Rockchip SPI Developer Guide

ID: RK-KF-YF-075

Release Version: V2.3.0

Release Date: 2020-11-02

Security Level: □Top-Secret □Secret □Internal ■Public

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
V2.1.0	Dingqiang Lin	2020-02-13	Adjust SPI slave configuration
V2.2.0	Dingqiang Lin	2020-07-14	Linux 4.19 DTS configuration change, Optimize document layout
V2.3.0	Dingqiang Lin	2020-11-02	Add comment for supporting spi-bus cs-gpios property
V2.3.1	Dingqiang Lin	2020-12-11	Update Linux 4.4 SPI slave description

Contents

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 SPI Device Configuration: RK SPI As Master Port

Kernel Configuration

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

DTS Node Configuration

```
&spi1 {                                  //Quote SPI controller node
    status = "okay";
    assigned-clocks = <&pmucru CLK_SPI0>;           //To specify SPI SCLK, you
can view the clock named spiclk in dtsi
    assigned-clock-rates = <200000000>;             //The corresponding clock
completes the assignment when parsing DTS
    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-cpha;                           //If configure it, cpha is 1
        spi-cpol;                           //If configure it,cpol is 1, the clk pin
remains high level.
        spi-lsb-first;                      //IO firstly transfer lsb
        status = "okay";                    //Enable device node
        spi-max-frequency = <24000000>;  //This is clock frequency of spi clk
output,witch does not exceed 50M.
    };
};
```

Configuration instructions for spiclk assigned-clock-rates and spi-max-frequency:

- spi-max-frequency is the output clock of SPI. spi-max-frequency is output after internal frequency division of SPI working clock spiclk in assigned-clock-rates. Since there are at least 2 internal frequency divisions, the relationship is that SPI assigned clock rates > = 2 * SPI Max frequency;
- Assume that we want 50MHz SPI IO rate，the configuration can be set as: spiclk assigned-clock-rates = <100000000>，spi-max-frequency = <50000000>.
- spiclk assigned-clock-rates should not be lower than 24M, otherwise there may be problems.
- If you need to configure spi-cpha, spiclk assigned-clock-rates <= 6M, 1M <= spi-max-frequency >= 3M.

2.3 SPI Device Configuration: RK SPI As Slave Port

The interfaces "spi_read" and "spi_write" of SPI slave are the same as SPI master.

2.3.1 Linux 4.4 configuration

Kernel Patch

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 {
    assigned-clocks = <&pmucru CLK_SPI0>;           //To specify SPI SCLK, you
can view the clock named spiclk in dtsi
    assigned-clock-rates = <200000000>;             //The corresponding clock
completes the assignment when parsing DTS
    spi_test@01 {
        compatible = "rockchip,spi_test_bus0_cs1";
        id = <1>;
        reg = <1>;
        //spi-max-frequency = <24000000>; is no need
        spi-slave-mode;      //if enble slave mode,just modify here
    };
};
```

Note:

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 8MHz
2. The Linux 4.4 framework does not make special optimization for SPI slave, so there are two kinds of transmission states:

    1. DMA transmission: after transmission initiation, the process enters the timeout mechanism of waiting for completion, and the DMA names of DMA transmission can be closed by adjusting "DMA names;" by DTS
    2. CPU transmission: while waits for the transmission to complete in the underlying driver, and CPU is busy
3. Using RK SPI as a slave, you can consider the following scenarios:
    1. Turn off DMA and block transmission only with CPU
    2. If the transmission is set to be greater than 32 bytes, DMA transmission is used, and the transmission waiting for completion timeout mechanism
    3. A GPIO is added between the master and slave devices, and the master device outputs the message to the slave device to transfer ready to reduce the CPU busy waiting time

2.3.2 Linux 4.19 configuration

Kernel Configuration

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

DTS configuration

```
&spi1 {
    status = "okay";
    assigned-clocks = <&pmucru CLK_SPI0>;
    assigned-clock-rates = <200000000>;
    dma-names = "tx","rx";
    spi-slave;                                            //enable slave mode
    slave {                                               //As spi-bus requied,
SPI slave sub-node should name start with "slave"
        compatible ="rockchip,spi_test_bus1_cs0";
        reg = <0>;
        id = <0>;
        //spi-max-frequency = <24000000>; is no need
    };
};
```

Note:

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 8MHz
2. In the actual use scenario, we can consider adding a GPIO between the master and the slave, and the master device outputs to notify the slave device to transfer ready to reduce the CPU busy waiting time

2.3.3 Tips for SPI slave test

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

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

2.4 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)
```

2.5 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.

Kernel Configuration

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

DTS Configuration

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

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

2.6 Support cs-gpios

Users can use the cs-gpios attribute of spi-bus to implement gpio simulation cs to extend SPI chip selection signal. Users can refer to the kernel document `Documentation/devicetree/bindings/spi/spi-bus.txt` to learn more about cs-gpios.

2.6.1 Linux 4.4 configuration

This support needs more support patches. Please contact RK Engineer for the corresponding patches.

2.6.2 Linux 4.19 configuration

Take spi1_cs2n in GPIO0_C4 for example:

Set the cs-gpio pin and reference it in the SPI node

```
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_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 &spi1m0_cs0n &spi1m0_cs1n &spi1_cs2n
&spi1m0_miso &spi1m0_mosi>;
+       pinctrl-1 = <&spi1m0_clk_hs &spi1m0_cs0n &spi1m0_cs1n &spi1_cs2n
&spi1m0_miso_hs &spi1m0_mosi_hs>
        status = "disabled";
};
```

SPI node reassigns CS pin

```
+&spi1 {
+       status = "okay";
+       max-freq = <48000000>;
+       cs-gpios = <0>, <0>, <&gpio0 RK_PC4 GPIO_ACTIVE_LOW>;   /* 该行定义：cs0-native，cs1-native，cs2-gpio */
        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>;
+       };
};
```

3. SPI Testing Driver in Kernel

3.1 Code Path

drivers/spi/spi-rockchip-test.c

3.2 SPI Testing Device Configuration

Kernel Path

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

DTS Configuraion

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

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=50
00000
[    0.532711] rockchip_spi_test_probe:poll_mode=0, type=0, enable_dma=0
//This is the mark of succesful register.
```

3.3 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.