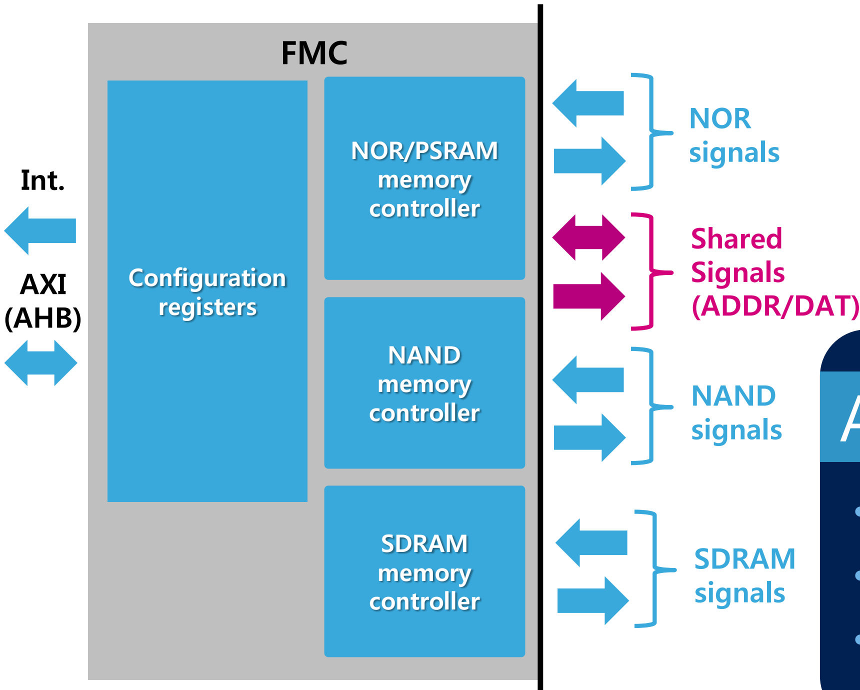
STM32 External Memory Control

ST Korea MCD Team

life.augmented

FMC (Flexible Memory Controller)

Overview



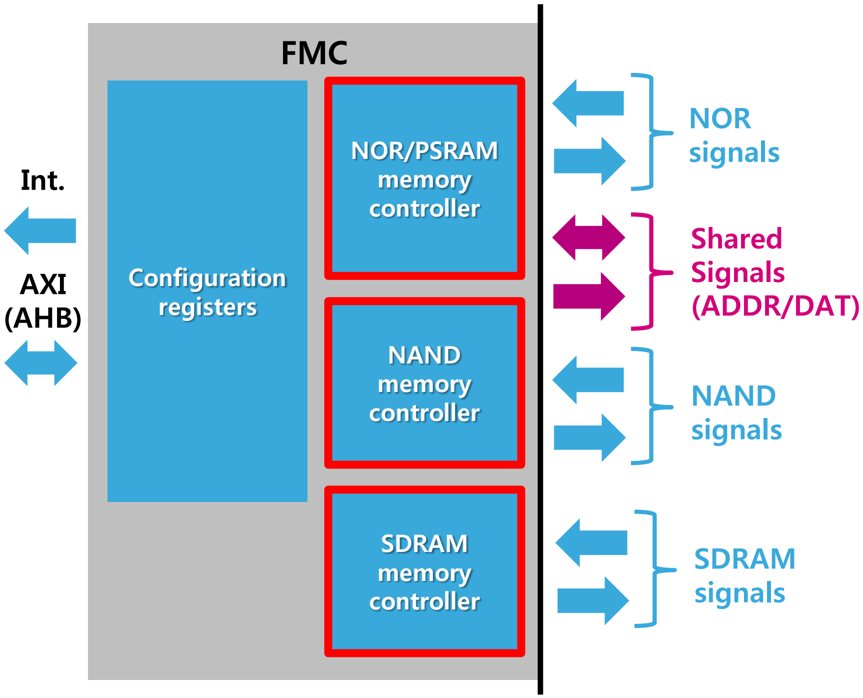
- FMC supports external memory via
  - NOR Flash/PSRAM controller
  - NAND memory controller
  - SDRAM memory controller

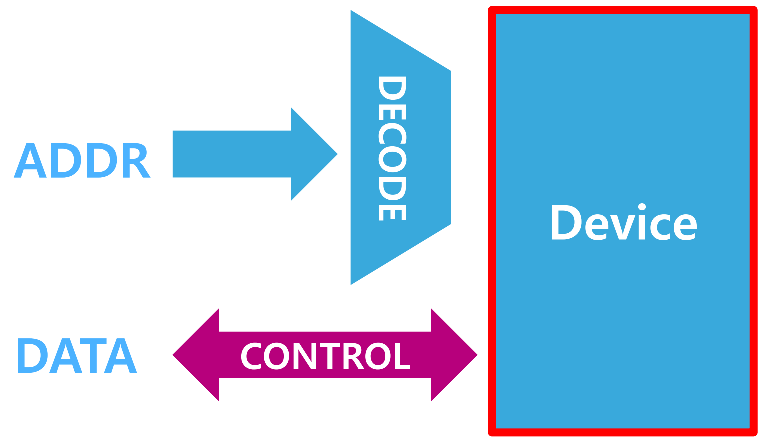
Application benefits

- RAM extension
- Flash memory extension
- Parallel interface (Intel 8080 / Motorola 6800)

Overview

- FMC supports external memory via
  - NOR Flash/PSRAM controller
  - NAND memory controller
  - SDRAM memory controller





Key features

- Fully independent banks
  - Four banks to support separate external memories
  - Independent Chip Select for each memory bank
  - Independent configuration for each memory bank

- Flexible configuration
  - FMC external access frequency is up to HCLK/2
  - Programmable timings to support a wide range of devices
  - 8- ,16- or 32-bit data bus
  - External asynchronous wait control
  - Extended mode (read timings and protocol different to write timings)
  - Supports burst mode access to synchronous devices (NOR Flash and PSRAM)
  - Write FIFO with 16 x32-bit depth

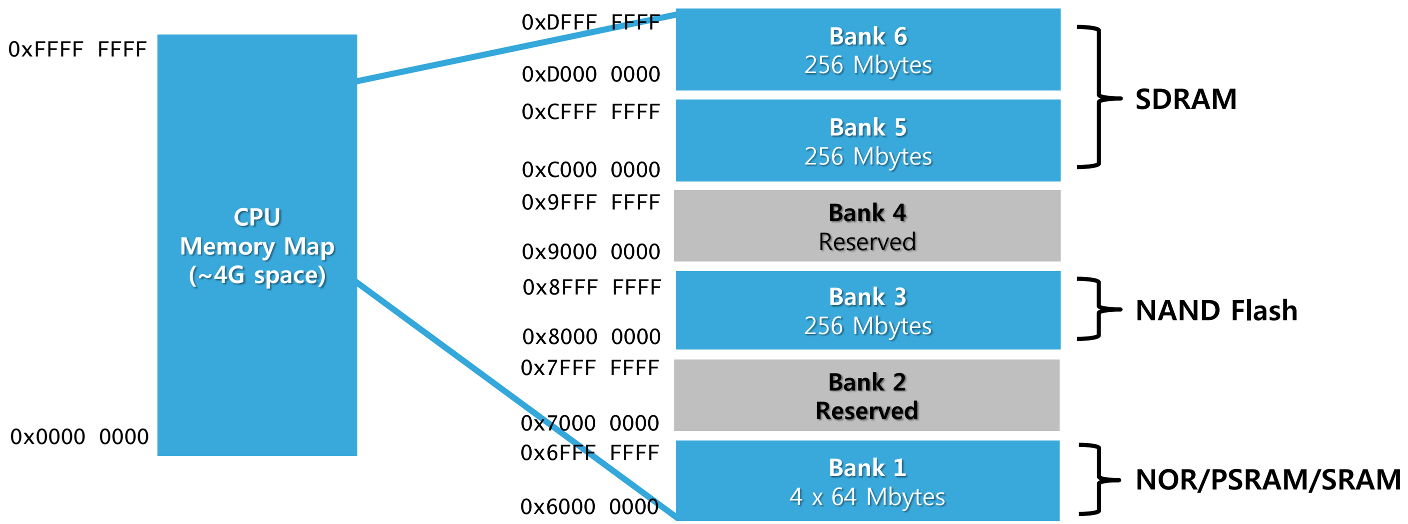
Supported devices

Compatible with a wide variety of interfaces and memories

- Static memory-mapped devices including
  - Static random access memory (SRAM)
  - Read-only memory (ROM)
  - NOR / OneNAND Flash memory
  - PSRAM
- NAND Flash memory
  - Includes ECC hardware to check up to 8 Kbytes of data read/written
  - 3 possible interrupt sources (level, rising edge and falling edge)
- SDRAM memory
  - Interfaces with Synchronous DRAM (SDRAM) memory-mapped

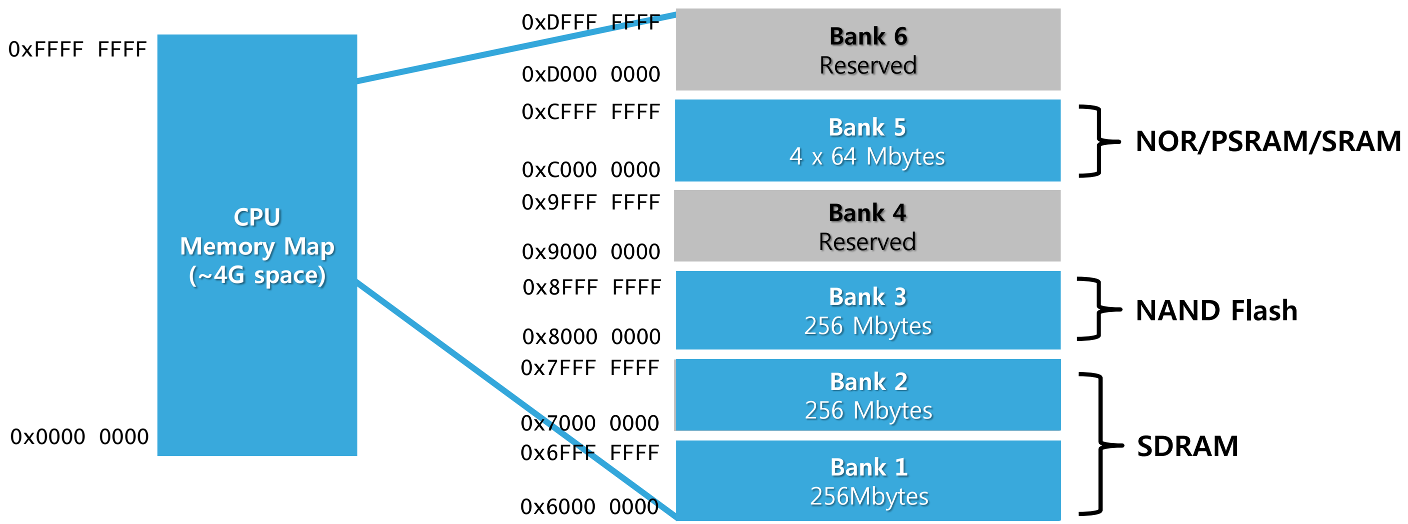
FMC Bank memory mapping

- External memories are divided into 4 fixed-size banks
  - Bank 1 (4 x 64 Mbytes) for NOR Flash, SRAM or PSRAM
  - Bank 3 (256 Mbytes) for NAND Flash
  - Banks 2 is used for SDRAM bank remap and Bank 4 is reserved.
  - Banks 5 & 6 for SDRAM



FMC Bank memory mapping

- External memories are divided into 4 fixed-size banks
  - Bank 1 (4 x 64 Mbytes) for NOR Flash, SRAM or PSRAM
  - Bank 3 (256 Mbytes) for NAND Flash
  - Banks 2 is used for SDRAM bank remap and Bank 4 is reserved.
  - Banks 5 & 6 for SDRAM

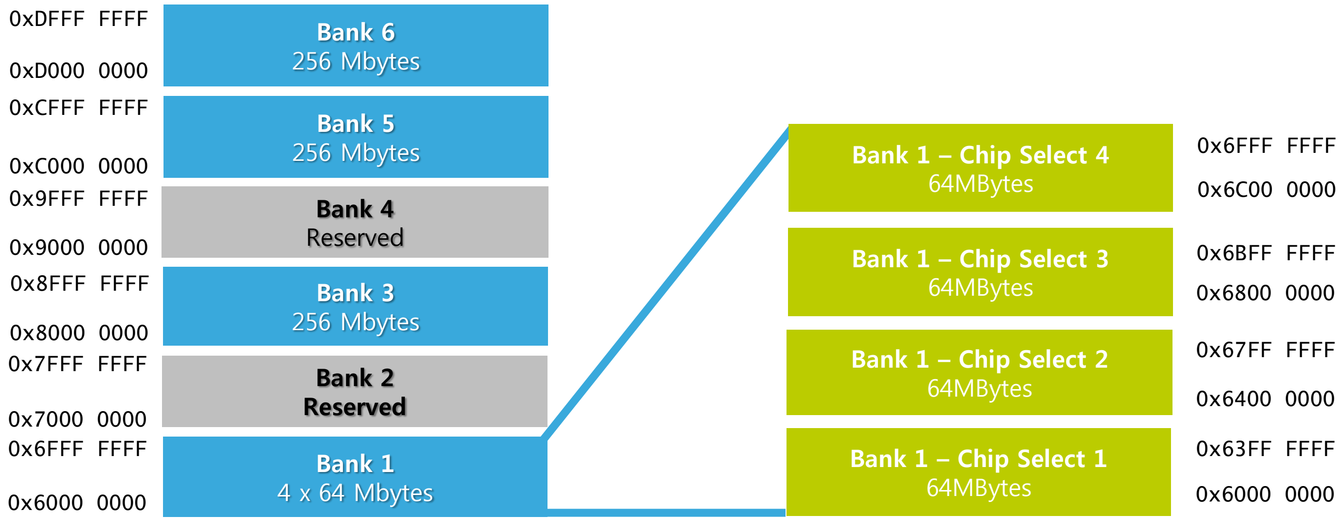


0x6000 0000번지로 SWAP하여 코드 수행 가능

life.augmented

NOR/PSRAM address mapping

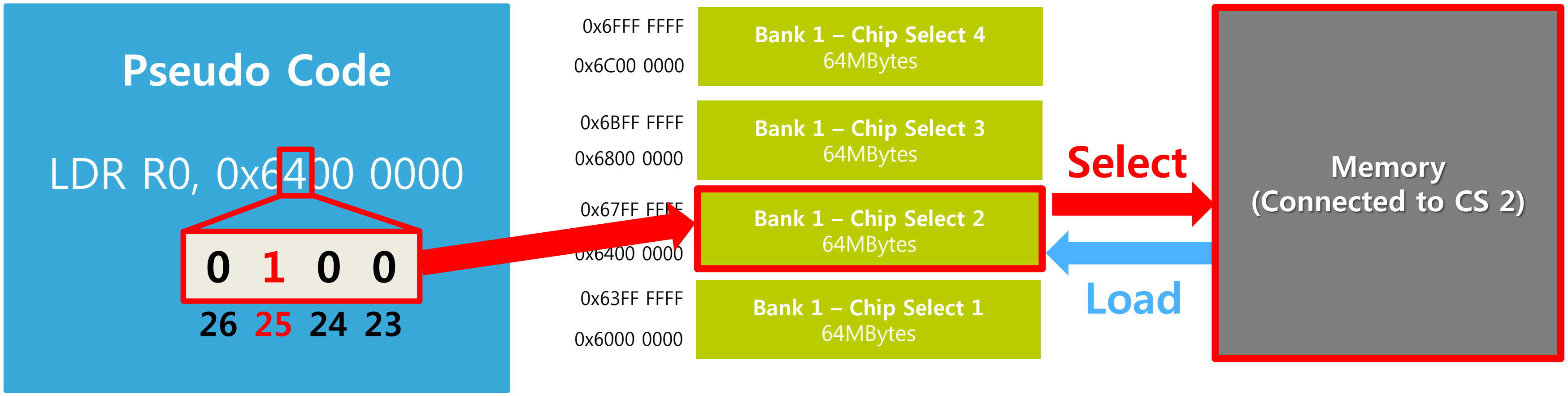
- Bank 1 is divided into 4 banks of 64 Mbytes each to interface with 4 external NOR / PSRAM memories (4 Chip Selects) which support
  - NOR Flash: 8/16/32-bit synchronous/asynchronous, multiplexed or non-multiplexed
  - SRAM: 8/16/32-bit
  - PSRAM: 8/16/32-bit synchronous/asynchronous



life.augmented

NOR/PSRAM address mapping

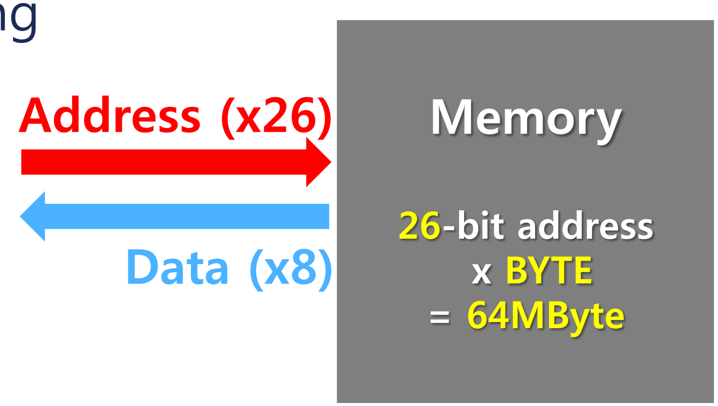
- NOR/PSRAM Controller Memory Mapping
  - **HADDR[27:26] :** Bank select
  - **HADDR[25:0] :**
    - 8-bit bus width : HADDR[25:0]
    - 16-bit bus width : HADDR[25:1] >> 1
    - 32-bit bus width : HADDR[25:2] >> 2

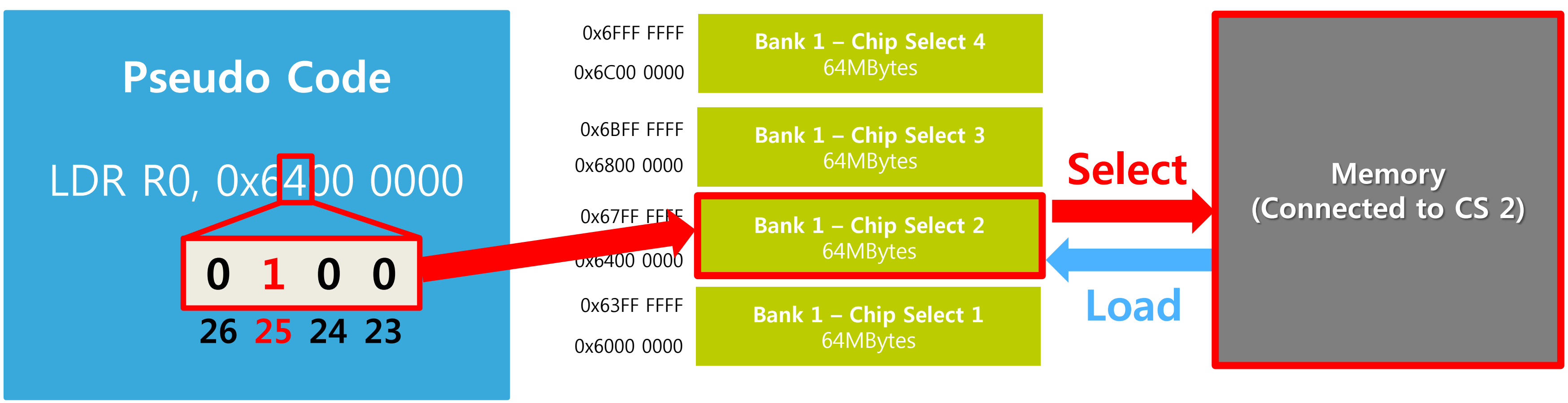


life.augmented

NOR/PSRAM address mapping

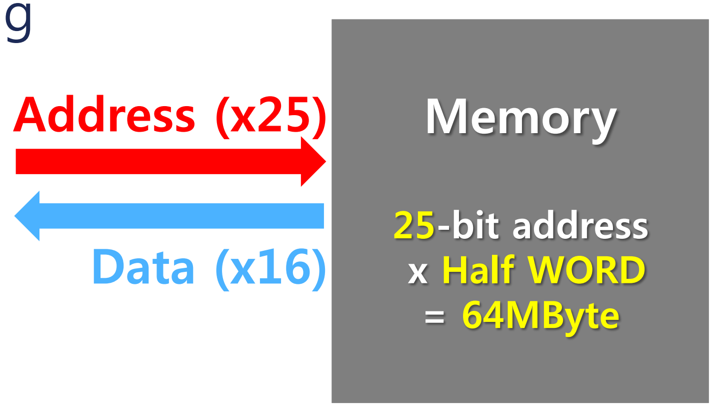
- NOR/PSRAM Controller Memory Mapping
  - **HADDR[27:26] :** Bank select
  - **HADDR[25:0] :**
    - 8-bit bus width : HADDR[25:0]
    - 16-bit bus width : HADDR[25:1] >> 1
    - 32-bit bus width : HADDR[25:2] >> 2

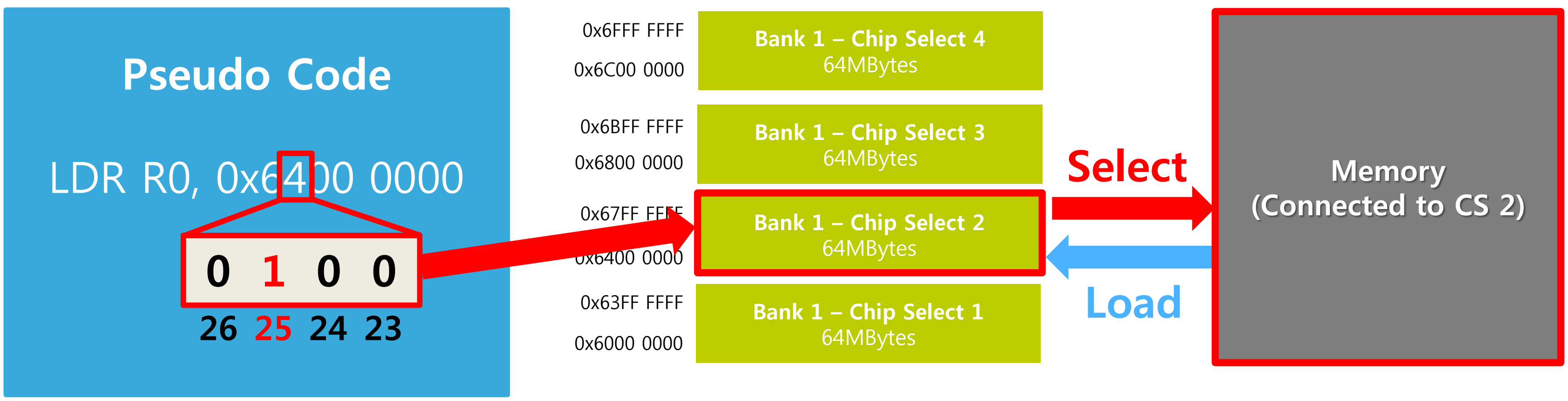




NOR/PSRAM address mapping

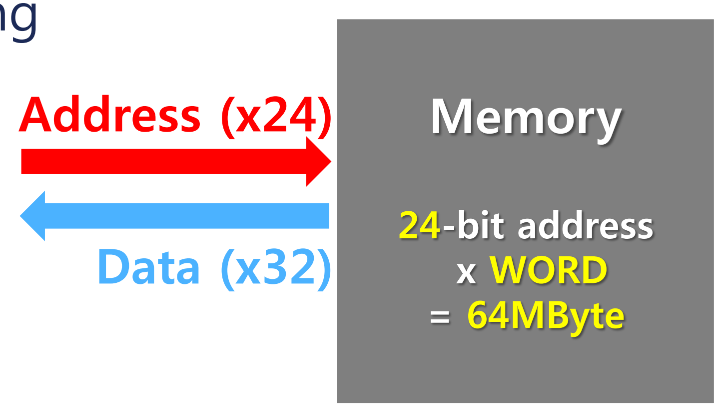
- NOR/PSRAM Controller Memory Mapping
  - **HADDR[27:26] :** Bank select
  - **HADDR[25:0] :**
    - 8-bit bus width : HADDR[25:0]
    - 16-bit bus width : HADDR[25:1] >> 1
    - 32-bit bus width : HADDR[25:2] >> 2





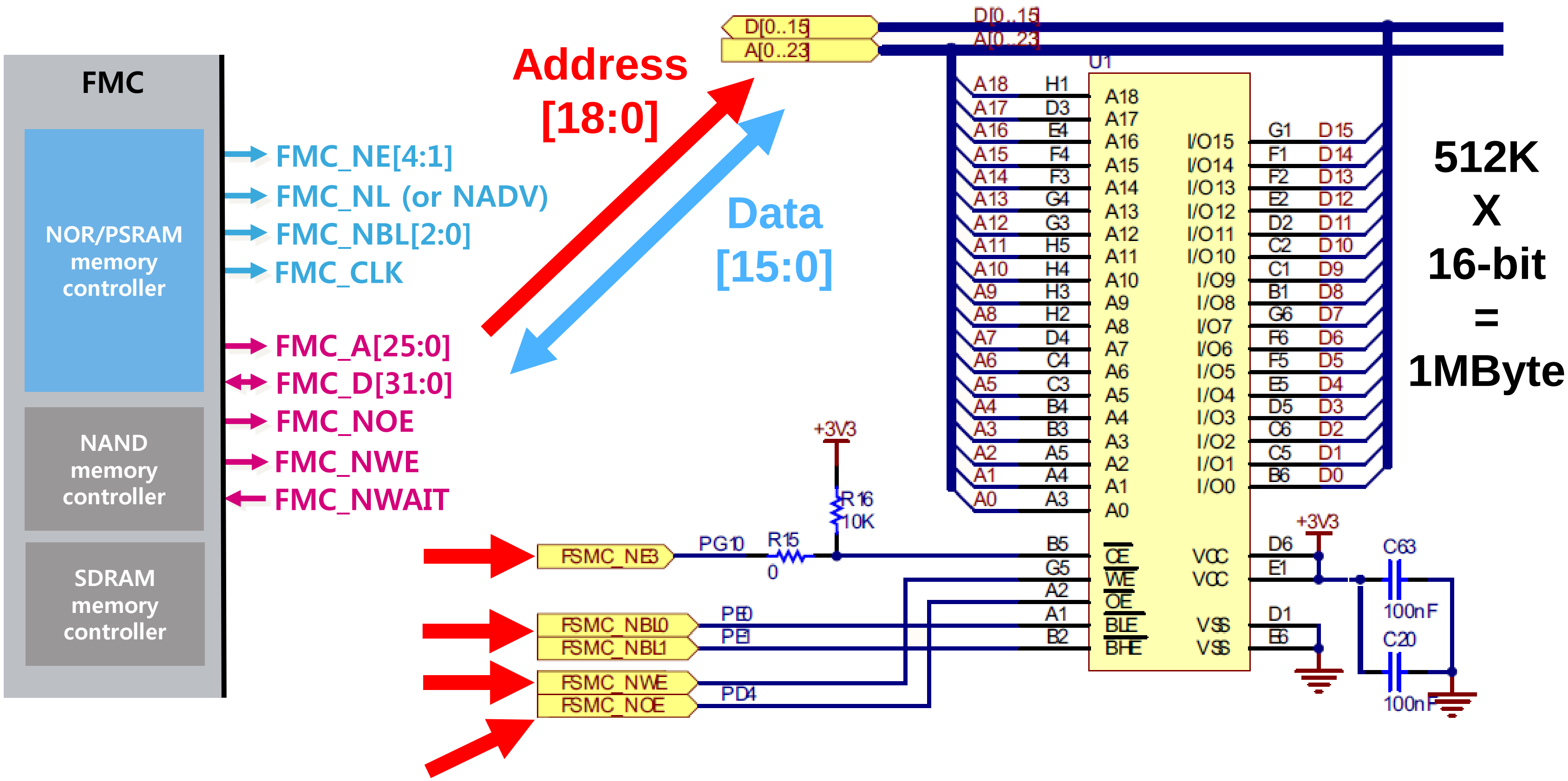
NOR/PSRAM address mapping

- NOR/PSRAM Controller Memory Mapping
  - **HADDR[27:26] :** Bank select
  - **HADDR[25:0] :**
    - 8-bit bus width : HADDR[25:0]
    - 16-bit bus width : HADDR[25:1] >> 1
    - 32-bit bus width : HADDR[25:2] >> 2

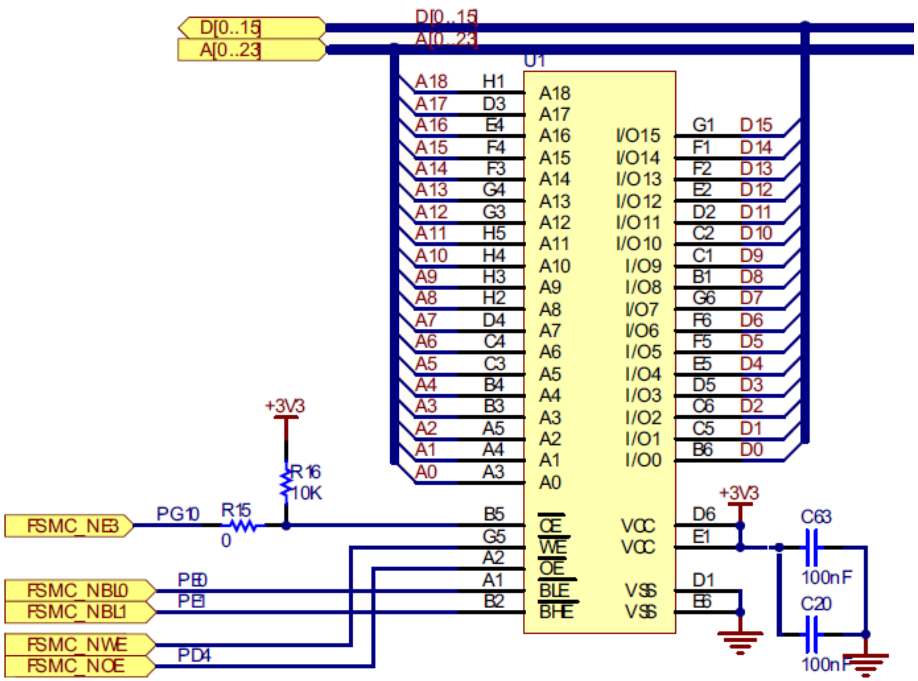


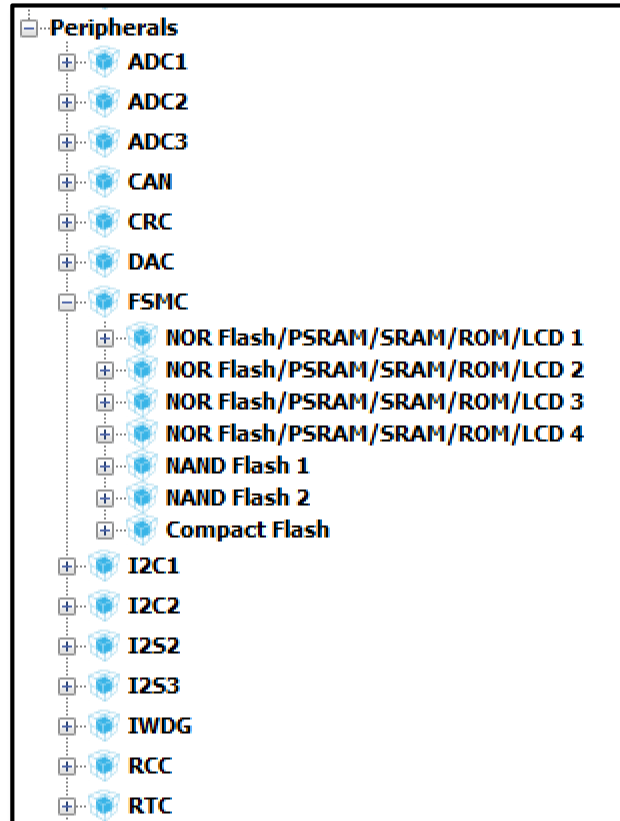
NOR/PSRAM interface signals with SRAM

- The FMC generates the appropriate signals to drive

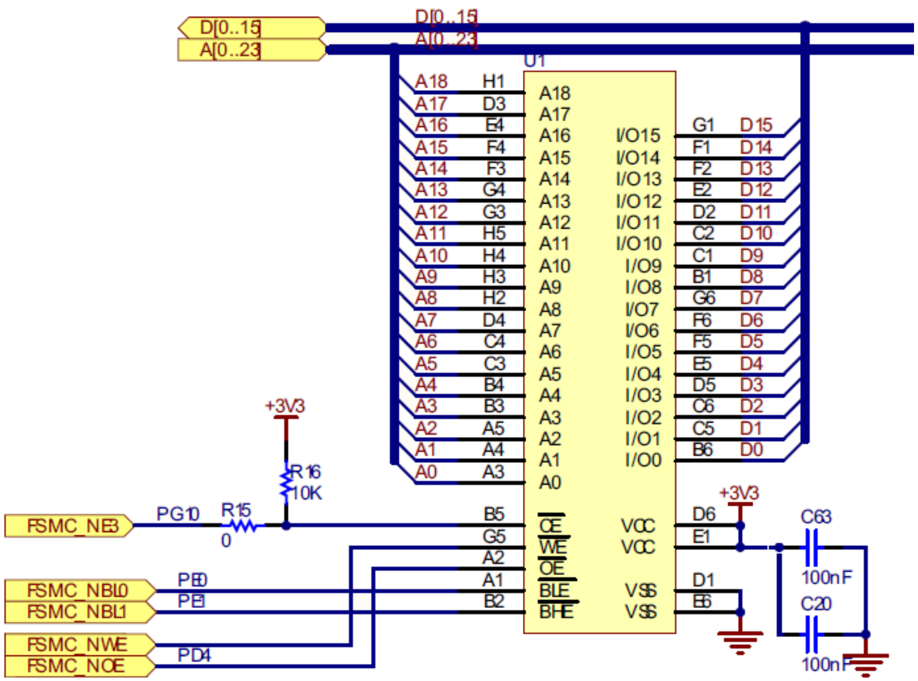


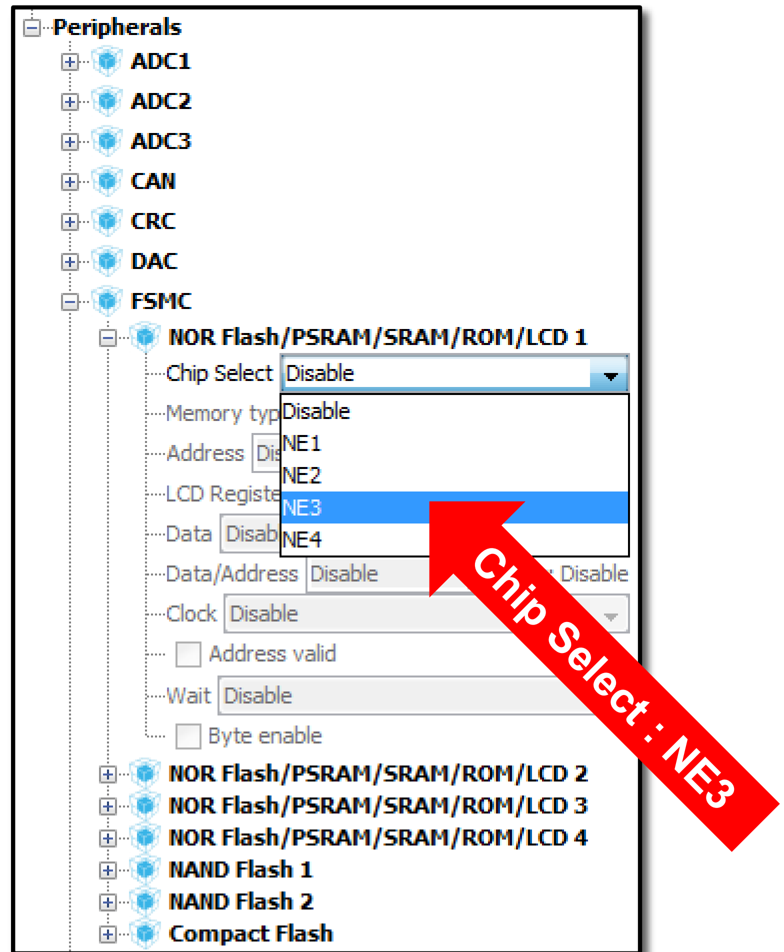
CubeMX Configuration for SRAM Connection



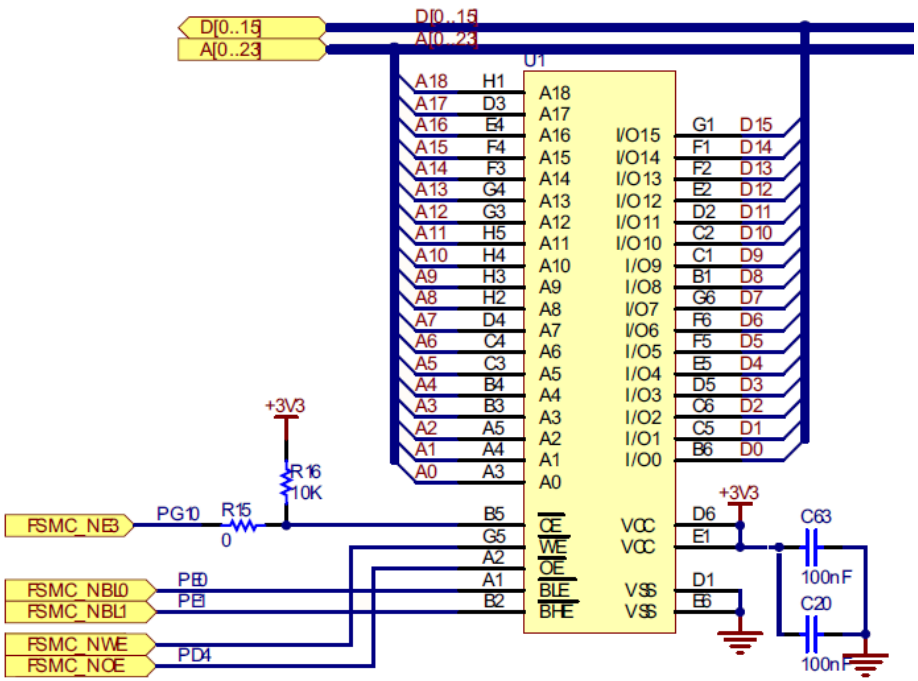


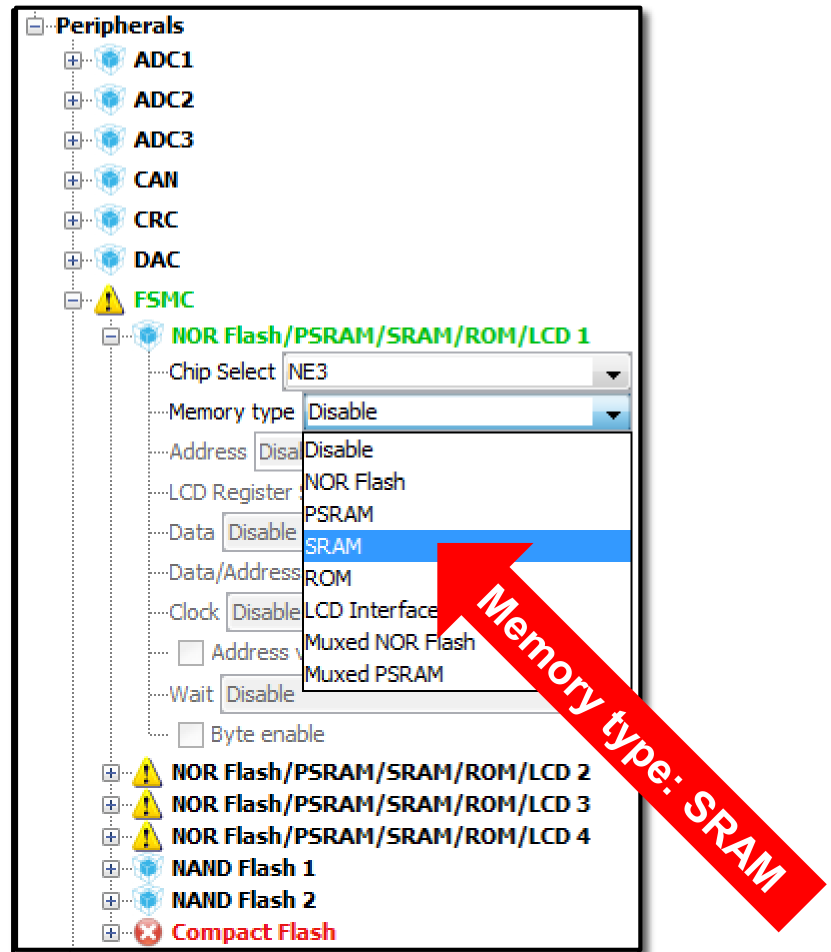
CubeMX Configuration for SRAM Connection



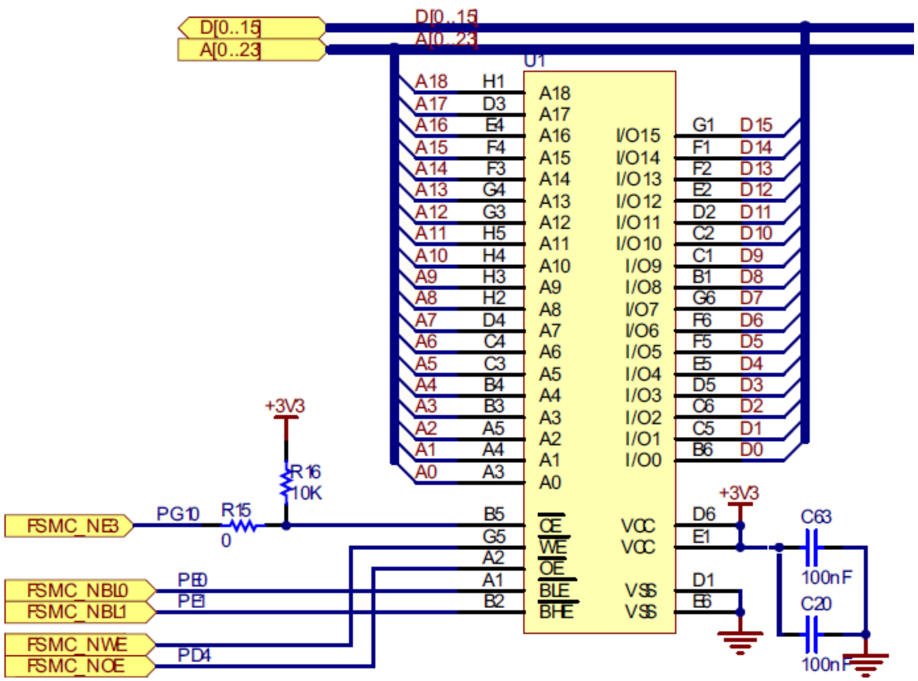


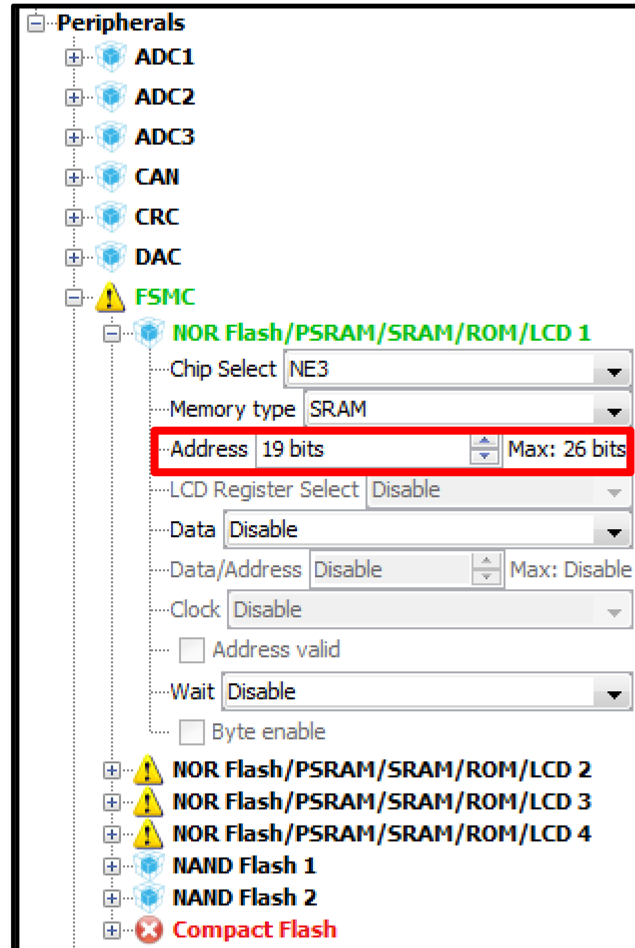
CubeMX Configuration for SRAM Connection



