Rockchip Linux SPI

文件标识：RK-KF-YF-020

发布版本：V2.0.0

日期：2019-12-03

文件密级：内部资料

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文介绍 Linux SPI 驱动原理和基本调试方法。

产品版本

芯片名称	内核版本
采用 linux4.4 的所有芯片	Linux4.4
采用 linux4.19 的所有芯片	Linux4.19

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

版本号	作者	修改日期	修改说明
V1.0.0	洪慧斌	2016-06-29	初始版本
V2.0.0	林鼎强	2019-12-03	新增 linux4.19 支持

目录

---

1 Rockchip SPI 功能特点

SPI （serial peripheral interface），以下是 linux 4.4 spi 驱动支持的一些特性：

- 默认采用摩托罗拉 SPI 协议
- 支持 8 位和 16 位
- 软件可编程时钟频率和传输速率高达 50MHz
- 支持 SPI 4 种传输模式配置
- 每个 SPI 控制器支持一个到两个片选

除以上支持，linux 4.19 新增以下特性：

- 框架支持 slave 和 master 两种模式

2 内核软件

2.1 代码路径

```
drivers/spi/spi.c             spi驱动框架
drivers/spi/spi-rockchip.c    rk spi各接口实现
drivers/spi/spidev.c          创建spi设备节点，用户态使用。
drivers/spi/spi-rockchip-test.c  spi测试驱动，需要自己手动添加到Makefile编译
Documentation/spi/spidev_test.c  用户态spi测试工具
```

2.2 内核配置

```
Device Drivers  --->
    [*] SPI support  --->
        <*>   Rockchip SPI controller driver
```

2.3 DTS 节点配置

```
&spi1 {                                 引用spi 控制器节点
status = "okay";
max-freq = <48000000>;                  spi内部工作时钟
dma-names = "tx","rx";                  使能DMA模式，一般通讯字节少于32字节的不建议用
    spi_test@10 {
        compatible ="rockchip,spi_test_bus1_cs0";  与驱动对应的名字
        reg = <0>;                      片选0或者1
        spi-max-frequency = <24000000>;   spi clk输出的时钟频率，不超过50M
        spi-cpha;                       如果有配，cpha为1
        spi-cpol;                       如果有配，cpol为1,clk脚保持高电平
        status = "okay";                使能设备节点
    };
};
```

一般只需配置以下几个属性就能工作了。

```
        spi_test@11 {
                compatible ="rockchip,spi_test_bus1_cs1";
                reg = <1>;
                spi-max-frequency = <24000000>;
                status = "okay";
        };
```

max-freq 和 spi-max-frequency 的配置说明：

- spi-max-frequency 是 SPI 的输出时钟，是 max-freq 分频后输出的，关系是 max-freq >= 2*spi-max-frequency。
- max-freq 不要低于 24M，否则可能有问题。
- 如果需要配置 spi-cpha 的话， max-freq <= 6M, 1M <= spi-max-frequency >= 3M。

2.3 SPI 设备驱动

设备驱动注册:

```
static int spi_test_probe(struct spi_device *spi)
{
        int ret;
        int id = 0;
        if(!spi)
            return -ENOMEM;
        spi->bits_per_word= 8;
        ret= spi_setup(spi);
        if(ret < 0) {
            dev_err(&spi->dev,"ERR: fail to setup spi\n");
            return-1;
        }
        return ret;
}
static int spi_test_remove(struct spi_device *spi)
{
        printk("%s\n",__func__);
        return 0;
}
static const struct of_device_id spi_test_dt_match[]= {
        {.compatible = "rockchip,spi_test_bus1_cs0", },
        {.compatible = "rockchip,spi_test_bus1_cs1", },
        {},
};
MODULE_DEVICE_TABLE(of,spi_test_dt_match);
static struct spi_driver spi_test_driver = {
        .driver = {
            .name  = "spi_test",
            .owner = THIS_MODULE,
            .of_match_table = of_match_ptr(spi_test_dt_match),
        },
        .probe = spi_test_probe,
        .remove = spi_test_remove,
};
static int __init spi_test_init(void)
{
        int ret = 0;
        ret = spi_register_driver(&spi_test_driver);
        return ret;
}
device_initcall(spi_test_init);
static void __exit spi_test_exit(void)
{
        return spi_unregister_driver(&spi_test_driver);
}
module_exit(spi_test_exit);
```

