一、基本概念（详细内容见 st 网站 stm32 应用笔记 AN2784）

1．FSMC 配置

控制一个 NOR 闪存存储器，需要 FSMC 提供下述功能：

●选择合适的存储块映射 NOR 闪存存储器：共有 4 个独立的存储块可以用于与 NOR 闪存、SRAM 和 PSRAM 存储器接口，每个存储块都有一个专用的片选管脚。

●使用或禁止地址/数据总线的复用功能。

●选择所用的存储器类型：NOR 闪存、SRAM 或 PSRAM。

●定义外部存储器的数据总线宽度：8 或 16 位。

●使用或关闭同步 NOR 闪存存储器的突发访问模式。

●配置等待信号的使用：开启或关闭，极性设置，时序配置。

●使用或关闭扩展模式：扩展模式用于访问那些具有不同读写操作时序的存储器。

因为 NOR 闪存/SRAM 控制器可以支持异步和同步存储器，用户只须根据存储器的参数配置使用到的参数。

FSMC 提供了一些可编程的参数，可以正确地与外部存储器接口。依存储器类型的不同，有些参数是不需要的。

当使用一个外部异步存储器时，用户必须按照存储器的数据手册给出的时序数据，计算和设置下列参数：

●ADDSET：地址建立时间

●ADDHOLD：地址保持时间

●DATAST：数据建立时间

●ACCMOD：访问模式 这个参数允许 FSMC 可以灵活地访问多种异步的静态存储器。共有 4 种扩展模式允许以不同的时序分别读写存储器。 在扩展模式下，FSMC_BTR 用于配置读操作，FSMC_BWR 用于配置写操作。(译注：如果读时序与写时序相同，只须使用 FSMC_BTR 即可。)

如果使用了同步的存储器，用户必须计算和设置下述参数：

●CLKDIV：时钟分频系数

●DATLAT：数据延时

如果存储器支持的话，NOR 闪存的读操作可以是同步的，而写操作仍然是异步的。

当对一个同步的 NOR 闪存编程时，存储器会自动地在同步与异步之间切换；因此，必须正确地设置所有的参数。

2．时序计算

如上所述，对于异步 NOR 闪存存储器或类似的存储，有不同的访问协议。首先要确定对特定存储器所需要使用的操作协议，选择的依据是不同的控制信号和存储器在读或写操作中的动作。

对于异步 NOR 闪存存储器，需要使用模式 2 协议。如果要使用的存储器有 NADV 信号，则需要使用扩展的模式 B 协议。

我们将使用模式 2 操作 M29W128FL，不使用任何扩展模式，即读和写操作的时序是一样的。这时 NOR 闪存控制器需要 3 个时序参数：ADDSET、DATAST 和 ADDHOLD。

需要根据 NOR 闪存存储器的特性和 STM32F10xxx 的时钟 HCLK 来这些计算参数。

基于图 3 和图 4 的 NOR 闪存存储器访问时序，可以得到下述公式：

写或读访问时序是存储器片选信号的下降沿与上升沿之间的时间，这个时间可以由 FSMC 时序参数的函数计算得到：

写/读访问时间 = ((ADDSET + 1) + (DATAST + 1)) × HCLK

在写操作中，DATAST 用于衡量写信号的下降沿与上升沿之间的时间参数：

写使能信号从低变高的时间 = tWP = DATAST × HCLK

为了得到正确的 FSMC 时序配置，下列时序应予以考虑：

●最大的读/写访问时间

●不同的 FSMC 内部延迟

●不同的存储器内部延迟

因此得到：

((ADDSET + 1) + (DATAST + 1)) × HCLK = max (tWC, tRC)

