AN4761
Application note

Using STM32L476/486 FSMC peripheral to drive external memories

Introduction

This application note describes the use of the STM32L476/486 FSMC (flexible static memory controller) peripheral to drive a set of external memories. For that aim, it gives an overview of STM32L476/486 FSMC. The document also presents memory interfacing examples that include typical FSMC configuration, timing computation method and hardware connection. In the document, FMC generic acronym substitutes FSMC.

This application note considers a 16-bit asynchronous NOR Flash memory, an 8-bit NAND Flash memory and a 16-bit asynchronous SRAM.

STM32CubeL4 firmware package and memory drivers corresponding to the memory types presented in this application note are available for download on STMicroelectronics website *www.st.com*.

Table 1 provides the list of products to which this application note applies.

Table 1. Applicable products

Type	Part numbers
Microcontroller	STM32L476QE, STM32L476QG, STM32L476VC, STM32L476VE, STM32L476VG, STM32L476ZE, STM32L476ZG, STM32L486QG, STM32L486VG, STM32L486ZG

Contents

1 Overview of the STM32L476/486 flexible static memory controller . . . 5

1.1 Interfacing asynchronous static memories (NOR Flash memory, SRAM) . 7

2 Interfacing a non-multiplexed, asynchronous 16-bit NOR Flash memory 9

2.1 FMC configuration 9

2.1.1 Typical use of FMC to interface with a NOR Flash memory 10

2.2 Timing computation 11

2.3 Hardware connection 13

3 Interfacing a non-multiplexed, asynchronous 16-bit SRAM 22

3.1 FMC configuration 22

3.1.1 Typical use of FMC to interface with an SRAM 23

3.2 Timing computation 23

3.3 Hardware connection 24

4 Interfacing an 8-bit NAND Flash memory 30

4.1 FMC configuration 30

4.1.1 Typical use of FMC to interface with a NAND memory 31

4.2 Timing computation 32

4.3 Hardware connection 34

4.4 Error correction code computation 40

4.4.1 Error correction code (ECC) computation overview 40

4.4.2 Error detection 40

5 STM32L476/486 FMC configuration in 100-pin packages 42

5.1 Interfacing FMC with a NOR Flash memory 42

6 Revision history 43

List of tables

Table 1. Applicable products . . . 1
Table 2. FMC operating modes . . . 7
Table 3. STM32L476/486 FMC asynchronous timings . . . 8
Table 4. NOR Flash memory timings . . . 12
Table 5. M29W128FL signal to FMC pin correspondence . . . 13
Table 6. IS61WV102416BL SRAM timings . . . 24
Table 7. IS61WV102416BLL signal to FMC pin correspondence . . . 24
Table 8. NAND512W3A2C Flash memory timings . . . 34
Table 9. NAND512W3A signal to FMC pin correspondence . . . 34
Table 10. NOR Flash memory connection to FMC . . . 42
Table 11. Document revision history . . . 43

List of figures

Figure 1. FMC block diagram . . . 6
Figure 2. FMC memory banks . . . 7
Figure 3. Asynchronous NOR Flash memory read access timing . . . 10
Figure 4. Asynchronous NOR Flash memory write access timing . . . 10
Figure 5. 16-bit NOR Flash memory: M29W128FL/GL connection to STM32L476/486 . . . 14
Figure 6. SRAM asynchronous read access timing . . . 22
Figure 7. SRAM asynchronous write access timing . . . 23
Figure 8. 16-bit SRAM: IS61WV102416BLL connection to STM32L476/486 . . . 25
Figure 9. FMC NAND bank sections . . . 30
Figure 10. NAND memory access timing . . . 31
Figure 11. 8-bit NAND Flash memory: NAND512W3A2C/NAND512W3A2B connection to STM32L476/486 . . . 35
Figure 12. Error detection flowchart . . . 40

1 Overview of the STM32L476/486 flexible static memory controller

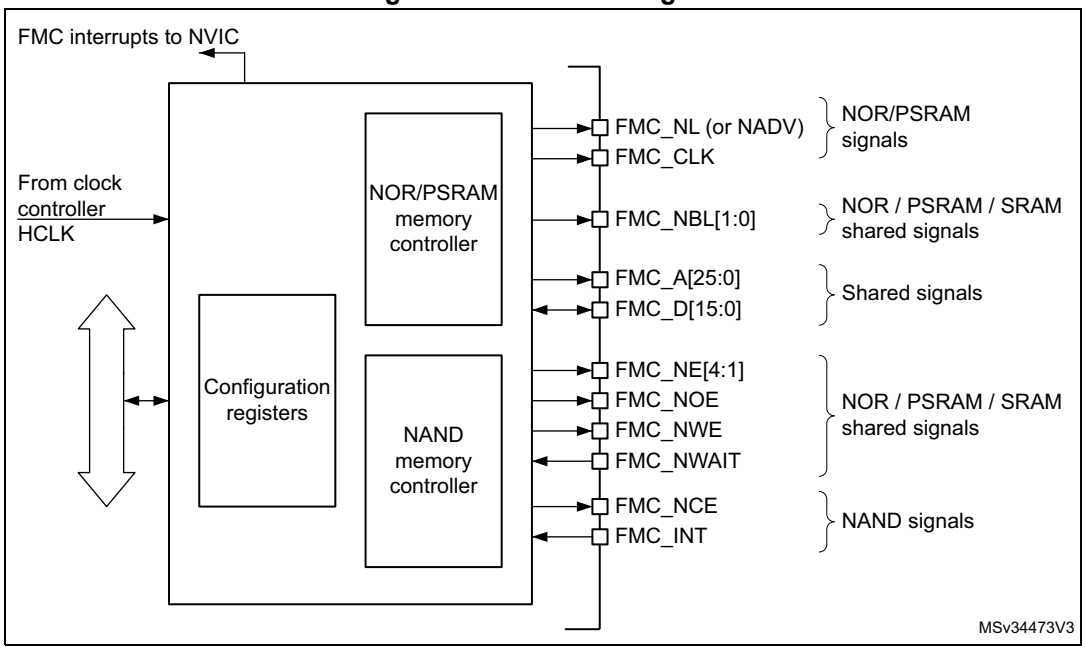
FMC has the following main features:

- interface with static-memory mapped devices including:
  - static random access memory (SRAM)
  - NOR Flash memory/OneNAND Flash memory
  - PSRAM (4 memory banks)
  - NAND Flash memory with ECC hardware to check up to 8 Kbyte of data
- burst mode access to synchronous devices (NOR Flash memory and PSRAM)
- programmable continuous clock output for asynchronous and synchronous accesses
- 8- or 16-bit data bus width
- independent chip select control for each memory bank
- independent configuration for each memory bank
- programmable timings to support a wide range of devices, in particular:
  - programmable wait states (up to 15)
  - programmable bus turnaround cycles (up to 15)
  - programmable output enable and write enable delays (up to 15)
  - independent read and write timings and protocol, so as to support the widest variety of memories and timings
- write enable and byte lane select outputs for use with PSRAM and SRAM devices
- translation of 32-bit wide AHB transactions into consecutive 16-bit or 8-bit accesses to external 16-bit or 8-bit devices
- write FIFO (can be disabled by setting the WFDIS bit)
- external asynchronous wait control

The FMC registers that define the external device type and associated characteristics are usually set at boot time and do not change until the next reset or power-up. However, it is possible to change the settings at any time.

Figure 1 illustrates the FMC block diagram.

Figure 1. FMC block diagram

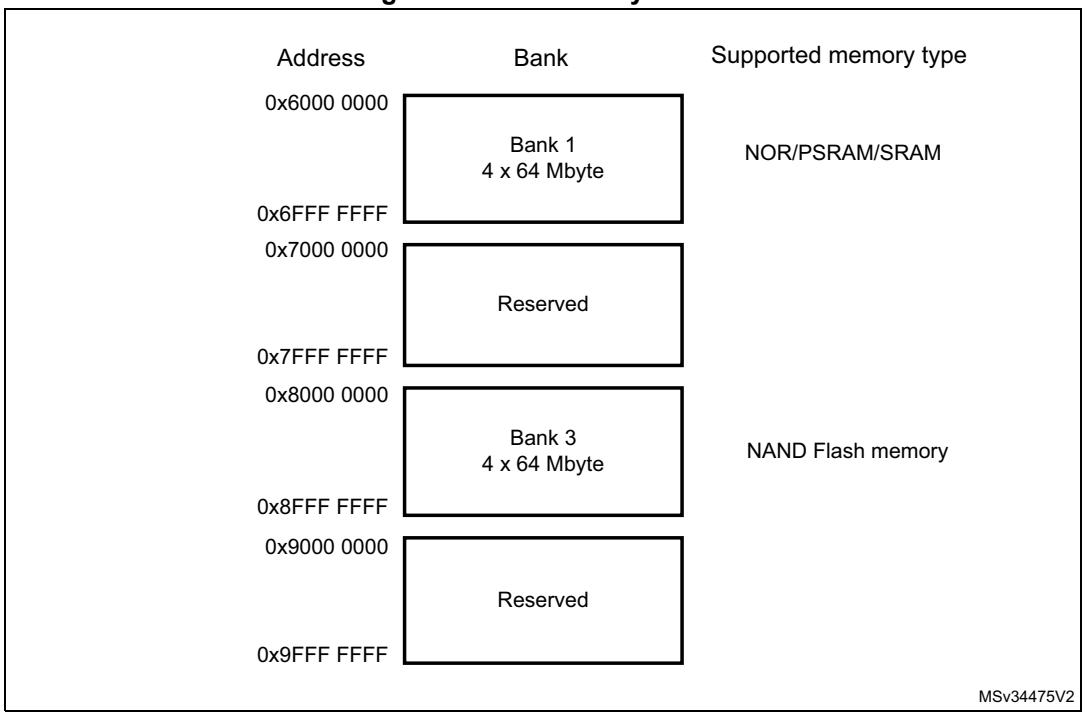


From the FMC point of view, the external memory is divided into two fixed-size banks of 256 Mbyte each, as shown in *Figure 2*:

- Bank 1 used by the NOR Flash memory/SRAM controller to address up to 4 memory devices. This bank is split into 4 regions with 4 dedicated Chip Select signals.
- Bank 3 used by the NAND Flash memory controller to address NAND Flash memory devices.

For each bank, the type of memory to be used is user-defined in the Configuration register.

Figure 2. FMC memory banks



1.1 Interfacing asynchronous static memories (NOR Flash memory, SRAM)

The interface signals are synchronized by the internal clock HCLK. This clock is not output to the memory

FMC always samples the data before de-asserting the chip select signal NE. This guarantees that the memory data-hold timing constraint is met (chip enable high to data transition, usually 0 ns min.).

- If extended mode is enabled (EXTMOD: bit is set in the FMC_BCRx register), there are up to four extended modes (A, B, C and D) and it is possible to mix modes A, B, C and D in read and write access. For example, read operation can be performed in mode A and write in mode B.
- If Extended mode is disabled, FMC operates in Mode1 or Mode2 as follows:
  - Mode 1 is the default mode when SRAM/ PSRAM memory type is selected (Bits 3:2 MTYP = 0x0 or 0x01 in the FMC_BCRx register).
  - Mode 2 is the default mode when NOR memory type is selected (Bits 3:2 MTYP = 0x10 in the FMC_BCRx register).

