

概述

SPI 协议是一通信协议，即串行外围接口，是一种高速全双工的通信总线。它被广泛地使用在多种外部设备进行通信，比如 ADC、音频解码器、LCD 等设备与 MCU 间，要求通讯速率较高的场合。

注意： 本文档主要以 SPC1168 为例进行介绍。

目录

1	SPI 特性	7
2	SPI 实例	9
2.1	Poll 传输模式	10
2.1.1	主机发送与接收	10
2.1.2	从机发送与接收	13
2.2	Bulk 传输模式	17
2.2.1	主机发送与接收	17
2.2.2	从机发送与接收	20
2.3	中断传输模式	24
2.3.1	主机发送与接收	24
2.3.2	从机发送与接收	28

图片列表

图 1-1: SPI 主从设备连接示意图	7
图 1-2: FIFO 压缩和非压缩模式	8

SPIN TROL

表格列表

表 1-1: 信号线功能描述8

SPIN TROL

版本历史

版本	日期	作者	状态	变更
A/0	2023-06-24	XuQing He	Outdated	首次发布。
C/0	2024-03-26	Jiali Zhou	Released	修改排版格式。

SPIN TROL

术语或缩写

术语或缩写	描述
/	/

SPIN TROL

1 SPI 特性

SPC1168 内建两个 SPI 单元。SPC1168 的 SPI 单元有以下特点：

- 支持全双工/单工传输；
- 支持 8、16、24、32 位帧格式传输；
- 数据传输最高数量可达 50Mbps；
- 数据流顺序为最高有效位优先；
- 内建宽度 32byte 深度 16 的发送和接收 FIFO；
- 支持可配的压缩 FIFO；

SPI 通信设备之间的常用的连接方式见图 1-1。在 SPI 通信过程中，只有主机有发送 SCK 的能力。如果从机有需求向主机发送数据，只能通过主机发送空数据产生时钟用来接收从机发送的数据。

图 1-1: SPI 主从设备连接示意图

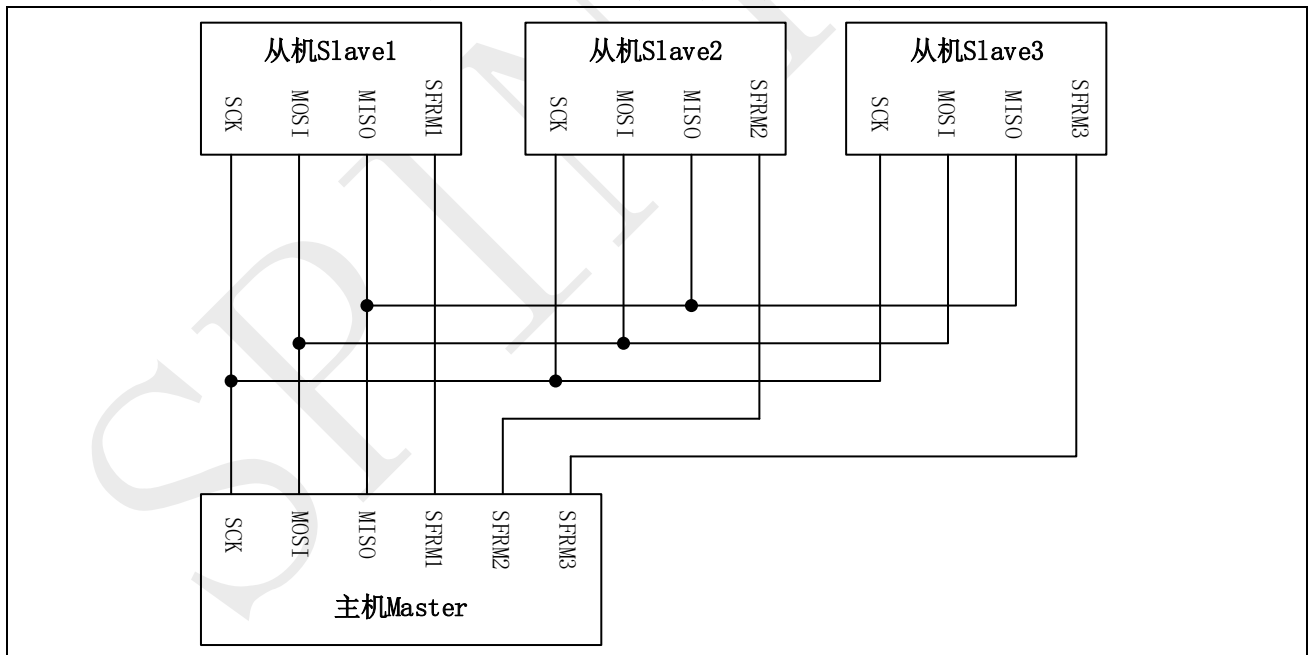
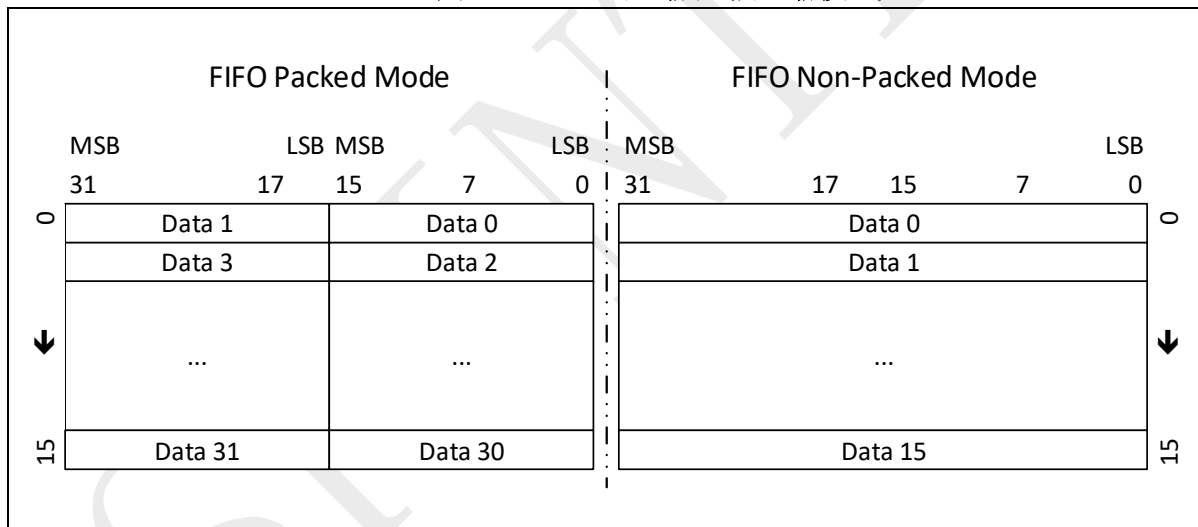


