A10 G-Sensor 驱动设计与移植文档

V1.0

2012-7-11

Revision History

Version	Date	Section/ Page	Changes compared to previous issue
1.0	20120711		initialition

目录

1. 文档说明

本文档列出已支持的一些 gsensor，并对已支持 gsensor 在 A1x 上的移植，进行总结归纳，旨在为大家提供一份详细的移植参考，从而获得统一的移植风格，以便于同事及客户的调试使用。
（本参考设计以 A10 的 linux-3.0 中 bma250 驱动为参考，并不断进行更新补充中，由于文档不断补充，代码也不断更新，有些地方可能和实际代码中有细微差别，请注意）

2. G-Sensor 驱动在系统中的层次

gsensor 驱动在系统中的层次如图 1 所示：

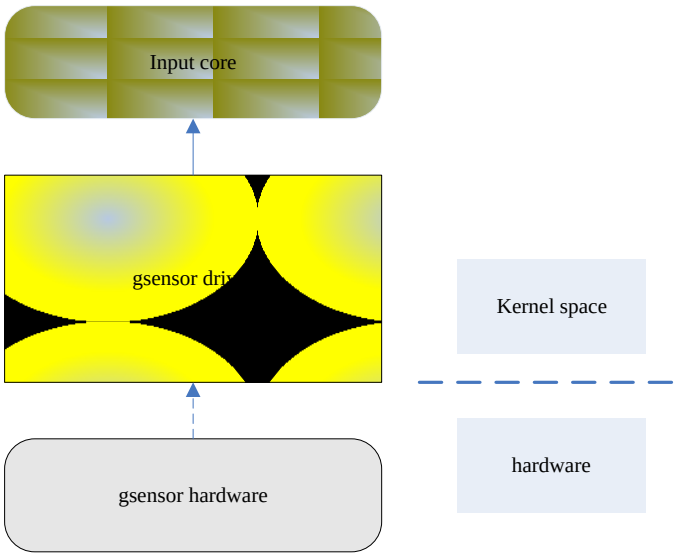


图 1 gsensor 驱动在系统中的位置

图中包含三个部分：gsensor hardware, gsensor driver, input core;

1. gsensor hardware: 是指主控，gsensor ic和gsensor 传感器构成的硬件平台，通过它可获得用户输入的原始数据；
2. gsensor driver（gsensor驱动）：是驻留于操作系统中，为gsensor hardware服务的一个内核模块；它将gsensor hardware采集到的原始数据，进行降噪，滤波，获得当前平板的空间状态，并按照操作系统的要求，将这些信息通过input core上报给操作系统。
3. Input core: 是linux为简化设备驱动程序开发，而开发的一个内核子系统；发给input core的数据将提供给操作系统使用。

实际使用时，驱动按照一定的时间间隔，通过数据总线，获取 gsensor hardware 采集到的数据，并按照操作系统的要求，将这些信息通过 input core 上报给操作系统。

3. G-Sensor 驱动工作流程

3.1 G-Sensor 驱动设计目标

在人机交互过程中，gsensor 起着举足轻重的作用，gsensor 作为输入设备，能感知当前 gsensor 传感器所处的空间状态，附着在 pad 上配合使用，能测量出 pad 在空间上的坐标状态，从而获知 pad 用户的操作意图：横竖屏切换，转弯等。

gsensor 驱动作为硬件与软件的一个桥梁，实现对 gsensor 控制器硬件初始化，获取 gsensor 控制器采集到的位置坐标信息（必要时，对数据进行滤波和用户操作意图识别），上报用户操作相关信息于操作系统，经上层系统处理后，正确响应用户的意图。

