> = set network variables (shows list of variables when run without arguments)
get_network <network id> <variable> = get network variables
scan = request new BSS scan
scan_results = get latest scan results
ap_scan <value> = set ap_scan parameter
quit = exit wpa_cli

6.2 Bluetooth 部分



6.2.1 相关代码分布

- packages/apps/Settings/src/com/android/settings/bluetooth/

  Bluetooth UI 层代码，用于控制 Bluetooth 的开启/关闭和相关的设置。

  包含设置部分与应用部分,

  设置部分的代码在 packages/apps/Settings/src/com/android/settings/bluetooth/

  应用部分的代码在 packages/apps/Bluetooth/

- frameworks/base/core/java/android/bluetooth/

  Bluetooth Framework 层代码，为 UI 层代码提供相关的函数调用。

  负责管理并使用底层的本地服务，并封装成系统服务，负责提供相关类给应用层使用。

  相关 Profile 及协议 UUID 定义请参考

  Frameworks/base/core/java/android/bluetooth/BluetoothUuid.java

- frameworks/base/core/jni/

  主要是 Android_bluetooth_*, android_server_Bluetooth*命名的源代码文件及头文件，生成的内容是 Android 的 JNI 库 libandroid_runtime.so 的一个部分。

- system/bluetooth/

  Bluetooth HAL 层代码及 patchram (firmware) 加载工具等。

  生成 libbluedroid.so 以及众多的相关工具及库，主要实现的功能是对蓝牙设备的管理，比如电源管理操作。蓝牙设备的电源管理，通过内核的 rfkill 机制实现。

- external/bluetooth/

  蓝牙协议栈 bluez 提供的相关工具及 bluetoothd 守护进程。

  生成 libbluetooth.so, bluetooth, hcidump 等众多的相关工具及库。BlueZ 提供用户空间的蓝牙方面的支持，包含一个主机控制协议(HCI)，以及其他众多的内核实现协议的接口。同时，实现了蓝牙的所有应用规范(BT Profile).

  相关 Profile 及协议 UUID 定义请参考 external/bluetooth/bluez/下相关文件。

- arch\arm\mach-s5p6442\bcm4329-bt.c

  在以上文件中进行 bcm4329 BT 部分相關 GPIO 的配置，實現 BT 的上電，下電及電源管理等事件的處理。

  蓝牙设备通常通过 USB,SDIO 或 UART 进行连接。USB,SDIO 通常使用标准的硬件接口和驱动，UART 通常需要使用芯片上的高速串口才能承载蓝牙协议中需要高速传输的部分。蓝牙部分的驱动程序在 kernel 的 drivers/bluetooth 目录下。蓝牙的协议层位于内核空间的较上层，Linux2.6.x 内核开始都已集成到 BlueZ.

6.2.2 相关配置与代码修改

- vender/sec/smdk6442/BoardConfig.mk

  开启蓝牙相关程序的编译选项:

```
BOARD_HAVE_BLUETOOTH:=true
BOARD_HAVE_BLUETOOTH_BCM:=true
```

- vender/sec/smdk6442/AndroidBoard.mk

  加載 BT firmware:

```
PRODUCT_COPY_FILES += \
    $(LOCAL_PATH)/../../sec_proprietary/bluetooth/BCM4329_BT.hcd:system/etc/firmware/BCM4329_BT.hcd
```

- external/libaudio/Android.mk

```
ifeq ($(BOARD_HAVE_BLUETOOTH),true)
    LOCAL_SHARED_LIBRARIES += liba2dp
endif
```