表 1-1: 信号线功能描述

MOSI	主设备输出/从设备输入引脚。主机的数据从这条信号线输出，从机由这条信号线读入主机发送的数据，即这条线上数据的方向为从主机到从机。
MISO	主设备输入/从设备输出引脚。从机的数据从这条信号线输出，主机由这条信号线读入从机发送的数据，即这条线上数据的方向为从机到主机。
SCK	时钟信号线，用于通讯数据同步。它由通讯主机产生，决定了通讯的速率。
SFRM	从设备的 SFRM 信号线设置为低电平，该从设备即被选中，即片选有效，主机开始与被选中的从设备进行通讯。SFRM 信号线置低电平为开始信号，SFRM 信号线被拉高作为结束信号。

SPC1168 支持压缩的数据 FIFO，例如传输的数据为 16bit。当使能 FIFO 工作在非压缩模式下时，Data0 数据会独占 FIFO 的一个条目；当使能 FIFO 工作在压缩模式下，Data0 和 Data1 会存储在 FIFO 的第一个条目如图 1-2，在读写数据时也是一次性访问 Data0 和 Data1。

图 1-2: FIFO 压缩和非压缩模式



2 SPI 实例

SPI 总线传输具有三种传输模式：**Bulk** 传输模式、**Poll** 传输模式、中断传输模式。**Bulk** 传输和 **Poll** 传输的区别是 **Bulk** 传输过程中会锁住 **SFRM** 信号一直占用 **I2C** 总线，其他两种模式在传输完一帧数据后会释放 **SFRM** 信号，让 **I2C** 总线空闲。

