SPC1169 I2C 使用指南

版本 A/0 – 2023 年 4 月

概述

I2C 总线用于连接微控制器及其外围设备，具有接口线少、控制简单、通信速率较高等优点，广泛应用于微控制器、LCD 驱动器、触摸屏、存储器、键盘等接口。

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 4 月 17 日	XuQing He	Released	首次发布。

术语或缩写

术语或缩写	描述

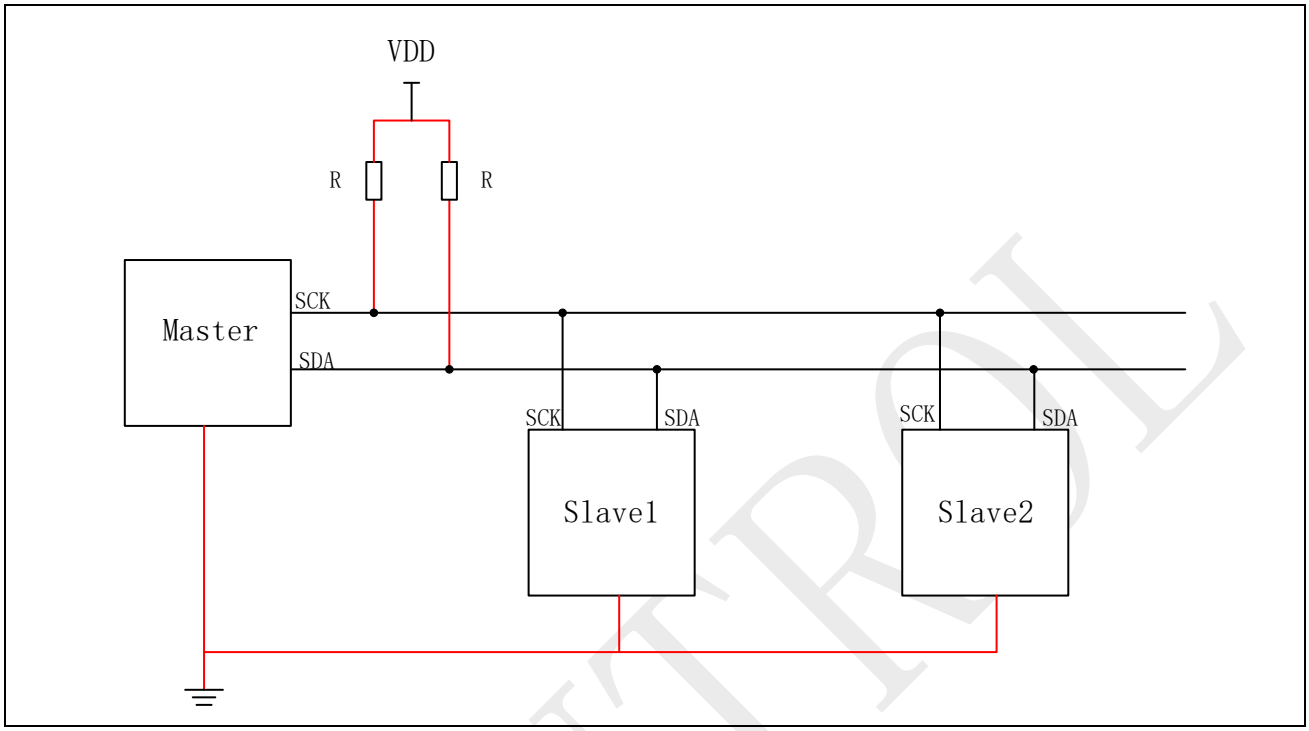
1 I2C 特性

SPC1169 内建一个 I2C 单元，其 I2C 单元有以下特点：

- 支持三种速度模式（标准速度模式 100kb/s、快速模式 400kb/s、高速模式 3.4Mb/s）；
- 具有灵活的时钟分频比可以实现不同的速率以及可编程的时钟极性和相位；
- 内建宽度 32byte 深度 16 的发送和接收 FIFO；
- 支持 7bit 和 10bit 地址寻址；

2 I2C 使用注意事项

图 2-1：I2C 连接示意图



I2C 接口采用开漏输出的方式输出，导致无法输出高电平，因此必须上拉电阻实现输出高电平的能力如**错误!未找到引用源。**所示。如果上拉了电阻，但在实际应用当中发现时钟的频率低于理论的时钟频率以及时钟的上升比较迟缓时，可以通过减小上拉电阻来提高驱动能力，适当改善时钟波形来提高频率。但上拉电阻并不是越小越好，对于上拉电阻的选择也是有一定的范围。可以产品实际情况计算出可以上拉电阻的最小值 R_{min} 和最大值 R_{max}。

根据 Spintrol 对应产品数据手册 I2C 的 IO 特性，I2C 的引脚的达到最小高电平阈值为 V_{DVDD5} - 0.5V，达到最大低电平的阈值为 0.5V，某时刻电压的计算表达式可以表示为：

$$V_t = V_{DVDD5}\left(1 - e^{-\frac{t}{RC}}\right)$$

其中 t 为自上电开始至达到 V_t 所需的时间，RC 是时间常数。以 V_{DVDD5}=5V 为例（以实际测量为准）计算到达最大低电平阈值时间 t_1 和到达最小高电平阈值时间 t_2。

