spi 接口使用说明书

2.0

2018.12.24

文档履历

版本号	日期	制/修订人	内容描述
1.0	2016.12.05	AWA1053	
2.0	2018.12.24	AWA1430	修改格式

目录

1. 概述

1.1 编写目的

介绍 Linux 内核中 SPI 子系统的接口及使用方法，为 SPI 设备驱动开发提供参考。

1.2 适用范围

适用于 allwinnertech 全系列平台。

1.3 相关人员

SPI 设备驱动、SPI 总线驱动的开发/维护人员。

2. 模块介绍

2.1 模块功能介绍

Linux 中 SPI 体系结构分为三个层次，如下图所示：

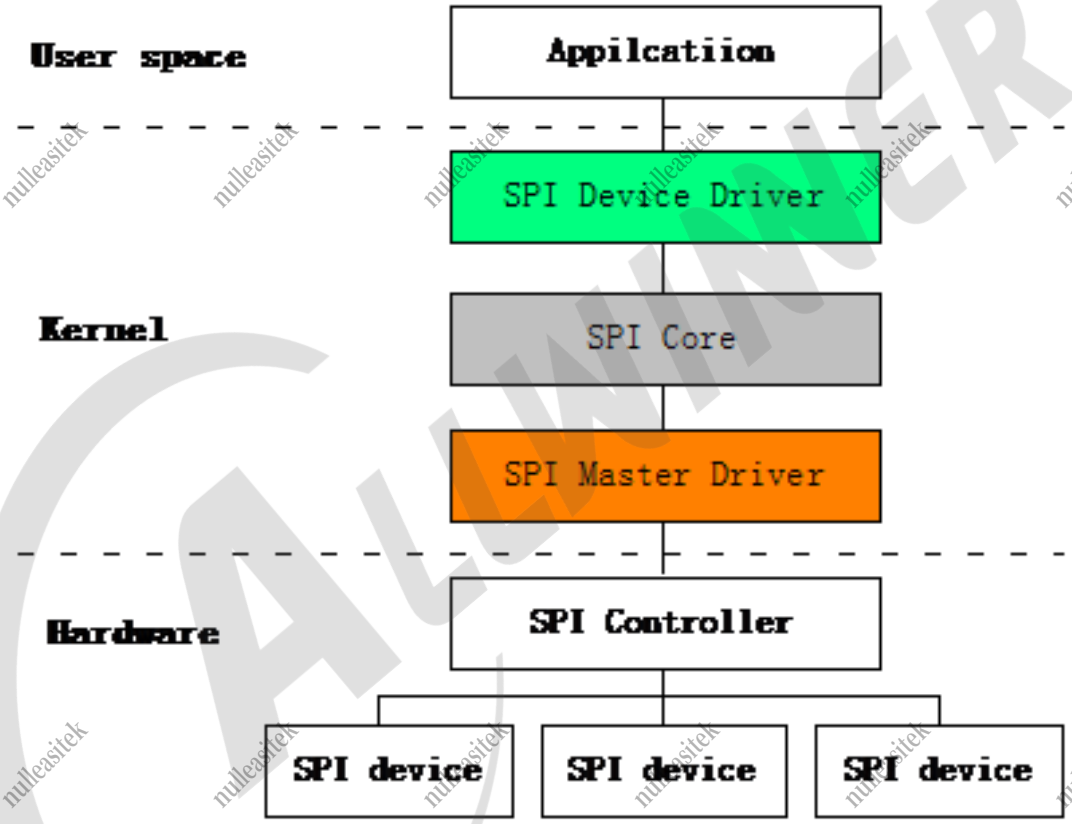

图 1: Linux SPI 体系结构图

1. 用户空间，包括所有使用 SPI 设备的应用程序；

2. 内核，也就是驱动部分；
3. 硬件，指实际物理设备，包括了 SPI 控制器和 SPI 外设。

其中，Linux 内核中的 SPI 驱动程序仅支持主设备，逻辑上又可以分为 3 个部分：1. SPI 核心（SPI Core）：实现对 SPI 总线驱动及 SPI 设备驱动的管理；

