



StarFive
赛昉科技

昉·惊鸿7110 SPI开发手册

昉·星光 2

版本：1.0

日期：2022/11/10

Doc ID: JH7110-DGCH-005

法律声明

阅读本文件前的重要法律告知。

版权注释

版权 ©上海赛昉科技有限公司，2023。版权所有。

本文档中的说明均基于“视为正确”提供，可能包含部分错误。内容可能因产品开发而定期更新或修订。上海赛昉科技有限公司（以下简称“赛昉科技”）保留对本协议中的任何内容进行更改的权利，恕不另行通知。

赛昉科技明确否认任何形式的担保、解释和条件，无论是明示的还是默示的，包括但不限于适销性、特定用途适用性和非侵权的担保或条件。

赛昉科技无需承担因应用或使用任何产品或电路而产生的任何责任，并明确表示无需承担任何及所有连带责任，包括但不限于间接、偶然、特殊、惩戒性或由此造成的损害。

本文件中的所有材料受版权保护，为赛昉科技所有。不得以任何方式修改、编辑或断章取义本文件中的说明，本文件或其任何部分仅限用于内部使用或教育培训。

联系我们：

地址：浦东新区盛夏路61弄张润大厦2号楼502，上海市，201203，中国

网站：<http://www.starfivetech.com>

邮箱：

- sales@starfivetech.com（销售）
- support@starfivetech.com（支持）

前言

关于本指南和技术支持信息

关于本手册

本手册主要为SDK开发和移植提供赛昉科技新一代SoC平台——昉·惊鸿7110的ISP编程基础和调试操作。

受众

本手册主要服务与与SPI相关驱动程序的开发人员。如果您正在开发其他模块，请与您的销售或支持顾问联系，获取我们在昉·惊鸿7110上的完整文档。






修订历史

表 0-1 修订历史

Version	发布说明	修订
1.0	2022/11/10	首次发布。

注释和注意事项

本指南中可能会出现以下注释和注意事项：

-  **提示：**
建议如何在某个主题或步骤中应用信息。
-  **注：**
解释某个特例或阐释一个重要的点。
-  **重要：**
指出与某个主题或步骤有关的重要信息。
-  **警告：**
表明某个操作或步骤可能会导致数据丢失、安全问题或性能问题。
-  **警告：**
表明某个操作或步骤可能导致物理伤害或硬件损坏。

目录

表格清单.....	5
插图清单.....	6
法律声明.....	ii
前言.....	iii
1. 简介.....	7
1.1. 功能介绍.....	7
1.2. 设备树概述.....	7
1.3. 设备树代码.....	8
1.4. 源代码结构.....	9
2. 配置.....	10
2.1. 内核菜单配置.....	10
2.2. 设备树配置.....	12
2.3. 板级配置.....	13
3. 驱动程序框架.....	15
3.1. 框图.....	15
4. 接口介绍.....	17
4.1. 接口定义.....	17
4.2. spi_register_driver.....	17
4.3. spi_message_init.....	17
5. 示例用例.....	19
6. 常见问题集.....	23

表格清单

表 0-1 修订历史..... iii



插图清单

图 1-1 设备树 workflow.....	8
图 2-1 Device Drivers.....	10
图 2-2 SPI support.....	11
图 2-3 SSP controller.....	12
图 3-1 框图.....	15
图 5-1 User mode SPI device driver support.....	21
图 5-2 SPI测试示例.....	22
图 6-1 DMA故障.....	23
图 6-2 DMA故障解决方案.....	23



StarFive
赛昉科技

1. 简介

串行外设接口（SPI）是微控制器和外设IC（如传感器、ADC、DAC、移位寄存器、SRAM等）之间应用的最广泛的接口之一。

1.1. 功能介绍

昉·惊鸿7110 SOC平台在SPI上具有以下特点和规格：

- 支持7个SPI接口。
- 支持串口主模式和串口从模式，使用软件配置可在这两种模式之间进行切换。
- 为输入和输出提供单独的数据位。
- 为TX和RX通道提供大小最高达8位×16位的可配置FIFO。
- 支持对TX和RX FIFO的DMA访问。