- vender/sec/smdk6442/init.rc

  Android 系统初始化脚本，需要加入以下語句以提供对 Bluetooth 的支持:

```
chmod 0770 /system/etc/bluez
chown bluetooth bluetooth /system/etc/bluez
chmod 0660 /system/etc/bluez/main.conf
chmod 0660 /system/etc/bluez/audio.conf
chmod 0660 /system/etc/bluez/input.conf
chmod 0660 /system/etc/dbus.conf
mkdir /system/etc/firmware 0770 bluetooth bluetooth
chmod 0770 /system/etc/firmware
chown bluetooth bluetooth /system/etc/firmware
chmod 0660 /system/etc/firmware/BCM4329_BT.hcd
chown bluetooth bluetooth /system/etc/firmware/BCM4329_BT.hcd
mkdir /data/misc/bluetoothd 0770 bluetooth bluetooth
chmod 0770 /data/misc/bluetoothd
chown bluetooth bluetooth /data/misc/bluetoothd

# open up access to bluetooth interface
# bluetooth power up/down interface
chown bluetooth bluetooth /sys/class/rfkill/rfkill0/type
chown bluetooth bluetooth /sys/class/rfkill/rfkill0/state
chmod 0660 /sys/class/rfkill/rfkill0/state

# Services for Bluetooth
service bluetoothd /system/bin/logwrapper /system/bin/bluetoothd -n -d
    socket bluetooth stream 660 bluetooth bluetooth
    socket dbus_bluetooth stream 660 bluetooth bluetooth
    # init.rc does not yet support applying capabilities, so run as root and
    # let hcid drop uid to bluetooth with the right linux capabilities
    group bluetooth net_bt_admin misc
    disabled

service hciattach /system/bin/brcm_patchram_plus --enable_hci --baudrate 460800 --patchram
```

```
/system/etc/firmware/BCM4329_BT.hcd /dev/s3c2410_serial0 --enable_lpm /dev/s3c2410_serial0
        group bluetooth net_bt_admin
        disabled

service hfag /system/bin/sdptool add --channel=10 HFAG
        user bluetooth
        group bluetooth net_bt_admin
        disabled
        oneshot

service hsag /system/bin/sdptool add --channel=11 HSAG
        user bluetooth
        group bluetooth net_bt_admin
        disabled
        oneshot

service opush /system/bin/sdptool add --channel=12 OPUSH
        user bluetooth
        group bluetooth net_bt_admin
        disabled
        oneshot

service hf /system/bin/sdptool add --channel=13 HF
        user bluetooth
        group bluetooth net_bt_admin
        disabled
        oneshot

service pbap /system/bin/sdptool add --channel=19 PBAP
        user bluetooth
        group bluetooth net_bt_admin
        disabled
        oneshot
```

注 1: 以上幾種服務均使用了 oneshot 方式表示只運行一次，bluttoothd 是 BlueZ 的守護進程，沒有使用 oneshot,啟動后需要通過 sdptool 啟動響應的 BlueZ 服務。

注 2: 相關藍牙底層協定：

HCI:藍牙協定中軟硬件之間的接口，可完成與藍牙硬件的互動

L2CAP:邏輯鏈路控制和配接協定，是 RFCOMM，SDP 等協定的基礎

SDP:服務發現協定，可存取其他服務的基礎協定

RFCOMM:端口通訊協定，為上層服務的實做提供了傳輸接口

SCO:同步資料交換協定，為語音等需要同步傳輸的服務提供了支援(不經過 HCI）

注 3: BlueZ 提供的命令工具：

hcidump:檢查 HCI 通訊的細節

hcitool:查看藍牙芯片狀態，發送特定命令等

rfcomm:建立 rfcomm 連接
sdptool:Service 相關工具
l2ping:L2CAP 協議連通性檢測工具

7. 遇到的问题及解决方法

7.1 Wi-Fi 部分

(1) Driver 加载上之后 firmware download 失败，eth0 始终 up 不起来，debug kernel 及 driver 时发现 driver 向 SDIO 发送命令 CMD53 且 data flag 为 100 时命令就会发送失败。后来发现是由于 SDIO 的 clock rate 过高导致，BCM4329 的 data sheet 中虽指出支持 50MHz 的 clock rate，但由于硬件 layout 缺陷等原因，SDIO 的 clock 实际达不到 50MHz，于是在 kernel 中将 SDIO 的 clock rate 调至 25MHz（修改“sdio.c”，应该可以更高），问题解决。另外，在 SMDK6442 板上使用无线传输文件时传 100 多 K 就出现 bus down 的 error，估计也是相同原因导致，尚未查证。

