Android 平台 GT818 驱动移植说明书

【程序总揽】

本程序针对 Android2.1 系统，移植的硬件平台为华天正科技开发的 Real6410（基于 S3C6410）。本驱动支持多指，能够在系统支持的情况下在主控进入关屏状态时自动调整 GT818 工作状态，达到省电的目的。

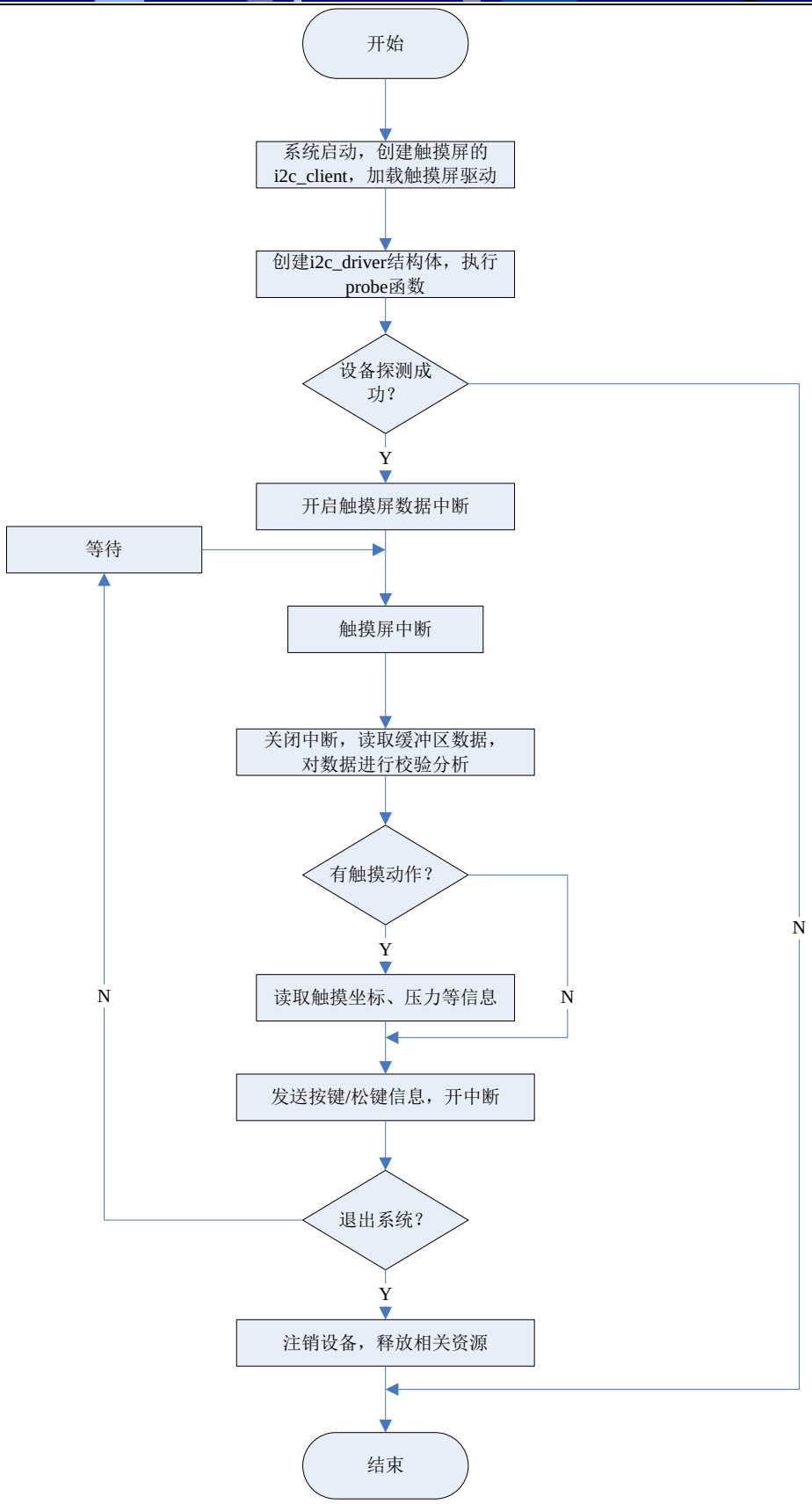
1.1 系统资源使用

资源类型	编号	作用
外部中断（可选）	硬件 IO： S3C64XX_GPL(10)	接收 Guitar 的数据中断，通知系统运行相应的坐标等数据处理
	系统中断号：119(（外部中断 18），配置为上升沿触发	
IO 口(必须有 SHUTDOWN)	154（S3C64XX_GPF(3)）	用 I2C 命令方式进行睡眠，用来唤醒芯片
I2C 适配器（SCL、SDA）	I2C0	Guitar 控制、数据传输

