M031 SPI從機最快速率傳輸

NuMicro®32位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例代碼使用 M031 來實現 SPI 從機以最快速率傳輸資料
BSP 版本	M031_Series_BSP_CMSIS_V3.05.000
開發平臺	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

本範例代碼主要演示 M031 作為 SPI 從機實現以最快速度接收和發送資料。

範例裡有兩個工程。SPI_Master 是 SPI 主機程式。它設置 SPI 時鐘為 16MHZ，向從機發送 256 位元組資料，然後再讀回資料，與發送的資料進行對比，判斷從機收發資料的正確性。SPI_Slave 是 SPI 從機程式。它收到主機的“寫”命令後，接收、保存並列印主機發送的 256 位元組資料；收到主機的“讀”命令後將 256 位元組資料回傳給主機。本文介紹了 SPI 速率和角色的設置、增加延遲間隔防止 RX FIFO 溢出、通訊前先寫入 TX FIFO 和增加 dummy 防止 TX 下溢。

1.1 原理

SPI 從機以最高速率通訊時，由於 MCU 時鐘頻率限制， MCU 讀寫 SPI 的速度比 SPI 傳送速率慢，無法實現同時收發資料，因此採用半雙工通訊。如圖 1-1 所示資料格式為：

命令（1 位元組）+位址（1 位元組）+ dummy（5 位元組）+資料（256 位元組）

命令決定從機是接收還是發送資料；位址決定資料保存的位置；dummy 用於預留從機解析命令和準備資料的時間。

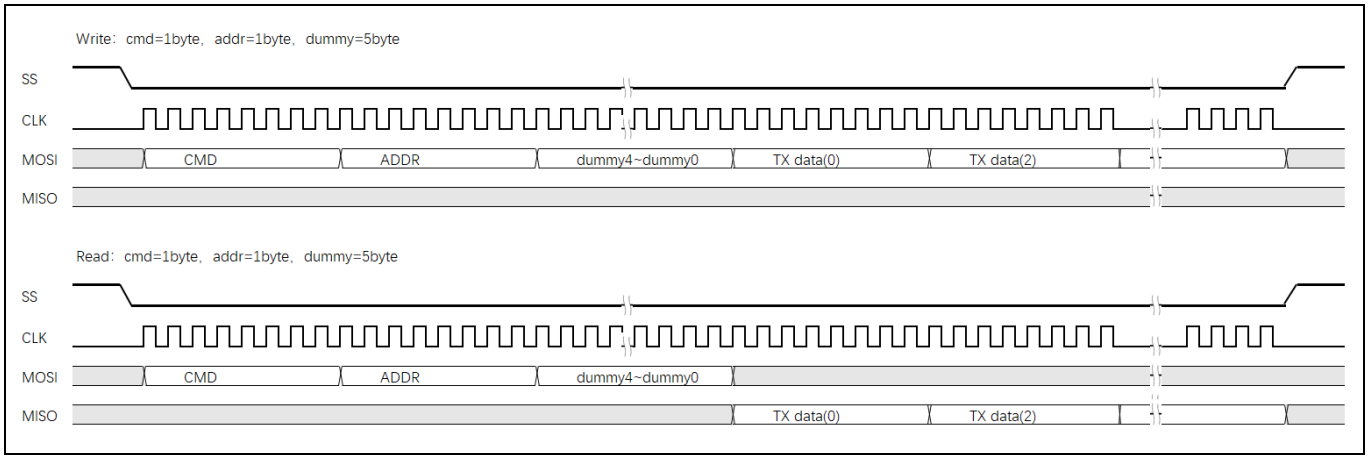


圖 1-1 SPI 收發時序

如果主機發送完命令和位址後立即傳輸資料，從機還未完成命令解析。從機無法及時讀取 RX FIFO，會發生 RX FIFO 溢出，導致從機接收丟資料；從機無法及時填寫 TX FIFO，會發生 TX 下溢中斷，導致從機無法發送資料。因此有必要在位址和資料之間增加 dummy。

