AN0004 SPI设备应用指南

!!! abstract "摘要"

> 本应用笔记以驱动SPI接口的OLED显示屏为例，说明了如何添加SPI设备驱动框架及底层硬件驱动，使用SPI设备驱动接口开发应用程序。并给出了在正点原子STM32F4探索者开发板上验证的代码示例。

1 本文的目的和结构

1.1 本文的目的和背景

串行外设接口（Serial Peripheral Interface Bus，SPI），是一种用于短程通信的同步串行通信接口规范，主要应用于单片机系统中。SPI主要应用于 EEPROM、FLASH、实时时钟、AD转换器、数字信号处理器和数字信号解码器等。在芯片的管脚上占用四根线或三根线，简单易用，因此越来越多的芯片集成了这种通信接口。

为了方便应用层程序开发，RT-Thread中引入了SPI设备驱动框架。本文说明了如何使用RT-Thread SPI设备驱动。

1.2 本文的结构

本文首先简要介绍了RT-Thread SPI设备驱动框架，然后在正点原子STM32F4探索者开发板上运行了SPI设备驱动示例代码。最后详细描述SPI设备驱动框架接口的使用方法及参数取值。

2 SPI设备驱动框架简介

RT-Thread SPI设备驱动框架把MCU的SPI硬件控制器虚拟成SPI总线（SPI BUS#n），总线上可以挂很多SPI设备(SPI BUS#0 CSm)，每个SPI设备只能挂载到一个SPI总线上。目前，RT-Thread已经实现了很多通用SPI设备的驱动，比如SD卡、各种系列Flash存储器、ENC28J60以太网模块等。SPI设备驱动框架的层次结构如下图所示。

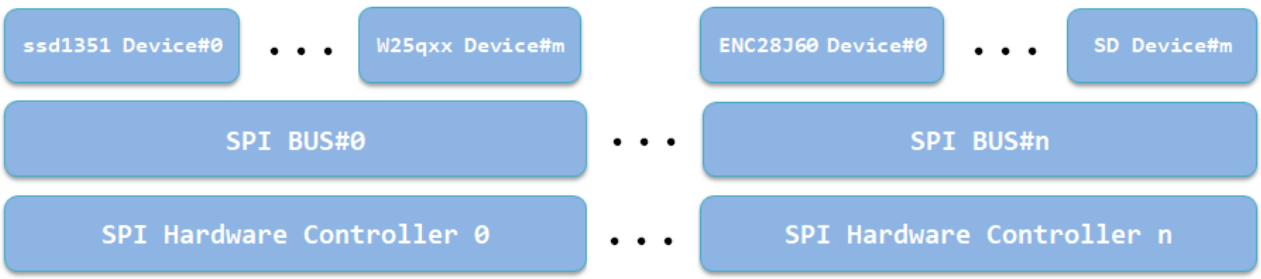

图2-1SPI设备驱动框架层次结构体

基于前面的介绍用户已经大致了解了RT-Thread SPI设备驱动框架，那么用户如何使用SPI设备驱动框架呢？

3 运行示例代码

本章节基于正点原子探索者STM32F4 开发板及SPI示例代码，给出了RT-Thread SPI设备驱动框架的使用方法。

3.1 示例代码软硬件资源

1. RT-Thread 源码
2. ENV工具
3. SPI设备驱动示例代码
4. 正点原子STM32F4探索者开发板
5. 1.5寸彩色OLED显示屏(SSD1351控制器)
6. MDK5

正点原子探索者STM32F4 开发板的MCU是STM32F407ZGT6，本示例使用USB转串口（USART1）发送数据及供电，使用SEGGER J-LINK连接JTAG调试，STM32F4 有多个硬件SPI控制器，本例使用 SPI1。彩色OLED显示屏板载SSD1351控制器，分辨率128*128。

STM32F4 与 OLED 显示屏管脚连接如下表所示：

STM32管脚	**OLED显示屏管脚**	**说明**
PA5	D0	SPI1 SCK，时钟
PA6		SPI1 MISO，未使用
PA7	D1	SPI1 MOSI，主机输出，从机输入
PC6	D/C	GPIO，输出，命令0/数据1选择
PC7	RES	GPIO，输出，复位，低电平有效
PC8	CS	GPIO，输出，片选，低电平有效
3.3V	VCC	供电
GND	GND	接地

