Application Note

SPC11X8/SPD11X8 I2C 单元使用指南

2021 年 11 月 – 版本 2

目录

表格列表

图片列表

1 I2C 概述

I2C 总线是一种多主机的同步串行总线，常用于与多种外部设备进行通信，是一种应用广泛的通信方式。

SPC11X8/SPD11X8 的 I2C 单元有以下特点：

- 兼容 I2C 标准 2.1 版本
- 支持标准模式（100kbps）、快速模式（400kbps）和高速模式（3.4Mbps）
- 支持时钟同步
- 支持 7bits、10bits 寻址
- 支持主/从配置
- 两个独立的接收/发送 FIFO，深度 16。
- 支持批量传输（bulk transmit）模式

在 I2C 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 和表 1-2 中 I2Cx 表示 I2C 模块编号，取值为 I2C。

表 1-1: I2C 宏定义列表

宏名	功能及参数说明
I2C_Enable(I2Cx)	使能 I2Cx
I2C_Disable(I2Cx)	禁用 I2Cx
I2C_MasterReadCmd(I2Cx)	设置 I2Cx 读取指令（主机有用）
I2C_WriteByte(I2Cx, u8Data)	I2Cx 写入 TxFIFO 发送数据 u8Data: 要发送的数据
I2C_ReadByte(I2Cx)	I2Cx 读取 RxFIFO 接收数据
I2C_EnableRxUnderflowInt(I2Cx)	使能 I2Cx RxFIFO 向下溢出中断
I2C_DisableRxUnderflowInt(I2Cx)	禁用 I2Cx RxFIFO 向下溢出中断
I2C_EnableRxOverflowInt(I2Cx)	使能 I2Cx RxFIFO 向上溢出中断
I2C_DisableRxOverflowInt(I2Cx)	禁用 I2Cx RxFIFO 向上溢出中断
I2C_EnableRxDataAvailableInt(I2Cx)	使能 I2Cx RxFIFO 满中断
I2C_DisableRxDataAvailableInt(I2Cx)	禁用 I2Cx RxFIFO 满中断
I2C_EnableTxOverflowInt(I2Cx)	使能 I2Cx TxFIFO 向上溢出中断
I2C_DisableTxOverflowInt(I2Cx)	禁用 I2Cx TxFIFO 向上溢出中断
I2C_EnableTxDataRequestInt(I2Cx)	使能 I2Cx TxFIFO 空中断
I2C_DisableTxDataRequestInt(I2Cx)	禁用 I2Cx TxFIFO 空中断
I2C_EnableReadRequestInt(I2Cx)	使能 I2Cx 读取请求中断（从机有用）
I2C_DisableReadRequestInt(I2Cx)	禁用 I2Cx 读取请求中断（从机有用）
I2C_EnableTxAbortInt(I2Cx)	使能 I2Cx 发送终止中断（主机有用）
I2C_DisableTxAbortInt(I2Cx)	禁用 I2Cx 发送终止中断（主机有用）
I2C_EnableRxDoneInt(I2Cx)	使能 I2Cx 主机接收结束中断（从机有用）
I2C_DisableRxDoneInt(I2Cx)	禁用 I2Cx 主机接收结束中断（从机有用）
I2C_EnableActivityInt(I2Cx)	使能 I2Cx 活动中断

