Rockchip Linux SPI

文件标识：RK-KF-YF-020

发布版本：V3.1.0

日期：2024-03-04

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 及以上内核的所有芯片	Linux4.19 及以上内核

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
V2.5.0	林鼎强	2021-12-27	新增 linux5.10 支持
V2.6.0	林鼎强	2023-06-25	新增 内核 SPI Slave 软件、rockchip,poll-only 支持和常见问题说明
V2.7.0	林鼎强	2023-08-15	增加 SPI 传输速率及 CPU 占用率高优化方向说明
V2.8.0	林鼎强	2023-10-23	增加 SPI 接口速率说明
V2.8.1	林鼎强	2023-10-24	修改 dts 中错误的节点属性
V2.9.0	林鼎强	2023-12-04	更新 Slave 专用驱动说明及须知
V3.0.0	林鼎强	2023-12-20	更新 SPI Slave 须知、删除 sram buffer 支持
V3.1.0	林鼎强	2024-03-04	添加 RK3576 支持

目录

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

SPI 接口速率

SOC	Master Mode 接口最高速率	Slave Mode 接口最高速率
RK3576	75MHz	50MHz
RK3562	50MHz	50MHz
RK3528	50MHz	25MHz
RV1106/RV1103	50MHz	50MHz
RK3588	50MHz	50MHz
RV1126/RV1109	50MHz	25MHz
RK3568	50MHz	50MHz
RK1808	50MHz	25MHz
RK3308	50MHz	25MHz
其他芯片平台	50MHz	16MHz

说明:

- 接口最高速率为理论速率，受设备走线 PCB 质量影响，以实测为准
- 部分平台由于 PLL 策略原因无法准确分频到上限值，实际以最大分频值为准

内核软件

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/spi-rockchip-slave.c rk spi slave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
```

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-clocks = <CLK_SPI1>;                   //默认不用配置，CLK_SPIn 请从
soc 对应的 dtsi 里确认
    //assigned-clock-rates = <200000000>;             //默认不用配置，SPI 设备工作时钟
值
    //dma-names;                                      //默认不用配置，关闭 DMA 支持，
仅支持 IRQ 传输
```

```
    //rockchip,poll-only;                          //默认不用配置，开启后强制使用
CPU 传输，仅支持 master mode 下配置
    //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
    //rockchip,autosuspend-delay-ms = <500>;       //默认不用配置，Runtime PM
autosuspend 延时，详细参考 “SPI 传输速率及 CPU 负载优化”
    //rockchip,rt;                                 //默认不用配置，将spi数据传输进程
放到SCHED_FIFO类中，其优先级为50
    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;

SPI 设备配置 —— RK 芯片作 Slave 端

关键补丁

推荐使用 SPI slave 源码 spi-rockchip-slave.c，由于 SDK 版本问题，建议先确认 SDK 是否有以下补丁：

```
commit 10cbf3c2c93fca6e5ec6c99b5bdb319ca0494d45
Author: Jon Lin <jon.lin@rock-chips.com>
Date:   Tue Nov 21 10:58:57 2023 +0800

    spi: rockchip-slave: Add code

    1.Implement one msg mechanism
    2.Support SRAM extension by dts rockchip,sram property

    Change-Id: I0fccc5d4347294488b5382ad3ba5ae72b35610f2
    Signed-off-By: Jon Lin <jon.lin@rock-chips.com>
```

说明：

- 如无该补丁，客户可直接通过 Redmine -> FAE 项目 -> 文档 -> 开发配置文档 -> SPI 路径获取。

内核配置

```
Device Drivers  --->
    [*] SPI support  --->
        [*]   SPI slave protocol handlers
        [*]   Rockchip SPI Slave controller driver
```

DTS 节点配置

