Rockchip Linux SPI

文件标识：RK-KF-YF-020

发布版本：V2.4.0

日期：2021-08-31

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

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
V2.1.0	林鼎强	2020-02-13	修改 SPI slave 配置
V2.2.0	林鼎强	2020-07-14	修订 Linux 4.19 DTS 相关配置，优化文档排版结构
V2.3.0	林鼎强	2020-11-02	新增 spi-bus cs-gpios 属性的支持说明
V2.3.1	林鼎强	2020-12-11	修订 Linux4.4 SPI slave 说明
V2.3.2	林鼎强	2021-07-06	增加参数配置说明、增加 cs-gpios 应用注意点
V2.4.0	林鼎强	2021-08-31	增加常见问题说明、减少冗余的配置

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 SPI 设备配置 —— RK 芯片作 Master 端

内核配置

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

DTS 节点配置

```
&spi1 {                                                 //引用spi 控制器节点
    status = "okay";
    //assigned-clock-rates = <200000000>;               //默认不用配置，SPI 设备工作时钟
值，io 时钟由工作时钟分频获取
    dma-names = "tx","rx";                              //使能DMA模式，通讯长度少于32字节
不建议用，置空赋值去掉使能，如 "dma-names;";
    //rx-sample-delay-ns = <10>;                        //默认不用配置，读采样延时，详细参
考 “常见问题”“延时采样时钟配置方案” 章节
    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-cpha;                                       //设置 CPHA = 1，不配置则为 0
        spi-cpol;                                       //设置 CPOL = 1，不配置则为 0
        spi-lsb-first;                                  //IO 先传输 lsb
        spi-max-frequency = <24000000>;                 //spi clk输出的时钟频率，不超过
50M
        status = "okay";                                //使能设备节点
    };
};
```

spiclk assigned-clock-rates 和 spi-max-frequency 的配置说明：

- spi-max-frequency 是 SPI 的输出时钟，由 SPI 工作时钟 spiclk assigned-clock-rates 内部分频后输出，由于内部至少 2 分频，所以关系是 spiclk assigned-clock-rates >= 2*spi-max-frequency；
- 假定需要 50MHz 的 SPI IO 速率，可以考虑配置（记住内部分频为偶数分频）spi_clk assigned-clock-rates = <100000000>，spi-max-frequency = <50000000>，即工作时钟 100 MHz（PLL 分频到一个不大于 100MHz 但最接近的值），然后内部二分频最终 IO 接近 50 MHz；
- spiclk assigned-clock-rates 不要低于 24M，否则可能有问题；
- 如果需要配置 spi-cpha 的话， 要求 spiclk assigned-clock-rates <= 6M，1M <= spi-max-frequency >= 3M。

2.3 SPI 设备配置 —— RK 芯片作 Slave 端

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

2.3.1 Linux 4.4 配置

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 节点配置

```
&spi0 {
    assigned-clocks = <&pmucru CLK_SPI0>;            //指定正确的 SPI sclk，可以通过
查看 dtsi 中命名为 spiclk 的时钟
    assigned-clock-rates = <200000000>;              //相应 clock 在解析 dts 时完成赋
值
    spi_test@01 {
        compatible = "rockchip,spi_test_bus0_cs1";
        id = <1>;
        reg = <1>;
        //spi-max-frequency = <24000000>;  这不需要配
        spi-slave-mode; //使能slave 模式， 只需改这里就行。
    };
};
```

注意：

1. The working clock must be more than 6 times of the IO clock sent by the master. For example, if the assigned clock rates are < 48000000 >, then the clock sent by the master must be less than 8m
   报错 笔记
   双语对照
2. 内核 4.4 框架并未对 SPI slave 做特殊优化，所以传输存在以下两种状态：
    1. DMA 传输：传输发起后流程进入等待 completion 的超时机制，可以通过 dts 调整 “ dma-names;” 来关闭 DMA 传输 dma-names

2. CPU 传输：while 在底层驱动等待传输完成，CPU 忙等

3. 使用 RK SPI 作为 slave，可以考虑以下几种场景：

    1. 关闭 DMA，仅使用 CPU 阻塞传输
    2. 传输均设置大于 32 byte，走 DMA 传输，传输等待 completion 超时机制
    3. 主从之间增加一个 gpio，主设备输出来通知从设备 transfer ready 来减少 CPU 忙等时间

