M031 SPI Slave Fastest Speed Transfer

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code is used for the M031 series microcontroller (MCU) as SPI slave to transfer data at the fastest speed.
BSP Version	M031_Series_BSP_CMSIS_V3.05.000
Hardware	NuMaker-M031SE V1.1

The information described in this document is the exclusive intellectual property of Hugoton 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. Overview

This example code is used for the M031 series microcontroller (MCU) as an SPI slave to receive and send data at the fastest speed.

There are two projects in the example. SPI_Master is an SPI master project, which sets SPI clock frequency to 16 MHz, sends 256 bytes of data to slave, reads back the data, compares it with the sent data, and checks whether the data sent and received by slave is correct. SPI_Slave is an SPI slave project, which receives, saves and prints the data sent by master after receiving “write” command from master; after receiving “read” command from master, the data will be sent back to master. This document introduces the example code of SPI clock frequency and role setting, increasing the suspend interval to prevent RX FIFO overflow, and filling in TX FIFO before communication and adding dummy to prevent TX underflow.

1.1 Principle

When SPI slave communicates at the fastest speed, due to the limitation of HCLK, MCU reads and writes SPI slower than SPI transfer. It is impossible to send and receive data at the same time, so half duplex communication is used. As shown in Figure 1-1, the data format is:

Command (1 byte) + Address (1 byte) + Dummy (5 bytes) + Data (256 bytes)

Wherein Command determines whether slave receives or sends data; Address determines where the data is stored; and Dummy is used to provide time to slave to parse commands and prepare data.

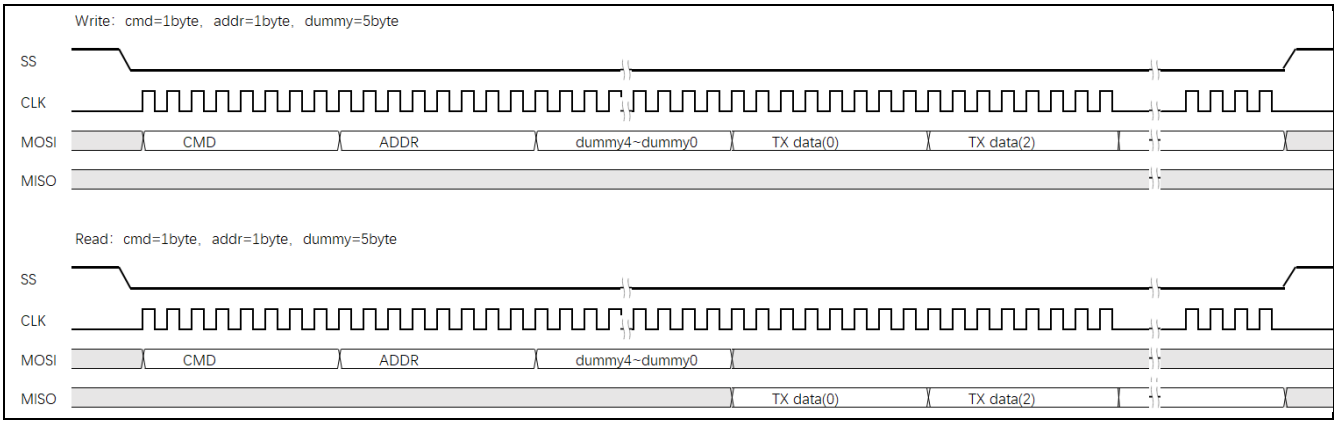


Figure 1-1 SPI Transfer Timing

If the master sends data immediately after sending the command and address, slave has not completed command parsing. Slave cannot read RX FIFO in time, RX FIFO overflow will occur, causing in data loss received by slave; Slave cannot write into TX FIFO in time, TX underflow interrupt will occur, causing in slave unable to send data. Dummy needs to be added between the address and the data.

