ARC

SPI

支持	www.armrunc.com http://armrunc.taobao.com

Change History

版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-03-06

目录

第13章 SPI

13.1 SPI 简介

SPI(Serial Peripheral Interface--串行外设接口)总线系统是一种同步串行数据连接标准。它是由摩托罗拉定义的，可工作在全双工模式。连接到 SPI 总线上的器件工作在主/从模式，主器件发起数据传输，SPI 总线上可以连接多个从器件，它们通过器件选择脚（CS）来区分。

13.1.1 SPI 总线接口

SPI 总线包含四个逻辑信号：

- SCLK：串行时钟，由主设备产生；
- MOSI：主设备输出，从设备输入，由主设备产生；
- MISO：主设备输入，从设备输出，由从设备产生；
- CS：从设备片选，低电平有效，由主设备产生。

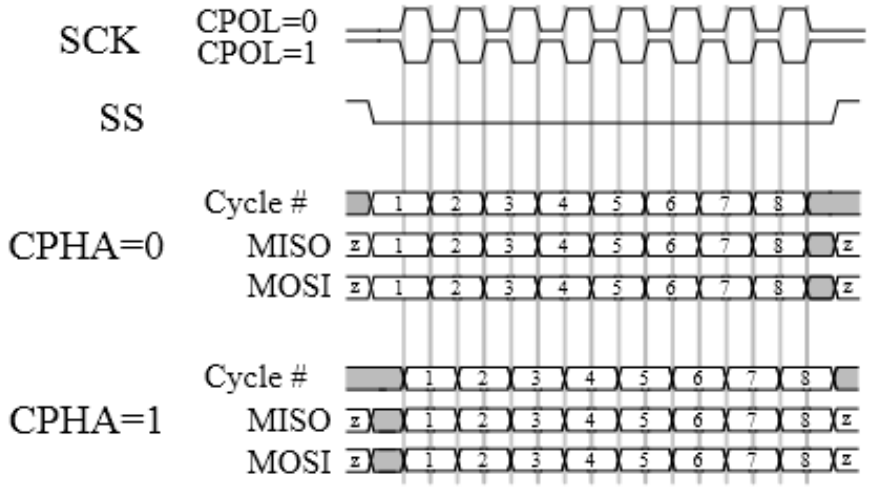
13.1.2 SPI 总线操作

SPI 总线可工作在一个主器件和一个或者多个从器件模式下，如果有多个从器件挂在 SPI 总线上，通过片选信号 CS 来识别。

主器件首先配置时钟来开始 SPI 传输，然后拉低目标从器件的片选信号 CS 来选中该从器件，在每个 SPI 时钟周期内，一个全双工的数据传输包括：

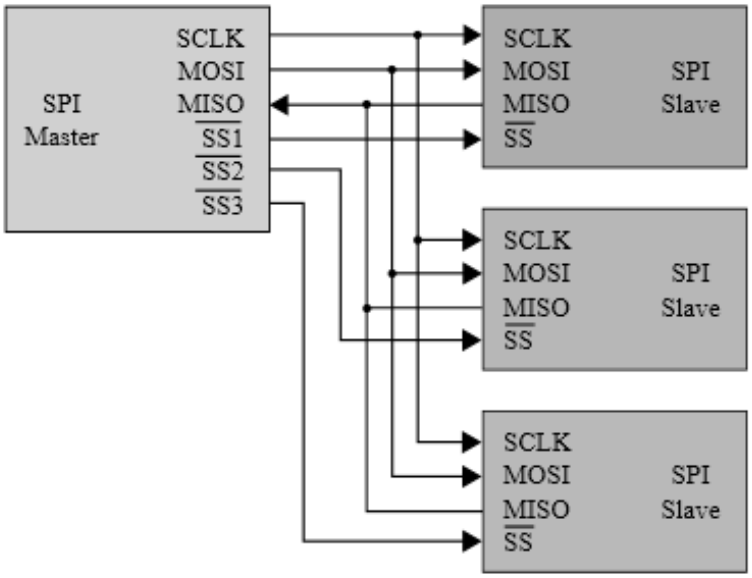
- 主器件发送一个比特位到 MOSI 线上，从器件从该线上读取数据；
- 从器件发送一个比特位到 MISO 线上，主器件从该线上读取数据。

