Application Note

SPC11X8/SPD11X8 SIO_SPI 使用指南

Revision 1 – July 2019

目录

表格列表

图片列表

1 SIO_SPI 单元

1.1 概述

SPI（Serial Peripheral Interface）接口是一种同步，双工的串行数据传输接口。它可以用于连接多种外设，例如 flash，ADC 等等。

用 SIO 实现的 SPI（SIO_SPI）支持摩托罗拉 SPI 协议定义的四种传输模式。传输波特率以及时序都可根据应用需求配置。

1.2 SIO_SPI 单元特性

- 低位有效的选择线（SFRM）；
- 时钟线相位极性定义的 4 种传输模式；
- 最高可达 12.5mps 的波特率；
- Master 或 Slave 模式；
- 双工模式或 Master 只发模式，Slave 只收模式；
- 8-bit 或 16-bit 帧长度；
- MSB-first 或 LSB-first；
- 独立的 Tx FIFO 和 Rx FIFO，FIFO 深度都为 6，宽度都为 16-bit；
- Rx FIFO 溢出错误标志位，会产生不可在 IP 级屏蔽的中断；
- 单次传输结束标志位，如果使能则产生中断；
- 多次传输结束标志位，如果使能则产生中断。

1.3 SIO_SPI 单元信号

表 1-1 描述了 SIO_SPI 信号列表。

表 1-1: SIO_SPI 信号列表

信号名	类型（Master/Slave）	Description
MOSI	输出/输入	Master 数据输出口/Slave 数据输入口
MISO	输入/输出	Master 数据输入口/Slave 数据输出口
SCLK	输出/输入	SPI 时钟口，Master 输出口/Slave 输入口
SFRM	输出/输入	Master 给 Slave 的选择信号

1.4 SIO_SPI 版本

表 1-2 罗列了 SIO_SPI 的所有版本以及目前可用的版本。

表 1- 2:　SIO_SPI 版本信息

Master/Slave	8-bit/16-bit	MSB/LSB First	可使用
Master	8-bit	MSB-first	否
Master	8-bit	LSB-first	否
Master	16-bit	MSB-first	**是**
Master	16-bit	LSB-first	否
Slave	8-bit	MSB-first	否
Slave	8-bit	LSB-first	否
Slave	16-bit	MSB-first	否
Slave	16-bit	LSB-first	否

2 功能介绍

2.1 时钟与复位

SIO_SPI 工作在 SIO_CLK 下，其最高工作频率为 50MHz。

SIO_SPI 可以被复位。复位后，所有寄存器都被复位到初值，Rx FIFO 和 Tx FIFO 被清空，当前传输会被中断。

2.2 管脚配置

SIO_SPI 支持灵活的管脚选择，用户可在 SDK 图形界面上选择最符合自己需求的管脚配置。

2.3 SIO_SPI 配置模式

对 SIO_SPI 的配置（传输模式，时序，波特率）应当在配置模式下完成，当配置结束，需将 SIO_SPI 调至运行模式以便于进行数据的发送和接收。

2.4 SIO_SPI 运行模式

数据的发送和接收只在 SIO_SPI 运行模式下完成。在运行模式下，SIO_SPI 按照顺序发送 Tx FIFO 里的数据直到 Tx FIFO 为空。接收的数据储存在 Rx FIFO 中

2.5 SIO_SPI 状态及中断

2.5.1 忙状态

SIO_SPI 通过 STS.BUSY 指示其是否在忙。如果状态位为 1，表明正在传送数据，反之则说明其空闲。

该状态位由硬件根据当前状态设置或清除，用户只可对其查询而不得修改。修改该状态位可导致错误。

2.5.2 单次传输结束标志位

SIO_SPI 通过 STS.OSTDONE 指示单次传输是否已完成。如果该状态位为 1，表明一次传输已结束，反之，没有一次传输结束。

该标志位由硬件设置，并只能由用户清除。用户应当写入 0 清除该标志位，用户对该位写 1 可导致错误。

如果使能（CTL1.OSTIE），该标志位产生 SIOB 中断

2.5.3 多次传输结束标志位

SIO_SPI 通过 STS.MLTDONE 指示多次传输（由 CTL1.MLTCNT 决定）是否已完成。如果该状态位为 1，表明设定的多次传输已结束，反之则没有结束。