SPI 极性 (CPOL)：时钟信号空闲时的电平状态，**CPOL** 为 0 时，时钟空闲电平为低，**CPOL** 为 1 时，时钟空闲电平为高。

SPI 相位 (CPHA)：采样数据的时刻，**CPHA** 为 0 时，在第一个边沿进行采数据，**CPHA=1** 时，在第二个边沿进行采数据。

CPOL=0,CPHA=0 时，时钟空闲电平为低电平，第一个边沿是由低变高，所以上升沿采样。

CPOL=0,CPHA=1 时，时钟空闲电平为低电平，第二个边沿是由高变低，所以下降沿采样。

CPOL=1,CPHA=0 时，时钟空闲电平为高电平，第一个边沿是由高变低，所以下降沿采样。

CPOL=1,CPHA=1 时，时钟空闲电平为高电平，第二个边沿是由低变高，所以上升沿采样。

2.1 Poll 传输模式

2.1.1 主机发送与接收

本示例中演示 SPI 作为主机使用 Poll 传输模式进行发送数据和接收数据。发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 主机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_MasterTransceive()函数进行发送与接收数据，并对数据进行校对；

Poll 传输（主机发送与接收）

```
#define          DEBUG_INFO          1
/*debug information flag*/
#define          T_BUFFER_SIZE      128
/*data size of receive and transmit*/
#define          DataWidth_8bit     8
/*data width size of SSP*/
#define          SSPBaudrate_1M     1000000
/*SSP baud rate*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
 * JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
#define          MOSI_GPIO          GPIO_10
#define          MISO_GPIO          GPIO_11
#define          MOSI_GPIO_FUNC     GPIO10_SPI_MOSI
#define          MISO_GPIO_FUNC     GPIO11_SPI_MISO
#define          SCLK_GPIO          GPIO_8
#define          SFRM_GPIO          GPIO_9
#define          SCLK_GPIO_FUNC     GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC     GPIO9_SPI_SFRM
#else
#define          MOSI_GPIO          GPIO_38
#define          MISO_GPIO          GPIO_39
#define          MOSI_GPIO_FUNC     GPIO38_SPI_MOSI
#define          MISO_GPIO_FUNC     GPIO39_SPI_MISO
#define          SCLK_GPIO          GPIO_8
#define          SFRM_GPIO          GPIO_9
#define          SCLK_GPIO_FUNC     GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC     GPIO9_SPI_SFRM
#endif

uint32_t        gu32BuffSize        = T_BUFFER_SIZE;
uint32_t        gau32TxBuf[T_BUFFER_SIZE];
uint32_t        gau32RxBuf[T_BUFFER_SIZE];
ErrorStatus     estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;
```

```

/*To check the datum sent and received are the same*/
for(i = 0; i < gu32BuffSize; i++)
{
    if (gau32RxBuf[i] != u8Data)
    {
        #if DEBUG_INFO
        printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau32RxBuf[i] );
        #endif

        return ERROR;
    }
    else
    {
        #if DEBUG_INFO
        printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
        #endif
    }

    u8Data++;
}

return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int)(gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Master TxRx data...\n");

    SSP_MasterTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)
    {
        printf("Success\n");
    }
}

/*****
*****
*
* @brief      In this case, Master sent data to slave and then read these datum
back from slave.
*
*            This demo code couple with another named 'SSP_Slave_TxRx_Polling'.
*
*****
*****/

int main()
{

```

```
FLASH_WALLOW();

#if defined(SPC1158)
FLASH_SetTiming(100000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
FLASH_SetTiming(200000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

Delay_Init();

/*
 * Init the UART
 *
 * 1.Set the GPIO34/35 as UART FUNC
 *
 * 2.Enable the UART CLK
 *
 * 3.Set the rest para
 */
GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
CLOCK_EnableModule(UART_MODULE);
UART_Init(UART, 38400);

/* Set GPIOs as SSP function */
GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);
GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

/* Set Output Strength(for high speed) */
GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

/*
 * Init SSP Baud Rate, Frame Format
 * 1) Set master mode
 * 2) Set frame data 8-bits width
 * 3) Set transfer speed 30Mbps
 * 4) Set pin SFRM active low
 * 5) Set SCLK Polarity: SCLK is low when bus idle
 * 6) Set SCLK Phase: sample data on SCLK rising edge
 */
SSP_Init(SSP, SSP_MASTER_MODE, DataWidth_8bit, SSPBaudrate_1M,
SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

/* Disable FIFO Packed Mode */
SSP_DisableFIFOPackMode(SSP);

/* Enable SSP */
SSP_Enable(SSP);
```

```
/* SSP Master Tx/Rx datum */
SSP_TxRx();

while (1)
{
}
}
```

2.1.2 从机发送与接收

本示例中演示 SPI 作为从机使用 Poll 传输模式进行发送数据和接收数据，发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 从机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_SlaveTransceive ()函数进行发送与接收数据，并对数据进行校对；