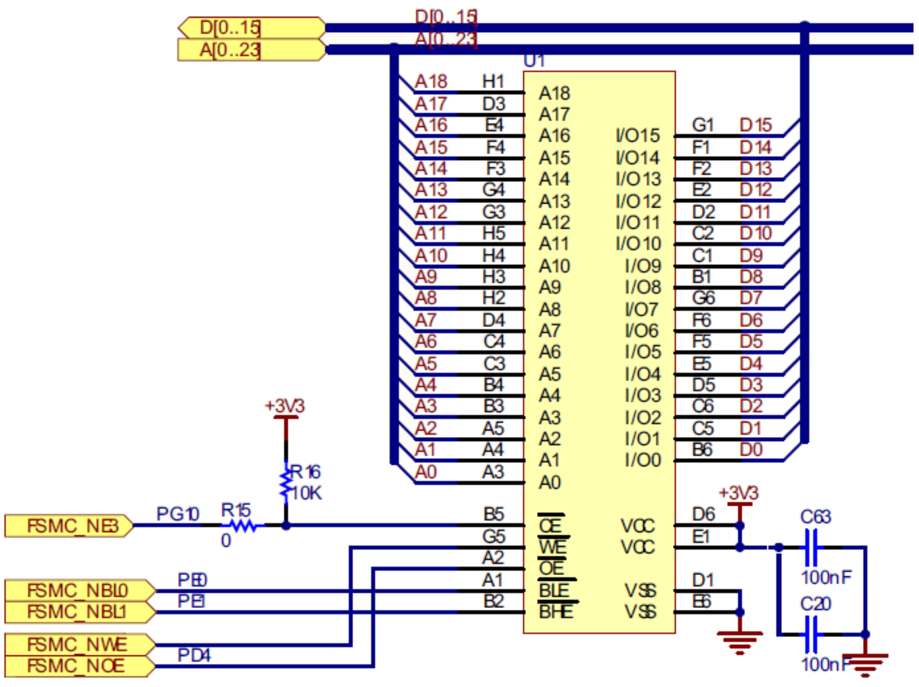


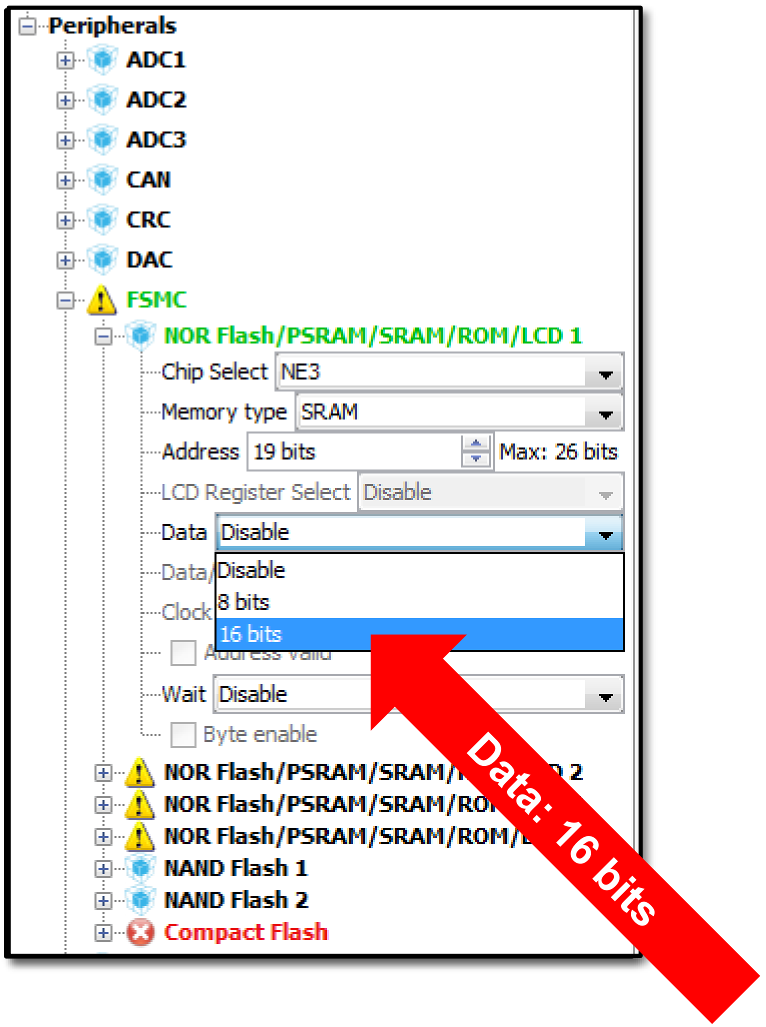
CubeMX Configuration for SRAM Connection



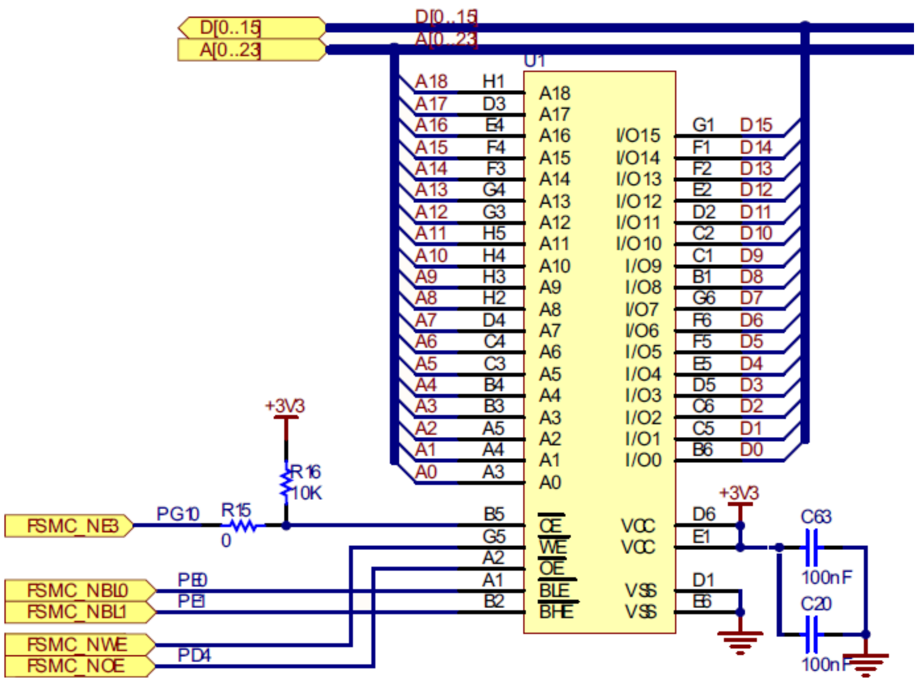


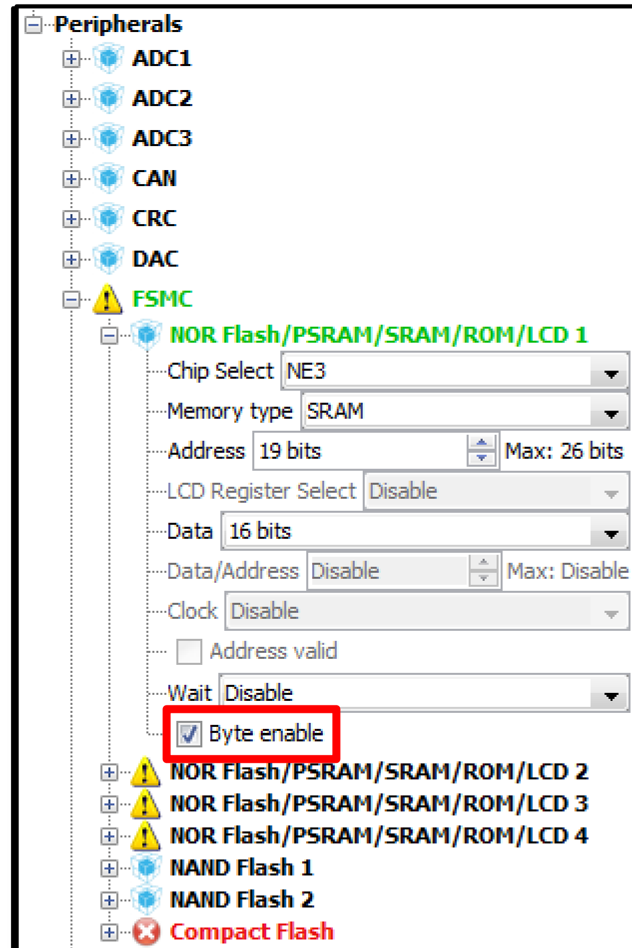
CubeMX Configuration for SRAM Connection





CubeMX Configuration for SRAM Connection

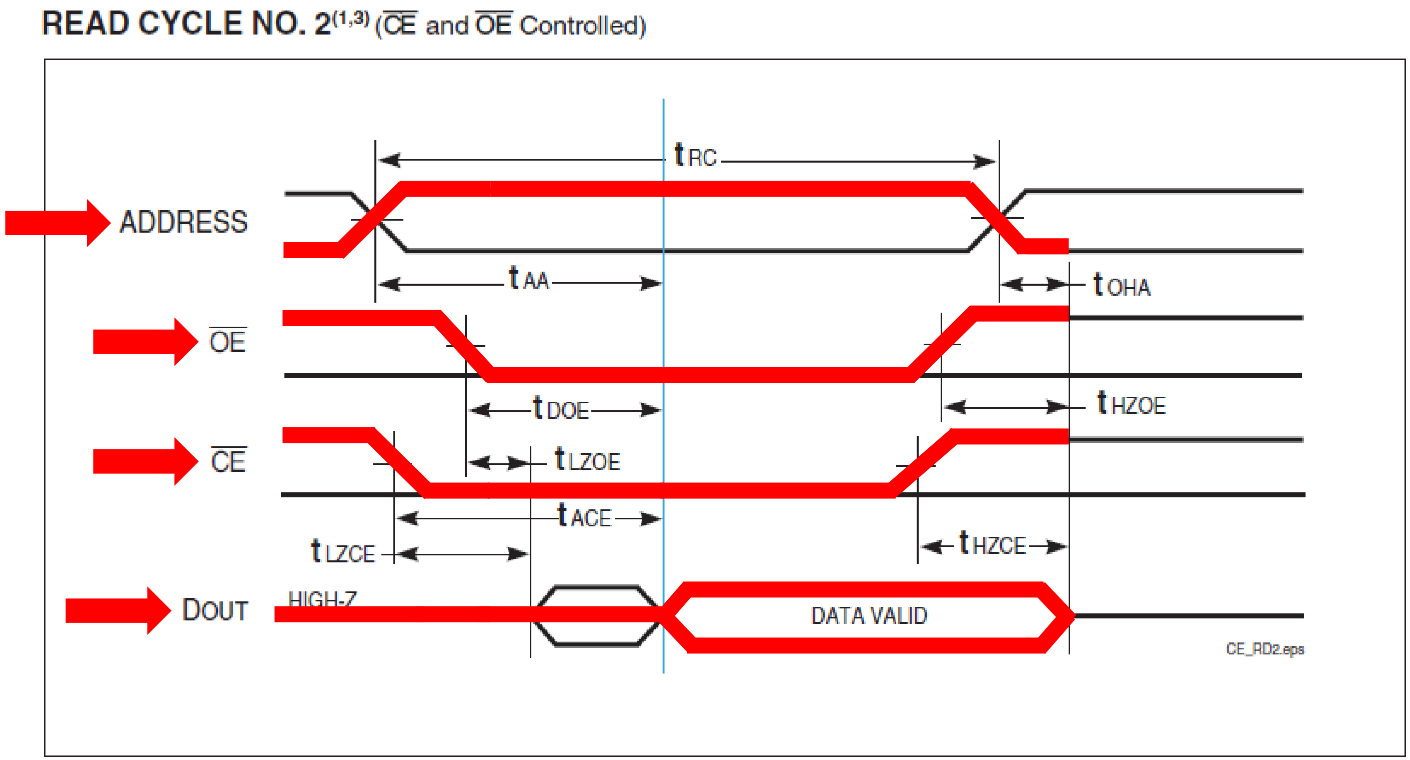




life.augmented

NOR/PSRAM Interface Protocol and Timing

- SRAM Interface Timing



NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/PSRAM	**N (default)**	**Don’t care**	**Mode 1**
	Y	0	Mode A
NOR	**N (default)**	**Don’t‘ care**	**Mode 2**
	Y	0	Mode B
	Y	1	Mode C
	Y	2	Mode D

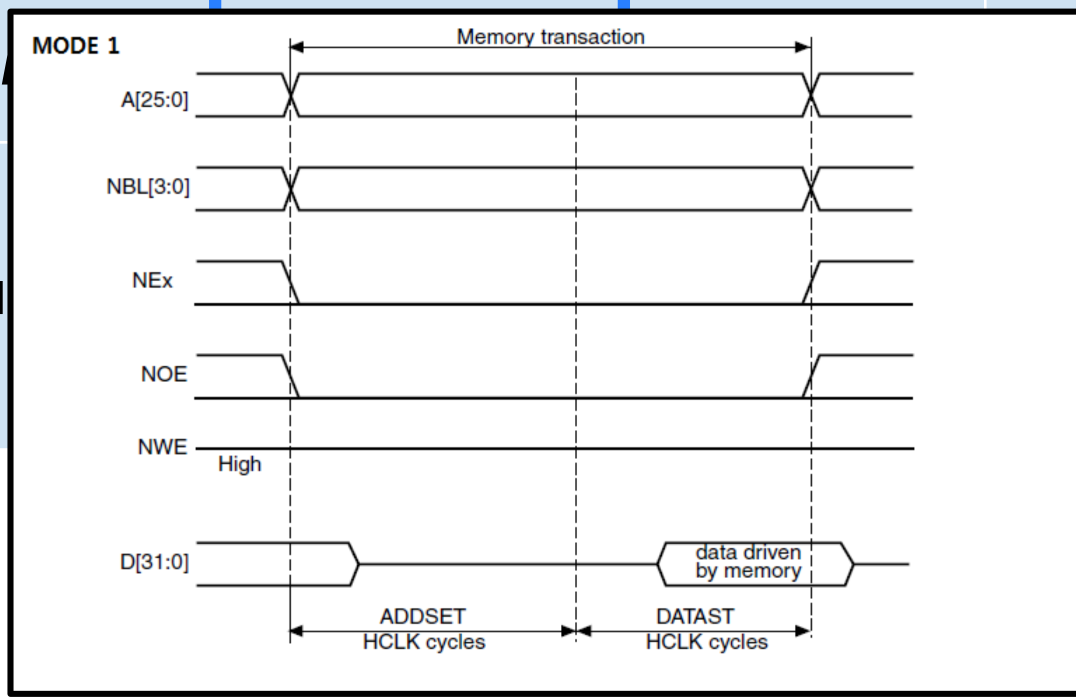
Use the common Read/Write timing

Use the separated Read/Write timing

NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

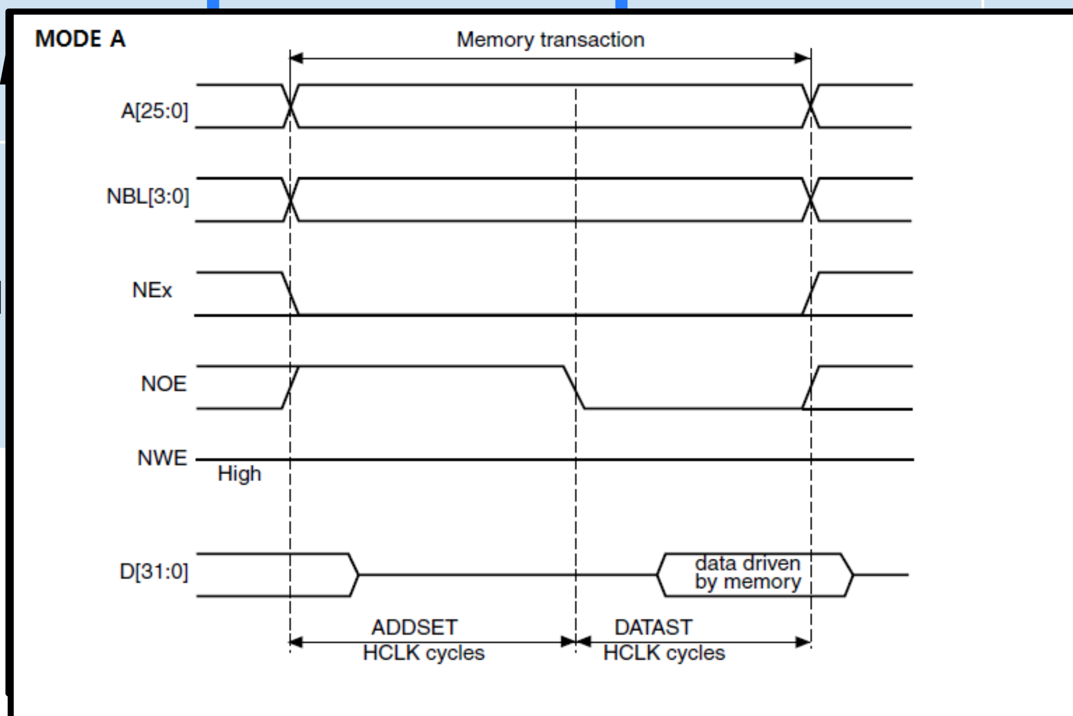
Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D



NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

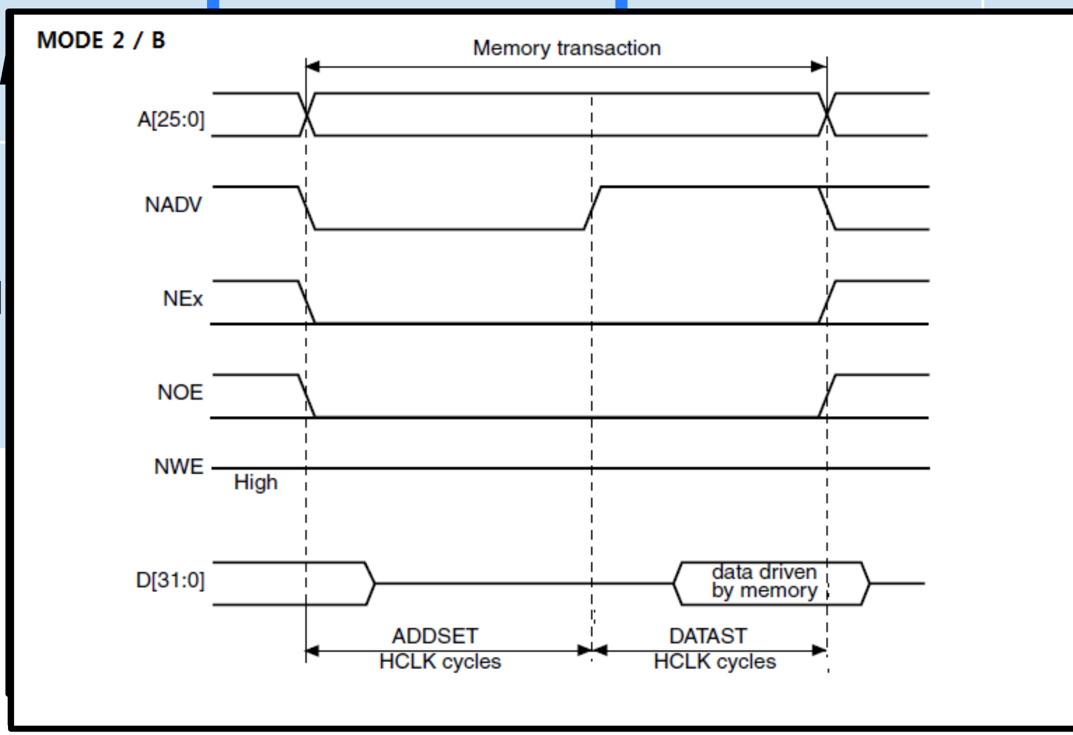
Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D



NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D

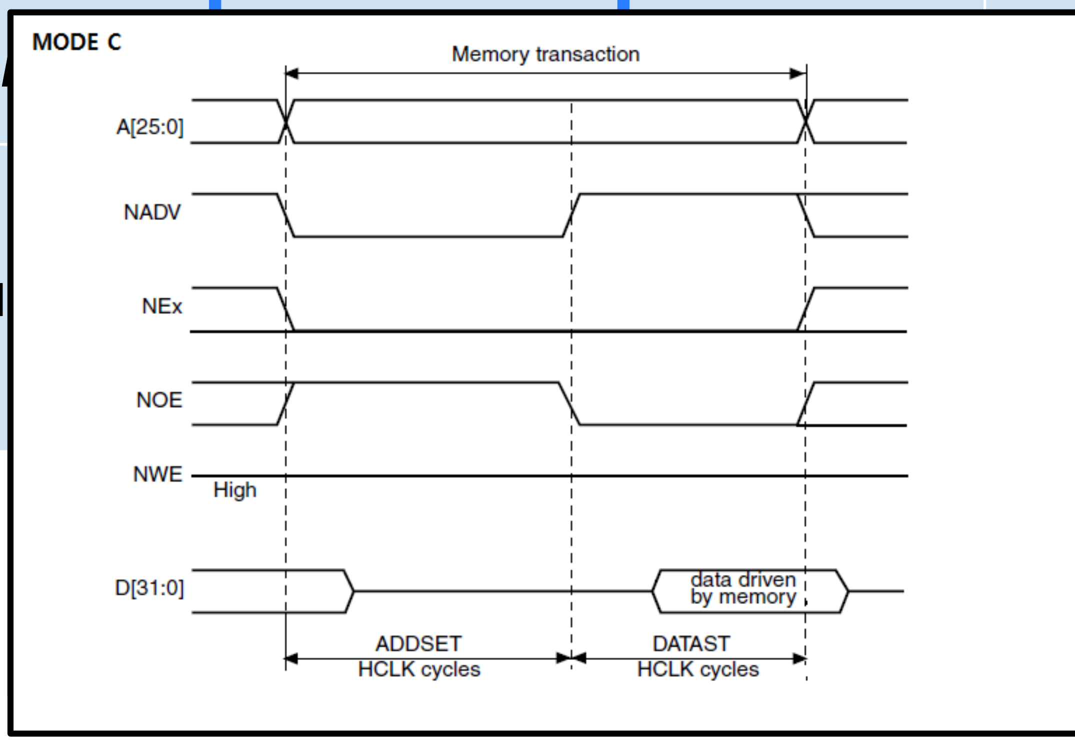


life.augmented

NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

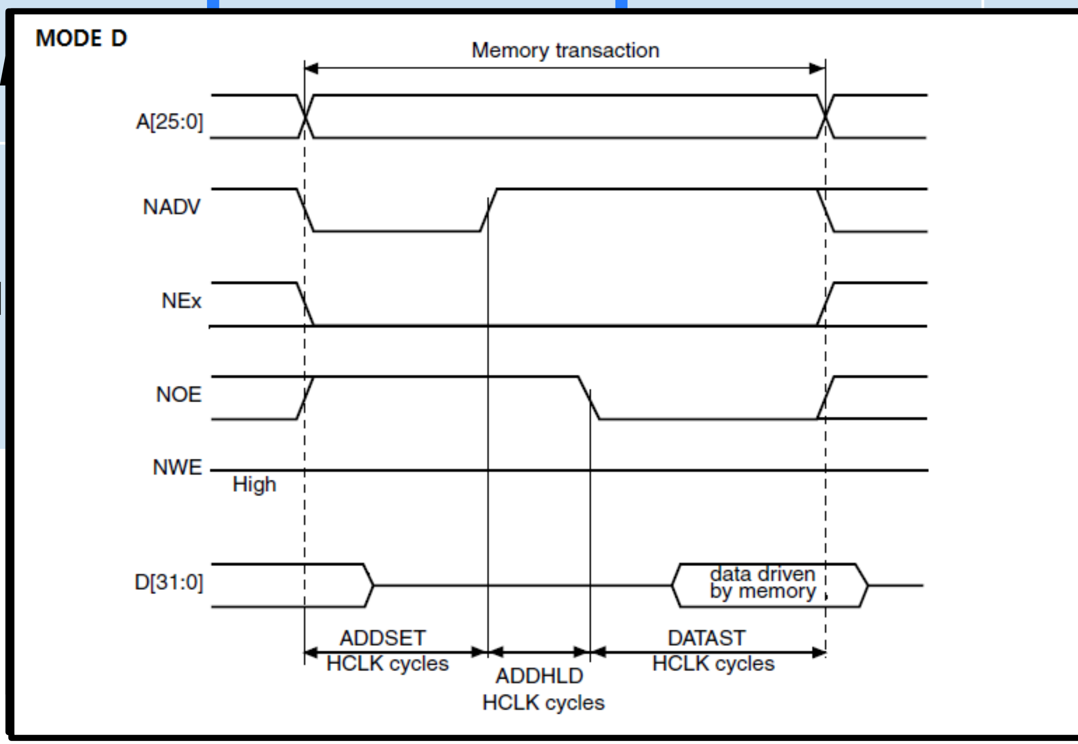
Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			**Mode 1**
			Mode A
N			**Mode 2**
			Mode B
			Mode C
			Mode D



NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D

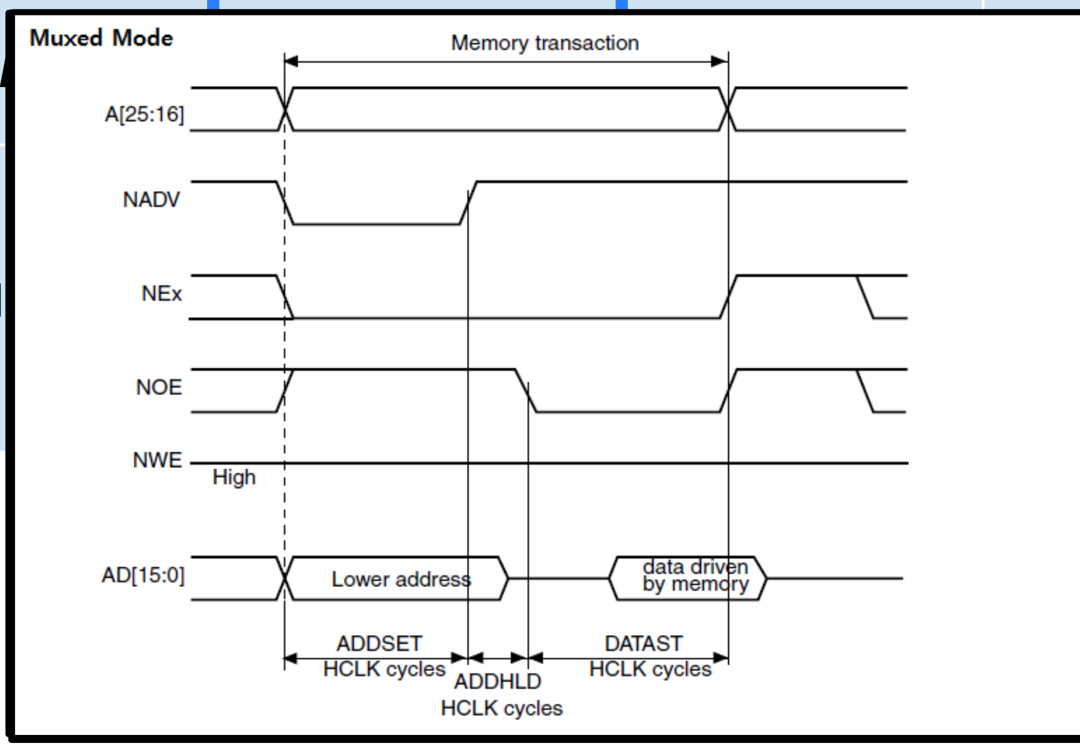


life.augmented

NOR/PSRAM Interface Protocol and Timing

- FMC NOR/PSRAM Access Mode

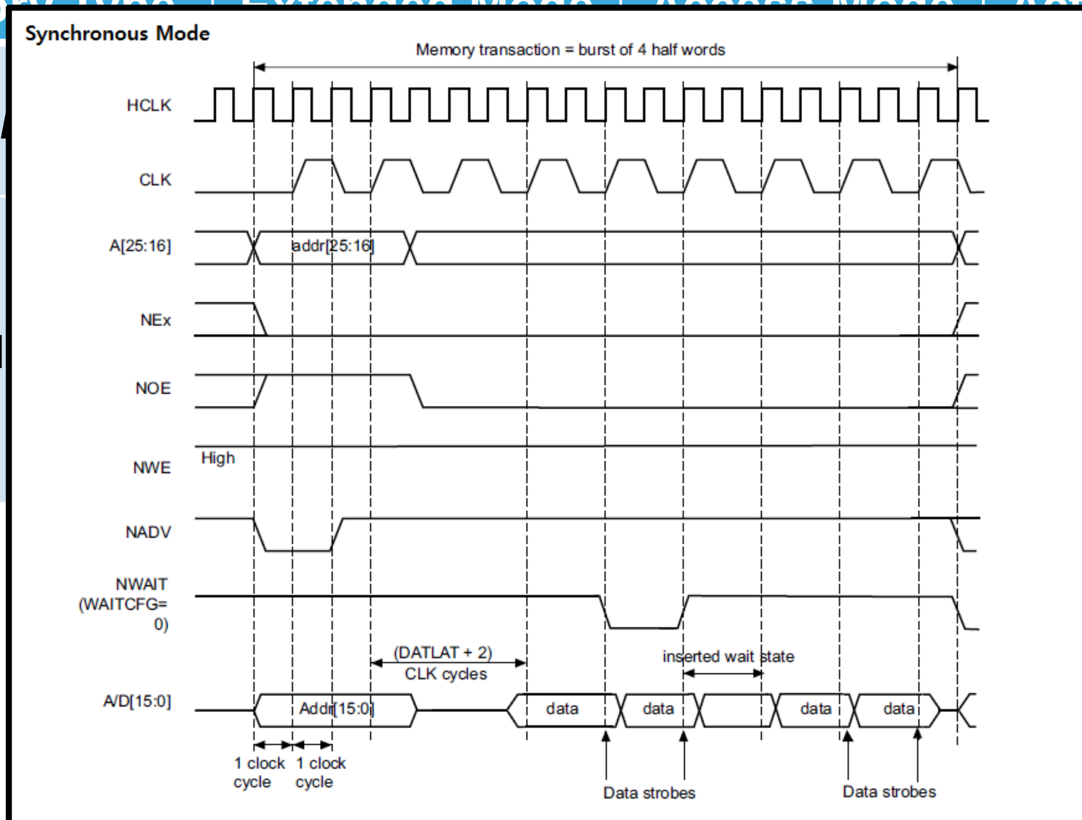
Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D



NOR/PSRAM Interface Protocol and Timing

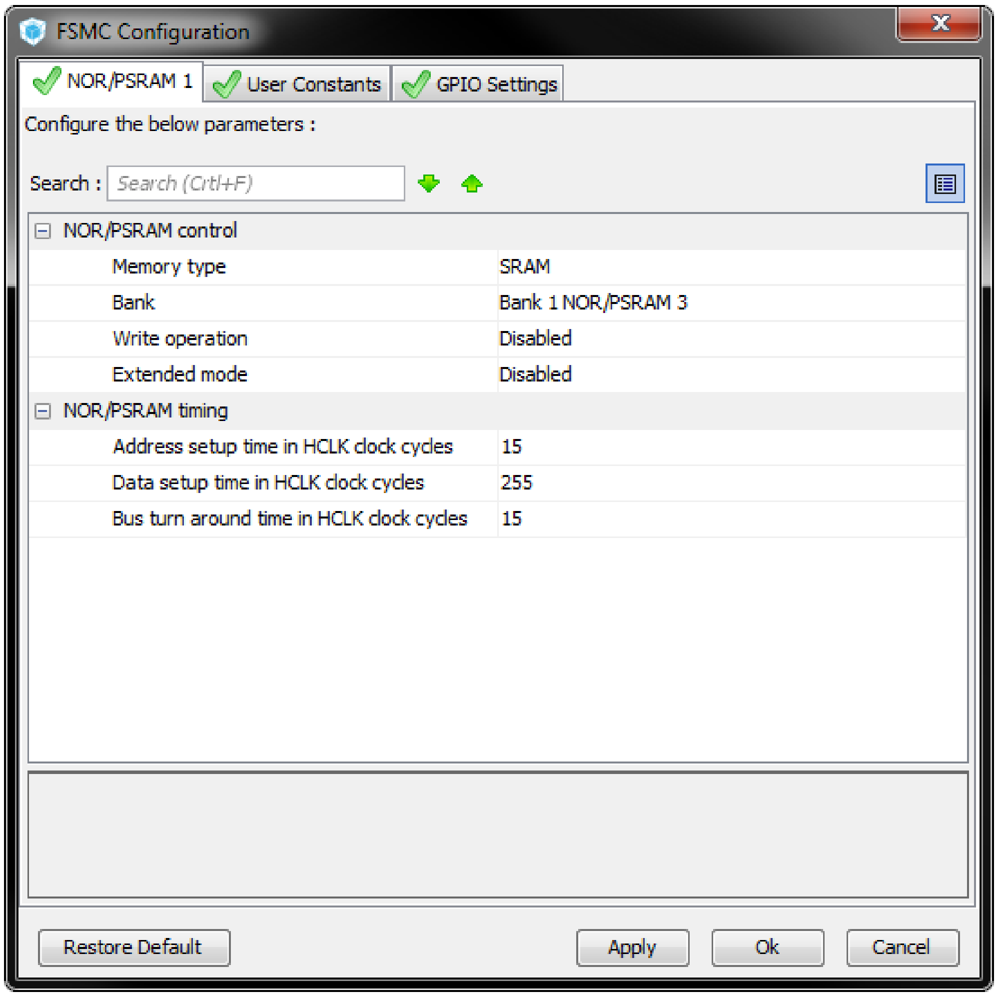
- FMC NOR/PSRAM Access Mode

Memory Type	Extended Mode	Access Mode	Actual Mode
SRAM/			Mode 1
			Mode A
N			Mode 2
			Mode B
			Mode C
			Mode D

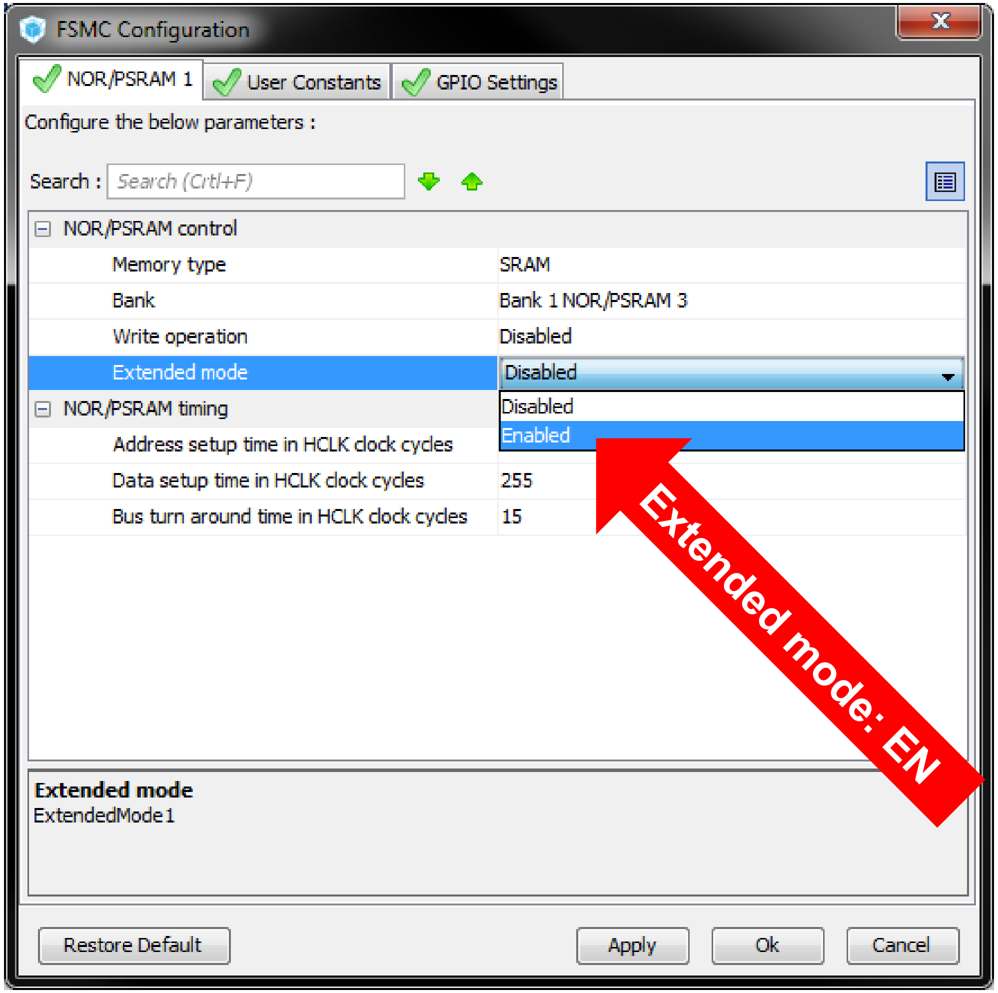


life.augmented

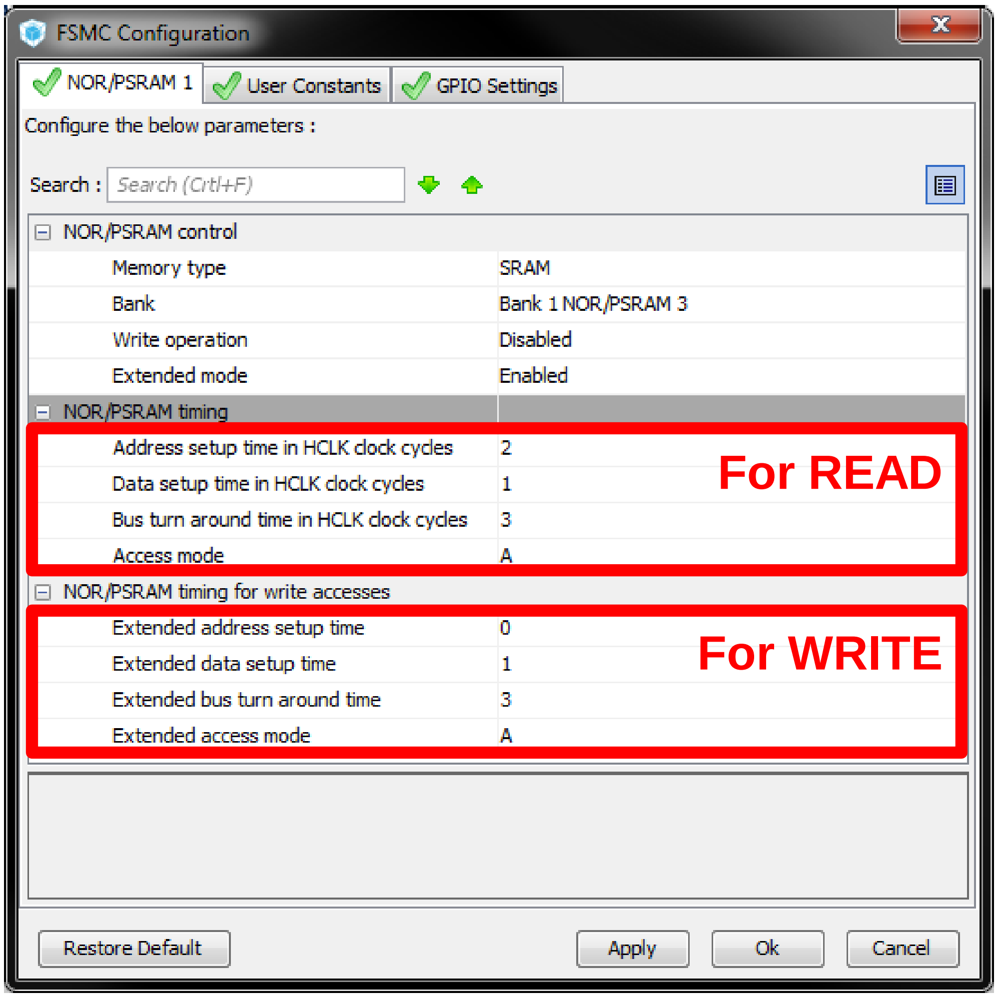
CubeMX FSMC Configuration for SRAM



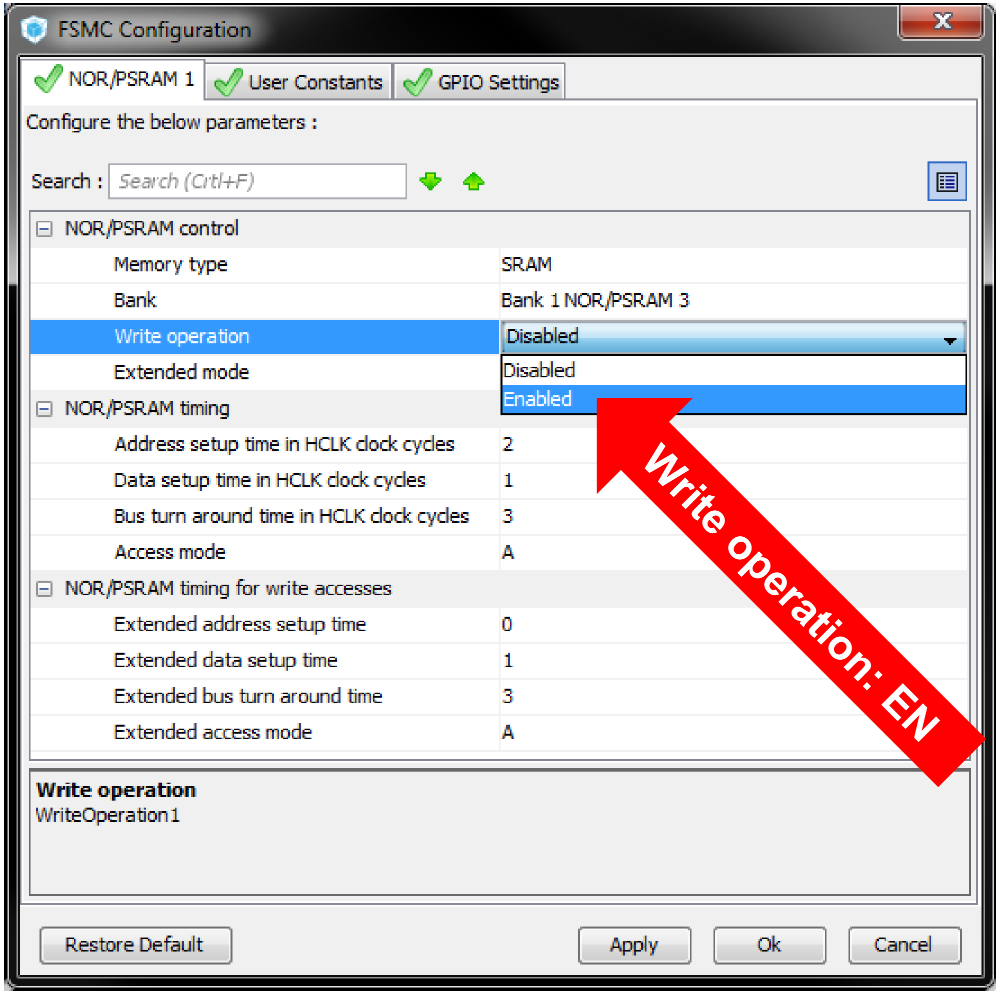
CubeMX FSMC Configuration for SRAM



CubeMX FSMC Configuration for SRAM

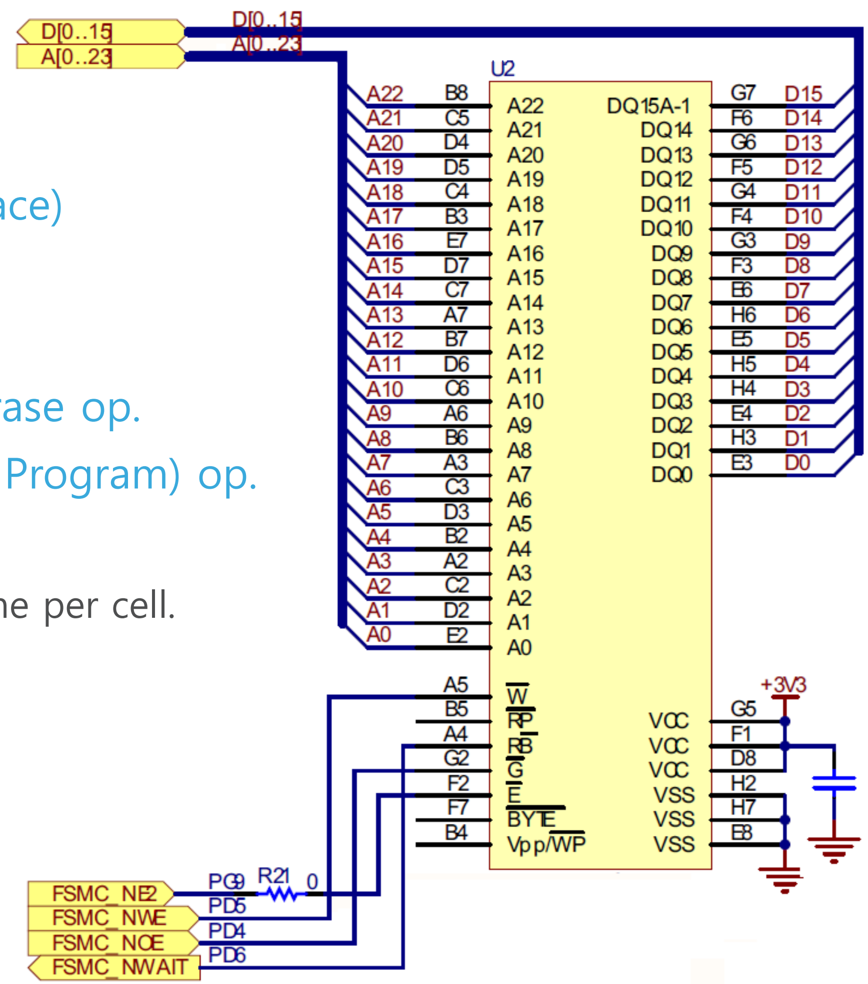


CubeMX FSMC Configuration for SRAM

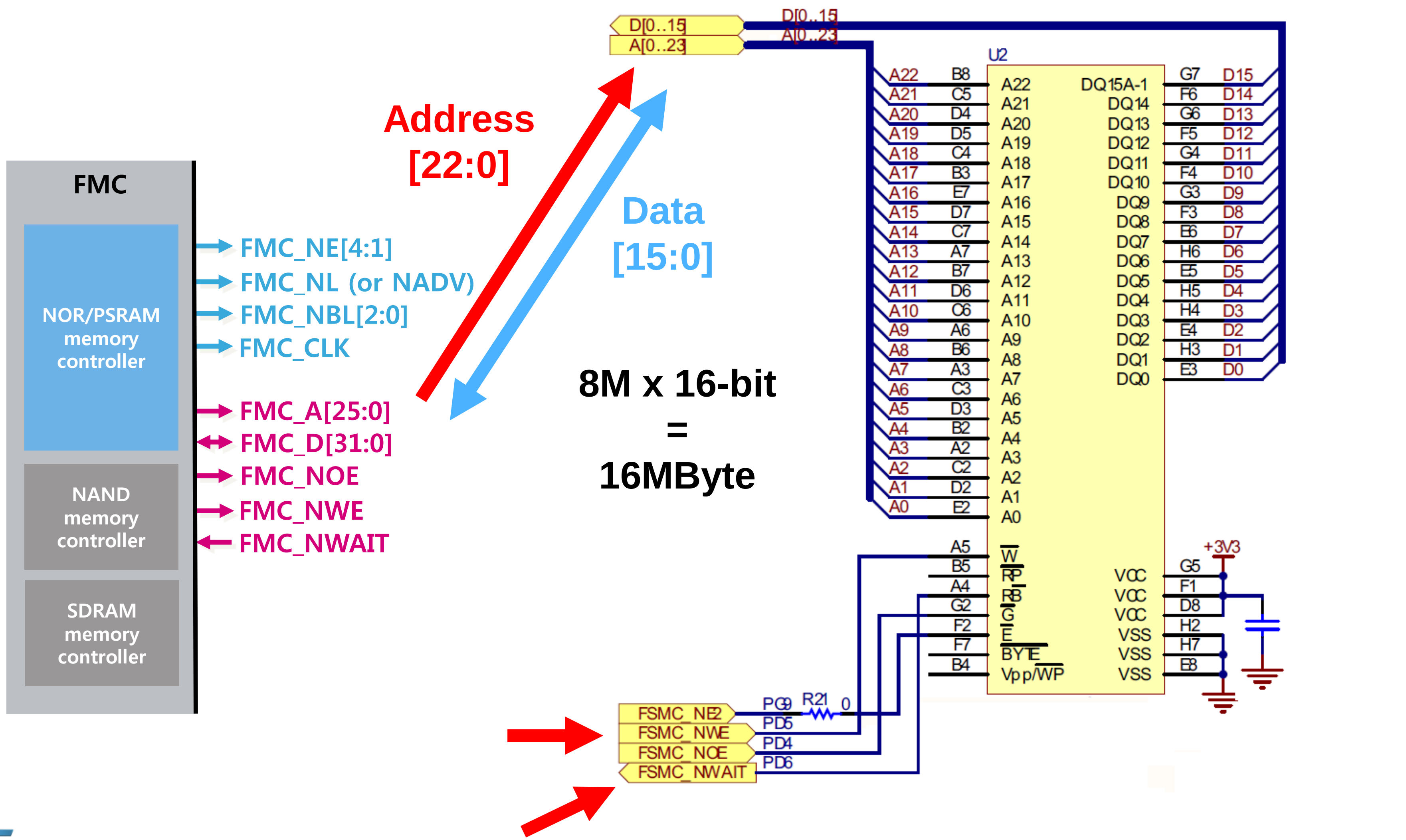


About NOR Flash Memory

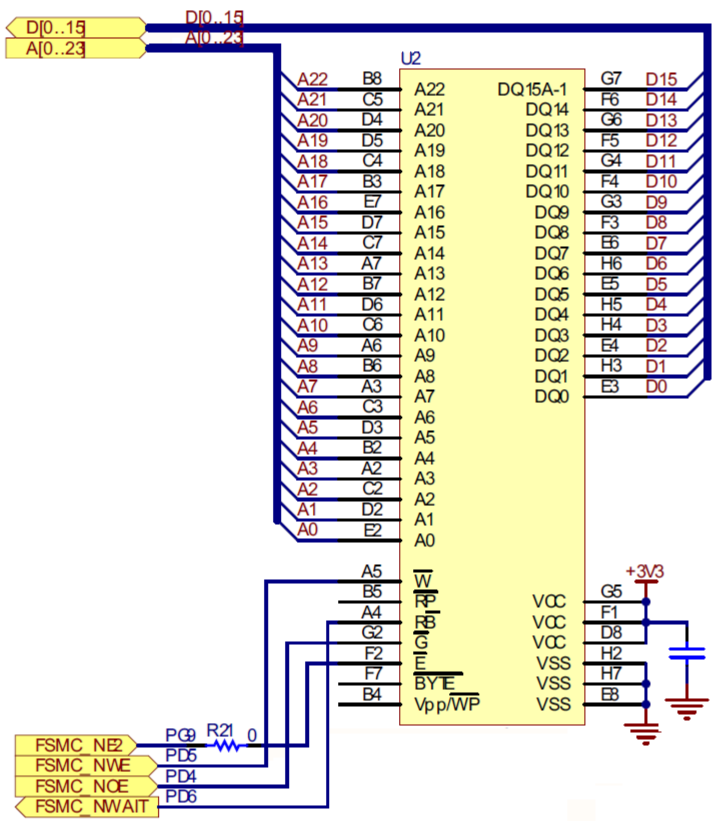
- NOR Flash memory
  - **Advantage**
    - Random read (XIP: Execute In Place)
    - Data stability
  - **Disadvantage**
    - Low performance on program/erase op.
    - Large unit for atomic erase (and Program) op.
    - Limited endurance
      - Generally 1~10K program/erase time per cell.