RX FIFO must be read in time when receiving from slave to prevent RX FIFO overflow. Therefore, the read speed must be faster than the transfer speed. When SPI clock is constant, increase interval between two successive transmit/receive by modifying suspend

interval. As shown in Figure 1-2, the time for MCU to read one byte is 640 ns and the time for SPI to transmit one byte is 500 ns. A suspend interval of 2.5 SPI clock (156 ns) periods needs to be inserted to ensure that the read speed is faster than the transfer speed.

Figure 1-2 The Time for SPI to Transmit One Byte and The Time for MCU to Read One Byte

TX FIFO must be written in time when sending from slave to prevent TX underflow interrupt. Therefore, command parsing has been completed and TX FIFO has been written before the dummy transmission is completed. As shown in Figure 1-3, the interval between master sending the first CLK and slave writing the first data to TX FIFO is 2.5 us, and the time for transmitting 1 byte is 500 ns. To ensure that the first data is written to TX FIFO before dummy is transmitted, the minimum time for transmitting command + address + dummy is 3 us(6 bytes). In this example code, the time is 3.5 us (7 bytes), as shown in Figure 1-1, command and address are 1 byte, so dummy needs 5 bytes.

Note: The length of command and address affects the receiving and parsing time. After changing the length, the time needs to be measured again.

