AN0102
入门指南

AT32 MCU SPI/I²S入门指南

前言

SPI(Serial Peripheral interface)是一种由 Motorola 最先推出的同步串行传输协议。SPI 是一种高速、全双工、同步的通信总线，使用简单高效。
I²S (Inter-IC Sound)总线，又称集成电路内置音频总线，是 Philips 为数字音频设备之间的音频数据传输而制定的一种总线标准。I²S 是一种同步、半双工的通信总线。AT32 部分型号增加了全双工功能及对应引脚，具体请参考本文 I²S 全双工章节。
AT32 控制器的大部分型号都是 SPI 和 I²S 共用 SPI 接口，根据软件编程配置来选择 SPI 还是 I²S 功能。本文分别介绍了 SPI 和 I²S 的几种模式/协议，配置流程和使用案例。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本 BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32F 系列

目录

表目录

图目录

1 SPI 接口概述

AT32 的 SPI 接口提供软件编程配置选项，根据软件编程配置方式不同，可以分别作为 SPI 和 I²S 使用。本文将分 SPI 和 I²S 分别介绍 SPI 接口作 SPI 或 I²S 的功能特性以及配置流程。

AT32 SPI特点：

- 可编程配置的全双工或半双工通信
  --全双工同步通信
  --半双工同步通信（可以根据软件编程配置选择传输方向：发送或接收）
- 可编程配置主/从模式
- 可编程配置的 CS 模式：硬件/软件 CS 模式
- 可编程配置的时钟极性和相位
- 可编程配置的数据传输顺序（MSB/LSB）
- 可编程配置的错误中断标志（CS脉冲异常，接收溢出错误，主模式错误，CRC校验错误）
- 数据接收/发送均支持DMA
- 兼容TI的SSP协议（即TI模式）

AT32 I²S特点：

- 可编程配置的操作模式
  --从设备发送
  --从设备接收
  --主设备发送
  --主设备接收
- 可编程配置的时钟极性
- 可编程配置的时钟频率（8KHz 到 192KHz）
- 可编程配置的数据位数（16 位，24 位，32 位）
- 可编程配置的声道位数（16 位，32 位）
- 可编程配置的音频协议
  --I²S飞利浦标准
  --高字节对齐标准（左对齐）
  --低字节对齐标准（右对齐）
  --PCM标准（长帧/短帧）
- 支持 I²S 全双工
- 支持 DMA 传输
- 在通讯期间可提供频率固定为 256 倍 Fs（音频采样频率）的外设主时钟

2 SPI 功能介绍

本章主要介绍 SPI 基本功能，以及 AT32 SPI 的各种附加可配置选项。

2.1 SPI 硬件接口

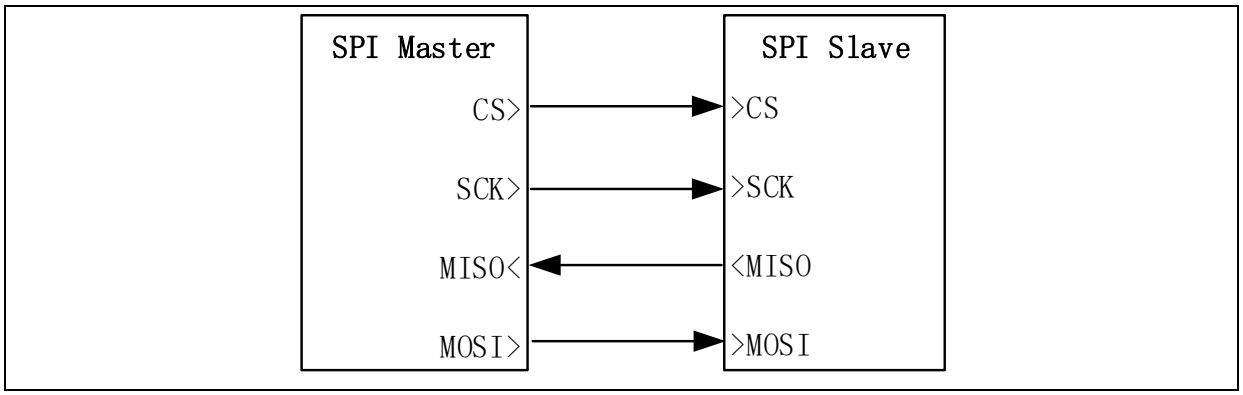
SPI 接口定义如下：

- CS：片选引脚（可选）。通常由主机输出，从机输入。后续 CS 管理章节作详细介绍。
- SCK：时钟引脚。由主机输出，从机输入。
- MISO：主收从发引脚。主机则作为接收数据引脚，从机则做为发送数据引脚。
- MOSI：主发从收引脚。主机则作为发送数据引脚，从机则作为接收数据引脚。

以下介绍几种常见的 SPI 通信接线方式。

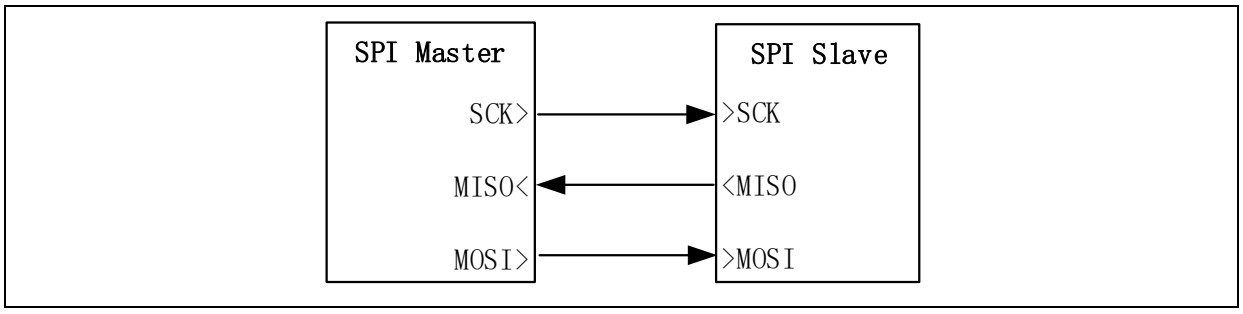
1）如下图 1，是单主单从、全双工、CS 硬件模式的应用接线示意图。

图 1 SPI 单主单从、全双工、硬件 CS 管理接线示意



2）如下图 2，是单主单从、全双工、CS 软件模式的应用接线示意图。在 CS 软件管理模式下，无需使用 CS 引脚，主/从机的 CS 引脚都可以释放给其他应用使用。此时，从机对 SWCSIL 位清 0 或置 1 等效于“硬件 CS 模式”下的 CS 引脚输入低电平和高电平。主机则必须将 SWCSIL 位置 1，以保证正确的处于主机模式。

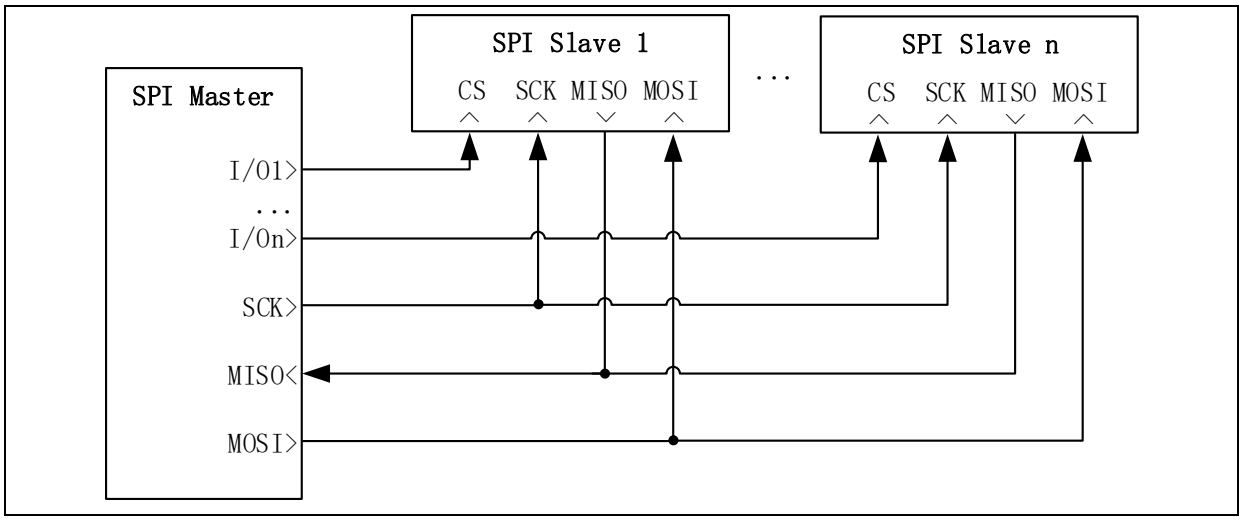
图 2 SPI 单主单从、全双工、软件 CS 管理接线示意



3）如下图 3，是单主多从全双工接线示意。

主机使用普通 I/O 口 1~n 输出以驱动从机 1~n 的片选 CS，主机的 SCK,MISO,MOSI 和从机的 SCK,MISO,MOSI 引脚一对多连接。此时 SPI 主机可以通过 I/O1~n 的选择与多个从机进行分时通信。此时从机必须使用硬件 CS 模式。主机可以使用软件 CS 模式，主机 CS 对应的引脚可以释放给其他应用使用。

图 3 SPI 单主多从、全双工接线示意

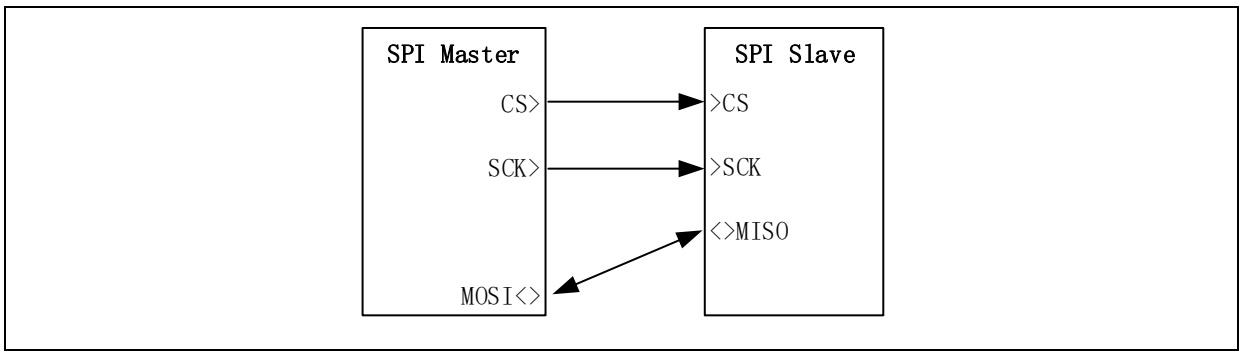


4）如下图 4，AT32 的 SPI 增加了 SPI 半双工模式，此模式下只需要使用 3 线即可通信。

主机使用 MOSI 进行数据收/发。当主机配置为半双工发送模式（SLBEN=1，SLBTD=1）时，主机 MOSI 为输出引脚，用于发送数据；当主机配置为半双工接收模式（SLBEN=1，SLBTD=0）时，主机 MOSI 为输入引脚，用于接收数据。

从机使用 MISO 进行数据收/发。当从机配置为半双工发送模式（SLBEN=1，SLBTD=1）时，从机 MISO 为输出引脚，用于发送数据；当从机配置为半双工接收模式（SLBEN=1，SLBTD=0）时，从机 MISO 为输入引脚，用于接收数据。

图 4 SPI 单主单从、半双工接线示意



2.2 SPI 数据接收和发送

如下图 5，用户软件程序可以读和写数据寄存器，其他部分全由硬件自动完成。

以全双工，硬件 CS 管理，8bit 数据格式为例。收发一笔数据的流程如下：

作为 SPI 主机时：

软件使能 SPI（SPIEN=1）后，硬件控制 CS 引脚输出低电平，以片选从机。之后软件往数据寄存器写入数据，如下图 5 的“软件写数据寄存器”处，相当于往发送缓冲器写入待发送数据，之后硬件会自动将发送缓冲器的待发送数据搬运到移位寄存器。同时将发送缓冲器空标志置 1（TDBE=1），以提醒软件又可以往数据寄存器写入下一笔待发送的数据。之后硬件立即开始在 SCK 引脚上输出时钟信号，并在 MOSI 引脚上发出待发送数据的第一 bit 数据。之后硬件从 MISO 引脚上读取待接收数据的第一 bit 数据并放入移位寄存器中（移位寄存器由于之前发送了 1bit 数据，此时正好有一 bit 位置可以存放收到的数据）。之后硬件会通过移位寄存器自动发送和接收剩下的 7bit 数据。之后硬件会将移位寄存器中接收到的 8bit 数据搬运到接收缓冲器中，并将接收缓冲器满标志置 1（RDBF=1），以提醒

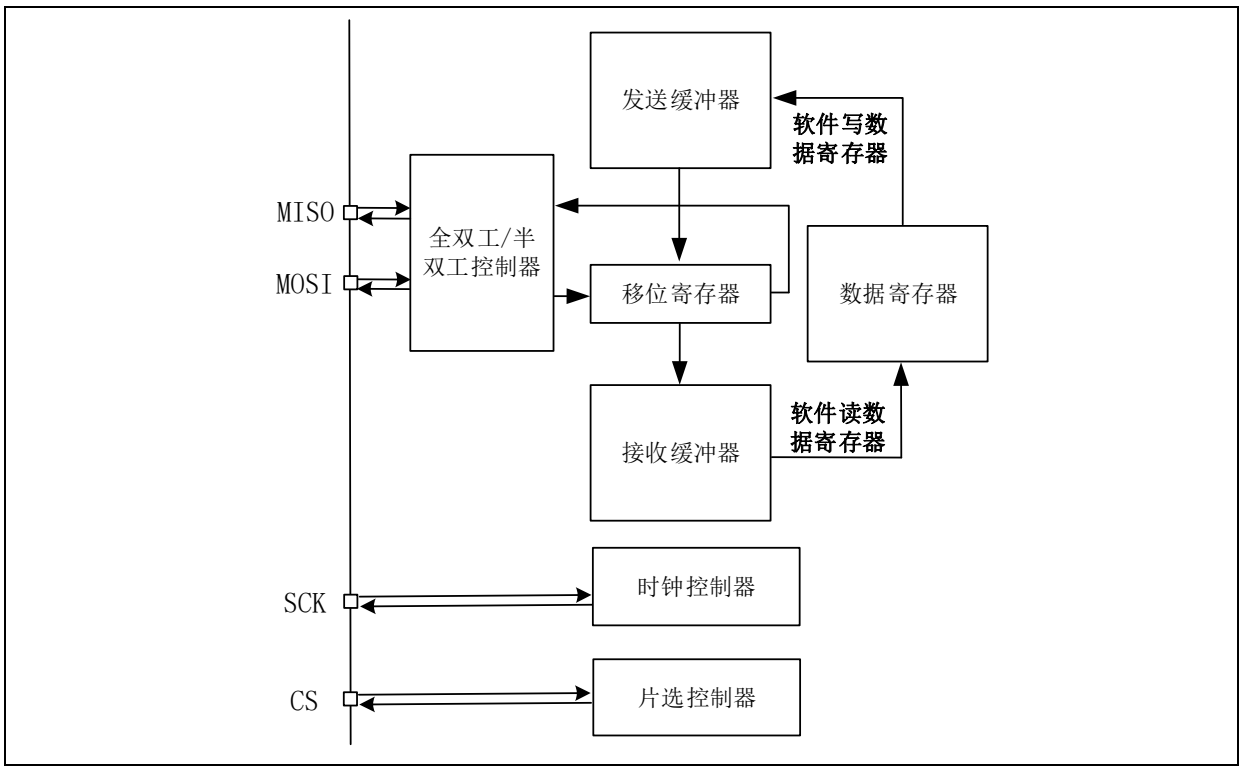
软件可以通过读取数据寄存器来读取刚刚接收到的这笔数据。

作为 SPI 从机时：

软件使能 SPI（SPIEN=1）后，软件往数据寄存器写入数据，相当于往发送缓冲器写入待发送数据，然后硬件会将发送缓冲器空标志清 0（TDBE=0），以提醒软件此时发送缓冲器已有待发送数据了。之后硬件检测 CS 引脚和 SCK 引脚，等待来自主机的片选和时钟信号。检测到片选和时钟信号之后，硬件自动将发送缓冲器的待发送数据搬运到移位寄存器并在 MISO 引脚上发出待发送数据的第一 bit。同时将发送缓冲器空标志置 1（TDBE=1），以提醒软件又可以往数据寄存器写入下一笔待发送的数据。之后硬件从 MOSI 引脚上读取待接收数据的第一 bit 数据并放入移位寄存器中（移位寄存器由于之前发送了 1bit 数据，此时正好有一 bit 位置可以存放收到的数据）。之后硬件根据在 SCK 上的时钟信号，会在 MISO 和 MOSI 引脚上发送和接收剩下的 7bit 数据。之后硬件会将移位寄存器中接收到的 8bit 数据搬运到接收缓冲器中，并将接收缓冲器满标志置 1（RDBF=1），以提醒软件可以通过读取数据寄存器来读取刚刚接收到的这笔数据。

也可以参考下一小节“SPI 时序”来理解 SPI 收发逻辑。

图 5 SPI 数据接收/发送框图



2.3 SPI 时序

本节介绍 SPI 通信时序。包括全双工和半双工的主/从通信时序。

2.3.1 SPI 全双工时序

以全双工、硬件 CS 管理、单主单从通信为例。

其中主机端相关配置如下：

MSTEN=1: 设备为主机；

SLBEN=0:全双工模式；

CLKPOL=0: SCK 空闲输出低电平；

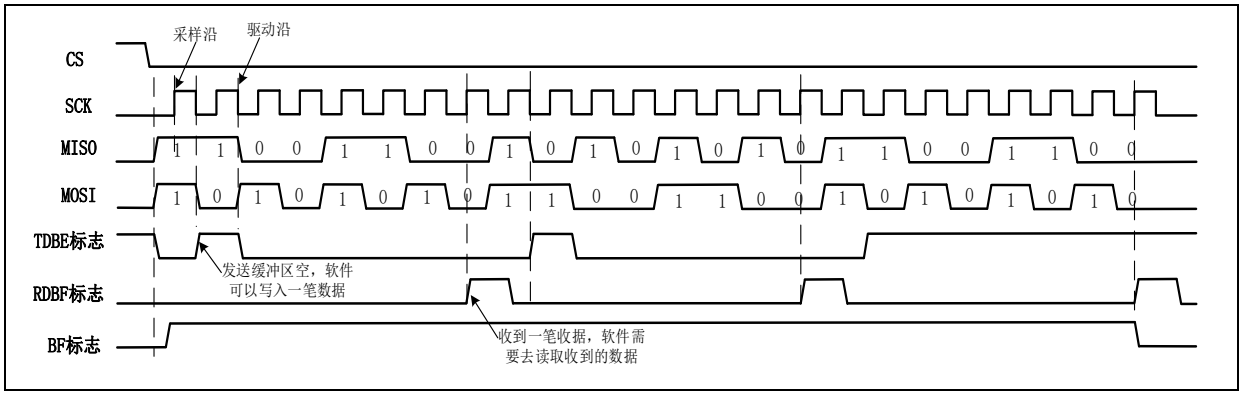
CLKPHA=0: SCK 第一个边沿开始进行数据捕获；

FBN=0: 帧位个数为 8bit；

SWCSEN=0，HWCSOE=1：使用硬件 CS 管理；

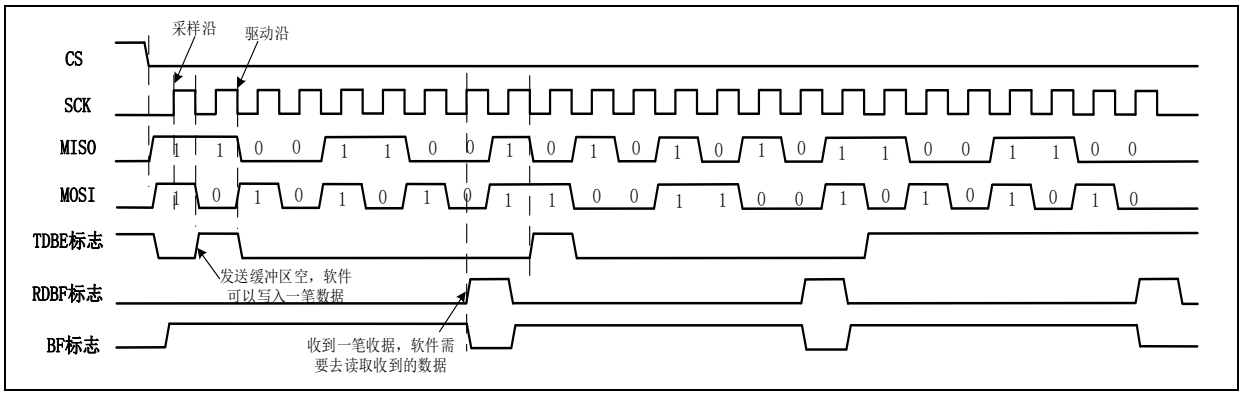
主机发送数据（MOSI）：0xAA,0xCC,0xAA；
从机发送数据(MISO)：0xCC,0xAA,0xCC。
主机端时序见下图 6：

