应用指南

32位 Cortex™-M0单片机
NuMicro® 系列

*怎样使用**SPI 2-Bit**模式*

目录

1 简介

本文档描述了怎样使用NUC1xx系列芯片SPI控制器的2-bit模式

1.1 概览

本文中，SPI控制器配置为2-bit主机，使用2个SPI flash（W25Q16BV,华邦spiFlash）作为从机。SPI主机使用串行时钟（SPICLKx），一个从机选择（SPISSx0）以及2通道数据输入/输出（MISOx0/1, MOSIx0/1）控制这两个SPI flash。

1.2 性能

- 支持主/从模式
- 支持1位和2位数据传输
- 可配置最大传输字1~32位,一次可传输1~2字
- MSB 或 LSB 在前传送
- 主模式下2个从机选择信号
- 支持字及字节睡眠
- 支持传送和接收DMA

1.3 结构

1.3.1 SPI模块图

下图描述了SPI控制器模块图。SPICLKx是SPI主机的串行时钟，在主机模式下SPISSx0 和 SPISSx1两脚是从机选择脚，在从机模式下只有SPISSx0是从机选择脚。MOSIx0 和 MISOx0引脚组合成第一通道数据收/发脚，MOSIx1 和 MISOx1引脚组合成第二通道数据收/发脚

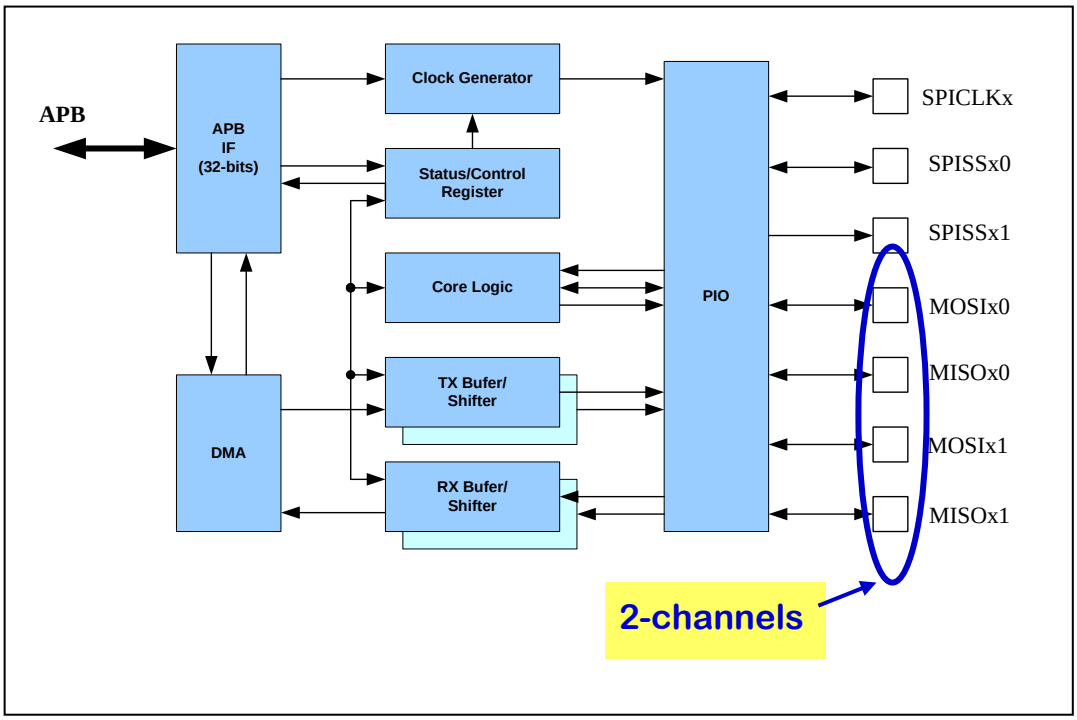


图 1 SPI 模块图

1.3.2 SPI应用框图

下图是应用框图，我们设定NUC100的SPI控制器2(SPI2)作为SPI 2-bit模式主机，SPICLK2是主机用来与两个SPI flash收发数据的时钟信号，SPISS20是SPI主机的从机选择输出，MISO20 和 MOSI20组成第一通道接到SPI flash0，MISO21 和 MOSI21组成第二通道接到SPI flash1。