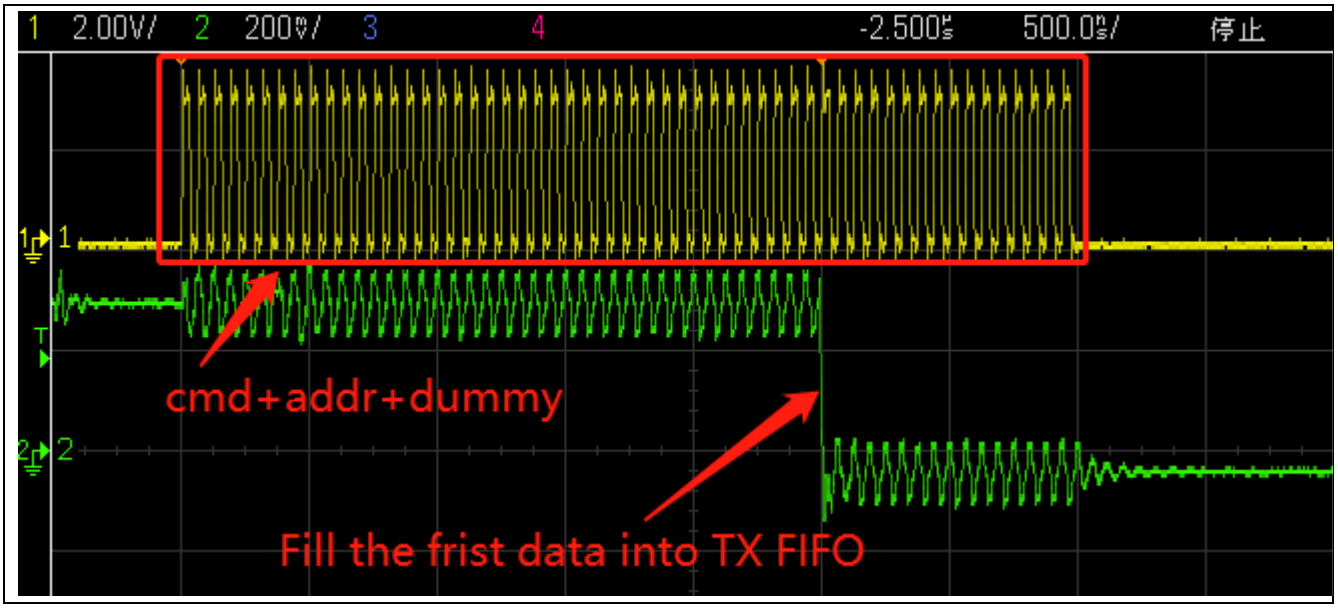


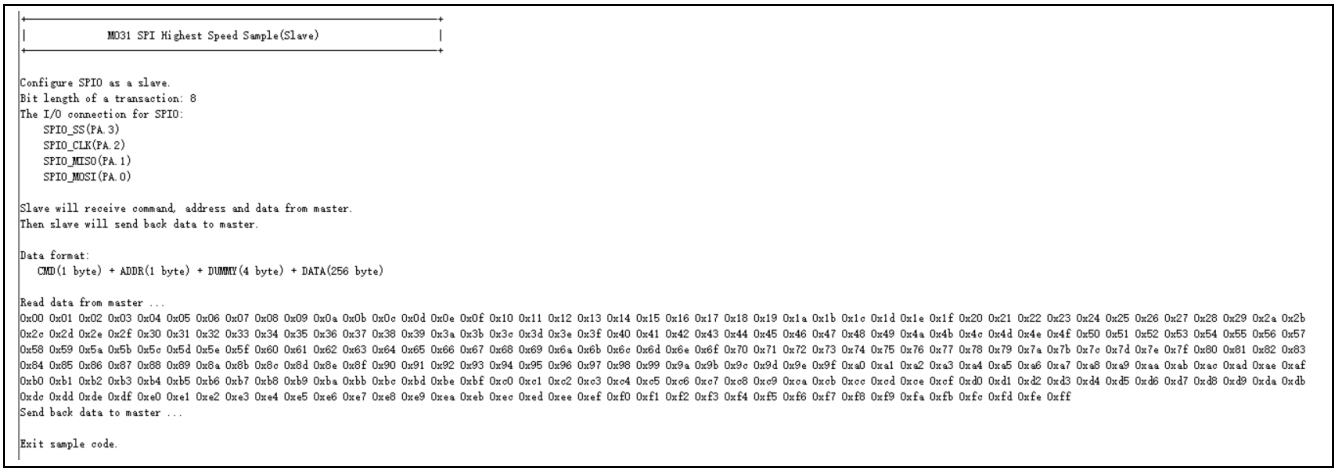
Figure 1-3 Interval between Transfer Start and Slave Writing the First Data to TX FIFO

1.2 Demo Result

If debug UARTs are connected to PC when executing the program, the UART tool can display information as shown in Figure 1-4 and Figure 1-5.

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

Figure 1-4 Master Execution Result



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

Figure 1-5 Slave Execution Result

2. Code Description

- SPI master code in *SPI_Master/main.c*

Define SPI parameters:

```
#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 */
```

Define SPI communication protocol:

When data width is 8 bits, TX FIFO and Rx FIFO is 8 levels. At this time, the total length of command + address + dummy cannot exceed 8 bytes.

```
/*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        5  /* Dummy length: byte */
/* Command opcode */
#define CMD_WRITE        2 /* Master write */
#define CMD_READ         3 /* Master read */
```

SPI initialization:

Master needs to set SPI clock and automatic slave selection.

```
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 master write:

Since slave does not have time to read data. It is necessary to set suspend interval before transmitting data.

```
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 master read:

In order to improve read speed, 8 bytes '0' are sent continuously and 8 bytes of data are read at one time.

```
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 slave code in *SPI_Slave/main.c*

SPI initialization:

Slave does not need to set SPI clock and slave selection.

```
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 slave receive:

```
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 slave send:

After entering the function, TX FIFO needs to be written to '0', and the length is equal to the sum of command + address + dummy. Otherwise, once the slave selection signal is active and TX FIFO is empty, TX underflow interrupt will occur and data cannot be sent.

After reading the address, use the time when master sends dummy to parse command and write data into TX FIFO.

```
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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M031SE V1.1
- Pin Connect
  - SPI0_MOSI (PA.0), SPI0_MISO (PA.1), SPI0_CLK (PA.2), SPI0_SS (PA.3) and GND of the two evaluation boards are connected together.
  - UART0 of both boards are connected to PC to control the execution of the sample code and display the execution results.