從機接收時必須及時讀走 RX FIFO，防止發生 RX FIFO 溢出。因此讀取速度必須比傳送速率快。當 SPI 時鐘固定時，插入延遲間隔，可以增加兩個位元組傳輸之間的時間間隔。如圖 1-2 SPI 傳輸 1 位元組的時間和 MCU 讀取 1 位元組的時間

所示 MCU 讀取一個位元元組需要 640 ns，而 SPI 傳輸一個位元組只需要 500 ns，至少需要再增加 2.5 個 SPI CLK（156 ns）的延遲間隔，才能保證讀取速度比傳送速率快。

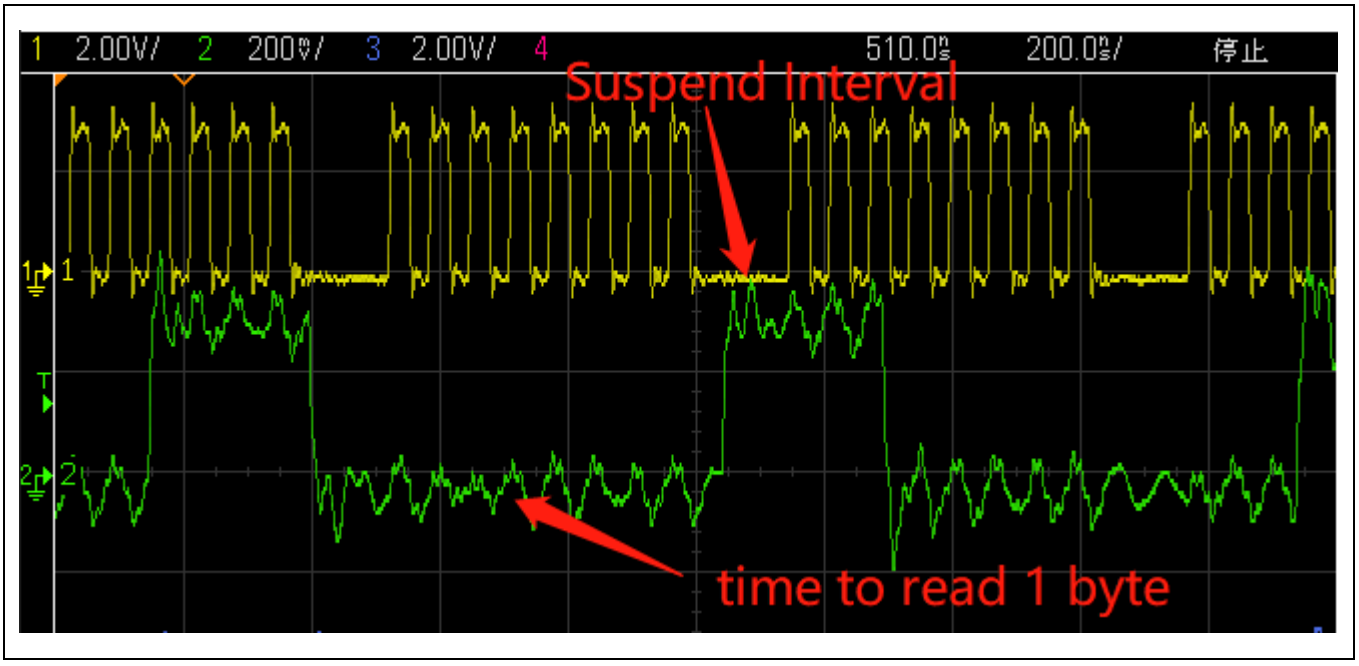


圖 1-2 SPI 傳輸 1 位元組的時間和 MCU 讀取 1 位元組的時間

從機發送時必須及時寫入 TX FIFO，防止發生 TX 下溢中斷。因此需要 dummy 傳完之前已經完成了命令解析並且往 TX FIFO 裡寫入過資料。如圖 1-3 所示主機發送第一個 CLK 到從機往 TX FIFO 裡寫入第一個資料需要 2.8 us，傳輸一個位元元組需要 500 ns。因此要保證 dummy 傳完前寫入第一個資料到 TX FIFO，傳輸命令+位址+dummy 的時間最小為 3 us（6 個位元組），本範例設置的 3.5 us（7 個位元組），如圖 1-1 所示命令和位址分別是 1 個位元組，那麼 dummy 就需要 5 個位元組。