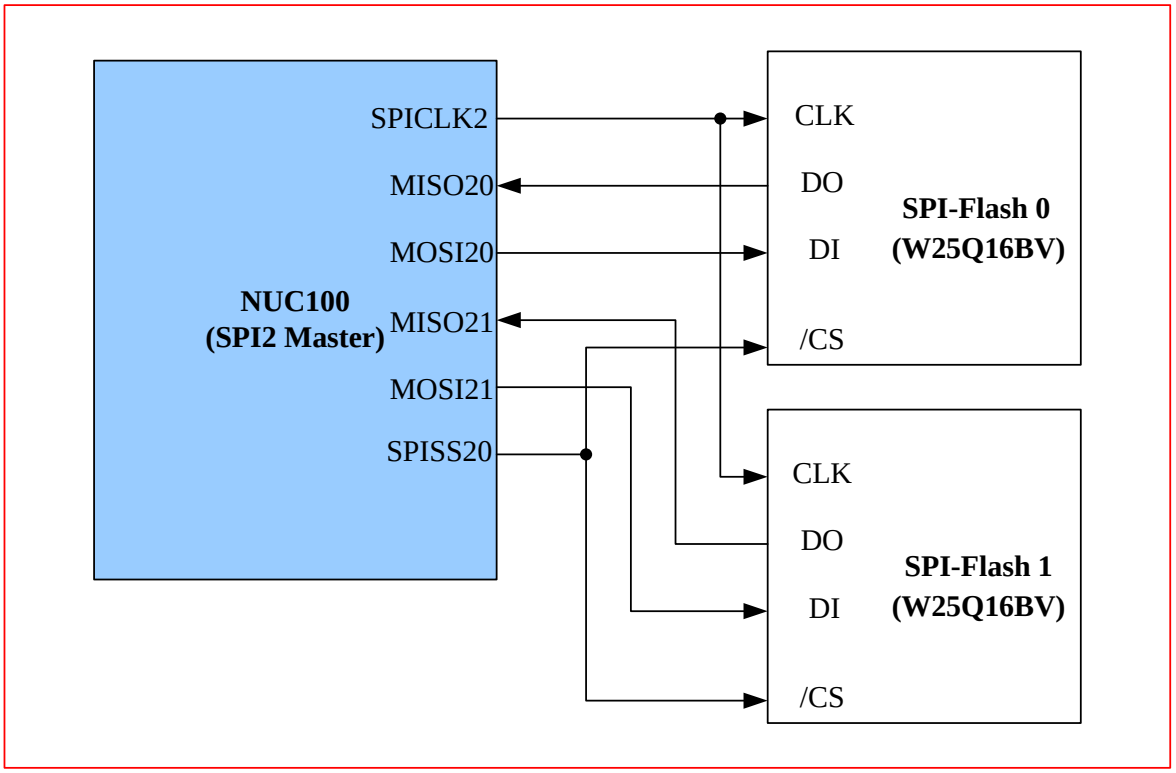


图 2 SPI 2-Bit 传送模式应用框图

1.3.3 2-Bit模式下的Tx/Rx寄存器和I/O口

在2-Bit模式中一次传送只能收发32位长度数据，MOSIx0 和 MISOx0引脚组成通道0，从SPI_Tx0寄存器发送数据，用SPI_Rx0接收数据。MOSIx1 和 MISOx1引脚组成通道1，从SPI_Tx1寄存器发送数据，用SPI_Rx1接收数据。下图描述了了寄存器和IO之间的关系：