图 6 硬件 CS 管理-全双工-主机通信时序



其中从机端相关配置如下：
MSTEN=0: 设备为从机
SLBEN=0:全双工模式；
CLKPOL=0: SCK 空闲低电平
CLKPHA=0: SCK 第一个边沿进行数据捕获
FBN=0: 帧位个数为 8bit
SWCSEN=0：使用硬件 CS 管理
主机发送数据(MOSI)：0xAA,0xCC,0xAA
从机发送数据(MISO)：0xCC,0xAA,0xCC
从机端时序见下图 7：

图 7 硬件 CS 管理-半双工-从机通信时序

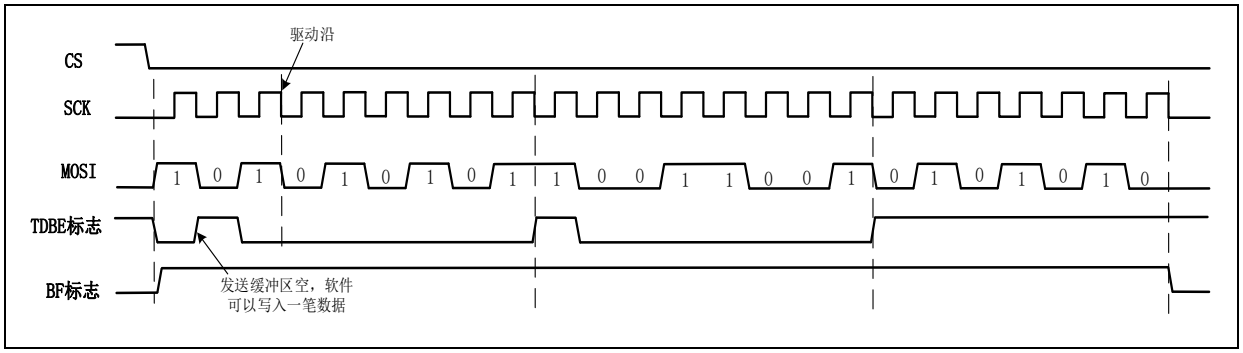


2.3.2 SPI 半双工时序

半双工时序下，有主机发送、从机接收、主机接收、从机发送 4 种模式。
1） 以半双工、硬件 CS 管理、主机发送-从机接收通信为例。
其中主机端相关配置如下：
MSTEN=1: 设备为主机；
SLBEN=1:使能单线双向半双工模式；
SLBTD=1:发送模式；
CLKPOL=0: SCK 空闲输出低电平；
CLKPHA=0: SCK 第一个边沿开始进行数据捕获；

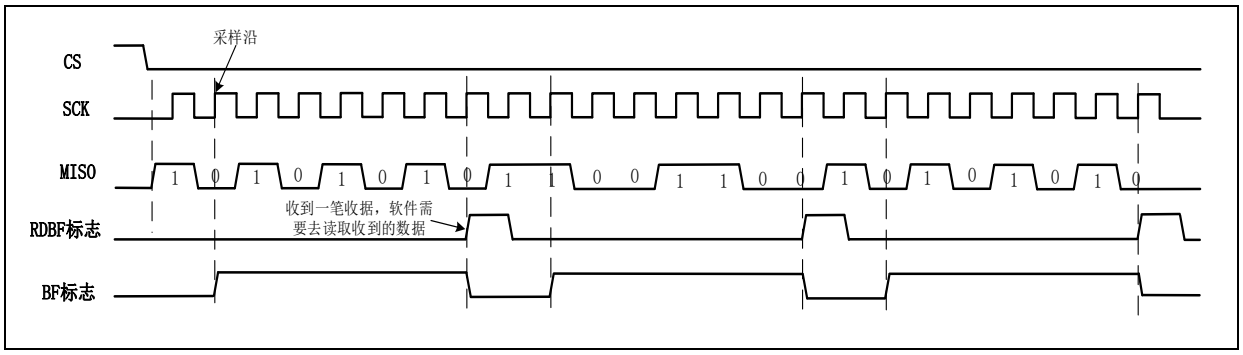
FBN=0: 帧位个数为 8bit；
SWCSEN=0，HWCSOE=1：使用硬件 CS 管理；
主机发送数据：0xAA,0xCC,0xAA；
主机端时序见下图 8：

图 8 硬件 CS 管理-半双工-主发时序



其中从机端相关配置如下：
MSTEN=0: 设备为从机；
SLBEN=1:使能单线双向半双工模式；
SLBTD=0:接收模式；
CLKPOL=0: SCK 空闲低电平；
CLKPHA=0: SCK 第一个边沿进行数据捕获；
FBN=0: 帧位个数为 8bit；
SWCSEN=0：使用硬件 CS 管理；
从机接收数据：0xAA,0xCC,0xAA；
从机端时序见下图 9：

图 9 硬件 CS 管理-半双工-从收时序



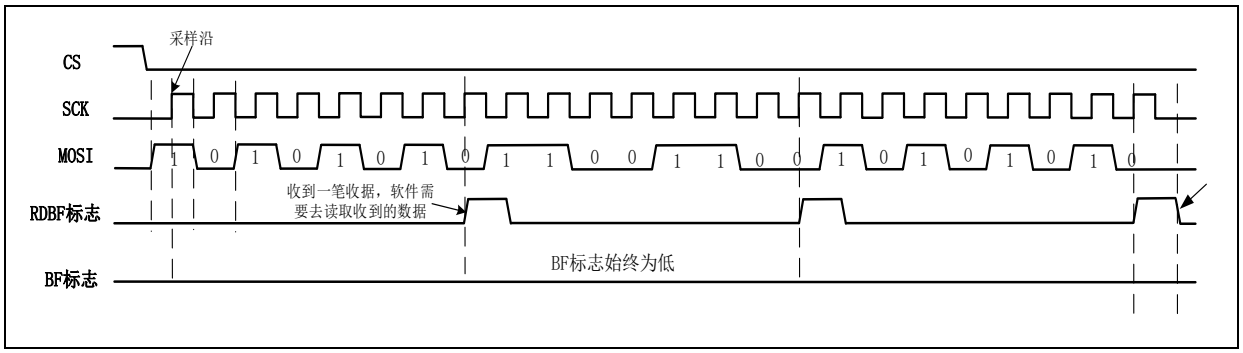
2） 以半双工、硬件 CS 管理、主机接收-从机发送通信为例。
其中主机端相关配置如下：
MSTEN=1: 设备为主机；
SLBEN=1:使能单线双向半双工模式；
SLBTD=0:接收模式；
CLKPOL=0: SCK 空闲输出低电平；
CLKPHA=0: SCK 第一个边沿开始进行数据捕获；
FBN=0: 帧位个数为 8bit；
SWCSEN=0，HWCSOE=1：使用硬件 CS 管理；
主机接收数据：0xAA,0xCC,0xAA；

主机端时序见下图 10：

此模式下，较为特别的是,SPI 主机一旦使能(SPIEN=1)，就会连续不断的输出时钟。因此，主机在接收完需要接收的数据后，需要关闭 SPI。在本例中，接收完连续的 3 笔数据后，也就是下图 10 的箭头处，需要关闭 SPI，输出多余的时钟。

另外，如下图 10，此模式下， BF 标志始终为低。

图 10 硬件 CS 管理-半双工-主收时序



其中从机端相关配置如下：

MSTEN=0: 设备为从机；

SLBEN=1:使能单线双向半双工模式；

SLBTD=1:发送模式；

CLKPOL=0: SCK 空闲低电平；

CLKPHA=0: SCK 第一个边沿进行数据捕获；

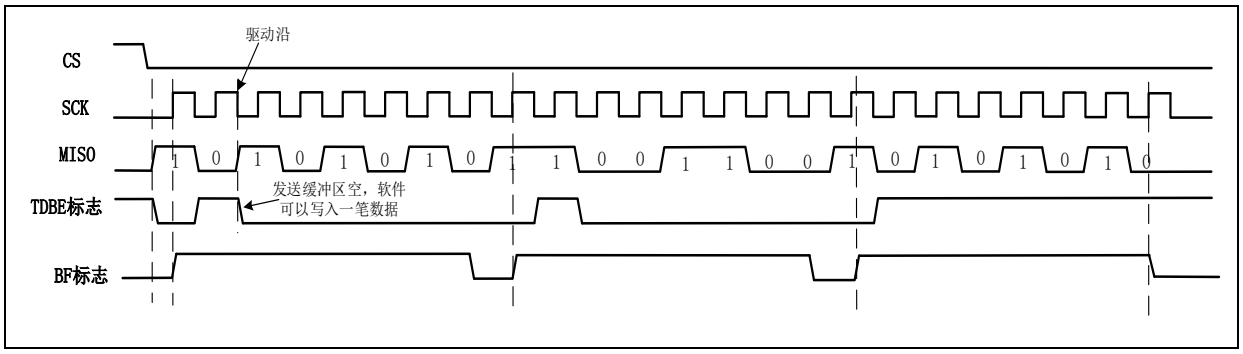
FBN=0: 帧位个数为 8bit；

SWCSEN=0：使用硬件 CS 管理；

从机发送数据：0xAA,0xCC,0xAA；

从机端时序见下图 11：

图 11 硬件 CS 管理-半双工-从发时序



2.4 SPI 数据收发方式

SPI 的接收和发送如前文所述，是通过操作数据寄存器来完成的。而根据操作数据寄存器的方法，SPI 有以下 3 种数据接收发送方式。

2.4.1 轮询方式

轮询方式不需要使能 SPI 中断和 DMA。只需要在 main 函数中不断读取 STS 寄存器，并判断 TDBE 和 RDBF 标志是否置起，以确定何时向数据寄存器写入待发送数据，何时从数据寄存器读取接收到的数据。比较耗费 CPU 资源，不建议使用此方式。具体请参考“案例 1-- SPI 全双工轮询方式通信”一节。

2.4.2 中断方式

中断方式需要开启“发送数据缓冲器空中断使能”和“接收数据缓冲器满中断使能”，即设置TDBEIE=1，RDBFIE=1。并配置和使能 SPI 中断。之后在 SPI 中断里进行数据的发送和接收，避免了轮询方式的等待耗时，占用 CPU 资源相对较少。具体请参考“案例 4-- SPI 半双工中断方式通信”或“案例 5-- SPI 半双工中断方式通信--加收发切换”一节。

2.4.3 DMA 方式

DMA 方式需要开启 SPI 的 DMA 接收/发送功能（DMAREN=1， DMATEN=1）。并配置 DMA 通道映射到 SPI 和使能 DMA。之后的接收和发送均由 DMA 完成，收/发过程不再需要软件参与，不占用 CPU 资源。具体请参考“案例 2-- SPI 全双工 DMA 方式通信”一节。

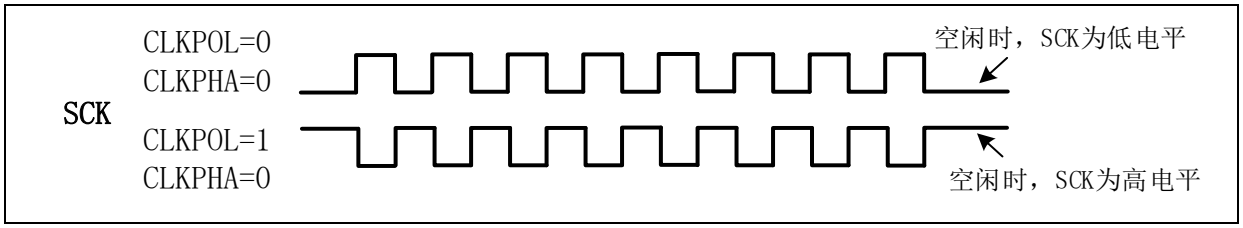
2.5 时钟控制器

SPI 协议采用同步传输。作主机时，需要时钟控制器产生通信时钟用于 SPI 接口的数据收发，并且需要将该通信时钟通过 SCK 引脚输出给从机，用于从机的数据收发；作从机时，需要外部提供通信时钟，从 SCK 引脚输入到 SPI 接口内部作为通信时钟使用。对用户而言，可软件配置的主要有极性、相位、分频系数三个参数。

2.5.1 极性

SPI 时钟极性，即空闲时 SCK 引脚输出的电平。通过配置 CLKPOL 位来选择 SPI 时钟极性。
如下图 12：CLKPOL=0 时，SCK 空闲为低电平；CLKPOL=1 时，SCK 空闲为高电平。

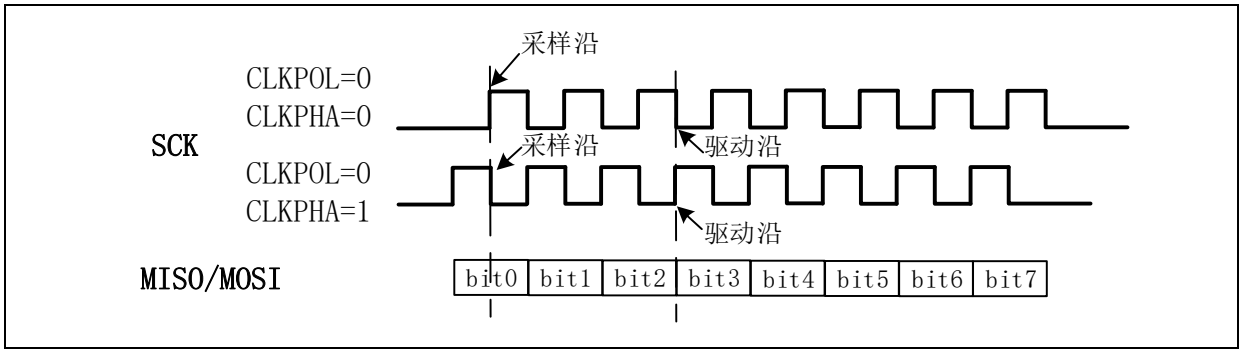
图 12 时钟极性对比



2.5.2 相位

SPI 时钟相位，即 SPI 数据捕获边缘。通过配置 CLKPHA 位来选择 SPI 时钟极性。
如下图 13：
CLKPHA=0 时，第一个边沿为捕获采样边沿。在本例中（CLKPOL=0），也就是上升沿为采样沿；相对的，本例中下降沿为驱动沿。
CLKPHA=1 时，第二个边沿为捕获采样边沿。在本例中（CLKPOL=0），也就是下降沿为采样沿；相对的，本例中上升沿为驱动沿。

图 13 时钟相位对比



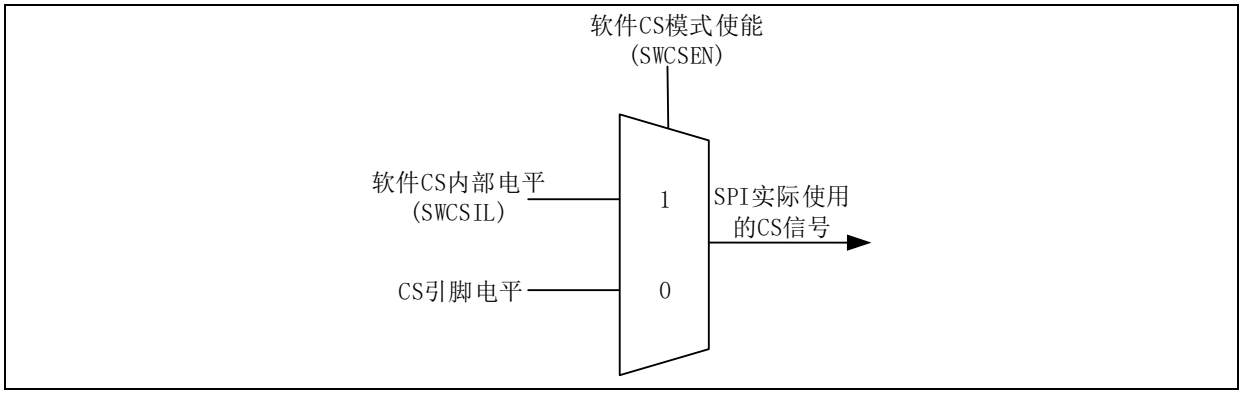
2.5.3 分频系数

SPI 时钟是从 APB 时钟分频得到的，通过配置 MDIV[3：0]和 MDIV3EN 来选择需要的分频系数，以确定 SPI 时钟。分频系数仅主机配置有效。从机需要外部提供时钟，因此此项配置无效。

2.6 CS 管理

SPI CS 包含硬件和软件管理两种模式，又根据主机/从机的不同，有不同的配置和应用场景。
在 CS 为输入模式时（HWCSOE=0），如下图 14，主机和从机均通过 SWCSEN 位来选择 SPI 实际使用的 CS 信号来自软件设置（SWCSIL）还是外部 CS 引脚电平。
在 CS 为输出模式时（HWCSOE=1），软件 CS 模式相关配置无效。

图 14 SPI 数据接收/发送框图



以下分别列出了主机和从机在不同的 CS 管理模式下的状态和应用场景。

主机 CS 管理：

下表 1 列出了几种主机 CS 管理的配置及应用场景。

表 1 SPI 主机 CS 管理配置

主机 CS 管理				
序号	SWCSEN （软件 CS 管理使能）	SWCSIL （软件 CS 设置）	CS 引脚输入电平 （硬件 CS 输入）	HWCSOE （硬件 CS 输出使能）
1	0	--	1	0
	主机状态：主机可以正常通信。 应用场景：用于多主机的 SPI 通信网络。此时 SPI 总线上没有其他主机使能，当前 SPI 可以作为主机进行通信。			
2	0	--	0	0
	主机状态：产生主模式错误。硬件自动置位主模式错误标志位（MMERR=1），退出主机模式（MSTEN=0），并关闭 SPI（SPIEN=0）。后续恢复操作请参考“MMERR：主模式错误”一节。 应用场景：用于多主机的 SPI 通信网络。此时 SPI 总线已有其他主机使能了，因此当前 SPI 需要退出主机状态。			
3	0	--	--	1
	主机状态：主机可以正常通信，CS 引脚为从机提供片选信号。 应用场景：单主单从的通信配置。			
4	1	1	--	--
	主机状态：主机可以正常通信。CS 引脚不提供片选状态。 应用场景：单主单从且从机也使用软件 CS 管理；单主多从，主机通过多个普通 I/O 驱动多个从机 CS 片选。			

注：“--”表示此项配置无效，建议保持默认配置。

从机 CS 管理：

下表 2 列出了从机 CS 管理的几种配置和应用场景。其中，从模式下 HWCSOE 配置无效，该位只在主模式下有效。

表 2 SPI 从机 CS 管理配置

从机 CS 管理				
序号	SWCSEN（软件 CS 管理使能）	SWCSIL（软件 CS 设置）	CS 引脚输入电平（硬件 CS 输入）	HWCSOE（硬件 CS 输出使能）
1	0	--	0	--
	从机状态：从机被使能，可以正常通信。 应用场景：单主单从/单主多从。			
2	0	--	1	--
	从机状态：从机未被选通，不能通信 应用场景：单主单从/单主多从。			
3	1	0	--	--
	从机状态：从机被选通，可以正常通信 应用场景：单主单从。			
4	1	1	--	--
	从机状态：从机未被选通，不能通信 应用场景：单主单从。			