主器件除了要配置时钟频率，还要配置时钟极性（CPOL）和时钟相位（CPHA）。两种时钟极性和时钟相位可以组合为四种工作模式，如下：



模式	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

典型的 SPI 总线连接示意图如下：



13.2 SPI 应用实例 ----- 读写 SPI flash

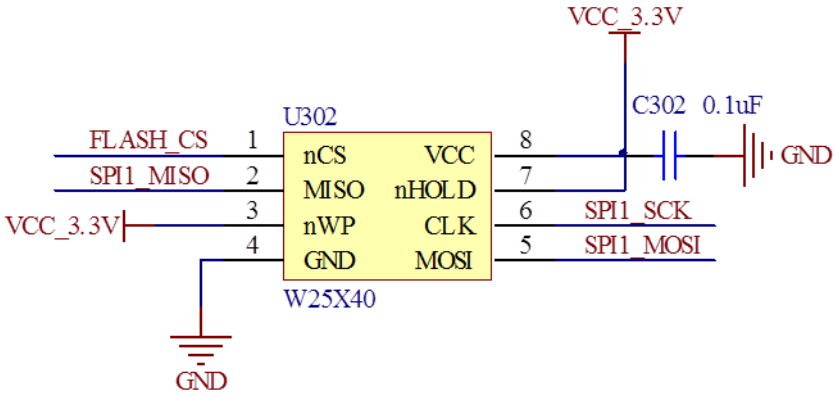
13.2.1 实例描述

本实例通过 STM32 SPI 硬件读写 SPI flash,然后通过串口打印出来，通过对比，读者能看到 SPI flash 访问是否正常。本实例可以支持多种 FLASH，并且很容易扩展支持其它的 flash，只要在文件 ARC_SPI_Flash.c 填表 spi_flash_list[] 即可。

13.2.2 硬件设计

该实例用到 PA4, PA5, PA6 和 PA7。GPIO 分配和硬件图如下：
注：SPI flash 型号不一定是 W25X40.

FLASH_CS	PA4
SPI1_SCK	PA5
SPI1_MISO	PA6
SPI1_MOSI	PA7



13.2.3 软件设计

本实例首先初始化 SysTick 用于精确延迟，初始化串口用于输出调试，然后再初始化 SPI 接口，SPI 接口配置为两线全双工主器件工作模式，数据传输大小为 8 比特，时钟极性，相位工作在模式 0（见上图）。SPI 波特率预分频系数为 4 （72MHz / 4）.

然后调用函数 ARC_SPI_FLASH_ID_check() 来识别 SPI flash，如果 flash 被识别，将会打印该 flash 描述，如果你的 flash 型号没有在 spi_flash_list[]，将会输出 flash not present or not recognized，请对照 flash 的规格书将该 flash 的信息加入到表 spi_flash_list[]即可。

SPI 写入之前必须确保所写的地址范围内的数据全部为 0XFF,否则在非 0XFF 处写入的数据将失败。所以在调用 flash 写函数 ARC_FLASH_WriteBuffer()之前，先调用函数 ARC_FLASH_EraseSector()将该 flash 擦除。

