1.1 概述

近期需要把 WiFi 无线网络功能移植到 iTOP-4412 开发平台，查阅了相关资料，经过一段时间的研究、调试,终于成功的将 WiFi 功能移植到了开发板上面，这里笔者记录移植过程及注意事项，方便以后工作需要。

iTOP-4412 开发板的 WiFi 模块与板卡之间的连接采用 SDIO 接口，WiFi 硬件模块使用的是 MTK 的 MT6620 芯片，MTK 提供了 Android4.0 及 Android4.4 的 driver， Porting Guid，有了这些就为我们的移植工作做了总体性的指导。

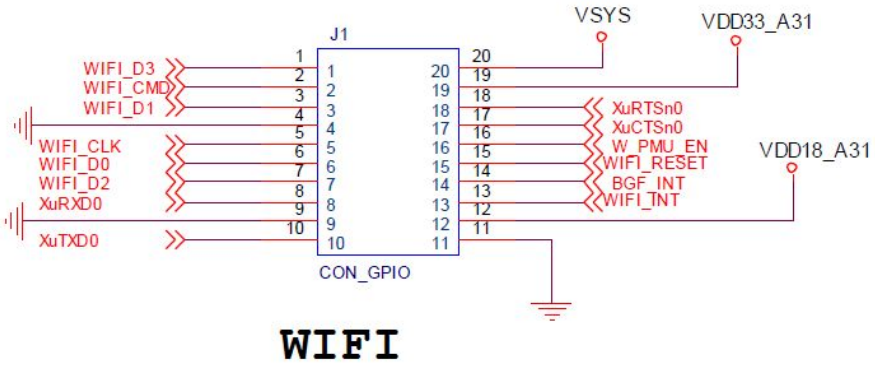
但是仅仅有 MTK 提供的文档还是远远不够的，毕竟硬件接口定义不同，kernel 版本也不同，Android 层与 MTK 提供的代码也有差异，这就需要我们在 MTK 文档的指导下，Step by Step 进行 Porting 工作.

移植环境:

1 iTOP-4412 精英版 + MT6620 WiFi 模块
2 kernel 3.0.15 version
3 Android4.4.4
4 Ubuntu12.04 64Bit 开发环境

1.2 硬件相关部分

下图为 WiFi 模块与开发板连接的引脚定义，通过该接口可以看出 WiFi 模块与 CPU 的交互接口.



WIFI

查看 WiFi 模块的原理图可知，WiFi 模块与 CPU 之间采用 SDIO 接口和串口进行数据和命令的交互工作，Pin1,2,3,5,6,7 为 SDIO 接口，另外还需要 Pin8，Pin10 UART 串口，另外 Pin18,Pin17 用于串口流控，实际是可以不使用流控功能。 MT6620 芯片是复合芯片，除了具备 WiFi 功能以外，还支持蓝牙，FM，GPS 功能，他们与 CPU 之间的通信需要串口，另外 MT6620 固件补丁的下载也是通过串口进行的，所以说串口是必不可少的硬件接口，即使您只使用该芯片的 WiFi 功能。

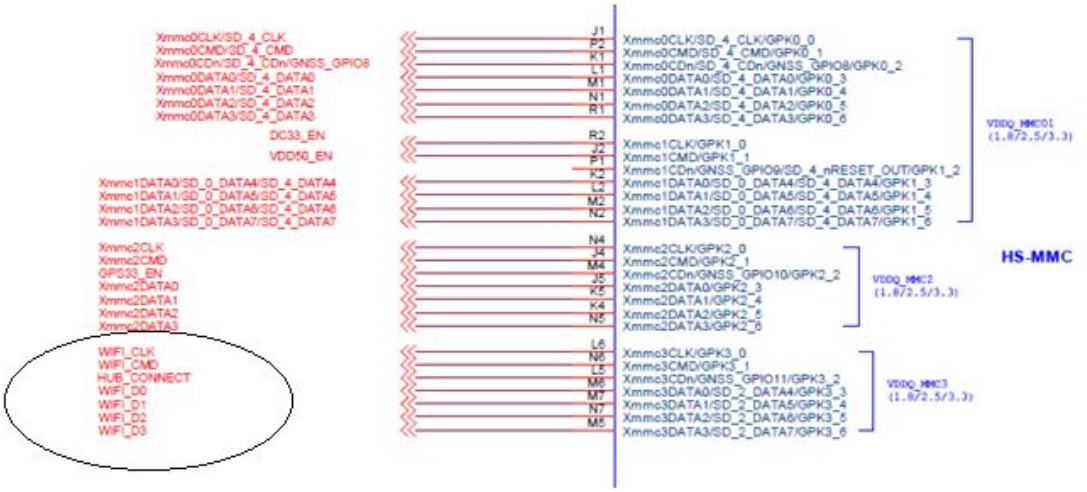
除了 SDIO 接口，UART 接口以外，还需要与 CPU 进行交互的接口包括：

