Linux SPI NG
开发指南

版本号: 2.0
发布日期: 2024.11.13

深圳市宇芯数码技术有限公司yfzy_01

版本历史

版本号	日期	制/修订人	内容描述
1.0	2023.01.03	AWA1979	初始版本
1.1	2023.04.10	AWA1979	增加 MR527 支持
1.2	2023.05.29	AWA1979	增加 AI985 支持
1.3	2023.05.31	AWA1979	1. 增加 A527 支持 2. 增加新老驱动区别说明
1.4	2023.07.23	AWA1979	1. 增加 T527 支持 2. 增加 fifo debug 节点
1.5	2023.11.02	AWA1979	1. 更新 dts 配置 2. 更新 slave 模式测试示例 3. 更新驱动调试节点
1.6	2024.02.23	AWA2155	1. 更新部分章节标题 2. 新增常见排查方法
1.7	2024.04.15	AWA1979	1. 增加 H728 支持 2. 增加 slave 模式全双工测试方法
1.8	2024.08.14	AWA2155	1. 增加 MR536 支持 2. 增加 camera 模式说明
1.9	2024.10.08	AWA1979	增加 V821 支持
2.0	2024.11.13	AWA1979	增加 A733 支持

深圳市宇芯数码技术有限公司yfzy_01

目　录

1 前言 **1**

1.1 文档简介 1

1.2 目标读者 1

1.3 适用范围 1

1.4 文档约定 2

1.4.1 标志说明 2

1.4.2 地址与数据描述方法约定 2

1.4.3 数值单位约定 2

1.5 相关术语介绍 3

1.5.1 硬件术语 3

1.5.2 软件术语 3

2 Allwinner SPI 功能特点 **4**

3 模块介绍及配置说明 **5**

3.1 模块配置说明 5

3.1.1 soc 级设备树配置 5

3.1.2 board 级设备树配置 5

3.1.2.1 引脚 PINMUX 配置 6

3.1.2.2 设备节点配置 6

3.1.3 内核配置 7

3.1.3.1 Allwinner Driver 配置 7

3.1.3.2 Kernel Driver 配置 10

3.2 源码结构介绍 13

3.3 内核框架介绍 14

3.3.1 用户空间 14

3.3.2 内核空间 15

3.3.2.1 SPI 驱动核心层（SPI Core） 15

3.3.2.2 SPI 设备驱动层（SPI Device Driver） 15

3.3.2.3 SPI 控制器驱动层（SPI Master Driver） 15

3.3.3 硬件 15

3.4 驱动框架介绍 15

3.4.1 Master 15

3.4.2 Flash 16

3.4.3 Slave 16

3.4.4 DBI 16

3.4.5 BIT 16

3.4.6 CAMERA 16

版权所有 © 珠海全志科技股份有限公司。保留一切权利

深圳市宇芯数码技术有限公司yfzy_01

4 模块接口说明 17
4.1 设备注册接口 . . . 17
4.1.1 spi_register_driver . . . 17
4.1.2 spi_unregister_driver . . . 17
4.2 数据传输接口 . . . 17
4.2.1 spi_message_init . . . 19
4.2.2 spi_message_add_tail . . . 19
4.2.3 spi_sync . . . 19
4.2.4 spi_async . . . 19
4.2.5 spi_slave_abort . . . 20

5 模块功能开发 21
5.1 Master 模式功能开发 . . . 21
5.1.1 Master 模式开发流程 . . . 21
5.1.1.1 软件配置 . . . 21
5.1.1.2 设备树配置 . . . 21
5.1.1.3 Master 模式注意事项 . . . 21
5.1.1.4 Master 模式编程示例 . . . 22
5.1.2 Master 模式测试流程 . . . 22
5.2 Slave 模式功能开发 . . . 24
5.2.1 Slave 模式开发流程 . . . 24
5.2.1.1 软件配置 . . . 24
5.2.1.2 设备树配置 . . . 24
5.2.1.3 Slave 模式注意事项 . . . 25
5.2.1.4 Slave 模式编程示例 . . . 25
5.2.2 Slave 模式测试流程 . . . 26
5.2.3 Slave 模式客户定制方法 . . . 29
5.3 Flash 模式功能开发 . . . 30
5.3.1 Flash 模式开发流程 . . . 30
5.3.1.1 软件配置 . . . 30
5.3.1.2 设备树配置 . . . 30
5.3.1.3 Flash 模式注意事项 . . . 31
5.3.1.4 Flash 模式编程示例 . . . 31
5.3.2 Flash 模式测试方法 . . . 31
5.4 DBI 模式功能开发 . . . 32
5.4.1 DBI 模式开发流程 . . . 32
5.4.1.1 软件配置 . . . 32
5.4.1.2 设备树配置 . . . 32
5.4.1.3 DBI 模式注意事项 . . . 33
5.4.1.4 DBI 模式编程示例 . . . 33
5.4.2 DBI 模式测试流程 . . . 33
5.5 BIT 模式功能开发 . . . 33
5.5.1 BIT 模式开发流程 . . . 33

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.5.1.1 软件配置 . . . 33
5.5.1.2 设备树配置 . . . 33
5.5.1.3 BIT 模式注意事项 . . . 34
5.5.1.4 BIT 模式编程示例 . . . 34
5.5.2 BIT 模式测试流程 . . . 34
5.6 CAMERA 模式功能开发 . . . 35
5.6.1 CAMERA 模式开发流程 . . . 35
5.6.1.1 软件配置 . . . 35
5.6.1.2 设备树配置 . . . 35
5.6.1.3 CAMERA 模式注意事项 . . . 36
5.6.1.4 CAMERA 模式编程示例 . . . 36
5.6.2 CAMERA 模式测试方法 . . . 36

6 调试方法 **38**
6.1 调试工具 . . . 38
6.2 驱动调试节点 . . . 38
6.2.1 debug_mask . . . 38
6.2.2 version . . . 38
6.3 控制器调试节点 . . . 38
6.3.1 info . . . 38
6.3.2 status . . . 39
6.3.3 fifo . . . 39
6.3.4 dump . . . 40
6.3.5 calibrate . . . 40

7 常见问题 **41**
7.1 如何区分当前使用是新/老驱动及驱动版本 . . . 41
7.1.1 通过 dtsi compatible 属性 . . . 41
7.1.2 通过内核配置 deconfig . . . 41
7.1.3 通过启动 log 打印 . . . 41
7.2 如何强制使用 CPU 传输 . . . 42
7.3 常见排查方法 . . . 42
7.4 FAQ . . . 42

版权所有 © 珠海全志科技股份有限公司。保留一切权利

插 图

图 3-1 Allwinner BSP 配置选项 7
图 3-2 Device Drivers 配置选项 8
图 3-3 SPI Drivers 配置选项 9
图 3-4 SPI Support for Allwinner SoCs 配置选项 10
图 3-5 Device Drivers 配置选项 11
图 3-6 SPI support 配置选项 12
图 3-7 SPI support 配置选项 13
图 3-8 Linux SPI 体系结构图 14
图 4-1 Linux SPI 数据传输流程 18
图 5-1 Slave CPU 32B 27
图 5-2 Slave DMA 2048B 28
图 5-3 SPI BIT 模式波形图 35
图 5-4 BF20A6 4-bit 模组实际效果图 37

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1 前言

1.1 文档简介

介绍 SPI NG 模块的使用方法，方便开发人员使用。

1.2 目标读者

SPI NG 模块的驱动开发/维护人员。

1.3 适用范围

从 A523 开始，启用重构后的全新 SPI 驱动，以适配内核新的 SPI 框架。

为了在主分支中兼容所有平台，旧驱动依旧保留。重构后的驱动将使用新名字 SPI-NG，其中 NG 为 next generation 的缩写。

表 1-1: 适用产品列表

产品名称	内核版本
A523	Linux-5.15
MR527	Linux-5.15
AI985	Linux-5.15
A527	Linux-5.15
T527	Linux-5.15
H728	Linux-5.15
MR536	Linux-5.15
V821	Linux-5.4

版权所有 © 珠海全志科技股份有限公司。保留一切权利

深圳市宇芯数码技术有限公司yfzy_01

A733 Linux-6.6

A733 Linux-6.6

TV323 Linux-5.15

1.4 文档约定

1.4.1 标志说明

注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-2: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

表 1-3: 数值单位约定

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

1.5 相关术语介绍

1.5.1 硬件术语

表 1-4: 硬件术语

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

1.5.2 软件术语

表 1-5: 软件术语

术语	解释说明
SPI Master	SPI 主模式
SPI Slave	SPI 从模式
SPI Device	挂在 SPI 总线上的设备
SPI Camera	使用 SPI 接口的 CAMERA 模组设备，此模式下，模组为主，SoC 为从

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2 Allwinner SPI 功能特点

SPI 是一种高速、高效率的串行外围设备接口。由 Motorola 公司提出，是一种高速、全双工、同步通信总线。SPI 以主从方式工作，通常是有一个主设备和一个或多个从设备，无应答机制。

全志的 SPI 控制器支持以下功能：

- 全双工同步串行接口
- Master/Slave 模式可配置
- 支持最大 100MHz 时钟频率
- 支持 SPI Mode0/1/2/3
- 支持多片选
- 片选和时钟的极性和相位可配置
- 支持 CPU(中断) 或 DMA 模式传输
- 支持 TX/RX FIFO 缓存
- 支持 Standard Single/Dual/Quad 数据模式（Slave 模式不支持 Dual/Quad）
- 支持 BIT 模式，用于 3Wire 场景，支持可编程 0~32bits 帧长度（仅支持 Master 模式，且不支持 DMA 和 FIFO 功能）
- 支持 DBI 模式，用于显示设备场景，用于传输视频数据 *
- 支持 Camera 模式，用于 SPI-Camera 场景，支持 Single/Dual/Quad-input 数据模式，具有 VSYNC/FRAMEHEAD/IDLEWAIT 三种硬件帧检测功能 *

说明

其中 DBI/Camera 模式并非所有平台的控制器都支持，详细情况见芯片手册说明

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 模块介绍及配置说明

3.1 模块配置说明

3.1.1 soc 级设备树配置

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，在设备树中配置参数相似，平台设备树文件的路径为：bsp/configs/内核版本/CHIP.dtsi(CHIP 为研发代号，如 sun55iw3p1 等)：

```
#include <dt-bindings/spi/sunxi-spi.h>
......
spi0: spi@4025000 {
   #address-cells = <1>;
   #size-cells = <0>;
   compatible = "allwinner,sunxi-spi-v1.3";      // 具体的设备，用于驱动和设备的绑定
   reg = <0x0 0x04025000 0x0 0x1000>;          // 模块寄存器地址
   interrupts = <GIC_SPI 16 IRQ_TYPE_LEVEL_HIGH>; // 总线中断号、中断类型
   clocks = <&ccu CLK_PLL_PERI0_300M>, <&ccu CLK_SPI0>, <&ccu CLK_BUS_SPI0>;  // 设备使用的时钟
   clock-names = "pll", "mod", "bus";          // 设备使用的时钟名称
   resets = <&ccu RST_BUS_SPI0>;               // 设备的reset时钟
   dmas = <&dma 22>, <&dma 22>;                // 控制器使用的dma通道号
   dma-names = "tx", "rx";                  // 控制器使用通道号对应的名字
   clock-frequency = <100000000>;              // 控制器支持的最大时钟频率（通常为100MHz）
   sunxi,spi-num-cs = <1>;                  // 控制器支持的最大片选数量（一般CS0默认引出，CS1~3根据IC平台决定）
   status = "disabled";                  // 控制器是否使能
};
```

为了在 SPI 总线驱动代码中区分每一个 SPI 控制器，需要在设备树中为每个 SPI 控制器指定别名：

```
aliases {
   ......
   spi0 = &spi0;
   ......
};
```

别名形式为字符串 “spi” 加连续编号的数字，在 SPI 总线驱动程序中可以通过of_alias_get_id函数获取对应 SPI 控制器的数字编号，从而区别每一个 SPI 控制器。

3.1.2 board 级设备树配置

配置文件路径为：/device/config/chips/{IC}/configs/{BOARD}/board.dts, 用于保存每一个板级平台设备差异化的信息的补充，具体配置如下：

3.1.2.1 引脚 PINMUX 配置

```
spi0_pins_default: spi0@0 {
    pins = "PC12", "PC2", "PC4"; /* clk, mosi, miso */
    function = "spi0";
    drive-strength = <10>;
};

spi0_pins_cs: spi0@1 {
    pins = "PC3";
    function = "spi0";
    drive-strength = <10>;
    bias-pull-up;   /* cs, hold, wp should be pulled up */
};

spi0_pins_sleep: spi0@2 {
    pins = "PC12", "PC2", "PC4", "PC3";
    function = "gpio_in";
    drive-strength = <10>;
};
```

