

# I2C 驱动实验

## 一、实验目的

1. 熟悉 I2C 协议的原理。
2. 熟悉 linux 下 I2C 的驱动构架。
3. 熟悉 Linux 驱动程序的编写。
4. 熟悉 Linux 下模块的加载等。

## 二、实验内容

- 1.编写 I2C 协议的驱动程序。
- 2.编写 Makefile。
- 3.以模块方式加载驱动程序。
- 4.编写 I2C 测试程序。

## 三、实验设备

- 1.硬件：PC 机，基于 ARM9 系统教学实验系统实验箱 1 台；网线；串口线，电压表。
- 2.软件：PC 机操作系统；Putty；服务器 Linux 操作系统；arm-v5t\_le-gcc 交叉编译环境。
- 3.环境：ubuntu12.04.4；文件系统版本为 filesys\_test；烧写的内核版本为 uImage\_mceb。源码及参考文档见附件 I2C 驱动实验文件夹。

## 四、预备知识

### 4.1 概述

I2C（Inter-Integrated Circuit）总线是由 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。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。I2C 总线支持任何 IC 生产工艺（CMOS、双极型）。通过串行数据（SDA）线和串行时钟（SCL）线在连接到总线的器件间传递信息。每个器件都有一个唯一的地址识别（无论是微控制器——MCU、LCD 驱动器、存储器或键盘接口），而且都可以作为一个发送器或接收器（由器件的功能决定）。任何被寻址的器件都被认为是从机。

I2C 总线（I2C bus，Inter-IC bus）是一个双向的两线连续总线，提供集成电路（ICs）之间的通信线路。I2C 总线是一种串行扩展技术，最早由 Philips 公司推出，广泛应用于电视，录像机和音频设备。I2C 总线的意思是“完成集成电路或功能单元之间信息交换的规范或协议”。Philips 公司推出的 I2C 总线采用一条数据线（SDA），加一条时钟线（SCL）来完成数据的传输及外围器件的扩展；对各个节点的寻址是软寻址方式，节省了片选线，标准的寻址字节 SLAM 为 7 位，可以寻址 127 个单元。

I2C 总线有三种数据传输速度：标准，快速模式和高速模式。标准的是 100Kbps，快速模式为 400Kbps，高速模式支持快至 3.4Mbps 的速度。所有的与次之传输速度的模式都是兼容的。I2C 总线支持 7 位和 10 位地址空间设备和在不同电压下运行的设备。

Linux 的 I2C 体系结构分为 3 个组成部分：

**I2C 核心：**I2C 核心提供了 I2C 总线驱动和设备驱动的注册，注销方法，I2C 通信方法（“algorithm”）上层的，与具体适配器无关的代码以及探测设备，检测设备地址的上层代码等。

**I2C 总线驱动：**I2C 总线驱动是对 I2C 硬件体系结构中适配器端的实现，适配器可由 CPU 控制，甚至可以直接集成在 CPU 内部。

**I2C 设备驱动：**I2C 设备驱动（也称为客户驱动）是对 I2C 硬件体系结构中设备端的实现，设备一般挂接在受 CPU 控制的 I2C 适配器上，通过 I2C 适配器与 CPU 交换数据。

## 4.2 实现的功能

1> 实现了将设备挂载 I2C 总线。

2> 实现了设备本身驱动注册。

## 4.3 基本原理

### 4.3.1 I2C 协议

以启动信号 START 来掌管总线，以停止信号 STOP 来释放总线；每次通信以 START 开始，以 STOP 结束；启动信号 START 后紧接着发送一个地址字节，其中 7 位为被控器件的地址码，一位为读/写控制位 R/W，R/W 位为 0 表示由主控向被控器件写数据，R/W 为 1 表示由主控向被控器件读数据；当被控器件检测到收到的地址与自己的地址相同时，在第 9 个时钟周期间反馈应答信号；每个数据字节在传送时都是高位（MSB）在前。

写通信过程：

- 主控在检测到总线空闲的状况下，首先发送一个 START 信号掌管总线。
- 发送一个地址字节（包括 7 位地址码和一位 R/W）。
- 当被控器件检测到主控发送的地址与自己的地址相同时发送一个应答信号（ACK）。
- 主控收到 ACK 后开始发送第一个数据字节。
- 被控器收到数据字节后发送一个 ACK 表示继续传送数据，发送 NACK 表示传送数据结束。
- 主控发送完全部数据后，发送一个停止位 STOP，结束整个通信并且释放总线。

读通信过程：

- 主控在检测到总线空闲的状况下，首先发送一个 START 信号掌管总线。
- 发送一个地址字节（包括 7 位地址码和一位 R/W）。
- 当被控器件检测到主控发送的地址与自己的地址相同时发送一个应答信号（ACK）。
- 主控收到 ACK 后释放数据总线，开始接收第一个数据字节。
- 主控收到数据后发送 ACK 表示继续传送数据，发送 NACK 表示传送数据结束。