3.2 模块和系统的工作关系

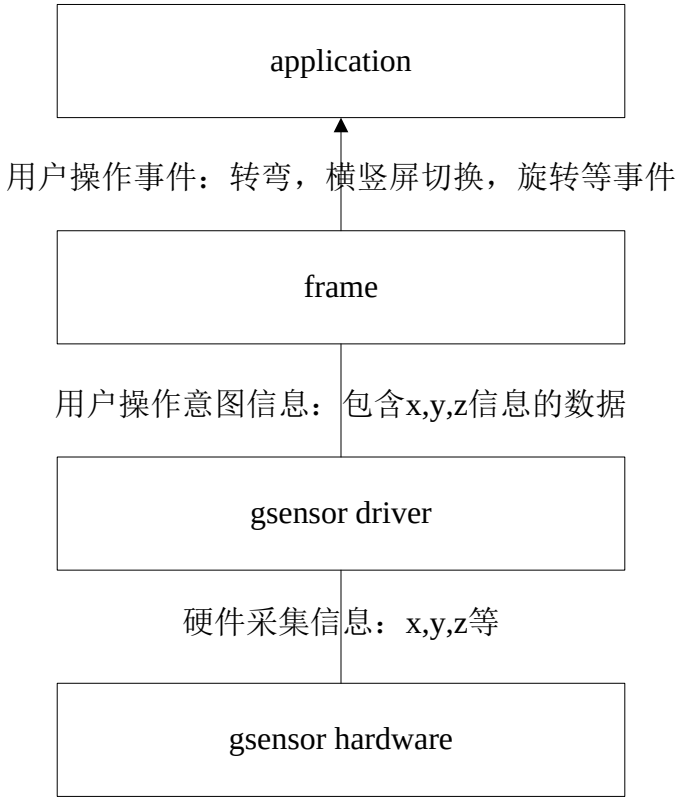


图 4 gsensor 驱动与系统的关系

图 4 为 gsensor 驱动与系统的关系，包括硬件采集、驱动层、框架层和应用层。

- 硬件采集的职能是获取 gsensor 传感器的输入，借助 fifo 暂存，最终提供给上层；
- 驱动层的职能是获取硬件信息，并完整的表达用户的操作意图，并按照上层应用或框架要求的协议上报；

- 框架层的职能是对驱动上报的数据，按照协议要求，进行解释，从而将这些数据转为转弯，旋转等事件；
- 应用层的职能是按照需要响应这些事件。

驱动和上层应用或框架，都需要真实的表达用户的操作意图，从而获得良好的用户体验；在驱动层，用户的意图，通过空间坐标信息表示；在上层应用或框架，用户的真实意图被解释为一个具体的动作，如旋转，转弯等。

3.3 G-Sensor 驱动设计规范

由 gsensor 驱动在系统中的层次，可知驱动负责采集到 gsensor 的信息，并准确及时的上报数据至 input core。驱动上报的数据，是被 input core 管理并被上层使用的，应符合 input core 和上层应用框架的要求；

- gsensor 驱动的设计必须符合整个系统的需求；
- gsensor 驱动的设计必须符合 Input 输入子系统的设计规范

3.3.1 系统对 G-Sensor 驱动的要求

- 接口：Gsensor 驱动，在设计上，不应自行决定是否上报，上报频率等，应提供接口，供上层应用控制驱动的运行和数据上报：包括使能控制 Enable， 上报时延 delay 等；通常通过 sys 文件系统提供，这部分实现，遵循标准的 linux 规范；
- 上报数据的方式：或者提供接口供上层访问（eg: ioctl），或者挂接在系统子系统上，使用系统子系统的接口，供上层使用(eg: input core)；
- 读取数据的支持：应满足读取数据的要求，进行相应的配置；本文以 i2c 总线为例，简要说明在 A1x 平台上，配置总线传输相关信息；

3.3.1.1. IIC 总线的配置

要使用 i2c 总线进行数据传输，需注册 i2c driver,创建 i2c-client，以便使用 2c-adapter 进行数据传输；要成功注册 i2c driver 有两种方式：

1. 使用 i2c_register_board_info：此方式，需要在系统启动时，进行相关信息的注，不利于模块化开发，现已不推荐；目前,在 2.6 内核上，还支持此方式，在 3． 0 上已不再支持；

2. 使用 detect 方式：在模块加载时，进行检测，在条件成立时，注册 i2c 设备相关信息，创建 i2c-client，并注册 i2c driver，执行 probe 操作;需要说明的是，此两种方式可共存，目前 2.6 就是这样的；在共存时，以 i2c_register_board_info 信息为更高优先级，在 i2c_register_board_info 已经占用设备的前提下，内核发现设备被占用，不会执行 detect, 因而不会有冲突。

3.3.2 Input 输入子系统设计规范

3.3.2.1. Input 子系统介绍

在Linux中，输入子系统是由输入子系统设备驱动层、输入子系统核心层(Input Core)和输入子系统事件处理层(Event Handler)组成。其中设备驱动层提供对硬件各寄存器的读写访问和将底层硬件的状态变化转换为标准的输入事件，再通过核心层提交给事件处理层；而核心层对下提供了设备驱动层的编程接口，对上又提供了事件处理层的编程接口；而事件处理层就为我们用户空间的应用程序提供了统一访问设备的接口以及对驱动层提交来的事件进行处理。所以这使得我们输入设备的驱动部分不在用关心对设备文件的操作，而是要关心对各硬件寄存器的操作和提交的输入事件。