3.1.2.2 设备节点配置

这里仅展示控制器节点的配置方法，其他模式或子节点配置见下文模块功能开发章节

```
&spi0 {
    pinctrl-0 = <&spi0_pins_default &spi0_pins_cs>;
    pinctrl-1 = <&spi0_pins_sleep>;
    pinctrl-names = "default", "sleep";
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_MASTER>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
    status = "okay";
};
```

- sunxi,spi-num-cs：定义该 SPI 控制器最多支持多少个片选。
- sunxi,spi-bus-mode：定义 SPI 控制器所处的状态，可配置为如下几种模式。
  - SUNXI_SPI_BUS_MASTER：处于 Master 模式，外接 SPI Device。
  - SUNXI_SPI_BUS_SLAVE：处于 Slave 模式，被其他 Master 访问。
  - SUNXI_SPI_BUS_DBI：处于 DBI 模式，外接 DBI 屏幕。
  - SUNXI_SPI_BUS_BIT：处于 BIT 模式，使用 3Wire 方式进行数据传输。
  - SUNXI_SPI_BUS_NOR：处于 NOR 模式，用于外接 SPI Nor Flash 的情况。
  - SUNXI_SPI_BUS_NAND：处于 NAND 模式，用于外接 SPI Nand Flash 的情况。
  - SUNXI_SPI_BUS_CAMERA：处于 CAMERA 模式，用于外接 SPI Camera 的情况
- sunxi,spi-cs-mode：定义 SPI 控制器对片选的控制方式，可配置为如下几种模式。
  - SUNXI_SPI_CS_AUTO：硬件自动控制，不需要驱动或软件介入。
  - SUNXI_SPI_CS_SOFT：软件手动控制，由驱动完成相关操作。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