DATAST × HCLK = tWP

DATAST 必须满足：

DATAST = (tAVQV + tsu(Data_NE) + tv(A_NE))/HCLK – ADDSET – 4

二、程序分析

```
 /*-- FSMC Configuration ------------------------------------------------------*/
  p.FSMC_AddressSetupTime = 0x05;     /*ADDSET   地址建立时间*/
  p.FSMC_AddressHoldTime = 0x00;    /*ADDHOLD  地址保持时间*/
  p.FSMC_DataSetupTime = 0x07;     /*DATAST  数据建立时间*/
  p.FSMC_BusTurnAroundDuration = 0x00;   /*BUSTURN  总线返转时间*/
  p.FSMC_CLKDivision = 0x00;      /*CLKDIV  时钟分频*/
  p.FSMC_DataLatency = 0x00;     /*DATLAT  数据保持时间*/
  p.FSMC_AccessMode = FSMC_AccessMode_B;   /*访问模式*/
/*NOR/SRAM 的存储块，共 4 个选项*/
  FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM2;
/*是否选择地址和数据复用数据线*/
  FSMC_NORSRAMInitStructure.FSMC_DataAddressMux =
FSMC_DataAddressMux_Disable;
/*连接到相应存储块的外部存储器类型*/
  FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_NOR;
/*存储器数据总线宽度*/
  FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth =
FSMC_MemoryDataWidth_16b;
/*使能或关闭同步 NOR 闪存存储器的突发访问模式设置是否使用迸发访问模式(应该就是连续读写模式吧)*/
  FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode =
FSMC_BurstAccessMode_Disable;
/*设置 WAIT 信号的有效电平*/
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity =
FSMC_WaitSignalPolarity_Low;
```

```
 /*设置是否使用环回模式*/
  FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;
/*设置 WAIT 信号有效时机*/
  FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive =
FSMC_WaitSignalActive_BeforeWaitState;
/*设定是否使能写操作*/
  FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
/*设定是否使用 WAIT 信号*/
  FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
/*使能或关闭扩展模式，扩展模式用于访问具有不同读写操作时序的存储器，设定是否使用单
独的写时序*/
  FSMC_NORSRAMInitStructure.FSMC_ExtendedMode =
FSMC_ExtendedMode_Disable;
/*设定是否使用异步等待信号*/
  FSMC_NORSRAMInitStructure.FSMC_AsyncWait = FSMC_AsyncWait_Disable;
/*设定是否使用迸发写模式*/
  FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
 /*设定读写时序*/
  FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;     //
  FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;     //

  FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);        //
  /* Enable FSMC Bank1_NOR Bank */
  FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM2, ENABLE);        //
}
```

