SPI Flash
Sample Code
Programming Guide

V1.00.001

Publication Release Date: Aug. 2011

Support Chips:
ISD9160

Support Platforms:
Nuvoton

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

This document introduces how to access Serial Flash Memory with SPI flash library API.

1.1. Feature

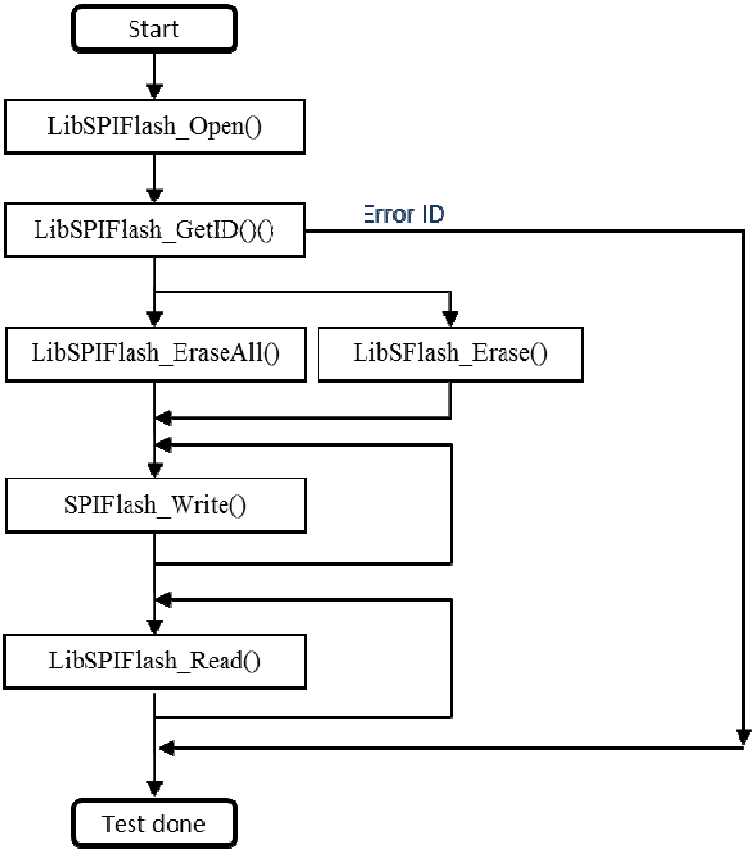
- Set test pattern and write pattern data to serial flash memory.
- Check the data consistency.

1.2. Limitation

- Use SPI interface as master to control the serial flash memory.
- Support W25Q10, W25Q20, W25Q40, W25Q80, W25Q16, and W25Q32.
- The size of test data is limited by the size of SPI Flash.

2. Calling Flow

2.1. Blocking APIs calling flow



2.2. API Usage Reference

- SPI Driver Reference Guide.pdf
- SPI Flash Reference Guide.pdf

3. Code Section – Smpl_SPIFlash.c

3.1. Main function

The main functions of Smpl_SPIFlash.c are

- Configure system clock
- LDO setting for SPI-flash power
- Configure GPIO pins to become SPI interface pins
- Initial SPI flash library
- Get SPI flash ID
- Write function enable
- Demo calling APIs
  - Whole chip erasing or 4KB sector erase
  - Write data from buffer
  - Read data to buffer and compare read/write buffers
- Test done

Configure system clock

The following codes are used to configure system clocks

```
UNLOCKREG();
SYSCLK->PWRCON.OSC49M_EN = 1;
SYSCLK->CLKSEL0.HCLK_S = 0; /* Select HCLK source as 48MHz */
SYSCLK->CLKDIV.HCLK_N  = 0;    /* Select no division        */
SYSCLK->CLKSEL0.OSCFSel = 0;  /* 1= 32MHz, 0=48MHz */
```

System clock register is normally write-protected. Before system clock setting, it needs "UNLOCKREG()" to unlock the protected register. After system clock configured, we must call "LOCKREG()" to lock these protected register.