1.2 系统运行流程

为了便于移植，程序中采用了中断和定时查询两种方式，并建议使用中断方式。以中断方式为例，系统的主运行流程如下所示：

1. 创建 Guitar 对应的 i2c_client 对象，其中完成了该适配器的相关信息设置；
2. 加载触摸屏驱动，注意该驱动必须在 I2C 总线驱动已经加载的情况下才能进行，否则 I2C 通信将出错。程序中将其加载优先级设为最低；
3. 创建驱动结构体，探测 Guitar 设备，并将其挂载到 I2C 总线驱动上；测试 I2C 通信，注册 input 设备，申请 IO 口和中断资源，完成 Guitar 初始化工作（如有需要，烧录升级在此进行）；
4. 开启触摸屏中断，等待数据接收；
5. 收到触摸屏数据中断，关中断；
6. 通过 I2C 传输数据，分析手指信息，；
7. 更新按键信息，上报手指坐标、按键等信息，开中断；
8. 重复 4-7 之间的循环。



【移植指南】

4. 准备工作

本驱动采用GPL许可证，代码没有采用模块方式，所以使用本驱动前需要重新编译内核。在编译内核前，

我们需要做好以下准备工作：

1、 在硬盘上或使用虚拟机（如 VmWare）安装 Linux 系统，推荐使用 Ubuntu9.10；

2、 安装 gcc 及 arm-linux-gcc 和其他编译工具，将 CC 和 LD 指定为 arm-linux-gcc;

3、 获取平台对应的 Android 源码，解压到自己的工作目录。

4、 了解移植平台的 IO 口、中断、I2C 总线的使用方式。

5. 内核编译配置

在编译代码前我们需要进行内核编译配置，可以使用下面命令中的一个：

#make xconfig （基于 QT 的图形配置界面，非常直观，推荐使用）

#make menuconfig (基于文本菜单的配置界面)

下面我们以 make xconfig 为例，将我们的驱动增加到内核中去。假定我们源代码目录为：

~/android/kernel/

将编写的源代码复制到 Linux 内核源代码的相应目录：

~/android/kernel/driver/input/touchscreen/goodix_touch.c

~/android/kernel/include/linux/goodix_touch.h

~/android/kernel/include/linux/goodix_update.h //如果不做烧录则可以不加

1. 在目录 touchscreen 的 Kconfig 文件中增加新源代码对应项目的编译配置选项：

```
#---------------------------------------------------------
config TOUCHSCREEN_GOODIX_GT818
    # 配置项名称
    tristate "S3C6410 TouchScreen Driver"
    #选择项，选择 Y 标识要将其编译进内核
    default y    #默认编译选项
    depends on I2C    #依赖项，本驱动必须工作在 I2C 总线驱动的基础上
    help    #帮助信息
        It is a android driver to support Gooidx's touchscreen whose name
        is guitar on s3c6410 platform. The touchscreen can support multi-touch
        not more than two fingers.
        Say Y here to enable the driver for the touchscreen on the
        S3C SMDK board.
        If unsure, say N.To compile this driver as a module, choose M here:
        the module will be called goodix_touch.ko.
#---------------------------------------------------------
```

注意：

当将编译选项设置为 M 时，编译生成的驱动需要我们手动挂载。如需要系统启动时自动挂载，需要将模块 goodix_touch.ko 拷贝到系统模块加载目录，并修改对应的启动过程的 rc 脚本。