Table 2. FMC operating modes

Asynchronous mode	Memory type	
	SRAM/PSRAM	NOR
Extended mode disabled	Mode 1	Mode2
Extended mode enabled	Mode A	Mode B, C, D

Table 3. STM32L476/486 FMC asynchronous timings

Symbols	Parameter	Value	Unit
HCLK	Internal AHB clock frequency	80	MHz
t_{HCLK}	Internal AHB clock cycle	12.5	ns
$t_{su(Data\_NE)} + t_{v(A\_NE)}$	Data to FMC_NEx high setup time + FMC_NEx low to FMC_A valid	$1t_{HCLK} + 1$	ns

Note: These timings extracted from STM32L476/486 datasheet are needed to compute the NOR and SRAM asynchronous mode timings.

2 Interfacing a non-multiplexed, asynchronous 16-bit NOR Flash memory

2.1 FMC configuration

To control a NOR Flash memory, FMC provides the following possible features:

- Bank select for mapping the NOR Flash memory

  There are 4 independent banks which can be used to interface with NOR Flash memory/SRAM/PSRAM memories, each bank has a separate Chip Select pin.
- address/data multiplexing enable/disable
- memory type select (NOR Flash memory/SRAM/PSRAM)
- external memory data bus width selection (8/16 bits)
- burst access mode enable/disable, for synchronous NOR Flash memories
- wait signal configuration - enable/disable, polarity setting and timing configuration
- extended mode enable/disable

  This mode is used to access the memory with a different timing configuration for read and write operations.
- write fifo enable/disable

As the NOR Flash memory/PSRAM controller can support asynchronous and synchronous memories, the user should select only the used parameters depending on the memory characteristics.

FMC also provides the possibility of programming several parameters to correctly interface with the external memory. Depending on the memory type, some parameters are not used.

In the case where an external asynchronous non-multiplexed memory is used, the user has to compute and set the following parameters depending on the information in the memory datasheet:

- ADDSET: address setup time
- DATAST: data setup time
- ACCMOD: access mode

  This parameter gives FMC the flexibility to access a wide variety of asynchronous static memories. There are four extended access modes (A, B, C and D) that allow write access while reading the memory with different timings, if the memory supports this kind of feature.

  When the extended mode is enabled, the FMC_BTRx register is used for read operations and the FMC_BWTRx register is used for write operations.

In the case where a synchronous memory is used, the user has to compute and set the following parameters:

- CLKDIV: clock divide ratio
- DATLAT: data latency

Note that NOR Flash memory read operations can be synchronous if the memory supports this mode, while write operations usually remain asynchronous.

When programming a synchronous NOR Flash memory, the memory automatically switches between the synchronous and the asynchronous mode, so in this case, all parameters have to be set correctly.

Figure 3 and *Figure 4* show the different timings during a typical NOR Flash memory access.

Figure 3. Asynchronous NOR Flash memory read access timing

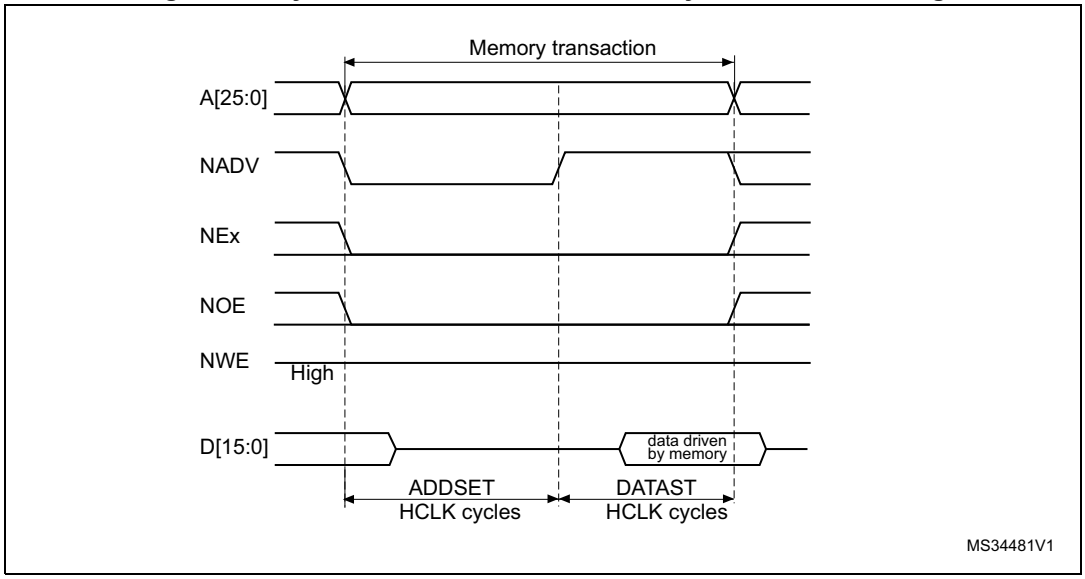
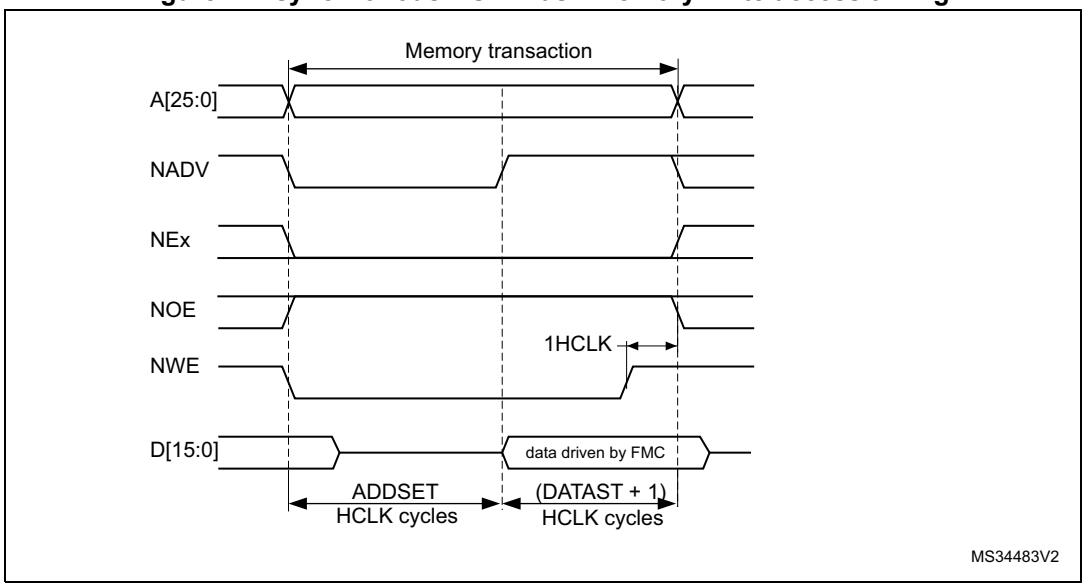


Figure 4. Asynchronous NOR Flash memory write access timing



2.1.1 Typical use of FMC to interface with a NOR Flash memory

The STM32L476/486 FMC has four different banks of 64 Mbyte to support NOR Flash memories/PSRAM and similar external memories.

The external memories share the address, data and control signals with the controller. Each external device is accessed by means of a unique Chip Select signal, but FMC performs only one access at a time to an external device.

Each bank is configured by means of dedicated registers. Configuration includes the different features and the timing parameters.

In this application note, the M29W128FL memory is used as a reference. The M29W128FL memory is a 16-bit, asynchronous, non-multiplexed NOR Flash memory. Based on these data, FMC is configured as follows:

Bank 1 - NOR/SRAM sub-bank 2 is selected to support the NOR Flash memory device:

- Bank 1 - NOR/SRAM sub-bank 2 is enabled: BCR2_MBKEN bit set to ‘1’.
- Memory type is NOR: BCR2_MTYP is set to ‘10’ to select the NOR memory type.
- Data bus width is 16-bit: BCR2_MWID is set to ‘01’ to select the 16-bit width.
- It is a non-multiplexed memory: BCR2_MUXEN is reset.

All remaining parameters must be kept cleared.

2.2 Timing computation

As described above, for asynchronous NOR Flash-like memories, there are different possible access protocols. The first step is therefore to define the kind of protocol that should be used with the specific memory. The choice depends on the different control signals and the behavior of the memory during read and write transactions.

In the case of an asynchronous NOR Flash memory, the Mode2 protocol will be used.

With the M29W128FL, we will use the Mode2 protocol. We will therefore not use any extended mode and the timings will be the same for read and write operations. In this case, the NOR memory controller needs two timing parameters: ADDSET and DATAST.

These parameters are computed according to the NOR Flash memory characteristics and according to the HCLK clock of the STM32L476/486.

Based on the NOR Flash memory access timings illustrated in *Figure 3* and *Figure 4*, the following equations are found:

- The write or read access time is the time between the falling edge and the rising edge of the memory Chip Select signal. This time is computed as a function of the FMC timing parameter:

  $(ADDSET + (DATAST + 1)) \times t_{HCLK} \geq$ Write access time
- In write operations, the DATAST parameter is measured between the falling edge and the rising edge of the write signal as follows:

  $t_{WP} = DATAST \times t_{HCLK} \geq$ Write Enable signal low to high

To have a correct configuration of the FMC timings, the timings have to take into account:

- the maximum read/write access time
- the different internal FMC delays
- the different internal memory delays

Hence, we have:

$(ADDSET + (DATAST + 1)) \times t_{HCLK} = \max (t_{WC}, t_{RC})$

$DATAST \times HCLK = t_{WP}$

For read access, DATAST must verify:

$$DATAST \geq (t_{AVQV} + t_{su(Data\_NE)} + t_{v(A\_NE)})/t_{HCLK} - ADDSET$$

Table 4 gives the meanings and values of the NOR Flash memory parameters.

Table 4. NOR Flash memory timings

Symbols	Parameter	Value		Unit
		M29W128xx70	S29GL128P90	
t_{WC}	Address valid to next address valid for write operation	70	90	ns
t_{RC}	Address valid to next address valid for read access	70	90	ns
t_{WP}	Write Enable low to Write Enable high	45	35	ns
t_{AVQV}	Address valid to output valid	70	90	ns

Using the memory timings in *Table 4* and the FMC asynchronous timings in *Table 3*, we can compute the following values:

- For the M29W128 NOR Flash memory, the timings are:
  - address setup time: 0x1
  - data setup time: 0x6
- For the S29GL128P NOR Flash memory, the timings are:
  - address setup time: 0x3
  - data setup time: 0x7

Note: The S29GL128P NOR Flash memory timings are also valid for the M29W128 NOR Flash memory.

2.3 Hardware connection

Table 5 gives the correspondence between the NOR Flash memory pins and the FMC pins and shows the GPIO configuration for each FMC pin.

In case of an 8-bit NOR Flash memory, the data/address bus is 8-bit wide and D8-D15 should not be connected to FMC. The unused FMC pins can be used for general purpose I/O.