三、例程：STM32 读写外 NOR FLASH 存储器 39VF1601

```
1. fsmc_nor..c
/******************** (C) COPYRIGHT 2008 STMicroelectronics ********************
* File Name          : fsmc_nor.c
* Author             : MCD Application Team
* Version            : V2.0.1
* Date               : 06/13/2008
* Description        : This file provides a set of functions needed to drive the
*                      M29W128FL, M29W128GL and S29GL128P NOR memories mounted
*                      on STM3210E-EVAL board.
*******************************************************************************
* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.
* AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT,
* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE
* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING
```

```
* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
*******************************************************************************/
/* Includes ------------------------------------------------------------------*/
#include "fsmc_nor.h"
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define Bank1_NOR2_ADDR       ((u32)0x64000000)
/* Delay definition */
#define BlockErase_Timeout    ((u32)0x00A00000)
#define ChipErase_Timeout     ((u32)0x30000000)
#define Program_Timeout       ((u32)0x00001400)
/* Private macro -------------------------------------------------------------*/
#define ADDR_SHIFT(A) (Bank1_NOR2_ADDR + (2 * (A)))
#define NOR_WRITE(Address, Data)  (*(vu16 *)(Address) = (Data))
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/*******************************************************************************
* Function Name  : FSMC_NOR_Init
* Description    : Configures the FSMC and GPIOs to interface with the NOR memory.
*                  This function must be called before any write/read operation
*                  on the NOR.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void FSMC_NOR_Init(void)
{
    FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
    FSMC_NORSRAMTimingInitTypeDef  p;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE |
                           RCC_APB2Periph_GPIOF | RCC_APB2Periph_GPIOG, ENABLE);
    /*-- GPIO Configuration ------------------------------------------------------*/
    /* NOR Data lines configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_8 | GPIO_Pin_9 |
                                  GPIO_Pin_10 | GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
                                  GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 |
```

```
                                GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    /* NOR Address lines configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
                                GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_12 | GPIO_Pin_13 |
                                GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_Init(GPIOF, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
                                GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;
    GPIO_Init(GPIOG, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    /* NOE and NWE configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
    GPIO_Init(GPIOD, &GPIO_InitStructure);
    /* NE2 configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_Init(GPIOG, &GPIO_InitStructure);
    /*-- FSMC Configuration ------------------------------------------------------*/
    p.FSMC_AddressSetupTime = 0x05;     /*ADDSET  地址建立时间*/
    p.FSMC_AddressHoldTime = 0x00;    /*ADDHOLD 地址保持时间*/
    p.FSMC_DataSetupTime = 0x07;     /*DATAST 数据建立时间*/
    p.FSMC_BusTurnAroundDuration = 0x00;   /*BUSTURN 总线返转时间*/
    p.FSMC_CLKDivision = 0x00;      /*CLKDIV 时钟分频*/
    p.FSMC_DataLatency = 0x00;     /*DATLAT 数据保持时间*/
    p.FSMC_AccessMode = FSMC_AccessMode_B;   /*访问模式*/
