全志科技 Allwinner Technology

Tina

I2C 驱动开发说明书 v1.0

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2016/8/19		初始版本

目　录

1. 概述

1.1. 编写目的

介绍 Linux 内核中 I2C 子系统的接口及使用方法，为 I2C 设备驱动的开发提供参考。

1.2. 适用范围

适用于 R6/R16/R18/R40/F35/R30 硬件平台。

1.3. 相关人员

公司开发人员、客户。

2. 模块介绍

2.1. 模块功能介绍

Linux 中 I2C 体系结构图 2.1 所示，

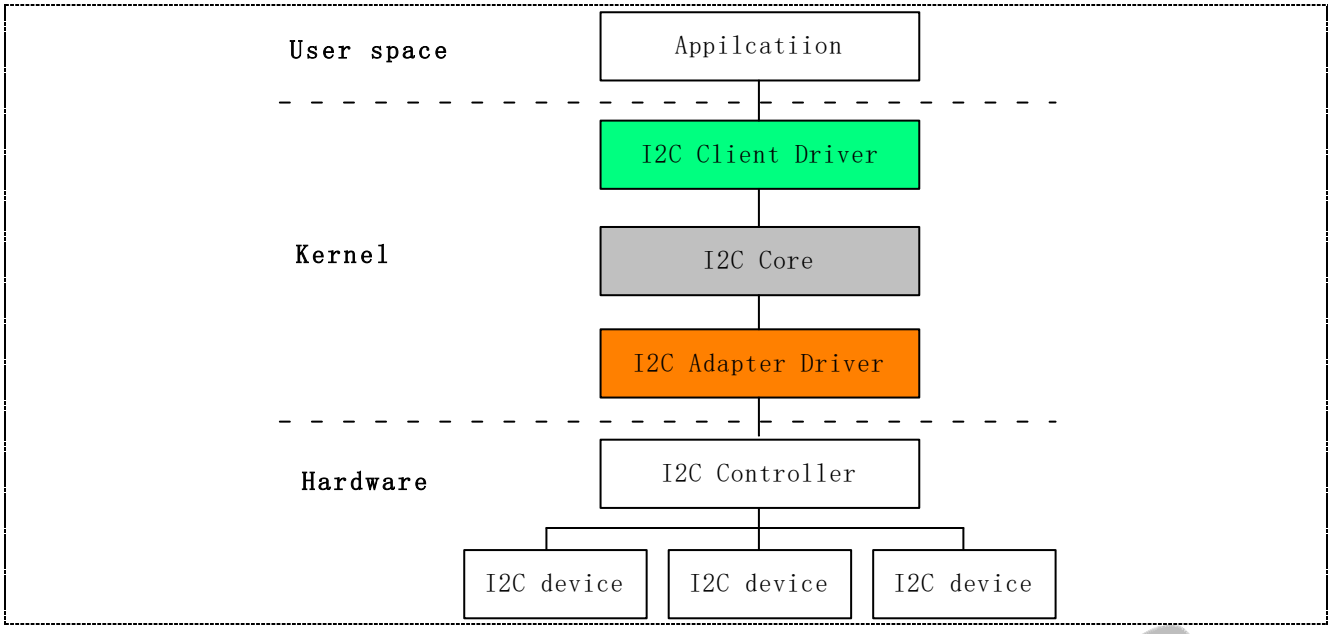

图 2.1 Linux I2C 体系结构图

图中用分割线分成了三个层次：

1．用户空间，包括所有使用 I2C 设备的应用程序；
2．内核，也就是驱动部分；
3．硬件，指实际物理设备，包括了 I2C 控制器和 I2C 外设。

其中，Linux 内核中的 I2C 驱动程序从逻辑上又可以分为 3 个部分：

1．I2C 核心（I2C Core）：实现对 I2C 总线驱动及 I2C 设备驱动的管理；
2．I2C 总线驱动（I2C adapter driver）：针对不同类型的 I2C 控制器，实现对 I2C 总线访问的具体方法；
3．I2C 设备驱动（I2C client driver）：针对特定的 I2C 设备，实现具体的功能，包括 read，write 以及 ioctl 等对用户层操作的接口。

