M031 SPI从机最快速率传输

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 M031 来实现 SPI 从机以最快速率传输数据
BSP 版本	M031_Series_BSP_CMSIS_V3.05.000
开发平台	NuMaker-M031SE V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例代码主要演示 M031 作为 SPI 从机实现以最快速度接收和发送数据。

范例里有两个工程。SPI_Master 是 SPI 主机程序。它设置 SPI 时钟为 16MHZ，向从机发送 256 字节数据，然后再读回数据，与发送的数据进行对比，判断从机收发数据是否正确。SPI_Slave 是 SPI 从机程序。它收到主机的“写”命令后，接收、保存并打印主机发送的 256 字节数据；收到主机的“读”命令后将 256 字节数据回传给主机。本文介绍了 SPI 速率和角色的设置、增加延时间隔防止 RX FIFO 溢出、通讯前先写入 TX FIFO 和增加 dummy 防止 TX 下溢。

1.1 原理

SPI 从机以最高速率通讯时，由于 MCU 时钟频率限制， MCU 读写 SPI 的速度比 SPI 传输速度慢，无法实现同时收发数据，因此采用半双工通讯。如图 1-1 所示数据格式为：

命令（1 字节）+ 地址（1 字节）+ dummy（5 字节）+ 数据（256 字节）

命令决定从机是接收还是发送数据；地址决定数据保存的位置；dummy 用于预留从机解析命令和准备数据的时间。

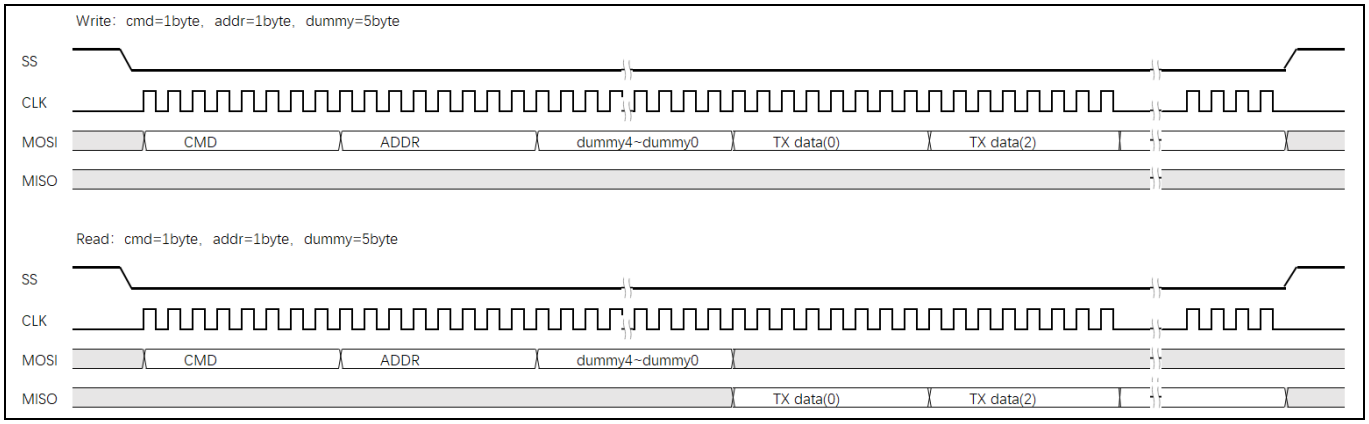


图 1-1 SPI 收发时序

如果主机发送完命令和地址后立即传输数据，从机还未完成命令解析。从机无法及时读取 RX FIFO，会发生 RX FIFO 溢出，导致从机接收丢数据；从机无法及时填写 TX FIFO，会发生 TX 下溢中断，导致从机无法发送数据。因此有必要在地址和数据之间增加 dummy。

从机接收时必须及时读走 RX FIFO，防止发生 RX FIFO 溢出。因此读取速度必须比传输速度快。当 SPI 时钟固定时，插入延时间隔，可以增加两个字节传输之间的时间间隔。如图 1-2 所示 MCU 读取一个字节需要 640 ns，而 SPI 传输一个字节只需要 500 ns，至少需要再增加 2.5 个 SPI CLK（156 ns）的延时间隔，才能保证读取速度比传输速度快。