- 主控发送完全部数据后，发送一个停止位 STOP，结束整个通信并且释放总线。

#### 4.3.2 总线信号时序分析

总线空闲状态：SDA 和 SCL 两条信号线都处于高电平，即总线上所有的器件都释放总线，两条信号线各自的上拉电阻把电平拉高。

启动信号 START：时钟信号 SCL 保持高电平，数据信号 SDA 的电平被拉低（即负跳变）。启动信号必须是跳变信号，而且在建立该信号前必须保证总线处于空闲状态。

停止信号 STOP：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），停止信号也必须是跳变信号。

数据传送：SCL 线呈现高电平期间，SDA 线上的电平必须保持稳定，低电平表示 0（此时的线电压为低电压），高电平表示 1（此时的电压由元器件的 VDD 决定）。只有在 SCL 线为低电平期间，SDA 上的电平允许变化。

应答信号 ACK：I2C 总线的数据都是以字节（8 位）的方式传送的，发送器件每发送一个字节之后，在时钟的第 9 个脉冲期间释放数据总线，由接收器发送一个 ACK（把数据总线的电平拉低）来表示数据成功接收。

无应答信号 NACK：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，NACK 有两种用途：一般表示接收器未成功接收数据字节；当接收器是主控器时，收到最后一个字节后，应发送一个 NACK 信号，以通知被控发送器结束数据发送，并释放总线，以便主控接收器发送一个停止信号 STOP。

#### 4.3.3 寻址约定

地址的分配方法有两种：

含 CPU 的智能器件，地址由软件初始化时定义，但不能与其他的器件有冲突。

不含 CPU 的非智能器件，由厂家在器件内部固化，不可改变。高 7 位为地址码，分为两部分：高 4 位属于固定地址不可改变，由厂家固化的统一地址；低 3 位为引脚设定地址，可以由外部引脚来设定（并非所有器件都可以设定）。

#### 4.3.4 I2C 驱动构架

如下图所示 1 所示，I2C 驱动框架大概可以分为 3 个组成部分：

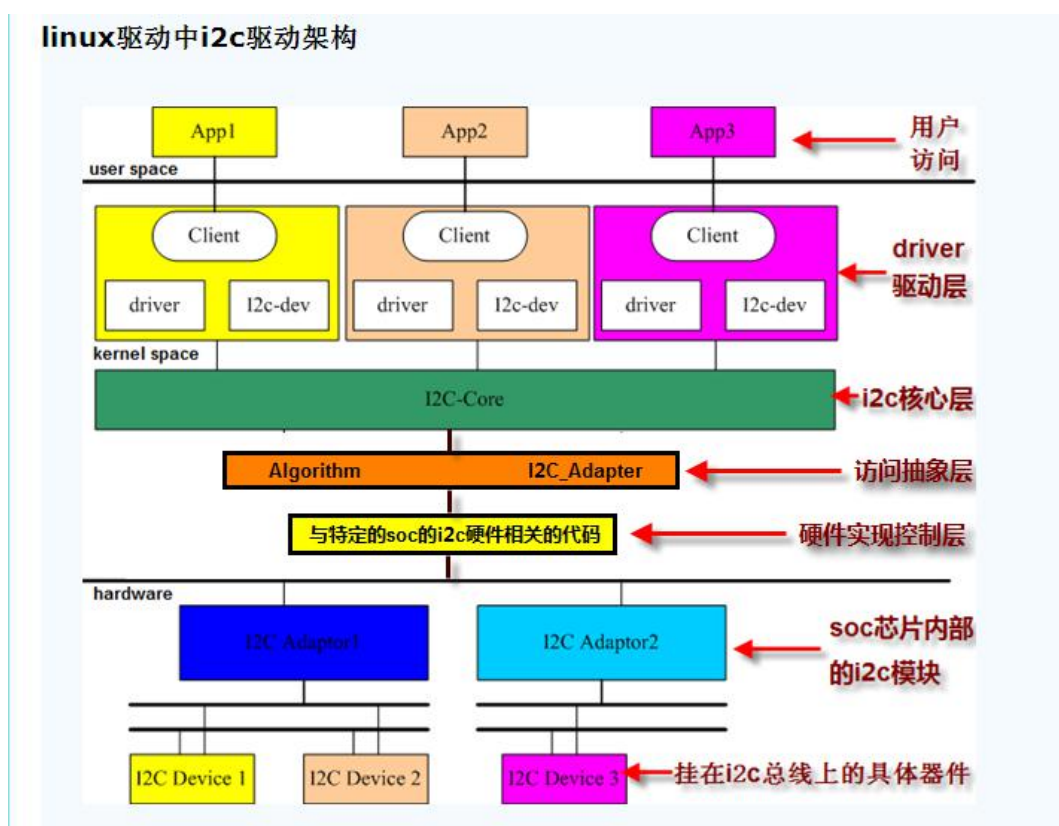


图 1 i2c 驱动框架

第一层：提供 i2c adapter 的硬件驱动，探测、初始化 i2c adapter（如申请 i2c 的 io 地址和中断号），驱动 soc 控制的 i2c adapter 在硬件上产生信号（start、stop、ack）以及处理 i2c 中断。覆盖图中的硬件实现层。