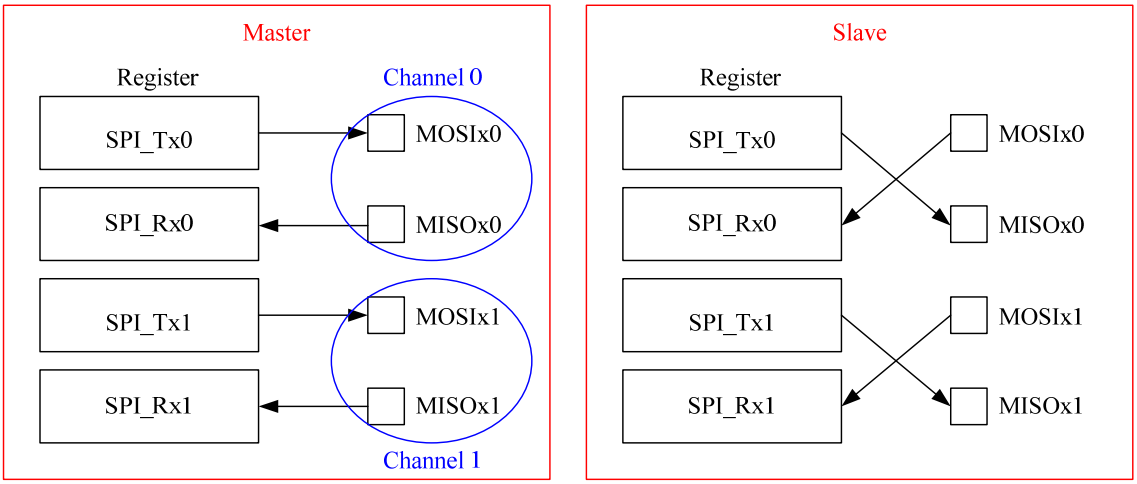


图 3 2-Bit 传送模式下 Tx/Rx 寄存器和 I/O 图

1.3.4 2-Bit传送模式时序图

下图描述SPI 2-Bit传送模式时序图：

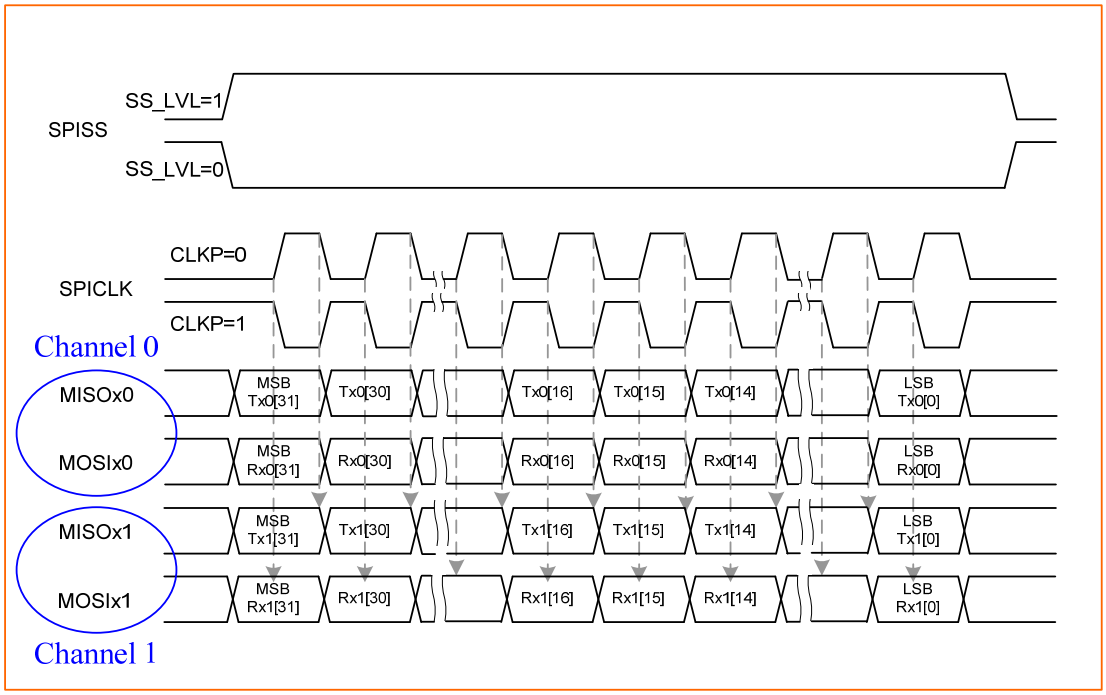


图 4 2-Bit 传送模式时序图

2 代码部分

2.1 主函数

主函数中首先检查了这2个SPI flash的MID和DID，然后擦除这2个flash，接着写数据0x00~0xff到flash0的第一页，写数据0xff~0x00到flash1的第一页，最后验证2个flash的数据

```
#define TEST_NUMBER 1
int main(void)
{
        uint32_t u32ByteCount, u32FlashAddress, u32PageNumber;
        uint8_t DataBuffer0[256];
        uint8_t DataBuffer1[256];

        /* Unlock the protected registers */
        UNLOCKREG();

        /* Enable the 12MHz oscillator oscillation */
        DrvSYS_SetOscCtrl(E_SYS_XTL12M, 1);

        /* HCLK clock source. 0: external 12MHz; 4:internal 22MHz RC oscillator */
        DrvSYS_SetHCLKSource(0);

        LOCKREG();

        /* HCLK clock frequency = HCLK clock source / (HCLK_N + 1) */
        DrvSYS_SetClockDivider(E_SYS_HCLK_DIV, 0);

        /* Configure SPI2 as a master, Type1 waveform, 32-bit transaction */
        DrvSPI_Open(eDRVSPI_PORT2, eDRVSPI_MASTER, eDRVSPI_TYPE1, 32);

        /* MSB first */
        DrvSPI_SetEndian(eDRVSPI_PORT2, eDRVSPI_MSB_FIRST);

        /* Disable the automatic slave select function of SS0. */
        DrvSPI_DisableAutoCS(eDRVSPI_PORT2);
```

```
/* Set the active level of slave select. */
DrvSPI_SetSlaveSelectActiveLevel(eDRVSPI_PORT2, eDRVSPI_ACTIVE_LOW_FALLING);

/* Configure SPI2 as 2-bit transfer mode */
DrvSPI_Set2BitSerialDataIOMode(eDRVSPI_PORT2, TRUE);

/* SPI clock rate 1MHz */
DrvSPI_SetClock(eDRVSPI_PORT2, 1000000, 0);