I2C_DisableActivityInt(I2Cx)	禁用 I2Cx 活动中断
I2C_EnableStopDetectInt(I2Cx)	使能 I2Cx 停止信号检测中断
I2C_DisableStopDetectInt(I2Cx)	禁用 I2Cx 停止信号检测中断
I2C_EnableStartDetectInt(I2Cx)	使能 I2Cx 开始信号检测中断
I2C_DisableStartDetectInt(I2Cx)	禁用 I2Cx 开始信号检测中断
I2C_EnableGeneralCallInt(I2Cx)	使能 I2Cx 广播中断
I2C_DisableGeneralCallInt(I2Cx)	禁用 I2Cx 广播中断
I2C_DisableAllInt(I2Cx)	禁用 I2Cx 所有中断
I2C_GetRxUnderflowIntFlag(I2Cx)	获取 I2Cx RxFIFO 向下溢出中断标志
I2C_GetRxOverflowIntFlag(I2Cx)	获取 I2Cx RxFIFO 向上溢出中断标志
I2C_GetRxDataAvailableIntFlag(I2Cx)	获取 I2Cx RxFIFO 满中断标志
I2C_GetTxOverflowIntFlag(I2Cx)	获取 I2Cx TxFIFO 向上溢出中断标志
I2C_GetTxDataRequestIntFlag(I2Cx)	获取 I2Cx TxFIFO 空中断标志
I2C_GetReadRequestIntFlag(I2Cx)	获取 I2Cx 读取请求中断标志（从机有用）
I2C_GetTxAbortIntFlag(I2Cx)	获取 I2Cx 发送终止中断标志（主机有用）
I2C_GetRxDoneIntFlag(I2Cx)	获取 I2Cx 主机接收结束中断标志（从机有用）
I2C_GetActivityIntFlag(I2Cx)	获取 I2Cx 活动中断标志
I2C_GetStopDetectIntFlag(I2Cx)	获取 I2Cx 停止信号检测中断标志
I2C_GetStartDetectIntFlag(I2Cx)	获取 I2Cx 开始信号检测中断标志
I2C_GetGeneralCallIntFlag(I2Cx)	获取 I2Cx 广播中断标志
I2C_GetRxUnderflowIntRawFlag(I2Cx)	获取 I2Cx RxFIFO 向下溢出中断原始标志
I2C_GetRxOverflowIntRawFlag(I2Cx)	获取 I2Cx RxFIFO 向上溢出中断原始标志
I2C_GetRxDataAvailableIntRawFlag(I2Cx)	获取 I2Cx RxFIFO 满中断原始标志
I2C_GetTxOverflowIntRawFlag(I2Cx)	获取 I2Cx TxFIFO 向上溢出中断原始标志
I2C_GetTxDataRequestIntRawFlag(I2Cx)	获取 I2Cx TxFIFO 空中断原始标志
I2C_GetReadRequestIntRawFlag(I2Cx)	获取 I2Cx 读取请求中断原始标志（从机有用）
I2C_GetTxAbortIntRawFlag(I2Cx)	获取 I2Cx 发送终止中断原始标志（主机有用）
I2C_GetRxDoneIntRawFlag(I2Cx)	获取 I2Cx 主机接收结束中断原始标志（从机有用）
I2C_GetActivityIntRawFlag(I2Cx)	获取 I2Cx 活动中断原始标志
I2C_GetStopDetectIntRawFlag(I2Cx)	获取 I2Cx 停止信号检测中断原始标志
I2C_GetStartDetectIntRawFlag(I2Cx)	获取 I2Cx 开始信号检测中断原始标志
I2C_GetGeneralCallIntRawFlag(I2Cx)	获取 I2Cx 广播中断原始标志
I2C_IsActivity(I2Cx)	获取 I2Cx 活动状态
I2C_IsTxNotFull(I2Cx)	获取 I2Cx TxFIFO 非满状态
I2C_IsTxEmpty(I2Cx)	获取 I2Cx TxFIFO 空状态
I2C_IsRxNotEmpty(I2Cx)	获取 I2Cx RxFIFO 非空状态
I2C_IsRxFull(I2Cx)	获取 I2Cx RxFIFO 满状态
I2C_IsMasterActivity(I2Cx)	获取 I2Cx 主机活动状态
I2C_IsSlaveActivity(I2Cx)	获取 I2Cx 从机活动状态
I2C_SetTxFIFOThreshold(I2Cx, u32TxLevel)	设置 I2Cx TxFIFO 触发深度 u32TxLevel: TxFIFO 深度小于等于该值，触发中断。
I2C_SetRxFIFOThreshold(I2Cx, u32RxLevel)	设置 I2Cx RxFIFO 触发深度 u32RxLevel: RxFIFO 深度大于等于该值，触发中断。

I2C_GetTxFIFOLevel(I2Cx)	获取 I2Cx TxFIFO 深度
I2C_GetRxFIFOLevel(I2Cx)	获取 I2Cx RxFIFO 深度
I2C_AckGeneralCall(I2Cx)	使能 I2Cx 响应广播
I2C_NotAckGeneralCall(I2Cx)	禁用 I2Cx 响应广播