说明

- **更多配置属性可参考内核文档Documentation/devicetree/bindings/spi/spi-controller.yaml**

3.1.3 内核配置

运行对应命令进入 kernel menuconfig 界面

3.1.3.1 Allwinner Driver 配置

选择 Allwinner BSP 选项进入下一级配置，如下图所示。

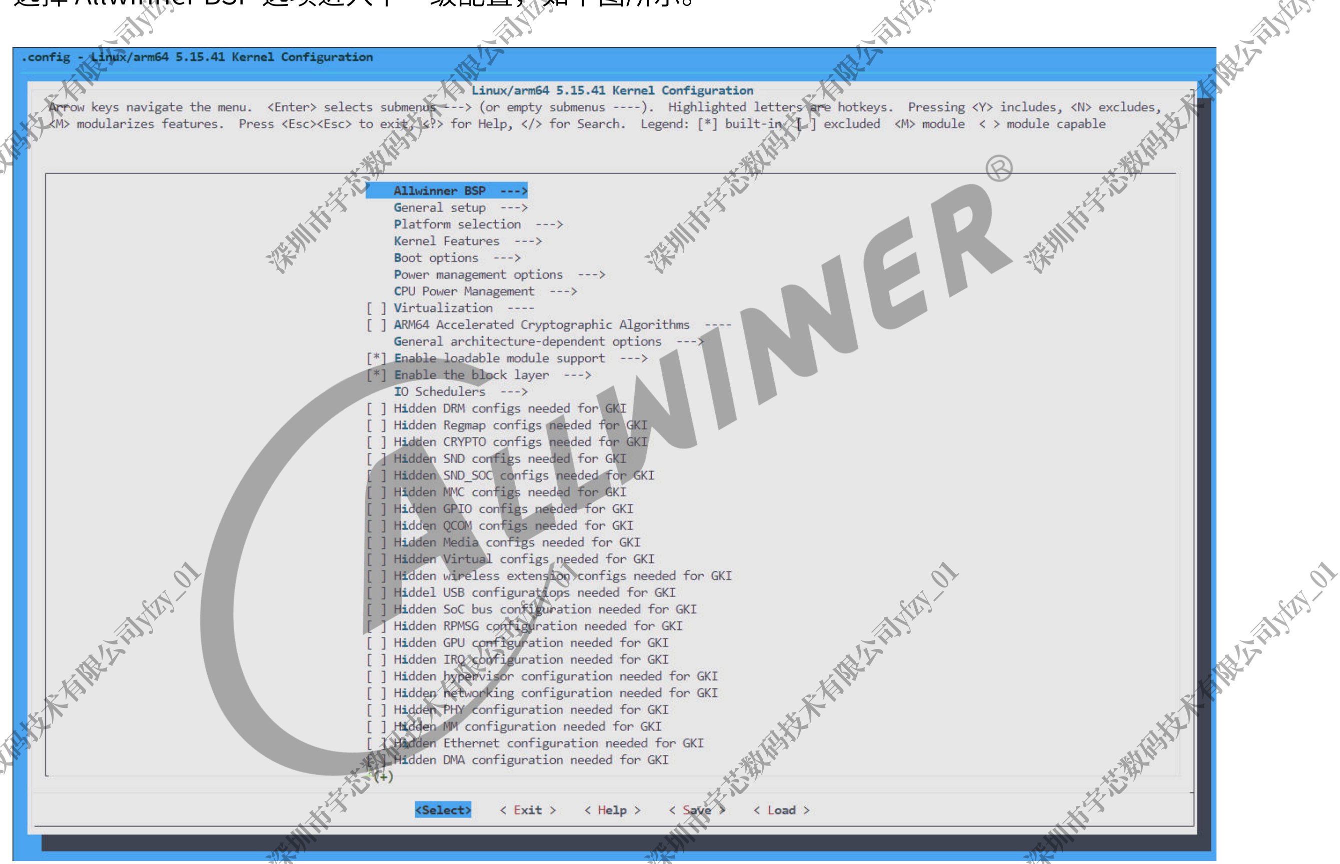


图 3-1: Allwinner BSP 配置选项

选择 Device Drivers 选项进入下一级配置，如下图所示。

深圳市芯数码技术有限公司yfzy_01

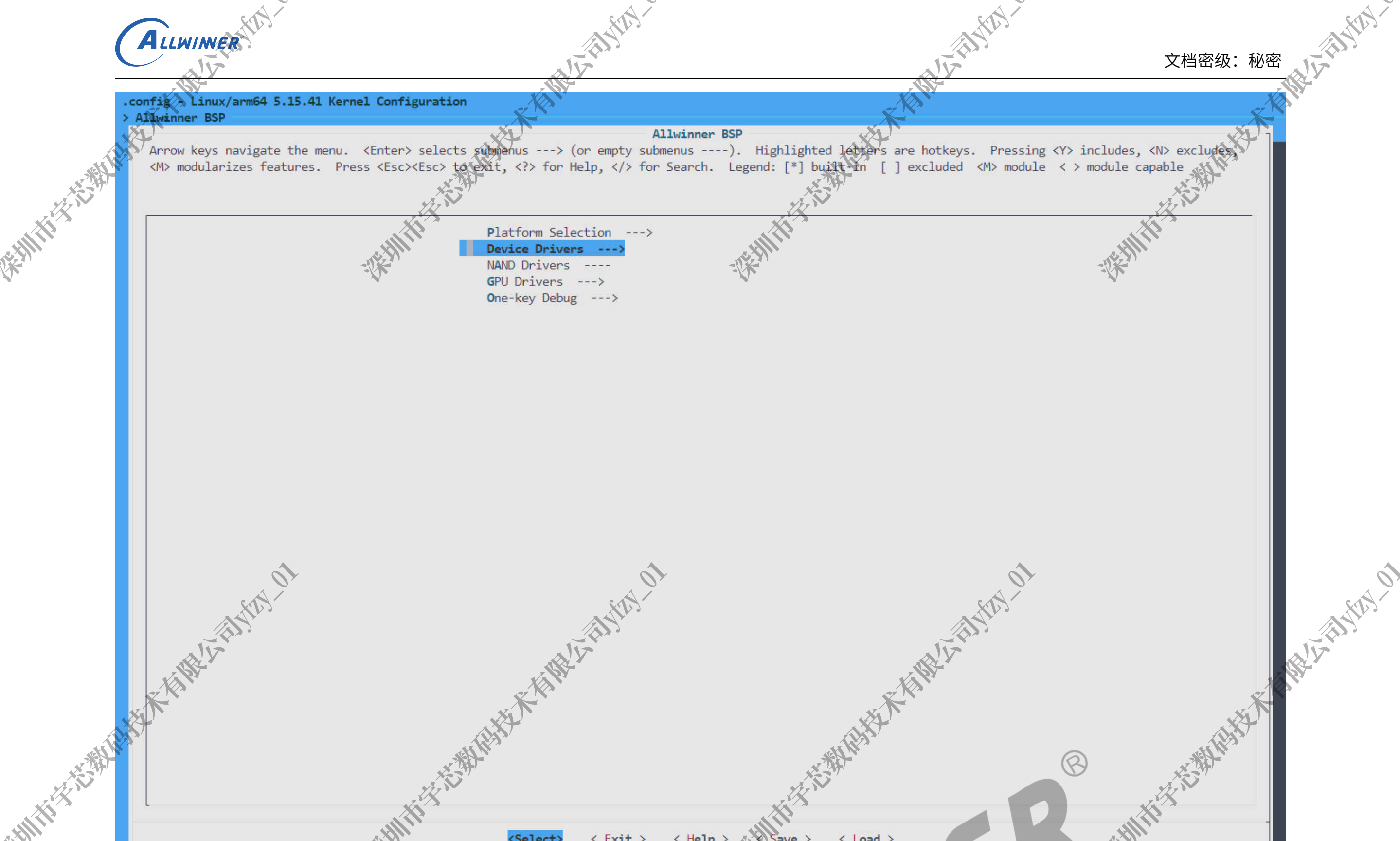


图 3-2: Device Drivers 配置选项

选择 SPI Drivers 选项进入下一级配置，如下图所示。

深圳市宇芯数码技术有限公司yfzy_01

```
.config - Linux/arm64 5.15.41 Kernel Configuration
> Allwinner BSP > Device Drivers
                                   Device Drivers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                Clock Drivers  --->
                Pinctrl Drivers  --->
                UART Drivers  --->
                Timer Drivers  --->
                DMA Drivers  --->
                RTC Drivers  --->
                Bus Drivers  --->
                Dump-Reg Drivers  --->
                Watchdog Drivers  --->
                I2C Drivers  --->
                SPI Drivers  --->
                SPI NG Drivers  --->
                LED Drivers  --->
                PWM Drivers  --->
                IR-TX Drivers  --->
                IR-RX Drivers  --->
                General Purpose ADC Drivers  --->
                Low Rate ADC Drivers  --->
                Resistive Touch Panel Drivers  --->
                USB Host Controller Drivers  --->
                USB Device Drivers  --->
                IOMMU Drivers  --->
                PCIe Drivers  --->
                PHY Drivers  --->
                Emac Drivers  --->
                Gmac Drivers  --->
                MBUS Drivers  ----
                MSGBOX Drivers  --->
                HWSPINLOCK Drivers  --->
                IRQ-Chip Drivers  --->
                Standby Debugging Drivers  --->
                Thermal Drivers  --->
                NVMEM Drivers  --->
                Power Domain Drivers  --->
                PMIC Drivers  --->
              (+)

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

图 3-3: SPI Drivers 配置选项

选择 SPI Support for Allwinner SoCs 选项，可选择直接编译进内核，也可编译成模块。如下图所示。

深圳市宇芯数码技术有限公司yfzy_01

版权所有 © 珠海全志科技股份有限公司。保留一切权利

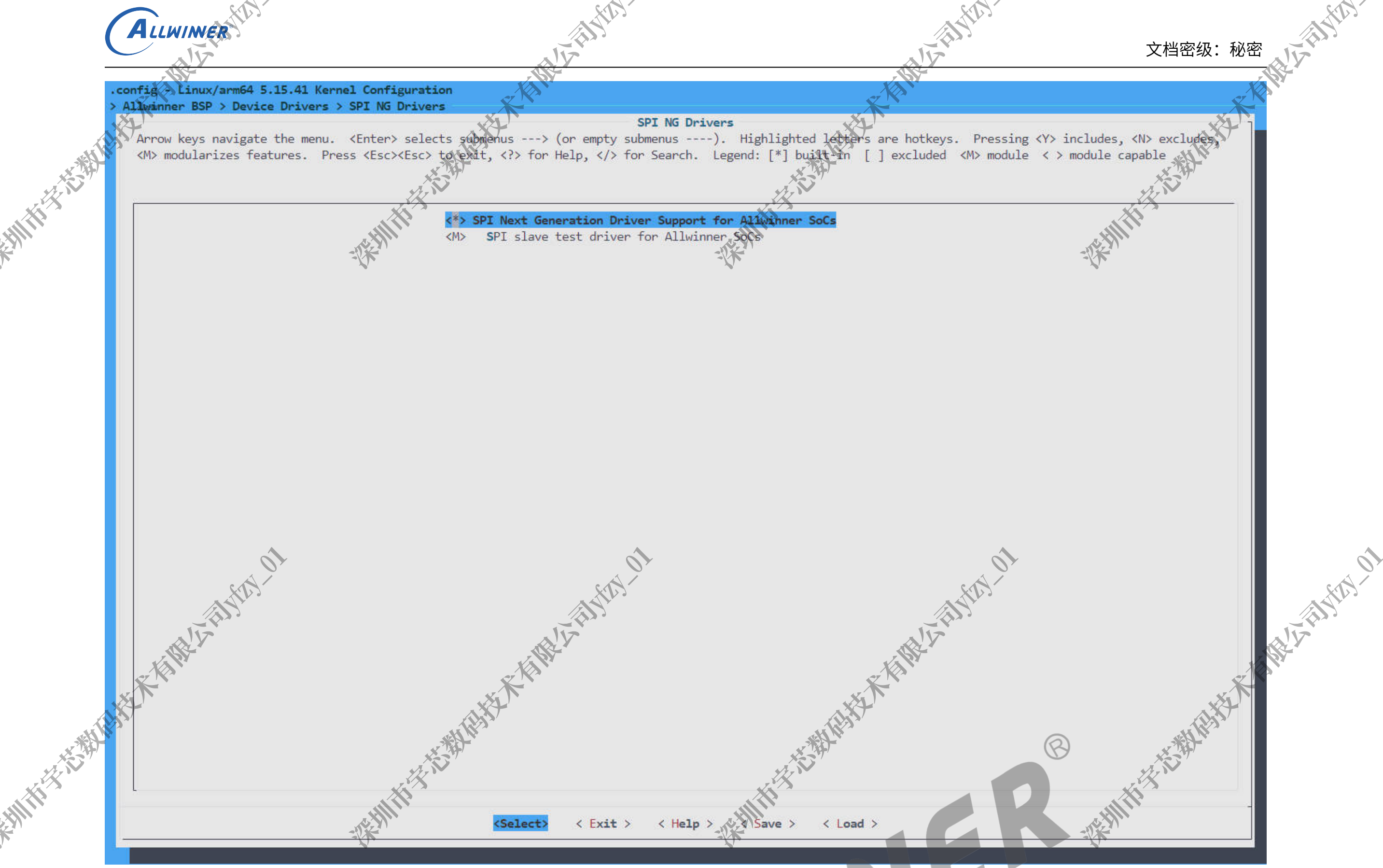


图 3-4: SPI Support for Allwinner SoCs 配置选项

3.1.3.2 Kernel Driver 配置

选择 Device Drivers 选项进入下一级配置，如下图所示。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

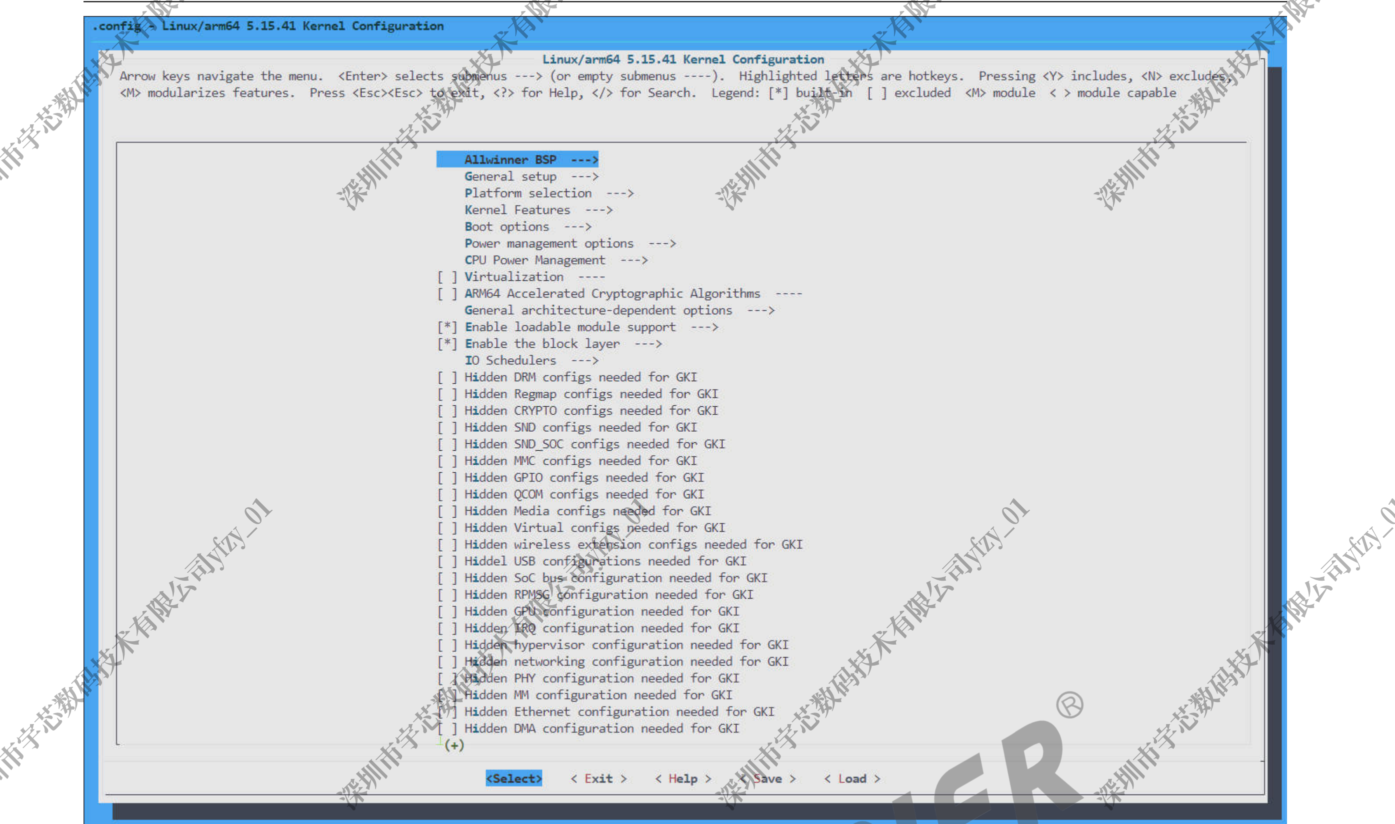


图 3-5: Device Drivers 配置选项

选择 SPI support 选项，进入下一级配置，如下图所示。

深圳市宇芯数码技术有限公司yfzy_01

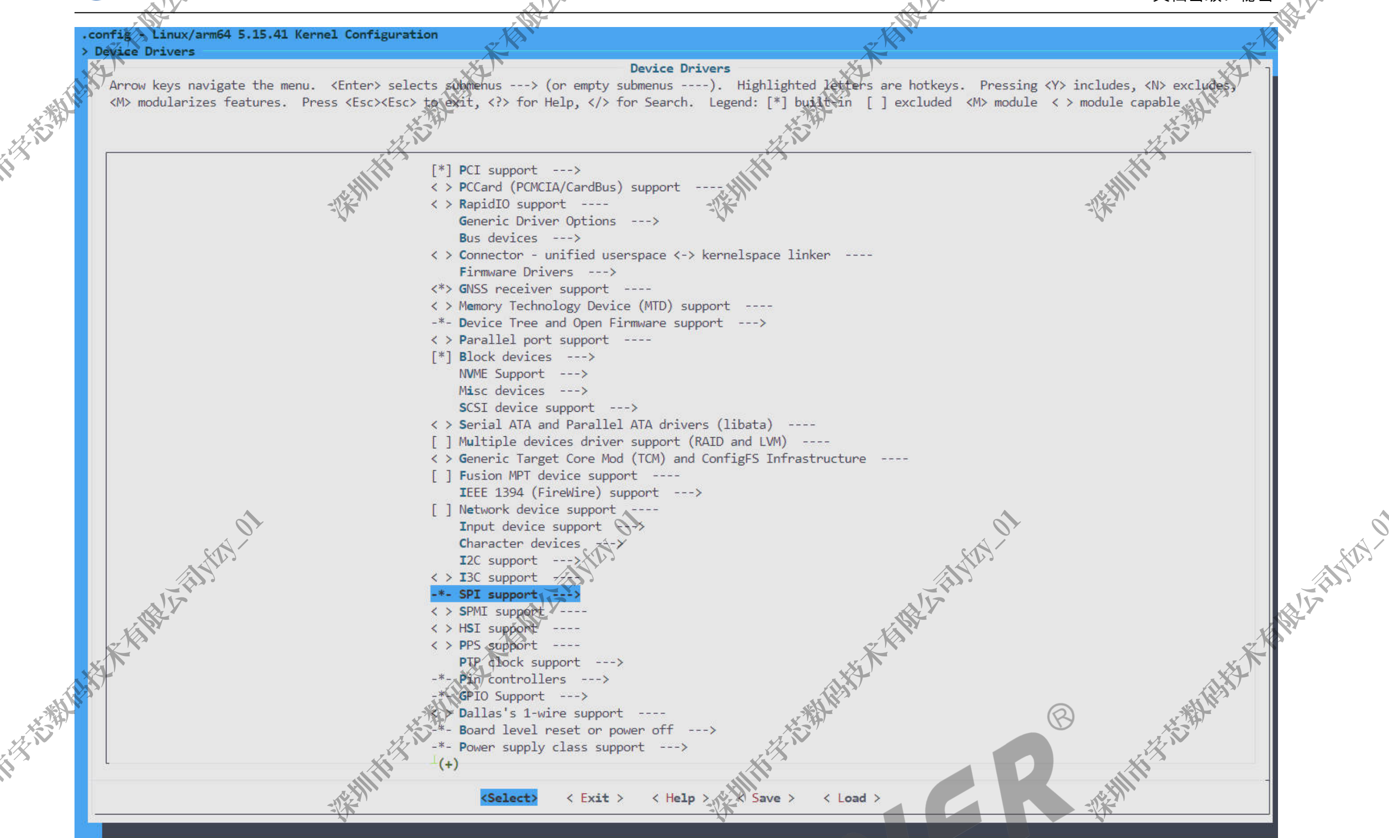


图 3-6: SPI support 配置选项

如果想要放开 spi 的一些调试打印，可以选上Debug support for SPI drivers。

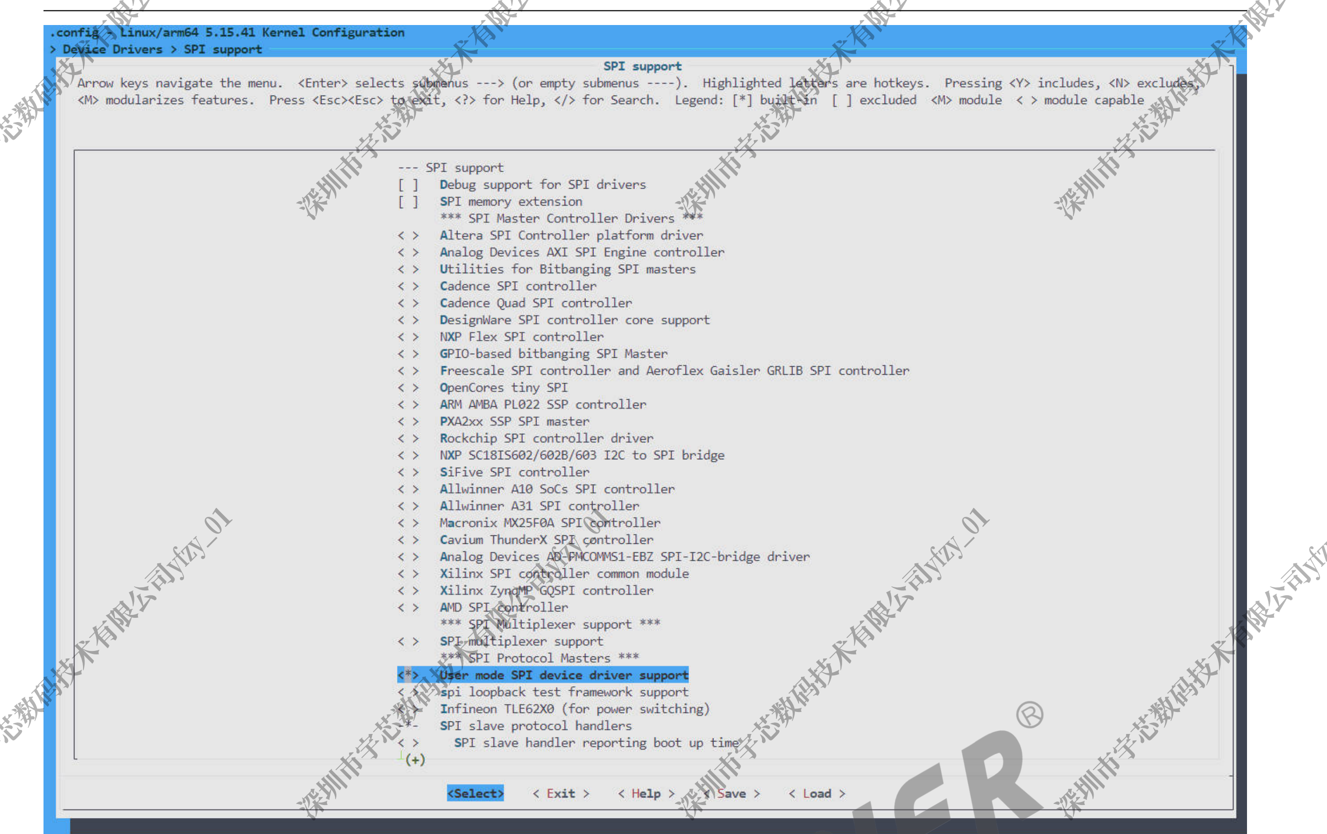
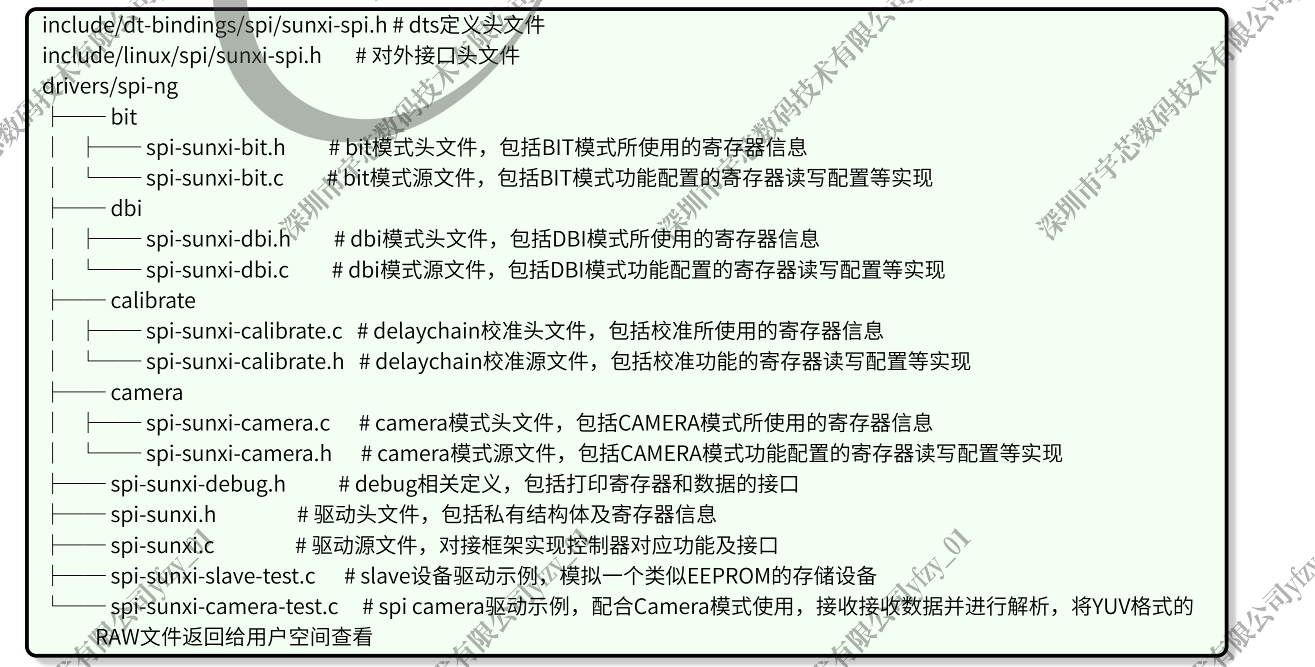


图 3-7: SPI support 配置选项

3.2 源码结构介绍

SPI 总线驱动的源代码位于 SDK 的 bsp/drivers/spi 目录下：