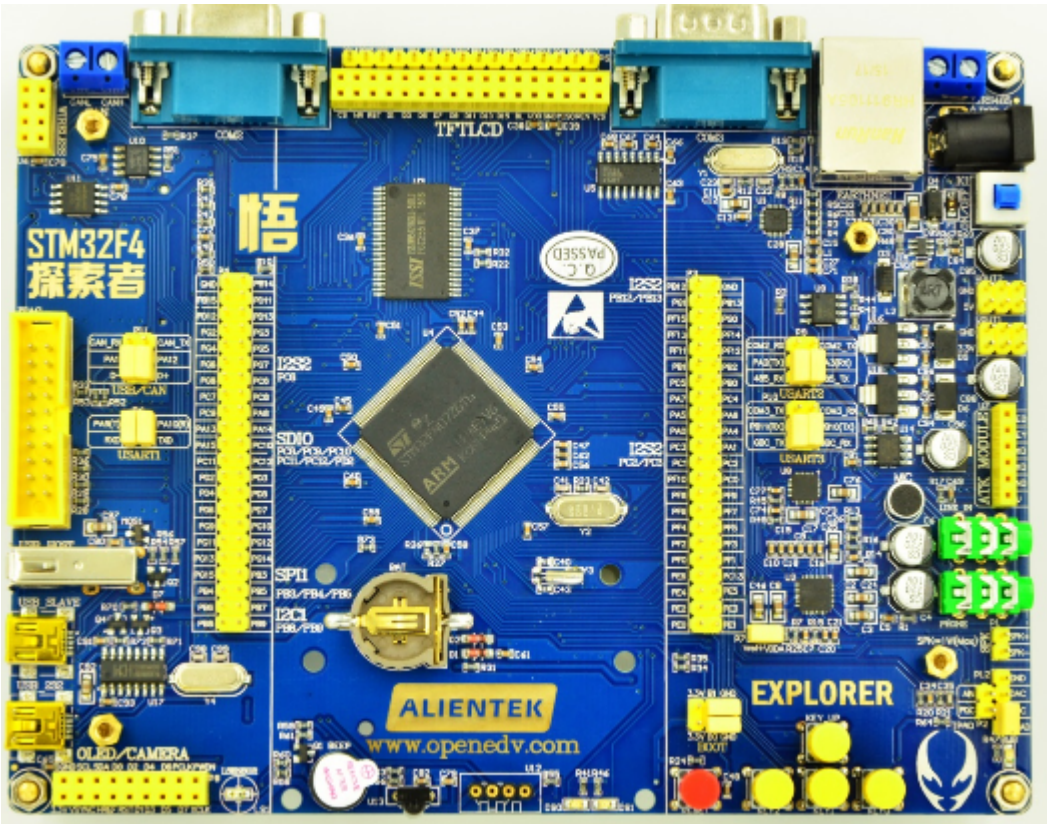

图3.1-1 正点原子开发板

图3.1-2 彩色OLED显示屏

SPI设备驱动示例代码包括`app.c、drv_ssd1351.c、drv_ssd1351.h`3个文件，`drv_ssd1351.c`是OLED显示屏驱动文件，此驱动文件包含了SPI设备ssd1351的初始化、挂载到系统及通过命令控制OLED显示的操作方法。由于RT-Thread上层应用API的通用性，因此这些代码不局限于具体的硬件平台，用户可以轻松将它移植到其它平台上。

3.2 配置工程

使用menuconfig配置工程：在env工具命令行使用cd 命令进入 rt-thread\bsp\stm32f4xx-HAL 目录，然后输入`menuconfig`命令进入配置界面。

- 修改工程芯片型号：修改 Device type为STM32F407ZG。

- 配置shell使用串口1：选中Using UART1，进入RT-Thread Kernel ---> Kernel Device Object菜单，修改the device name for console为uart1。
- 开启SPI总线及设备驱动并注册SPI总线到系统：进入RT-Thread Components ---> Device Drivers菜单，选中Using SPI Bus/Device device drivers，RT-Thread Configuration界面会默认选中Using SPI1，spi1总线设备会注册到操作系统。