Pin16 PMU_EN ，使能引脚，默认低电平状态，高电平有效。

Pin15 WiFI_RESET 引脚，默认低电平状态，高电平有效,用于复位 WiFi 芯片；

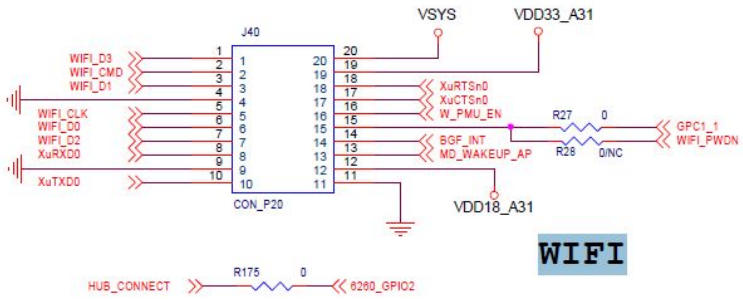
Pin 13 WiFi INT WiFI 的中断引脚，用于告知 CPU 有数据来了;

以上这些引脚需要在 BSP 中配置，WiFI 的驱动会通过这些引脚与 MT6620 芯片进行通信;另外需要强调的是 WiFi 模块占用 CPU 的 MMC3 端口，也可以说就是 SDIO 总线，关于 MMC，SD，SDIO 总线的来历这里不再描述，下图为核心板 WiFi 相关部分：



在 iTOP-4412 精英版中 mmc0,mmc1 总线用于 eMMC，mmc2 用于 TF 卡，mmc3 用于 SDIO WiFi，关于 wifi 的其他硬件连接比如串口，使能（GPIO），复位（GPIO），中断（GPIO）这里不再贴出，具体的请查看开发板的原理图。

这里重点强调一下 HUB_CONNECT，也就是上图中的 L5 引脚，该引脚复用为 mmc 总线的 CD 引脚，也就是 Card Detect 引脚，用于告知 MMC3 控制器有设备插入，属于中断引脚，我们需要把该引脚与 CPU 的 某个 GPIO 引脚连接：



以上为 iTOP4412 底板原理图 WiFi 接口部分，请注意 HUB_CONNECT 引脚通过 0 欧电阻与 6060_GPIO2 进行了连接，6060_GPIO2 为核心板 GPIO 引脚。

这样连接的目的是通过软件输出 6060_GPIO2 低电平，从而 HUB_CONNECT 引脚为低电平输入状态，MMC3 控制器认为有设备插入到了 MMC3 总线上面，原理同 TF 卡,SD 卡的检测。

以上为进行 Porting 前的准备工作，当然需要万用表，示波器工具进行辅助的检测，查看 WiFi 模块的工作电压是否正常，GPIO 的当前状态，MMC 总线上面的时钟及是否有数据从 MMC3 控制器输出等等.

1.3 Kernel

1.3.1 概述

iTO-P4412 开发板采用的内核是 Linux 3.0.15 版本，MTK 官方给的移植 Porting 没有说明针对具体的 kernel 版本，由于是 Android4.4，所以 kernel 应该是 3.0 以后的版本或者更高支持；

首先按照 PoringGuid 的指导说明，把 New 和 Modify 文件夹下面关于 kernel 部分的修改放到我们的 kernel 代码里面，MT6620 的驱动分两个部分，一部分放在 driver/misc/目录下面，文件夹名称 mediatek,里面存放的是 WMT，既 wireless manage tools，里面提供了与 MT6620 download firmware patch ，enable /disable WIFi 芯片，power on, power off 操作的相关驱动部分，及 SDIO 总线设备接口驱动 Host Interface drivers，这些驱动工作正常后才开始加载 WiFi 网络相关驱动。

