Application Note

32-bit Cortex™-M0 MCU
NuMicro® Family

How to use AD7793 via SPI?

Table of Contents-

1 INTRODUCTION

This application note introduces SPI structure of NUC1xx and supports a simple code to control AD7793 via SPI interface of NUC1xx.

1.1 Feature

- Support master or slave operation
- Support 1 and 2-bit serial data IN/OUT
- Configurable data length of transfer word up to 32 bits
- Variable output serial clock frequency in master mode
- Provide burst mode operation, transmit/receive can be executed up to two times in one transfer
- MSB or LSB first data transfer
- 2 slave/device select lines when it is set as the master mode, and 1 slave/device select line when it is set as slave mode
- Fully static synchronous design with one clock domain
- Byte Suspend Sleep Mode
- Support two programmable serial output clock frequency.

1.2 Structure

The Serial Peripheral Interface (SPI) is a synchronous serial data communication protocol which operates in full duplex mode. Devices communicate in master/slave mode with 4-wire bi-direction interface.

NUC1xx series contain some sets of SPI controller performing a serial-to-parallel conversion on data received from a peripheral device, and a parallel-to-serial conversion on data transmitted to a peripheral device. Each SPI set can drive up to 2 external peripherals, but is time-sharing and can not operate simultaneously. It also can be driven as the slave device when the SLAVE bit (CNTRL[18]) is set.

Each controller can generate an individual interrupt signal when data transfer is finished and can be cleared by writing 1 to the respective interrupt flag. The active level of device/slave select signal can be programmed to low active or high active on SSR[SS_LVL] bit, which depends on the connected peripheral. Writing a divisor into DIVIDER register can program the frequency of serial clock output when it is as the master. If the VARCLK_EN bit in SPI_CNTRL[23] is enabling, the serial clock can be set as two programmable frequencies which are defined in DIV and DIV2. The format of the variable frequency is defined in VARCLK.

This master/slave core contains two 32-bit transmit/receive buffers, and can provide burst mode operation. It supports variable length transfer and the maximum transmitted/received length can be up to 64 bits.

The controller also supports two bits data mode which is defined in the SPI_CNTL[22]. When the TWOB bit, in SPI_CNTL[22], is enabling, it can transmits and receives two bit serial data out/in the serial buffer. The 1st bit channel transmits the data from SPI_TX0 and receives the data into SPI_RX0. The 2nd bit channel transmits the data from SPI_TX1 and receives the data into SPI_RX1.