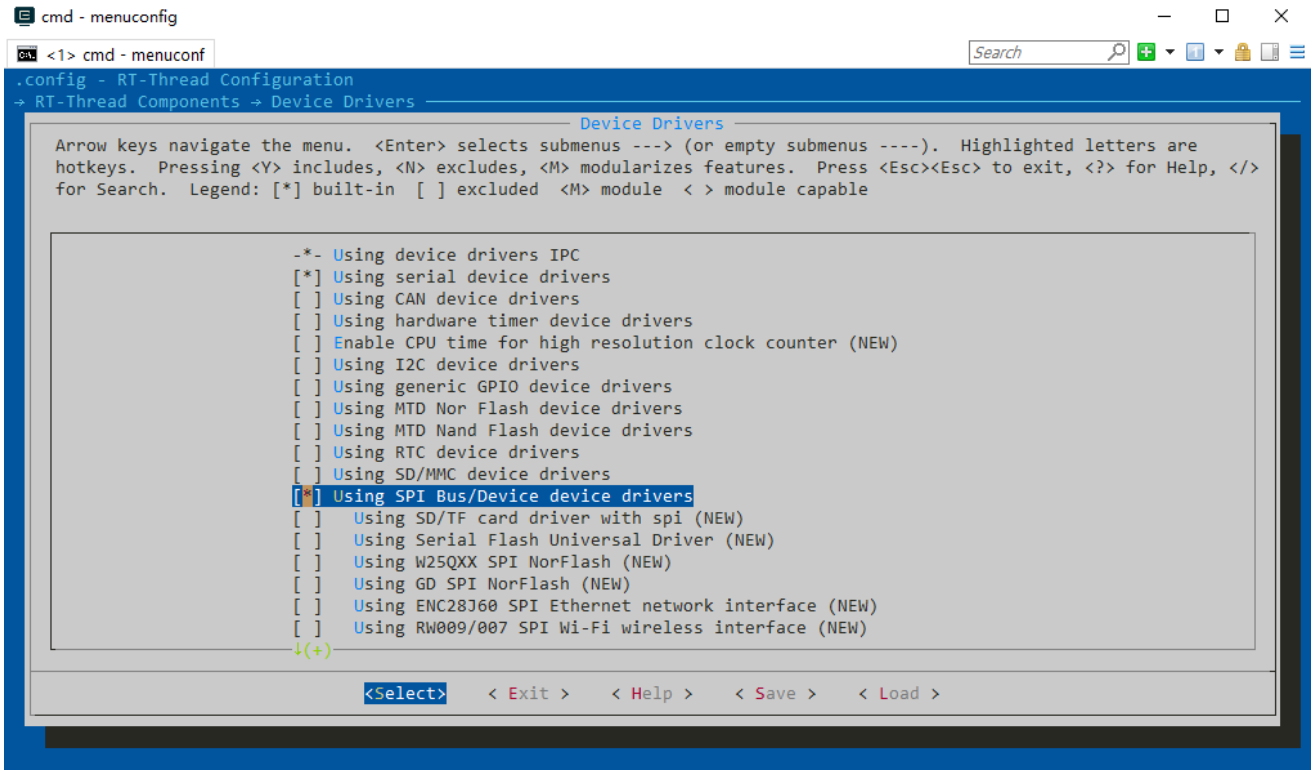

图3.2-1 使用menuconfig开启SPI

- 开启GPIO驱动：进入RT-Thread Components ---> Device Drivers菜单，选中Using generic GPIO device drivers。OLED屏需要2个额外的GPIO用于DC、RES信号，SPI总线驱动也需要对片选管脚进行操作，都需要调用系统的GPIO驱动接口。

生成新工程及修改调试选项：退出menuconfig配置界面并保存配置，在ENV命令行输入`scons --target=mdk5 -s`命令生成mdk5工程，新工程名为project。使用MDK5打开工程，修改调试选项为J-LINK。