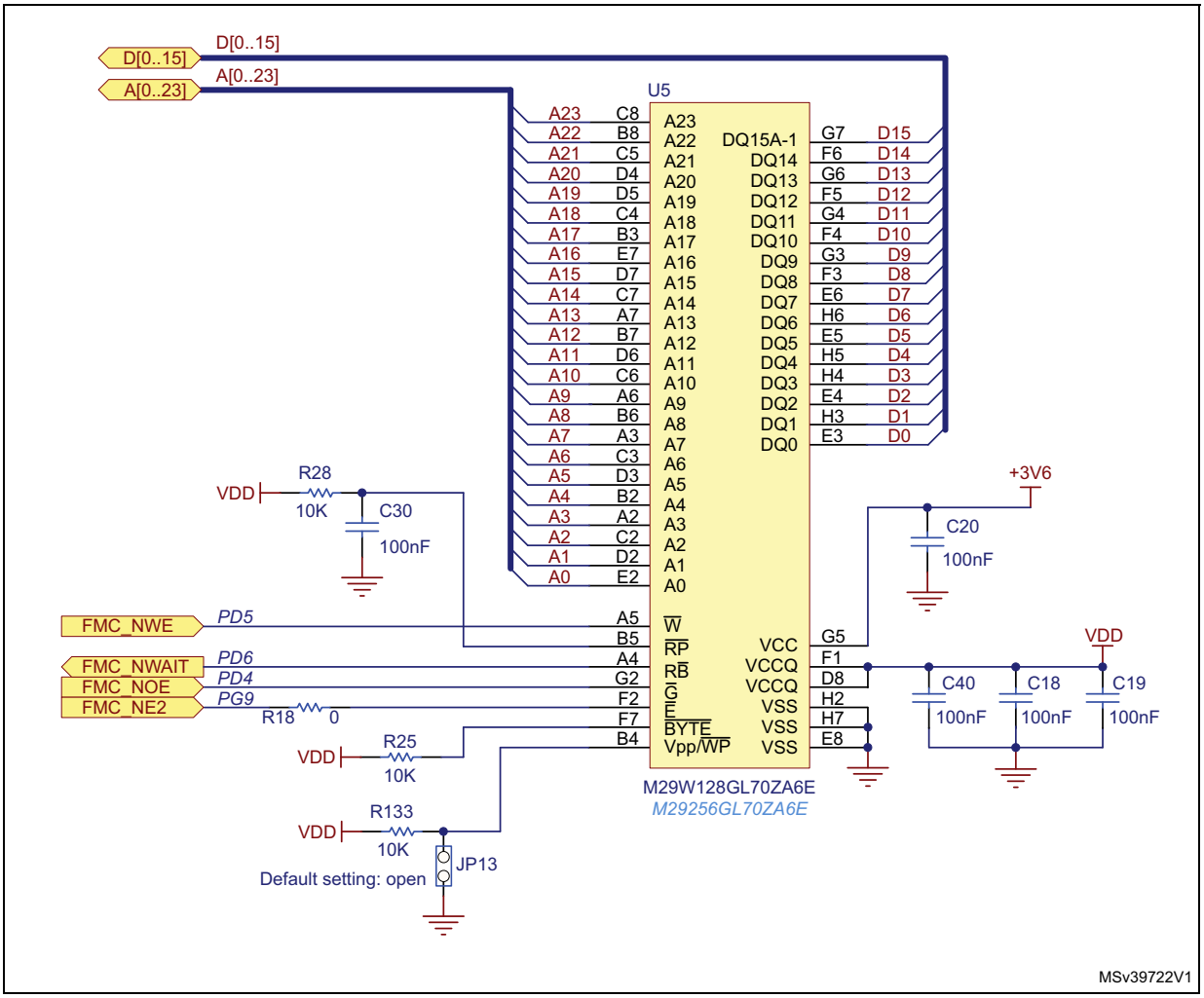
In case of a synchronous memory, the FMC_CLK pin should be connected to the memory clock pin.

Table 5. M29W128FL signal to FMC pin correspondence

Memory signals	FMC signals	Pin / Port assignment	Pin / Port configuration	Signal description
A0-A22	A0-A22	Port F/Port G/Port E/Port D	AF push-pull	Address A0-A22
DQ0-DQ7	D0-D7	Port D/Port E	AF push-pull	Data D0-D7
DQ8-DQ14	D8-D14	Port D/Port E	AF push-pull	Data D8-D14
DQ15A-1	D15	PD10	AF push-pull	Data D15
$\overline{E}$	NE2	PG9	AF push-pull	Chip Enable
$\overline{G}$	NOE	PD4	AF push-pull	Output Enable
$\overline{W}$	NWE	PD5	AF push-pull	Write Enable

Figure 5 shows a typical connection between an STM32L476/486 microcontroller and the M29W128FL NOR Flash memory.

Figure 5. 16-bit NOR Flash memory: M29W128FL/GL connection to STM32L476/486



Note: The GPIO PD6 pin is used to provide a Ready/Busy output signal for NOR Flash memories (in this case the application needs to poll on the state of this pin to guarantee correct operation).
In the example shown below, this pin is not used with the M29W128FL and M29W128GL NOR Flash memories. It is however required for the S29GL128P NOR Flash memory.

A firmware example is available hereafter.