Input subsys的架构图如图所示：

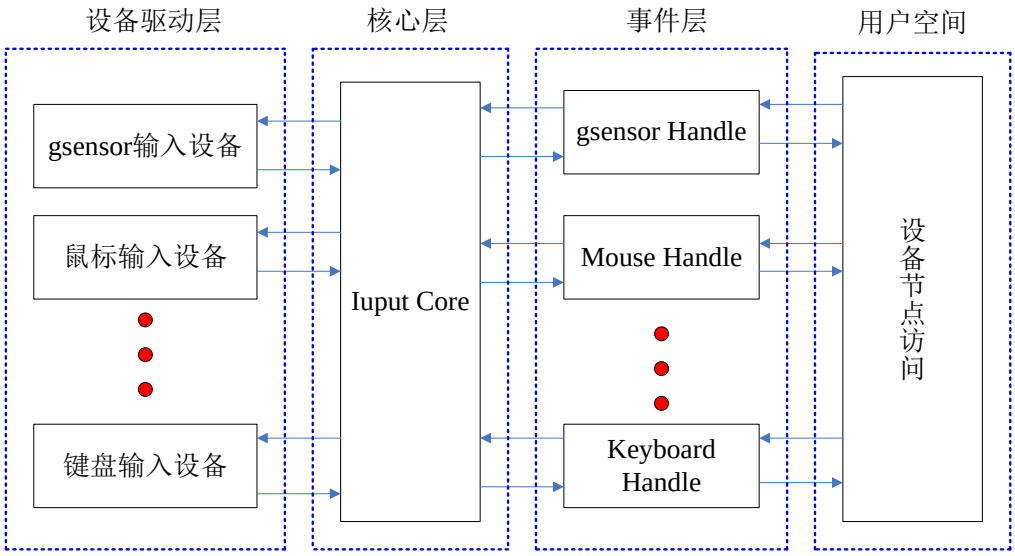


图 2 input subsystem 的架构

3.3.2.2. Input 子系统设备驱动层实现原理

在 Linux 中，Input 设备用 input_dev 结构体描述，定义在 input.h 中。设备的驱动只需按照如下步骤就可实现了。

1).在驱动模块加载函数中设置 Input 设备支持 input 子系统的哪些事件；

2).将 Input 设备注册到 input 子系统中；

3).在 Input 设备发生输入操作时(如：键盘被按下/抬起、检测到 gsensor 状态、鼠标被移动/单击/抬起时等)，提交所发生的事件及对应的键值/坐标等状态。

软件的设计流程如图 3 所示。

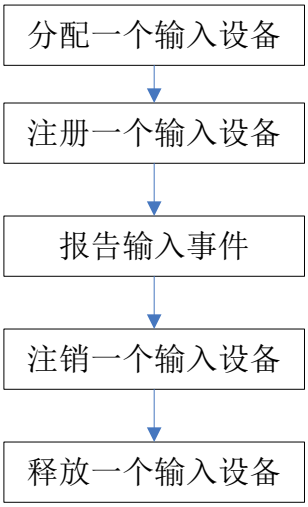


图 3 input 子系统软件设计流程

3.3.2.3. G一Sensor driver 与 input core 之间主要的接口

申请 input_dev 结构：

```
struct input_dev *input_allocate_device(void)
```

注册输入设备，并和对应的 handler 处理函数挂钩：

```
int __must_check input_register_device(struct input_dev *)
```

注销 Input 设备：

```
void input_unregister_device(struct input_dev *)
```

上报坐标值（绝对值）：

```
static inline void input_report_abs(struct input_dev *dev, unsigned int code, int value)
```

上报坐标值结束时的同步信号：

```
static inline void input_sync(struct input_dev *dev)
```

这部分接口是 linux input subsystem 对外提供的接口，gsensor driver 使用这些接口，向 input subsystem 注册设备和上报数据。

Gsensor driver 与 input core 之间主要的数据结构 input_dev,根据各种输入信号的类型来建立类型为 unsigned long 的数组，数组的每一位代表一种信号类型，内核中会对其进行置位或清位操作来表示事件的发生和被处理。此结构代表了一个输入设备，包含了一个输入设备所需要的信息，如支持的事件类型等；其具体用法可以参考 include\linux\input.h;

```
struct input_dev {
    void *private;
     const char *name;
     const char *phys;
      const char *uniq;
      struct input_id id;
      unsigned long evbit[NBITS(EV_MAX)];
      unsigned long  keybit[NBITS(KEY_MAX)];
      unsigned long relbit[NBITS(REL_MAX)];
      unsigned long absbit[NBITS(ABS_MAX)];
      unsigned long mscbit[NBITS(MSC_MAX)];
      unsigned long ledbit[NBITS(LED_MAX)];
      unsigned long sndbit[NBITS(SND_MAX)];
      unsigned long ffbit[NBITS(FF_MAX)];
      unsigned long swbit[NBITS(SW_MAX)];

   .....
};
```