I2C 总线驱动主要实现了适用于特定 I2C 控制器的总线读写方法，并注册到 Linux 内核的 I2C 架构，I2C 外设就可以通过 I2C 架构完成设备和总线的适配。但是总线驱动本身并不会进行任何的通讯，它只是提供通讯的实现，等待设备驱动来调用其函数。

I2C Core 的管理正好屏蔽了 I2C 总线驱动的差异，使得 I2C 设备驱动可以忽略各种总线控制器的不同，不用考虑其如何与硬件设备通讯的细节。

2.2. 相关术语介绍

术语	介绍
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
I2C	Inter-Integrated Circuit，用于 CPU 与外设通信的一种串行总线。
TWI	Normal Two Wire Interface，Sunxi 平台中的 I2C 控制器名称。
I2C Adapter	I2C Core 将所有 I2C 控制器称作 I2C 适配器，可以理解成控制器的软件名称。
I2C Client	指 I2C 从设备。
smbus	System Management Bus，系统管理总线，基于 I2C 操作原理，是一个两线接口，通过它各设备之间以及设备与系统的其他部分之间可以互相通信。

3. 模块配置

3.1. sys_config.fex 配置说明

在不同的 Sunxi 硬件平台中，TWI 控制器的数目也不同，但对于每一个 TWI 控制器来说，在 sys_config.fex 中配置参数相似，如下：

```
[twi0]
twi0_used                       = 1
twi0_scl              = port:PB00<2><default><default><default>
twi0_sda               = port:PB01<2><default><default><default>
```

其中，twi0_used 置为 1 表示使能，0 表示不使能；twi0_scl 和 twi0_sda 用于配置相应的 GPIO。

对于 I2C 设备，可以把设备节点填充作为相应 TWI 控制器的子节点。TWI 控制器驱动的 probe 函数透过 of_i2c_register_devices()，自动展开作为其子节点的 I2C 设备。

```
[twi0/twi_board0]
compatible             = "atmel,24c16";
reg                     = 0x50;
```

其中，twi0/twi_board0：表示挂在总线 twi0 下的设备 twi_board0；compatible 表征具体的设备，用于驱动和设备的绑定；reg 表示设备使用的地址。

3.2. menuconfig 配置说明

在命令行中进入内核根目录，执行 make kernel_menuconfig 进入配置主界面，并按以下配置路径操作：

```
Device Drivers
 └──>I2C support
        └──>I2C HardWare Bus support
               └──>SUNXI I2C controller
```

