Rockchip
瑞芯微电子

Rockchip
I2C 开发指南

发布版本:1.0

日期:2017.02

前言

概述

产品版本

芯片名称	内核版本
RK3328	Linux3.10
RK3228H	Linux3.10

读者对象

本文档（本指南）主要适用于以下工程师：
技术支持工程师
软件开发工程师

修订记录

日期	版本	作者	修改说明
2017-02-17	V1.0	WDC	初始版本

目录

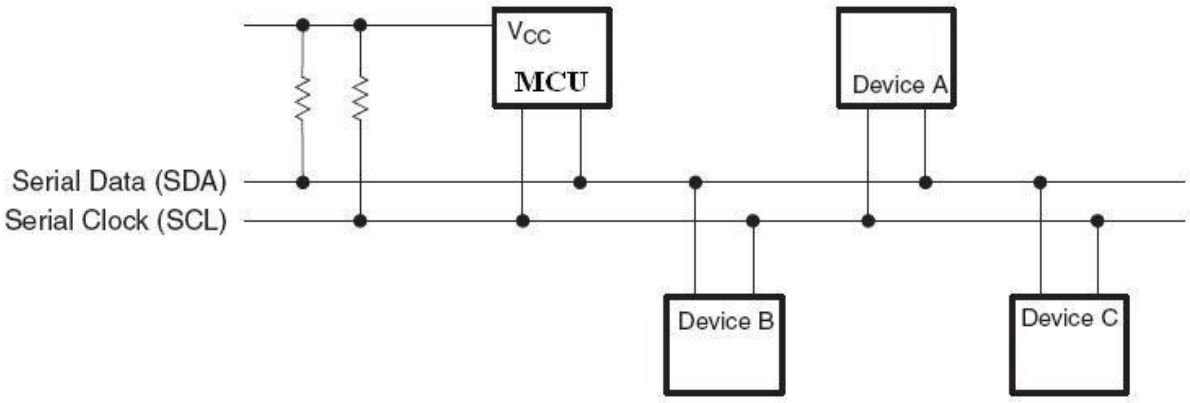
1 RockchipI2C 功能特点

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; I2C 总线控制器通过串行数据（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）。

Rockchip I2C 控制器支持下列功能：

- 兼容 I2C 与 SMBus 总线
- 支持主模式下的 I2C 总线
- 软件可编程时钟频率和传输速率高达 1000kbps
- 支持 7 位和 10 位寻址模式
- 一次中断或轮询至多 32 个字节的数据传输
- 时钟拉伸和等待状态

下图为 I2C 总线的硬件连接方式，需要上拉电阻，改变上拉电阻大小可调节 I2C 总线的上拉强度：



I²C 总线连线图

2 DTS 节点配置

Dts 节点配置可参考 Linux kernel 目录下的文件：

```
Documentation/devicetree/bindings/i2c/i2c-rk3x.txt
```

需要配置项：

1. I2C 速率配置：请查阅 device 的 datasheet，确定可适配的 clock，一般配置 400k，100k(默认，可不填)，200k，1000k；

400k 示例：

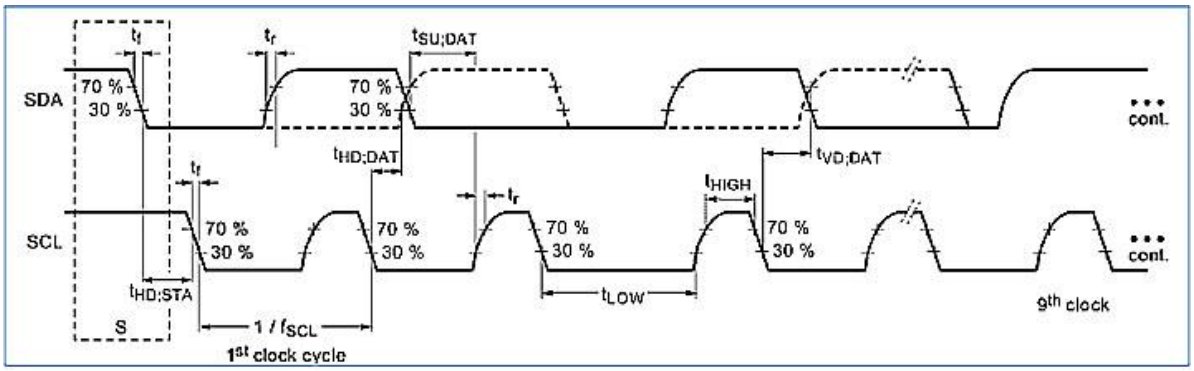
```
clock-frequency = <400000>;
```

2. i2c_clk 上升沿时间,下降沿时间；

当需要 I2C 速率配置超过 100k 时，i2c_clk 上升沿和下降沿时间一定需要通过示波器测量得出；因为 I2C 协议标准对上升沿和下降沿时间有规定，特别是上升沿时间，如果超过了协议规定的最大值，则 I2C 通讯可能失败，下面是协议规定的上升下降沿时间范围：