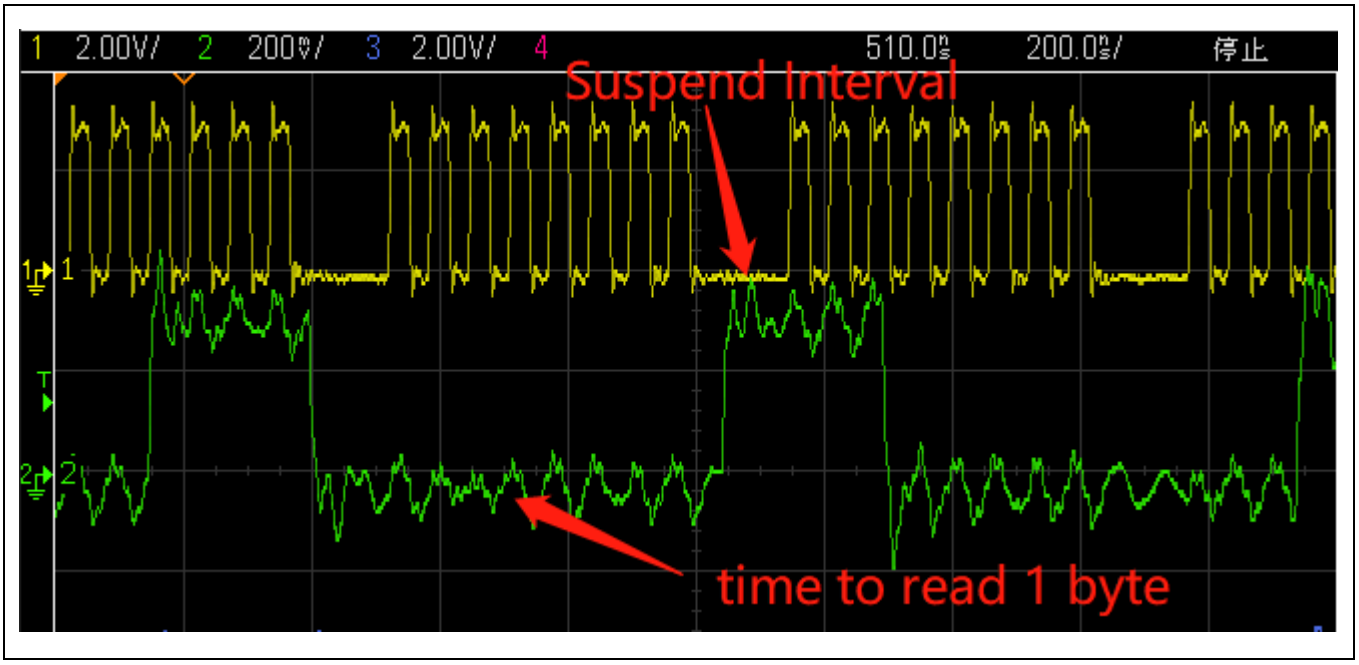


图 1-2 SPI 传输 1 字节的时间和 MCU 读取 1 字节的时间

从机发送时必须及时写入 TX FIFO，防止发生 TX 下溢中断。因此需要 dummy 传完之前已经完成了命令解析并且往 TX FIFO 里写入过数据。如图 1-3 所示主机发送第一个 CLK 到从机往 TX FIFO 里写入第一个数据需要 2.5 us，传输一个字节需要 500ns。因此要保证 dummy 传完前写入第一个数据到 TX FIFO，传输命令+地址+dummy 的时间最小为 3 us（6 个字节），本范例设置的 3.5 us（7 个字节）,如图 1-1 所示命令和地址分别是 1 个字节，那么 dummy 就需要 5 个字节。