注：“--”表示此项配置无效，建议保持默认配置。

2.7 CRC 校验

AT32 SPI 具有独立的发送和接收 CRC 计算单元，用户可通过软件配置使能此功能。使能 CRC 校验之后：硬件会对发送的数据进行 CRC 计算，并将计算得到的 CRC 校验码放在 SPI_TCRC 寄存器中， CRC 校验码紧接在数据之后发送；且硬件会在接收一笔数据时对接收到的数据进行 CRC 计算，并将计算得到的 CRC 校验码放在 SPI_RCRC 寄存器中，硬件会在一笔连续的数据接收完成后将最后接收到的 CRC 校验码与 SPI_RCRC 寄存器中的校验码进行比较，如果不符，CRC 校验错误位（CCERR）会置起，若使能了错误中断（ERRIE=1），将会产生中断。

另外，SPI 通信方式使用 DMA 和使用轮询/中断方式时，CRC 功能的软件操作步骤有区别，如下：

轮询/中断方式：

① 配置 CRC 多项式：配置 SPI_CPOLY 寄存器和需要通信的 SPI 一致（默认值为 0x0007）；

② 使能 CRC 功能：CCEN=1；

③ 使能 SPI：SPIEN=1；

④ 发送待发送的数据 data_1~data_n；

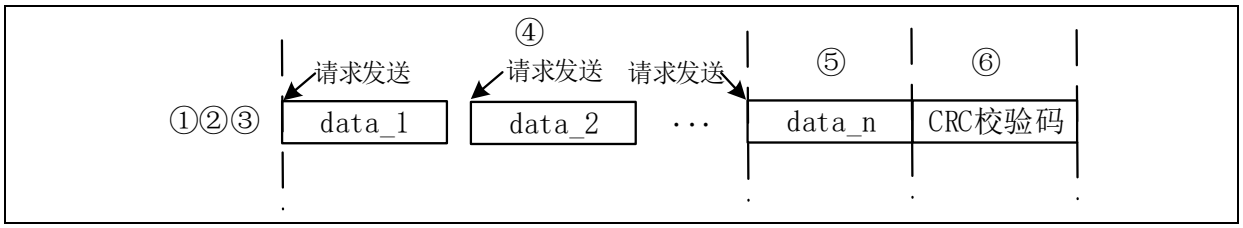
⑤ 请求发送 CRC 校验码：NTC=1。

注：NTC 置 1 的时间参考下图 15 的⑤段，需要在最后一笔数据（data_n）写入数据寄存器到发送完成之间。实际编写软件代码时，建议在 data_n 写入数据寄存器之后立即将 NTC 置 1。

⑥ 硬件自动在 data_n 发送完成后立即发送 CRC 校验码（此时无需软件参与）。

轮询方式使用 CRC 功能可参考“案例 6-- SPI CRC 功能使用”。

图 15 CRC 使用流程（轮询/中断方式）



DMA 方式：

SPI 在使用 DMA 方式通信时，不需要软件代码去设置请求发送 CRC 校验码，即不需要去置位 NTC 位。具体步骤如下：

① 配置 DMA：请参考“案例 2-- SPI 全双工 DMA 方式通信”的 DMA 配置部分；

② 配置 CRC 多项式：配置 SPI_CPOLY 寄存器和需要通信的 SPI 一致（默认值为 0x0007）；

③ 使能 CRC 功能：CCEN=1；

④ 使能 SPI：SPIEN=1；

⑤ 数据及 CRC 接收完成后需关闭 SPI 和 DMA，之后再开启，准备下一次通信。

2.8 TI 模式（TI SSP 协议）

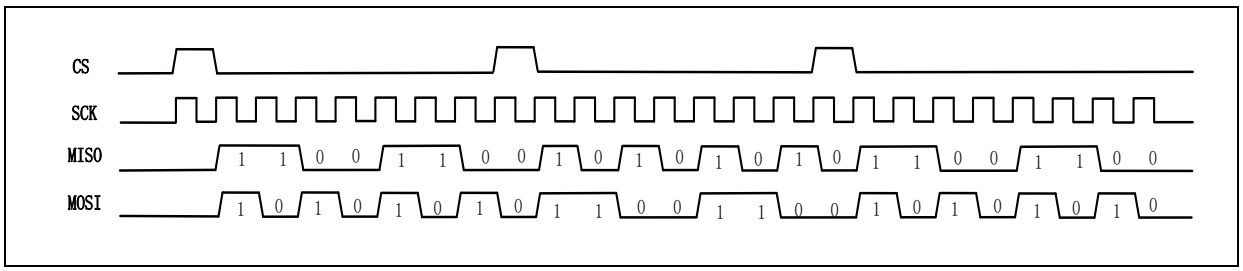
AT32 SPI 接口支持 TI SSP 协议，即 TI 模式。用户可以通过将 TIEN 位置 1 来使能 TI 模式。

使能 TI 模式后，SPI 接口将按照 TI 协议要求产生时钟 SCK，和片选信号 CS。也就是 CS 软/硬件管理相关控制位、时钟极性/相位相关控制位配置无效，请保持默认设置。使能 TI 模式后，帧格式（8/16bit）配置、CRC 校验、DMA 等功能仍然可以使用。

TI 模式下，连续和不连续通信稍有区别。连续通信时，只有第一笔数据发送时有一个 dummy CLK；不连续通信时，每笔数据都有一个 dummy CLK。参考下图 16 和图 17。

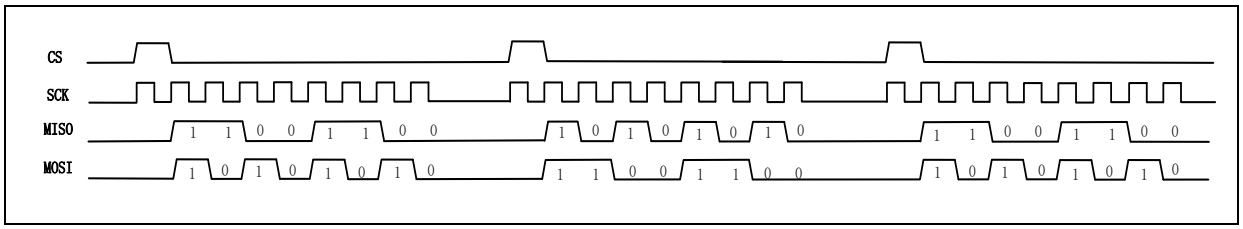
TI 模式连续通信时序图见下图 16。主机发送数据（MOSI）：0xAA,0xCC,0xAA；从机发送数据（MISO）：0xCC,0xAA,0xCC。

图 16 TI 模式连续通信时序图



TI 模式不连续通信时序图见下图 17。主机发送数据（MOSI）：0xAA,0xCC,0xAA；从机发送数据（MISO）：0xCC,0xAA,0xCC。

图 17 TI 模式不连续通信时序图



注：AT32 只有部分型号支持 TI 模式。AT32F425/F435/F437 等型号支持 TI 模式，AT32F403/F403A/F407/F413/F415等型号不支持TI模式。具体请参考各型号的RM。

2.9 SPI 错误

AT32 的 SPI 有 4 种错误标志。这 4 种错误标志共用一个错误中断使能位（ERRIE）。也就是当 ERRIE=1 时，产生以下 4 种错误的任一错误就会进入 SPI 错误中断。

2.9.1 CSPAS-- CS 脉冲异常置位标志

在 TI 模式下，当 SPI 作从机使用时，SPI 会在数据传输期间侦测非预期的 CS 脉冲，当发生 CS 脉冲异常置位错误，CSPAS 被硬件置 1，如果开启了错误中断使能（ERRIE=1），则会产生错误中断。之后可以通过软件读 SPI_STS 清除该位。当侦测到 CS 脉冲异常时，为避免数据错误，软件应当关闭 SPI 从机，重新配置 SPI 主机，再打开 SPI 从机以重新开始通信。

注：仅在 TI 模式下，CSPAS 错误标志才有效。

2.9.2 ROERR--接收器溢出错误标志

当 SPI 数据寄存器已暂存一笔待读数据时，又收到一笔新数据，则会发生接收器溢出错误，ROERR 被硬件置 1，如果开启了错误中断使能（ERRIE=1），则会产生错误中断。发生溢出错误之后，SPI_DT 寄存器存放的是最早收到的那笔数据，发生溢出错误之后收到的数据都被丢弃。

之后依次读取 SPI_DT 寄存器和 SPI_STS 寄存器可清除该标志位。

2.9.3 MMERR--主模式错误标志

参考“CS 管理”一节，当 SPI 主机为 CS 输入模式且 CS 引脚输入低电平时，会产生主模式错误，MMERR 位会被硬件置 1，且硬件会强制 SPI 进入从机模式（MSTEN 清 0）并关闭 SPI（SPIEN 清 0）。如果开启了错误中断使能（ERRIE=1），则会产生错误中断。

清除 MMERR 标志位并从错误状态恢复请严格按照以下步骤执行：

① 拉高 CS 引脚电平；

② 执行一次对 SPI_STS 寄存器的读或写操作；

③ 执行一次对 SPI_CTRL1 寄存器的写操作；（不需要改变 SPI_CTRL1 的值，只需要写这个动作）

④ 之后硬件会自动清除 MMERR 标志（此步骤不需要软件代码参与）；

⑤ 之后软件可根据需要重新配置主/从模式和使能 SPI。

2.9.4 CCERR--CRC 校验错误标志

当使能了 CRC 功能后，如果接收到的 CRC 校验码（发送方发送的校验码）和 SPI_RCRC 的校验码（接收方根据接收到的数据计算的校验码）不符，则硬件将 CCERR 位置 1，如果开启了错误中断使能（ERRIE=1），则会产生错误中断。

由于发生了 CRC 校验错误，软件程序应当丢弃之前接收到的一笔数据，并重新通信。软件对 CCERR 位写 0 可清除该标志位。

2.10 SPI 中断

如下图 18，SPI 有一个全局中断向量。SPI 中断有三个中断源：接收缓冲器满、发送缓冲器空、通信错误（详见上一节“SPI 错误”）。这三个中断源分别有对应的使能位。

下图中，3 个中断使能位定义如下：

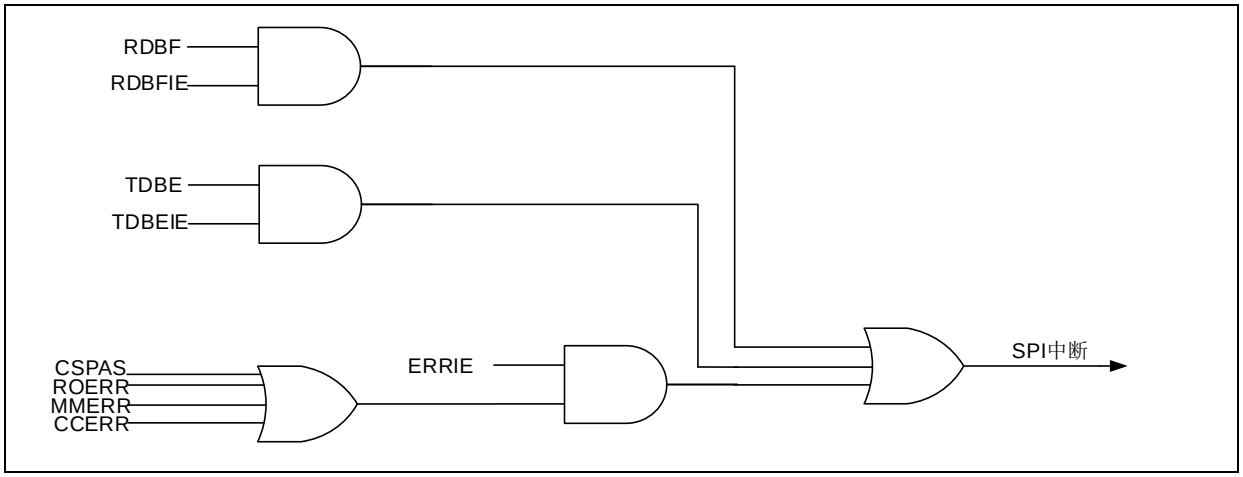
- RDBFIE：接收数据缓冲器满中断使能
- TDBEIE：发送数据缓冲器空中断使能
- ERRIE: 错误中断使能

下图中，5 个标志位定义如下：

- ROERR：接收器溢出错误
- MMERR：主模式错误
- CCERR：CRC 校验错误

- RDBF：接收数据缓冲器满
- TDBE：发送数据缓冲器空

图 18 SPI 中断示意



3 I²S 功能介绍

3.1 I²S 硬件接口

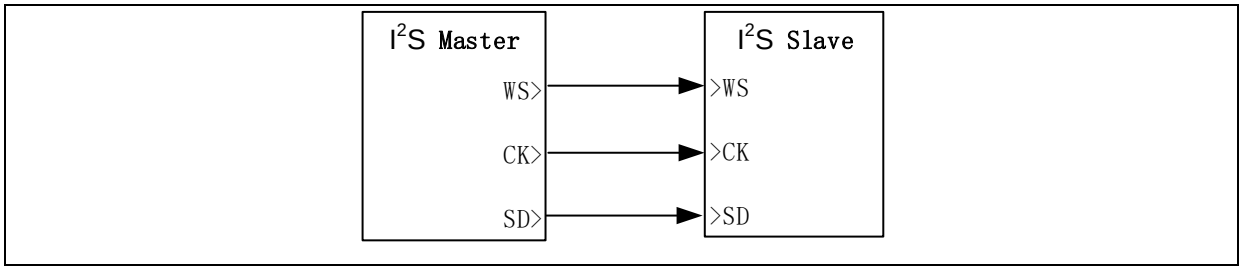
I²S 接口定义如下：

- WS：声道选择。由主机输出，从机输入。
- CK：时钟。由主机输出，从机输入。
- MCK：主时钟输出（可选）。在通讯期间可提供频率固定为 256 倍 Fs 的外设主时钟，仅在作为主机时有效。
- SD：数据。主机/从机接收或发送数据均通过这个引脚。

以下是常见的 I²S 通信接线方式。

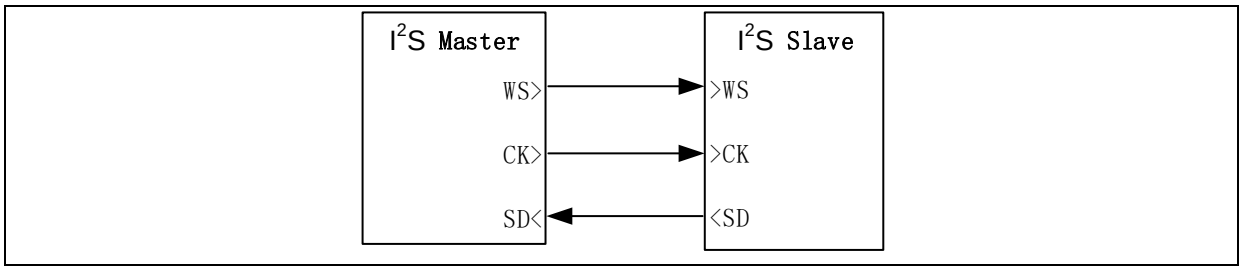
如下图 19，是 I²S 主发从收的接线示意图。

图 19 I²S 主发从收通信接线示意



如下图 20，是 I²S 主收从发的接线示意图。

图 20 I²S 主收从发通信接线示意



3.2 I²S 数据接收和发送

I²S 数据收发模块和上文的 SPI 共用数据寄存器和 DMA 映射，如上图 5 中的接收/发送缓冲器、移位寄存器也都是共用的，因此软件代码操作类似，请参考下文的 I²S 案例一节。不过 SPI 默认是全双工，而 I²S 默认是半双工。AT32 也为用户提供 I²S 全双工功能，后文 I²S 全双工一节详细介绍。

3.3 I²S 音频协议

AT32 的 I²S 支持以下 4 种标准协议：

- 飞利浦标准：STDSLE=0b00；
- 高字节对齐标准（左对齐）：STDSLE=0b01；
- 低字节对齐标准（右对齐）：STDSLE=0b10；
- PCM 标准：STDSLE=0b11。

  其中 PCM 标准又分为：
  - PCM 长帧同步：PCMFSSEL=1；
  - PCM 短帧同步：PCMFSSEL=0。

下图 21 在“16 位数据-32 位声道格式”，I²S 时钟极性为低（I2SCLKPOL=0）条件下对比了几种音

频标准一帧数据的通信时序。

飞利浦模式下：WS 为低电平表示正在传输的声道为左声道，WS 为高电平表示正在传输的声道为右声道。数据左对齐，即高位对齐，低位的 16 个 bit 被硬件填充为 0。

MSB 标准下：WS 为高电平表示正在传输的声道为左声道，WS 为低电平表示正在传输的声道为右声道。数据左对齐，即高位对齐，低位的 16 个 bit 被硬件填充为 0。

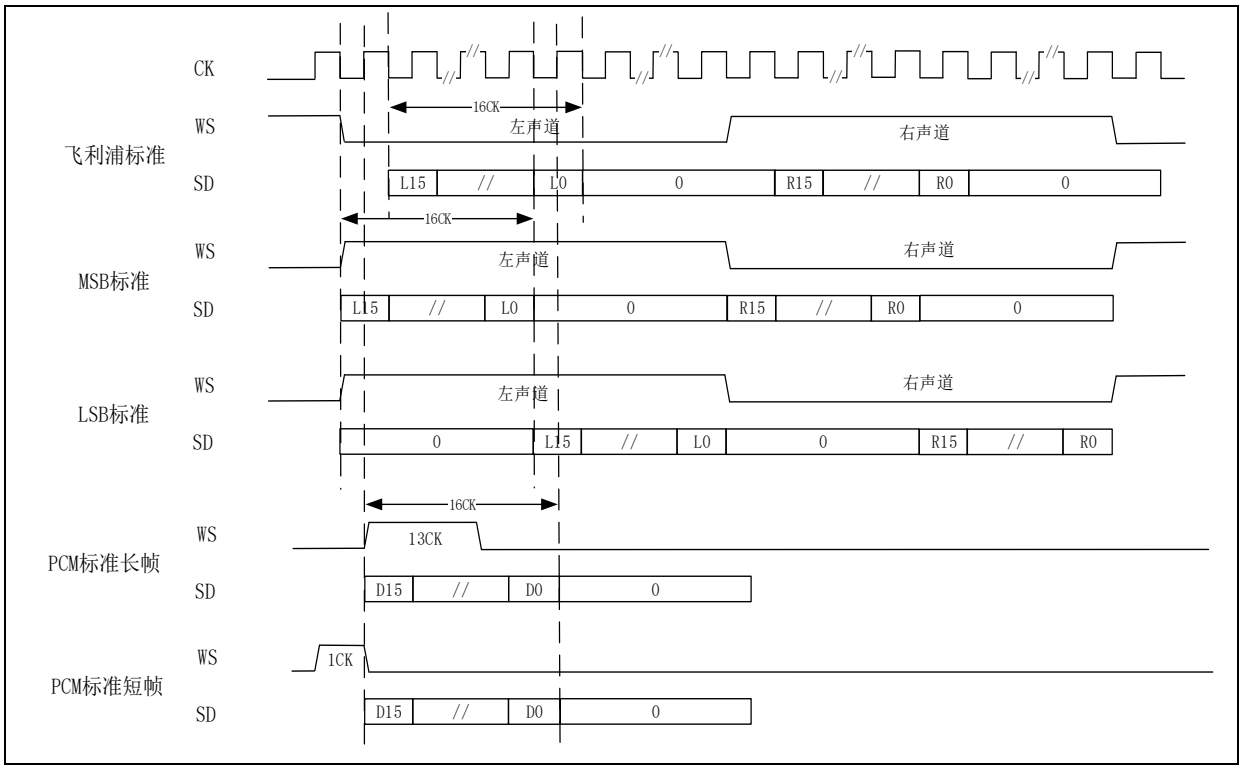
LSB 标准下：WS 为高电平表示正在传输的声道为左声道，WS 为低电平表示正在传输的声道为右声道。数据右对齐，即低位对齐，高位的 16 个 bit 被硬件填充为 0。