第二层：提供 i2c adapter 的 algorithm，用具体适配器的 xxx\_xferf() 函数来填充 i2c\_algorithm 的 master\_xfer 函数指针，并把赋值后的 i2c\_algorithm 再赋值给 i2c\_adapter 的 algo 指针。覆盖图中的访问抽象层、i2c 核心层。

第三层：实现 i2c 设备驱动中的 i2c\_driver 接口，用具体的 i2c device 设备的 attach\_adapter()、detach\_adapter() 方法赋值给 i2c\_driver 的成员函数指针。实现设备 device 与总线（或者叫 adapter）的挂接。覆盖图中的 driver 驱动层。

第四层：实现 i2c 设备所对应的具体 device 的驱动，i2c\_driver 只是实现设备与总线的挂接，而挂接在总线上的设备则是千差万别的，例如 eeprom 和 ov2715 显然不是同一类的 device，所以要实现具体设备 device 的 write()、read()、ioctl() 等方法，赋值给 file\_operations，然后注册字符设备（多数是字符设备）。覆盖图

中的 driver 驱动层。

第一层和第二层又叫 i2c 总线驱动(bus)，第三第四属于 i2c 设备驱动(device driver)。

在 linux 驱动架构中，几乎不需要驱动开发人员再添加 bus，因为 linux 内核几乎集成所有总线 bus，如 usb、pci、i2c 等等。并且总线 bus 中的(与特定硬件相关的代码)已由芯片提供商编写完成，例如 TI davinci 平台 i2c 总线 bus 与硬件相关的代码在内核目录/drivers/i2c/buses 下的 i2c-davinci.c 源文件中；三星的 s3c-2440 平台 i2c 总线 bus 为/drivers/i2c/buses/i2c-s3c2410.c。

第三第四层与特定 device 相干的就需要驱动工程师来实现了。

#### 4.3.5 架构层次分类

Linux I2C 驱动体系结构主要由 3 部分组成，即 I2C 核心、I2C 总线驱动和 I2C 设备驱动。I2C 核心是 I2C 总线驱动和 I2C 设备驱动的中间枢纽，它以通用的、与平台无关的接口实现了 I2C 中设备与适配器的沟通。I2C 总线驱动填充 i2c\_adapter 和 i2c\_algorithm 结构体。I2C 设备驱动填充 i2c\_driver 和 i2c\_client 结构体。

##### (1) I2C 核心框架

I2C 核心框架具体实现在/drivers/i2c 目录下的 i2c-core.c 和 i2c-dev.c。I2C 核心框架提供了核心数据结构的定义和相关接口函数，用来实现 I2C 适配器 驱动和设备驱动的注册、注销管理，以及 I2C 通信方法上层的、与具体适配器无关的代码，为系统中每个 I2C 总线增加相应的读写方法。

##### (2) I2C 总线驱动

I2C 总线驱动具体实现在/drivers/i2c 目录下 buses 文件夹。例如：Linux I2C GPIO 总线驱动为 i2c\_gpio.c。I2C 总线算法在/drivers/i2c 目录下 algos 文件夹。例如：Linux I2C GPIO 总线驱动算法实现在 i2c\_algo\_bit.c。I2C 总线驱动定义描述具体 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。经过 I2C 总线驱动的的代码，可以为我们控制 I2C 产生开始位、停止位、读写周期以及从设备的读写、产生 ACK 等，此部分驱动已经编译进内核。

##### (3) I2C 设备驱动

I2C 设备驱动具体实现放在在/drivers/i2c 目录下 chips 文件夹。I2C 设备驱动是对具体 I2C 硬件驱动的实现。I2C 设备驱动通过 I2C 适配器与 CPU 通信。其中主要包含 i2c\_driver 和 i2c\_client 数据结构，i2c\_driver 结构对应一套具体的驱动方法，例如：probe、remove、suspend 等，需要自己申明。i2c\_client 数据结构由内核根据具体的设备注册信息自动生成，设备驱动根据硬件具体情况填充。