Poll 模式（从机发送与接收）

```
#define          DEBUG_INFO          1
/*debug information flag*/
#define          T_BUFFER_SIZE       128
/*data size of receive and transmit*/
#define          DataWidth_8bit      8
/*data width size of SSP*/
#define          SSPBaudrate_1M      1000000
/*SSP baud rate*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
 * JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
#define          MOSI_GPIO            GPIO_10
#define          MISO_GPIO            GPIO_11
#define          MOSI_GPIO_FUNC      GPIO10_SPI_MOSI
#define          MISO_GPIO_FUNC      GPIO11_SPI_MISO
#define          SCLK_GPIO            GPIO_8
#define          SFRM_GPIO            GPIO_9
#define          SCLK_GPIO_FUNC      GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC      GPIO9_SPI_SFRM
#else
#define          MOSI_GPIO            GPIO_38
#define          MISO_GPIO            GPIO_39
#define          MOSI_GPIO_FUNC      GPIO38_SPI_MOSI
#define          MISO_GPIO_FUNC      GPIO39_SPI_MISO
#define          SCLK_GPIO            GPIO_8
#define          SFRM_GPIO            GPIO_9
```

```

#define          SCLK_GPIO_FUNC          GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC          GPIO9_SPI_SFRM
#endif

uint32_t        gu32BuffSize            = T_BUFFER_SIZE;
uint32_t        gau32TxBuf[T_BUFFER_SIZE];
uint32_t        gau32RxBuf[T_BUFFER_SIZE];
ErrorStatus     estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;

    /*To check the datum sent and received are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau32RxBuf[i] != u8Data)
        {
            #if DEBUG_INFO
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau32RxBuf[i] );
            #endif

            return ERROR;
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
            #endif
        }

        u8Data++;
    }

    return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int) (gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Slave TxRx data...\n");

    SSP_SlaveTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)
    {
        printf("Success\n");
    }
}

```

```
/*
*****
*****
*
* @brief      In this case, slave sent datum received from master back to
master.
*
*              This demo code couple with another named 'SSP_Master_Polling_TxRx'.
*
*****
*****/

int main()
{
    FLASH_WALLOW();

#ifdef SPC1158
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
    * Init the UART
    *
    * 1.Set the GPIO34/35 as UART FUNC
    *
    * 2.Enable the UART CLK
    *
    * 3.Set the rest para
    */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Set GPIOs as SSP function */
    GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);
    GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
    GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
    GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

    /* Set Output Strength(for high speed) */
    GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

    /*
    * Init SSP Baud Rate, Frame Format
    *
    * 1) Set master mode
    *
    */

```

```
* 2) Set frame data 8-bits width
*
* 3) Set transfer speed 30Mbps
*
* 4) Set pin SFRM active low
*
* 5) Set SCLK Polarity: SCLK is low when bus idle
*
* 6) Set SCLK Phase: sample data on SCLK rising edge
*/
SSP_Init(SSP, SSP_SLAVE_MODE, DataWidth_8bit, SSPBaudrate_1M,
SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

/* Disable FIFO Packed Mode */
SSP_DisableFIFOPackMode(SSP);

/* Enable SSP */
SSP_Enable(SSP);

/* SSP Slave Tx/Rx datum */
SSP_TxRx();

while (1)
{
}
}
```


2.2 Bulk 传输模式

2.2.1 主机发送与接收

本示例中演示 SPI 作为主机使用 Bulk 传输模式进行发送数据和接收数据。发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 主机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_MasterB2BTransceive ()函数进行发送与接收数据，并对数据进行校对；

