## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Author** | **Changes** |
| 2022/4/3 | 1.0 | Luoming | Initial version. |

**一．IP Description:**

1. **The Block Diagram**



## **Feature set: （功能列表）：**

* Compatible with Motorola’s SPI specification
* 4 to 32 bits SPI master/slave support
* Master and slave support TI mode
* APB compatible interface
* Support DMA 4 words burst or single word transfer
* Full duplex synchronous operation
* Configurable master or slave support
* 16bit Programmable Baud Rate Generator
* The SPI clock can be up to “pclk/2” (pclk is APB CLOCK) in SPI master mode, and can be up to “pclk/4” in SPI slave mode (that is APB 120-MHz Clock Rate for up to 30-Mbaud Operation)
* Serial clock with programmable polarity and phase
* MSB or LSB first data transmit and receive
* Support one master and multi-slave operation
* Support 4-32bits data to be synchronously transmitted and received simultaneously in slave mode
* 32 words depth FIFO for buffered transmit and receive
* Interrupt-driven operation:

Transmitter FIFO empty, Transmitter FIFO underrun,

Receiver Data Available, Receiver overrun, RX FIFO Full,

Receive Completed in SPI master mode, Transmitter empty

## 二．**Feature comparison with STM32: （跟STM32的功能比较）**

**1. Chip-Platform:**

|  |  |  |  |
| --- | --- | --- | --- |
| 符号描述：   1. ✅支持 2. ❌不支持 3. ❌硬件有，但SDK计划不支持 | AT103 | STM32 |  |
| 3线全双工同步传输 | ✅ | ✅ |  |
| 带或不带第三根双向数据线的双线单工同步传输 | ❌ | ✅ |  |
| 8或16位传输帧格式选择 | ✅ | ✅ |  |
| 主或从操作 | ✅ | ✅ |  |
| 支持多主模式 | ❌ | ✅ |  |
| 8个主模式波特率预分频系数(最大为fPCLK/2) | ✅ | ✅ |  |
| 从模式频率 (最大为fPCLK/2) | (最大为fPCLK/4) | ✅ |  |
| 主模式和从模式下均可以由软件或硬件进行NSS管理：主/从操作模式的动态改变 | ❌ | ✅ |  |
| 可编程的时钟极性和相位 | ✅ | ✅ |  |
| 可编程的数据顺序，MSB在前或LSB在前 | ✅ | ✅ |  |
| 可触发中断的专用发送和接收标志 | ✅ | ✅ |  |
| SPI总线忙状态标志 | ❌ | ✅ |  |
| 支持可靠通信的硬件CRC  ─ 在发送模式下，CRC值可以被作为最后一个字节发送  ─ 在全双工模式中对接收到的最后一个字节自动进行CRC校验 | ❌ | ✅ |  |
| 可触发中断的主模式故障、过载以及CRC错误标志 | ✅无CRC错误标志 | ✅ |  |
| 支持DMA功能的1字节发送或接收缓冲器：产生发送和接受请求 | ✅支持DMA功能的多字节发送和接收缓冲器：产生发送和接受请求 | ✅ |  |
| 支持I2S音频协议接口 | ❌ | ✅大容量产品和互联型产品上 |  |