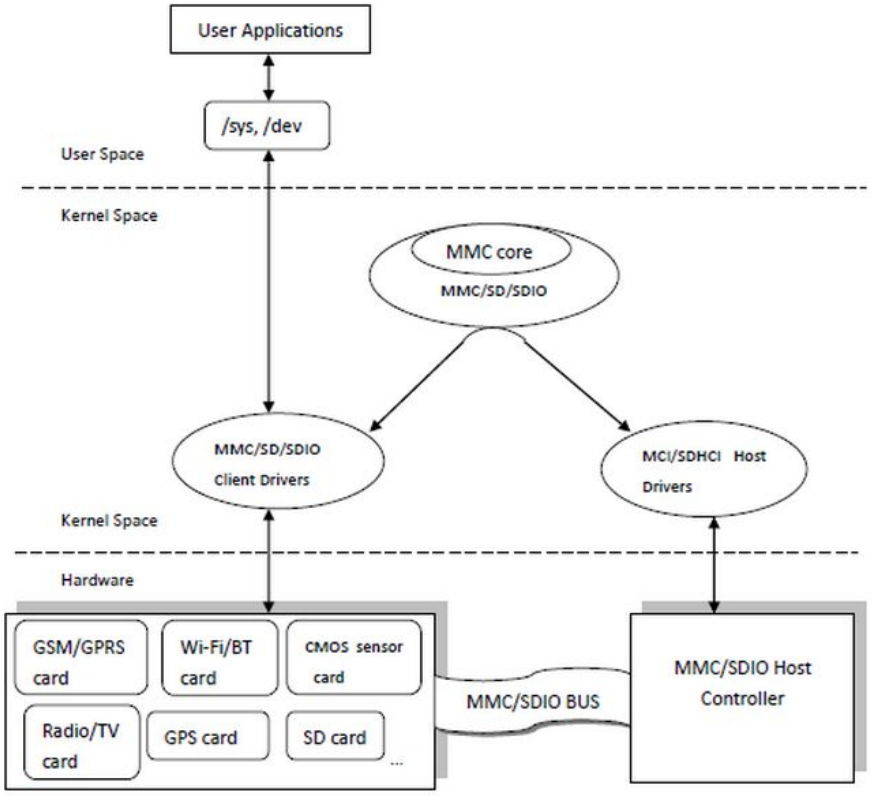
我们以驱动库 .ko 的形式编译驱动模块，driver/misc/mediatek/ 库文件与 WiFi 网络库文件 列表如下::

mtk_hif_sdio.ko ----mmc 总线相关接口，mmc 总线发现 SDIO 设备，分配总线地址后，会与该驱动进行适配.适配成功后该驱动会调用 WiFi 网络驱动；
mtk_stp_uart.ko-----串口相关驱动,通过串口下载固件补丁,设置芯片参数;
mtk_stp_wmt.ko-----core 部分，提供 WiFi 上电，断电等等相关操作;
mtk_wmt_wifi.ko ----字符设备驱动，创建设备节点用于与用户空间交互;
wlan_mt6620.ko -- -- WiFi 网络相关驱动,不需要我们进行修改;

另外这里附一张 MMC 驱动框架图：

因为我们的 MT6620 模块挂载到了 MMC 总线上面，属于 MMC 子系统的工作实例，我们非常有必要熟悉一下 mmc 驱动架构，是我们移植工作非常重要的一部分，关于 MMC 子系统的详细介绍这里不再说明.