注意：命令和地址的长度影响接收和解析的时间，更改长度后需要重新测量时间。

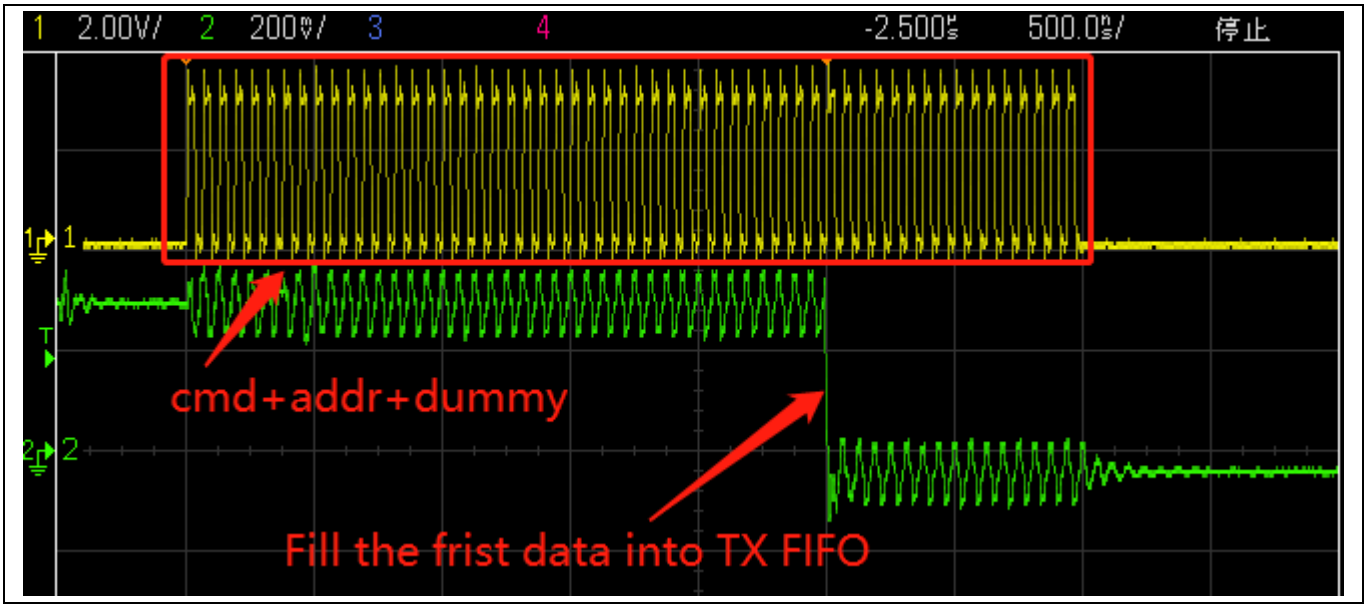


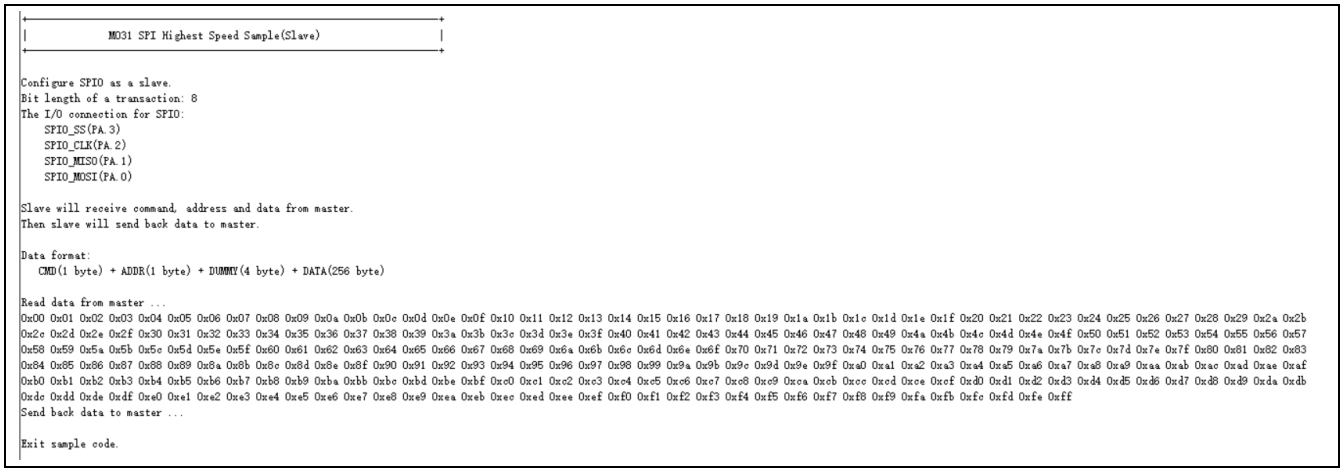
图 1-3 从传输启动到从机往 TX FIFO 里写入第一个数据的时间间隔

1.2 执行结果

如果将调试串口都连接到电脑上，程序执行时串口工具里将显示执行结果如图 1-4 和图 1-5 所示。

```
+------------------------------------------------------------------------+
|                  M031 SPI Highest Speed Sample(Master)                 |
+------------------------------------------------------------------------+

Configure SPI0 as a master.
Bit length of a transaction: 8
The I/O connection for SPI0:
    SPI0_SS(PA.3)
    SPI0_CLK(PA.2)
    SPI0_MISO(PA.1)
    SPI0_MOSI(PA.0)

Master will send command, address and data to slave.
Then master will receive data sent back from slave.

Data format:
   CMD(1 byte) + ADDR(1 byte) + DUMMY(4 byte) + DATA(256 byte)

The SPI master configuration is ready.
Press any key to continue.

Write data to Slave...
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b
0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82 0x83
0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb
0xdc 0xdd 0xde 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7 0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff [OK]
Read data& Compare ...
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b
0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82 0x83
0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb
0xdc 0xdd 0xde 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7 0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff [OK]


Exit Sample code.
```

图 1-4 SPI 主机的执行结果



```
+-----------------------------------------------------------------+
|              M031 SPI Highest Speed Sample(Slave)                |
+-----------------------------------------------------------------+