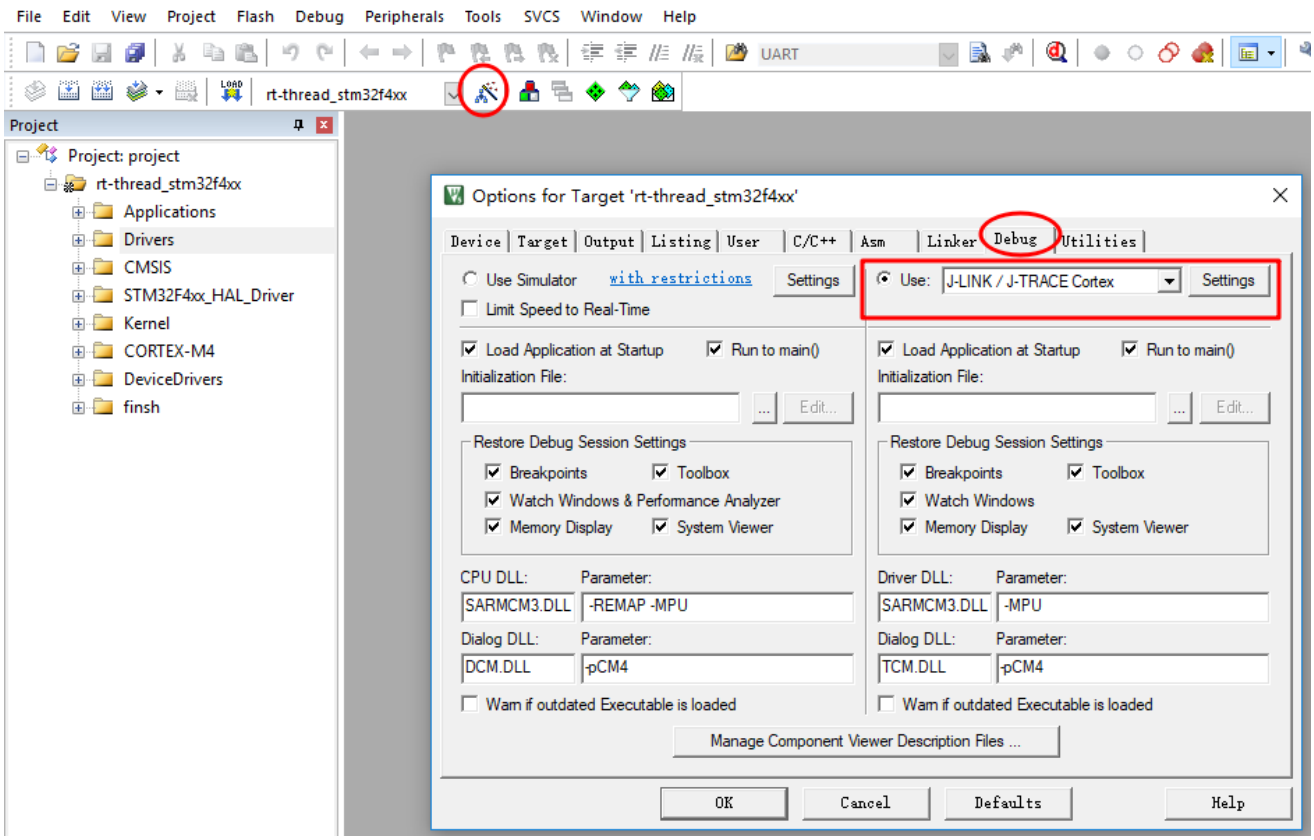

图3.2-2 修改调试选项

使用list_device命令查看SPI总线：添加SPI底层硬件驱动无误后，在终端PuTTY(打开对应端口，波特率配置为115200)使用`list_device`命令就能看到SPI总线。同样可以看到我们使用的UART设备和PIN设备。

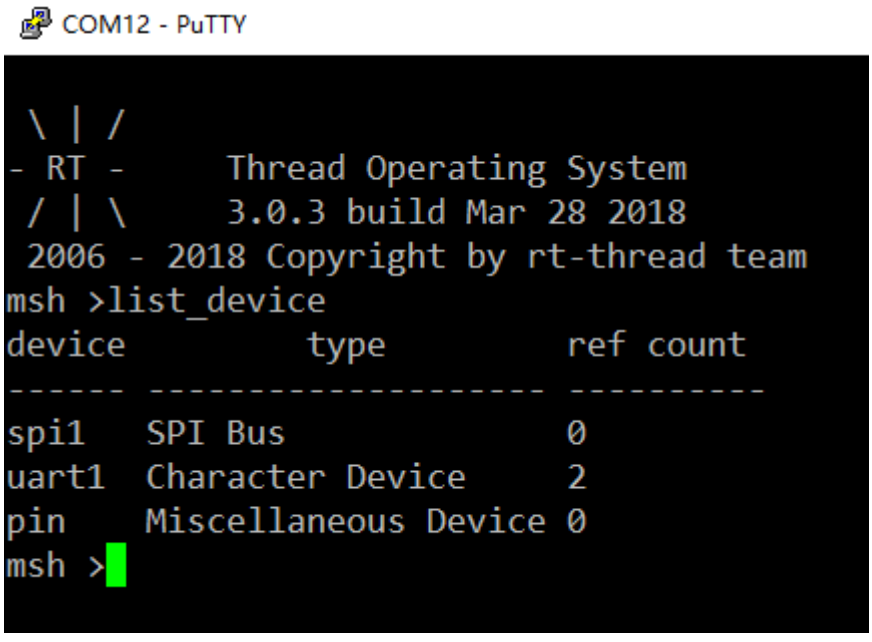

图3.2-3使用list_device命令查看系统设备

3.3 添加示例代码

将SPI设备驱动示例代码里的`app.c`拷贝到\rt-thread\bsp\stm32f4xx-HAL\applications目录下。`drv_ssd1351.c、drv_ssd1351.h`拷贝到\rt-thread\bsp\stm32f4xx-HAL\drivers目录下，并将它们添加到工程中对应分组。如图所示：

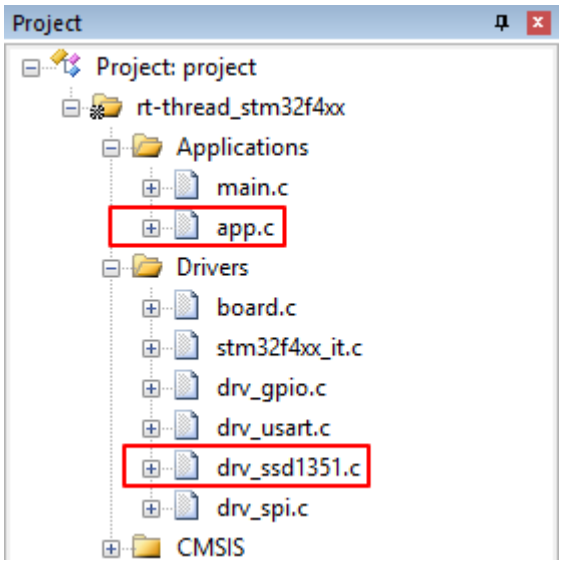

图3.3-1 添加示例代码到工程

在`main.c`中调用`app_init()`，`app_init()`会创建一个oled线程，线程会循环展示彩虹颜色图案和正方形颜图案。

`main.c`调用测试代码源码如下:

```
#include <rtthread.h>
#include <board.h>

extern int app_init(void);

int main(void)
{
  /* user app entry */

  app_init();

  return 0;
}
```