**2.API Design:**

|  |  |  |
| --- | --- | --- |
|  | AT103 | STM32F10x\_StdPeriph\_Lib\_V3.5.0 |
| Init  /de-init | **SPI\_I2S\_DeInit(SPI\_TypeDef\* SPIx);**  **SPI\_Init(SPI\_TypeDef\* SPIx,**  **SPI\_InitTypeDef\* SPI\_InitStruct);**  **SPI\_StructInit(SPI\_InitTypeDef\* SPI\_InitStruct);** | **void SPI\_I2S\_DeInit(SPI\_TypeDef\* SPIx);**  **void SPI\_Init(SPI\_TypeDef\* SPIx,**  **SPI\_InitTypeDef\* SPI\_InitStruct);**  **void SPI\_StructInit(SPI\_InitTypeDef\* SPI\_InitStruct);** |
| Config | **void SPI\_Cmd(SPI\_TypeDef\* SPIx,**  **FunctionalState NewState);**  **void SPI\_DataSizeConfig(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_DataSize);**  **void SPI\_BiDirectionalLineConfig(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_Direction);**  **Void FlagStatus SPI\_I2S\_GetFlagStatus(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_I2S\_FLAG);** | **void SPI\_Cmd(SPI\_TypeDef\* SPIx,**  **FunctionalState NewState);**  **void SPI\_DataSizeConfig(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_DataSize);**  **void SPI\_BiDirectionalLineConfig(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_Direction);**  **Void FlagStatus SPI\_I2S\_GetFlagStatus(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_I2S\_FLAG);**  **void SPI\_I2S\_ClearFlag(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_I2S\_FLAG);** |
| DMA | **void SPI\_I2S\_DMACmd(SPI\_TypeDef\* SPIx,**  **uint32\_t SPI\_I2S\_DMAReq，**  **FunctionalState NewState)；** | **void SPI\_I2S\_DMACmd(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_I2S\_DMAReq,**  **FunctionalState NewState);** |
| INT | **SPI\_I2S\_ITConfig(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT,**  **FunctionalState NewState);**  **ITStatus SPI\_I2S\_GetITStatus(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT);**  **void SPI\_I2S\_ClearITPendingBit(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT)；** | **SPI\_I2S\_ITConfig(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT,**  **FunctionalState NewState);**  **ITStatus SPI\_I2S\_GetITStatus(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT);**  **void SPI\_I2S\_ClearITPendingBit(SPI\_TypeDef\* SPIx,**  **uint8\_t SPI\_I2S\_IT)；** |
| Tx/Rx  Data | **void SPI\_I2S\_SendData(SPI\_TypeDef\* SPIx,**  **uint16\_t Data);**  **uint16\_t SPI\_I2S\_ReceiveData(SPI\_TypeDef\* SPIx);** | **void SPI\_I2S\_SendData(SPI\_TypeDef\* SPIx,**  **uint16\_t Data);**  **uint16\_t SPI\_I2S\_ReceiveData(SPI\_TypeDef\* SPIx);** |
| NSS | **void SPI\_SSOutputCmd(SPI\_TypeDef\* SPIx,**  **FunctionalState NewState);** | **void SPI\_NSSInternalSoftwareConfig(SPI\_TypeDef\* SPIx,**  **uint16\_t SPI\_NSSInternalSoft);**  **void SPI\_SSOutputCmd(SPI\_TypeDef\* SPIx,**  **FunctionalState NewState);** |
| CRC |  | **void SPI\_TransmitCRC(SPI\_TypeDef\* SPIx);**  **void SPI\_CalculateCRC(SPI\_TypeDef\* SPIx,**  **FunctionalState NewState);**  **uint16\_t SPI\_GetCRC(SPI\_TypeDef\* SPIx, uint8\_t SPI\_CRC);**  **uint16\_t SPI\_GetCRCPolynomial(SPI\_TypeDef\* SPIx);** |
| I2S |  | **void I2S\_Init(SPI\_TypeDef\* SPIx,**  **I2S\_InitTypeDef\* I2S\_InitStruct);**  **void I2S\_StructInit(I2S\_InitTypeDef\* I2S\_InitStruct);**  **void I2S\_Cmd(SPI\_TypeDef\* SPIx, FunctionalState NewState);** |

1. **AT103-SDK design**
2. **Driver**

**1.1 Data structure design:**

**1.1.1 at103.h/SPI\_TypeDef : SPI MemMap & RegistorDescriptor**

/\*\*

\* @brief SPI Peripheral Interface

\*/

typedef struct

{

} SPI\_TypeDef;

#define SPI1 ((SPI\_TypeDef \*)SPI1\_BASE)