(2) 无法 enable Wi-Fi，查看所有文件夹都已被成功创建，后来发现是文件夹权限不对，都为 root:root，导致 Wi-Fi 相关进程无法操作相关文件，故失败（logcat 中得知 Wi-Fi 进程读取相关文件失败）。解决方法是在 init.rc 中加入对所有 Wi-Fi 相关文件（夹）的权限及所有者控制。

(3) Wi-Fi 连上无线 AP 后重启 AP 然后 Wi-Fi 就无法再次连上 AP，现象是 Wi-Fi Settings 处总是显示正在获取 IP 地址，然后就 unsuccessful 断掉了。用 ps 命令查看进程发现有多个 dhcpcd 进程存在。试着在 HAL，framework，及 app 层代码中用 setprop ctl.stop dhcpcd 都无法将其结束。后发现是因为 dhcpcd 进程启动后默认会 fork 出一个子进程作为守护进程，然后 kill 掉父进程，这就导致 init 进程无法到 dhcpcd 的 PID 从而无法将其结束掉。解决方法：在 dhcpcd 命令后加上 -dd 参数就不会出现这个问题了。

(4) Wi-Fi 无法连接加密后的 AP（WEP，WPA/WPA2），且用 wl scan 扫描不到任何 AP（部分 source 可扫描到），后查证是因为所用的 firmware 与 driver 版本不匹配，且所用的 firmware 文件不是 for end user 的，而是用于 mfg test 的。更换为正确的 firmware 后问题解决。注：使用“wl ver”命令可以查看当前使用的 firmware 版本。

(5) 手动下命令 dhcpcd eth0 时报如下错误：

.....

eth0: send_raw_packet: Bad file descriptor

.....

导致 dhcpcd 无法获得 IP；解决方法：在 kernel menuconfig 选项中：Networking Support → Networking Options → 选中 Packet socket；重新编译并使用新的 kernel 后问题解决。

后记：这个方法同时解决了 Android UI 里 Wi-Fi 无法开启（主要是 wpa_supplicant 没起得来）的问题；此时单独运行 wpa_supplicant 时会报如下错误：

socket(PF_PACKET): Address family not supported by protocol

7.2 Bluetooth 部分

(1) 无法启用 Bluetooth，无明显现象，后发现是由于 init.rc 中将某些需要操作串口设备(/dev/s3c2410_serial0)的服务 owern 设为了 bluetooth，导致它们无权操作串口设备。解决方法是注册这部分服务时不要设置 owner，让它们默认从属于 root 即可。其他程序及目录的权限也要多加注意。

(2) BT patchram(firmware)无法down至设备内存。检查三个地方：

a. 连接到设备的那颗32K RTC clock是否正常工作（需要设置 S3C2410_RTCCON_CLKEN 这个寄存器位）；

b. setup_sdhci.c 里 s5p6442_setup_sdhci2_cfg_gpio( ) 函数中有没有误操作到 BT reset pin(GPG2[2])；

c. Setting GPG2CONPDN to 0x2a9a,change GPG2CONPDN to Output1.具体分析过程及解决方法可以参考本人另一篇文章之 BT_Analysis.pdf.

GPG2CONPDN	Bit	Description
GPG2[n]	[2n+1:2n] n=0~6	00 = Output 0 01 = Output 1 10 = Input 11 = Previous state

(3) Bluetooth 最初通過 A2DP 播放音乐无法听到声音，在 external/libaudio/目录的 Android.mk 文件中添加如下代码后问题解决。特别提醒，大家注意，Bluetooth 的 A2DP 与 Audio 芯片本身的解码并没有关系，此部分建议重点关注 AudioFinger，本人在 A2DP 最初的 debug 过程中因为对此不了解，造成了方向的偏失。

```
ifeq ($(BOARD_HAVE_BLUETOOTH),true)
  LOCAL_SHARED_LIBRARIES += liba2dp
endif
```