Configure SPI0 as a slave.
Bit length of a transaction: 8
The I/O connection for SPI0:
    SPI0_SS(PA.3)
    SPI0_CLK(PA.2)
    SPI0_MISO(PA.1)
    SPI0_MOSI(PA.0)

Slave will receive command, address and data from master.
Then slave will send back data to master.

Data format:
   CMD(1 byte) + ADDR(1 byte) + DUMMY(4 byte) + DATA(256 byte)

Read data from master ...
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b
0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57
0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82 0x83
0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb
0xdc 0xdd 0xde 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7 0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff
Send back data to master ...

Exit sample code.
```

图 1-5 SPI 从机的执行结果

2. 代码介绍

- SPI 主机，代码位于 *SPI_Master/main.c*

定义 SPI 参数:

```
#define SPI_PORT         SPI0
#define SPI_CLK_FREQ     16000000   /* SPI clock: HZ */
#define DATA_WIDTH       8          /* data width: bit */
#define FIFO_LEVEL       8          /* FIFO level */
#define DATA_LENGTH      256        /* data length */
```

定义 SPI 通讯协议:

当数据宽度为 8 位时，TX 和 RX FIFO 深度为 8，命令 + 地址 + dummy 的总长度不能超过 8 字节。

```
/*Note: when the width is 8 bits, TX FIFO and Rx FIFO are 8 levels, so the total length of
  CMD, address, dummy cannot exceed 8 bytes */
#define CMD_LEN          1  /* Command length : byte */
#define ADDR_LEN         1  /* Address length: byte */
/* Dummy is added to give slave more time to fill TX FIFO. Avoid generating TX underflow
   interrupt. The length of dummy is affected by SPI speed and address length */
#define DUMMY_LEN        4  /* Dummy length: byte */
/* Command opcode */
#define CMD_WRITE        2 /* Master write */
#define CMD_READ         3 /* Master read */
```