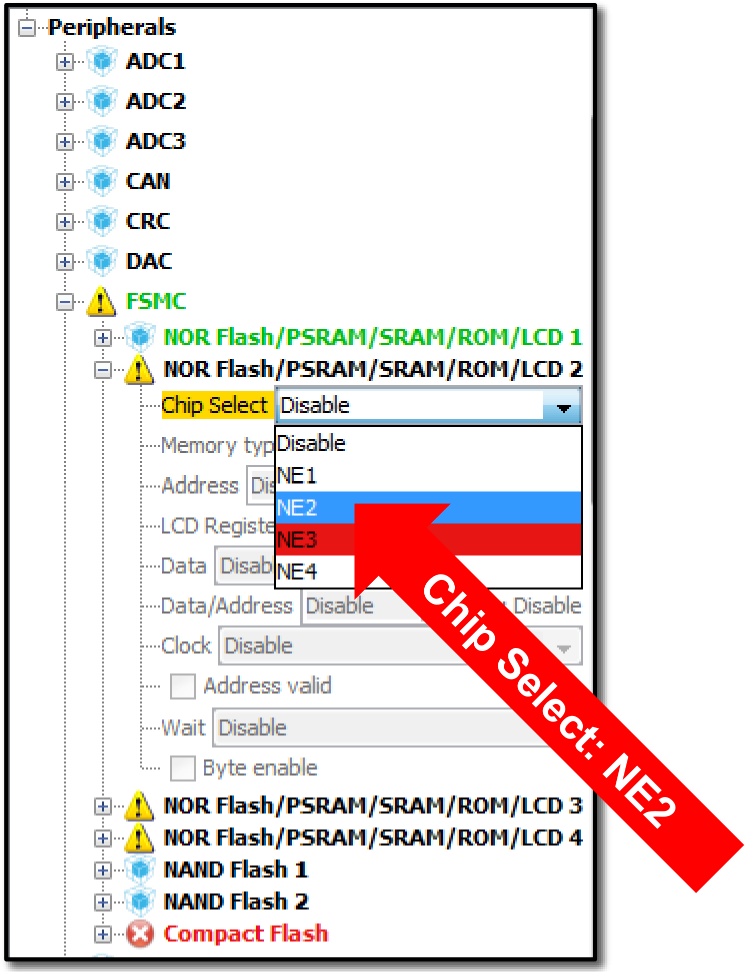


NOR/PSRAM interface signals with NOR

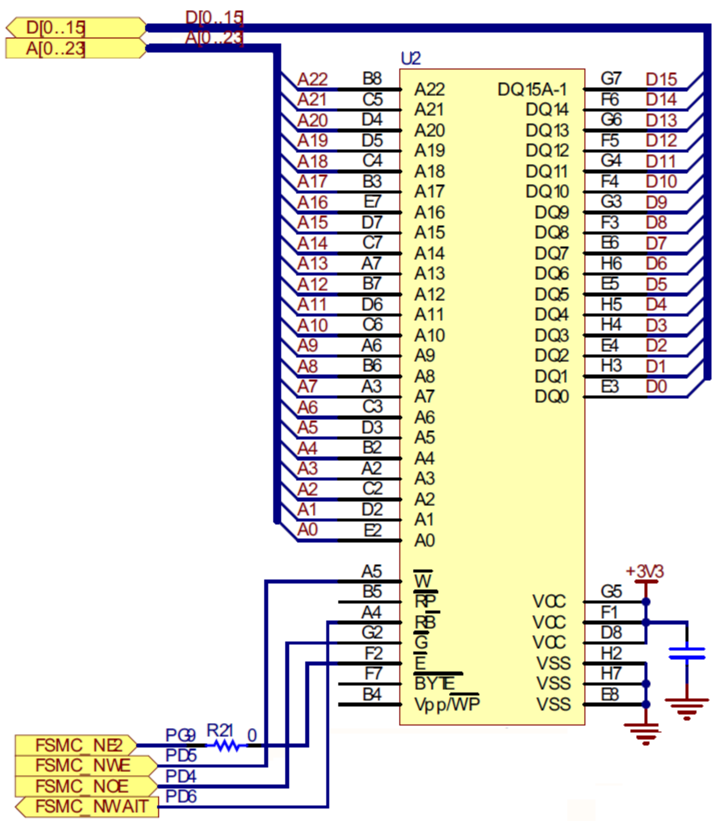


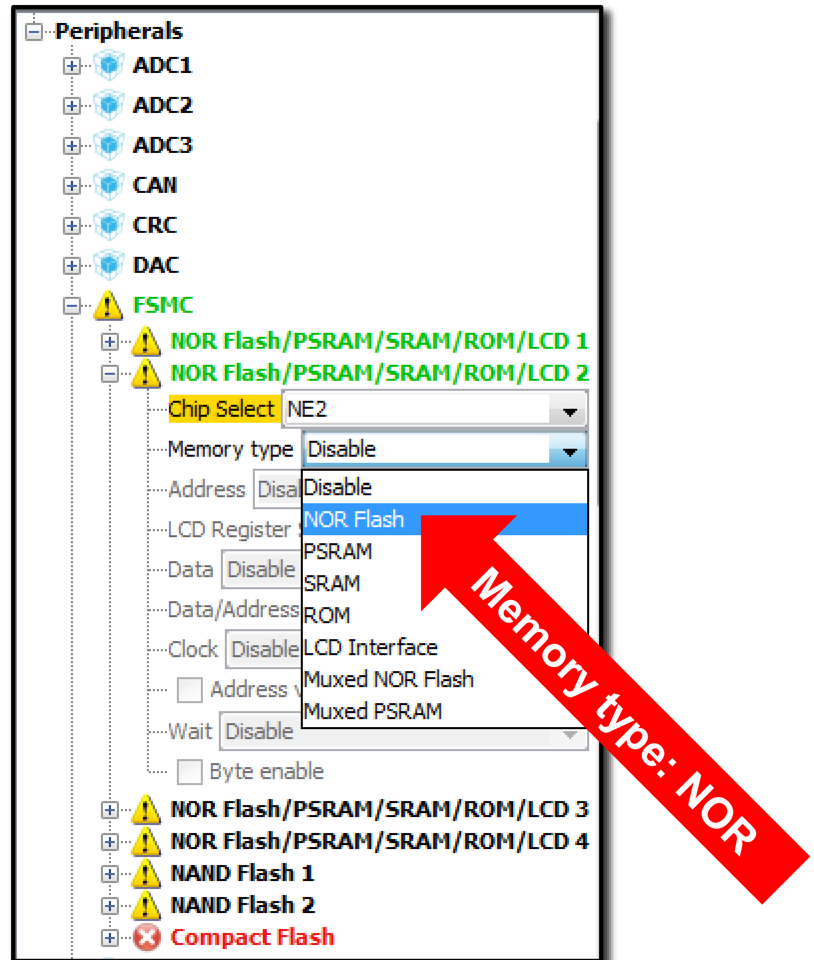
CubeMX Configuration for NOR Connection



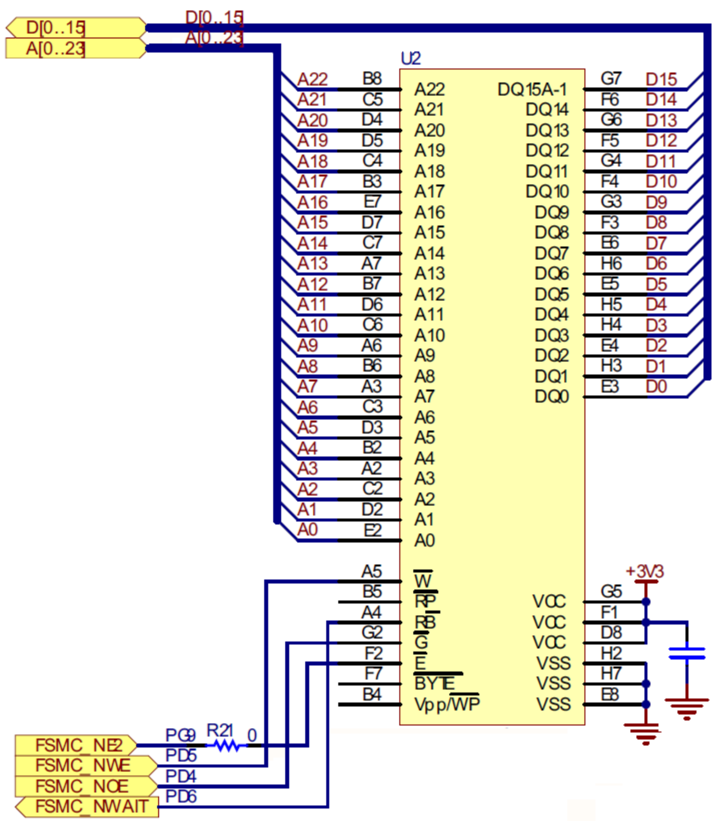


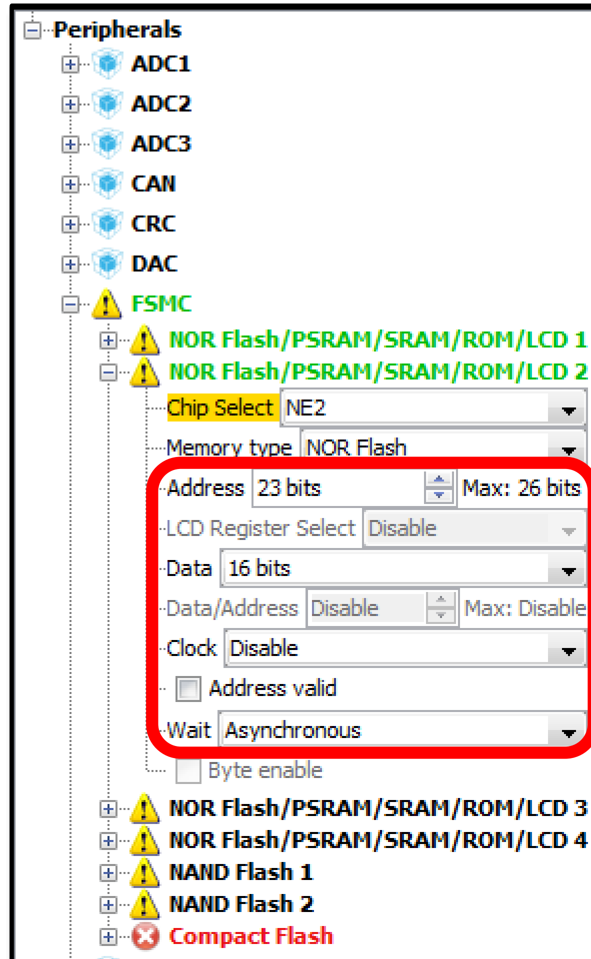
CubeMX Configuration for NOR Connection





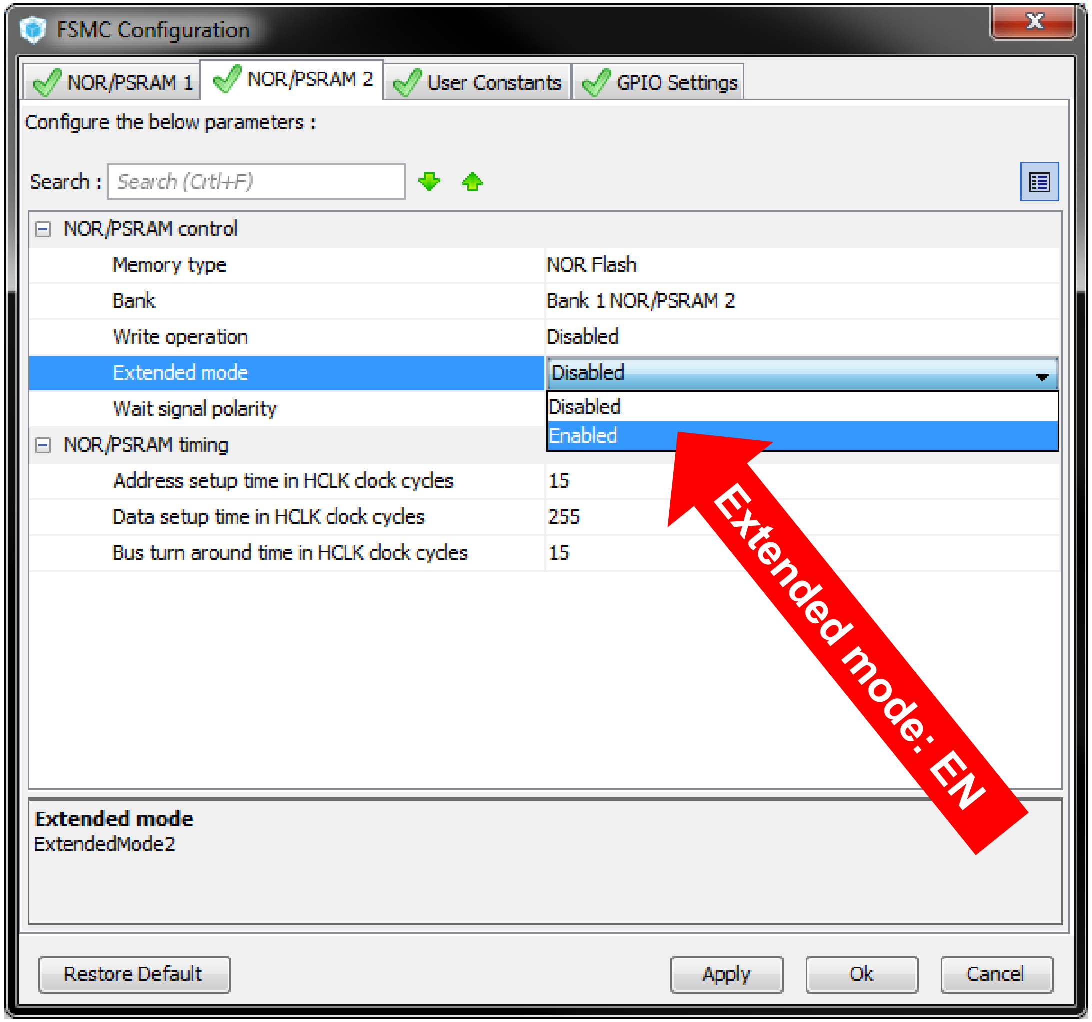
CubeMX Configuration for NOR Connection



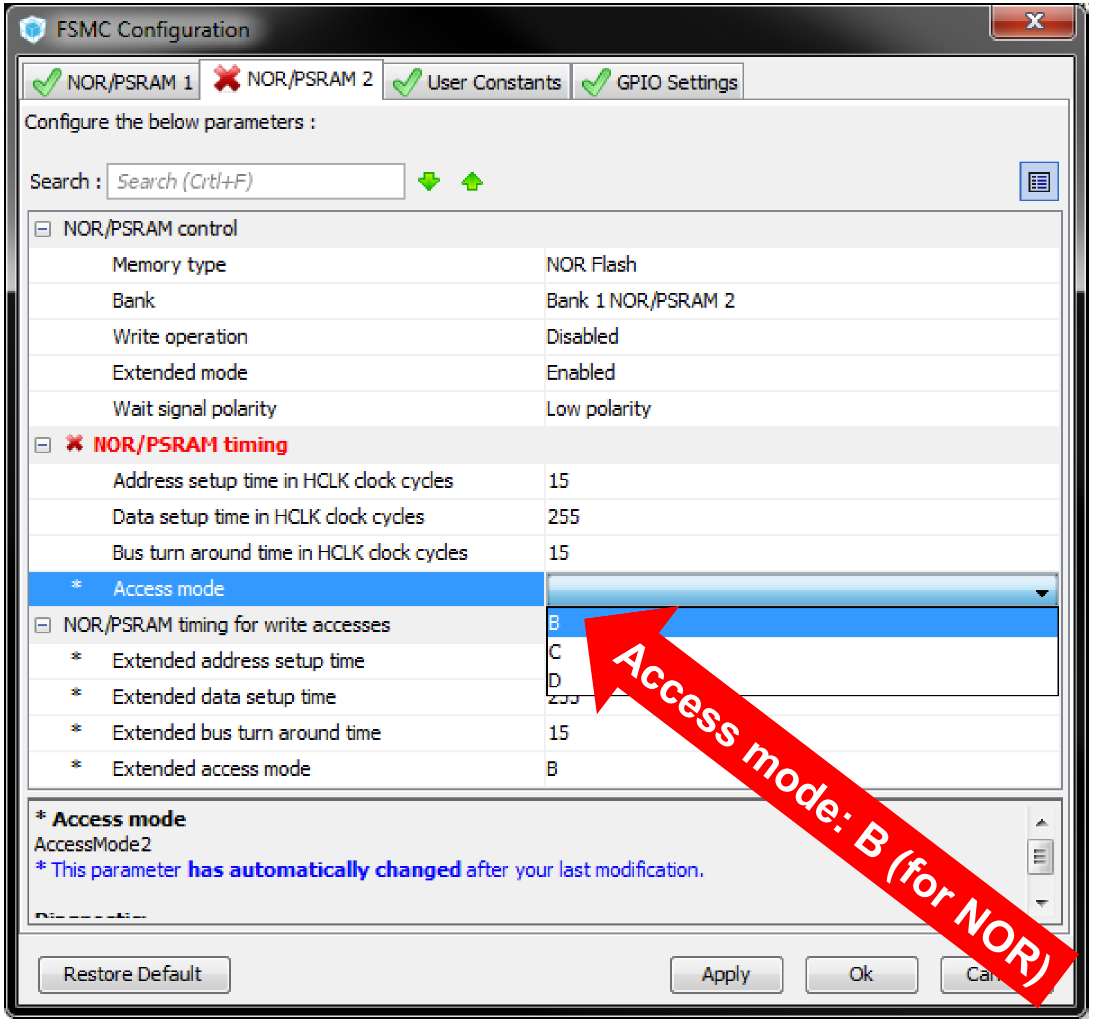


life.augmented

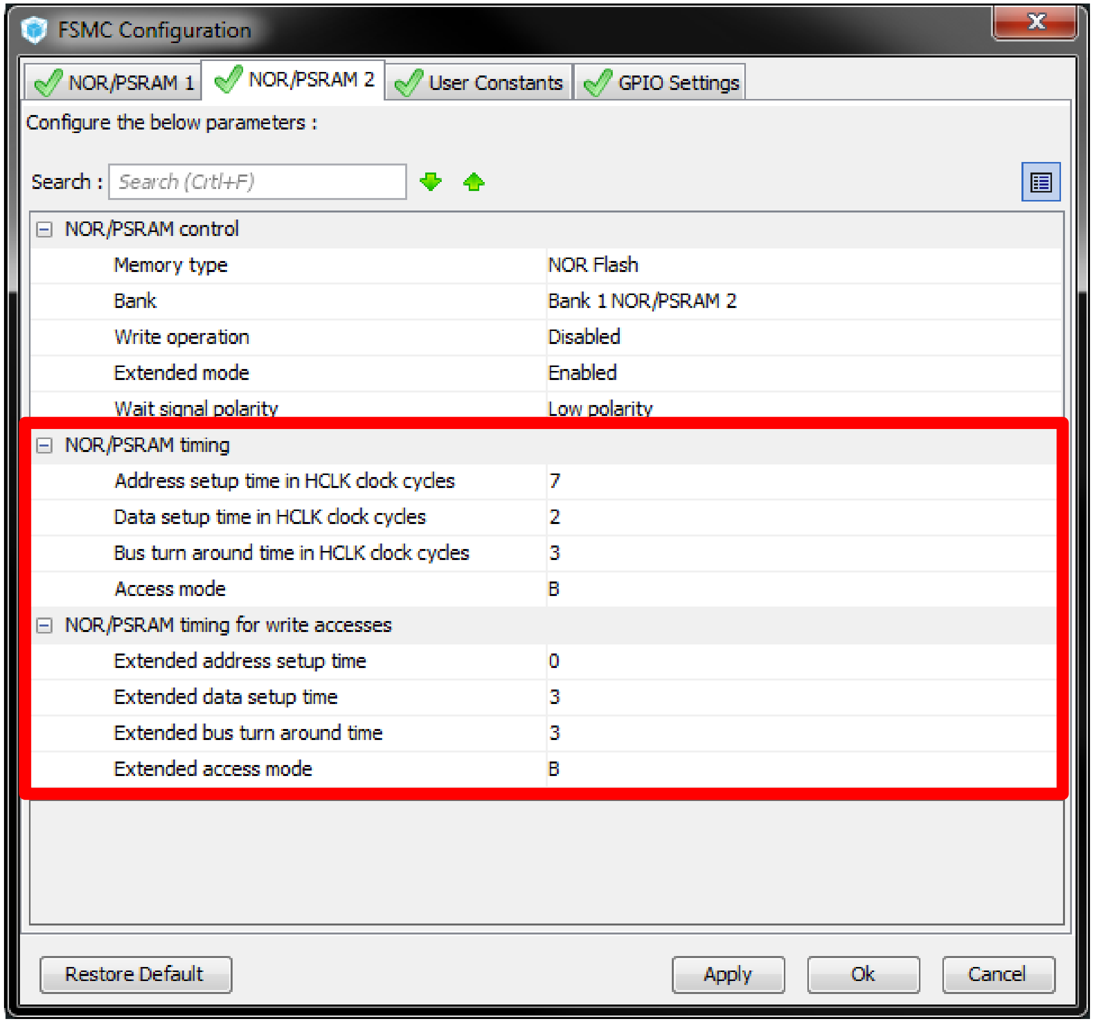
CubeMX FSMC Configuration for NOR



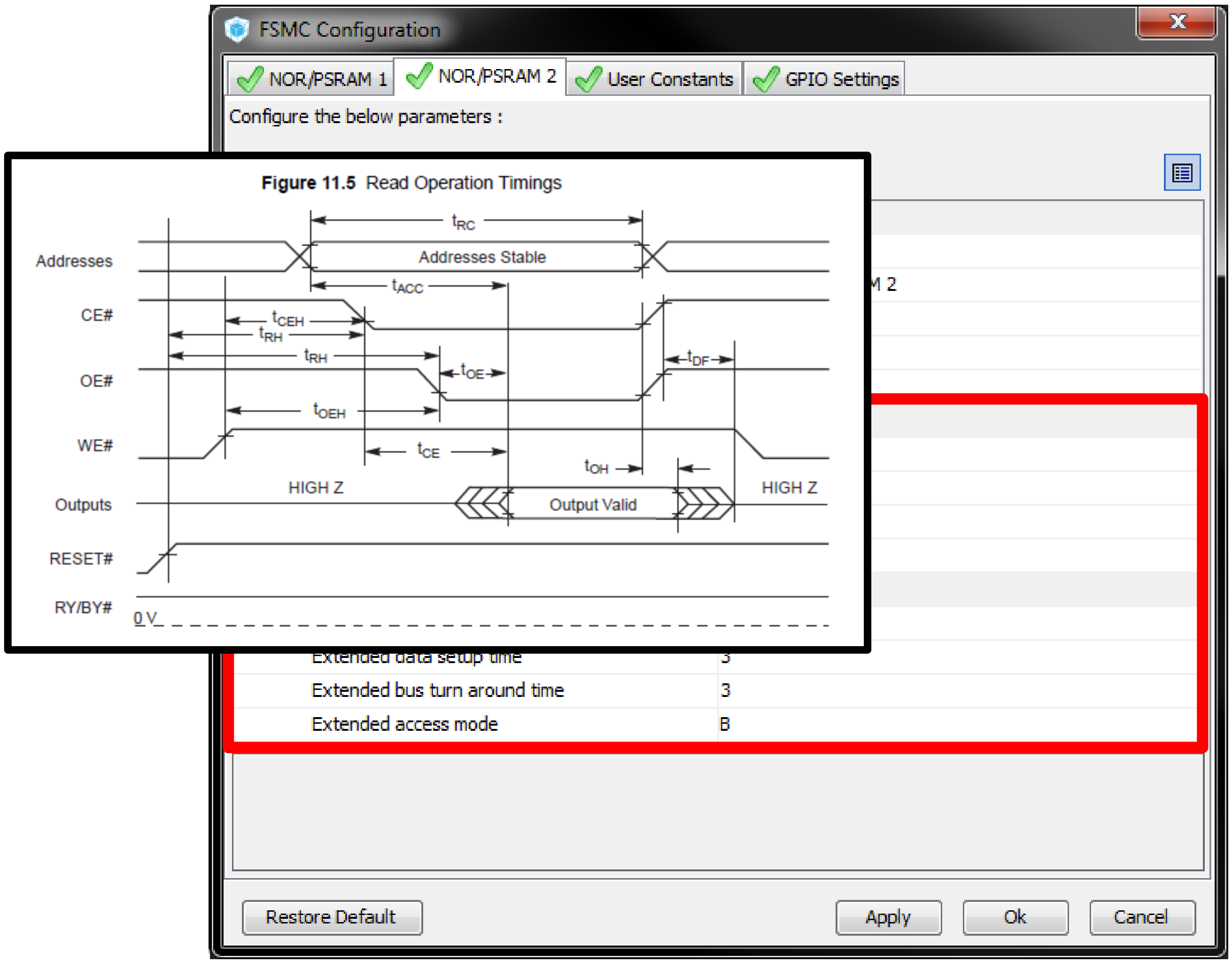
CubeMX FSMC Configuration for NOR



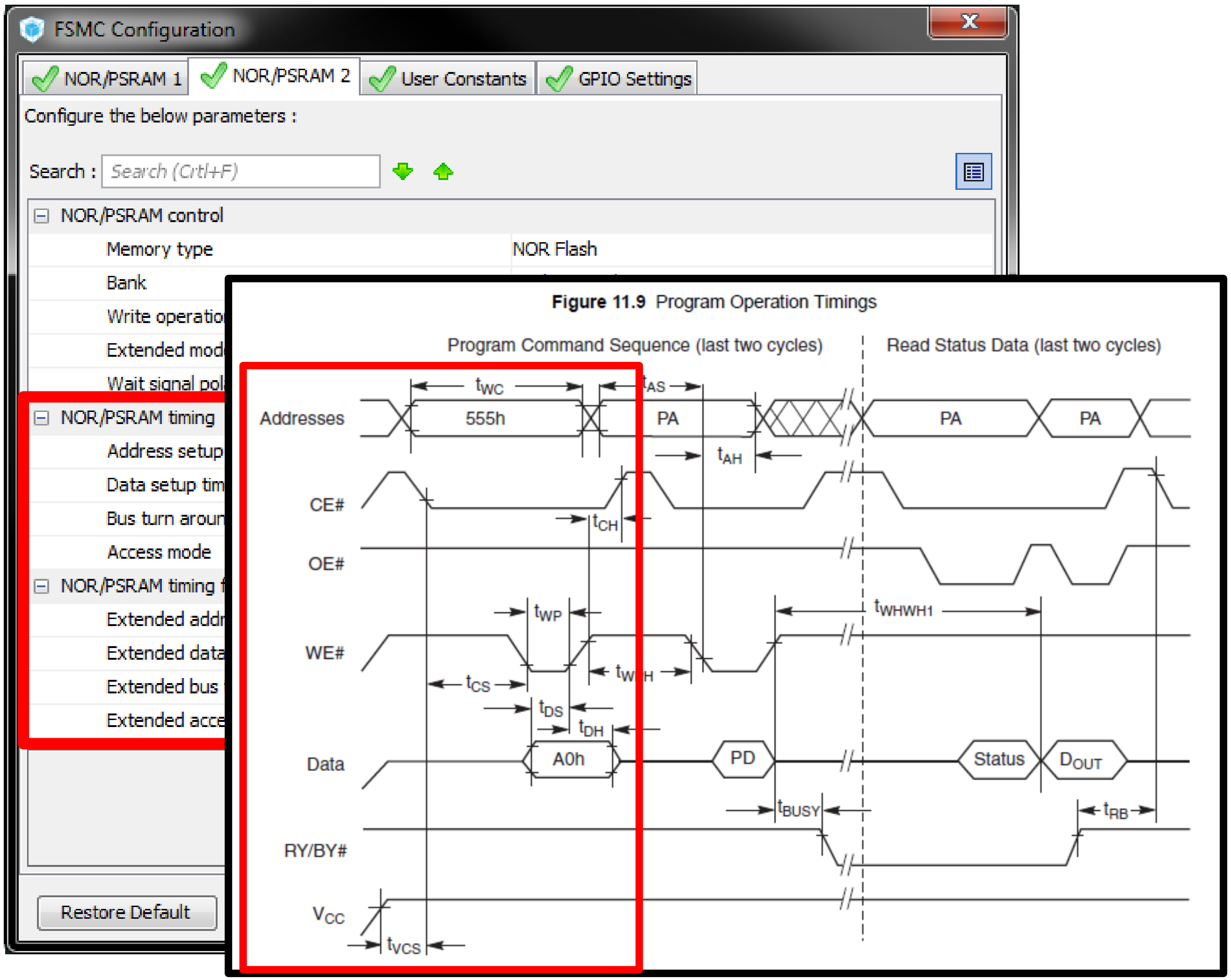
CubeMX FSMC Configuration for NOR



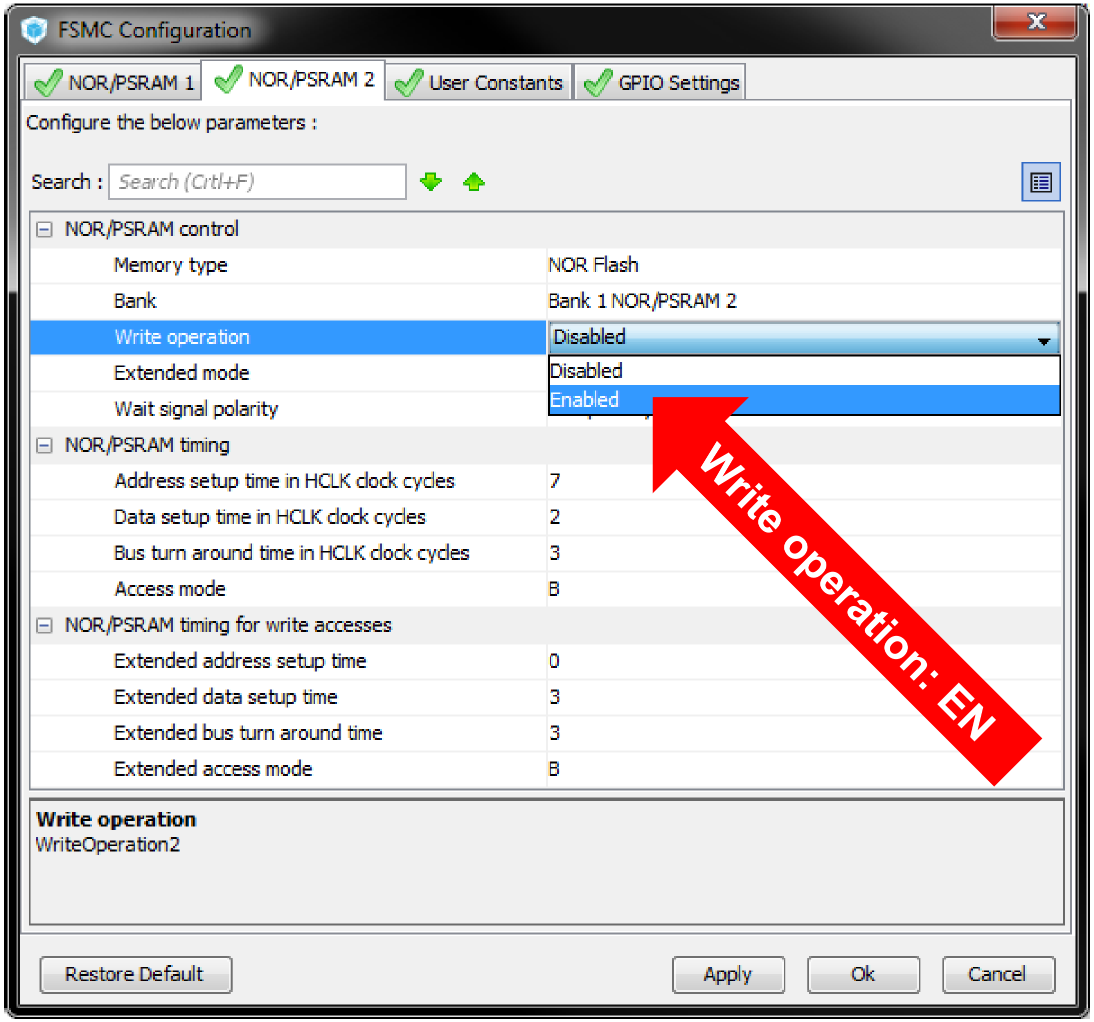
CubeMX FSMC Configuration for NOR



CubeMX FSMC Configuration for NOR



CubeMX FSMC Configuration for NOR



NOR Flash Memory HAL Code Example


```
/* BANK1-CS2 NOR */
HAL_NOR_Read_ID(&hnor2, &norID);
HAL_NOR_ReturnToReadMode(&hnor2);

HAL_NOR_Erase_Block(&hnor2, 0x64000000, 0);
HAL_NOR_GetStatus(&hnor2, 0x64000000, 500);

HAL_NOR_Program(&hnor2, (uint32_t *)0x64000000, (uint16_t *)"ST");
HAL_NOR_GetStatus(&hnor2, 0x64000000, 1);
```

NOR Flash Memory Command Sequence

- Memory Command (8-bit bus)

	Command (Notes)	Cycles	Bus Cycles (Notes 1–5) First Addr	First Data	Second Addr	Second Data	Third Addr	Third Data	Fourth Addr	Fourth Data	Fifth Addr	Fifth Data	Sixth Addr	Sixth Data
	Read (6)	1	RA	RD										
	Reset (7)	1	XXX	F0										
Autoselect (8,9)	Manufacturer ID	4	AAA	AA	555	55	AAA	90	X00	01				
	Device ID (8)	6	AAA	AA	555	55	AAA	90	X02	XX7E	X1C	(8)	X1E	(8)
	Sector Protect Verify (10)	4	AAA	AA	555	55	AAA	90	[SA]X04	(10)				
	Secure Device Verify (11)	4	AAA	AA	555	55	AAA	90	X06	(11)				

- Memory Command (16-bit bus)

	Command (Notes)	Cycles	Bus Cycles (Notes 1–5) First Addr	First Data	Second Addr	Second Data	Third Addr	Third Data	Fourth Addr	Fourth Data	Fifth Addr	Fifth Data	Sixth Addr	Sixth Data
	Read (6)	1	RA	RD										
	Reset (7)	1	XXX	F0										
Autoselect (8,9)	Manufacturer ID	4	555	AA	2AA	55	555	90	X00	01				
	Device ID (8)	6	555	AA	2AA	55	555	90	X01	227E	X0E	(8)	X0F	(8)
	Sector Protect Verify (10)	4	555	AA	2AA	55	555	90	[SA]X02	(10)				
	Secure Device Verify (11)	4	555	AA	2AA	55	555	90	X03	(11)				

NOR Flash Memory Command Sequence

- Memory Command (8-bit bus)

Command (Notes)		Cycles	Bus Cycles (Notes 1–5)											
			First		Second		Third		Fourth		Fifth		Sixth	
			Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read (6)		1	RA	RD										
Reset (7)		1	XXX	F0										
Autoselect (8,9)	Manufacturer ID	4	AAA	AA	555	55	AAA	90	X00	01				
Autoselect (8,9)	Device ID (8)	6	AAA	AA	555	55	AAA	90	X02	XX7E	X1C	(8)	X1E	(8)
Autoselect (8,9)	Sector Protect Verify (10)	4	AAA	AA	555	55	AAA	90	[SA]X04	(10)				
Autoselect (8,9)	Secure Device Verify (11)	4	AAA	AA	555	55	AAA	90	X06	(11)				

- Memory Command (16-bit bus)

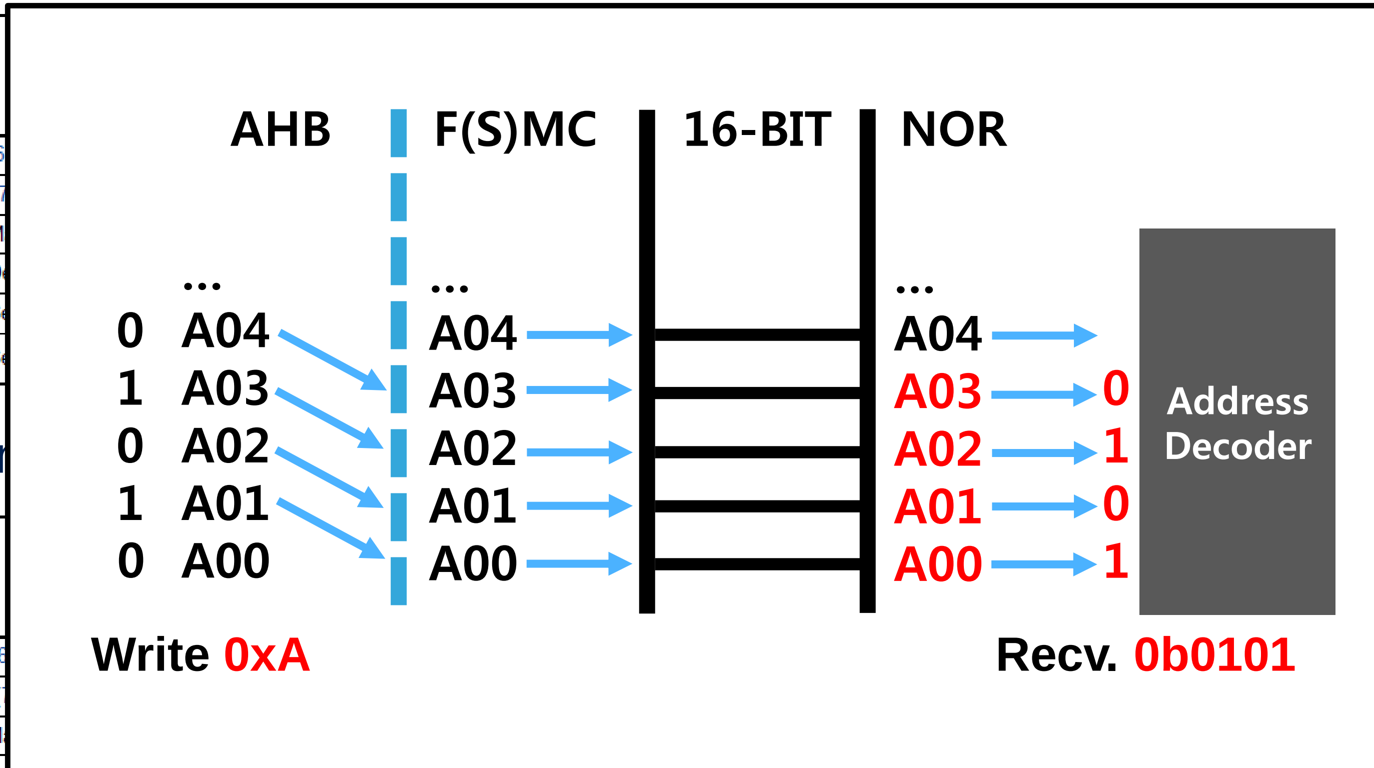
Command (Notes)		Cycles	Bus Cycles (Notes 1–5)											
			First		Second		Third		Fourth		Fifth		Sixth	
			Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read (6)		1	RA	RD										
Reset (7)		1	XXX	F0										
Autoselect (8,9)	Manufacturer ID	4	555	AA	2AA	55	555	90	X00	01				
Autoselect (8,9)	Device ID (8)	6	555	AA	2AA	55	555	90	X01	227E	X0E	(8)	X0F	(8)
Autoselect (8,9)	Sector Protect Verify (10)	4	555	AA	2AA	55	555	90	[SA]X02	(10)				
Autoselect (8,9)	Secure Device Verify (11)	4	555	AA	2AA	55	555	90	X03	(11)				

life.augmented

NOR Flash Memory Command Sequence

- Memory Command (8-bit bus)

