基于 I2C 的嵌入式多点触摸屏幕驱动设计

引言

随着嵌入式设备的开发和推广，触摸屏作为新式输入设备已经随处可见，手机、PDA、MID 以及 ATM 机等设备都已经用到了触摸屏。而科技在不断发展，触摸屏也由一开始的 4 线式单点电阻触摸屏发展到今天的各种多点式电容触摸屏。本文通过对以 cypress 7958 为代表的 I2C 总线接口电容式多点触摸屏的研究，设计了针对 Linux 操作系统的多点触摸的屏幕驱动，以及不运行操作系统前提下的单片机对触摸屏的驱动，取得了良好的效果。

1 研究平台介绍

1.1 ARM11 处理器 S3C6410X

S3C6410X 是基于 ARM1176JZFS 核的用于手持、移动等终端设备的通用处理器。S3C6410X 是一款低功率、高性价比、高性能的用于移动电话和通用处理 RSIC 处理器。为 2.5G 和 3G 通信服务提供了优化的硬件性能，采用 64/32 位的内部总线架构，融合了 AXI、AHB、APB 总线。还有很多强大的硬件加速器，包括运动视频处理、音频处理、2D 加速、显示处理和缩放。

1.2 电容式多点触摸屏

电容式触摸屏在触摸屏 4 边均镀上狭长的电极，在导电体内形成一个低电压交流电场。在触摸屏幕时，由于人体电场，手指与导体层间会形成一个耦合电容，4 边电极发出的电流会流向触点，而电流强度与手指到电极的距离成正比，位于触摸屏幕后的控制器会计算电流的比例及强弱，准确算出触摸点的位置。电容触摸屏的双玻璃不但能保护导体及感应器，更有效地防止外在环境因素对触摸屏造成影响，就算屏幕沾有污秽、尘埃或油渍，电容式触摸屏依然能准确算出触摸位置。与电阻触摸屏相对比，电容式触摸屏就是支持多点触摸的人机交互方式，普通电阻式触摸屏只能进行单一点的触控。

1.3 ARM 工具链

本文针对 ARM 核的单片机使用了 arm none linux gnueabi 4.3.2 交叉编译链，实现对 ARM 支持的二进制文件编译，用以成功编译 ARMLinux 2.6.28 内核。

1.4 移植条件

对于本文所述内容，所有支持 Linux 操作系统运行的处理器（包括嵌入式处理器）都可以运行，而所有支持 I2C 总线协议的单片机也可以在不使用操作系统的前提下将触摸屏作为一种普通输入设备进行使用。

2 研究过程

图 1 显示了本文中针对嵌入式 Linux 平台下的驱动软硬件结构体系。

图 1 驱动软硬件结构体系

2.1 I2C 设备在平台部分声明

CYPRESS 7958 多点触摸屏的 I2C 地址为 0x20，在使用前需要在平台设备处进行 I2C 设备声明，这样才可以使 Linux 驱动找到其对应的 I2C 地址进行操作。首先要声明该 I2C 设备结构体，代码如下：

```
static struct i2c_board_info i2c_devs1 [] __initdata={
{I2C_BOARD_INFO（“cypress 7958”， 0x20）， }， /*cypress 7958 touch panel controller*/
};
```

然后在 static void __init smdk6410_machine_init（void）函数中声明该 I2C 设备：

```
i2c_register_board_info（1， i2c_devs1， ARRAY_SIZE（i2c_devs1））;
```

2.2 Cypress 7958 驱动部分设计

2.2.1 注册和注销模块

首先建立 I2C 驱动结构体，cypress_7958_driver，代码如下：

```
static struct i2c_driver cypress_7958_driver={
.probe=cypress_7958_probe，
.remove=cypress_7958_remove，
.id_table=cypress_7958_id，
.driver={
.name=CYPRESS_7958_NAME，
},
};
```

然后建立_INIT 初始化函数与_EXIT 注销设备函数：static int __devinit cypress_7958_ts_init（void），static void __exit cypress_7958_exit（void），通过 i2c_add_driver 与 i2c_del_driver 函数进行 I2C 设备的注册与注销。

2.2.2 触摸屏驱动入口函数的设计

由上节中声明的 I2C 结构体得知，在设备被检查到的时候进入 static int synaptics_ts_probe（struct i2c_client *client， const struct i2c_device_id *id）函数，在该函数中需要进行触摸屏工作模式的初始化，对作为输入设备的触摸屏驱动在 Linux 平台下的设备名注册，同时初始化触摸事件触发时引起的中断操作。

