doi: 10.3969/j.issn.1006-1576.2011.06.024

# Linux 下基于 I2C 的电源管理芯片驱动设计

于海航，杜刚，石仁协
（中国地质大学 信息工程学院，北京 100083）

**摘要**：为了提高驱动运行效率，对 Linux 下内部集成电路（inter-integrated circuit，I2C）设备的电源管理芯片驱动进行设计。介绍 I2C 总线时序以及 Linux 内核中 I2C 总线所特有的体系结构，在分析 Linux 内核 I2C 总线驱动框架的基础上，结合具体的电源管理芯片 AXP192，介绍了 I2C 设备驱动的实现方法。结果证明，该设计能有效地提高效率。

**关键词**：I2C 总线；Linux；电源管理芯片；驱动
**中图分类号**：TP303⁺.3　**文献标志码**：A

# Power Management Chip Drive Design Based on I2C Under Linux

Yu Haihang, Du Gang, Shi Renxie
(Dept. of Information Engineering, China University of Geosciences, Beijing 100083, China)

Abstract: In order to improve the drive running efficiency, Design the power management chip drive of I2C (inter-integrated circuit) under Linux. The I2C bus timing and the special architecture of the I2C bus in the Linux kernel are introduced. On the basis of analysis Linux kernel I2c bus drive architecture, combine power management chip AXP192, introduce its realization method of I2C equipment drive. The result shows that the design can improve the efficiency effectively.

Keywords: I2C bus; Linux; power management chip; drive

## 0　引言

内部集成电路（inter-integrated circuit，I2C）总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。I2C 总线结构简单、使用方便。由于接口直接在组件之上，因此 I2C 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本[1]。I2C 总线现已应用于各种服务与管理场合，来实现配置或掌握组件的功能状态，如电池信息、系统温度等参数。Linux 系统下 I2C 的驱动程序具有清晰的层次结构，借助于成熟的驱动的例子用户很容易开发出针对自己产品的相应驱动。因此，笔者分析了 Linux 系统下 I2C 驱动结构，并在此基础上实现了一个具体的 I2C 设备的驱动。

## 1　I2C 总线概述与时序

### 1.1　I2C 总线介绍

I2C 总线是由串行数据线（serial data lines，SDA）和串行时钟线（serial clock lines，SCL）构成的串行总线，可发送和接收数据。所有的主/从设备都挂载在这条总线上，每个从设备都有唯一的设备地址，这样才能被主设备唯一识别。

### 1.2　I2C 总线信号时序

SDA 和 SCL 2 条信号线都处于高电平，即总线空闲状态，2 条信号线各自的上拉电阻把电平拉高；时钟信号 SCL 保持高电平期间，SDA 从高到低的跳变作为 I2C 总线的起始信号如图 1;时钟信号 SCL 保持高电平，数据线被释放，使得 SDA 返回高电平（即正跳变）作为 I2C 总线的停止信号如图 1。开始信号产生后,传送首字节的前 7 位为从设备地址，第 8 位为传输方向位，如果第 8 位为 0，表示这是一次写操作，否则表示这是一次读请求，数据传输的基本单位是 1 个字节（8 位），而且从最高位开始传送，传送的字节数不限，如图 2[2]。发送完一个字节后的下一个 CLK 周期（第 9 个时钟脉冲）为 ACK 周期，发送端发送完一个字节后会置 SDA 为高电平，即在 ACK 周期开始时置 SDA 为高电平，接收端接收完一个字节后，会立刻在 ACK 周期内将 SDA 由高电平翻转为低电平，这便产生了一个 ACK 信号，发送端在 ACK 周期检测到了 SDA 的电平跳变后，便认为收到了 ACK 信号，可以传输下一个字节，如图 3。若在时钟的第 9 个脉冲期间发送器释放数据总线，接收器不拉低数据总线表示一个 NACK，即认为传输出错。

---

收稿日期：2011-02-28；修回日期：2011-03-25
作者简介：于海航（1988—），男，山东人，硕士，从事嵌入式开发研究。

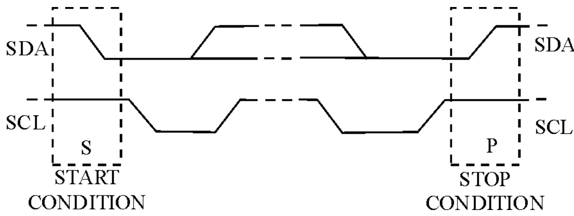


图 1　I2C 总线起始信号与结束信号[1]