到达最大低电平阈值时间 t_1 为：

$$V_{t_1} = 0.5 = V_{DVDD5}\left(1 - e^{-\frac{t_1}{RC}}\right)$$

则

$$t_1 = 0.1053605 * RC$$

到达最小高电平阈值时间 t_2 为：

$$V_{t_2} = V_{DVDD5} - 0.5 = V_{DVDD5}\left(1 - e^{-\frac{t_2}{RC}}\right)$$

则

$$t_2 = 2.3025850 * RC$$

由V_{t_1}到达V_{t_2}时间T为：

$$T = t_2 - t_1 = 2.1972245 * RC$$

由于 IIC 设计指标规定了 SCL/SDA 上升时间 t_r 最大值，预估总线电容 C_b。标准模式 t_r 时间为 1000ns，快速模式 t_r 时间为 300ns，快速 plus 模式 t_r 时间为 120ns。标准模式 C_b 电容为 400pF，快速模式 C_b 电容为 400pF，快速模式 plus C_b 电容为 550pF。

所以可以由以上表达式推算出 R_{max} 为：

$$R_{max} = \frac{t_r}{2.1972245 * C_b}$$

且 IIC 设计指标规定上拉电阻最小值可表示为：

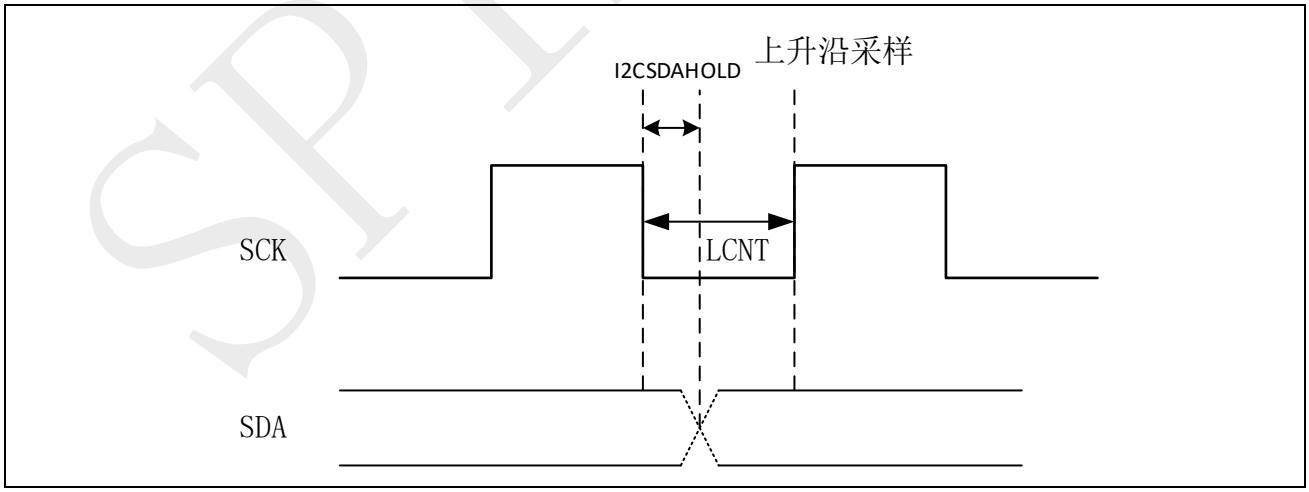
$$R_{min} = \frac{V_{DVDD5} - V_{OL}}{I_{OL}}$$

其中 V_{OL} 为最大低电平阈值 0.5，I_{OL} 为引脚的额定最小电流（根据设置引脚的输出强度不同而不同，当 STRENGTH 为 1 时，最小电流为 26.1mA）。

需要注意的是，I2C 在通信的过程不仅与上拉电阻的选择有关，还与 I2C 的通信的距离有关。I2C 总线适用于短距离通信，一般通信距离在 20cm 左右才能保证 I2C 的信号质量。为确保主从设备通信的信号质量，两设备之间需要共地用来确保两设备一致的基准电平，否则在通信过程中容易产生毛刺干扰通信数据。

在 Spintrol 的 I2C 控制器中提供了 I2CSDAHOLD 寄存器配置，方便用户根据自己的实际情况灵活调节数据在低电平哪个时刻进行发送数据，但 I2CSDAHOLD 的配置不能超过实际 I2C 时钟的 LCNT 低电平时间，如图 2-2 所示。