3.3.2.4. Event 设备

Event 设备在用户空间大多使用 read,ioctl,poll 等文件系统的接口进行操作，read 用于读取输入信息，ioctl 用于获取和设置信息，poll 调用可以进行用户空间的阻塞，当内核有按键等中断时，通过在中断中唤醒 poll 的内核实现，这样在用户空间的 poll 调用也可以返回。

```
static const struct file_operations evdev_fops = {
        .owner =        THIS_MODULE,
        .read =         evdev_read,
        .write =        evdev_write,
        .poll =         evdev_poll,
        .open =         evdev_open,
        .release =      evdev_release,
        .unlocked_ioctl = evdev_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl = evdev_ioctl_compat,
#endif
        .fasync =       evdev_fasync,
        .flush =        evdev_flush
};
```

Event 设备在文件系统中的设备节点为:/dev/input/eventX.主设备号为13，次设备号递增生成，为64~95，各个具体的设备为 misc,gsensor,touchscreen,keyboard 等。Event 输入驱动的架构如下图 4 所示：

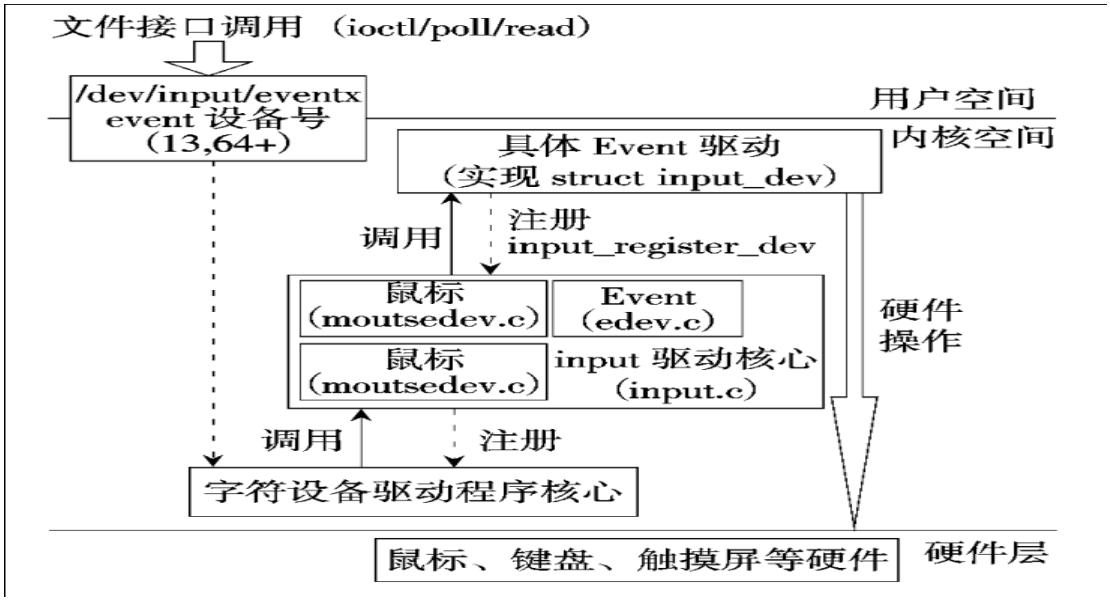


图 4 Event 输入驱动架构图

对于 Event 设备分为同步设备，键盘，相对设备(鼠标)，绝对设备(gsensor)等。Event 驱动程序通过 Input 驱动程序的统一函数进行注册：

```
int __must_check input_register_device(struct input_dev *)
```

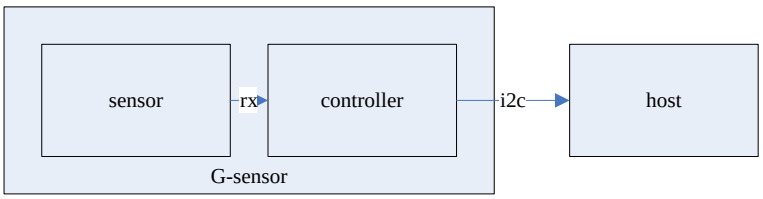
具体的 Event 驱动程序需要定义 struct input_dev 结构体，并且通过 input_register_device()函数进行注册。

3.4 G-Sensor 模块硬件说明

gsensor 硬件，负责获取 gsensor 传感器所处的空间状态信息，存放于 fifo 中，供主控使用，不同的硬件平台，数据准备好后，告知主控的方式及主控获取数据的方式略有不同。

告知主控的方式：gsensor 作为传感器，本身无法区分哪些数据是应该上报的，哪些数据是无效的，它只能接受主控的控制，以主控主动查询为主；

主控获取数据的方式：通过 ahb, i2c, spi,usb 等方式获取都是可能的。以下以一种典型的硬件连接为例，描述 gsensor 传感器，gsensor ic，主控之间的连接关系；



