StarFive 赛昉科技

# Software SDK Developer Guide for SPI

VisionFive 2

Version: 1.0

Date: 2022/11/10

Doc ID: JH7110-DGEN-005

# Legal Statements

Important legal notice before reading this documentation.

### PROPRIETARY NOTICE



Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.



### Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

# Preface

About this guide and technical support information.

## About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the SPI of the StarFive next generation SoC platform - JH7110.

## Audience

This document mainly serves the SPI relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.

## Revision History

**Table 0-1 Revision History**

| Version | Released | Revision |
|---|---|---|
| 1.0 | | First official release. |

## Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

# Contents

# List of Tables

# List of Figures

# 1. Introduction

*Serial peripheral interface (SPI)* is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others.

## 1.1. Function Introduction

The JH7110 SoC Platform supports the following features and specifications on SPI.

- Support 7 SPI interfaces.
- Support serial-master mode and serial-slave mode. Use software configuration to switch between the two modes.
- Provide separate data bits for input and output.
- Provide configurable FIFO with size up to 8-bit × 16-bit for both TX and RX channels.
- Support DMA access for TX and RX FIFO.

## 1.2. Device Tree Overview

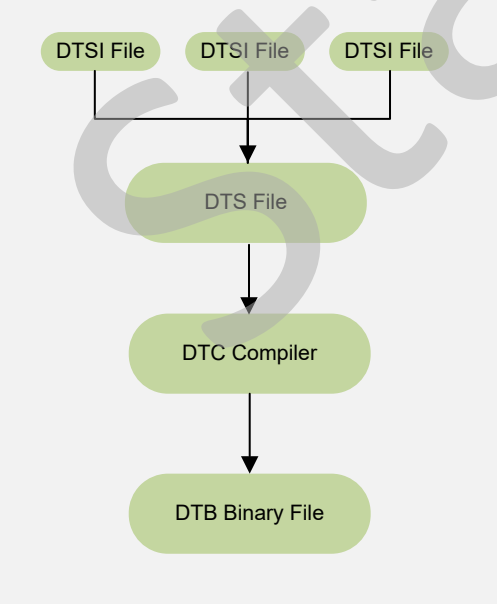
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

**Figure 1-1 Device Tree Workflow**



## 1.3. Device Tree Source Code

### Overview Structure

The device tree source code of JH7110 is listed as follows:

```
linux
├── arch
│   ├── riscv
│   │   ├── boot
│   │   │   ├── dts
│   │   │   │   └── starfive
│   │   │   │       ├── codecs
│   │   │   │       │   ├── sf_pdm.dtsi
│   │   │   │       │   ├── sf_pwmdac.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
│   │   │   │       ├── jh7110-evb-can-pdm-pwmdac.dts
│   │   │   │       ├── jh7110-evb.dts
│   │   │   │       ├── jh7110-evb.dtsi
│   │   │   │       ├── jh7110-evb-dvp-rgb2hdmi.dts
│   │   │   │       ├── jh7110-evb-pcie-i2s-sd.dts
│   │   │   │       ├── jh7110-evb-pinctrl.dtsi
│   │   │   │       ├── jh7110-evb-spi-uart2.dts
│   │   │   │       ├── jh7110-evb-uart1-rgb2hdmi.dts
│   │   │   │       ├── jh7110-evb-uart4-emmc-spdif.dts
│   │   │   │       ├── jh7110-evb-uart5-pwm-i2c-tdm.dts
│   │   │   │       ├── jh7110-fpga.dts
│   │   │   │       ├── jh7110-visionfive-v2.dts
│   │   │   │       ├── Makefile
│   │   │   │       └── vf2-overlay
│   │   │   │           ├── Makefile
│   │   │   │           └── vf2-overlay-uart3-i2c.dts
```

### SoC Platform

The device tree source code of the JH7110 SoC platform is in the following path:

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

### VisionFive 2

The device tree source code of the VisionFive 2 *Single Board Computer (SBC)* is in the following path:

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

## 1.4. Source Code Structure

The following list shows the source code of SPI since Linux 5.15.

- `drivers/spi/spi.c`: The file contains source code for the SPI driver framework.
- `drivers/spi/spidev.c`: The file contains the source code to create the SPI device node and use it in user mode.