SPI 初始化：

主机需要设置 SPI 时钟和 AutoSS。

```
void SPI_Init(void)
{
    /* Configure SPI_PORT as a master, MSB first, 8-bit transaction, SPI Mode-0 timing,
       clock is 16MHz */
    SPI_Open(SPI_PORT, SPI_MASTER, SPI_MODE_0, DATA_WIDTH, SPI_CLK_FREQ);

    /* Disable auto SS function, control SS signal manually. */
    SPI_DisableAutoSS(SPI_PORT);
}
```

SPI 主机写:

由于从机来不及读取数据，传输数据之前需要设置延时间隔。

```
void SpiMaster_Write(uint32_t StartAddress, uint8_t *u8DataBuffer, uint32_t len)
{
    uint32_t i;
```

```
    /* /CS: active */
    SPI_SET_SS_LOW(SPI_PORT);

    /* Send Command: 0x02, Page program */
    SPI_WRITE_TX(SPI_PORT, CMD_WRITE);

    /* Send 8-bit start address */
    SPI_WRITE_TX(SPI_PORT, StartAddress & 0xFF);

    /* Send DUMMY_LEN*8 '0', dummy */
    for (i = 0; i < DUMMY_LEN; i++)
        SPI_WRITE_TX(SPI_PORT, 0x00);

    while (SPI_IS_BUSY(SPI_PORT));

    /* Write data */
    /* Set suspend interval. To prevent slave's RX FIFO overflow caused by slave too late
       to receive data */
    SPI_SET_SUSPEND_CYCLE(SPI_PORT, 0x02);

    i = 0;

    while (i < len)
    {

        if (!SPI_GET_TX_FIFO_FULL_FLAG(SPI_PORT))
        {
            SPI_WRITE_TX(SPI_PORT, u8DataBuffer[i++]);
        }
    }

    /* Wait tx finish */
    while (SPI_IS_BUSY(SPI_PORT));

    /* /CS: de-active */
    SPI_SET_SS_HIGH(SPI_PORT);

    /* Clear suspend interval */
    SPI_SET_SUSPEND_CYCLE(SPI_PORT, 0x00);
    SPI_ClearRxFIFO(SPI_PORT);
}
```

SPI 主机读：

为了提高读取数据的速度，采取连续发送 8 字节'0'，再一次性读取 8 字节数据的方式。

```
void SpiMaster_Read(uint32_t StartAddress, uint8_t *u8DataBuffer, uint32_t len)
{
    uint32_t i = 0, len_by_level;

    /* /CS: active */
    SPI_SET_SS_LOW(SPI_PORT);

    /* Send Command: 0x03, Read data */
    SPI_WRITE_TX(SPI_PORT, CMD_READ);
```

```
/* Send 8-bit start address */
 SPI_WRITE_TX(SPI_PORT, StartAddress  & 0xFF);

 /* Send DUMMY_LEN*8 '0', dummy */
 for (i = 0; i < DUMMY_LEN; i++)
     SPI_WRITE_TX(SPI_PORT, 0x00);

 while (SPI_IS_BUSY(SPI_PORT));

 /* Slear RX buffer */
 SPI_ClearRxFIFO(SPI_PORT);

 /* Read data */
 i = 0;
 len_by_level = len / FIFO_LEVEL * FIFO_LEVEL;

 /* Data length greater than 8 */
 while (i < len_by_level)
 {
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);
     SPI_WRITE_TX(SPI_PORT, 0x00);

     while (SPI_IS_BUSY(SPI_PORT));

     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
 }

 /* The data length cannot be divided by 8 */
 while (i < len)
 {
     SPI_WRITE_TX(SPI_PORT, 0x00);

     while (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT));

     u8DataBuffer[i++] = SPI_READ_RX(SPI_PORT);
 }

 /* Wait tx finish */
 while (SPI_IS_BUSY(SPI_PORT));

 /* /CS: de-active */
 SPI_SET_SS_HIGH(SPI_PORT);
```

```
}
```