```
&spi1 {
    compatible = "rockchip,spi-slave";       //优先使用 SPI slave 专用驱动
    status = "okay";
    //ready-gpios = <&gpio1 RK_PD2 GPIO_ACTIVE_LOW>;//建议配置，SPI slave 完成传输配置的信号，详细参考“内核 SPI Slave 软件”章节
    //rockchip,cs-inactive-disable;                 //默认不用配置，当 SPI master 时序 tod_cs（Clk Rise To CS Rise Time） 超过多个 io 时钟周期，应开启配置来屏蔽检测 cs 释放动作
    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>;                                  //片选仅支持 0
        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
        status = "okay";                            //使能设备节点
    };
};
```

说明：

- RK SPI 默认使能 DMA 传输，slave mode 不建议关闭 DMA 传输。当一笔传输超过控制器缓存数量，软件会配置为 DMA 传输，来避免中断传输相应不及时。

SPI Slave 须知

建议设置 performance

当 master 速率超过一定频率后，建议传输过程设置 performance 模式，避免传输过程 DRAM 变频导致控制器缓存溢出：

- bits_per_word = 8btis，master io 速率超过 5MHz
- bits_per_word = 16btis，master io 速率超过 10MHz

参考代码如下：

```
diff --git a/drivers/spi/spi-rockchip-test.c b/drivers/spi/spi-rockchip-test.c
index 544d6038919a..c1037153ff86 100644
--- a/drivers/spi/spi-rockchip-test.c
+++ b/drivers/spi/spi-rockchip-test.c
@@ -36,6 +36,8 @@
 #include <linux/platform_data/spi-rockchip.h>
 #include <linux/uaccess.h>
 #include <linux/syscalls.h>
+#include <soc/rockchip/rockchip-system-status.h>
+#include <dt-bindings/soc/rockchip-system-status.h>

 #define MAX_SPI_DEV_NUM 10
 #define SPI_MAX_SPEED_HZ        12000000
```

```
@@ -242,8 +244,10 @@ static ssize_t spi_test_write(struct file *file,
                }

                start_time = ktime_get();
+               rockchip_set_system_status(SYS_STATUS_PERFORMANCE);
                for (i = 0; i < times; i++)
                        spi_read_slt(id, rxbuf, size);
+               rockchip_clear_system_status(SYS_STATUS_PERFORMANCE);
                end_time = ktime_get();
                cost_time = ktime_sub(end_time, start_time);
                us = ktime_to_us(cost_time);
```

说明:

- 建议所有 slave mode 传输行为都应在 performance mode 下运行
- set/clear performance 接口有一定的时间开销，所以建议业务上层设置，避免频繁调用
- 如果缓存溢出，slave 无法完成 DMA 传输，会阻塞无法退出，通过打印 SPI->SPI_RISR 寄存器可以确认是否出现缓存溢出

建议设置 16bits 宽度

最大限度利用 slave fifo 容量，加速且最小 burst 2，能加速 slave 端的 DMA 传输速率，避免 fifo 因为来不及搬移造成堆叠

其他须知

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`

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)
```

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@0 {
        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"

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`。

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。

Linux 4.19 及以上内核配置

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@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>;
+       };
};
```

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

内核测试软件

代码路径

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

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@0 {
        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@1 {
        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=1600
0000
[    0.457308]
rockchip_spi_test_probe:name=spi_test_bus0_cs1,bus_num=0,cs=1,mode=11,speed=1600
0000
```

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。

内核 SPI Slave 软件

简介

背景

SPI 主从之间传输通常遵循特定协议，如 SPI Nor 兼容 JEDEC SDFP 协议，RK SPI slave 作为设备端传输也应遵循特定的协议，由于协议无范式，所以 RK 提供自定义的传输协议和设备驱动以供客户参考。

Linux SPI slave 驱动框架限制：

- 使用传输队列，虽然队列唤醒后的线程优先级较高，但受调度影响不能完全保证实时性

RK SPI slave mode 限制：

- 每次传输需重新发起 SPI 控制器配置，因此为确保 SPI master 能够获知 RK SPI slave 完成传输配置从而发起数据传输，RK SPI slave 端需增加 side-band 信号做 ready 状态位