/* Check MID & DID */
SpiFlashx2_ReadMidDid();

/* Erase SPI flash */
SpiFlashx2_ChipErase();

/* Wait ready */
SpiFlashx2_WaitReady();

/* initial source data */
for(u32ByteCount=0; u32ByteCount<256; u32ByteCount++)
{
        DataBuffer0[u32ByteCount] = u32ByteCount;
}

/* Program SPI flash */
u32FlashAddress = 0;
for(u32PageNumber=0; u32PageNumber<TEST_NUMBER; u32PageNumber++)
{
        /* page program */
        SpiFlashx2_PageProgram(DataBuffer0, u32FlashAddress, 256);
        SpiFlashx2_WaitReady();
        u32FlashAddress += 0x100;
}
```

```
/* clear data buffer */
for(u32ByteCount=0; u32ByteCount<256; u32ByteCount++)
{
        DataBuffer0[u32ByteCount] = 0;
        DataBuffer1[u32ByteCount] = 0;
}

/* Verify SPI flash */
u32FlashAddress = 0;
for(u32PageNumber=0; u32PageNumber<TEST_NUMBER; u32PageNumber++)
{
        SpiFlashx2_ReadData(DataBuffer0, DataBuffer1, u32FlashAddress, 256);
        u32FlashAddress += 0x100;

        for(u32ByteCount=0; u32ByteCount<256; u32ByteCount++)
        {
                if(DataBuffer0[u32ByteCount]!=u32ByteCount)
                        while(1); /* Verify Error! */

                if((uint8_t)DataBuffer1[u32ByteCount]!=(uint8_t)~DataBuffer0[u32ByteCount])
                        while(1); /* Verify Error! */
        }

        /* clear data buffer */
        for(u32ByteCount=0; u32ByteCount<256; u32ByteCount++)
        {
                DataBuffer0[u32ByteCount] = 0;
                DataBuffer1[u32ByteCount] = 0;
        }
}

DrvSPI_Close(eDRVSPI_PORT2);

return 1;
```

```
}
```