1.2. 设备树概述

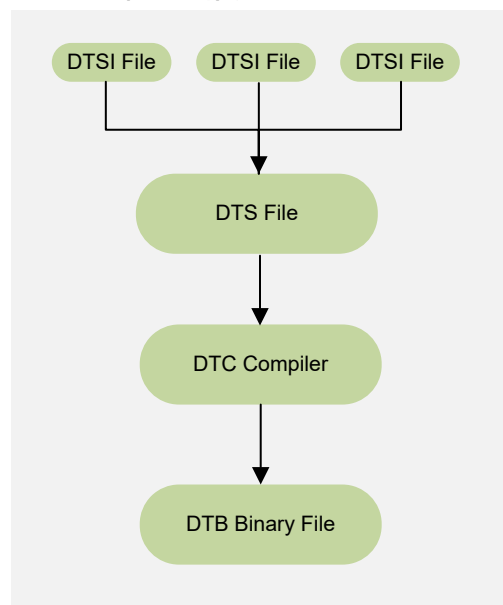
自Linux 3.x以来，系统就引入了设备树作为数据结构和语言来描述硬件配置。设备树是硬件设置的系统可读描述，这样操作系统不必硬编码机器的详细信息。

一个设备树主要有以下呈现形式。

- 设备树编译器（DTC）：用于将设备树编译为系统可读的二进制文件的工具。
- 设备树源码（DTS）：人类可读的设备树描述文件。您可以在此文件中找到目标参数并修改硬件配置。
- 设备树源码信息（DTSI）：可包括在设备树描述中的人类可读的头文件。您可以在此文件中找到目标参数并修改硬件配置。
- 设备树块（DTB）：系统可读设备树二进制blob文件，在系统中烧录以供执行。

下图显示了上述形式的关系（工作流）。

图 1-1 设备树工作流程



1.3. 设备树代码

总体结构

昉·惊鸿7110的设备树代码如下：

```

linux
├── arch
│   ├── riscv
│   │   ├── boot
│   │   │   ├── dts
│   │   │   │   └── starfive
│   │   │   │       ├── codecs
│   │   │   │       │   ├── sf_pdm.dtsi
│   │   │   │       │   ├── sf_pwm dac.dtsi
│   │   │   │       │   ├── sf_spdif.dtsi
│   │   │   │       │   ├── sf_tdm.dtsi
│   │   │   │       │   └── sf_wm8960.dtsi
│   │   │   │       ├── evb-overlay
│   │   │   │       │   ├── jh7110-evb-overlay-can.dts
│   │   │   │       │   ├── jh7110-evb-overlay-rgb2hdmi.dts
│   │   │   │       │   ├── jh7110-evb-overlay-sdio.dts
│   │   │   │       │   ├── jh7110-evb-overlay-spi.dts
│   │   │   │       │   ├── jh7110-evb-overlay-uart4-emmc.dts
│   │   │   │       │   ├── jh7110-evb-overlay-uart5-pwm.dts
│   │   │   │       │   └── Makefile
│   │   │   │       ├── jh7110-clk.dtsi
│   │   │   │       ├── jh7110-common.dtsi
│   │   │   │       └── jh7110.dtsi
  