图 2-2：I2CSDAHOLD 数据保持时间配置



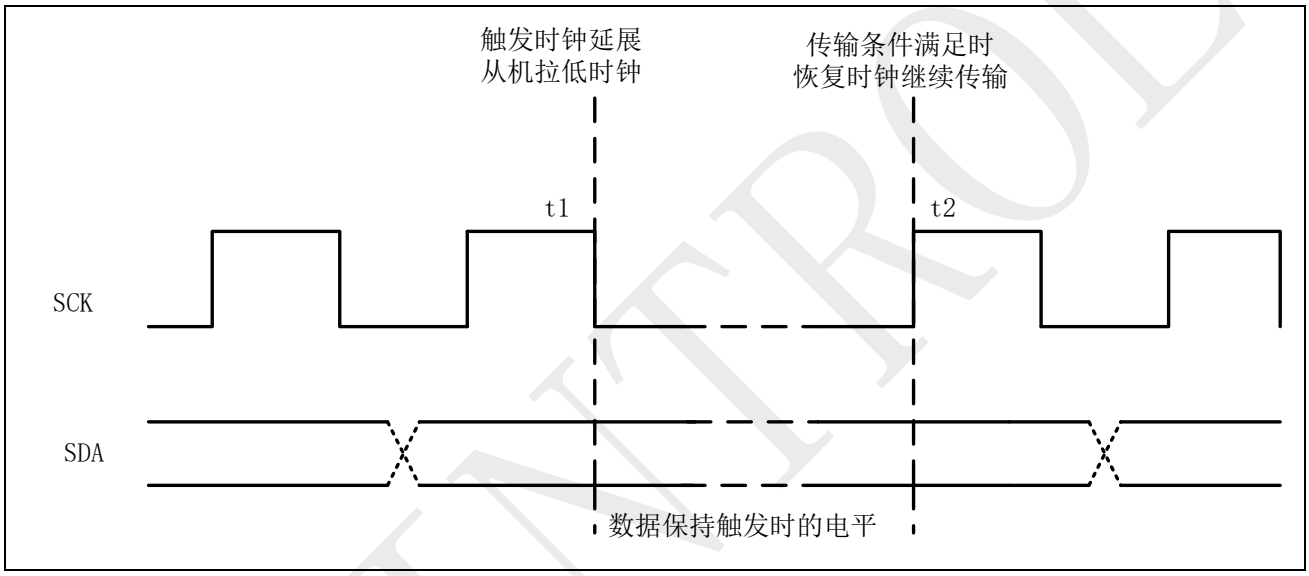
I2CSDAHOLD 为时钟下降沿开始到数据开始发送的时间。当 I2C 设备进行发送时，可以设置 I2CSDAHOLD 的值进行调节数据发送的时间；当 I2C 设备进行接收时，设置 I2CSDAHOLD 时间无效。

I2C 当作从机时，还具有时钟延展功能，是否使能时钟延展功能可以通过 CLKSTRETCH 寄存器进行配置。I2C 的时钟延展功能就是从机拉低时钟，从而主动终止数据传输；在以下两种情况下从机将拉低时钟：

- 当从机发送数据，且发送 FIFO 为空；
- 当从机接收数据，且接收 FIFO 为满；

当以上状态不存在时，从机将释放时钟。如图 2-3 所示，当 t1 时刻从机进行接收数据，但接收 FIFO 已经满不能接收数据时，就会触发时钟延展功能，从机会把时钟拉低，数据信号维持触发前的状态，直到接收 FIFO 不为满时时，从机将释放时钟，从而恢复数据传输。

图 2-3：时钟延展



3 I2C 实例

I2C 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，数据传输格式如图 3-1 所示。

START 信号：时钟为高电平时数据线由高变低。

STOP 信号：时钟为高电平时数据线有低变高。

Bulk 传输模式：所有数据一次性传输，只发一次 START 信号、STOP 信号、从设备地址。

Poll 传输模式：每发一个数据，会发一次 START 信号、STOP 信号、从设备地址。

中断传输模式：每发发送 FIFO 深度的数据，会发送一次 START 信号、STOP 信号、从设备地址。

图 3-1：三种模式数据传输格式

3.1 Bulk 传输模式

3.1.1 主机发送与接收

本示例中演示 I2C 作为主机使用 Bulk 传输模式进行发送数据和接收数据。当示例的 MASTER_TRANSMIT 宏为 1 时进行发送数据，当 MASTER_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_EnableInt()函数使能探测 STOP 信号中断，通过判断是否进入中断来判断通信是否完成；
- 调用 I2C_MasterInit()函数初始化 I2C 为 master 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_MasterBulkWrite()或者 I2C_MasterBulkRead()函数进行发送或者接收数据；

Bulk 传输(主机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#define            DEBUG_INFO            1              /* the controlling flag of
printf info*/
#define            MASTER_TRANSMIT        1              /* 1: master transmit 0:
master receive */
#define            Tx_Full_INT_TH         0              /* threshold will trigger
Tx full INT */
#define            Rx_Empty_INT_TH        0              /* threshold will trigger
Rx empty INT */
#define            T_BUFFER_SIZE          128
#define            I2C_SPEED              400000
#define            I2C_Slave_ADDR         0x9            /* IIC Slave ADDR */

uint32_t           gu32BuffSize           = T_BUFFER_SIZE;
uint8_t            gau8TxBuf[T_BUFFER_SIZE];
uint8_t            gau8RxBuf[T_BUFFER_SIZE];
uint32_t           gu32_cnt_i2c_stop_isr;
ErrorStatus        estatus;

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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}


void Master_Bulk_TxRX_data(I2C_REGS* I2Cx)
{
    int i;
    uint8_t u8Data = 0;
    uint32_t u32IsrCnt = 0;                                  /* Interrupt Check
variable */


    if (MASTER_TRANSMIT)
    {
```

```
        /* Generate random datum to transmit */
        for(i=0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

        printf("Master Tx data...\n");
        I2C_MasterBulkWrite(I2Cx, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8TxBuf,
gu32BuffSize);
    }
    else
    {
        printf("Master Rx data...\n");

        /*Read the data had sent to the slave back and put them in 'gau8RxBuf'*/
        I2C_MasterBulkRead(I2Cx, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
gu32BuffSize);

