Rockchip SPI Developer Guide

ID: RK-KF-YF-075

Release Version: V2.7.0

Release Date: 2023-08-15

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 and above	Linux 4.19 and above

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	Jon Lin	2019-12-09	Support Linux 4.19
V2.1.0	Jon Lin	2020-02-13	Adjust SPI slave configuration
V2.2.0	Jon Lin	2020-07-14	Linux 4.19 DTS configuration change, Optimize document layout
V2.3.0	Jon Lin	2020-11-02	Add comment for supporting spi-bus cs-gpios property
V2.3.1	Jon Lin	2020-12-11	Update Linux 4.4 SPI slave description
V2.3.2	Jon Lin	2021-07-06	Add more add configuration description、Add more cs-gpios description
V2.4.0	Jon Lin	2021-08-31	Add FAQs and reduce redundant configurations
V2.5.0	Jon Lin	2021-12-27	Support Linux 5.10
V2.6.0	Jon Lin	2023-06-22	Added SPI Slave Software In Kernel、rockchip,poll-only support and explanation of common problems
V2.7.0	Jon Lin	2023-08-15	Explanation of the optimization direction for increasing SPI transmission rate and high CPU usage

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-clock-rates = <CLK_SPI1>; //Not configured by default, depend on
soc dtsi
    //assigned-clock-rates = <200000000>;//Not configured by default, spi
controller work clock
    //dma-names;                        //Not configured by default, turn off
DMA support, only supports IRQ transmission
```

```
    //rockchip,poll-only;               //Not configured by default,  turn to
use CPU transmission, only master mode supported
    //rx-sample-delay-ns = <10>;        //Not configured by default, Read
sampling delay. Please refer to "FAQ" and "Delay sampling clock configuration"
for details
    //rockchip, autosuspend-delay-ms = <500>; //Not configured by default,
Runtime PM autosuspend delay, refer to "SPI Transfer Rate and CPU Usage
Optimization Directions" for details.
    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.

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 Configuration of Linux 4.4

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@1 {
        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 Configuration of Linux 4.19 and above

Kernel Configuration

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

DTS configuration

```
&spi1 {
    status = "okay";
    spi-slave;                          //enable slave mode
    //assigned-clock-rates = <CLK_SPI1>; //Not configured by default, depend on
soc dtsi
    //assigned-clock-rates = <200000000>;//Not configured by default, spi
controller work rate
    //dma-names;                        //Not configured by default, turn off
DMA support, only supports IRQ transmission
    //ready-gpios = <&gpio1 RK_PD2 GPIO_ACTIVE_LOW>; //Not configured by default,
SPI slave completes the transmission configuration signal, please refer to the
"SPI Slave Software In Kernel" chapter for details
    //rockchip,cs-inactive-disable;     //Not configured by default, When SPI
master timing tod_cs (Clk Rise To CS Rise Time) exceeds multiple IO clock cycles,
the configuration should be enabled to mask and detect the cs release action
    slave {                             //As spi-bus requied, SPI slave sub-node
should name start with "slave"
        compatible ="rockchip,spi_test_bus1_cs0";
        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
```

```
        spi-max-frequency = <50000000>;  //Slave does not output clk, but
requires setting the controller's working clock through this configuration. It is
recommended to fix the configuration to 50MHz
        status = "okay";                 //Enable device node
    };
};
```

Note:

- 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
- The RK spi slave universal driver only supports IO clocks within 16MHz, and the docking spi master should be limited to this rate range

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:

```
#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);
```

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@0 {
        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

For the demo of spidev operation, please refer to:

- Kernel 4.4 Documentation/spi/spidev_test.c
- Kernel 4.19 and later tools/spi/spidev_test.c
- After the kernel project is compiled, enter the corresponding path and enter the following command to directly compile the standard SPI app program:

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

- It supports the configuration of SPI slave devices. Refer to "SPI Device Configuration: RK SPI As Slave Port", in which the DTS configuration sub node should remain "rockchip, spidev"

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 Configuration of Linux 4.4

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

2.6.2 Configuration of Linux 4.19 and above

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_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 node reassigns CS pin

```
+&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@0 {
                compatible = "rockchip,spi_test_bus1_cs0";
...
+       spi_test@2 {
+               compatible = "rockchip,spi_test_bus1_cs2";
+               id = <2>;
+               reg = <0x2>;
+               spi-cpha;
+               spi-cpol;
+               spi-lsb-first;
+               spi-max-frequency = <16000000>;
+       };
};
```

Note:

- If you want to extend cs with gpio, all cs should be converted to gpio function and supported by cs-gpios property.