该实例的主要代码如下：
文件 SPI_FLASH_main.c：

```
/**
  * @brief   Main program, SPI flash write/read example.
  * @param   None
  * @retval None
  */
int main(void)
{
    uint8_t   Tx_Buffer[] = "ARC SPI FLASH Example";
    uint8_t   Rx_Buffer[512] = "flash not present or not recognized\n";
    uint16_t BufferSize = sizeof(Tx_Buffer) / sizeof(*Tx_Buffer);
    uint32_t FlashAddr = 0x0;

    ARC_SysTick_Init();

    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);
```

```
    ARC_SPI_Flash_Init();
    /*!< Enable the SPI1    */
    SPI_Cmd(SPI1, ENABLE);

    ARC_SPI_FLASH_ID_check();

    if (spi_flash_found)
    {
        /* Erase SPI FLASH Sector to write on */
        ARC_FLASH_EraseSector(FlashAddr);

        /* Write Tx_Buffer data to SPI FLASH memory */
        ARC_FLASH_WriteBuffer(Tx_Buffer, FlashAddr, BufferSize);

        /* Read data from SPI FLASH memory */
        ARC_FLASH_ReadBuffer(Rx_Buffer, FlashAddr, BufferSize);
    }

    while (1)
    {
        printf("Tx: %s\n", Tx_Buffer);
        printf("Rx: %s\n", Rx_Buffer);
        ARC_SysTick_Delay(1000);
    }
}
```