表 1-2: I2C 函数列表

函数名	功能及参数说明
void I2C_SpeedInit(I2C_REGS* I2Cx, uint32_t u32Speed);	I2Cx 速度初始化。 1) u32Speed: 指定协议速率
void I2C_MasterInit(I2C_REGS* I2Cx, uint32_t u32Speed);	I2Cx 主机初始化。 1) u32Speed: 协议速率 标准模式: <= 100 kbps 快速模式: <= 400 kbps 高速模式: <= 3.4 Mbps
void I2C_SlaveInit(I2C_REGS* I2Cx, I2C_AddrModeEnum eAddrMode, uint16_t u16SlvAddr, uint32_t u32Speed);	I2Cx 从机初始化。 1) eAddrMode: 7 位地址或者 10 位地址 2) u16SlvAddr: 从机地址 3) u32Speed: 协议速率
void I2C_EnableGeneralCall(I2C_REGS* I2Cx);	使能 I2Cx 广播模式
void I2C_DisableGeneralCall(I2C_REGS* I2Cx);	禁用 I2Cx 广播模式
void I2C_EnableStartByte(I2C_REGS* I2Cx);	使能 I2Cx StartByte
void I2C_DisableStartByte(I2C_REGS* I2Cx);	禁用 I2Cx StartByte
void I2C_ClearInt(I2C_REGS* I2Cx, I2C_IntEnum eIntType);	清除中断。 1) eIntType: 中断类型
void I2C_MasterWrite(I2C_REGS* I2Cx, I2C_AddrModeEnum eAddrMode, uint_t u16TargetAddr, uint8_t *pu8WriteBuffer, uint32_t u32Count);	I2Cx 主机发送数据。 1) eAddrMode: 7 位地址或者 10 位地址 2) u16TargetAddr: 目标从机地址 3) pu8WriteBuffer: 发送数据缓冲区地址 4) u32Count: 发送数据数量
void I2C_SlaveRead(I2C_REGS* I2Cx, uint8_t *pu8ReadBuffer, uint32_t u32Count);	I2Cx 从机接收数据。 1) pu8ReadBuffer: 接收数据缓冲区地址 2) u32Count: 接收数据数量
void I2C_MasterRead(I2C_REGS* I2Cx,	I2Cx 主机读取数据。

I2C_AddrModeEnum eAddrMode, uint16_t u16TargetAddr, uint8_t *pu8ReadBuffer, uint32_t u32Count);	1) eAddrMode: 7 位地址或者 10 位地址 2) u16TargetAddr: 目标从机地址 3) u8ReadBuffer: 接收数据缓冲区地址 4) u32Count:接收数据数量
void I2C_SlaveWrite(I2C_REGS* I2Cx, uint8_t *pu8WriteBuffer, uint32_t u32Count);	I2Cx 从机发送数据。 1) pu8ReadBuffer：发送数据缓冲区地址 2) u32Count：发送数据数量
void I2C_MasterBulkWrite(I2C_REGS* I2Cx, I2C_AddrModeEnum eAddrMode, uint16_t u16TargetAddr, uint8_t *pu8WriteBuffer, uint32_t u32Count);	I2Cx 主机批量发送数据。 1) eAddrMode: 7 位地址或者 10 位地址 2) u16TargetAddr: 目标从机地址 3) pu8WriteBuffer: 发送数据缓冲区地址 4) u32Count: 发送数据数量
void I2C_SlaveBulkRead(I2C_REGS* I2Cx, uint8_t *pu8ReadBuffer, uint32_t u32Count);	I2Cx 从机批量接收数据。 1) pu8ReadBuffer: 接收数据缓冲区地址 2) u32Count: 接收数据数量
void I2C_MasterBulkRead(I2C_REGS* I2Cx, I2C_AddrModeEnum eAddrMode, uint16_t u16TargetAddr, uint8_t *pu8ReadBuffer, uint32_t u32Count);	I2Cx 主机批量读取数据。 1) eAddrMode: 7 位地址或者 10 位地址 2) u16TargetAddr: 目标从机地址 3) pu8ReadBuffer: 接收数据缓冲区地址 4) u32Count: 接收数据数量
void I2C_SlaveBulkWrite(I2C_REGS* I2Cx, uint8_t *pu8WriteBuffer, uint32_t u32Count);	I2Cx 从机批量发送数据。 1) pu8ReadBuffer: 发送数据缓冲区地址 2) u32Count: 发送数据数量