    FSMC_NORSRAMInitStructure.FSMC_Bank = FSMC_Bank1_NORSRAM2;    // NOR/SRAM 的存储块，共 4 个选项
    FSMC_NORSRAMInitStructure.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;  //  是都选择地址和数据复用数据线
    FSMC_NORSRAMInitStructure.FSMC_MemoryType = FSMC_MemoryType_NOR;   // 连接到相应存储块的外部存储器类型
    FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;  //存储器数据总线宽度
    FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode = FSMC_BurstAccessMode_Disable;   // 使能或关闭同步 NOR 闪存存储器的突发访问模式
    //设置是否使用迸发访问模式(应该就是连续读写模式吧)
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;   // 设置 WAIT 信号的有效电平
```

```
    FSMC_NORSRAMInitStructure.FSMC_WrapMode = FSMC_WrapMode_Disable;    // 设置是否使用环回模式
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState; // 设置 WAIT 信号有效时机
    FSMC_NORSRAMInitStructure.FSMC_WriteOperation = FSMC_WriteOperation_Enable;  //   设定是否使能写操作
    FSMC_NORSRAMInitStructure.FSMC_WaitSignal = FSMC_WaitSignal_Disable;   //    设定是否使用 WAIT 信号
    FSMC_NORSRAMInitStructure.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;  // 使能或关闭扩展模式，扩展模式用于访问具有不同读写操作时序的存储器
    // 设定是否使用单独的写时序
    FSMC_NORSRAMInitStructure.FSMC_AsyncWait = FSMC_AsyncWait_Disable;  //  设定是否使用异步等待信号
    FSMC_NORSRAMInitStructure.FSMC_WriteBurst = FSMC_WriteBurst_Disable;  //  设定是否使用迸发写模式
    FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = &p;    //  设定读写时序
    FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct = &p;    //

    FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);        //
    /* Enable FSMC Bank1_NOR Bank */
    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM2, ENABLE);        //
}
/*******************************************************************************
* Function Name  : FSMC_NOR_ReadID
* Description    : Reads NOR memory's Manufacturer and Device Code.
* Input          : - NOR_ID: pointer to a NOR_IDTypeDef structure which will hold
*                    the Manufacturer and Device Code.
* Output         : None
* Return         : None
*******************************************************************************/
void FSMC_NOR_ReadID(NOR_IDTypeDef* NOR_ID)
{
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x0090);
    NOR_ID->Manufacturer_Code = *(vu16 *) ADDR_SHIFT(0x0000);
    NOR_ID->Device_Code1 = *(vu16 *) ADDR_SHIFT(0x0001);
    NOR_ID->Device_Code2 = *(vu16 *) ADDR_SHIFT(0x000E);
    NOR_ID->Device_Code3 = *(vu16 *) ADDR_SHIFT(0x000F);
}
/*******************************************************************************
* Function Name  : FSMC_NOR_EraseBlock
```

```
* Description    : Erases the specified Nor memory block.
* Input          : - BlockAddr: address of the block to erase.
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
*******************************************************************************/
NOR_Status FSMC_NOR_EraseBlock(u32 BlockAddr)
{
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x0080);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE((Bank1_NOR2_ADDR + BlockAddr), 0x30);
    return(FSMC_NOR_GetStatus(BlockErase_Timeout));
}
/*******************************************************************************
* Function Name  : FSMC_NOR_EraseChip
* Description    : Erases the entire chip.
* Input          : None
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
*******************************************************************************/
NOR_Status FSMC_NOR_EraseChip(void)
{
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x0080);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x0010);
    return(FSMC_NOR_GetStatus(ChipErase_Timeout));
}
/*******************************************************************************
* Function Name  : FSMC_NOR_WriteHalfWord
* Description    : Writes a half-word to the NOR memory.
* Input          : - WriteAddr : NOR memory internal address to write to.
*                  - Data : Data to write.
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
```

```