Gsensor在硬件上，只有i2c连接，这些连接信息，需要事先告知驱动，从而从指定的设备上读取数据；这些连接信息，通过sysconfig描述，在驱动中使用；

3.5 G-Sensor 驱动软件操作说明

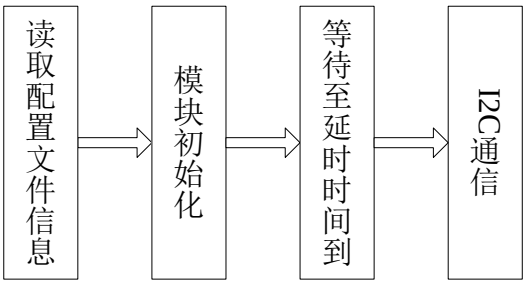


图 7 gsensor 驱动工作流程

gsensor 模块的工作流程如图 7 所示：

- 读取 sys_config1.fex 中的配置信息，设置是否启用 gsensor 模块，gsensor 使用的数据传输相关配置；
- 在启用 gsensor 模块的前提下，模块初始化中将设备注册到 Input 子系统中，
- 在使能 gsensor 的条件下，在设定时延到时后，通过 I2C 总线读取数据并上报到上层。

4. 源码结构和配置

4.1 G-Sensor 驱动源码的位置

目前，A1x 系列平台上，已支持 bma250, mma7660, mxc622x 等 gsensor，可供其他 gsensor ic 移植时参考

源码目录:

.\exdroid\lichee\linux-2.6.36\drivers\gsensor\

或者 ..\exdroid\lichee\linux-3.0\drivers\hwmon

4.2 gsensor 驱动的配置

gsensor 驱动的配置有三个部分，分别为：

- sysconfig 文件 gsensor 是否启用、gsensor 驱动的名字参数等配置；
- linux 层驱动的注册以及生成最终的目标文件；
- android 层驱动的拷贝与加载；

4.2.1. sys_config1.fex 文件配置

```
;---------------------------------------------------------------------------
; G sensor configuration
; gs_twi_id ---  TWI ID for controlling Gsensor (0: TWI0, 1: TWI1, 2: TWI2)
;---------------------------------------------------------------------------
[gsensor_para]
gsensor_used            = 1
gsensor_name            = "bma250"
gsensor_twi_id          = 1
gsensor_twi_addr        = 0x18
gsensor_int1            =
gsensor_int2            =
```

文件配置说明如下：

gsensor_used	是否使用 gsensor
gsensor_name	为驱动的名字，与相应的驱动中设置的名字对应
gsensor_twi_id	使用哪组 I2C
gsensor_twi_addr	I2C 设备地址（7 位地址）

4.2.2. Linux 层配置

添加一个新的 gsensor 驱动需要修改 Kconfig 文件和 Makefile 文件以使得能够在 menuconfig 中选中 gsensor 驱动并编译生成模块或者是直接编译进内核。现以 GT801 为例，将源码拷贝到目录: ..\exdroid\lichee\linux-2.6.36\drivers\gsensor 下，按照文件 Kconfig 与 Makefile 的形式，将 gsensor 的驱动注册进去，如下：

Kconfig 文件：

```
config SENSORS_BMA250
        tristate "BMA250 acceleration sensor support"
        depends on I2C
        help
          If you say yes here you get support for Bosch Sensortec's
          acceleration sensors BMA250.
```

Makefile 文件：

```
obj-$(CONFIG_SENSORS_BMA250)            += bma250.o
```

两个文件中设置的名字必须一致，即“CONFIG_SENSORS_BMA250”需一致，编译之后会自动的生成.Ko 文件。添加之后可以到系统中查看是否添加成功，在编译服务器上，目录为 workspace\exdroid\lichee\linux-2.6.36 上，输入命令：

```
make ARCH=arm menuconfig
```

进入目录 Device Drivers\Input device support 即可看到添加的驱动

4.2.3. Android 层配置

添加 GSENSOR 驱动模块对应的 hal 层。关于 hal 层的说明，请参考另外一份更加详细的说明文档《android 传感器框架说明》。Hal 层代码中需要注意几个编译开关选项，具体说明如下：

SW_BOARD_USES_GSENSOR_TYPE	Hal 层编译开关，表示 gsensor 类型，常见的 bma250，mm7660 等
SW_BOARD_GSENSOR_DIRECT_X	Gsensor x 轴的方向，当定义成 true 时，x 轴取正值，当定义为 false 时，x 轴取负值
SW_BOARD_GSENSOR_DIRECT_Y	Gsensor y 轴的方向，当定义成 true 时，y 轴取正值，当定义为 false 时，x 轴取负值

SW_BOARD_GSENSOR_DIRECT_Z	Gsensor z 轴的方向，当定义成 true 时，z 轴取正值，当定义为 false 时，z 轴取负值
SW_BOARD_GSENSOR_XY_REVERT	XY 轴对调，当设为 TRUE 时，x 轴变为原来 y 轴