The main goal of this example is to provide the basics of how to use the FMC firmware library and the associated NOR Flash memory driver to perform erase/read/write operations on the M29W128FL, M29W128GL or S29GL128P NOR Flash memories. The FMC NOR timings in the example are set for M29W128GL memory.

```
/* Includes ------------------------------------------------------------
---*/
#include "stm32l4xx_hal.h"

/* Private define ------------------------------------------------------
---*/
#define NOR_DEVICE_ADDR         ((uint32_t)0x64000000)

/* #define NOR_MEMORY_WIDTH     FMC_NORSRAM_MEM_BUS_WIDTH_8  */
#define NOR_MEMORY_WIDTH        FMC_NORSRAM_MEM_BUS_WIDTH_16

#define NOR_BURSTACCESS         FMC_BURST_ACCESS_MODE_DISABLE
/* #define NOR_BURSTACCESS      FMC_BURST_ACCESS_MODE_ENABLE*/

#define NOR_WRITEBURST          FMC_WRITE_BURST_DISABLE
/* #define NOR_WRITEBURST       FMC_WRITE_BURST_ENABLE */

/* NOR operations Timeout definitions */
#define BLOCKERASE_TIMEOUT      ((uint32_t)0x00A00000)  /* NOR block erase
timeout */
#define CHIPERASE_TIMEOUT       ((uint32_t)0x30000000)  /* NOR chip erase
timeout  */
#define PROGRAM_TIMEOUT         ((uint32_t)0x00004400)  /* NOR program
timeout     */

/* NOR Ready/Busy signal GPIO definitions */
#define NOR_READY_BUSY_PIN      GPIO_PIN_6
#define NOR_READY_BUSY_GPIO     GPIOD
#define NOR_READY_STATE         GPIO_PIN_SET
#define NOR_BUSY_STATE          GPIO_PIN_RESET

#define BUFFER_SIZE         ((uint32_t)16)
#define WRITE_READ_ADDR     ((uint32_t)0x8000)
#define MANUFACTURER_CODE   ((uint16_t)0x0020)
#define DEVICE_CODE1        ((uint16_t)0x227E)
#define DEVICE_CODE2        ((uint16_t)0x2221)
#define DEVICE_CODE3       ((uint16_t)0x2200)  /*00h for M29W128GL70ZA6E.*/

#define NOR_BANK_ADDR       ((uint32_t)0x64000000)
#define NOR_TIMEOUT_VALUE   ((uint32_t)0xFFFF)

/**
  * @brief  NOR status definition
  */
#define NOR_STATUS_OK            0x00
#define NOR_STATUS_INIT_ERROR    0x01
```

```
#define NOR_STATUS_READ_ID_ERROR 0x02
#define NOR_STATUS_ID_ERROR      0x03
#define NOR_STATUS_ERASE_ERROR   0x04
#define NOR_STATUS_WRITE_ERROR   0x05
#define NOR_STATUS_READ_ERROR    0x06

/* Private variables ---------------------------------------------------------*/
static NOR_HandleTypeDef         hNOR;
static FMC_NORSRAM_TimingTypeDef NOR_Timing;
static NOR_IDTypeDef             NOR_Id;

/* Read/Write Buffers */
uint16_t aTxBuffer[BUFFER_SIZE] = {0x00, 0x01, 0x02, 0x03,
                                   0x04, 0x05, 0x06, 0x07,
                                   0x08, 0x09, 0x0A, 0x0B,
                                   0x0C, 0x0D, 0x0E, 0x0F};
uint16_t aRxBuffer[BUFFER_SIZE] = {0};

/* Test function -------------------------------------------------------------*/
int Test_NOR(void)
{
  uint16_t *pdata = NULL;
  uint32_t index  = 0;
  uint32_t startaddress = 0;

  /*##-1- Configure the NOR device ###################################*/
  hNOR.Instance  = FMC_NORSRAM_DEVICE;
  hNOR.Extended  = FMC_NORSRAM_EXTENDED_DEVICE;

  /* NOR device configuration */
  NOR_Timing.AddressSetupTime       = 1;
  NOR_Timing.AddressHoldTime        = 1;
  NOR_Timing.DataSetupTime          = 6;
  NOR_Timing.BusTurnAroundDuration  = 0;
  NOR_Timing.CLKDivision            = 2;
  NOR_Timing.DataLatency            = 2;
  NOR_Timing.AccessMode             = FMC_ACCESS_MODE_B;

  hNOR.Init.NSBank                  = FMC_NORSRAM_BANK2;
  hNOR.Init.DataAddressMux          = FMC_DATA_ADDRESS_MUX_DISABLE;
  hNOR.Init.MemoryType              = FMC_MEMORY_TYPE_NOR;
  hNOR.Init.MemoryDataWidth         = FMC_NORSRAM_MEM_BUS_WIDTH_16;
  hNOR.Init.BurstAccessMode         = FMC_BURST_ACCESS_MODE_DISABLE;
```

```
  hNOR.Init.WaitSignalPolarity      = FMC_WAIT_SIGNAL_POLARITY_LOW;
  hNOR.Init.WaitSignalActive        = FMC_WAIT_TIMING_BEFORE_WS;
  hNOR.Init.WriteOperation          = FMC_WRITE_OPERATION_ENABLE;
  hNOR.Init.WaitSignal              = FMC_WAIT_SIGNAL_DISABLE;
  hNOR.Init.ExtendedMode            = FMC_EXTENDED_MODE_DISABLE;
  hNOR.Init.AsynchronousWait        = FMC_ASYNCHRONOUS_WAIT_DISABLE;
  hNOR.Init.WriteBurst              = FMC_WRITE_BURST_DISABLE;
  hNOR.Init.ContinuousClock         = FMC_CONTINUOUS_CLOCK_SYNC_ONLY;
  hNOR.Init.WriteFifo               = FMC_WRITE_FIFO_DISABLE;
  hNOR.Init.PageSize                = FMC_PAGE_SIZE_NONE;

  /* Initialize the NOR controller */
  if(HAL_NOR_Init(&hNOR, &NOR_Timing, &NOR_Timing) != HAL_OK)
  {
    return NOR_STATUS_INIT_ERROR;
  }

  /*##-2- Read & Check the NOR device IDs ##########################*/
  /* Read NOR memory ID */
  if(HAL_NOR_Read_ID(&hNOR, &NOR_Id) != HAL_OK)
  {
    /* NOR read ID Error */
    return NOR_STATUS_READ_ID_ERROR;
  }

   /* Test the NOR ID correctness */
  if((NOR_Id.Manufacturer_Code != (uint16_t)MANUFACTURER_CODE) ||
     (NOR_Id.Device_Code1 != (uint16_t)DEVICE_CODE1) ||
     (NOR_Id.Device_Code2 != (uint16_t)DEVICE_CODE2) ||
     (NOR_Id.Device_Code3 != (uint16_t)DEVICE_CODE3))
  {
    /* NOR ID not correct */
    return NOR_STATUS_ID_ERROR;
  }

  /*##-3- Erase NOR memory #########################################*/
  /* Return to read mode */
  HAL_NOR_ReturnToReadMode(&hNOR);

  /* Erase the NOR memory block to write on */
  HAL_NOR_Erase_Block(&hNOR, WRITE_READ_ADDR, NOR_BANK_ADDR);

  /* Return the NOR memory status */
  if(HAL_NOR_GetStatus(&hNOR, NOR_BANK_ADDR, NOR_TIMEOUT_VALUE) !=
HAL_NOR_STATUS_SUCCESS)
```

```
  {
    /* Erase Error */
    return NOR_STATUS_ERASE_ERROR;
  }

  /*##-4- NOR memory read/write access ############################*/
  /* Write data to the NOR memory */
  pdata = aTxBuffer;
  index = BUFFER_SIZE;
  startaddress = NOR_BANK_ADDR + WRITE_READ_ADDR;
  while(index > 0)
  {
    /* Write data to NOR */
    HAL_NOR_Program(&hNOR, (uint32_t *)startaddress, pdata);

    /* Read NOR device status */
    if(HAL_NOR_GetStatus(&hNOR, NOR_BANK_ADDR, NOR_TIMEOUT_VALUE) !=
HAL_NOR_STATUS_SUCCESS)
    {
      return NOR_STATUS_WRITE_ERROR;
    }

    /* Update the counters */
    index--;
    startaddress += 2;
    pdata++;
  }

  /* Read back data from the NOR memory */
  if(HAL_NOR_ReadBuffer(&hNOR, NOR_BANK_ADDR + WRITE_READ_ADDR, aRxBuffer,
BUFFER_SIZE) != HAL_OK)
  {
    return NOR_STATUS_READ_ERROR;
  }

  /* aRxBuffer shall be identical to aTxBuffer */

  return NOR_STATUS_OK;
}

/**
  * @brief NOR MCU Support Package Initialization
  *        This function configures the hardware resources used in this
example:
  *           - Peripheral's clock enable
  *           - Peripheral's GPIO Configuration
```

```
  * @param hnor: NOR handle pointer
  * @retval None
  */
void HAL_NOR_MspInit(NOR_HandleTypeDef *hnor)
{
  GPIO_InitTypeDef gpio;

  /* Enable FMC clock */
  __HAL_RCC_FMC_CLK_ENABLE();

  /* Enable GPIOs clock */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_PWREx_EnableVddIO2();

  /* Common GPIO configuration */
  gpio.Mode      = GPIO_MODE_AF_PP;
  gpio.Pull      = GPIO_PULLUP;
  gpio.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
  gpio.Alternate = GPIO_AF12_FMC;

  /*## Data Bus #######*/
  /* GPIOD configuration */
  gpio.Pin   = GPIO_PIN_0  | GPIO_PIN_1  | GPIO_PIN_8 | GPIO_PIN_9 | \
               GPIO_PIN_10 | GPIO_PIN_14 | GPIO_PIN_15;
  HAL_GPIO_Init(GPIOD, &gpio);

  /* GPIOE configuration */
  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;
  HAL_GPIO_Init(GPIOE, &gpio);

  /*## Address Bus #######*/
  /* GPIOF configuration */
  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;
  HAL_GPIO_Init(GPIOF, &gpio);

  /* GPIOG configuration */
  gpio.Pin   = GPIO_PIN_0  | GPIO_PIN_1  | GPIO_PIN_2  | \
```

```
                GPIO_PIN_3  | GPIO_PIN_4  | GPIO_PIN_5  | \
                GPIO_PIN_13 | GPIO_PIN_14;
  HAL_GPIO_Init(GPIOG, &gpio);

  /* GPIOD configuration */
  gpio.Pin   = GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13;
  HAL_GPIO_Init(GPIOD, &gpio);

  /* GPIOE configuration */
  gpio.Pin   = GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6;
  HAL_GPIO_Init(GPIOE, &gpio);

  /*## NOE and NWE configuration #######*/
  gpio.Pin = GPIO_PIN_4 | GPIO_PIN_5;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## Enable Bank pin configuration #######*/
  gpio.Pin = GPIO_PIN_9;
  HAL_GPIO_Init(GPIOG, &gpio);

  /*## NE3 configuration #######*/
  gpio.Pin = GPIO_PIN_10;
  HAL_GPIO_Init(GPIOG, &gpio);

  /*## NBL0, NBL1 configuration #######*/
  gpio.Pin = GPIO_PIN_0 | GPIO_PIN_1;
  HAL_GPIO_Init(GPIOE, &gpio);

  /*## Configure PD6 for NOR memory Ready/Busy signal ######*/
  gpio.Pin = GPIO_PIN_6;
  HAL_GPIO_Init(GPIOD, &gpio);
}

/**
  * @brief  NOR BSP Wait for Ready/Busy signal.
  * @param  hnor: Pointer to NOR handle
  * @param  Timeout: Timeout duration
  * @retval None
  */
void HAL_NOR_MspWait(NOR_HandleTypeDef *hnor, uint32_t Timeout)
{
  uint32_t timeout = Timeout;

  /* Polling on Ready/Busy signal */
```

```
  while((HAL_GPIO_ReadPin(NOR_READY_BUSY_GPIO, NOR_READY_BUSY_PIN) !=
NOR_BUSY_STATE) && (timeout > 0))
  {
    timeout--;
  }

  timeout = Timeout;

  /* Polling on Ready/Busy signal */
  while((HAL_GPIO_ReadPin(NOR_READY_BUSY_GPIO, NOR_READY_BUSY_PIN) !=
NOR_READY_STATE) && (timeout > 0))
  {
    timeout--;
  }
}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

3 Interfacing a non-multiplexed, asynchronous 16-bit SRAM

3.1 FMC configuration

SRAM and NOR Flash memories share the same FMC banks. The protocol to be used depends on the selected memory type.

To control an SRAM, FMC provides the following possible features:

- address/data multiplexing feature enable/disable
- memory type select (NOR/SRAM/PSRAM)
- external memory data bus width select (8/16 bits)
- extended mode enable/disable

  This mode is used to access the memory with a different timing configuration for read and write operations.

Like for NOR Flash memories, the user has to compute and set the following parameters as a function of the information in the SRAM datasheet:

ADDSET: address setup time

DATAST: data setup time

Figure 6 and *Figure 7* show the different timings for a typical SRAM access.

Figure 6. SRAM asynchronous read access timing

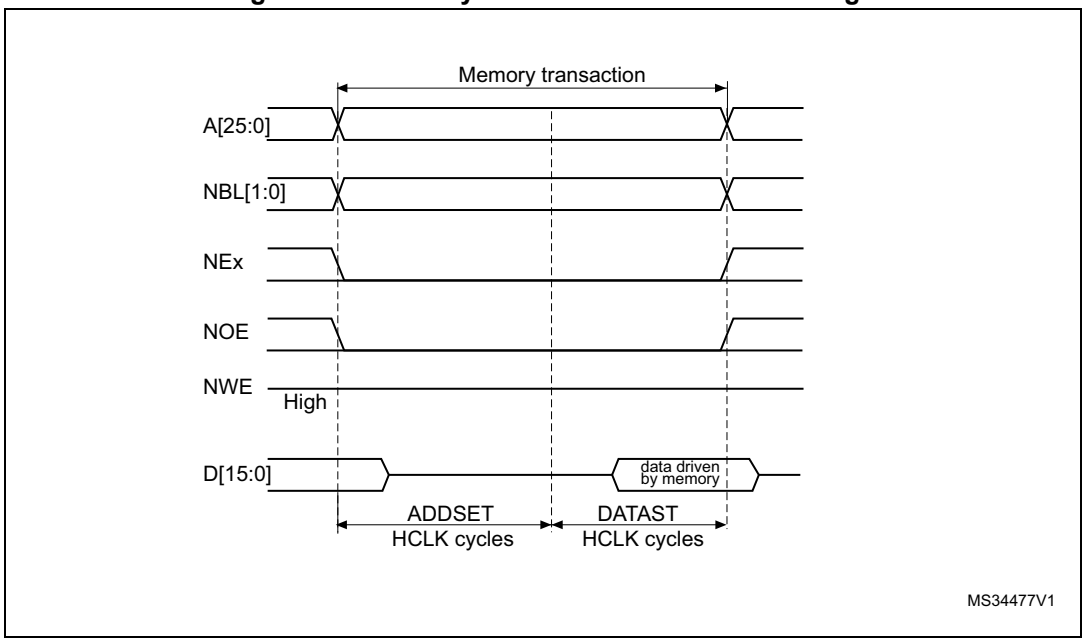
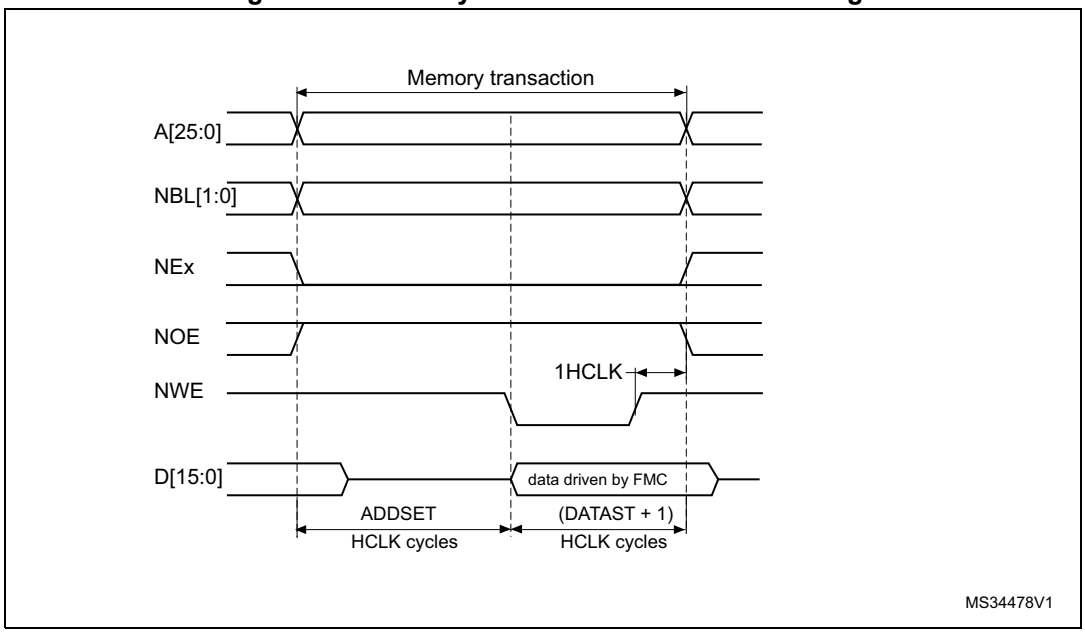


Figure 7. SRAM asynchronous write access timing



3.1.1 Typical use of FMC to interface with an SRAM

In this application note, the IS61WV102416BLL memory is used as the reference.

The IS61WV102416BLL memory is a non-multiplexed, asynchronous, 16-bit memory. Bank 1 - NOR/PSRAM sub-bank 1 is selected to support the SRAM device. Based on these data, FMC is configured as follows:

- Bank 1 - NOR/PSRAM sub-bank 1 is enabled: BCR1_MBKEN bit set to ‘1’.
- Memory type is SRAM: BCR1_MTYP is set to ‘00’ to select the SRAM memory type.
- Data bus width is 16 bits: BCR1_MWID is set to ‘01’ to select the 16-bit width.
- The memory is non-multiplexed: BCR1_MUXEN is reset.

All remaining parameters must be kept cleared.

3.2 Timing computation

The SRAM shares the same banks and configuration register as the NOR Flash memory. As a result, the timing computation method is the same as described in detail in the NOR Flash memory section (*Section 2.2: Timing computation on page 11*).

FMC is configured on the basis of the SRAM access timings illustrated in *Figure 6* and *Figure 7*, and taking into account the following:

- the maximum read/write access time
- the different internal FMC delays
- the different internal memory delays

Hence the following equations:

$(ADDSET + (DATAST + 1)) \times t_{HCLK} \geq \max(t_{WC})$

$DATAST \times t_{HCLK} = t_{PWE1}$

For read access, DATAST must verify:

$$DATAST \geq (t_{AA} + t_{su(Data\_NE)} + t_{v(A\_NE)})/t_{HCLK} - ADDSET$$

Table 6 gives the meanings and values of the SRAM parameters.

Table 6. IS61WV102416BL SRAM timings

Symbols	Parameter	Value	Unit
t_{WC}	Write cycle time	12	ns
t_{RC}	Read cycle time	12	ns
t_{PWE1}	Write Enable low pulse width	8	ns
t_{AA}	Address access time	12	ns

Using the above described formulas, the memory timings in *Table 6* and the asynchronous timings in *Table 3*, we have:

- address setup time: 0x1
- data setup time: 0x1

3.3 Hardware connection

Table 7 gives the correspondence between SRAM pins and FMC pins and shows the GPIO configuration for each FMC pin.

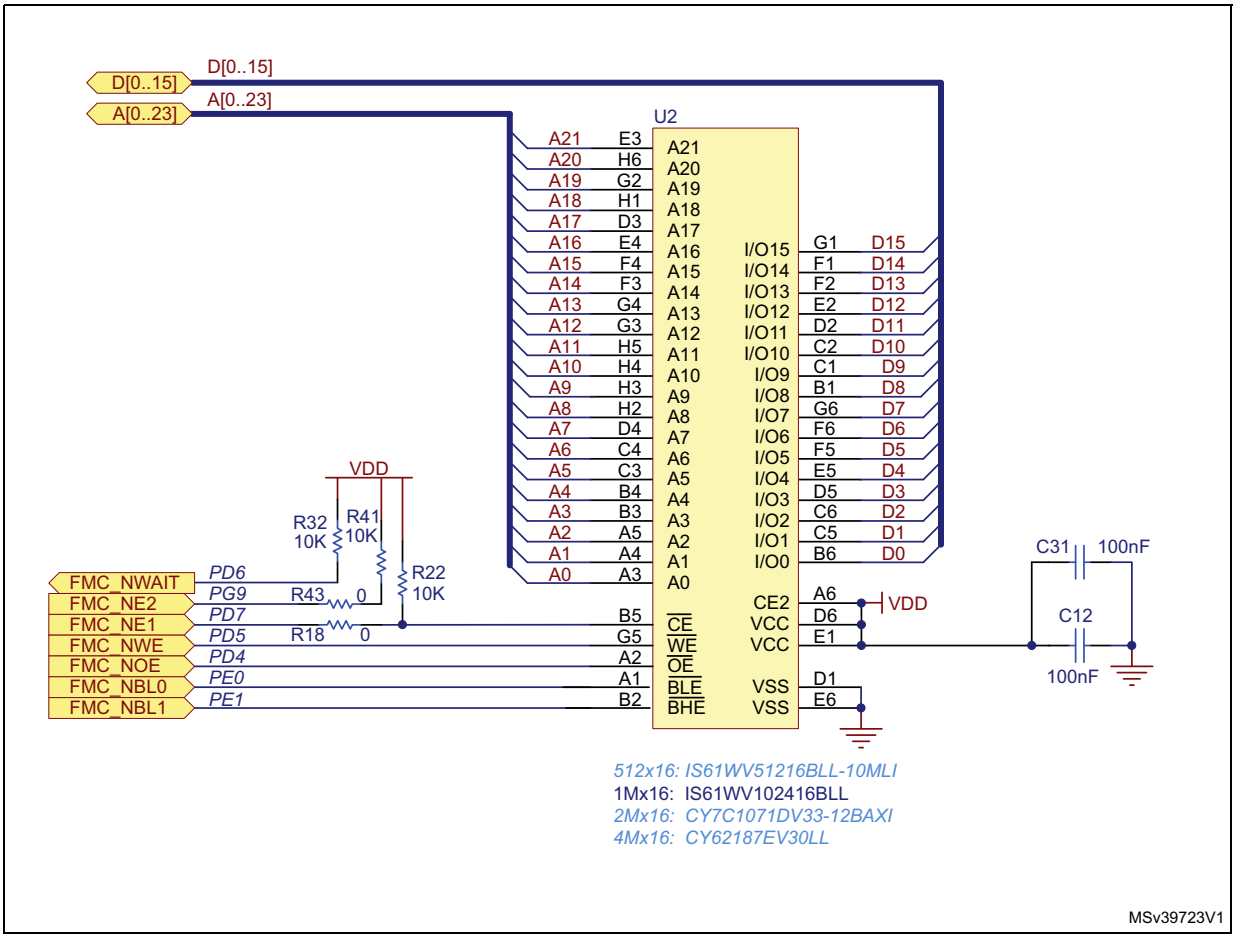
In case of an 8-bit SRAM, the data/address bus is 8 bits wide and D8-D15 should not be connected to FMC. The unused FMC pins can be used as general purpose I/O pins

Table 7. IS61WV102416BLL signal to FMC pin correspondence

Memory signals	FMC signals	Pin / Port assignment	Pin / Port configuration	Signal description
A0-A18	A0-A18	Port F/Port G/Port E/ Port D	AF push-pull	Address A0-A18
I/O0-I/O15	D0-D15	Port D/Port E	AF push-pull	Data D0-D15
$\overline{CE}$	NE1	PD7	AF push-pull	Chip Enable
$\overline{OE}$	NOE	PD4	AF push-pull	Output Enable
$\overline{WE}$	NWE	PD5	AF push-pull	Write Enable
$\overline{LB}$	NBL0	PE0	AF push-pull	Lower byte control
$\overline{UB}$	NBL1	PE1	AF push-pull	Upper byte control

Figure 8 shows a typical connection between an STM32L476/486 microcontroller and an IS61WV102416BLL SRAM.

Figure 8. 16-bit SRAM: IS61WV102416BLL connection to STM32L476/486



A firmware example is available hereafter.

The main goal of this example is to provide the basics of how to use the FMC firmware library and the associate SRAM driver to perform read/write operations on the IS61WV102416BLL SRAM memory.