Bulk 传输（主机发送与接收）

```
#define          DEBUG_INFO          1          /*debug
information flag*/
#define          T_BUFFER_SIZE       128        /*data size
of receive and transmit*/
#define          DataWidth_8bit      8          /*data width
size of SSP*/
#define          SSPBaudrate_1M      1000000   /*SSP baud
rate*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
 * JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
#define          MOSI_GPIO           GPIO_10
#define          MISO_GPIO           GPIO_11
#define          MOSI_GPIO_FUNC      GPIO10_SPI_MOSI
#define          MISO_GPIO_FUNC      GPIO11_SPI_MISO
#define          SCLK_GPIO           GPIO_8
#define          SFRM_GPIO           GPIO_9
#define          SCLK_GPIO_FUNC      GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC      GPIO9_SPI_SFRM
#else
#define          MOSI_GPIO           GPIO_38
#define          MISO_GPIO           GPIO_39
#define          MOSI_GPIO_FUNC      GPIO38_SPI_MOSI
#define          MISO_GPIO_FUNC      GPIO39_SPI_MISO
#define          SCLK_GPIO           GPIO_8
#define          SFRM_GPIO           GPIO_9
#define          SCLK_GPIO_FUNC      GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC      GPIO9_SPI_SFRM
#endif

uint32_t        gu32BuffSize        = T_BUFFER_SIZE;
uint32_t        gau32TxBuf[T_BUFFER_SIZE];
uint32_t        gau32RxBuf[T_BUFFER_SIZE];
ErrorStatus     estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;
```

```

/*To check the datum sent and received are the same*/
for(i = 0; i < gu32BuffSize; i++)
{
    if (gau32RxBuf[i] != u8Data)
    {
        #if DEBUG_INFO
        printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau32RxBuf[i] );
        #endif

        return ERROR;
    }
    else
    {
        #if DEBUG_INFO
        printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
        #endif
    }

    u8Data++;
}

return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int)(gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Master TxRx data...\n");

    SSP_MasterB2BTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)
    {
        printf("Success\n");
    }
}

/*****
*****
*
* @brief      In this case, Master sent datum to slave and then read these datum
back with Back-to-Back mode. There two ways
*              to trigger Back-to-Back mode:
*              1).TX FIFO is not empty while 'SFRM' signal is valid period.
*              2).If the data width is less then 16bits, and when the Back-to-
Back mode is enable. what's more, there is
*              a key point : The max speed of Back-to-Back mode is 4M,
because of the limit of MCU can cause the TX
*              FIFO empty to breakout the Back-to-Back mode.
*
*****/

```

```
*          This demo code couple with another named
'SSP_Slave_B2B_TxRx_Polling'.
*
*****
*****/

int main()
{
    FLASH_WALLOW();

#ifdef SPC1158
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Set GPIOs as SSP function */
    GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);
    GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
    GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
    GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

    /* Set Output Strength(for high speed) */
    GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

    /*
     * Init SSP Baud Rate, Frame Format
     * 1) Set master mode
     * 2) Set frame data 8-bits width
     * 3) Set transfer speed 30Mbps
     * 4) Set pin SFRM active low
     * 5) Set SCLK Polarity: SCLK is low when bus idle
     * 6) Set SCLK Phase: sample data on SCLK rising edge
     */
}
```

```

    SSP_Init(SSP, SSP_MASTER_MODE, DataWidth_8bit, SSPBaudrate_1M,
    SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

    /* Disable FIFO Packed Mode */
    SSP_DisableFIFOPackMode(SSP);

    /* Enable SSP */
    SSP_Enable(SSP);

    /* SSP Master Tx/Rx datum */
    SSP_TxRx();

    while (1)
    {
    }
}

```

2.2.2 从机发送与接收

本示例中演示 SPI 作为从机使用 Bulk 传输模式进行发送数据和接收数据。发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 从机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_SlaveTransceive ()函数进行发送与接收数据，并对数据进行校对；

Bulk 传输（从机发送与接收）

```

#define          DEBUG_INFO          1          /*debug
information flag*/
#define          T_BUFFER_SIZE       128       /*data size
of receive and transmit*/
#define          DataWidth_8bit      8         /*data width
size of SSP*/
#define          SSPBaudrate_1M     1000000   /*SSP baud
rate*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
    #define          MOSI_GPIO          GPIO_10
    #define          MISO_GPIO          GPIO_11
    #define          MOSI_GPIO_FUNC     GPIO10_SPI_MOSI
    #define          MISO_GPIO_FUNC     GPIO11_SPI_MISO
    #define          SCLK_GPIO          GPIO_8
    #define          SFRM_GPIO          GPIO_9
    #define          SCLK_GPIO_FUNC     GPIO8_SPI_SCLK
    #define          SFRM_GPIO_FUNC     GPIO9_SPI_SFRM
#else

```

```

#define          MOSI_GPIO          GPIO_38
#define          MISO_GPIO          GPIO_39
#define          MOSI_GPIO_FUNC     GPIO38_SPI_MOSI
#define          MISO_GPIO_FUNC     GPIO39_SPI_MISO
#define          SCLK_GPIO          GPIO_8
#define          SFRM_GPIO          GPIO_9
#define          SCLK_GPIO_FUNC     GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC     GPIO9_SPI_SFRM
#endif

uint32_t        gu32BuffSize        = T_BUFFER_SIZE;
uint32_t        gau32TxBuf[T_BUFFER_SIZE];
uint32_t        gau32RxBuf[T_BUFFER_SIZE];
ErrorStatus     estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;

    /*To check the datum sent and received are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau32RxBuf[i] != u8Data)
        {
            #if DEBUG_INFO
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
            gau32RxBuf[i] );
            #endif

            return ERROR;
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
            #endif
        }

        u8Data++;
    }

    return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int) (gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Slave TxRx data...\n");

    SSP_SlaveTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)