注：LSB 标准下，数据发送顺序还是高位在前。

PCM 标准长帧下：没有左右声道之分，WS 的脉冲代表同步信号而不是左右声道。WS 脉冲长度为 13 个 CK 长度。

PCM 标准短帧下：没有左右声道之分，WS 的脉冲代表同步信号而不是左右声道。WS 脉冲长度为 1 个 CK 长度。

图 21 I²S 各音频标准时序对比



3.4 I²S 帧格式

AT32 的 I²S 支持 3 种数据位个数选择：

- 16 位：I2SDBN=0b00
- 24 位：I2SDBN=0b01
- 32 位：I2SDBN=0b10

AT32 的 I²S 支持 2 种声道位个数选择：

- 16 位：I2SDBN =0
- 32 位：I2SDBN =1

数据位个数和声道位个数组合有以下 4 种帧格式：

① 16 位数据-16 位声道格式：I2SDBN=0b00，I2SDBN =0；

② 16 位数据-32 位声道格式：I2SDBN=0b00，I2SDBN =1；

③ 24 位数据-32 位声道格式：I2SDBN=0b01，I2SDBN =1；

④ 32 位数据-32 位声道格式：I2SDBN=0b10，I2SDBN =1；

上述的第②、③帧格式中，由于声道位个数>数据位个数。此时硬件会自动将多余的声道位填充为 0。可参考上图 21 中，“16 位数据-32 位声道格式”的时序。

上述的①、②帧格式，在接收/发送数据时，仅需要读取/写入一次 16bit 的 DT 寄存器；而③、④帧格式，在接收/发送数据时，需要读取/写入两次 16bit 的 DT 寄存器。

3.5 I²S 时钟控制器

3.5.1 I²S 采样率（Fs）

常见的音频的采样频率有：192KHz、96kHz、48kHz、44.1kHz、32kHz、22.05kHz、16kHz、11.025kHz、8kHz 等。而实际上 AT32 的 I²S 可以产生以上范围内的任何采样率。

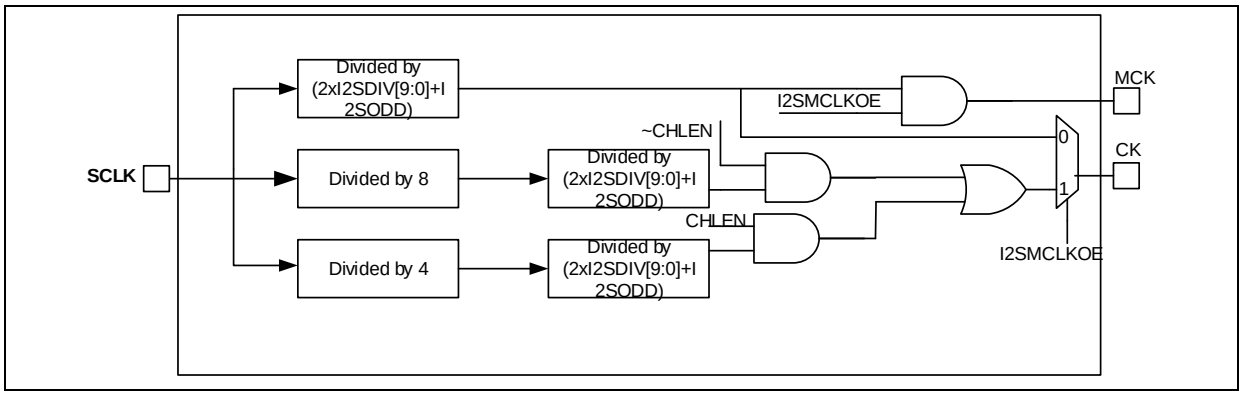
例如采样率为 192K，则表示每秒可传输 192K 帧音频数据。对于飞利浦/MSB/LSB 标准而言，一帧音频数据包含左、右通道两笔 16/32bit 的数据；对于 PCM 标准而言，一帧数据仅包含一笔 16/32bit 的数据。

3.5.2 I²S 时钟（CK）和主时钟（MCK）

如下图 22，I²S 的时钟从系统时钟 SCLK 分频得来。不过使用 I²S 时仍然需要使能挂载在 APB 时钟总线上的 SPI 接口时钟（SPIxEN=1），以用于 I²S 逻辑控制器等。

- 当需要输出主时钟（I2SMCLKOE=1），声道位个数为 16bit（I2SDBN=0）时：Fs=SCLK/[(16*2)* ((2*I2SDIV) + I2SODD)*8]；
- 当需要输出主时钟（I2SMCLKOE=1），声道位个数为 32bit（I2SDBN=1）时：Fs=SCLK/[(32*2)* ((2*I2SDIV) + I2SODD)*4]；
- 当不需要输出主时钟（I2SMCLKOE=0），声道位个数为 16bit（I2SDBN=0）时：Fs=SCLK/[(16*2)* ((2*I2SDIV) + I2SODD)]；
- 当不需要输出主时钟（I2SMCLKOE=0），声道位个数为 32bit（I2SDBN=1）时：Fs=SCLK/[(32*2)* ((2*I2SDIV) + I2SODD)]；

图 22 I²S 时钟



如下表 3，示例了系统时钟为 240MHz 时，配置 192K 的采样率推荐的 I²S 时钟配置方案。

表 3 I²S 时钟配置方案示例

SCLK (MHz)	MCK	Target Fs (Hz)	16bit				32bit			
			I2S DIV	I2S_ODD	RealFs	Error	I2S DIV	I2S_ODD	RealFs	Error
240	NO	192000	19	1	192307	1.2%	10	0	187500	2.34%
240	YES	192000	2	1	187500	2.34%	2	1	187500	2.34%

3.6 I²S 全双工

AT32F403A/F407/F435/F437/F425 等型号支持 I²S 全双工，AT32F403/F413/F415/F421 等型号不支持 I²S 全双工，具体请参考对应型号 RM 文档。不同型号 MCU 的 I²S 全双工实现方式也有所区别，本文会分别介绍 AT32F403A/F407/F435/F437 和 AT32F425 的 I²S 全双工功能。

3.6.1 AT32F403A/F407/F435/F437 的 I²S 全双工

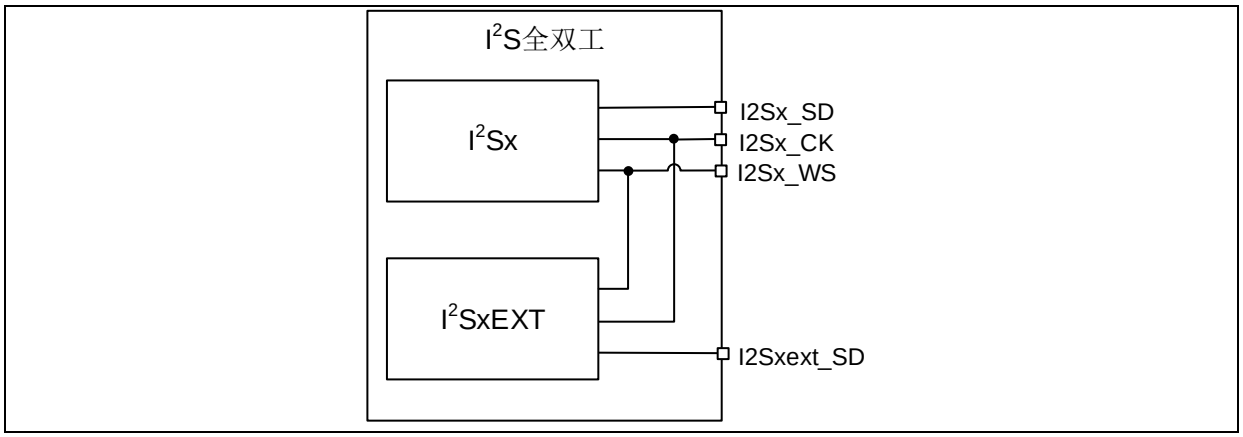
AT32F403A/F407/F435/F437 为了支持 I²S 全双工模式，额外例化了两个 I²S 模块（I²S2EXT，I²S3EXT）。I²S2 可与 I²S2EXT 组合在一起支持全双工模式，I²S3 可与 I²S3EXT 组合在一起支持全双工模式。代码可参考“案例 3—AT32F403A/F407/F435/F437 I2S 全双工 DMA 方式通信”。

在 I²Sx 可与 I²SxEXT 组成全双工模块时，I²Sx 可配置为主或从模式，I²SxEXT 只能配置为从模式。I²SxEXT 共享 I²Sx 的 CK 和 WS，对应的有以下几种配置方式：

- I²Sx 主发（OPERSEL=0b10）；I²SxEXT 从收（OPERSEL=0b01）；
- I²Sx 主收（OPERSEL=0b11）；I²SxEXT 从发（OPERSEL=0b00）。
- I²Sx 从发（OPERSEL=0b00）；I²SxEXT 从收（OPERSEL=0b01）；
- I²Sx 从收（OPERSEL=0b01）；I²SxEXT 从发（OPERSEL=0b00）。

注：I²S2EXT 和 I²S3EXT 只用于 I²S 全双工模式，而不能单独使用。

图 23 I²S 全双工结构图

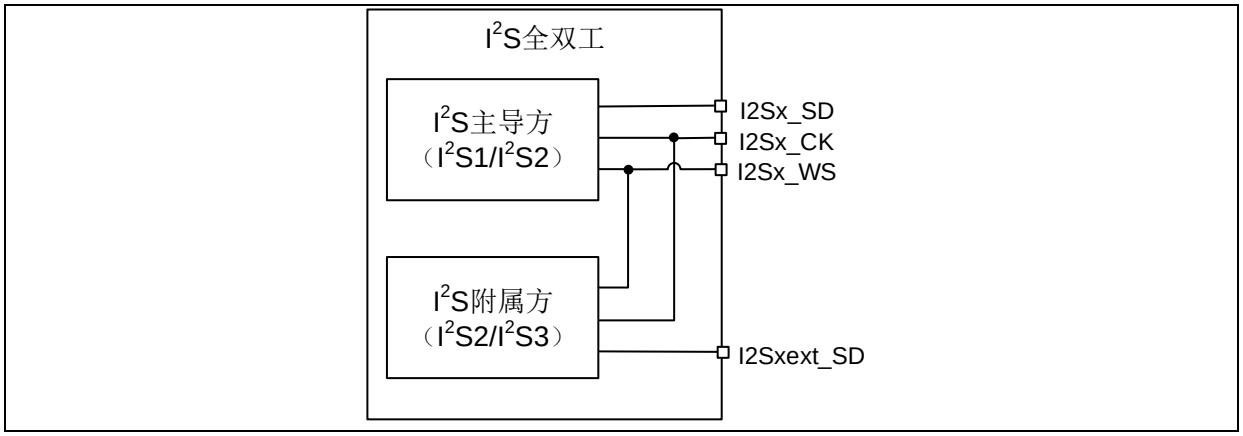


3.6.2 AT32F425 的 I²S 全双工

AT32F425 的 I²S 全双工实现方式和 AT32F403A/F407/F435/F437 不同。AT32F425 可以通过设置 SCFG_CFG2 寄存器中的 I2S_FD 将两个 SPI 组合在一起实现 I²S 全双工。代码可参考“案例 4—AT32F425 I2S 全双工 DMA 方式通信”。CK 和 WS 由 I²S 主导方提供，附属方的 CK 和 WS 对应引脚可以释放给其他功能使用。根据 I2S_FD 配置，有以下几种组合方式：

- I2S_FD=0b00：I²S1~3 各自独立半双工工作；
- I2S_FD=0b01：I²S1 和 I²S3 组成全双工模式，其中 I²S1 为主导方，I²S3 为附属方。
- I2S_FD=0b10：I²S2 和 I²S3 组成全双工模式，其中 I²S2 为主导方，I²S3 为附属方。
- I2S_FD=0b11：I²S1 和 I²S2 组成全双工模式，其中 I²S1 为主导方，I²S2 为附属方。

图 24 I²S 全双工结构图



3.7 I²S 错误

3.7.1 ROERR--接收器溢出错误标志

此错误标志和 SPI 一样，请参考前文“2.9.2 ROERR--接收器溢出错误标志”。

3.7.2 TUERR--发送器欠载错误标志位

在 I²S 从发模式下，如果在 CK 上检测到了数据驱动沿，但新的发送数据并没有被写入数据寄存器，则会产生发送器欠载错误，此时硬件会将 TUERR 位置 1。软件读 SPI_STS 寄存器可清除该标志位。此错误标志仅使用 I²S 时有效。

3.8 I²S 中断

I²S 和 SPI 共用一个全局中断向量，即下图 25 的“I²S 中断”和上图 18 中的“SPI 中断”共用同一个中断向量。I²S 中断有三个中断源：接收缓冲器满、发送缓冲器空、通信错误（详见上一节“I²S 错误”）。这三个中断源分别有对应的使能位。

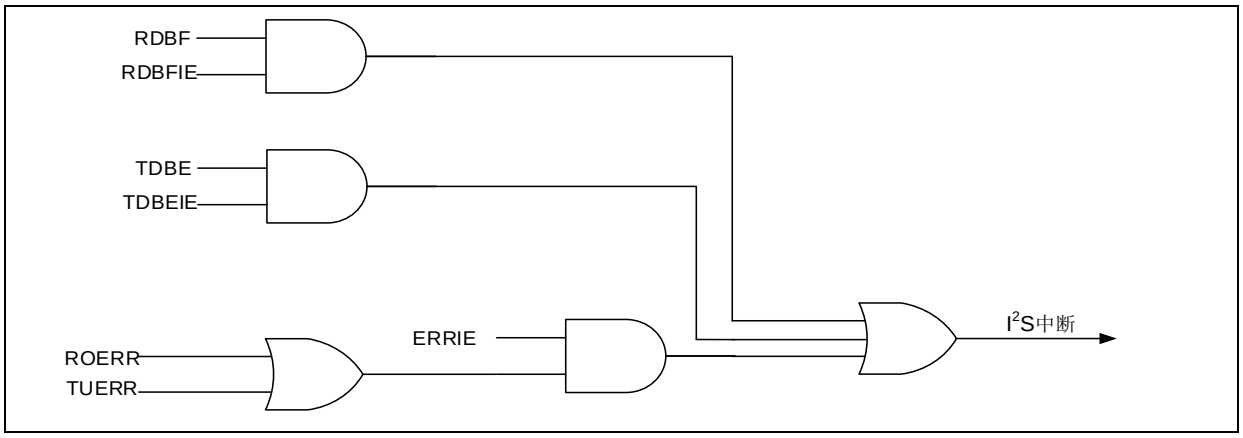
下图中，3 个中断使能位定义如下：

- RDBFIE：接收数据缓冲器满中断使能（和 SPI 一样）
- TDBEIE：发送数据缓冲器空中断使能（和 SPI 一样）
- ERRIE: 错误中断使能（和 SPI 一样）

下图中，4 个标志位定义如下：

- ROERR：接收器溢出错误（和 SPI 一样）
- TUERR：发送器欠载错误（仅 I²S 有这个错误标志）
- RDBF：接收数据缓冲器满（和 SPI 一样）
- TDBE：发送数据缓冲器空（和 SPI 一样）

图 25 I²S 中断示意



4 SPI 案例

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

4.1 案例 1-- SPI 全双工轮询方式通信

4.1.1 功能简介

实现 SPI2 和 SPI3 之间的全双工轮询通信。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)
pc2(miso)	<--->	pc11 (miso)
pd4(mosi)	<--->	pc12(mosi)

4.1.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\spi\fullduplex_polling

4.1.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 通信配置；
- 开始轮询通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机，SPI3 为主机*/
  /* transfer procedure:the "BUFFER_SIZE" data transfer */
  while(tx_index < BUFFER_SIZE)/*循环收发 BUFFER_SIZE 次*/
  {
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);/*等待 SPI3 发送缓冲区空*/
    spi_i2s_data_transmit(SPI2, spi2_buffer_tx[tx_index]);/*向 SPI2 发送缓冲区写入待发送数据*/
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*向 SPI3 发送缓冲区写入待发送数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI2 接收缓冲区满*/
    spi2_buffer_rx[rx_index] = spi_i2s_data_receive(SPI2);/*读取 SPI2 收到的数据*/
```

```
    while(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI3 接收缓冲区满*/
    spi3_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI3);/*读取 SPI3 收到的数据*/
  }
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对 SPI3 收到的数据是否正确*/
  /* master &slave mode switch */
  spi_enable(SPI3, FALSE);/*关闭 SPI3*/
  spi_enable(SPI2, FALSE);/*关闭 SPPI2*/
  spi_init_struct.master_slave_mode =SPI_MODE_SLAVE;/*将 SPI3 配置为从机*/
  spi_init(SPI3, &spi_init_struct);
  spi_init_struct.master_slave_mode =SPI_MODE_MASTER;/*将 SPI2 配置为主机*/
  spi_init(SPI2, &spi_init_struct);
  tx_index = 0;
  rx_index = 0;
  for(index = 0; index < BUFFER_SIZE; index++)/*将自定义的 SPI3 接收 buffer 清 0*/
    spi3_buffer_rx[index] = 0;
  for(index = 0; index < BUFFER_SIZE; index++)/*将自定义的 SPI2 接收 buffer 清 0*/
    spi2_buffer_rx[index] = 0;
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
  /* transfer procedure:the "BUFFER_SIZE" data transfer */
  while(tx_index < BUFFER_SIZE) /*循环收发 BUFFER_SIZE  次*/
  {
    while(spi_i2s_flag_get(SPI2, SPI_I2S_TDBE_FLAG) == RESET);/*等待 SPI2 发送缓冲区空*/
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index]);/*向 SPI3 发送缓冲区写入待发送数据*/
    spi_i2s_data_transmit(SPI2, spi2_buffer_tx[tx_index++]);/*向 SPI2 发送缓冲区写入待发送数据*/
    while(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI3 接收缓冲区满*/
    spi3_buffer_rx[rx_index] = spi_i2s_data_receive(SPI3);/*读取 SPI3 收到的数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI2 接收缓冲区满*/
    spi2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2);/*等待 SPI2 接收缓冲区满*/
  }
  /* test result:the data check */
  transfer_status3 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  transfer_status4 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对 SPI3 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS) && \
     (transfer_status3 == SUCCESS) && (transfer_status4 == SUCCESS))/*判断是否以上两轮通信都正确*/
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
```

```
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ SPI 通信配置函数代码描述

```
static void spi_config(void)
{
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/

  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/

  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力
为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;/*选取 pin10*/
```

```
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_11;/*选取 pin11*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC11*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE11, GPIO_MUX_6);/*配置 PC11 复用为 MUC_6 功能，即为 SPI3_MISO 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;/*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD4*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
}
```

4.1.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.2 案例 2-- SPI 全双工 DMA 方式通信

4.2.1 功能简介

实现 SPI2 和 SPI3 之间的全双工 DMA 通信。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)

pc2(miso) <---> pc11 (miso)
pd4(mosi) <---> pc12(mosi)

4.2.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\spi\motorola_fullduplex_dma

4.2.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 的通信配置和 DMA 配置；
- 开始 DMA 通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机，SPI3 为主机*/
  dma_channel_enable(DMA1_CHANNEL1, TRUE);/*使能 DMA1 CHANNEL1:对应 SPI2 发送*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 CHANNEL2:对应 SPI2 接收*/
  dma_channel_enable(DMA1_CHANNEL4, TRUE);/*使能 DMA1 CHANNEL4:对应 SPI3 发送*/
  dma_channel_enable(DMA1_CHANNEL3, TRUE);/*使能 DMA1 CHANNEL3:对应 SPI3 接收*/
  while(dma_flag_get(DMA1_FDT2_FLAG) == RESET);
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对 SPI3 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
```

```
  }
}
```

■ SPI 通信配置及 DMA 配置函数代码描述

```
static void spi_config(void)
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dma_reset(DMA1_CHANNEL1);/*复位 DMA1 channel1，使 channel1 处于默认配置*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dma_reset(DMA1_CHANNEL3);/*复位 DMA1 channel3，使 channel3 处于默认配置*/
  dma_reset(DMA1_CHANNEL4);/*复位 DMA1 channel4，使 channel4 处于默认配置*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_SPI2_TX);/*弹性映射配置：DMA1 channel1 对应 SPI2 发送*/
  dmamux_init(DMA1MUX_CHANNEL2, DMAMUX_DMAREQ_ID_SPI2_RX);/*弹性映射配置：DMA1 channel2 对应 SPI2 接收*/
  dmamux_init(DMA1MUX_CHANNEL3, DMAMUX_DMAREQ_ID_SPI3_TX);/*弹性映射配置：DMA1 channel3 对应 SPI3 发送*/
  dmamux_init(DMA1MUX_CHANNEL4, DMAMUX_DMAREQ_ID_SPI3_RX);/*弹性映射配置：DMA1 channel4 对应 SPI3 接收*/
  dma_default_para_init(&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = BUFFER_SIZE;/*设置 DMA buffer 长度：和 SPI 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;/*DMA 内存数据宽度：BYTE。和自定义的 SPI 收/发 buffer 格式一致*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;/*DMA 外设数据宽度：BYTE。和 SPI 的数据格式一致*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_tx;/*内存地址：SPI2 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL1, &dma_init_struct);/*将 DMA1 channel1 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_rx;/*内存地址：SPI2 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct);/*将 DMA1 channel2 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_tx;/*内存地址：SPI3 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL3, &dma_init_struct);/*将 DMA1 channel3 设置为以上配置*/
```