MMC子系统框架



1.3.2 代码修改

1 根据硬件连接情况配置必要的平台资源

修改文件: kernel/iTop4412_Kernel_3.0/arch/arm/mach-exynos/mach-itop4412.c

关键函数 1: 该函数配置 WiFi 相关的 GPIO 引脚为初始化输出状态，或者配置为中断状态 WIFI 驱动会改变这些引脚的状态，这里仅仅是初始化.

```
static void __init mtk_combo_init(void)
 {

        //MT66XX PMUEN
        if(gpio_request(EXYNOS4_GPC1(0), "GPC1_0"))
        {
                printk(KERN_ERR "failed to request GPC1_0 for MT6620 PMUEN control\n");
        }
```

```
        //MT66XX SYSRST
        if(gpio_request(EXYNOS4_GPC1(1), "GPC1_1"))
        {
       printk(KERN_ERR "failed to request GPC1_1 for MT6620    SYSRST control\n");

        }
        s3c_gpio_cfgpin(EXYNOS4_GPC1(0), S3C_GPIO_OUTPUT);
        s3c_gpio_cfgpin(EXYNOS4_GPC1(1), S3C_GPIO_OUTPUT);

        gpio_direction_output(EXYNOS4_GPC1(0), 0);
        gpio_direction_output(EXYNOS4_GPC1(1), 0);


        gpio_free(EXYNOS4_GPC1(0));
        gpio_free(EXYNOS4_GPC1(1));

        mdelay(5);

        //need config eint models for Wifi & BGA Interupt
        if (gpio_request(EXYNOS4_GPX2(5), "WiFi INT"))
                printk(KERN_WARNING "MT6620 WiFi INT(GPX2.5) Port request error!!!\n");
        else    {
                s3c_gpio_setpull(EXYNOS4_GPX2(5), S3C_GPIO_PULL_NONE);
                s3c_gpio_cfgpin(EXYNOS4_GPX2(5), S3C_GPIO_SFN(0xF));
                gpio_free(EXYNOS4_GPX2(5));
        }

        if (gpio_request(EXYNOS4_GPX2(4), "BGF INT"))
                printk(KERN_WARNING "MT6620 BGA INT(GPX2.4) Port request error!!!\n");
        else    {
                s3c_gpio_setpull(EXYNOS4_GPX2(4), S3C_GPIO_PULL_NONE);
                s3c_gpio_cfgpin(EXYNOS4_GPX2(4), S3C_GPIO_SFN(0xF));
                gpio_free(EXYNOS4_GPX2(4));
        }

        //normal it is high level
        if (gpio_request(EXYNOS4_GPX3(2),    "6260_GPIO2")!=0) {
                printk("[mt6620] ERROR:Cannot request 6260_GPIO2\n");
          } else {
                gpio_direction_output(EXYNOS4_GPX3(2), 1);/* WLAN_CHIP_PWD */
                gpio_set_value(EXYNOS4_GPX3(2), 1);
                mdelay(100);
                gpio_free(EXYNOS4_GPX3(2));
}
```

```
    return; }
```

关键函数 2: *setup_mt6620_wlan_power_for_onoff*

该函数为导出函数，WIFi 驱动会调用该函数，该函数关键地方是让 MMC 控制器驱动扫描 MMC 总线上面的设备，MMC 扫描到了 WiFI 模块才会加载相应的 WiFi 驱动，这里是主动让 MMC 扫描，我们的 SD 卡是采用中断触发的方式扫描，他们本质上都是扫描 MMC 总线上面的新设备，然后加载对应的设备驱动，具体的可以看一下 MMC 子系统相关内容.

函数所属文件: kernel/iTop4412_Kernel_3.0/arch/arm/mach-exynos/mach-itop4412.c

```
void setup_mt6620_wlan_power_for_onoff(int on)
{
    int chip_pwd_low_val;
    int outValue;

    printk("[mt6620] +++ %s : wlan power %s\n",__func__, on?"on":"off");

#if 1
    if (on) {
        outValue = 0;
    } else {
        outValue = 1;
    }

   if (gpio_request(EXYNOS4_GPX3(2),    "6260_GPIO2")!=0) {
        printk("[mt6620] ERROR:Cannot request 6260_GPIO2\n");
    } else {
        gpio_direction_output(EXYNOS4_GPX3(2), 1);/* WLAN_CHIP_PWD */
        gpio_set_value(EXYNOS4_GPX3(2), outValue);
        mdelay(100);
        gpio_free(EXYNOS4_GPX3(2));
   }

    if(on)
    {
     //need reset on mt6620 ? need test.....
    }
#endif

    extern void sdhci_s3c_sdio_card_detect(struct platform_device *pdev);
```

```
    // mdelay(200);

    //need sdhc controler check wifi catd states.....
    sdhci_s3c_sdio_card_detect(&s3c_device_hsmmc3);

    printk("[mt6620] --- %s\n",__func__);

}
EXPORT_SYMBOL(setup_mt6620_wlan_power_for_onoff);
```