2. 在目录 touchscreen 的 makefile 文件中增加对新源代码的编译条目；

```
#这里的二进制文件名必须与源文件名移植，如 goodix_touch
```

obj-$(CONIDG_ TOUCHSCREEN_GOODIX_GT818) += goodix_touch.o

3. 进入内核源码目录，执行 make xconfig，查看 driver 项中是否增加了 GOODIX_TOUCHSCREEN 条目。

6. 如果只需要支持一款触摸屏，在内核设置中需要我们设定只编译一个触摸屏的驱动，此时与之前触摸屏的关联设置需要屏蔽。以 S3C 开发板提供的电阻屏驱动为例：

自带电阻屏的驱动配置选项为 TOUCHSCREEN_S3C 或 TOUCHSCREEN_S3C_ANDROID，而 s3c_ts_set_platdata、s3c_device_ts 是在 arch/arm/plat-s3c/dev-ts.c 文件中定义的，而这个源文件必须在 CONFIG_TOUCHSCREEN_S3C 或者 CONFIG_TOUCHSCREEN_S3C_ANDROID 有定义的情况下才会被编译，因此对这这个源文件中的函数或者变量的引用相应地应该加上诸如下面的这个编译控制宏：

```
#if defined(CONFIG_TOUCHSCREEN_S3C) || defined(CONFIG_TOUCHSCREEN_S3C_ANDROID)
……
#endif
```

这样能够避免修改了内核编译配置而造成的编译错误，并方便地通过 make config 设置来调整我们对触摸屏的支持。

7. 多指驱动支持

要支持多点触摸，首先需要 linux 内核的支持，也就是 input 子系统需要提供多点触摸的接口，其次是 GUI 库要能处理多点触摸的事件，后者在 Android2.0 及以上版本已经支持。内核方面，自 2.6.30 以后的内核的 input 子系统开始增加了多点触摸的支持，而比这更早的版本我们可以通过修改 input 子系统源文件来完成支持。

以内核 2.6.28 为例，主要修改的部分为：（红色部分为添加部分）

在文件（内核源文件目录)/include/input.h

```
#define SYN_REPORT          0
#define SYN_CONFIG          1
#define SYN_MT_REPORT       2

//-----------------------------------------
#define ABS_MT_TOUCH_MAJOR 0x30 /* Major axis of touching ellipse */
#define ABS_MT_TOUCH_MINOR 0x31 /* Minor axis (omit if circular) */
#define ABS_MT_WIDTH_MAJOR 0x32 /* Major axis of approaching ellipse */
#define ABS_MT_WIDTH_MINOR 0x33 /* Minor axis (omit if circular) */
#define ABS_MT_ORIENTATION 0x34 /* Ellipse orientation */
#define ABS_MT_POSITION_X 0x35 /* Center X ellipse position */
#define ABS_MT_POSITION_Y 0x36 /* Center Y ellipse position */
#define ABS_MT_TOOL_TYPE 0x37 /* Type of touching device */
#define ABS_MT_BLOB_ID 0x38 /* Group a set of packets as a blob */
#define ABS_MT_TRACKING_ID 0x39 /* Unique ID of initiated contact */
```

```
#define ABS_MT_PRESSURE 0x3a /* Pressure on contact area */
//-----------------------------------------------------------
#define ABS_MAX                 0x3f
#define ABS_CNT             (ABS_MAX+1)

/*
* MT_TOOL types
*/
#define MT_TOOL_FINGER 0
#define MT_TOOL_PEN 1
```

在同一文件添加一下相应处理函数：

```
static inline void input_mt_sync(struct input_dev *dev)
{
    input_event(dev, EV_SYN, SYN_MT_REPORT, 0);
}
```

在文件（内核源文件目录)/drivers/input/input.c 添加：

```
static unsigned int __initdata    input_abs_bypass_init_data[]    = {
    ABS_MT_TOUCH_MAJOR,
    ABS_MT_TOUCH_MINOR,
    ABS_MT_WIDTH_MAJOR,
    ABS_MT_WIDTH_MINOR,
    ABS_MT_ORIENTATION,
    ABS_MT_POSITION_X,
    ABS_MT_POSITION_Y,
    ABS_MT_TOOL_TYPE,
    ABS_MT_BLOB_ID,
    ABS_MT_TRACKING_ID,
    ABS_MT_PRESSURE,
    0
};

static unsigned long input_abs_bypass[BITS_TO_LONGS(ABS_CNT)];

static void input_handle_event(struct input_dev *dev,
            ……
                disposition = INPUT_PASS_TO_HANDLERS;
            }
            break;
        case SYN_MT_REPORT:
```

```
                dev->sync = 0;
                disposition = INPUT_PASS_TO_HANDLERS;
                break;
            }
            break;
        ……

        case EV_ABS:
            if (is_event_supported(code, dev->absbit, ABS_MAX)) {
            if (test_bit(code, input_abs_bypass)) {
                            disposition = INPUT_PASS_TO_HANDLERS;
                            break;
                }

                value = input_defuzz_abs_event(value,
                        dev->abs[code], dev->absfuzz[code]);

static void __init input_init_abs_bypass(void)
{
    const unsigned int *p;
    for (p = input_abs_bypass_init_data; *p; p++)
        input_abs_bypass[BIT_WORD(*p)] |= BIT_MASK(*p);
}
static int __init input_init(void)
{
    int err;
    input_init_abs_bypass();
    err = class_register(&input_class);
```