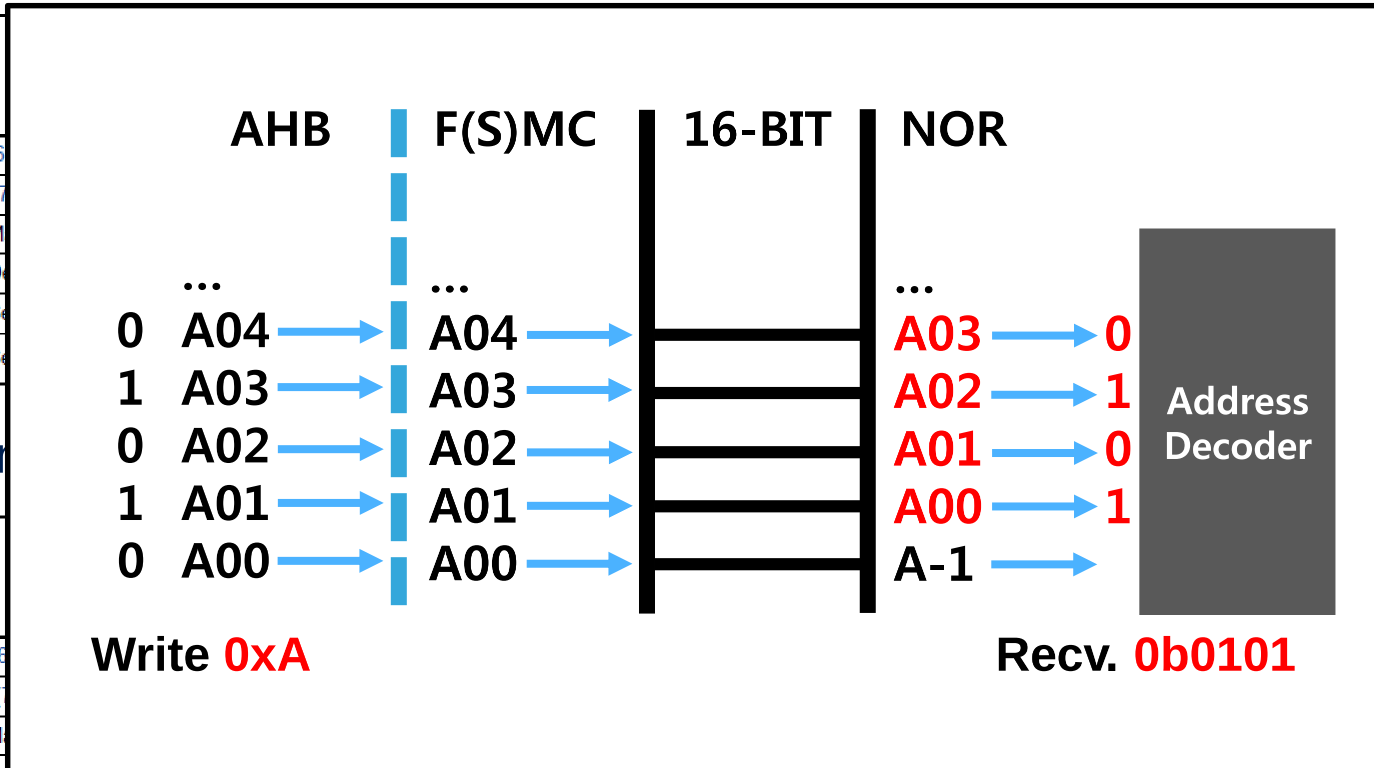
- Mer



Autoselect (8,9)	Sector Protect Verify (10)	4	555	AA	2AA	55	555	90	[SA]X02	(10)					
	Secure Device Verify (11)	4	555	AA	2AA	55	555	90	X03	(11)					

NOR Flash Memory Command Sequence

- Memory Command (8-bit bus)

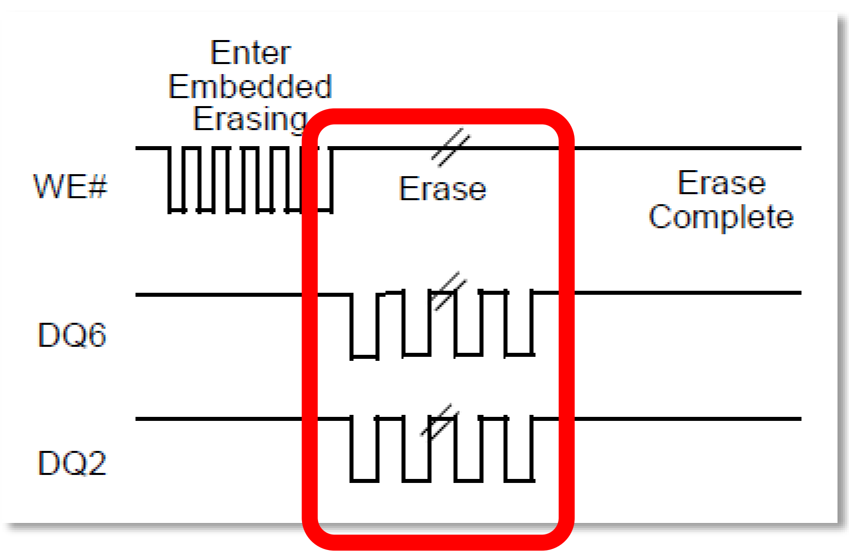


Autoselect (8,9)	Sector Protect Verify (10)	4	555	AA	2AA	55	555	90	[SA]X02	(10)				
	Secure Device Verify (11)	4	555	AA	2AA	55	555	90	X03	(11)				

NOR Flash Memory Polling Sequence


```
/* BANK1-CS2 NOR */
HAL_NOR_Read_ID(&hnor2, &norID);
HAL_NOR_ReturnToReadMode(&hnor2);

HAL_NOR_Erase_Block(&hnor2, 0x64000000, 0);
HAL_NOR_GetStatus(&hnor2, 0x64000000, 500);

HAL_NOR_Program(&hnor2, (uint32_t *)0x64000000, (uint16_t *)"ST");
HAL_NOR_GetStatus(&hnor2, 0x64000000, 1);
```

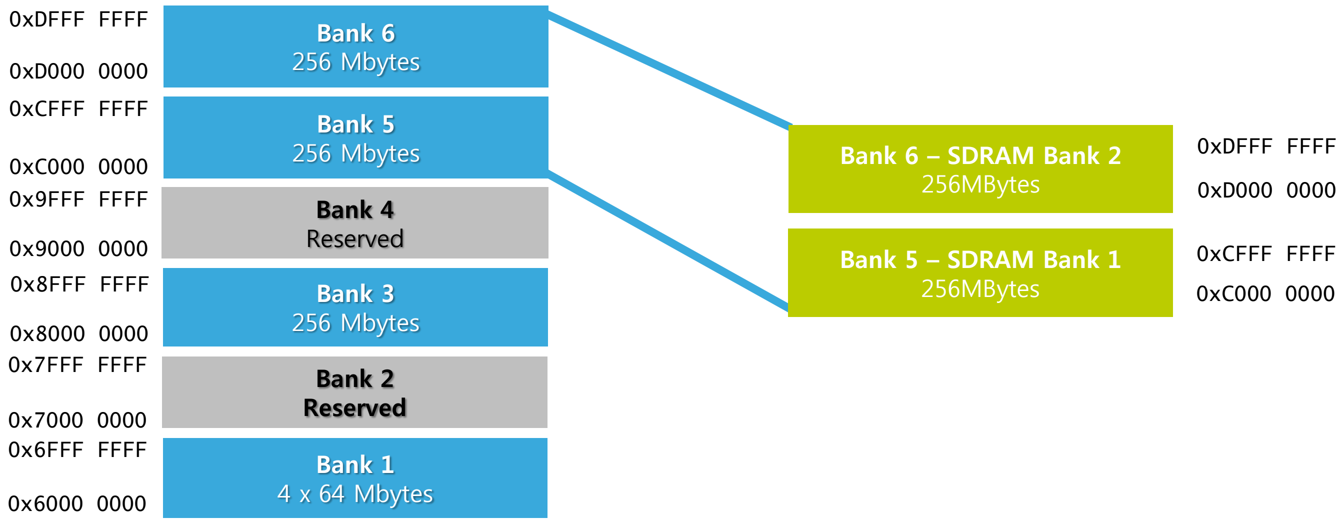
- NOR Status checking by bus polling (DQn)



life.augmented

SDRAM address mapping

- Fully programmable SDR (single data rate) SDRAM interface
- Up to 512MB continues memory range split into two banks, can be seen as a single device.
  - Each Bank can address up to 256MBytes memory



life.augmented

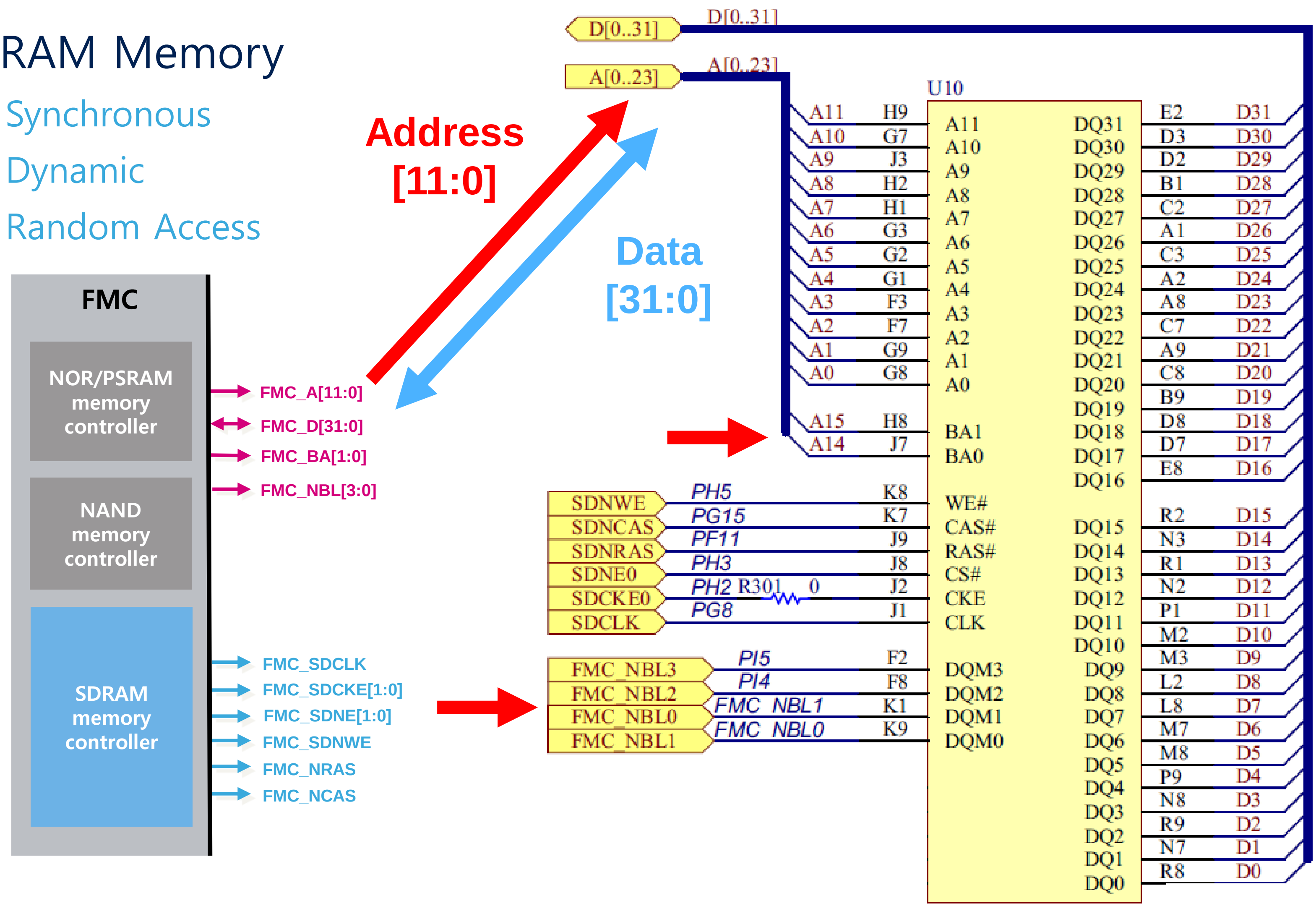
SDRAM address mapping

- Fully programmable SDR (single data rate) SDRAM interface
- Up to 512MB continues memory range split into two banks, can be seen as a single device.
  - Each Bank can address up to 256MBytes memory

life.augmented

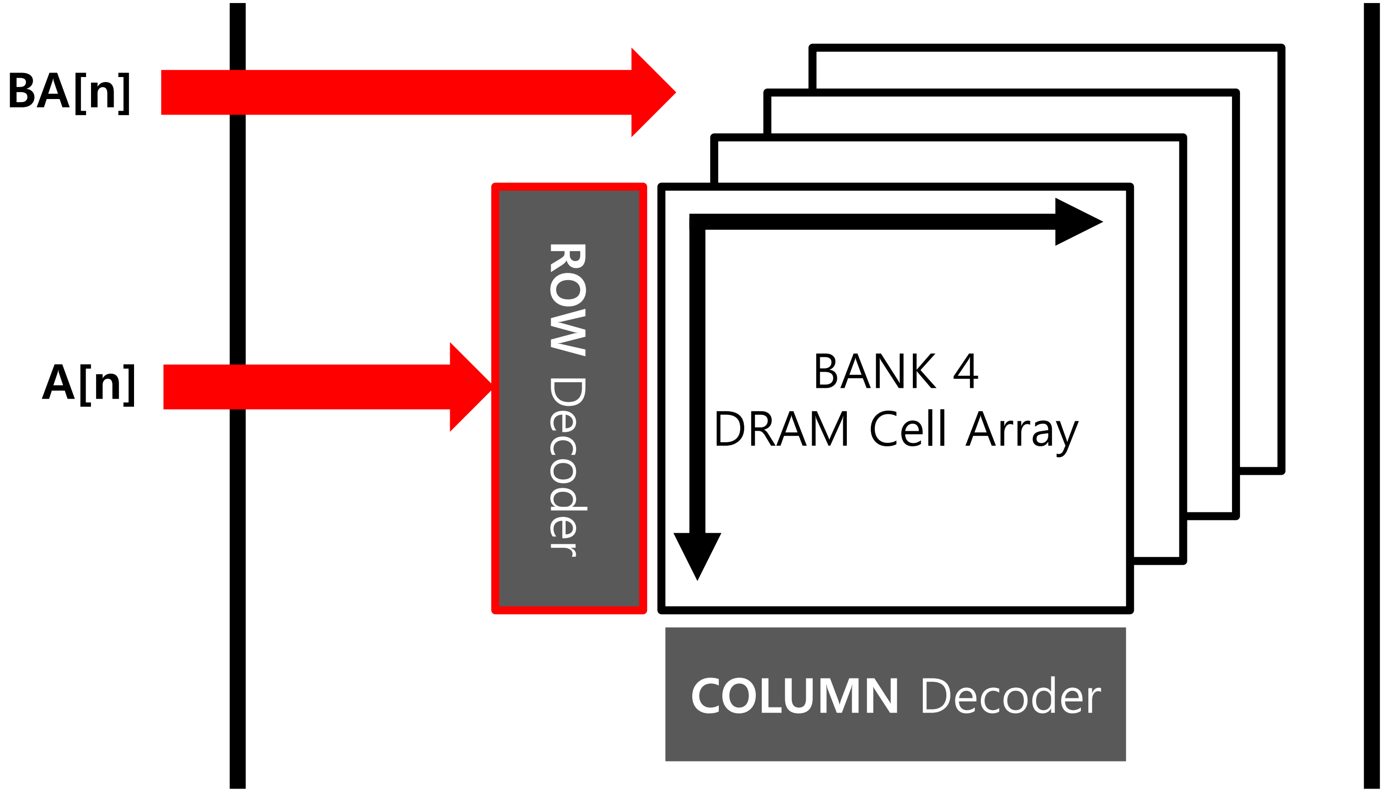
SDRAM interface signals

- SDRAM Memory
  - Synchronous
  - Dynamic
  - Random Access



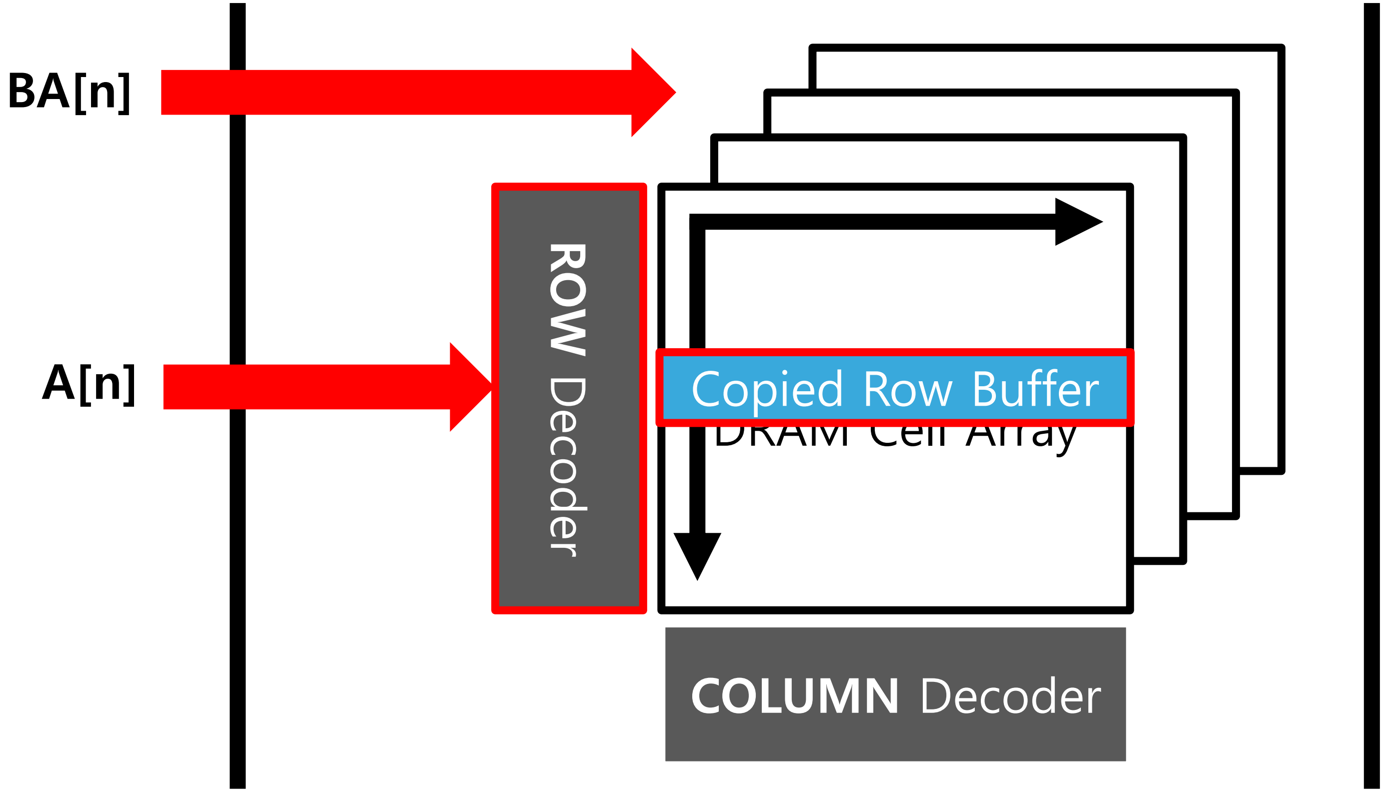
About SDRAM Memory

- SDRAM Memory Structure



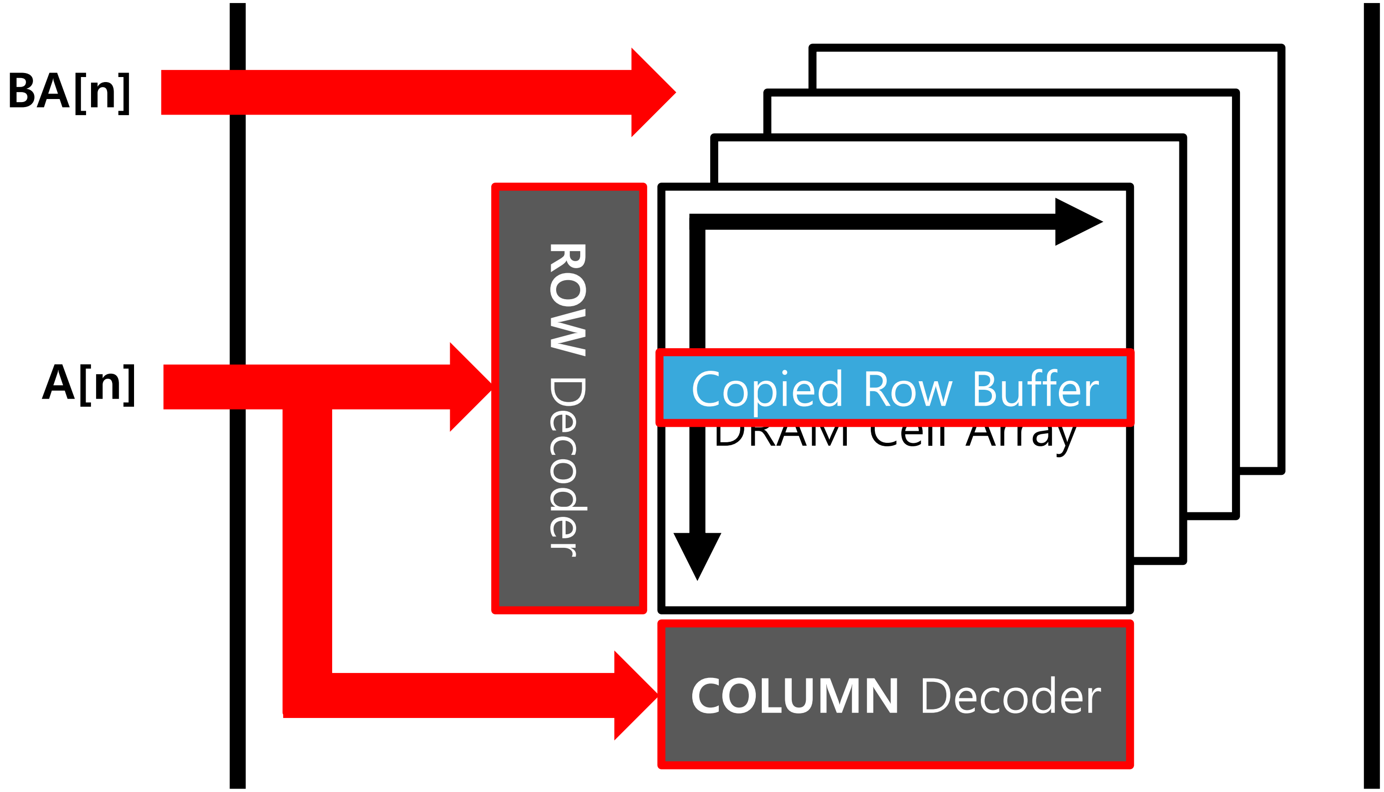
About SDRAM Memory

- SDRAM Memory Structure



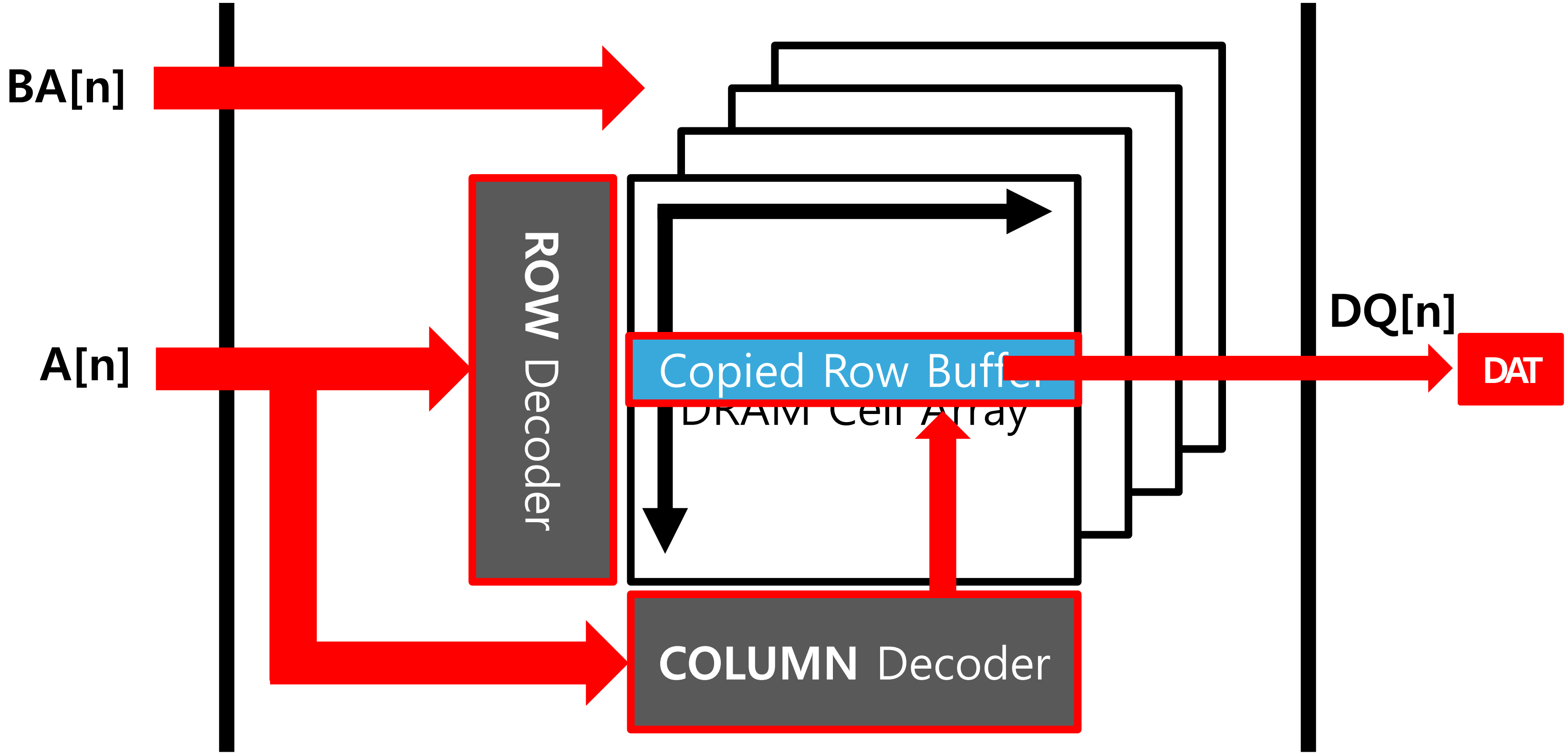
About SDRAM Memory

- SDRAM Memory Structure



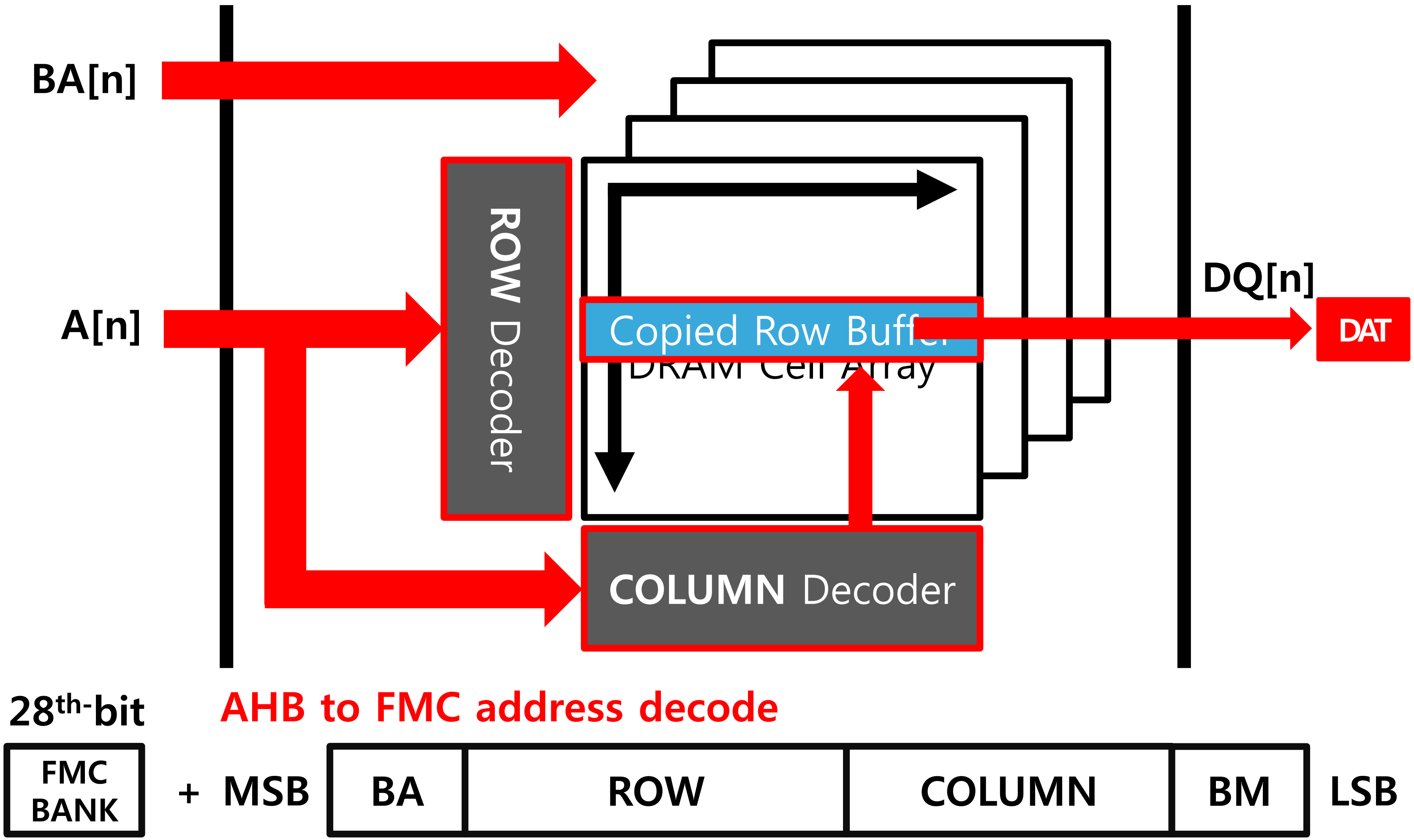
About SDRAM Memory

- SDRAM Memory Structure



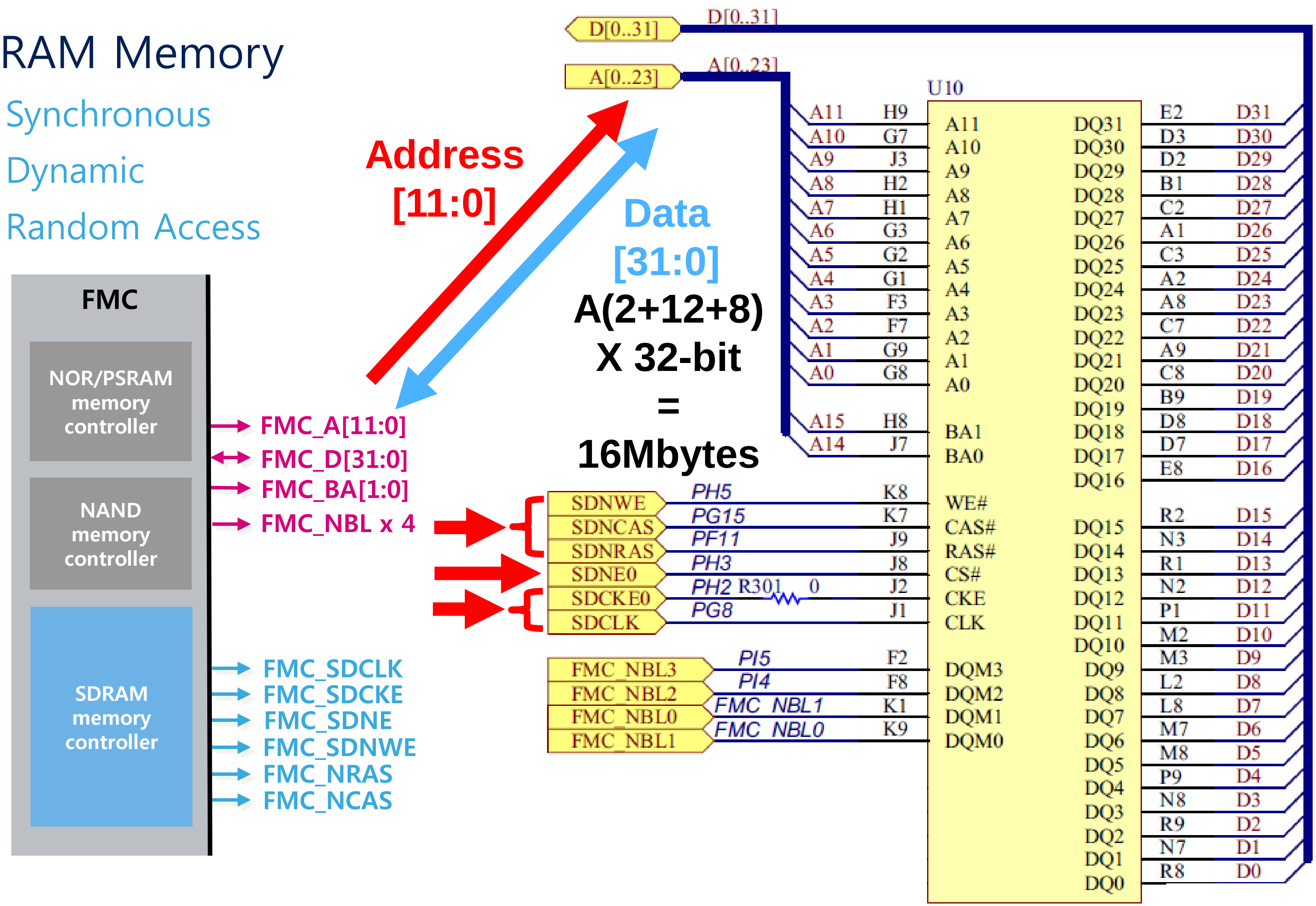
About SDRAM Memory

- SDRAM Memory Structure



SDRAM interface signals

- SDRAM Memory
  - Synchronous
  - Dynamic
  - Random Access

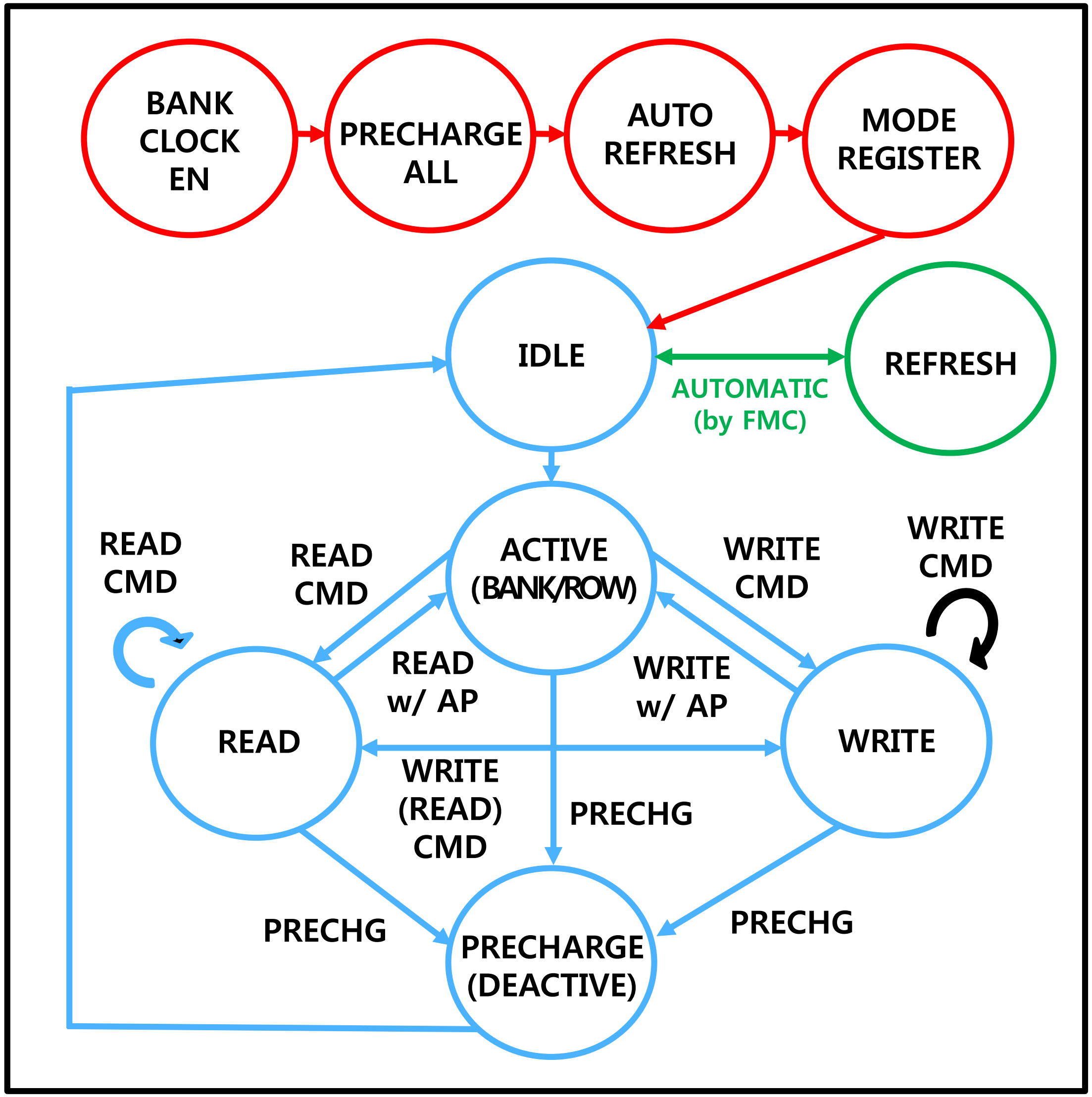


SDRAM Commands

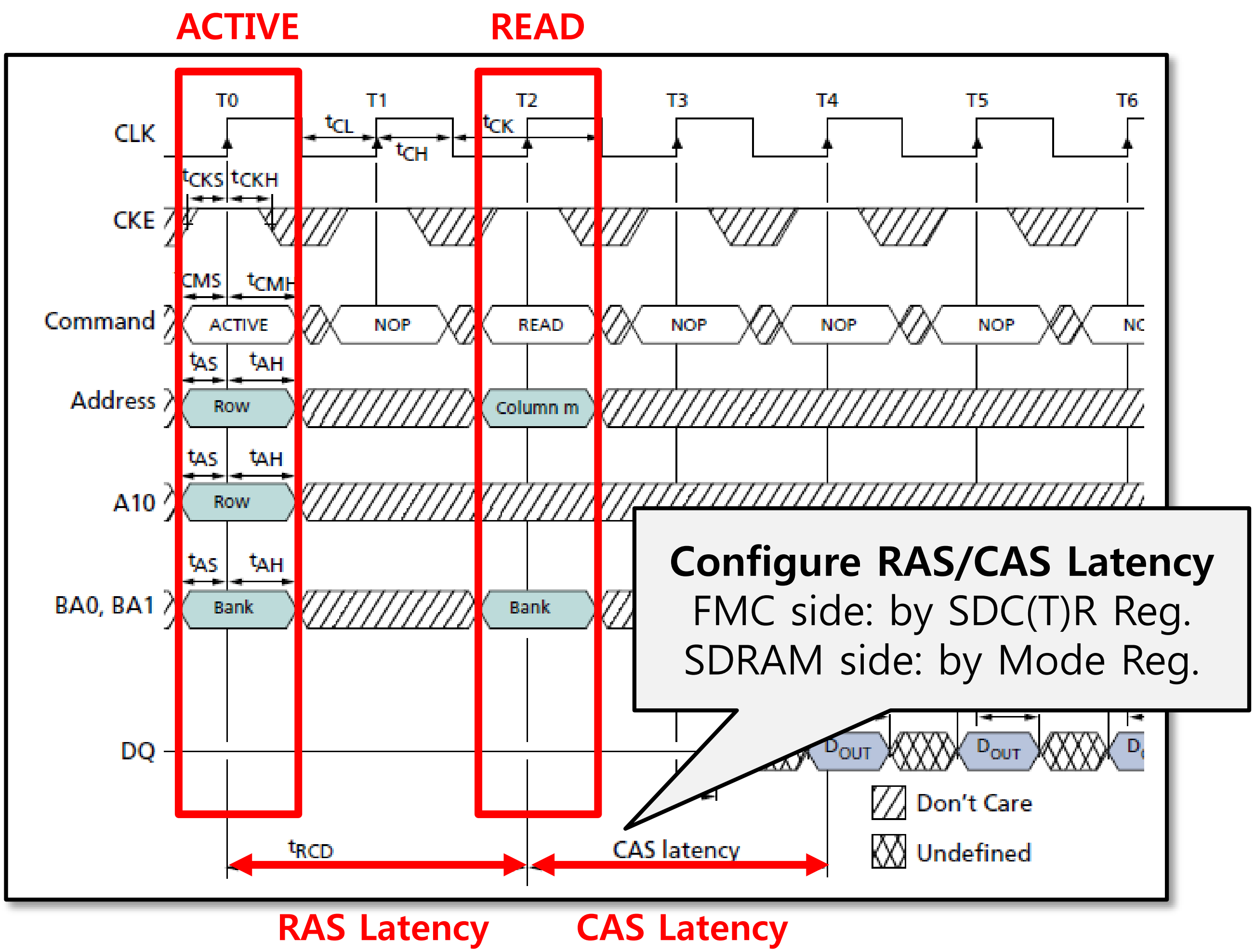
Name (Function)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQ	Notes
COMMAND INHIBIT (NOP)	H	X	X	X	X	X	X	
NO OPERATION (NOP)	L	H	H	H	X	X	X	
ACTIVE (select bank and activate row)	L	L	H	H	X	Bank/row	X	2
READ (select bank and column, and start READ burst)	L	H	L	H	L/H	Bank/col	X	3
WRITE (select bank and column, and start WRITE burst)	L	H	L	L	L/H	Bank/col	Valid	3
BURST TERMINATE	L	H	H	L	X	X	Active	4
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	X	Code	X	5
AUTO REFRESH or SELF REFRESH (enter self refresh mode)	L	L	L	H	X	X	X	6, 7
LOAD MODE REGISTER	L	L	L	L	X	Op-code	X	8
Write enable/output enable	X	X	X	X	L	X	Active	9
Write inhibit/output High-Z	X	X	X	X	H	X	High-Z	9

Notes: CKE is HIGH for all commands shown except SELF REFRESH.

