SPI Driver Sample Code Reference Guide

V1.00.001

Publication Release Date: Sep. 2011

Support Chips:

ISD9160

Support Platforms:

NuvotonPlatform_Keil

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1 Introduction

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-directional interface. The ISD91XX series contains an SPI controller performing a serial-to-parallel conversion of data received from an external device, and a parallel-to-serial conversion of data transmitted to an external device. The SPI controller can be set as a master with up to 2 slave select (SSB) address lines to access two slave devices; it also can be set as a slave controlled by an off-chip master device. The Smpl_DrvSPI.c and Smpl_DrvSPI_Tx.c demo the usage of ISD9160 SPI IP.

1.1 Feature

- Demo SPI master and slave mode.
- Demo burst, two port mode.
- Demo byte endian and length setting.
- Demo interrupt method.

1.2 Limitation

- Max SPI clock is 12MHz when system clock is 49.152MHz.
- Max SPI transmit length is 32 bit.
- Burst mode support maximum of 2 words

2 Block Diagram

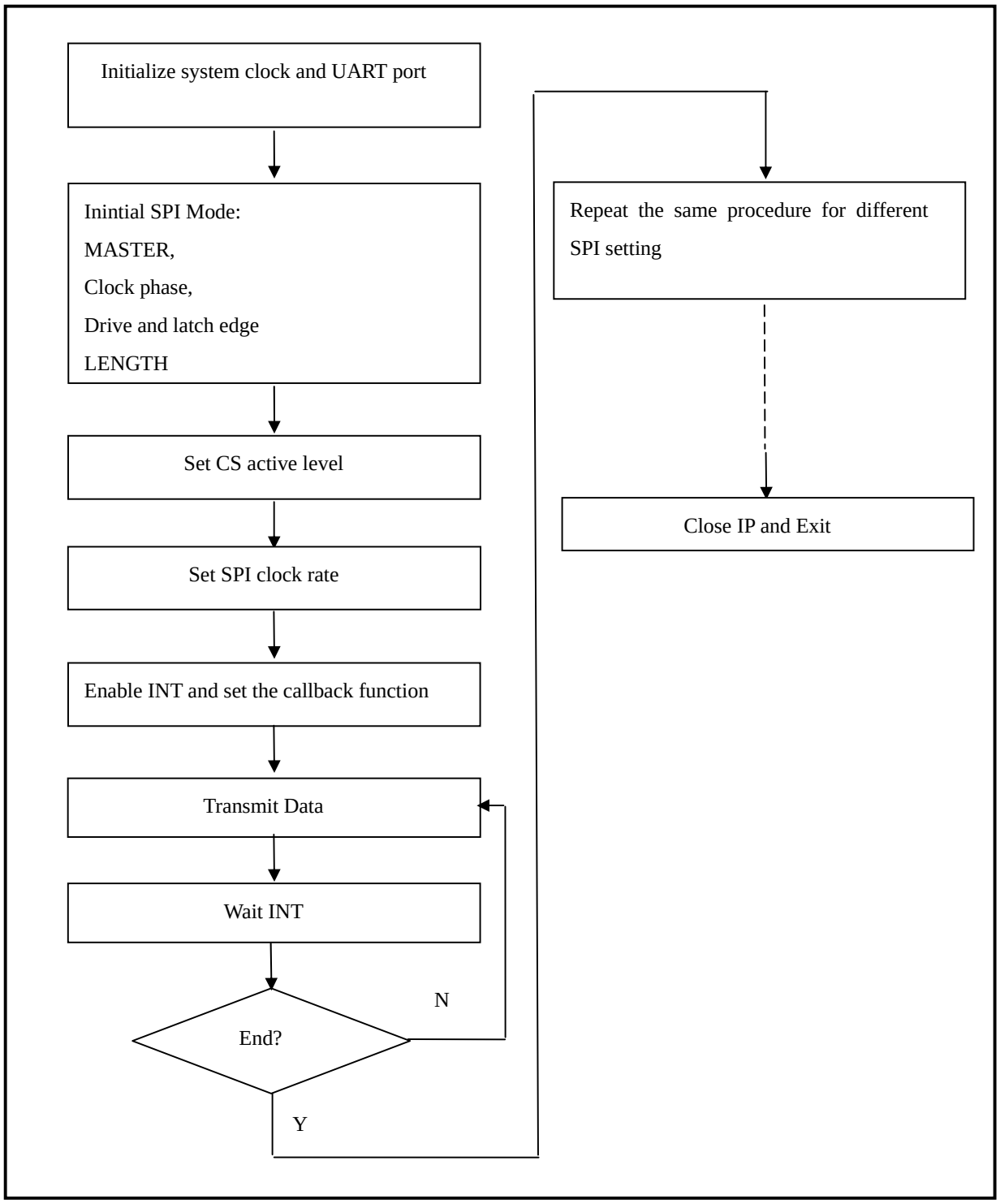

Figure 2-1: Main flow of sample code

3 Calling Sequence

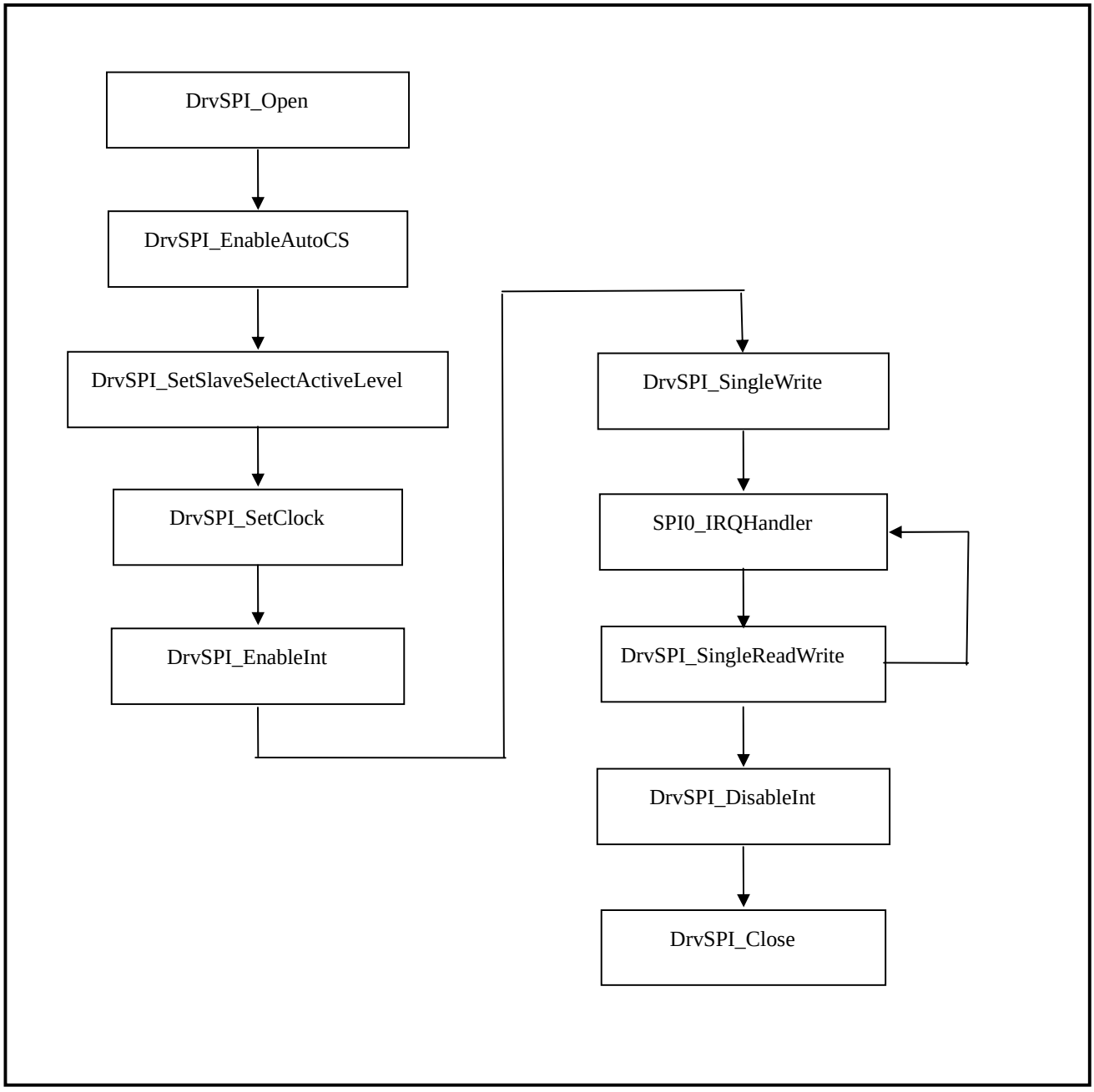

Figure 3-1: API call flow

3.1 API Usage Reference

- SPI Driver User Guide.pdf

4 Code Section

4.1 Constant and Variable

There is one interrupt flag “SPI0_INT_Flag”used in interrupt callback function. Define WAIT_FOR_NEXT will hold program when each section SPI demo completes. The TEST_COUNT decides transfer buffer length. TEST_CYCLE decides how many transfer cycles are repeated. The buffer au32SourceData hold source data to transfer. The buffer au32DestinationData holds the data received.

```
#define WAIT_FOR_NEXT
#define TEST_COUNT 255
#define TEST_CYCLES 255

uint32_t au32SourceData[TEST_COUNT];
uint32_t au32DestinationData[TEST_COUNT];
volatile uint32_t SPI0_INT_Flag;
```

4.2 Main Function

Select internal oscillator

```
/* Unlock the protected registers */
UNLOCKREG();
SYSCLK->CLKSEL0.HCLK_S = 0;
/* Select HCLK source as OSC49.152MHz */
SYSCLK->CLKDIV.HCLK_N    = 0;      /* Select no division                  */