其他版本的内核的 input 子系统文件可以根据 http://www.linuxhq.com/kernel/ 上提供的文件更新比较来作出对应的修改。

8. New-style 形式驱动信息配置（probe 方式）

如果采用的是 New-style 驱动探测方式，在挂载驱动前需要预先建立在主板的 i2c_board_info 数组中添加 I2C 地址、名字、中断等相关结构信息，以便于系统自动建立 i2c_client 用于 probe，需要修改的文件为：

```
//mach-smdk6410.c
//i2c 总线 0 上需要建立的设备 client 列表，挂载在哪个适配器列表上取决于硬件配置
static struct i2c_board_info i2c_devs0[] __initdata = {
    { I2C_BOARD_INFO("24c08", 0x50), },
/*  { I2C_BOARD_INFO("WM8580", 0x1b), },      */
}
```

修改为：

```
static struct i2c_board_info i2c_devs0[] __initdata = {
    { I2C_BOARD_INFO("24c08", 0x50, },
```

```
    { I2C_BOARD_INFO("Goodix-TS", 0x5d),      //0x5d 为 I2C 从设备 7 位地址，需要根据实际情况修改
                                              //注意名字"Goodix-TS"与代码中 GOODIX_I2C_NAME 的值一致
        .irq=IRQ_EINT(18),                    }, //程序中使用的系统中断号，示例代码已在头文件中指定
    ……,                                       //需要根据实际情况修改
};
```

与此对应的是，如果采用的 adapter 方式（LEGACY）的 I2C 驱动，需要手动建立 client。请注意确保驱动中使用的 i2c 总线 i2c_board_info 数组上不存在与 GT80X 相同地址的 client。另芯片缺省从设备地址是 0x5d，如果一致，请注释掉头文件中的 NO_DEFAULT_ID 宏定义，如果需要修改成别的设备地址（还提供了两个地址，0x37 和 0x14），请必须同时打开 INT_PORT 和 NO_DEFAULT_ID 的宏定义，也就是说必须要有 INT 口才可以设置其它从设备地址。

9. Guitar 触摸屏的配置信息

Guitar 触摸屏在上电初始化的过程中，需要主控发送配置信息，配置信息共分为 106 字节（红色部分），适合不同机型按键板的配置信息由汇顶科技公司面向 windows 系统的 GuitarTestPlatform 工具自动产生，由于 IC 不同算法其配置信息也不全相同，因此需要根据读出的版本信息来发送相应的配置，具体配置信息详细内容可参阅 Guitar 的用户手册。下面以 4.3 inch 屏为例：

```
static int goodix_init_panel(struct goodix_ts_data *ts)
{
    short ret=-1;
    int ic_size = 0;
    uint8_t config_info_c[] = {          //试例
    0x06,0xA2,
    0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E,
    0x10,0x12,0xE0,0x00,0xD0,0x00,0xC0,0x00,
    0xB0,0x00,0xA0,0x00,0x90,0x00,0x80,0x00,
    0x70,0x00,0x00,0x00,0x10,0x00,0x20,0x00,
    0x30,0x00,0x40,0x00,0x50,0x00,0x60,0x00,
    0x00,0x00,0x01,0x13,0x80,0x88,0x90,0x14,
    0x15,0x40,0x0F,0x0F,0x0A,0x50,0x3C, 0x0C,
    0x00,0x00,MAX_FINGER_NUM,(TOUCH_MAX_WIDTH&0xff),(TOUCH_MAX_WIDTH>>8),(TOUCH_MAX_HEIGHT&0xff),(TOUCH_MAX_HEIGHT>>8),0x00,
    0x00,0x46,0x5A,0x00,0x00,0x00,0x00,0x03,
    0x19,0x05,0x00,0x00,0x00,0x00,0x00,0x00,
    0x20,0x10,0x00,0x04,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x38,
    0x00,0x3C,0x28,0x00,0x00,0x00,0x00,0x00,
        0x00,0x01
};
```

```
        uint8_t config_info_d[] = {          //Touch key devlop board
        0x06,0xA2,
        ……//内容略
        …….
    };
        ret = goodix_read_version(ts);
        if (ret < 0)
          return ret;

    if((ts->version&0xff) < TPD_CHIP_VERSION_D_FIRMWARE_BASE)
     {
            dev_info(&ts->client->dev," Guitar Version: C\n");
            config_info_c[57] = (config_info_c[57]&0xf7)|(INT_TRIGGER<<3); //中断方式，头文件定义
            ret=i2c_write_bytes(ts->client,config_info_c, (sizeof(config_info_c)/sizeof(config_info_c[0])));
      }
     else
       {
            dev_info(&ts->client->dev," Guitar Version: D\n");
            config_info_d[57] = (config_info_d[57]&0xf7)|(INT_TRIGGER<<3);
            ret=i2c_write_bytes(ts->client,config_info_d, (sizeof(config_info_d)/sizeof(config_info_d[0])));
        }
    if (ret < 0)
        return ret;
    msleep(10);
    return 0;
}
```