注意：命令和位址的長度影響接收和解析的時間，更改長度後需要重新測量時間。

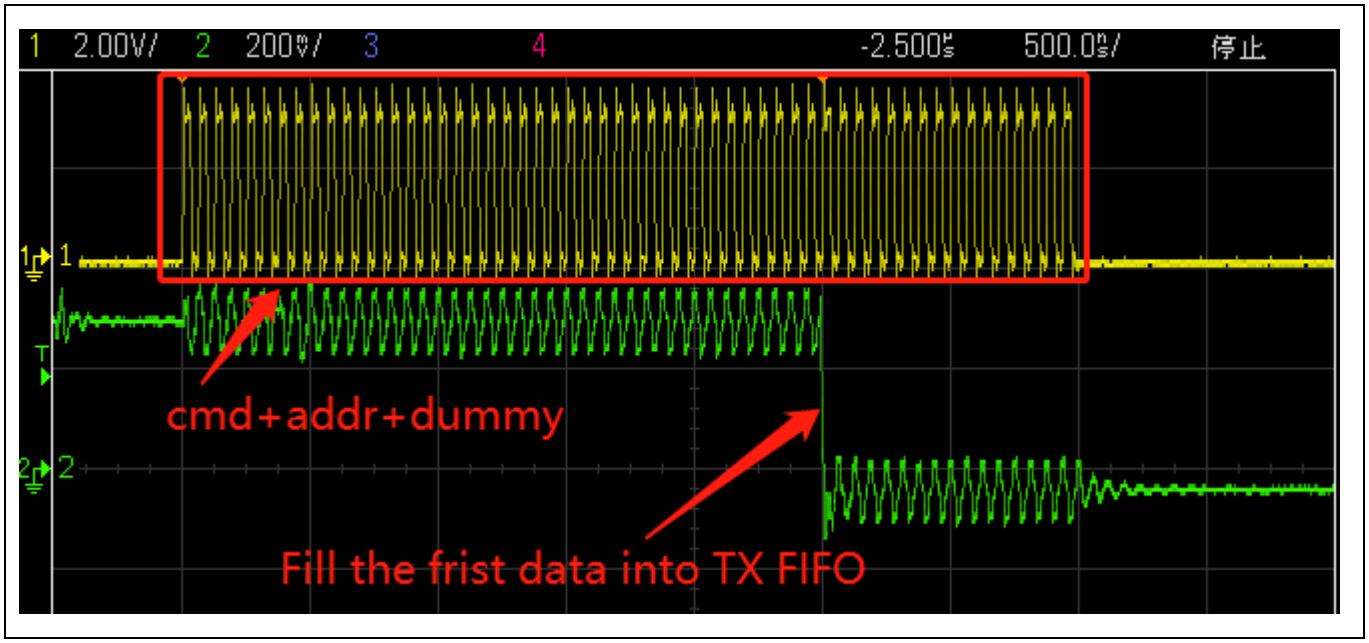


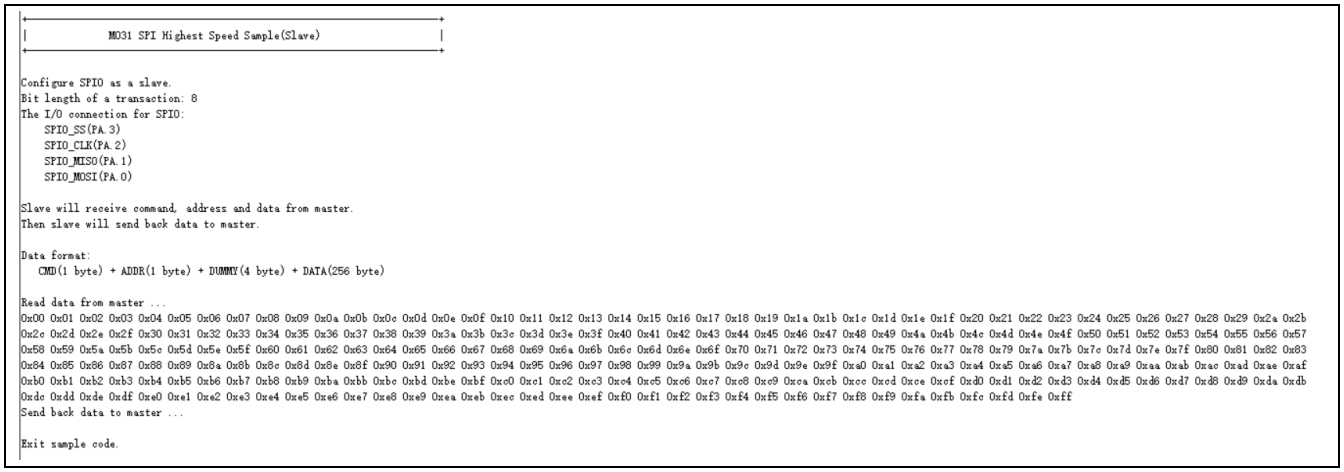
圖 1-3 從傳輸啟動到從機往 TX FIFO 裡寫入第一個資料的時間間隔

1.2 執行結果

如果將調試串口都連接到電腦上，程式執行時串口工具裡將顯示執行結果如圖 1-4 和圖 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.
```

圖 1-4 SPI 主機的執行結果



```
+-------------------------------------------------------------------+
|              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.
```

圖 1-5 SPI 從機的執行結果

2. 代碼介紹

- SPI 主機，代碼位於 *SPI_Master/main.c*

定義 SPI 參數:

```
#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 */
```

定義 SPI 通訊協定:

當資料寬度為 8 位元時，TX 和 RX FIFO 深度為 8，命令+位址+dummy 的總長度不能超過 8 位元組。

```
/*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：

主機需要設置 SPI 時鐘和 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 主機寫:

由於從機來不及讀取資料，傳輸資料之前需要設置延遲間隔。

```
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 主機讀：

為了提高讀取資料的速度，採取連續發送 8 位元組'0'，再一次性讀取 8 位元組資料的方式。

```
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));

 /* Clear 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 從機，代碼位於 *SPI_Slave/main.c*

SPI 初始化：

從機不需要設置時鐘和 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 從機接收:

```
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 從機發送：

進入函數後需要先將 TX FIFO 寫入'0'，長度等於命令+位址+dummy 的總和。否則一旦片選信號有效，TX FIFO 為空，會發生 TX 下溢，無法發送資料。

讀完位址後利用主機發送 dummy 的時間，解析命令並將需要發送的資料寫入 TX FIFO 裡。

```
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. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M031_Series_BSP_CMSIS_V3.05.000
- IDE 版本
  - Keil uVersion5.26

3.2 硬體需求

- 電路元件
  - NuMaker-M031SE V1.1
- 示意圖
  - 將兩塊開發板的 SPI0_MOSI (PA.0)、SPI0_MISO (PA.1)、SPI0_CLK (PA.2)、SPI0_SS (PA.3)、GND 分別連接在一起。
  - 兩塊開發板的 UART0 都連接到電腦上。以控制範例代碼的執行和顯示執行結果。

