应用指南

32位 Cortex™-M0 单片机
NuMicro® 系列

怎样通过SPI使用AD7793?

目录

1 简介

本应用指南介绍了NUC1xx的SPI结构，并通过一个简单程序讲解如何通过SPI接口控制AD7793

1.1 性能

- 支持主/从操作
- 支持1位和2位数据传输
- 可配置最大传输32位数据
- 主模式下输出时钟频率可变
- 支持突发模式，一次传输可以发送/接收2次
- 支持MSB或LSB在前传输模式
- 主模式下2条主/从选择线，从模式下1条选择线
- 同一组时钟下，全静态同步设计
- 字节睡眠模式
- 支持两组可编程串行时钟频率输出

1.2 结构

串行设备接口（SPI）是一个全双工同步串行数据通信协议，设备使用4线双向接口在主/从模式下通信。

NUC1xx系列MCU内置了一些SPI控制器，用来将接收到的终端设备数据进行串并转换，以及在向设备发送数据时进行并串转换。每个SPI可以最多驱动两个外设，但是此操作是分时进行的，不可同时操作。当SLAVE(CNTRL[18])位被置1时，也可以当作从设备使用。

当数据传送结束时，每组SPI控制器可以单独产生中断，相应中断标志位写1清零。根据外设配置SSR[SS_LVL]可以设定主/从设备的使能电平是高还是低。在主机模式下，对DIVIDER写入值，可以配置传输时钟的频率。如果SPI_CNTRL[23]寄存器的VARCLK_EN 位被使能，传输时钟可根据DIV和DIV2 ，被分为两组不同的频率。可变频率根据VARCLK来确定

该主/从机内核包含2组32位输出/输入数据缓存，可支持突发传输 (burst mode)模式，并支持各种不同长度的变量数据模式，最多可支持64 位数据传输

通过配置SPI_CNTL[22]，MCU也可支持2位数据传输模式，当SPI_CNTL[22]寄存器的TWOB位使能，即可传输2位数据的输入/输出。第一位数据从SPI_TX0输出或从SPI_RX0输入，第二位数据从SPI_TX1输出或从SPI_RX0输入。

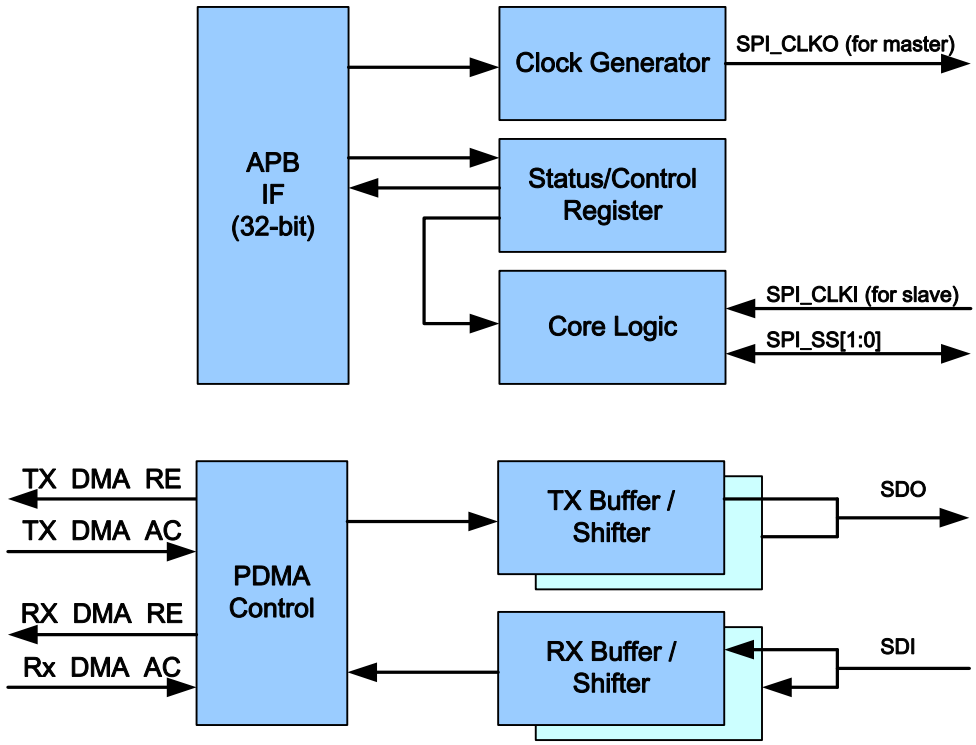

APB
IF
(32-bit)
Clock Generator
SPI_CLKO (for master)
Status/Control
Register
Core Logic
SPI_CLKI (for slave)
SPI_SS[1:0]
TX DMA RE
TX DMA AC
RX DMA RE
Rx DMA AC
PDMA
Control
TX Buffer /
Shifter
SDO
RX Buffer /
Shifter
SDI