（1） Cypress 7958 模式初始化

作为多点触摸屏幕，Cypress 7958 有很多相关的配置寄存器，本文中不再赘述，初始化部分仅需对屏幕是否工作在正常工作模式下进行检查，通过读取 0x28 地址的寄存器，如果值为 0x07，则屏幕工作正常，否则返回错误值。

```
ret=i2c_smbus_read_byte_data（ts->client， 0x28）；
if（ret！=0x07）{
printk（KERN_ERR，“Cypress Detect Errorn”）；
return ret;
}
```

（2） 输入设备名注册

创建 struct input_dev 结构体，通过 input_allocate_device（）函数进行设备名的创建，然后通过 set_bit 函数进行输入设备功能声明。因为是多点触摸屏，可以产生 EV_SYN，EV_KEY，BTN_TOUCH，BTN_2（多点触摸），EV_ABS 等功能，故对之进行声明：

```
set_bit（EV_SYN， ts->input_dev->evbit）；
set_bit（EV_KEY， ts->input_dev->evbit）；
set_bit（BTN_TOUCH， ts->input_dev->keybit）；
set_bit（BTN_2， ts->input_dev->keybit）；
set_bit（EV_ABS， ts->input_dev->evbit）；
```

然后完成对事件的具体配置：

```
input_set_abs_params（ts->input_dev， ABS_X， 0， max_y， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_Y， 0， max_x， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_PRESSURE， 0， 255， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_TOOL_WIDTH， 0， 15， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_HAT0X， 0， max_y， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_HAT0Y， 0， max_x， 0， 0）；
input_set_abs_params（ts->input_dev， ABS_MT_POSITION_X， 0， max_y， 0，
0）；
input_set_abs_params（ts->input_dev， ABS_MT_POSITION_Y， 0， max_x， 0，
0）；
input_set_abs_params（ts->input_dev， ABS_MT_TOUCH_MAJOR， 0， 255，
0， 0）；
input_set_abs_params（ts->input_dev， ABS_MT_WIDTH_MAJOR， 0， 15， 0，
0）；
```

最后通过 input_register_device（ts->input_dev）函数完成对该设备名的注册。

（3） 驱动事件产生中断函数初始化

Cypress 7958 触摸屏在触摸事件产生时会在 IRQ 引脚产生一个低电平信号，将该引脚连接到 GPN（15）引脚上，同时创建 GPIO 中断函数：

```
s3c_gpio_cfgpin（S3C64XX_GPN（15），S3C_GPIO_SFN（2））；
client->irq=gpio_to_irq（S3C64XX_GPN（15））；
irqflags=IRQF_TRIGGER_LOW;
```

然后通过 ret=request_irq（client->irq， cypress_7958_irq_handler， irqflags， client->name， ts）进行中断函数申请。创建 cypress_7958_irq_handler 函数：

```
static irqreturn_t cypress_7958_irq_handler（int irq， void *dev_id）{
struct synaptics_ts_data *ts=dev_id;
//int ret=gpio_get_value（S3C64XX_GPN（15））；
//printk（“%s:ret=%dn”，__func__，ret）；
disable_irq_nosync（ts->client->irq）；
queue_work（cypress_7958_wq， &ts->work）；
return IRQ_HANDLED;
}
```

当驱动事件被触发之后通过 queue_work 函数进入驱动工作区 cypress_7958_wq，进行驱动层对应用层的信息上报。

2.2.3 触摸屏工作区函数设计

触摸屏工作区函数需要完成事件信息获取以及驱动层对应用层的信息上报功能，通过 INIT_WORK（&ts->work， cypress_7958_work_func）函数完成驱动工作区函数的初始化声明，在驱动事件中断产生之后进入工作区函数 cypress_7958_work_func。

（1） 触摸屏事件信息获取

Cypress 7958 的事件触发信息存储在寄存器中，只需要通过 i2c_smbus_read_byte_data 函数对其寄存器信息进行读取即可完成其事件信息的获取，也可以通过 i2c_transfer 完成对其寄存器信息的批量读取：

```
buf［0］=i2c_smbus_read_byte_data（ts->client， 0x12）；
buf［1］=i2c_smbus_read_byte_data（ts->client， 0x13）；
buf［2］=i2c_smbus_read_byte_data（ts->client， 0x14）；
buf［3］=i2c_smbus_read_byte_data（ts->client， 0x15）；
buf［4］=i2c_smbus_read_byte_data（ts->client， 0x16）；
buf［5］=i2c_smbus_read_byte_data（ts->client， 0x17）；
buf［6］=i2c_smbus_read_byte_data（ts->client， 0x18）；
buf［7］=i2c_smbus_read_byte_data（ts->client， 0x19）；
buf［8］=i2c_smbus_read_byte_data（ts->client， 0x1a）；
buf［9］=i2c_smbus_read_byte_data（ts->client， 0x1b）；
buf［10］=i2c_smbus_read_byte_data（ts->client， 0x1c）；
buf［11］=i2c_smbus_read_byte_data（ts->client， 0x1d）；
buf［12］=i2c_smbus_read_byte_data（ts->client， 0x1e）；
buf［13］=i2c_smbus_read_byte_data（ts->client， 0x1f）；
```

