文档版本	V1.0.0
发布日期	20221026

APT32F110x 基于 CSI 库 SPI 应用指南

目录

1 概述

本文介绍了在APT32F110x中使用SPI的应用范例。

2. 适用的硬件

该例程使用于 APT32F110x 系列学习板

3. 应用方案代码说明

3.1 SPI 配置

基于 APT32F110x 完整的 CSI 库文件系统，可以对 SPI 进行配置。

- **硬件配置：**

SPI 是可以配置为主机或者从机接口模块，可以用来跟其它外设进行同步串行通讯，发送和接收都有一个 16 位宽，8 地址深的 FIFO。具有独立可控制的发送 FIFO 中断，接收 FIFO 中断和溢出中断，支持内部环回测试模式。兼容摩托罗拉(Motorola) SPI。

注意 SPI 的主/从模式通信，首先硬件接线正确，CS-CS，CLK-CLK，MISO，MOSI。主机模式频率和从机模式频率要参数一致，比如波特率、数据长度。

1 主 1 从连接方式：

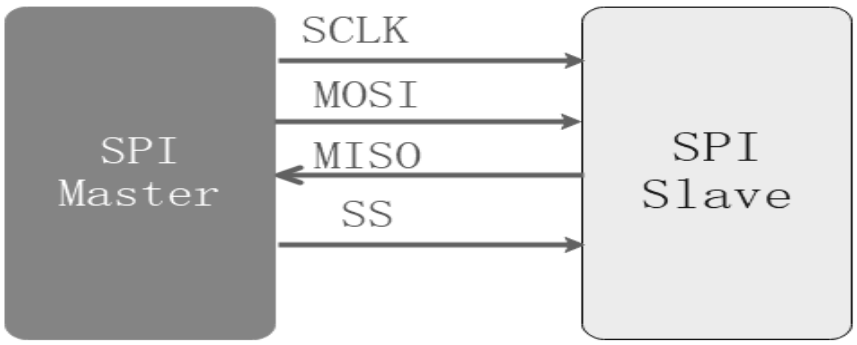

图 3.1.1 SPI 1 主 1 从定义

1 主多从连接方式：

时钟、数据脚都并接，主机通过 CS 脚来控制不同的设备使能。

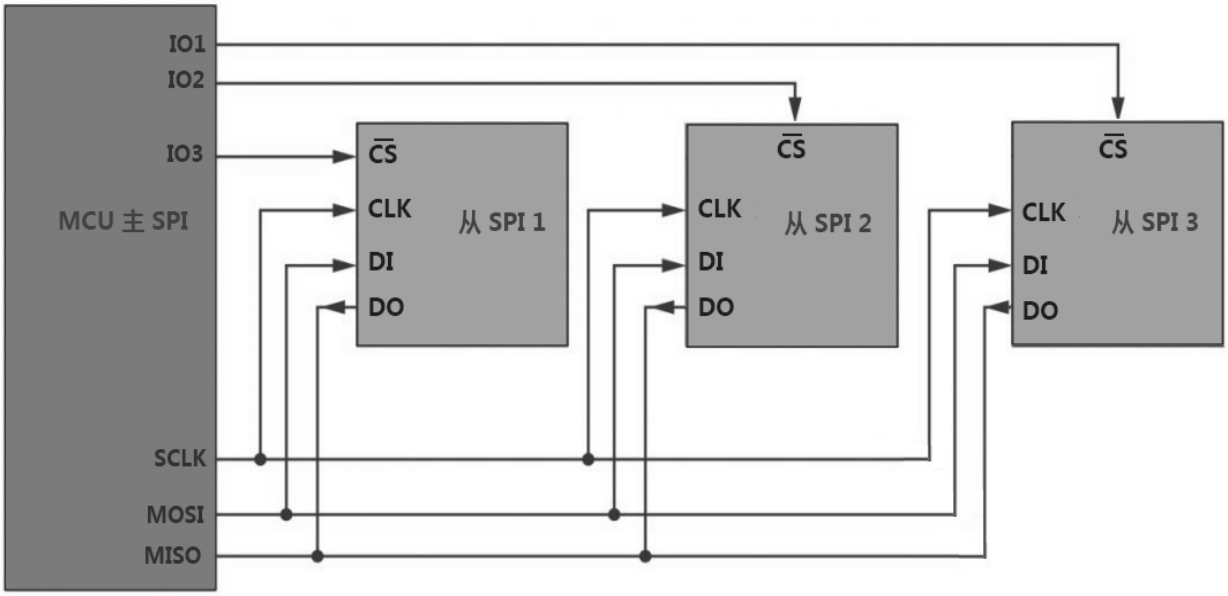

图 3.1.2 SPI　1 主多从定义

- **管脚描述：**

管脚名称	功能	I/O类型	说明
SPICLK	SPI 串行时钟	I/O	–
SPIMOSI	主机输出从机输入	I/O	–
SPIMISO	主机输入从机输出	I/O	–
SPIFSS	帧，从机选择 (作为主机时) 帧输入 (作为从机时)	I/O	–

图 3.1.3 SPI 管脚定义

SPICLK：串行时钟（主机输出）

SPIMOSI：主输出从机输入或主机输出从机输入（主机输出的数据）

SPIMISO：主输入从输出或主输入从输出（从输出的数据输出）

SPIFSS：从机选择（通常为低电平有效，主机输出）

时钟：

从机模式下， PCLK ≥ 12 x SPICLK .主机模式下，PCLK ≥ 2 x SPICLK.

举例：主机模式，要产生 1MHz 的比特率（SPICLK = 1MHz），PCLK 至少为 2MHz

3.2 SPI 主机配置

系统时钟为内部主频 48MHz，SPI 设置主机模式，进行测试数据发送。

- **管脚配置：**

SCLK--PA0.8 / MOSI--PA0.6 / MISO--PA0.9 / NSS--PA0.7

```
void spi_config(void)
{
    uint8_t bySendData[16] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
    csi_spi_config_t t_SpiConfig;