图 1 SPI 控制器框图

1.3 SPI时序图(Master/Slave)

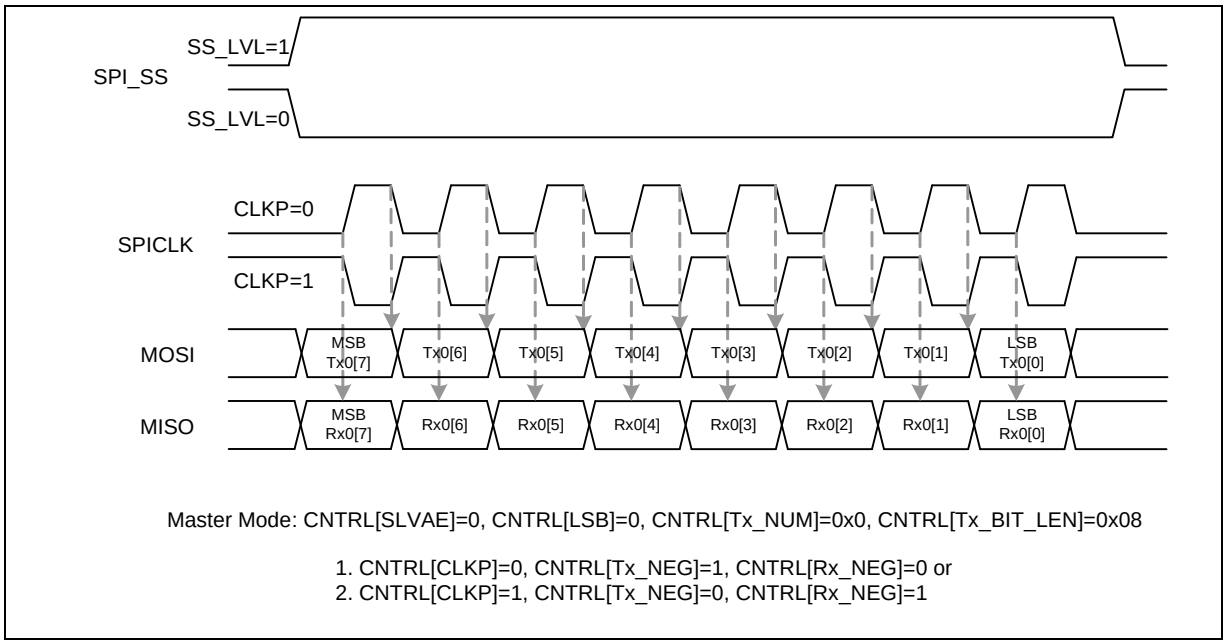


图 2 MICROWIRE/SPI 时序 (主)