只有发送正确的配置信息才能让 Guitar 运行在最佳状态，注意不同模组的配置信息一般都不相同。发送错误的配置信息还可能导致无法读取版本号。

10. 系统资源使用

Guitar 的硬件 I2C 传输有最大速率限制（具体速率限制可以通过查询 Guitar 主控编程说明，不超过 600K），过大将通信失败，需要预设移植系统的 I2C 总线速度低于此速度；I2C 系统的上拉电阻应保证小于一定阈值，请注意 i2c 工作电压和 Guitar 的工作电压；

驱动中使用的 SHUTDOWN 管脚为 GPF3，移植时需要修改为移植芯片上的对应管脚。

系统使用的中断管脚为 GPL10（系统中断 119），移植时需要根据具体的芯片手册来修改系统中断号。

为了便于移植调试，这里可以采用编译控制宏的方式来方便地控制 GPIO 的使用：

中断不使用为例：

请将 goodix_touch.h 文件中的

……

#define INT_PORT S3C64XX_GPL(10) //Int IO port

……

删除或注释掉；

不使用中断管脚时驱动将以 16ms 的周期查询坐标。

11. 头文件需要设定的部分

A. Touch 芯片的分辨率（注意，配置信息中也要相应配置）与屏的像素。头文件中的 TOUCH_MAX_HEIGHT 和

TOUCH_MAX_WIDTH，如未要求我司修改，默认分别是 7680 和 5120。一般屏的长边对应 TOUCH_MAX_HEIGHT，短边对应 TOUCH_MAX_WIDTH，请根据实际屏的像素的大小设置，比如 800×600 的像素，TOUCH_MAX_HEIGHT 设为 800，TOUCH_MAX_WIDTH 设为 600。

B. 上报坐标方式设置。如果采用轮询上报坐标，请注释#define INT_PORT 的定义。如果采用中断上报坐

标，需要设置中断是上升沿触发还是下降沿触发（注意，配置信息中也要相应配置），INT_TRIGGER 定义为 1 代表上升沿触发，为 0 代表下降沿触发。

C. 手指数的设置（注意，配置信息中也要相应配置）。MAX_FINGER_NUM 定义为 5 代表 5 指，本芯片最

大支持 5 指。

D. 按键的设置（注意，配置信息中也要相应配置）。头文件中需要定义 HAVE_TOUCH_KEY，而 touch_key_array 数组就定义了相应的按键，按键的顺序通过调整其在数组元素的位置。最多支持 4 个按键。

12、 上报坐标

由于每个平台定义的 X，Y 有可能不同，如果发现 X，Y 方向是反的，可以将程序中 goodix_ts_work_func 函数中

```
input_report_abs(ts->input_dev, ABS_MT_POSITION_X,
(*(coor_point+3*(position-1)+1)));

input_report_abs(ts->input_dev, ABS_MT_POSITION_Y,
SCREEN_MAX_WIDTH-(*(coor_point+3*(position-1))));
```