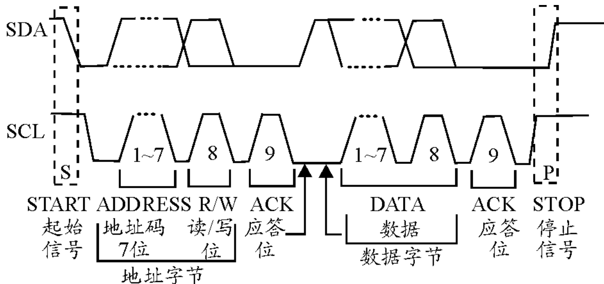


图 2　I2C 总线传输数据时序图

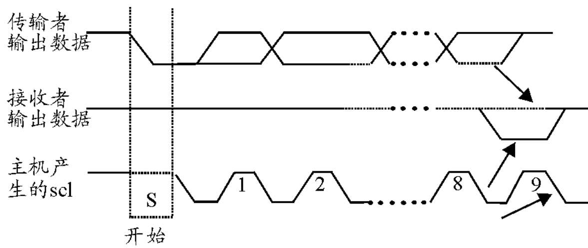


图 3　I2C 总线 ACK 时序图[1]

## 2　Linux 下 I2C 驱动框架

Linux I2C 框架中各个部分的关系如图 4，Linux 内核中 I2C 体系结构可以分为 3 个层次：I2C 框架、I2C 总线适配器驱动以及 I2C 设备驱动。

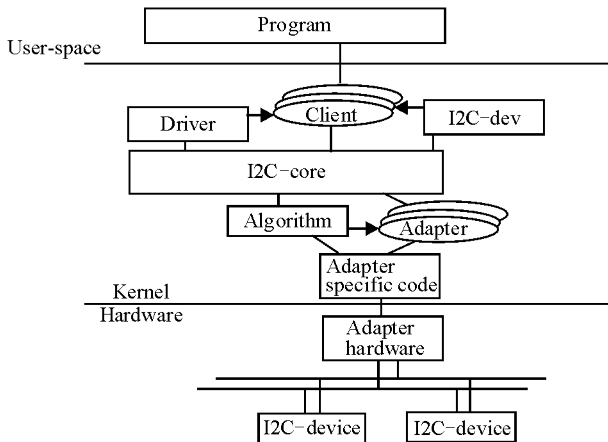


图 4　I2C 框架关系图[3]

### 2.1　I2C 框架

I2C.h 和 I2C-core.c 为 I2C 框架的主体，提供了核心数据结构的定义、I2C 适配器驱动和设备驱动的注册、注销管理，I2C 通信方法上层的、与具体适配器无关的代码、检测设备地址的上层代码等；I2C-dev.c 用于创建 I2C 适配器的/dev/I2C/%d 设备节点，提供 I2C 设备访问方法等[4]。

### 2.2　I2C 总线适配器驱动

在 kernel\drivers\I2C\busses 目录下，定义描述具体 I2C 总线适配器的 I2C_adapter 数据结构、实现在具体 I2C 适配器上的 I2C 总线通信方法，并由 I2C_algorithm 数据结构进行描述[1]。

### 2.3　I2C 设备驱动

定义描述具体设备的 I2C_client 和可能的私有数据结构、借助 I2C 框架的 I2C_probe 函数实现注册设备的 attach_adapter 方法、提供设备可能使用的地址范围、以及设备地址检测成功后创建 I2C_client 数据结构的回调函数。

## 3　linux 下 I2C 设备驱动的具体实现

### 3.1　AXP192 芯片介绍

AXP192 是高度集成的电源系统管理芯片，针对单芯锂电池（锂离子或锂聚合物）且需要多路电源转换输出的应用，提供简单易用而又可以灵活配置的完整电源解决方案，充分满足目前日益复杂的应用处理器系统对于电源相对复杂而精确控制的要求。AXP192 提供了一个与主机通讯的两线串行通讯接口：two wire serial interface(TWSI)，应用处理器可以通过这个接口去打开或关闭某些电源输出，设置它们的电压，访问内部寄存器和多种测量数据，包括 FuelGauge。

### 3.2　AXP192 驱动实现

由于 Linux 系统下已经实现了 I2C 框架，I2C 总线适配器的驱动，通过 I2C-dev.c 文件提供了一个通用的 I2C 设备的驱动程序，因此，驱动程序的开发主要集中在 AXP192 设备驱动程序这一层，用来实现针对 AXP192 设备的数据格式的解释以及一些专用功能，特别是对设备寄存器的读写。下面是定义一个 I2C_driver 的结构体以及实现总线初始化时的设备检测加载、设备删除时的数据操作。其中 I2C_driver 结构体为：

```
static struct I2C_driver AXP192_driver = {
    .driver = {
        .owner   = THIS_MODULE,
        .name    = "AXP192",
    },
    .probe          = AXP192_probe,
    .remove         = AXP192_remove,
    .id_table = AXP192_id,
```

```
        .detect          = AXP192_detect,
        .address_data = &addr_data,
};
```