#define SPI2 ((SPI\_TypeDef \*)SPI2\_BASE)

**1.1.2 at103\_spi.h/ SPI\_InitTypeDef**

/\*\*

\* @brief SPI Init Structure definition

\*/

typedef struct

{

uint16\_t SPI\_Direction;

uint16\_t SPI\_Mode;

uint16\_t SPI\_DataSize;

uint16\_t SPI\_CPOL;

uint16\_t SPI\_CPHA;

uint16\_t SPI\_NSS;

uint16\_t SPI\_BaudRatePrescaler;

uint16\_t SPI\_FirstBit;

uint16\_t SPI\_CRCPolynomial;

}SPI\_InitTypeDef;

**1.1.3 at103\_spi.c/library fuction**

**1.1.3.1函数SPI\_DeInit**

|  |  |
| --- | --- |
| 函数名 | SPI\_DeInit |
| 函数原形 | void SPI\_DeInit(SPI\_TypeDef\* SPIx) |
| 功能描述 | 将外设 SPIx 寄存器重设为缺省值 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 对 SPI1，RCC\_APB2PeriphClockCmd().  SPI2，RCC\_APB1PeriphClockCmd(). |

**1.1.3.2 函数SPI\_Init**

|  |  |
| --- | --- |
| 函数名 | SPI\_Init |
| 函数原形 | void SPI\_Init(SPI\_TypeDef\* SPIx, SPI\_InitTypeDef\* SPI\_InitStruct) |
| 功能描述 | 根据 SPI\_InitStruct 中指定的参数初始化外设 SPIx 寄存器 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.3 函数**SPI\_StructInit

|  |  |
| --- | --- |
| 函数名 | SPI\_StructInit |
| 函数原形 | void SPI\_StructInit(SPI\_InitTypeDef\* SPI\_InitStruct) |
| 功能描述 | 把 SPI\_InitStruct 中的每一个参数按缺省值填入 |
| 输入参数 | SPI\_InitStruct：指向结构 SPI\_InitTypeDef 的指针，待初始化 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.4 函数**SPI\_ Cmd

|  |  |
| --- | --- |
| 函数名 | SPI\_ Cmd |
| 函数原形 | void SPI\_Cmd(SPI\_TypeDef\* SPIx, FunctionalState NewState) |
| 功能描述 | 使能或者失能 SPI 外设 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  NewState: 外设 SPIx 的新状态 ENABLE 或者 DISABLE。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.5 函数**SPI\_ITConfig

|  |  |
| --- | --- |
| 函数名 | SPI\_ITConfig |
| 函数原形 | void SPI\_ITConfig(SPI\_TypeDef\* SPIx, u16 SPI\_IT, FunctionalState NewState) |
| 功能描述 | 使能或者失能 SPI 外设 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_IT：待使能或者失能的 SPI 中断源;同时使能SPI中断；  NewState: SPIx 中断的新状态 ENABLE 或者 DISABLE。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.6 函数**SPI\_ DMACmd

|  |  |
| --- | --- |
| 函数名 | SPI\_ DMACmd |
| 函数原形 | void SPI\_DMACmd(SPI\_TypeDef\* SPIx, u16 SPI\_DMAReq, FunctionalState NewState) |
| 功能描述 | 使能或者失能指定 SPI 的 DMA 请求 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_DMAReq：待使能或者失能的 SPI DMA 传输请求;设定txtlf或rxtlf;  NewState: SPIx DMA 传输的新状态 ENABLE 或者 DISABLE.。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.7 函数**SPI\_ SendData

|  |  |
| --- | --- |
| 函数名 | SPI\_ SendData |
| 函数原形 | void SPI\_SendData(SPI\_TypeDef\* SPIx, u16 Data) |
| 功能描述 | 通过外设 SPIx 发送一个数据 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  Data: 待发送的数据。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.8 函数**SPI\_ ReceiveData

