



## Application Note

---

### SPC1068 SSP 单元使用指南

---

Revision 1 – September 2017

目录

1    SSP 概述.....5

2    SSP 各种模式实例 .....7

     2.1    SSP 主模式收发实例..... 7

     2.2    SSP 发送中断实例..... 10

3    修订记录 .....12

## 表格列表

表 1-1. SSP API 列表 .....	5
表 3-1. 文档修订记录 .....	12

## 图片列表

# 1 SSP 概述

SSP 单元是一个同步串行通信单元，常常被用于与多种外部设备进行通信，比如 ADC、音频解码器、通信解码器，是一种应用广泛的通信接口。

SPC1068 的 SSP 单元有以下特点：

- 可支持 TI 的 SSP 通信协议
- 可支持 Motorola 的 SPI 通信接口
- 最大速度 25M bps
- 支持灵活的位宽配置，硬件支持 4~32 位宽
- 支持主/从配置
- 带有 16x32 bit 或 32x16bit 的接收/发送 FIFO
- 高位左对齐，MSB 位先发送

在 SSP 的驱动库中，已经有下列驱动函数可供调动，可大大方便用户使用和理解。

表 1-1. SSP API 列表

API Name	Description
SSP_Enable(SSPx)	使能 SSPx
SSP_Disable(SSPx)	禁用 SSPx
SSP_EnableRxFifoOverflowInt(SSPx)	使能 SSPx 接收 FIFO 超上限中断
SSP_DisableRxFifoOverflowInt(SSPx)	禁用 SSPx 接收 FIFO 超上限中断
SSP_EnableTxFifoUnderflowInt(SSPx)	使能 SSPx 发送 FIFO 超下限中断
SSP_DisableTxFifoUnderflowInt(SSPx)	禁用 SSPx 发送 FIFO 超下限中断
SSP_EnableBitCountErrorInt(SSPx)	使能 SSPx Bit Count 错误中断
SSP_DisableBitCountErrorInt(SSPx)	禁用 SSPx Bit Count 错误中断
SSP_EnableRxTimeoutInt(SSPx)	使能 SSPx 接收超时中断
SSP_DisableRxTimeoutInt(SSPx)	禁用 SSPx 接收超时中断
SSP_EnableTxFifoEmptyInt(SSPx)	使能 SSPx 发送 FIFO 为空中断
SSP_DisableTxFifoEmptyInt(SSPx)	禁用 SSPx 发送 FIFO 为空中断
SSP_EnableRxFifoFullInt(SSPx)	使能 SSPx 接收 FIFO 满中断
SSP_DisableRxFifoFullInt(SSPx)	禁用 SSPx 接收 FIFO 满中断
SSP_EnableLoopBackMode(SSPx)	使能 SSPx 回环模式，TX 连接 RX
SSP_DisableLoopBackMode(SSPx)	禁用 SSPx 回环模式
SSP_GetTxFifoLevel(SSPx)	获取 SSPx 发送 FIFO 中数据个数
SSP_GetRxFifoLevel(SSPx)	获取 SSPx 接收 FIFO 中数据个数
SSP_SetRxFifoTriggerLevel(SSPx,u8RxLevel)	设置 SSPx 接收 FIFO 的触发门限为 u8RxLevel
SSP_SetTxFifoTriggerLevel(SSPx,u8TxLevel)	设置 SSPx 发送收 FIFO 的触发门限为 u8TxLevel
SSP_IsTxFifoNotFull(SSPx)	返回 SSPx 发送 FIFO 是否非满状态
SSP_IsRxFifoNotEmpty(SSPx)	返回 SSPx 接收 FIFO 是否非空状态
SSP_IsBusy(SSPx)	返回 SSPx 是否正忙状态
SSP_IsTxFifoServiceRequest(SSPx)	返回 SSPx 是否发生发送 FIFO 事件状态