```


2. 配置

2.1. 内核菜单配置

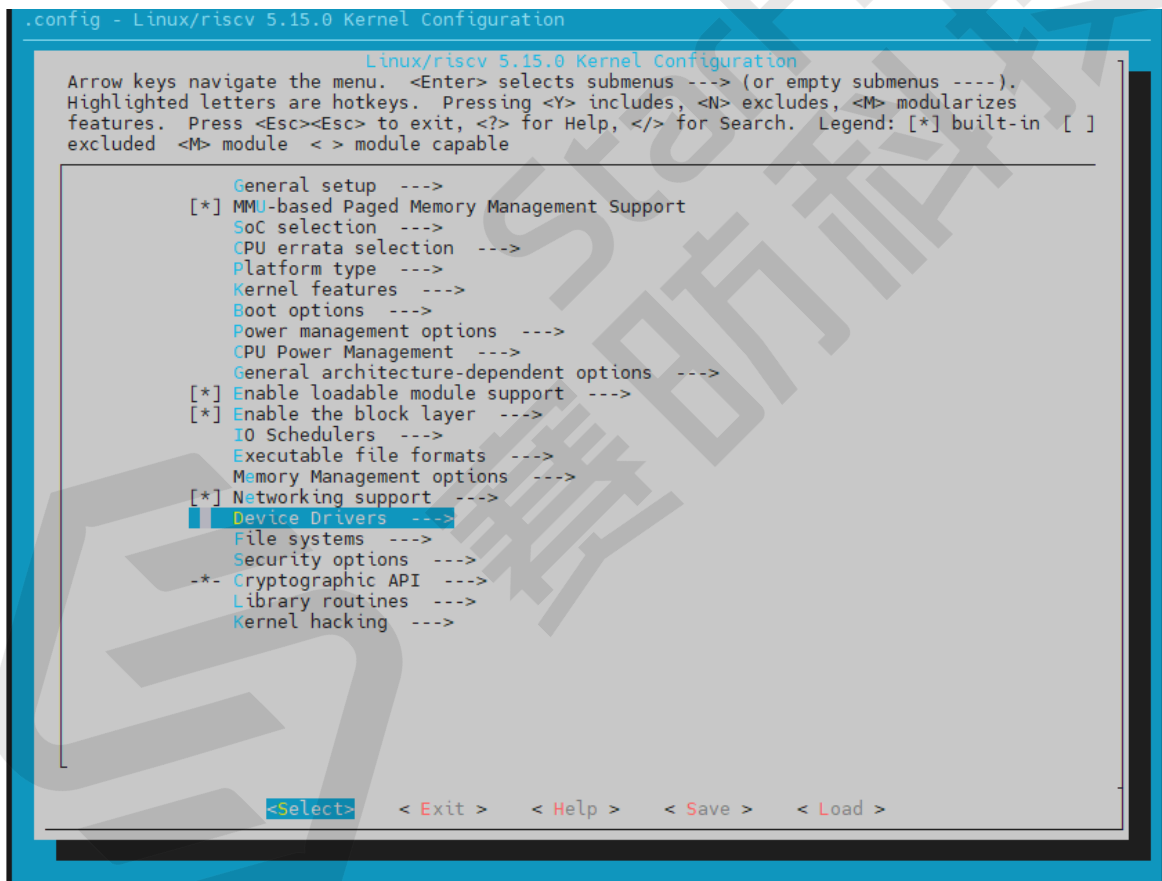
执行以下步骤，创建SPI内核配置：

1. 在freelight-u-sdk的根目录下，输入以下命令以进入内核菜单配置GUI。

```
make linux-menuconfig
```

2. 进入Device Drivers菜单。

图 2-1 Device Drivers



3. 选择SPI support菜单。

图 2-2 SPI support

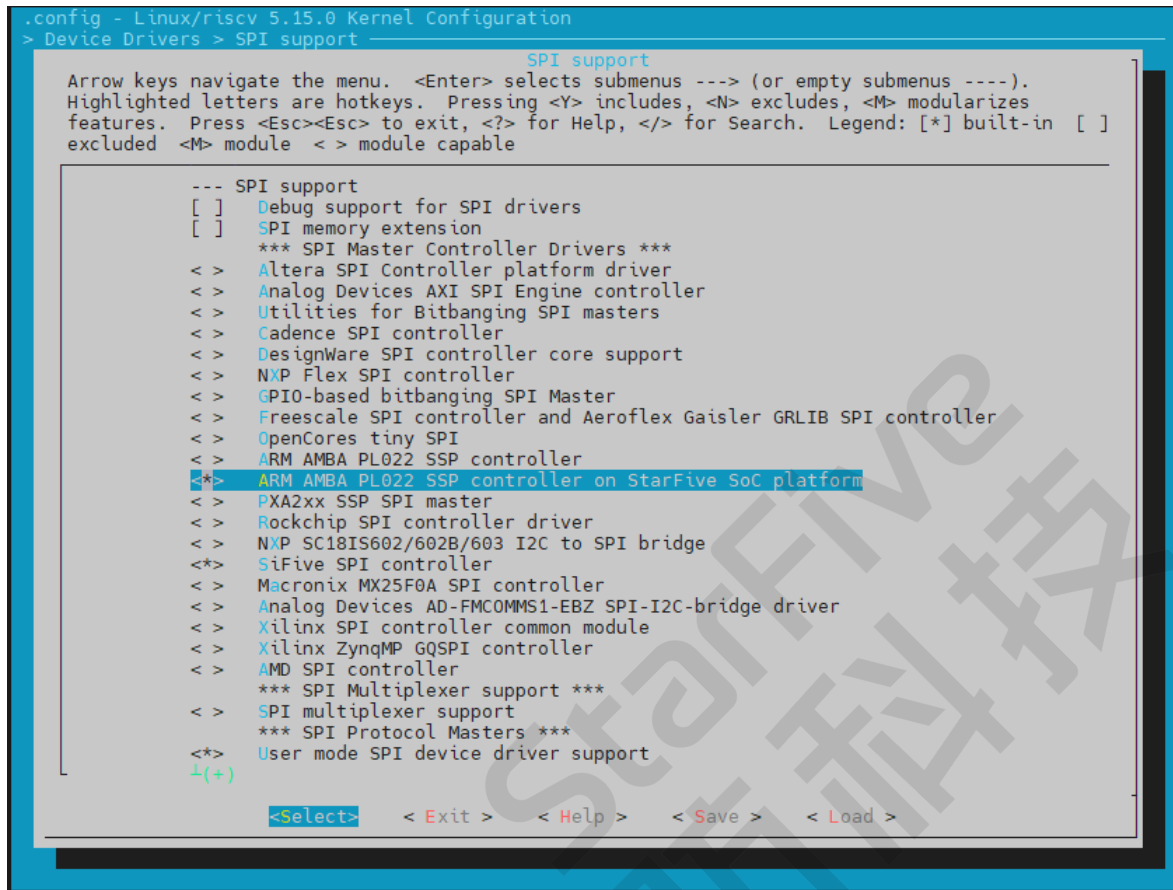
```

.config - Linux/riscv 5.15.0 Kernel Configuration
> 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
^(-)
< > Connector - unified userspace <-> kernelspace linker ----
Firmware Drivers ----
< > GNSS receiver support ----
< > Memory Technology Device (MTD) support ----
-* Device Tree and Open Firmware support ---->
< > Parallel port support ----
[*] Block devices ---->
    NVME Support ---->
    Misc devices ---->
    SCSI device support ---->
<*> Serial ATA and Parallel ATA drivers (libata) ---->
[ ] Multiple devices driver support (RAID and LVM) ----
< > Generic Target Core Mod (TCM) and ConfigFS Infrastructure ----
[ ] Fusion MPT device support ----
IEEE 1394 (FireWire) support ---->
[*] Network device support ---->
    Input device support ---->
    Character devices ---->
    I2C support ---->
< > I3C support ----
[*] SPI support ---->
< > SPMI support ----
< > HSI support ----
< > PPS support ----
    PTP clock support ---->
