Linux I2C framework(3)_I2C consumer

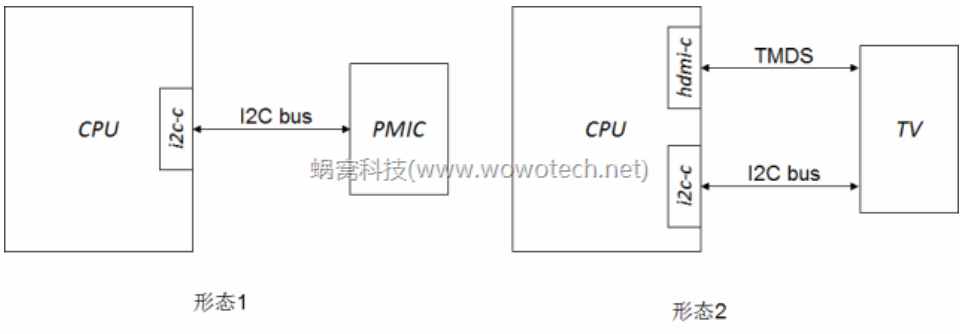
作者：wowo 发布于：2016-3-23 22:05 分类：通信类协议

1. 前言

本文从I2C consumer的角度，介绍怎么在linux中，利用I2C framework提供的接口，编写I2C slave device的驱动程序。

2. 两种设备形态

嵌入式系统中，I2C总线上连接的slave device，有两种形态，如下：



图片1 I2C slave device的两种形态

> 形态1，CPU和设备之间的所有数据交互，都是通过I2C总线进行，没有其它方式，如PMIC、Audio codec等。
>
> 形态2，I2C只是CPU和设备之间进行数据交互的一种，例如HDMI，图像以及音频数据通过TDMS接口传输，EDID等信息的交互通过I2C总线（在HDMI协议中称作DDC接口）。

这两种设备形态决定了设备在设备模型中的位置：

> 形态1比较简单，以PMIC为例，可以把它看作I2C bus上的一个设备；
>
> 形态2就复杂了，以TV为例，它一部分功能可看作I2C bus上的一个设备，另一部分是却是platform bus（HDMI Controller）上的一个设备，它的设备驱动要怎么写？一般是以其主要功能为准，TV的主要功能明显是音视频传输，因此应该当做一个platform设备。

在设备模型中的位置不同，最终可以体现在I2C slave device在DTS中的描述方式的不同，具体如下。

形态1，pmic的DTS node是i2c1的一个child node，I2C core负责该设备的创建和注册，以及和其driver的probe等操

站内搜索

请输入关键词 搜索

功能

留言板
评论列表

最新评论

linuxer
@rafal：多谢您的回复，不过我暂时想远离memory b...
rafal
"由于存在Invalidate Queue这中东西，因此，C...
wowo
@向往：我们还没有系统性的整理文件系统的文章啊，所以这几个问...
wowo
@dong：所谓的角色反转就是：前一次A是master，B是...
dong
“master和slave都要生成各自的LTK/EDIV/R...
dong
顶一下wowo，看了一个月BLE,终于入门了，wowo帖子帮...

文章分类

Linux内核分析(9)
统一设备模型(12)
电源管理子系统(42)
中断子系统(14)
进程管理(13)
内核同步机制(17)
GPIO子系统(5)
时间子系统(14)
通信类协议(7)
内存管理(18)
图形子系统(1)
文件系统(3)
TTY子系统(5)
u-boot分析(3)
Linux应用技巧(13)
软件开发(6)
基础技术(13)
蓝牙(16)
ARMv8A Arch(13)
显示(3)
基础学科(9)

作：

```
/* arch/arm/boot/dts/imx6dl-riotboard.dts */

&i2c1 {
        clock-frequency = <100000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c1>;
        status = "okay";
        ...
        pmic: pf0100@08 {
                compatible = "fsl,pfuze100";
                ...
        };
        ...
};
```