```
include/dt-bindings/spi/sunxi-spi.h # dts定义头文件
include/linux/spi/sunxi-spi.h      # 对外接口头文件
drivers/spi-ng
├── bit
│   ├── spi-sunxi-bit.h      # bit模式头文件，包括BIT模式所使用的寄存器信息
│   └── spi-sunxi-bit.c      # bit模式源文件，包括BIT模式功能配置的寄存器读写配置等实现
├── dbi
│   ├── spi-sunxi-dbi.h      # dbi模式头文件，包括DBI模式所使用的寄存器信息
│   └── spi-sunxi-dbi.c      # dbi模式源文件，包括DBI模式功能配置的寄存器读写配置等实现
├── calibrate
│   ├── spi-sunxi-calibrate.c  # delaychain校准头文件，包括校准所使用的寄存器信息
│   └── spi-sunxi-calibrate.h  # delaychain校准源文件，包括校准功能的寄存器读写配置等实现
├── camera
│   ├── spi-sunxi-camera.c   # camera模式头文件，包括CAMERA模式所使用的寄存器信息
│   └── spi-sunxi-camera.h   # camera模式源文件，包括CAMERA模式功能配置的寄存器读写配置等实现
├── spi-sunxi-debug.h      # debug相关定义，包括打印寄存器和数据的接口
├── spi-sunxi.h            # 驱动头文件，包括私有结构体及寄存器信息
├── spi-sunxi.c            # 驱动源文件，对接框架实现控制器对应功能及接口
├── spi-sunxi-slave-test.c   # slave设备驱动示例，模拟一个类似EEPROM的存储设备
└── spi-sunxi-camera-test.c   # spi camera驱动示例，配合Camera模式使用，接收接收数据并进行解析，将YUV格式的
    RAW文件返回给用户空间查看
```

其中 bit/dbi/camera 为独立功能，配置可单独裁剪

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.3 内核框架介绍

Linux 中 SPI 体系结构分为三个层次，如下图所示。

应用空间

user application

spi user mode device driver

内核空间

spi device driver

spidev

SPI Core

spi transfer

...

spi message

SPI Master Driver

transfer_one_message

硬件

hardware

图 3-8: Linux SPI 体系结构图

3.3.1 用户空间

包括所有使用 SPI 设备的应用程序，在这一层用户可以根据自己的实际需求，将 spi 设备进行一些特殊的处理，此时控制器驱动程序并不清楚和关注设备的具体功能，SPI 设备的具体功能是由用户层程序完成的。例如，和 MTD 层交互以便把 SPI 接口的存储设备实现为某个文件系统，和 TTY 子系统交互把 SPI 设备实现为一个 TTY 设备，和网络子系统交互以便把一个 SPI 设备实现为一个网络设备，等等。当然，如果是一个专有的 SPI 设备，我们也可以按设备的协议要求，实现自己的专有协议驱动。同时这部分我们不用关注。

深圳市芯数码技术有限公司yfzy_01

3.3.2 内核空间

内核空间我们同样的会分为一下三部分：

3.3.2.1 SPI 驱动核心层（SPI Core）

SPI Core 是 Linux 内核用来维护和管理 spi 的核心部分，SPI Core 提供操作接口函数，允许一个 spi master，spi driver 和 spi device 初始化时在 SPI Core 中进行注册，以及退出时进行注销。

3.3.2.2 SPI 设备驱动层（SPI Device Driver）

SPI Driver 是对应于 spi 设备端的驱动程序，通过接口函数向 SPI Core 进行注册，SPI Driver 的作用是将 spi 设备挂接到 spi 总线上。

3.3.2.3 SPI 控制器驱动层（SPI Master Driver）

SPI Master 针对不同类型的 spi 控制器硬件，实现 spi 总线的硬件访问操作。SPI Master 通过接口函数向 SPI Core 注册一个控制器。

3.3.3 硬件

这一层是实际的物理器件，其中包括我们的 spi 控制器以及与控制器相连的各个 spi 子设备，通过 spi 总线能够与 cpu 进行数据的交互。

3.4 驱动框架介绍

根据 SPI 控制器集成了 BYTE/BIT/DBI 三个 IP 的特性，将驱动分 3 个 IP 域和 5 个功能逻辑块：master/slave/flash/dbi/bit。其中 BIT/DBI 为一一对应关系，BYTE 则对应 Master/Flash/Slave 三个不同功能逻辑块。

3.4.1 Master

该模式属于 BYTE 模式的一种，数据的最小单位为 BYTE，即一次发送会产生 8 个 CLK 将一个 Byte 数据全部发送出去。支持 Single/Dual/Quad 三种模式，常见的使用常见为通用 SPI 外设。

3.4.2 Flash

该模式属于 BYTE 模式的一种，但驱动专门针对 Nor/Nand Flash 传输过程做了优化，当外设连接的是 Flash 的情况下，可以提高读写性能。支持 1-1-1，1-1-2，1-1-4 三种 SPI-Flash 模式。只能使用于 SPI-Flash 外设。

3.4.3 Slave

该模式属于 BYTE 模式的一种，但此时控制器处于从机模式下，等待其他 Master 的访问，在该模式下仅支持使用 Single Mode 进行数据传输。针对数据同步的问题，增加 ready 脚的 sync 机制，该机制为纯软件逻辑，需要对方的 Master 端的适配支持，才能正常使用。若不用 ready 同步机制，则需要上层应用在传输时进行相应的 delay 等待。

3.4.4 DBI

该模式专门针对 SPI 显示外设设计，使用 DBI 协议。支持 3 线或四线传输，且同时兼容多种视频数据格式。

3.4.5 BIT

该模式专门针对特殊外设设计，数据的最小单位为 BIT，即可以将数据分成单个 Bit 在 MOSI/MISO 上进行发送或接受，一个 CLK 对应一个 bit，长度为 0~32 可编程。常见的使用场景为 3Wire 模式。

3.4.6 CAMERA

该模式专门针对 SPI 摄像头模组设计，此时控制器处于从机模式下，支持 Single/Dual/Quad-input 数据模式，具有 VSYNC/FRAMEHEAD/IDLEWAIT 三种硬件帧检测功能

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 模块接口说明

4.1 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 SPI 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)
```

4.1.1 spi_register_driver

- 原型:int spi_register_driver(struct spi_driver *sdrv)
- 作用: 注册一个 SPI 设备驱动
- 参数：
  - 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
- 返回：
  - 0: 成功
  - 其他: 失败

4.1.2 spi_unregister_driver

- 原型：void spi_unregister_driver(struct spi_driver *sdrv)
- 作用：注销一个 SPI 设备驱动。
- 参数：
  - 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。
- 返回：无

4.2 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。内核线程使用队列实现了异步传输的功能，对于同一个数据传输的发起者，既然异步方式无需等待数据传输完成即可返回，返回后，该发起者可以立刻又发起一个 message，而这时上一个 message 还没有处理完。对于另外一个不同的发起者来说，也有可能同时发起一次 message 传输请求。

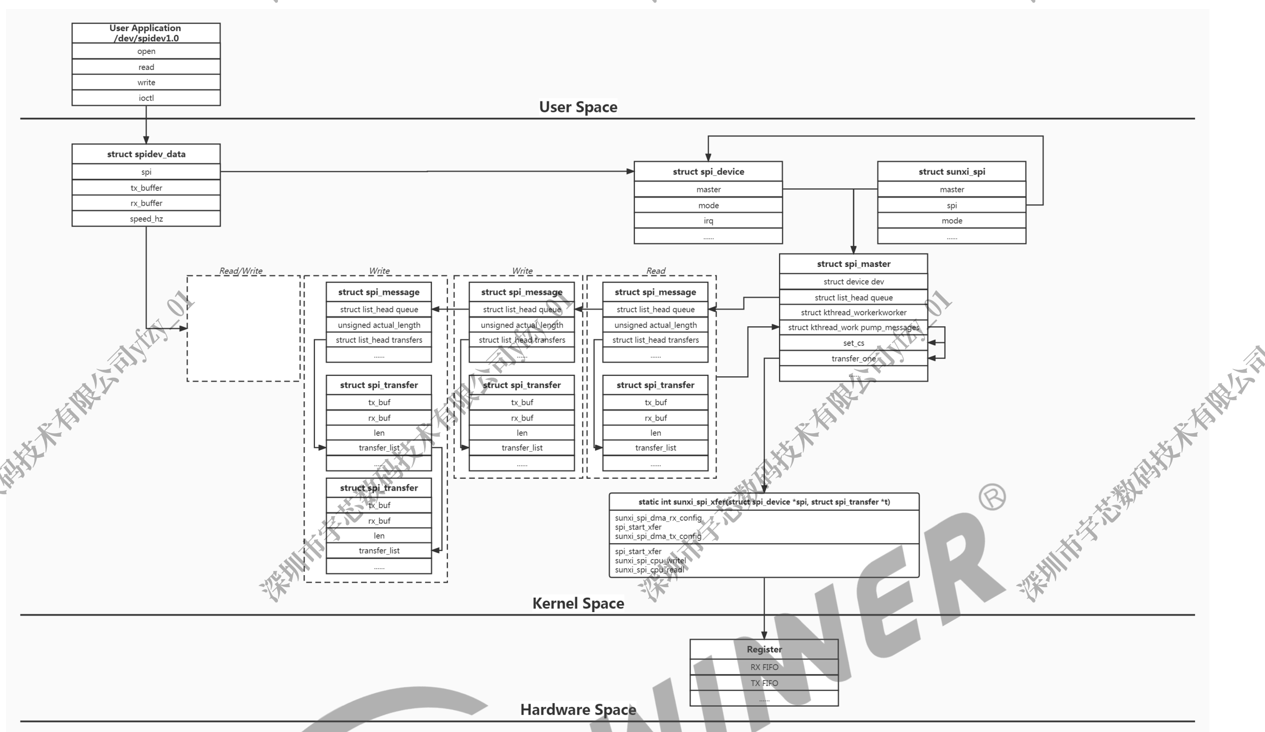


图 4-1: Linux SPI 数据传输流程

```
1   struct spi_transfer {
2       const void  *tx_buf;
3       void        *rx_buf;
4       unsigned    len;
5
6       dma_addr_t tx_dma;
7       dma_addr_t rx_dma;
8
9       unsigned    cs_change:1;
10      u8      bits_per_word;
11      u16     delay_usecs;
12      u32     speed_hz;
13
14      struct list_head transfer_list ;
15  };
16
17  struct spi_message {
18      struct list_head     transfers;
19      struct spi_device    *spi;
20      unsigned        is_dma_mapped:1;
21      void            (*complete)(void *context);
22      void            *context;
23      unsigned        actual_length;
24      int         status;
25      struct list_head     queue;
26      void            *state;
```

```
27 };
```