2. SPI 总线驱动（SPI Master Driver）：针对不同类型的 SPI 控制器，实现对 SPI 总线访问的具体方法；
3. SPI 设备驱动（SPI Device Driver）：针对特定的 SPI 设备，实现具体的功能，包括 read，write 以及 ioctl 等对用户层操作的接口。SPI 总线驱动主要实现了适用于特定 SPI 控制器的总线读写方法，并注册到 Linux 内核的 SPI 架构，SPI 外设就可以通过 SPI 架构完成设备和总线的适配。但是总线驱动本身并不会进行任何的通讯，它只是提供通讯的实现，等待设备驱动来调用其函数。SPI Core 的管理正好屏蔽了 SPI 总线驱动的差异，使得 SPI 设备驱动可以忽略各种总线控制器的不同，不用考虑其如何与硬件设备通讯的细节。

2.2 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台
SPI	Serial Peripheral Interface，同步串行外设接口
SPI Master	SPI 主设备
SPI Device	指 SPI 外部设备

2.3 模块配置介绍

2.3.1 board.dts 配置说明

在不同的Sunxi 硬件平台中，SPI 控制器的数目也不同，但对于每一个 SPI 控制器来说，在 board.dts 中配置参数相似，如下：

```
spi1: spi@05011000 {
    pinctrl-0 = <&spi1_pins_a &spi1_pins_b>;
    pinctrl-1 = <&spi1_pins_c>;
    spi_slave_mode = <0>;
    status = "disable";
    spi_board1 {
        device_type = "spi_board1";
        compatible = "rohm,dh2228fv";
        spi-max-frequency = <0x5f5e100>;
```

```
            reg = <0x0>;
            spi-rx-bus-width = <0x1>;
            spi-tx-bus-width = <0x1>;
        };
    };
```

其中：>pinctrl-0: 正常使用时的 pin 状态；>pinctrl-1: 休眠使用的 pin 状态；>spi_slave_mode:spi 设备的模式；>status：为 okay 表示使能，disable 表示不使能；
>spi_board1:spi 设备；>device_type：设备的名字；>compatible: 匹配设备的驱动；>spi-max-frequency: 设备的通讯频率；>reg：设备地址长度；>spi-rx-bus-width：接收的数据线数量：>spi-tx-bus-width：发送的数据线数量。

2.3.2 menuconfig 配置说明

在命令行中进入内核 linux 目录，执行 make ARCH=arm menuconfig（64 位平台执行 make ARCH=arm64 menuconfig）进入配置主界面，并按以下步骤操作。