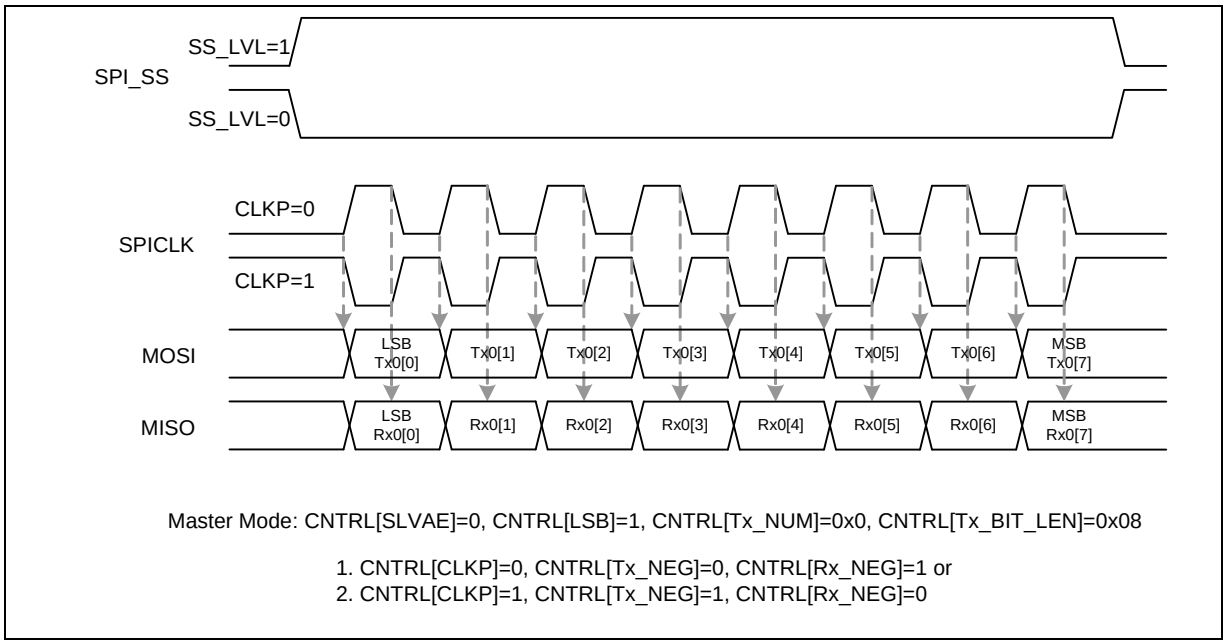
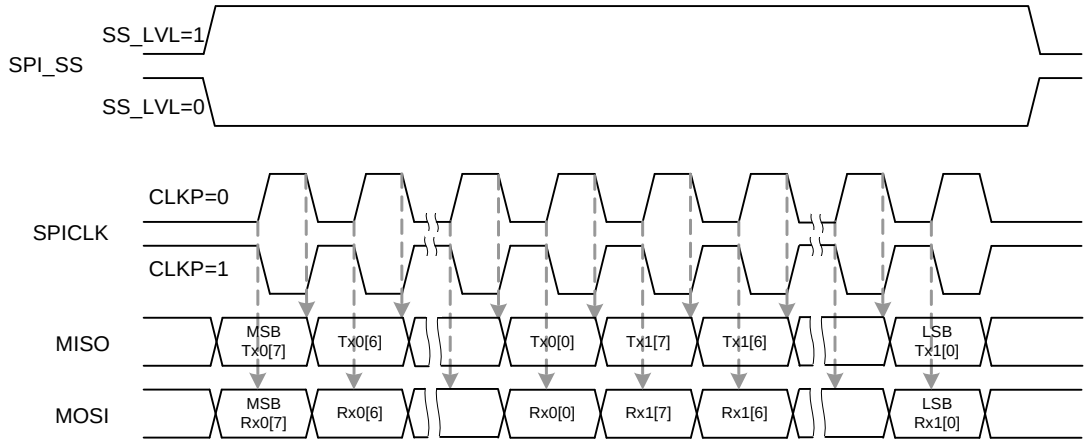


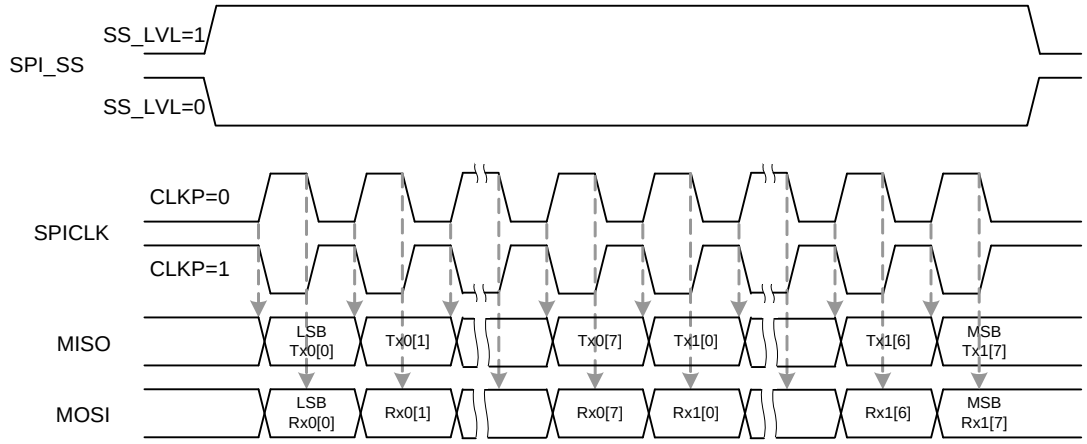
图 3 相位变化的 SCLK 时钟时序(主)