#### 4.3.6 I2C 体系文件构架

在 Linux 内核源代码中的 driver 目录下包含一个 i2c 目录，如下图 2 所示：

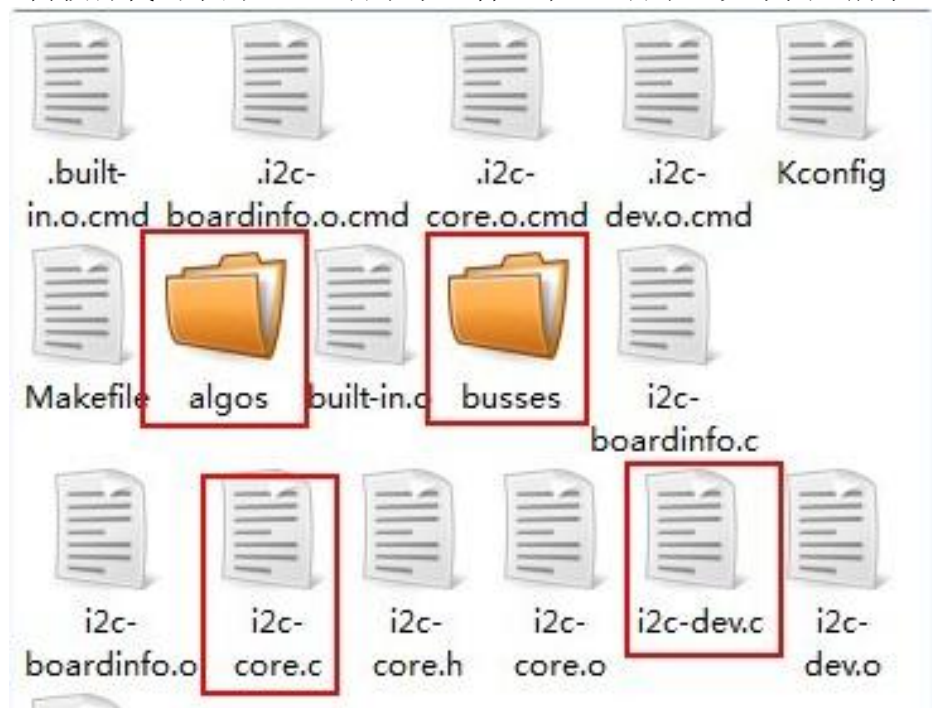


图 2 文件夹架构

i2c-core.c 这个文件实现了 I2C 核心的功能以及/proc/bus/i2c\*接口。

i2c-dev.c 实现了 I2C 适配器设备文件的功能，每一个 I2C 适配器都被分配一个设备。通过适配器访设备时的主设备号都为 89，次设备号为 0-255。I2c-dev.c 并没有针对特定的设备而设计，只是提供了通用的 read(),write(),和 ioctl()等接口，应用层可以借用这些接口访问挂接在适配器上的 I2C 设备的存储空间或寄存器，并控制 I2C 设备的工作方式。

busses 文件夹这个文件中包含了一些 I2C 总线的驱动，如针对 S3C2410，S3C2440，S3C6410 等处理器的 I2C 控制器驱动为 i2c-s3c2410.c。

algos 文件夹实现了一些 I2C 总线适配器的 algorithm。

下面张图展示是 linux 内核和芯片提供商为我们的驱动程序提供了 i2c 驱

动的框架，以及框架底层与硬件相关的代码的实现，如图 3 所示。剩下的就是针对挂载在 i2c 两线上的 i2c 设备了 device，如 x1205,ov5640, tvp5151,触摸屏，而编写的具体设备驱动了，这里的设备就是硬件接口外挂载的设备，而非硬件接口本身（soc 硬件接口本身的驱动可以理解为总线驱动）。