```
/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Private define ------------------------------------------------------------*/
#define SRAM_BANK_ADDR          ((uint32_t)0x60000000)

/* #define SRAM_MEMORY_WIDTH    FMC_NORSRAM_MEM_BUS_WIDTH_8  */
#define SRAM_MEMORY_WIDTH       FMC_NORSRAM_MEM_BUS_WIDTH_16
/* #define SRAM_MEMORY_WIDTH    FMC_NORSRAM_MEM_BUS_WIDTH_32 */

#define SRAM_TIMEOUT            ((uint32_t)0xFFFF)
```

```
#define BUFFER_SIZE             ((uint32_t)16)
#define WRITE_READ_ADDR         ((uint32_t)0x0800)

/**
  * @brief  SRAM status definition
  */
#define SRAM_STATUS_OK           0x00
#define SRAM_STATUS_INIT_ERROR   0x01

/* Private variables ---------------------------------------------------------*/
static SRAM_HandleTypeDef hsram;
static FMC_NORSRAM_TimingTypeDef SRAM_Timing;

/* Read/Write Buffers */
uint16_t aTxBuffer[BUFFER_SIZE] = {0x00, 0x01, 0x02, 0x03,
                                   0x04, 0x05, 0x06, 0x07,
                                   0x08, 0x09, 0x0A, 0x0B,
                                   0x0C, 0x0D, 0x0E, 0x0F};
uint16_t aRxBuffer[BUFFER_SIZE] = {0};

/* Counter index */
uint32_t uwIndex = 0;

/* Test function -------------------------------------------------------------*/
int Test_SRAM(void)
{
  /*##-1- Configure the SRAM device ###################################*/
  hsram.Instance  = FMC_NORSRAM_DEVICE;
  hsram.Extended  = FMC_NORSRAM_EXTENDED_DEVICE;

  /* SRAM device configuration */
  SRAM_Timing.AddressSetupTime       = 1;
  SRAM_Timing.AddressHoldTime        = 1;
  SRAM_Timing.DataSetupTime          = 1;
  SRAM_Timing.BusTurnAroundDuration  = 0;
  SRAM_Timing.CLKDivision            = 2;
  SRAM_Timing.DataLatency            = 2;
  SRAM_Timing.AccessMode             = FMC_ACCESS_MODE_A;

  hsram.Init.NSBank             = FMC_NORSRAM_BANK1;
  hsram.Init.DataAddressMux     = FMC_DATA_ADDRESS_MUX_DISABLE;
  hsram.Init.MemoryType         = FMC_MEMORY_TYPE_SRAM;
  hsram.Init.MemoryDataWidth    = SRAM_MEMORY_WIDTH;
```