LDO setting

ISD9160 operation voltage is 2.4V~5.5V, but SPI-flash is 2.7V~3.6V. ISD9160 is built-in the LDO to provide the power for SPI-flash. Programmer needs to consider the LDO on/off based on the target board circuit.

```
void LdoOn(void)
{
     SYSCLK->APBCLK.ANA_EN=1;
     ANA->LDOPD.PD=0;
```

```
        ANA->LDOSET=3;      //Set LDO output @ 3.3V
    }
```

Configure GPIO pins to become SPI interface pins

SPI interface is shared with GPIO. Before calling LibSPIFlash_Open, programmer should enable the GPIO pins as the SPI interface pins. These codes are in the function SpiFlashInit().

```
//Enable SPI shared function pins
    SYS->GPA_ALT.GPA0           = 1;   //MOSIO
    SYS->GPA_ALT.GPA2           = 1; // SSB0
    SYS->GPA_ALT.GPA1           = 1; // SCLK
    SYS->GPA_ALT.GPA3           = 1; // MISO0

//Reset the ISD9160 SPI
    SYSCLK->APBCLK.SPI0_EN      = 1;
    SYS->IPRSTC2.SPI0_RST       = 1;
    SYS->IPRSTC2.SPI0_RST       = 0;
```

Initial SPI flash library

Users must initial the SPI flash library for each SPI flash mounted on SPI interface,. The initial procedures are doing in the “SPIFlash_Open()”. The following API is called in SpiFlashInit().

```
LibSPIFlash_Open(&g_SPIFLASH);
```

Get SPI flash ID

After calling “LibSPIFlash_Open()”, users can call “LibSPIFlash_GetID()” to get the SPI flash manufacture ID & device ID. User can use them to judge whether the SPI-flash initialization successful or not.

```
    u32temp= LibSPIFlash_GetID(&g_SPIFLASH);
    printf("\n Manufacture ID: %2X \n", u32temp>>16);
    printf("\n Device ID: %4X \n", u32temp&0xFFFF);
    if ((u32temp==0)||(u32temp==0xFFFFFF))
        goto Error;
```

Write function enable

SPI flash is designed with normally write protected. Before each erase or write command, programmer needs to send the write enable command. In each SPI-flash library API, it sends the command before operation. User does not need to care again.

Demo calling APIs

Before write data into SPI flash, user has to erase the SPI-flash. User can select whole chip erase or 4KB erase based on application requirement.

- Erase whole chip
  Calling "SPIFlash_EraseAll()" can erase whole chip. After whole chip erased and SPI flash is in ready state, program will return from this API.

```
    LibSPIFlash_EraseAll(&g_SPIFLASH);
```

- Erase a 4KBytes sector
  This is the minimum size for erase. Programmer has to read the remained valid data then write back to flash if the programming data is less than a sector size.

```
    iRet=LibSPIFlash_Erase(&g_SPIFLASH, u32EraseStartAddr, u32EraseLength);
```

  The size parameter "u32EraseLength" should be the multiple of 4096. The LibSPIFlash_Erase() will not erase the sector if it is not enough than a sector.

- Write data
  For writing data to SPI Flash, programmer can call "LibSPIFlash_Write()". It uses page (256B) program to write the SPI flash, the size parameter should be the multiple of 256.
  Because this APIs need to align data buffer on 4 byte alignment, therefore we must use the following codes to align buffer on 4 bytes.

```
__align(4)     uint8_t WriteBuffer[BUFFER_SAMPLECOUNT];
```

```
    LibSPIFlash_Write(&g_SPIFLASH, RECORD_START_ADDR,  (uint32_t *) WriteBufferAddr,
BUFFER_SAMPLECOUNT);
```

  After all data is written and SPI flash is in ready state, program will return from this API.

- Read data
  For reading data from SPI Flash, programmer can call "LibSPIFlash_FastRead()" or "LibSPIFlash_Fast()". Suggest normally use "LibSPIFlash_FastRead()" to get better reading performance.
  Because this APIs need to align data buffer on 4 byte alignment, therefore we must use the following codes to align buffer on 4 bytes.

```
__align(4)     uint8_t ReadBuffer[BUFFER_SAMPLECOUNT];
```

```
    LibSPIFlash_FastRead(&g_SPIFLASH, RECORD_START_ADDR,  (uint32_t *)ReadBufferAddr,
BUFFER_SAMPLECOUNT);
```

4. Revision History

Version	Date	Description
V1.00.001	Aug. 29, 2010	• Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

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