因为最终产品电路设计可能不一样，导致 gsensor 可能贴的方向不一样，所以会出现添加 gsensor 后，用户实际使用中会出现 gsensor 方向不对的现象，比如本来是 0°放置的时候，界面却旋转 90°，这个时候就需要调整上面的几个参数中的 SW_BOARD_GSENSOR_XY_REVERT，将其设置为 true，然后再根据 x、y、z 三个轴的实际效果选择是否设置其值。

在 android 上移植 gsensor 还必须将驱动拷贝到 android 打包目录的对应文件夹中，并在 init.sunxi 中将其加载。具体修改方法如下（以 aino 为例）：

1、 android2.3.4

在 android2.3.4\device\softwinner\crane-aino\mkfs.sh 中 mkimg_root 函数中添加 cp $DRV_DIR/bma250.ko $PRODUCT_ROOT/drv/bma250.ko

2、 android4.0.x

android4.0.x 会在 extract-bsp 的时候将其拷贝到对应目录。

Android4.0.x 对应的目录已经更改为：$PRODUCT_ROOT/system/vendor/modules/

3、 修改 init.sunxi.rc 文件，android2.3.x 修改 init.sun4i.rc,android4.0.x

修改 init.sun5i.rc，在对应的文件中添加：insmod /drv/bma250.ko

5. 主要数据结构说明

5.1 gsensor 驱动主要数据结构

5.1.1 struct i2c_driver bma250_driver

struct i2c_driver bma250_driver：该变量完成 gsensor 驱动的主要工作，匹配设备名，设备的侦测等，文件操作结构体如下所示。

```
static struct i2c_driver bma250_driver = {
        .class = I2C_CLASS_HWMON,
        .driver = {
                .owner          = THIS_MODULE,
                .name           = SENSOR_NAME,
        },
        .id_table       = bma250_id,
        .probe                  = bma250_probe,
        .remove                 = bma250_remove,
        .address_list = u_i2c_addr.normal_i2c,
};
```

5.1.2 相关接口介绍

static int gsensor_fetch_sysconfig_para(void)；

static int gsensor_detect(struct i2c_client *client, struct i2c_board_info *info)；

gsensor_fetch_sysconfig_para	从 sysconfig.fex 文件中获取配置信息
gsensor_detect	对硬件进行检测，判断硬件是否存在

5.1.3 struct bma250_data

struct bma250_data :代表了 gsensor 驱动所需要的信息的集合，用于帮助实现对采样信息的处理。

```
struct bma250_data {
        struct i2c_client *bma250_client;
        atomic_t delay;
        atomic_t enable;
        unsigned char mode;
        struct input_dev *input;
        struct bma250acc value;
        struct mutex value_mutex;
        struct mutex enable_mutex;
        struct mutex mode_mutex;
        struct delayed_work work;
        struct work_struct irq_work;
#ifdef CONFIG_HAS_EARLYSUSPEND
        struct early_suspend early_suspend;
#endif
};
```

5.1.4 Delayed_work

Delayed_work 一般用来触发定时的操作，在定时时间到后，完成指定操作，通过不断的触发定时操作，实现按照一定频率的执行指定操作；

```
struct delayed_work {
        struct work_struct work;
        struct timer_list timer;
};
```

5.1.5 struct bma250acc

struct bma250acc 用于记录采样时获得的信息。

```
struct bma250acc{
    s16     x,
            y,
            z;
} ;
```

5.2 gsensor 驱动实现事件支持

Struct input_dev 中有若干成员，描述设备支持的事件，该事件的属性，从而在上层能控制数据的传递。

Linux 中输入设备的事件类型有(这里只列出了常用的一些，更多请看 linux/input.h 中)：

EV_SYN	同步事件
EV_ABS	绝对坐标
EV_MSC	其它

5.3 驱动报告事件

上报的事件类型常用类型如下所示：

ABS_X	X axis
ABS_Y	Y axis
ABS_Z	Z axis

在 gsensor 驱动中，在上报完坐标值后，因为包含多个采样值（x,y,z），需要上报一个同步信号，即 input _sync，表明完成了一次完整的数据上报。一次完整的上报过程如下：

```
input_report_abs(bma250->input, ABS_X, acc.x);
input_report_abs(bma250->input, ABS_Y, acc.y);
input_report_abs(bma250->input, ABS_Z, acc.z);
input_sync(bma250->input);
```