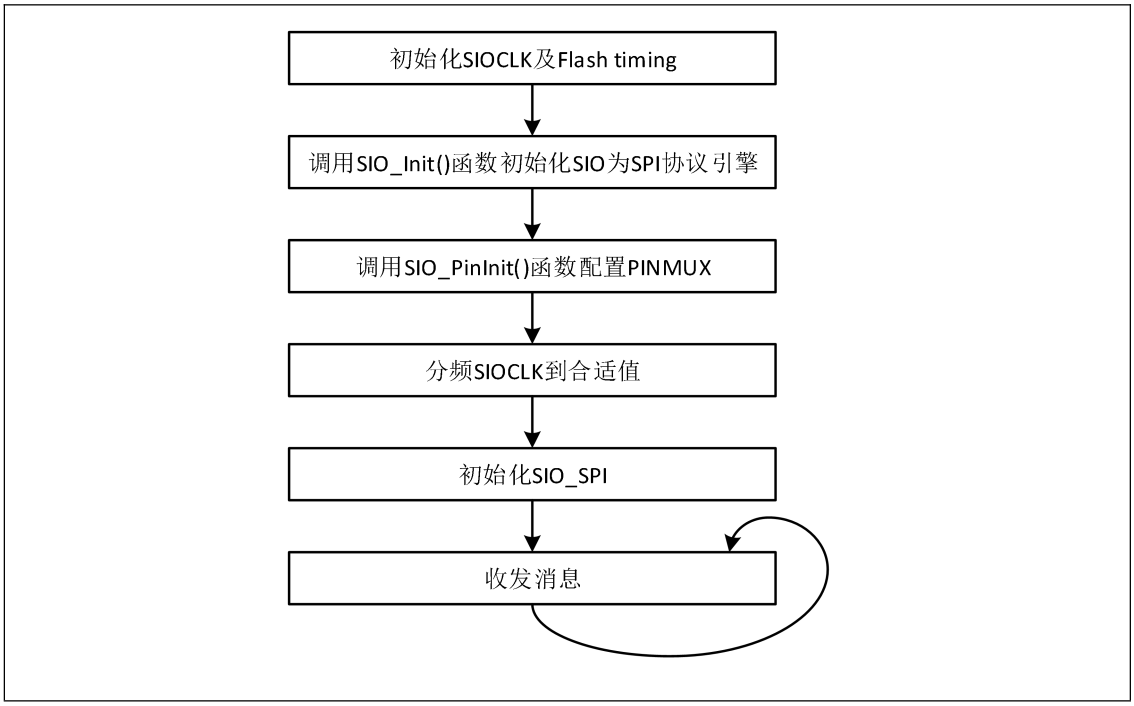
该标志位由硬件设置，并只能由用户清除。用户应当写入 0 清除该标志位，用户对该位写 1 可导致错误。

如果使能（CTL1.MLTIE），该标志位产生 SIOB 中断

3 操作方式

图 3-1 给出了基于 SIO 的 SPI 在使用时的具体操作流程。Spintrol 提供了相应的软件库来简化该系统的使用。

图 3-1: 基于 SIO 的 SPI 操作流程



3.1 初始化 SIOCLK 及 Flash timing

用户可以通过 CLOCK 以及 SIOCLKCTL 寄存器来配置 SIO 时钟，包括时钟源，使能和分频比。具体可参见《SPC11X8/SPD11X8 Technical Reference Manual》的第 3 章。初始化时建议使用 200Mhz 以快速配置 SIO 为 SPI。

3.2 初始化 SIO 为 SPI 协议引擎

SPC11X8/SPD11X8 SDK 中提供了 SIO_Init()函数，下载固件到 SIO 模块，将其初始化成 SPI 擎。用户只需要在代码中直接调用该函数即可。

3.3 配置 PINMUX

用户需要根据所用的管脚，配置相应的 PINMUX，将管脚切换至 SIO 的输入输出通道。具体可参见《SPC11X8/SPD11X8 Technical Reference Manual》的第 4.3 章节和第 4.4 章节。SPC11X8/SPD11X8 SDK 中提供了 SIO_PinInit()函数便于快速完成 PINMUX 的配置，用户只需要在代码中直接调用该函数即可。

4 函数介绍

Spintrol 提供了基础的 SIO_SPI 配置函数，可协助用户快速上手使用。本章节着重介绍这些函数的使用方式及需要特别注意的地方。主要的 API 函数如表 4-1 所示。