        /*To check the datum sent and recieved are the same*/
        Check_Receive_Data();
    }

    /*
    * In the interface of 'I2C_MasterBulkWriteData()', master will wait for the
Tx FIFO empty
    * after this bulk, in other words, the master will sent a 'STOP' CMD to the
IIC, and
    * then sent a 'RESTART' at the next bulk. we can count the bulk we has sent
to get
    * the INT count had entered.
    */
    u32IsrCnt += 1;

    /* Wait for I2C INT done */
    Delay_Us(300);

    /*
    * Check interrupt counter, if the INT count is not equal the bulk we has
sent, there
    * must be something wrong had happened.
    */
    if( gu32_cnt_i2c_stop_isr != u32IsrCnt )
    {
        printf("[Error]@%4d: I2C ISR counter not right. expect %d, but is %d",
__LINE__,
                                u32IsrCnt, gu32_cnt_i2c_stop_isr);
    }
    else
    {
        printf("Success\n");
    }
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
```

```
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPD1179)
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);
    #else
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO32, PIN_GPIO32_I2C_SCL);
    PIN_SetChannel(PIN_GPIO33, PIN_GPIO33_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO32, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO33, PIN_OUT_STRENGTH_20MA);
    #endif

    /*
    * Tx/Rx FIFO threshold set as 0 means letting MCU process data once
    * FIFO has one entry data.
    */
    I2C_SetTxFIFOThreshold(I2C, Tx_Full_INT_TH);
    I2C_SetRxFIFOThreshold(I2C, Rx_Empty_INT_TH);

    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Enable the INT of detecting the STOP of I2C */
    I2C_EnableInt(I2C,I2C_INT_STOP_DETECT);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Initial I2C as Master */
    estatus = I2C_MasterInit(I2C, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC master initial FAIL] I2C clock is not fast enough to
support the speed\n");
        return 0;
    }

    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Master start to transmit and receive data */
    Master_Bulk_TxRX_data(I2C);
```

```
    while(1)
    {

    }
}


void I2C_IRQHandler(void)
{
    if(I2C_GetIntFlag(I2C,I2C_INT_STOP_DETECT))
    {
        gu32_cnt_i2c_stop_isr ++ ;

        I2C_ClearInt(I2C, I2C_INT_STOP_DETECT|I2C_INT_GLOBAL);
    }
    else
    {
        printf("[%s Error] Stop detect interrupt error\n", __func__);
    }
}
```

3.1.2 从机发送与接收

本示例中演示 I2C 作为从机使用 Bulk 传输模式进行发送数据和接收数据。当示例的 SLAVE_TRANSMIT 宏为 1 时进行发送数据，当 SLAVE_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_EnableInt()函数使能探测 STOP 信号中断，通过判断是否进入中断来判断通信是否完成；
- 调用 I2C_SlaveInit()函数初始化 I2C 为 slave 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_SlaveBulkWrite()或者 I2C_SlaveBulkRead()函数进行发送或者接收数据；

Bulk 传输(从机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#define               DEBUG_INFO              1                       /* The
controlling flag of printf info*/
#define               SLAVE_TRANSMIT           0                       /* 1:
slave transmit 0: slave receive */
#define               Tx_Full_INT_TH           0                       /*
Threshold will trigger Tx full INT */
#define               Rx_Empty_INT_TH          0                       /*
Threshold will trigger Rx empty INT */
#define               T_BUFFER_SIZE           128
#define               I2C_Slave_ADDR           0x9                      /* IIC
Slave ADDR */
```

```
#define                I2C_SPEED                400000

uint32_t               gu32BuffSize               = T_BUFFER_SIZE;
uint8_t                gau8TxBuf[T_BUFFER_SIZE];
uint8_t                gau8RxBuf[T_BUFFER_SIZE];
uint32_t               gu32_cnt_i2c_stop_isr;
ErrorStatus            estatus;

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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}

void Slave_Bulk_TxRX_data(I2C_REGS* I2Cx)
{
    int i;
    uint8_t u8Data = 0;
    uint32_t u32IsrCnt = 0;                                    /*Interrupt Check
variable*/

    if (SLAVE_TRANSMIT)
    {
        /* Generate random datum to transmit */
        for(i=0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

        printf("Slave Tx data...\n");
        I2C_SlaveBulkWrite(I2Cx, gau8TxBuf, gu32BuffSize);
    }
    else
    {
        printf("Slave Rx data...\n");
        I2C_SlaveBulkRead(I2Cx, gau8RxBuf, gu32BuffSize);

        /*To check the datum sent and recieved are the same*/
        Check_Receive_Data();
    }

    /*
    * The master will sent a 'STOP' CMD to the IIC, and then sent a 'RESTART'
at the
```

```
    * next bulk. we can count the bulk we has sent to get the INT count had
entered.
    */
    u32IsrCnt += 1;

    /* Wait for I2C INT done */
    Delay_Us(300);

    /*
    * Check interrupt counter, if the INT count is not equal the bulk we has
sent, there
    * must be something wrong had happened.
    */
    if( gu32_cnt_i2c_stop_isr != u32IsrCnt )
    {
        printf("[Error]@%4d: I2C ISR counter not right. expect %d, but is %d",
__LINE__,
                            u32IsrCnt, gu32_cnt_i2c_stop_isr);
    }
    else
    {
        printf("Success\n");
    }
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPD1179)
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);
    #else
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO32, PIN_GPIO32_I2C_SCL);
    PIN_SetChannel(PIN_GPIO33, PIN_GPIO33_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO32, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO33, PIN_OUT_STRENGTH_20MA);
    #endif
```

```
    /*
    * Tx/Rx FIFO threshold set as 0 means letting MCU process data once
    * FIFO has one entry data.
    */
    I2C_SetTxFIFOThreshold(I2C, Tx_Full_INT_TH);
    I2C_SetRxFIFOThreshold(I2C, Rx_Empty_INT_TH);

    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Enable the INT of detecting the STOP of I2C */
    I2C_EnableInt(I2C,I2C_INT_STOP_DETECT);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Initial I2C as Slave and set the speed as 400K */
    estatus = I2C_SlaveInit(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC slave initial FAIL] I2C clock is not fast enough to support
the speed\n");
        return 0;
    }

    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable MCU INT Request */
    I2C_Enable(I2C);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Slave start to transmit and receive data */
    Slave_Bulk_TxRX_data(I2C);

    while(1)
    {

    }
}


void I2C_IRQHandler(void)
{
    if(I2C_GetIntFlag(I2C,I2C_INT_STOP_DETECT))
    {
        gu32_cnt_i2c_stop_isr ++ ;

        I2C_ClearInt(I2C, I2C_INT_STOP_DETECT|I2C_INT_GLOBAL);
    }
    else
    {
        printf("[%s Error] Stop detect interrupt error\n", __func__);
    }
}
```