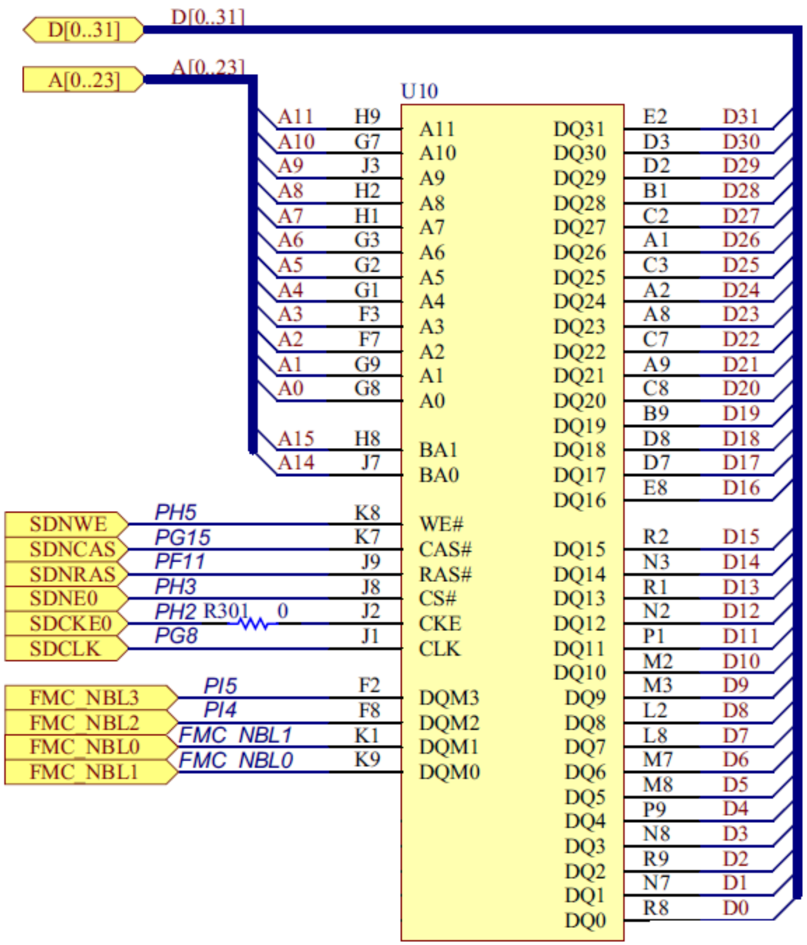
SDRAM State Diagram

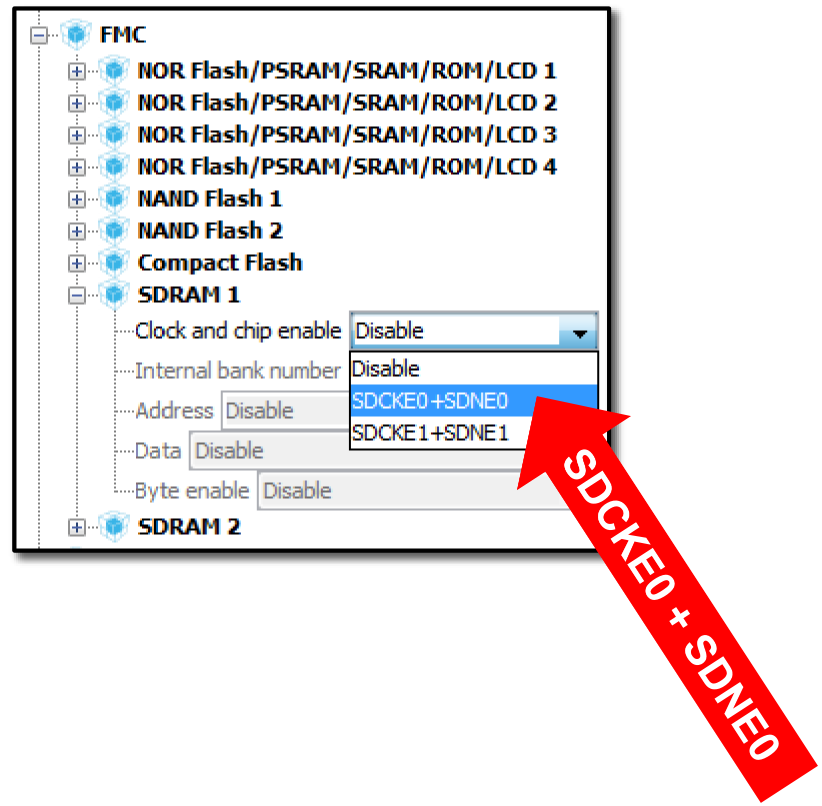


SDRAM READ Operation

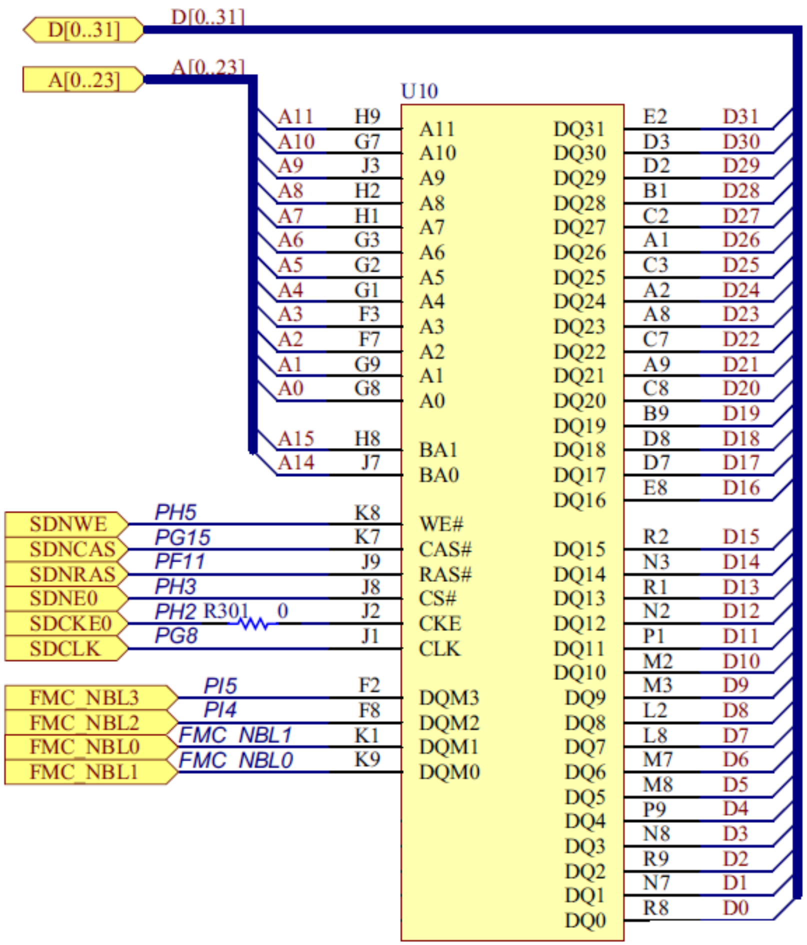


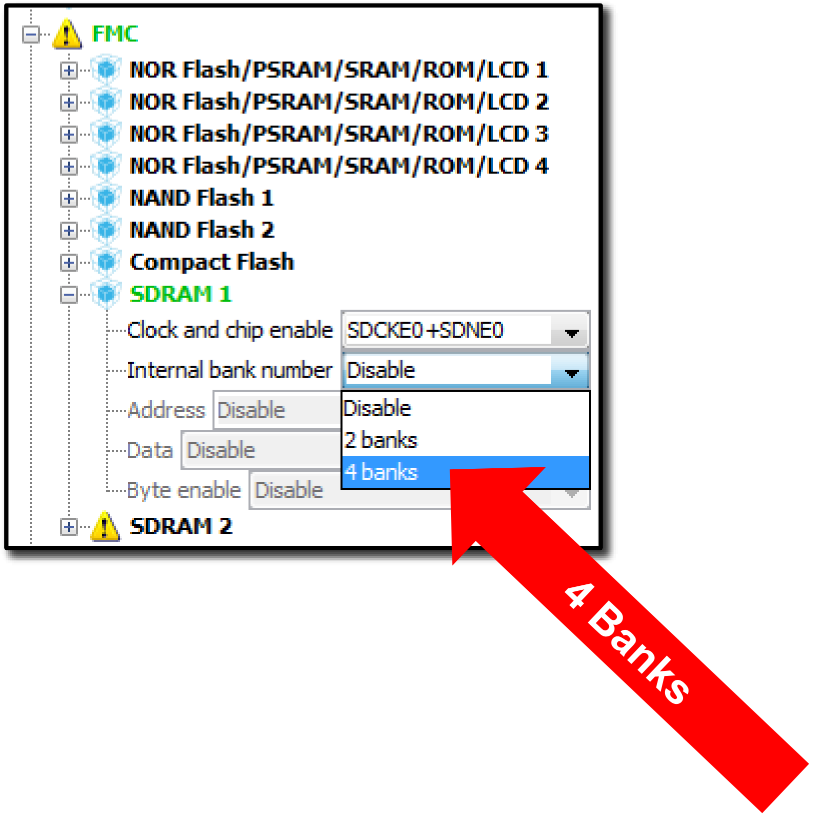
CubeMX Configuration for SDRAM Connection



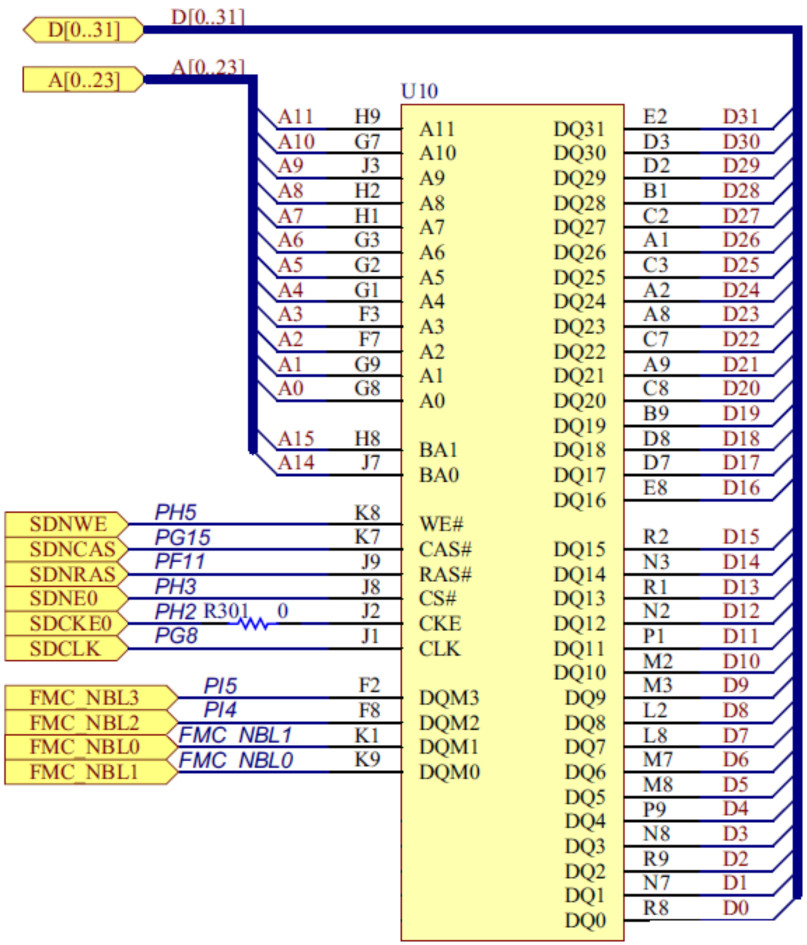


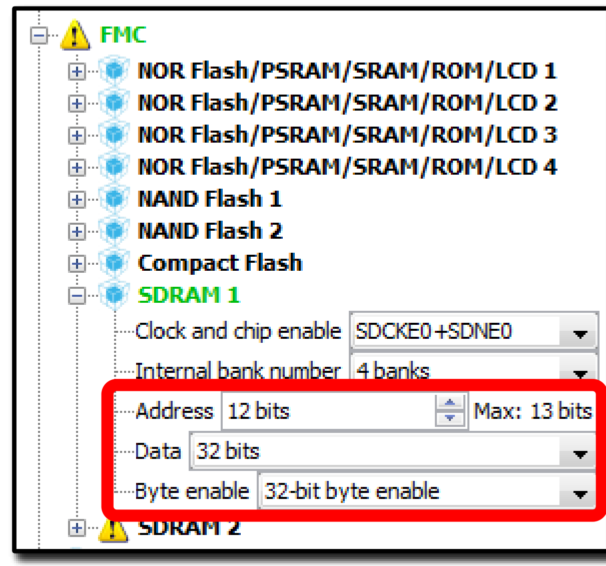
CubeMX Configuration for SDRAM Connection





CubeMX Configuration for SDRAM Connection



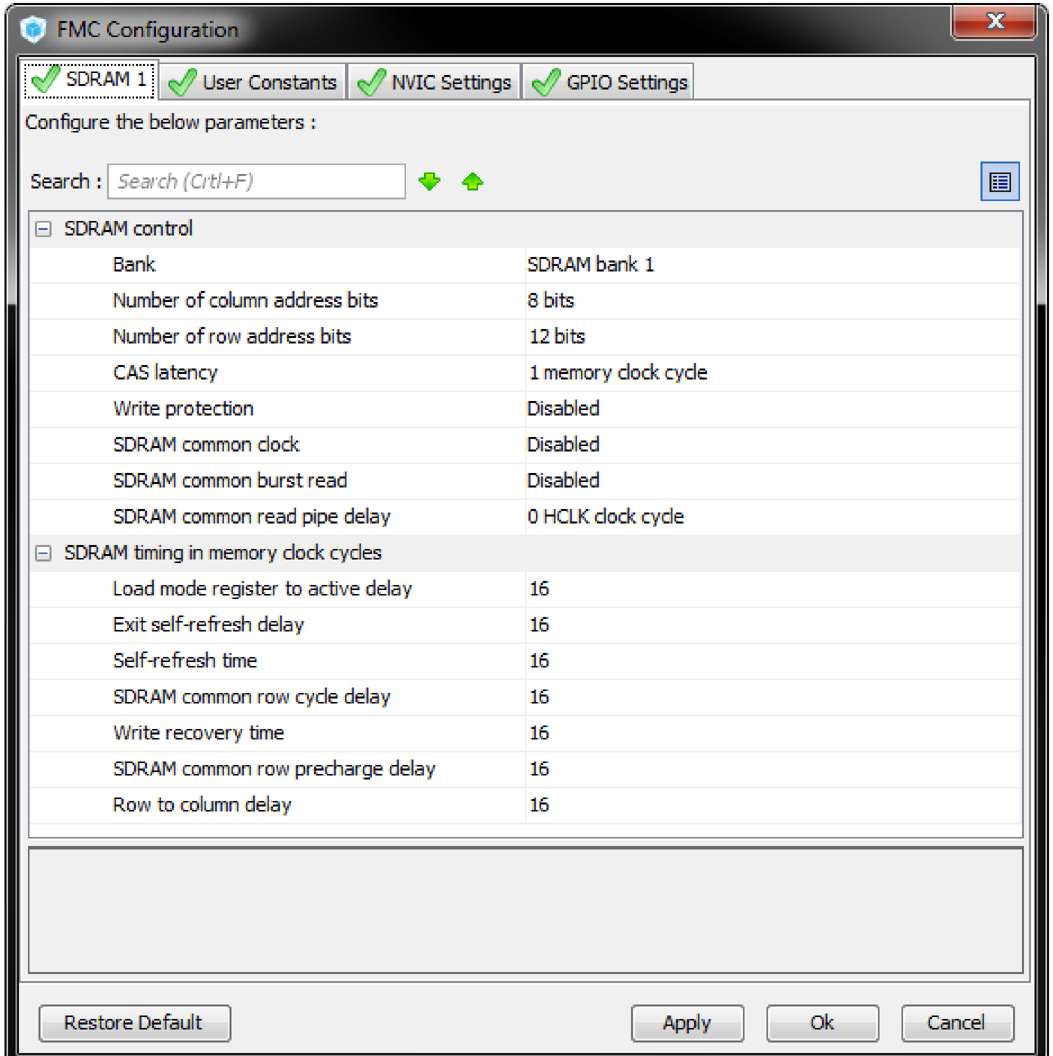


life.augmented

CubeMX FMC Configuration for SDRAM

AHB HCLK = 180MHz

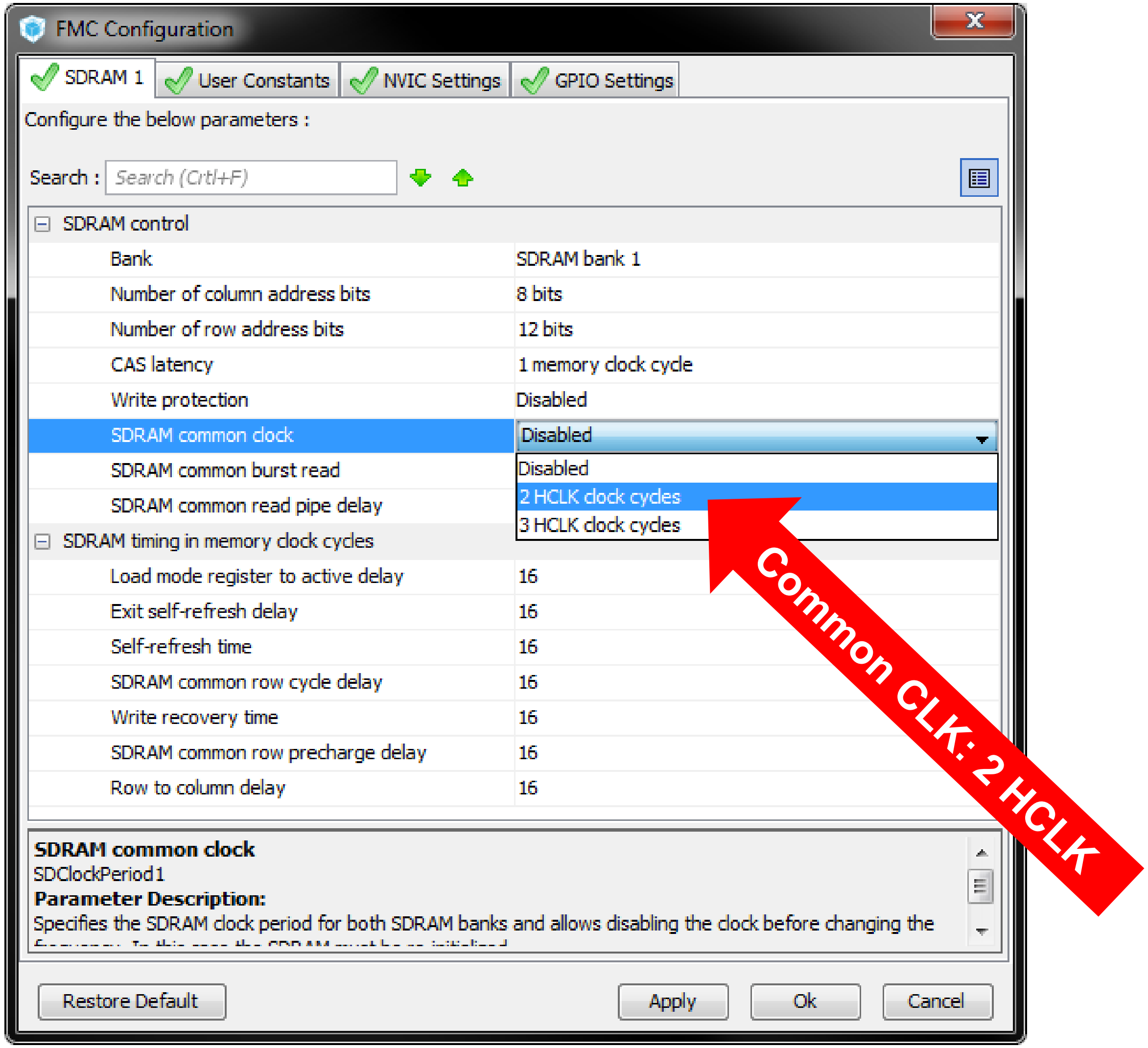
FMC CLK = 90MHz
(clock period: 11.111ns)



CubeMX FMC Configuration for SDRAM

AHB HCLK = 180MHz

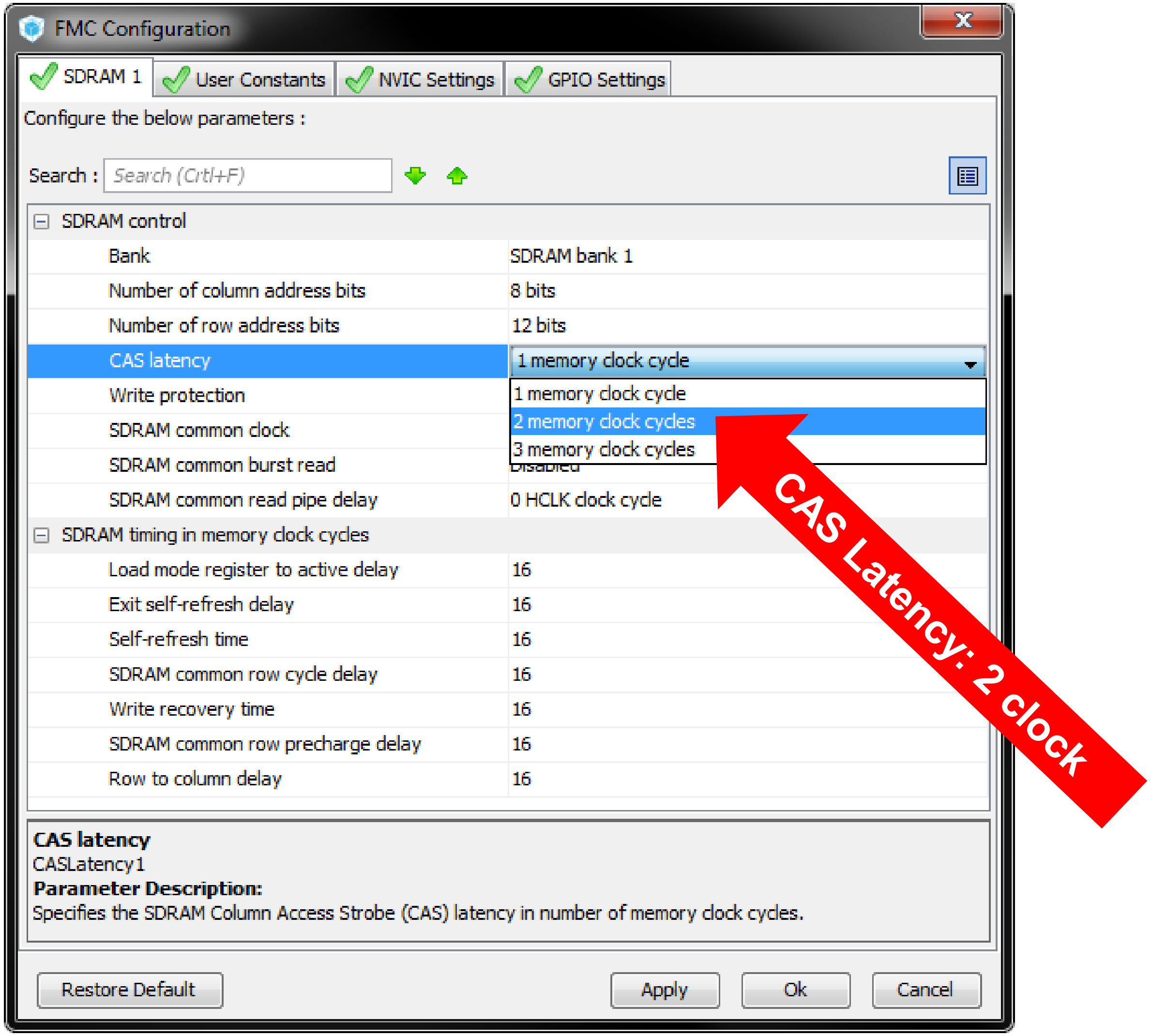
FMC CLK = 90MHz
(clock period: 11.111ns)



CubeMX FMC Configuration for SDRAM

AHB HCLK = 180MHz

FMC CLK = 90MHz
(clock period: 11.111ns)

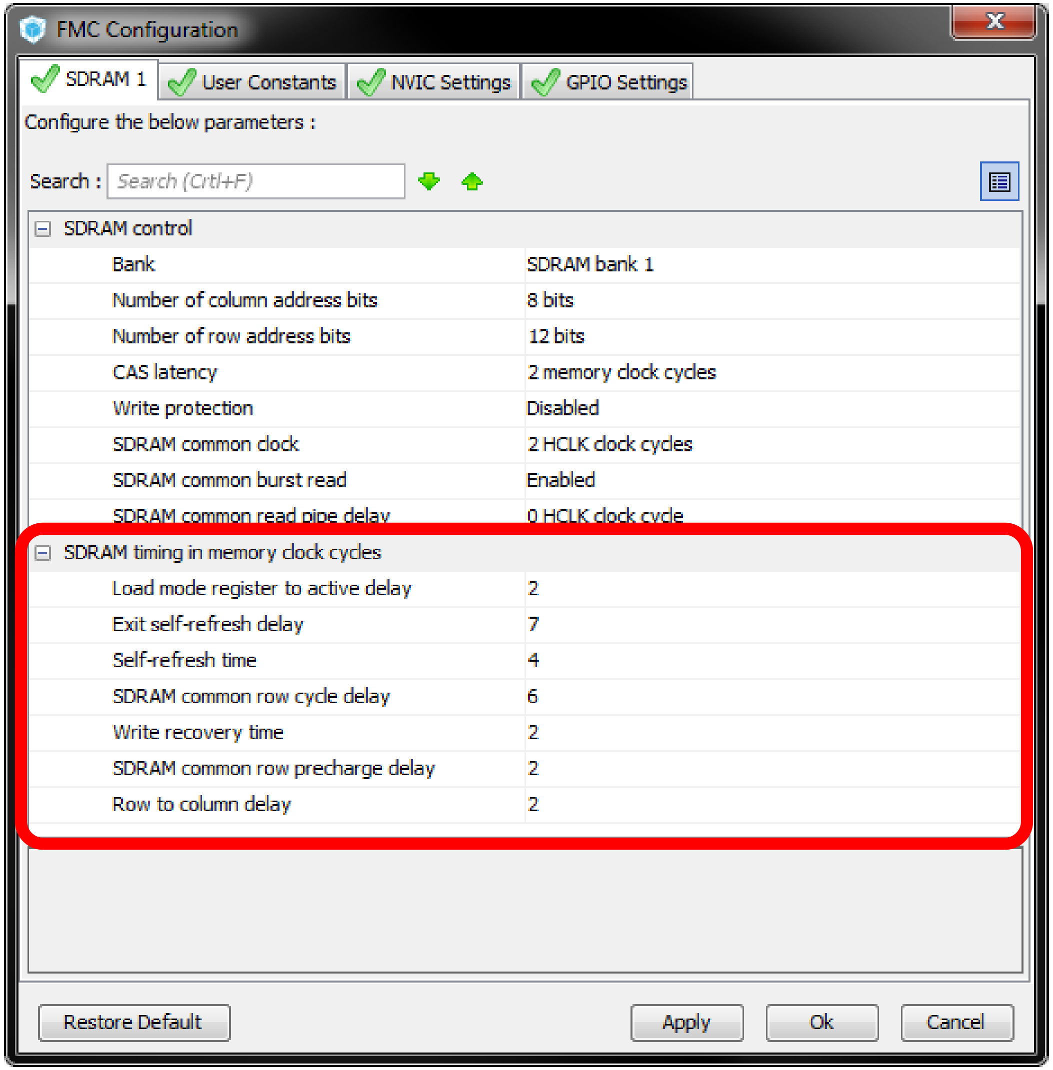


CubeMX FMC Configuration for SDRAM

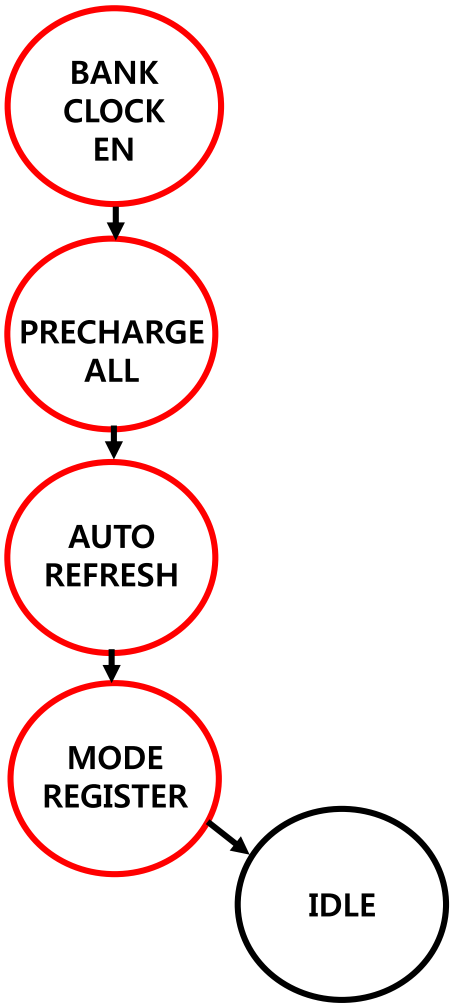
AHB HCLK = 180MHz

FMC CLK = 90MHz
(clock period: 11.111ns)

CAS Latency = 3 clock



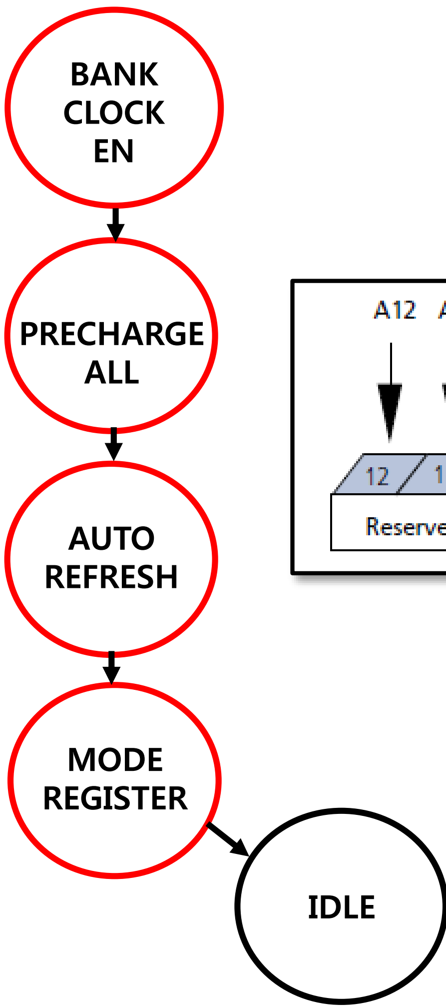
SDRAM Initialize Sequence



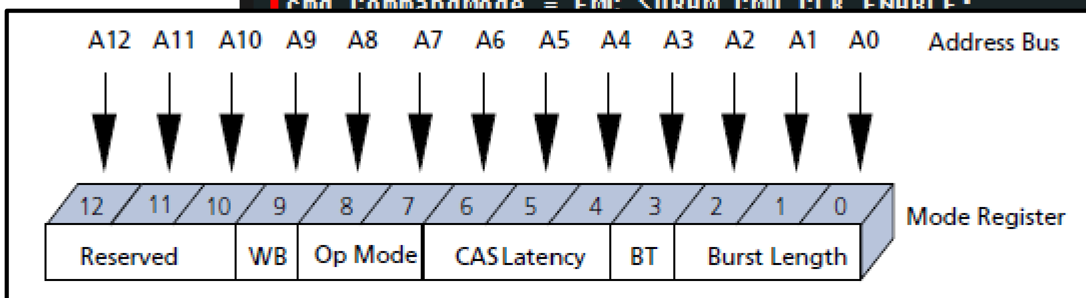

```
void SDRAM_Initialize(SDRAM_HandleTypeDef *hsdram)
{
  FMC_SDRAM_CommandTypeDef cmd;

  /* Step 1. Configure a clock configuration enable command */
  cmd.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
  cmd.CommandMode = FMC_SDRAM_CMD_CLK_ENABLE;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 2. Insert 100 us minimum delay */
  HAL_Delay(1);
  /* Step 3. Configure a PALL (precharge all) command */
  cmd.CommandMode = FMC_SDRAM_CMD_PALL;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 4. Configure a Auto-Refresh command */
  cmd.CommandMode = FMC_SDRAM_CMD_AUTOREFRESH_MODE;
  cmd.AutoRefreshNumber = 2;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 5. Program the external memory mode register */
  cmd.CommandMode = FMC_SDRAM_CMD_LOAD_MODE;
  cmd.ModeRegisterDefinition = 0x0220;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 6. Set the refresh rate counter
   * = 15.625us x freq(90MHz = 11.111ns) - 20
   * = 1386
   */
  HAL_SDRAM_ProgramRefreshRate(hsdram, 1386);
}
```

life.augmented

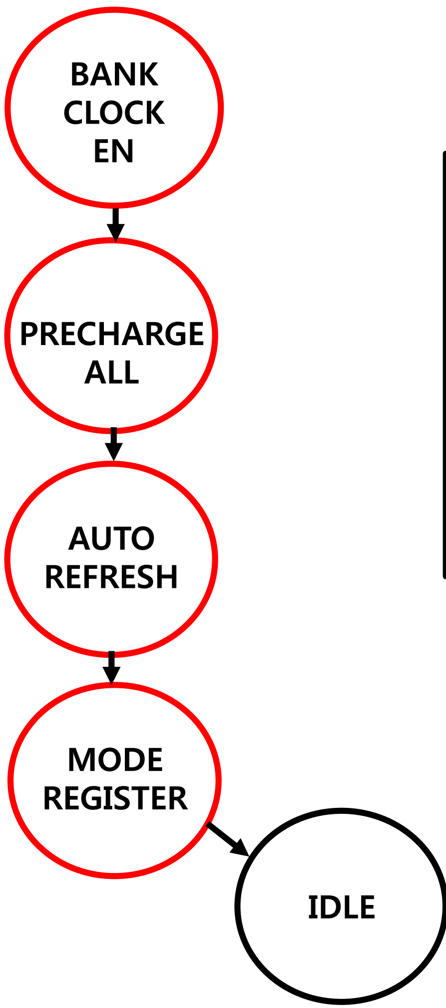
SDRAM Initialize Sequence



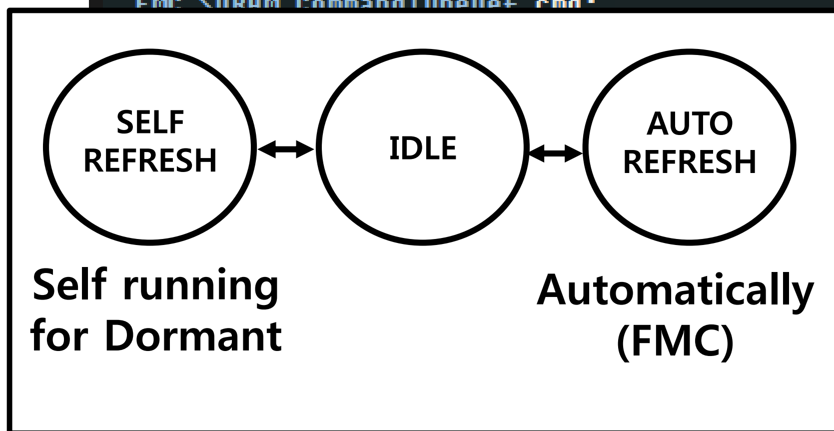

```
void SDRAM_Initialize(SDRAM_HandleTypeDef *hsdram)
{
  FMC_SDRAM_CommandTypeDef cmd;

  /* Step 1. Configure a clock configuration enable command */
  cmd.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
  cmd.CommandMode = FMC_SDRAM_CMD_CLK_ENABLE;
  ...
                                                    */
  ...
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 5. Program the external memory mode register */
  cmd.CommandMode = FMC_SDRAM_CMD_LOAD_MODE;
  cmd.ModeRegisterDefinition = 0x0220;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 6. Set the refresh rate counter
   * = 15.625us x freq(90MHz = 11.111ns) - 20
   * = 1386
   */
  HAL_SDRAM_ProgramRefreshRate(hsdram, 1386);
}
```



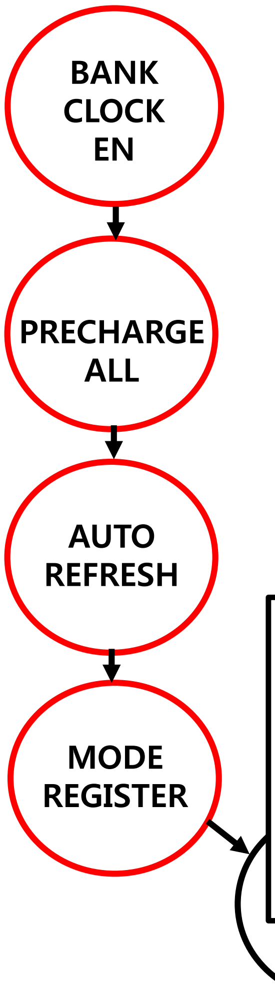
SDRAM Initialize Sequence

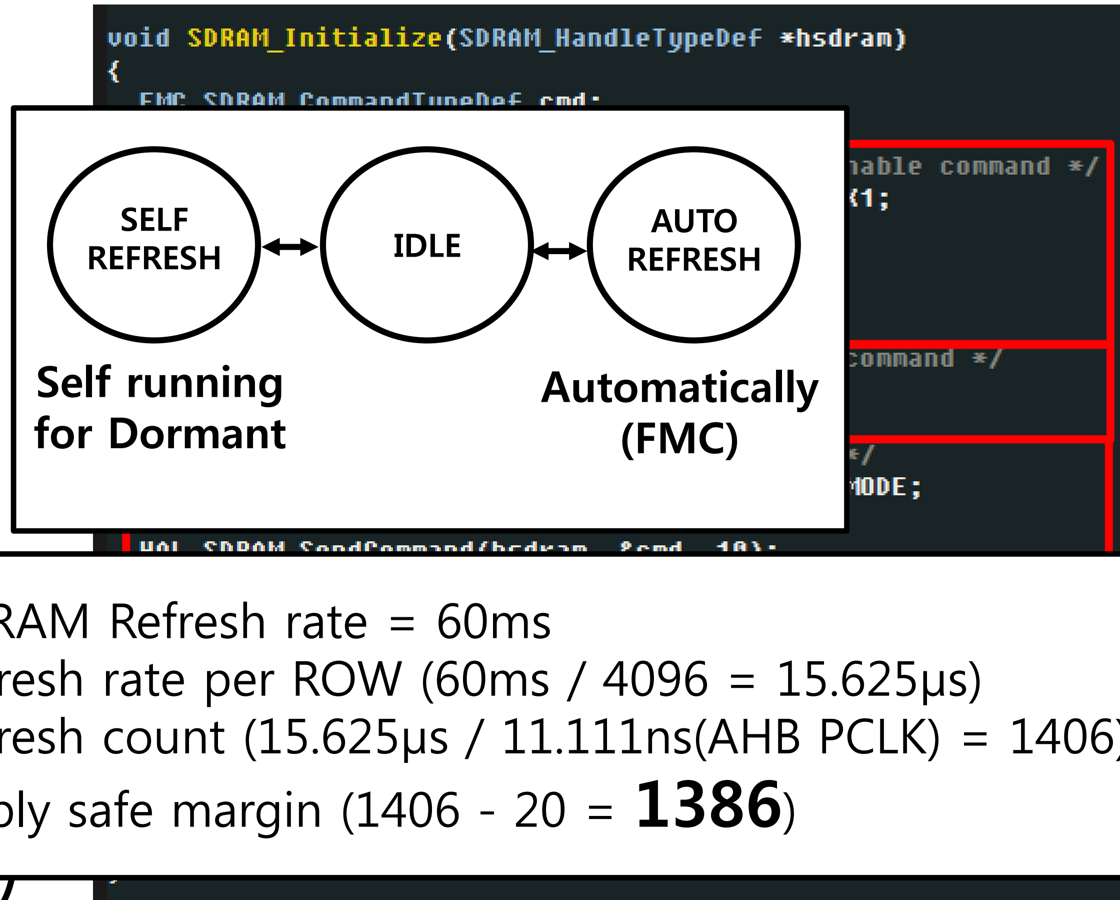



```
void SDRAM_Initialize(SDRAM_HandleTypeDef *hsdram)
{
  FMC_SDRAM_CommandTypeDef cmd;
                                                   nable command */
                                                   (1;

                                                   command */

                                                   */
                                                   MODE;

  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 5. Program the external memory mode register */
  cmd.CommandMode = FMC_SDRAM_CMD_LOAD_MODE;
  cmd.ModeRegisterDefinition = 0x0220;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Step 6. Set the refresh rate counter
   * = 15.625us x freq(90MHz = 11.111ns) - 20
   * = 1386
   */
  HAL_SDRAM_ProgramRefreshRate(hsdram, 1386);
}
```



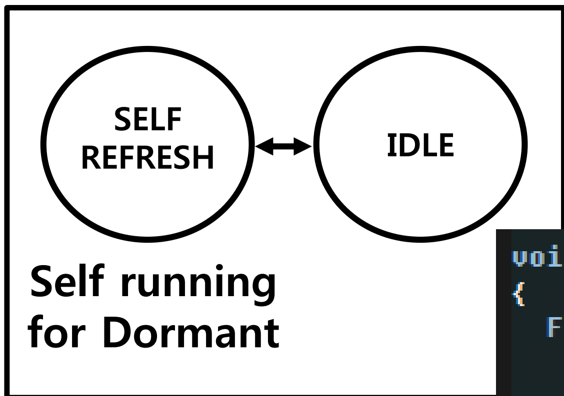
SDRAM Initialize Sequence





1. SDRAM Refresh rate = 60ms
2. Refresh rate per ROW (60ms / 4096 = 15.625μs)
3. Refresh count (15.625μs / 11.111ns(AHB PCLK) = 1406)
4. Apply safe margin (1406 - 20 = **1386**)

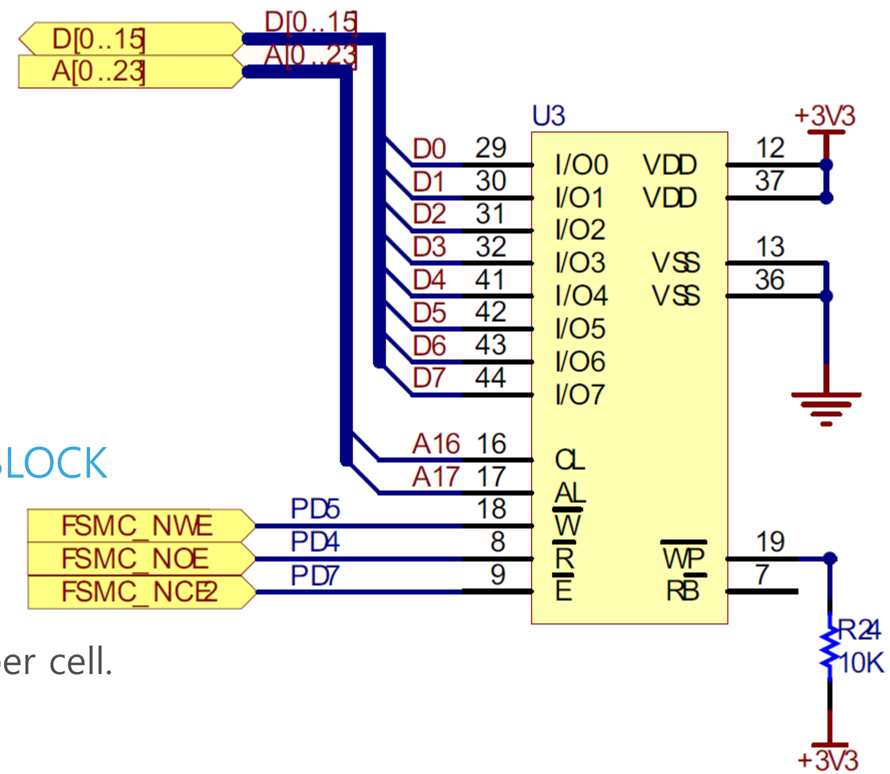
SDRAM Initialize Sequence



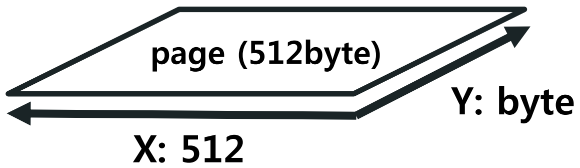

```c
void STOP_EntryExit(SDRAM_HandleTypeDef *hsdram)
{
  FMC_SDRAM_CommandTypeDef cmd;

  __asm volatile ("cpsid i");

  /* Issue self-refresh mode */
  cmd.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
  cmd.CommandMode = FMC_SDRAM_CMD_SELFREFRESH_MODE;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);
  /* Enter STOP mode */
  HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON, PWR_STOPENTRY_WFI);
  SystemClock_Config();
  /* Trun to normal mode */
  cmd.CommandMode = FMC_SDRAM_CMD_NORMAL_MODE;
  HAL_SDRAM_SendCommand(hsdram, &cmd, 10);

  __asm volatile ("cpsie i");
}
```

About NAND Flash Memory

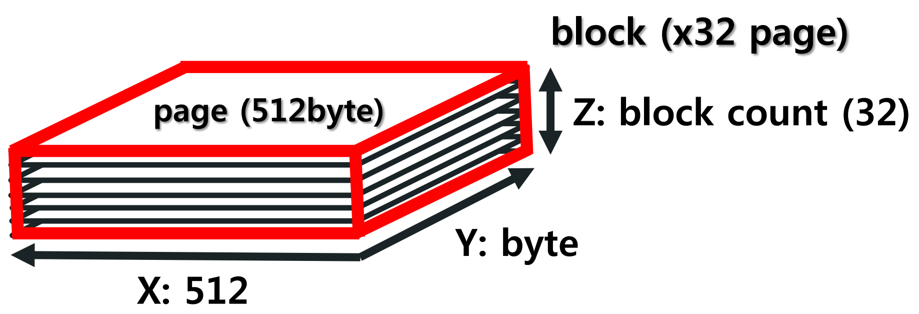
- NAND Flash memory
  - **Advantage**
    - Non-volatile
    - High capacities and cost.
    - Fast program/erase op.
    - Simple interface
  - **Disadvantage**
    - Read/Program w/ PAGE, Erase w/ BLOCK
    - Bad cell management
    - Worst endurance
      - Generally 3~4K program/erase time per cell.