6. gsensor 驱动移植扩展说明

在 gsensor 方案移植过程中，所有与平台相关的，可以被其他 gsensor 驱动复用的代码，都尽量封装成函数，以便复用。

6.1 移植 Demo

移植步骤，按照 gsensor 工作的流程：加载驱动，获取数据，上报数据，分为三个部分：

- a1x 平台相关的部分；
- 获取数据部分；
- 上报数据部分

➢ a1x 平台相关的部分

Init 函数中增加读取配置文件 sysconfig 的部分，并支持 i2c detect 方式：
这部分完成根据配置文件的获取和解析，并注册 i2c driver,正确前提下，应执行 probe，继续其他数据结构的配置；
修改之前：

```
static struct i2c_driver bma250_driver = {
	.driver = {
		.owner  = THIS_MODULE,
		.name   = SENSOR_NAME,
	},
	.id_table	= bma250_id,
	.probe		= bma250_probe,
	.remove		= bma250_remove,

};

static int __init BMA250_init(void)
{
	bma_dbg("bma250: init\n");
	return i2c_add_driver(&bma250_driver);
}
```

修改之后：

```
static struct i2c_driver bma250_driver = {
        .class = I2C_CLASS_HWMON,
        .driver = {
                .owner          = THIS_MODULE,
                .name           = SENSOR_NAME,
        },
        .id_table       = bma250_id,
        .probe                  = bma250_probe,
        .remove                 = bma250_remove,
        .address_list = u_i2c_addr.normal_i2c,
};

static int __init BMA250_init(void)
{
        int ret = -1;
        bma_dbg("bma250: init\n");

        if(gsensor_fetch_sysconfig_para()){
                printk("%s: err.\n", __func__);
                return -1;
        }

        printk("%s: after fetch_sysconfig_para:  normal_i2c: 0x%hx. normal_i2c[1]: 0x%hx \n", \
        __func__, u_i2c_addr.normal_i2c[0], u_i2c_addr.normal_i2c[1]);

        bma250_driver.detect = gsensor_detect;

        ret = i2c_add_driver(&bma250_driver);

        return ret;
}
```

还有部分函数存在调用上的细微差异，请参考已支持的驱动源码。

- 获取数据部分

  硬件在准备好数据后，或者通过中断告知主控，主控发起查询，或者主控主动查询；主控通常通过 i2c 等总线访问设备；

  典型的做法是使用定时器，按照一定的间隔，查询设备，这部分代码，与具体的平台无关，可复用供应商提供的源码；

- 上报数据部分

  上报数据于操作系统，通常是通过 input 子系统上报；在某些驱动中，或许会提供 ioctrl 的方式供主控获取数据，不过 android 上层最终是从 event 节点获取数据，已经成为事实上的规范，这种方式已经不被推荐。

  上报数据部分代码，与体的平台无关，可复用供应商提供的源码；

  最后，编译 gsensor 驱动，结合相应的硬件进行调试。

6.2 G-Sensor 驱动调试步骤

1）确保硬件各个管脚的连接顺序正确；

2）上电，测试各个管脚信号的电压正常，只有在保证硬件正常的情况下，进行软件驱动调试，方可保证驱动能够正常工作（该处最容易被很多软件开发人员忽视，务必注意，方可节省大部分时间）

3）将串口打印信息打开，串口打印信息设置：在打包工具中的 crane-win-v2\wboot\bootfs\linux 目录下的 params 和 paramsr 两个文件中的语句的最后加入 loglevel=9 即可。gsensor 驱动中所有的打印信息打开，查看驱动程序的配置信息读取状态以及 I2C 的初始化状态。

4）查看 probe 是否成功，如 probe 不成功，根据打印信息定位驱动的运行情况，是因为什么原因导致失败。

5）当 probe 成功之后，gsensor 没反应，查看打印信息，是否 enable，确保 enable。

6）查看 i2c 通信状态，当串口打印信息显示 i2C 通信失败时，主要有以下两个原因，一是硬件上的，各个信号线接触不良，所以出现通信失败时，检查各引脚接触情况和电压情况。二是因为 I2C 的地址不正确导致，因为 i2C 地址为 7 位地址，所以可能是因为在配置的时候没有移位或者是触控 IC 有多个 I2C 地址，导致地址不匹配。在已知 i2C 地址的情况下，可以通过尝试的方法，进行 I2C 地址的匹配；在不知道 I2C 地址的情况下，可以通过扫描的方法查看在哪一个地址时，有应答，即可知道 I2C 通信地址，在将正确的地址填写 sysconfig 配置文件中即扫描 i2c 地址的示例代码如下所示：

