

I2C最全干货- (2) 驱动篇-之基于linux的mpu6050驱动

原创 土豆居士 一口Linux 2020-06-23 22:30

收录于合集

#所有原创 194 #arm 22



本文以三星exynos4412为例讲解mpu6050陀螺仪的数据读取驱动的实现。通过本篇文章，读者可以掌握基于Linux驱动I2C编写方法。

I2C核心 (i2c_core)

I2C核心维护了i2c_bus结构体，提供了I2C总线驱动和设备驱动的注册、注销方法，维护了I2C总线的驱动、设备链表，实现了设备、驱动的匹配探测。此部分代码由Linux内核提供。

I2C总线驱动

I2C总线驱动维护了I2C适配器数据结构 (i2c_adapter) 和适配器的通信方法数据结构 (i2c_algorithm) 。所以I2C总线驱动可控制I2C适配器产生start、stop、ACK等。此部分代码由具体的芯片厂商提供，比如Samsung、高通。

I2C设备驱动

I2C设备驱动主要维护两个结构体：i2c_driver和i2c_client，实现和用户交互的文件操作集合fops、cdev等。此部分代码就是驱动开发者需要完成的。

Linux内核中描述I2C的四个核心结构体

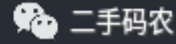
1) i2c_client—挂在I2C总线上的I2C从设备

每一个i2c从设备都需要用一个i2c_client结构体来描述，i2c_client对应真实的i2c物理设备device。

```

struct i2c_client {
    unsigned short flags;      //标志位 (读或者写)
    unsigned short addr;      //7位的设备地址 (低7位)
    char name[I2C_NAME_SIZE]; //设备的名字, 用来和i2c_driver匹配
    struct i2c_adapter *adapter; //所依附的i2c适配器, 即所属哪条i2c总线
    struct device dev;         //继承的设备结构体
    int irq;                   //设备申请的中断号
    struct list_head detected; //已经被发现的设备链表
};

```



二手码农

但是i2c_client不是我们自己写程序去创建的, 而是通过以下常用的方式自动创建的:

方法一: 分配、设置、注册i2c_board_info

方法二: 获取adapter调用i2c_new_device

方法三: 通过设备树 (devicetree) 创建

方法1和方法2通过platform创建, 这两种方法在内核3.0版本以前使用所以在这不详细介绍; 方法3是最新的方法, 3.0版本之后的内核都是通过这种方式创建的, 文章后面的案例就按方法3。

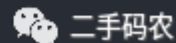
2) i2c_adapter

I2C总线适配器, 即soc中的I2C总线控制器, 硬件上每一对I2C总线都对应一个适配器来控制它。在Linux内核代码中, 每一个adapter提供了一个描述它的结构(struct i2c_adapter), 再通过i2c core层将i2c设备与i2c adapter关联起来。主要用来完成i2c总线控制器相关的数据通信, 此结构体在芯片厂商提供的代码中维护。

```

struct i2c_adapter {
    struct module *owner;
    unsigned int class;          //允许匹配的设备的类型
    const struct i2c_algorithm *algo; //指向适配器的驱动程序, 实现发送数据的算法
    struct device dev;           //指向适配器的设备结构体
    char name[48];               //适配器的名字
};

```



二手码农

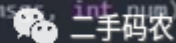
3) i2c_algorithm

I2C总线数据通信算法, 通过管理I2C总线控制器, 实现对I2C总线上数据的发送和接收等操作。亦可以理解为I2C总线控制器 (适配器adapter) 对应的驱动程序, 每一个适配器对应一个驱动程序, 用来描述适配器和设备之间的通信方法, 由芯片厂商去实现的。

```

struct i2c_algorithm {
    //传输函数指针, 指向实现IIC总线通信协议的函数
    int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg *msgs, int num);
};