3.2 Poll 传输模式

3.2.1 主机发送与接收

本示例中演示 I2C 作为从机使用 Poll 传输模式进行发送数据和接收数据。当示例的 MASTER_TRANSMIT 宏为 1 时进行发送数据，当 MASTER_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_EnableInt()函数使能探测 STOP 信号中断，通过判断是否进入中断来判断通信是否完成；
- 调用 I2C_MasterInit ()函数初始化 I2C 为 master 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_MasterWrite()或者 I2C_MasterRead()函数进行发送或者接收数据；

Polling 模式(主机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#define            DEBUG_INFO            1             /* the controlling flag of
printf info*/
#define            MASTER_TRANSMIT        1             /* 1: master transmit 0:
master receive */
#define            Tx_Full_INT_TH         0             /* threshold will trigger
Tx full INT */
#define            Rx_Empty_INT_TH        0             /* threshold will trigger
Rx empty INT */
#define            T_BUFFER_SIZE          128
#define            I2C_SPEED              400000
#define            I2C_Slave_ADDR         0x9           /* IIC Slave ADDR */

uint32_t           gu32BuffSize           = T_BUFFER_SIZE;
uint8_t            gau8TxBuf[T_BUFFER_SIZE];
uint8_t            gau8RxBuf[T_BUFFER_SIZE];
uint32_t           gu32_cnt_i2c_stop_isr;
ErrorStatus        estatus;

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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
```

```
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}

void Master_TxRX_data(I2C_REGS* I2Cx)
{
    int i;
    uint8_t u8Data = 0;
    uint32_t u32IsrCnt = 0;                                    /*Interrupt Check
variable*/

    if (MASTER_TRANSMIT)
    {
        /* Generate random datum to transmit */
        for(i=0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

        printf("Master Tx data...\n");
        I2C_MasterWrite(I2Cx, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8TxBuf,
gu32BuffSize);
    }
    else
    {
        printf("Master Rx data...\n");

        /*Read the data had sent to the slave back and put them in 'gau8RxBuf'*/
        I2C_MasterRead(I2Cx, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
gu32BuffSize);

        /*To check the datum sent and recieved are the same*/
        Check_Receive_Data();
    }

    u32IsrCnt += gu32BuffSize;

    /* Wait for I2C INT done */
    Delay_Us(300);

    /*
    * Check interrupt counter, if the INT count is not equal the bytes we has
sent, there
    * must be something wrong had happened.
    */
    if( gu32_cnt_i2c_stop_isr != u32IsrCnt)
    {
        printf("[Error]@%4d: I2C ISR counter not right. expect %d, but is %d",
__LINE__,
                                u32IsrCnt, gu32_cnt_i2c_stop_isr);
    }
    else
    {
        printf("Success\n");
    }
```

```
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPD1179)
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);
    #else
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO32, PIN_GPIO32_I2C_SCL);
    PIN_SetChannel(PIN_GPIO33, PIN_GPIO33_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO32, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO33, PIN_OUT_STRENGTH_20MA);
    #endif

    /*
    * Tx/Rx FIFO threshold set as 0 means letting MCU process data once
    * FIFO has one entry data.
    */
    I2C_SetTxFIFOThreshold(I2C, Tx_Full_INT_TH);
    I2C_SetRxFIFOThreshold(I2C, Rx_Empty_INT_TH);

    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Enable the INT of detecting the STOP of I2C */
    I2C_EnableInt(I2C,I2C_INT_STOP_DETECT);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Initial I2C as Master */
    estatus = I2C_MasterInit(I2C, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC master initial FAIL] I2C clock is not fast enough to
support the speed\n");
```

```
        return 0;
    }

    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Master start to transmit and receive data */
    Master_TxRX_data(I2C);

    while(1)
    {

    }
}


void I2C_IRQHandler(void)
{
    if(I2C_GetIntFlag(I2C,I2C_INT_STOP_DETECT))
    {
        gu32_cnt_i2c_stop_isr ++ ;