    //
    csi_pin_set_mux(PA07, PA07_OUTPUT);
    csi_pin_output_mode(PA07, GPIO_PUSH_PULL);
    csi_spi_nss_high(PA07);

    csi_pin_set_mux(PA08,PA08_SPI0_SCK);
    csi_pin_set_mux(PA09,PA09_SPI0_MISO);
    csi_pin_set_mux(PA06,PA06_SPI0_MOSI);

    t_SpiConfig.bySpiMode = SPI_MASTER;
    t_SpiConfig.bySpiPolarityPhase = SPI_FORMAT_CPOL0_CPHA1;
    t_SpiConfig.bySpiFrameLen = SPI_FRAME_LEN_8;
    t_SpiConfig.wSpiBaud = 2000000;
    t_SpiConfig.byInt= SPI_INTSRC_NONE;
    t_SpiConfig.byTxMode = SPI_TX_MODE_POLL;
    csi_spi_init(SPI0,&t_SpiConfig);

    my_printf("the spi master send demo!\n");
    mdelay(500);
}
void main()
{
    system_init();
    board_init();
            while(1)
            {
            csi_spi_nss_low(PA07);
             csi_spi_send(SPI0, bySendData,16);
             csi_spi_nss_high(PA07);
             mdelay(100);
```

```
    }
}
```

- **代码说明:**

  csi_pin_set_mux(PA07, PA07_OUTPUT); ----- 配置 IO 为输出，片选

  csi_pin_set_mux(PA08,PA08_SPI0_SCK); ----- 配置 IO 为 SPI SCK

  csi_pin_set_mux(PA09,PA09_SPI0_MISO); ----- 配置 IO 为 SPI MISO

  csi_pin_set_mux(PA06,PA06_SPI0_MOSI); ----- 配置 IO 为 SPI MOSI

  csi_spi_init(); ----- 配置 SPI 初始化

  csi_spi_send(); ----- 发送数据

- **函数参数说明：**

  csi_spi_init(SPI0,&t_SpiConfig);