表 4-1: SIO_SPI 单元驱动函数

API 函数	说明
SIO_Init(void)	初始化 SIO 为 SIO_SPI，详见 4.1
SIO_PinInit(void)	初始化 SIO 管脚，详见 4.1
SIO_SPI_ReceiveData()	返回接收数据
SIO_SPI_SendData(u16Data)	发送数据 u16Data
SIO_SPI_Run()	切换到运行模式
SIO_SPI_Stop()	停止运行模式（配置模式）
SIO_SPI_SetMultipleTransferCnt(eDataNum)	设置多次传输数目为 eDataNum
SIO_SPI_GetMultipleTransferCnt()	返回多次传输数目
SIO_SPI_EnableTxOnly()	使能只发模式
SIO_SPI_DisableTxOnly()	禁用只发模式
SIO_SPI_SetCPOL(u1Val)	设置 CPOL 为 u1Val
SIO_SPI_GetCPOL()	获取当前 CPOL
SIO_SPI_SetCPHA(u1Val)	设置 CPHA 为 u1Val
SIO_SPI_GetCPHA()	获取当前 CPHA
SIO_SPI_SetLeadingTime(u15Time)	设置 Leading time 为 u15Time
SIO_SPI_GetLeadingTime()	获取当前 Leading time
SIO_SPI_SetTrailingTime(u15Time)	设置 Trailing time 为 u15Time
SIO_SPI_GetTrailingTime()	获取当前 Trailing time
SIO_SPI_SetIdlingTime(u15Time)	设置 Idling time 为 u15Time
SIO_SPI_GetIdlingTime()	获取当前 Idling time
SIO_SPI_SetHalfBitDuration(u15Time)	设置半比特宽度为 u15Time
SIO_SPI_GetHalfBitDuration()	获取当前半比特宽度
SIO_SPI_EnableMultipleTransferDoneInt()	使能多次传输结束中断
SIO_SPI_DisableMultipleTransferDoneInt()	禁用多次传输结束中断
SIO_SPI_EnableSingleTransferDoneInt()	使能单次传输结束中断
SIO_SPI_DisableSingleTransferDoneInt()	禁用单次传输结束中断
SIO_SPI_ClearMultipleTransferDoneInt()	清除多次传输结束中断位
SIO_SPI_ClearSingleTransferDoneInt()	清除单次传输结束中断位
SIO_SPI_GetSingleTransferDoneIntStatus()	获取单次传输结束标志位
SIO_SPI_GetMultipleTransferDoneIntStatus()	获取多次传输结束标志位
SIO_SPI_GetBusyStatus()	获取忙标志位
SIO_SPI_GetRxOverflowIntStatus()	获取接收 FIFO 是否溢出状态
SIO_SPI_GetRxEmptyStatus()	获取接收 FIFO 是否空状态
SIO_SPI_GetRxFullStatus()	获取接收 FIFO 是否满状态
SIO_SPI_GetTxEmptyStatus()	获取发送 FIFO 是否空状态

SIO_SPI_GetTxFullStatus()	获取发送 FIFO 是否满状态
SIO_SPI_Reset(void)	复位 SIO_SPI
SIO_SPI_Init(uint16_t u16IdlingTime, uint16_t u16LeadingTime, uint16_t u16TrailingTime, uint32_t u32Baudrate, uint8_t u1CPOL, uint8_t u1CPHA, uint8_t u1TxOnly, uint8_t u3MltCnt)	初始化 SIO_SPI，详见 4.2
SIO_SPI_InitEasy(uint32_t u32Baudrate)	简易初始化 SIO_SPI，详见 4.3
SIO_SPI_Send(uint16_t *pu16Buf, uint32_t u32Count)	发送一组数据函数 1） pu16Buffer 发送数组指针 2） u32Count 发送数据数量
SIO_SPI_Receive(uint16_t *pu16Buf, uint32_t u32Count)	查询 RxFIFO 空状态来接收一组数据函数，该函数返回数据量 1）pu8Buffer 接收数组指针 2）u32Count 接收数据数量
SIO_SPI_RecvWithoutCheck(uint16_t *pu16Buf, uint32_t u32Count)	接收指定数目数据，不查询 RxFIFO 空状态

4.1 初始化 SIO 为 SIO_SPI 及管脚

用户可以通过 SIO_Init()函数配置 SIO 为 SIO_SPI。建议按照如下步骤配置：

（1） 务必在 main 函数一开始先调用 FLASH_WALLOW()以及 FLASH_SetTiming()，再调用 CLOCK_InitWithRCO()初始化时钟；

（2） 调用 SIO_Init ()配置 SIO 为 SIO_SPI；

（3） 调用 SIO_PinInit()切换到选定的 gpio 管脚

（4） 如果需要分频 SIO_CLK，切记 SIO_CLK 不得超过 50MHz。

示例代码 4-1：

```
void main()
{
/*** Step 1: Flash timing setting ***/
FLASH_WALLOW();
FLASH_SetTiming(200000000);
/*** Step 2: Clock initial ***/
CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
/*** Step 3: SIO initial ***/
SIO_Init();
/*** Step 4: SIO pin initial ***/
SIO_PinInit();
/*** Step 5: divide SIO_CLK ***/
CLOCK->SIOCLKCTL.bit.DIV = 3;

}
```

