

A33

AndroidL 通话驱动模块使用 说明书

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2014-12-31		初始版本

Confidential

目 录

A33	- 1 -
AndroidL 通话驱动模块使用说明书	- 1 -
1. 前言	- 4 -
1.1 编写目的	- 4 -
1.2 适用范围	- 4 -
1.3 相关人员	- 4 -
2. 驱动模块概述	- 4 -
3 驱动模块的使用	- 4 -
3.1. 驱动模块适用范围	- 4 -
3.2. 驱动模块配置信息	- 4 -
3.3. 驱动模块信息	- 5 -
3.4. 驱动模块加载	- 6 -
3.5. 驱动模块加载资源要求	- 6 -
3.6. 驱动模块调试测试	- 6 -
3.7. 功能模块接口	- 6 -
3.7.1. sw_module_mdelay	- 6 -
3.7.2. modem_get_config	- 7 -
3.7.3. modem_pin_init	- 7 -
3.7.4. modem_pin_exit	- 7 -
3.7.5. modem_vbat	- 7 -
3.7.6. modem_reset	- 8 -
3.7.7. modem_sleep	- 8 -
3.7.8. modem_dldo_on_off	- 8 -
3.7.9. modem_power_on_off	- 8 -
3.7.10. modem_rf_disable	- 9 -
3.7.11. modem_irq_init	- 9 -
3.7.12. modem_irq_exit	- 9 -
3.7.13. modem_early_suspend	- 9 -
3.7.14. modem_early_resume	- 10 -
3.8. 使用示例	- 10 -
3.8.1. 编译配置	- 10 -
3.8.2. mu509 驱动	- 11 -

1. 前言

1.1 编写目的

主要介绍 2G/3G 模组驱动功能模块的使用方法,以及客户如何通过使用功能模块提供的方法来实现 2G/3G 模组的驱动。

1.2 适用范围

适用于 A33 等平台的 2G/3G 模组驱动实现。

1.3 相关人员

主要供 A33 平台 2G/3G 模组驱动实现,及调试人员参考。

2. 驱动模块概述

2G/3G 驱动模块划分为三个部分:通话驱动模块、功能模块、以及各 2G/3G 模组驱动;主要完成 AP 及 BP 间的通信,包括 BP 的上电、断电、AP 唤醒/休眠 BP、以及复位 BP、禁止/使能 RF 等功能。客户可以根据我们提供的功能模块的接口方法来实现 2G/3G 模组驱动。

通话驱动模块代码实现位于\linux-3.4\drivers\misc\sw_3g_module\。

注意:Android 的配置方法可参考《A33-通话模组配置》。

3 驱动模块的使用

3.1. 驱动模块适用范围

此驱动模块适用于所有支持 2G/3G 模组的设备;不支持移动网络的设备不需要此驱动模块。在使用具体的 2G/3G 模组时候,需要单独提供此 2G/3G 模组的驱动模块(驱动实现此 2G/3G 模组的上电/断电、休眠/唤醒、以及禁止/使能 RF、复位等操作)。

补充:电话语音输出(蓝牙通话等)部分请参考《A33 Android 电话音频接口》

3.2. 驱动模块配置信息

linux 配置主要在 sys_config.fex 文件中完成.具体参数各 2G/3G 模组会有不同。驱动实现人员需要根据 2G/3G 模组的 spec 来配置下面的各项参数(包括 GPIO

引脚的定义等)。下面以 mu509 模组为例。

gpio 的形式: Port:端口+组内序号<功能分配><内部电阻状态><驱动能力><输出电平状态>

```
;3G configuration
;-----
[3g_para]
3g_used      = 1                // 使能 2G/3G 模组驱动
3g_usbc_num = 3
3g_uart_num = 4                //使用的串口数量
bb_name      = "mu509"         // 模组名称
bb_vbat      =
bb_on        =
bb_pwr_on    = port:PL09<1><default><default><0> // power 引脚
bb_wake      = port:PL11<1><default><default><0> // AP 唤醒/休眠 BP 的 sleep 引脚
bb_rf_dis    =
bb_rst       = port:PL10<1><default><default><0> // reset 引脚
bb_dldo      = "axp22_aldo1"
bb_dldo_min_uV = 5000000
bb_dldo_max_uV = 5000000

[wakeup_src_para]
bb_wake_ap    = port:PL08<4><default><default><0> //唤醒 AP 引脚
```