```
  hsram.Init.BurstAccessMode     = FMC_BURST_ACCESS_MODE_DISABLE;
  hsram.Init.WaitSignalPolarity = FMC_WAIT_SIGNAL_POLARITY_LOW;
  hsram.Init.WaitSignalActive    = FMC_WAIT_TIMING_BEFORE_WS;
  hsram.Init.WriteOperation      = FMC_WRITE_OPERATION_ENABLE;
  hsram.Init.WaitSignal          = FMC_WAIT_SIGNAL_DISABLE;
  hsram.Init.ExtendedMode        = FMC_EXTENDED_MODE_DISABLE;
  hsram.Init.AsynchronousWait    = FMC_ASYNCHRONOUS_WAIT_DISABLE;
  hsram.Init.WriteBurst          = FMC_WRITE_BURST_DISABLE;
  hsram.Init.ContinuousClock     = FMC_CONTINUOUS_CLOCK_SYNC_ONLY;
  hsram.Init.WriteFifo           = FMC_WRITE_FIFO_DISABLE;
  hsram.Init.PageSize            = FMC_PAGE_SIZE_NONE;

  /* Initialize the SRAM controller */
  if(HAL_SRAM_Init(&hsram, &SRAM_Timing, &SRAM_Timing) != HAL_OK)
  {
    return SRAM_STATUS_INIT_ERROR;
  }

  /*##-2- SRAM memory read/write access ############################*/
  /* Write data to the SRAM memory */
  for(uwIndex = 0; uwIndex < BUFFER_SIZE; uwIndex++)
  {
    *(__IO uint16_t *)(SRAM_BANK_ADDR + WRITE_READ_ADDR + 2 * uwIndex) =
aTxBuffer[uwIndex];
  }

  /* Read back data from the SRAM memory */
  for(uwIndex = 0; uwIndex < BUFFER_SIZE; uwIndex++)
  {
    aRxBuffer[uwIndex] = *(__IO uint16_t *)(SRAM_BANK_ADDR +
WRITE_READ_ADDR + 2 * uwIndex);
  }

  /* aRxBuffer shall be identical to aTxBuffer */

  return SRAM_STATUS_OK;
}

/**
  * @brief SRAM MCU Support Package Initialization
  *        This function configures the hardware resources used in this
example:
  *           - Peripheral's clock enable
  *           - Peripheral's GPIO Configuration
  * @param hnor: NOR handle pointer
  * @retval None
```

```
  */
void HAL_SRAM_MspInit(SRAM_HandleTypeDef *hsram)
{
  GPIO_InitTypeDef gpio;

  /* Enable FMC clock */
  __HAL_RCC_FMC_CLK_ENABLE();

  /* Enable GPIOs clock */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_PWREx_EnableVddIO2();

  /* Common GPIO configuration */
  gpio.Mode      = GPIO_MODE_AF_PP;
  gpio.Pull      = GPIO_PULLUP;
  gpio.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
  gpio.Alternate = GPIO_AF12_FMC;

  /*## Data Bus #######*/
  /* GPIOD configuration */
  gpio.Pin   = GPIO_PIN_0  | GPIO_PIN_1  | GPIO_PIN_8 | GPIO_PIN_9 | \
               GPIO_PIN_10 | GPIO_PIN_14 | GPIO_PIN_15;
  HAL_GPIO_Init(GPIOD, &gpio);

  /* GPIOE configuration */
  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;
  HAL_GPIO_Init(GPIOE, &gpio);

  /*## Address Bus #######*/
  /* GPIOF configuration */
  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;
  HAL_GPIO_Init(GPIOF, &gpio);

  /* GPIOG configuration */
  gpio.Pin   = GPIO_PIN_0  | GPIO_PIN_1  | GPIO_PIN_2  | \
               GPIO_PIN_3  | GPIO_PIN_4  | GPIO_PIN_5  | \
               GPIO_PIN_13 | GPIO_PIN_14;
```

```
  HAL_GPIO_Init(GPIOG, &gpio);

  /* GPIOD configuration */
  gpio.Pin   = GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13;
  HAL_GPIO_Init(GPIOD, &gpio);

  /* GPIOE configuration */
  gpio.Pin   = GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6;
  HAL_GPIO_Init(GPIOE, &gpio);

  /*## NOE and NWE configuration #######*/
  gpio.Pin = GPIO_PIN_4 | GPIO_PIN_5;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## Enable Bank pin configuration #######*/
  gpio.Pin = GPIO_PIN_7;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## NE3 configuration #######*/
  gpio.Pin = GPIO_PIN_10;
  HAL_GPIO_Init(GPIOG, &gpio);