2.2 检查SPI Flash的MID和DID

SPI主机发送0x90命令到这两个SPI flash,读回各自的制造ID(MID)和设备ID(DID)，检查读到的数据是否等于0xEF14，如果数据正确则NUC1xx SPI主机和两个flash从机连接成功。

```
// ***************************************
// For W25Q16BV, Manufacturer ID: 0xEF; Device ID: 0x14
void SpiFlashx2_ReadMidDid(void)
{
    uint32_t au32SourceData[2];
    uint32_t au32DestinationData[2];

    /* configure transaction length as 8 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0x90, Read Manufacturer/Device ID */
    au32SourceData[0] = 0x90;
    au32SourceData[1] = 0x90;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

    /* configure transaction length as 24 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 24);

    /* send 24-bit '0', dummy */
    au32SourceData[0] = 0x0;
    au32SourceData[1] = 0x0;
   DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
```

```
        while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

        /* configure transaction length as 16 bits */
        DrvSPI_SetBitLength(eDRVSPI_PORT2, 16);

        /* receive */
        au32SourceData[0] = 0x0;
        au32SourceData[1] = 0x0;
        DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

        /* wait */
        while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

        /* /CS: de-active */
        DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

        /* dump Rx register */
        DrvSPI_DumpRxRegister(eDRVSPI_PORT2, &au32DestinationData[0], 2);

        /* check the MID and DID */
        if ((au32DestinationData[0] & au32DestinationData[1] & 0xffff) != 0xEF14)
                while (1);  /* MID & DID Error */

}
```

2.3 擦除SPI Flash

SPI主机可以发送0xC7命令到这两个SPI flash将其擦除

```
void SpiFlashx2_ChipErase(void)
{
    uint32_t au32SourceData[2];

    /* configure transaction length as 8 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0x06, Write enable */
    au32SourceData[0] = 0x06;
    au32SourceData[1] = 0x06;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

    /* /CS: de-active */
    DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0xC7, Chip Erase */
    au32SourceData[0] = 0xc7;
    au32SourceData[1] = 0xc7;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}
```

```
        /* /CS: de-active */
        DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);
}
```

2.4 读SPI Flash的状态寄存器 1

SPI主机发送0x05命令到这两个SPI Flash可以读回各自的状态寄存器1

```
uint32_t SpiFlash2_ReadStatusReg1(void)
{
        uint32_t au32SourceData[2];
        uint32_t au32DestinationData[2];

        /* configure transaction length as 16 bits */
        DrvSPI_SetBitLength(eDRVSPI_PORT2, 16);

        /* /CS: active */
        DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

        /* send Command: 0x05, Read status register 1 */
        au32SourceData[0] = 0x0500;
        au32SourceData[1] = 0x0500;
        DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

        /* wait */
        while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

        /* /CS: de-active */
        DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

        /* dump Rx register */
        DrvSPI_DumpRxRegister(eDRVSPI_PORT2, &au32DestinationData[0], 2);

        return ((au32DestinationData[0] | au32DestinationData[1]) & 0xFF);

}
```

2.5 检查SPI flash的BUSY位

SPI主机读回SPI flash状态寄存器1的值并检查SPI flash是否正忙

```
void SpiFlashx2_WaitReady(void)
{
        uint32_t ReturnValue;

        do{
                ReturnValue = SpiFlash2_ReadStatusReg1();
                ReturnValue = ReturnValue & 1;
        }while(ReturnValue!=0); // check the BUSY bit

}
```