（2） 触摸屏事件信息上报

通过对 buf 数组的分析，获取当前事件具体信息，然后通过 input_report 系列函数进行事件信息的应用层上报：

```
if（fingermark==2）{
input_report_key（ts->input_dev，ABS_MT_TRACKING_ID，0）；
input_report_abs（ts->input_dev， ABS_MT_TOUCH_MAJOR， f1z）；
input_report_abs（ts->input_dev， ABS_MT_POSITION_X， f1x）；
```

```
input_report_abs（ts->input_dev， ABS_MT_POSITION_Y， f1y）；
input_mt_sync（ts->input_dev）；
input_report_key（ts->input_dev，ABS_MT_TRACKING_ID，1）；
input_report_abs（ts->input_dev， ABS_MT_TOUCH_MAJOR， f2z）；
input_report_abs（ts->input_dev， ABS_MT_POSITION_X， f2x）；
input_report_abs（ts->input_dev， ABS_MT_POSITION_Y， f2y）；
input_mt_sync（ts->input_dev）
input_sync（ts->input_dev）；
}
else if（fingermark==1）{
input_report_key（ts->input_dev，ABS_MT_TRACKING_ID，0）；
input_report_abs（ts->input_dev， ABS_MT_TOUCH_MAJOR， f1z）；
input_report_abs（ts->input_dev， ABS_MT_POSITION_X， f1x）；
input_report_abs（ts->input_dev， ABS_MT_POSITION_Y， f1y）；
input_mt_sync（ts->input_dev）；
input_sync（ts->input_dev）；
}
else{
input_report_abs（ts->input_dev， ABS_MT_TOUCH_MAJOR， 0）；
input_mt_sync（ts->input_dev）；
input_sync（ts->input_dev）；
}
```

2.3 Cypress 7958 驱动在内核中的移植

通过改写 Makefile 与 KCONFIG 完成 Cypress 7985 在内核中的移植，以帮助 GCC 工具链实现对内核的编译。

2.3.1 Kconfig 的修改

在/driver/input/touchscreen/Kconfig 中添加如下语句：

```
config TOUCHSCREEN_CYPRESS
tristate “CYPRESS 7958 touchscreens”
help
Say Y here if you have a CYPRESS 7958 touchscreen connected to your system.
If unsure， say N.
```

以实现将文件编译选项添加到 MAKE MENUCONFIG 中。由于触摸屏驱动属于系统基本输入设备驱动，本身调用了 I/O 中断，不能实现模块编译，只能完全编译进内核。在后续的研发中发现可以使用时钟中断将其模块化编译进内核，但由于时钟中断影响 UCLINUX 时间片的运行，故弃之不用。

2.3.2 Makefile 的修改

然后在/driver/input/touchscreen/Makefile 中添加对应编译信息：

obj $（CONFIG_TOUCHSCREEN_CYPRESS） +=touchscreen_cypress.o

最终在编译选项中将/MAKEFILE 中的 ARCH 选项设置为 S3C6410，在 make menuconfig 命令之后的选项中选择 TOUCHSREEN_CYPRESS 选项并选择编译进内核。

结语

本设计以 I2C 方式对多点触摸屏进行驱动，通过嵌入式 Linux 将多点触摸输入方式应用到嵌入式应用系统中，丰富了单一的键盘输入与单点输入方式， 减小了系统尺寸，提高了系统的可靠性。使得嵌入式系统的输入方式简单易行，同时也增强了嵌入式系统与人之间的通信能力，简化了繁琐的调试。采用三星公司的 S3C6410 ARM11 处理器，加快了实验的操作步骤。实践证明，该设计驱动多点触摸屏幕的速度以及稳定性满足调试要求。该设计只需对底层驱动进行简单修改，就可直接应用于单片机以及其他全部可以运行 Linux 的嵌入式系统中。