  /*## NBL0, NBL1 configuration #######*/
  gpio.Pin = GPIO_PIN_0 | GPIO_PIN_1;
  HAL_GPIO_Init(GPIOE, &gpio);
}
```

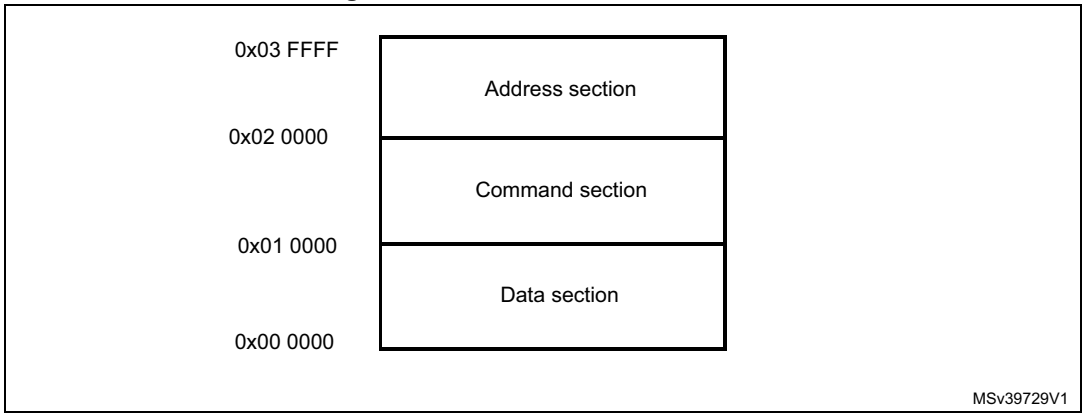
4 Interfacing an 8-bit NAND Flash memory

NAND Flash memories are accessed in accordance with a specific protocol. To write to or read from the NAND Flash memory, it is necessary to:

1. Send a command to the NAND Flash memory
2. Send the address to write to or read from
3. Read or write the data

The FMC NAND banks are divided into three sections to allow the user to easily program the NAND Flash memory: data section, address section, and command section.

Figure 9. FMC NAND bank sections



In fact, the three sections are the representation of the real NAND Flash memory. By writing to any location in the command section, the user writes the command to the NAND Flash memory. By writing to any location in the address section, the user specifies the address of the read or write operation. Depending on the structure of the used NAND Flash memory, four or five operations are required to write the address. By writing to or reading from any location in the data section, the data are written to or read from the address previously sent to the address section.

4.1 FMC configuration

To control a NAND Flash memory, FMC provides the following possible features:

- memory Ready/$\overline{\text{Busy}}$ signal use enable/disable as the wait input for FMC
- memory Ready/$\overline{\text{Busy}}$ signal use enable/disable as the interrupt source for FMC

  The interrupt can be generated with three possible configurations:
  - on the rising edge of the Ready/$\overline{\text{Busy}}$ signal: when the memory has just completed an operation and the new status in ready
  - on the falling edge of the Ready/$\overline{\text{Busy}}$ signal: when the memory starts the new operation
  - on the high level of the Ready/$\overline{\text{Busy}}$ signal: when the memory is ready
- NAND Flash memory data bus width select (8/16bit)
- ECC computation logic enable/disable
- ECC page size select (256, 512, 1024, 2048, 4096 or 8192 bytes)

FMC also provides the possibility of programming the timings for the different NAND Flash memory sections separately: common section and attribute section. The timings are the following:

- **Setup time**

  It is the time (in HCLK) required to set up the address before the command assertion. That is, It is the time from address valid to the start of the read or write operation.
- **Wait time**

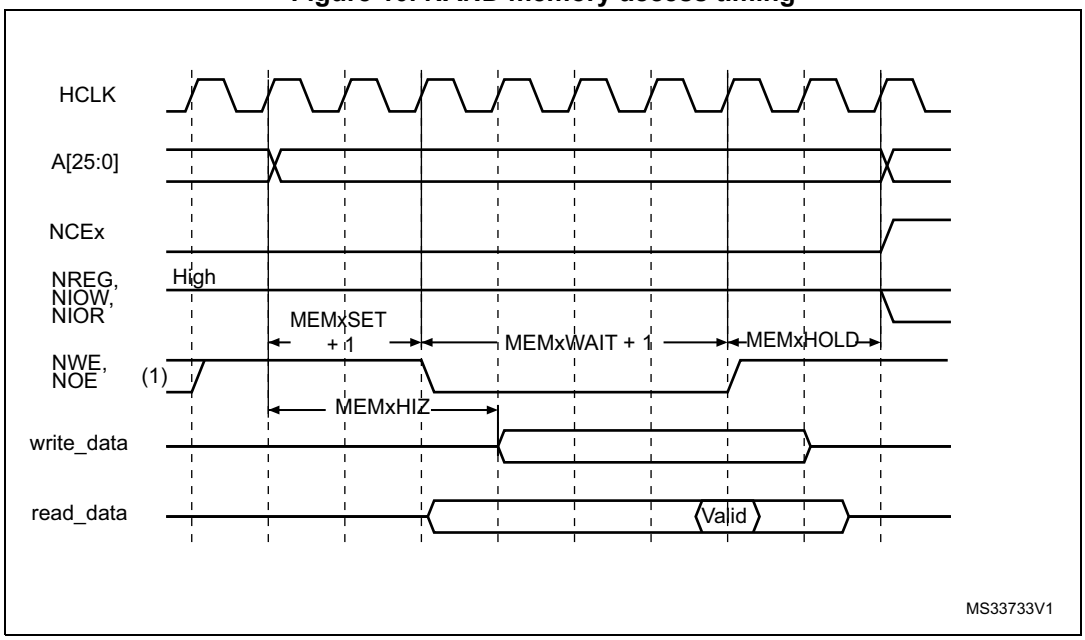
  It is the time (in HCLK) required to assert the command. That is, it is the time taken by the NOE and NWE signals to become de-asserted.
- **Hold time**

  It is the time (in HCLK) during which the address is held after the command deassertion. That is, it is the time between the deassertion of the NOE and NWE signals and the end of the operation cycle.
- **Data-bus HiZ time**

  It is only valid for write operations and corresponds to the time (in HCLK) during which the data bus is kept in the high-impedance state after the start of a write access. That is, it is the time from address valid to data bus driven.

Figure 10 shows the different timings for a typical NAND Flash memory access.

Figure 10. NAND memory access timing



1. NOE remains high (inactive) during write accesses. NWE remains high (inactive) during read accesses.

4.1.1 Typical use of FMC to interface with a NAND memory

The STM32L476/486 FMC NAND Flash memory controller can configure Bank3 to support NAND Flash memories.

The banks are selected using the Chip Select signals, as each bank is associated with a specific Chip Select.

To enable communication with the NAND Flash memory devices, the FMC NAND Flash memory controller has to be initialized to meet the characteristics of the NAND Flash memory devices: features, timings, data width, etc.

In this application note, the Numonyx NAND512W3A is used as the reference. This memory has the same access protocol as many other NAND Flash memories on today's market.

NAND512W3A characteristics:

- NAND interface: x8 bus width, multiplexed address/ data
- page size - x8 device: (512 + 16 spare) bytes
- page Read/Program timings:
  - random access: 12 µs (3 V)/15 µs (1.8 V) (max.)
  - sequential access: 30 ns (3 V)/50 ns (1.8 V) (min.)
  - page program time: 200 µs (typ.)

Bank3 is selected to support the NAND Flash memory device. Based on these data, FMC is configured as follows:

- Bank3 is enabled: PCR_PBKEN bit set to '1'.
- Memory type is NAND Flash memory: PCR_PTYP is set to '1' to select the NAND Flash memory type.
- Data bus width is 8 bit: PCR_PWID is set to '00' to select 8-bit width.
- ECC page size is 512 bytes: PCR_ECCPS is set to '001' to set the ECC computation page size to 512 bytes.
- ECC hardware calculation on/off as needed: PCR_ECCEN is set or reset accordingly.
- Wait feature may or not be enabled depending on the user's application: PCR_PWAITEN is set or reset as needed.

The Ready/$\overline{\text{Busy}}$ memory signal can be connected to the FMC_NWAIT pin, and in this case the Wait feature must be used to manage the NAND Flash memory operations.

When using the NAND Flash memory with the wait feature enabled, the controller waits for the NAND Flash memory to be ready to become active before starting a new access. While waiting, the controller maintains the NCE signal active (low).

Generally, the Ready/$\overline{\text{Busy}}$ signal is an open-drain output. To connect this signal to the STM32L476/486 microcontroller, the corresponding pin must be configured as input pull-up.

The Ready/$\overline{\text{Busy}}$ signal can be used as an interrupt source for FMC and, in this case, the CPU can perform other tasks during NAND Flash memory operations.
Three FMC configurations make it possible to use this signal as an interrupt. For that purpose, the IREN, IFEN or ILEN bits in the SR register are used to select the rising edge, the falling edge or the high level of the NAND Flash memory Ready/$\overline{\text{Busy}}$ signal.

4.2 Timing computation

Besides configuring the different features that are to be used with the NAND Flash memory, the user has to initialize the controller to meet the memory timings.

As described in *Section 4.1*, FMC is able to program four different timings for the common space and the attribute space independently: **Setup time, Wait time, Hold time and Data-bus HiZ time**.

These parameters are computed according to the NAND Flash memory characteristics and the STM32L476/486 HCLK clock.

Based on the NAND Flash memory access timings shown in *Figure 10*, the following equations are found:

- The write or read access time is the time between the falling edge and the rising edge of the NAND Flash memory Chip Select signal. It is computed as a function of the FMC timing parameter:

  $((SET + 1) + (WAIT + 1) + (HOLD)) \times t_{HCLK} \geq$ Write/Read access
- The Wait time is measured between the falling edge and the rising edge of the Write/Read Enable signal:

  Write/Read Enable signal low to high = $(WAIT + 1) \times t_{HCLK}$
- For write access, the HIZ parameter is the time measured between the falling edge of the Chip Select signal and the data setup on bus:

  $(HIZ) \times t_{HCLK} \geq$ Chip Select setup time to Data setup.
- The HOLD timing is given in the NAND datasheet as follows:

  $(HOLD) \times t_{HCLK}$ = Write Enable High to Chip Enable / Address/Command Latch High

To make sure of the correct timing configuration of FMC, the timings have to take into consideration:

- the maximum read/write access time
- the different internal FMC delays
- the different internal memory delays

Hence, we have the following equations following the NAND512W3A NAND datasheet:

- $(SET + 1) \times t_{HCLK} \geq \max(t_{CS}, t_{CLS}, t_{ALS}, t_{CLR}, t_{AR}) - t_{WP}$
- $(WAIT + 1) \times t_{HCLK} \geq \max(tWP, tRP)$
- $(HIZ) \times t_{HCLK} \geq \max(t_{CS}, t_{ALS}, t_{CLS}) + (t_{WP} - t_{DS})$
- $(HOLD) \times t_{HCLK} \geq \max(t_{CH}, t_{CLH}, t_{ALH})$
- $((WAIT + 1) + (HOLD) + (SET + 1)) \times t_{HCLK} \geq \max(t_{WC/RC})$

Considering the different timings in FMC and the memory, the equations become:

- WAIT must verify:

$(WAIT + 1) \times t_{HCLK} \geq (t_{REA} + t_{su(D-NOE)})$

$WAIT \geq (t_{REA} + t_{su(D-NOE)}) / t_{HCLK} - 1$

Note: *$t_{su(D-NOE)}$ is specified in the STM32L476/486 datasheets.*
$t_{su(D-NOE)}$ = 25 ns

Table 8 gives the meanings and values of the NAND512W3A2C memory parameters.

Table 8. NAND512W3A2C Flash memory timings

Symbol	Parameter	Value	Unit
t_{CEA}	Chip Enable low to output valid	35	ns
t_{WP}	Write Enable low to Write Enable high	15	ns
t_{RP}	Read Enable low to Read Enable high	15	ns
t_{CS}	Chip Enable low to Write Enable high	20	ns
t_{ALS}	AL setup time	15	ns
t_{CLS}	CL Setup time	15	ns
t_{CH}	$\overline{E}$ Hold time	5	ns
t_{ALH}	AL Hold time	5	ns
t_{CLH}	CL Hold time	5	ns
t_{DS}	Data Valid to Write Enable high	15	ns
t_{WC}	Write Enable low to Write Enable low	30	ns
t_{RC}	Read Enable low to Read Enable low	30	ns
t_{REA}	Read Enable low to Output Valid	18	ns

Using the above described formulas, the memory timings in *Table 8* and the asynchronous timings in *Table 3*, we have:

- Setup time: 0x0
- Wait time: 0x3
- Hold time: 0x1
- HiZ time: 0x2

4.3 Hardware connection

Table 9 gives the correspondence between the NAND Flash memory pins and the FMC pins and shows the GPIO configuration for each FMC pin.

In case of a 16-bit NAND Flash memory, the data/address bus is 16 bits wide and the remaining FMC data/address bus are used.

Table 9. NAND512W3A signal to FMC pin correspondence

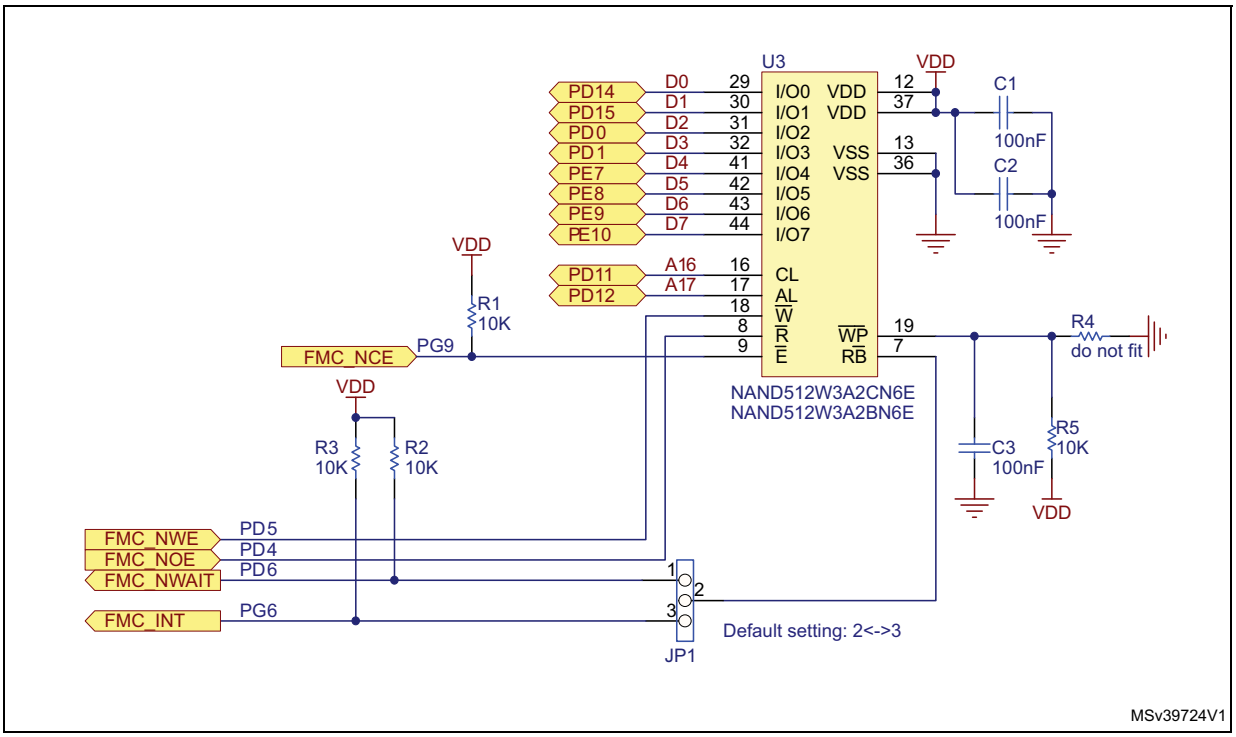
Memory signals	FMC signals	Pin / Port assignment	Pin / Port configuration	Signal description
AL	ALE/A17	PD11	AF push-pull	Address Latch Enable
CL	CLE/A16	PD12	AF push-pull	Command Latch Enable
I/O0-I/O7	D0-D7	Port D/Port E	AF push-pull	Data D0-D7
$\overline{E}$	NCE	PG9	AF push-pull	Chip Enable
$\overline{R}$	NOE	PD4	AF push-pull	Output Enable

Table 9. NAND512W3A signal to FMC pin correspondence (continued)

Memory signals	FMC signals	Pin / Port assignment	Pin / Port configuration	Signal description
$\overline{W}$	NWE	PD5	AF push-pull	Write Enable
$R\overline{B}$	NWAIT/INT	PD6/PG6	Input pull-up	Ready/$\overline{Busy}$ signal

Figure 11 illustrates a typical connection between the STM32L476/486 microcontroller and the NAND512W3A memory.

Figure 11. 8-bit NAND Flash memory: NAND512W3A2C/NAND512W3A2B connection to STM32L476/486



A firmware example is available hereafter.

The main goal of this example is to provide the basics of how to use the FMC firmware library and the associated NAND Flash memory driver to perform erase/read/write operations using the FMC wait feature on the NAND512W3A2 memory.