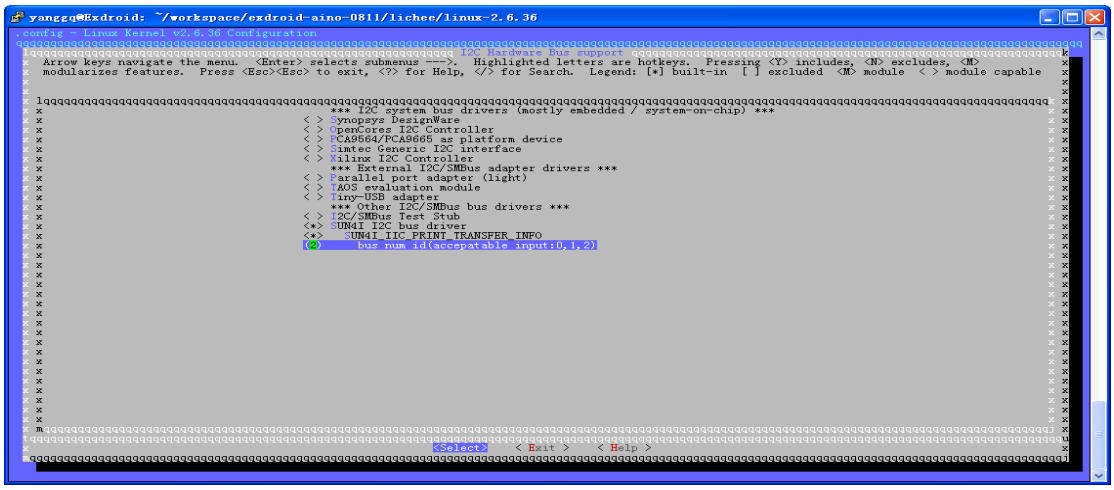
```
static int goodix_iic_test(struct i2c_client * client)
{
      struct i2c_msg msg;
      int ret=-1;
      uint8_t data[0];
      int i;
      for(i =0; i<256;i++)
      {
      msg.flags = !I2C_M_RD;//写消息
      msg.addr = i;
      msg.len = 1;
      msg.buf = data;

      ret=i2c_transfer(client->adapter, &msg,1);
      if(ret == 1)
      {
        printk("IIC TEST OK addr = %x\n",i);
        break;
      }
      mdelay(1000);
   }
      return ret;
}
```

若以上两种方法都不能正确进行 i2c 传输，则打开 i2c 传输打印，查看传输打印状态， 在编译服务器上，目录为 workspace\exdroid\lichee\linux-2.6.36 上，输入命令：

```
make ARCH=arm menuconfig
```

选择 Device Drivers->I2C support->I2C Hardware Bus support->SUN4I_IIC_PRINT_TRANSFER_INFO，输入 Y 进入 bus num id(accepatable input:0,1,2)(new),输入数值，，若希望打印信息，数值对应相应的 IIC

号，gsensorIC 用的是第二组，因此选择数值为 2，若不希望打印信息，输入 N 退出保存即可。进行修改后，需要重新编译打包之后才能生效。



7）可使用 adb 工具查看驱动是否加载以及 gsensor 是否有反应，adb 工具需要安装设备对应的驱动。使用 adb shell 工具查看驱动是否存在于机器中以及驱动是否已经加载，以及 gsensor 之后是否有反应。同时可以作为简单的调试工具，修改好的驱动 PUSH 到机器中，重启系统之后即可运行新的驱动，不用重新打包（配置文件内容除外）。使用到的命令如下所示：

```
adb shell   登录设备的shell
adb push xx.ko /drv     将触摸驱动通过adb工具push到机器中
cd   /drv  进入drv目录
ls   *.ko  查看机器中已经有了那些驱动
lsmod   查看系统中已经加载了那些模块
rmmod  ** 卸载驱动（注：不用加后缀）
insmod  **.ko  加载驱动
getevent  查看系统中已经注册了那些input设备(当触摸有效时，触摸屏幕，会有相应的打印信息)
```

8）使用 adb 工具时，push 失败，是因为文件没有可写的权限，将系统设置为可写即可。

v1.6 版本的 Android 处于安全性考虑，将 system, root 文件系统挂接成只读文件系统，开发人员在开发过程中，可以重新 remount 成读写文件系统。

1. 运行时

```
adb shell
mount -o remount,rw /dev/block/nandc /system
mount -o remount,rw /dev/root /
```

2. 编译时修改：

去掉 init.sun4i.rc 2 句话

```
mount ext4 /dev/block/nandc /system ro remount
mount rootfs rootfs / ro remount
```

6.3 现有驱动的使用

现已 mma7660 的使用为例，说明使用 gsensor 驱动的注意事项。

- sysconfig 注意事项
  i2c_dev_name: 用于 i2c 驱动的匹配，在 sysconfig 中指定，应为"mma7660"
- android 层注意事项
  input_dev_name: 用于上层从/dev/input/event*节点获取数据，驱动中已指定为"mma7660"，需 android 层配合使用；
- 获取数据
  请先确保加载驱动： lsmod 察看；
  若加载驱动后，getevent 没有得到数据，或者从对应 event 节点没有得到数据；请先执行以下操作：
  cd /sys/class/input/input3
  echo 1 > enable

Declaration