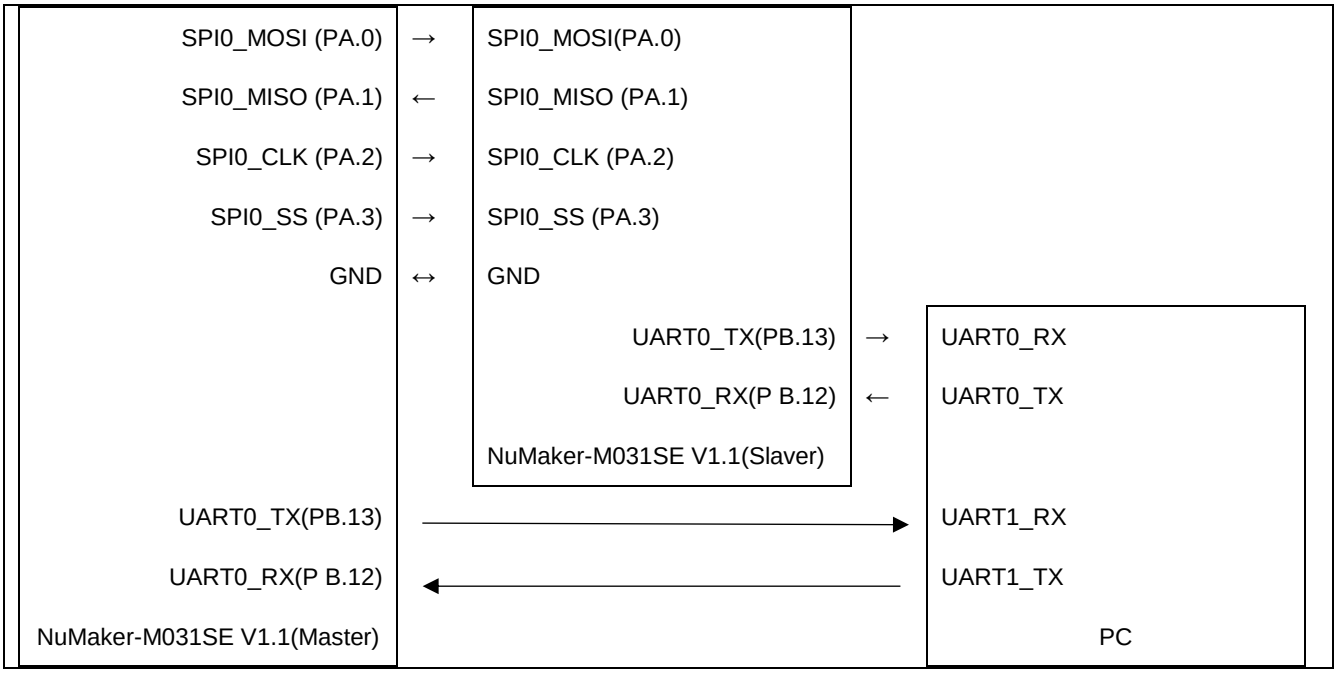


Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

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

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M031_SPI_Slave_Fastest_Speed.uvmpw*.
2. Set SPI_Master as active project, and enter KEIL compile mode
   - Set SPI_Master as Active project
   - Build
   - Download
3. Set SPI_Slave as active project, and enter KEIL compile mode
   - Set SPI_Master as Active project
   - Build
   - Download
4. Print master debug information
   - Open UART tool
   - Baud rate: 115200 bps
5. Print slave debug information
   - Open UART tool
   - Baud rate: 115200 bps
6. Run
   - Press reset key on NuMaker-M031SE V1.1, and execute slave code.
   - Press reset key on NuMaker-M031SE V1.1, and execute master code.

6. Revision History

Date	Revision	Description
2022.08.10	1.00	Initial version.

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.