文件 ARC_SPI_Flash.c：

```
/**
  * @brief   Set flash index and the found status.
  * @param   flash_index: the flash index.
  * @param   flash_found: the found status.
  * @retval None.
  */
void ARC_SPI_FLASH_set_index(uint32_t flash_index, uint8_t flash_found)
{
    spi_flash_index = flash_index;
    spi_flash_found = flash_found;
}

/**
  * @brief   get flash index and the found status.
  * @param   *flash_index: the flash index.
  * @retval the flash found status.
```

```
   */
uint8_t ARC_SPI_FLASH_get_index(uint32_t *flash_index)
{
      *flash_index = 0;
      if(spi_flash_found != 0)
            *flash_index = spi_flash_index;
      return spi_flash_found;
}

/**
   * @brief    Check the ID of the flash, including manufacture ID and device ID.
   * @param    None
   * @retval None.
   */
void ARC_SPI_FLASH_ID_check(void)
{
      uint32_t i = 0, flash_list_count = 0;
      uint32_t FlashID = 0;

      flash_list_count = sizeof(spi_flash_list)/sizeof(SPI_FLASH_CMD);


      for(i = 0; i < flash_list_count; i++)
      {
            ARC_FLASH_CS_LOW();
            ARC_SPI_SendByte(SPI1, spi_flash_list[i].flash_cmd_rdid);
            FlashID |= (ARC_SPI_SendByte(SPI1, FLASH_DUMMY_BYTE) &
0xFF) << 16;
            FlashID |= (ARC_SPI_SendByte(SPI1, FLASH_DUMMY_BYTE) &
0xFF) << 8;
            FlashID |= ARC_SPI_SendByte(SPI1, FLASH_DUMMY_BYTE) &
0xFF;

            ARC_FLASH_CS_HIGH();

            if(spi_flash_list[i].flash_id == FlashID)
            {
                  printf("%s found\n", spi_flash_list[i].flash_desc);
                  ARC_SPI_FLASH_set_index(i, 1);
                  return;
            }
      }
      printf("flash not present or not recognized\n");
      ARC_SPI_FLASH_set_index(0, 0);
```

```
}

/**
   * @brief    Initialize SPI flash.
   * @param    None
   * @retval None.
   */
void ARC_SPI_Flash_Init()
{
      ARC_SPI_Flash_RCC_Init();
      ARC_SPI_Flash_GPIO_Init();
      ARC_SPI_PARAM_Init(SPI1);
}

/**
   * @brief    Polls the status of the Write In Progress (WIP) flag in the FLASH's
   *              status register and loop until write opertaion has completed.
   * @param    None
   * @retval None
   */
void ARC_FLASH_WaitForWriteEnd(void)
{
      uint8_t flashstatus = 0;

      /*!< Select the FLASH: Chip Select low */
      ARC_FLASH_CS_LOW();

      /*!< Send "Read Status Register" instruction */
      ARC_SPI_SendByte(SPI1,
spi_flash_list[spi_flash_index].flash_cmd_rdsr);

      /*!< Loop as long as the memory is busy with a write cycle */
      do
      {
            /*!< Send a dummy byte to generate the clock needed by the FLASH
            and put the value of the status register in FLASH_Status variable */
            flashstatus = ARC_SPI_SendByte(SPI1, FLASH_DUMMY_BYTE);
      }while ((flashstatus & FLASH_WIP_FLAG) == SET); /* Write in progress */

      /*!< Deselect the FLASH: Chip Select high */
      ARC_FLASH_CS_HIGH();
}

/**
```

```
  * @brief  Enables the write access to the FLASH.
  * @param  None
  * @retval None
  */
void ARC_Flash_WriteEnable(void)
{
    /*!< Select the FLASH: Chip Select low */
    ARC_FLASH_CS_LOW();

    /*!< Send "Write Enable" instruction */
    ARC_SPI_SendByte(SPI1,
spi_flash_list[spi_flash_index].flash_cmd_wren);

    /*!< Deselect the FLASH: Chip Select high */
    ARC_FLASH_CS_HIGH();
}


/**
  * @brief  Erases the specified FLASH sector.
  * @param  SectorAddr: address of the sector to erase.
  * @retval None
  */
void ARC_FLASH_EraseSector(uint32_t FlashAddr)
{
    /*!< Send write enable instruction */
    ARC_Flash_WriteEnable();

    /*!< Sector Erase */

    /*!< Select the FLASH: Chip Select low */
    ARC_FLASH_CS_LOW();
    /*!< Send Sector Erase instruction */
    ARC_SPI_SendByte(SPI1, spi_flash_list[spi_flash_index].flash_cmd_se);
    /*!< Send SectorAddr high nibble address byte */
    ARC_SPI_SendByte(SPI1, (FlashAddr & 0xFF0000) >> 16);
    /*!< Send SectorAddr medium nibble address byte */
    ARC_SPI_SendByte(SPI1, (FlashAddr & 0xFF00) >> 8);
    /*!< Send SectorAddr low nibble address byte */
    ARC_SPI_SendByte(SPI1, FlashAddr & 0xFF);
    /*!< Deselect the FLASH: Chip Select high */
    ARC_FLASH_CS_HIGH();

    /*!< Wait the end of Flash writing */
```

```
    ARC_FLASH_WaitForWriteEnd();
}

/**
  * @brief  Writes more than one byte to the FLASH with a single WRITE cycle
  *         (Page WRITE sequence).
  * @note   The number of byte can't exceed the FLASH page size.
  * @param  pBuffer: pointer to the buffer  containing the data to be written
  *         to the FLASH.
  * @param  WriteAddr: FLASH's internal address to write to.
  * @param  NumByteToWrite: number of bytes to write to the FLASH, must be equal
  *         or less than "FLASH_PAGESIZE" value.
  * @retval None
  */
void ARC_FLASH_WritePage(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)
{
    /*!< Enable the write access to the FLASH */
    ARC_Flash_WriteEnable();

    /*!< Select the FLASH: Chip Select low */
    ARC_FLASH_CS_LOW();
    /*!< Send "Write to Memory " instruction */
    ARC_SPI_SendByte(SPI1, spi_flash_list[spi_flash_index].flash_cmd_write);
    /*!< Send WriteAddr high nibble address byte to write to */
    ARC_SPI_SendByte(SPI1, (WriteAddr & 0xFF0000) >> 16);