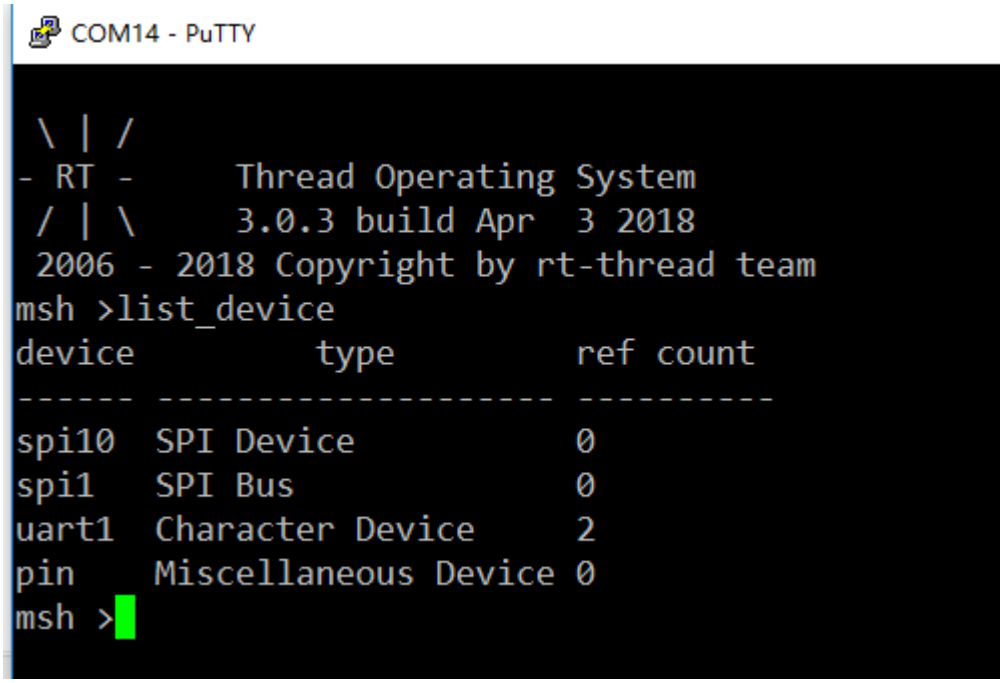

图3.3-2 使用list_device命令查看SPI设备驱动

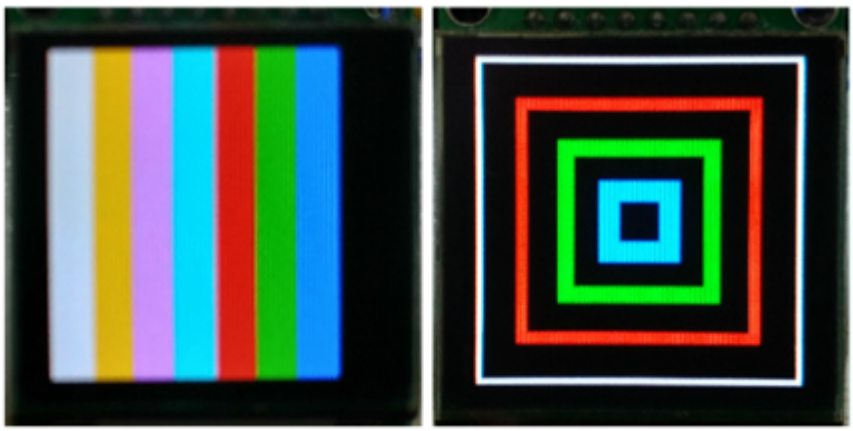

图3.3-3 实验现象

4 SPI设备驱动接口使用详解

按照前文的步骤，相信读者能很快的将RT-Thread SPI设备驱动运行起来，那么如何使用SPI设备驱动接口开发应用程序呢？

RT-Thread SPI设备驱动使用流程大致如下：

1. 定义SPI设备对象，调用`rt_spi_bus_attach_device()`挂载SPI设备到SPI总线。
2. 调用`rt_spi_configure()`配置SPI总线模式。
3. 使用`rt_spi_transfer()`等相关数据传输接口传输数据。

接下来本章节将详细讲解示例代码使用到的主要的SPI设备驱动接口。

4.1 挂载SPI设备到总线

用户定义了SPI设备对象后就可以调用此函数挂载SPI设备到SPI总线。

函数原型:

```
rt_err_t rt_spi_bus_attach_device(struct rt_spi_device *device,
                                  const char          *name,
                                  const char          *bus_name,
                                  void                *user_data)
```

参数	描述
device	SPI设备句柄
name	SPI设备名称
bus_name	SPI总线名称
user_data	用户数据指针

函数返回：成功返回RT_EOK，否则返回错误码。

此函数用于挂载一个SPI设备到指定的SPI总线，向内核注册SPI设备，并将user_data保存到SPI设备device里。

注意

1. 用户首先需要定义好SPI设备对象device
2. 推荐SPI总线命名原则为spix， SPI设备命名原则为spixy，如 本示例的spi10 表示挂载在在 spi1 总线上的 0 号设备。
3. SPI总线名称可以在msh shell输入list_device 命令查看，确定SPI设备要挂载的SPI总线。
4. user_data一般为SPI设备的CS引脚指针，进行数据传输时SPI控制器会操作此引脚进行片选。