2.3.2 Linux 4.19 配置

内核配置

```
Device Drivers  --->
    [*] SPI support  --->
        [*]   SPI slave protocol handlers
```

DTS 节点配置

```
&spi1 {
    status = "okay";
    dma-names = "tx","rx";
    spi-slave;                                          //使能 slave 模式
    slave {                                             //按照框架要求，SPI slave
子节点的命名需以 "slave" 开始
        compatible ="rockchip,spi_test_bus1_cs0";
        reg = <0>;
        id = <0>;
    };
};
```

注意：

- 实际使用场景可以考虑主从之间增加一个 gpio，主设备输出来通知从设备 transfer ready 来减少 CPU 忙等时间

2.3.3 SPI Slave 测试须知

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。

例如：在第三章节的基础上：

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

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

2.4 SPI 设备驱动介绍

设备驱动注册:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/of.h>
#include <linux/spi/spi.h>

static int spi_test_probe(struct spi_device *spi)
{
    int ret;

    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;
}
module_init(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)
```

2.5 User mode SPI device 配置

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。

内核配置

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

DTS 配置

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

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

设备节点的读写操作例程请参照:

- 内核 4.4 Documentation/spi/spidev_test.c
- 内核 4.19 及以后 tools/spi/spidev_test.c
- 可在内核工程编译后，进入对应路径，输入以下命令直接编译标准 SPI app 程序：

```
make CROSS_COMPILE=~/path-to-toolchain/gcc-xxxxx-toolchain/bin/xxxx-linux-gnu-
    # 选择 kernel 所用 CROSS_COMPILE
```

支持配置为 SPI slave 设备，参考 “SPI 设备配置 —— RK 芯片做 Slave 端”，其中 DTS 配置 sub node 应保持为 "rockchip,spidev"

2.6 cs-gpios 支持

用户可以通过 spi-bus 的 cs-gpios 属性来实现 gpio 模拟 cs 以扩展 SPI 片选信号，cs-gpios 属性详细信息可以查阅内核文档 `Documentation/devicetree/bindings/spi/spi-bus.txt`。

2.6.1 Linux 4.4 配置

该支持需要较多支持补丁，请联系 RK 工程师获取相应的补丁。

2.6.2 Linux 4.19 配置

以 SPI1 设定 GPIO0_C4 为 spi1_cs2n 扩展脚为例。

设置 **cs-gpio** 脚并在 **SPI** 节点中引用

```
diff --git a/arch/arm/boot/dts/rv1126-evb-v10.dtsi b/arch/arm/boot/dts/rv1126-
evb-v10.dtsi
index 144e9edf1831..c17ac362289e 100644
--- a/arch/arm/boot/dts/rv1126-evb-v10.dtsi
+++ b/arch/arm/boot/dts/rv1126-evb-v10.dtsi

&pinctrl {
        ...
+
+       spi1 {
+               spi1_cs0n: spi1-cs1n {
+                       rockchip,pins =
+                               <0 RK_PC2 RK_FUNC_GPIO
&pcfg_pull_up_drv_level_0>;
+               };
+               spi1_cs1n: spi1-cs1n {
+                       rockchip,pins =
+                               <0 RK_PC3 RK_FUNC_GPIO
&pcfg_pull_up_drv_level_0>;
+               };
+               spi1_cs2n: spi1-cs2n {
+                       rockchip,pins =
+                               <0 RK_PC4 RK_FUNC_GPIO
&pcfg_pull_up_drv_level_0>;
+               };
+       };
};

diff --git a/arch/arm/boot/dts/rv1126.dtsi b/arch/arm/boot/dts/rv1126.dtsi
index 351bc668ea42..986a85f13832 100644
--- a/arch/arm/boot/dts/rv1126.dtsi
+++ b/arch/arm/boot/dts/rv1126.dtsi

