Rockchip SPI Developer Guide

ID: RK-KF-YF-075

Release Version: V2.0.0

Release Date: 2019-12-09

Security Level: Non-confidential

Fuzhou Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website： www.rock-chips.com

Customer service Tel： +86-4007-700-590

Customer service Fax： +86-591-83951833

Customer service e-Mail： fae@rock-chips.com

Preface

Overview

This article introduces the Linux SPI driver principle and basic debugging methods.

Product Version

Chipset	Kernel Version
All chips develop in linux4.4	Linux 4.4
All chips develop in linux4.19	Linux 4.19

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers
Software development engineers

Revision History

Version	Author	Date	Change Description
V1.0.0	Huibin Hong	2016-06-29	Initial version
V2.0.0	Dingqiang Lin	2019-12-09	Support Linux 4.19

Content

---

1 Feature of Rockchip SPI

The serial peripheral interface is called SPI, the following are some of the features supported by the Linux 4.4 SPI driver:

- Motorola SPI protocol is used by default
- Supports 8-bit and 16-bit
- Software programmable clock frequency and transfer rate up to 50MHz
- Support 4 transfer mode configurations of SPI
- One or two chips selects per SPI controller

the following are some of the new features supported by the Linux 4.19 SPI driver:

- Support both slave and master mode

2 Kernel Software

2.1 Code Path

```
drivers/spi/spi.c               /* SPI Driver framework */
drivers/spi/spi-rockchip.c      /* RK SPI  implement of interface */
drivers/spi/spidev.c            /* Create  SPI  device node for using */
drivers/spi/spi-rockchip-test.c /* SPI test driver, it needs to add to Makefile
compiler manually. */
Documentation/spi/spidev_test.c /* SPI test tool in user state */
```

2.2 Kernel Configuration

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

2.3 DTS Node Configuration

```
&spi1 {                               //Quote SPI controller node
status = "okay";
max-freq = <48000000>;                //SPI internal clock
dma-names = "tx","rx";                //Enable DMA mode, it is not recommended if
the general communication byte is less than 32 bytes
    spi_test@10 {
        compatible ="rockchip,spi_test_bus1_cs0";    //The name corresponding to
the driver
        reg = <0>;                    //Chip select 0 or 1
        spi-max-frequency = <24000000>; //This is clock frequency of spi clk
output,witch does not exceed 50M.
        spi-cpha;                     //If configure it, cpha is 1
        spi-cpo;                      //If configure it,cpol is 1, the clk pin
remains high level.
        status = "okay";              //Enable device node
    };
};
```

Typically, SPI can work by only configuring the following properties.

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

Configuration instructions for max-freq and spi-max-frequency:

- spi-max-frequency is the output clock of the SPI, which is output after max-freq was divided, and the relationship between them is max-freq >= 2*spi-max-frequency.
- max-freq should not be lower than 24M, otherwise there may be problems.
- If you need to configure spi-cpha, max-freq <= 6M, 1M <= spi-max-frequency >= 3M.

2.3 SPI Device Driver

Register device driver:

```
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);
```

For SPI read and write operations, please refer to `include/linux/spi/spi.h`.

```
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 Configuration

User mode SPI device means operating the SPI interface in user space directly, which makes it convenient for many SPI peripheral drivers run in user space.

There is no need to change the kernel to facilitate driver development.

2.4.1 Kernel Configuration

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

2.4.2 DTS Configuration

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

2.4.3 Kernel Patch

```
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);
```

Note: The legacy kernels may not have 2.4.1 and 2.4.3 and need to be added manually. If there already have both cores, just add 2.4.2.

2.4.4 Using Instruction

After the driver device is successfully registered, a device like this name will be displayed: /dev/spidev1.1

Please refer to Documentation/spi/spidev_test.c

2.5 Independent SPI slave configuration

About kernel patch of the slave, please check if your code contains the following patches, if not, please add the patch:

```
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 configuration：

```
    &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>; " is no need
                spi-cpha;
                spi-cpol;
                spi-slave-mode; //if enble slave mode,just modify here
        };
    };
```

Note: max-freq must be more than 6 times larger than master clk, such as max-freq = <48000000>; the clock given by master must be less than 8M.

Testing:

If SPI working as slave, you must start" slave read" and then start "master write". Otherwise, the slave will not finish reading and the master has finished writing.

If it is slave write , then master read, also needs to start slave write first, because only slave sends clock, slave will work, and master will sent or received data immediately.

Based on the third chapter:

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

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

3 SPI Testing Driver in Kernel

3.1 Kernel Driver

drivers/spi/spi-rockchip-test.c
You need to add below :

```
drivers/spi/Makefile
+obj-y                                  += spi-rockchip-test.o
```

3.2 DTS Configuraion

```
&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>;       //This attribute is used to distinguish
different SPI slave devices in "spi-rockchip-test.c".
                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;
        };
};
```

3.3 Driver 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
//If you donnot use it,please ignore it.
[    0.531929]
rockchip_spi_test_probe:name=spi_test_bus1_cs0,bus_num=32766,cs=0,mode=0,speed=5
000000
[    0.532711] rockchip_spi_test_probe:poll_mode=0, type=0, enable_dma=0
//This is the mark of succesful register.
```

3.4 Test Command

```
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
```

The above means:

Echo type id number of loops transfer length > /dev/spi_misc_test

Echo setspeed id frequency (in Hz) > /dev/spi_misc_test

You can modify the test case by yourself if you want.

4 FAQ

- Confirm that the driver is running before debugging
- Ensure that the IOMUX configuration of the SPI 4 pins is correct .
- Confirm that during the TX sending, the TX pin has a normal waveform, CLK has a normal CLOCK signal, and the CS signal is pulled low.
- If the clock frequency is high, considering increasing the drive strength to improve the signal.