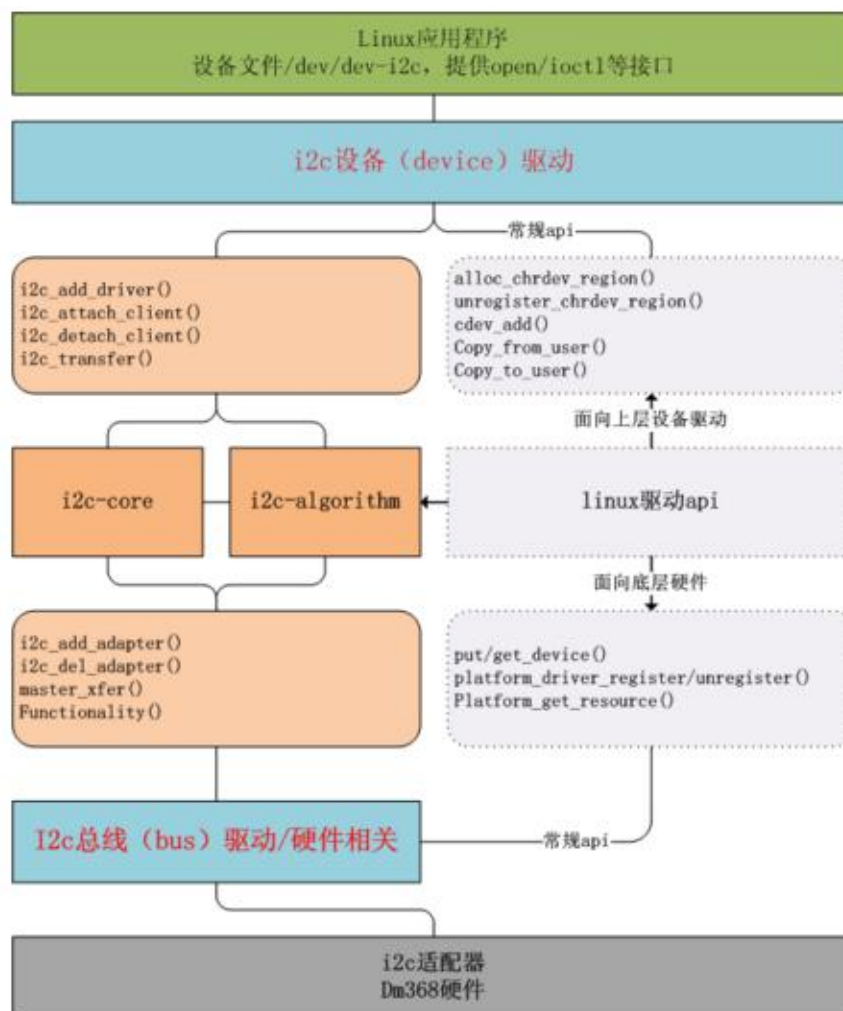


图 3 函数结构代码梳理图

#### 4.4 总体硬件结构设计

外接名为 SY 的设备通过 I2C 与 DM365 的电路连接, 硬件电路图如图 4 所示:



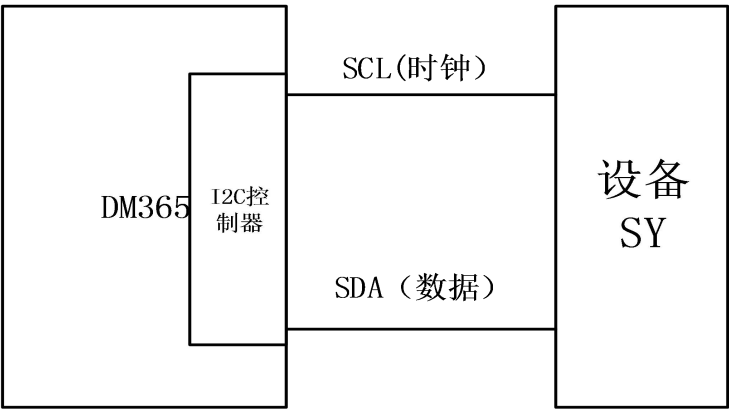


图 4 I2C 硬件电路图

4.5 软件框架

4.5.1 软件流程

设备名为 SY 的设备驱动注册流程如图 5 所示

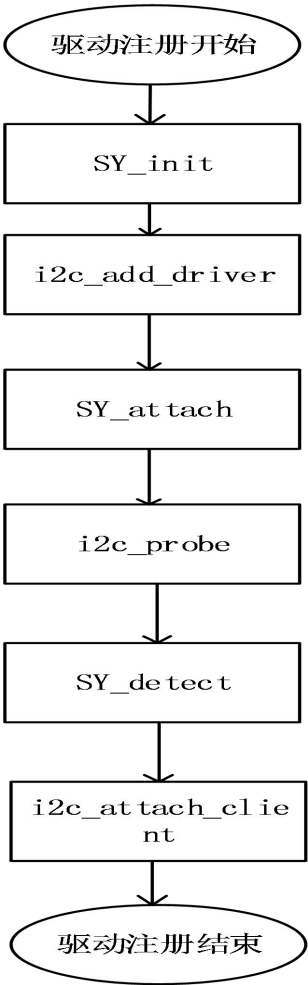


图 5 I2C 注册流程图

在 SY\_init 函数中，使用了 I2C 总线提供的 i2c\_add\_driver 函数，向 I2C 总线添加了 SY 的设备驱动，从而引起设备探测函数 i2c\_probe 的调用，这会使在该函数下的 SY\_detect 函数通过调用 I2C 总线驱动提供的 i2c\_attach\_client 函数，向内核的 I2C 总线注册 SY 设备，从而完成驱动的注册。