3.3. 驱动模块信息

通话驱动模块、以及 2G/3G 模组驱动均位于 \linux-3.4\drivers\misc\sw_3g_module\目录, 目前包含通话模块驱动以及各个 modem 的驱动。

```
|---sw_3g_module
|---core.c /* 驱动模块的功能实现, 读取配置文件, 设置引脚等, 2G/3G 模组 驱动编写人员即通过此模块提供的功能函数来实现特定 2G/3G 模组驱动 */
|---driver.c /* 通话驱动模块实现 */
|---sw_module.h /*驱动重要数据结构的定义、以及 core.c 功能函数的声明*/
|---cwm600.c /* 2G/3G 驱动模组实现 */
|---em55.c /* 2G/3G 驱动模组实现 */
|---g3.c /* 2G/3G 驱动模组实现 */
|---mu509.c /* 2G/3G 驱动模组实现 */
|---usi6276.c /* 2G/3G 驱动模组实现 */
|---wm5608.c /* 2G/3G 驱动模组实现 */
```

3.4. 驱动模块加载

通话及 2G/3G 模组驱动是直接编译到内核的，因此不需要通过 `insmod` 的方式进行加载。但是在编译版本的时候需要选择编译选项。具体编译配置请参考 3.8 节。

3.5. 驱动模块加载资源要求

暂无

3.6. 驱动模块调试测试

可以通过添加打印 `log` 的方法来进行调试(暂未定义用于控制 `log` 信息的宏)，使用 `modem_dbg` 宏打印输出 `log` 信息。

(1)如何判断模组上电是否成功：

由于各个模组的差异性，判断方法略有不同，需要根据模组的 `spec`，查看模组上电成功之后各个管脚的高低电平进行判断。

(2)如何判断模组是否正常工作：

一般我们可以通过拨打设备号码的方式来判断。如果能够成功呼叫设备的号码，则表明模组上电、工作正常。为了能够正常的注册到网络，一定要注意焊接天线。

3.7. 功能模块接口

功能模块实现的接口定义在 `core.c` 文件中，所有的 2G/3G 模组的驱动程序都是基于这些功能接口来完成的。接口函数在 `sw_module.h` 头文件中都有声明。

3.7.1. `sw_module_mdelay`

原型：`void sw_module_mdelay(u32 time)`

功能：由于不同的 2G/3G 模组，各操作有不同的时序要求，此方法主要提供延时功能，单位 毫秒。

参数：`time`-延时时长，单位 `ms`。

返回值：无

3.7.2. modem_get_config

原型: s32 modem_get_config(struct sw_modem *modem)

功能: 获取 sys_config.fex 文件关于 2G/3G 的配置信息 (3g_used、3g_usbc_num、3g_uart_num、bb_name、bb_vbat、bb_pwr_on、bb_rst、bb_rf_dis、bb_wake_ap、bb_wake、bb_dldo、bb_dldo_min_uV、bb_dldo_max_uV), 如果还有额外的配置信息需要读取, 则客户或者 驱动人员需要在此方法中添加, 并扩展 sw_modem 数据结构。

参数: modem-2G/3G 模组对象 返回值: 0-成功; !0-失败

3.7.3. modem_pin_init

原型: s32 modem_pin_init(struct sw_modem *modem)

功能: 初始化 2G/3G 模组的 gpio 引脚(只有在 sys_config.fex 中配置了引脚才会被初始化, 目前包括 bb_vbat、bb_pwr_on、bb_rst、bb_wake、bb_rf_dis 引脚, 均为输出引脚)。

参数: modem-2G/3G 模组对象。

返回值: 0-成功; !0-失败。

3.7.4. modem_pin_exit

原型: s32 modem_pin_exit(struct sw_modem *modem)

功能: 释放 2G/3G 模组的 gpio 引脚(包括所有在 modem_pin_init 中配置的 gpio 引脚: bb_vbat、bb_pwr_on、bb_rst、bb_wake、bb_rf_dis)。

参数: modem-2G/3G 模组对象。

返回值: 0-成功; !0-失败。

3.7.5. modem_vbat

原型: void modem_vbat(struct sw_modem *modem, u32 value)

功能: 操作 2G/3G 模组的 bb_vbat 引脚; 拉低或拉高。

参数: modem: 2G/3G 模组对象。

value: 0-表示拉低; 1-表示拉高 返回值: 无。

3.7.6. modem_reset

原型: void modem_reset(struct sw_modem *modem, u32 value)