4.2.1 spi_message_init

- 原型：void spi_message_init(struct spi_message *m)
- 作用：初始化一个 SPI message 结构，主要是清零和初始化 transfer 队列。
- 参数：
  - m:spi_message 类型的指针。
- 返回：无

4.2.2 spi_message_add_tail

- 原型：void spi_message_add_tail(struct spi_transfer *t, struct spi_message *m)
- 作用：向 SPI message 中添加一个 transfer。
- 参数：
  - t: 指向待添加到 SPI transfer 结构;
  - m:spi_message 类型的指针。
- 返回：无

4.2.3 spi_sync

- 原型：int spi_sync(struct spi_device *spi, struct spi_message *message)
- 作用：数据传输同步接口，该函数会阻塞至传输完成才会返回。
- 参数：
  - 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。
- 返回：
  - 0: 成功
  - 小于 0: 失败

4.2.4 spi_async

- 原型：int spi_async(struct spi_device *spi, struct spi_message *message)
- 作用：数据传输异步接口，该函数不会阻塞立刻返回，当数据真正完成后通过回调通知设备驱动。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- 参数：
  - 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。
- 返回：
  - 0: 成功
  - 小于 0: 失败

4.2.5 spi_slave_abort

- 原型：int spi_slave_abort(struct spi_device *spi)
- 作用：当控制器处于从模式时，中止正在进行的传输
- 参数：
  - spi: 指向当前的 SPI 设备;
- 返回：
  - 0: 成功
  - 小于 0: 失败

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5 模块功能开发

5.1 Master 模式功能开发

5.1.1 Master 模式开发流程

5.1.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_SPI_NG
CONFIG_SPI_SPIDEV
```

5.1.1.2 设备树配置

```
&spi1 {
    ......
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_MASTER>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
    status = "okay";

    spidev0 {
        compatible = "rohm,dh2228fv";
        reg = <0x0>;
        spi-max-frequency = <100000000>;
        spi-rx-bus-width = <0x1>;
        spi-tx-bus-width = <0x1>;
        status = "okay";
    };
};
```

5.1.1.3 Master 模式注意事项

- 用户空间测试依赖于/dev/spidevX.Y节点
- SPI 模式配置有许多选项，如：SPI_CPHA|SPI_CPOL|SPI_LSB_FIRST|SPI_CS_HIGH 等

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.1.1.4 Master 模式编程示例

内核空间驱动编程示例参考kernel/drivers/spi/spidev.c

用户空间应用编程示例参考bsptest/cases/spi/apps/spidev-test.c或kernel/tools/spi/spidev_test.c

更多详细说明参考内核文档kernel/Documentation/spi/spidev.rst

- 打开设备并配置相关参数

```
int mode = SPI_MODE_0;
int bits = 8;
int speed = 10000000;
int fd;

fd = open("/dev/spidevX.Y", O_RDWR);
ioctl(fd, SPI_IOC_WR_MODE32, &mode); // set spi mode
ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits); // set bit per word
ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed); // set max speed hz
```

- 收发数据

构造传输结构体，调用 ioctl 进行数据传输

```
struct spi_ioc_transfer tr = {
    .tx_buf = (unsigned long)tx,
    .rx_buf = (unsigned long)rx,
    .tx_nbits = 1,
    .rx_nbits = 1,
    .len = len,
    .speed_hz = speed,
};

ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
```

若只发不收，则 rx_buf 指针填为 NULL；若只收不发，则 tx_buf 指针填为 NULL；若同时收发，则指针指向对应数据。

若传输时需要使用 single/dual/quad 模式，则将对应方向的 nbits 参数写为 1/2/4。

- 关闭设备

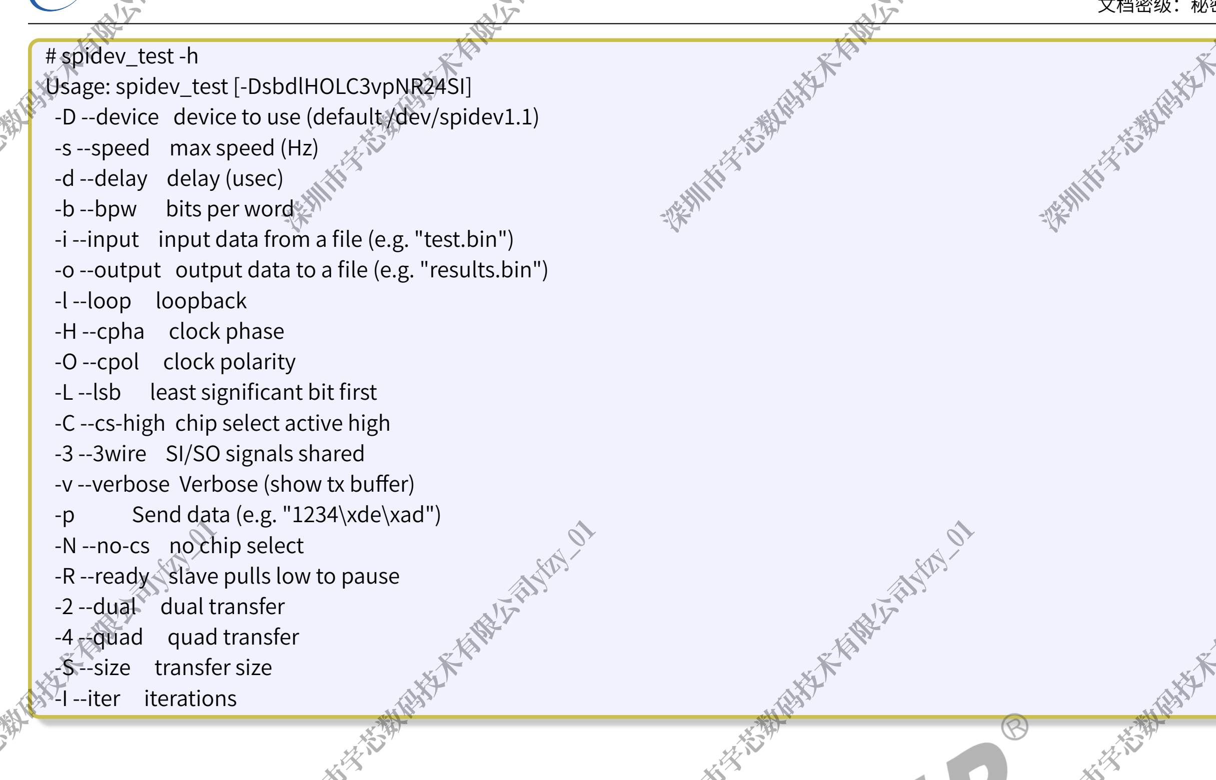
```
close(fd);
```

5.1.2 Master 模式测试流程

硬件上将 SPI 的 MOSI/MISO 引脚短接，使用回环自发自收进行测试。

- Master 测试工具使用方法

版权所有 © 珠海全志科技股份有限公司。保留一切权利



```
# spidev_test -h
Usage: spidev_test [-DsbdlHOLC3vpNR24SI]
  -D --device   device to use (default /dev/spidev1.1)
  -s --speed    max speed (Hz)
  -d --delay    delay (usec)
  -b --bpw      bits per word
  -i --input    input data from a file (e.g. "test.bin")
  -o --output   output data to a file (e.g. "results.bin")
  -l --loop     loopback
  -H --cpha     clock phase
  -O --cpol     clock polarity
  -L --lsb      least significant bit first
  -C --cs-high  chip select active high
  -3 --3wire    SI/SO signals shared
  -v --verbose  Verbose (show tx buffer)
  -p            Send data (e.g. "1234\xde\xad")
  -N --no-cs    no chip select
  -R --ready    slave pulls low to pause
  -2 --dual     dual transfer
  -4 --quad     quad transfer
  -S --size     transfer size
  -I --iter     iterations
```

- CPU 模式回环自测

低于 FIFO 大小数据量的传输，将会使用 CPU 进行搬运

```
/ # spidev-test -D /dev/spidev3.0 -s 100000000 -S 32 -I 1 -v
spi mode: 0x0
bits per word: 8
max speed: 100000000 Hz (100000 kHz)
TX | 52 5D 4D 8A 1A A3 56 DA DF D1 1E 26 9A 41 01 84 0B 4C FC F0 12 F7 34 87 D2 F7 7E 11 DC 35 45 2F  |R]M...V....&.A...L
        ....4...~..5E/|
RX | 52 5D 4D 8A 1A A3 56 DA DF D1 1E 26 9A 41 01 84 0B 4C FC F0 12 F7 34 87 D2 F7 7E 11 DC 35 45 2F  |R]M...V....&.A...L
        ....4...~..5E/|
total size   : 32 B
total time   : 55.33 us
averange rate: 0.55 MB/s
averange time: 55.33 us
```

- DMA 模式回环自测

超过 FIFO 大小数据量的传输，将会使用 DMA 进行搬运

```
/ # spidev-test -D /dev/spidev3.0 -s 10000000 -S 1024 -I 1
spi mode: 0x0
bits per word: 8
max speed: 10000000 Hz (10000 kHz)
total size   : 1.00 KB
total time   : 985.00 us
averange rate: 0.99 MB/s
averange time: 985.00 us
```

- DMA 模式性能测试

深圳市宇芯数码技术有限公司yfzy_01

将速率提高至最高 100MHz，测试接口带宽性能

```
/ # spidev-test -D /dev/spidev3.0 -s 100000000 -S 4096 -I 1024
spi mode: 0x0
bits per word: 8
max speed: 100000000 Hz (100000 kHz)
total size   : 4.00 MB
total time   : 391.14 ms
averange rate: 10.23 MB/s
averange time: 381.97 us
```

100MHz 下的理论值为 12.5MB/s，当前 Master 驱动能达到 81.84% 的性能

5.2 Slave 模式功能开发

5.2.1 Slave 模式开发流程

5.2.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_SPI_NG
CONFIG_AW_SPI_SLAVETEST
```

5.2.1.2 设备树配置

考虑到稳定性，Slave 端建议使用 10MHz 以下的频率

```
&spi2 {
  ......
  clock-frequency = <10000000>;
  pinctrl-names = "default", "sleep";
  sunxi,spi-bus-mode = <SUNXI_SPI_BUS_SLAVE>;
  sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
  status = "okay";

  slave {
    compatible = "sunxi,spislave";
    reg = <0x0>;
    spi-max-frequency = <10000000>;
    slave-cachesize = <4096>;
    status = "okay";
  };
};
```

若要使用 ready 同步功能，需要在 master/slave 两边添加用于同步的引脚，配置如下：

- Master 节点

```
&spi1 {
    ......
    ready-gpio = <&pio PC 15 GPIO_ACTIVE_LOW>;
    ......
};
```

- Slave 节点

```
&spi2 {
    ......
    ready-gpio = <&pio PC 16 GPIO_ACTIVE_LOW>;
    ......
};
```