2.6 SPI Flash编程

SPI主机发送0x02命令到SPI flash，可以对其编程

```
void SpiFlashx2_PageProgram(uint8_t *DataBuffer, uint32_t StartAddress, uint32_t ByteCount)
{
    uint32_t au32SourceData[2];
    uint32_t Counter;

    /* configure transaction length as 8 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0x06, Write enable */
    au32SourceData[0] = 0x06;
    au32SourceData[1] = 0x06;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

    /* /CS: de-active */
    DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0x02, Page program */
    au32SourceData[0] = 0x02;
    au32SourceData[1] = 0x02;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}
```

```
        /* configure transaction length as 24 bits */
        DrvSPI_SetBitLength(eDRVSPI_PORT2, 24);

        /* send 24-bit start address */
        au32SourceData[0] = StartAddress;
        au32SourceData[1] = StartAddress;
        DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

        /* wait */
        while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

        /* configure transaction length as 8 bits */
        DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

        for(Counter=0; Counter<ByteCount; Counter++)
        {
                /* send data to program */
                 au32SourceData[0] = DataBuffer[Counter];
                 au32SourceData[1] = ~au32SourceData[0];
                DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

                 /* wait */
                 while (DrvSPI_Busy(eDRVSPI_PORT2)) {}
        }

        /* /CS: de-active */
        DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

}
```

2.7 从SPI Flash读数据

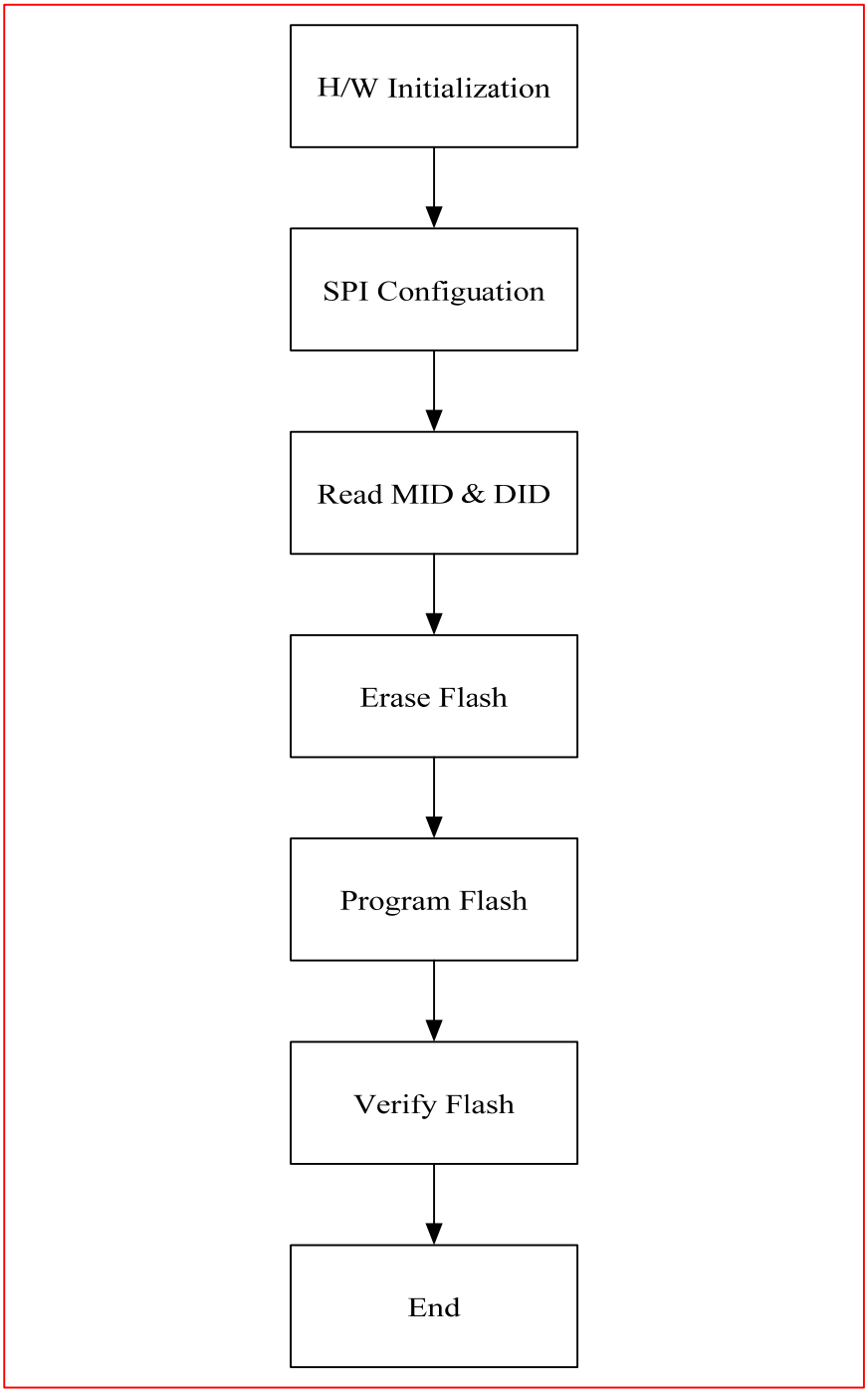
SPI主机发送0x03命令到SPI Flash可以读其数据

```
void SpiFlashx2_ReadData(uint8_t *DataBuffer0, uint8_t *DataBuffer1, uint32_t StartAddress, uint32_t ByteCount)
{
    uint32_t au32SourceData[2];
    uint32_t au32DestinationData[2];
    uint32_t Counter;

    /* configure transaction length as 8 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

    /* /CS: active */
    DrvSPI_SetCS(eDRVSPI_PORT2, eDRVSPI_SS0);

    /* send Command: 0x03, Read data */
    au32SourceData[0] = 0x03;
    au32SourceData[1] = 0x03;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

    /* configure transaction length as 24 bits */
    DrvSPI_SetBitLength(eDRVSPI_PORT2, 24);

    /* send 24-bit start address */
    au32SourceData[0] = StartAddress;
    au32SourceData[1] = StartAddress;
    DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

    /* wait */
    while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

    /* configure transaction length as 8 bits */
```

```
        DrvSPI_SetBitLength(eDRVSPI_PORT2, 8);

        for(Counter=0; Counter<ByteCount; Counter++)
        {
                /* receive */
                au32SourceData[0] = 0x0;
                au32SourceData[1] = 0x0;
                DrvSPI_BurstWrite(eDRVSPI_PORT2, &au32SourceData[0]);

                /* wait */
                while (DrvSPI_Busy(eDRVSPI_PORT2)) {}

                /* dump Rx register */
                DrvSPI_DumpRxRegister(eDRVSPI_PORT2, &au32DestinationData[0], 2);
                DataBuffer0[Counter] = (uint8_t) au32DestinationData[0];
                DataBuffer1[Counter] = (uint8_t) au32DestinationData[1];
        }

        /* /CS: de-active */
        DrvSPI_ClrCS(eDRVSPI_PORT2, eDRVSPI_SS0);

}
```