SYSCLK->CLKSEL0.OSCFSel = 0;       /* 1= 32.768MHz, 0=49.152MHz */
LOCKREG();
/* HCLK clock frequency = HCLK clock source / (HCLK_N + 1) */
DrvSYS_SetClockDivider(E_SYS_HCLK_DIV, 0);
```

Set up UART port to 115200 baud, 8 bit, no parity, stop bit 1..

```
DrvGPIO_InitFunction(FUNC_UART0);

param.u32BaudRate        = 115200;
param.u8cDataBits        = DRVUART_DATABITS_8;
param.u8cStopBits        = DRVUART_STOPBITS_1;
param.u8cParity          = DRVUART_PARITY_NONE;
param.u8cRxTriggerLevel  = DRVUART_FIFO_1BYTES;
param.u8TimeOut          = 0;
DrvUART_Open(UART_PORT0, &param);
```

Set the source data and clear the destination buffer

```
for(u32DataCount=0; u32DataCount<TEST_COUNT; u32DataCount++)
{
au32SourceData[u32DataCount] = (u32DataCount & 0xff) +
                               (((u32DataCount & 0xff)+1) << 8 ) +
                               (((u32DataCount & 0xff)+2) <<16 ) +
                               (((u32DataCount & 0xff)+3) <<24 ) ;
                              au32DestinationData[u32DataCount] = 0;
}
```