[*] Pin controllers ---->
    GPIO Support ---->
< > Dallas's 1-wire support ----
↑(+)
```

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

4. 选择SSP controller选项。

图 2-3 SSP controller



5. 保存更改，并退出内核配置对话框。

2.2. 设备树配置

SPI的设备树源代码在以下路径：

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

下面的代码为设置SPI0的示例。

```

spi0: spi@10060000 {
    compatible = "arm,pl022", "arm,primecell";
    reg = <0x0 0x10060000 0x0 0x10000>;
    clocks = <&clkgen JH7110_SPI0_CLK_APB>;
    clock-names = "apb_pclk";
    resets = <&rstgen RSTN_U0_SSP_SPI_APB>;
    reset-names = "rst_apb";
    interrupts = <38>;
    /* shortage of dma channel that not be used */
    /* dmas = <&dma 14 1>, <&dma 15 1>;*/
    /* dma-names = "rx","tx";*/
    arm,primecell-periphid = <0x00041022>;
}

```

```

num-cs = <1>;
#address-cells = <1>;
#size-cells = <0>;
status = "disabled";

```

以下提供了对上述代码块中的参数说明。

- **compatible**: 兼容性信息，用于连接驱动程序和目标设备。
- **reg**: 寄存器基本地址“0x10060000”和范围“0x10000”。
- **clocks**: SPI模块使用到的时钟。
- **clock-names**: 上述时钟的名称。
- **resets**: SPI模块使用到的复位信号。
- **reset-names**: 上述复位信号的名称。
- **interrupts**: 硬件中断ID。
- **primecell-periphid**: SPI设备的外设ID。
- **num-cs**: 片选信号的总数。
- **status**: SPI模块的工作状态。要启用模块，请将此位设置为“okay”；要禁用该模块，请将此位设置为“disabled”。