ABS_MT_POSITION_X 和 ABS_MT_POSITION_Y 位置对调

13、在线升级

为了让客户可以在不拆机的情况下升级 guitar 以获得最好的工作状态，因此在驱动中加了支持固件升级模块，该模块支持上电自动检测版本并升级最新版本方式。GOODIX 提供的驱动参考代码默认自带固件升级模块，并依赖于 AUTO_UPDATE_GT818 与 CONFIG_TOUCHSCREEN_GOODIX_IAP 宏编译开关和升级数据 gt818_update.h（文件数组大小为 3824 byte 错误的数据文件将导致 IC 无法正常工作）。

```
#define AUTO_UPDATE_GT818          //启动开机自动检测升级功能

#define CONFIG_TOUCHSCREEN_GOODIX_IAP    //开启 APK 升级接口
```

若开了自动检测升级模式，则上电执行 probe 时会调用函数 gt818_downloader，根据当前 IC 版本是否比可升级版本高来决定是否需要升级，如果当前版本比可升级版本高，则不会升级，否则会自动升级为最新程序；

Probe 中自动升级入口

```
#ifdef AUTO_UPDATE_GT818
    goodix_read_version(ts);
    ret = gt818_downloader( ts, goodix_gt818_firmware, update_path);
    if(ret < 0)
    {
            dev_err(&client->dev, "Warnning: GT818 update might be ERROR!\n");
    }
#endif
```

APK 升级读写节点：

```
#ifdef CONFIG_TOUCHSCREEN_GOODIX_IAP
    goodix_proc_entry = create_proc_entry("goodix-update", 0666, NULL);
    if(goodix_proc_entry == NULL)
    {
        dev_info(&client->dev, "Couldn't create proc entry!\n");
        ret = -ENOMEM;
        goto err_create_proc_entry;
    }
    else
    {
        dev_info(&client->dev, "Create proc entry success!\n");
        goodix_proc_entry->write_proc = goodix_update_write;
        goodix_proc_entry->read_proc = goodix_update_read;
        goodix_proc_entry->owner =THIS_MODULE;
    }
#endif
```

APK 升级方式即上层 APK 通过 goodix_update_write 发送升级数据文件路径给驱动，然后从文件中读取所需的数据存到 goodix_gt818_firmware 数组中，再调用烧录模块

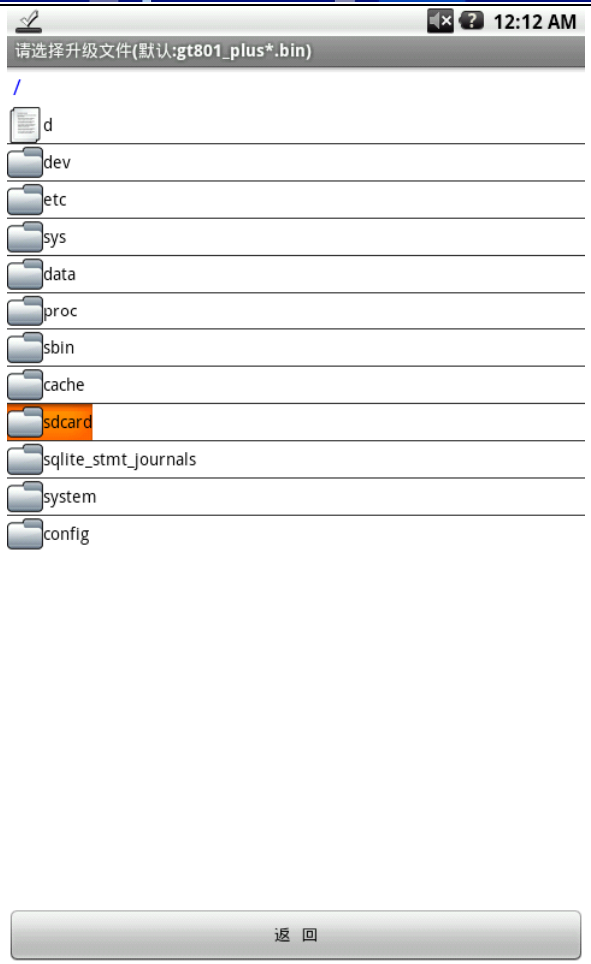
```
ret = gt818_downloader( ts, goodix_gt818_firmware, update_path);
```