本文示例代码底层驱动 `drv_ssd1351.c` 中 `rt_hw_ssd1351_config()` 挂载ssd1351设备到SPI总线源码如下：

```
#define SPI_BUS_NAME                "spi1"  /* SPI总线名称 */
#define SPI_SSD1351_DEVICE_NAME     "spi10" /* SPI设备名称 */

... ...

static struct rt_spi_device spi_dev_ssd1351; /* SPI设备ssd1351对象 */
static struct stm32_hw_spi_cs  spi_cs;  /* SPI设备CS片选引脚 */

... ...

static int rt_hw_ssd1351_config(void)
{
    rt_err_t res;

    /* oled use PC8 as CS */
    spi_cs.pin = CS_PIN;
    rt_pin_mode(spi_cs.pin, PIN_MODE_OUTPUT);    /* 设置片选管脚模式为输出 */



    res = rt_spi_bus_attach_device(&spi_dev_ssd1351, SPI_SSD1351_DEVICE_NAME, SPI_BUS_NAME,
```

```
    (void*)&spi_cs);
    if (res != RT_EOK)
    {
        OLED_TRACE("rt_spi_bus_attach_device!\r\n");
        return res;
    }

    ... ...
}


```

4.2 配置SPI模式

挂载SPI设备到SPI总线后，为满足不同设备的时钟、数据宽度等要求，通常需要配置SPI模式、频率参数。

SPI从设备的模式决定主设备的模式，所以SPI主设备的模式必须和从设备一样两者才能正常通讯。

函数原型:

```
rt_err_t rt_spi_configure(struct rt_spi_device *device,
                          struct rt_spi_configuration *cfg)
```

参数	描述
device	SPI设备句柄
cfg	SPI传输配置参数指针

函数返回：返回RT_EOK。

此函数会保存cfg指向的模式参数到device里，当device调用数据传输函数时都会使用此配置信息。

struct rt_spi_configuration 原型如下：

```
struct rt_spi_configuration
{
    rt_uint8_t mode;        //spi模式
    rt_uint8_t data_width;  //数据宽度，可取8位、16位、32位
    rt_uint16_t reserved;   //保留
    rt_uint32_t max_hz;     //最大频率
};
```

模式/mode:使用`spi.h`中的宏定义，包含MSB/LSB、主从模式、 时序模式等，可取宏组合如下。

```
/* 设置数据传输顺序是MSB位在前还是LSB位在前 */
#define RT_SPI_LSB      (0<<2)                      /* bit[2]: 0-LSB */
#define RT_SPI_MSB      (1<<2)                      /* bit[2]: 1-MSB */

/* 设置SPI的主从模式 */
```

```
#define RT_SPI_MASTER   (0<<3)                      /* SPI master device */
#define RT_SPI_SLAVE    (1<<3)                      /* SPI slave device */

/* 设置时钟极性和时钟相位 */
#define RT_SPI_MODE_0   (0 | 0)                     /* CPOL = 0, CPHA = 0 */
#define RT_SPI_MODE_1   (0 | RT_SPI_CPHA)           /* CPOL = 0, CPHA = 1 */
#define RT_SPI_MODE_2   (RT_SPI_CPOL | 0)           /* CPOL = 1, CPHA = 0 */
#define RT_SPI_MODE_3   (RT_SPI_CPOL | RT_SPI_CPHA) /* CPOL = 1, CPHA = 1 */

#define RT_SPI_CS_HIGH  (1<<4)                      /* Chipselect active high */
#define RT_SPI_NO_CS    (1<<5)                      /* No chipselect */
#define RT_SPI_3WIRE    (1<<6)                      /* SI/SO pin shared */
#define RT_SPI_READY    (1<<7)                      /* Slave pulls low to pause */

```

数据宽度/data_width:根据SPI主设备及SPI从设备可发送及接收的数据宽度格式设置为8位、16位或者32位。

最大频率/max_hz:设置数据传输的波特率，同样根据SPI主设备及SPI从设备工作的波特率范围设置。

注意

挂载SPI设备到SPI总线后必须使用此函数配置SPI设备的传输参数。

本文示例代码底层驱动`drv_ssd1351.c`中`rt_hw_ssd1351_config()`配置SPI传输参数源码如下：

```
static int rt_hw_ssd1351_config(void)
{
    ... ...

    /* config spi */
    {
        struct rt_spi_configuration cfg;
        cfg.data_width = 8;
        cfg.mode = RT_SPI_MASTER | RT_SPI_MODE_0 | RT_SPI_MSB;
        cfg.max_hz = 20 * 1000 *1000; /* 20M,SPI max 42MHz,ssd1351 4-wire spi */

        rt_spi_configure(&spi_dev_ssd1351, &cfg);
    }

    ... ...