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		unit
		Min	Max	Min	Max	Min	Max	
fSCL	SCL clock frequency		100		400		1000	KHZ
Tr	rise time of both SDA and SCL signals		1000	20	300		120	ns
Tf	fall time of both SDA and SCL signals		300	20× (VDD/5.5V	300	20× (VDD/5.5V	300	ns

上升沿 Tr，下降沿 Tf，分别取 30%~70%的波形时间：



3. 未配置以上两项，则将默认按上升沿和下降沿的 max 值来计算，得到的 clk 速度将近 90k。默认使用 100k 的 max 值是 1000ns，符合多数硬件，所以如果要求不高的话可以不配置。
4. i2c1+es8316 codec 示例说明，需要 i2c 时钟 400k，示波器所测得 Tr=164ns，Tf=15ns：

```
&i2c1 {
status = "okay";
i2c-scl-rising-time-ns = <164>;
i2c-scl-falling-time-ns = <15>;
clock-frequency = <400000>;

        es8316: es8316@10 {
                #sound-dai-cells = <0>;
compatible = "everest,es8316";
```

```
    reg = <0x10>;
    clocks = <&cru SCLK_I2S_8CH_OUT>;
    clock-names = "mclk";
    spk-con-gpio = <&gpio0 11 GPIO_ACTIVE_HIGH>;
    hp-det-gpio = <&gpio4 28 GPIO_ACTIVE_LOW>;
            };
    };
```

3 代码使用 I2C 接口

驱动文件所在 kernel 位置：drivers/i2c/busses/i2c-rk3x.c，驱动的实现方式可以参考此驱动代码，接下来是读写使用用例。

A. 读数据使用示例：


```
staticint i2c_read_bytes(struct i2c_client *client,
u8cmd, u8 *data, u8 data_len)
{
struct i2c_msg msgs[2];
int ret;
        u8 *buffer;

buffer = kzalloc(data_len, GFP_KERNEL);
if (!buffer)
return-ENOMEM;;

msgs[0].addr = client->addr;
msgs[0].flags = client->flags;
msgs[0].len = 1;
msgs[0].buf = &cmd;

msgs[1].addr = client->addr;
msgs[1].flags = client->flags | I2C_M_RD;
msgs[1].len = data_len;
msgs[1].buf = buffer;

ret = i2c_transfer(client->adapter, msgs, ARRAY_SIZE(msgs));
if (ret < 0)
dev_err(&client->adapter->dev, "i2c read failed\n");
else
memcpy(data, buffer, data_len);

kfree(buffer);
return ret;
}
```

B. 写数据使用示例：

```
staticint i2c_write_bytes(struct i2c_client *client,
u8cmd, u8 *data, u8 data_len)
{
struct i2c_msg msgs[1];
        u8 *buffer;
int ret = 0;

buffer = kzalloc(data_len + 1, GFP_KERNEL);
if (!buffer)
return-ENOMEM;

buffer[0] = cmd;
memcpy(buffer + 1, data, data_len);

msgs[0].addr = client->addr;
msgs[0].flags = client->flags;
msgs[0].len = data_len + 1;
msgs[0].buf = buffer;

ret = i2c_transfer(client->adapter, msgs, ARRAY_SIZE(msgs));
if (ret < 0)
dev_err(&client->adapter->dev, "i2c write failed\n");

kfree(buffer);
return ret;
}
```

4 常见问题

1. 当 log："timeout, ipd: 0x00, state: 1"出现时，请检查硬件上拉是否给电或者 I2C pin 脚的 iomux 值是否设置正确；
2. 如果调用 i2c_transfer 返回值为-6 时候，表示为 NACK 错误，即对方设备无应答响应，这种情况一般为外设的问题，常见的有以下几种情况：
   A. I2C 地址错误，解决方法是测量 i2c 波形，确认是否 i2c 设备地址错误；
   B. I2C slave 设备不处于正常工作状态，比如未给电，错误的上电时序等；
   C. 时序不符合 I2C slave 设备所要求也会产生 NACK 信号，比如下面的第三点；
3. 当外设对于读时序要求中间是 stop 信号,而不是 repeat start 信号的时候，需要调用两次 i2c_transfer，分别将写寄存器地址的操作与读数据操作，作两次 I2C 调用，修改如下：

```
staticint i2c_read_bytes(struct i2c_client *client,
u8cmd, u8 *data, u8 data_len)
{
struct i2c_msg msgs[2];
int ret;
        u8 *buffer;

buffer = kzalloc(data_len, GFP_KERNEL);
if (!buffer)
return-ENOMEM;;

msgs[0].addr = client->addr;
msgs[0].flags = client->flags;
msgs[0].len = 1;
msgs[0].buf = &cmd;
ret = i2c_transfer(client->adapter, msgs, 1);
if (ret < 0) {
dev_err(&client->adapter->dev, "i2c read failed\n");
                kfree(buffer);
            return ret;
        }

msgs[1].addr = client->addr;
msgs[1].flags = client->flags | I2C_M_RD;
msgs[1].len = data_len;
msgs[1].buf = buffer;

ret = i2c_transfer(client->adapter, &msgs[1], 1);
if (ret < 0)
dev_err(&client->adapter->dev, "i2c read failed\n");
else
memcpy(data, buffer, data_len);
```

```
kfree(buffer);
return ret;
}
```