  对 spi 初始化参数的结构体进行配置 **csi_spi_config_t t_SpiConfig;**

  t_SpiConfig.bySpiMode ----配置 SPI 模式，主机/从机

  t_SpiConfig.bySpiPolarityPhase ----SPI 空闲电平及相位

  t_SpiConfig.bySpiFrameLen ---- 帧数据长度

  t_SpiConfig.wSpiBaud -----通讯速率

  t_SpiConfig.byInt ----配置是否开启中断

  t_SpiConfig.byTxMode ----发送模式

- **测试波形：**

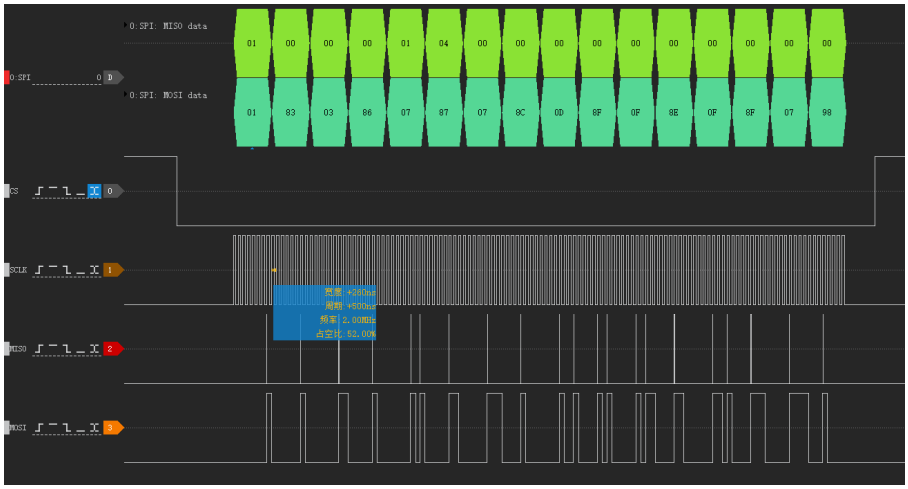

图 3.2.1 SPI 主发从收波形

3.3 SPI 从机配置

系统时钟选用内部主频 48MHZ，在中断中进行接收数据。

- **管脚配置：**

SCLK--PA0.8 / MOSI--PA0.6 / MISO--PA0.9 / NSS--PA0.7

```
void spi_slave_receive_int_demo(void)
{
    uint8_t byReceData[16] = {0};
    uint8_t byIndex = 0;
    csi_spi_config_t t_SpiConfig;

    //端口配置
    csi_pin_set_mux(PA07,PA07_SPI0_NSS);
    csi_pin_set_mux(PA08,PA08_SPI0_SCK);
    csi_pin_set_mux(PA09,PA09_SPI0_MISO);
    csi_pin_set_mux(PA06,PA06_SPI0_MOSI);

    t_SpiConfig.bySpiMode = SPI_SLAVE;
    t_SpiConfig.bySpiPolarityPhase = SPI_FORMAT_CPOL0_CPHA1;
    t_SpiConfig.bySpiFrameLen = SPI_FRAME_LEN_8;
    t_SpiConfig.wSpiBaud = 2000000;
    t_SpiConfig.byInt=   SPI_INTSRC_RXIM;
    t_SpiConfig.byRxMode = SPI_RX_MODE_INT;
    csi_spi_init(SPI0,&t_SpiConfig);

    my_printf("the spi slave receive int demo!\n");
    mdelay(100);
```

```
}
void main()
{
    system_init();
    board_init();
    while(1)
    {
        csi_spi_receive(SPI0,byReceData,16);
        while(SPI_STATE_BUSY==csi_spi_get_state(SPI_RECV));
        my_printf("\nslave receive data is:");
        for(byIndex = 0; byIndex<16; byIndex++)
        {
            my_printf(" %d", byReceData[byIndex]);
        }
    }
}
```

- **代码说明：**

  csi_spi_init(); ----- 用于配置 SPI 初始化

- **函数参数说明：**

  csi_spi_init(SPI0,&t_SpiConfig);

  对 spi 初始化参数的结构体进行配置 **csi_spi_config_t t_SpiConfig**，解析同上;