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@0 {
        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@1 {
        compatible = "rockchip,spi_test_bus0_cs1";
```

```
        id = <1>;
        reg = <1>;
        spi-max-frequency = <24000000>;
    };
};
```

Driver 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 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. SPI Slave Software In Kernel

4.1 Introduction

Background

The transmission between SPI masters and slaves usually follows specific protocols, such as SPI No compatible with JEDEC SDFP protocol, and RK SPI slave, as a device side transmission, should also follow specific protocols. As the protocol has no paradigm, RK provides customized transmission protocols and device drivers for customer reference.

Linux SPI slave driver framework limitations:

- Using a transmission queue, although the thread priority after queue wake-up is higher, real-time performance cannot be fully guaranteed due to scheduling constraints

RK SPI slave mode restrictions:

- Each transmission requires a restart of the SPI controller configuration. Therefore, to ensure that the SPI master can know that the RK SPI slave has completed the transmission configuration and initiated data

transmission, the RK SPI slave end needs to add a side band signal as a ready status bit

Transport Protocol

RK SPI slave transmission protocol:

- The RK SPI slave transmission requires specifying ready-gpios to notify the SPI master. The basic process is as follows:

  step1：slave start spi_sync

  step2：slave ready, output GPIO_SLV_READY signal

  step3：master confirm slave ready, then begin to transfer

  step4：The slave receives sufficient clks from the master to complete the transmission

  step5：slave idle， release GPIO_SLV_READY signal

- Define two types of packages:
  Ctrl packet: 2B cmd, 2B addr (application buffer offset address defined by RK slave), 4B data (usually used to specify the transmission length of subsequent data packets)

  data packet

- Define two types of transmission:
  Ctrl transmission, only containing 1 ctrl packet

  Data transfer, including two SPI transfers of 1 ctrl packet and 1 data packet

- spidev_rkslv support SPI_OBJ_APP_RAM_SIZE Bytes application buffer for transmission buffer， The data transmission 1 ctrl packet 2B addr initiated by the SPI master points to the cache offset address

Device Driver

Driver source code:

```
drivers/spi/spidev-rkslv.c
drivers/spi/spidev-rkmst.c
```

Source code introduction:

drivers/spi/spidev-rkslv.c：

```
static int spidev_rkslv_ctrl_receiver_thread(void *p)
//Establish a thread and repeatedly initiate transfers within the thread
{
    while (1)
        spidev_rkslv_xfer(spidev);
}

static int spidev_rkslv_xfer(struct spidev_rkslv_data *spidev)
 //Transmission entrance
{
    spidev_slv_read(spidev, spidev->ctrlbuf, SPI_OBJ_CTRL_MSG_SIZE);    //1 ctrl
packet to obtain and parse the transmission type
    switch (ctrl->cmd) {                                                //1 data
packet, define data packets based on the transmission type and complete sending
and receiving
    case SPI_OBJ_CTRL_CMD_INIT:
            /* to-do */
```

```
    case SPI_OBJ_CTRL_CMD_READ:
            /* to-do */
    case SPI_OBJ_CTRL_CMD_WRITE:
            /* to-do */
    case SPI_OBJ_CTRL_CMD_DUPLEX:
            /* to-do */
    }
}

static const struct file_operations spidev_rkslv_misc_fops = {}
 //Register misc device test interface
```

drivers/spi/spidev-rkmst.c：