关键结构体：该结构体告诉 WiFi 驱动相关部分使用了平台的哪些 GPIO 资源.
结构体所属文件: kernel/iTop4412_Kernel_3.0/arch/arm/mach-exynos/mach-itop4412.c

```
  static struct mtk_wmt_platform_data mtk_wmt_pdata = {
        .pmu    =EXYNOS4_GPC1(0),    //RK30SDK_WIFI_GPIO_POWER_N,//RK30_PIN0_PB5,
//MUST set to pin num in target system
        .rst  =  EXYNOS4_GPC1(1),//RK30SDK_WIFI_GPIO_RESET_N,//RK30_PIN3_PD0,  //MUST
set to pin num in target system
        .bgf_int=EXYNOS4_GPX2(4),
//IRQ_EINT(20),//RK30SDK_WIFI_GPIO_BGF_INT_B,//RK30_PIN0_PA5,//MUST set to pin num in
target system if use UART interface.
        .urt_cts = -EINVAL, // set it to the correct GPIO num if use common SDIO, otherwise set
it to -EINVAL.
        .rtc = -EINVAL, //Optipnal. refer to HW design.
        .gps_sync = -EINVAL, //Optional. refer to HW design.
        .gps_lna = -EINVAL, //Optional. refer to HW design.
    };
    static struct mtk_sdio_eint_platform_data mtk_sdio_eint_pdata = {
        .sdio_eint                                                              =
EXYNOS4_GPX2(5),//IRQ_EINT(21)  ,//RK30SDK_WIFI_GPIO_WIFI_INT_B,//53,  //MUST set pin
num in target system.
    };
    static struct platform_device mtk_wmt_dev = {
        .name = "mtk_wmt",
        .id = 1,
        .dev = {
        .platform_data = &mtk_wmt_pdata,
        },
    };
    static struct platform_device mtk_sdio_eint_dev = {
        .name = "mtk_sdio_eint",
        .id = 1,
```

```
        .dev = {
        .platform_data = &mtk_sdio_eint_pdata,
        },
};
```

2 WIFI 驱动导出函数.

文件：
kernel/iTop4412_Kernel_3.0/drivers/misc/mediatek/combo_mt66xx/wmt/platform/vendor/wmt_plat.c

修改函数： *wmt_plat_sdio_ctrl*
函数说明：该函数会调用我们上面导出的接口，让 MMC 总线控制器扫描新设备

```
INT32 wmt_plat_sdio_ctrl (WMT_SDIO_SLOT_NUM sdioPortType, ENUM_FUNC_STATE on)
{
    int ret = 0;

        extern void setup_mt6620_wlan_power_for_onoff(int on);

    if (FUNC_OFF == on)    {
        /* add control logic here to generate SDIO CARD REMOVAL event to mmc/sd
          * controller. SDIO card removal operation and remove success messages
          * are expected.
          */

        //add by dg 2015-04-14
        setup_mt6620_wlan_power_for_onoff(0);
    }
    else {
        /* add control logic here to generate SDIO CARD INSERTION event to mmc/sd
          * controller. SDIO card detection operation and detect success messages
          * are expected.
          */

        //add by dg 2015-04-14
        setup_mt6620_wlan_power_for_onoff(1);

    }
        //extern int omap_mmc_update_mtk_card_status(int state);
        //ret = omap_mmc_update_mtk_card_status((FUNC_OFF == on)? 0: 1);
```

```
    WMT_INFO_FUNC(KERN_INFO "%s, on=%d, ret=%d\n", __FUNCTION__, on, ret);
    return ret;
}
```

以上两个文件的修改最为关键，当然您还需要配置 MMC3 的相关引脚为 MMC 工作状态，默认情况下面 MMC3 相关引脚为复用引脚中的 GPIO 状态，我们需要配置为 MMC 总线状态，笔者在调试过程中总是发现 MMC 总线上面没有命令或者数据输出，后发现默认情况下 MMC3 相关引脚并没有配置成 MMC 工作模式，查看 Exynos4412 Datasheet 后才发现这一问题.修改工作模式后，此问题得到解决.

MTK 官方给的移植文档中会告诉你需要在原始内核代码里面增加哪些文件，如何在 make menuconfig 中配置相关部分，这里就不再详细描述.