#### 4.5.2 操作函数的接口函数和结构体

##### 1> 驱动源码函数分析

函数: **MODULE\_LICENSE("GPL");**

功能: 将模块的许可协议设置为 GPL 许可,必不可少的。

函数: **module\_init(SY\_init);**

功能: module\_init 是内核模块一个宏。其用来声明模块的加载函数，也就是使用 insmod 命令加载模块时，调用的函数 SY\_init()。

函数: **module\_exit(SY\_exit);**

功能: module\_exit 是内核模块一个宏。其用来声明模块的释放函数，也就是使用 rmmod 命令卸载模块时，调用的函数 SY\_exit ()。

函数: **static int SY\_init (void)**

功能: 加载函数调用驱动注册函数实现驱动程序在内核的注册，同时还有可能对设备进行初始化，在驱动程序加载被调用。

函数: **static void SY\_exit(void)**

功能: 卸载函数调用解除注册函数实现驱动程序在内核的中的解除注册，同时在驱动程序卸载时被调用。

函数: **i2c\_add\_driver(&SYdriver);**

功能: 调用核心层函数，注册 SY\_driver 结构体

函数: **static int SY\_attach(struct i2c\_adapter \*adapter)**

功能: 在 attach 中通过 probe 探测到总线上的设备并把设备和驱动建立连接

以完成设备的初始化。

函数: **i2c\_attach\_client(SY\_client);**

功能: 等要卸载驱动时, 调用 SY\_detech\_adapter 函数

## 五、实验步骤

本次实验将执行下面几个步骤:

**步骤 1:** 编译 I2C 驱动 (在服务器上进行)

创建 I2C 驱动文件夹, 执行 `mkdir I2C`, 编写 I2C 驱动程序。

设备驱动源码参考文件夹 I2C 驱动实验/I2C/i2c.c。

测试程序源码参考文件夹 I2C 驱动实验/I2C/i2c\_test.c

(注: 文件夹名称以及驱动设备名称采用学生名字字母)。

编写驱动程序编译成模块所需要的 Makefile 文件, 执行 `vim Makefile`

```
KDIR:= /home/slho/kernel/ linux-2-6-18_pro500 //编译驱动模块依赖的内核路径, 修改为使用服务器
//上内核的路径

CROSS_COMPILE    = arm_v5t_le-
CC               = $(CROSS_COMPILE)gcc
.PHONY: modules clean
obj-m := i2c.o           //表明有个模块要从目标文件 i2c.o 建立. 在从目标文件建立后结果
                        模? //块命名为 i2c.ko

modules:
    make -C $(KDIR) M=`pwd` modules //根据提供的内核生成 i2c.o
clean:
    make -C $(KDIR) M=`pwd` modules clean
```

执行 `make` 命令, 成功后会生成 `i2c.ko` 等文件, 如图 6 所示,

```
slh0@ubuntu:~/pncompany/modules/I2C$ make
make -C /home/slho/kernel/linux-2-6-18_pro500 M=`pwd` modules
make[1]: Entering directory `/home/slho/kernel/linux-2-6-18_pro500'
Building modules, stage 2.
MODPOST
make[1]: Leaving directory `/home/slho/kernel/linux-2-6-18_pro500'
slh0@ubuntu:~/pncompany/modules/I2C$ ls
i2c.c  i2c.ko  i2c.mod.c  i2c.mod.o  i2c.o  i2c_test  i2c_test.c  Makefile  Module.symvers
slh0@ubuntu:~/pncompany/modules/I2C$
```

图 6 编译驱动模块

`i2c.ko` 驱动文件生成成功后, 将其复制到挂载的文件系统 `modules` 的目录下。

(命令 `sudo cp i2c.ko /home/XXX/nfs/ filesystem_test/modules`)

**步骤 2:** 挂载文件系统, 登入板子, 手动加载驱动模块

1> 连接好实验箱设备, 打开 `putty` 并登入串口终端。开启电源, 挂载文件系统

fileSYS\_test, 板子启动完成后按 enter 键输入 root, 再按 enter 键获取操作权限, 如下图 7 所示:

```
zjut login: root
[ 186.900000] nfs: server 192.168.1.3 not responding, still trying
[ 186.900000] nfs: server 192.168.1.3 OK

Welcome to MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921).

login[717]: root login on 'console'

/*****Set QT environment*****/

[root@zjut ~]#
```

图 7 登入板子获取 root 权限

2> 在驱动加载之前查看当前已经加载的驱动模块, 使用命令 lsmod。如图 8:

```
[root@zjut ~]# lsmod
Module                  Size  Used by    Tainted: P
dm365mmap 5336 0 - Live 0xbfc1000
edmak 13192 0 - Live 0xbfb0000
irqk 8552 0 - Live 0xbfb8000
cmemk 28172 0 - Live 0xbfb0000
ov5640_i2c 9572 1 - Live 0xbfac000
rtnet5572sta 53620 0 - Live 0xbf19d000
rt5572sta 1574024 1 rtnet5572sta, Live 0xbf01b000
rtutil5572sta 79988 2 rtnet5572sta,rt5572sta, Live 0xbf006000
egalax_i2c 16652 0 - Live 0xbf000000
```