2.3. 板级配置

板级设备树文件（DTSI文件）存储所有其他板级设备中相同的信息。（例如，common.dtsi、pinctrl.dtsi和evb.dts等文件。）

common.dtsi文件在以下路径：

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
```

文件中，spi0有以下设置。

```

&spi0 {
    pinctrl-names = "default";
    pinctrl-0 = <&ssp0_pins>;
    status = "disabled";
    spi_dev0: spi@0 {
        compatible = "rohm,dh2228fv";
        pl022,com-mode = <1>;
        spi-max-frequency = <10000000>;
        reg = <0>;
        status = "okay";
    };
};

```

以下为上述代码中配置位的描述：

- **spi-max-frequency**：编辑此位以配置SPI的通信时钟频率。
- **status**：编辑此位以定义是否启用此模块。

昉·星光 2 板级配置

`pinctrl.dtsi`文件包含pin控制配置。文件在以下路径：

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
```

以下代码块提供了spi0使用的pin的示例，包括**tx**（收发器）、**rx**（接收器）、**clk**（时钟）和**cs**（片选）信号。

```
ssp0_pins: ssp0-pins {
    ssp0-pins_tx {
        sf,pins = <PAD_GPIO52>;
        sf,pinmux = <PAD_GPIO52_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
        sf,pin-gpio-dout = <GPO_SPI0_SSPTXD>;
        sf,pin-gpio-doen = <OEN_LOW>;
    };

    ssp0-pins_rx {
        sf,pins = <PAD_GPIO53>;
        sf,pinmux = <PAD_GPIO53_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
        sf,pin-gpio-doen = <OEN_HIGH>;
        sf,pin-gpio-din = <GPI_SPI0_SSPRXD>;
    };

    ssp0-pins_clk {
        sf,pins = <PAD_GPIO48>;
        sf,pinmux = <PAD_GPIO48_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
        sf,pin-gpio-dout = <GPO_SPI0_SSPCLKOUT>;
        sf,pin-gpio-doen = <OEN_LOW>;
    };

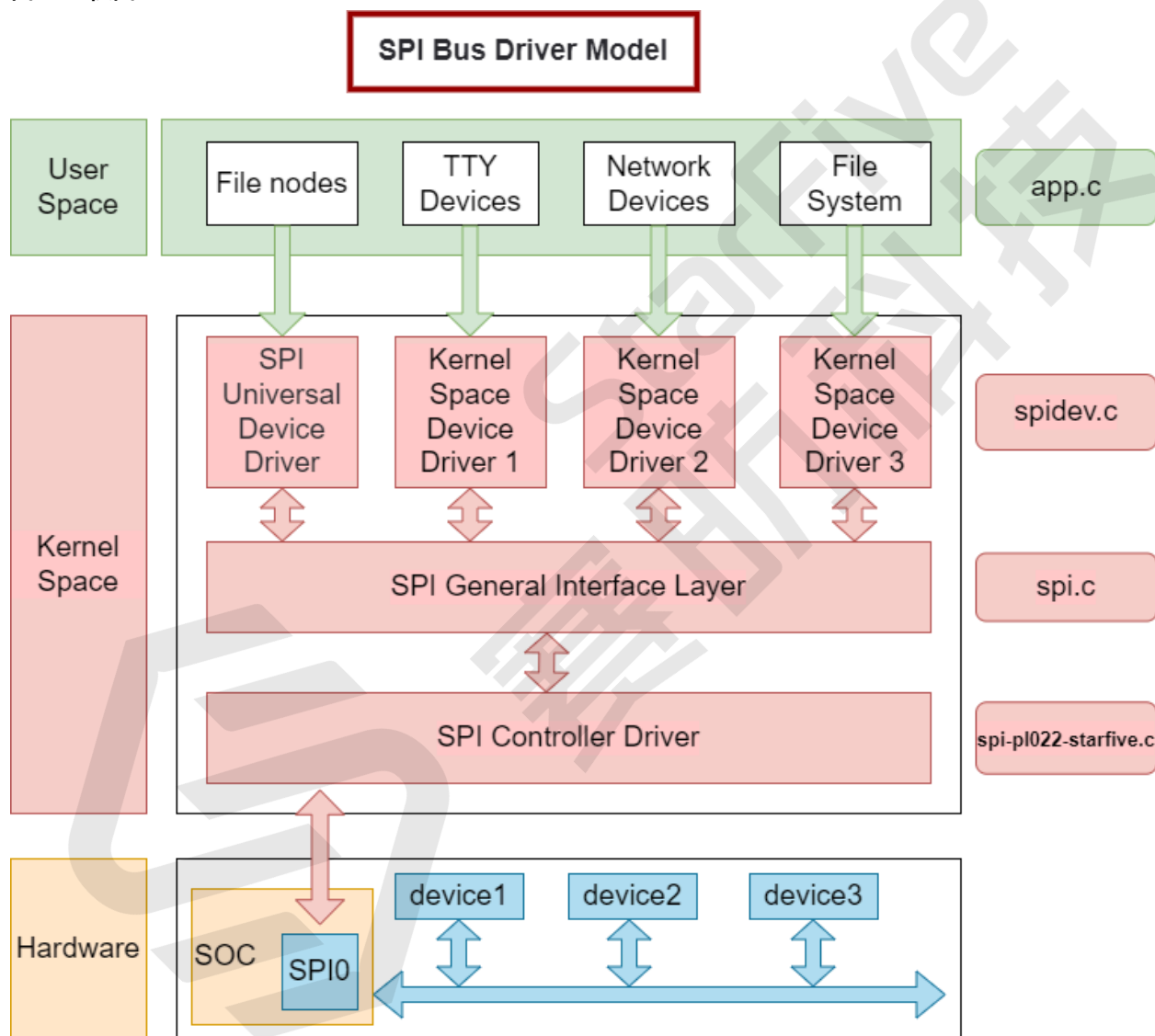
    ssp0-pins_cs {
        sf,pins = <PAD_GPIO49>;
        sf,pinmux = <PAD_GPIO49_FUNC_SEL 0>;
        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
        sf,pin-gpio-dout = <GPO_SPI0_SSPFSSOUT>;
        sf,pin-gpio-doen = <OEN_LOW>;
    };
};
```