1.4 用户态空间

下面我们描述一下采用 Linux 系统和 Android 系统的用户都需要注意的地方：

驱动层移植完成后， MTK 的 Porting Guid 会告诉你需要在用户态运行 wmt_launcher 工具，作为后台的一个服务程序运行，该服务会配置串口的工作参数，下载固件补丁到 MT6620 中，他的源代码相对比较简单，只有一个.c 文件：

原始文件位于 MTK 发布包：
APEX_Android_4.4_MP_SW_package_V2.0/APEX_Android_4.4_MP_001_panda_combo_mt66xx_Package_Common/New/hardware/mediatek/wmt/ stp_uart_launcher.c

修改后的文件位于 iTOP-4412 Android4.4 发布包：
iTop4412_KK4.4/hardware/mediatek/wmt/stp_uart_launcher.c

修改点主要在串口参数配置上，由于内核版本不同，串口设置参数也略有不同.
具体修改可以使用代码比对工具进行比较。

另外需要说明的是运行 wmt_launcher 的运行参数 跟 MTK 给的移植文档有点不同，Porting Guid 里面推荐串口波特率使用 921600, 而在 iTOP-4412 的板子上面采用该值会工作不正常，导致固件补丁无法下载，开始怀疑板卡不支持该波特率，后使用串口测试工具专门针对这个串口进行 921600 测试，也没发现问题，后没有继续查找，而是运行 wmt_launcher 时采用 115200 波特率：

```
wmt_launcher -b 115200 -d /dev/ttySAC0 -p /system/etc/firmware &
```

注意： 如果您的操作系统使用的是 Linux 而不是 Android，需要修改 stp_uart_launcher.c 原始代码里面有 Android 特有的属性相关部分，Linux 系统不具有这个特性，我们提供了修改好的文件： stp_uart_launcher-linux-ok.c ，用户可以作为参考，该文件与原始文件 stp_uart_launcher-ori.c，及正常工作的文件 stp_uart_launcher.c 位于相同目录下面.

运行 wmt_launcher 服务后，然后执行 "echo 1 > /dev/wmtWifi"命令，如果工作正常，会产生 wlan0 网络节点，如果没有产生设备节点，中间会提示出错信息，需要根据信息查找相关问题，默认情况下 WiFi 驱动的调试级别为 DEBUG 级别，可以提升调试级别为更高，当然不要忘记把 Kernel 控制台输出级别也设置的高一些，驱动的输出信息依赖于驱动代码设置的调试级别及 Kernel 的控制台级别两部分。

调试信息多一些，方便定位与分析问题，驱动工作正常后需要把调试级别恢复为正常状态，过多的调试信息输出会影响驱动的工作效率和工作的结果，笔者在调试 MMC 部分由于把 MMC 总线的调试信息全部放开，导致 MMC 工作效率降低，WiFi 相关驱动总是适配不到 SDIO 设备，因为 WiFI 驱动会按一定的循环次数查找 SDIO 设备，由于 SDIO 相关驱动工作效率很低（大量的调试信息输出引起），导致 WiFi 驱动轮询次数结束了都没有匹配到设备。

如果产生了 wlan0 设备节点，那么下一步就是移植 wpa_supplicant 及 wpa_cli 程序了，Android4.4 采用的 wpa_supplicanat_8 版本，而不是以前 Android4.0 采用的 wpa_supplicant 版本，他们之间的差异还是比较大的，显著的一个区别是 Android4.4 里面 wpa_supplicant_8 使用的是 NL80211 驱动库。而 Android4.0 中的 wpa_supplicant 采用的是 WEXT 驱动库，如果您使用的是 Android4.4 的内核版本，运行的是 Linux 系统，那么需要您移植 wpa_supplicanat_8 到 Linux 文件系统中。

Android4.4 系统包含 wpa_supplicant_8 代码，编译 Android4.4 时会编译产生 wpa_supplicant_8. wpa_cli 为 wpa_supplicant 的客户端程序，可以使用该程序扫描无线网络，设置网络的 ESSID 和密码，连接到无线网络，Linux 用户需要使用 wpa_cli 进行网络连接。Android 用户也可以在命令行中使用该工具验证 WiFi 驱动及 wpa_supplicant_8 是否工作正常.

这些都没有问题后我们需要移植 HAL 层相关代码。

1.5 HAL 层移植

HAL 层移植相对简单，MT6620 采用的是 Android 的 WiFi 架构，没有经过修改，按照 MTK 的指导文档移植即可,这里需要注意的是 wifi.c 文件和 init.connectivity.rc 文件.