        I2C_ClearInt(I2C, I2C_INT_STOP_DETECT|I2C_INT_GLOBAL);
    }
    else
    {
        printf("[%s Error] Stop detect interrupt error\n", __func__);
    }
}
```

3.2.2 从机发送与接收

本示例中演示 I2C 作为从机使用 Poll 传输模式进行发送数据和接收数据。当示例的 SLAVE_TRANSMIT 宏为 1 时进行发送数据，当 SLAVE_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_EnableInt()函数使能探测 STOP 信号中断，通过判断是否进入中断来判断通信是否完成；
- 调用 I2C_SlaveInit()函数初始化 I2C 为 slave 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_SlaveWrite()或者 I2C_SlaveRead()函数进行发送或者接收数据；

Poll 模式(从机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
```

```
#endif

#define               DEBUG_INFO              1                           /* The
controlling flag of printf info*/
#define               SLAVE_TRANSMIT           0                           /* 1:
slave transmit 0: slave receive */
#define               Tx_Full_INT_TH           0                           /*
Threshold will trigger Tx full INT */
#define               Rx_Empty_INT_TH          0                           /*
Threshold will trigger Rx empty INT */
#define               T_BUFFER_SIZE            128
#define               I2C_Slave_ADDR           0x9                         /* IIC
Slave ADDR */
#define               I2C_SPEED                400000

uint32_t              gu32BuffSize              = T_BUFFER_SIZE;
uint8_t               gau8TxBuf[T_BUFFER_SIZE];
uint8_t               gau8RxBuf[T_BUFFER_SIZE];
uint32_t              gu32_cnt_i2c_stop_isr;
ErrorStatus           estatus;

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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}

void Slave_TxRX_data(I2C_REGS* I2Cx)
{
    int i;
    uint8_t u8Data = 0;
    uint32_t u32IsrCnt = 0;                                 /* Interrupt Check
variable */

    if (SLAVE_TRANSMIT)
    {
        /* Generate random datum to transmit */
        for(i = 0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

        printf("Slave Tx data...\n");
        I2C_SlaveWrite(I2Cx, gau8TxBuf, gu32BuffSize);
```

```
    }
    else
    {
        printf("Slave Rx data...\n");

        /* Read the data had sent to the master just now back and put them into
the 'gau8RxBuf' */
        I2C_SlaveRead(I2Cx, gau8RxBuf, gu32BuffSize);

        /*To check the datum sent and recieved are the same*/
        Check_Receive_Data();
    }


    u32IsrCnt += gu32BuffSize;

    /* Wait for I2C INT done */
    Delay_Us(300);

    /*
    * Check interrupt counter, if the INT count is not equal the bytes we has
sent, there
    * must be something wrong had happened.
    */
    if( gu32_cnt_i2c_stop_isr != u32IsrCnt )
    {
        printf("[Error]@%4d: I2C ISR counter not right. expect %d, but is %d",
__LINE__,
                                u32IsrCnt, gu32_cnt_i2c_stop_isr);
    }
    else
    {
        printf("Success\n");
    }
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);

    /*
    * Tx/Rx FIFO threshold set as 0 means letting MCU process data once
```

```
    * FIFO has one entry data.
    */
    I2C_SetTxFIFOThreshold(I2C, Tx_Full_INT_TH);
    I2C_SetRxFIFOThreshold(I2C, Rx_Empty_INT_TH);

    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Enable the INT of detecting the STOP of I2C */
    I2C_EnableInt(I2C,I2C_INT_STOP_DETECT);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Initial I2C as Slave and set the speed as 400K */
    estatus = I2C_SlaveInit(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC slave initial FAIL] I2C clock is not fast enough to support
the speed\n");
        return 0;
    }

    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Slave start to transmit and receive data */
    Slave_TxRX_data(I2C);

    while(1)
    {

    }
}


void I2C_IRQHandler(void)
{
    if(I2C_GetIntFlag(I2C,I2C_INT_STOP_DETECT))
    {
        gu32_cnt_i2c_stop_isr ++ ;

        I2C_ClearInt(I2C, I2C_INT_STOP_DETECT|I2C_INT_GLOBAL);
    }
    else
    {
        printf("[%s Error] Stop detect interrupt error\n", __func__);
    }
}
```

3.3 中断传输模式

3.3.1 主机发送与接收

本示例中演示 I2C 作为主机使用中断传输模式进行发送数据和接收数据。当示例的 MASTER_TRANSMIT 宏为 1 时进行发送数据，当 MASTER_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_MasterInit ()函数初始化 I2C 为 master 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_SetAddressMode()与 I2C_SetTargetAddress()函数进行设置 I2C 地址模式以及设备地址；
- 当为接收时需要调用 I2C_MasterReadCmd()函数给从机发送读请求；
- 调用 I2C_SetTxFIFOThreshold()或者 I2C_SetRxFIFOThreshold()函数设置发送或者接收 FIFO 的阈值；
- 调用 I2C_EnableInt()函数使能发送与接收中断
- 在中断服务函数中根据发送与接收 FIFO 的状态进行发送或者接收数据；