功能: 操作 2G/3G 模组的 bb_rst 引脚; 拉低或拉高。

参数: modem-2G/3G 模组对象。

value: 0-表示拉低; 1-表示拉高 返回值: 无。

3.7.7. modem_sleep

原型: void modem_sleep(struct sw_modem *modem, u32 value)

功能: 操作 2G/3G 模组的 bb_wake 引脚; 拉高或拉低。

参数: modem-2G/3G 模组对象。

value: 0-表示拉低; 1-表示拉高 返回值: 无。

3.7.8. modem_dldo_on_off

原型: void modem_dldo_on_off(struct sw_modem *modem, u32 on)

功能: 使能或禁止电压校准器。

参数: modem-2G/3G 模组对象。

value: 0-表示打开; 1-表示关闭 返回值: 无。

3.7.9. modem_power_on_off

原型: void modem_power_on_off(struct sw_modem *modem, u32 value)

功能：操作 2G/3G 模组的 bb_pwr_on 引脚；拉低或拉高。

参数：modem-2G/3G 模组对象。

value: 0-表示拉低；1-表示拉高 返回值：无。

3.7.10.modem_rf_disable

原型：void modem_rf_disable(struct sw_modem *modem, u32 value)

功能：操作 2G/3G 模组的 bb_rf_dis 引脚；拉低或拉高。

参数：modem-2G/3G 模组对象。

value: 0-表示拉低；1-表示拉高 返回值：无。

3.7.11.modem_irq_init

原型：int modem_irq_init(struct sw_modem *modem, enum gpio_eint_trigtype trig_type)

功能：设置 2G/3G 模组中 BP 唤醒 AP 的 GPIO 引脚(bb_wake_ap)中断触发类型，以及请求 中断处理句柄。

参数：modem-为 2G/3G 模组对象；trig_type-中断触发类型(上升沿/下降沿)。

返回值：0-成功；!0-失败。

3.7.12.modem_irq_exit

原型：int modem_irq_exit(struct sw_modem *modem)

功能：释放申请的中断(bb_wake_ap 引脚中断), 通常在 2G/3G 模组停止后调用。

参数：modem-为 2G/3G 模组对象。

返回值：0-成功；!0-失败。

3.7.13.modem_early_suspend

原型: `void modem_early_suspend(struct sw_modem *modem)`

功能: 在 2G/3G 模组休眠的时候, 使能 BP 唤醒 AP 的 gpio 引脚 (bb_wake_ap)。

参数: modem-为 2G/3G 模组对象。

返回值: 无。

3.7.14.modem_early_resume

原型: `void modem_early_resume (struct sw_modem *modem)`

功能: 在 2G/3G 模组唤醒的时候, 关闭 BP 唤醒 AP 的 gpio 引脚 (bb_wake_ap)。

参数: modem-为 2G/3G 模组对象。

返回值: 无。

3.8. 使用示例

3.8.1.编译配置

首先, 需要在 `./linux-x.x/drivers/misc/sw_3g_module/` 目录下创建 2G/3G 模组驱动文件, 如添加 `mu509.c`。

其次, 需要在 `./linux-x.x/drivers/misc/sw_3g_module/Kconfig` 中加入驱动对应配置的描述, 如添加 `mu509.c` 驱动, 则在 `Kconfig` 文件中添加如下信息 (choice 选项我们在发布代码中已经添加, 添加一个新的模组时不需要重覆添加, HUAWEI_MU509 选项是 `mu509` 驱动新添加的内容)。

```
choice
    prompt "3G modem support"
    depends on SW_3G_MODULE
    help
        3G modem slecet.

config HUAWEI_MU509
    boolean "huawei mu509"
    help
        huawei mu509 modem support.
```

Kconfig 的主要作用是：配置模组是否加载。编译的时候，通过它来选定使用的 2G/3G 模组驱动。

以华为 mu509 模组为例，进入到 ./linux3.x/ 执行 " make ARCH=arm menuconfig " 命令

```

->选择" Device Drivers --->"
    ->选择" Misc devices --->"
        ->选择" softwinner 3G module driver --->"
            ->选择" 3G modem support()"
                ->选择" huawei mu509"
  
```

选定模组之后，回退到上一级目录；还需要选择 2G/3G 模组所支持的休眠/唤醒模式，以华为 mu509 模组为例：

```

->选择" 3G suspend/resume mode support ()"
    ->选择" ()bp sleep by gpio, ap wakeup by gpio"
  