3. 驱动程序框架

3.1. 框图

下图显示了SPI驱动程序框架的3个层级。

图 3-1 框图



以下是对上图中每一层的描述。

用户空间

用户空间层包括使用SPI设备的所有应用程序。在这一层中，用户可以根据他们的特定需求定制他们的SPI设备。

内核空间

内核空间层可分为以下三个部分。

- SPI设备驱动程序层：

Linux内核并不提供特定的SPI设备驱动程序，因为SPI上可能连接各种设备。用户必须使用通用的SPI设备驱动程序，该驱动程序只能以同步模式与SPI设备通信。因此，该层只支持一些简单而非需要消耗大量数据的设备。

在这一层中，我们提供了`spidev.c`作为标准的SPI驱动程序，而`spi-nand.c`作为SPI的NAND驱动程序。

- SPI通用接口封装层：

为了简化SPI驱动程序的编程，减少驱动程序的耦合，Linux内核为控制器和协议打包封装了一些通用的驱动程序，形成了SPI通用接口封装层。

在这一层中，我们提供了Linux自带的驱动程序`spi.c`。

- SPI控制器驱动程序层：

这一层是我们关注的重点，它将在文档的后面部分中详细介绍。

在这一层中，我们提供了驱动程序`spi-pl022-star5.c`。

硬件

硬件层是物理设备层。在这一层中，SPI控制器和所连接的SPI设备通过SPI总线与CPU进行通信。

4. 接口介绍

4.1. 接口定义

SPI的接口定义在该文件中：`include/linux/spi/spi.h`，主要的接口有`spi_register_driver`和`spi_message_init`。