进行数据烧录。

升级过程中会使用到中断脚与复位脚，具体动作请参考 818 编程指南，请在升级完后保证中断口为输入口且中断使能，以免读不到坐标。

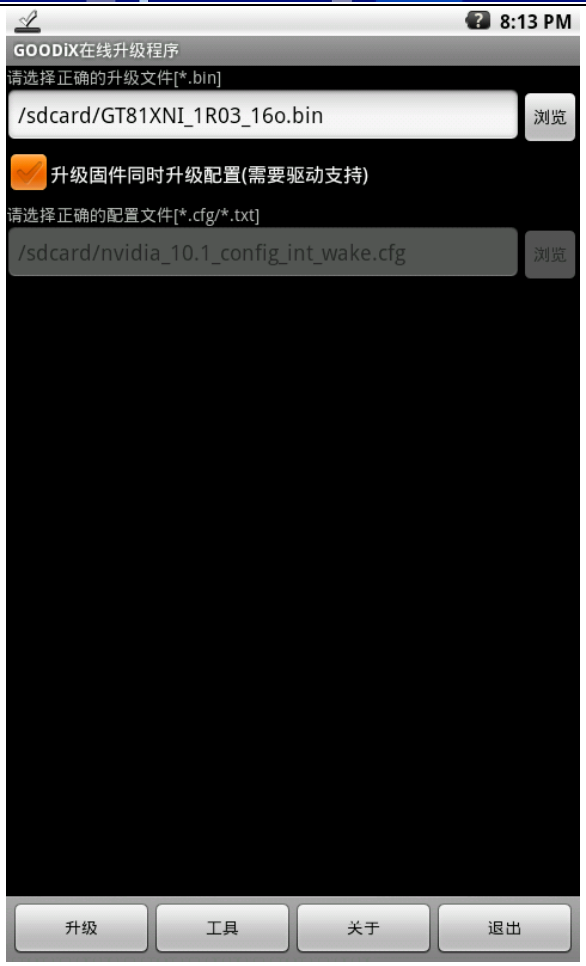
安装 GOODIX 提供的 Android 固件升级程序 GOODiX 在线升级程序.apk，程序运行后的界面如下图所示：

第一个路径栏用于选择用于升级中要使用固件文件(*.bin)，第二个路径栏用于选择升级过程中要使用的配置文件(目前暂不开放)，程序会自动保存上次升级所使用升级文件的路径，通过”浏览”按钮可以从存储器中选择升级文件相应的升级文件，选择升级文件的界面如下图所示：



进入相应文件夹选择用于升级的*.bin 文件时，程序将弹出警告提示，如果您确认该文件为 GOODIX 提供的正确的 bin 文件，请选择”强制升级”返回升级主界面，否则请”重新选择”正确的 bin 文件。（注意：错误的升级文件将导致触摸屏失效，请慎重操作！）

选择完升级文件后，升级主界面的地址栏中将显示您所选择的路径，如果升级同时不需要升级配置，则配置地址栏显示为不可用状态：



点击”升级”按钮进入升级后，将出现确认升级对话框，可以点击”是”直接进入升级，或者点击”否”返回升级主界面：

进入升级后，将在屏幕中央打印出升级过程的各个步骤，并用进度条显示升级进度，可以对比升级前后所读取的版本号，来判断升级是否成功：

如果升级失败，触摸屏将可能无法正常工作，重启系统后升级程序将自动运行，并倒计时 10s 后自动进入升级，如果是由于上次升级选择的文件错误导致触摸屏失效，请删除上次升级的文件，并将正确的升级文件保持与上次升级的文件名和路劲都相同，然后重新启动系统，让程序自动升级，如果程序找不到上次升级的文件，则会出现以下提示，此时请将升级文件按照程序的按照上次升级的路径和文件名存放，然后重启系统，程序将自动进行升级：

如果路径不对则会出现如下提示：

点击工具按键则会进入工具栏界面，在该界面可以进行配置信息的读写与读取版本号：

注意：

1、 由于该模式会与 gt818_update.h 比较软件版本，因此当开了 AUTO_UPDATE_GT818 时，只能升比当前更高的版本；
2、 升级前请确保系统电量充足，以防升级过程中出现意外；
3、 请确保升级文件为 GOODIX 提供的对应的 GT818 触摸模组的 BIN 文件或 gt818_update.h；
4、 升级结束后请把相关 APK 删除，防止误操作；

【注意事项】

1、 程序中可以设定支持的手指数目，双指情况良好；可以支持更多手指，但由于 Android 2.3 以前文件系统的原因有手指 ID 交换问题；在 Android 2.3 以后 Google 已经修改该 Bug。

主要问题：以三指为例，操作时如果先按键的手指先离开，可能发生 ID 交换或其他情况。