传输协议

RK SPI slave 传输协议：

- RK SPI slave 传输要求指定 ready-gpios 来通知 SPI master，基本流程：

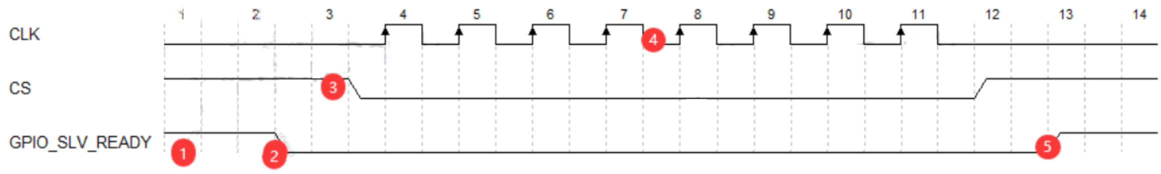
  step1：slave 主动发起 spi_sync

  step2：slave ready，使能 GPIO_SLV_READY 信号

  step3：master 确认 slave ready 后发起传输

  step4：slave 接收来自 master 发出的足够的 clk 后完成传输

  step5：slave idle，释放 GPIO_SLV_READY 信号



- 定义两种包类型：
  ctrl packet：2B cmd，2B addr（RK slave 定义的 application buffer 偏移地址），4B data（通常用于指定之后 data 包的传输长度）

  data packet
- 定义两种传输类型：
  ctrl 传输，仅包含 1 ctrl packet

  data 传输，包含 1 ctrl packet 和 1 data packet 的两笔 SPI 传输
- spidev_rkslv 支持 SPI_OBJ_APP_RAM_SIZE 长度的 application buffer 用于缓存传输数据，SPI master 发起的 data 传输 1 ctrl packet 2B addr 指向该缓存偏移地址

设备驱动

关键补丁：

```
commit d2fef34977c1a7aab3837d29ac8dc3b5378a2754 (HEAD -> develop-4.19)
Author: Jon Lin <jon.lin@rock-chips.com>
Date:   Wed Dec 20 12:02:14 2023 +0800

    spi: spidev_rkslv: Support dynamic adjustment of system performance

    If the DRAM frequency conversion jitters during the transmission process,
    it will cause the DMA to be unable to transport SPI FIFO data in a timely
    manner, resulting in FIFO overflow/underflow.

    Clear performance status for short cmd packet and Set the performance
    status for data packet.

    Change-Id: I65532ba309677a8d98c8277875a3bd358ca44e44
    Signed-off-by: Jon Lin <jon.lin@rock-chips.com>
```

说明：

- 由于 SDK 版本问题，建议先确认 SDK 是否有以下补丁，如无该补丁，客户可直接通过 Redmine -> FAE 项目 -> 文档 -> 开发配置文档 -> SPI 路径获取

驱动源码：

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

源码简介：

drivers/spi/spidev-rkslv.c：

```
static int spidev_rkslv_ctrl_receiver_thread(void *p)                //建立线程，线程内重复发起传输
{
    while (1)
        spidev_rkslv_xfer(spidev);
```

```
}

static int spidev_rkslv_xfer(struct spidev_rkslv_data *spidev)          //传输入口
{
    spidev_slv_read(spidev, spidev->ctrlbuf, SPI_OBJ_CTRL_MSG_SIZE);    //1 ctrl packet，获取并解析传输类型
    switch (ctrl->cmd) {                                                //1 data packet，根据传输类型，定义 data packet 并完成收发
    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 = {}          //注册misc device 测试接口
```

drivers/spi/spidev-rkmst.c:

```
static int spidev_rkmst_xfer(struct spidev_rkmst_data *spidev, void *tx, void *rx, u16 addr, u32 len)  //传输入口
{
    spidev_rkmst_ctrl(spidev, cmd, addr, len);                           //1 ctrl packet，定义传输类型
    switch (cmd) {                                                       //1 data packet，根据传输类型，定义 data packet 并完成收发
    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 = {}          //注册misc device 测试接口
```