首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

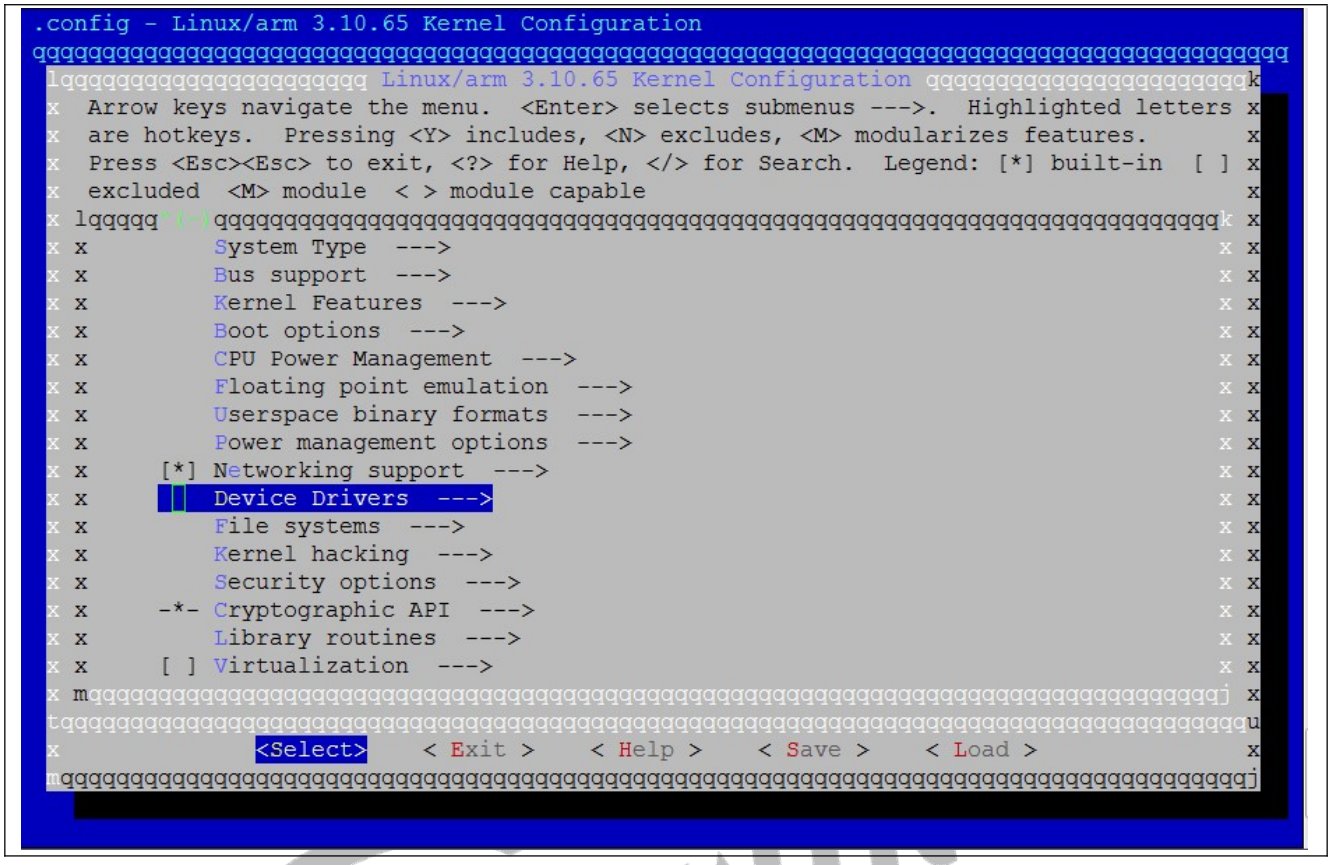

图 2.2 Device Drivers 选项配置

然后，选择 I2C support 选项，进入下一级配置，如下图所示：

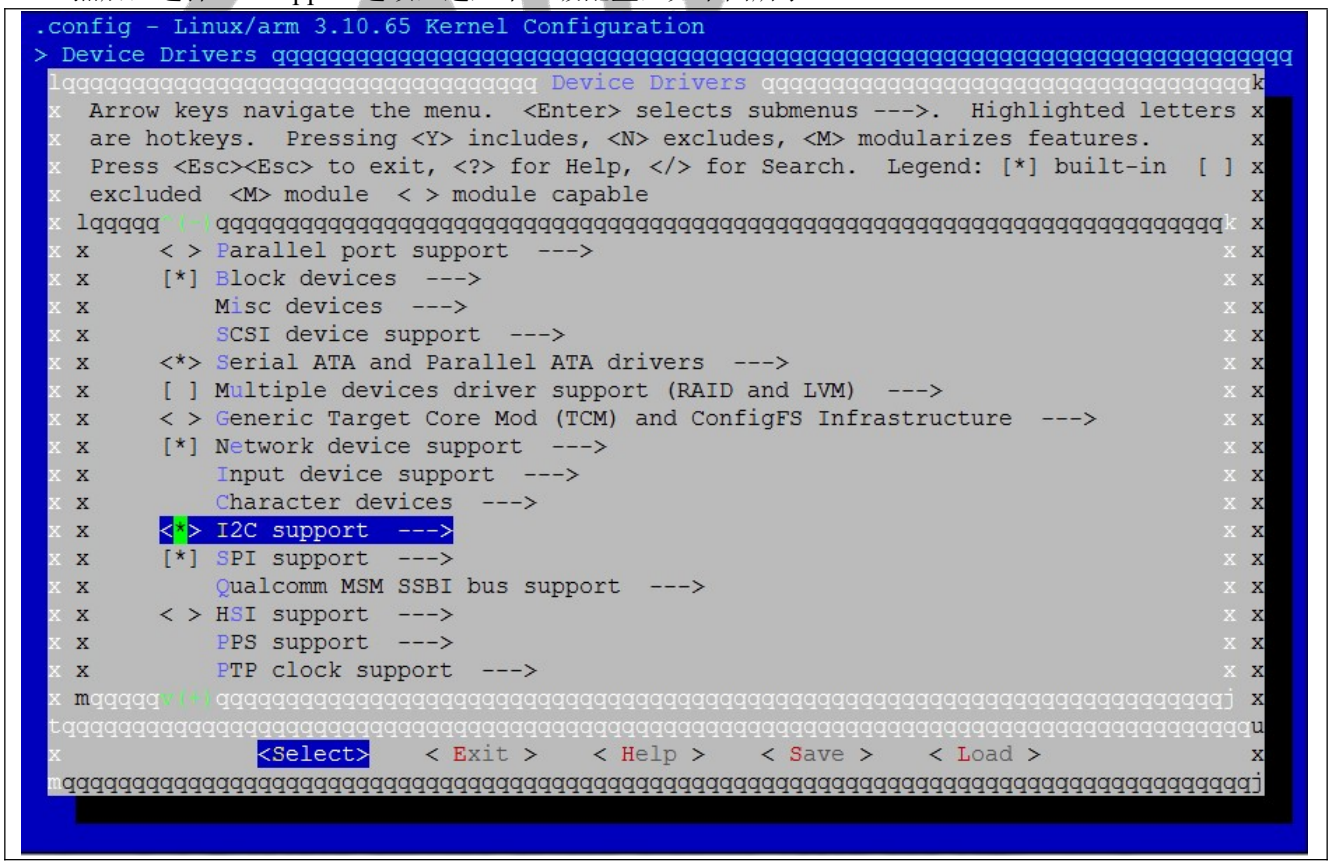

图 2.3 I2C support 选项配置

接着，选择 I2C HardWare Bus support 选项，进入下一级配置，如下图：

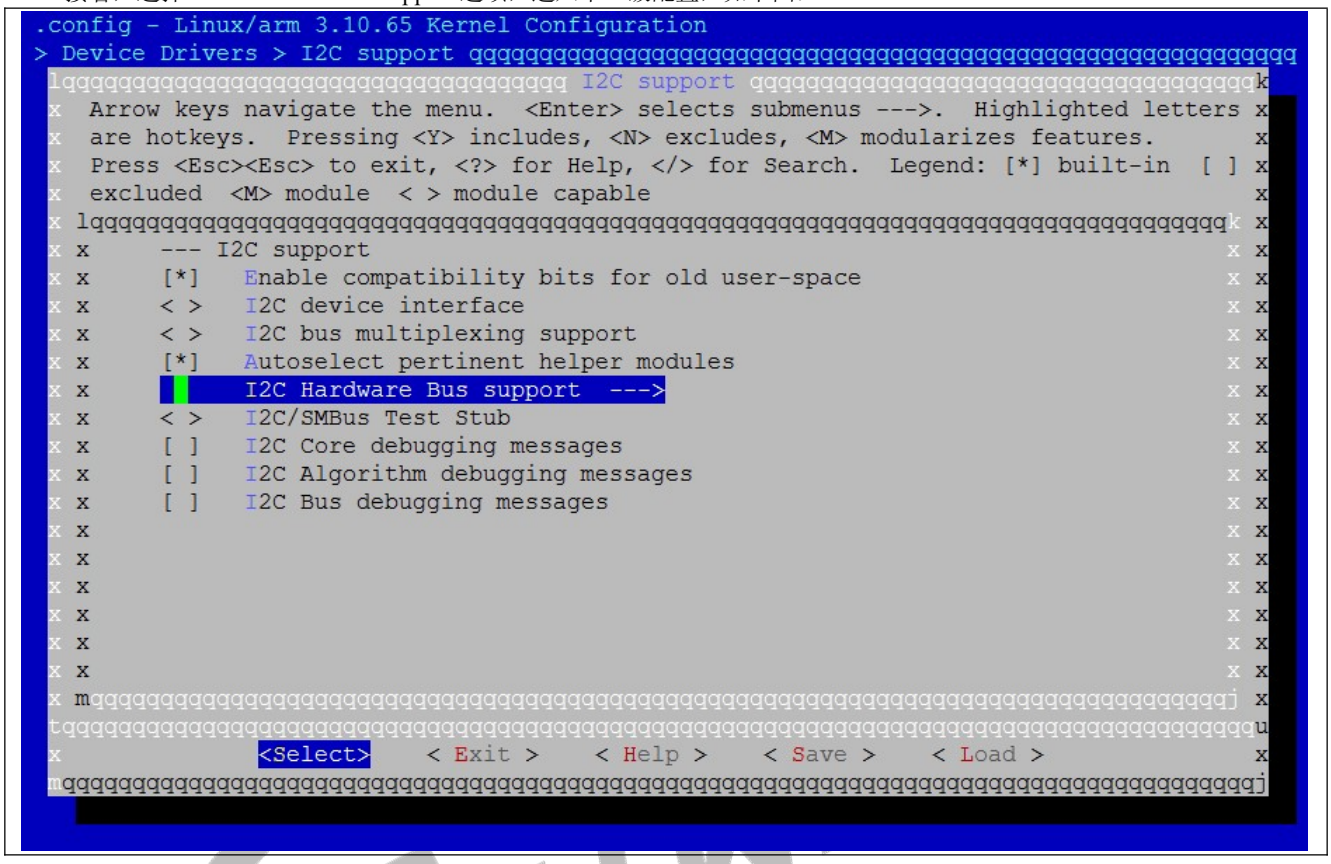

图 2.4 I2C HardWare Bus support 选项配置

最后，选择 SUNXI I2C controller 选项，可选择<*>表示直接编译进内核，也可以选择<M>表示编译成模块。如下图所示：

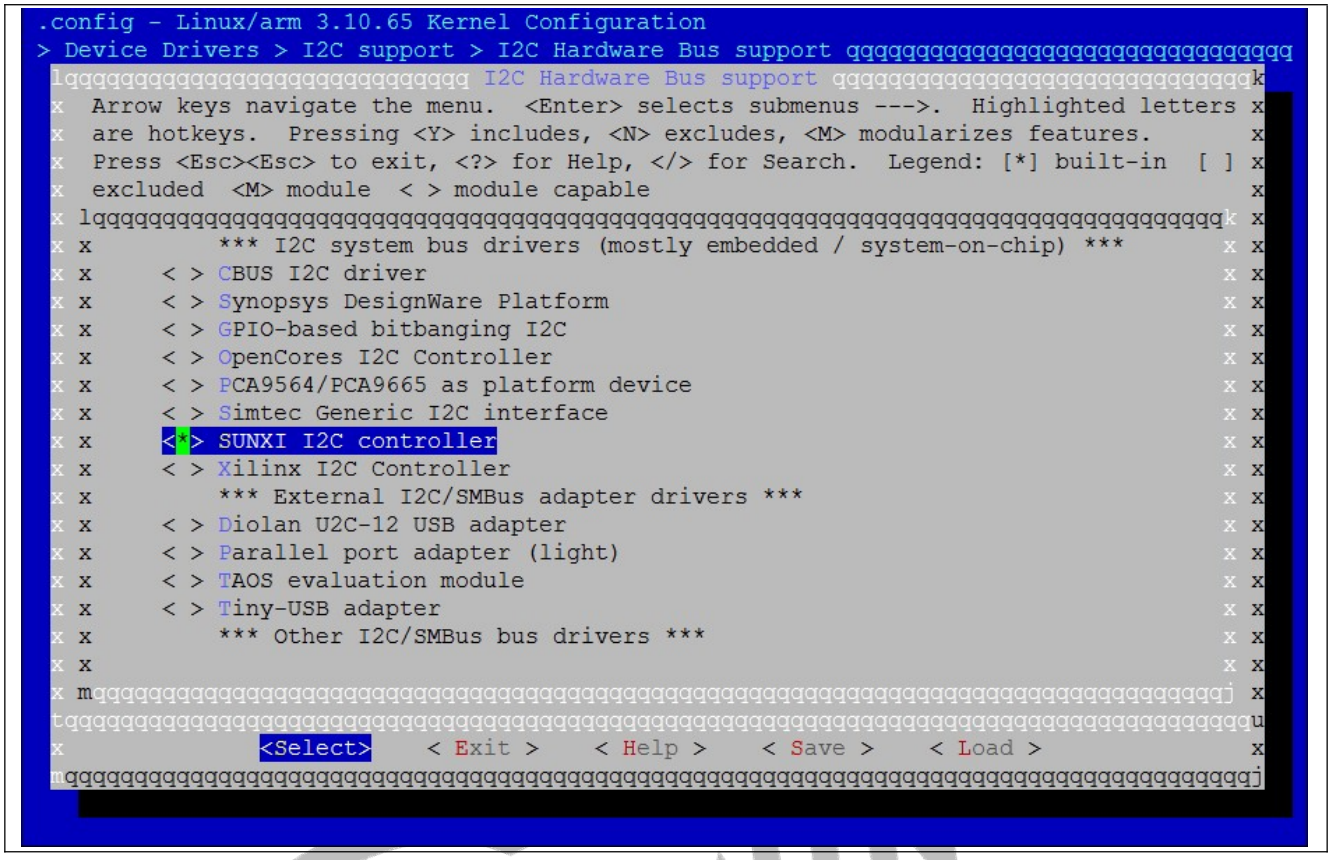

图 2.5 SUNXI I2C controller 选项配置

3.3. 源码结构介绍

I2C 总线驱动的源代码位于内核在 linux-xxx/drivers/i2c/busses 目录下（linux-xxx 中“xxx”表示内核版本，如 linux-3.10）：

```
├── busses
│   ├── i2c-sunxi.c          // I2C 控制器驱动代码
│   ├── i2c-sunxi.h          // I2C 控制器驱动定义了一些宏、数据结构
```

4. 接口描述

4.1. 设备注册接口

定义在 linux-xxx\include\linux\i2c.h。