4.2 初始化 SIO_SPI

Spintrol 提供打包好的 SIO_SPI 配置函数完成所有配置功能。该函数执行以下步骤：

（1） 切换 SIO_SPI 到配置模式

（2） 根据当前 SIO_CLK 频率及设定的波特率计算分频并配置。需要注意的是，寄存器设置值 = SIO_CLK 频率/波特率/2 – 1。计算值会被取整，因而波特率值为离散的数值，用户应当根据需要的波特率配置合适的 SIO_CLK，否则波特率将存在误差。

（3） 配置时序寄存器

（4） 配置 CPHA 和 CPOL，多次传输数目以及是否双工模式

（5） 取消中断使能

（6） 切换 SIO_SPI 到运行模式。

表 4- 2: SIO_SPI_Init()函数介绍

SIO_SPI_Init(uint16_t u16IdlingTime, uint16_t u16LeadingTime, uint16_t u16TrailingTime, uint32_t u32Baudrate, uint8_t u1CPOL, uint8_t u1CPHA, uint8_t u1TxOnly, uint8_t u3MltCnt)	
Parameter	**Description**
u16IdlingTime	最小 idling time，以半个 SCLK 周期为单位，禁止设置为 0
u16LeadingTime	最小 leading time，以半个 SCLK 周期为单位
u16TrailingTime	最小 Trailing time，以半个 SCLK 周期为单位
u32BaudRate	波特率设置，以赫兹为单位，不得高于 12.5MHz
u1CPOL	时钟 idle 状态极性，设 0 为低电平，设 1 为高电平
u1CPHA	时钟采样相位设定，设 0 第一个时钟沿采集，设 1 第二个时钟沿采集
u1TxOnly	只发不收，设 0 为双工模式，设 1 为只发不收模式
u3MltCnt	多次传输数目设定，可设为 1-6

示例代码

```
/*所有 timing 都设定为半个 SCLK 周期，12.5MHz 波特率传输，CPOL=0，CPHA=0，双工模式，多次传输设为 4 */
SIO_SPI_Init(1, 1, 1, 12500000, 1, 0, 0, 4);
```

用户应当尤其注意波特率不可任意选择，SIO_SPI 仅支持离散的波特率值。用户还当注意在发送数据前自行打开需要的中断使能。

4.3 简易初始化 SIO_SPI

Spintrol 提供最常用的 SIO_SPI 配置函数，用户只需指定波特率。调用该函数，CPOL = 0，CPHA = 0，所有时序设置为半个 SCLK，双工模式，且多次传输设为 1。该函数调用上节函数配置，上节中对波特率的限制亦适用本节。

表 4- 3: SIO_SPI_InitEasy()函数介绍

SIO_SPI_InitEasy(uint32_t u32Baudrate)	
Parameter	**Description**
u32BaudRate	波特率设置，以赫兹为单位，不得高于 12.5MHz

示例代码

```
/*12.5MHz 波特率传输，CPOL=0，CPHA=0，双工模式，多次传输设为 1 */
SIO_SPI_InitEasy(12500000);
```