- `drivers/amba/bus.c`: The file contains the source code to register platform devices through AMBA bus.
- `driver/spi/spi-pl022-starfive.c`: The file contains the source code of SPI controller driver on the StarFive JH7110 platform.
- `tools/spi/spidev_test.c`: The file contains the SPI test tool in user mode.

# 2. Configuration

## 2.1. Kernel Menu Configuration

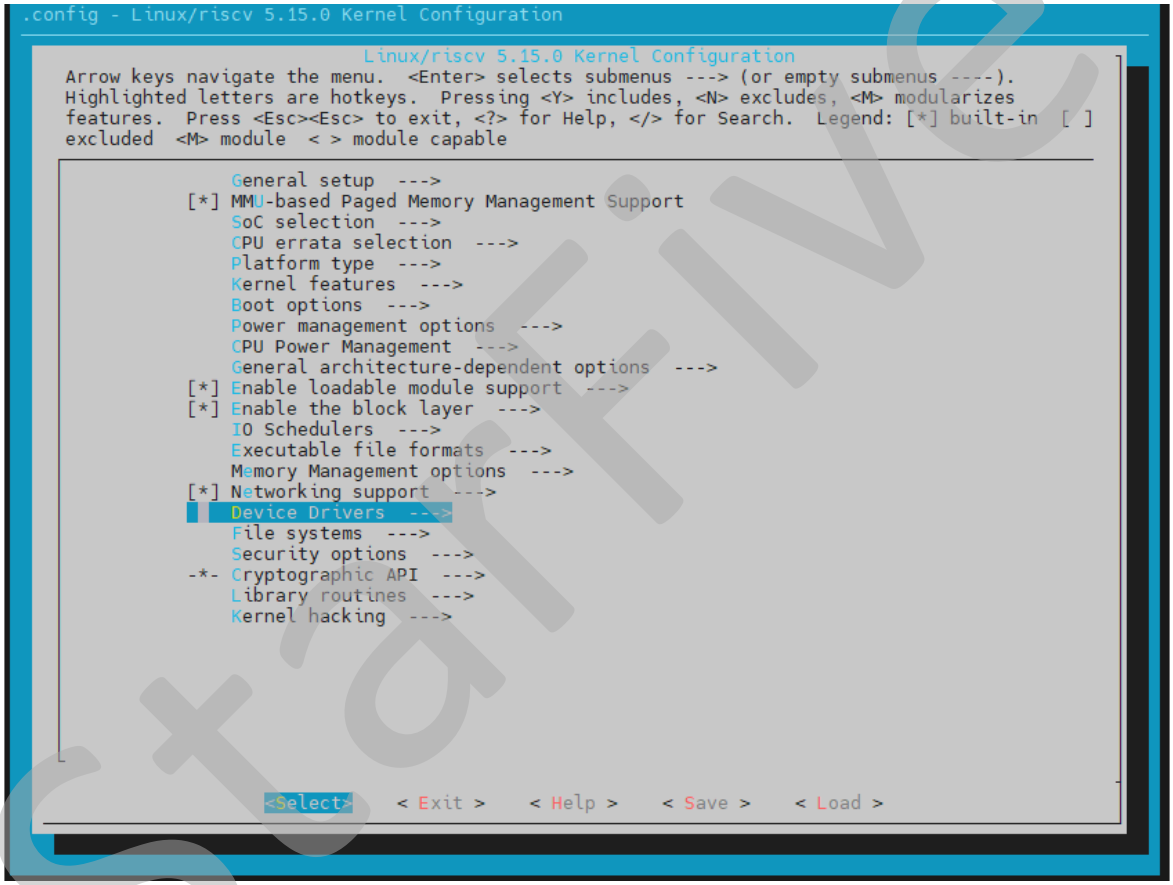
Follow the steps below to enable the kernel configuration for SPI.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

```
make linux-menuconfig
```