*******************************************************************************/
NOR_Status FSMC_NOR_WriteHalfWord(u32 WriteAddr, u16 Data)
{
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00A0);
    NOR_WRITE((Bank1_NOR2_ADDR + WriteAddr), Data);
    return(FSMC_NOR_GetStatus(Program_Timeout));
}
/*******************************************************************************
* Function Name  : FSMC_NOR_WriteBuffer
* Description    : Writes a half-word buffer to the FSMC NOR memory.
* Input          : - pBuffer : pointer to buffer.
*                  - WriteAddr : NOR memory internal address from which the data
*                    will be written.
*                  - NumHalfwordToWrite : number of Half words to write.
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
*******************************************************************************/
NOR_Status FSMC_NOR_WriteBuffer(u16* pBuffer, u32 WriteAddr, u32 NumHalfwordToWrite)
{
    NOR_Status status = NOR_ONGOING;
    do
    {
        /* Transfer data to the memory */
        status = FSMC_NOR_WriteHalfWord(WriteAddr, *pBuffer++);
        WriteAddr = WriteAddr + 2;
        NumHalfwordToWrite--;
    }
    while((status == NOR_SUCCESS) && (NumHalfwordToWrite != 0));

    return(status);
}
/*******************************************************************************
* Function Name  : FSMC_NOR_ProgramBuffer
* Description    : Writes a half-word buffer to the FSMC NOR memory. This function
*                  must be used only with S29GL128P NOR memory.
* Input          : - pBuffer : pointer to buffer.
*                  - WriteAddr: NOR memory internal address from which the data
*                    will be written.
*                  - NumHalfwordToWrite: number of Half words to write.
```

```
*                    The maximum allowed value is 32 Half words (64 bytes).
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
*******************************************************************************/
NOR_Status FSMC_NOR_ProgramBuffer(u16* pBuffer, u32 WriteAddr, u32 NumHalfwordToWrite)
{
    u32 lastloadedaddress = 0x00;
    u32 currentaddress = 0x00;
    u32 endaddress = 0x00;
    /* Initialize variables */
    currentaddress = WriteAddr;
    endaddress = WriteAddr + NumHalfwordToWrite - 1;
    lastloadedaddress = WriteAddr;
    /* Issue unlock command sequence */
    NOR_WRITE(ADDR_SHIFT(0x005555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    /* Write Write Buffer Load Command */
    NOR_WRITE(ADDR_SHIFT(WriteAddr), 0x0025);
    NOR_WRITE(ADDR_SHIFT(WriteAddr), (NumHalfwordToWrite - 1));
    /* Load Data into NOR Buffer */
    while(currentaddress <= endaddress)
    {
        /* Store last loaded address & data value (for polling) */
        lastloadedaddress = currentaddress;

        NOR_WRITE(ADDR_SHIFT(currentaddress), *pBuffer++);
        currentaddress += 1;
    }
    NOR_WRITE(ADDR_SHIFT(lastloadedaddress), 0x29);

    return(FSMC_NOR_GetStatus(Program_Timeout));
}
/*******************************************************************************
* Function Name  : FSMC_NOR_ReadHalfWord
* Description    : Reads a half-word from the NOR memory.
* Input          : - ReadAddr : NOR memory internal address to read from.
* Output         : None
* Return         : Half-word read from the NOR memory
*******************************************************************************/
u16 FSMC_NOR_ReadHalfWord(u32 ReadAddr)
{
```

```
    NOR_WRITE(ADDR_SHIFT(0x005555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x002AAA), 0x0055);
    NOR_WRITE((Bank1_NOR2_ADDR + ReadAddr), 0x00F0 );
    return(*(vu16 *)((Bank1_NOR2_ADDR + ReadAddr)));
}
/*******************************************************************************
* Function Name  : FSMC_NOR_ReadBuffer
* Description    : Reads a block of data from the FSMC NOR memory.