3 调用顺序

3.1 综合SPI控制器和SPI Flash



3.2 NUC100 设定

1. 硬件初始化：使能12MHz晶振，设置HCLK时钟源及其分频
2. 配置SPI控制器2为主机，2-bit传送模式，数据在下降沿发送，上升沿接收，禁止自动从机选择输出，从机选择有效电平为低，比特率1MHz
3. 读回MID和DID检查连接是否成功
4. 整片擦除这两个SPI flash
5. 对这两个SPI Flash的第一页写入互补的数据
6. 从两个SPI Flash中读出数据并检查
7. 关闭SPI控制器2

3.3 有关的API参考：

1. SPI Driver Reference Guide.doc

4 运行环境设置及结果

4.1 测试 Smpl_SPI_Flashx2

这个SPI示例程序使用2-bit传送模式控制两个SPI flash，Smpl_SPI_Flashx2可以用Keil MDK编译并通过ICE下载到NUC1xx系列学习板，然后用户可以在ICE环境下执行代码，或者复位学习板运行片内Flash程序

4.2 结果

擦除这两个Flash，写数据0x00~0xff到flash0的第一页，写数据0xff~0x00到flash1的第一页

5 修订历史

版本.	日期	描述
1.00	2010-5-12	1. 初次发布

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Further more, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur.

Nuvoton customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from such improper use or sales.

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.