```

```

{
    printf("Success\n");
}

}

/*****
*****
*
* @brief      In this case, Slave sent the datum received from master back to
master with Back-to-Back mode. There two ways
*
*             to trigger Back-to-Back mode:
*
*             1).TX FIFO is not empty while 'SFRM' signal is valid period.
*
*             2).If the data width is less then 16bits, and when the Back-
to-Back mode is enable. what's more, there
*
*             is a key point : The max speed of Back-to-Back mode is 4M,
because of the limit of MCU can cause the
*
*             TX FIFO empty to breakout the Back-to-Back mode.
*
*
*             This demo code couple with another named
'SSP_Master_B2B_TxRx_Polling'.
*
*****
*****/

int main()
{
    FLASH_WALLOW();

#ifdef SPC1158
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
    * Init the UART
    *
    * 1.Set the GPIO34/35 as UART FUNC
    *
    * 2.Enable the UART CLK
    *
    * 3.Set the rest para
    */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Set GPIOs as SSP function */
    GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);

```

```
GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

/* Set Output Strength(for high speed) */
GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

/*
 * Init SSP Baud Rate, Frame Format
 *
 * 1) Set master mode
 *
 * 2) Set frame data 8-bits width
 *
 * 3) Set transfer speed 30Mbps
 *
 * 4) Set pin SFRM active low
 *
 * 5) Set SCLK Polarity: SCLK is low when bus idle
 *
 * 6) Set SCLK Phase: sample data on SCLK rising edge
 */
SSP_Init(SSP, SSP_SLAVE_MODE, DataWidth_8bit, SSPBaudrate_1M,
SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

/* Disable FIFO Packed Mode */
SSP_DisableFIFOPackMode(SSP);

/* Enable SSP */
SSP_Enable(SSP);

/* SSP Slave Tx/Rx datum */
SSP_TxRx();

while (1)
{
}
}
```

2.3 中断传输模式

2.3.1 主机发送与接收

本示例中演示 SPI 作为主机使用中断传输模式进行发送数据和接收数据。发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 主机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_SetTxFIFOTriggerThreshold() 函数进行设置发送 FIFO 的阈值，以及调用 SSP_EnableTxDataRequestInt ()函数使能 TX FIFO 为空时发生中断；
- 在中断服务函数中调用 SSP_MasterTransceive ()函数进行发送与接收数据，并对数据进行校对；

中断模式（主机发送与接收）

```

#define          DEBUG_INFO          1          /*debug
information flag*/
#define          T_BUFFER_SIZE       128        /*data
size of receive and transmit*/
#define          DataWidth_8bit      8          /*data
width size of SSP*/
#define          SSPBaudrate_1M      1000000   /*SSP
baud rate*/
#define          TX_FOFI_TH          0          /*TX FIFO
Threshold*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
#define          MOSI_GPIO            GPIO_10
#define          MISO_GPIO            GPIO_11
#define          MOSI_GPIO_FUNC       GPIO10_SPI_MOSI
#define          MISO_GPIO_FUNC       GPIO11_SPI_MISO
#define          SCLK_GPIO            GPIO_8
#define          SFRM_GPIO            GPIO_9
#define          SCLK_GPIO_FUNC       GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC       GPIO9_SPI_SFRM
#else
#define          MOSI_GPIO            GPIO_38
#define          MISO_GPIO            GPIO_39
#define          MOSI_GPIO_FUNC       GPIO38_SPI_MOSI
#define          MISO_GPIO_FUNC       GPIO39_SPI_MISO
#define          SCLK_GPIO            GPIO_8
#define          SFRM_GPIO            GPIO_9
#define          SCLK_GPIO_FUNC       GPIO8_SPI_SCLK
#define          SFRM_GPIO_FUNC       GPIO9_SPI_SFRM
#endif
    
```



```

uint32_t      gu32BuffSize      = T_BUFFER_SIZE;
uint32_t      gau32TxBuf[T_BUFFER_SIZE];
uint32_t      gau32RxBuf[T_BUFFER_SIZE];

uint32_t      gu32_cnt_ssp_tx_isr      = 0;
uint32_t      gu32_cnt_ssp_rx_isr      = 0;
ErrorStatus   estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;

    /*To check the datum sent and received are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau32RxBuf[i] != u8Data)
        {
            #if DEBUG_INFO
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau32RxBuf[i] );
            #endif

            return ERROR;
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
            #endif
        }

        u8Data++;
    }

    return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;
    gu32_cnt_ssp_tx_isr = 0;
    gu32_cnt_ssp_rx_isr = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int)(gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Master TxRx data...\n");

    /* Enable TxFIFO Empty Interrupt */
    SSP_EnableTxDataRequestInt(SSP);

    /* Wait transmit data finish */
    while (gu32_cnt_ssp_tx_isr < gu32BuffSize) {}

    /* Wait receive data finish */
    while (gu32_cnt_ssp_rx_isr < gu32BuffSize) {}
}