spi1: spi@ff5b0000 {
```

```
        compatible = "rockchip,rv1126-spi", "rockchip,rk3066-spi";
        reg = <0xff5b0000 0x1000>;
        interrupts = <GIC_SPI 11 IRQ_TYPE_LEVEL_HIGH>;
        #address-cells = <1>;
        #size-cells = <0>;
        clocks = <&cru CLK_SPI1>, <&cru PCLK_SPI1>;
        clock-names = "spiclk", "apb_pclk";
        dmas = <&dmac 3>, <&dmac 2>;
        dma-names = "tx", "rx";
        pinctrl-names = "default", "high_speed";
-       pinctrl-0 = <&spi1m0_clk &spi1m0_cs0n &spi1m0_cs1n &spi1m0_miso
&spi1m0_mosi>;
-       pinctrl-1 = <&spi1m0_clk_hs &spi1m0_cs0n &spi1m0_cs1n &spi1m0_miso_hs
&spi1m0_mosi_hs>;
+       pinctrl-0 = <&spi1m0_clk &spi1_cs0n &spi1_cs1n &spi1_cs2n &spi1m0_miso
&spi1m0_mosi>;
+       pinctrl-1 = <&spi1m0_clk_hs &spi1_cs0n &spi1_cs1n &spi1_cs2n
&spi1m0_miso_hs &spi1m0_mosi_hs>
        status = "disabled";
};
```

SPI 节点重新指定 **cs** 脚

```
+&spi1 {
+       status = "okay";
+       max-freq = <48000000>;
+       cs-gpios = <&gpio0 RK_PC2 GPIO_ACTIVE_LOW>, <&gpio0 RK_PC3
GPIO_ACTIVE_LOW>, <&gpio0 RK_PC4 GPIO_ACTIVE_LOW>;
        spi_test@00 {
                compatible = "rockchip,spi_test_bus1_cs0";
...
+       spi_test@02 {
+               compatible = "rockchip,spi_test_bus1_cs2";
+               id = <2>;
+               reg = <0x2>;
+               spi-cpha;
+               spi-cpol;
+               spi-lsb-first;
+               spi-max-frequency = <16000000>;
+       };
};
```

注释：

- 如果要扩展 cs-gpio，则所有 cs 都要转为 gpio function，用 cs-gpios 扩展来支持

3. 内核测试软件

3.1 代码路径

```
drivers/spi/spi-rockchip-test.c
```

3.2 SPI 测试设备配置

内核补丁

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

DTS 配置

```
&spi0 {
    status = "okay";
    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_test@01 {
        compatible = "rockchip,spi_test_bus0_cs1";
        id = <1>;
        reg = <1>;
        spi-max-frequency = <24000000>;
    };
};
```

驱动 **log**

```
[    0.457137]
rockchip_spi_test_probe:name=spi_test_bus0_cs0,bus_num=0,cs=0,mode=11,speed=16000
000
[    0.457308]
rockchip_spi_test_probe:name=spi_test_bus0_cs1,bus_num=0,cs=1,mode=11,speed=16000
000
```

3.3 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

4.1 SPI 无信号

- 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
- 如何简单判断 SPI DMA 是否使能，串口打印如无以下关键字则 DMA 使能成功：

```
[    0.457137] Failed to request TX DMA channel
[    0.457237] Failed to request RX DMA channel
```

4.2 如何编写 SPI 应用代码

请选择合适的目标函数接口再编写驱动。

自定义 **SPI** 设备驱动

参考 “SPI 设备驱动介绍” 编写，实例如： drivers/spi/spi-rockchip-test.c。

基于 **spidev** 标准设备节点编写的应用程序

参考 “User mode SPI device 配置” 章节。

4.3 延迟采样时钟配置方案

对于 SPI io 速率较高的情形，正常 SPI mode 可能依旧无法匹配外接器件输出延时，RK SPI master read 可能无法采到有效数据，需要启用 SPI rsd 逻辑来延迟采样时钟。

RK SPI rsd（read sample delay）控制逻辑有以下特性：

- 可配值为 0，1，2，3
- 延时单位为 1 spi_clk cycle，即控制器工作时钟，详见 "SPI 设备配置章节“

rx-sample-delay 实际延时为 dts 设定值最接近的 rsd 有效值为准，以 spi_clk 200MHz，周期 5ns 为例：

rsd 实际可配延迟为 0，5ns，10ns，15ns，rx-sample-delay 设定 12ns，接近有效值 10ns，所以最终为 10ns 延时。