4.1.1. i2c_add_driver

类别	介绍
函数原型	#define i2c_add_driver(driver) i2c_register_driver(THIS_MODULE, driver) int i2c_register_driver(struct module *owner, struct i2c_driver *driver)
参数	driver，i2c_driver 类型的指针，其中包含了 I2C 设备的名称、probe、detect 等接口信息。
返回	成功：0 失败：其它
功能描述	注册一个 I2C 设备驱动。从代码可以看 i2c_add_driver 是一个宏，由函数 i2c_register_driver 实现。

其中，结构 i2c_driver 的定义如下：

```
struct i2c_driver {
        unsigned int class;
        /* Notifies the driver that a new bus has appeared or is about to be
         * removed. You should avoid using this, it will be removed in a
         * near future.
         */
        int (*attach_adapter)(struct i2c_adapter *) __deprecated;
        int (*detach_adapter)(struct i2c_adapter *) __deprecated;

        /* Standard driver model interfaces */
        int (*probe)(struct i2c_client *, const struct i2c_device_id *);
        int (*remove)(struct i2c_client *);
        /* driver model interfaces that don't relate to enumeration  */
        void (*shutdown)(struct i2c_client *);
        int (*suspend)(struct i2c_client *, pm_message_t mesg);
        int (*resume)(struct i2c_client *);
        /* Alert callback, for example for the SMBus alert protocol.
         * The format and meaning of the data value depends on the protocol.
         * For the SMBus alert protocol, there is a single bit of data passed
         * as the alert response's low bit ("event flag").
         */
        void (*alert)(struct i2c_client *, unsigned int data);
        /* a ioctl like command that can be used to perform specific functions
         * with the device.
         */
        int (*command)(struct i2c_client *client, unsigned int cmd, void *arg);
        struct device_driver driver;
        const struct i2c_device_id *id_table;

        /* Device detection callback for automatic device creation */
        int (*detect)(struct i2c_client *, struct i2c_board_info *);
        const unsigned short *address_list;
        struct list_head clients;
};
```

I2C 设备驱动可能支持多种型号的设备，可以在.id_table 中给出所有支持的设备信息。

4.1.2. i2c_del_driver

类别	介绍
函数原型	void i2c_del_driver(struct i2c_driver *driver)
参数	driver，i2c_driver 类型的指针，包含有待卸载的 I2C 驱动信息
返回	无
功能描述	注销一个 I2C 设备驱动。

i2c.h 中还给出了快速注册的 I2C 设备驱动的宏：module_i2c_driver()，定义如下：

```
#define module_i2c_driver(__i2c_driver)     module_driver(__i2c_driver, i2c_add_driver,
i2c_del_driver)
```