适配器加载探测函数原型：

```
static int pcf8575_probe(struct I2C_client *client, const struct I2C_device_id *id)
```

设备移除函数原型：

```
static int pcf8575_remove(struct I2C_client *client)
```

探测芯片设备函数原型：

```
static int pcf8575_detect(struct I2C_client *client, int kind,struct I2C_board_info *info)
```

其中，对 AXP192 寄存器的读写有严格的时序，图 5 是 AXP192 读写时序图。从图中可以看出：在写操作时只有一次 I2C 交互，只需要一个 I2C_msg 数据结构即可。用户进程在写入前准备好 2 个字节的数组，第 1 个数组元素为待写入的寄存器编号，第 2 个字节为写入的数值，而无需对 I2C_master_send 函数做任何修改。而在读操作时需要 2 次交互，且以“Repeated Start”的形式出现。所以就不得不使用两个 I2C_msg 数据结构：第 1 个 I2C_msg 数据结构完成写操作，写入的数值为待读出的寄存器编号；第 2 个 I2C_msg 数据结构完成读操作，从先前指定的寄存器中读出数据。对 AXP192 寄存器进行读写操作的函数原型如下：其中 PHY_I2C_ADDRESS 为 AXP192 设备物理地址。下面的读写函数每次只能读写一个字节，若想连续地读写多个字节，可以令写函数调用下面的函数来实现对设备寄存器读写多个字节。在设备驱动初始化的过程中需要对 AXP192 的寄存器进行设置，可以通过 axp_read_ri()函数对寄存器进行赋值，并通过 axp_write_ri()函数读取寄存器的值来进行验证初始化的正确性。

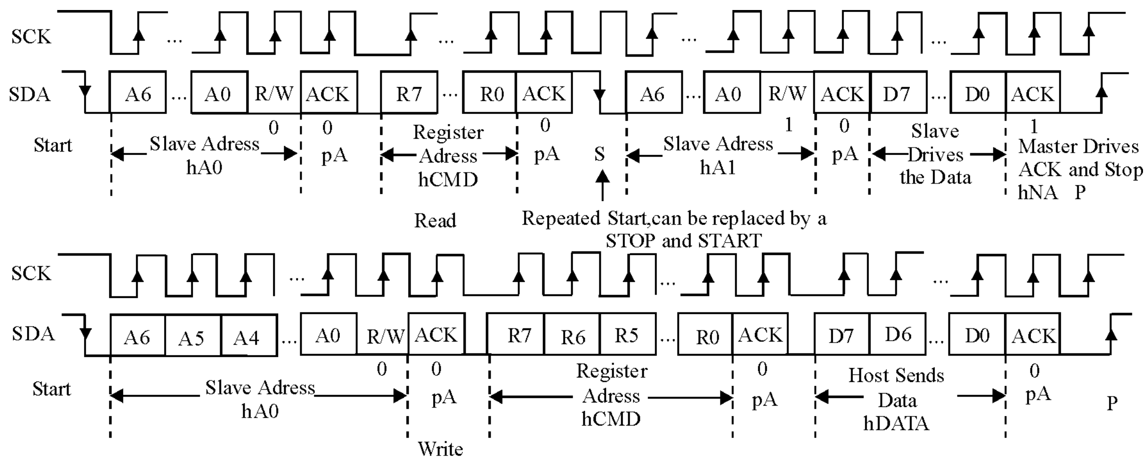


图 5　AXP192 读写时序

```
static int axp_read_ri(unsigned char offset, unsigned char *data, short len)
{
        struct I2C_adapter *adap;
        struct I2C_msg msgs[2];
        unsigned char test = offset;
        adap = I2C_get_adapter(0);
        if (!adap)
                return 0;
        // set offset
        msgs[0].addr= HY_I2C_ADDRESS>>1;
        msgs[0].flags = 0;
        msgs[0].len      = 1;
        msgs[0].buf      = &test;
        // read data
        msgs[1].addr= PHY_I2C_ADDRESS>>1;
        msgs[1].flags = I2C_M_RD;
        msgs[1].len      = len;
        msgs[1].buf      = data;
        // read from DDC line
        return I2C_transfer(adap, msgs, 2) ;
}
static int axp_write_ri(unsigned char offset, unsigned char data)
{
        struct I2C_adapter *adap;
        struct I2C_msg msgs[2];
        unsigned char buffer[2]={0,0};
        buffer[0] = offset;
        buffer[1] = data;
        adap = I2C_get_adapter(0);
        if (!adap)
                return 0;
// write
```

（下转第 90 页）