选择 Device Drivers 选项进入下一级配置，如下图所示：

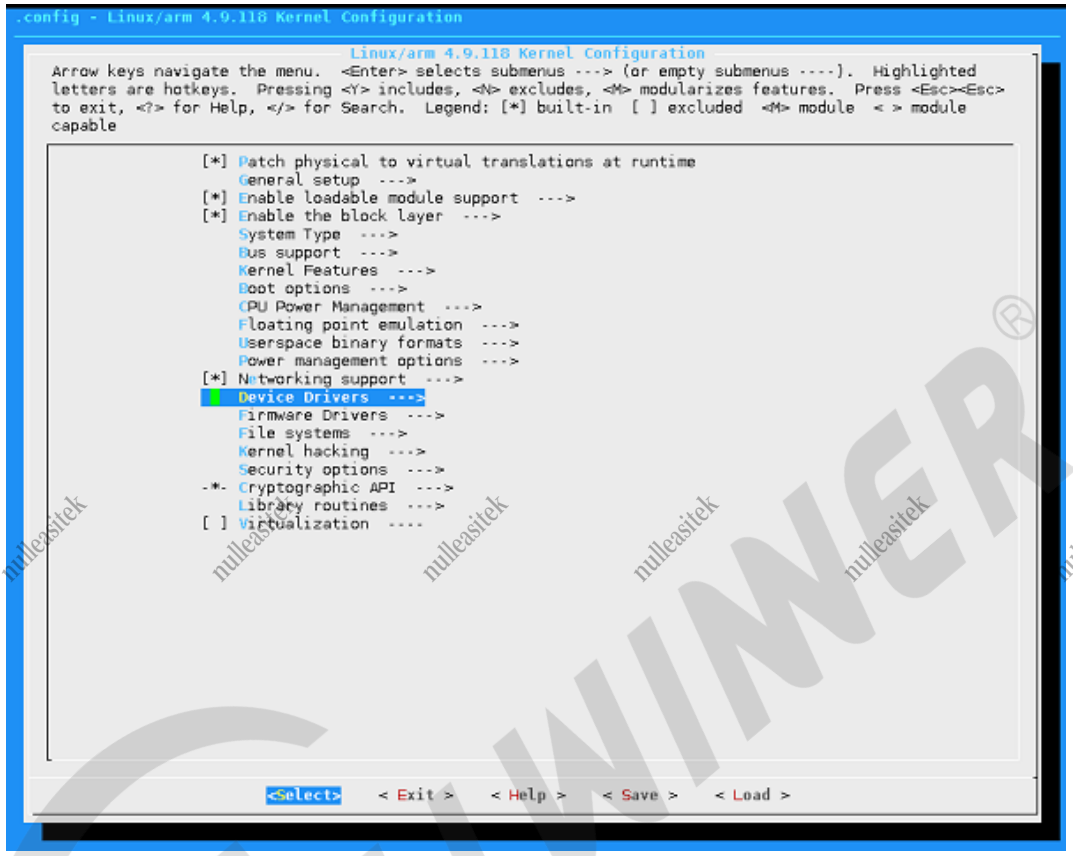

图 2: Device Drivers 配置选项

选择 SPI support 选项，进入下一级配置，如下图所示：

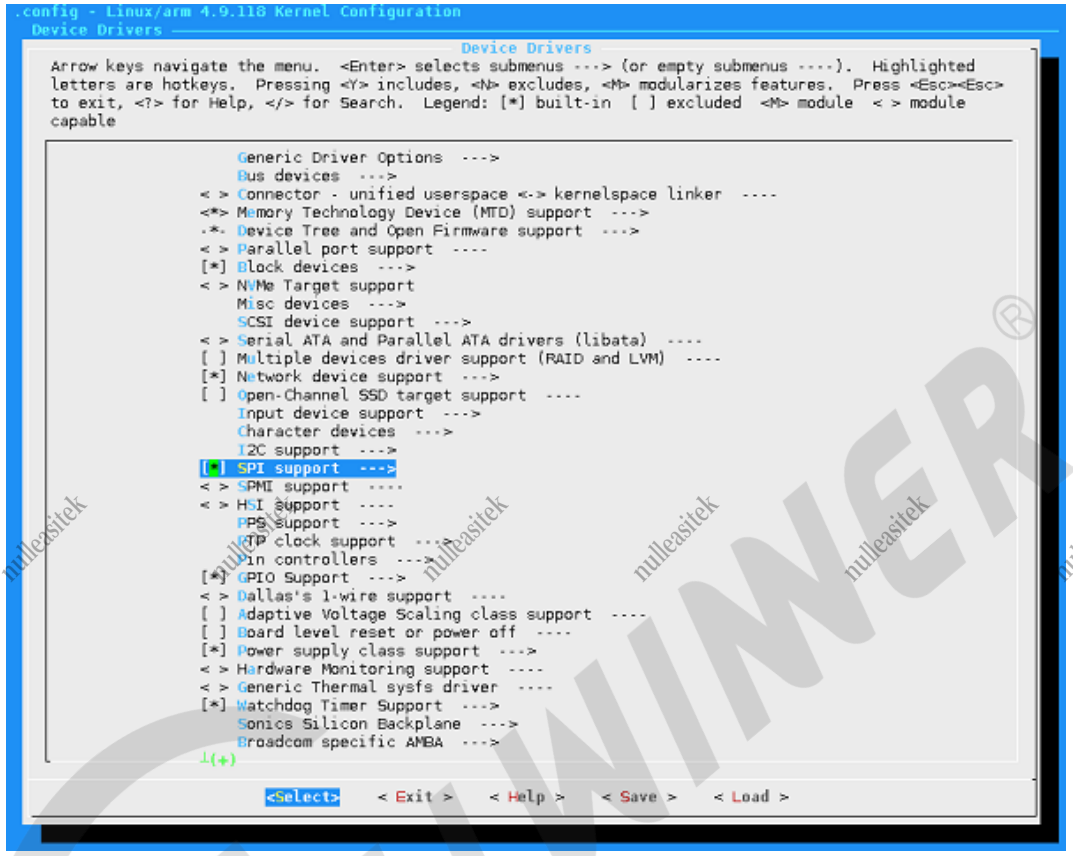

图 3: SPI support 配置选项

选择 SUNXI SPI Controller 选项，可选择直接编译进内核，也可编译成模块。如下图所示：

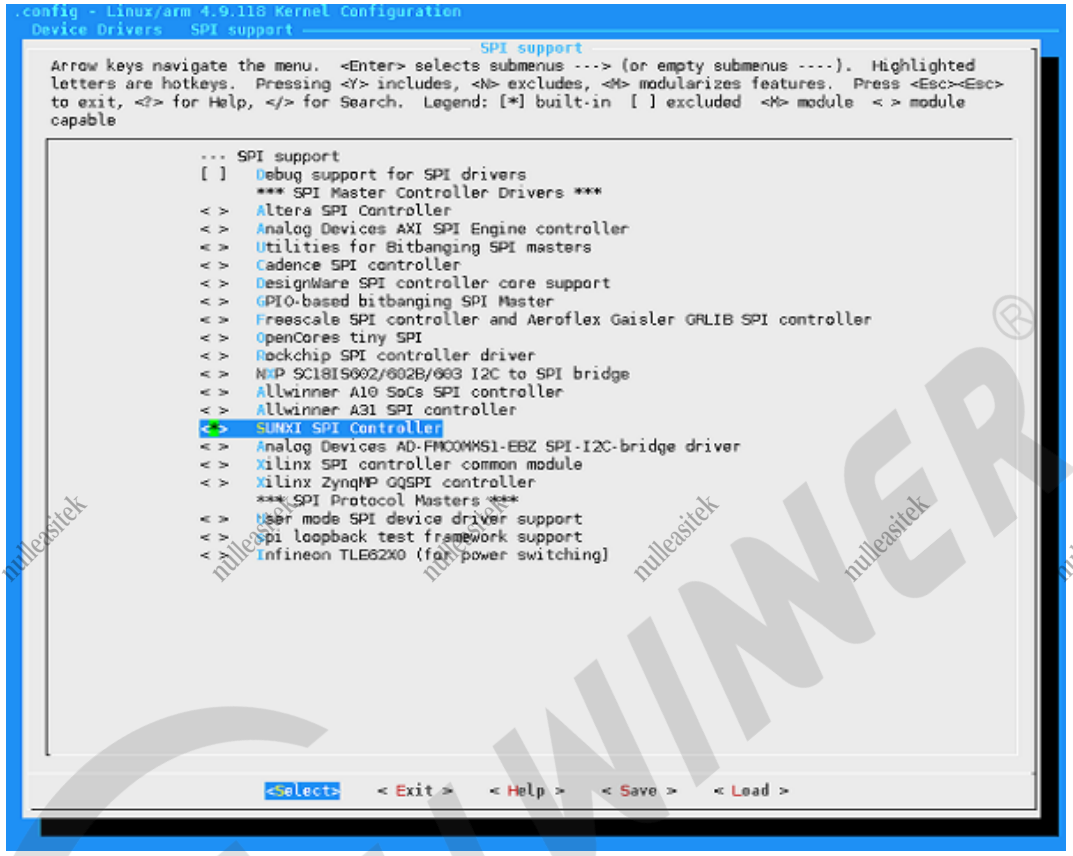

图 4: SUNXI SPI Controller 配置选项

2.4 源码结构介绍

SPI 总线驱动的源代码位于内核在 drivers/spi 目录下：

```
drivers/spi/
├── spi-sunxi.c // Sunxi平台的SPI控制器驱动代码
├── spi-sunxi.h // 为Sunxi平台的SPI控制器驱动定义了一些宏、数据结构
```

3. 接口描述

3.1 设备注册接口

接口定义在 include/linux/spi/spi.h，主要包含 spi_register_driver 与 spi_unregister_driver 接口，其中给出了快速注册的 I2C 设备驱动的宏 module_spi_driver()，定义如下：

```
#define module_spi_driver(__spi_driver)\
module_driver(__spi_driver, spi_register_driver,\
    spi_unregister_driver)
```

3.1.1 spi_register_driver()