宏`module_spi_driver ()`用于快速注册一个SPI设备。

以下代码为一个示例：

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

4.2. spi_register_driver

该接口有以下参数。

- 简介：

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

- 描述：该接口用于注册一个SPI设备的驱动程序。

- 参数：

- `sdrv`：`spi_driver`类型，包括SPI设备名、探测接口信息等。

- 返回值：

- 成功：0。
- 失败：除0外的其他值。

4.3. spi_message_init

该接口有以下参数。

- 简介：

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

- 描述：该接口用于初始化SPI信息结构，以清除或初始化传输队列。

- 参数：

- **m**: SPI信息类型。
- **返回值**: 无。



5. 示例用例

以下列出了昉·惊鸿7110 SPI的典型用例。

查找原始内核驱动程序

驱动程序文件在以下路径：

```
freelight-u-sdk/linux/drivers/spi/spidev.c
```

该驱动程序是一个Linux嵌入式SPI设备驱动程序。

注册一个SPI驱动程序

您可以使用[spi_register_driver \(第 17页\)](#)接口来注册一个SPI驱动程序，作为SPI消息读写的基础。

以下代码为一个示例：

```
static int __init spidev_init(void)
{
    int status;

    /* Claim our 256 reserved device numbers. Then register a class
     * that will key udev/mdev to add/remove /dev nodes. Last, register
     * the driver which manages those device numbers.
     */
    BUILD_BUG_ON(N_SPI_MINORS > 256);
    status = register_chrdev(SPIDEV_MAJOR, "spi", &spidev_fops);
    if (status < 0)
        return status;

    spidev_class = class_create(THIS_MODULE, "spidev");
    if (IS_ERR(spidev_class)) {
        unregister_chrdev(SPIDEV_MAJOR, spidev_spi_driver.driver.name);
        return PTR_ERR(spidev_class);
    }

    status = spi_register_driver(&spidev_spi_driver);
    if (status < 0) {
        class_destroy(spidev_class);
        unregister_chrdev(SPIDEV_MAJOR, spidev_spi_driver.driver.name);
    }
    return status;
}
module_init(spidev_init);
```

配置SPI驱动程序

同时，确保您已在dts文件中为SPI控制器添加了子设备的设备信息描述。

以下代码以spi0为例：

```
&spi0 {
    pinctrl-names = "default";
    pinctrl-0 = <&ssp0_pins>;
    status = "disabled";
    spi_dev0: spi@0 {
        compatible = "rohm,dh2228fv";
        pl022,com-mode = <1>;
        spi-max-frequency = <10000000>;
        reg = <0>;
        status = "okay";
    };
};
```

配置文件spi_dev0包含以下参数。

- **compatible**：驱动程序的兼容性信息。
- **pl022,com-mode**：驱动程序的通信模式。以下为可用值：
 - 0：轮询
 - 1：中断
 - 2：DMA
- **spi-max-frequency**：从设备最大频率值



注：

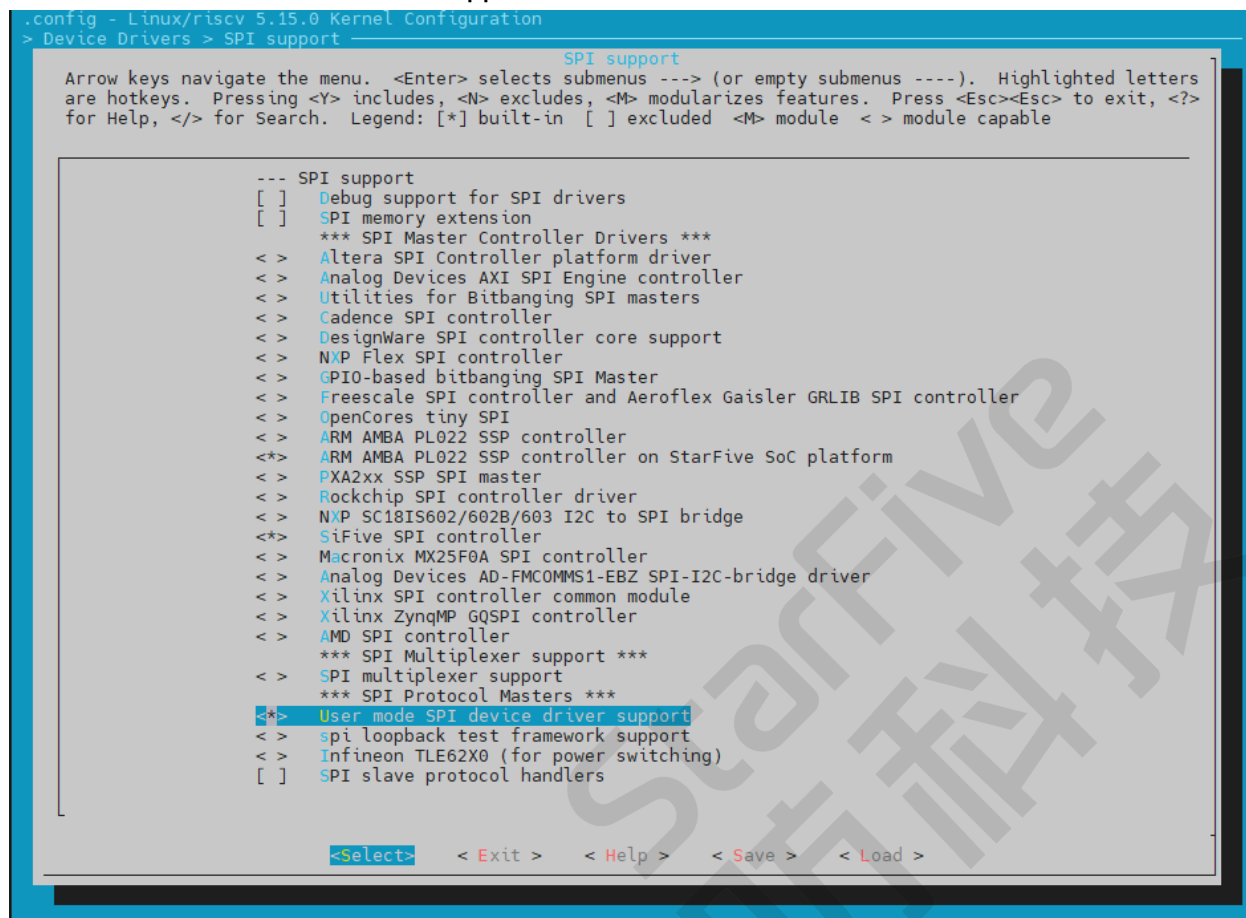
确保您根据实际情况设置了正确的最大频率值，如果配置了不正确的值，可能会导致传输中的数据丢失。