- SPI 从机，代码位于 *SPI_Slave/main.c*

SPI 初始化：

从机不需要设置时钟和 SS。

```
void SPI_Init(void)
{
    /* Configure SPI_PORT as a slave, MSB first, 8-bit transaction, SPI Mode-0 timing*/
    SPI_Open(SPI_PORT, SPI_SLAVE, SPI_MODE_0, DATA_WIDTH, (uint32_t) NULL);
}
```

SPI 从机接收:

```
uint32_t SpiSlave_Read(uint32_t *StartAddress, uint8_t *u8DataBuffer)
{
    uint32_t len = 0;
    uint8_t Cmd = 0, cnt = 0;

    *StartAddress = 0;

    /*Clear RX FIFO */
    SPI_ClearRxFIFO(SPI_PORT);

    /* Read Command: 0x02, write to slave */
    while (1)
    {
        /* Check RX EMPTY flag */
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
            Cmd = SPI_READ_RX(SPI_PORT); /* Read RX FIFO */

        if (Cmd == CMD_WRITE) break;
    }

    /* Read 8-bit start address */
    do
    {
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
        {
            *StartAddress += (*StartAddress << 8) + SPI_READ_RX(SPI_PORT);
            cnt++;
        }
    } while (cnt < ADDR_LEN);

    u8DataBuffer += *StartAddress;

    /* Read DUMMY_LEN*8 bit '0', dummy */
    cnt = 0;

    do
    {
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
```

```
        {
            SPI_READ_RX(SPI_PORT);
            cnt++;
        }
    } while (cnt < DUMMY_LEN);

    /* Read data */
    do
    {
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
        {
            u8DataBuffer[len++] = SPI_READ_RX(SPI_PORT);
        }
    } while (len < DATA_LENGTH);

    return len;
}
```

SPI 从机发送：

进入函数后需要先将 TX FIFO 写入'0'，长度等于命令 + 地址 + dummy 的总和。否则一旦片选信号有效，TX FIFO 为空，会发生 TX 下溢，无法发送数据。