    /*!< Send WriteAddr medium nibble address byte to write to */
    ARC_SPI_SendByte(SPI1, (WriteAddr & 0xFF00) >> 8);
    /*!< Send WriteAddr low nibble address byte to write to */
    ARC_SPI_SendByte(SPI1, WriteAddr & 0xFF);

    /*!< while there is data to be written on the FLASH */
    while (NumByteToWrite--)
    {
        /*!< Send the current byte */
        ARC_SPI_SendByte(SPI1, *pBuffer);
        /*!< Point on the next byte to be written */
        pBuffer++;
    }

    /*!< Deselect the FLASH: Chip Select high */
```

```
    ARC_FLASH_CS_HIGH();

    /*!< Wait the end of Flash writing */
    ARC_FLASH_WaitForWriteEnd();
}

/**
  * @brief   Writes block of data to the FLASH. In this function, the number of
  *          WRITE cycles are reduced, using Page WRITE sequence.
  * @param   pBuffer: pointer to the buffer  containing the data to be written
  *          to the FLASH.
  * @param   WriteAddr: FLASH's internal address to write to.
  * @param   NumByteToWrite: number of bytes to write to the FLASH.
  * @retval None
  */
void ARC_FLASH_WriteBuffer(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t
NumByteToWrite)
{
    uint8_t num_of_page = 0, num_of_single = 0, offset_in_page = 0,
num_remained_in_1st_page = 0, temp = 0;
    uint16_t spi_flash_page_size =
spi_flash_list[spi_flash_index].flash_spi_pagesize;

    offset_in_page = WriteAddr % spi_flash_page_size;
    num_remained_in_1st_page = spi_flash_page_size - offset_in_page;
    num_of_page =  NumByteToWrite / spi_flash_page_size;
    num_of_single = NumByteToWrite % spi_flash_page_size;

    if (offset_in_page == 0) /*!< WriteAddr is FLASH_PAGESIZE aligned  */
    {
        if (num_of_page == 0) /*!< NumByteToWrite < FLASH_PAGESIZE */
        {
            ARC_FLASH_WritePage(pBuffer, WriteAddr, NumByteToWrite);
        }
        else /*!< NumByteToWrite > FLASH_PAGESIZE */
        {
            while (num_of_page--)
            {
                ARC_FLASH_WritePage(pBuffer, WriteAddr,
spi_flash_page_size);
                WriteAddr +=  spi_flash_page_size;
                pBuffer += spi_flash_page_size;
            }
```

```
                if (num_of_single != 0)
                {
                    ARC_FLASH_WritePage(pBuffer, WriteAddr,
num_of_single);
                }
            }
        }
        else /*!< WriteAddr is not FLASH_PAGESIZE aligned  */
        {
            if (num_of_page == 0) /*!< NumByteToWrite < FLASH_PAGESIZE */
            {
                if (num_of_single > num_remained_in_1st_page) /*!<
(NumByteToWrite + WriteAddr) > FLASH_PAGESIZE */
                {
                    temp = num_of_single - num_remained_in_1st_page;

                    ARC_FLASH_WritePage(pBuffer, WriteAddr,
num_remained_in_1st_page);
                    WriteAddr +=  num_remained_in_1st_page;
                    pBuffer += num_remained_in_1st_page;

                    ARC_FLASH_WritePage(pBuffer, WriteAddr, temp);
                }
                else
                {
                    ARC_FLASH_WritePage(pBuffer, WriteAddr,
NumByteToWrite);
                }
            }
            else /*!< NumByteToWrite > FLASH_PAGESIZE */
            {
                NumByteToWrite -= num_remained_in_1st_page;
                num_of_page =  NumByteToWrite / spi_flash_page_size;
                num_of_single = NumByteToWrite % spi_flash_page_size;

                ARC_FLASH_WritePage(pBuffer, WriteAddr,
num_remained_in_1st_page);
                WriteAddr +=  num_remained_in_1st_page;
                pBuffer += num_remained_in_1st_page;

                while (num_of_page--)
                {
                    ARC_FLASH_WritePage(pBuffer, WriteAddr,
spi_flash_page_size);
```

```
                WriteAddr +=    spi_flash_page_size;
                pBuffer += spi_flash_page_size;
            }

            if (num_of_single != 0)
            {
                ARC_FLASH_WritePage(pBuffer, WriteAddr,
num_of_single);
            }
        }
    }
}

/**
  * @brief   Reads a block of data from the FLASH.
  * @param   pBuffer: pointer to the buffer that receives the data read from
the FLASH.
  * @param   ReadAddr: FLASH's internal address to read from.
  * @param   NumByteToRead: number of bytes to read from the FLASH.
  * @retval None
  */
void ARC_FLASH_ReadBuffer(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t
NumByteToRead)
{
    /*!< Select the FLASH: Chip Select low */
    ARC_FLASH_CS_LOW();

    /*!< Send "Read from Memory " instruction */
    ARC_SPI_SendByte(SPI1,
spi_flash_list[spi_flash_index].flash_cmd_read);

    /*!< Send ReadAddr high nibble address byte to read from */
    ARC_SPI_SendByte(SPI1, (ReadAddr & 0xFF0000) >> 16);
    /*!< Send ReadAddr medium nibble address byte to read from */
    ARC_SPI_SendByte(SPI1, (ReadAddr& 0xFF00) >> 8);
    /*!< Send ReadAddr low nibble address byte to read from */
    ARC_SPI_SendByte(SPI1, ReadAddr & 0xFF);

    while (NumByteToRead--) /*!< while there is data to be read */
    {
        /*!< Read a byte from the FLASH */
        *pBuffer = ARC_SPI_SendByte(SPI1, FLASH_DUMMY_BYTE);
        /*!< Point to the next location where the byte read will be saved */
        pBuffer++;
```

```
    }

    /*!< Deselect the FLASH: Chip Select high */
    ARC_FLASH_CS_HIGH();
}
```