* Input          : - pBuffer : pointer to the buffer that receives the data read
*                    from the NOR memory.
*                  - ReadAddr : NOR memory internal address to read from.
*                  - NumHalfwordToRead : number of Half word to read.
* Output         : None
* Return         : None
*******************************************************************************/
void FSMC_NOR_ReadBuffer(u16* pBuffer, u32 ReadAddr, u32 NumHalfwordToRead)
{
    NOR_WRITE(ADDR_SHIFT(0x05555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x02AAA), 0x0055);
    NOR_WRITE((Bank1_NOR2_ADDR + ReadAddr), 0x00F0);
    for(; NumHalfwordToRead != 0x00; NumHalfwordToRead--) /* while there is data to read */
    {
        /* Read a Halfword from the NOR */
        *pBuffer++ = *(vu16 *)((Bank1_NOR2_ADDR + ReadAddr));
        ReadAddr = ReadAddr + 2;
    }
}
/*******************************************************************************
* Function Name  : FSMC_NOR_ReturnToReadMode
* Description    : Returns the NOR memory to Read mode.
* Input          : None
* Output         : None
* Return         : NOR_SUCCESS
*******************************************************************************/
NOR_Status FSMC_NOR_ReturnToReadMode(void)
{
    NOR_WRITE(Bank1_NOR2_ADDR, 0x00F0);
    return(NOR_SUCCESS);
}
/*******************************************************************************
* Function Name  : FSMC_NOR_Reset
* Description    : Returns the NOR memory to Read mode and resets the errors in
```

```
*                  the NOR memory Status Register.
* Input          : None
* Output         : None
* Return         : NOR_SUCCESS
*******************************************************************************/
NOR_Status FSMC_NOR_Reset(void)
{
    NOR_WRITE(ADDR_SHIFT(0x005555), 0x00AA);
    NOR_WRITE(ADDR_SHIFT(0x002AAA), 0x0055);
    NOR_WRITE(Bank1_NOR2_ADDR, 0x00F0);
    return(NOR_SUCCESS);
}
/*******************************************************************************
* Function Name  : FSMC_NOR_GetStatus
* Description    : Returns the NOR operation status.
* Input          : - Timeout: NOR progamming Timeout
* Output         : None
* Return         : NOR_Status:The returned value can be: NOR_SUCCESS, NOR_ERROR
*                  or NOR_TIMEOUT
*******************************************************************************/
NOR_Status FSMC_NOR_GetStatus(u32 Timeout)
{
    u16 val1 = 0x00, val2 = 0x00;
    NOR_Status status = NOR_ONGOING;
    u32 timeout = Timeout;
    /* Poll on NOR memory Ready/Busy signal -----------------------------------*/
    while((GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6) != RESET) && (timeout > 0))
    {
        timeout--;
    }
    timeout = Timeout;

    while((GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6) == RESET) && (timeout > 0))
    {
        timeout--;
    }

    /* Get the NOR memory operation status ------------------------------------*/
    while((Timeout != 0x00) && (status != NOR_SUCCESS))
    {
        Timeout--;
        /* Read DQ6 and DQ5 */
```

```
        val1 = *(vu16 *)(Bank1_NOR2_ADDR);
        val2 = *(vu16 *)(Bank1_NOR2_ADDR);
        /* If DQ6 did not toggle between the two reads then return NOR_Success */
        if((val1 & 0x0040) == (val2 & 0x0040))
        {
            return NOR_SUCCESS;
        }
        if((val1 & 0x0020) != 0x0020)
        {
            status = NOR_ONGOING;
        }
        val1 = *(vu16 *)(Bank1_NOR2_ADDR);
        val2 = *(vu16 *)(Bank1_NOR2_ADDR);

        if((val1 & 0x0040) == (val2 & 0x0040))
        {
            return NOR_SUCCESS;
        }
        else if((val1 & 0x0020) == 0x0020)
        {
            return NOR_ERROR;
        }
    }
    if(Timeout == 0x00)
    {
        status = NOR_TIMEOUT;
    }
    /* Return the operation status */
    return(status);
}
/******************* (C) COPYRIGHT 2008 STMicroelectronics *****END OF FILE****/
```