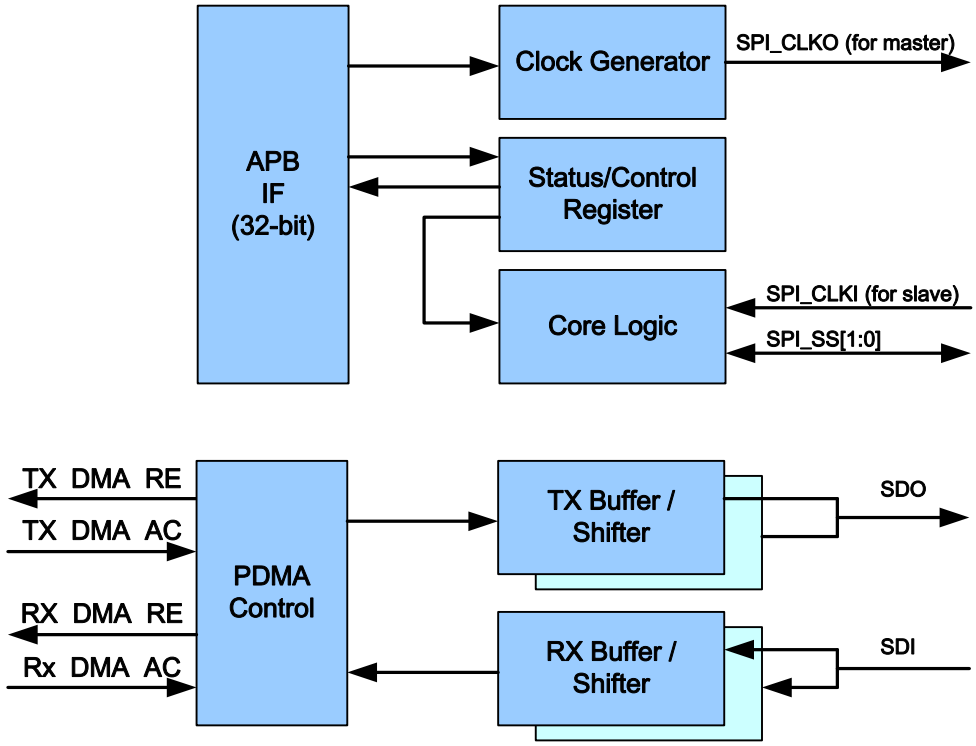
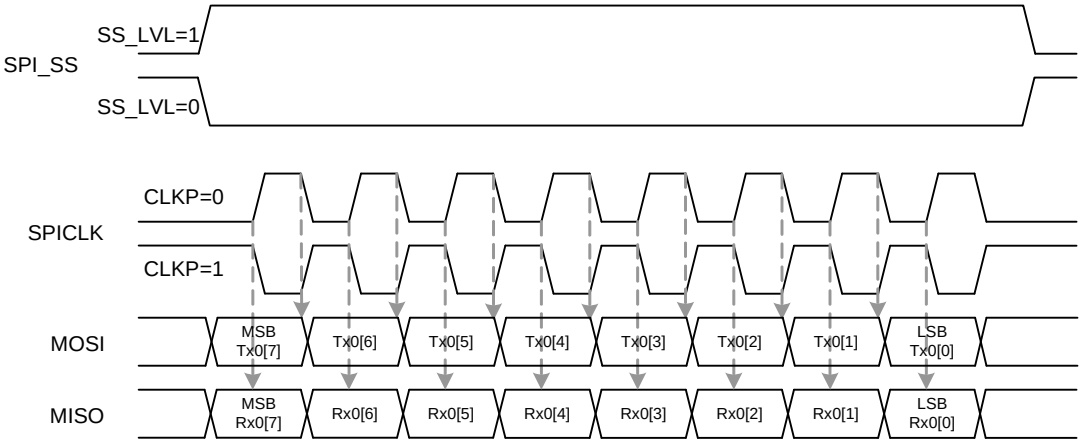


Figure 1 SPI Controller Block Diagram

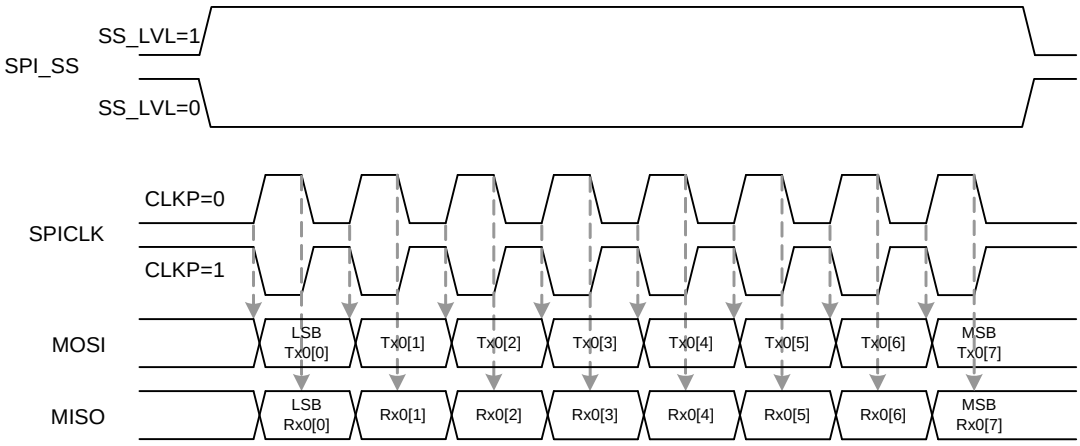
1.3 SPI Timing Diagram (Master/Slave)