Slave Mode: CNTRL[SLVAE]=1, CNTRL[LSB]=0, CNTRL[Tx_NUM]=0x01, CNTRL[Tx_BIT_LEN]=0x08

1. CNTRL[CLKP]=0, CNTRL[Tx_NEG]=1, CNTRL[Rx_NEG]=0 or
2. CNTRL[CLKP]=1, CNTRL[Tx_NEG]=0, CNTRL[Rx_NEG]=1

图 4 MICROWIRE /SPI 时序(从)



Master Mode: CNTRL[SLVAE]=1, CNTRL[LSB]=1, CNTRL[Tx_NUM]=0x01, CNTRL[Tx_BIT_LEN]=0x08

1. CNTRL[CLKP]=0, CNTRL[Tx_NEG]=0, CNTRL[Rx_NEG]=1 or
2. CNTRL[CLKP]=1, CNTRL[Tx_NEG]=1, CNTRL[Rx_NEG]=0

图 5 相位变化的 SCLK 时钟时序 (从)

2 如何编程SPI

2.1 SPI的编程流程

1. 设置APBCLK[SPIx_EN]位使能SPIx时钟，设置IPRSTC2[SPIx_RST]位复位SPIx模块
2. 选择GPxMFP中SPIx_SS0, SPIx_CLK, SPI0_MISOx 和 SPIx_MOSI0 使能复用脚的SPI接口
3. 设置SPI_DIVIDER寄存器中的值,定义SPIx时钟= HCLK/(DIVIDER+1).
4. 设置CNTRL寄存器SLAVE位定义主/从模式，然后设置LSB, CLKP, TX_NEG 和RX_NEG 定义SPI的时序
5. 设置SSR寄存器中ASS, SS_LVL以及SSR位定义SPIx_SS引脚状态