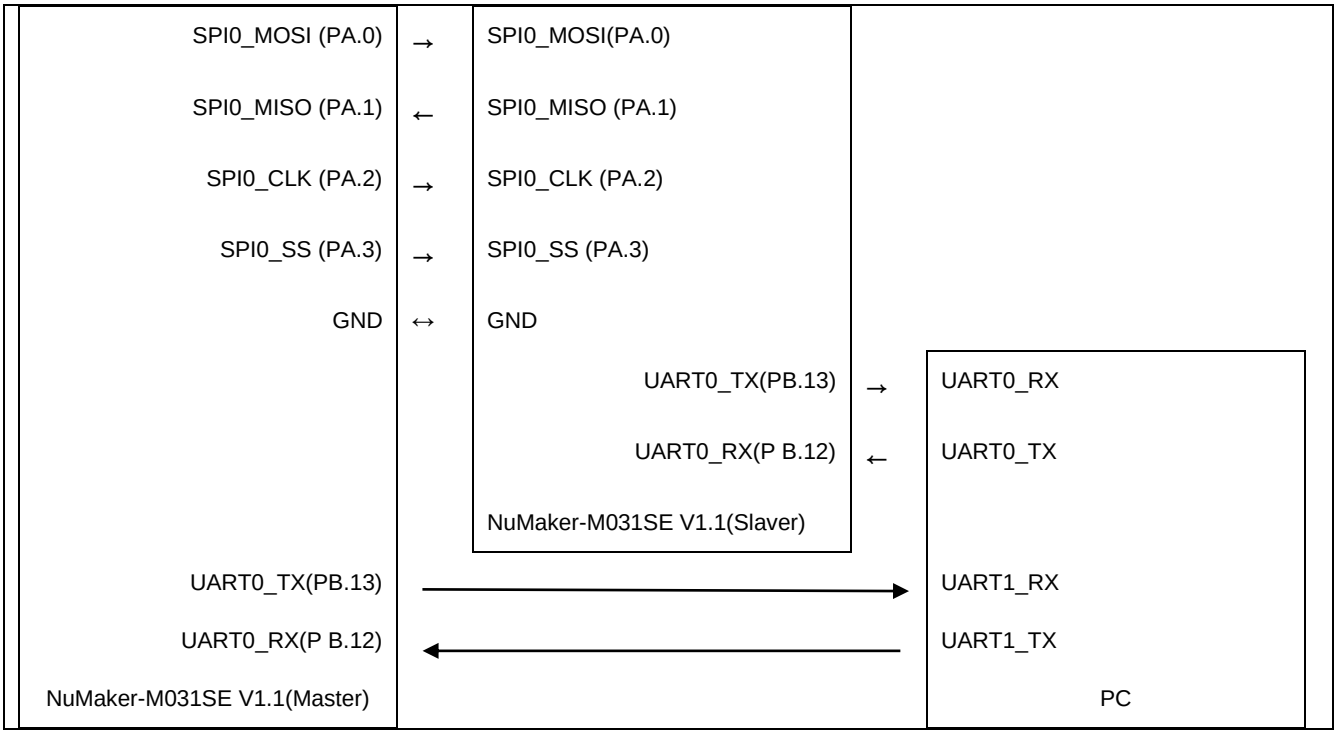


圖 3-1 引腳連接

4. 目錄資訊

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

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑，按兩下 *M031_SPI_Slave_Fastest_Speed.uvmpw*。
2. 進入 SPI_Master 編譯模式介面
   - 設置 SPI_Master 為 Active 工程
   - 編譯
   - 下載代碼至記憶體
3. 進入 SPI_Slave 編譯模式介面
   - 設置 SPI_Master 為 Active 工程
   - 編譯
   - 下載代碼至記憶體
4. 列印主機調試資訊
   - PC 上打開串口調試工具
   - 串列傳輸速率：115200bps
5. 列印從機調試資訊
   - PC 上打開串口調試工具
   - 串列傳輸速率：115200bps
6. 運行
   - 從機按重置鍵，執行代碼
   - 主機按重置鍵，執行代碼

6. 修訂紀錄

Date	Revision	Description
2022.08.10	1.00	初始發布。

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.