3.1.1.1 函数原型

```
int spi_register_driver(struct spi_driver *sdrv)
```

3.1.1.2 功能描述

注册一个 SPI 设备驱动。

3.1.1.3 返回值

返回 0 表示成功，返回其他值表示失败。

3.1.1.4 参数说明

sdrv，spi_driver 类型的指针，其中包含了 SPI 设备的名称、probe 等接口信息。SPI 设备驱动可能支持多种型号的设备，可以在.id_table 中给出所有支持的设备信息。其中，结构 spi_driver 的定义如下：

```
struct spi_device_id {
        char name[SPI_NAME_SIZE];
        kernel_ulong_t driver_data /* Data private to the driver */
                __attribute__((aligned(sizeof(kernel_ulong_t))));
};

struct spi_driver {
        const struct spi_device_id *id_table;
        int (*probe)(struct spi_device *spi);
        int (*remove)(struct spi_device *spi);
        void (*shutdown)(struct spi_device *spi);
        int (*suspend)(struct spi_device *spi, pm_message_t mesg);
        int (*resume)(struct spi_device *spi);
        struct device_driver driver;
};
```

3.1.2 spi_unregister_driver()

3.1.2.1 函数原型

```
void spi_unregister_driver(struct spi_driver *sdrv)
```

3.1.2.2 功能描述

注销一个 SPI 设备驱动。

3.1.2.3 参数说明

sdrv，spi_driver 类型的指针，其中包含了 SPI 设备的名称、probe 等接口信息。

3.1.2.4 返回值

无

3.2 数据传输接口

SPI 设备驱动使用 ``struct spi_message" 向 SPI 总线请求读写 I/O。一个 spi_message 中包含了一个操作序列，每一个操作称作 spi_transfer，这样方便 SPI 总线驱动中串行的执行一个个原子的序列。spi_message 和 spi_transfer 的定义也在 spi.h 中：

```
struct spi_transfer {
        const void *tx_buf;
        void *rx_buf;
        unsigned len;

        dma_addr_t tx_dma;
        dma_addr_t rx_dma;

        unsigned cs_change:1;
        u8 bits_per_word;
        u16 delay_usecs;
        u32 speed_hz;

        struct list_head transfer_list;
};

struct spi_message {
        struct list_head transfers;
        struct spi_device *spi;
        unsigned is_dma_mapped:1;
        void (*complete)(void *context);
        void *context;
        unsigned actual_length;
        int status;
        struct list_head queue;
        void *state;
```

```
};
```

3.2.1 spi_message_init()

3.2.1.1 函数原型

```
void spi_message_init(struct spi_message *m)
```

3.2.1.2 功能描述

初始化一个 SPI message 结构，主要是清零和初始化 transfer 队列。

3.2.1.3 参数说明

m，spi_message 类型的指针。

3.2.1.4 返回值】：无

3.2.2 spi_message_add_tail()

3.2.2.1 函数原型

```
void spi_message_add_tail(struct spi_transfer *t, struct spi_message *m)
```

3.2.2.2 功能描述

向 SPI message 中添加一个 transfer。

3.2.2.3 参数说明

t，指向待添加到 SPI transfer 结构; m，spi_message 类型的指针。

3.2.2.4 返回值

无

3.2.3 spi_sync()

3.2.3.1 函数原型

```
nt spi_sync(struct spi_device *spi, struct spi_message *message)
```

3.2.3.2 功能描述

启动、并等待 SPI 总线处理完指定的 SPI message。

3.2.3.3 参数说明

spi，指向当前的 SPI 设备; m，spi_message 类型的指针，其中有待处理的 SPI transfer 队列。

3.2.3.4 返回值

0，成功；小于 0，失败。

4. demo

4.1 drivers/mtd/device/m25p80.c

此 SPI 设备是一个 Nor Flash，需要 MTD 架构的支持。其中调用调用 spi_register_driver() 注册 SPI 驱动，用户主要工作是实现 probe 函数和要通过 SPI message 实现数据的读写。

```
static struct platform_driver sunxi_spi_driver = {
        .probe = sunxi_spi_probe,
        .remove = sunxi_spi_remove,
        .driver = {
                .name = SUNXI_SPI_DEV_NAME,
                .owner = THIS_MODULE,
                .pm = SUNXI_SPI_DEV_PM_OPS,
                .of_match_table = sunxi_spi_match,
        },
};

static int __init sunxi_spi_init(void)
{
        return platform_driver_register(&sunxi_spi_driver);
}

static void __exit sunxi_spi_exit(void)
{
        platform_driver_unregister(&sunxi_spi_driver);
}

subsys_initcall(sunxi_spi_init);
module_exit(sunxi_spi_exit);
```

5. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.