- **接收数据：**

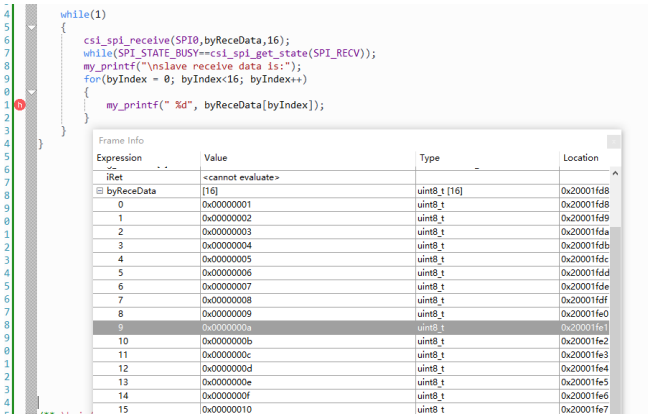

图 3.3.1 SPI 从机接收

3.4 SPI 配置 DMA 发送

系统时钟选用内部主频 48MHZ，通过 DMA 发送数据。

- **管脚配置：**

SCLK--PA0.8 / MOSI--PA0.6 / MISO--PA0.9 / NSS--PA0.7

```
void spi_etcb_dma_send(void)
{
    uint8_t bySdData[100]= {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

    csi_dma_ch_config_t tDmaConfig;
    csi_etb_config_t        tEtbConfig;
    csi_spi_config_t        t_SpiConfig;

    for(uint8_t i=0; i<100; i++)
    {
            bySdData[i] = i;
    }

    //dma para config
    tDmaConfig.bySrcLinc        = DMA_ADDR_CONSTANT;
    tDmaConfig.bySrcHinc        = DMA_ADDR_INC;
    //tDmaConfig.bySrcLinc      = DMA_ADDR_INC;
    //tDmaConfig.bySrcHinc      = DMA_ADDR_CONSTANT;
    tDmaConfig.byDetLinc        = DMA_ADDR_CONSTANT;
    tDmaConfig.byDetHinc        = DMA_ADDR_CONSTANT;
    tDmaConfig.byDataWidth      = DMA_DSIZE_8_BITS;
    tDmaConfig.byReload         = DMA_RELOAD_DISABLE;
    tDmaConfig.byTransMode      = DMA_TRANS_ONCE;
    tDmaConfig.byTsizeMode    = DMA_TSIZE_ONE_DSIZE;
    tDmaConfig.byReqMode        = DMA_REQ_HARDWARE;
    tDmaConfig.wInt                     = DMA_INTSRC_NONE;

    //etcb para config
    tEtbConfig.byChType = ETB_ONE_TRG_ONE_DMA;
    tEtbConfig.bySrcIp     = ETB_SPI0_TXSRC;
    tEtbConfig.byDstIp     = ETB_DMA_CH0;
    tEtbConfig.byTrgMode = ETB_HARDWARE_TRG;

    //端口配置
    csi_pin_set_mux(PA07, PA07_OUTPUT);
    csi_pin_output_mode(PA07, GPIO_PUSH_PULL);
    csi_spi_nss_high(PA07);
    csi_pin_set_mux(PA08,PA08_SPI0_SCK);
    csi_pin_set_mux(PA09,PA09_SPI0_MISO);
    csi_pin_set_mux(PA06,PA06_SPI0_MOSI);
```

```
    //spi para config
    t_SpiConfig.bySpiMode = SPI_MASTER;
    t_SpiConfig.bySpiPolarityPhase = SPI_FORMAT_CPOL0_CPHA1;
    t_SpiConfig.bySpiFrameLen = SPI_FRAME_LEN_8;
    t_SpiConfig.wSpiBaud = 12000000;
    t_SpiConfig.byInt= SPI_INTSRC_NONE;

    csi_etb_init();
    csi_etb_ch_config(ETB_CH8, &tEtbConfig);

    csi_dma_soft_rst(DMA);
    csi_dma_ch_init(DMA, 0, &tDmaConfig);

    csi_spi_init(SPI0,&t_SpiConfig);

  void main()
  {
      system_init();
    board_init();
    while(1)
    {
    csi_spi_nss_low(PA07);
    csi_spi_send_dma(SPI0,(void *)bySdData,100, 0);
    csp_dma_t *ptDmaChBase = (csp_dma_t *)DMA_REG_BASE(DMA, 0);
    while(csp_dma_get_curr_htc(ptDmaChBase));       //等待直到 dma 传输完成
    while(csp_spi_get_sr(SPI0) & SPI_BSY);          //wait for transmition finish
    csi_spi_nss_high(PA07);
    }
  }
```