2 I2C 各种模式实例

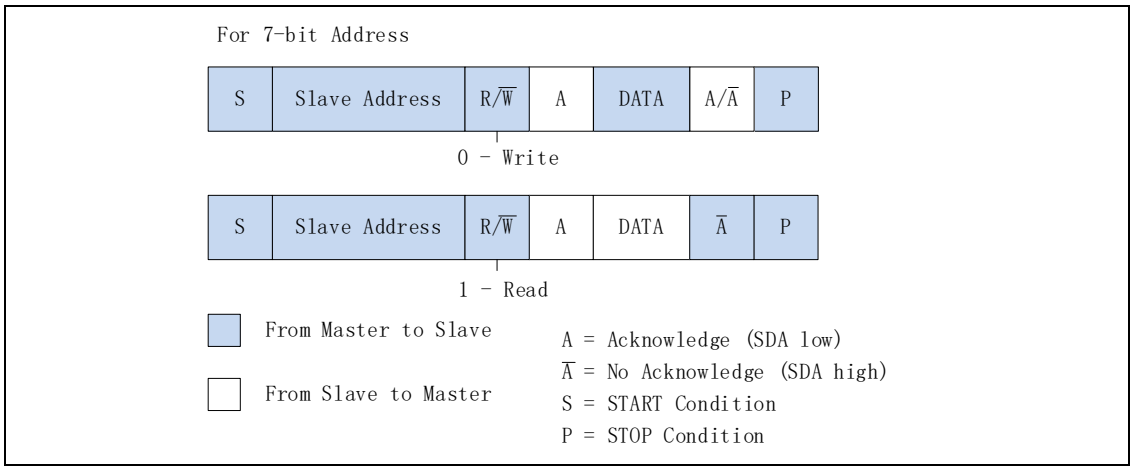
2.1 I2C 主机模式和从机模式收发实例

在这个例子里，会演示如何使用 I2C 的基本功能，收发数据。
该实例需要两颗 SPC11X8/SPD11X8 芯片，分别设置 I2C 为主机模式和从机模式，从机地址为 0x09（地址宽度 7 位），其余配置相同。数据帧配置如下：

- 波特率 400kbps
- 地址宽度 7bits
- 数据宽度 8 bits

如果涉及 I2C 的简单应用，用户可以依照这个例子进行设定。波形如下所示：

图 2-1: I2C 波形格式



主机和从机全局变量定义，代码如下：

Example Code

```
#define     T_BUFFER_SIZE     128
#define     I2C_Slave_ADDR    0x9         /* IIC Slve ADDR */
#define     Rx_Empty_INT_TH   0x0         /* Rx threshold as 12 entry */
#define     Tx_Full_INT_TH    0xF
#define     I2C_SPEED         400000

uint32_t    gu32BuffSize              = T_BUFFER_SIZE;
uint32_t    gu32_cnt_i2c_stop_isr;
uint8_t     gau8TxBuf[T_BUFFER_SIZE];
uint8_t     gau8RxBuf[T_BUFFER_SIZE];
uint8_t     gau8Buf[T_BUFFER_SIZE];
ErrorStatus estatus;
```

主机和从机都需要配置系统时钟 HCLK 为 200MHz，并且打开串口，代码如下：

Example Code

```
FLASH_WALLOW();
FLASH_SetTiming(200000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

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);
```

主机和从机中断函数代码如下：

Example Code

```
volatile uint32_t u32Dummy;

if( I2C_GetStopDetectIntRawFlag(I2C) )
{
    u32Dummy = I2C->I2CSTOPDETCLR.all;
    gu32_cnt_i2c_stop_isr ++ ;
}
else
{
    printf("[Error]@%4d: Stop detect interrupt error", __LINE__);
}
```