|  |  |
| --- | --- |
| 函数名 | SPI\_ ReceiveData |
| 函数原形 | u16 SPI\_ReceiveData(SPI\_TypeDef\* SPIx) |
| 功能描述 | 返回通过 SPIx 最近接收的数据 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设 |
| 输出参数 | 无 |
| 返回值 | 接收到的字 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.9 函数**SPI\_SSOutputCmd

|  |  |
| --- | --- |
| 函数名 | SPI\_SSOutputCmd |
| 函数原形 | void SPI\_SSOutputCmd(SPI\_TypeDef\* SPIx, FunctionalState NewState) |
| 功能描述 | 使能或者失能指定的 SPI SS 输出 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  NewState:使能或者失能指定的 SPI SS 输出ENABLE 或者 DISABLE。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.10 函数**SPI\_DataSizeConfig

|  |  |
| --- | --- |
| 函数名 | SPI\_DataSizeConfig |
| 函数原形 | void SPI\_DataSizeConfig(SPI\_TypeDef\* SPIx, u16 SPI\_DatSize) |
| 功能描述 | 设置选定的 SPI 数据大小 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_DataSize：SPI 数据大小。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.11 函数**SPI\_BiDirectionalLineConfig

|  |  |
| --- | --- |
| 函数名 | SPI\_BiDirectionalLineConfig |
| 函数原形 | void SPI\_BiDirectionalLineConfig(SPI\_TypeDef\* SPIx, u16 SPI\_Direction) |
| 功能描述 | 选择指定 SPI 在双向模式下的数据传输方向 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_Direction:选择SPI在双向模式下的数据传输方向。 |
| 输出参数 | 无 |
| 返回值 | 无 |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.12 函数**SPI\_I2S\_GetFlagStatus

|  |  |
| --- | --- |
| 函数名 | SPI\_I2S\_GetFlagStatus |
| 函数原形 | FlagStatus SPI\_I2S\_GetFlagStatus(SPI\_TypeDef\* SPIx, uint16\_t SPI\_I2S\_FLAG) |
| 功能描述 | 检查指定的 SPI 标志位设置与否 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_I2S\_FLAG：待检查的 SPI 标志位 。 |
| 输出参数 | 无 |
| 返回值 | SPI\_I2S\_FLAG 的新状态（SET 或者 RESET） |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.13 函数**SPI\_I2S\_GetITStatus

|  |  |
| --- | --- |
| 函数名 | SPI\_I2S\_GetITStatus |
| 函数原形 | ITStatus SPI\_I2S\_GetITStatus(SPI\_TypeDef\* SPIx, uint8\_t SPI\_I2S\_IT) |
| 功能描述 | 检查指定的 SPI 中断发生与否 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_I2S\_IT：待检查的 SPI 中断源 。 |
| 输出参数 | 无 |
| 返回值 | SPI\_IT 的新状态 （SET 或者 RESET） |
| 先决条件 | 无 |
| 被调用函数 | 无 |

**1.1.3.14 函数**SPI\_I2S\_ClearITPendingBit

|  |  |
| --- | --- |
| 函数名 | SPI\_I2S\_ClearITPendingBit |
| 函数原形 | void SPI\_I2S\_ClearITPendingBit(SPI\_TypeDef\* SPIx, uint8\_t SPI\_I2S\_IT) |
| 功能描述 | 清除 SPIx 的中断待处理位 |
| 输入参数 | SPIx：x 可以是 1 或者 2，来选择 SPI 外设;  SPI\_I2S\_IT：待清除的 SPI 中断源 。 |
| 输出参数 | 无 |
| 返回值 | SPI\_IT 的新状态 （SET 或者 RESET） |
| 先决条件 | 无 |
| 被调用函数 | 无 |

1. **Examples**
   1. **./project/examples/spi/main.c**
      1. **功能概述**

用查询标志方式实现SPI1和SPI2之间的四线双工通信。

第一步：SPI1作为主机，SPI2 作为从机,完成它们之间的32Bytes数据传输，检查SPI1接收的数据和SPI2发送的数据，SPI2接收的数据和SPI1发送的数据，是否一致，debug 输出结果。