1.5.1 wifi.c 文件

wifi.c 文件的路径
iTop4412_KK4.4/hardware/libhardware_legacy/wifi.c

该文件会与 wpa_supplicnat 服务进行通信，是 Android 进行 WIFi 控制的 HAL 层的实现，根据 logcat 输出信息判断 WiFi 工作流程哪里出了问题，笔者修改了 wifi.c 文件的宏定义：

```
static char primary_iface[PROPERTY_VALUE_MAX];
// TODO: use new ANDROID_SOCKET mechanism, once support for multiple
// sockets is in


//dg cancel for mt6620
//#define WIFI_DRIVER_MODULE_NAME1      "rtl8188eu"
//#define WIFI_DRIVER_MODULE_PATH1          "/system/lib/modules/rtl8188eu.ko"
//#define WIFI_DRIVER_MODULE_NAME2      "rtl8192cu"
//#define WIFI_DRIVER_MODULE_PATH2          "/system/lib/modules/rtl8192cu.ko"
//#define WIFI_DRIVER_MODULE_NAME3      "rt5370sta"
//#define WIFI_DRIVER_MODULE_PATH3          "/system/lib/modules/rt5370sta.ko"
```

由于我们在 init.connectivity.rc 里面加载了 WiFi 驱动库及运行 wmt_launcher 服务，所以不再需要 wif.c 加载驱动了，直接注释掉相关宏即可.

wifi.c 的 int wifi_load_driver() 函数会设置 wifi 的相关属性：

```
static const char DRIVER_PROP_NAME[]       = "wlan.driver.status";
property_set(DRIVER_PROP_NAME, "ok");
```

设置属性后会触发 WiFi 上电操作，因为我们在 init.connectivity.rc 设置了属性触发：

```
# monitor property and power on/off wlan
on property:wlan.driver.status=ok
    write /dev/wmtWifi "1"

on property:wlan.driver.status=unloaded
    write /dev/wmtWifi "0"
```

wifi.c 会启动 wpa_suplicant 服务:

```
int wifi_start_supplicant(int p2p_supported)
```

该函数会查找 wpa_supplicant 服务是否已经运行，如没有运行会启动该服务.

1.5.2 init.connectivity.rc 文件

init.connectivity.rc 原始文件有 MTK 提供：
iTop4412_KK4.4/hardware/mediatek/config/combo_mt66xx/ init.combo_mt66xx.rc

原始文件名称为 init.combo_mt66xx.rc，拷贝到 ramdisk 的 root 目录下面名称变更为 init.connectivity.rc 文件 。

我们在该文件增加了加载驱动模块库操作，运行 wmt_lanucher 服务操作，另外需要注意文件原有的创建 wifi 相关目录操作，及修改权限，变更拥有者，这些 command 非常的重要，比如:

```
mkdir /data/misc/wifi 0770 wifi wifi
mkdir /data/misc/wifi/sockets 0770 wifi wifi
mkdir /data/misc/wpa_supplicant 0770 wifi wifi
mkdir /data/misc/p2p_supplicant 0770 wifi wifi
chown wifi wifi /data/misc/wifi/wpa_supplicant.conf
chown wifi wifi /data/misc/wifi/p2p_supplicant.conf
chmod 0660 /data/misc/wifi/wpa_supplicant.conf
chmod 0660 /data/misc/wifi/p2p_supplicant.conf
```

wpa_supplicant 会在/data/misc/wifi/sockets 目录下面创建 wlan0 文件节点用于与外部程序 wpa_cli 或者 Android 层服务进行通信.

这样在 Android4.4 的 Setting 里面打开 WiFi，就可以扫描到热点，连接互联网了，启动 WiFi 之前注意关闭有线连接，否则会存在网络访问冲突.

注意: Android 系统第一次运行是没有开启 WiFi 功能的，如果您的开发板上面有 WiFi 模块，且手动开启了 Android Setting 界面的 WiFi 功能 ，那么请不要把 WiFi 模块从底板上面拆除，否则 Android 启动过程中因为找不到 WiFi 模块 ，会频繁 打印出调试信息，导致 Android 启动失败 ，如果确实产生了这个问题，请重新烧写 Android 系统此问题即可解决.

1.5 总结

以上作为 iTOP-4412 开发平台移植 WiFi 功能的过程总结，即将发布的 Android4.4 的 Kernel 及 Android 层代码均包含 Porting 后的代码，也就是 Wifi 正常工作的代码，方便大家学习和产品研发.

如果您在实际的项目中需要 WiFi 功能，请参考我们的原理图设计硬件，尽量使用相同的 WiFi 资源，比如 WiFi 使能配置引脚，WiFi 中断配置引脚，WiFi 复位引脚，串口配置引脚等等，这样您只需要关注硬件部分，驱动使用我们移植好的即可，否则需要您修改 WiFi 引脚配置，进行必要的调试工作，增加自己的工作量。