2.2 示例代码

2.2.1 AD7793.c

此代码支持两种SPI操作模式，如果定义了SPI_EN_ASS，自动从选择模式可以在设置SPx->CNTRL.GO_BUSY位后自动使能或禁止SPI_SS脚，或者在手动从选择模式下，SPI_SS脚被SPIx->SSR.SSR控制

```
#include <stdio.h>
#include "NUC1xx.h"
#include "AD7793.h"

// Define AD7793 register size and mode (read/write)
REGISTER_TYPE AD7793_REG[ALL_REGS]={
  {8, REG_RD},                          // Status Reg
  {16,REG_RD_WR},                         // Mode Reg
  {16,REG_RD_WR},                         // Configuration Reg
  {24,REG_RD},                          // Data Reg
  {8, REG_RD},                          // ID Reg
  {8, REG_RD_WR},                         // IO Reg
  {24,REG_RD_WR},                         // Offset Reg
  {24,REG_RD_WR},                         // Full-Scale Reg
};

#ifdef SPI_EN_ASS
/*---------------------------------------------------------------------------------------------------------*/
/* Enable automatic slave select                                       */
/*---------------------------------------------------------------------------------------------------------*/
// Initialize required GPIO ports and SPI interface
```

```
void  InitSPI(void)
{
        /* Step 1. Enable and Select SPI clock source*/
        SYSCLK->APBCLK.SPI0_EN  =1;
        SYS->IPRSTC2.SPI0_RST   =1;
        SYS->IPRSTC2.SPI0_RST   =0;

        /* Step 2. GPIO initial */
        SYS->GPCMFP.SPI0_SS0      =1;
        SYS->GPCMFP.SPI0_CLK      =1;
        SYS->GPCMFP.SPI0_MISO0    =1;
        SYS->GPCMFP.SPI0_MOSI0    =1;

        /* Step 3. Set SCLK */
        SPI0->DIVIDER.DIVIDER=4;    //HCLK/(DIVIDER+1) = 22.1184Mhz/(4+1) = 4.42568Mhz

        /* Step 4. Select Operation mode */
        SPI0->CNTRL.SLAVE = 0;      //Master mode

        SPI0->CNTRL.LSB = 0;        //MSB
//      SPI0->CNTRL.TX_BIT_LEN = 8; //Length:8 bit

        SPI0->CNTRL.CLKP = 1;       //SCLK idle high
        SPI0->CNTRL.TX_NEG = 1;     //Change SDO on falling edge of SCLK
        SPI0->CNTRL.RX_NEG = 0;     //Latch SDI on rising edge of SCLK

        SPI0->SSR.ASS = 1;          //Disable automatic slave select
        SPI0->SSR.SS_LVL = 0;       //SS low active
        SPI0->SSR.SSR = 1;          //SPI0_SS0
}


void  wAD7793(uint8_t RegAdr,uint32_t RegData)
{
```

```
  while(SPI0->CNTRL.GO_BUSY);                  //Check busy
  SPI0->CNTRL.TX_BIT_LEN = 8 + AD7793_REG[RegAdr].len;  //Set length of TX and RX
  SPI0->TX[0] = (RS(RegAdr)<<AD7793_REG[RegAdr].len) + RegData; //Store data into TX buffer
  SPI0->CNTRL.GO_BUSY=1;                       //Start TX and RX
}
uint32_t rAD7793(uint8_t RegAdr)
{
  while(SPI0->CNTRL.GO_BUSY);                  //Check busy
  SPI0->CNTRL.TX_BIT_LEN = 8 + AD7793_REG[RegAdr].len;  //Set length of TX and RX
  SPI0->TX[0] = (RW + RS(RegAdr))<<AD7793_REG[RegAdr].len;//Store data into TX buffer
  SPI0->CNTRL.GO_BUSY=1;                       //Start TX and RX

  while(SPI0->CNTRL.GO_BUSY);                  //Check busy
  return SPI0->RX[0]&((1<<AD7793_REG[RegAdr].len)-1);
}

void ResetAD7793(void)
{
  while(SPI0->CNTRL.GO_BUSY);         //Check busy