5.2.1.3 Slave 模式注意事项

- Master 的 MOSI 接 Slave 的 MOSI，Master 的 MISO 接 Slave 的 MISO，如果接错会导致数据收发异常

5.2.1.4 Slave 模式编程示例

内核空间驱动编程示例参考bsp/drivers/spi-ng/spi-sunxi-slave-test.c

用户空间应用编程示例参考bsptest/cases/spi/apps/spislave-test.c

本示例站在 master 的角度展示了向 slave 写入数据再回读的过程

- 打开设备

```
int mode = SPI_MODE_0;
int bits = 8;
int speed = 10000000;
int fd;

fd = open("/dev/spidevX.Y", O_RDWR);
ioctl(fd, SPI_IOC_WR_MODE32, &mode); // set spi mode
ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits); // set bit per word
ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed); // set max speed hz
```

- 写入数据

```
char head_buf[PKT_HEAD_LEN] = { SUNXI_OP_WR, 1, 0x00, 0x00, 0x00, 0x10 };
struct spi_ioc_transfer tr[2];

tr[0].tx_buf = (unsigned long)head_buf;
tr[0].tx_nbits = 1;
tr[0].rx_buf = (unsigned long)NULL;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
tr[0].rx_nbits = 0;
tr[0].len = sizeof(head_buf);
tr[0].speed_hz = speed;

tr[1].tx_buf = (unsigned long)tx_buf;
tr[1].tx_nbits = 1;
tr[1].rx_buf = NULL;
tr[1].len = len;
tr[1].speed_hz = speed;

ioctl(fd, SPI_IOC_MESSAGE(1), &tr[0]);
usleep(10000);
ioctl(fd, SPI_IOC_MESSAGE(1), &tr[1]);
```

- 回读数据

```
char head_buf[PKT_HEAD_LEN] = { SUNXI_OP_RD, 1, 0x00, 0x00, 0x00, 0x10 };
struct spi_ioc_transfer tr[2];

tr[0].tx_buf = (unsigned long)head_buf;
tr[0].tx_nbits = 1;
tr[0].rx_buf = (unsigned long)NULL;
tr[0].rx_nbits = 0;
tr[0].len = sizeof(head_buf);
tr[0].speed_hz = speed;

tr[1].tx_buf = NULL;
tr[1].rx_buf = (unsigned long)rx_buf;
tr[1].rx_nbits = 1;
tr[1].len = len;
tr[1].speed_hz = speed;

ioctl(fd, SPI_IOC_MESSAGE(1), &tr[0]);
usleep(10000);
ioctl(fd, SPI_IOC_MESSAGE(1), &tr[1]);
```

- 关闭设备

```
close(fd);
```

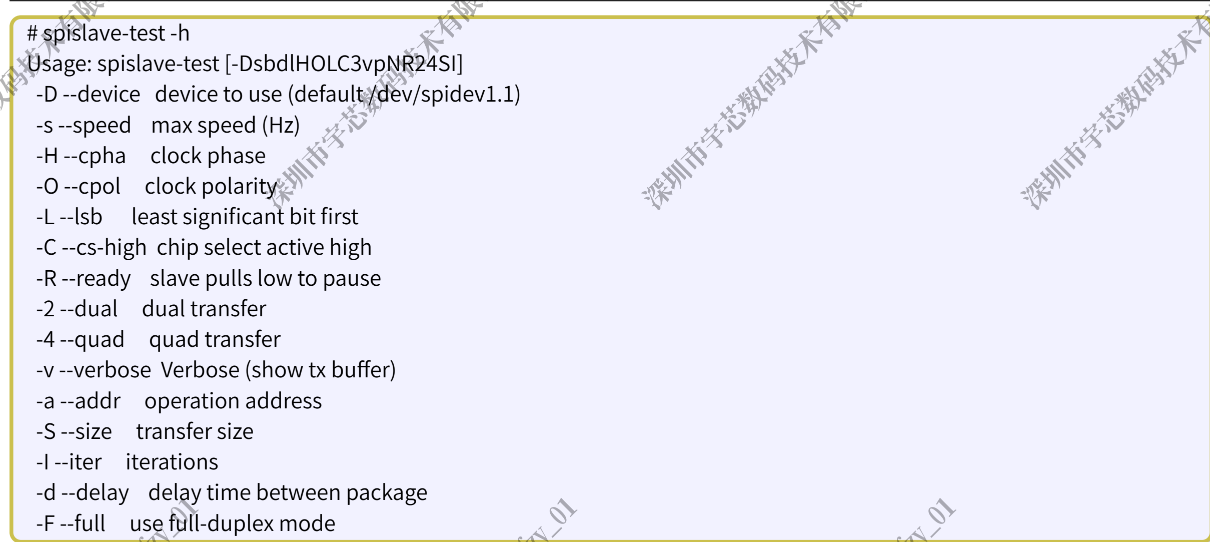
5.2.2 Slave 模式测试流程

使用 2 路 SPI 接口，一个做 Master，一个做 Slave，并将 Master 与 Slave 的信号线按顺序依次连接。

- 加载 Slave 设备驱动

```
# insmod spi-sunxi-slave-test.ko
```

- Slave 测试工具使用方法



```
# spislave-test -h
Usage: spislave-test [-DsbdlHOLC3vpNR24SI]
  -D --device   device to use (default /dev/spidev1.1)
  -s --speed    max speed (Hz)
  -H --cpha     clock phase
  -O --cpol     clock polarity
  -L --lsb      least significant bit first
  -C --cs-high  chip select active high
  -R --ready    slave pulls low to pause
  -2 --dual     dual transfer
  -4 --quad     quad transfer
  -v --verbose  Verbose (show tx buffer)
  -a --addr     operation address
  -S --size     transfer size
  -I --iter     iterations
  -d --delay    delay time between package
  -F --full     use full-duplex mode
```

若使用 ready 同步功能则加上-R 参数，Master/Slave 将自动通过该 gpio 的翻转信号进行数据同步。

若使用 delay 等待功能则加上-d 参数，Master 将会在等待相应时间后再进行数据操作。

若使用全双工模式功能则加上-F 参数，Master/Slave 将在数据包使用全双工方式进行数据收发。

- CPU 模式半双工收发测试

```
/ # spislave-test -D /dev/spidev0.0 -R -s 10000000 -a 0 -S 32 -v
spi mode: 0x80
max speed: 10000000 Hz (10000 kHz)
op addr : 0
op size : 32
package head : { 0x01 0x00 0x00 0x00 0x20 }
package head : { 0x03 0x00 0x00 0x00 0x20 }
TX | A0 07 B0 44 47 9E CD 15 AC 0E CC 85 E8 CF 46 D4 16 F9 4B F8 3F 5A F7 A0 91 22 02 10 B1 BD B6 51  |...DG.........F...K.?Z
     ..."....Q|
RX | A0 07 B0 44 47 9E CD 15 AC 0E CC 85 E8 CF 46 D4 16 F9 4B F8 3F 5A F7 A0 91 22 02 10 B1 BD B6 51  |...DG.........F...K.?Z
     ..."....Q|

/ # hexdump  /sys/class/spi_slave/spi1/spi1.0/sunxi-slave-test
0000000 07a0 44b0 9e47 15cd 0eac 85cc cfe8 d446
0000010 f916 f84b 5a3f a0f7 2291 1002 bdb1 51b6
0000020 ffff ffff ffff ffff ffff ffff ffff ffff
*
0001000
```

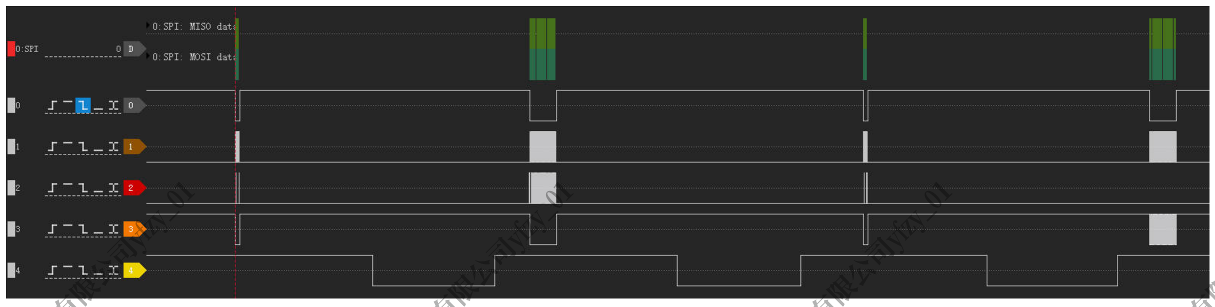


图 5-1: Slave CPU 32B

- CPU 模式全双工收发测试

```
/ # spislave-test -D /dev/spidev0.0 -s 10000000 -a 0 -S 32 -I 1 -d 5000 -F -v
spi mode: 0x0
max speed: 10000000 Hz (10000 kHz)
op addr : 0
op size : 32
package head : { 0x03 0x01 0x00 0x00 0x00 0x20 }
package head : { 0x03 0x01 0x00 0x00 0x00 0x20 }
TX0 | 0F 48 9D 8B 18 1B 49 28 3D AD E2 29 6A 59 7E 54 EE 9D 09 1E 09 88 DE 84 BD F8 7E 49 60 0B D5 6C  |.H....I(=..)jY~T
        ..........~I`..l|
RX0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  |................................|
TX1 | BB 0F 6D 93 E1 67 A4 31 0C DC 4D AA 3F 23 32 8D 9C 8B 28 21 3A 53 B0 1B 31 9F 21 E8 A2 DE 5F C3  |..m..g.1..M
        .?#2...(!:S..1.!..._.|
RX1 | 0F 48 9D 8B 18 1B 49 28 3D AD E2 29 6A 59 7E 54 EE 9D 09 1E 09 88 DE 84 BD F8 7E 49 60 0B D5 6C  |.H....I(=..)jY~T
        ..........~I`..l|

/ # hexdump /sys/class/spi_slave/spi1/spi1.0/sunxi-slave-test
0000000 0fbb 936d 67e1 31a4 dc0c aa4d 233f 8d32
0000010 8b9c 2128 533a 1bb0 9f31 e821 dea2 c35f
0000020 ffff ffff ffff ffff ffff ffff ffff ffff
*
0001000
```

- DMA 模式半双工收发测试

```
/ # spislave-test -D /dev/spidev0.0 -R -s 10000000 -a 0 -S 2048
spi mode: 0x80
max speed: 10000000 Hz (10000 kHz)
op addr : 0
op size : 2048

/ # hexdump  /sys/class/spi_slave/spi1/spi1.0/sunxi-slave-test
0000000 1f89 c9a0 398f 433d a5da 7fd7 8c0b 1757
0000010 1de3 6033 49d7 381a a556 aba4 40fb 8443
0000020 e360 ef4e 8b1c f633 0a30 3b75 cc96 7a52
0000030 85e9 c1da f4ce 24f9 9d9a 95d0 13dd 3d1a
......
00007d0 e04c cd0d a110 bbd4 3426 2236 a904 debc
00007e0 4c0b d35e 1d50 2a50 4377 ce0f 1254 a093
00007f0 a0f2 026d 4141 67bd f376 7a8a 469d a858
0000800 ffff ffff ffff ffff ffff ffff ffff ffff
*
0001000
```