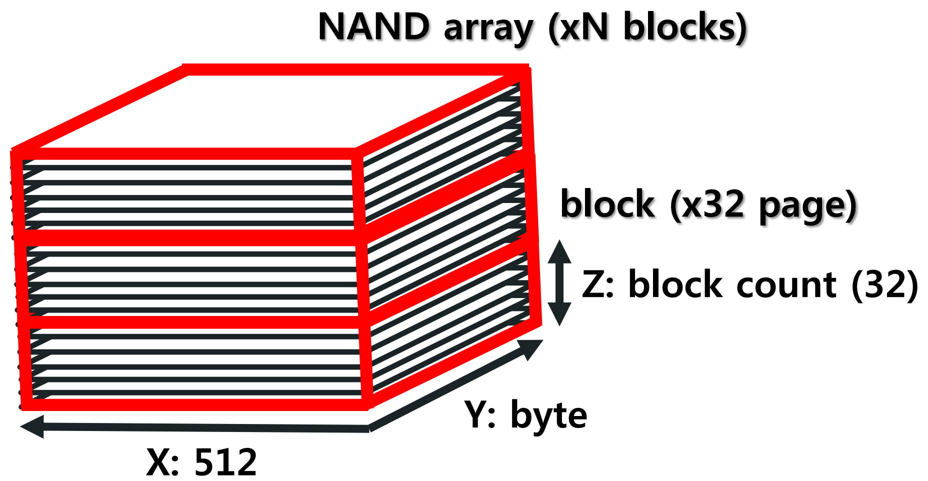
About NAND Flash Memory Structure



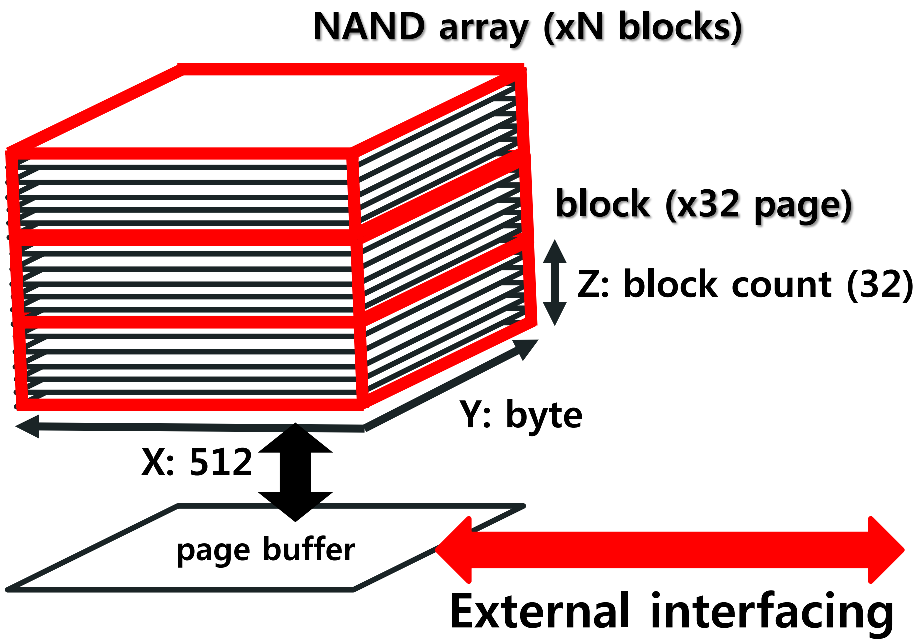
About NAND Flash Memory Structure



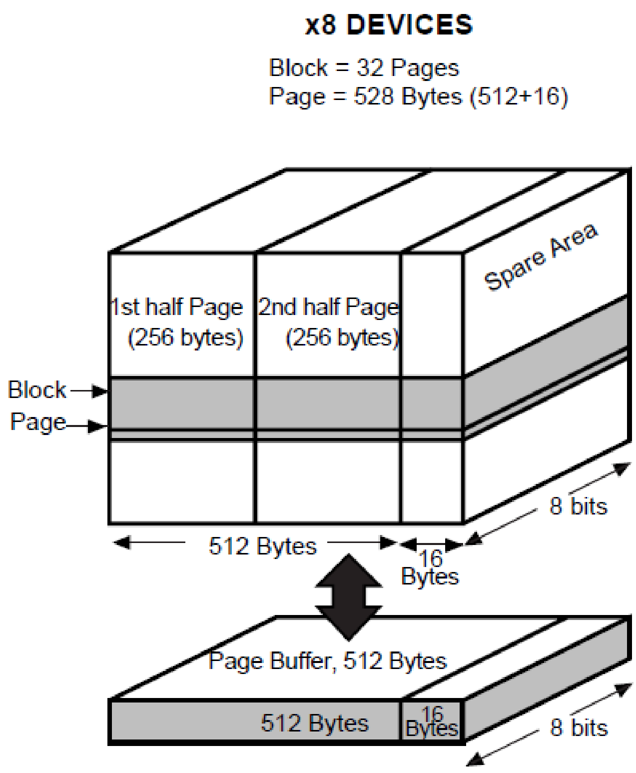
About NAND Flash Memory Structure



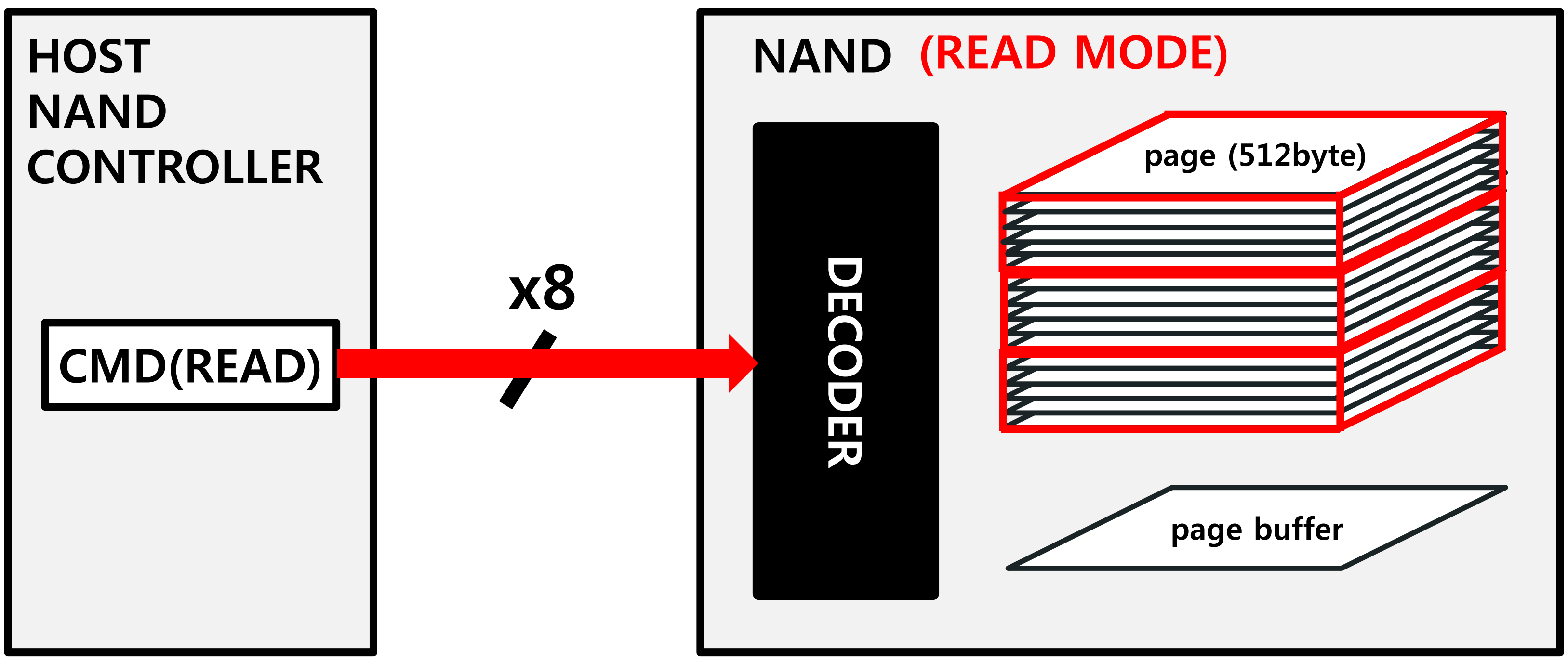
About NAND Flash Memory Structure



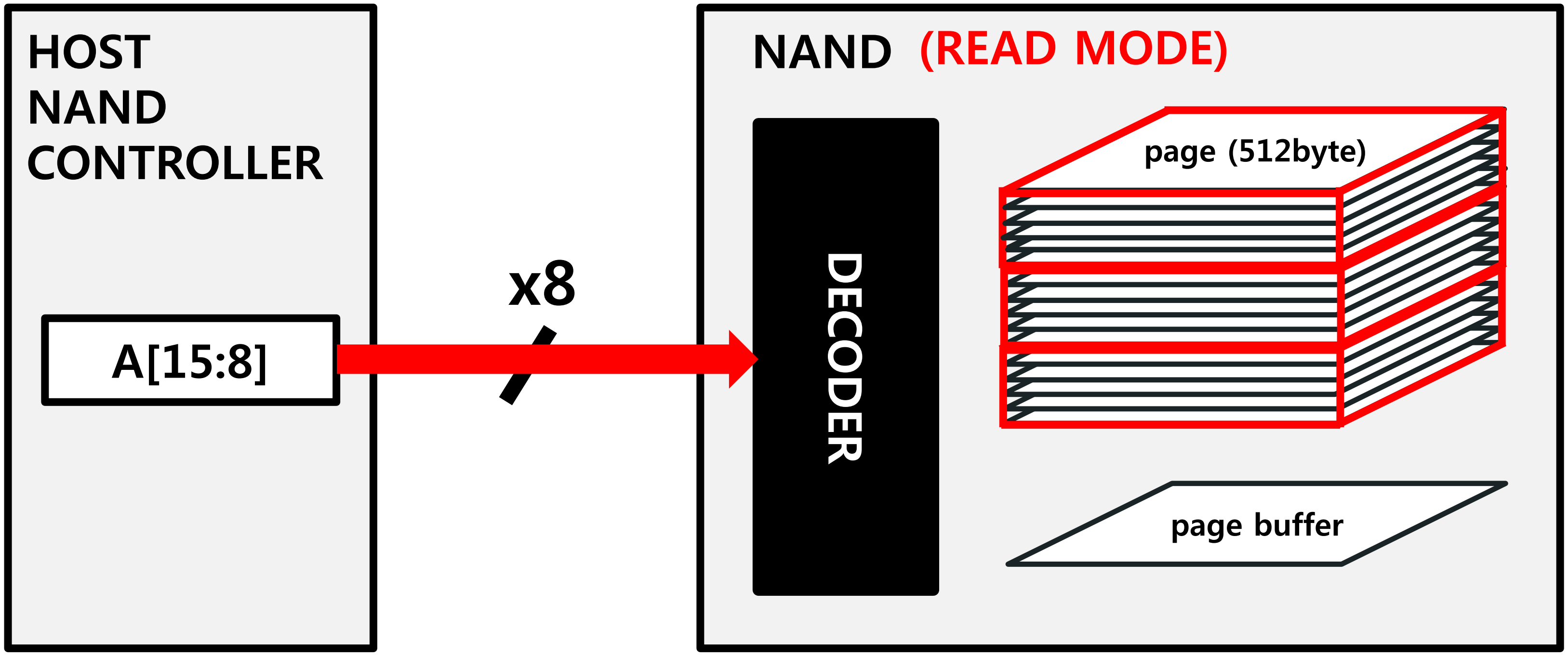
About NAND Flash Memory Structure



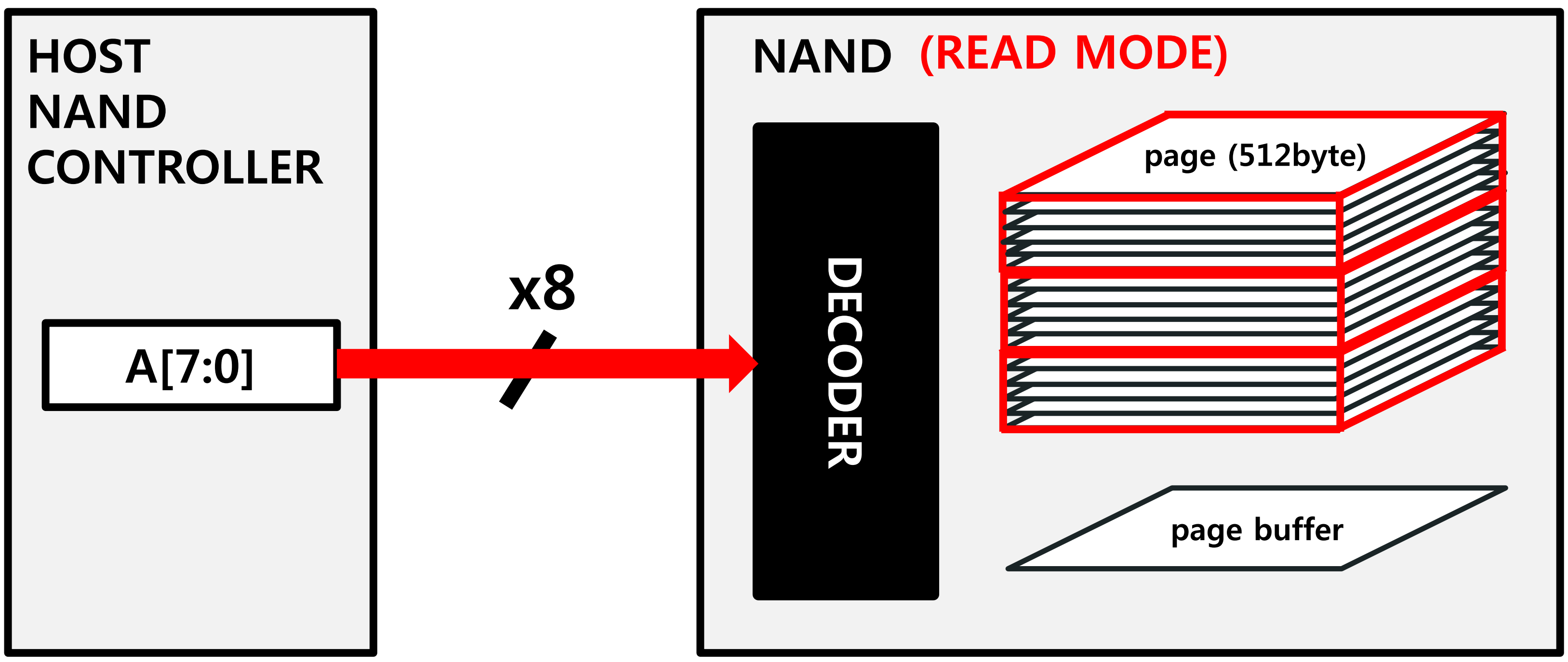
About NAND Flash Interface Protocol



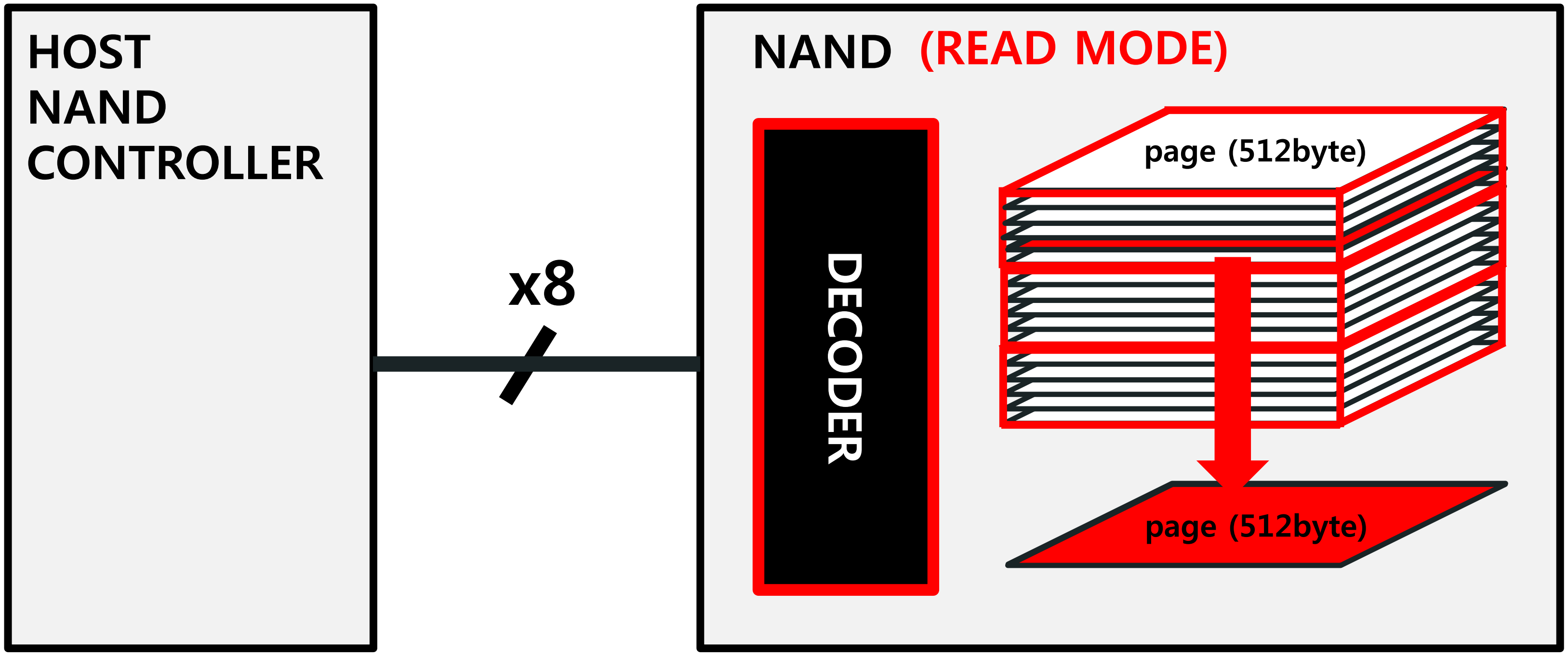
About NAND Flash Interface Protocol



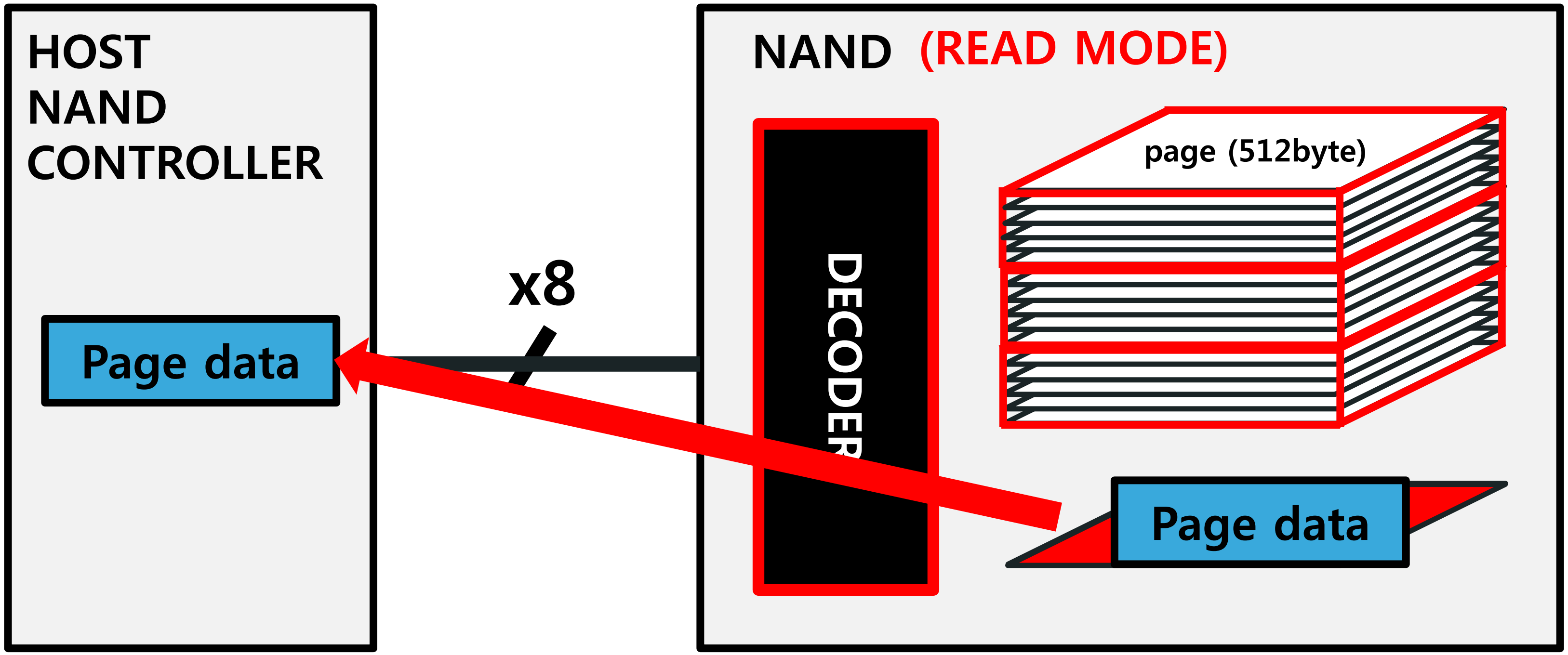
About NAND Flash Interface Protocol



About NAND Flash Interface Protocol

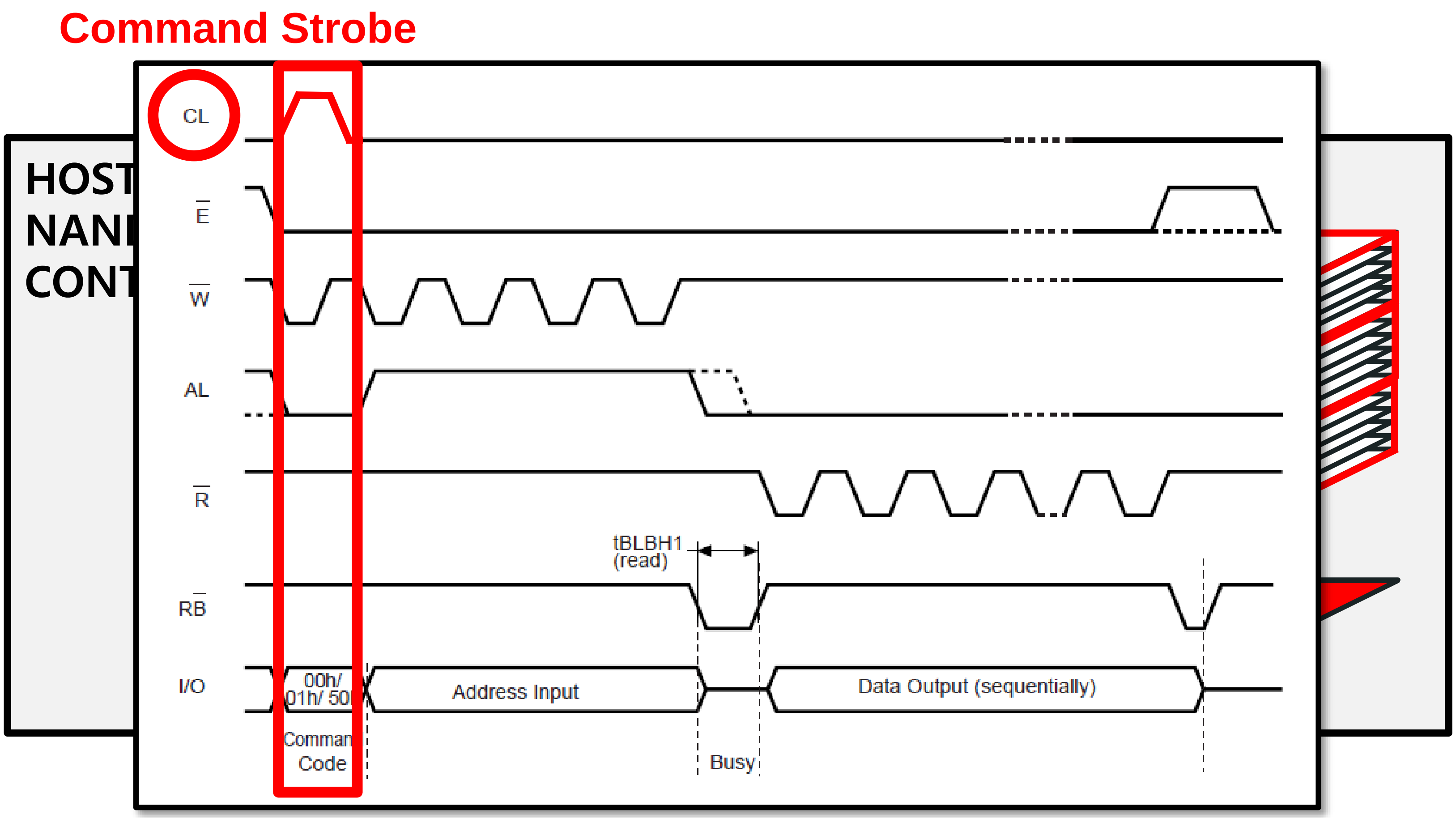


About NAND Flash Interface Protocol



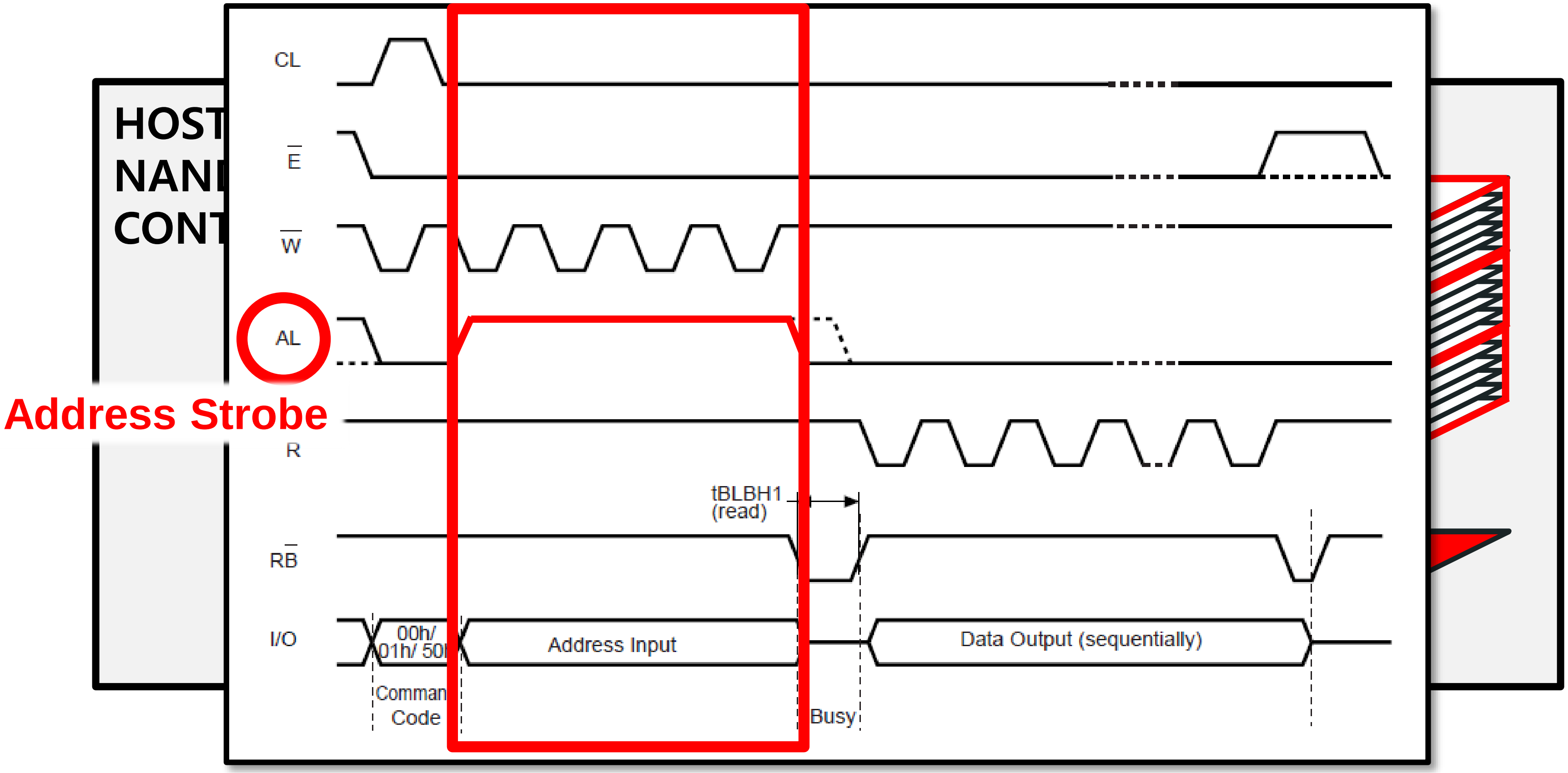
life.augmented

About NAND Flash Interface Protocol

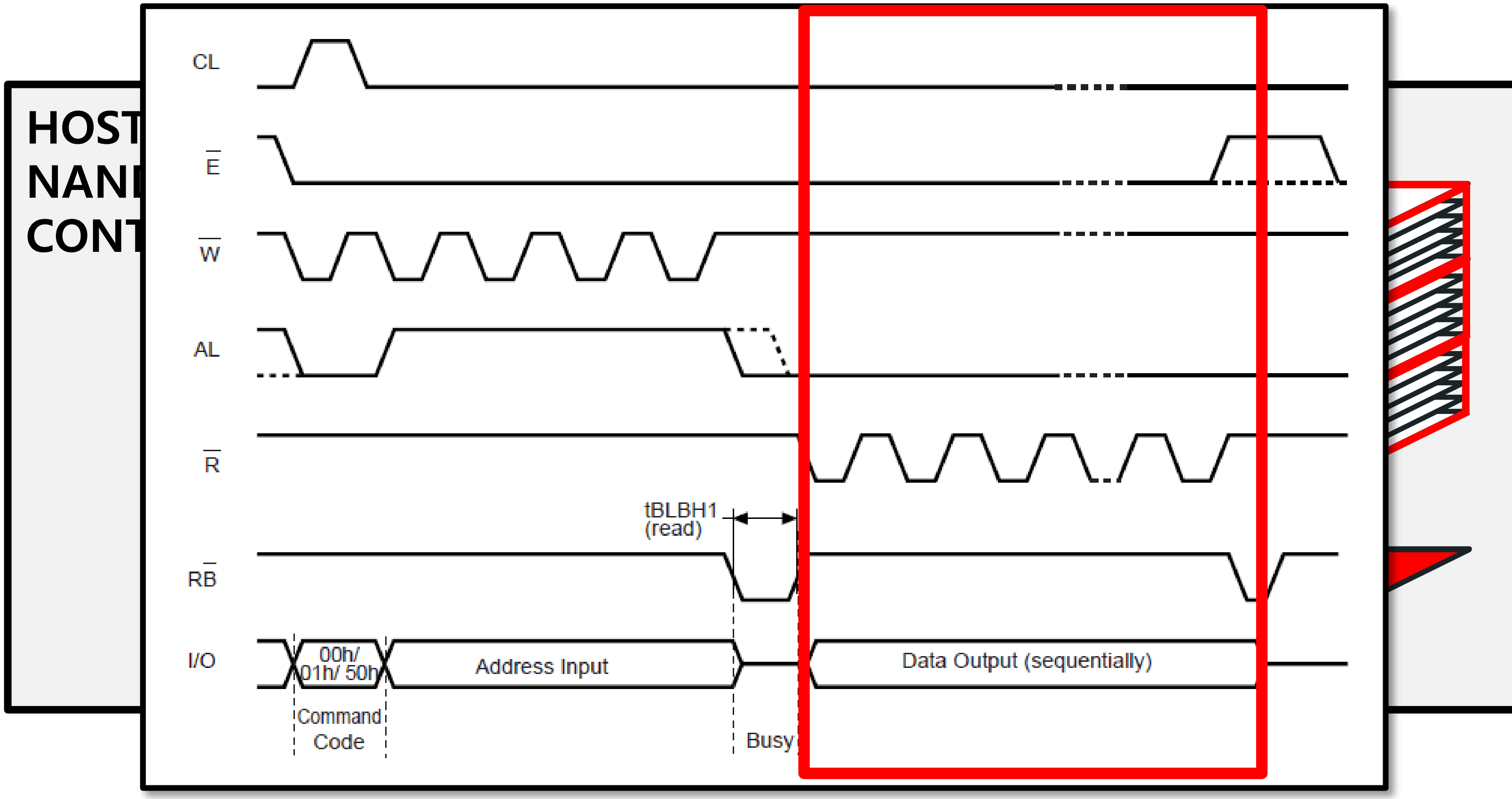


life.augmented

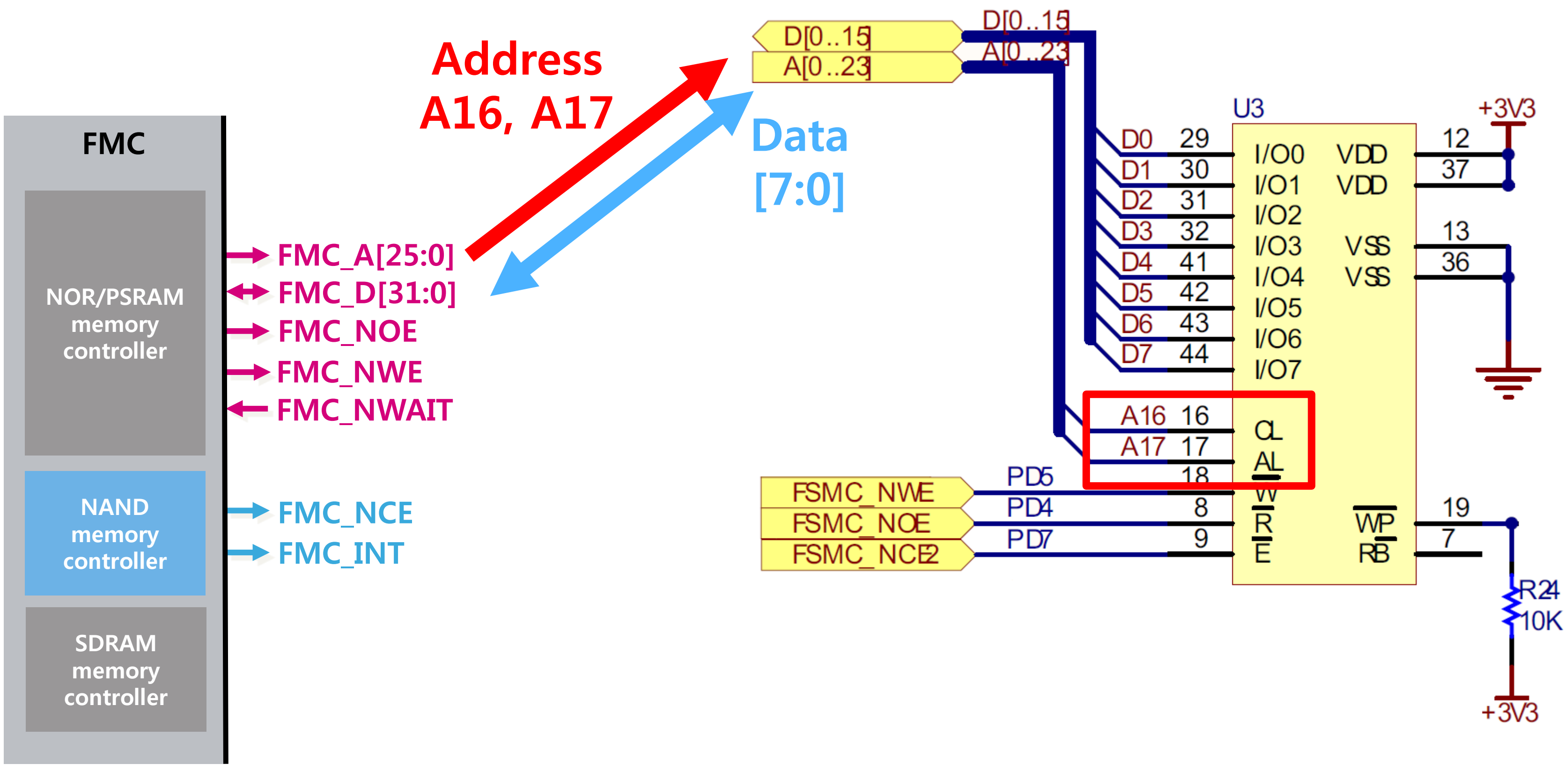
About NAND Flash Interface Protocol



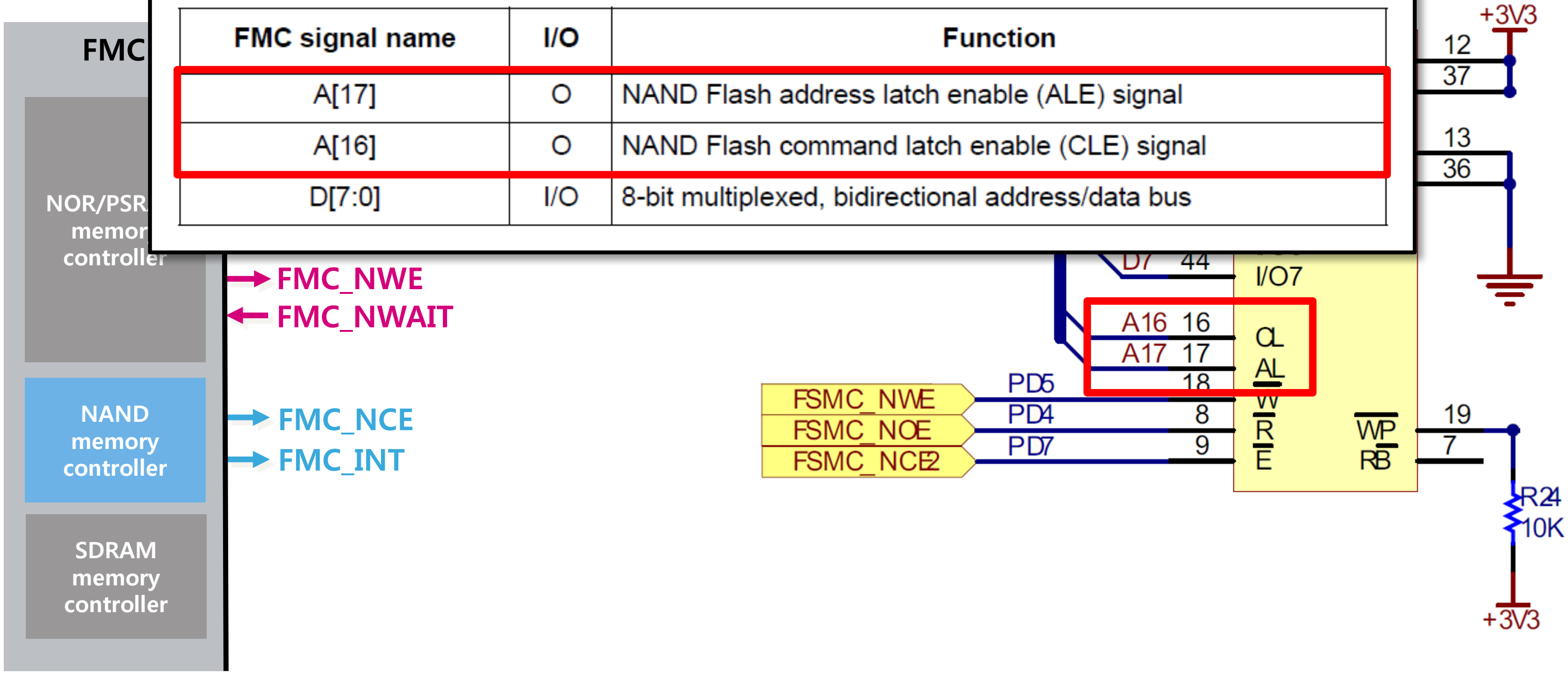
About NAND Flash Interface Protocol



NAND interface signals



NAND interface signals



8-bit NAND Flash memory

Table 75. 8-bit NAND Flash

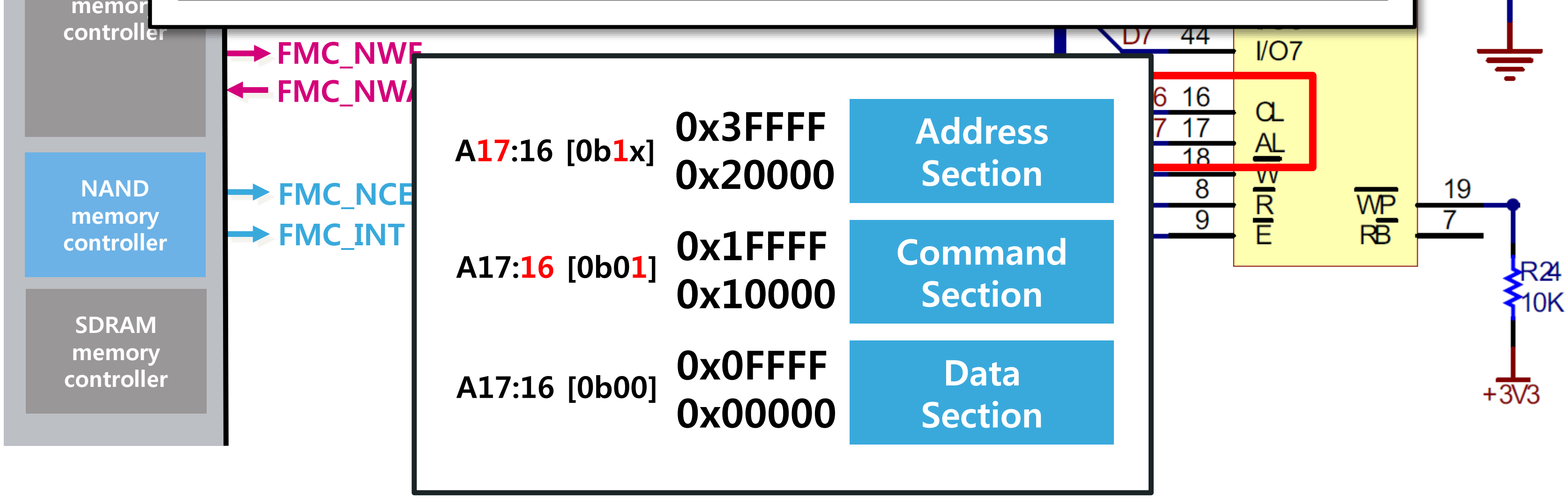
FMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus

NAND interface signals

8-bit NAND Flash memory

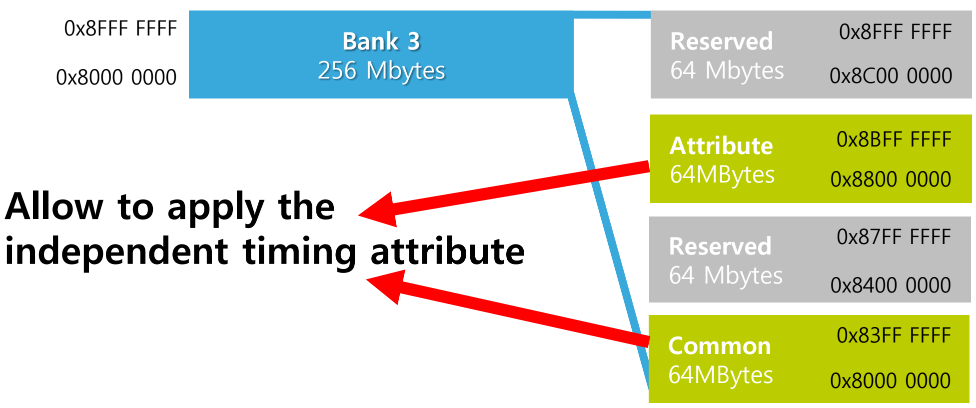
Table 75. 8-bit NAND Flash

FMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus



NAND address mapping

- Bank 3 is used to support NAND Flash memory through two memory spaces
  - Common and Attribute memory space
- Each memory space is divided into 3 subsections
  - Data section (64 Kbytes): Used to read or write data
  - Command section (64 Kbytes): Used to send a command to NAND Flash memory
  - Address section (128 Kbytes): Used to specify the NAND Flash memory address



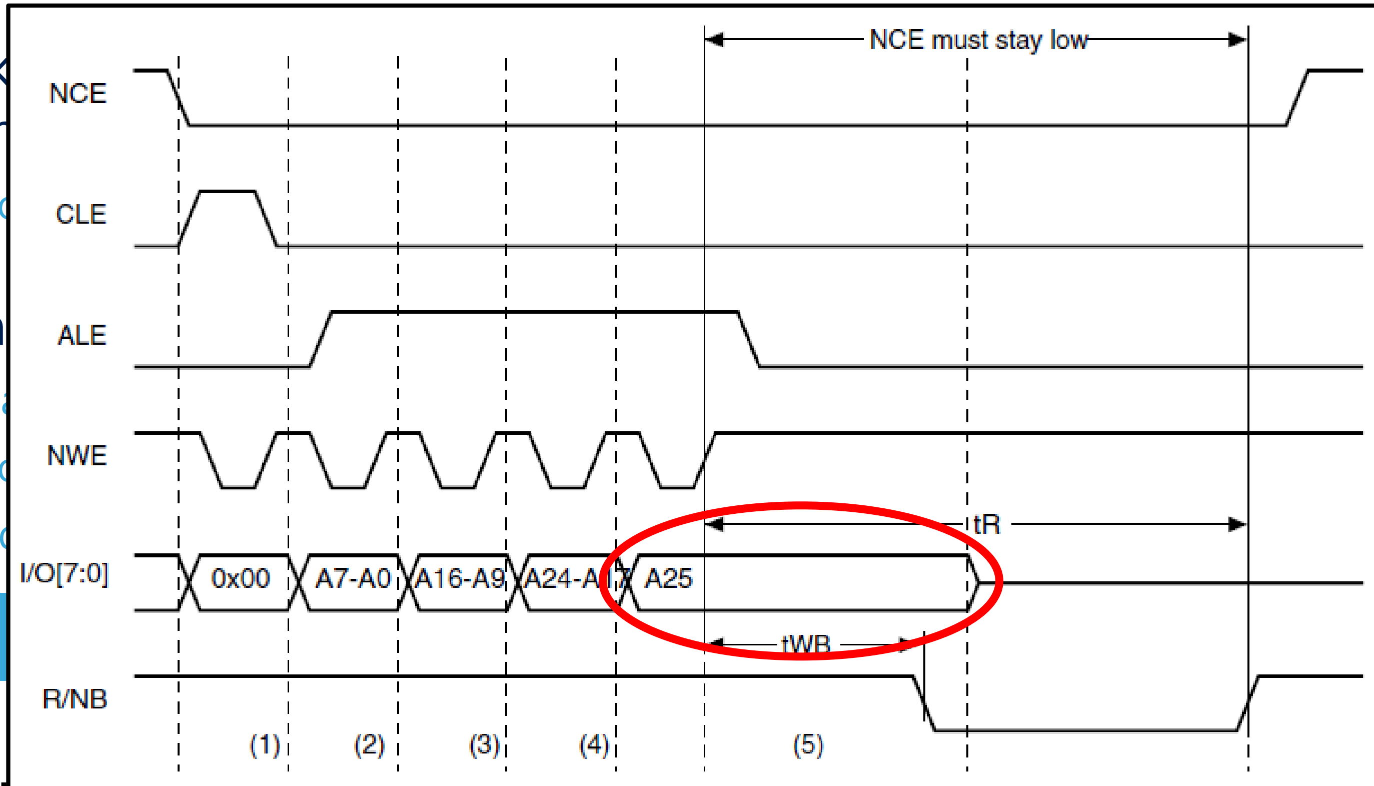
life.augmented

NAND address mapping

- Bank
  mem
  - Co
- Each
  - Da
  - Co … memory
  - Ad … address

0x8FFF FFFF

0x8000 0000



Allow to apply the independent timing attribute

Reserved 64 Mbytes 0x87FF FFFF 0x8400 0000

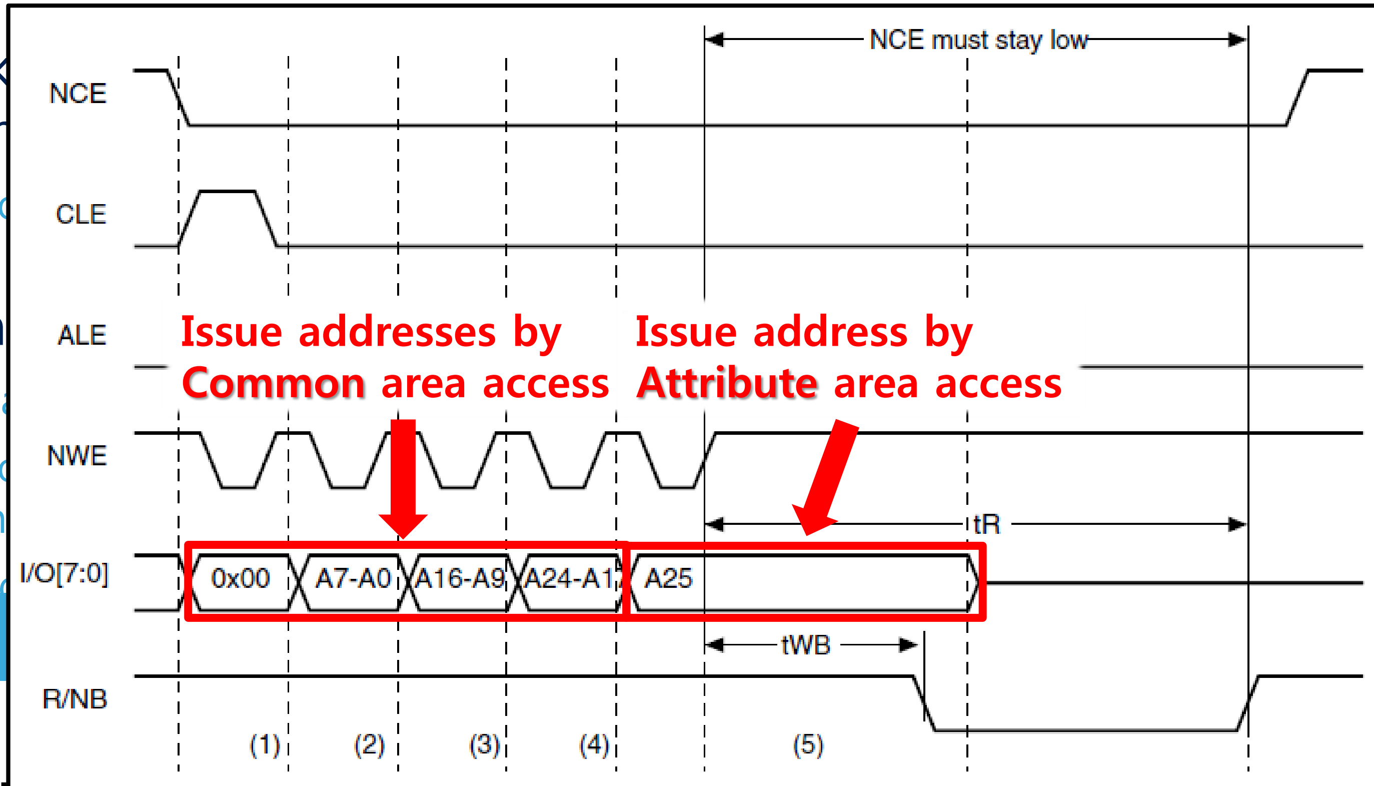
Common 64MBytes 0x83FF FFFF 0x8000 0000

NAND address mapping

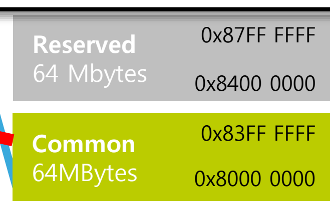
- Bank
  mem
  - Co
- Each
  - Da
  - Co
    m
  - A ... address