4.1.3. i2c_register_board_info

类别	介绍
函数原型	int i2c_register_board_info(int busnum, struct i2c_board_info const *info, unsigned n)
参数	busnum，I2C 控制器编号；info，提供 I2C 设备名称、I2C 设备地址信息；n，要注册的 I2C 设备个数。
返回	成功：0 失败：其它
功能描述	向某个 I2C 总线注册 I2C 设备信息，I2C 子系统通过此接口保存 I2C 总线和 I2C 设备的适配关系。

注意，注册 I2C 设备信息的方式除了 i2c_register_board_info()，还可以通过 I2C 设备驱动的 detect 接口实现，此接口会在 I2C 子系统注册一个 I2C adapter（即 I2C 控制器）或注册一个 I2C 设备驱动时调用。

4.2. 数据传输接口

I2C 设备驱动使用"struct i2c_msg"向 I2C 总线请求读写 I/O。

一个 i2c_msg 中包含了一个 I2C 操作，通过调用 i2c_transfer 接口触发 I2C 总线的数据收发。i2c_transfer 支持多个 i2c_msg，处理时按串行的顺序依次执行。

i2c_msg 的定义也在 i2c.h 中：

```
struct i2c_msg {
        __u16 addr;     /* slave address                        */
        __u16 flags;
#define I2C_M_TEN                       0x0010   /* this is a ten bit chip address */
#define I2C_M_RD                        0x0001   /* read data, from slave to master */
#define I2C_M_NOSTART                   0x4000      /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_REV_DIR_ADDR              0x2000   /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_IGNORE_NAK                0x1000      /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_NO_RD_ACK                 0x0800      /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_RECV_LEN                  0x0400   /* length will be first received byte */
        __u16 len;                              /* msg length                   */
        __u8 *buf;                      /* pointer to msg data                  */
};
```

4.2.1. i2c_transfer

类别	介绍
函数原型	int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num)
参数	client，指向当前 I2C 设备的实例；buf，用于保存接收到的数据缓存；count，数据缓存 buf 的长度。
返回	>0：已经处理的 msg 个数 <0：失败
功能描述	完成 I2C 总线和 I2C 设备之间的一定数目的 I2C message 交互。

4.2.2. i2c_master_recv

类别	介绍
函数原型	int i2c_master_recv(const struct i2c_client *client, char *buf, int count)
参数	client，指向当前 I2C 设备的实例；buf，用于保存接收到的数据缓存；count，数据缓存 buf 的长度。
返回	>0：成功接收的字节数 <0：失败
功能描述	通过封装 i2c_transfer 完成一次 I2c 接收操作。

4.2.3. i2c_master_send

类别	介绍
函数原型	int i2c_master_send(const struct i2c_client *client, char *buf, int count)
参数	client，指向当前 I2C 从设备的实例；buf，要发送的数据；count，要发送的数据长度。
返回	>0：成功发送的字节数 <0：失败
功能描述	通过封装 i2c_transfer 完成一次 I2c 发送操作。

4.2.4. i2c_smbus_read_byte

类别	介绍
函数原型	s32 i2c_smbus_read_byte(const struct i2c_client *client)
参数	client，指向当前 I2C 从设备。
返回	>0：读到的数据 <0：失败
功能描述	从 I2C 总线读取一个字节。（内部是通过 i2c_transfer 实现，以下几个接口同。）

4.2.5. i2c_smbus_write_byte

类别	介绍
函数原型	s32 i2c_smbus_read_byte(const struct i2c_client *client, u8 value)
参数	client，指向当前 I2C 从设备；value，要写入的数值。
返回	0：成功 <0：失败

功能描述	从 I2C 总线写入一个字节。

4.2.6. i2c_smbus_read_byte_data

类别	介绍
函数原型	s32 i2c_smbus_read_byte_data(const struct i2c_client *client, u8 command)
参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）。
返回	>0：读到的数据 <0：失败
功能描述	从 I2C 设备指定偏移处读取一个字节。