```



二手码农


4) i2c_driver

用于管理I2C的驱动程序和i2c设备 (client) 的匹配探测, 实现与应用层交互的文件操作集合fops、cdev等。

```

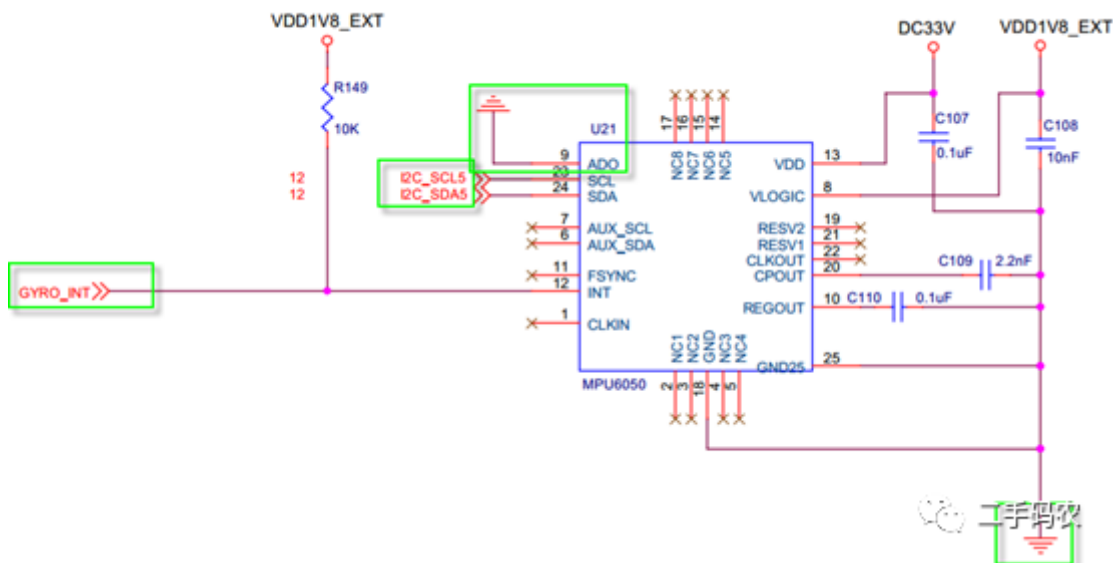
struct i2c_driver {
    /* i2c_driver与i2c_client匹配成功调用的函数 */
    int (*probe)(struct i2c_client *, const struct i2c_device_id *);
    int (*remove)(struct i2c_client *);
    struct device_driver driver;
    /* 设备的ID表, 用来匹配用platform方法创建的i2c_client */
    const struct i2c_device_id *id_table;
};

```

 二手码农

填写设备树节点信息:

硬件电路图如下:



 二手码农

I2C_SDA5 AF21 XspiMISO0/I2C_5_SDA/GPB_2
I2C_SCL5 AF21 XspiMOSI0/I2C_5_SCL/GPB_3

由上图所示硬件使用的是I2C通道5,

29.6.1 Register Map Summary

- Base Address: 0x1386_0000, 0x1387_0000, 0x1388_0000, 0x1389_0000, 0x138A_0000, 0x138B_0000, 0x138C_0000, 0x138D_0000, 0x138E_0000

 二手码农

查找exnos4412的datasheet 29.6.1节, 对应的基地址为0x138B0000.

XEINT27/KP_ROW11/ALV_DBG23/GPX3_3 DU H9 GYRO_INT

由上图可知中断引脚复用的是GPX3_3.

参考I2C最全干货- (1) 裸机操作篇, mpu6050从设备地址为0x68.

综上设备树中描述I2C设备信息

```

i2c@138B0000 {
    samsung,i2c-sda-delay = <100>;
    samsung,i2c-max-bus-freq = <20000>;
    pinctrl-0 = <&i2c5_bus>;
}

```

基地址是 138B0000
通道5

```
pinctrl-names = "default";
status = "okay";

mpu6050-3-asix@68 {
    compatible = "invensense,mpu6050";
    reg= <0x68>;
    interrupt-parent = <&gpx3>;
    interrupts= <3 2>;
};
```