形态2，hdmi的DTS node位于根目录，作为platform device存在，它的DDC功能所使用的i2c2，是以一个变量的形式引用的：

```
/* arch/arm/boot/dts/imx6dl-riotboard.dts */

&hdmi {
       ddc-i2c-bus = <&i2c2>;
        status = "okay";
};
```

这两种不同的DTS描述，决定了最终的I2C slave device driver有不同的编写方式，具体请参考后面章节的描述。

3. 驱动编写步骤

针对第2章所描述的两种不同的设备形态，有两种驱动编写方法。

3.1 形态1

1）根据硬件的连接方式，确定该设备所从属的I2C controller（在I2C framework中称作I2C adapter），例如第2章例子中的i2c1。

2）在I2C adapter的DTS node中，添加该设备的DTS描述，其格式和正常的platform device一致。

3）DTS描述中的compatible关键字用于设备和驱动的probe，如“compatible = "fsl,pfuze100";”，其它字段根据实际情况自行添加。

4）编写该设备的驱动程序，完成如下内容（具体可参考drivers/regulator/pfuze100-regulator.c）：

a）定义一个struct i2c_driver类型的变量，并调用module_i2c_driver接口将其注册到I2C core中。

b）该变量包含应包含一个DTS中的“compatible ”字段相同的of_match_table，以及一个probe接口。

5）由“Linux I2C framework(2)_I2C provider”的描述可知，I2C framework core会在每一个I2C adapter注册时，为它下面所有的slave device创建struct i2c_client结构，并匹配对应的struct i2c_driver变量，调用driver的probe接口。

6）i2c_driver的probe接口的输入参数是struct i2c_client类型的指针（代表I2C slave device），以struct i2c_client指针为参数，可以调用i2c_master_send/i2c_master_recv接口进行简单的I2C传输，同时，也可以通过该指针获得所属的I2C adapter指针，然后通过i2c_transfer接口，进行更为复杂的read、write操作（可参考“drivers/base/regmap/regmap-

技术漫谈(12)
项目专区(0)
X Project(26)

随机文章
Linux DMA Engine framework(2)_功能介绍及解接口分析
linux kernel的中断子系统之（七）：GIC代码分析
X-013-UBOOT-使能autoboot功能
Linux2.6.11版本：classic RCU的实现
ARMv8之Observability

文章存档
2017年9月(2)
2017年8月(4)
2017年7月(4)
2017年6月(3)
2017年5月(3)
2017年4月(1)
2017年3月(8)
2017年2月(6)
2017年1月(5)
2016年12月(6)
2016年11月(11)
2016年10月(9)
2016年9月(6)
2016年8月(9)
2016年7月(5)
2016年6月(8)
2016年5月(8)
2016年4月(7)
2016年3月(5)
2016年2月(5)
2016年1月(6)
2015年12月(6)
2015年11月(9)
2015年10月(9)
2015年9月(4)
2015年8月(3)
2015年7月(7)
2015年6月(3)
2015年5月(6)
2015年4月(9)
2015年3月(9)
2015年2月(6)
2015年1月(6)
2014年12月(17)
2014年11月(8)
2014年10月(9)
2014年9月(7)
2014年8月(12)
2014年7月(6)
2014年6月(6)
2014年5月(9)
2014年4月(9)
2014年3月(7)
2014年2月(3)
2014年1月(4)

i2c.c”中的regmap_i2c_read接口）。

3.2 形态2

1）根据硬件的连接方式，确定该设备所从属的I2C controller（在I2C framework中称作I2C adapter），例如第2章例子中的i2c2。

2）将该设备（如HDMI）当做一个platform device，并按照platform device的通用方法，提供DTS描述、编写platform driver，可参考第2章中描述的hdmi的例子。