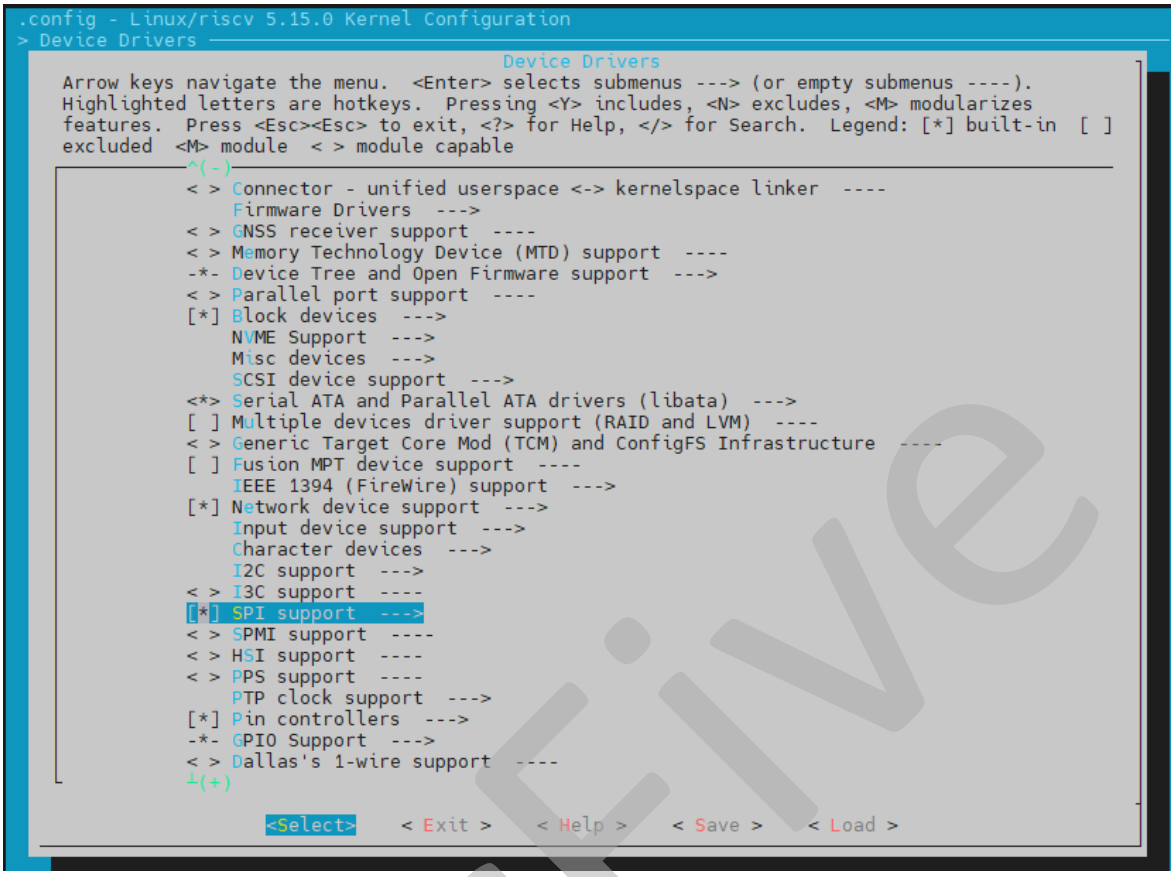
2. Enter the **Device Drivers** menu.