  SPI0->CNTRL.TX_BIT_LEN = 0;         //Set length for 32 bits
  SPI0->TX[0] = 0xFFFFFFFF;           //MOSI keep high
  SPI0->CNTRL.GO_BUSY=1;              //Start TX and RX
}
#else
/*---------------------------------------------------------------------------------------------------------*/
/* Disable automatic slave select                                        */
/*---------------------------------------------------------------------------------------------------------*/
void  InitSPI(void)
{
        /* Step 1. Enable and Select SPI clock source*/
        SYSCLK->APBCLK.SPI0_EN  =1;
        SYS->IPRSTC2.SPI0_RST   =1;
        SYS->IPRSTC2.SPI0_RST   =0;
```

```
        /* Step 2. GPIO initial */
        SYS->GPCMFP.SPI0_SS0      =1;
        SYS->GPCMFP.SPI0_CLK      =1;
        SYS->GPCMFP.SPI0_MISO0    =1;
        SYS->GPCMFP.SPI0_MOSI0    =1;

        /* Step 3. Set SCLK */
        SPI0->DIVIDER.DIVIDER     =4;//HCLK/(DIVIDER+1)     =     22.1184Mhz/(4+1)     =
4.42568Mhz

        /* Step 4. Select Operation mode */
        SPI0->CNTRL.SLAVE = 0;              //Master mode

        SPI0->CNTRL.LSB = 0;        //MSB
//      SPI0->CNTRL.TX_BIT_LEN = 8; //Length:8 bit

        SPI0->CNTRL.CLKP = 1;       //SCLK idle high
        SPI0->CNTRL.TX_NEG = 1;     //Change SDO on falling edge of SCLK
        SPI0->CNTRL.RX_NEG = 0;     //Latch SDI on rising edge of SCLK

        SPI0->SSR.ASS = 0;          //Disable automatic slave select
        SPI0->SSR.SS_LVL = 0;       //SS low active
//      SPI0->SSR.SSR = 1;          //SPI0_SS0
}


void  wAD7793(uint8_t RegAdr,uint32_t RegData)
{
  SPI0->SSR.SSR = 1;                                        //SPI0_SS0 active
  SPI0->CNTRL.TX_BIT_LEN = 8 + AD7793_REG[RegAdr].len;  //Set length of TX and RX
  SPI0->TX[0] = (RS(RegAdr)<<AD7793_REG[RegAdr].len) + RegData; //Store data into TX buffer
  SPI0->CNTRL.GO_BUSY=1;                                //Start TX and RX
  while(SPI0->CNTRL.GO_BUSY);                               //Check busy
  SPI0->SSR.SSR = 0;                                        //SPI0_SS0 inactive
}
```

```
uint32_t rAD7793(uint8_t RegAdr)
{
  SPI0->SSR.SSR = 1;                                          //SPI0_SS0 active
  SPI0->CNTRL.TX_BIT_LEN = 8 + AD7793_REG[RegAdr].len;  //Set length of TX and RX
  SPI0->TX[0] = (RW + RS(RegAdr))<<AD7793_REG[RegAdr].len;//Store data into TX buffer
  SPI0->CNTRL.GO_BUSY=1;                                      //Start TX and RX

  while(SPI0->CNTRL.GO_BUSY);                                 //Check busy
  SPI0->SSR.SSR = 0;                                          //SPI0_SS0 inactive

  return SPI0->RX[0]&((1<<AD7793_REG[RegAdr].len)-1);
}

void ResetAD7793(void)
{
  SPI0->SSR.SSR = 1;                  //SPI0_SS0 active
  SPI0->CNTRL.TX_BIT_LEN = 0;         //Set length for 32 bits
  SPI0->TX[0] = 0xFFFFFFFF;     //MOSI keep high
  SPI0->CNTRL.GO_BUSY=1;              //Start TX and RX
  while(SPI0->CNTRL.GO_BUSY);         //Check busy
  SPI0->SSR.SSR = 0;                  //SPI0_SS0 inactive
}
#endif
```