用户应当尤其注意波特率不可任意选择，SIO_SPI 仅支持离散的波特率值。用户还当注意在发送数据前自行打开需要的中断使能。

5 SIO_SPI 实例

5.1 主模式收发实例

这个例子演示如何使用 SIO_SPI 收发数据。该例子使用 SIO_SPI Master 16bit MSB 优先版本。其他配置情况为：

- 波特率 12.5Mhz
- CPOL=1， CPHA=1

该例子配合一个 SPI 从模块进行验证，用户可通过打印信息查看信息传输情况，也可严格按照例子中发送和接收的数据来配置从模块发送的数据，实现数据自动检查及报错。

示例代码 5-1（以 SPC1168 为例）：

```
#include <stdio.h>
#include "spc1168.h"
#include "sio_spi0.h"

#define SPI_CPOL 1
#define SPI_CPHA 1
#define LOOP_TIME 10

uint16_t au16TxData[16];
uint16_t au16RxData[16];

int main(void)
{
  uint32_t i,j ;
  uint16_t u16RxData;
  uint16_t u16TempData;

  uint32_t ip_cpol, ip_cpha;

  /* Set flash timing for 200Mhz */
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);

  /* Init clock */
  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);


  /* Enable UART clock */
  CLOCK->UARTCLKCTL.bit.EN = 1;
```

```
/* Configure GPIO pin as UART function */
PINMUX->GPIO34.bit.MUXSEL = 1;
PINMUX->GPIO35.bit.MUXSEL = 1;

/* UART Init */
UART_Init(UART, 38400);

/*Init SIO as SIO_SPI*/
SIO_Init();

/* Set SIO Clock DIV*/
CLOCK->SIOCLKCTL.bit.DIV = 3;

/* Init SIO_SPI */
SIO_SPI0_Init(1, 1, 1, 12500000, SPI_CPOL, SPI_CPHA, 0, 4);

/* Configure selected GPIO pin as SIO function */
SIO_PinInit();

/* Print current configuration for check */
printf("SystemCoreClock = %d MHz \n", SystemCoreClock / 1000000);
printf("HCLK DIV = %d, SIOCLK DIV = %d \n", CLOCK->HCLKCTL.bit.DIV,
        CLOCK->SIOCLKCTL.bit.DIV);
printf("SIO_SPI Half bit duration = %d \n",
        SIO_SPI0_GetHalfBitDuration());

/* Show CPOL and CPHA */
u16TempData = SIO_SPI0->CTL1;
ip_cpol = (u16TempData & SIO_SPI0_CTL1_ALL_CPOL_Msk) >>
           SIO_SPI0_CTL1_ALL_CPOL_Pos;
ip_cpha = (u16TempData & SIO_SPI0_CTL1_ALL_CPHA_Msk) >>
           SIO_SPI0_CTL1_ALL_CPHA_Pos;
printf("CPOL = %1d, CPHA = %1d \n\n", ip_cpol, ip_cpha);

printf("SPC1168 SPI_SIO Master 16b TX/RX test begins...\n\n\n");

/* data value base */
u16TempData = 0xA5F0;

/* Prepare vip_tx data for check */
for (i = 0; i < 16; i++)
{
  u16TempData += 0x1234;
```

```
    au16RxData[i] = u16TempData;
  }

  /* Prepare tx data for sending */
  for (i = 0; i < 16; i++)
  {
     u16TempData -= 0x1234;
     au16TxData[i] = u16TempData;
  }


  /* Start test: send data and check with received data for LOOP_TIME
times */
  for (i= 0 ; i < LOOP_TIME; i++)
  {
    printf("Loop %d :\n\n", i);

    for (j = 0; j < 16; j++)
    {
      /* Send one 16-bit data */
      SIO_SPI0_SendData(au16TxData[j]);

      /* Wait until SIO_SPI complete a single transfer */
      while(SIO_SPI0_GetSingleTransferDoneIntStatus() == 0);

      /* Clear single transfer done flag*/
      SIO_SPI0_ClearSingleTransferDoneInt();

      /* Get received data from Rx FIFO */
      u16RxData = SIO_SPI0_ReceiveData();

      /* Check with vip_tx sent data */
      if (u16RxData != au16RxData[j])
      {
        printf("FAIL: wait for 0x%04x but received 0x%04x \n",
                au16RxData[j], u16RxData);
        return 1;
      }
      else
      {
        printf("PASS: received 0x%04x as expected \n", au16RxData[j]);
      }
    }
```

```
    printf("\nTest PASS!\n");
  }


  while(1)
  {
  }
}
```