Master Mode: CNTRL[SLVAE]=0, CNTRL[LSB]=0, CNTRL[Tx_NUM]=0x0, 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

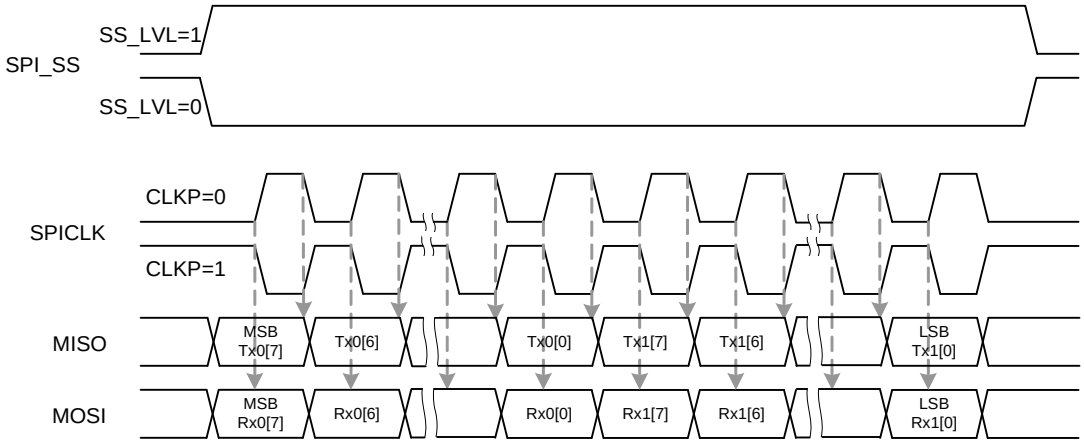
Figure 2 MICROWIRE/SPI Timing (Master)



Master Mode: CNTRL[SLVAE]=0, CNTRL[LSB]=1, CNTRL[Tx_NUM]=0x0, 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

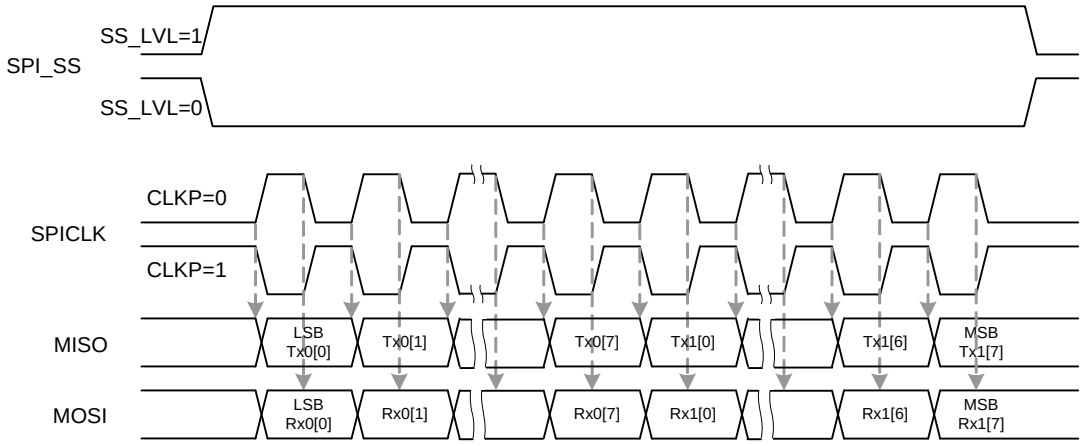
Figure 3 Alternate Phase SCLK Clock Timing (Master)



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

Figure 4 MICROWIRE /SPI Timing (Slave)



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

Figure 5 Alternate Phase SCLK Clock Timing (Slave)