I2C 主机配置代码如下：

Example Code

```
int main()
{
    // Init I2C Pinmux
    GPIO_SetPinChannel(GPIO_38, GPIO38_BIT_MUXSEL_I2C_SDA);
    GPIO_SetPinChannel(GPIO_39, GPIO39_BIT_MUXSEL_I2C_SCL);

    // Set Output Strength
    GPIO_SetOutStrength(GPIO_38, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(GPIO_39, GPIO_OUT_STRENGTH_20MA);

    // FIFO threshold
    I2C_SetTxFIFOThreshold(I2C, 0);
    I2C_SetRxFIFOThreshold(I2C, 0);

    // Disable All Interrupt
    I2C_DisableAllInt(I2C);

    // Enable I2C STOP detect interrupt
    I2C_EnableStopDetectInt(I2C);

    // Clear All Interrupt
    I2C_ClearInt(I2C, I2C_INT_ALL);

    // Init I2C as Master
    estatus = I2C_MasterInit(I2C, 400000);
    if(estatus == ERROR)
    {
      printf("[IIC master init FAIL] I2C clock is not fast enough to
              support the speed\n");
      return 0;
    }

    // Enable CM4 Interrupt Request
    NVIC_EnableIRQ(I2C_IRQn);

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

    while(1)
    { }
}
```

I2C 从机配置代码如下：

Example Code

```
int main()
{
    // Init I2C Pinmux
    GPIO_SetPinChannel(GPIO_38, GPIO38_BIT_MUXSEL_I2C_SDA);
    GPIO_SetPinChannel(GPIO_39, GPIO39_BIT_MUXSEL_I2C_SCL);

    // Set Output Strength
    GPIO_SetOutStrength(GPIO_38, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(GPIO_39, GPIO_OUT_STRENGTH_20MA);

    // FIFO threshold
    I2C_SetTxFIFOThreshold(I2C, 0);
    I2C_SetRxFIFOThreshold(I2C, 0);

    // Disable All Interrupt
    I2C_DisableAllInt(I2C);

    // Enable I2C STOP detect interrupt
    I2C_EnableStopDetectInt(I2C);

    // Clear All Interrupt
    I2C_ClearInt(I2C, I2C_INT_ALL);

    // Init I2C as Slave
    estatus = I2C_SlaveInit(I2C, I2C_ADDR_7BIT, 0x09);
    if(estatus == ERROR)
    {
      printf("[IIC slave init FAIL] I2C clock is not fast enough to support
              the speed\n");
      return 0;
    }

    // Enable CM4 Interrupt Request
    NVIC_EnableIRQ(I2C_IRQn);

    /* Master start to transmit and I data */
    Slave_TxRX_data(I2C);

    while(1){}
}
```

I2C 主机收发实例代码如下：

Example Code

```
void Master_TxRX_data(void)
{
  int I;
  uint32_t u32IsrCnt = 0;                    /*Interrupt Check variable*/

  /* Generate random dtas to transmit */
  for(i=0; I < gu32BuffSize; i++)
    gau8TxBuf[i] = rand() & 0xFF;

  printf("Master Tx data...\n");
  I2C_MasterWrite(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8TxBuf,
                  gu32BuffSize);
  /*
   * In the interface of 'I2C_MasterWriteData()', master will wait for
  *the Tx FIFO empty after every byte, in other words, the master will
  * sent a 'STOP' CMD to the IIC, and then sent a 'RESTART' at the next
  * byte. We can count the bytes we has sent to get the INT count had
  * entered.
   */
  u32IsrCnt += gu32BuffSize;

  printf("Master Rx data...\n");
  /* Read the data had sent to the slave back and put them in 'gau8RxBuf'
*/
  I2C_MasterRead(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  /* To check the datum sent and I are the same */
  for(i=0; I < gu32BuffSize; i++)
  {
    if( gau8TxBuf[i] != gau8RxBuf[i])
    {
      printf("[Error]@%4d: TX(%p,0x%02X) != RX(%p,0x%02X)\n",
               __LINE__,&gau8TxBuf[i], gau8TxBuf[i], &gau8RxBuf[i],
               gau8RxBuf[i] );
    }
    else
    {
#if DEBUG_INFO
      printf("%3d: TX(%p,0x%02X) == RX(%p,0x%02X)\n", I,
           &gau8TxBuf[i], gau8TxBuf[i], &gau8RxBuf[i], gau8RxBuf[i] );
```

```
#endif
    }
  }

  /* the aim of fllowing code is to check if the slave Tx/Rx is ok */
  printf("Master Rx data...\n");
  I2C_MasterRead(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
                 gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  printf("Master Tx data...\n");
  /* Sent the data had I back to the slave */
  I2C_MasterWrite(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
                  gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  /*
   * 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);
}
```