第二步：SPI1作为从机，SPI2 作为主机,完成它们之间的32Bytes数据传输，检查SPI2接收的数据和SPI1发送的数据，SPI1接收的数据和SPI2发送的数据，是否一致，debug 输出结果。

* + 1. **使用库函数**

**SPI\_Init(SPIy, &SPI\_InitStructure);**

**SPI\_Cmd(SPIy, ENABLE);**

**SPI\_SSOutputCmd(SPIy, ENABLE);**

**SPI\_I2S\_GetFlagStatus(SPIy, SPI\_I2S\_FLAG\_TXE)；**

**SPI\_I2S\_SendData(SPIz, SPIz\_Buffer\_Tx[TxIdx])；**

**SPI\_I2S\_ReceiveData(SPIy)；**

* 1. **./project/examples/spi\_interrupt/main.c**
     1. **功能概述**

用中断方式实现SPI1和SPI2之间的三线单工通信。

第一步：SPI1作为主机发送，SPI2 作为从机接收,完成它们之间的单向32Bytes数据传输，检查SPI2接收的数据和SPI1发送的数据，是否一致，debug 输出结果。

第二步：SPI1作为从机接收，SPI2 作为主机发送,完成它们之间的单向32Bytes数据传输，检查SPI1接收的数据和SPI2发送的数据，是否一致，debug 输出结果。

* + 1. **使用库函数**

**SPI\_I2S\_DeInit(SPIy);**

**SPI\_Init(SPIy, &SPI\_InitStructure);**

**SPI\_I2S\_ClearITPendingBit(SPIy, SPI\_I2S\_IT\_ALL);**

**SPI\_I2S\_ITConfig(SPIy, SPI\_I2S\_IT\_TXE, ENABLE);**

**SPI\_Cmd(SPIy, ENABLE);**

**SPI\_SSOutputCmd(SPIy, ENABLE);**

**SPI\_I2S\_SendData(SPIz, SPIz\_Buffer\_Tx[TxIdx])；**

**SPI\_I2S\_ReceiveData(SPIy)；**

* 1. **./project/examples/spi\_dma/main.c**
     1. **功能概述**

用dma传输方式实现SPI1和SPI2之间的四线双工通信。

SPI1的TXFIFO通过DMA channel1 与memory建立DMA通道，RXFIFO通过DMA channel2 与memory建立DMA通道;SPI2的TXFIFO通过DMA channel3 与memory建立DMA通道，RXFIFO通过DMA channel4 与memory建立DMA通道;配置SPI和DMA,触发DMA通道传输，SPI1作为SPI主机，SPI2作为SPI从机，通过SPI总线收发64byte数据，SPI通信完成后，检查memory中SPI1接收数据缓冲和SPI2发送数据缓冲，SPI1发送数据缓冲和SPI2接收数据缓冲，是否一致，debug 输出结果。

* + 1. **使用库函数**

**DMA\_Cmd(DMA\_Channelx, DISABLE);**

**DMA\_DeInit(DMA\_Channelx);**

**DMA\_StructInit(&DMA\_InitStructure);**

**DMA\_Init(DMA\_Channelx, &DMA\_InitStructure);**

**SPI\_I2S\_DeInit(SPIx);**

**SPI\_Init(SPx, &SPI\_InitStructure);**

**SPI\_I2S\_DMACmd(SPI\_MASTER, SPI\_I2S\_DMAReq\_Tx, ENABLE);**

**DMA\_Cmd(DMA\_Channelx, ENABLE);**

**SPI\_Cmd(SPIx, ENABLE);**

**SPI\_SSOutputCmd(SPIx, ENABLE);**

**DMA\_GetFlagStatus(DMA\_FLAG\_TEx)；**

**DMA\_ClearFlag(DMA\_FLAG\_GLx);**