**Figure 2-1 Device Drivers**



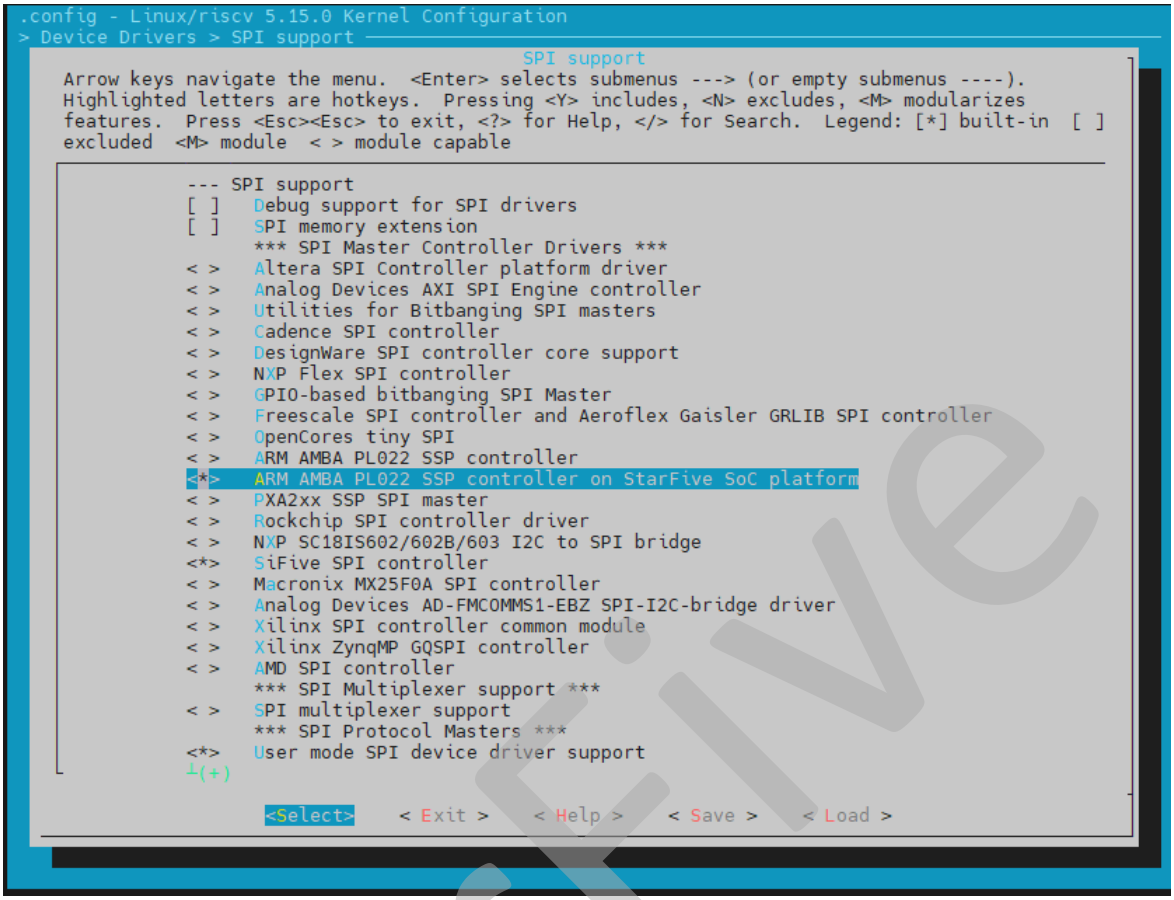

3. Select the **SPI support** menu.

**Figure 2-2 SPI Support**




4. Select the **SSP controller** option.

**Figure 2-3 SSP Controller**




5. Save your change before you exit the kernel configuration dialog.

## 2.2. Device Tree Configuration

The device tree of SPI is stored in the following path:

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

The following code block provides an example of how to set SPI0.

```
spi0: spi@10060000 {
        compatible = "arm,pl022", "arm,primecell";
        reg = <0x0 0x10060000 0x0 0xl0000>;
        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";
```

The following list provides explanations for the parameters included in the above code block.

- **compatible**: Compatibility information, used to associate the driver and its target device.
- **reg**: Register base address "`0x10060000`" and range "`0x10000`".

- **clocks**: The clocks used by the SPI module.
- **clock-names**: The names of the above clocks.
- **resets**: The reset signals used by the SPI module.
- **reset-names**: The names of the above reset signals.
- **interrupts**: Hardware interrupt ID.
- **primecell-periphid**: The peripheral ID of the SPI device.
- **num-cs**: The total count of the chip select signals.
- **status**: The work status of the SPI module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

## 2.3. Board Level Configuration

The board level device tree file (DTSI file) stores all differentiation information for each board-level device. (For example, `common.dtsi`, `pinctrl.dtsi`, and `evb.dts`, etc. )

The `common.dtsi` file is stored in the following path:

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

In the file, spi0 has the following settings.

```
&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";
        };
};
```

The following list provides more descriptions for the configuration bits.

- **spi-max-frequency**: Edit this bit to configure the communication clock frequency of SPI.
- **status**: Edit this bit to define whether this module is enabled.

### VisionFive 2 Board Level Configuration

The pinctrl.dtsi file contains the pin control configuration. The file is stored in the following path:

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

The following code block provides an example of the pins used by spi0 including **tx** (Transceiver), **rx** (Receiver), **clk** (clock), and **cs** (chip select) signals.

```
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>;
```