I2C 从机收发实例代码如下：

Example Code

```
void Slave_TxRX_data(I2C_REGS* I2Cx)
{
  int I;
  uint32_t u32IsrCnt = 0;                /* Interrupt Check variable */

  /* The aim of fllowing code is to check if the Master Tx/Rx is ok */
  printf("Slave Rx data...\n");
  I2C_SlaveRead(I2C, gau8RxBuf, gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  printf("Slave Tx data...\n");
  /* Sent the data had I back to the master */
  I2C_SlaveWrite(I2C, gau8RxBuf, gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  /* Generate random dtas to transmit */
  for(i=0; I < gu32BuffSize; i++)
      gau8TxBuf[i] = rand() & 0xFF;

  /* The aim of fllowing code is to check if the Tx/Rx of slave itself
is ok */
  printf("Slave Tx data...\n");
  I2C_SlaveWrite(I2C, gau8TxBuf, gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  printf("Slave Rx data...\n");
  /* Read the data had sent to the master just now back and put them into
the 'gau8RxBuf' */
  I2C_SlaveRead(I2C, gau8RxBuf, gu32BuffSize);
  u32IsrCnt += gu32BuffSize;

  /* Wait until I2C idle */
  while( I2C_IsActivity(I2Cx) ) { }

  /* To check the datum sent and I are the same */
  for(i=0; i<gu32BuffSize; i++)
  {
      if( gau8TxBuf[i] != gau8RxBuf[i])
          printf("[Error]@%4d: %3d: TX(%p,0x%02X) != RX(%p,0x%02X)\n",
                  __LINE__, I, &gau8TxBuf[i], gau8TxBuf[i],
                  &gau8RxBuf[i], gau8RxBuf[i] );
      else
```

```
  #if DEBUG_INFO
        printf("%3d: TX(%p,0x%02X) == RX(%p,0x%02X)\n", I,
          &gau8TxBuf[i], gau8TxBuf[i], &gau8RxBuf[i], gau8RxBuf[i] );
  #endif
  }
  /*
   * 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);
}
```

程序下载后，主机和从机同时运行:

- 串口波特率设定为 38400bps，可以看到发送数据和接收数据一致，则代码运行正常。
- 通过示波器观察波形与图一致。

2.2 I2C 主机模式和从机模式批量收发（bulk transmit）实例

I2C 批量传输模式是发送地址后连续发送多个数据的一种传输方式。该模式需要在数据传输阶段 TxFIFO 非空。

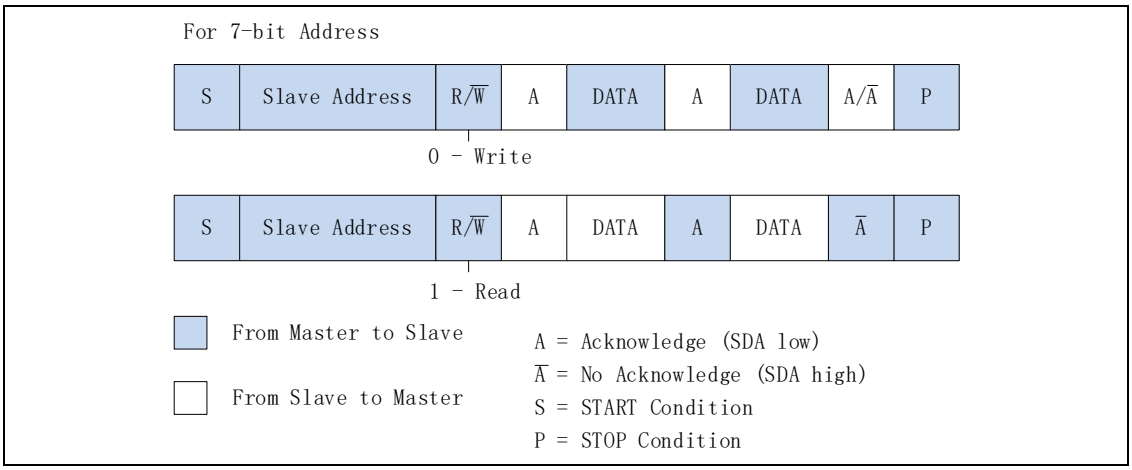
在这个例子里，会演示 I2C 使用批量模式收发数据。

该实例需要两颗 SPC11X8/SPD11X8 芯片，分别设置 I2C 为主机模式和从机模式，从机地址为 0x09（地址宽度 7 位），其余配置相同。数据帧配置如下：