```
static int spidev_rkmst_xfer(struct spidev_rkmst_data *spidev, void *tx, void
*rx, u16 addr, u32 len)  //Transmission entrance
{
    spidev_rkmst_ctrl(spidev, cmd, addr, len);                          //1 ctrl
packet, defining the transmission type
    switch (cmd) {                                                      //1 data
packet, define data packets based on the transmission type and complete sending
and receiving
    case SPI_OBJ_CTRL_CMD_READ:
            /* to-do */
    case SPI_OBJ_CTRL_CMD_WRITE:
            /* to-do */
    case SPI_OBJ_CTRL_CMD_DUPLEX:
            /* to-do */
    }
}

static const struct file_operations spidev_rkmst_misc_fops = {}
 //Register misc device test interface
```

Implement business

The purpose of providing "SPI Slave Software In Kernel" is to provide reference for protocol and device drivers, and the end customer should also define their product requirements on the application buffer of the slave end to achieve business.

4.2 SPI Slave Testing Device Configuration

Defconfig configuration:

```
CONFIG_SPI_SLAVE_ROCKCHIP_OBJ=y
```

RK SPI slave end dts reference configuration:

```
&spi1 {
    status = "okay";
    spi-slave;
    rockchip,cs-inactive-disable;                               //RK internal
interconnection using RK Linux SPI master driver, and the tod_cs takes a long
time
```

```
    ready-gpios = <&gpio1 RK_PD3 GPIO_ACTIVE_LOW>;           //Please set to the
actual GPIO used
    slave {
        compatible = "rockchip,spi-obj-slave";
        reg = <0x0>;
        spi-cpha;
        spi-cpol;
        spi-lsb-first;
        spi-max-frequency = <50000000>;
    };
};
```

RK SPI master dts reference configuration:

```
&spi0 {
    status = "okay";
    spi_test@00 {
        compatible = "rockchip,spi-obj-master";
        reg = <0x0>;
        spi-cpha;
        spi-cpol;
        spi-lsb-first;
        spi-max-frequency = <16000000>;
        ready-gpios = <&gpio1 RK_PD2 GPIO_ACTIVE_LOW>;
    };
};
```

4.3 Test Command

SPI master initiates single packet data transmission testing

```
echo cmd addr length > /dev/spidev_rkmst_misc
```

Note:

- cmd: Support read/write/duplex
- addr: Refers to the offset of the peer slave application buffer, in Bytes, which only supports decimal input
- length: The length of the data packet, in Bytes, only supports decimal input
- The example is as follows:

```
echo write 128 128 > /dev/spidev_rkmst_misc
echo read 128 128 > /dev/spidev_rkmst_misc
echo loop 128 128 > /dev/spidev_rkmst_misc
```

SPI master initiates Test automation

```
echo autotest length loops > /dev/spidev_rkmst_misc
```

Note:

- autotest: Fixed input, test full duplex data transmission first, then test read and write data transmission, and output rate results
- The test defaults to using the opposite slave application buffer offset address 0

- length: The length of the data packet, in Bytes, only supports decimal input
- loops: Set the number of pressure testing cycles
- The strength is as follows:

```
echo autotest 1024 64 > /dev/spidev_rkmst_misc
```

SPI slave testing

```
echo appmem 0 256 > ./dev/spidev_rkslv_misc      #Print application buffer data
echo verbose 1 > ./dev/spidev_rkslv_misc         #Enable the transmission process
debug log, echo verbose 0, and close printing
```

5. FAQ

5.1 SPI no signal

- 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.
- How to simply judge whether SPI DMA is enabled or not? If the serial port printing does not have the following keywords, DMA is enabled successfully:

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

5.2 How to design application code in SPI

Please select the appropriate object function interface before writing the driver.

Custom SPI device driver

Refer to "SPI Device Driver", for example: drivers/spi/spi-rockchip-test.c.

Application program based on spidev standard device node

Refer to "User mode SPI device Configuration"

5.3 Delay sampling clock configuration

In the case of high SPI IO rate, the normal SPI mode may still not match the output delay of external devices, and RK SPI master read may not be able to sample valid data. SPI RSD logic needs to be enabled to delay the sampling clock.

RK SPI RSD (read sample delay) control logic has the following characteristics:

- The assignable values are 0, 1, 2, 3
- The delay unit is 1 spi_clk cycle, i.e. the working clock of the controller, see "SPI Device Configuration" for details

rx-sample-delay actual delay is the RSD effective value closest to the DTS set value, subject to spi_clk 200MHz, cycle 5ns, for example:

The actual configurable delay of RSD is 0, 5ns, 10ns and 15ns. RX sample delay is set to 12ns, which is close to the effective value of 10ns, so the final delay is 10ns.

5.4 SPI transmission method description

The master mode supports IRQ, DMA, and CPU transmission, while the slave mode supports IRQ and DMA transmission. The default transmission mode is the combination of IRQ/DMA:

- When the transmission length is less than the fifo depth, IRQ transmission is used. By default, SOC with kernel versions 4.19 and above is used, and the fifo depth is 64
- When the transmission length >= fifo use DMA transmission

IRQ transmission characteristics:

- When the data < fifo depth, one interrupt is triggered at a time
- When the data >= fifo depth and IRQ transmission is used, the fifo watermark is set to half fifo, usually 32 items, and a transmission roughly triggers items/32 interrupts

DMA transmission characteristics:

- Do not trigger SPI controller interrupt, use DMA to transfer finished call back callback

5.5 SPI Transfer Rate and CPU Usage Optimization Directions

The reasons for slow SPI transfer rates and high CPU usage under heavy IO load are usually related to small transfer granularity and frequent transfer requests, which involve the following:

- SPI thread scheduling
- Interrupt scheduling (refer to the "SPI Transfer Mode Explanation" section to confirm if interrupt transfers are used)
- CPU idle scheduling

Recommended optimization directions:

1. Enable auto runtime suspend with a delay of 500ms (specific value may vary based on actual testing). Modify the DTS node by adding the `rockchip,autosuspend-delay-ms` property.
2. Reduce CPU load: Switch to IRQ-based transfers, which may have advantages over DMA. Refer to the "Switch to IRQ Transfers" subsection for patch reference.
3. Reduce CPU load: If using DMA transfers, modify the TX DMA waterline to reduce the time CPU spends waiting for FIFO transfers to complete in the DMA callback function. Refer to the "Modify SPI Waterline" patch.

Patch references:

Switch to IRQ Transfers

```
diff复制代码diff --git a/arch/arm/boot/dts/rv1126-evb-v10.dtsi
b/arch/arm/boot/dts/rv1126-evb-v10.dtsi
index 86dd23482d97..2cea93d2423f 100644
--- a/arch/arm/boot/dts/rv1126-evb-v10.dtsi
+++ b/arch/arm/boot/dts/rv1126-evb-v10.dtsi
@@ -1367,6 +1367,7 @@
        status = "okay";
        max-freq = <48000000>;
        cs-gpios = <0>, <0>, <&gpio0 RK_PC4 GPIO_ACTIVE_LOW>;
+       dma-names;
        spi_test@00 {
```

Modify SPI Waterline

```
diff复制代码diff --git a/drivers/spi/spi-rockchip.c b/drivers/spi/spi-rockchip.c
index 27fd6f671b12..bd0fa8c5f8c3 100644
--- a/drivers/spi/spi-rockchip.c
+++ b/drivers/spi/spi-rockchip.c
@@ -616,7 +616,8 @@ static void rockchip_spi_config(struct rockchip_spi *rs,
        else
                writel_relaxed(rs->fifo_len / 2 - 1, rs->regs +
ROCKCHIP_SPI_RXFTLR);

-       writel_relaxed(rs->fifo_len / 2 - 1, rs->regs + ROCKCHIP_SPI_DMATDLR);
+       // writel_relaxed(rs->fifo_len / 2 - 1, rs->regs + ROCKCHIP_SPI_DMATDLR);
+       writel_relaxed(11, rs->regs + ROCKCHIP_SPI_DMATDLR);
        writel_relaxed(rockchip_spi_calc_burst_size(xfer->len / rs->n_bytes) - 1,
                       rs->regs + ROCKCHIP_SPI_DMARDLR);
        writel_relaxed(dmacr, rs->regs + ROCKCHIP_SPI_DMACR);
```

Note: The provided patch references are specific to the Rockchip SPI driver and RV1126-EVB-V10 DTS file. Adjustments may be needed based on your specific hardware and software configurations.