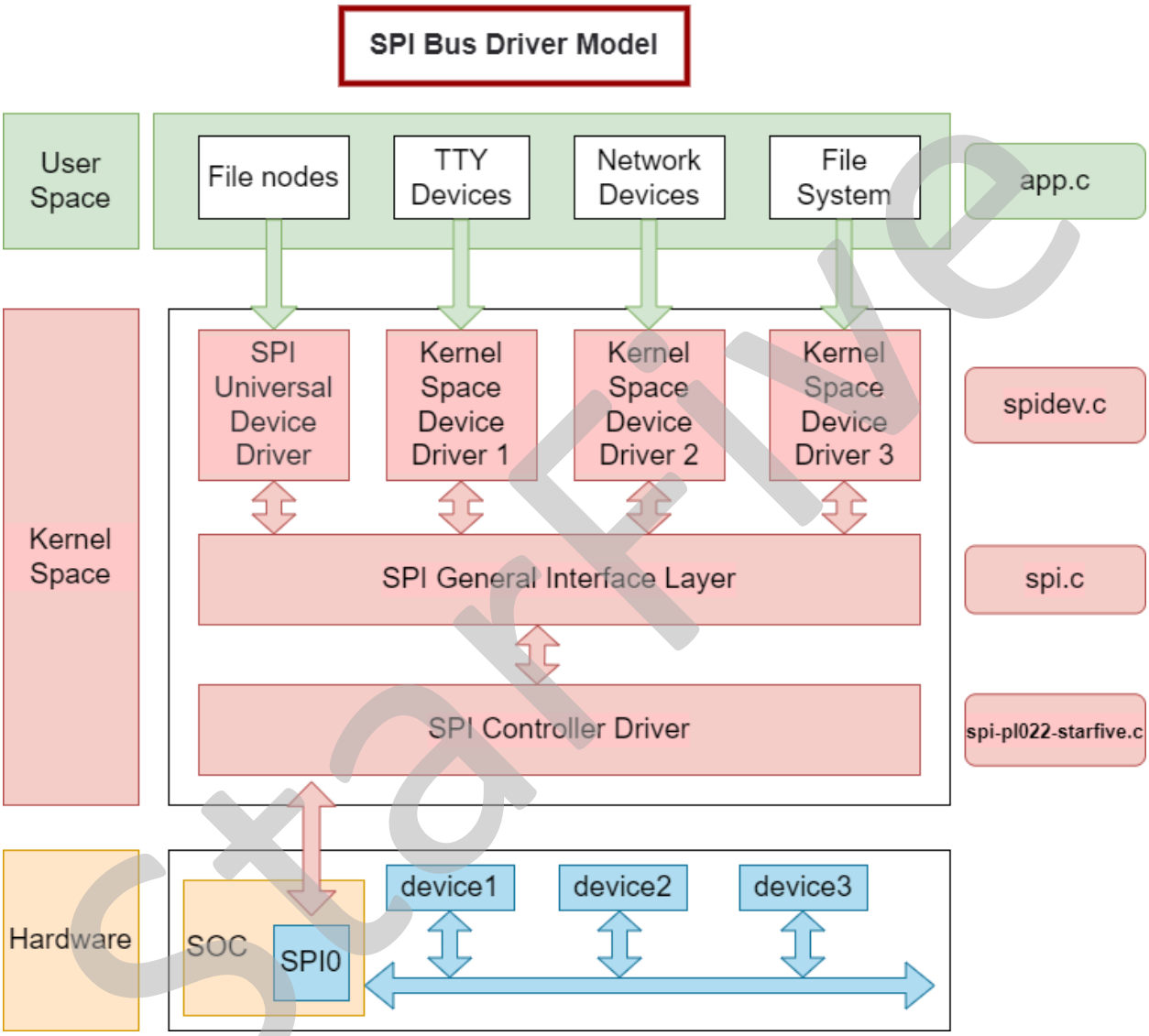
```
                sf,pin-gpio-din =  <GPI_SPI0_SSPRXD>;
        };

        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. Driver Framework

## 3.1. Block Diagram

The following block diagram shows the 3 layers of the SPI driver framework.

**Figure 3-1 Block Diagram**




The following sections describe each layer in the above diagram.

### User Space

The user space layer includes all the applications which use the SPI device. In this layer, users can customize their SPI device based on their specific needs.

### Kernel Space

The kernel space layer can be divided into the following 3 parts.

- SPI Device Driver Layer:

  The Linux kernel does not provide a specific SPI device driver, since the devices on an SPI may vary from case to case. Users have to use the general SPI device driver which can only communicate with the SPI device in synchronous modes. So only some simple and not-so-much-data-consumptive devices are supported in this layer.

  In this layer, we provide `spidev.c` as the standard SPI driver and `spi-nand.c` as the NAND driver for SPI.

- SPI General Interface Package Layer:

  To simplify SPI driver programming and reduce driver program coupling, Linux kernel packages some general programs of drivers for controllers and protocols to form the SPI general interface package layer.

  In this layer, we provide the Linux original driver `spi.c`.

- SPI Controller Driver Layer:

  This layer is the focus of our attention, it will be introduced in detail in the following parts of the document.

  In this layer, we provide the driver `spi-pl022-starfive.c`.

### Hardware

The hardware layer is the physical device layer. In this layer, SPI controllers and the connected SPI devices communicate with CPU via SPI bus.