读完地址后利用主机发送 dummy 的时间，解析命令并将需要发送的数据写入 TX FIFO 里。

```
uint32_t SpiSlave_Write(uint32_t *StartAddress, uint8_t *u8DataBuffer)
{
    uint32_t len = 0;
    uint8_t Cmd = 0, cnt = 0;

    *StartAddress = 0;

    /* Clear RX FIFO, TX FIFO */
    SPI_ClearRxFIFO(SPI_PORT);
    SPI_ClearTxFIFO(SPI_PORT);

    /* Write to TX FIFO ADDR_LEN + DUMMY_LEN + 1 byte */
    for (cnt = 0; cnt < (ADDR_LEN + DUMMY_LEN + 1); cnt++)
        SPI_WRITE_TX(SPI_PORT, 0x00);

    cnt = 0;

    /* Read Command: 0x03, read data from slave */
    do
    {
        /* Check RX EMPTY flag */
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
        {
            Cmd = SPI_READ_RX(SPI_PORT); /* Read RX FIFO */
        }
    } while (Cmd != CMD_READ);

    /* Read 8-bit start address */
    do
```

```
    {
        if (SPI_GET_RX_FIFO_EMPTY_FLAG(SPI_PORT) == 0)
        {
            *StartAddress += (*StartAddress << 8) + SPI_READ_RX(SPI_PORT);
            cnt++;
        }
    } while (cnt < ADDR_LEN);

    u8DataBuffer += *StartAddress;

    /* Write data */
    do
    {
        if (SPI_GET_TX_FIFO_FULL_FLAG(SPI_PORT) == 0)
        {
            SPI_WRITE_TX(SPI_PORT, u8DataBuffer[len++]); /* Write to TX FIFO */
        }
    } while (len < DATA_LENGTH);

    /* Wait for the end */
    while ((SPI_GET_TX_FIFO_EMPTY_FLAG(SPI_PORT) == 0) || (SPI_IS_BUSY(SPI_PORT)));

    return len;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M031_Series_BSP_CMSIS_V3.05.000
- IDE 版本
  - Keil uVersion 5.26

3.2 硬件需求

- 电路组件
  - NuMaker-M031SE V1.1
- 示意图
  - 将两块开发板的 SPI0_MOSI (PA.0)、 SPI0_MISO (PA.1)、SPI0_CLK (PA.2)、SPI0_SS (PA.3)、GND 分别连接在一起。
  - 两块开发板的 UART0 都连接到电脑上。以控制范例代码的执行和显示执行结果。

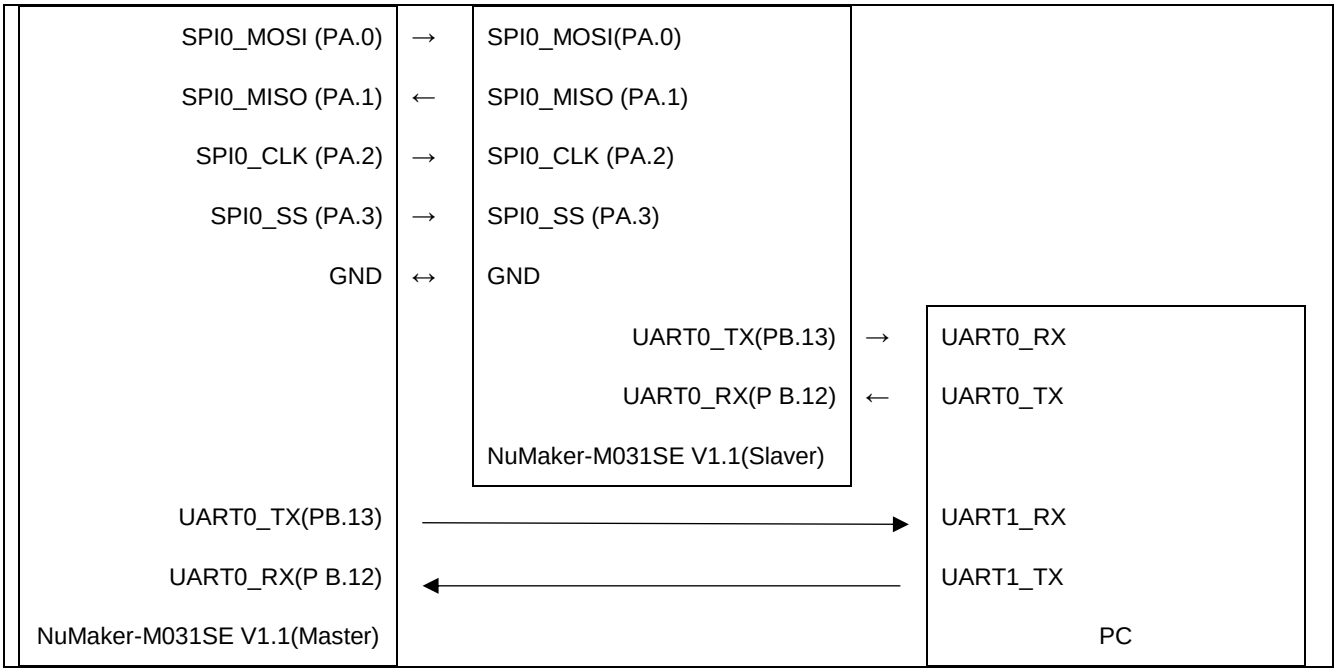


图 3-1 引脚连接

4. 目录信息

EC_M031_Slave_Fastest_Speed_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径，双击 *M031_SPI_Slave_Fastest_Speed.uvmpw*。
2. 进入 SPI_Master 编译模式界面
   - 设置 SPI_Master 为 Active 工程
   - 编译
   - 下载代码至内存
3. 进入 SPI_Slave 编译模式界面
   - 设置 SPI_Master 为 Active 工程
   - 编译
   - 下载代码至内存
4. 打印主机调试信息
   - PC 上打开串口调试工具
   - 波特率：115200bps
5. 打印从机调试信息
   - PC 上打开串口调试工具
   - 波特率：115200bps
6. 运行
   - 从机按复位键，执行代码
   - 主机按复位键，执行代码

6. 修订纪录

Date	Revision	Description
2022.08.10	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.