0x8FFF FFFF

0x8000 0000



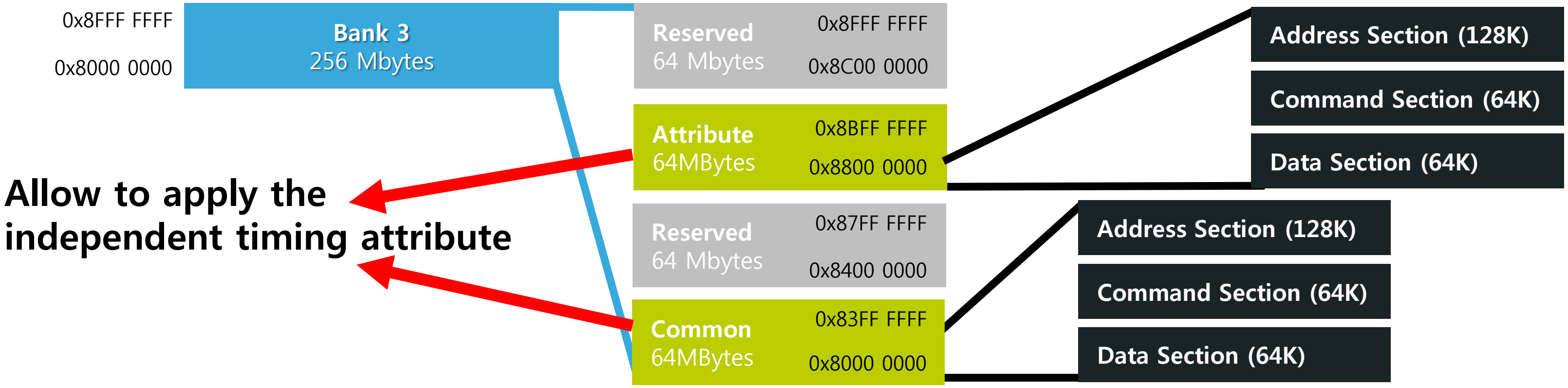
Allow to apply the independent timing attribute



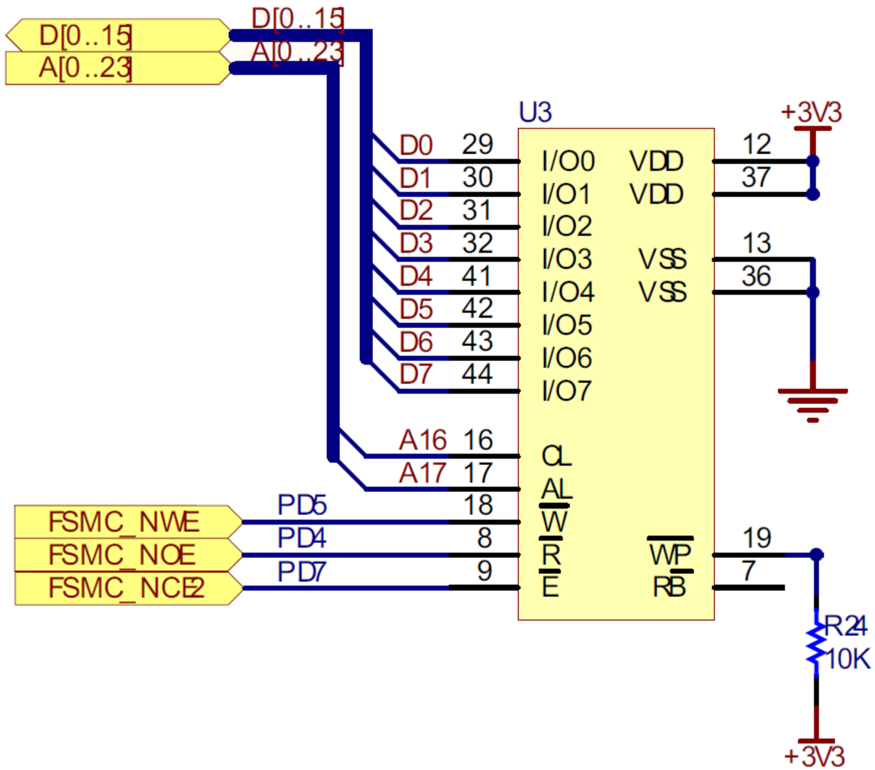
life.augmented

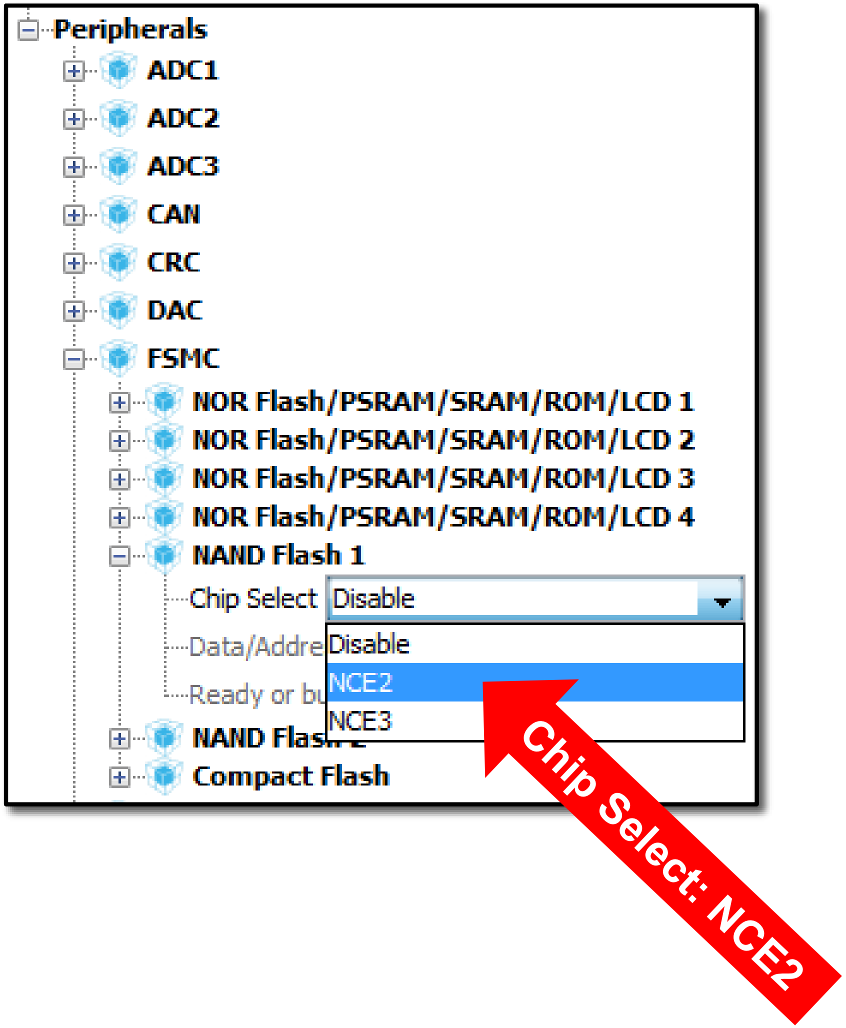
NAND address mapping

- Bank 3 is used to support NAND Flash memory through two memory spaces
  - Common and Attribute memory space
- Each memory space is divided into 3 subsections
  - Data section (64 Kbytes): Used to read or write data
  - Command section (64 Kbytes): Used to send a command to NAND Flash memory
  - Address section (128 Kbytes): Used to specify the NAND Flash memory address

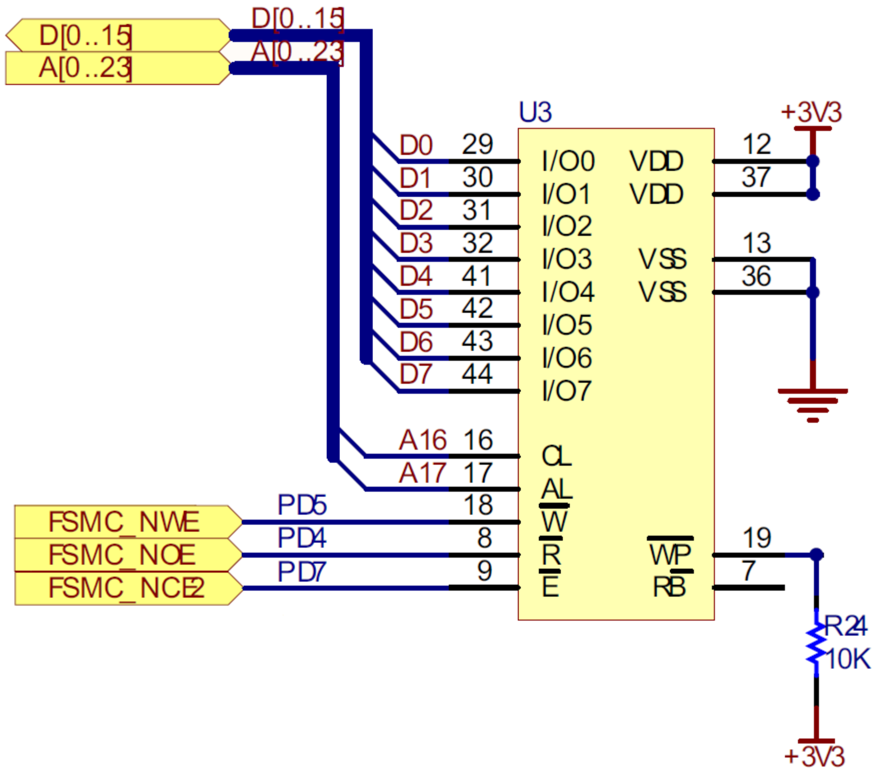


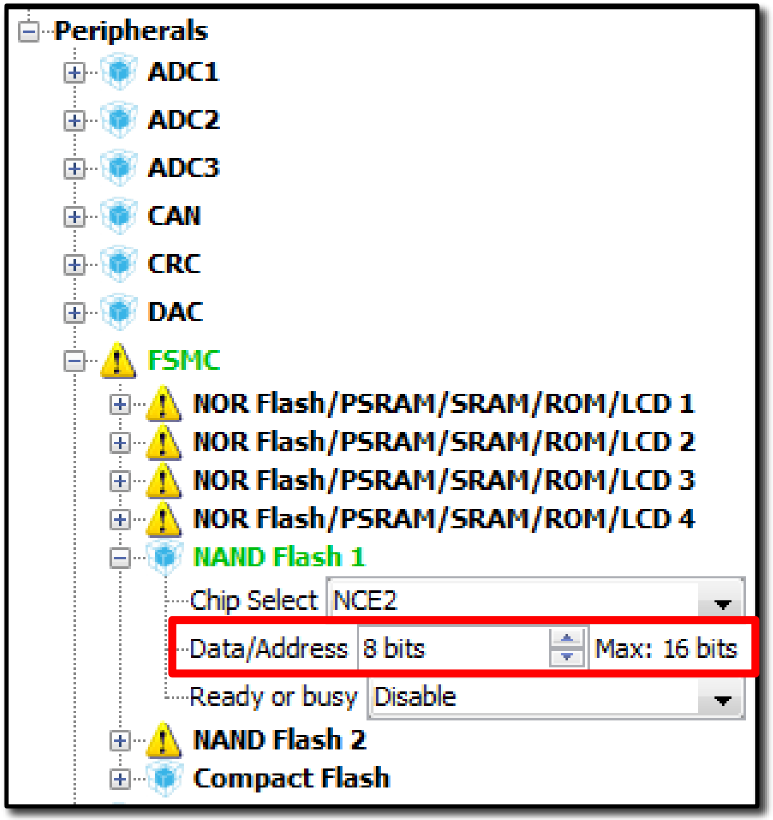
CubeMX Configuration for NAND Connection



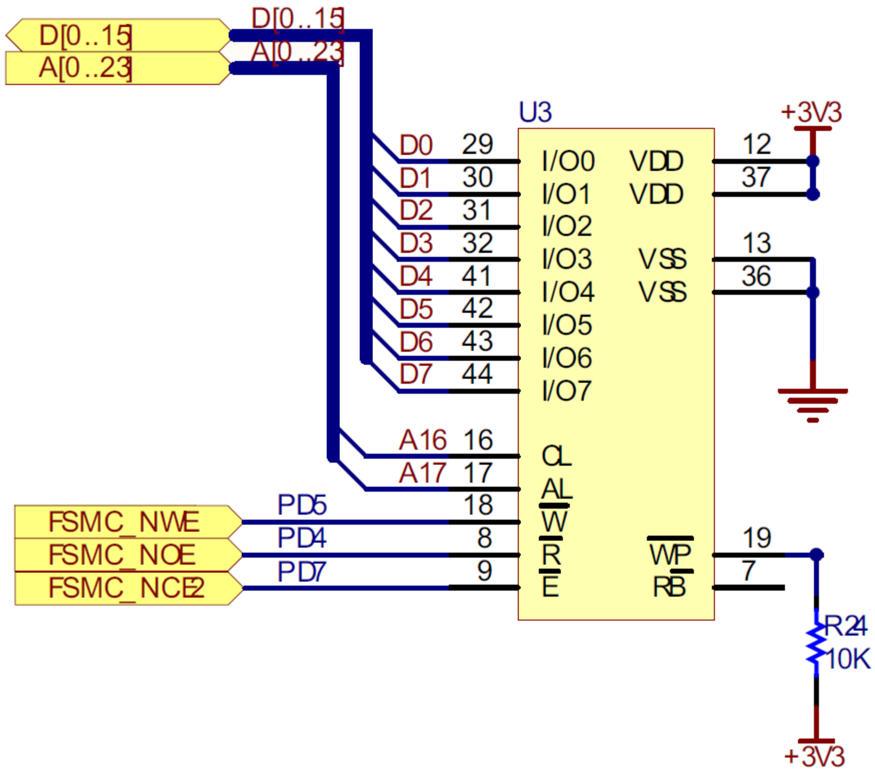


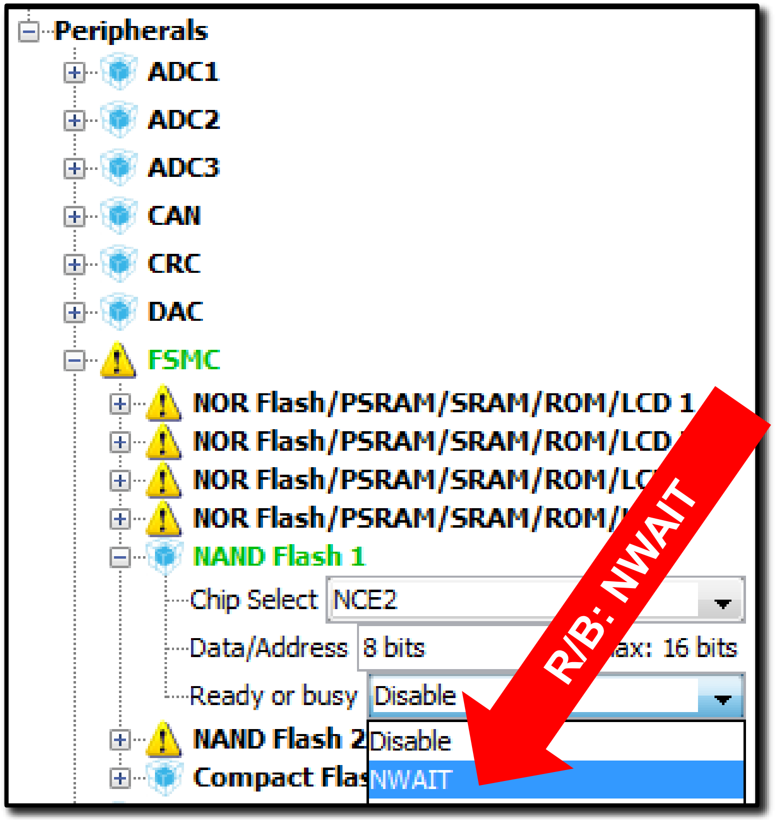
CubeMX Configuration for NAND Connection



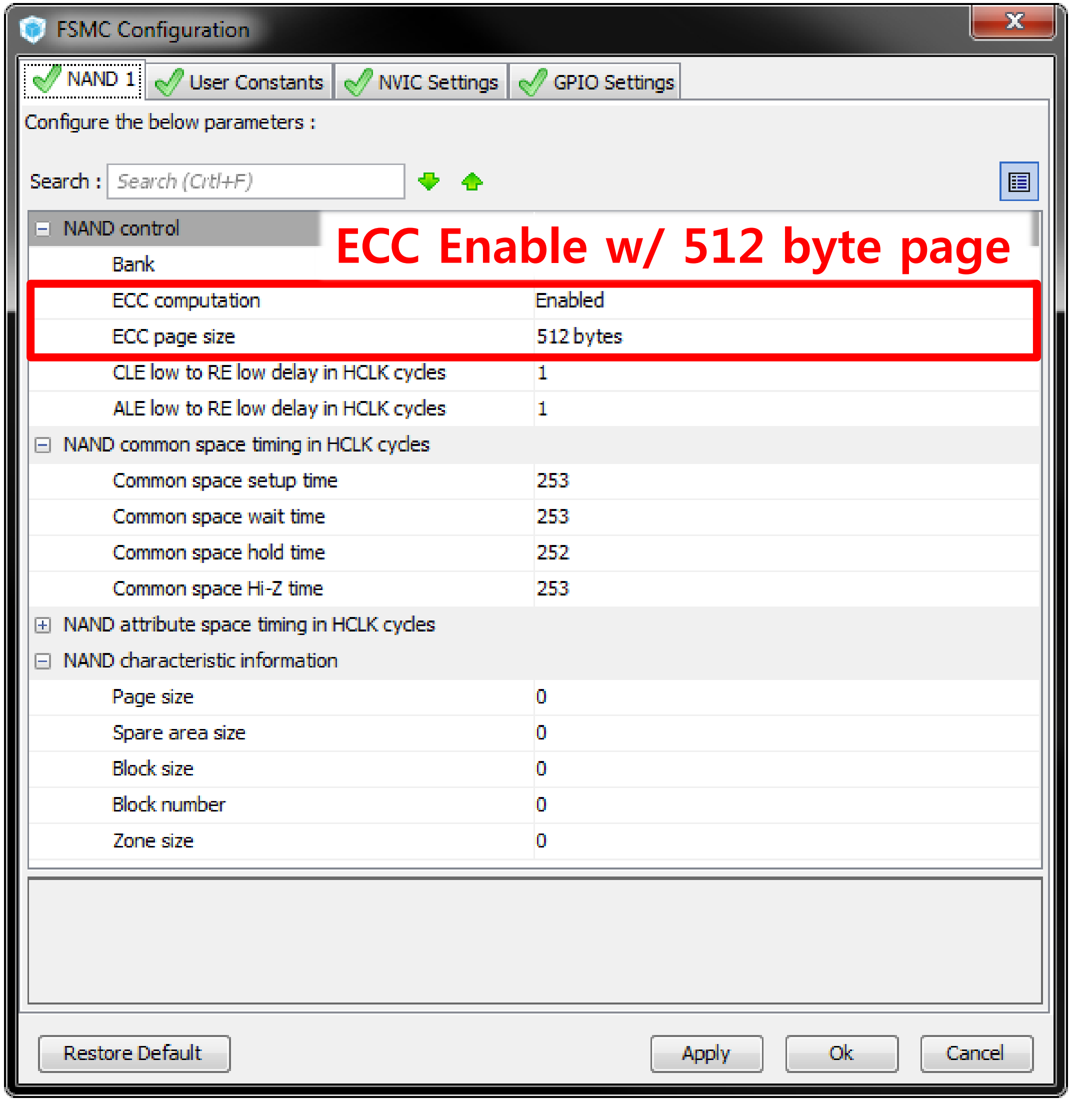


CubeMX Configuration for NAND Connection



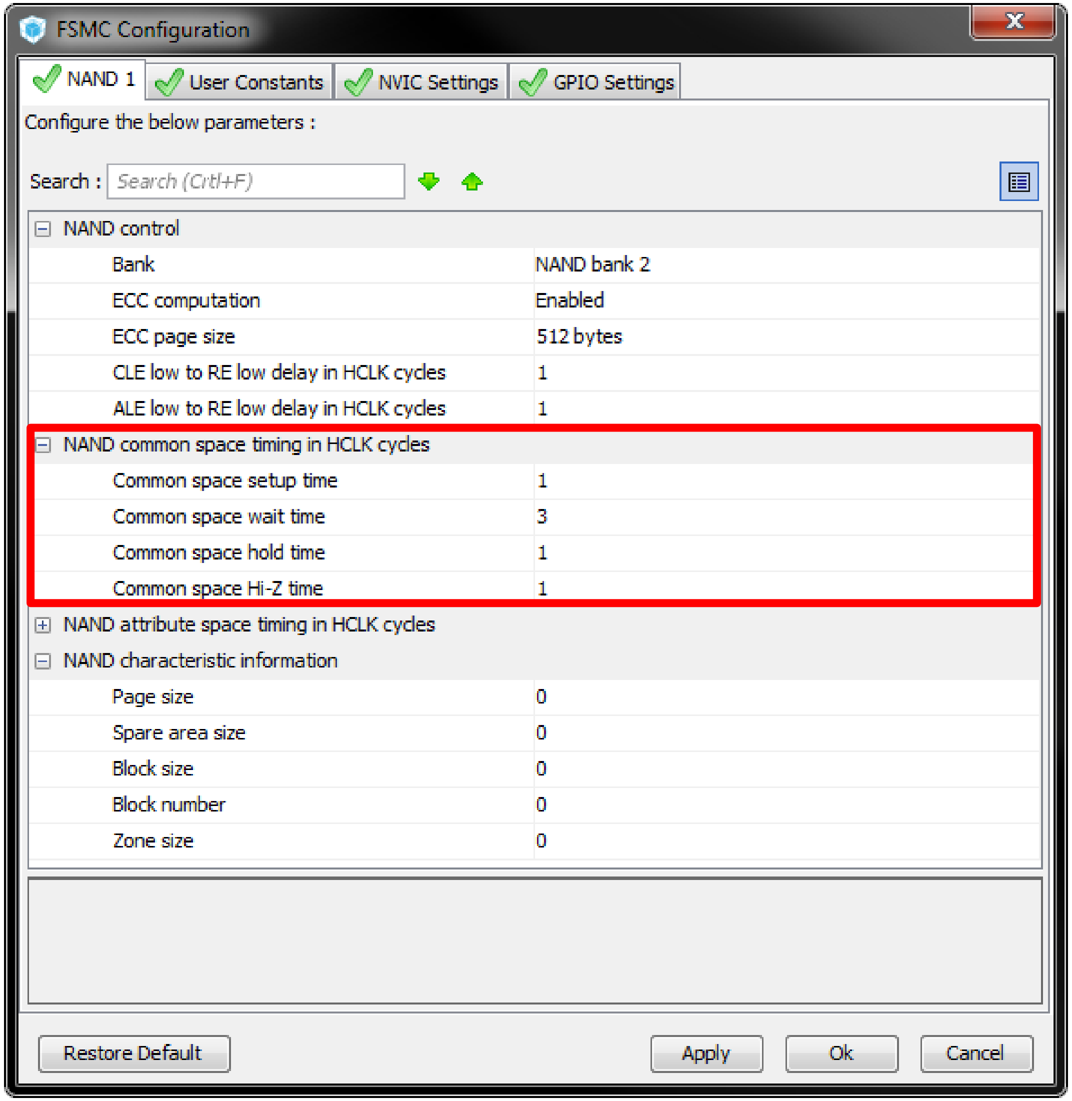


CubeMX FSMC Configuration for NAND

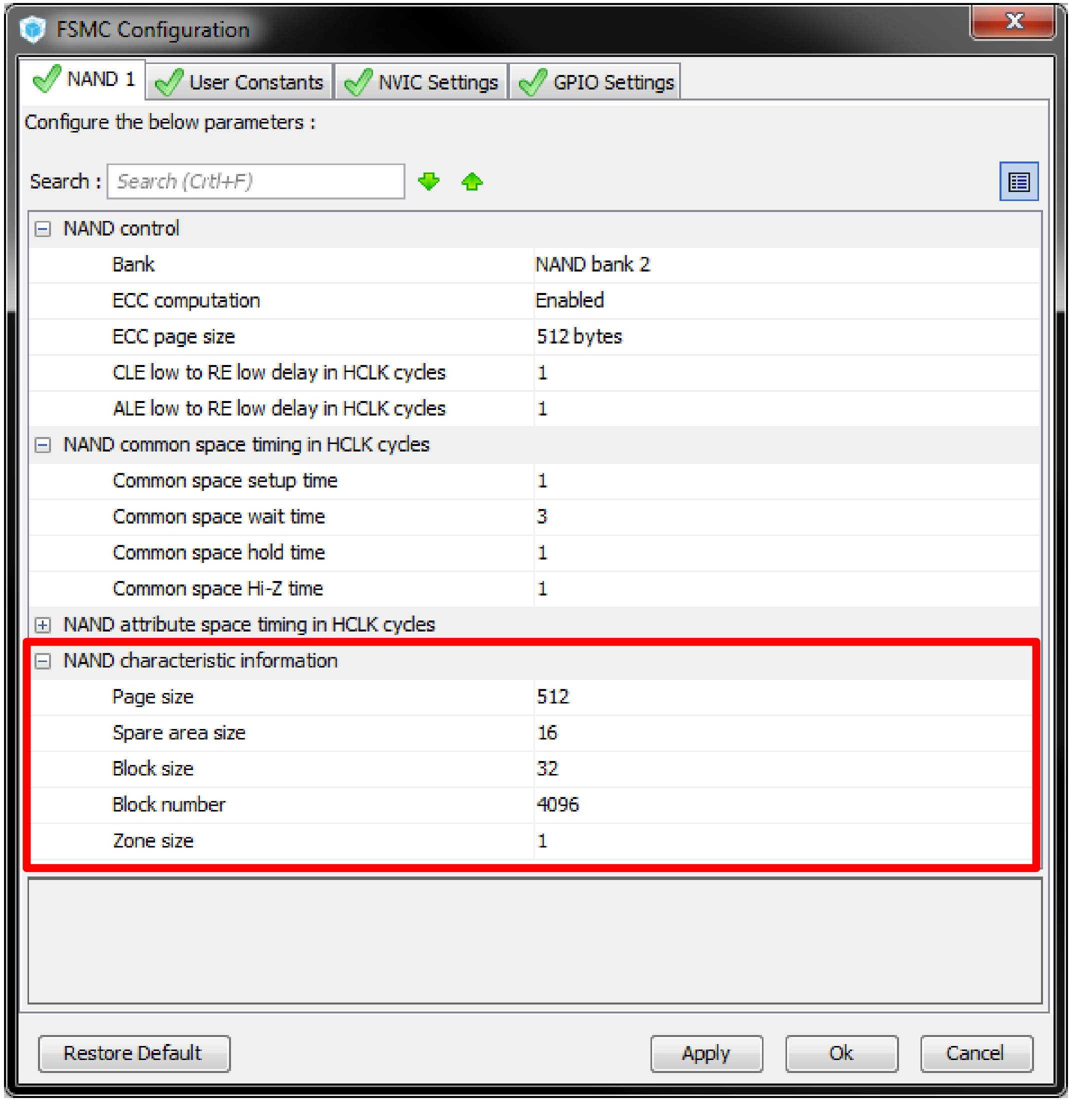


life.augmented

CubeMX FSMC Configuration for NAND



CubeMX FSMC Configuration for NAND



NAND Control Code


```
HAL_StatusTypeDef HAL_NAND_Read_ID(NAND_HandleTypeDef *hnand,
                                   NAND_IDTypeDef *pNAND_ID)
{
  __IO uint32_t data = 0;

  __HAL_LOCK(hnand);

  hnand->State = HAL_NAND_STATE_BUSY;

  /* Send Read ID command sequence */
  *(__IO uint8_t *)((uint32_t)(NAND_DEVICE | CMD_AREA))  = NAND_CMD_READID;
  *(__IO uint8_t *)((uint32_t)(NAND_DEVICE | ADDR_AREA)) = 0x00;

  /* Read the electronic signature from NAND flash */
  data = *(__IO uint32_t *)NAND_DEVICE;

  /* Return the data read */
  pNAND_ID->Maker_Id    = ADDR_1st_CYCLE(data);
  pNAND_ID->Device_Id   = ADDR_2nd_CYCLE(data);
  pNAND_ID->Third_Id    = ADDR_3rd_CYCLE(data);
  pNAND_ID->Fourth_Id   = ADDR_4th_CYCLE(data);

  /* Update the NAND controller state */
  hnand->State = HAL_NAND_STATE_READY;

  __HAL_UNLOCK(hnand);

  return HAL_OK;
}
```

NAND Control Code

```
{
  __HAL_UNLOCK(hnand);

  return HAL_OK;
}
```

Table 9. Commands

Command	Bus Write Operations[1]		
	1st CYCLE	2nd CYCLE	3rd CYCLE
Read A	00h	-	-
Read B	01h[2]	-	-
Read C	50h	-	-
Read Electronic Signature	90h	-	-
Read Status Register	70h	-	-
Page Program	80h	10h	-
Copy Back Program	00h	8Ah	10h
Block Erase	60h	D0h	-
Reset	FFh	-	-

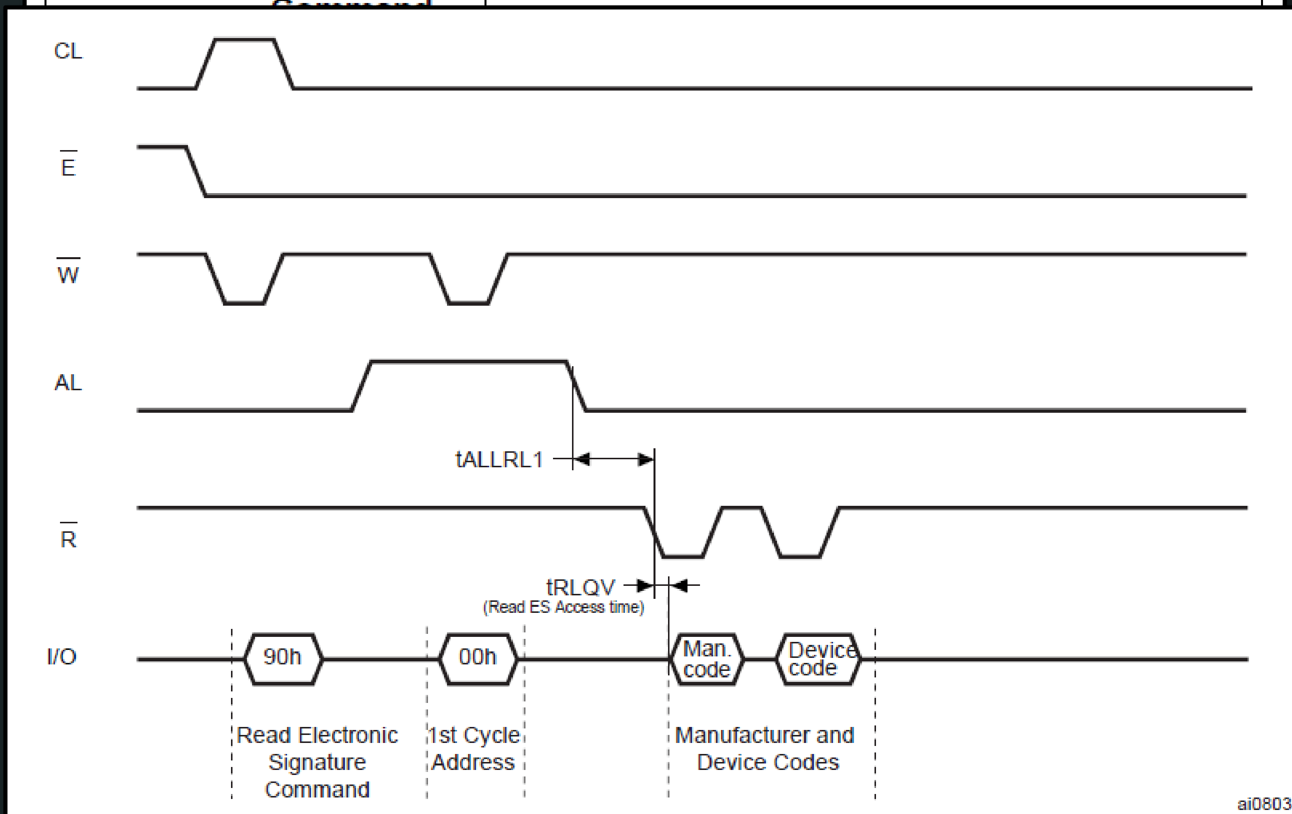
life.augmented

NAND Control Code

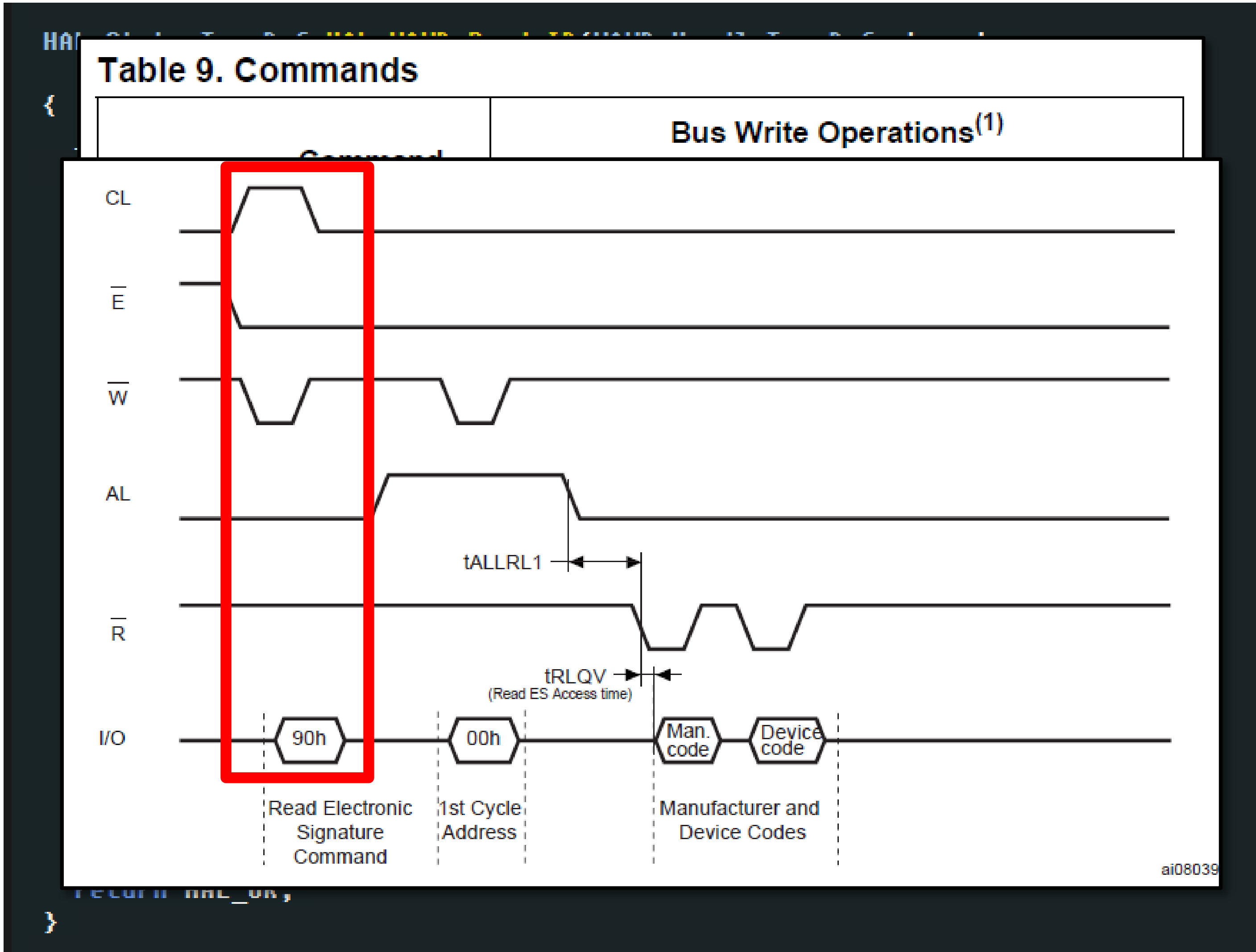
```
HA
{
return HAL_OK;
}
```

Table 9. Commands

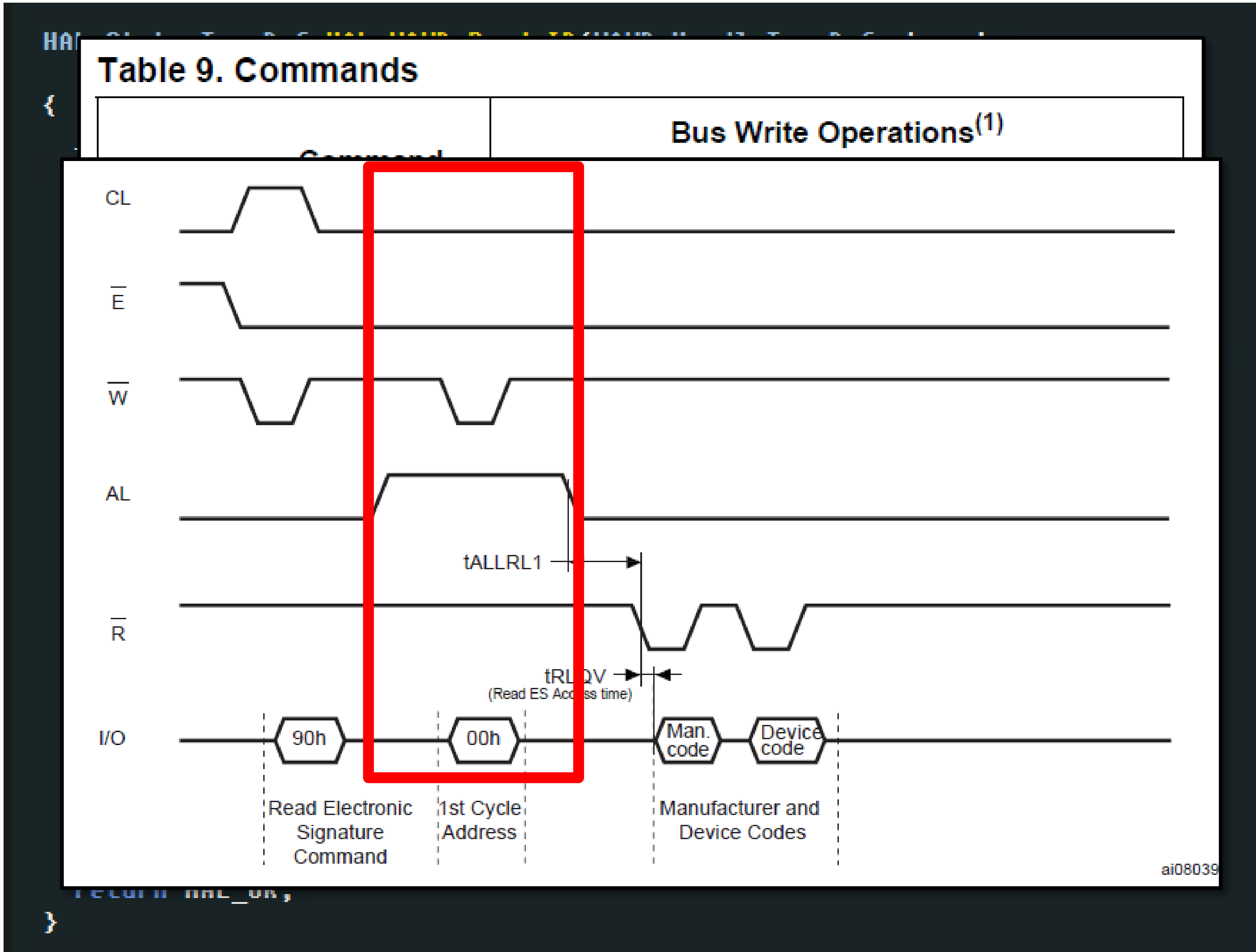
Command	Bus Write Operations(1)



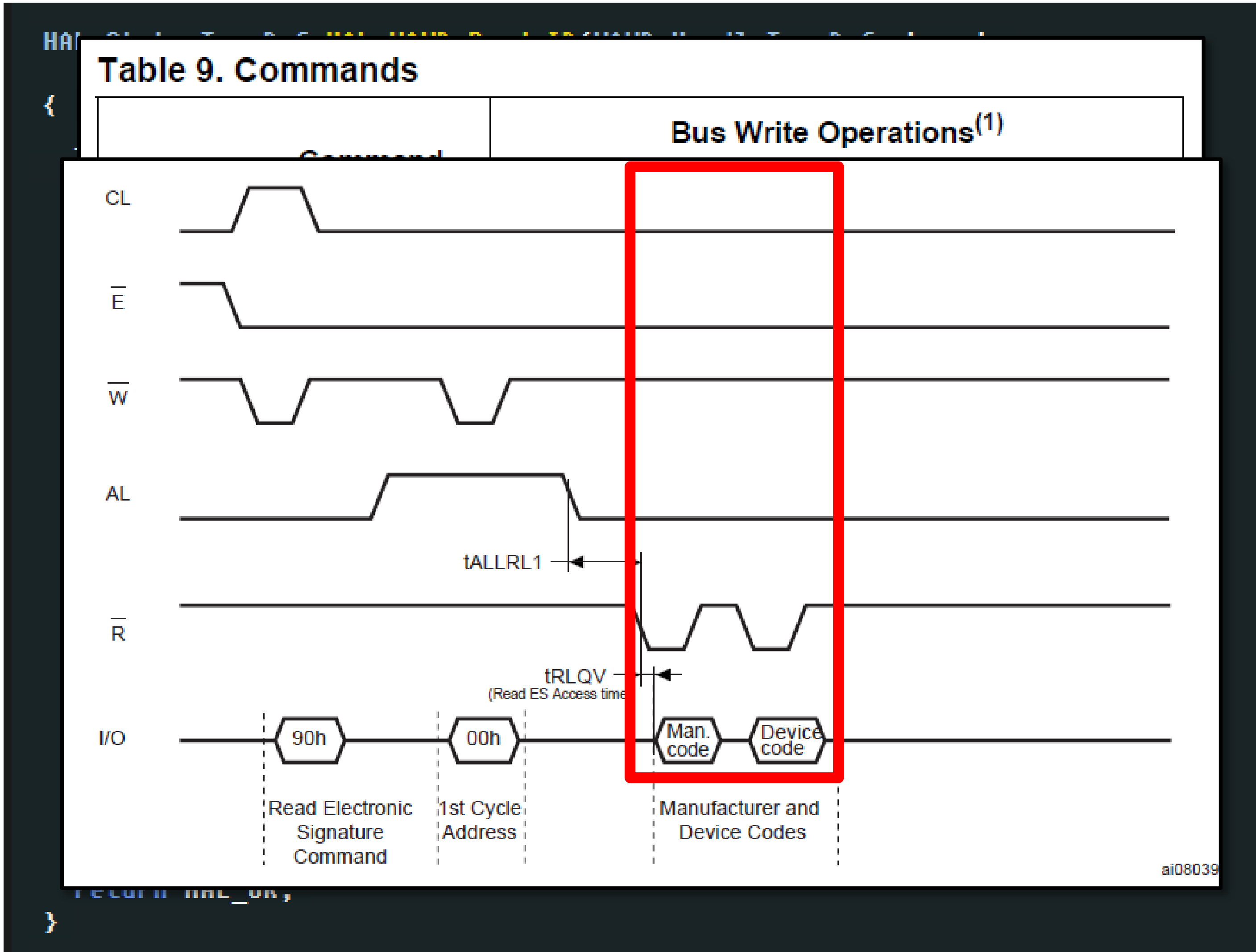
NAND Control Code



NAND Control Code



NAND Control Code



NAND Control Code

```
HAL_StatusTypeDef HAL_NAND_Read_ID(NAND_HandleTypeDef *hnand,
                                   NAND_IDTypeDef *pNAND_ID)
{
  __IO uint32_t data = 0;

  __HAL_LOCK(hnand);

  hnand->State = HAL_NAND_STATE_BUSY;

  /* Send Read ID command sequence */
  *(__IO uint8_t *)((uint32_t)(NAND_DEVICE | CMD_AREA))  = NAND_CMD_READID;
  *(__IO uint8_t *)((uint32_t)(NAND_DEVICE | ADDR_AREA)) = 0x00;

  /* Read the electronic signature from NAND flash */
  data = *(__IO uint32_t *)NAND_DEVICE;

  /* Return the data read */
  pNAND_ID->Maker_Id    = ADDR_1st_CYCLE(data);
  pNAND_ID->Device_Id   = ADDR_2nd_CYCLE(data);
  pNAND_ID->Third_Id    = ADDR_3rd_CYCLE(data);
  pNAND_ID->Fourth_Id   = ADDR_4th_CYCLE(data);

  /* Update the NAND controller state */
  hnand->State = HAL_NAND_STATE_READY;

  __HAL_UNLOCK(hnand);

  return HAL_OK;
}
```