文件 ARC_SPI.c：

```
/**
  * @brief  Initialize SPI parameters
  * @param  *SPIx: the SPI device ID.
  * @retval None
  */
void ARC_SPI_PARAM_Init(SPI_TypeDef *SPIx)
{
    SPI_InitTypeDef  SPI_InitStructure;

    /*!< SPI configuration */
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
    SPI_InitStructure.SPI_BaudRatePrescaler =
SPI_BaudRatePrescaler_256;
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;
    SPI_Init(SPIx, &SPI_InitStructure);
}


/**
  * @brief  Sends a byte through the SPI interface and return the byte
received
  *         from the SPI bus.
  * @param  *SPIx: the SPI device ID.
  * @param  byte: byte to send.
  * @retval The value of the received byte.
  */
uint8_t ARC_SPI_SendByte(SPI_TypeDef *SPIx, uint8_t byte)
{
    /*!< Loop while DR register in not empty */
    while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
```

```
    /*!< Send byte through the SPI1 peripheral */
    SPI_I2S_SendData(SPIx, byte);

    /*!< Wait to receive a byte */
    while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);

    /*!< Return the byte read from the SPI bus */
    return SPI_I2S_ReceiveData(SPIx);
}
```

文件 ARC_RCC.c

```
/**
  * @brief   Configures SPI Flash clocks.
  * @param   None
  * @retval None
  */
void ARC_SPI_Flash_RCC_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_AFIO, ENABLE);

    /*!< FLASH_SPI Periph clock enable */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
}
```

文件 ARC_GPIO.c

```
/**
  * @brief   Configures SPI GPIO ports.
  * @param   None
  * @retval None
  */

/*
        ---------------
        | CLK   | PA5   |
        ---------------
        | MISO | PA6   |
        ---------------
        | MOSI | PA7   |
        ---------------
        | CS    | PA4   |
        ---------------
*/
void ARC_SPI_Flash_GPIO_Init()
```

```
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*!< Configure FLASH_SPI pins: SCK */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure FLASH_SPI pins: MOSI */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure FLASH_SPI pins: MISO */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*!< Configure FLASH_CS_PIN pin: FLASH CS pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```