对 spi 读写操作请参考 include/linux/spi/spi.h，以下简单列出几个

```
static inline int
spi_write(struct spi_device *spi,const void *buf, size_t len)
static inline int
spi_read(struct spi_device *spi,void *buf, size_t len)
static inline int
spi_write_and_read(structspi_device *spi, const void *tx_buf, void *rx_buf,
size_t len)
```

2.4 User mode SPI device 配置说明

User mode SPI device 指的是用户空间直接操作 SPI 接口，这样方便众多的 SPI 外设驱动跑在用户空间，不需要改到内核，方便驱动移植开发。

2.4.1 内核配置

```
Device Drivers  --->
    [*] SPI support  --->
        [*]   User mode SPI device driver support
```

2.4.2 DTS 配置

```
&spi0 {
    status = "okay";
    max-freq = <50000000>;
    spi_test@00 {
        compatible = "rockchip,spidev";
        reg = <0>;
        spi-max-frequency = <5000000>;
    };
};
```

2.4.3 内核补丁

```
diff --git a/drivers/spi/spidev.c b/drivers/spi/spidev.c
index d0e7dfc..b388c32 100644
--- a/drivers/spi/spidev.c
+++ b/drivers/spi/spidev.c
@@ -695,6 +695,7 @@ static struct class *spidev_class;
static const struct of_device_id spidev_dt_ids[] = {
        { .compatible = "rohm,dh2228fv" },
        { .compatible = "lineartechnology,ltc2488" },
+       { .compatible = "rockchip,spidev" },
        {},
};
MODULE_DEVICE_TABLE(of, spidev_dt_ids);
```

说明：较旧的内核可能没有 2.4.1 和 2.4.3 ，需要手动添加，如果已经包含这两个的内核，只要添加 2.4.2 即可。

2.4.4 使用说明

驱动设备加载注册成功后，会出现类似这个名字的设备：/dev/spidev1.1

请参照 Documentation/spi/spidev_test.c

2.5 SPI 做 slave

使用的接口和 master 模式一样，都是 spi_read 和 spi_write。

内核补丁，请先检查下自己的代码是否包含以下补丁，如果没有，请手动打上补丁：

```
diff --git a/drivers/spi/spi-rockchip.c b/drivers/spi/spi-rockchip.c
index 060806e..38eecdc 100644
```

```
--- a/drivers/spi/spi-rockchip.c
+++ b/drivers/spi/spi-rockchip.c
@@ -519,6 +519,8 @@ static void rockchip_spi_config(struct rockchip_spi *rs)
        cr0 |= ((rs->mode & 0x3) << CR0_SCPH_OFFSET);
        cr0 |= (rs->tmode << CR0_XFM_OFFSET);
        cr0 |= (rs->type << CR0_FRF_OFFSET);
+       if (rs->mode & SPI_SLAVE_MODE)
+               cr0 |= (CR0_OPM_SLAVE << CR0_OPM_OFFSET);

        if (rs->use_dma) {
                if (rs->tx)
@@ -734,7 +736,7 @@ static int rockchip_spi_probe(struct platform_device
*pdev)

        master->auto_runtime_pm = true;
        master->bus_num = pdev->id;
-       master->mode_bits = SPI_CPOL | SPI_CPHA | SPI_LOOP;
+       master->mode_bits = SPI_CPOL | SPI_CPHA | SPI_LOOP | SPI_SLAVE_MODE;
        master->num_chipselect = 2;
        master->dev.of_node = pdev->dev.of_node;
        master->bits_per_word_mask = SPI_BPW_MASK(16) | SPI_BPW_MASK(8);
diff --git a/drivers/spi/spi.c b/drivers/spi/spi.c
index dee1cb8..4172da1 100644
--- a/drivers/spi/spi.c
+++ b/drivers/spi/spi.c
@@ -1466,6 +1466,8 @@ of_register_spi_device(struct spi_master *master,
struct device_node *nc)
                spi->mode |= SPI_3WIRE;
        if (of_find_property(nc, "spi-lsb-first", NULL))
                spi->mode |= SPI_LSB_FIRST;
+       if (of_find_property(nc, "spi-slave-mode", NULL))
+               spi->mode |= SPI_SLAVE_MODE;

        /* Device DUAL/QUAD mode */
        if (!of_property_read_u32(nc, "spi-tx-bus-width", &value)) {
diff --git a/include/linux/spi/spi.h b/include/linux/spi/spi.h
index cce80e6..ce2cec6 100644
--- a/include/linux/spi/spi.h
+++ b/include/linux/spi/spi.h
@@ -153,6 +153,7 @@ struct spi_device {
 #define        SPI_TX_QUAD     0x200                   /* transmit with 4
wires */
 #define        SPI_RX_DUAL     0x400                   /* receive with 2
wires */
 #define        SPI_RX_QUAD     0x800                   /* receive with 4
wires */
+#define        SPI_SLAVE_MODE  0x1000                  /* enable spi slave
mode */
        int                     irq;
        void                    *controller_state;
        void                    *controller_data;
```