2.2.2 Smpl_SPI.c

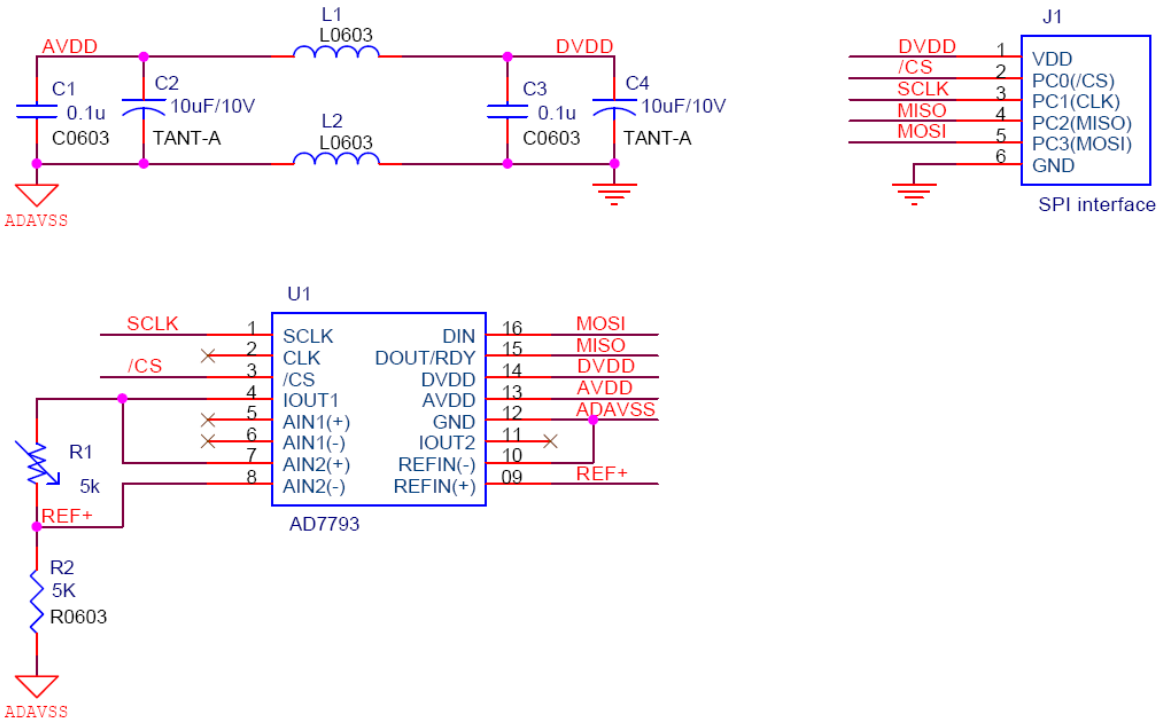
查询AD7793状态并显示ADC结果

```
#include <stdio.h>
#include "NUC1xx.h"
#include "AD7793.h"



void InitUART(void)
{
        /* Step 1. Enable and Select UART clock source*/
```

```
    UNLOCKREG();
    SYSCLK->PWRCON.XTL12M_EN = 1;
    LOCKREG();

    SYSCLK->APBCLK.UART0_EN = 1;//Enable UART clock
    SYSCLK->CLKSEL1.UART_S = 0;     //Select external 12Mhz for UART clock source

    SYSCLK->CLKDIV.UART_N = 0;      //UART clock source = 12Mhz;

    /* Step 2. GPIO initial */
    SYS->GPBMFP.UART0_RX    =1;
    SYS->GPBMFP.UART0_TX    =1;

    /* Step 3. Set BaudRate */
    UART0->BAUD.DIVX_EN = 1;
    UART0->BAUD.DIVX1   = 1;
    UART0->BAUD.DIV = 12000000 / 115200  -2;

    /* Step 4. Select Operation mode */
    UART0->FCR.TFR =1;              //Reset Tx FIFO
    UART0->FCR.RFR =1;              //Reset Rx FIFO

    UART0->FCR.RFITL = 0;//Set Rx Trigger Level -1byte FIFO
    UART0->LCR.PBE      = 0;//Disable parity
    UART0->LCR.WLS      = 3;//8 data bits
    UART0->LCR.NSB      = 0;//Enable 1 Stop bit
}

/*---------------------------------------------------------------------------
  MAIN function
  ---------------------------------------------------------------------------*/
int32_t main (void)
{
    InitUART();
    printf("\nUART initial ok\n");
```

```
        InitSPI();
        ResetAD7793();
        wAD7793(CONF_REG,0x0011);//Gain = 1, external reference, select AIN2(+)-AIN2(-)
        wAD7793(IO_REG,0x0A);          //Both current source connect to pin IOUT1. (200uA mode)
        printf("\n");
        printf("+---------------------------------------+\n");
        printf("|          AD7793 setting               |\n");
        printf("+---------------------------------------+\n");
        printf("|STAT_REG:%x\n",rAD7793(STAT_REG));
        printf("|MODE_REG:%x\n",rAD7793(MODE_REG));
        printf("|CONF_REG:%x\n",rAD7793(CONF_REG));
        printf("|ID_REG:%x\n",rAD7793(ID_REG));
        printf("|IO_REG:%x\n",rAD7793(IO_REG));
        printf("|OFFS_REG:%x\n",rAD7793(OFFS_REG));
        printf("|FUL_SC_REG:%x\n",rAD7793(FUL_SC_REG));
        printf("+---------------------------------------+\n");

        do
        {
                while(rAD7793(STAT_REG)&0x80);                  //Wait ADC data ready
                printf("DATA_REG:%x\n",rAD7793(DATA_REG));//print the result of ADC
                printf("Please press any key to update ADC, except \"ESC\".\n");
        }while(GetChar()!=0x1B);
        //Continuous or not

        printf("Exit\n");
}
```

3 电路图


L1
L0603
AVDD
DVDD
C1
0.1u
C0603
C2
10uF/10V
TANT-A
L2
L0603
C3
0.1u
C0603
C4
10uF/10V
TANT-A
ADAVSS
J1
DVDD
/CS
SCLK
MISO
MOSI
1
2
3
4
5
6
VDD
PC0(/CS)
PC1(CLK)
PC2(MISO)
PC3(MOSI)
GND
SPI interface
U1
SCLK
/CS
R1
5k
REF+
R2
5K
R0603
ADAVSS
SCLK
CLK
/CS
IOUT1
AIN1(+)
AIN1(-)
AIN2(+)
AIN2(-)
DIN
DOUT/RDY
DVDD
AVDD
GND
IOUT2
REFIN(-)
REFIN(+)
MOSI
MISO
DVDD
AVDD
ADAVSS
REF+
AD7793

4 修订历史

版本.	日期	描述
1.00	2010-3-1	1. 初次发布
1.01	2010-4-8	1. 移除寄存器描述

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.