```

4.3 数据传输

SPI设备挂载到SPI总线并配置好相关SPI传输参数后就可以调用RT-Thread提供的一系列SPI设备驱动数据传输函数。

rt_spi_transfer_message()

函数原型:

```
struct rt_spi_message *rt_spi_transfer_message(struct rt_spi_device  *device,
                                struct rt_spi_message *message)
```

参数	描述
device	SPI设备句柄
message	消息指针

函数返回： 成功发送返回RT_NULL，否则返回指向剩余未发送的message

此函数可以传输一连串消息，用户可以很灵活的设置message结构体各参数的数值，从而可以很方便的控制数据传输方式。

struct rt_spi_message原型如下：

```
struct rt_spi_message
{
    const void *send_buf;          /* 发送缓冲区指针 */
    void *recv_buf;                /* 接收缓冲区指针 */
    rt_size_t length;              /* 发送/接收 数据字节数 */
    struct rt_spi_message *next;   /* 指向继续发送的下一条消息的指针 */

    unsigned cs_take    : 1;       /* 值为1，CS引脚拉低，值为0,不改变引脚状态 */
    unsigned cs_release : 1;       /* 值为1，CS引脚拉高，值为0,不改变引脚状态 */
};
```

SPI是一种全双工的通信总线，发送一字节数据的同时也会接收一字节数据，参数length为传输一次数据时发送/接收的数据字节数，发送的数据为send_buf指向的缓冲区数据，接收到的数据保存在recv_buf指向的缓冲区。若忽视接收的数据则recv_buf值为NULL，若忽视发送的数据只接收数据，则send_buf值为NULL。

参数next是指向继续发送的下一条消息的指针，若只发送一条消息，则此指针值置为NULL。

rt_spi_send()

函数原型:

```
rt_size_t rt_spi_send(struct rt_spi_device *device,
                      const void           *send_buf,
                      rt_size_t             length)
```

参数	描述
device	SPI设备句柄
send_buf	发送缓冲区指针
length	发送数据的字节数

函数返回： 成功发送的数据字节数

调用此函数发送send_buf指向的缓冲区的数据，忽略接收到的数据。

此函数等同于调用`rt_spi_transfer_message()`传输一条消息，message参数配置如下：

```
    struct rt_spi_message msg;

    msg.send_buf   = send_buf;
    msg.recv_buf   = RT_NULL;
    msg.length     = length;
    msg.cs_take    = 1;
    msg.cs_release = 1;
    msg.next       = RT_NULL;
```

注意

调用此函数将发送一次数据。开始发送数据时片选开始，函数返回时片选结束。

本文示例代码底层驱动`drv_ssd1351.c`调用`rt_spi_send()`向SSD1351发送指令和数据的函数源码如下：

```
rt_err_t ssd1351_write_cmd(const rt_uint8_t cmd)
{
    rt_size_t len;

    rt_pin_write(DC_PIN, PIN_LOW);    /* 命令低电平 */

    len = rt_spi_send(&spi_dev_ssd1351, &cmd, 1);

    if (len != 1)
    {
        OLED_TRACE("ssd1351_write_cmd error. %d\r\n",len);
        return -RT_ERROR;
    }
    else
    {
        return RT_EOK;
    }

}

rt_err_t ssd1351_write_data(const rt_uint8_t data)
{
    rt_size_t len;

    rt_pin_write(DC_PIN, PIN_HIGH);        /* 数据高电平 */

    len = rt_spi_send(&spi_dev_ssd1351, &data, 1);

    if (len != 1)
    {

        OLED_TRACE("ssd1351_write_data error. %d\r\n",len);
```

```
        return -RT_ERROR;
    }
    else
    {
        return RT_EOK;
    }
}

```

rt_spi_recv()

函数原型:

```
rt_size_t rt_spi_recv(struct rt_spi_device *device,
                      void                 *recv_buf,
                      rt_size_t             length)
```

参数	描述
device	SPI设备句柄
recv_buf	接受缓冲区指针
length	接受到的数据字节数

函数返回：成功接受的数据字节数

调用此函数将保存接受到的数据到recv_buf指向的缓冲区。

此函数等同于调用`rt_spi_transfer_message()`传输一条消息，message参数配置如下：

```
    struct rt_spi_message msg;

    msg.send_buf   = RT_NULL;
    msg.recv_buf   = recv_buf;
    msg.length     = length;
    msg.cs_take    = 1;
    msg.cs_release = 1;
    msg.next       = RT_NULL;
```

注意

调用此函数将接受一次数据。开始接收数据时片选开始，函数返回时片选结束。

rt_spi_send_then_send()

函数原型:

```
rt_err_t rt_spi_send_then_send(struct rt_spi_device *device,
                               const void           *send_buf1,
                               rt_size_t             send_length1,
                               const void           *send_buf2,
                               rt_size_t             send_length2);
```

参数	描述
device	SPI总线设备句柄
send_buf1	发送缓冲区1数据指针
send_length1	发送缓冲区数据字节数
send_buf2	发送缓冲区2数据指针
send_length2	发送缓冲区2数据字节数

函数返回： 成功返回RT_EOK，否则返回错误码

此函数可以连续发送2个缓冲区的数据，忽略接收到的数据。发送send_buf1时片选开始，发送完send_buf2后片选结束。

此函数等同于调用`rt_spi_transfer_message()`传输2条消息，message参数配置如下：

```
    struct rt_spi_message msg1,msg2;

    msg1.send_buf   = send_buf1;
    msg1.recv_buf   = RT_NULL;
    msg1.length     = send_length1;
    msg1.cs_take    = 1;
    msg1.cs_release = 0;
    msg1.next       = &msg2;

    msg2.send_buf   = send_buf2;
    msg2.recv_buf   = RT_NULL;
    msg2.length     = send_length2;
    msg2.cs_take    = 0;
    msg2.cs_release = 1;
    msg2.next       = RT_NULL;
```

rt_spi_send_then_recv()