- **reg**：从设备的寄存器地址偏移量。
- **status**：从设备的状态。以下为可用值：
 - **okay**：从设备工作正常。
 - **disabled**：从设备被禁用。

配置内核菜单

在内核菜单配置页面，请选择**User mode SPI device driver support**选项。

图 5-1 User mode SPI device driver support



构建SPI文件

完成固件安装后，按照以下步骤构建SPI文件。

1. 在/dev/文件夹下找到spidevX.0 (X=1-7) 设备。
2. 在文件上执行读写操作。或者您可以使用Linux SPI工具，并在路径下运行以下命令：

```
freelight-u-sdk/linux/tools
```

3. 执行以下命令，构建用于测试的SPI文件。

```
# make spi
```

结果： freelight-u-sdk/linux/tools/spi路径下生成可执行文件spidev_test。

测试SPI文件

按照以下步骤测试已生成额SPI文件。

1. 将生成的文件复制到SoC中，并连接SPI上的I/O接口的TX和RX。
2. 然后运行以下测试命令：

```
# /spidev_test -D /dev/spidevX.0 -v -p data
```

结果：系统将显示您要传输的内容，测试示例如下：

图 5-2 SPI测试示例

```
# ls /dev/spi*
/dev/spidev1.0 /dev/spidev3.0 /dev/spidev5.0 /dev/spidev7.0
/dev/spidev2.0 /dev/spidev4.0 /dev/spidev6.0
# ./spidev_test -D /dev/spidev1.0 -v -p string_to_send
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 kHz)
TX | 73 74 72 69 6E 67 5F 74 6F 5F 73 65 6E 64 |string_to_send|
RX | 73 74 72 69 6E 67 5F 74 6F 5F 73 65 6E 64 |string_to_send|
# █
```

6. 常见问题集

以下列出了有关SPI的常见问题。

DMA故障

问题描述： DMA模式无法从内核加载日志。

下图为一个加载失败的示例。

图 6-1 DMA故障

```
[ 6.625672] ssp-pl022 10060000.spi: ARM PL022 driver for StarFive SoC platform, device ID: 0x00041022
[ 6.634827] ssp-pl022 10060000.spi: mapped registers from 0x0000000010060000 to (____ptrval____)
[ 6.643704] ssp-pl022 10060000.spi: Failed to work in dma mode, work without dma!
```

分析： 这是正常情况，因为DMA没有在设备树中配置。DMA通道较少，不能完全满足需求，SPI不使用DMA通道进行传输。

解决方法： 如果要使用DMA通道进行传输，则应更改设备树并添加DMA配置。

下图中方框部分提供了一个示例解决方案。

图 6-2 DMA故障解决方案

```
spi0: spi@10060000 {
    compatible = "arm,pl022", "arm,primecell";
    reg = <0x0 0x10060000 0x0 0x10000>;
    clocks = <&clkgen JH7110_SPI0_CLK_APB>;
    clock-names = "apb_pclk";
    resets = <&rstgen RSTN_U0_SSP_SPI_APB>;
    reset-names = "rst_apb";
    interrupts = <38>;
    /* shortage of dma channel that not be used */
    dmas = <&dma 14 1>, <&dma 15 1>;
    dma-names = "rx", "tx";
    arm,primecell-periphid = <0x00041022>;
    num-cs = <1>;
    #address-cells = <1>;
    #size-cells = <0>;
    status = "disabled";
};
```