图 8 当前加载的驱动模块

3> 执行 cd /sys/bus/i2c, 如图 9 所示。查看当前加载的设备地址和设备名。所有的 I2C 设备都在 sysfs 文件系统中显示。在当前目录文件夹下的 devices 下, 是当前挂载在 I2C 上的设备地址。在 drivers 目录下是挂载 I2C 上的设备驱文件。

```
[root@zjut /]# cd /sys/bus/i2c/
[root@zjut i2c]# ls
devices  drivers
[root@zjut i2c]# ls devices/
0-0018  0-003c
[root@zjut i2c]# ls drivers/
OV5640 channel0 Video Decoder I2C driver
aic3x I2C Codec
davinci_evm
dev_driver
egalax_i2c
i2c_adapter
[root@zjut i2c]#
```

图 9 设备地址和设备文件名

4> 执行 insmod /modules/i2c.ko。在驱动加载成功后会打印出设备的 I2C 注册地址。具体添加设备的地址查看数据手册, 每一个设备都会有一个固定地址。如下

图 10 所示。例如添加设备地址为 0x6f，转化成二进制为一个 8 位的数据。I2C 协议地址为 7 位。

```
[root@zjut /]# insmod /modules/i2c.ko
[ 1007.460000] SY_init
[ 1007.470000] SY_detect
[ 1007.480000] SY_probe with name = SY, addr = 0x6f
[root@zjut /]#
```

图 10 手动加载驱动模块

5> 驱动加载成功之后，查看 devices 和 drivers 目录，如下图 11 所示：

```
[root@zjut /]# cd /sys/bus/i2c/
[root@zjut i2c]# ls devices/
0-0004 0-0018 0-003c 0-006f
[root@zjut i2c]# ls drivers/
OV5640 channel0 Video Decoder I2C driver
SY
aic3x I2C Codec
davinci_evm
dev_driver
egalax_i2c
fm1188-i2c
i2c_adapter
[root@zjut i2c]#
```

图 11 设备 SY 注册成功图

6> 执行 cat /proc/devices,显示加载的 I2C 设备驱动程序创建了一个主设备号为 245,名为 SY 的设备节点，如图 12 所示：

```
[root@zjut i2c]# cat /proc/devices
Character devices:
 1 mem
 4 /dev/vc/0
 4 tty
 4 ttyS
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 7 vcs
10 misc
13 input
14 sound
21 sg
29 fb
81 video4linux
89 i2c
90 mtd
108 ppp
116 alsa
128 ptm
136 pts
180 usb
188 ttyUSB
189 usb_device
199 dm365_gpio
245 SY
246 dm365
```

图 12 节点创建成功

**步骤 3:** 编写测试程序，并进行调试。（在服务器上进行）

测试程序在文件夹 I2C 驱动实验/I2C/i2c\_test 下，i2c\_test.c 就是对应的测试文件。用来查看设备的地址。

在服务器编写测试程序的各函数后，接下来是使用交叉编译工具编译测试程序，并将编译后生成的可执行文件挂载到实验箱的板子上运行调试。

```
#arm_v5t_le-gcc i2c_test.c -o i2c_tets
```

交叉编译生成可执行文件 i2c\_test。编译成功后，可看见 i2c\_test 文件，将可执行文件 i2c\_test 复制到板子挂载的文件系统 filesys\_test/opt/dm365。执行如下命令 i2c\_test，如下图 13 所示。读出当前设备地址。

```
[root@zjut dm365]# i2c_test
reading data is OK
SY address = 0x6f
[root@zjut dm365]#
```

图 13 执行测试程序

**步骤 4:** 具体 I2C 设备实例（X1205 时钟芯片）

在上面的步骤中，模拟了一个名为 SY 的设备挂载在 I2C 上，这个硬件设备并不存在，所以无法进行读写。但实验箱上具体的挂在 I2C 的硬件设备有触摸屏，ov5640 芯片，tvp5151 芯片，以及回音抵消模块 fm1188，RTC 时钟芯片等设备，我们以 RTC 时钟设备 X1205 为例。RTC 芯片是通过 I2C 总线与主芯片 DM365 相连接，电路图连接图如图 14 所示。