Configure SPI port parameter. Master mode, clock phase, data drive and latch relation is Type1, transmit length is 32 bits. CS pin active low. SCLK clock rate is 2.5MHz. The INT callback function is SPI0_Callback.

```
/* Configure SPI0 as a master, 32-bit transaction */
DrvSPI_Open(eDRVSPI_PORT0, eDRVSPI_MASTER, eDRVSPI_TYPE1, 32);
/* Enable the automatic slave select function of SS0. */
DrvSPI_EnableAutoCS(eDRVSPI_PORT0, eDRVSPI_SS0);
/* Set the active level of slave select. */
DrvSPI_SetSlaveSelectActiveLevel(eDRVSPI_PORT0, eDRVSPI_ACTIVE_LOW_FALLING);
/* Enable the automatic slave select function and set the specified slave select pin. */
DrvSPI_EnableAutoCS(eDRVSPI_PORT0, eDRVSPI_SS0);
/* SPI clock rate 2.5MHz */
DrvSPI_SetClock(eDRVSPI_PORT0, 2500000, 0);
/* Enable the SPI0 interrupt and install the callback function. */
DrvSPI_EnableInt(eDRVSPI_PORT0, SPI0_Callback, 0);
```

Show the SPI driver version, bit length, clock rate and wait for user input.

```
printf("\n\n");
printf("+-----------------------------------------------------------------------+\n");
printf("|                    SPI Driver Sample Code                             |\n");
printf("|                                                                       |\n");
printf("+-----------------------------------------------------------------------+\n");
printf("\n");
printf("SPI Driver version: %x\n", DrvSPI_GetVersion());
printf("Configure SPI0 as a master. Loopback MISO to MOSI for test.\n");
printf("Bit length of a transaction: 32\n");
printf("SPI clock rate: %d Hz\n", DrvSPI_GetClock1(eDRVSPI_PORT0));
printf("The I/O connection for SPI0 loopback:\n ");
printf("       SPI0_MISO0(GPA3) <--> SPI0_MISO0(GPA0)\n\n");
printf("Please connect loopback, and press any key to start transmission.\n");
getchar();
```

Call SpiCheck to transfer data.

```
SpiCheck(TEST_CYCLES);
```

Transfer data in SpiCheck. Use DrvSPI_SingleWrite to transfer first data. Then wait INT and transfer other data.

```
/* write the first data of source buffer to Tx register of SPI0. And start transmission. */
DrvSPI_SingleWrite(eDRVSPI_PORT0, &au32SourceData[0]);

while(1){
        if(SPI0_INT_Flag==1){
                SPI0_INT_Flag = 0;
                if(u32DataCount<(TEST_COUNT-1))     {
                /* Read the previous retrived data and trigger next transfer. */
                DrvSPI_SingleReadWrite(eDRVSPI_PORT0, &au32DestinationData[u32DataCount],
                                                        au32SourceData[u32DataCount+1]);
                u32DataCount++;
                }else{
                /* Just read the previous retrived data but trigger next transfer,
                                                because this is the last transfer. */
                DrvSPI_DumpRxRegister(eDRVSPI_PORT0, &au32DestinationData[u32DataCount], 1);
                break;
                }
}
```

Call DrvSPI_Close to close SPI port.

```
DrvSPI_Close(eDRVSPI_PORT0);
```

4.3 SPI0_IRQHandler

In SPI Driver, SPI0_IRQHandler will be executed when SPI interrupt occurs. The callback function is provided to execute user specified action.

```
void SPI0_IRQHandler(void)
{
        // write '1' to clear SPI0 interrupt flag
        SPI0->CNTRL.IF = 1;

        if(g_sSpiHandler[0].pfncallback != NULL)
        {
            g_sSpiHandler[0].pfncallback(g_sSpiHandler[0].u32userData);
        }
}
```

To set up the callback function, use DrvSPI_EnableInt to hook up the SPI0_Callback function.

```
DrvSPI_EnableInt(eDRVSPI_PORT0, SPI0_Callback, 0);
```

User specifies action when SPI interrupt occur. Here only SPI0_INT_Flag is set. The transfer routine will use this flag to check SPI transfer done or not by polling this flag.

```
void SPI0_Callback(uint32_t u32UserData)
{
        SPI0_INT_Flag = 1;
}
```

5 Execution Environment Setup and Result

- Prepare ISD9160 EVB
- Connect UART to user’s host UART port by female to female cable.
- Compile it under Keil environment.
- Execute the program and the result will show on host console window.

6 Revision History

Version	Date	Description
V1.00.01	Sep. 2011	Created