```
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_rx;/*内存地址：SPI3 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL4, &dma_init_struct);/*将 DMA1 channel4 设置为以上配置*/
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  spi_i2s_dma_transmitter_enable(SPI2,TRUE);/*使能 SPI2 DMA 发送功能*/
  spi_i2s_dma_transmitter_enable(SPI3,TRUE);/*使能 SPI3 DMA 发送功能*/
  spi_i2s_dma_receiver_enable(SPI2,TRUE);/*使能 SPI2 DMA 接收功能*/
  spi_i2s_dma_receiver_enable(SPI3,TRUE);/*使能 SPI3 DMA 接收功能*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力
为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;/*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功
能，即为 SPI3_SCK 功能*/
  /* spi3 miso pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;/*配置 GPIO 上拉*/
```

```
  gpio_initstructure.gpio_pins              = GPIO_PINS_11;/*选取 pin11*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC11*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE11, GPIO_MUX_6);/*配置 PC11 复用为 MUC_6 功能，即为 SPI3_MISO 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;/*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD4*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
}
```

4.2.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.3 案例 3-- SPI 只收通信

4.3.1 功能简介

实现 SPI2 和 SPI3 之间的单向只收通信。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)
pd4(mosi)	<--->	pc12(mosi)

4.3.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\spi\only_receive_mode_polling

4.3.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 的通信配置；
- 开始轮询通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机只收，SPI3 为主机全双工*/
  /* transfer procedure:the "BUFFER_SIZE" data transfer */
  while(tx_index < BUFFER_SIZE)/*循环收发 BUFFER_SIZE 次*/
  {
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);/*等待 SPI3 发送缓冲区空*/
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*向 SPI3 发送缓冲区写入待发送数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI2 接收缓冲区满*/
    spi2_buffer_rx[rx_index] = spi_i2s_data_receive(SPI2);/*读取 SPI2 收到的数据*/
  }
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if(transfer_status1 == SUCCESS)
  {
    at32_led_on(LED2);/*数据正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

- SPI 通信配置代码描述

```
static void spi_config(void)
```

```
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_SIMPLEX_RX;/*配置 SPI 为只收模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type          = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力为强*/
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode               = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins               = GPIO_PINS_10;/*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins               = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
```

```
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;/*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD4*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
while(gpio_input_data_bit_read(GPIOD, GPIO_PINS_1) != RESET);/*等待从机的 SCK 拉低后再进行后续的 SPI 配置*/
}
```

4.3.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.4 案例 4-- SPI 半双工中断方式通信

4.4.1 功能简介

实现 SPI2 和 SPI3 之间的半双工通信。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)
pc2(miso)	<--->	pc12(mosi)

4.4.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_xxxx\examples\spi\halfduplex_interrupt

4.4.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 设置 SPI2 和 SPI3 的中断处理函数；
- 配置 SPI2 和 SPI3 的通信配置；
- 开始中断通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
```

```
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机只收，SPI3 为主机全双工*/
  while(rx_index < BUFFER_SIZE);/*等待接收完预期的 BUFFER_SIZE 个数据*/
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if(transfer_status1 == SUCCESS)
  {
    at32_led_on(LED2);/*数据正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ 中断处理函数代码描述

```
void SPI3_I2S3EXT_IRQHandler(void)
{
  if(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) != RESET)/*判断是否 SPI3 发送缓冲区空标志置起*/
  {
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*发送一笔待发送数据*/
    if(tx_index == BUFFERSIZE)
    {
      spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, FALSE);/*发送完预期的 BUFFERSIZE 笔数据后，关闭 SPI3 发送缓冲区空中断*/
    }
  }
}
void SPI2_I2S2EXT_IRQHandler(void)
{
  if(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) != RESET)/*判断是否 SPI2 接收缓冲区满标志置起*/
  {
    spi2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2);/*读取一笔收到的数据*/
  }
}
```

■ SPI 通信配置函数代码描述

```
static void spi_config(void)
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_TX;/*配置 SPI 为半双工发送模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_RX;/*配置 SPI 为半双工接收模式
*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  nvic_irq_enable(SPI3_I2S3EXT_IRQn, 0, 0);/*使能 SPI3 中断向量并配置优先级*/
  nvic_irq_enable(SPI2_I2S2EXT_IRQn, 0, 0);/*使能 SPI2 中断向量并配置优先级*/
  spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, TRUE);/*使能 SPI3 发送缓冲区空中断*/
  spi_i2s_interrupt_enable(SPI2, SPI_I2S_RDBF_INT, TRUE);/*使能 SPI2 接收缓冲区满中断*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力
为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;/*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功
能，即为 SPI3_SCK 功能*/
  /* spi3 mosi pin */
```

```
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
while(gpio_input_data_bit_read(GPIOD, GPIO_PINS_1) != RESET);/*等待从机的 SCK 拉低后再进行后续的 SPI 配置*/
}
```

4.4.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.5 案例 5-- SPI 半双工中断方式通信--加收发切换

4.5.1 功能简介

实现 SPI2 和 SPI3 之间的半双工收发切换通信。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)
pc2(miso)	<--->	pc12(mosi)

4.5.2 资源准备

1) 硬件环境：

   一块 AT32F437 的 AT-START BOARD

2) 软件环境：

   project\at_start_f437\examples\spi\ halfduplex_transceiver_switch

4.5.3 软件设计

1) 配置流程

   - 配置 SPI2 和 SPI3 对应的 GPIO；
   - 设置 SPI2 和 SPI3 的中断处理函数；
   - 配置 SPI2 和 SPI3 的通信配置；

- 开始中断通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机只收，SPI3 为主机全双工*/
  while(rx_index < BUFFER_SIZE);/*等待接收完预期的 BUFFER_SIZE 个数据*/
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  /* config spi2 send spi1 receive */
  spi_enable(SPI3, FALSE);/*关闭 SPI3*/
  spi_enable(SPI2, FALSE);/*关闭 SPI2*/
  rx_index = 0;/*接收计数清零*/
  tx_index = 0;/*发送计数清零*/
  spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, FALSE);/*关闭 SPI3 发送缓冲区空中断*/
  spi_i2s_interrupt_enable(SPI2, SPI_I2S_RDBF_INT, FALSE);/*关闭 SPI2 接收缓冲区满中断*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_RX;/*重新配置 SPI3 为半双工只收*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*仍配置 SPI3 为主机*/
  spi_init(SPI3, &spi_init_struct);

  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_TX;/*重新配置 SPI2 为半双工只发*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*仍配置 SPI2 为从机*/
  spi_init(SPI2, &spi_init_struct);
  spi_i2s_interrupt_enable(SPI3, SPI_I2S_RDBF_INT, TRUE);/*使能 SPI3 接收缓冲区满中断*/
  spi_i2s_interrupt_enable(SPI2, SPI_I2S_TDBE_INT, TRUE);/*使能 SPI2 发送缓冲区空中断*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  while(rx_index < BUFFER_SIZE);/*等待接收完预期 BUFFER_SIZE 个数据*/
  /* test result:the data check */
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对接收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*数据都正确则点亮 LED2*/
  }
  else
  {
```

```
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ 中断处理函数代码描述

```
void SPI3_I2S3EXT_IRQHandler(void)
{
  if(SPI3->ctrl2_bit.tdbeie != RESET)/*判断是否使能了 SPI3 发送缓冲区空中断使能*/
  {
    if(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) != RESET) /*判断是否 SPI3 发送缓冲区空标志置起*/
    {
      spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*发送一笔待发送数据*/
      if(tx_index == BUFFERSIZE)
      {
        spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, FALSE); /*发送完预期的 BUFFERSIZE 笔数据后，关闭 SPI3 发送缓冲区空中断*/
      }
    }
  }
  if(SPI3->ctrl2_bit.rdbfie != RESET) /*判断是否使能了 SPI3 接收缓冲区满中断使能*/
  {
    if(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) != RESET) /*判断是否 SPI3 接收缓冲区满标志置起*/
    {
      spi_enable(SPI3, FALSE);/*关闭主机 SPI3。因为 SPI 主机只收模式下，一旦使能了 SPI，就会连续不断的输出时钟。所以在中断/轮询通信模式下，为避免来不及接收数据，主机每收到一笔数据先关闭 SPI，接收完这笔数据后，再使能 SPI，继续发出时钟，以接收下一笔数据。如果使用 DMA 通信方式，数据接收无需软件处理，则无需每笔数据后关闭 SPI*/
      spi3_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI3);/*读取接收到的数据*/
      spi_enable(SPI3, TRUE);/*使能 SPI3，继续输出时钟，继续通信*/
      if(rx_index == BUFFERSIZE)
      {
        spi_i2s_interrupt_enable(SPI3, SPI_I2S_RDBF_FLAG, FALSE);/*接收完预期的 BUFFERSIZE 个数据后，关闭 SPI3 接收缓冲区满中断*/
        spi_enable(SPI3, FALSE);/*关闭 SPI3，避免输出多余的时钟*/
      }
    }
  }
}
void SPI2_I2S2EXT_IRQHandler(void)
{
  if(SPI2->ctrl2_bit.tdbeie != RESET) /*判断是否使能了 SPI2 发送缓冲区空中断使能*/
  {
```

```
    if(spi_i2s_flag_get(SPI2, SPI_I2S_TDBE_FLAG) != RESET) /*判断是否 SPI2 发送缓冲区空标志置起*/
    {
      spi_i2s_data_transmit(SPI2, spi2_buffer_tx[tx_index++]);/*发送一笔待发送数据*/
      if(tx_index == BUFFERSIZE)
      {
        spi_i2s_interrupt_enable(SPI2, SPI_I2S_TDBE_INT, FALSE);/*发送完预期的 BUFFERSIZE 笔数据之后，关闭 SPI2 发送缓冲区空中断*/
      }
    }
  }
  if(SPI2->ctrl2_bit.rdbfie != RESET)/*判断是否使能了 SPI2 接收缓冲区满中断使能*/
  {
    if(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) != RESET) /*判断是否 SPI2 接收缓冲区满标志置起*/
    {
      spi2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2); /*读取接收到的数据*/
    }
  }
}
```

■ SPI 通信配置函数代码描述

```
static void spi_config(void)
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_TX;/*配置 SPI 为半双工发送模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_RX;/*配置 SPI 为半双工接收模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  nvic_irq_enable(SPI3_I2S3EXT_IRQn, 0, 0);/*使能 SPI3 中断向量并配置优先级*/
  nvic_irq_enable(SPI2_I2S2EXT_IRQn, 0, 0);/*使能 SPI2 中断向量并配置优先级*/
  spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, TRUE);/*使能 SPI3 发送缓冲区空中断*/
  spi_i2s_interrupt_enable(SPI2, SPI_I2S_RDBF_INT, TRUE);/*使能 SPI2 接收缓冲区满中断*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
```

```
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;/*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
while(gpio_input_data_bit_read(GPIOD, GPIO_PINS_1) != RESET);/*等待从机的 SCK 拉低后再进行后续的 SPI 配置*/
}
```

4.5.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.6 案例 6-- SPI CRC 功能使用

4.6.1 功能简介

实现 SPI2 和 SPI3 之间的全双工通信和 CRC 校验。接线如下：

spi2		spi3
pd1(sck)	<--->	pc10(sck)
pc2(miso)	<--->	pc11 (miso)
pd4(mosi)	<--->	pc12(mosi)

4.6.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\spi\crc_transfer_polling

4.6.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 通信配置；
- 开始轮询通信和 CRC 校验。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机，SPI3 为主机*/

  /* transfer procedure:the "BUFFER_SIZE-1" data transfer */
  while(tx_index < BUFFER_SIZE - 1)/*轮询发送和接收 BUFFER_SIZE – 1 个数据*/
  {
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);/*等待 SPI3 发送缓冲区空*/
    spi_i2s_data_transmit(SPI2, spi2_buffer_tx[tx_index]);/*向 SPI2 发送缓冲区写入待发送数据*/
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*向 SPI3 发送缓冲区写入待发送数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI2 接收缓冲区满*/
    spi2_buffer_rx[rx_index] = spi_i2s_data_receive(SPI2);/*读取 SPI2 收到的数据*/
    while(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) == RESET);/*等待 SPI3 接收缓冲区满*/
    spi3_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI3);/*读取 SPI3 收到的数据*/
  }
  while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);/*等待 SPI3 发送缓冲区空标志置起*/
  while(spi_i2s_flag_get(SPI2, SPI_I2S_TDBE_FLAG) == RESET); /*等待 SPI2 发送缓冲区空标志置起*/
  /* transfer procedure:the last data and crc transfer */
  spi_i2s_data_transmit(SPI2, spi2_buffer_tx[tx_index]);/*SPI2 发送最后一笔数据*/
```

```
  spi_crc_next_transmit(SPI2);/*立即请求下一笔发送 CRC 校验码，也就是上一步发送的最后一笔数据发送完成后，会立即发送 CRC 校验码*/
  spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index]); /*SPI3 发送最后一笔数据*/
  spi_crc_next_transmit(SPI3); /*立即请求下一笔发送 CRC 校验码，也就是上一步发送的最后一笔数据发送完成后，会立即发送 CRC 校验码*/
  while(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) == RESET); /*等待 SPI3 接收缓冲区满*/
  spi3_buffer_rx[rx_index] = spi_i2s_data_receive(SPI3); /*读取 SPI3 收到的数据*/
  while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET); /*等待 SPI2 接收缓冲区满*/
  spi2_buffer_rx[rx_index] = spi_i2s_data_receive(SPI2); /*读取 SPI2 收到的数据*/
  while(spi_i2s_flag_get(SPI3, SPI_I2S_RDBF_FLAG) == RESET);/*再一次等待 SPI3 接收缓冲区满，也就是等待 CRC 校验码接收完成*/
  while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET); /*再一次等待 SPI2 接收缓冲区满，也就是等待 CRC 校验码接收完成*/
  /* test result:the data and crc check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 接收到的数据是否正确*/
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE); /*核对 SPI3 接收到的数据是否正确*/
  if((spi_i2s_flag_get(SPI3, SPI_CCERR_FLAG)) != RESET)/*判断 SPI3 CRC 校验是否正确—通过判断 CRC 错误标志是否置起*/
  {
    spi_i2s_flag_clear(SPI3, SPI_CCERR_FLAG);
    transfer_status2 = ERROR;
  }
  if((spi_i2s_flag_get(SPI2, SPI_CCERR_FLAG)) != RESET) /*判断 SPI2 CRC 校验是否正确—通过判断 CRC 错误标志是否置起*/
  {
    spi_i2s_flag_clear(SPI2, SPI_CCERR_FLAG);
    transfer_status1 = ERROR;
  }
  crc1_value = spi_i2s_data_receive(SPI3);/*软件读取一次数据寄存器，以清除接收缓冲区满标志*/
  crc2_value = spi_i2s_data_receive(SPI2);/*软件读取一次数据寄存器，以清除接收缓冲区满标志*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))/*判断 CRC 校验是否都正确*/
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ SPI 通信配置函数代码描述

```
static void spi_config(void)
{
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  spi_crc_polynomial_set(SPI3, 7);/*设置 SPI3 CRC 多项式（主从的多项式需设置为一样）：7*/
  spi_crc_polynomial_set(SPI2, 7); /*设置 SPI2 CRC 多项式（主从的多项式需设置为一样）：7*/
  spi_crc_enable(SPI3, TRUE);/*使能 SPI3 CRC 功能*/
  spi_crc_enable(SPI2, TRUE);/*使能 SPI2 CRC 功能*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;/*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 miso pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_11;/*选取 pin11*/
```

```
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC11*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE11, GPIO_MUX_6);/*配置 PC11 复用为 MUC_6 功能，即为 SPI3_MISO 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;/*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD4*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
}
```

4.6.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.7 案例 7-- SPI TI 模式使用（TI SSP 协议）

4.7.1 功能简介

实现 SPI2 和 SPI3 之间的 TI 模式 DMA 通信。接线如下：

spi2		spi3
pd0(cs)	<--->	pa4(cs)
pd1(sck)	<--->	pc10(sck)
pc2(miso)	<--->	pc11 (miso)
pd4(mosi)	<--->	pc12(mosi)

4.7.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\spi\ti_fullduplex_dma

4.7.3 软件设计

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 的通信配置和 DMA 配置；
- 开始 DMA 通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机，SPI3 为主机*/
  dma_channel_enable(DMA1_CHANNEL1, TRUE);/*使能 DMA1 CHANNEL1:对应 SPI2 发送*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 CHANNEL2:对应 SPI2 接收*/
  dma_channel_enable(DMA1_CHANNEL4, TRUE);/*使能 DMA1 CHANNEL4:对应 SPI3 发送*/
  dma_channel_enable(DMA1_CHANNEL3, TRUE);/*使能 DMA1 CHANNEL3:对应 SPI3 接收*/
  while(dma_flag_get(DMA1_FDT2_FLAG) == RESET);
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数
据是否正确*/
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对 SPI3 收到的数
据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

- SPI 通信配置及 DMA 配置函数代码描述

```
static void spi_config(void)
{
```

```
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dma_reset(DMA1_CHANNEL1);/*复位 DMA1 channel1，使 channel1 处于默认配置*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dma_reset(DMA1_CHANNEL3);/*复位 DMA1 channel3，使 channel3 处于默认配置*/
  dma_reset(DMA1_CHANNEL4);/*复位 DMA1 channel4，使 channel4 处于默认配置*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_SPI2_TX);/*弹性映射配置：DMA1
channel1 对应 SPI2 发送*/
  dmamux_init(DMA1MUX_CHANNEL2, DMAMUX_DMAREQ_ID_SPI2_RX);/*弹性映射配置：DMA1
channel2 对应 SPI2 接收*/
  dmamux_init(DMA1MUX_CHANNEL3, DMAMUX_DMAREQ_ID_SPI3_TX);/*弹性映射配置：DMA1
channel3 对应 SPI3 发送*/
  dmamux_init(DMA1MUX_CHANNEL4, DMAMUX_DMAREQ_ID_SPI3_RX);/*弹性映射配置：DMA1
channel4 对应 SPI3 接收*/
  dma_default_para_init(&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = BUFFER_SIZE;/*设置 DMA buffer 长度：和 SPI 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;/*DMA 内存数据宽度：
BYTE。和自定义的 SPI 收/发 buffer 格式一致*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加
一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;/*DMA 外设数据宽
度：BYTE。和 SPI 的数据格式一致*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）
*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_tx;/*内存地址：SPI2 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL1, &dma_init_struct);/*将 DMA1 channel1 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_rx;/*内存地址：SPI2 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct);/*将 DMA1 channel2 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_tx;/*内存地址：SPI3 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL3, &dma_init_struct);/*将 DMA1 channel3 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_rx;/*内存地址：SPI3 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL4, &dma_init_struct);/*将 DMA1 channel4 设置为以上配置*/
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
```

```
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI 为软件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  spi_i2s_dma_transmitter_enable(SPI2,TRUE);/*使能 SPI2 DMA 发送功能*/
  spi_i2s_dma_transmitter_enable(SPI3,TRUE);/*使能 SPI3 DMA 发送功能*/
  spi_i2s_dma_receiver_enable(SPI2,TRUE);/*使能 SPI2 DMA 接收功能*/
  spi_i2s_dma_receiver_enable(SPI3,TRUE);/*使能 SPI3 DMA 接收功能*/
  spi_ti_mode_enable(SPI3, TRUE);/*使能 SPI3 TI 模式*/
  spi_ti_mode_enable(SPI2, TRUE);/*使能 SPI2 TI 模式*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
/* spi3 cs pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL; /*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /*配置 GPIO 驱动能力为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN; /*配置 GPIO 下拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX; /*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins = GPIO_PINS_4; /*选取 pin4/
  gpio_init(GPIOA, &gpio_initstructure);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE4, GPIO_MUX_6) /*配置 PA4 为 MUC_6 功能，即为 SPI3_CS*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN; /*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins = GPIO_PINS_10; /*选取 pin10*/
  gpio_init(GPIOC, &gpio_initstructure); /*按以上配置设置 PC10*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6); /*配置 PC10 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 miso pin */
```