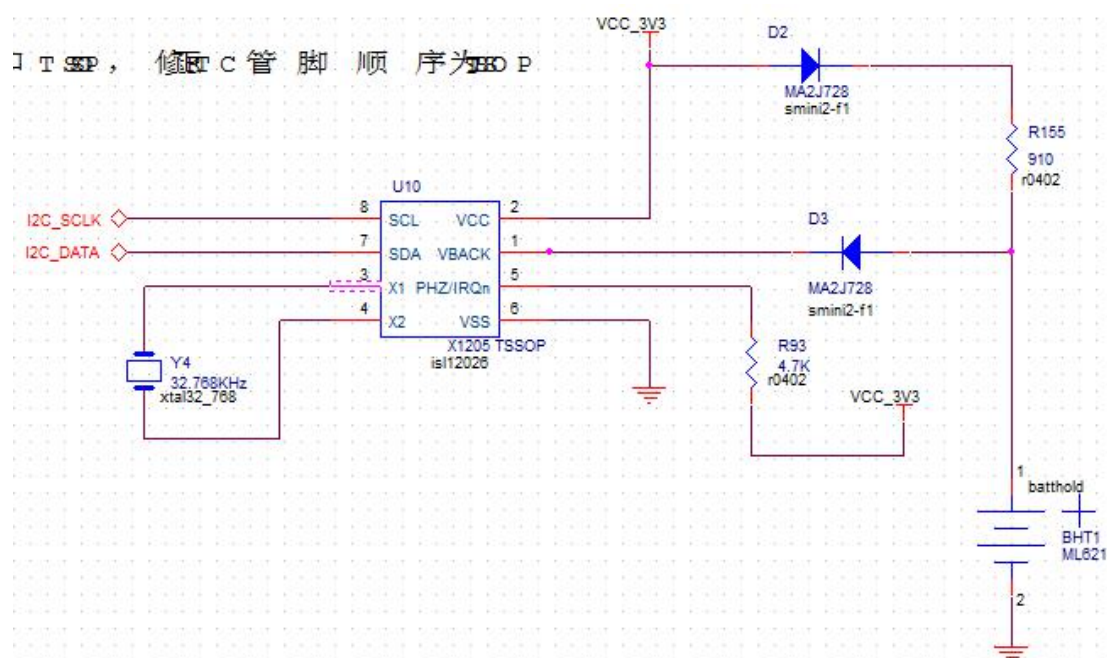


图 14 RTC 时钟电路连接图

关于 RTC 驱动原理以及部分操作可以参考 RTC 时钟驱动实验文档，附件中给出了 X1205 的驱动程序和测试程序。

设备驱动源码参考文件夹 I2C 驱动实验/RTC/rtc-x1205.c 及其 makefile。

测试程序源码参考文件夹 I2C 驱动实验/RTC/rtc\_test.c。

内核参考文件夹 I2C 驱动实验/RTC/内核/uImage\_wlwszbs

在编译好 rtc-x1205.c 与 makefile 后，输入 make，编译驱动，并将得到的驱动文件放到挂载的文件系统中

```
sudo cp rtc_x1205.ko /home/XXX/nfs/ filesystem_test/modules
```

编译好测试程序后，输入：arm\_v5t\_le-gcc rtc\_test.c -o rtc\_tets，进行编译，再把编译好的文件移入要挂载文件系统中

```
sudo cp rtc_test /home/XXX/nfs/ filesystem_test/opt/dm365
```

可以向 RTC 芯片 X1205 内写入时间并且读出来。

启动试验箱，对之前的文件系统重新挂载。

1> 使用 insmod /modules/rtc-x1205.ko 加载 RTC 驱动模块，如下图 15 所示；

```
[root@zjut ~]# lsmod
Module                  Size Used by    Tainted: P
dm365mmmap 5336 0 - Live 0xbfb1bd000
edmak 13192 2 - Live 0xbfb1b8000
irqk 8552 0 - Live 0xbfb1b4000
cmemk 28172 0 - Live 0xbfb1ac000
rtnet5572sta 53620 0 - Live 0xbfb19d000
rt5572sta 1574024 1 rtnet5572sta, Live 0xbfb01b000
rtutil5572sta 79988 2 rtnet5572sta,rt5572sta, Live 0xbfb006000
egallax_i2c 16652 0 - Live 0xbfb000000
[root@zjut ~]# insmod /modules/rtc-x1205.ko
[ 74.720000] x1205 0-006f: chip found, driver version 1.0.7
[ 74.720000] x1205 0-006f: rtc intf: proc
[ 74.730000] x1205 0-006f: rtc intf: dev (254:0)
[ 74.730000] x1205 0-006f: rtc core: registered x1205 as rtc0
[root@zjut ~]#
```

图 15 加载 rtc-x1205.ko

2> 执行测试程序 rtc\_test，根据提示输入数字 1，即可通过 I2C 总线读取 RTC 芯片 X1205 的时间参数,如下图 16 所示：

```
[root@zjut dm365]# rtc_test
Now you can choose
1 to select get_curtime
2 to select set_curtime
3 to get_alarmtime
1
Current RTC date/time is
2000-256-0, 00:00:36.
```



图 16 读取芯片时间

3> 可以任意设置 RTC 时间，执行测试程序 `rtc_test`。根据提示输入数字 2，如下图所示，依次输入年/月/日/时/分/秒。再输入任意数字结束当前操作。可以通过 I2C 向 RTC 芯片 X1205 写入时间。再次执行测试程序 `rtc_test`，输入数字 1，检测是否把时间参数写入芯片里。

```
[root@zjut dm365]# rtc_test
Now you can choose
1 to select get_curtime
2 to select set_curtime
3 to get_alarmtime
2
first is year
second is month
third is day
fourth is hour
fifth is minute
sixth is second
2016
12
12
9
28
00
1
Current RTC date/time is
2016-12-12, 09:28:00.
```

图 17 设置任意 RTC 时间