dts 配置：

```
&spi0 {
    max-freq = <48000000>;   //spi internal clk, don't modify
    spi_test@01 {
            compatible = "rockchip,spi_test_bus0_cs1";
            id = <1>;
            reg = <1>;
            //spi-max-frequency = <24000000>;  这不需要配
            spi-cpha;
            spi-cpol;
            spi-slave-mode; 使能slave 模式， 只需改这里就行。
    };
};
```

注意：max-freq 必须是 master clk 的 6 倍以上，比如 max-freq = <48000000>; master 给过来的时钟必须小于 8M。

测试：

spi 做 slave， 要先启动 slave read，再启动 master write，不然会导致 slave 还没读完，master 已经写完了。

slave write，master read 也是需要先启动 slave write，因为只有 master 送出 clk 后，slave 才会工作，同时 master

会立即发送或接收数据。

在第三章节的基础上：

先 master : `echo write 0 1 16 > /dev/spi_misc_test`

再 slave: `echo read 0 1 16 > /dev/spi_misc_test`

3 SPI 内核测试驱动

3.1 内核驱动

```
drivers/spi/spi-rockchip-test.c
需要手动添加编译：
drivers/spi/Makefile
+obj-y                                  += spi-rockchip-test.o
```

3.2 DTS 配置

```
&spi0 {
        status = "okay";
        max-freq = <48000000>;   //spi internal clk, don't modify
        //dma-names = "tx", "rx";   //enable dma
        pinctrl-names = "default";  //pinctrl according to you board
        pinctrl-0 = <&spi0_clk &spi0_tx &spi0_rx &spi0_cs0 &spi0_cs1>;
        spi_test@00 {
                compatible = "rockchip,spi_test_bus0_cs0";
                id = <0>;       //这个属性spi-rockchip-test.c用来区分不同的spi从设备的
                reg = <0>;   //chip select  0:cs0  1:cs1
                spi-max-frequency = <24000000>;   //spi output clock
                //spi-cpha;      //not support
                //spi-cpol;     //if the property is here it is 1:clk is high, else 0:clk is low  when idle
        };

        spi_test@01 {
                compatible = "rockchip,spi_test_bus0_cs1";
                id = <1>;
                reg = <1>;
                spi-max-frequency = <24000000>;
                spi-cpha;
                spi-cpol;
                spi-slave-mode; 使能slave 模式， 只需改这里就行。
        };
};
```

3.3 驱动 log

```
[    0.530204] spi_test spi32766.0: fail to get poll_mode, default set 0
[    0.530774] spi_test spi32766.0: fail to get type, default set 0
[    0.531342] spi_test spi32766.0: fail to get enable_dma, default set 0
以上这几个没配的话，不用管
[    0.531929] rockchip_spi_test_probe:name=spi_test_bus1_cs0,bus_num=32766,cs=0,mode=0,speed=5000000
[    0.532711] rockchip_spi_test_probe:poll_mode=0, type=0, enable_dma=0
这是驱动注册成功的标志
```

3.4 测试命令

```
echo write 0 10 255 > /dev/spi_misc_test
echo write 0 10 255 init.rc > /dev/spi_misc_test
echo read 0 10 255 > /dev/spi_misc_test
echo loop 0 10 255 > /dev/spi_misc_test
echo setspeed 0 1000000 > /dev/spi_misc_test
```

echo 类型 id 循环次数 传输长度 > /dev/spi_misc_test

echo setspeed id 频率（单位 Hz） > /dev/spi_misc_test

如果需要，可以自己修改测试 case。

4 常见问题

- 调试前确认驱动有跑起来
- 确保 SPI 4 个引脚的 IOMUX 配置无误
- 确认 TX 送时，TX 引脚有正常的波形，CLK 有正常的 CLOCK 信号，CS 信号有拉低
- 如果 clk 频率较高，可以考虑提高驱动强度来改善信号