# 4. Interface Description

## 4.1. Interface Definition

The interface definition of SPI is stored in the following file: `include/linux/spi/spi.h`, the primary interfaces are **spi_register_driver** and **spi_message_init**.

The macro **module_spi_driver()** is used to register an SPI device in a short while.

The following code block shows an example.

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

## 4.2. spi_register_driver

The interface has the following parameters.

- **Synopsis**:

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

- **Description**: The interface is used to register the driver of an SPI device.
- **Parameter**:
  - **sdrv**: `spi_driver` type, includes the SPI device name, probe interface information and so on.
- **Return**:
  - **Success**: 0.
  - **Failure**: Any value other than 0.

## 4.3. spi_message_init

The interface has the following parameters.

- **Synopsis**:

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

- **Description**: The interface is used to initialize an SPI message structure to clear or initialize a transfer queue.
- **Parameter**:
  - **m**: The SPI message type.
- **Return**: None.

# 5. Example Use Case

The following section lists a typical use case of JH7110 SPI.

### Locating the Original Kernel Driver

The driver file is stored in the following path.

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

The driver is a Linux embedded SPI device driver.

### Registering an SPI Driver

You can use the spi_register_driver *(on page 17)* interface to register an SPI driver, as a basis for SPI message reading and writing.

The following code block provides an example.