```

```

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)
    {
        printf("Success\n");
    }
}

/*****
*****
*
* @brief      In this case, Master sent data to slave and then read these datum
back from slave.
*
*           This demo code couple with another named 'SSP_Slave_TxRx_INT'.
*
*****
*****/

int main()
{
    FLASH_WALLOW();

#ifdef SPC1158
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
    * Init the UART
    *
    * 1.Set the GPIO34/35 as UART FUNC
    *
    * 2.Enable the UART CLK
    *
    * 3.Set the rest para
    */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Set GPIOs as SSP function */
    GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);
    GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
    GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
    GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

    /* Set Output Strength(for high speed) */
    GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);

```

```
GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

/*
 * Init SSP Baud Rate, Frame Format
 * 1) Set master mode
 * 2) Set frame data 8-bits width
 * 3) Set transfer speed 30Mbps
 * 4) Set pin SFRM active low
 * 5) Set SCLK Polarity: SCLK is low when bus idle
 * 6) Set SCLK Phase: sample data on SCLK rising edge
 */
SSP_Init(SSP, SSP_MASTER_MODE, DataWidth_8bit, SSPBaudrate_1M,
SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

/* Disable FIFO Packed Mode */
SSP_DisableFIFOPackMode(SSP);

/* Set Tx FIFO Threshold */
SSP_SetTxFIFOTriggerThreshold(SSP, TX_FOFI_TH);

/* Disable Tx FIFO Empty Interrupt */
SSP_DisableTxDataRequestInt(SSP);

/* Enable SSP */
SSP_Enable(SSP);

/* Enable CM4 Interrupt Request */
NVIC_EnableIRQ(SSP_IRQn);

/* SSP Master Tx/Rx datum */
SSP_TxRx();

while (1)
{
}

void SSP_IRQHandler(void)
{
    /* At first, send data to slave */
    if (gu32_cnt_ssp_tx_isr < gu32BuffSize)
    {
        SSP_MasterTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

        gu32_cnt_ssp_tx_isr += gu32BuffSize;
        gu32_cnt_ssp_rx_isr += gu32BuffSize;

        /* Disable Tx FIFO Empty Interrupt */
        SSP_DisableTxDataRequestInt(SSP);
    }
}
```

2.3.2 从机发送与接收

本示例中演示 SPI 作为从机使用中断传输模式进行发送数据和接收数据。发送的数据宽度为 8bit，速度为 1M，极性 CPOL=0、相位 CPHA=0。其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SPI 的 GPIO；
- 调用 SSP_Init()函数进行配置 SPI 从机模式、极性与相位、波特率，需要与从机的配置一致，并调用 SSP_Enable()函数使能 SPI 模块；
- 调用 SSP_SlaveTransceive ()函数进行发送与接收数据，并对数据进行校对；