中断模式(主机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define            DEBUG_INFO            1              /* the controlling flag of
printf info*/
#define            MASTER_TRANSMIT        1              /* 1: master transmit 0:
master receive */
#define            RX_INT_TH             0xf            /* Tx thresgold as 1 entry
*/
#define            TX_INT_TH             0x0            /* Tx thresgold as 1 entry
*/
#define            TxRx_DATA_LEN          16             /* Send receive 16 Byte
*/
#define            T_BUFFER_SIZE          128
#define            I2C_SPEED             400000
#define            I2C_Slave_ADDR         0x9            /* IIC Slave ADDR */


uint32_t           gu32BuffSize           = T_BUFFER_SIZE;
uint8_t            gau8TxBuf[T_BUFFER_SIZE];
uint8_t            gau8RxBuf[T_BUFFER_SIZE];
uint32_t           num                   = 0;
uint32_t           u32IsrCnt              = 0;

ErrorStatus        estatus;

uint8_t            u32DetectAck           = 0;
```

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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}


void Master_TxRX_data(I2C_REGS* I2Cx)
{
    uint32_t i;
    uint8_t u8Data = 0;

    /* Set Address Mode */
    I2C_SetAddressMode(I2Cx, I2C_ADDR_7BIT);

    /* Set Target Slave Address*/
    I2C_SetTargetAddress(I2Cx, I2C_Slave_ADDR);

    if (MASTER_TRANSMIT)
    {
        /* Generate random datum to transmit */
        for(i = 0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

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

        I2C_SetTxFIFOThreshold(I2Cx, TX_INT_TH);

        I2C_EnableInt(I2Cx, I2C_INT_TX_REQ);
    }
    else
    {
        printf("Master Rx data...\n");
        for (i = 0; i < TxRx_DATA_LEN; i++)
        {
            I2C_MasterReadCmd(I2Cx);

            /* Wait I2C ACK */
            if (u32DetectAck == 0)
            {
                while(I2C_GetIntRawFlag(I2C, I2C_INT_ACK_DETECT) == 0) {}
```

```
                u32DetectAck = 1;
            }
        }

        I2C_SetRxFIFOThreshold(I2Cx, RX_INT_TH);

        I2C_EnableInt(I2Cx, I2C_INT_RX_REQ);
    }
}



int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPD1179)
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);
    #else
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO32, PIN_GPIO32_I2C_SCL);
    PIN_SetChannel(PIN_GPIO33, PIN_GPIO33_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO32, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO33, PIN_OUT_STRENGTH_20MA);
    #endif

    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Init I2C as Master */
    estatus = I2C_MasterInit(I2C, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC master initial FAIL] I2C clock is not fast enough to
support the speed\n");
        return 0;
    }
```

```
    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Master start to transmit and receive data */
    Master_TxRX_data(I2C);

    while(1)
    {
    }
}

void I2C_IRQHandler(void)
{
    int i;

    if (MASTER_TRANSMIT && (u32IsrCnt < T_BUFFER_SIZE))
    {
        /* Wait I2C Bus Idle */
        while(I2C_GetStatus(I2C, I2C_STS_ACTIVITY)) { }

        if (I2C_GetStatus(I2C, I2C_STS_TX_EMPTY))
        {
            for (i = 0; i < TxRx_DATA_LEN; i++)
            {
                I2C_WriteByte(I2C, gau8TxBuf[u32IsrCnt++]);

                /* Wait I2C ACK */
                if (u32DetectAck == 0)
                {
                    while(I2C_GetIntRawFlag(I2C, I2C_INT_ACK_DETECT) == 0) {}
                    u32DetectAck = 1;
                }
            }
        }
    }
    else if (!MASTER_TRANSMIT && (u32IsrCnt < T_BUFFER_SIZE))
    {
        /* Read RxFIFO, Read Data */
        while (I2C_GetStatus(I2C, I2C_STS_RX_NOT_EMPTY))
        {
            gau8RxBuf[u32IsrCnt++] = I2C_ReadByte(I2C);
        }

        for (i = 0; i < TxRx_DATA_LEN; i++)
        {
            I2C_MasterReadCmd(I2C);
        }
    }

    if (u32IsrCnt == T_BUFFER_SIZE)
    {
        if (MASTER_TRANSMIT)
        {
            I2C_DisableInt(I2C,I2C_INT_TX_REQ);
        }
        else
        {
            I2C_DisableInt(I2C,I2C_INT_RX_REQ);
            Check_Receive_Data();
```

```
        }

        printf("Success\n");
    }

    I2C_ClearInt(I2C, I2C_INT_ALL);
}
```

3.3.2 从机发送与接收

本示例中演示 I2C 作为从机使用中断传输模式进行发送数据和接收数据。当示例的 SLAVE_TRANSMIT 宏为 1 时进行发送数据，当 SLAVE_TRANSMIT 宏为 0 时进行接收数据，其配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 I2C 的 GPIO；
- 调用 I2C_SlaveInit()函数初始化 I2C 为 slave 设备，并初始化速度为 400K；
- 调用 I2C_Enable()函数进行使能 I2C 设备；
- 调用 I2C_SetTxFIFOThreshold()或者 I2C_SetRxFIFOThreshold()函数设置发送或者接收 FIFO 的阈值；
- 调用 I2C_EnableInt()函数使能发送与接收中断
- 在中断服务函数中根据发送与接收 FIFO 的状态进行发送或者接收数据；