2.main.c

```
/******************** (C) COPYRIGHT 2008 STMicroelectronics ********************
* File Name          : main.c
* Author             : MCD Application Team
* Version            : V2.0.1
* Date               : 06/13/2008
* Description        : Main program body
********************************************************************************
* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.
* AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT,
```

```
* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE
* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING
* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
*******************************************************************************/
/* Includes ------------------------------------------------------------------*/
#include "fsmc_nor.h"
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define BUFFER_SIZE        0x400
#define WRITE_READ_ADDR    0x8000
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
GPIO_InitTypeDef GPIO_InitStructure;
ErrorStatus HSEStartUpStatus;
u16 TxBuffer[BUFFER_SIZE];
u16 RxBuffer[BUFFER_SIZE];
u32 WriteReadStatus = 0, Index = 0;
NOR_IDTypeDef NOR_ID;
/* Private function prototypes -----------------------------------------------*/
void RCC_Configuration(void);
void NVIC_Configuration(void);
void Fill_Buffer(u16 *pBuffer, u16 BufferLenght, u32 Offset);
/* Private functions ---------------------------------------------------------*/
/*******************************************************************************
* Function Name  : main
* Description    : Main program.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
int main(void)
{
#ifdef DEBUG
    debug();
#endif
    /* System Clocks Configuration */
    RCC_Configuration();
    /* NVIC Configuration */
    NVIC_Configuration();
    /* PF.06 and PF.07  config to drive LD1 and LD2 ****************************/
    /* Enable GPIOF clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOF, ENABLE);
```

```
/* Configure PF.06 and PF.07 as Output push-pull */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOF, &GPIO_InitStructure);

/* Write/read to/from FSMC SRAM memory  ************************************/
/* Enable the FSMC Clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);
/* Configure FSMC Bank1 NOR/SRAM2 */
FSMC_NOR_Init();

/* Read NOR memory ID */
FSMC_NOR_ReadID(&NOR_ID);
FSMC_NOR_ReturnToReadMode();
/* Erase the NOR memory block to write on */
FSMC_NOR_EraseBlock(WRITE_READ_ADDR);
/* Write data to FSMC NOR memory */
/* Fill the buffer to send */
Fill_Buffer(TxBuffer, BUFFER_SIZE, 0x3210);
FSMC_NOR_WriteBuffer(TxBuffer, WRITE_READ_ADDR, BUFFER_SIZE);
/* Read data from FSMC NOR memory */
FSMC_NOR_ReadBuffer(RxBuffer, WRITE_READ_ADDR, BUFFER_SIZE);
/* Read back NOR memory and check content correctness */
for(Index = 0x00; (Index < BUFFER_SIZE) && (WriteReadStatus == 0); Index++)
{
    if(RxBuffer[Index] != TxBuffer[Index])
    {
        WriteReadStatus = Index + 1;
    }
}
if(WriteReadStatus == 0)
{ /* OK */
  /* Turn on LD1 */
    GPIO_SetBits(GPIOF, GPIO_Pin_6);
}
else
{ /* KO */
  /* Turn on LD2 */
    GPIO_SetBits(GPIOF, GPIO_Pin_7);
}
```

```
    while(1)
    {
    }
}
/*******************************************************************************
* Function Name  : RCC_Configuration
* Description    : Configures the different system clocks.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RCC_Configuration(void)
{
    /* RCC system reset(for debug purpose) */
    RCC_DeInit();
    /* Enable HSE */
    RCC_HSEConfig(RCC_HSE_ON);
    /* Wait till HSE is ready */
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
    if(HSEStartUpStatus == SUCCESS)
    {
        /* Enable Prefetch Buffer */
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
        /* Flash 2 wait state */
        FLASH_SetLatency(FLASH_Latency_2);

        /* HCLK = SYSCLK */
        RCC_HCLKConfig(RCC_SYSCLK_Div1);

        /* PCLK2 = HCLK */
        RCC_PCLK2Config(RCC_HCLK_Div1);
        /* PCLK1 = HCLK/2 */
        RCC_PCLK1Config(RCC_HCLK_Div2);
        /* PLLCLK = 8MHz * 9 = 72 MHz */
        RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
        /* Enable PLL */
        RCC_PLLCmd(ENABLE);
        /* Wait till PLL is ready */
        while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
        {
        }
        /* Select PLL as system clock source */
```

```
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
        /* Wait till PLL is used as system clock source */
        while(RCC_GetSYSCLKSource() != 0x08)
        {
        }
    }
}
/*******************************************************************************
* Function Name  : NVIC_Configuration
* Description    : Configures Vector Table base location.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void NVIC_Configuration(void)
{
#ifdef  VECT_TAB_RAM
    /* Set the Vector Table base location at 0x20000000 */
    NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  /* VECT_TAB_FLASH  */
    /* Set the Vector Table base location at 0x08000000 */
    NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif
}
/*******************************************************************************
* Function name : Fill_Buffer
* Description   : Fill the global buffer
* Input         : - pBuffer: pointer on the Buffer to fill
*                 - BufferSize: size of the buffer to fill
*                 - Offset: first value to fill on the Buffer
* Output param  : None
*******************************************************************************/
void Fill_Buffer(u16 *pBuffer, u16 BufferLenght, u32 Offset)
{
    u16 IndexTmp = 0;
    /* Put in global buffer same values */
    for(IndexTmp = 0; IndexTmp < BufferLenght; IndexTmp++)
    {
        pBuffer[IndexTmp] = IndexTmp + Offset;
    }
}
#ifdef  DEBUG
```

```c
/*******************************************************************************
* Function Name  : assert_failed
* Description    : Reports the name of the source file and the source line number
*                  where the assert_param error has occurred.
* Input          : - file: pointer to the source file name
*                  - line: assert_param error line source number
* Output         : None
* Return         : None
*******************************************************************************/
void assert_failed(u8* file, u32 line)
{
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* Infinite loop */
    while(1)
    {
    }
}
#endif
```