3）DTS描述中，使用一个变量，指向其I2C adapter的DTS节点，例如：“ddc-i2c-bus = <&i2c2>; ”。

4）在platform driver的probe函数中，以“ddc-i2c-bus ”参数，调用of_parse_phandle接口，获取I2C adapter的device node（即i2c的device node），然后调用of_find_i2c_adapter_by_node获取相应的I2C adapter指针，如下：

```
/* drivers/gpu/drm/panel/panel-simple.c */

ddc = of_parse_phandle(dev->of_node, "ddc-i2c-bus", 0);
if (ddc) {
	panel->ddc = of_find_i2c_adapter_by_node(ddc);
	of_node_put(ddc);

	if (!panel->ddc) {
		err = -EPROBE_DEFER;
		goto free_backlight;
	}
}
```

5）获得struct i2c_adapter指针后，即可通过i2c_transfer接口，即可进行read、write操作。

4. 关键数据结构和API介绍

4.1 i2c client

由“Linux I2C framework(1)_概述”可知，I2C framework使用struct i2c_client抽象I2C slave device（对应设备模型中的struct device），具体如下：

```
1: /* include/linux/i2c.h */
2:
3: /**
4:  * struct i2c_client - represent an I2C slave device
5:  * @flags: I2C_CLIENT_TEN indicates the device uses a ten bit chip address;
6:  *	I2C_CLIENT_PEC indicates it uses SMBus Packet Error Checking
7:  * @addr: Address used on the I2C bus connected to the parent adapter.
8:  * @name: Indicates the type of the device, usually a chip name that's
9:  *	generic enough to hide second-sourcing and compatible revisions.
10:  * @adapter: manages the bus segment hosting this I2C device
```

1）flags，指示该I2C slave device一些特性，包括：

I2C_CLIENT_PEC，indicates it uses SMBus Packet Error Checking；
I2C_CLIENT_TEN，indicates the device uses a ten bit chip address；
I2C_CLIENT_WAKE，该设备具备wakeup的能力。

2）addr，该设备的7-bit的slave地址。

3）adapter，该设备所在的I2C controller。

4）irq，irq number（如果有的话）。

通常情况下，struct i2c_client变量是由I2C core在register adapter的时候，解析adapter的child node自行创建的（具体可参考“Linux I2C framework(2)_I2C provider”），该数据结构中的有些信息，可通过DTS配置，包括：

```
xxx:xxx@08 {
        reg = <0x08>;                   /* 对应struct i2c_client中的'addr'*/
        interrupts = <16 8>;            /* 对应struct i2c_client中的'irq'*/
        wakeup-source;                  /* 对应flags中的I2C_CLIENT_WAKE */
};
```

4.2 I2C adapter

I2C数据传输（read or write），需要以struct i2c_adapter为操作对象，具体可参考“Linux I2C framework(2)_I2C provider”。

4.3 i2c msg

I2C数据传输的单位，具体可参考“Linux I2C framework(2)_I2C provider”。

4.4 编写I2C slave driver所需使用的API

```
1: /* include/linux/i2c.h */
2:
3: /* must call put_device() when done with returned i2c_client device */
4: extern struct i2c_client *of_find_i2c_device_by_node(struct device_node *node);
5:
6: /* must call put_device() when done with returned i2c_adapter device */
7: extern struct i2c_adapter *of_find_i2c_adapter_by_node(struct device_node *node);
```

通过DTS节点获取相应的client或者adapter指针，3.2中的例子已经说明了of_find_i2c_adapter_by_node函数的使用场景。

```
1: /* include/linux/i2c.h */
2:
3: /*
4:  * The master routines are the ones normally used to transmit data to devices
5:  * on a bus (or read from them). Apart from two basic transfer functions to
6:  * transmit one message at a time, a more complex version can be used to
7:  * transmit an arbitrary number of messages without interruption.
8:  * @count must be be less than 64k since msg.len is u16.
9:  */
10: extern int i2c_master_send(const struct i2c_client *client, const char *buf,
11:                            int count);
12: extern int i2c_master_recv(const struct i2c_client *client, char *buf,
```

i2c数据传输有关的接口有两类：

一类是以i2c client为参数，进行简单的数据收发，包括i2c_master_send/i2c_master_recv。该方法只可以通过标准方式，发送或者接收一定数量的数据。