```
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_11;/*选取 pin11*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC11*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE11, GPIO_MUX_6);/*配置 PC11 复用为 MUC_6 功能，即为 SPI3_MISO 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_12;/*选取 pin12*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC12*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 cs pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN; /*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_0;/*选取 pin0*/
  gpio_init(GPIOD, &gpio_initstructure); /*按以上配置设置 PD0/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7); /*配置 PD0 为 MUC_7 功能，即为 SPI2_CS*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;/*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;/*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD1*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD11 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_2;/*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);/*按以上配置设置 PC2*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);/*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;/*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);/*按以上配置设置 PD4*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
}
```

4.7.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

4.8 案例 8-- SPI 读写 FLASH（W25Q128）

4.8.1 功能简介

实现 SPI2 读/写 flash（w25q128）。接线如下：

spi2　　w25q128

pd0(cs) <---> cs pin
pd1(sck) <---> clk pin
pc2(miso) <---> di pin
pd4(mosi) <---> do pin

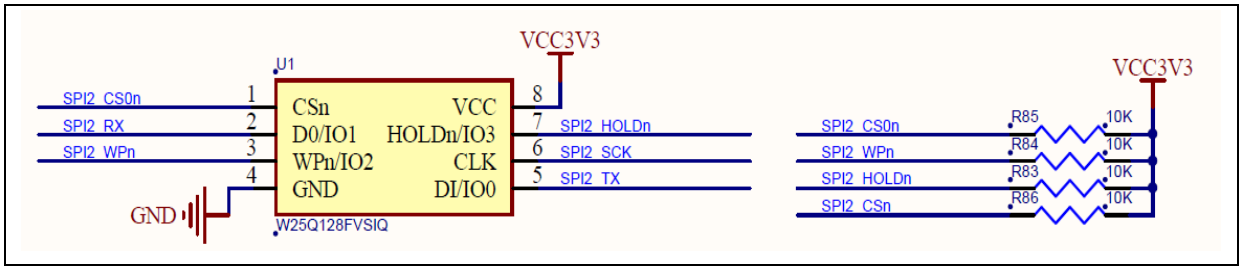
4.8.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD 和 w25q128 电路板

w25q128 建议电路如下图 26：

图 26 w25q128 建议电路



2) 软件环境：

project\at_start_f437\examples\spi\w25q_flash

4.8.3 软件设计

1) 配置流程

- 配置 SPI2 和对应 GPIO；
- 读取 flash ID；
- 擦除 flash 扇区；
- 向 flash 写入数据；
- 读取上一步 flash 写入数据的区域。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  __IO uint32_t flash_id_index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  tx_data_fill();/*填充待发送数据*/
  uart_print_init(115200);/*初始化串口打印*/
  spiflash_init();/*初始化与 flash 通信的 spi 接口：包括 spi 配置，gpio 配置*/
  flash_id_index = spiflash_read_id();/*读 flash ID*/
  if(flash_id_index != W25Q128)/*判断读到的 flash ID 是否正确*/
  {
```

```
    printf("flash id check error!\r\n");
    for(index = 0; index < 50; index++)
    {
      at32_led_toggle(LED2);
      at32_led_toggle(LED3);
      delay_ms(200);
    }
    return 1;
  }
  else
  {
    printf("flash id check success! id: %x\r\n", flash_id_index);
  }
  /* erase sector */
  spiflash_sector_erase(FLASH_TEST_ADDR / SPIF_SECTOR_SIZE);/*擦除 flash 待写入的扇区*/
  /* write data */
  spiflash_write(buffer_tx, FLASH_TEST_ADDR, BUF_SIZE);/*向 flash 写入待写入的数据*/
  /* read data */
  spiflash_read(buffer_rx, FLASH_TEST_ADDR, BUF_SIZE);/*读取上一步写入的数据*/
  /* printf read data */
  printf("Read Data: ");
  for(index = 0; index < BUF_SIZE; index++)
  {
    printf("%x ", buffer_rx[index]);
  }
  /* test result:the data check */
  transfer_status = buffer_compare(buffer_rx, buffer_tx, BUF_SIZE);/*判断读出的数据和写入的数据是否符合，以判断 flash 读写操作是否成功*/
  /* test result indicate:if success ,led2 lights */
  if(transfer_status == SUCCESS)
  {
    printf("\r\nflash data read write success!\r\n");
    at32_led_on(LED2);
  }
  else
  {
    printf("\r\nflash data read write ERROR!\r\n");
    at32_led_off(LED2);
```

```
  }
  while(1)
  {
  }
}
```

■ spi 接口配置函数和 gpio 配置函数代码描述

```
void spiflash_init(void)
{
  gpio_init_type gpio_initstructure;
  spi_init_type spi_init_struct;
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);
  /* software cs, pd0 as a general io to control flash cs */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_OUTPUT;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_0;
  gpio_init(GPIOD, &gpio_initstructure);
  /* sck */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_pins           = GPIO_PINS_1;
  gpio_init(GPIOD, &gpio_initstructure);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);
  /* miso */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_2;
  gpio_init(GPIOC, &gpio_initstructure);
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5);
  /* mosi */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOD, &gpio_initstructure);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);
  FLASH_CS_HIGH();/*先拉高 CS，避免和 flash 发生意外通信*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*开启 SPI 时钟*/
  spi_default_para_init(&spi_init_struct);
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_MSB;
```

```
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_HIGH;
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;
  spi_init(SPI2, &spi_init_struct);/*配置 SPI 接口--需和 flash 一致*/
  spi_enable(SPI2, TRUE);/*使能 SPI 接口*/
}
```

- flash 擦除函数代码描述

```
void spiflash_sector_erase(uint32_t erase_addr)
{
  erase_addr *= SPIF_SECTOR_SIZE; /*将扇区地址换算成 byte 地址*/
  spiflash_write_enable();/*向 flash 写入“写使能”命令*/
  spiflash_wait_busy();/*等待上一个命令操作完成，flash busy 标志清零*/
  FLASH_CS_LOW();/*拉低片选引脚，使能 flash*/
  spi_byte_write(SPIF_SECTORERASE);/*向 flash 写入“擦除”命令*/
  spi_byte_write((uint8_t)((erase_addr) >> 16));/*分 3 个 byte 写入 24bit 需要擦除的地址*/
  spi_byte_write((uint8_t)((erase_addr) >> 8));
  spi_byte_write((uint8_t)erase_addr);
  FLASH_CS_HIGH();/*拉高片选引脚，失能 flash*/
  spiflash_wait_busy();/*等待上一个命令操作完成，flash busy 标志清零*/
}
```

- flash 读数据函数代码描述

```
void spiflash_read(uint8_t *pbuffer, uint32_t read_addr, uint32_t length)
{
  FLASH_CS_LOW();
  spi_byte_write(SPIF_READDATA); /*向 flash 发送“读取数据”命令*/
  spi_byte_write((uint8_t)((read_addr) >> 16)); /*分 3byte 发送需要读取数据的起始地址*/
  spi_byte_write((uint8_t)((read_addr) >> 8));
  spi_byte_write((uint8_t)read_addr);
  spi_bytes_read(pbuffer, length);/*通过读取 SPI 数据寄存器读取 length 长度的数据*/
  FLASH_CS_HIGH();/*拉高片选引脚，失能 flash*/
}
```

- flash 读 ID 函数代码描述

```
uint16_t spiflash_read_id(void)
{
  uint16_t wreceivedata = 0;
  FLASH_CS_LOW();/*拉低 CS，以片选使能 flash*/
  spi_byte_write(SPIF_MANUFACTDEVICEID);/*向 flash 发送“读取 ID”命令*/
  spi_byte_write(0x00);/*提供读取 ID 需要的 dummy 时序*/
  spi_byte_write(0x00);
  spi_byte_write(0x00);
  wreceivedata |= spi_byte_read() << 8;/*读取 flash ID*/
```

```
  wreceivedata |= spi_byte_read();
  FLASH_CS_HIGH();/*拉高片选引脚，失能 flash*/
  return wreceivedata;
}
```

■ flash 写函数代码描述

```
void spiflash_write(uint8_t *pbuffer, uint32_t write_addr, uint32_t length)
{
  uint32_t sector_pos;
  uint16_t sector_offset;
  uint16_t sector_remain;
  uint16_t index;
  uint8_t *spiflash_buf;
  spiflash_buf = spiflash_sector_buf;
  /* sector address */
  sector_pos = write_addr / SPIF_SECTOR_SIZE;/*计算带写入地址的扇区号*/
  /* address offset in a sector */
  sector_offset = write_addr % SPIF_SECTOR_SIZE;/*计算待写入地址的扇区内偏移地址*/
  /* the remain in a sector */
  sector_remain = SPIF_SECTOR_SIZE - sector_offset;/*计算待写入扇区的剩余长度*/
  if(length <= sector_remain)
  {
    /* smaller than a sector size */
    sector_remain = length;/*如果待写入数据长度效于扇区剩余长度，则将剩余长度设置为待写入数据长度
*/
  }
  while(1)
  {
    /* read a sector */
    spiflash_read(spiflash_buf, sector_pos * SPIF_SECTOR_SIZE, SPIF_SECTOR_SIZE);/*读待写入地址
区域的数据*/
    /* validate the read erea */
    for(index = 0; index < sector_remain; index++)
    {
      if(spiflash_buf[sector_offset + index] != 0xFF)/*判断待写入地址区域是否为空(是否为 0xFF)*/
      {
        /* there are some data not equal 0xff, so this secotr needs erased */
        break;
      }
    }
    if(index < sector_remain)
    {
      /* erase the sector */
      spiflash_sector_erase(sector_pos);/*如果待写入地址区域非空（有非 0xFF 的数据），那么先擦除要写
入的扇区*/
```

```
        /* copy the write data */
        for(index = 0; index < sector_remain; index++)
        {
          spiflash_buf[index + sector_offset] = pbuffer[index];/*将本次待写入数据拷贝到本次的写 buffer*/
        }
        spiflash_write_nocheck(spiflash_buf, sector_pos * SPIF_SECTOR_SIZE, SPIF_SECTOR_SIZE); /*
向 flash 写入待写入的数据 */
      }
      else
      {
        /* write directly in the erased area */
        spiflash_write_nocheck(pbuffer, write_addr, sector_remain);/*如果待写入地址区域全为空（全是
0xFF），那么直接向 flash 写入待写入的数据*/
      }
      if(length == sector_remain)
      {
        /* write end */
        break;/*如果待写入的数据长度和本次写入的数据长度一样，则结束写操作*/
      }
      else
      {
        /* go on writing */
        sector_pos++;/*否则，将待写入扇区号加一*/
        sector_offset = 0;/*将下一个待写入扇区内地址偏移设为 0*/
        pbuffer += sector_remain;/*待写入的数据指针向后移 sector_remain（上次写入的数据长度）*/
        write_addr += sector_remain;/*待写入的地址指针向后移 sector_remain（上次写入的数据长度）*/
        length -= sector_remain;/*剩余待写入的数据长度减 sector_remain（上次写入的数据长度）*/
        if(length > SPIF_SECTOR_SIZE)
        {
          /* could not write the remain data in the next sector */
          sector_remain = SPIF_SECTOR_SIZE;/*如果剩余待写数据长度大于一个扇区长度，则设置下一次
待写入的长度为一个扇区长度。后续循环进行写操作*/
        }
        else
        {
          /* could write the remain data in the next sector */
          sector_remain = length;/*否则，设置下一次待写入数据长度等于剩余待写入数据长度。后续循环进
行写操作*/
        }
      }
    }
}
```

4.8.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 FLASH 通信正常。

4.9 案例 9-- SPI 使用 jtag 引脚的配置

4.9.1 功能简介

实现 SPI2 和 SPI3 之间，使用特殊的 jtag 引脚的全双工轮询通信。接线如下：

spi2		spi3
pd0(cs)	<--->	pa15(cs)
pd1(sck)	<--->	pb3 (sck)
pc2(miso)	<--->	pa13(miso)
pd4(mosi)	<--->	pa14(mosi)

4.9.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\use_jtagpin_hardwarecs_dma

4.9.3 软件设计

注意：本文仅介绍了 AT32F435/F437 的 SPI 使用 jtag 引脚的配置，其他型号的使用配置稍有不同，请参照对应型号的 BSP 使用。

1) 配置流程

- 配置 SPI2 和 SPI3 对应的 GPIO；
- 配置 SPI2 和 SPI3 通信配置；
- 开始轮询通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  delay_us(100);/*开始运行代码之前（配置 JTAG 相关引脚为 SPI 功能），加一点延时，以确保程序被完整的下载*/
  gpio_config();/*SPI2 和 SPI3 GPIO 初始化*/
  spi_config();/*配置 SPI2 和 SPI3。其中 SPI2 为从机，SPI3 为主机*/
  dma_channel_enable(DMA1_CHANNEL1, TRUE);/*使能 DMA1 CHANNEL1:对应 SPI2 发送*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 CHANNEL2:对应 SPI2 接收*/
  dma_channel_enable(DMA1_CHANNEL4, TRUE);/*使能 DMA1 CHANNEL4:对应 SPI3 发送*/
  dma_channel_enable(DMA1_CHANNEL3, TRUE);/*使能 DMA1 CHANNEL3:对应 SPI3 接收*/
  while(dma_flag_get(DMA1_FDT2_FLAG) == RESET);
  /* test result:the data check */
  transfer_status1 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, BUFFER_SIZE);/*核对 SPI2 收到的数据是否正确*/
  transfer_status2 = buffer_compare(spi3_buffer_rx, spi2_buffer_tx, BUFFER_SIZE);/*核对 SPI3 收到的数
```

```
据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ SPI 通信配置函数代码描述

```
static void spi_config(void)
{
  dma_init_type dma_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dma_reset(DMA1_CHANNEL1);/*复位 DMA1 channel1，使 channel1 处于默认配置*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dma_reset(DMA1_CHANNEL3);/*复位 DMA1 channel3，使 channel3 处于默认配置*/
  dma_reset(DMA1_CHANNEL4);/*复位 DMA1 channel4，使 channel4 处于默认配置*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_SPI2_TX);/*弹性映射配置：DMA1
channel1 对应 SPI2 发送*/
  dmamux_init(DMA1MUX_CHANNEL2, DMAMUX_DMAREQ_ID_SPI2_RX);/*弹性映射配置：DMA1
channel2 对应 SPI2 接收*/
  dmamux_init(DMA1MUX_CHANNEL3, DMAMUX_DMAREQ_ID_SPI3_TX);/*弹性映射配置：DMA1
channel3 对应 SPI3 发送*/
  dmamux_init(DMA1MUX_CHANNEL4, DMAMUX_DMAREQ_ID_SPI3_RX);/*弹性映射配置：DMA1
channel4 对应 SPI3 接收*/
  dma_default_para_init(&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = BUFFER_SIZE;/*设置 DMA buffer 长度：和 SPI 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;/*DMA 内存数据宽度：
BYTE。和自定义的 SPI 收/发 buffer 格式一致*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加
一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;/*DMA 外设数据宽
度：BYTE。和 SPI 的数据格式一致*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）
*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_tx;/*内存地址：SPI2 发送 buffer 的地址*/
```

```
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL1, &dma_init_struct);/*将 DMA1 channel1 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi2_buffer_rx;/*内存地址：SPI2 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct);/*将 DMA1 channel2 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_tx;/*内存地址：SPI3 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从内存到外设*/
  dma_init(DMA1_CHANNEL3, &dma_init_struct);/*将 DMA1 channel3 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)spi3_buffer_rx;/*内存地址：SPI3 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL4, &dma_init_struct);/*将 DMA1 channel4 设置为以上配置*/
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟*/
  spi_default_para_init(&spi_init_struct);/*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;/*配置 SPI 为全双工模式*/
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;/*配置 SPI 为主机模式*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 时钟分频为 8 分频*/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_LSB;/*配置 SPI 数据格式为 LSB（低位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;/*配置 SPI 数据位数为每笔 8bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲时为低电平*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个时钟边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_HARDWARE_MODE;/*配置 SPI 为硬件 CS 管理模式*/
  spi_init(SPI3, &spi_init_struct);/*将 SPI3 设置为以上配置*/
  spi_init_struct.master_slave_mode = SPI_MODE_SLAVE;/*配置 SPI 为从机模式*/
  spi_init(SPI2, &spi_init_struct);/*将 SPI2 设置为以上配置*/
  spi_hardware_cs_output_enable(SPI3, TRUE);/*使能 SPI3 的 CS 引脚为输出模式*/
  spi_i2s_dma_transmitter_enable(SPI2,TRUE);/*使能 SPI2 DMA 发送功能*/
  spi_i2s_dma_transmitter_enable(SPI3,TRUE);/*使能 SPI3 DMA 发送功能*/
  spi_i2s_dma_receiver_enable(SPI2,TRUE);/*使能 SPI2 DMA 接收功能*/
  spi_i2s_dma_receiver_enable(SPI3,TRUE);/*使能 SPI3 DMA 接收功能*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
}
```

■ SPI GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE); /*使能 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE); /*使能 GPIOB 时钟*/
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*使能 GPIOC 时钟*/
```

```
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*使能 GPIOD 时钟*/
  gpio_default_para_init(&gpio_initstructure);/*将 GPIO 初始化变量设置为默认值*/
  /* spi3 cs pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;/*配置 GPIO 为推挽模式*/
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;/*配置 GPIO 驱动能力为强*/
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;/*配置 GPIO 上拉*/
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;/*配置 GPIO 为复用*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_15;/*选取 pin15*/
  gpio_init(GPIOA, &gpio_initstructure);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE15, GPIO_MUX_6); /*配置 PA15 复用为 MUC_6 功能，即为 SPI3_CS 功能*/
  /* spi3 sck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN; /*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_3; /*选取 pin3*/
  gpio_init(GPIOB, &gpio_initstructure);
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE3, GPIO_MUX_6); /*配置 PB3 复用为 MUC_6 功能，即为 SPI3_SCK 功能*/
  /* spi3 miso pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP; /*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_13; /*选取 pin13*/
  gpio_init(GPIOA, &gpio_initstructure);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE13, GPIO_MUX_6); /*配置 PA13 复用为 MUC_6 功能，即为 SPI3_MISO 功能*/
  /* spi3 mosi pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP; /*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_14; /*选取 pin14*/
  gpio_init(GPIOA, &gpio_initstructure);
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE14, GPIO_MUX_6); /*配置 PA14 复用为 MUC_6 功能，即为 SPI3_MOSI 功能*/
  /* spi2 cs pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP; /*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_0; /*选取 pin0*/
  gpio_init(GPIOD, &gpio_initstructure);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7); /*配置 PD0 复用为 MUC_7 功能，即为 SPI2_CS 功能*/
  /* spi2 sck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN; /*配置 GPIO 下拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_1; /*选取 pin1*/
  gpio_init(GPIOD, &gpio_initstructure);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6); /*配置 PD1 复用为 MUC_6 功能，即为 SPI2_SCK 功能*/
  /* spi2 miso pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP; /*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins           = GPIO_PINS_2; /*选取 pin2*/
  gpio_init(GPIOC, &gpio_initstructure);
```

```
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_5); /*配置 PC2 复用为 MUC_5 功能，即为 SPI2_MISO 功能*/
  /* spi2 mosi pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP; /*配置 GPIO 上拉*/
  gpio_initstructure.gpio_pins              = GPIO_PINS_4; /*选取 pin4*/
  gpio_init(GPIOD, &gpio_initstructure);
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);} /*配置 D4 复用为 MUC_6 功能，即为 SPI2_MOSI 功能*/
```

4.9.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 SPI2 和 SPI3 通信正常。

5 I2S 案例

5.1 案例 1-- I²S 半双工 DMA 方式通信

5.1.1 功能简介

实现 I²S2 和 I²S3 之间的半双工 DMA 通信。接线如下：

I²S2		I²S3
pd0(ws)	<--->	pa4(ws)
pd1(ck)	<--->	pc10(ck)
pd4(sd)	<--->	pc12(sd)

5.1.2 资源准备

1) 硬件环境：

一块 AT32F437 的 AT-START BOARD

2) 软件环境：

project\at_start_f437\examples\i2s\halfduplex_dma

5.1.3 软件设计

1) 配置流程

- 配置 I²S2 和 I²S3 对应的 GPIO；
- 配置 I²S2 和 I²S3 的通信配置和 DMA 配置；
- 开始 DMA 通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*I2S2 和 I2S3 相关 GPIO 初始化*/
  i2s_config();/*I2S2 I2S3 初始化（含 I2S2 使能）*/
  i2s_enable(SPI3, TRUE);/*使能 SPI3（即 I2S3）*/
  while(dma_flag_get(DMA1_FDT1_FLAG) == RESET);/*等待 I2S2 接收完成*/
  transfer_status = buffer_compare(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 收到的数据是否正确*/
  if(transfer_status==SUCCESS)
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
```

```
  }
}
```