(4) 通过 A2DP 播放音乐时有停顿、延时状况，造成此状况的原因是在串口初始化时对串口波特率的设置较低，此外，造成该状况的可能原因還有 BlueZ, buffer size.具体状况需要根据实际情况而定。

(5) 对蓝牙模块的功耗实现，目前已达成 BT 在 Sleep 模式开启状态下的功耗为 0.13uA, 配对状态下的功耗为 0.4mA. 具体分析过程及解决方法可以参考本人另外一篇文章之 BCM4329_Power_Consumption.pdf.

(6) Fixed Bluetooth default status can't enable in sometimes.

- \system\bluetooth\bluedroid\bluetooth.c

```
int bt_enable() {
    LOGV(__FUNCTION__);

    int ret = -1;
    int hci_sock = -1;
    int attempt;
    //for_ticket455_s
    set_bluetooth_power(0);
    usleep(10000);
    //for_ticket455_e
    if (set_bluetooth_power(1) < 0) goto out;
    //for_ticket455_s
```

```
sleep(2);
//for_ticket455_e
LOGI("Starting hciattach daemon");
if (property_set("ctl.start", "hciattach") < 0) {
        LOGE("Failed to start hciattach");
        goto out;
}

// Try for 10 seconds, this can only succeed once hciattach has sent the
// firmware and then turned on hci device via HCIUARTSETPROTO ioctl
for (attempt = 1000; attempt > 0;  attempt--) {
        hci_sock = create_hci_sock();
        if (hci_sock < 0) goto out;

        if (!ioctl(hci_sock, HCIDEVUP, HCI_DEV_ID)) {
                break;
        }
        close(hci_sock);
        usleep(10000);  // 10 ms retry delay

}
if (attempt == 0) {
        LOGE("%s: Timeout waiting for HCI device to come up", __FUNCTION__);
//for_ticket455_s
        #if 1
        //If hciattach failed,kill it then we can start hciattach deamon again.
        LOGI("Stopping hciattach deamon");
        if (property_set("ctl.stop", "hciattach") < 0) {
        LOGE("Error stopping hciattach");
        }
        #endif
        //for_ticket455_e
        goto out;
}

LOGI("Starting bluetoothd deamon");
if (property_set("ctl.start", "bluetoothd") < 0) {
        LOGE("Failed to start bluetoothd");
        goto out;
}
sleep(HCID_START_DELAY_SEC);

ret = 0;
```

PDF-XCHANGE Click to buy NOW! www.docu-track.com


```
    out:
        if (hci_sock >= 0) close(hci_sock);
        return ret;
    }
```

8. 结束语

(1) 应首先配置好相关 GPIO PIN，确保硬件 GPIO PIN 连接正确。

(2) 对于 WiFi,应首先确保能够正确编译出驱动文件。

(3) 尝试 wpa_supplicant, hciattach 初始化。

(4) 将 wpa_supplicant, hciattach 的初始化整合到 init.rc 中。

(5) 关于如何在 kernel 中引用 ANDROID 的 tool，在此以 HCITOOL 为例进行说明，

```
static char eppconfig_path[256] = "/system/xbin/hcitool";
static char *envp[] = { "HOME=/", "TERM=linux", "PATH=/usr/bin:/bin", NULL };
char *argv[] = { eppconfig_path, "-i hci0 cmd 0x3f 0x27 0x1 0x0 0x0 0x1 0x1 0x0 0x1 0x1
0x0 0x0 0x0 0x1",NULL };
call_usermodehelper(eppconfig_path, argv, envp, UMH_WAIT_PROC);
```

以上，可结合 kernel\arch\arm\mach-s5p6442\bcm4329_bt.c 进行。

由于本人初学 Android，水平有限，以上难免存在错误及不足，欢迎广大专家，读者批评指正，谨以此篇献给广大 Android 的初学者以及对 Android 系统 BSP 开发有兴趣的同仁!