5.2 中断收发实例

这个例子演示如何使用 SIO_SPI 收发数据。该例子使用 SIO_SPI Master 16bit MSB-First 版本。其他配置情况为：

- 波特率 12.5Mhz
- CPOL=1， CPHA=1

该例子配合一个 SPI 从模块进行验证，用户可通过打印信息查看信息传输情况，也可严格按照例子中发送和接收的数据来配置从模块发送的数据，实现数据自动检查及报错。

示例代码 5-2（以 SPC1168 为例）：

```
#include <stdio.h>
#include "spc1168.h"
#include "sio_spi0.h"

#define SPI_CPOL 1
#define SPI_CPHA 1
#define LOOP_TIME 10
#define MLTCNT 6

uint16_t au16TxData[16];
uint16_t au16RxData[16];
uint16_t u16LoopTime = 0;

/* PIN SELECTION of Committed File*/
/* SCLK: 23 */
/* SFRM: 30 */
/* MISO: 22 */
/* MOSI: 39 */

int main(void)
{
  uint32_t i,j ;
  uint16_t u16TempData;

  uint32_t ip_cpol, ip_cpha;


  /* Set flash timing for 200Mhz */
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);

  /* Init clock */
  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
```

```
/* Enable UART clock */
CLOCK->UARTCLKCTL.bit.EN = 1;

/* Configure GPIO pin as UART function */
PINMUX->GPIO34.bit.MUXSEL = 1;
PINMUX->GPIO35.bit.MUXSEL = 1;

/* UART Init */
UART_Init(UART, 38400);

/*Init SIO as SIO_SPI*/
SIO_Init();

/* Set SIO Clock DIV*/
CLOCK->SIOCLKCTL.bit.DIV = 3;

/* Init SIO_SPI */
SIO_SPI0_Init(1, 1, 1, 12500000, SPI_CPOL, SPI_CPHA, 0, MLTCNT);

/* Configure selected GPIO pin as SIO function */
SIO_PinInit();

/* Enable multiple transfer done int */
SIO_SPI0_EnableMultipleTransferDoneInt();

/* Enable SSP interrupt in CPU side */
NVIC_EnableIRQ(SIO0A_IRQn);
NVIC_EnableIRQ(SIO0B_IRQn);

/* Print current configuration for check */
printf("SystemCoreClock = %d MHz \n", SystemCoreClock / 1000000);
printf("HCLK DIV = %d, SIOCLK DIV = %d \n", CLOCK->HCLKCTL.bit.DIV,
        CLOCK->SIOCLKCTL.bit.DIV);
printf("SIO_SPI Half bit duration = %d \n",
        SIO_SPI0_GetHalfBitDuration());

/* Show CPOL and CPHA */
u16TempData = SIO_SPI0->CTL1;
ip_cpol = (u16TempData & SIO_SPI0_CTL1_ALL_CPOL_Msk) >>
           SIO_SPI0_CTL1_ALL_CPOL_Pos;
ip_cpha = (u16TempData & SIO_SPI0_CTL1_ALL_CPHA_Msk) >>
           SIO_SPI0_CTL1_ALL_CPHA_Pos;
```

```
  printf("CPOL = %1d, CPHA = %1d \n\n", ip_cpol, ip_cpha);
  printf("CTL1 = 0x%04X\n", u16TempData);

  printf("SPC1168 SPI_SIO Master 16b TX/RX test begins...\n\n\n");

  /* data value base */
  u16TempData = 0xA5F0;

  /* Prepare vip_tx data for check */
  for (i = 0; i < 16; i++)
  {
    u16TempData += 0x1234;
    au16RxData[i] = u16TempData;
  }

  /* Prepare tx data for sending */
  for (i = 0; i < 16; i++)
  {
      u16TempData -= 0x1234;
      au16TxData[i] = u16TempData;
  }

  /* Start test: send multiple data */
  printf("\nLoop %d :\n\n", u16LoopTime);

  for (j = 0; j < MLTCNT; j++)
  {
    /* Send one 16-bit data */
    //printf("Fill data =  0x%04X\n", au16TxData[j]);
    SIO_SPI0_SendData(au16TxData[j]);
  }

  while(1)
  {
  }
}

void SIO0A_IRQHandler()
{
  /* Get Rx overflow flag */
  if (SIO_SPI0_GetRxOverflowIntStatus() == 1)
  {
    printf("FAIL : Rx FIFO overflow ! \n");
    while(1);
```

```
  }
}

void SIO0B_IRQHandler()
{
    uint32_t j;
    uint32_t u16RxData;

    /* Clear multiple transfer int */
    SIO_SPI0_ClearMultipleTransferDoneInt();

    /* Check received data */
    for (j = 0; j < MLTCNT; j++)
    {
    /* Get received data from Rx FIFO */
    u16RxData = SIO_SPI0_ReceiveData();

    /* Check with expected data */
    if (u16RxData != au16RxData[j])
    {
      printf("FAIL: wait for 0x%04x but received 0x%04x \n",
               au16RxData[j], u16RxData);
      while(1);
    }
    else
    {
      printf("PASS: received 0x%04x as expected \n", au16RxData[j]);
    }
    }

  u16LoopTime++;
  if (u16LoopTime >= LOOP_TIME)
  {
    printf("\nTest PASS!\n");
    while(1);
  }
  else
  {
    printf("\nLoop %d :\n", u16LoopTime);
  }

  /* Send multiple data */
  for (j = 0; j < MLTCNT; j++)
  {
```

```
    /* Send one 16-bit data */
    SIO_SPI0_SendData(au16TxData[j]);
  }
}
```

6 寄存器

6.1 寄存器地址映射

表 6- 1: SIO_SPI Module Base Address

Peripheral Module	Base Address
SIO_SPI	0x4000 B000

表 6- 2: SIO_SPI Register Map

Register	Offset	Description	Reset Value
FIFODAT	0x00	TX/RX FIFO Data Register	0x00000000
CTL0	0x08	SPI Control Register 0	0x00000000
CTL1	0x0C	SPI Control Register 1	0x00000000
LEADTIM	0x10	SPI Leading Time Register	0x00000000
TRAILTIM	0x14	SPI Trailing Time Register	0x00000000
IDLETIM	0x18	SPI Idling Time Register	0x00000000
HALFBD	0x1C	SPI Half Bit Duration Register	0x00000000
STS	0x20	SPI Status Register	0x00000000
FIFOSTS	0x7C	FIFO Status Register	0x00000044

6.2 SIO_SPI 寄存器

表 6- 3 through 表 6- 20 provide the SIO_SPI module related register details.