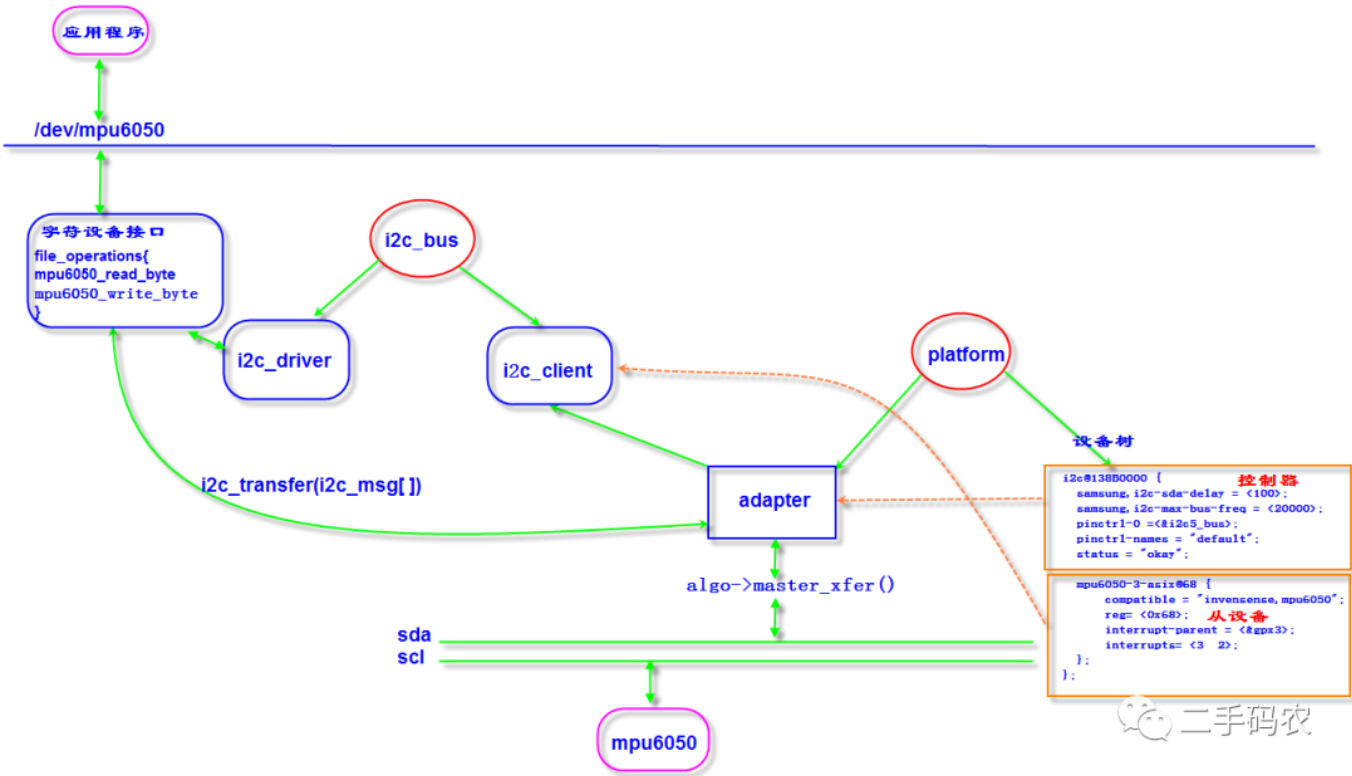
从设备地址

中断父节点

中断index=3，中断触发方式：下降沿触发


其中 外面节点 i2c@138B0000{}是i2c控制器设备树信息，子节点 mpu6050-3-asix@68{}是从设备mpu6050的设备树节点信息。

结构体之间关系如下：



1. 设备树节点分为控制器和从设备两部分，控制器节点信息会通过platform总线与控制器驱动匹配，
控制器驱动已经由内核提供，结构体如下：

```
static struct platform_driver s3c24xx_i2c_driver = {
    .probe      = s3c24xx_i2c_probe,
    .remove     = s3c24xx_i2c_remove,
    .id_table   = s3c24xx_driver_ids,
    .driver     = {
        .owner   = THIS_MODULE,
        .name    = "s3c-i2c",
        .pm      = S3C24XX_DEV_PM_OPS,
        .of_match_table = of_match_ptr(s3c24xx_i2c_match),
    },
};
```

 二手码农

2. 从设备节点信息最终会通过i2c_bus与i2c_driver匹配，i2c_driver需要由开发者自己注册，并实现字符设备接口和创建设备节点/dev/mpu6050;
3. 用户通过字符设备节点/dev/mpu6050调用内核的注册的接口函数mpu6050_read_byte、mpu6050_write_byte;
4. 内核的i2c core模块提供了i2c协议相关的核心函数，在实现读写操作的时候，需要通过一个重要的函数i2c_transfer()，这个函数是i2c核心提供给设备驱动的，通过它发送的数据需要被打包成i2c_msg结构，这个函数最终会回调相应i2c_adapter->i2c_algorithm->master_xfer()接口将i2c_msg对象发送到i2c物理控制器。

应用实例，实现mpu6050驱动，读取温度：

【注】实例所用soc是exynos4412，为三星公司所出品，所以i2c控制器设备树节点信息可以参考linux内核根目录下文件：Documentation\devicetree\bindings\i2c\i2c-s3c2410.txt。
不同的公司设计的i2c控制器设备树节点信息填写格式不尽相同，需要根据具体产品填写。


编写驱动代码

分配、设置、注册i2c_driver结构体

```
struct i2c_driver mpu6050_driver = {
    .driver = {
        .name = "mpu6050",
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(mpu6050_of_match),
    },

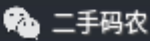
    .probe = mpu6050_probe,
    .remove = mpu6050_remove,
};

/* 可在模块加载函数中将i2c_driver结构体注册到i2c_bus管理的链表中，与i2c设备进行匹配 */
i2c_add_driver(&mpu6050_driver);
```

 二手码农

i2c总线驱动模型属于设备模型中的一类，同样struct i2c_driver结构体继承于struct driver，匹配方法和设备模型中讲的一样，这里要去匹配设备树，所以必须实现i2c_driver结构体中的driver成员中的of_match_table成员：

```
/* 用来匹配mpu6050的设备树节点 */
static struct of_device_id mpu6050_of_match[] = {
    {.compatible = "invensense,mpu6050"},
    {}
};
```