- 波特率 400kbps
- 地址宽度 7bits
- 数据宽度 8 bits

波形如下所示：

图 2-2: I2C 批量传输波形格式



主机和从机全局变量定义，代码如下：

Example Code

```
#define  DEBUG_INFO      1    /* the contolling flag of printf info*/
#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;
```

主机和从机都需要配置系统时钟 HCLK 为 200MHz，并且打开串口，代码如下：

Example Code

```
FLASH_WALLOW();
FLASH_SetTiming(200000000);
/* Disable flash write access after flash operation had done */
FLASH_WDIS();

CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

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);
```

主机和从机中断函数代码如下：

Example Code

```
volatile uint32_t u32Dummy;

if( I2C_GetStopDetectIntRawFlag(I2C) )
{
    u32Dummy = I2C->I2CSTOPDETCLR.all;
    gu32_cnt_i2c_stop_isr ++ ;
}
else
{
    printf("[Error]@%4d: Stop detect interrupt error", __LINE__);
}
```

I2C 主机配置代码如下：

Example Code

```
int main()
{
    // Init I2C Pinmux
    GPIO_SetPinChannel(GPIO_38, GPIO38_BIT_MUXSEL_I2C_SDA);
    GPIO_SetPinChannel(GPIO_39, GPIO39_BIT_MUXSEL_I2C_SCL);

    // Set Output Strength
    GPIO_SetOutStrength(GPIO_38, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(GPIO_39, GPIO_OUT_STRENGTH_20MA);

    // FIFO threshold
    I2C_SetTxFIFOThreshold(I2C, 0);
    I2C_SetRxFIFOThreshold(I2C, 0);

    // Disable All Interrupt
    I2C_DisableAllInt(I2C);

    // Enable I2C STOP detect interrupt
    I2C_EnableStopDetectInt(I2C);

    // Clear All Interrupt
    I2C_ClearInt(I2C, I2C_INT_ALL);

    // Init I2C as Master
    estatus = I2C_MasterInit(I2C, 400000);
    if(estatus == ERROR)
    {
      printf("[IIC master init FAIL] I2C clock is not fast enough to
support the speed\n");
      return 0;
    }

    // Enable CM4 Interrupt Request
    NVIC_EnableIRQ(I2C_IRQn);
    /* Master start to transmit and I data */
    Master_Bulk_TxRX_data();

    while(1) {}
}
```

I2C 从机配置代码如下：

Example Code

```
int main()
{
    // Init I2C Pinmux
    GPIO_SetPinChannel(GPIO_38, GPIO38_BIT_MUXSEL_I2C_SDA);
    GPIO_SetPinChannel(GPIO_39, GPIO39_BIT_MUXSEL_I2C_SCL);

    // Set Output Strength
    GPIO_SetOutStrength(GPIO_38, GPIO_OUT_STRENGTH_20MA);
    GPIO_SetOutStrength(GPIO_39, GPIO_OUT_STRENGTH_20MA);

    // FIFO threshold
    I2C_SetTxFIFOThreshold(I2C, 0);
    I2C_SetRxFIFOThreshold(I2C, 0);

    // Disable All Interrupt
    I2C_DisableAllInt(I2C);

    // Enable I2C STOP detect interrupt
    I2C_EnableStopDetectInt(I2C);

    // Clear All Interrupt
    I2C_ClearInt(I2C, I2C_INT_ALL);

    // Init I2C as Slave
    estatus = I2C_SlaveInit(I2C, I2C_ADDR_7BIT, 0x09);
    if(estatus == ERROR)
    {
      printf("[IIC slave init FAIL] I2C clock is not fast enough to support
the speed\n");
      return 0;
    }

    // Enable CM4 Interrupt Request
    NVIC_EnableIRQ(I2C_IRQn);
    /* Master start to transmit and receive data */
    Slave_TxRX_data(I2C);

    while(1) {}
}
```