另一类是以i2c adapter和i2c msg为参数，可以更为灵活的read或者write数据，包括i2c_transfer。使用该方法可以以struct i2c_msg为参数，一次读取、或者写入、或者读取加写入，一定数量的数据。

原创文章，转发请注明出处。蜗窝科技，www.wowotech.net。

标签: Linux I2C consumer slave

« Debian下的WiFi实验（二）：无线网卡自动连接AP | systemd：为何要创建一个新的init系统软件»

评论：

小白
2017-09-04 23:42

wowo你好！在of_i2c_register_device函数中是解析i2c的slaver设备的信息，主要是填充struct i2c_board_info结构，然后内核使用i2c_new_device(adap, &info)解析成client信息，我看了源码，并没有看到struct i2c_client中的irq信息的解析！不知道是不是哪里没有注意到！在此请教！难道这个irq信息的填充，内核并没有帮我们填充吗？我们自己的驱动中解析？但是想想也不合理，因为platform_device解析资源的时候都会顺带将irq信息解析。

```
static struct i2c_client *of_i2c_register_device(struct i2c_adapter *adap,
                        struct device_node *node)
{
    struct i2c_board_info info = {};

    addr_be = of_get_property(node, "reg", &len);
    addr = be32_to_cpup(addr_be);

    info.addr = addr;
    info.of_node = of_node_get(node);
    info.archdata = &dev_ad;

    result = i2c_new_device(adap, &info);

    return result;
}
```

回复

wowo
2017-09-05 10:31

@小白：i2c_device_probe中解析的：

```
        if (!client->irq && dev->of_node) {
                int irq = of_irq_get(dev->of_node, 0);

                if (irq == -EPROBE_DEFER)
                        return irq;
                if (irq < 0)
                        irq = 0;

                client->irq = irq;
        }
```

回复

小白
2017-09-05 23:05

@wowo：谢谢

回复

emeralddream
2016-08-31 09:04

亲爱的版主：
请教一个问题。如果我要向flash写入一个字节的数据。
方法1：

```
    msg[0].addr = client->addr;
    msg[0].flags = client->flags;
    msg[0].len = 1;
    msg[0].buf = &wr_reg;

    msg[1].addr = client->addr;
    msg[1].flags = client->flags;
    msg[1].len = 1;
    msg[1].buf = buf;
```

方法2：

```
        char *bufs[2] ={
```

```
        {wr_reg},{*buf}
    };
    msg[0].addr = client->addr;
msg[0].flags = client->flags;
msg[0].len = 2;
msg[0].buf = &bufs;
```

就是把写入的地址和数据 分成2个不同的 MSG，或者在同一个msg。有没有什么不同？分成2个msg会不会在传送写入地址后，又发送1帧i2c从机地址+wr，然后再发要写的数据。

回复

wowo

2016-08-31 09:21

@emeralddream：是由区别的。具体的行为，和Controller的默认配置有关，msg的flags参数也可以影响，具体可以参考http://www.wowotech.net/comm/i2c_provider.html中有关的说明。

回复

Peter

2016-07-28 22:29

Audio Codec 有音频数据接口(eg: I2S/PCM 等)用于传输数字音频信号，而i2c总线 主要用于配置寄存器；因此应该属于形态2

回复

wowo

2016-07-29 08:43

@Peter：多谢提醒，你说的是对的:-)

回复

ohyes158

2017-04-24 15:29

@wowo：对于形态2的i2c slave，其实也不属于platform
依然是i2c device，只是从类设备来说，会从属于音频(音频codec)或者视频子系统（v4l2 subdev）

回复

Musiclover

2016-03-25 23:01

刚好学习i2c就看见wowo更新文章，我真是有福。

回复

wowo

2016-03-26 11:10

@Musiclover：哈哈，客气了，希望多提宝贵意见~~

回复

发表评论：

昵称

邮件地址 (选填)

个人主页 (选填)

发表评论

Copyright @ 2013-2015 蜗窝科技 All rights reserved. Powered by emlog