实现业务

提供"内核 SPI slave 软件"的目的在于提供协议和设备驱动的参考，最终客户还应在 slave 端的 application buffer 上定义自己的产品需求以实现业务。

SPI Slave 测试设备配置

defconfig 配置:

```
CONFIG_SPI_SLAVE_ROCKCHIP_OBJ=y
```

RK SPI slave 端 dts 参考配置:

```
&spi1 {
    status = "okay";
    spi-slave;
    rockchip,cs-inactive-disable;                           //RK 内部互联使用 RK
Linux SPI master 驱动, tod_cs 较长
    ready-gpios = <&gpio1 RK_PD3 GPIO_ACTIVE_LOW>;          //请设置为实际所用
GPIO
    slave {
        compatible = "rockchip,spi-obj-slave";
        reg = <0x0>;
        spi-cpha;
        spi-cpol;
        spi-lsb-first;
        spi-max-frequency = <50000000>;
    };
};
```

RK SPI master 端 dts 参考配置:

```
&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>;          //请设置为实际所用
GPIO
    };
};
```

测试命令

SPI master 发起单包数据传输测试

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

说明:

- cmd： 支持 read/write/duplex
- addr：为对端 slave application buffer 偏移，单位 Bytes，仅支持 10 进制输入
- length：为 data packet 长度，单位 Bytes，仅支持 10 进制输入
- 实例如下:

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

SPI master 发起自动化测试

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

说明：

- autotest：固定输入，先测试全双工数据传输，再测试读写数据传输，并输出速率结果
- 测试默认使用对端 slave application buffer 偏移地址 0
- length：为 data packet 长度，单位 Bytes，仅支持 10 进制输入
- loops：设定压测循环次数
- 实力如下：

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

SPI slave 测试

```
echo appmem 0 256 > ./dev/spidev_rkslv_misc      #打印 application buffer 数据
echo verbose 1 > ./dev/spidev_rkslv_misc         #开启传输传输过程 debug log，echo
verbose 0 关闭打印
```

常见问题

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
  ```

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。

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。

SPI 传输方式说明

master mode 支持 IRQ、DMA 和 CPU 传输，slave mode 支持 IRQ 和 DMA 传输，默认都为 IRQ/DMA 组合传输方式：

- 当传输长度 < fifo 深度时，使用 IRQ 传输，默认使用 4.19 及以上内核版本的 SOC，fifo 深度为 64
- 当传输长度 >= fifo 深度时，使用 DMA 传输

IRQ 传输特性：

- 当数据 < fifo 深度时，一次传触发 1 个中断
- 当数据 >= fifo 深度且使用 IRQ 传输时，fifo 水线设置为半 fifo，通常为 32 item，一次传输大致上触发 items / 32 次中断

DMA 传输特性：

- 不触发 spi 控制器中断，使用 DMA 传输 finished call back 回调

SPI 传输速率及 CPU 占用率高优化方向

通常 SPI 传输速率慢、IO 高负载下 CPU 占用率高的原因是因为：SPI 传输粒度小，且传输次数多，频繁发起传输从而涉及较多的调度，例如：

- SPI 线程调度
- 中断调度，参考 “SPI 传输方式说明” 章节先确认是否使用到中断传输
- CPU idle 调度

建议优化方向：

1. 开启 auto runtime，延时设置为 500ms，具体值以实测为准，修改点为 dts 节点添加 rockchip,autosuspend-delay-ms 属性
2. 降低 CPU 负载：改用 IRQ 传输，相对 DMA 可能会有优势，补丁参考 “改为 IRQ 传输” 小节
3. 降低 CPU 负载：如为 DMA 传输，可修改 TX DMA 水线来降低 CPU 在 DMA 回调函数中等待 fifo 传输完成的时间，补丁参考 “修改 SPI 水线”

修改补丁参考：

改为 IRQ 传输

```
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 {
```

修改 SPI 水线

```
diff --git a/drivers/spi/spi-rockchip.c b/drivers/spi/spi-rockchip.c
index 27fd6f671b12..bd0fa8c5f8c3 100644
```

```diff
--- 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);
```