主机收发实例代码如下：

Example Code

```
void Master_Bulk_TxRX_data(void)
{
  int I;
  uint32_t u32IsrCnt = 0;                /* Interrupt Check variable */

  /* Generate random datum to transmit */
  for(i=0; I < gu32BuffSize; i++)
    gau8TxBuf[i] = rand() & 0xFF;

  printf("Master Tx data...\n");
  I2C_MasterBulkWrite(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8TxBuf,
                      gu32BuffSize);

  /*
   * 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;

  printf("Master Rx data...\n");
  /*Read the data had sent to the slave back and put them in 'gau8RxBuf'*/
  I2C_MasterBulkRead(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
                     gu32BuffSize);
  u32IsrCnt += 1;

  /*To check the datum sent and I are the same*/
  for(i=0; I < gu32BuffSize; i++)
  {
    if( gau8TxBuf[i] != gau8RxBuf[i])
    {
      printf("[Error]@%4d: TX(%p,0x%02X) != RX(%p,0x%02X)\n",
        __LINE__,&gau8TxBuf[i], gau8TxBuf[i],
       &gau8RxBuf[i], gau8RxBuf[i] );
    }
    else
    {
  #if DEBUG_INFO
      printf("%3d: TX(%p,0x%02X) == RX(%p,0x%02X)\n", I,
          &gau8TxBuf[i], gau8TxBuf[i], &gau8RxBuf[i], gau8RxBuf[i] );
```

```
  #endif
    }
  }

  /*The aim of fllowing code is to check if the slave Tx/Rx is ok*/
  printf("Master Rx data...\n");
  I2C_MasterBulkRead(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
                     gu32BuffSize);
  u32IsrCnt += 1;

  printf("Master Tx data...\n");
  /*Sent the data had I back to the slave*/
  I2C_MasterBulkWrite(I2C, I2C_ADDR_7BIT, I2C_Slave_ADDR, gau8RxBuf,
                      gu32BuffSize);
  u32IsrCnt += 1;

  /*
   * 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);
}
```

I2C 从机收发实例代码如下：

Example Code

```
void Slave_TxRX_data(I2C_REGS* I2Cx)
{
  int I;
  uint32_t u32IsrCnt = 0;                /*Interrupt Check variable*/

  /* The aim of fllowing code is to check if the Master Tx/Rx is ok */
  printf("Slave Rx data...\n");
  I2C_SlaveBulkRead(I2C, gau8RxBuf, gu32BuffSize);
  /*
   * The master will sent a 'STOP' CMD to the IIC, and then sent a 'RESTART'
    * at thenext bulk. We can count the bulk we has sent to get the INT
    * count had entered.
    */
  u32IsrCnt += 1;

  printf("Slave Tx data...\n");
  /* Sent the datum had I back to the master */
  I2C_SlaveBulkWrite(I2C, gau8RxBuf, gu32BuffSize);
  u32IsrCnt += 1;

  /* Generate random dtas to transmit */
  for(i=0; I < gu32BuffSize; i++)
      gau8TxBuf[i] = rand() & 0xFF;

  /* The aim of fllowing code is to check if the Tx/Rx of slave itself
is ok */
  printf("Slave Tx data...\n");
  I2C_SlaveBulkWrite(I2C, gau8TxBuf, gu32BuffSize);
  u32IsrCnt += 1;

  printf("Slave Rx data...\n");
  /* Read the datum had sent to the master just now back and put them
into the 'gau8RxBuf' */
  I2C_SlaveBulkRead(I2C, gau8RxBuf, gu32BuffSize);
  u32IsrCnt += 1;

  /* Wait until I2C idle */
  while( I2C_IsActivity(I2Cx) ) { }

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

```
        if( gau8TxBuf[i] != gau8RxBuf[i])
            printf("[Error]@%4d: %3d: TX(%p,0x%02X) != RX(%p,0x%02X)\n",
                    __LINE__, I, &gau8TxBuf[i], gau8TxBuf[i],
                    &gau8RxBuf[i], gau8RxBuf[i] );
        else
    #if DEBUG_INFO
            printf("%3d: TX(%p,0x%02X) == RX(%p,0x%02X)\n", I,
                    &gau8TxBuf[i], gau8TxBuf[i], &gau8RxBuf[i],
                     gau8RxBuf[i] );
    #endif
    }
    /*
     * 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);

}
```

程序下载后，俩颗芯片同时运行:

- 串口波特率设定为 38400bps，可以看到发送数据和接收数据一致，则代码运行正常。
- 通过示波器观察波形与图一致。

3 修订记录

表 3-1: 文档修订记录

日期	版本	修改内容
2019-07-25	1	初始版本
2021-11-13	2	1. 更新表 1-2。