```
/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Private define ------------------------------------------------------------*/
#define NAND_DEVICE_ADDR   ((uint32_t)0x70000000)
#define BUFFER_SIZE        ((uint32_t)16)
#define WRITE_READ_ADDR    ((uint32_t)0x20800)
```

```
/**
  * @brief  NOR status definition
  */
#define NAND_STATUS_OK            0x00
#define NAND_STATUS_INIT_ERROR    0x01
#define NAND_STATUS_READ_ID_ERROR 0x02
#define NAND_STATUS_ERASE_ERROR   0x03
#define NAND_STATUS_WRITE_ERROR   0x04
#define NAND_STATUS_READ_ERROR    0x05

/* Private variables ---------------------------------------------------------*/
static NAND_HandleTypeDef hNAND;                    /* NAND handle */
static FMC_NAND_PCC_TimingTypeDef NAND_ComTiming; /* Common memory space timing    */
static FMC_NAND_PCC_TimingTypeDef NAND_AttTiming; /* Attribute memory space timing */
static NAND_IDTypeDef NAND_ID = {0};

uint8_t aTxBuffer[BUFFER_SIZE] = {0x00, 0x01, 0x02, 0x03,
                                  0x04, 0x05, 0x06, 0x07,
                                  0x08, 0x09, 0x0A, 0x0B,
                                  0x0C, 0x0D, 0x0E, 0x0F};
uint8_t aRxBuffer[BUFFER_SIZE]= {0};

/* Test function -------------------------------------------------------------*/
int Test_NAND(void)
{
  NAND_AddressTypedef Address;

  /*##-1- Configure the NAND device ###################################*/
  hNAND.Instance = FMC_NAND_DEVICE;

  /* NAND device configuration */
  NAND_ComTiming.SetupTime = 0x00;
  NAND_ComTiming.WaitSetupTime = 3;
  NAND_ComTiming.HoldSetupTime = 1;
  NAND_ComTiming.HiZSetupTime = 2;
  NAND_AttTiming.SetupTime = 1;
  NAND_AttTiming.WaitSetupTime = 3;
  NAND_AttTiming.HoldSetupTime = 1;
  NAND_AttTiming.HiZSetupTime = 2;

  hNAND.Init.NandBank = FMC_NAND_BANK3;
  hNAND.Init.Waitfeature = FMC_NAND_PCC_WAIT_FEATURE_ENABLE;
```

```
hNAND.Init.MemoryDataWidth = FMC_NAND_PCC_MEM_BUS_WIDTH_8;
hNAND.Init.EccComputation = FMC_NAND_ECC_ENABLE;
hNAND.Init.ECCPageSize = FMC_NAND_ECC_PAGE_SIZE_512BYTE;
hNAND.Init.TCLRSetupTime = 0;
hNAND.Init.TARSetupTime = 0;
hNAND.Info.BlockNbr = 0x04;
hNAND.Info.BlockSize = 0x20;
hNAND.Info.ZoneSize = 0x400;
hNAND.Info.PageSize = 0x200;
hNAND.Info.SpareAreaSize = 0x10;

/* NAND controller initialization */
if(HAL_NAND_Init(&hNAND, &NAND_ComTiming, &NAND_AttTiming) != HAL_OK)
{
  return NAND_STATUS_INIT_ERROR;
}

/*##-2- Read & check the NAND device IDs ##########################*/
/* Read the NAND memory ID */
if(HAL_NAND_Read_ID(&hNAND, &NAND_ID) != HAL_OK)
{
  return NAND_STATUS_READ_ID_ERROR;
}

/*##-3- Erase NAND memory ##########################################*/
Address.Page = 3;
Address.Zone = 1;
Address.Block = 5;

/* Send NAND erase block operation */
if (HAL_NAND_Erase_Block(&hNAND, &Address) != HAL_OK)
{
  return NAND_STATUS_ERASE_ERROR;
}

/*##-4- NAND memory read/write access ##############################*/
/* Write data to the NAND memory */
if (HAL_NAND_Write_Page(&hNAND, &Address, aTxBuffer, 1) != HAL_OK)
{
  return NAND_STATUS_WRITE_ERROR;
}
/* Read back data from the NAND memory */
if (HAL_NAND_Read_Page(&hNAND, &Address, aRxBuffer, 1) != HAL_OK)
{
  return NAND_STATUS_READ_ERROR;
```

```
  }

  return NAND_STATUS_OK;
}

/**
  * @brief NAND MCU Support Package Initialization
  *        This function configures the hardware resources used in this
example:
  *           - Peripheral's clock enable
  *           - Peripheral's GPIO Configuration
  * @param hnor: NOR handle pointer
  * @retval None
  */
void HAL_NAND_MspInit(NOR_HandleTypeDef *hnor)
{
  GPIO_InitTypeDef gpio;

  /* Enable FMC clock */
  __HAL_RCC_FMC_CLK_ENABLE();

  /* Enable GPIOs clock */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_PWREx_EnableVddIO2();

  /* Common GPIO configuration */
  gpio.Mode      = GPIO_MODE_AF_PP;
  gpio.Pull      = GPIO_PULLUP;
  gpio.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
  gpio.Alternate = GPIO_AF12_FMC;

  /*## Data Bus #######*/
  /* GPIOD configuration */
  gpio.Pin   = GPIO_PIN_0  | GPIO_PIN_1  | GPIO_PIN_8 | GPIO_PIN_9 | \
               GPIO_PIN_10 | GPIO_PIN_14 | GPIO_PIN_15;
  HAL_GPIO_Init(GPIOD, &gpio);

  /* GPIOE configuration */
  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;
  HAL_GPIO_Init(GPIOE, &gpio);
```

```
  /*##  Command Latch Enable - Address Latch Enable ##*/
  gpio.Pin = GPIO_PIN_11 | GPIO_PIN_12;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## NOE and NWE configuration #######*/
  gpio.Pin = GPIO_PIN_4 | GPIO_PIN_5;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## NCE pin configuration   #######*/
  gpio.Pin = GPIO_PIN_9;
  HAL_GPIO_Init(GPIOG, &gpio);

  /*## NWAIT NAND pin configuration #####*/
  gpio.Pin = GPIO_PIN_6;
  HAL_GPIO_Init(GPIOD, &gpio);

  /*## INT3 NAND pin configuration #####*/
  gpio.Pin = GPIO_PIN_7;
  HAL_GPIO_Init(GPIOG, &gpio);
}
```

4.4 Error correction code computation

4.4.1 Error correction code (ECC) computation overview

The FMC NAND Flash memory controller includes two pieces of error correction code computation hardware, one for each NAND Flash memory block.

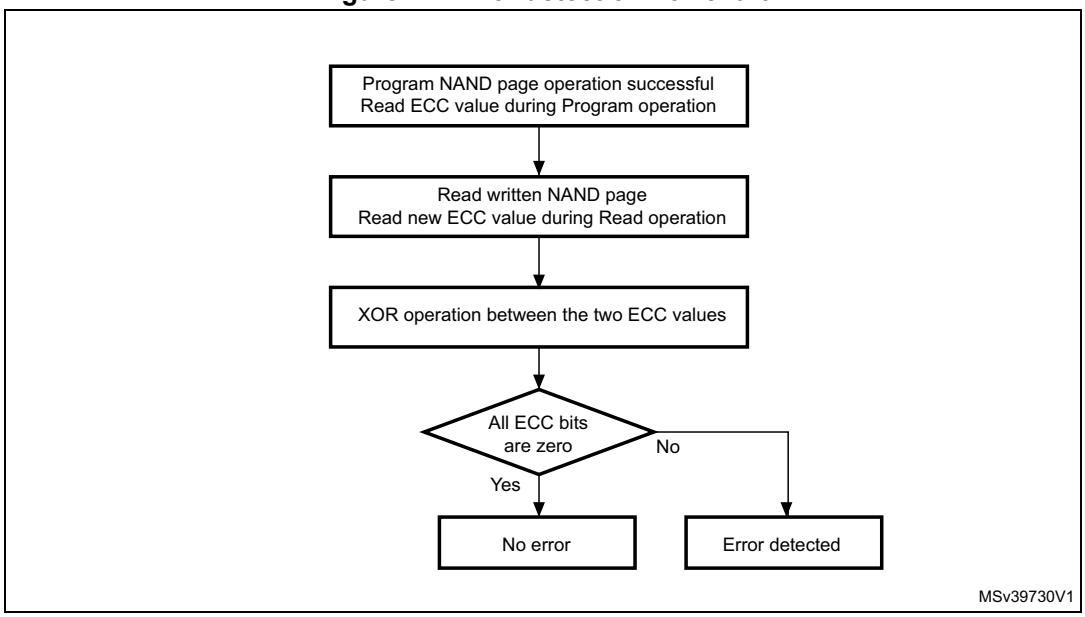
The ECC can be performed for page sizes of 256, 512, 1024, 2048, 4096 or 8192 bytes, depending on the ECC page size configured by the user. Depending on the configured page size, the ECC code will be 22, 24, 26, 28, 30 or 32 bits.

To even improve the error coverage, the user can read/write the NAND Flash memory page with a reduced ECC page size. This is possible when starting and stopping the ECC computation after the desired number of bytes to check. In this case, the ECC code is only calculated for the bytes written and read.

The error correction code algorithm implemented in FMC can perform 1-bit error correction and 2-bit error detection per page read from or written to the NAND Flash memory. It is based on the Hamming algorithm and consists in calculating the row and column parity.

4.4.2 Error detection

Figure 12. Error detection flowchart



When an error occurs during the write operation, this error is either correctable or uncorrectable depending on the ECC XOR operation:

- **Case of a correctable error**

  The ECC XOR operation contains 11-bit data at 1. And each pair parity is 0x10 or 0x01.
- **Case of an ECC error**

  The ECC XOR operation contains only one bit at 1.
- **Case of an uncorrectable error**

  The ECC XOR operation is random data. In this case the page data cannot be corrected.

Based on the flowchart shown in *Figure 12*, the correction software is easy to implement.

The first step consists in detecting whether an error occurred during the write operation. If that was the case, the second step consists in determining if the error is correctable or not. If it is correctable, then the third step consists in correcting the error.

The error correction is based on the second ECC generated after the read operation. The error location can be identified from this code. Usually, the following data are extracted from the ECC:

P1024, P512, P256, P128, P64, P32, P16, P8, P4, P2, P1, where Px are the line and column parity.

In case of an 8-bit memory, P4, P2, P1 define the error bit position. And P1024, P512, P256, P128, P64, P32, P16, P8 define the error byte position.

5 STM32L476/486 FMC configuration in 100-pin packages

FMC is present in devices delivered in both 144-pin and 100-pin packages. For devices in 100-pin packages, however, only some FMC banks can be used because not all pins are available.

5.1 Interfacing FMC with a NOR Flash memory

In devices that come in 100-pin packages, only Bank 1 (NOR/PSRAM 1) can be used to interface with a NOR Flash memory. This is because the NE2, NE3 and NE4 pins are not available in these packages.

Likewise, the A0-A15 pins are not available, so the NOR Flash controller should be used in multiplexed mode to use the data bus for both address and data.

In 8-bit muxed mode, the FMC address signals A[15:8] are available on the AD[15:8] pins.

Table 10 shows how to connect a NOR Flash memory to the FMC peripheral of devices delivered in 100-pin packages.

Table 10. NOR Flash memory connection to FMC

8-/16-bit NOR/PSRAM pins	FMC pins	100-pin package
A[15:0]	A[15:0]	DA[15:0]
A[23:16]	A[23:16]	A[23:16]
DQ[15:0]	D[15:0]	D[15:0]
$\overline{W}$	NWE	NWE
$\overline{E}$	NE1/NE2/NE3/NE4	NE1
$\overline{G}$	NOE	NOE
ADV	NADV	NADV
R/B	NWAIT	NWAIT
UB, LB	NBL[1:0]	NBL[1:0]

6 Revision history

Table 11. Document revision history

Date	Revision	Changes
23-Oct-2015	1	Initial release.