```

最后，需要在 ./linux-x.x/drivers/misc/sw_3g_module/Makefile 文件尾部加入 2G/3G 模组驱动编译目标文件。如新添加了一个模组—mu509.c：

```
obj-$(CONFIG_HUAWEI_MU509) += mu509.o
```

注意：CONFIG_HUAWEI_MU509 的红色部分必须是 Kconfig 文件中配置的模组名称。

3.8.2.mu509驱动

以 mu509 驱动为例，简单介绍通话模组驱动的实现过程。

(1) 定义通话模组信息描述宏

```

#define DRIVER_DESC          SW_DRIVER_NAME
#define DRIVER_VERSION        "1.0"
#define DRIVER_AUTHOR         "Javen Xu"
#define MODEM_NAME            "mu509"
  
```

(2) 模组设备数据初始化

```

//g_mu509 模块驱动结构
static struct sw_modem g_mu509;

//模组驱动名称
static char g_mu509_name[] = MODEM_NAME;
  
```

```
//mu509 的平台设备初始化
static struct platform_device mu509_device = {
    .name                = SW_DRIVER_NAME,
    .id                  = -1,

    .dev = {
        .platform_data  = &g_mu509,
    },
};
```

(3) 根据 sw_module_op 结构定义并实现 mu509 的操作函数

sw_modem_ops 结构如下。

```
struct sw_modem_ops{
    // 根据模组 spec 定义的时序来实现开关机
    void (*power) (struct sw_modem *modem, u32 on);

    //根据模组 spec 定义的时序来实现模组的重置操作
    void (*reset) (struct sw_modem *modem);

    //根据模组 spec 定义的时序来实现模组的休眠唤醒操作
    void (*sleep) (struct sw_modem *modem, u32 sleep);

    //如果需要，根据模组 spec 定义的时序来实现模组 RF 的关闭和开启
    void (*rf_disable) (struct sw_modem *modem, u32 disable);

    //启动函数、关闭
    int (*start) (struct sw_modem *modem);
    int (*stop) (struct sw_modem *modem);

    void (*early_suspend) (struct sw_modem *modem);
    void (*early_resume) (struct sw_modem *modem);

    //休眠、唤醒函数
    int (*suspend) (struct sw_modem *modem);
    int (*resume) (struct sw_modem *modem);
};
```

sw_module_ops 结构在 mu509 驱动中的初始化如下。每个函数的具体实现与模组特性相关，需根据模组的使用说明文档实现。

```
static struct sw_modem_ops mu509_ops = {
    .power                = mu509_power,
```

```

.reset          = mu509_reset,
.sleep          = mu509_sleep,
.rf_disable     = mu509_rf_disable,

.start          = mu509_start,
.stop           = mu509_stop,

.early_suspend  = modem_early_suspend,
.early_resume   = modem_early_resume,

.suspend        = mu509_suspend,
.resume         = mu509_resume,
};

```

(4) 模组驱动的 init、exit 方法

Mu509 的 init 方法为 mu509_init。

```

static int __init mu509_init(void)
{
    int ret = 0;

    memset(&g_mu509, 0, sizeof(struct sw_modem));

    /* gpio */
    ret = modem_get_config(&g_mu509); //获取 sys_config.fex 中配置的通话模
                                     // 组参数
    if(ret != 0){
        modem_err("err: mu509_get_config failed\n");
        goto get_config_failed;
    }

    if(g_mu509.used == 0){
        modem_err("mu509 is not used\n");
        goto get_config_failed;
    }

    ret = modem_pin_init(&g_mu509); //用获取参数初始化 mu509
                                     // sw_modem 管脚
    if(ret != 0){
        modem_err("err: mu509_pin_init failed\n");
        goto pin_init_failed;
    }

    // if(g_mu509.name[0] == 0){

```

```

        strcpy(g_mu509.name, g_mu509_name);
//    }
    g_mu509.ops = &mu509_ops;

    modem_dbg("%s modem init\n", g_mu509.name);

    platform_device_register(&mu509_device);    //注册设备信息

    return 0;
pin_init_failed:

get_config_failed:

    modem_dbg("%s modem init failed\n", g_mu509.name);

    return -1;
}

```

Mu509 的 exit 函数为 mu509_exit()。

```

static void __exit mu509_exit(void)
{
    platform_device_unregister(&mu509_device);    //注销设备
}

```

(5) 加载驱动

```

late_initcall(mu509_init);
//module_init(mu509_init);
module_exit(mu509_exit);

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(MODEM_NAME);
MODULE_VERSION(DRIVER_VERSION);
MODULE_LICENSE("GPL");

```