SSP_IsRxFifoServiceRequest(SSPx)	返回 SSPx 是否发生接收 FIFO 事件状态
SSP_IsRxFifoOverflow(SSPx)	返回 SSPx 接收 FIFO 是否溢出状态
SSP_IsRxTimeoutPending(SSPx)	返回 SSPx 是否出现接收超时状态
SSP_IsTxFifoUnderflow(SSPx)	返回 SSPx 发送 FIFO 是否下溢状态
SSP_IsBusySyncSlaveClk(SSPx)	返回 SSPx 是否正在同步时钟状态
SSP_IsBitCountError(SSPx)	返回 SSPx 是否发生 Bit Count 错误状态
SSP_IsTxFifoHasOddSample(SSPx)	返回 SSPx 发送 FIFO 中是否有奇数个数据状态
SSP_IsRxFifoHasOddSample(SSPx)	返回 SSPx 接收 FIFO 中是否有奇数个数据状态
SSP_ClearRxFifoOverflowInt(SSPx)	清除 SSPx 接收 FIFO 溢出中断位
SSP_ClearTxFifoUnderflowInt(SSPx)	清除 SSPx 发送 FIFO 下溢中断位
SSP_ClearRxTimeoutInt(SSPx)	清除 SSPx 接收 FIFO 超时中断位
SSP_ClearBitCountErrorInt(SSPx)	清除 SSPx 接收 Bit Count 错误中断位
SSP_SetRxTimeout(SSPx, u32Timeout)	设置 SSPx 接收超时时长为 u32Timeout
SSP_SendData(SSPx, u32Data)	SSPx 发送数据 u32Data
SSP_ReceiveData(SSPx)	返回 SSPx 接收数据
void SSP_Init(SSP_REGS* SSPx, uint8_t u8MasterOrSlave, uint8_t u8DataSize, uint32_t u32Baudrate, SSP_FramePolarityEnum u1FramePolarity, SSP_ClockIdleLevelEnum u1ClockIdleLevel, SSP_SendDataEdgeEnum u1DataSendEdge)	初始化 SSP，参数为 1) u8MasterOrSlave - 主：1； 从：0 2) u8DataSize - 数据位宽：1~32 3) u32Baudrate - SSP 速率 4) u1FramePolarity - Frame 信号低/高有效 5) u1ClockIdleLevel - Clock 在 Idle 状态时是低/高电平 6) u1DataSendEdge - 数据在 Clock 的上升/下降沿发送
void SSP_InitEasy(SSP_REGS* SSPx, uint8_t u8MasterOrSlave, uint8_t u8DataSize, uint32_t u32Baudrate)	初始化 SSP，参数为： 1) u8MasterOrSlave - 主：1； 从：0 2) u8DataSize - 数据位宽：1~32 3) u32Baudrate - SSP 速率 其他假定条件为： 1) 时钟上升沿发送数据 2) FRM 信号为低有效 3) 时钟信号在空闲状态为低
void SSP_Send(SSP_REGS* SSPx, uint32_t *pu8Buffer, uint32_t u32Offset, uint32_t u32Count)	发送一组数据函数 1) pu8Buffer 发送数组指针 2) u32Offset 发送数组偏移 3) u32Count 发送数据数量
uint32_t SSP_Recv(SSP_REGS* SSPx, uint32_t *pu8Buffer, uint32_t u32Offset, uint32_t u32Count)	接收一组数据函数 1) pu8Buffer 接收数组指针 2) u32Offset 接收数组偏移 3) u32Count 接收数据数量

## 2 SSP 各种模式实例

### 2.1 SSP 主模式收发实例

在这个例子里，会演示如何使用 SSP 的基本功能，收发数据。在这个例子里，SSP 被设定为回环模式，即发出的数据将被自身接收。SSP 本身的配置情况为：

- 波特率 200000
- 位宽 8 bit
- FRM 低有效（CS 信号）
- CLK 空闲为低
- 数据在 CLK 上升沿发送

如果涉及 SSP 的简单应用，用户可以依照这个例子进行设定。

**重要 Note:**

SPC1068 在不同模式时(Master/Slave)，需手动配置两个 Data 管脚的方向，代码如下：

#### Example Code

```
/* 主机模式 */
GPIO_SetPinChannel (GPIO_19,GPIO19_SPI_CLK);
GPIO_SetPinChannel (GPIO_20,GPIO20_SPI_FRM);
GPIO_SetPinChannel (GPIO_21,GPIO21_SPI_MI);
GPIO_SetPinChannel (GPIO_22,GPIO22_SPI_MO);

/* 从机模式 */
GPIO_SetPinChannel (GPIO_19,GPIO19_SPI_CLK);
GPIO_SetPinChannel (GPIO_20,GPIO20_SPI_FRM);
GPIO_SetPinChannel (GPIO_21,GPIO21_SPI_SO);
GPIO_SetPinChannel (GPIO_22,GPIO22_SPI_SI);
```