- **代码说明：**

  csi_etb_init(); ----- 用于使能 ETCB 功能

  csi_etb_ch_config(); ----- 用于初始化 ETCB

  csi_dma_soft_rst(); ----- 用于软件复位 DMA 模块

  csi_spi_send_dma(); ----- 用于使能 DMA 通道

- **函数参数说明：**

  csi_etb_ch_config(ETB_CH8, &tEtbConfig);对 ETCB 初始化参数的结构体进行配置

tEtbConfig.byChType ---- 选择 ETCB 触发方式

tEtbConfig.bySrcIp ---- 选择触发源

tEtbConfig.byDstIp ---- 目标

tEtbConfig.byTrgMode ---- 通道触发模式

csi_dma_ch_init(DMA, 0, &tDmaConfig); 对 DMA 初始化参数的结构体进行配置

tDmaConfig.bySrcLinc ----低位传输原地址

tDmaConfig.bySrcHinc ----高位传输原地址

tDmaConfig.byDetLinc ----低位传输目标地址

tDmaConfig.byDetHinc ----高位传输目标地址

tDmaConfig.byDataWidth ----传输数据宽度 8bit 16bit 32bit

tDmaConfig.byReload ----禁止/开启 自动重载

tDmaConfig.byTransMode ----DMA 服务模式(传输模式)

tDmaConfig.byTsizeMode ----传输数据大小，一个 DSIZE，即 DSIZE 定义大小

tDmaConfig.byReqMode ----DMA 请求模式

tDmaConfig.wInt ----中断开/关

- **发送数据：**

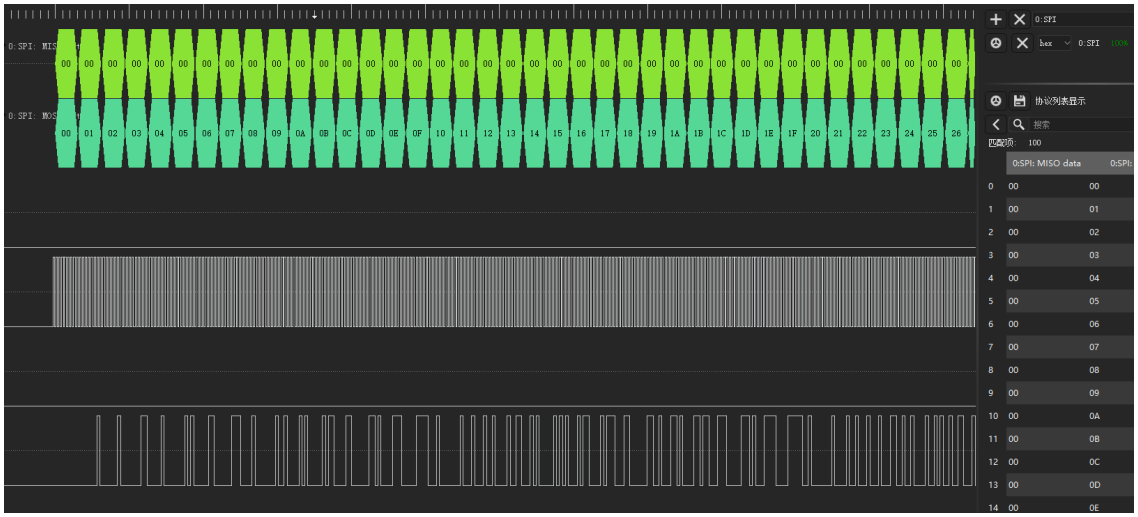

图 3.4.1 SPI DMA

4. 程序下载和运行

1．将目标板与仿真器连接，分别为 VDD、SCLK、SWIO、GND

2. 将 SPI 主机功能脚与对应的 SPI 从机设备功能脚进行连接

3. 程序编译后仿真运行

4. 进行读写数据，查看图 3.2.1 图 3.3.1 图 3.4.1 进行验证。