4.2.7. i2c_smbus_write_byte_data

类别	介绍
函数原型	s32 i2c_smbus_write_byte_data(const struct i2c_client *client, u8 command, u8 value)
参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）；value，要写入的数值。
返回	0：成功 <0：失败
功能描述	从 I2C 设备指定偏移处写入一个字节。

4.2.8. i2c_smbus_read_word_data

类别	介绍
函数原型	s32 i2c_smbus_read_word_data(const struct i2c_client *client, u8 command)
参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）。
返回	>0：读到的数据 <0：失败
功能描述	从 I2C 设备指定偏移处读取一个 word 数据（两个字节，适用于 I2C 设备寄存器是 16 位的情况）。

4.2.9. i2c_smbus_write_word_data

类别	介绍
函数原型	s32 i2c_smbus_write_word_data(const struct i2c_client *client, u8 command, u16 value)
参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）；value，要写入的数值。
返回	0：成功 <0：失败
功能描述	从 I2C 设备指定偏移处写入一个 word 数据。

4.2.10. i2c_smbus_read_block_data

类别	介绍
函数原型	s32 i2c_smbus_read_block_data(const struct i2c_client *client, u8 command, u8 *values)

参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）；value，用于保存读取到的数据。
返回	>0：读取到的数据长度 <0：失败
功能描述	从 I2C 设备指定偏移处读取一块数据。

4.2.11. i2c_smbus_write_block_data

类别	介绍
函数原型	s32 i2c_smbus_write_block_data(const struct i2c_client *client, u8 command, u8 length, const u8 *values)
参数	client，指向当前 I2C 从设备；command，I2C 协议数据的第 0 字节命令码（即偏移值）；length，要写入的数据长度；value，要写入的数据。
返回	0：成功 <0：失败
功能描述	从 I2C 设备指定偏移处写入一块数据（长度最大 32 字节）。

5. Demo

5.1. drivers\hwmon\bma250.c

此 I2C 设备是一个 Gsensor，使用 detect 方式完成 I2C 设备和 I2C 总线的适配，代码如下，主要目的是完成 info->type 的赋值。

```
static int gsensor_detect(struct i2c_client *client, struct i2c_board_info *info)
{
        struct i2c_adapter *adapter = client->adapter;
        int ret;
        …
      if (twi_id == adapter->nr) {
        for (i2c_num = 0; i2c_num < (sizeof(i2c_address)/sizeof(i2c_address[0]));i2c_num++) {
                client->addr = i2c_address[i2c_num];
                pr_info("%s:addr= 0x%x,i2c_num:%d\n",__func__,client->addr,i2c_num);
                ret= i2c_smbus_read_byte_data(client,BMA250_CHIP_ID_REG);
                pr_info("Read ID value is :%d",ret);
                if ((ret &0x00FF) == BMA250_CHIP_ID) {
                        pr_info("Bosch Sensortec Device detected!\n" );
                        strlcpy(info->type, SENSOR_NAME, I2C_NAME_SIZE);
                        return 0;
                } else if((ret &0x00FF) == BMA150_CHIP_ID) {
```

```
                    pr_info("Bosch Sensortec Device detected!\n" \
                    "BMA150 registered I2C driver!\n");
                    strlcpy(info->type, SENSOR_NAME, I2C_NAME_SIZE);
                    return 0;
                } else if((ret &0x00FF) == BMA250E_CHIP_ID) {
                    pr_info("Bosch Sensortec Device detected!\n" \
                    "BMA250E registered I2C driver!\n");
                    strlcpy(info->type, SENSOR_NAME, I2C_NAME_SIZE);
                    return 0;
                }
            }
            pr_info("%s:Bosch Sensortec Device not found, \
            maybe the other gsensor equipment! \n",__func__);
            return -ENODEV;
        } else {
            return -ENODEV;
        }
}
```

bma250.c 将 Gsensor 注册成一个 input 设备，定时向上层报告坐标数据，这样应用层就可以采用类似鼠标键盘设备的方式读取到坐标数据。

可以看出，input 接口是 Gsensor 这个设备和上层应用的接口，而第 3 章所描述的 I2C 接口是 Gsensor 和 I2C adapter 总线控制器的接口。

在参考这个例子上，重点关注 I2C 接口的使用即可。

6. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgment to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.