如果和设备树匹配成功，那么就会调用probe函数

```
/* 匹配函数,设备树中的mpu6050结点对应转换为一个client结构体 */
static int mpu6050_probe(struct i2c_client * client, const struct i2c_device_id * id)
{
    mpu6050_dev.client = client;
    /* 注册设备号 */
    mpu6050_dev.devno = MKDEV(MAJOR, MINOR);
    register_chrdev_region(mpu6050_dev.devno, 1, "mpu6050");

    cdev_init(&mpu6050_dev.cdev, &mpu6050_fops);
    mpu6050_dev.cdev.owner = THIS_MODULE;
    cdev_add(&mpu6050_dev.cdev, mpu6050_dev.devno, 1);

    return 0;
}
```



实现文件操作集合


```
struct file_operations mpu6050_fops = {
    .owner = THIS_MODULE,
    .open = mpu6050_open,
    .release = mpu6050_release,
    .read = mpu6050_read,
};

static int mpu6050_open(struct inode * inode, struct file * file)
{
    /* 初始化mpu6050 */
    mpu6050_write_byte(mpu6050_dev.client,0x1A, 0x00);
    ...

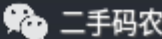
    return 0;
}

static int mpu6050_release(struct inode * inodep, struct file * file)
{
    return 0;
}

static long mpu6050_read(struct file * fp, char __user * buf, size_t count, loff_t * loff)
{
    int temp;

    temp = mpu6050_read_byte(mpu6050_dev.client, TEMP_OUT_L);
    temp |= mpu6050_read_byte(mpu6050_dev.client, TEMP_OUT_H) << 8;

    copy_to_user(buf, &temp, count);
    return count;
}
```



如何填充i2c_msg?

根据mpu6050的datasheet可知，向mpu6050写入1个data和读取1个值的时序分别如下图所示。

Single-Byte Write Sequence

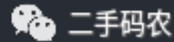
Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

基于Linux的i2c架构编写驱动程序，我们需要用struct i2c_msg结构体来表示上述所有信息。

```
struct i2c_msg {
    __u16 addr;    /* slave address */
    __u16 flags;   /* 1 - 读 0 - 写 */
    __u16 len;     /* msg length */
    __u8 *buf;     /* 要发送的数据 */
};
```



编写i2c_msg信息原则如下：

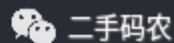
1. 有几个S信号，msg数组就要有几个元素；
2. addr为从设备地址，通过i2c总线调用注册的probe函数的参数i2c_client传递下来；
3. len的长度不包括S、AD、ACK、P；
4. buf为要发送或者要读取的DATA的内存地址。

综上所述：

1. Single-Byte Write Sequence时序只需要1个i2c_msg,len值为2，buf内容为是RA、DATA；
2. Single-Byte Read Sequence时序需要2个i2c_msg,len值分别都为1，第1个msg的buf是RA，第2个msg的buf缓冲区用于存取从设备发送的DATA。

```
/* 读取mpu6050中一个字节的数，将读取的数据的地址返回 */
static int mpu6050_read_byte(struct i2c_client * client, unsigned char reg_addr)
{
    /* 要读取的那个寄存器的地址 */
    char txbuf = reg_addr;
    /* 用来接收读到的数据 */
    char rxbuf[1];
    /* i2c_msg指明要操作的从机地址，方向，缓冲区 */
    struct i2c_msg msg[] = {
        {client->addr, 0, 1, &txbuf}, //0表示写，向从机写要操作的寄存器的地址
        {client->addr, I2C_M_RD, 1, rxbuf}, //读数据
    };
    /* 通过i2c_transfer函数操作msg */
    i2c_transfer(client->adapter, msg, 2); //执行2条msg
    return rxbuf[0];
}

static int mpu6050_write_byte(struct i2c_client * client, unsigned char reg_addr, unsigned char data)
{
    /* 要写的那个寄存器的地址和要写的数据 */
    char txbuf[] = {reg_addr, data}; /* 1个msg，写两次 */
    struct i2c_msg msg[] = { {client->addr, 0, 2, txbuf} };
    i2c_transfer(client->adapter, msg, 1);
    return 0;
}
```



参考文献：

mpu6050 datasheet、exynos4412 datasheet以及源代码下载请关注订阅号，回复'exynos4412'，即可获取。

本公众号定期发布关于嵌入式的原创干货。敬请关注'二手码农'。

收录于合集 #所有原创 194

上一篇
I2C最全干货- (3) 驱动篇-之内核架构分析

下一篇
I2C最全干货- (1) 裸机操作篇

喜欢此内容的人还喜欢

570个常用的Linux命令，1349页Linux命令速查手册（附PDF）
一口Linux