中断模式(从机发送与接收)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define             DEBUG_INFO              1               /* the controlling flag of
printf info*/
#define             SLAVE_TRANSMIT          0               /* 1: slave transmit 0:
slave receive */
#define             RX_INT_TH               0xf             /* Rx thresgold as 16
entry */
#define             TX_INT_TH               0x0             /* Tx thresgold as 1 entry
*/
#define             TxRx_DATA_LEN           16              /* Send receive 16 Byte
*/
#define             T_BUFFER_SIZE           128
#define             I2C_SPEED               400000
#define             I2C_Slave_ADDR          0x9             /* IIC Slave ADDR */


uint32_t            gu32BuffSize            = T_BUFFER_SIZE;
uint8_t             gau8TxBuf[T_BUFFER_SIZE];
uint8_t             gau8RxBuf[T_BUFFER_SIZE];
uint32_t            num                     = 0;
uint32_t            u32IsrCnt               = 0;
```

```
ErrorStatus        estatus;


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

    /*To check the datum sent and recieved are the same*/
    for(i = 0; i < gu32BuffSize; i++)
    {
        if (gau8RxBuf[i] != u8Data)
        {
            printf("[Error]@%4d: TX(0x%02X) != RX(0x%02X)\n", __LINE__, u8Data,
gau8RxBuf[i] );
        }
        else
        {
            #if DEBUG_INFO
            printf("%3d: TX(0x%02X) == RX(0x%02X)\n", i, u8Data, gau8RxBuf[i] );
            #endif
        }

        u8Data++;
    }
}


void Slave_TxRX_data(I2C_REGS* I2Cx)
{
    int i;
    uint8_t u8Data = 0;

    if (SLAVE_TRANSMIT)
    {
        /* Generate random datum to transmit */
        for(i = 0; i < gu32BuffSize; i++)
        {
            gau8TxBuf[i] = u8Data++;
            printf("gau8TxBuf[%d] = 0x%x\n", i, gau8TxBuf[i]);
        }

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

        I2C_SetTxFIFOThreshold(I2C, TX_INT_TH);

        I2C_EnableInt(I2C, I2C_INT_TX_REQ);
    }
    else
    {
        printf("Slave Rx data...\n");

        I2C_SetRxFIFOThreshold(I2C, RX_INT_TH);

        I2C_EnableInt(I2C, I2C_INT_RX_REQ);
    }
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);
```

```
    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPD1179)
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_I2C_SCL);
    PIN_SetChannel(PIN_GPIO15, PIN_GPIO15_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO14, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO15, PIN_OUT_STRENGTH_20MA);
    #else
    /* Initial the I2C PIN */
    PIN_SetChannel(PIN_GPIO32, PIN_GPIO32_I2C_SCL);
    PIN_SetChannel(PIN_GPIO33, PIN_GPIO33_I2C_SDA);

    /*
    * Set Output Strength as 20mA, in case of many more
    * slaves are linking to the master.
    */
    PIN_SetOutStrength(PIN_GPIO32, PIN_OUT_STRENGTH_20MA);
    PIN_SetOutStrength(PIN_GPIO33, PIN_OUT_STRENGTH_20MA);
    #endif


    /* Disable All INT */
    I2C_DisableInt(I2C,I2C_INT_ALL);

    /* Clear All INT */
    I2C_ClearInt(I2C, I2C_INT_ALL);

    /* Initial I2C as Slave and set the speed as 400K */
    estatus = I2C_SlaveInit(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, I2C_SPEED);
    if(estatus == ERROR)
    {
        printf("[IIC slave initial FAIL] I2C clock is not fast enough to support
the speed\n");
        return 0;
    }

    /* Enable MCU INT Request */
    NVIC_EnableIRQ(I2C_IRQn);

    /* Enable I2C */
    I2C_Enable(I2C);

    /* Slave start to transmit and receive data */
    Slave_TxRX_data(I2C);

    while(1)
    {
    }
}
```

```
void I2C_IRQHandler(void)
{
    int i;

    if (SLAVE_TRANSMIT && (u32IsrCnt < T_BUFFER_SIZE))
    {
        if (I2C_GetStatus(I2C, I2C_STS_TX_EMPTY))
        {
            for (i = 0; i < TxRx_DATA_LEN; i++)
            {
                /* Wait Until Detect Read Request */
                while(!I2C_GetIntRawFlag(I2C, I2C_INT_READ_REQ)) { }

                /* Clear Event Flag */
                I2C_ClearInt(I2C, I2C_INT_READ_REQ);

                /* Write TxFIFO, Send Data */
                I2C_WriteByte(I2C, gau8TxBuf[u32IsrCnt++]);
            }
        }

    }
    else if (!SLAVE_TRANSMIT && (u32IsrCnt < T_BUFFER_SIZE))
    {
        while (I2C_GetStatus(I2C, I2C_STS_RX_NOT_EMPTY))
        {
            gau8RxBuf[u32IsrCnt++] = I2C_ReadByte(I2C);
        }
    }


    if (u32IsrCnt == T_BUFFER_SIZE)
    {
        if (SLAVE_TRANSMIT)
        {
            I2C_DisableInt(I2C,I2C_INT_TX_REQ);
        }
        else
        {
            I2C_DisableInt(I2C,I2C_INT_RX_REQ);
            Check_Receive_Data();
        }

        printf("Success\n");
    }

    I2C_ClearInt(I2C, I2C_INT_ALL);
}
```