2 HOW TO PROGRAM SPI

2.1 PROGRAM FLOW OF SPI

1. Set the **SPIx_EN** bit in **APBCLK** register to enable SPIx clock source and control **SPIx_RST** bit in **SPI0_RST** register to reset SPIx modular.
2. Select **SPIx_SS0**, **SPIx_CLK**, **SPI0_MISOx** and **SPIx_MOSI0** in **GPxMFP** register to enable Multi-function pin for SPI interface.
3. Select **DIVIDER** bit in **DIVIDER** register to define SPIx clock = HCLK/(DIVIDER+1).
4. Select **SLAVE** bit in **CNTRL** register to define slave or master mode and then select **LSB**, **CLKP**, **TX_NEG** and **RX_NEG** in **CNTRL** register to define clock timing of SPI.
5. Select **ASS**, **SS_LVL** and **SSR** bit in **SSR** register to define SPIx_SS pin status.

2.2 Sample code

2.2.1 AD7793.c

This sample code supports two operation mode to control SPI. First, automatic slave select mode can automatic active or inactive SPI_SS pin after setting SPI0->CNTRL.GO_BUSY if defining SPI_EN_ASS. Or else the SPI_SS pin is controlled by SPI0->SSR.SSR at manual slave select mode.

```
#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

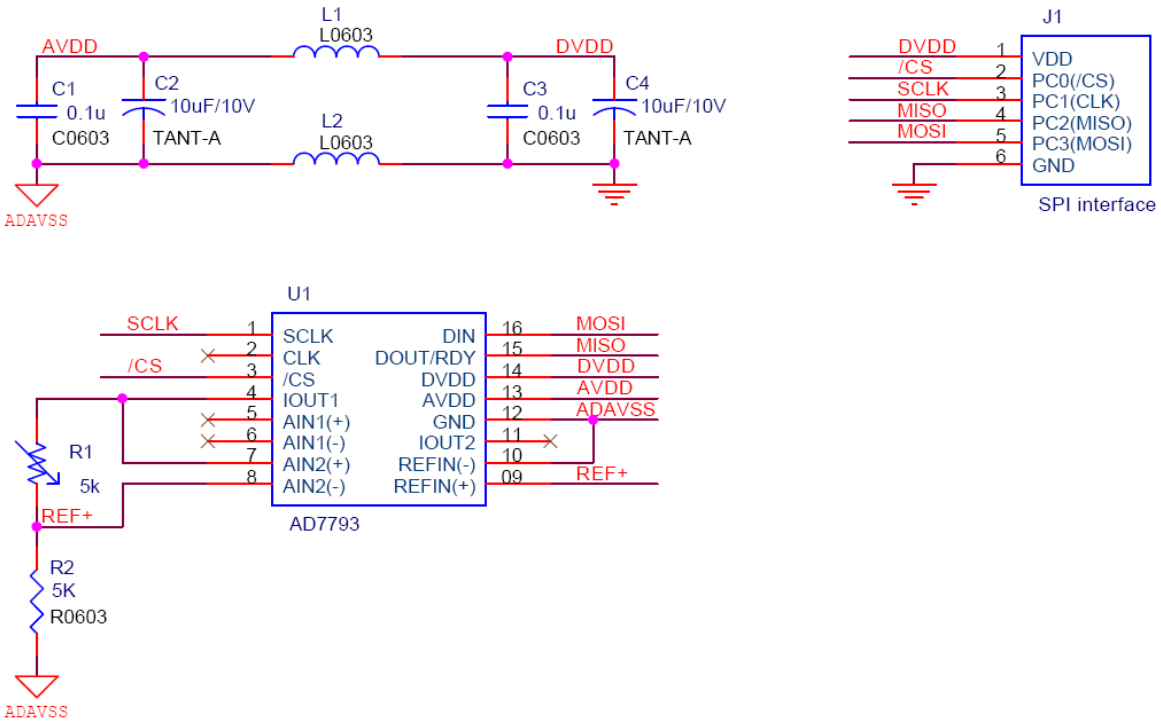
Polling AD7793 status and display the result data of ADC.

```
#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 SCHEMATIC


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 REVISION HISTORY

REV.	DATE	DESCRIPTION
1.00	March 1, 2010	1. Initially issued.
1.01	April 8, 2010	1. Remove register description

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.