■ I2S 配置及 DMA 配置函数代码描述

```
static void i2s_config(void)
{
  dma_init_type dma_init_struct;
  i2s_init_type i2s_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dma_reset(DMA1_CHANNEL1);/*复位 DMA1 channel1，使 channel1 处于默认配置*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_SPI2_RX);/*弹性映射配置：DMA1 channel1 对应 SPI2 接收（也就是 I2S 接收）*/
  dmamux_init(DMA1MUX_CHANNEL2, DMAMUX_DMAREQ_ID_SPI3_TX);/*弹性映射配置：DMA1 channel2 对应 SPI3 发送（也就是 I2S 发送）*/
  dma_default_para_init(&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = 32;/*设置 DMA buffer 长度：和 I2S 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;/*DMA 内存数据宽度：半字。*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;/*DMA 外设数据宽度：半字*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/
  dma_init_struct.memory_base_addr = (uint32_t)i2s3_buffer_tx;/*内存地址：I2S3 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt);/*外设地址：SPI3 数据寄存器地址（I2S3 和 SPI3 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct);/*将 DMA1 channel2 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)i2s2_buffer_rx;/*内存地址：I2S2 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt);/*外设地址：SPI2 数据寄存器地址（I2S2 和 SPI2 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY;/*数据传输方向：从外设到内存*/
  dma_init(DMA1_CHANNEL1, &dma_init_struct);/*将 DMA1 channel1 设置为以上配置*/
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟（也就是使能 I2S3 时钟）*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟（也就是使能 I2S2 时钟）*/
  i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;/*设置 I2S 为飞利浦标准*/
  i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;/*设置 I2S 帧格式为：数据位数为 16bit，声道位数为 32bit*/
```

```
  i2s_init_struct.mclk_output_enable = TRUE;/*使能 MCLK 时钟输出*/
  i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;/*配置 I2S 采样率为 48K*/
  i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;/*设置时钟空闲电平为低电平*/
  i2s_init_struct.operation_mode = I2S_MODE_MASTER_TX;/*设置 I2S 为主机发送模式*/
  i2s_init(SPI3, &i2s_init_struct);/*将 SPI3(也就是 I2S3)设置为以上配置*/
  i2s_init_struct.operation_mode =I2S_MODE_SLAVE_RX;/*设置 I2S 为从机接收模式*/
  i2s_init(SPI2, &i2s_init_struct);/*将 SPI2(也就是 I2S2)设置为以上配置*/
  dma_channel_enable(DMA1_CHANNEL1, TRUE);/*使能 DMA1 channel1*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 channel2*/
  spi_i2s_dma_receiver_enable(SPI2, TRUE);/*使能 SPI2（即 I2S2）DMA 接收使能*/
  spi_i2s_dma_transmitter_enable(SPI3, TRUE);/*使能 SPI3（即 I2S3）DMA 发送使能*/
  i2s_enable(SPI2, TRUE);/*使能 SPI2（即 I2S2）*/
}
```

■ GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);/*开启 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*开启 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*开启 GPIOD 时钟*/
  /* master ws pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOA, &gpio_initstructure);/*配置 PA4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PA4 复用为 mux6，也就是
I2S3 WS 功能*/

  /* master ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC10 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 mux6，也
就是 I2S3 CK 功能*/
  /* master sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_12;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC12 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 mux6，也
就是 I2S3 SD 功能*/
  /* master mck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
```

```
  gpio_initstructure.gpio_pins              = GPIO_PINS_7;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC7 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE7, GPIO_MUX_6);/*配置 PC12 复用为 mux6，也就是 I2S3 MCK 功能*/
  /* slave ws pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins              = GPIO_PINS_0;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD0 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7);/*配置 PD0 复用为 mux7，也就是 I2S2 WS 功能*/
  /* slave ck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD1 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD1 复用为 mux6，也就是 I2S2 CK 功能*/
  /* slave sd pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 mux6，也就是 I2S2 CK 功能*/
}
```

5.1.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 I²S2 和 I²S3 通信正常。

5.2 案例 2-- I²S 半双工中断方式通信

5.2.1 功能简介

实现 I²S2 和 I²S3 之间的半双工中断方式通信。接线如下：

I²S2		I²S3
pd0(ws)	<--->	pa4(ws)
pd1(ck)	<--->	pc10(ck)
pd4(sd)	<--->	pc12(sd)

5.2.2 资源准备

1) 硬件环境：

   一块 AT32F437 的 AT-START BOARD

2) 软件环境：

   project\at_start_f437\examples\i2s\halfduplex_interrupt

5.2.3 软件设计

1) 配置流程

- 配置 I²S2 和 I²S3 对应的 GPIO；
- 配置 I²S2 和 I²S3 的通信配置；
- 配置中断函数；
- 开始中断通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*I2S2 和 I2S3 相关 GPIO 初始化*/
  i2s_config(I2S_DATA_16BIT_CHANNEL_32BIT, I2S_AUDIO_FREQUENCY_48K);/*配置 I2S2 和 I2S3 帧格式为：16bit 数据，32bit 声道；采样率为 48K*/
  while(rx_index < 32);/*等待 I2S2 接收完成 32 笔数据*/
  /* test result:the data check */
  transfer_status1 = buffer_compare(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 接收到的数据是否正确*/
  for(index = 0; index < 32; index++)  i2s2_buffer_rx[index] = 0;/*将 I2S2 接收数据 buffer 清零*/
  tx_index = 0;/*将 I2S3 发送计数清零*/
  rx_index = 0;/*将 I2S2 接收计数清零*/
  i2s_config(I2S_DATA_24BIT_CHANNEL_32BIT, I2S_AUDIO_FREQUENCY_16K); /*配置 I2S2 和 I2S3 帧格式为：24bit 数据，32bit 声道；采样率为 16K*/
  while(rx_index < 32);
  /* test result:the data check */
  transfer_status2 = buffer_compare_24bits(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

- I2S 配置函数代码描述

```
static void i2s_config(void)
{
```

```
  i2s_init_type i2s_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟（也就是使能 I2S3 时钟）*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟（也就是使能 I2S2 时钟）*/
  nvic_irq_enable(SPI3_I2S3EXT_IRQn, 0, 0);/*配置和使能 SPI3（即 I2S3）中断*/
  nvic_irq_enable(SPI2_I2S2EXT_IRQn, 0, 0);/*配置和使能 SPI2（即 I2S2）中断*/
  spi_i2s_reset(SPI2);/*复位 SPI2（即 I2S2），以将相关寄存器恢复到默认配置*/
  spi_i2s_reset(SPI3);/*复位 SPI2（即 I2S2），以将相关寄存器恢复到默认配置*/
  i2s_default_para_init(&i2s_init_struct);/*给 I2S 初始化变量赋默认值*/
  i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;/*设置 I2S 为飞利浦标准*/
  i2s_init_struct.data_channel_format = format;/*按照 format 设置 I2S 帧格式*/
  i2s_init_struct.mclk_output_enable = TRUE;/*使能 MCLK 时钟输出*/
  i2s_init_struct.audio_sampling_freq = freq;/*按照 freq 配置 I2S 采样率*/
  i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;/*设置时钟空闲电平为低电平*/
  i2s_init_struct.operation_mode = I2S_MODE_MASTER_TX;/*设置 I2S 为主机发送模式*/
  i2s_init(SPI3, &i2s_init_struct);/*将 SPI3(也就是 I2S3)设置为以上配置*/
  i2s_init_struct.operation_mode =I2S_MODE_SLAVE_RX;/*设置 I2S 为从机接收模式*/
  i2s_init(SPI2, &i2s_init_struct);/*将 SPI2(也就是 I2S2)设置为以上配置*/
  spi_i2s_interrupt_enable(SPI2, SPI_I2S_RDBF_INT, TRUE);/*使能 SPI2（即 I2S2）接收中断*/
  spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, TRUE);/*使能 SPI3（即 I2S3）发送中断*/
  i2s_enable(SPI2, TRUE);/*使能 SPI2（即 I2S2）*/
  i2s_enable(SPI3, TRUE);/*使能 SPI3（即 I2S3）*/
}
```

■ I2S 中断配置函数代码描述

```
/*SPI2(即 I2S2)中断处理函数*/
void SPI2_I2S2EXT_IRQHandler(void)
{
  if(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) != RESET) /*判断是否 SPI2(即 I2S2)接收缓冲区满标志置起*/
  {
    i2s2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2); /*读取一笔收到的数据*/
  }
}

/*SPI3(即 I2S3)中断处理函数*/
void SPI3_I2S3EXT_IRQHandler(void)
{
  if(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) != RESET) /*判断是否 SPI3(即 I2S3)发送缓冲区空标志置起*/
  {
    spi_i2s_data_transmit(SPI3, i2s3_buffer_tx[tx_index++]);/*发送一笔待发送数据*/
    if(tx_index == 32)
    {
```

```
        spi_i2s_interrupt_enable(SPI3, SPI_I2S_TDBE_INT, FALSE); /*发送完预期的 32 笔数据后，关闭
SPI3 发送缓冲区空中断*/
      }
    }
}
```

■ GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);/*开启 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);/*开启 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);/*开启 GPIOD 时钟*/
  /* master ws pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOA, &gpio_initstructure);/*配置 PA4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PA4 复用为 mux6，也就是
I2S3 WS 功能*/
  /* master ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC10 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6);/*配置 PC10 复用为 mux6，也
就是 I2S3 CK 功能*/
  /* master sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_12;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC12 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6);/*配置 PC12 复用为 mux6，也
就是 I2S3 SD 功能*/
  /* master mck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_7;
  gpio_init(GPIOC, &gpio_initstructure);/*配置 PC7 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE7, GPIO_MUX_6);/*配置 PC12 复用为 mux6，也就
是 I2S3 MCK 功能*/
  /* slave ws pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_0;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD0 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7);/*配置 PD0 复用为 mux7，也就是
```

```
I2S2 WS 功能*/
  /* slave ck pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins              = GPIO_PINS_1;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD1 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6);/*配置 PD1 复用为 mux6，也就是
I2S2 CK 功能*/
  /* slave sd pin */
  gpio_initstructure.gpio_pull              = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins              = GPIO_PINS_4;
  gpio_init(GPIOD, &gpio_initstructure);/*配置 PD4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6);/*配置 PD4 复用为 mux6，也就是
I2S2 CK 功能*/
}
```

5.2.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 I²S2 和 I²S3 通信正常。

5.3 案例 3-- AT32F403A/F407/F435/F437 I2S 全双工 DMA 方式通信

5.3.1 功能简介

实现 I²S2 和 I²S3 之间的全双工 DMA 通信。接线如下：

I²S2		I²S3
pd0(ws)	<--->	pa4 (ws)
pd1(sck)	<--->	pc10(sck)
pc2(ext_sd) rx	<--->	pc11(ext_sd) tx
pd4(sd) tx	<--->	pc12(sd) rx

5.3.2 资源准备

1) 硬件环境：

   一块 AT32F437 的 AT-START BOARD

2) 软件环境：

   project\at_start_xxxx\examples\i2s\fullduplex_dma

5.3.3 软件设计

1) 配置流程

   - 配置 I²S2 和 I²S3 对应的 GPIO；
   - 配置 I²S2 和 I²S3 的通信配置和 DMA 配置；
   - 开始 DMA 通信。

2) 代码介绍

   - main 函数代码描述

```
int main(void)
{
```

```
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*I2S2 和 I2S3 相关 GPIO 初始化*/
  i2s_config();/*I2S2 I2S3 初始化*/
  i2s_enable(SPI3, TRUE); /*使能 SPI3（即 I2S3）*/
  i2s_enable(I2S3EXT, TRUE); /*使能 I2S3 扩展模块*/
  i2s_enable(I2S2EXT, TRUE); /*使能 SPI2（即 I2S2）*/
  i2s_enable(SPI2, TRUE); /*使能 I2S2 扩展模块*/
  while(dma_flag_get(DMA1_FDT2_FLAG) == RESET);/*等待 I2S2 接收完成*/
  transfer_status = buffer_compare(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 收到的数据是否正确*/
  transfer_status2 = buffer_compare(i2s3_buffer_rx, i2s2_buffer_tx, TXBUF_SIZE); /*核对 I2S3 收到的数据是否正确*/
  if((transfer_status1 == SUCCESS) &&(transfer_status2 == SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ I2S 配置及 DMA 配置函数代码描述

```
static void i2s_config(void)
{
  dma_init_type dma_init_struct;
  i2s_init_type i2s_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dma_reset(DMA1_CHANNEL1);/*复位 DMA1 channel1，使 channel1 处于默认配置*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dma_reset(DMA1_CHANNEL3);/*复位 DMA1 channel3，使 channel3 处于默认配置*/
  dma_reset(DMA1_CHANNEL4);/*复位 DMA1 channel4，使 channel4 处于默认配置*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dmamux_init(DMA1MUX_CHANNEL1, DMAMUX_DMAREQ_ID_SPI2_TX); /*弹性映射配置：DMA1 channel1 对应 SPI2 发送（也就是 I2S2 发送）*/
  dmamux_init(DMA1MUX_CHANNEL2, DMAMUX_DMAREQ_ID_I2S2_EXT_RX); /*弹性映射配置：DMA1 channel2 对应 I2S2_EXT 接收*/
  dmamux_init(DMA1MUX_CHANNEL3, DMAMUX_DMAREQ_ID_SPI3_RX); /*弹性映射配置：DMA1 channel3 对应 SPI3 接收（也就是 I2S3 接收）*/
  dmamux_init(DMA1MUX_CHANNEL4, DMAMUX_DMAREQ_ID_I2S3_EXT_TX); /*弹性映射配置：DMA1 channel4 对应 I2S3_EXT 发送*/
```

```
  dma_default_para_init(&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = 32;/*设置 DMA buffer 长度：和 I2S 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;/*DMA 内存数据宽度：半字。*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;/*DMA 外设数据宽度：半字*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/

  dma_init_struct.memory_base_addr = (uint32_t)i2s2_buffer_tx; /*内存地址：I2S2 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt); /*外设地址：SPI2 数据寄存器地址（I2S2 和 SPI2 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL;/*数据传输方向：内存到外设*/
  dma_init(DMA1_CHANNEL1, &dma_init_struct); /*将 DMA1 channel1 设置为以上配置*/

  dma_init_struct.memory_base_addr = (uint32_t)i2s2_buffer_rx; /*内存地址：I2S2 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(I2S2EXT->dt); /*外设地址：I2S2EXT 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY; /*数据传输方向：外设到内存*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct); /*将 DMA1 channel2 设置为以上配置*/

  dma_init_struct.memory_base_addr = (uint32_t)i2s3_buffer_rx; /*内存地址：I2S3 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI3->dt); /*外设地址：SP3 数据寄存器地址（I2S3 和 SPI3 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY; /*数据传输方向：外设到内存*/
  dma_init(DMA1_CHANNEL3, &dma_init_struct); /*将 DMA1 channel3 设置为以上配置*/

  dma_init_struct.memory_base_addr = (uint32_t)i2s3_buffer_tx; /*内存地址：I2S3 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(I2S3EXT->dt); /*外设地址：I2S3EXT 数据寄存器地址*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*数据传输方向：内存到外设*/
  dma_init(DMA1_CHANNEL4, &dma_init_struct); /*将 DMA1 channel4 设置为以上配置*/

  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟（也就是使能 I2S3 时钟）*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟（也就是使能 I2S2 时钟）*/
  i2s_default_para_init(&i2s_init_struct);/*给 I2S 初始化变量赋默认值*/
  i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;/*设置 I2S 为飞利浦标准*/
  i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;/*设置 I2S 帧格式为：数据位数为 16bit，声道位数为 32bit*/
  i2s_init_struct.mclk_output_enable = TRUE;/*使能 MCLK 时钟输出*/
```

```
  i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;/*配置 I2S 采样率为 48K*/
  i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;/*设置时钟空闲电平为低电平*/
  i2s_init_struct.operation_mode =I2S_MODE_MASTER_TX;/*设置 I2S 为主机发送模式*/
  i2s_init(SPI2, &i2s_init_struct);/*将 SPI2(也就是 I2S2)设置为以上配置*/
  i2s_init_struct.operation_mode =I2S_MODE_SLAVE_RX; /*设置 I2SEXT 为从机接收模式*/
  i2s_init(I2S2EXT, &i2s_init_struct);
  i2s_init_struct.operation_mode = I2S_MODE_SLAVE_RX;/*设置 I2S 为从机接收模式*/
  i2s_init(SPI3, &i2s_init_struct);/*将 SPI3(也就是 I2S3)设置为以上配置*/
  i2s_init_struct.operation_mode =I2S_MODE_SLAVE_TX; /*设置 I2SEXT 为从机发送模式*/
  i2s_init(I2S3EXT, &i2s_init_struct);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);/*使能 DMA1 channel1*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 channel2*/
  dma_channel_enable(DMA1_CHANNEL3, TRUE);/*使能 DMA1 channel3*/
  dma_channel_enable(DMA1_CHANNEL4, TRUE);/*使能 DMA1 channel4*/
  spi_i2s_dma_transmitter_enable(SPI2, TRUE); /*使能 SPI2（即 I2S2）DMA 发送使能*/
  spi_i2s_dma_receiver_enable(I2S2EXT, TRUE); /*使能 I2S2EXT DMA 接收使能*/
  spi_i2s_dma_receiver_enable(SPI3, TRUE); /*使能 SPI3（即 I2S3）DMA 接收使能*/
  spi_i2s_dma_transmitter_enable(I2S3EXT, TRUE); /*使能 I2S3EXT DMA 发送使能*/
}
```

■ GPIO 配置函数代码描述

```
static void gpio_config(void)
{
gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE); /*开启 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE); /*开启 GPIOC 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE); /*开启 GPIOD 时钟*/
  /* master ws pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_0;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD0 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7); /*配置 PD0 复用为 mux7，也就
是 I2S2 WS 功能*/
  /* master ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins           = GPIO_PINS_1;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD1 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6); /*配置 PD1 复用为 mux6，也就
是 I2S2 CK 功能*/
   /* master ext_sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_2;
```