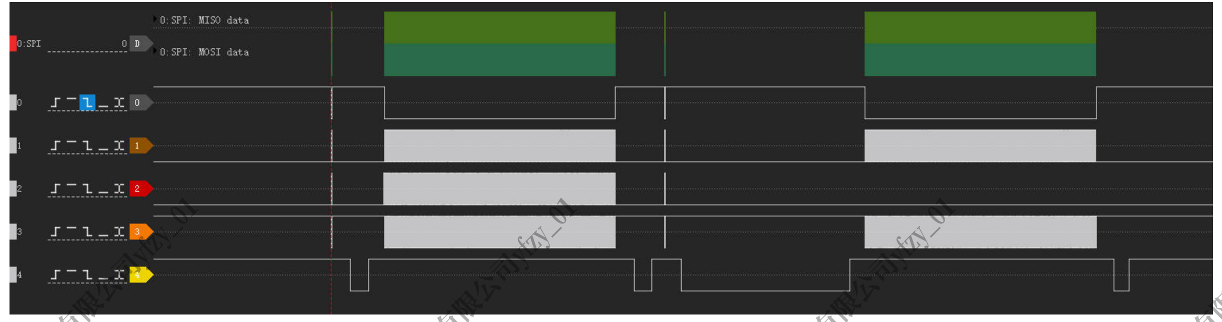


图 5-2: Slave DMA 2048B

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- DMA 模式全双工收发测试

```
/ # spislave-test -D /dev/spidev0.0 -s 10000000 -a 0 -S 128 -I 1 -d 5000 -F -v
spi mode: 0x0
max speed: 10000000 Hz (10000 kHz)
op addr : 0
op size : 128
package head : { 0x03 0x01 0x00 0x00 0x00 0x80 }
package head : { 0x03 0x01 0x00 0x00 0x00 0x80 }
TX0 | 10 F5 42 A7 51 E5 9E AB 69 D8 53 D9 5A D4 69 17 76 4B 4B 8B 11 0C F0 EB 4F 28 36 CC 0C ED 76 EF |..B.Q...i.S.Z.i.vKK
    .....O(6...v.|
TX0 | 12 A4 2C F9 A0 EC BC E9 9C AF 12 63 1C 94 56 7E BD C5 52 26 C3 82 EA 8B 7C 59 B6 AC 0D 65 C3 CB |..,........c..V~..R
    &....|Y...e..|
TX0 | 06 69 4E 4A A2 91 37 13 57 41 28 DF 2E 99 28 72 F5 86 6B 55 9C 99 03 86 18 67 B4 77 8B D4 75 E1 |.iNJ..7.WA(...(r..kU
    .....g.w..u.|
TX0 | 21 AF 37 99 EB A6 DF 48 F3 39 F5 8A C0 33 A4 A1 7B 52 54 85 1E 2D 57 F6 24 F2 DF 9A DA D5 E2 43 |!.7....H.9...3..{RT..-
    W.$......C|
RX0 | BB 0F 6D 93 E1 67 A4 31 0C DC 4D AA 3F 23 32 8D 9C 8B 28 21 3A 53 B0 1B 31 9F 21 E8 A2 DE 5F C3 |..m..g.1..M
    .?#2...(!:S..1.!..._.|
RX0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................................|
RX0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................................|
RX0 | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................................|
TX1 | A1 D1 3E D3 1F 70 7F 3C 98 CF 72 29 0B 5C D5 79 0D B8 F2 9C 70 AF B7 59 C4 A3 01 91 A1 75 05 EC |..>..p.<..r).\.y....p..
    Y.....u..|
TX1 | 3A 5D 2F 3D 05 90 D8 0C 11 D2 7B 1E 51 92 83 68 22 E9 BE F2 12 7F 6B 86 3A 8E BC D2 40 63 E3 80 |:]/=......{.Q..h".....k
    .:...@c..|
TX1 | 90 58 8F 11 CC 8C 1C B3 6C 0D ED 36 44 F1 65 2E DB 44 D0 0D E6 D3 AC 5A C7 41 46 E2 10 B3 46 6B |.X......l..6D.e..D.....
    Z.AF...Fk|
TX1 | 67 8C 04 D3 33 EE 2C F8 AF A7 B0 80 5E 36 14 C5 50 B2 EB 3F 2B FE 75 4A 2F 1A 7C 9F CE 7E 76 5E |g...3.,.....^6..P..?+.
    uJ/.|..~v^|
RX1 | 10 F5 42 A7 51 E5 9E AB 69 D8 53 D9 5A D4 69 17 76 4B 4B 8B 11 0C F0 EB 4F 28 36 CC 0C ED 76 EF |..B.Q...i.S.Z.i.vKK
    .....O(6...v.|
RX1 | 12 A4 2C F9 A0 EC BC E9 9C AF 12 63 1C 94 56 7E BD C5 52 26 C3 82 EA 8B 7C 59 B6 AC 0D 65 C3 CB |..,........c..V~..R
    &....|Y...e..|
RX1 | 06 69 4E 4A A2 91 37 13 57 41 28 DF 2E 99 28 72 F5 86 6B 55 9C 99 03 86 18 67 B4 77 8B D4 75 E1 |.iNJ..7.WA(...(r..kU
    .....g.w..u.|
RX1 | 21 AF 37 99 EB A6 DF 48 F3 39 F5 8A C0 33 A4 A1 7B 52 54 85 1E 2D 57 F6 24 F2 DF 9A DA D5 E2 43 |!.7....H.9...3..{RT..-
    W.$......C|

/ # hexdump /sys/class/spi_slave/spi1/spi1.0/sunxi-slave-test
0000000 d1a1 d33e 701f 3c7f cf98 2972 5c0b 79d5
0000010 b80d 9cf2 af70 59b7 a3c4 9101 75a1 ec05
0000020 5d3a 3d2f 9005 0cd8 d211 1e7b 9251 6883
0000030 e922 f2be 7f12 866b 8e3a d2bc 6340 80e3
0000040 5890 118f 8ccc b31c 0d6c 36ed f144 2e65
0000050 44db 0dd0 d3e6 5aac 41c7 e246 b310 6b46
0000060 8c67 d304 ee33 f82c a7af 80b0 365e c514
0000070 b250 3feb fe2b 4a75 1a2f 9f7c 7ece 5e76
0000080 ffff ffff ffff ffff ffff ffff ffff ffff
*
0001000
```

5.2.3 Slave 模式客户定制方法

AW 提供的 Slave 驱动只是一个使用示例，其功能是模拟了一个内存设备，可以被 Master 进行访问，进行数据传输，验证方法即是 Master 对 Slave 设备进行写数据，再回读，数据一致即测试成

功。客户可以参考该驱动进行定制修改。

5.3 Flash 模式功能开发

5.3.1 Flash 模式开发流程

5.3.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_SPI_NG
```

其余配置参考《Linux_SPINAND_开发指南》或《Linux_SPINOR_开发指南》

5.3.1.2 设备树配置

SPI NOR Flash 设备树配置参考如下：

```
&spi0 {
    pinctrl-0 = <&spi0_pins_default &spi0_pins_cs>;
    pinctrl-1 = <&spi0_pins_sleep>;
    pinctrl-names = "default", "sleep";
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_NOR>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_SOFT>;
    status = "okay";

    spi_board0 {
        device_type = "spi_board0";
        compatible = "spi-nor";
        spi-max-frequency = <100000000>;
        m25p,fast-read = <1>;
        /*individual_lock;*/
        reg = <0x0>;
        spi-rx-bus-width = <4>;
        spi-tx-bus-width = <4>;
        status = "okay";
    };
};
```

SPI NAND Flash 设备树配置参考如下：

```
&spi0 {
    pinctrl-0 = <&spi0_pins_default &spi0_pins_cs>;
    pinctrl-1 = <&spi0_pins_sleep>;
    pinctrl-names = "default", "sleep";
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_NAND>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_SOFT>;
    status = "okay";

    spi-nand@0 {
```

```
    compatible = "spi-nand";
    spi-max-frequency = <100000000>;
    reg = <0x0>;
    spi-rx-bus-width = <4>;
    spi-tx-bus-width = <4>;
    status = "okay";
  };
};
```

5.3.1.3 Flash 模式注意事项

- 当sunxi,spi-bus-mode配置为SUNXI_SPI_BUS_NOR/SUNXI_SPI_BUS_NAND时，sunxi,spi-cs-mode会强制使用SUNXI_SPI_CS_SOFT模式
- spi-max-frequency是该 SPI 外设的最大输出时钟频率，配置时不可超过父节点中配置的 SPI 控制器最大时钟频率clock-frequency
- spi-rx-bus-width/spi-tx-bus-width是 spi 的读写线宽，根据 Flash 特性可配置为 1/2/4

5.3.1.4 Flash 模式编程示例

参考《Linux_SPINAND_ 开发指南》或《Linux_SPINOR_ 开发指南》

5.3.2 Flash 模式测试方法

参考 Flash 引脚封装图，将 SPI 控制器引脚与 Flash 对应的引脚连接。参考对应开发指南适配完成系统启动后，使用 dd 命令测试：

```
#time dd if=/dev/mtd3 of=/dev/null bs=4096 count=10240
命令分析：
  time:      命令常用于测量一个命令的运行时间。
  dd:        用指定大小的块拷贝一个文件，并在拷贝的同时进行指定的转换。
  if=文件名：   输入文件名，缺省为标准输入。即指定源文件。< if=input file >
  /dev/mtd3:   物理分区，根据你实际使用情况选择，对它的读取会产生IO
  of=文件名：   输出文件名，缺省为标准输出。即指定目的文件。< of=output file >
  /dev/null:   是一个伪设备，相当于回收站，of到该设备不会产生IO（也可以指定存在的文件，但会产生IO）
  bs=bytes:    同时设置读入/输出的块大小为bytes个字节。
  count=blocks：仅拷贝blocks个块，块大小等于ibs指定的字节数。
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.4 DBI 模式功能开发

5.4.1 DBI 模式开发流程

5.4.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_SPI_NG
CONFIG_AW_SPI_NG_DBI
```

其余配置参考《Linux_LCD_ 开发指南》

5.4.1.2 设备树配置

```
&spi1 {
    ......
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_DBI>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
    status = "okay";
};

&lcd_fb0 {
    lcd_used = <1>;
    lcd_driver_name = "kld2844b";
    lcd_if = <1>;
    lcd_dbi_if = <4>;
    lcd_data_speed = <60>;
    lcd_spi_bus_num = <1>;
    lcd_x = <240>;
    lcd_y = <320>;
    lcd_pixel_fmt = <0>;
    lcd_dbi_fmt = <3>;
    lcd_rgb_order = <0>;
    lcd_width = <60>;
    lcd_height = <95>;
    lcd_pwm_used = <1>;
    lcd_pwm_ch = <6>;
    lcd_pwm_freq = <5000>;
    lcd_pwm_pol = <1>;
    lcd_frm = <1>;
    lcd_gamma_en = <1>;
    fb_buffer_num = <2>;
    lcd_backlight = <100>;
    lcd_fps = <60>;
    lcd_dbi_te = <1>;
    lcd_dbi_clk_mode = <0>;
    lcd_gpio_0 = <&pio PD 4 GPIO_ACTIVE_LOW>;
    /*lcd_spi_dc_pin = <&pio PD 5 GPIO_ACTIVE_HIGH>;*/
    status = "okay";
};
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.4.1.3 DBI 模式注意事项

- DBI 模式下，SPI 的其他功能（如 Master/Slave/Flash 等）将无法使用

5.4.1.4 DBI 模式编程示例

参考《Linux_LCD_ 开发指南》

5.4.2 DBI 模式测试流程

将 SPI 与屏幕对应的引脚连接，按照开发指南适配启动后，屏幕能点亮并显示内容

5.5 BIT 模式功能开发

5.5.1 BIT 模式开发流程

5.5.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_SPI_NG
CONFIG_AW_SPI_NG_BIT
CONFIG_SPI_SPIDEV
```

5.5.1.2 设备树配置

```
&spi0 {
    ......
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_BIT>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
    status = "okay";

    spidev@0 {
        compatible = "rohm,dh2228fv";
        reg = <0x0>;
        spi-3wire; # 3线bit模式
        spi-max-frequency = <10000000>;
        spi-rx-bus-width = <1>;
        spi-tx-bus-width = <1>;
        status = "okay";
    };
};
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.5.1.3 BIT 模式注意事项

- 用户空间测试依赖于/dev/spidevX.Y节点
- BIT 模式仅支持 CPU 传输方式，不支持 DMA 和全双工模式

5.5.1.4 BIT 模式编程示例

BIT 模式大部分的流程跟 Master 非常类似，可直接参考Master 模式编程示例，其中有差异的部分是会通过 bpw 代替 len 来决定一次传输的长度。

- 打开设备并配置相关参数

```
int bits = 12;

ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits); // set bit per word
```

- 收发数据

```
struct spi_ioc_transfer tr = {
    ......
    .bits_per_word = bits,
};

ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
```

5.5.2 BIT 模式测试流程

将 SPI 信号线接入示波器或逻辑分析仪查看波形是否正确

```
# /bsptest/bin/spidev-test -D /dev/spidev1.0 -s 10000000 -S 4 -I 1 -3 -b 12 -v
spi mode: 0x10
bits per word: 12
max speed: 10000000 Hz (10000 kHz)
TX | 7C FD 89 4C __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  ||..L|
RX | 00 00 00 00 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  |....|
rx/tx buffer is not same, data error!!!
total size   : 4 B
total time   : 46.17 us
averange rate: 0.08 MB/s
averange time: 46.17 us
```