```
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);
```

### Configuring the SPI Driver

In the meantime, make sure you have added device information descriptions of the sub-devices for the SPI controller in the dts file.

The following code block shows an example of spi0.

```
&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";
};
```

The configuration file contains the following parameters for **spi_dev0**.

- **compatible**: The compatibility information of the driver.
- **pl022,com-mode**: The communication mode of the driver. The following values are available.
  - **0**: Polling.
  - **1**: Interrupt
  - **2**: DMA
- **spi-max-frequency**: The maximum frequency of the slave device.

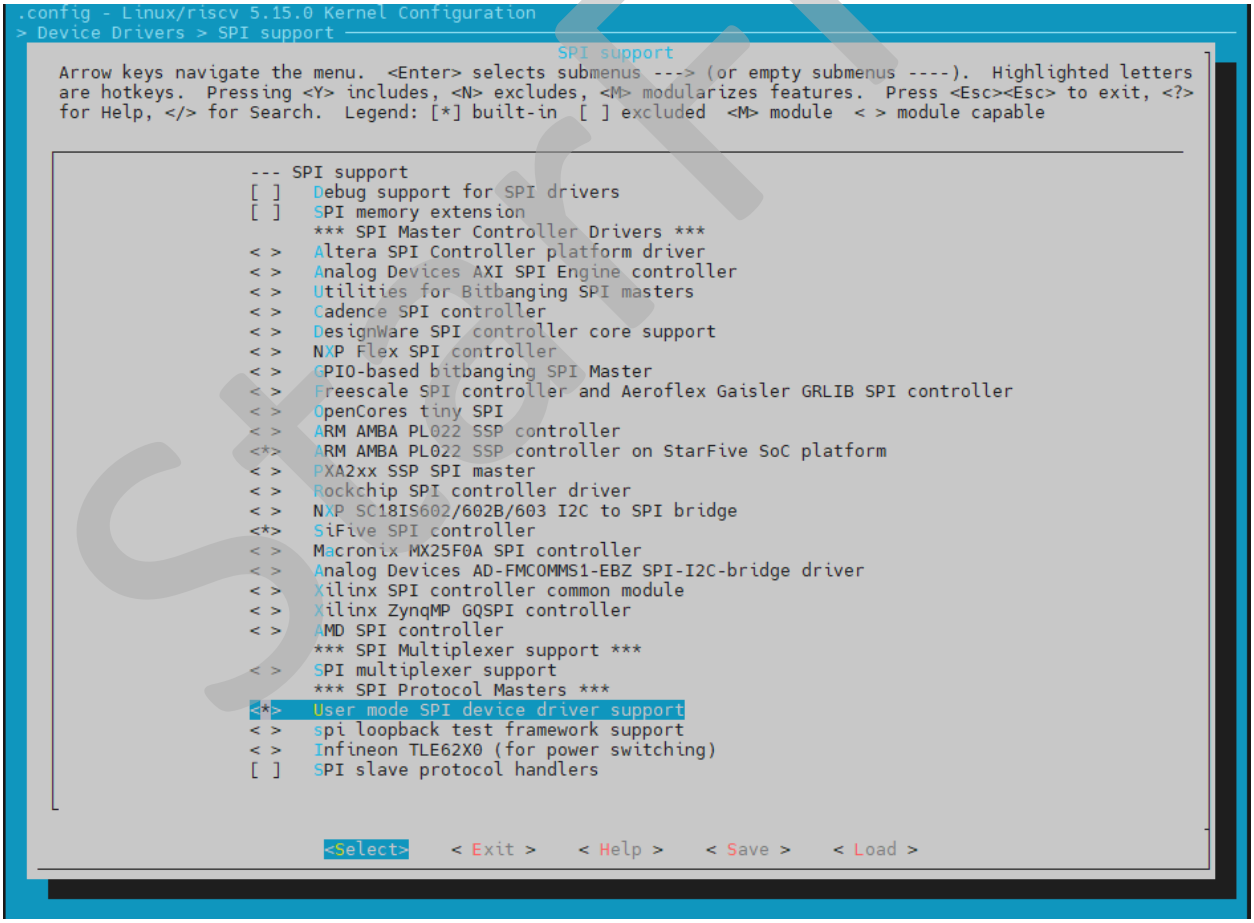
**Note:**
Make sure you set the a proper value for maximum frequency based on your real conditions, data loss in transmission maybe caused if an improper value is configured.

- **reg**: The register address offset of the slave device.
- **status**: Status of the slave device. The following values are available.
  - **okay**: The slave device is working properly.
  - **disabled**: The slave device has been disabled.

### Configuring Kernel Menu

In the kernel menu configuration, make sure you have selected the **User mode SPI device driver support** option.

**Figure 5-1 User Mode SPI Support**




### Building the SPI File

After you have completed the firmware installation, follow the steps below to build the SPI file.

1. Find the `spidevX.0` (X = 1-7) device under the `/dev` folder.
2. Perform reading and writing operations on the file. Or you can use the Linux SPI tool, and run the following command under path:

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

3. Use the following command to build SPI file for test.

   ```
   # make spi
   ```

   **Result**: An executable file named `spidev_test` is generated under the `freelight-u-sdk/linux/tools/spi` path.

### Testing the SPI File

Follow the steps below to test your generated SPI file.

1. Copy the generated file to the SoC and connect the TX and RX of the I/O port on the SPI.
2. Then run the following command to test:

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

   **Result**: The displayed data is the content to be transferred. The test example is as follows.

**Figure 5-2 SPI Test Example**

# 6. FAQ

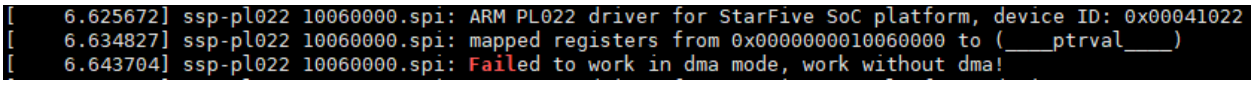
The following section shows the *Frequently Asked Questions (FAQ)* of JH7110 SPI.

### DMA Failure

**Problem description**: DMA mode failed to load logs from the kernel.

The following image displays an example failure screen.

**Figure 6-1 DMA Failure**




**Analysis**: This is a normal case, since DMA is not configured in device tree. There are fewer DMA channels and cannot fully meet the demand, and SPI does not use the DMA channel to transfer.

**Solution**: If you want to use DMA channel to transfer, you should change the device tree and add DMA configuration.

The highlighted section in the following image provides an example solution.

**Figure 6-2 DMA Failure Solution**

```
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";
};
```