```
  gpio_init(GPIOC, &gpio_initstructure); /*配置 PC2 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_6); /*配置 PC2 复用为 mux6，也就是 I2S2 EXT_SD 功能*/
  /* master sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6); /*配置 PD4 复用为 mux6，也就是 I2S2 SD 功能*/
  /* slave ws pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA4 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE4, GPIO_MUX_6); /*配置 PA4 复用为 mux6，也就是 I2S3 WS 功能*/
  /* slave ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins           = GPIO_PINS_10;
  gpio_init(GPIOC, &gpio_initstructure); /*配置 PC10 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE10, GPIO_MUX_6); /*配置 PC10 复用为 mux6，也就是 I2S3 CK 功能*/
  /* slave ext_sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_11;
  gpio_init(GPIOC, &gpio_initstructure); /*配置 PC11 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE11, GPIO_MUX_5); /*配置 PC11 复用为 mux5，也就是 I2S3 EXT_SD 功能*/
  /* slave sd pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins           = GPIO_PINS_12;
  gpio_init(GPIOC, &gpio_initstructure); /*配置 PC12 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE12, GPIO_MUX_6); /*配置 PC12 复用为 mux6，也就是 I2S3 SD 功能*/
```

5.3.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 I²S2 和 I²S3 全双工通信正常。

5.4 案例 4-- AT32F425 I2S 全双工 DMA 方式通信

5.4.1 功能简介

实现 I²S1 和 I²S2 组成一个全双工 I²S，组成的全双工 I²S 接口定义如下：

pa4(ws)
pa5(ck)
pa6(mck)
pa7(sd tx)

pb14(ext_sd rx)

由于 AT32F425 只有 3 个 spi（I²S），只能组成一个全双工 I²S，所以不能在同一块 AT-START 上用两个全双工 I²S 进行通信。所以本案例使用一个全双工 I²S 进行自收自发通信，接线如下：

pa4(ws) <---> NC

pa5(ck) <---> NC

pa6(mck) <---> NC

pa7(sd) <---> pb14(ext_sd)

用户也可以准备两块 AT-START 板子对测。使用两块板子时，需要注意一块 AT-START 配置为主机，一块 AT-START 配置为从机。本案例中的代码仅展示配置为主机，在一块 AT-START 上进行自发自收的配置。

5.4.2 资源准备

1) 硬件环境：

一块 AT32F425 的 AT-START BOARD

2) 软件环境：

project\at_start_f425\examples\i2s\fullduplex_dma

5.4.3 软件设计

1) 配置流程

- 配置 I²S1 和 I²S2 对应的 GPIO；
- 配置 I²S1 和 I²S2 的通信配置和 DMA 配置；
- 开始 DMA 通信。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  tx_data_fill();/*填充待发送数据 buffer*/
  gpio_config();/*I2S2 和 I2S3 相关 GPIO 初始化*/
  i2s_config();/*I2S2 I2S3 初始化*/
  i2s_enable(SPI1, TRUE); /*使能 SPI1（即 I2S1）*/
  i2s_enable(SPI2, TRUE); /*使能 SPI2（即 I2S2）*/
  while(dma_flag_get(DMA1_FDT2_FLAG) == RESET);/*等待 I2S 发送完成*/
  while(dma_flag_get(DMA1_FDT3_FLAG) == RESET); /*等待 I2S 接收完成*/
  transfer_status1 = buffer_compare(i2s2_buffer_rx, i2s1_buffer_tx, TXBUF_SIZE); /*核对 I2S 收到的数据是否正确*/
  if(transfer_status1 == SUCCESS)
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
```

```
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ I2S 配置及 DMA 配置函数代码描述

```
static void i2s_config(void)
{
  dma_init_type dma_init_struct;
  i2s_init_type i2s_init_struct;
  crm_periph_clock_enable(CRM_DMA1_PERIPH_CLOCK, TRUE);/*使能 DMA1 时钟*/
  dma_reset(DMA1_CHANNEL2);/*复位 DMA1 channel2，使 channel2 处于默认配置*/
  dma_reset(DMA1_CHANNEL3);/*复位 DMA1 channel3，使 channel3 处于默认配置*/
  dmamux_enable(DMA1, TRUE);/*开启 DMA1 弹性映射功能*/
  dma_flexible_config(DMA1, FLEX_CHANNEL2, DMA_FLEXIBLE_SPI1_TX); /*弹性映射配置：DMA1 channel2 对应 SPI1 发送，也就是 I2S1 发送，也就是 I2S 全双工的发送*/
  dma_flexible_config(DMA1, FLEX_CHANNEL3, DMA_FLEXIBLE_SPI2_RX); /*弹性映射配置：DMA1 channel3 对应 SPI2 接收，也就是 I2S2 接收，也就是 I2S 全双工的接收*/
  dma_default_para_init (&dma_init_struct);/*将 DMA 初始化变量置为默认值*/
  dma_init_struct.buffer_size = TXBUF_SIZE;/*设置 DMA buffer 长度：和 I2S 通信数据长度一致*/
  dma_init_struct.memory_data_width = DMA_MEMORY_DATA_WIDTH_HALFWORD;/*DMA 内存数据宽度：半字。*/
  dma_init_struct.memory_inc_enable = TRUE;/*内存地址自增：使能（每收/发一个数据后，内存地址要加一）*/
  dma_init_struct.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_HALFWORD;/*DMA 外设数据宽度：半字*/
  dma_init_struct.peripheral_inc_enable = FALSE;/*外设地址自增：关闭（一直是 SPI 数据寄存器，不变）*/
  dma_init_struct.priority = DMA_PRIORITY_HIGH;/*优先级：高优先级*/
  dma_init_struct.loop_mode_enable = FALSE;/*循环模式：关闭*/
  dma_init_struct.memory_base_addr = (uint32_t)i2s1_buffer_tx; /*内存地址：I2S 发送 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI1->dt); /*外设地址：I2S1 数据寄存器地址（I2S1 和 SPI1 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_MEMORY_TO_PERIPHERAL; /*数据传输方向：内存到外设*/
  dma_init(DMA1_CHANNEL2, &dma_init_struct); /*将 DMA1 channel2 设置为以上配置*/
  dma_init_struct.memory_base_addr = (uint32_t)i2s2_buffer_rx; /*内存地址：I2S 接收 buffer 的地址*/
  dma_init_struct.peripheral_base_addr = (uint32_t)&(SPI2->dt); /*外设地址：I2S2 数据寄存器地址（I2S1 和 SPI1 共用数据寄存器）*/
  dma_init_struct.direction = DMA_DIR_PERIPHERAL_TO_MEMORY; /*数据传输方向：外设到内存*/
  dma_init(DMA1_CHANNEL3, &dma_init_struct); /*将 DMA1 channel3 设置为以上配置*/
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE);/*使能 SPI3 时钟（也就是使能 I2S3 时钟）*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE);/*使能 SPI2 时钟（也就是使能 I2S2 时
```

```
钟）*/
  crm_periph_clock_enable(CRM_SCFG_PERIPH_CLOCK, TRUE);/*使能系统配置时钟，用于后续配置 I2S 全双工组合选择*/
  scfg_i2s_full_duplex_config(SCFG_FULL_DUPLEX_I2S1_I2S2);/*配置 I2S1 和 I2S2 组成全双工 I2S。其中 I2S1 作为主导方，提供 WS、CK、MCK 和 SD；I2S2 作为附属方，仅提供 SD_EXT*/
  i2s_default_para_init(&i2s_init_struct);/*给 I2S 初始化变量赋默认值*/
  i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;/*设置 I2S 为飞利浦标准*/
  i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;/*设置 I2S 帧格式为：数据位数为 16bit，声道位数为 32bit*/
  i2s_init_struct.mclk_output_enable = TRUE;/*使能 MCLK 时钟输出*/
  i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;/*配置 I2S 采样率为 48K*/
  i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW;/*设置时钟空闲电平为低电平*/
  i2s_init_struct.operation_mode =I2S_MODE_MASTER_TX;/*设置 I2S 为主机发送模式*/
  i2s_init(SPI1, &i2s_init_struct);/*将 SPI1(也就是 I2S1)设置为以上配置*/
  i2s_init_struct.operation_mode = I2S_MODE_SLAVE_RX;/*设置 I2S 为从机接收模式*/
  i2s_init(SPI2, &i2s_init_struct);/*将 SPI2(也就是 I2S2)设置为以上配置*/
  dma_channel_enable(DMA1_CHANNEL2, TRUE);/*使能 DMA1 channel2*/
  dma_channel_enable(DMA1_CHANNEL3, TRUE);/*使能 DMA1 channel3*/
  spi_i2s_dma_transmitter_enable(SPI1, TRUE); /*使能 SPI1（即 I2S1）DMA 发送使能*/
  spi_i2s_dma_receiver_enable(SPI2, TRUE); /*使能 SPI2（即 I2S2）DMA 接收使能*/
}
```

■ GPIO 配置函数代码描述

```
static void gpio_config(void)
{
gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE); /*开启 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE); /*开启 GPIOB 时钟*/
  /* master ws pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_4;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA4 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE4, GPIO_MUX_0); /*配置 PA4 复用为 mux0，也就是 I2S WS 功能*/
  /* master ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins           = GPIO_PINS_5;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA5 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE5, GPIO_MUX_0); /*配置 PA5 复用为 mux0，也就是 I2S CK 功能*/
   /* master mck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
```

```
  gpio_initstructure.gpio_pins                = GPIO_PINS_6;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA6 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE6, GPIO_MUX_0); /*配置 PA6 复用为 mux0，也就是
I2S mck 功能*/
  /* master sd pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins                = GPIO_PINS_7;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA7 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE7, GPIO_MUX_0); /*配置 PA7 复用为 mux0，也就是
I2S SD 功能*/
  /* slave ext_sd pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins                = GPIO_PINS_14;
  gpio_init(GPIOB, &gpio_initstructure); /*配置 PB14 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE14, GPIO_MUX_3); /*配置 PB14 复用为 mux3，也
就是 I2S EXT_SD 功能*/
```

5.4.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 I²S 全双工通信正常。

5.5 案例 5-- I²S 和 SPI 功能切换通信

5.5.1 功能简介

实现 SPI2 和 SPI3 之间通信切换到 I²S2 和 I²S3 之间通信。接线如下：

spi2/i2s2		spi3/i2s3
pd0(cs/ws)	<--->	pa15(cs/ws)
pd1(sck/ck)	<--->	pb3(sck/ck)
pd4(mosi/sd)	<--->	pb5(mosi/sd)

5.5.2 资源准备

1) 硬件环境：

   一块 AT32F437 的 AT-START BOARD

2) 软件环境：

   project\at_start_f437\examples\i2s\spii2s_switch_halfduplex_polling

5.5.3 软件设计

1) 配置流程

- 配置 SPI2/I²S2 和 SPI3/I²S3 对应的 GPIO；
- 配置 I²S2 和 I²S3 的通信配置（主发从收）；
- 开始 I²S2 和 I²S3 的轮询通信，并核对数据；
- 配置 SPI2 和 SPI3 的通信配置（SPI3 为主机半双工只发；SPI2 为从机单向只收）；
- 开始 SPI2 和 SPI3 的轮询通信，并核对数据；
- 配置 I²S2 和 I²S3 的通信配置（主收从发）；

- 开始 I²S2 和 I²S3 的轮询通信，并核对数据。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  __IO uint32_t index = 0;
  system_clock_config();/*系统时钟配置*/
  at32_board_init();/*LED 初始化*/
  gpio_config();/*I2S2 和 I2S3 相关 GPIO 初始化*/
  i2s_config(I2S_MODE_MASTER_TX, I2S_MODE_SLAVE_RX);/*将 SPI2/3 接口配置为 I2S 模式。且 SPI3 作为主机发送方；SPI2 作为从机接收方。*/
  i2s_enable(SPI2, TRUE);/*使能 SPI2，即 I2S2*/
  i2s_enable(SPI3, TRUE);/*使能 SPI3，即 I2S3*/
  while(rx_index < 32)/*轮询收/发 32 次*/
  {
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);
    spi_i2s_data_transmit(SPI3, i2s3_buffer_tx[tx_index++]);/*I2S3 发送数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);
    i2s2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2);/*I2S2 接收数据*/
  }
  /* test result:the data check */
  transfer_status1 = buffer_compare(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 收到的数据是否正确*/
  tx_index = 0;/*将发送计数清零*/
  rx_index = 0;/*将接收计数清零*/
  spi_config();/*将 SPI2/3 接口配置为 SPI 模式。且 SPI3 为主机半双工只发模式；SPI2 为从机单向只收模式*/
  while(rx_index < 32)/*轮询收/发 32 次*/
  {
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);
    spi_i2s_data_transmit(SPI3, spi3_buffer_tx[tx_index++]);/*SPI3 发送数据*/
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);
    spi2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2); /*SPI2 接收数据*/
  }

  /* test result:the data check */
  transfer_status2 = buffer_compare(spi2_buffer_rx, spi3_buffer_tx, 32); /*核对 SPI2 收到的数据是否正确*/
  for(index = 0; index < 32; index++)  i2s2_buffer_rx[index] = 0;/*将 I2S2 接收 buffer 清零*/
  tx_index = 0; /*将发送计数清零*/
  rx_index = 0; /*将接收计数清零*/
  i2s_config(I2S_MODE_SLAVE_TX, I2S_MODE_MASTER_RX); /*将 SPI2/3 接口配置为 I2S 模式。且 I2S2 作为主机接收方；I2S3 作为从机发送方。*/
  while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);/*等待从机 I2S3 发送缓冲区为空后据*/
  spi_i2s_data_transmit(SPI3, i2s3_buffer_tx[tx_index++]);/*先往数据寄存器预填一笔待发送数*/
  i2s_enable(SPI3, TRUE);/*再使能从机 I2S3*/
  i2s_enable(SPI2, TRUE);/*  再使能主机 I2S2*/
```

```
  while(rx_index < 32)/*轮询收/发 32 笔数据*/
  {
    while(spi_i2s_flag_get(SPI2, SPI_I2S_RDBF_FLAG) == RESET);
    i2s2_buffer_rx[rx_index++] = spi_i2s_data_receive(SPI2);/* I2S2 接收数据*/
    while(spi_i2s_flag_get(SPI3, SPI_I2S_TDBE_FLAG) == RESET);
    spi_i2s_data_transmit(SPI3, i2s3_buffer_tx[tx_index++]);/* I2S3 发送数据*/
  }
  /* test result:the data check */
  transfer_status3 = buffer_compare(i2s2_buffer_rx, i2s3_buffer_tx, 32);/*核对 I2S2 收到的数据是否正确*/
  /* test result indicate:if success ,led2 lights */
  if((transfer_status1 == SUCCESS) && (transfer_status2 == SUCCESS) && (transfer_status3 ==
SUCCESS))
  {
    at32_led_on(LED2);/*都正确则点亮 LED2*/
  }
  else
  {
    at32_led_off(LED2);/*否则关闭 LED2*/
  }
  while(1)
  {
  }
}
```

■ I2S 配置函数代码描述

```
static void i2s_config(i2s_operation_mode_type i2s3_mode, i2s_operation_mode_type i2s2_mode)
{
  i2s_init_type i2s_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE); /*使能 SPI3 时钟（也就是使能 I2S3 时
钟）*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE); /*使能 SPI2 时钟（也就是使能 I2S2 时
钟）*/
  spi_i2s_reset(SPI2); /*复位 SPI2（即 I2S2），以将相关寄存器恢复到默认配置*/
  spi_i2s_reset(SPI3); /*复位 SPI3（即 I2S3），以将相关寄存器恢复到默认配置*/
  i2s_default_para_init(&i2s_init_struct);/*给 I2S 初始化变量赋默认值*/
  i2s_init_struct.audio_protocol = I2S_AUDIO_PROTOCOL_PHILLIPS;/*设置 I2S 为飞利浦标准*/
  i2s_init_struct.data_channel_format = I2S_DATA_16BIT_CHANNEL_32BIT;/*设置 I2S 帧格式为：数据
6bit，声道 32bit*/
  i2s_init_struct.mclk_output_enable = FALSE; /*关闭 MCLK 时钟输出*/
  i2s_init_struct.audio_sampling_freq = I2S_AUDIO_FREQUENCY_48K;/*设置 I2S 采样率为 48K*/
  i2s_init_struct.clock_polarity = I2S_CLOCK_POLARITY_LOW; /*设置时钟空闲电平为低电平*/
  i2s_init_struct.operation_mode = i2s3_mode;/*按照 i2s3_mode  设置 I2S3 模式*/
  i2s_init(SPI3, &i2s_init_struct); /*将 SPI3(也就是 I2S3)设置为以上配置*/
  i2s_init_struct.operation_mode =i2s2_mode; /*按照 i2s2_mode  设置 I2S2 模式*/
  i2s_init(SPI2, &i2s_init_struct); /*将 SPI2(也就是 I2S2)设置为以上配置*/
```

```
}
```

■ SPI 配置函数代码描述

```
static void spi_config(void)
{
  spi_init_type spi_init_struct;
  crm_periph_clock_enable(CRM_SPI3_PERIPH_CLOCK, TRUE); /*使能 SPI3 时钟*/
  crm_periph_clock_enable(CRM_SPI2_PERIPH_CLOCK, TRUE); /*使能 SPI2 时钟*/
  spi_i2s_reset(SPI2); /*复位 SPI2，以将相关寄存器恢复到默认配置*/
  spi_i2s_reset(SPI3); /*复位 SPI3，以将相关寄存器恢复到默认配置*/
  spi_default_para_init(&spi_init_struct); /*给 SPI 初始化变量赋默认值*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_HALF_DUPLEX_TX;/*配置 SPI 为半双工只发模式*/
  spi_init_struct.master_slave_mode =SPI_MODE_MASTER;/*配置 SPI 为主机*/
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_8;/*配置 SPI 分频系数为 8/
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_MSB;/*配置 SPI 数据传输格式为 MSB（高位在前）*/
  spi_init_struct.frame_bit_num = SPI_FRAME_16BIT; /*配置 SPI 数据格式为 16bit*/
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_LOW;/*配置 SPI 时钟空闲电平为低*/
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;/*配置 SPI 在第二个边沿采样*/
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;/*配置 SPI CS 管理为软件模式*/
  spi_init(SPI3, &spi_init_struct);/*设置 SPI3 为以上配置*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_SIMPLEX_RX;/*配置 SPI 为只收模式*/
  spi_init_struct.master_slave_mode =SPI_MODE_SLAVE;/*配置 SPI 为从机*/
  spi_init(SPI2, &spi_init_struct); /*设置 SPI2 为以上配置*/
  spi_enable(SPI2, TRUE);/*使能 SPI2*/
  spi_enable(SPI3, TRUE);/*使能 SPI3*/
}
```

■ GPIO 配置函数代码描述

```
static void gpio_config(void)
{
  gpio_init_type gpio_initstructure;
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE); /*开启 GPIOA 时钟*/
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE); /*开启 GPIOB 时钟*/
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE); /*开启 GPIOD 时钟*/
  /* spi3/i2s3 ws pin */
  gpio_initstructure.gpio_out_type       = GPIO_OUTPUT_PUSH_PULL;
  gpio_initstructure.gpio_pull           = GPIO_PULL_UP;
  gpio_initstructure.gpio_mode           = GPIO_MODE_MUX;
  gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_initstructure.gpio_pins           = GPIO_PINS_15;
  gpio_init(GPIOA, &gpio_initstructure); /*配置 PA15 为上拉复用引脚*/
  gpio_pin_mux_config(GPIOA, GPIO_PINS_SOURCE15, GPIO_MUX_6); /*配置 PA15 复用为 mux6，也就是 I2S3 WS/SPI3 CS 功能*/
  /* spi3/i2s3 ck pin */
  gpio_initstructure.gpio_pull           = GPIO_PULL_DOWN;
```

```
  gpio_initstructure.gpio_pins                = GPIO_PINS_3;
  gpio_init(GPIOB, &gpio_initstructure); /*配置 PB3 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE3, GPIO_MUX_6); /*配置 PB3 复用为 mux6，也就是 I2S3 CK/SPI3 SCK 功能*/
  /* spi3/i2s3 sd pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins                = GPIO_PINS_5;
  gpio_init(GPIOB, &gpio_initstructure); /*配置 PB5 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE5, GPIO_MUX_6); /*配置 PB5 复用为 mux6，也就是 I2S3 SD/SPI3 MOSI 功能*/
  /* spi2/i2s2 ws pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins                = GPIO_PINS_0;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD0 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_7); /*配置 PD0 复用为 mux7，也就是 I2S2 SD/SPI2 MOSI 功能*/
  /* spi2/i2s2 ck pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_DOWN;
  gpio_initstructure.gpio_pins                = GPIO_PINS_1;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD1 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE1, GPIO_MUX_6); /*配置 PD1 复用为 mux6，也就是 I2S2 CK/SPI2 SCK 功能*/
  /* spi2/i2s2 sd pin */
  gpio_initstructure.gpio_pull                = GPIO_PULL_UP;
  gpio_initstructure.gpio_pins                = GPIO_PINS_4;
  gpio_init(GPIOD, &gpio_initstructure); /*配置 PD4 为下拉复用引脚*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_6); /*配置 PD4 复用为 mux6，也就是 I2S2 SD/SPI2 MOSI 功能*/
}
```

5.5.4 实验效果

AT-START BOARD 的 LED2 亮起，说明 I²S2 和 I²S3 通信正常，SPI2 和 SPI3 也通信正常。

6 文档版本历史

表 4 文档版本历史

日期	版本	变更
2022.01.07	2.0.0	最初版本
2022.05.12	2.0.1	1、更改“I2S时钟控制器”章节的采样率计算公式； 2、更改“SPI读写FLASH案例”的SPI和对应GPIO。

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A) 对安全性有特别要求的应用，如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。