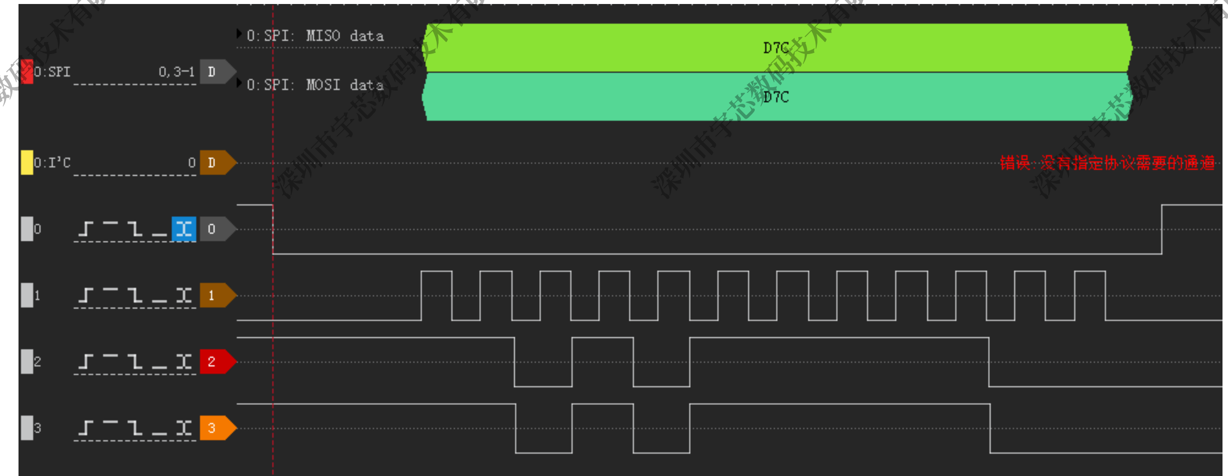


图 5-3: SPI BIT 模式波形图

5.6 CAMERA 模式功能开发

5.6.1 CAMERA 模式开发流程

5.6.1.1 软件配置

打开 kernel menuconfig 中如下配置项

```
CONFIG_AW_TWI
CONFIG_AW_SPI_NG
CONFIG_AW_SPI_NG_CAMERA
CONFIG_AW_SPI_CAMERATEST
```

5.6.1.2 设备树配置

这里以 4-Bit framehead 模式为例，配置如下：

```
&spi3 {
    clock-frequency = <24000000>;
    pinctrl-0 = <&spi3_pins_default &spi3_pins_cs>;
    pinctrl-1 = <&spi3_pins_sleep>;
    pinctrl-names = "default", "sleep";
    sunxi,spi-bus-mode = <SUNXI_SPI_BUS_CAMERA>;
    sunxi,spi-cs-mode = <SUNXI_SPI_CS_AUTO>;
    status = "disabled";

    bf20a6_4bit: slave@3 {
        compatible = "sunxi,spicamera";
        reg = <0x0>;
        spi-cs-high;
        spi-cpol;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
        spi-rx-bus-width = <4>;
        pixel-horizontal = <640>;
        pixel-vertical = <480>;
        pixel-bytes = <2>;
        camera-type = <2>; /* 0:vsync 1:vsyncinput 2:framehead 3:idlewait */
        pwdn-gpio = <&pio PK 14 GPIO_ACTIVE_LOW>;
        avdd-supply = <&reg_aldo1>;
        avdd-microvolt = <2800000>;
        start-of-frame = /bits/ 8 <0xff 0x00 0x00 0xab>;
        start-of-line = /bits/ 8 <0xff 0x00 0x00 0x80>;
        end-of-line = /bits/ 8 <0xff 0x00 0x00 0x9d>;
        end-of-frame = /bits/ 8 <0xff 0x00 0x00 0xb6>;
        status = "disabled";
    };
};
```

5.6.1.3 CAMERA 模式注意事项

- CAMERA 模式本质是 Slave 功能，但不可与上述 Slave 功能混淆

5.6.1.4 CAMERA 模式编程示例

内核空间驱动编程示例参考bsp/drivers/spi-ng/spi-sunxi-camera-test.c

5.6.2 CAMERA 模式测试方法

以 bf20a6 模组为例，将 SPI 与 bf20a6 按照引脚连接

```
/ # insmod /lib/modules/*/spi-sunxi-camera-test.ko
[   50.265167] sunxi-spi-camera spi2.0: get pixel h_640 v_480 bytes_2
[   50.290303] sun55iw5-pinctrl 2000000.pinctrl: supply vcc-ph not found, using dummy regulator
[   50.320253] sunxi-spi-camera spi2.0: camera get pwdn gpio 224
[   50.338754] sunxi-spi-camera spi2.0: camera get type 2
[   50.345755] sunxi-spi-camera spi2.0: get flag start-of-frame: ff 00 00 ab
[   50.368948] sunxi-spi-camera spi2.0: get flag start-of-line: ff 00 00 80
[   50.388991] sunxi-spi-camera spi2.0: get flag end-of-line: ff 00 00 9d
[   50.397705] sunxi-spi-camera spi2.0: get flag end-of-frame: ff 00 00 b6
[   50.469015] sunxi-spi-camera spi2.0: raw buffer len 618248
[   50.496755] sunxi-spi-camera spi2.0: img buffer len 614400
/ # echo on > /sys/class/spi_slave/spi2/spi2.0/pwdn
[   50.903756] sunxi-spi-camera spi2.0: spi camera power on
# ./BF20A6_YUV_6M_AW_4BIT_ENCODE.sh 0 0x6e
/ # echo > /sys/class/spi_slave/spi2/spi2.0/capture
[  111.890857] sunxi-spi-camera spi2.0: camera capture start
[  113.021656] sunxi-spi-camera spi2.0: buffer decode size 614400 start
[  113.044927] sunxi-spi-camera spi2.0: buffer decode size 614400 done
[  113.053530] sunxi-spi-camera spi2.0: camera capture done
/ # dd if=/sys/class/spi_slave/spi2/spi2.0/sunxi-camera-test-img of=/capture.bin
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

📖 说明

配置脚本需根据不同模组进行适配，不随 SDK 发布

将 capture.bin 文件用 adb 拉到电脑上，用 RawViewer.exe 打开

0#0 YUYV 640x480 scale=1:1 -spi_camera.bin

x=271 y= 2 Y=254 U=128 V=128 w=16 h= 0 b8=1 b4= 5

图 5-4: BF20A6 4-bit 模组实际效果图

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6 调试方法

6.1 调试工具

见上文章节Master 模式功能开发及Slave 模式功能开发的描述和使用方法。

6.2 驱动调试节点

6.2.1 debug_mask

用于动态打开调试打印以跟踪每次数据传输前后的寄存器状态变化即收发的数据内容

```
enum {
    SUNXI_SPI_DEBUG_DUMP_REG   = BIT(0),
    SUNXI_SPI_DEBUG_DUMP_DATA  = BIT(1),
};
```

当前有 spi/dbi 两个节点控制，向其写入对应的 bit 掩码，即可打开对应打印信息

```
echo 3 > /sys/module/spi_sunxi_ng/parameters/spi_debug_mask
echo 3 > /sys/module/spi_sunxi_ng/parameters/dbi_debug_mask
```

6.2.2 version

用于查看当前软件驱动的版本号

```
cat /sys/module/spi_sunxi_ng/version
```

6.3 控制器调式节点

6.3.1 info

打印 SPI 控制器的硬件资源信息。

```
/ # cat /sys/devices/platform/soc@3000000/4025000.spi/info
IP Version = V1.3
pdev->id   = 0
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
pdev->name = 4025000.spi
pdev->num_resources = 2
pdev->resource = [mem 0x04025000-0x04025fff flags 0x200]
ctlr->bus_num = 0
ctlr->num_chipselect = 1
ctlr->dma_alignment  = 128
ctlr->mode_bits = 3855
ctlr->dma_tx = [dma0chan0]
ctlr->dma_rx = [dma0chan1]
ctlr->min_speed_hz = 187500
ctlr->max_speed_hz = 100000000
sspi->base_addr = 0xffffffc0093e1000
sspi->irq = 118
sspi->bus_mode = 16
sspi->bus_num  = 0
sspi->bus_freq = 100000000
sspi->cs_mode  = 1
sspi->cs_num   = 1
sspi->pre_speed_hz = 0
sspi->bus_sample_mode  = 0 [auto]
sspi->spi_sample_mode  = 0x0
sspi->spi_sample_delay = 0x0
```

6.3.2 status

打印 SPI 控制器运行状态信息。

```
/ # cat /sys/devices/platform/soc@3000000/4025000.spi/status
ctlr->flags = 0x20
ctlr->busy = 0
ctlr->running = 1
sspi->mode_type = 0
sspi->result = 0
```

6.3.3 fifo

打印 SPI 控制器的 fifo 状态（包括硬件 fifo 大小，触发水线及数据的实时数量）。

```
/ # cat /sys/devices/platform/soc@3000000/4025000.spi/fifo
tx fifo size      : 64
rx fifo size      : 128
tx trigger level  : 32
rx trigger level  : 64

tx shift buf count: 0
tx buffer count   : 0
tx fifo   count   : 0
rx buffer count   : 0
rx fifo   count   : 0
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.3.4 dump

打印 SPI 控制器的寄存器信息

```
/ # cat /sys/devices/platform/soc@3000000/4025000.spi/dump
[ 7781.698377] 0x04025000: 00010003 00000087 000000c7 00000000
[ 7781.704639] 0x04025010: 00000000 00000032 00200040 00000000
[ 7781.710888] 0x04025020: 00000000 00000002 00002000 00000000
[ 7781.717138] 0x04025030: 00000000 00000000 00000000 00000000
[ 7781.723385] 0x04025040: 000000a0 00000000 00000000 00000000
[ 7781.729628] 0x04025400: 00000000 00000000 00000000 00000000
```

6.3.5 calibrate

针对 Flash 模式，用于扫描并校准当前速率下的可用延时窗口

```
/ # echo > /sys/devices/platform/soc@3000000/4025000.spi/calibrate
[  161.393648] mode(0): ----------------------------------------------------------------
[  161.404807] mode(1): ----------------------------------------------------------------
[  161.416357] mode(2): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[  161.427200] mode(3): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[  161.438869] mode(4): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[  161.450083] mode(5): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[  161.461251] mode(6): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[  161.470175] sunxi:spi-4025000.spi:[INFO]: calibrate suitable sample mode:0 delay:31
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7 常见问题

7.1 如何区分当前使用是新/老驱动及驱动版本

7.1.1 通过 dtsi compatible 属性

老驱动一般名字是：allwinner,sunXXi-spi（如 allwinner,sun8i-spi、allwinner,sun50i-spi）

而新驱动名字是：allwinner,sunxi-spi-vX.Y（如 allwinner,sunxi-spi-v1.3）

7.1.2 通过内核配置 deconfig

老驱动配置为：CONFIG_AW_SPI

新驱动配置为：CONFIG_AW_SPI_NG

7.1.3 通过启动 log 打印

老驱动使用 pr_xxx 接口进行打印，启动 log 如下：

```
spi spi0: supply spi not found, using dummy regulator
sunxi_spi_resource_get()2482 - [spi0] SPI MASTER MODE
sunxi_spi_resource_get()2524 - Failed to get sample mode
sunxi_spi_resource_get()2529 - Failed to get sample delay
sunxi_spi_resource_get()2533 - sample_mode:-1431633921 sample_delay:-1431633921
sunxi_spi_clk_init()2577 - [spi0] mclk 100000000
sunxi_spi_probe()3028 - [spi0]: driver probe succeed, base f005c000, irq 43
```

新驱动使用 dev_xxx 接口进行打印，启动 log 如下：

```
sunxi-spi-ng 4025000.spi: bus is in flash mode, cs must use software contorl
sunxi-spi-ng 4025000.spi: bus sample mode old
sunxi-spi-ng 4025000.spi: supply spi not found, using dummy regulator
sunxi-spi-ng 4025000.spi: bus num_0 mode_4 freq_100000000
sunxi-spi-ng 4025000.spi: cs num_1 mode_1
sunxi-spi-ng 4025000.spi: spi fifosize tx_64 rx_128
sunxi-spi-ng 4025000.spi: Request DMA channel dma0chan0(tx) and dma0chan1(rx)
sunxi-spi-ng 4025000.spi: init mclk freq 100000000
sunxi-spi-ng 4025000.spi: probe succeed (Version 2.3.4)
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7.2 如何强制使用 CPU 传输

- 关闭 DMA 相关配置宏 CONFIG_DMADEVICES/CONFIG_AW_DMA/CONFIG_DMA_ENGINE
- 注释 dtsi 中 SPI 控制器的 dma-names = “tx”，“rx” 配置信息
- 将 SPI 驱动中的 sspi->use_dma 强制修改为 false

以上三种办法任选其一即可。

7.3 常见排查方法

- 确保 SPI 驱动正常注册
- 确保板级设备树 SPI 和 PINMUX 配置无误
- 如果 SPI DMA 传输异常，强制使用 CPU 传输看看有没有问题
- 其他问题请联系 FAE 或者提 Aservice 单

7.4 FAQ

- [FAQ1285] SPI 需要支持 SPI_NO_CS 模式
- [FAQ1527] T3_linux5.10 spi 设置低波特率通信失败
- [FAQ1560] Linux SPI 驱动中如何使用自定义 GPIO 口用作 CS 功能
- [FAQ1567] T3 linux5.10 uboot spi2 支持
- [FAQ1773] SPI 控制器发出的 CLK 信号不连续
- [FAQ2234] R_SPI 使用 DMA0(SYS_DMA) 通信超时/失败

版权所有 © 珠海全志科技股份有限公司。保留一切权利

著作权声明

版权所有 ©2025 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

ALLWINNER、ALLWINNER 全志科技、全志科技、ALLWINNER®（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。

深圳市宇芯数码技术有限公司yfzy_01