表 6- 3: SIO_SPI FIFO Data Register (FIFODAT) Layout

FIFODAT (TX/RX FIFO Data Register) Offset: 0x0 Default: 0x00000000 Access: SIO_SPI -> FIFODAT.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
DATA							
7	6	5	4	3	2	1	0
DATA							

表 6- 4: SIO_SPI FIFO Data Register (FIFODAT) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED_31_16	RO	0x0	Reserved
15:0	DATA	RW	0x0	SPI FIFO data entry Write this register puts data to TX FIFO Read this register gets data from RX FIFO

表 6- 5: SIO_SPI Control Register 0 (CTL0) Layout

CTL0 (SPI Control Register 0) Offset: 0x8 Default: 0x00000000 Access: SIO_SPI -> CTL0.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED							
7	6	5	4	3	2	1	0
RESERVED							RUN

表 6- 6: SIO_SPI Control Register 0 (CTL0) Field Description

Bits	Field Name	Type	Reset	Description
31:1	RESERVED_31_1	RO	0x0	Reserved
0	RUN	RW	0x0	SPI normal run enable 0: Configure SPI (i.e. configure CTL1) 1: SPI is running (i.e. data transfer) Before setting this bit to 1, CTL1, LEADTIM, IDLETIM, TRAILTIM, HALFBD shall be configured

表 6- 7: SIO_SPI Control Register 1 (CTL1) Layout

CTL1 (SPI Control Register 1) Offset: 0xC Default: 0x00000000 Access: SIO_SPI -> CTL1.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED						OSTIE	MLTIE
7	6	5	4	3	2	1	0
RESERVED	MLTCNT			RESERVED	TXONLY	CPOL	CPHA

表 6- 8: SIO_SPI Control Register 1 (CTL1) Field Description

Bits	Field Name	Type	Reset	Description
31:10	RESERVED_31_10	RO	0x0	Reserved
9	OSTIE	RW	0x0	One-shot transfer interrupt enable Generate interrupt upon each transfer 0: Disable 1: Enable
8	MLTIE	RW	0x0	Multi-data transfer interrupt enable Generate interrupt upon transfer every data 0: Disable 1: Enable
7	RESERVED_7	RO	0x0	Reserved

6:4	MLTCNT	RW	0x0	Data counts that regards as a multi-data transfer 1: Regards every 1 data transfer as a complete multi-data transfer 2: Regards every 2 data transfer as a complete multi-data transfer 3: Regards every 3 data transfer as a complete multi-data transfer 4: Regards every 4 data transfer as a complete multi-data transfer 5: Regards every 5 data transfer as a complete multi-data transfer 6: Regards every 6 data transfer as a complete multi-data transfer
3	RESERVED_3	RO	0x0	Reserved
2	TXONLY	RW	0x0	TX only 0: Full TX/RX mode 1: Disable RX and only allow TX
1	CPOL	RW	0x0	Clock polarity setting 0: The inactive or idle state SPI clock is low 1: The inactive or idle state SPI clock is high
0	CPHA	RW	0x0	Clock phase setting 0: Start sample data on the first SPI clock edge after the start of a frame. 1: Start sample data on the second SPI clock edge after the start of a frame.

表 6- 9: SIO_SPI Leading Time Register (LEADTIM) Layout

LEADTIM (SPI Leading Time Register) Offset: 0x10 Default: 0x00000000 Access: SIO_SPI -> LEADTIM.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED	VAL						
7	6	5	4	3	2	1	0
VAL							

表 6- 10: SIO_SPI Leading Time Register (LEADTIM) Field Description

Bits	Field Name	Type	Reset	Description
31:15	RESERVED_31_15	RO	0x0	Reserved
14:0	VAL	RW	0x0	Set leading time to LEADTIM SIO clocks (Note: Leading time is 2^15 SIO clocks if LEADTIM=0)

表 6- 11: SIO_SPI Trailing Time Register (TRAILTIM) Layout

TRAILTIM (SPI Trailing Time Register) Offset: 0x14 Default: 0x00000000 Access: SIO_SPI -> TRAILTIM.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED	VAL						
7	6	5	4	3	2	1	0
VAL							

表 6- 12: SIO_SPI Trailing Time Register (TRAILTIM) Field Description

Bits	Field Name	Type	Reset	Description
31:15	RESERVED_31_15	RO	0x0	Reserved
14:0	VAL	RW	0x0	Set trailing time to TRAILTIM SIO clocks (Note: trailing time is 2^15 SIO clocks if TRAILTIM=0)