中断模式（从机发送与接收）

```

#define          DEBUG_INFO          1          /*debug
information flag*/
#define          T_BUFFER_SIZE       128       /*data size
of receive and transmit*/
#define          DataWidth_8bit      8        /*data width
size of SSP*/
#define          SSPBaudrate_1M      1000000   /*SSP baud
rate*/

/*
 * If you use the 1168 and 1158 development boards, you cannot download using
 * JLink
 * because GPIO38 and GPIO39 are already in use.
 */
#if defined(SPD1148) || defined(SPD1178) || defined(SPD1188)
    #define      MOSI_GPIO            GPIO_10
    #define      MISO_GPIO            GPIO_11
    #define      MOSI_GPIO_FUNC      GPIO10_SPI_MOSI
    #define      MISO_GPIO_FUNC      GPIO11_SPI_MISO
    #define      SCLK_GPIO           GPIO_8
    #define      SFRM_GPIO           GPIO_9
    #define      SCLK_GPIO_FUNC      GPIO8_SPI_SCLK
    #define      SFRM_GPIO_FUNC      GPIO9_SPI_SFRM
#else
    #define      MOSI_GPIO            GPIO_38
    #define      MISO_GPIO            GPIO_39
    #define      MOSI_GPIO_FUNC      GPIO38_SPI_MOSI
    #define      MISO_GPIO_FUNC      GPIO39_SPI_MISO
    #define      SCLK_GPIO           GPIO_8
    #define      SFRM_GPIO           GPIO_9
    #define      SCLK_GPIO_FUNC      GPIO8_SPI_SCLK
    #define      SFRM_GPIO_FUNC      GPIO9_SPI_SFRM
#endif

uint32_t        gu32BuffSize          = T_BUFFER_SIZE;
uint32_t        gau32TxBuf[T_BUFFER_SIZE];
uint32_t        gau32RxBuf[T_BUFFER_SIZE];
ErrorStatus     estatus;

static ErrorStatus Check_Receive_Data(void)
{
    int i;
    uint8_t u8Data = 0;

    /*To check the datum sent and received are the same*/
    for(i = 0; i < gu32BuffSize; i++)

```

```

{
    if (gau32RxBuf[i] != u8Data)
    {
        #if DEBUG_INFO
        printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau32RxBuf[i] );
        #endif

        return ERROR;
    }
    else
    {
        #if DEBUG_INFO
        printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau32RxBuf[i] );
        #endif
    }

    u8Data++;
}

return SUCCESS;
}

void SSP_TxRx(void)
{
    int i;
    uint8_t u8Data = 0;

    /* Prepare data to be send */
    for (i = 0; i < (int)(gu32BuffSize); i++)
    {
        gau32TxBuf[i] = u8Data++;
    }

    printf("Slave TxRx data...\n");
    SSP_SlaveTransceive(SSP, gau32TxBuf, gau32RxBuf, gu32BuffSize);

    estatus = Check_Receive_Data();
    if (estatus == SUCCESS)
    {
        printf("Success\n");
    }
}

/*****
*****
*
* @brief      In this case, Slave with polling mode sent the datum back to
master received from master in INT mode.
*
*            This demo code couple with another named 'SSP_Master_TxRx_INT'.
*
*****
*****/

int main()
{
    FLASH_WALLOW();

#if defined(SPC1158)

```

```
FLASH_SetTiming(100000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
FLASH_SetTiming(200000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

Delay_Init();

/*
 * Init the UART
 *
 * 1.Set the GPIO34/35 as UART FUNC
 *
 * 2.Enable the UART CLK
 *
 * 3.Set the rest para
 */
GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
CLOCK_EnableModule(UART_MODULE);
UART_Init(UART, 38400);

/* Set GPIOs as SSP function */
GPIO_SetPinChannel(SCLK_GPIO, SCLK_GPIO_FUNC);
GPIO_SetPinChannel(SFRM_GPIO, SFRM_GPIO_FUNC);
GPIO_SetPinChannel(MOSI_GPIO, MOSI_GPIO_FUNC);
GPIO_SetPinChannel(MISO_GPIO, MISO_GPIO_FUNC);

/* Set Output Strength(for high speed) */
GPIO_SetOutStrength(SCLK_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(SFRM_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MOSI_GPIO, GPIO_OUT_STRENGTH_20MA);
GPIO_SetOutStrength(MISO_GPIO, GPIO_OUT_STRENGTH_20MA);

/*
 * Init SSP Baud Rate, Frame Format
 *
 * 1) Set master mode
 *
 * 2) Set frame data 8-bits width
 *
 * 3) Set transfer speed 30Mbps
 *
 * 4) Set pin SFRM active low
 *
 * 5) Set SCLK Polarity: SCLK is low when bus idle
 *
 * 6) Set SCLK Phase: sample data on SCLK rising edge
 */
SSP_Init(SSP, SSP_SLAVE_MODE, DataWidth_8bit, SSPBaudrate_1M,
SSP_FRM_ACTIVE_LOW,
        SSP_CLK_IDLE_LOW, SSP_SEND_ON_FALLING_EDGE);

/* Disable FIFO Packed Mode */
SSP_DisableFIFOPackMode(SSP);
```

```
/* Enable SSP */
SSP_Enable(SSP);

/* SSP Slave Tx/Rx datum */
SSP_TxRx();

while (1)
{
}
}
```

SPIN
TROL