当设定为主机模式，并且选择 GPIO21 作为数据输入管脚，选择 GPIO22 作为数据输出，当作为从机时，则刚好相反。

SSP 主模式收发实例代码如下：

#### Example Code

```
/* Enable UART and SSP clock */
CLOCK_EnableModule(UART_MODULE);
CLOCK_EnableModule(SSP_MODULE);

/* Configure GPIO pin as UART function */
GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
/* UART Init */
UART_Init(UART, 38400);

/* Init SSP as 1) Master Mode 2) 8 bit frame width, baudrate 2M bps */
SSP_InitEasy(SSP, SSP_MASTER_MODE, 8, 2000000);

/* Enable loopback mode for test purpose */
SSP_EnableLoopBackMode(SSP);
/* Enable SSP */
SSP_Enable(SSP);

/* Print data to be sent */
printf("Data to Send = %4X    ", u8DataToSend);

/* Send data */
SSP_SendData(SSP, u8DataToSend);

while(1)
{
    /* Wait until SSP receive buffer is not empty, eg. SSP received data */
    while(!SSP_IsRxFifoNotEmpty(SSP)) {}

    /* Get SSP receive data from buffer */
    u8DataReceived = SSP_ReceiveData(SSP);
    /* Print data received */
    printf("Data Received = %4X\n", u8DataReceived);

    Delay_ms(1000);

    u8DataToSend = u8DataReceived + 1;
    /* Print data to send */
    printf("Data to Send = %4X    ", u8DataToSend);
}
```



```
/* Send data */  
SSP_SendData (SSP, u8DataToSend);  
}
```

程序下载后，观察程序运行，串口波特率设定为 38400HZ，可以看到发送数据和接收数据一致，则代码运行正常。

## 2.2 SSP 发送中断实例

在很多应用场景，比如 SSP 高速通信中，需要按照最大速率进行发送，这时候当 SSP 空闲时就要启动下一次发送。如果有这样的需求，可以使用 SPC1068 SSP 的发送中断，当 SPC1068 监测到 SSP 的发送 FIFO 快空的时候，可以重新把数据装满发送 FIFO，通过这种机制，可以保证 SSP 一直发送，没有空闲。

具体配置如下：

关于 SSP 的基本配置这里不再重复，参考之前的 2.1 例子。

### 1) 中断使能部分

#### Example Code

```
/* Set Tx FIFO trigger threshold */
SSP_SetTxFifoTriggerLevel(SSP, 0);

/* Enable Tx FIFO empty interrupt */
SSP_EnableTxFifoEmptyInt(SSP);

/* Enable SSP interrupt in CPU side */
NVIC_EnableIRQ(SSP_IRQn);
```

### 2) 中断函数部分

中断函数部分主要实现了读取 SSP 接收的数据，打印接收数据，打印发送数据，再一次发送数据的功能，其中的延时是为了让用户更好的通过打印消息观察实验结果。

#### Example Code

```
void SSP_IRQHandler()
{
    int i;

    /* Receive Data */
    SSP_Rcv(SSP, u8DataToRev, 0, 8);
    printf("Rev  (0) %X, (1) %X, (2) %X, (3) %X, (4) %X, (5) %X, (6) %X, (7) %X \n\n",
           u8DataToRev[0],
           u8DataToRev[1],
           u8DataToRev[2],
           u8DataToRev[3],
           u8DataToRev[4],
           u8DataToRev[5],
           u8DataToRev[6],
           u8DataToRev[7]);

    Delay_ms(2000);

    for(i = 0; i < 8; i++)
```

```
{
    u8DataToSend[i]++;
}

/* Send Data */
printf("Send (0) %X, (1) %X, (2) %X, (3) %X, (4) %X, (5) %X, (6) %X, (7) %X \n",
    u8DataToSend[0],
    u8DataToSend[1],
    u8DataToSend[2],
    u8DataToSend[3],
    u8DataToSend[4],
    u8DataToSend[5],
    u8DataToSend[6],
    u8DataToSend[7]);
SSP_Send(SSP, u8DataToSend, 0, 8);

Delay_ms(100);
}
```

具体的实验结果，可以通过串口工具或者 ISP 工具，看看接收数据和发送数据是否一致，是否被打印。

### 3 修订记录

表 3-1. 文档修订记录

日期	版本	修改内容
2017-09-14	1	初始版本