表 6- 13: SIO_SPI Idling Time Register (IDLETIM) Layout

IDLETIM (SPI Idling Time Register) Offset: 0x18 Default: 0x00000000 Access: SIO_SPI -> IDLETIM.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED	VAL						
7	6	5	4	3	2	1	0
VAL							

表 6- 14: SIO_SPI Idling Time Register (IDLETIM) Field Description

Bits	Field Name	Type	Reset	Description
31:15	RESERVED_31_15	RO	0x0	Reserved
14:0	VAL	RW	0x0	Set idling time to IDLETIM SIO clocks (Note: idling time is 2^15 SIO clocks if IDLETIM=0)

表 6- 15: SIO_SPI Half Bit Duration Register (HALFBD) Layout

HALFBD (SPI Half Bit Duration Register) Offset: 0x1C Default: 0x00000000 Access: SIO_SPI -> HALFBD.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED		VAL					
7	6	5	4	3	2	1	0
VAL							

表 6- 16: SIO_SPI Half Bit Duration Register (HALFBD) Field Description

Bits	Field Name	Type	Reset	Description
31:14	RESERVED_31_14	RO	0x0	Reserved
13:0	VAL	RW	0x0	Set bit duration to 2x HALFBD SIO clocks. Equivalent baudrate is SIO clock frequency scaled down by 2xHALFBD

表 6- 17: SIO_SPI Status Register (STS) Layout

STS (SPI Status Register) Offset: 0x20 Default: 0x00000000 Access: SIO_SPI -> STS.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED							
7	6	5	4	3	2	1	0
RESERVED					BUSY	OSTDONE	MLTDONE

表 6- 18: SIO_SPI Status Register (STS) Field Description

Bits	Field Name	Type	Reset	Description
31:3	RESERVED_31_3	RO	0x0	Reserved
2	BUSY	RO	0x0	SPI engine busy status 0: No data TX/RX 1: Data TX/RX is ongoing
1	OSTDONE	RW	0x0	A signal transfer finished status 0: Read a 0 indicates a complete one-shot transfer does not occur Write a 0 clears this bit 1: Read a 1 indicates a complete one-shot transfer An SIOB interrupt will be generated if CTL1.OSTIE=1 This bit should not be written as 1

0	MLTDONE	RW	0x0	All transfer finished status 0: Read a 0 indicates a complete multi-data transfer does not occur Write a 0 clears this bit 1: Read a 1 indicates a complete multi-data transfer An SIOB interrupt will be generated if CTL1.MLTIE=1 This bit should not be written as 1

表 6- 19: SIO_SPI FIFO Status Register (FIFOSTS) Layout

FIFOSTS (FIFO Status Register) Offset: 0x7C Default: 0x00000044 Access: SIO_SPI -> FIFOSTS.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15 RESERVED	14	13	12	11	10	9	8
RESERVED							
7 TXFULL	6 TXEMPTY	5 RESERVED	4	3 RXFULL	2 RXEMPTY	1 RXOVF	0 SIOEN

表 6- 20: SIO_SPI FIFO Status Register (FIFOSTS) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED_31_16	RO	0x0	Reserved
15	RESERVED_15	RO	0x0	Reserved
14:8	RESERVED_14_8	RO	0x0	Reserved
7	TXFULL	RO	0x0	TX FIFO full status 0: TX FIFO is not full 1: TX FIFO is full This bit is self-cleared when SPI engine fetch a data from TX FIFO for transmission
6	TXEMPTY	RO	0x1	TX FIFO empty status 0: TX FIFO is not empty 1: TX FIFO is empty This bit is self-cleared when write new data to TX FIFO via FIFODAT register
5:4	RESERVED_5_4	RO	0x0	Reserved
3	RXFULL	RO	0x0	RX FIFO full status 0: RX FIFO is not full 1: RX FIFO is full (i.e. There are 6 data in the RX FIFO) This bit is self-cleared when read RX FIFO via FIFODAT register

2	RXEMPTY	RO	0x1	RX FIFO empty status 0: RX FIFO is not empty 1: RX FIFO is empty This bit is self-cleared when SPI engine put a new received data to RX FIFO
1	RXOVF	RO	0x0	RX FIFO overflow status 0: RF FIFO is not overflow (i.e. Total data number is below 7) 1: RF FIFO overflow (i.e. Received more than 6 data) This bit is self-cleared when read RX FIFO via FIFODAT register An SIOA interrupt will be generated upon overflow
0	SIOEN	RW	0x0	SIO enable 0: Disable 1: Enable

7 修订记录

表 7- 1: 文档修订记录

日期	版本	修改内容
2019-07-25	1	初始版本