函数原型:

```
rt_err_t rt_spi_send_then_recv(struct rt_spi_device *device,
                               const void           *send_buf,
                               rt_size_t             send_length,
                               void                 *recv_buf,
                               rt_size_t             recv_length);
```

参数	描述
device	SPI总线设备句柄
send_buf	发送缓冲区数据指针
send_length	发送缓冲区数据字节数
recv_buf	接收缓冲区数据指针，spi是全双工的，支持同时收发
length	接收缓冲区数据字节数

函数返回： 成功返回RT_EOK，否则返回错误码

此函数发送第一条消息send_buf时开始片选，此时忽略接收到的数据，然后发送第二条消息，此时发送的数据为空，接收到的数据保存在recv_buf里，函数返回时片选结束。

此函数等同于调用`rt_spi_transfer_message()`传输2条消息，message参数配置如下：

```
    struct rt_spi_message msg1,msg2;

    msg1.send_buf   = send_buf;
    msg1.recv_buf   = RT_NULL;
    msg1.length     = send_length;
    msg1.cs_take    = 1;
    msg1.cs_release = 0;
    msg1.next       = &msg2;

    msg2.send_buf   = RT_NULL;
    msg2.recv_buf   = recv_buf;
    msg2.length     = recv_length;
    msg2.cs_take    = 0;
    msg2.cs_release = 1;
    msg2.next       = RT_NULL;
```

`rt_spi_sendrecv8()`和`rt_spi_sendrecv16()`函数是对此函数的封装，`rt_spi_sendrecv8()`发送一个字节数据同时收到一个字节数据，`rt_spi_sendrecv16()`发送2个字节数据同时收到2个字节数据。

4.4 SPI设备驱动应用

本文示例使用SSD1351显示图像信息，首先需要确定信息在显示器上的行列起始地址，调用`ssd1351_write_cmd()`向SSD1351发送指令，调用`ssd1351_write_data()`向SSD1351发送数据，源代码如下：

```
void set_column_address(rt_uint8_t start_address, rt_uint8_t end_address)
{
    ssd1351_write_cmd(0x15);              //   Set Column Address
    ssd1351_write_data(start_address);    //   Default => 0x00 (Start Address)
    ssd1351_write_data(end_address);      //   Default => 0x7F (End Address)
}
void set_row_address(rt_uint8_t start_address, rt_uint8_t end_address)
{
    ssd1351_write_cmd(0x75);              //   Set Row Address
    ssd1351_write_data(start_address);    //   Default => 0x00 (Start Address)
    ssd1351_write_data(end_address);      //   Default => 0x7F (End Address)
}
```

5 参考

本文所有相关的API

SPI设备驱动框架所有API	头文件
rt_spi_bus_register()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_bus_attach_device()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_configure ()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_send_then_send()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_send_then_recv()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_transfer()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_transfer_message()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_take_bus()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_release_bus()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_take()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_release()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_recv()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_send()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_sendrecv8()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_sendrecv16()	rt-thread\components\drivers\include\drivers\spi.h
rt_spi_message_append()	rt-thread\components\drivers\include\drivers\spi.h

示例代码相关API	位置
ssd1351_write_cmd()	drv_ssd1351.c
ssd1351_write_data()	drv_ssd1351.c
rt_hw_ssd1351_config()	drv_ssd1351.c

其他核心API详解

rt_spi_take_bus()

函数原型:

```
rt_err_t rt_spi_take_bus(struct rt_spi_device *device);
```

参数	描述
device	SPI设备句柄

函数返回： 成功返回RT_EOK，否则返回错误码

设备调用此函数可以占有SPI总线资源，其他设备则不能使用SPI总线。

rt_spi_release_bus()

函数原型:

```
rt_err_t rt_spi_release_bus(struct rt_spi_device *device);
```

参数	描述
device	SPI设备句柄

函数返回： 成功返回RT_EOK，否则返回错误码

设备调用`rt_spi_take_bus()`获取总线资源后需要调用此函数释放SPI总线资源，这样其他设备才能访问SPI总线。

rt_spi_take()

函数原型:

```
rt_err_t rt_spi_take(struct rt_spi_device *device);
```

参数	描述
device	SPI设备句柄

函数返回： 返回0

调用此函数则片选开始。

rt_spi_release()

函数原型:

```
rt_err_t rt_spi_release(struct rt_spi_device *device);
```

参数	描述
device	SPI设备句柄

函数返回： 返回0

调用此函数则片选结束。

rt_spi_message_append()

函数原型:

```
rt_inline void rt_spi_message_append(struct rt_spi_message *list,
                                     struct rt_spi_message *message)
```

参数	描述
list	消息链表指针
message	消息指针

函数返回： 无返回值

调用此函数向消息链表list里面插入一条消息message。

讨论和反馈

欢迎登陆RT-Thread开发者社区进行交流

参考文献

RT-Thread编程手册

关注RT-Thread公众号

扫一扫关注RT-Thread微信公众号