Write to Command Section
(Issue CLE(A16))

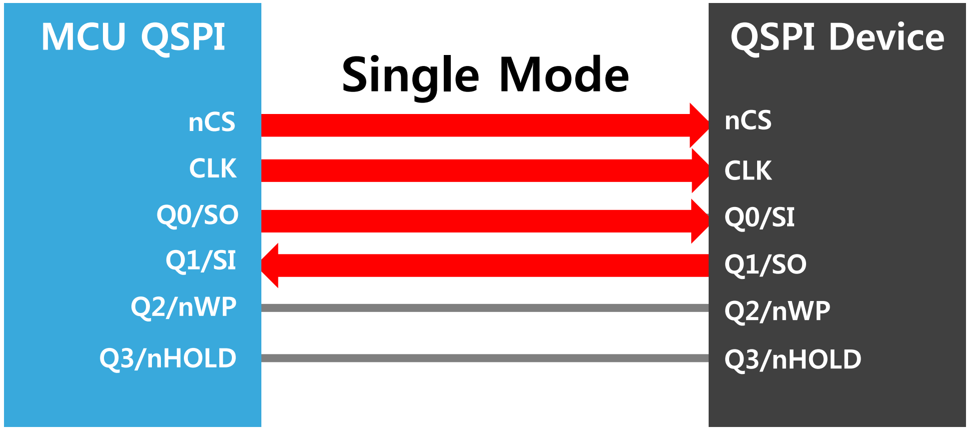
Write to Address Section
(Issue ALE(A17))

1xRead word transaction will be divided to sub-transaction (4xRead byte) by FMC

life.augmented

Quad-SPI Overview

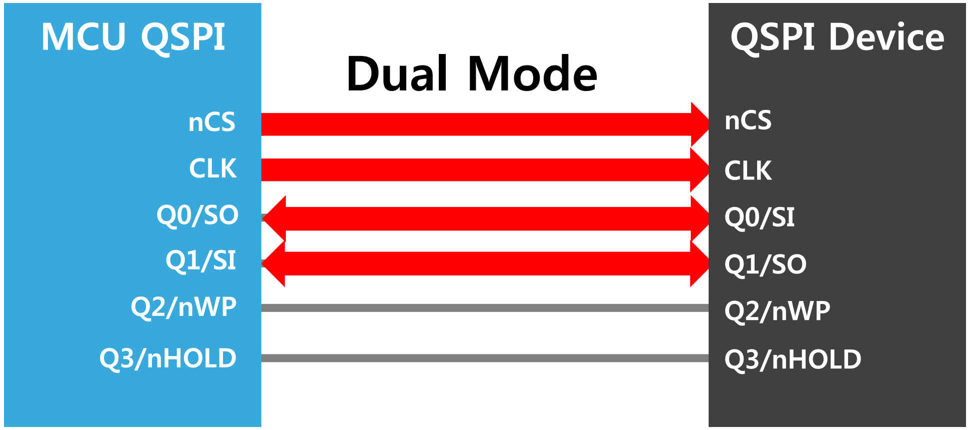
- Three operating modes
  - **Indirect**: all the operations are performed through registers (classical SPI)
  - **Status polling**: periodical read of the flash status registers (interrupt generation)
  - **Memory mapped**: External flash seen as internal for read operations (XIP)
- Communication interface for single/dual/quad SPI flash memories



Quad-SPI Overview

- Three operating modes
  - **Indirect**: all the operations are performed through registers (classical SPI)
  - **Status polling**: periodical read of the flash status registers (interrupt generation)
  - **Memory mapped**: External flash seen as internal for read operations (XIP)

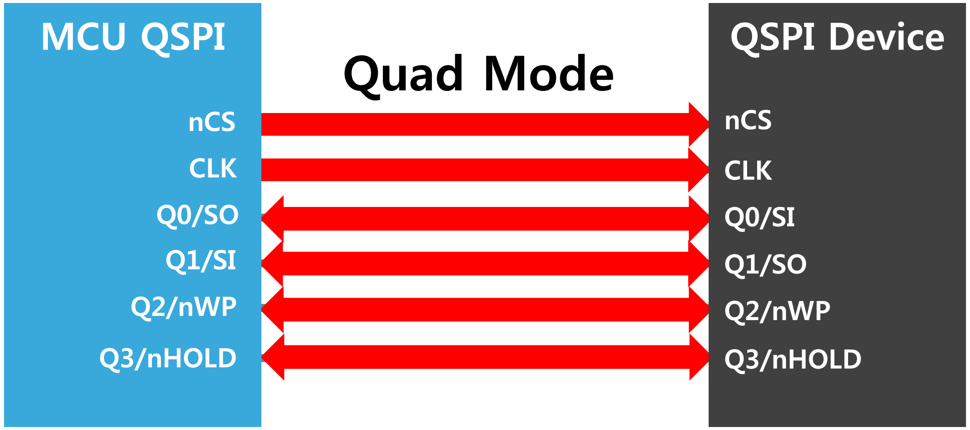
- Communication interface for single/dual/quad SPI flash memories



Quad-SPI Overview

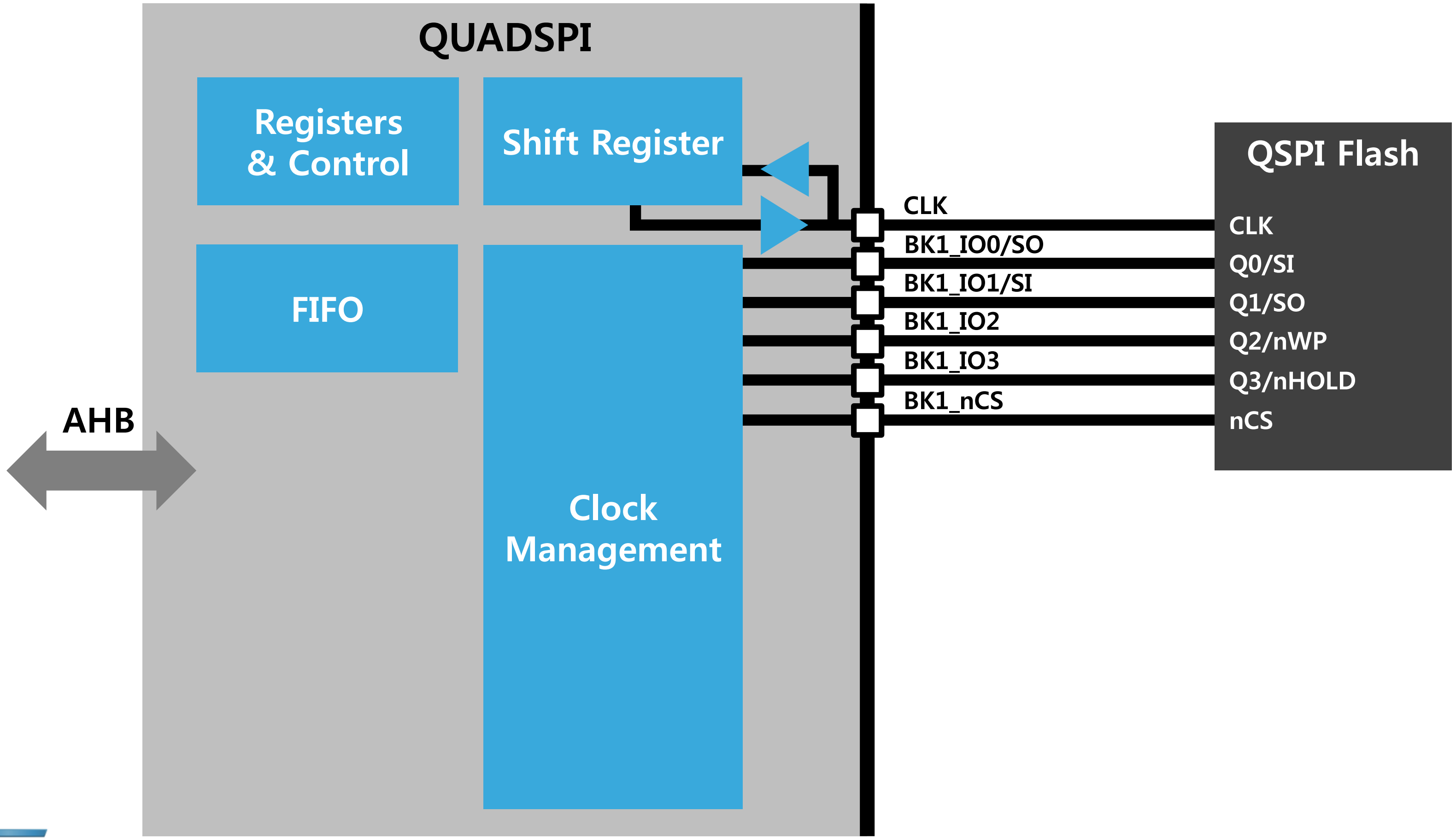
- Three operating modes
  - **Indirect**: all the operations are performed through registers (classical SPI)
  - **Status polling**: periodical read of the flash status registers (interrupt generation)
  - **Memory mapped**: External flash seen as internal for read operations (XIP)

- Communication interface for single/dual/quad SPI flash memories



Quad-SPI Overview

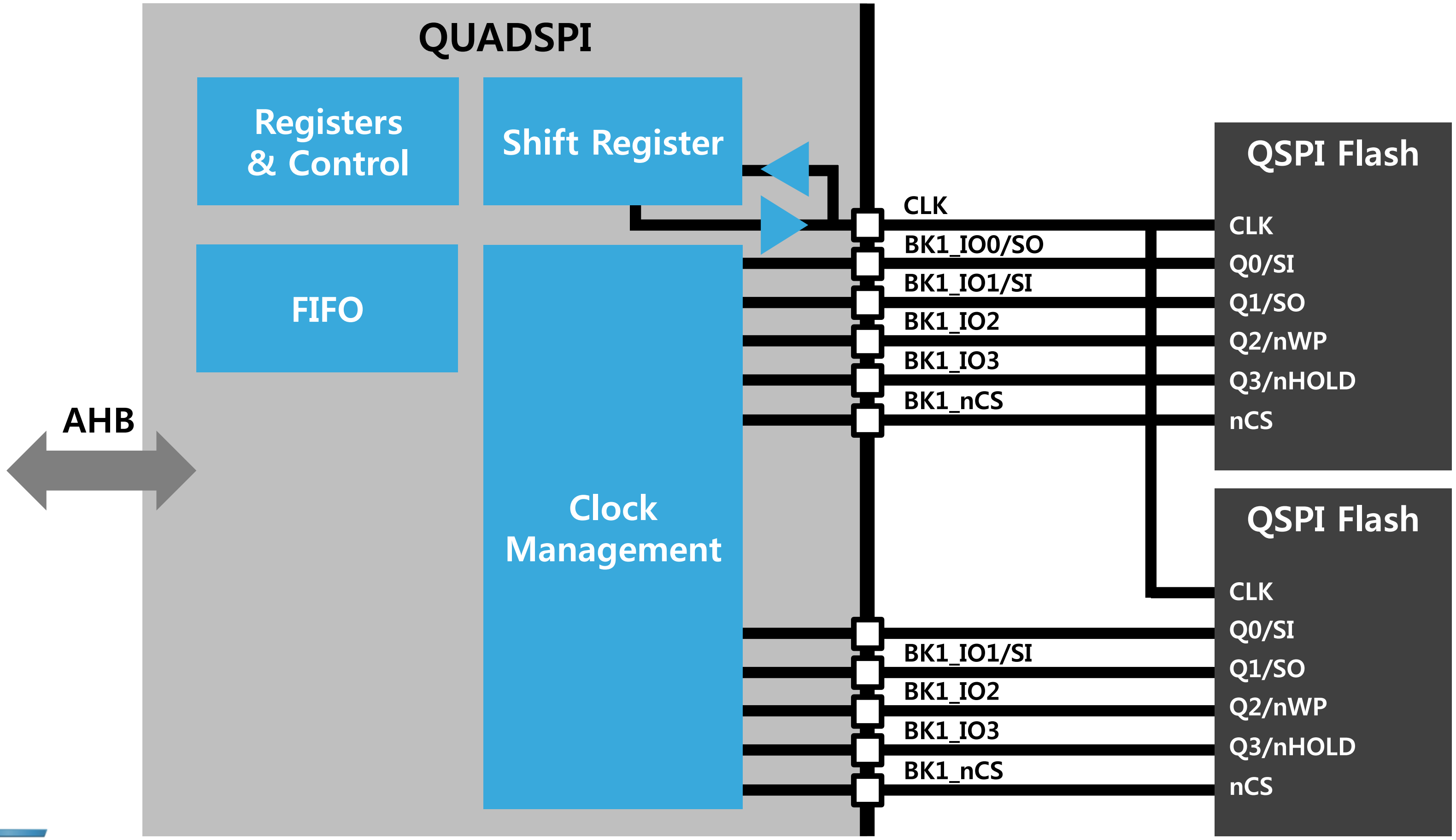
- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



life.augmented

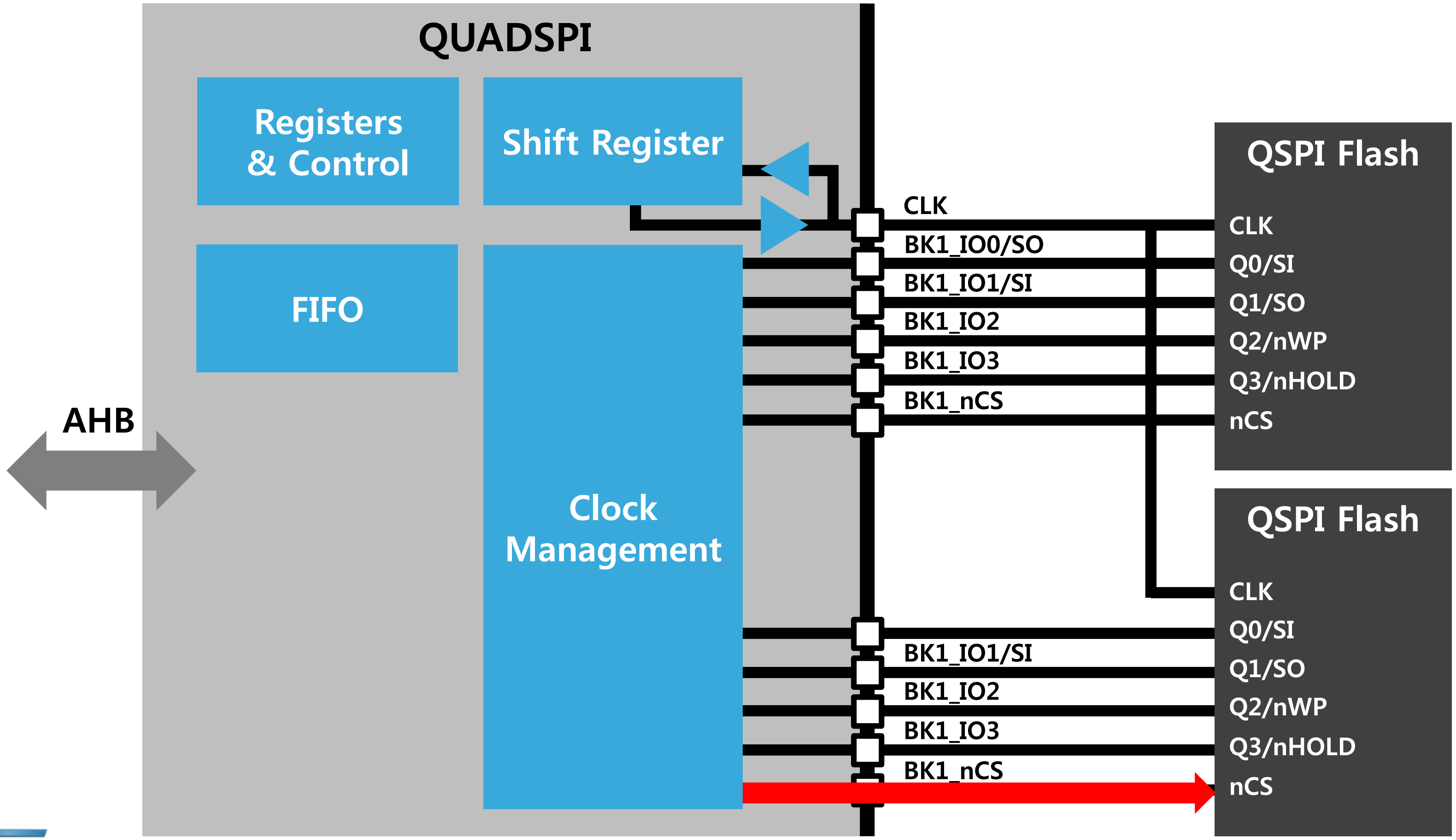
Quad-SPI Overview

- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



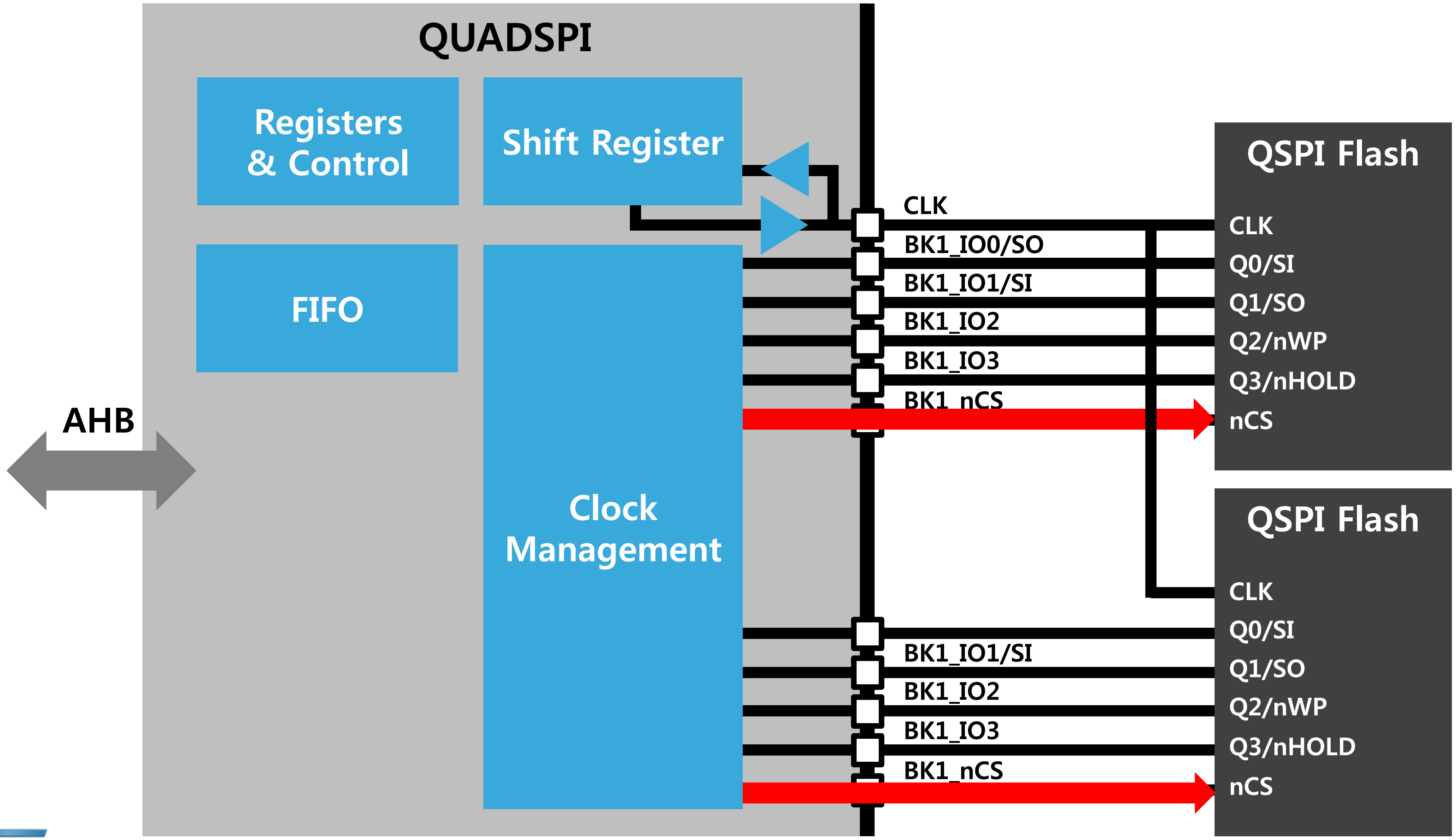
Quad-SPI Overview

- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



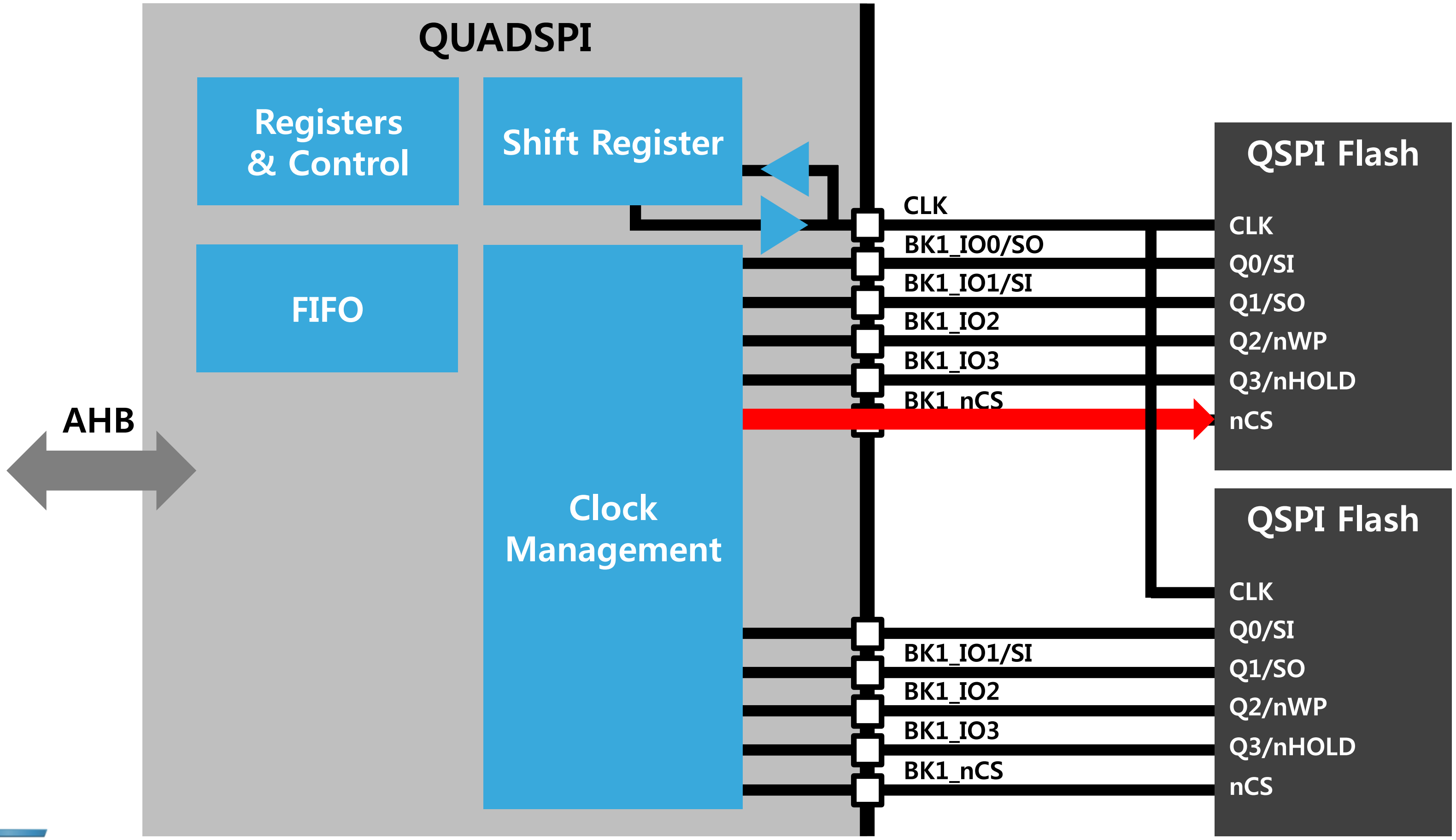
Quad-SPI Overview

- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



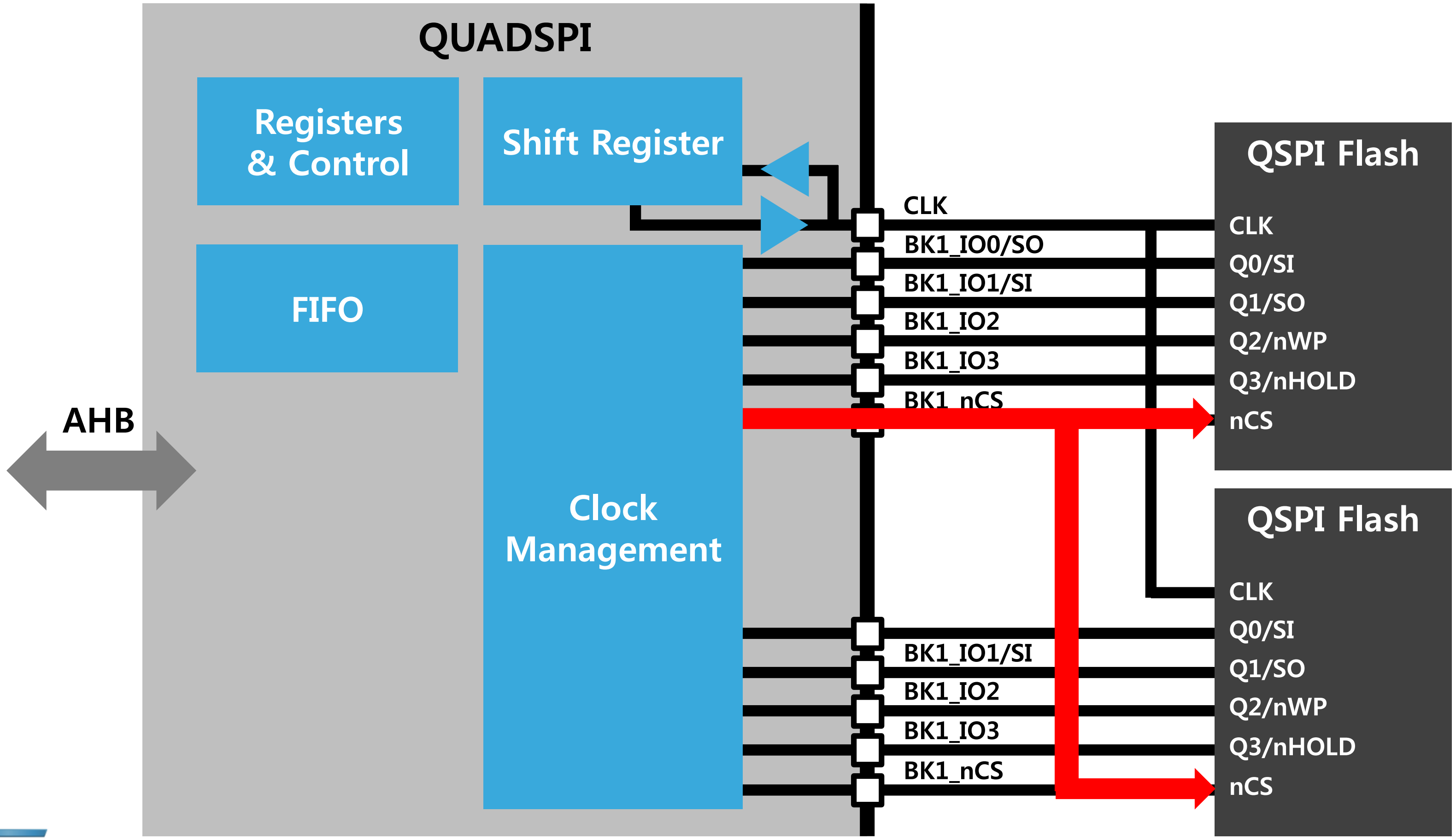
Quad-SPI Overview

- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



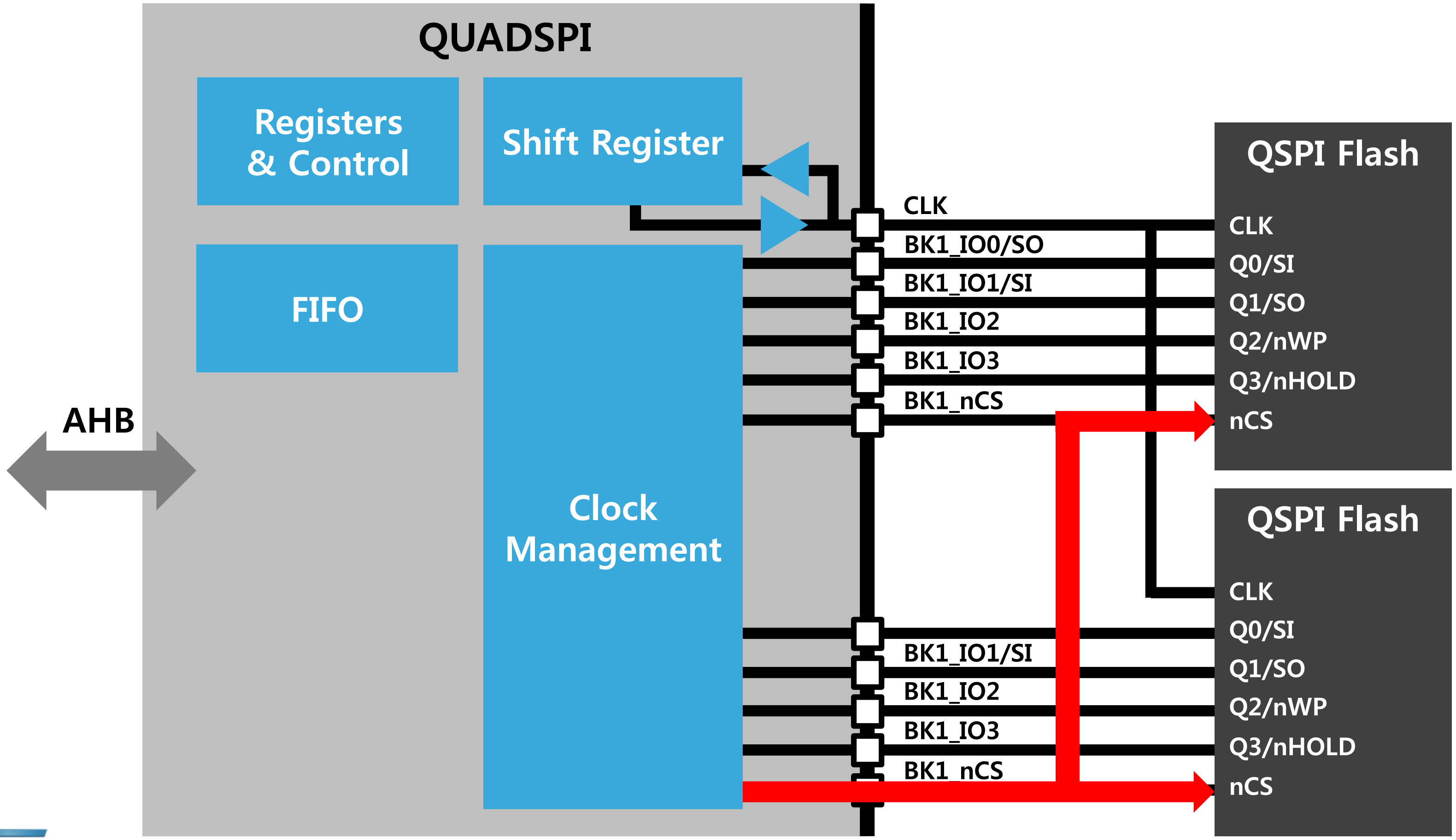
Quad-SPI Overview

- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



Quad-SPI Overview

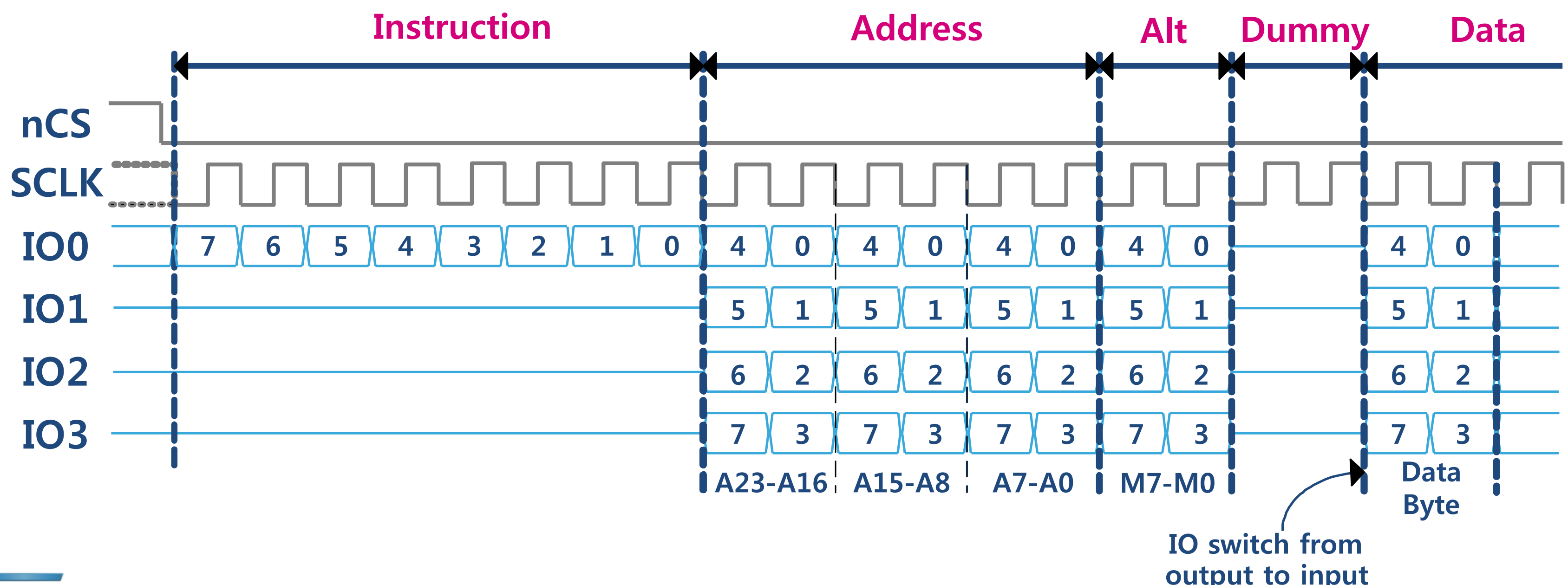
- Access two flashes in parallel with the same frame format and the same instruction (8-bit par cycle)



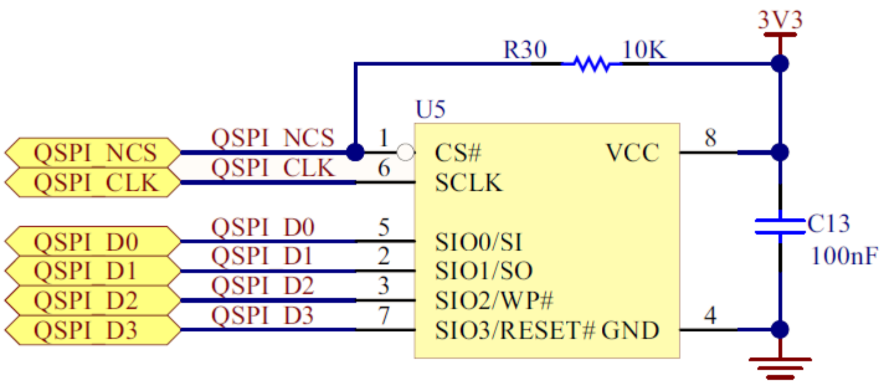
Flexible frame format

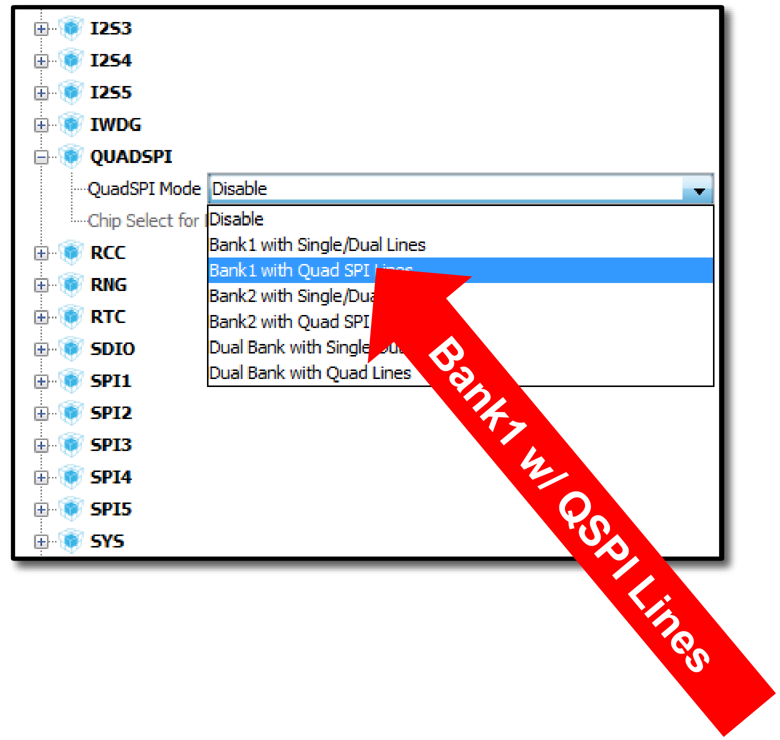
- Each of the 5 stages(phases) is fully configurable
  - Enabled or not
  - Length (8-bit to 32-bit)
  - Number of lanes (1/2/4)

- Exemple of Read configuration
  - Instruction(1 lane), Address/Alternate/Data(4 lanes), 2 dummy cycles

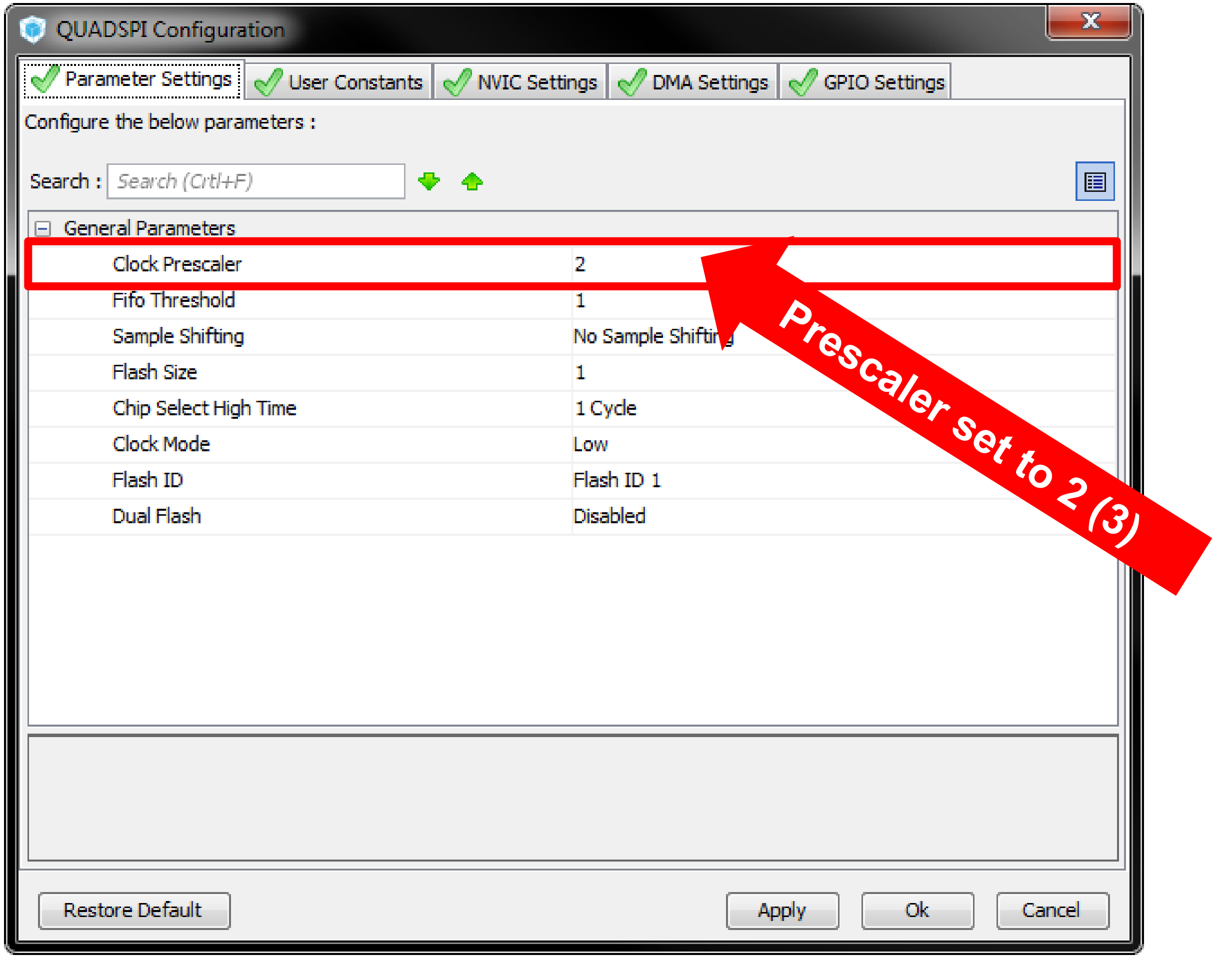


CubeMX Configuration for QSPI Connection

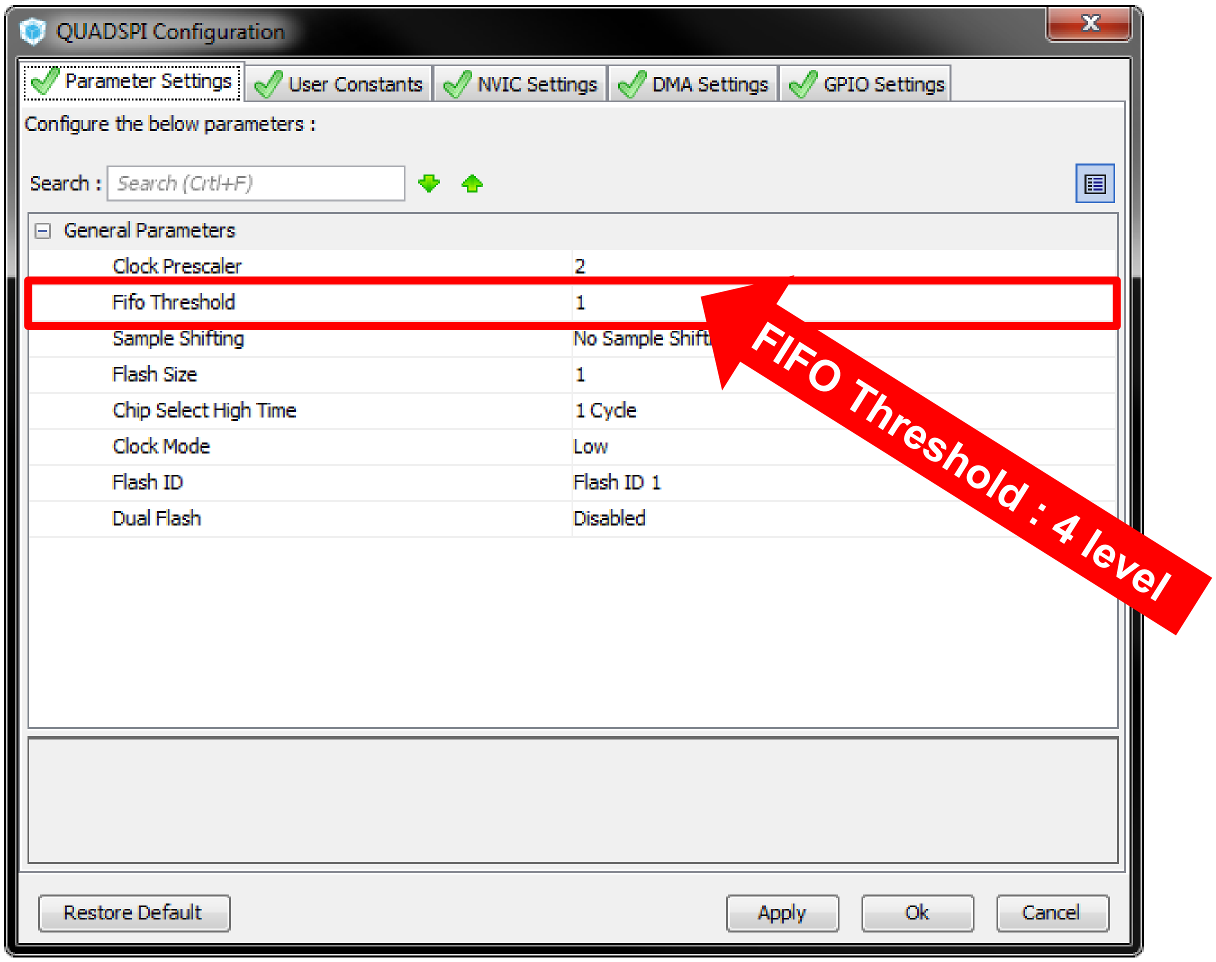




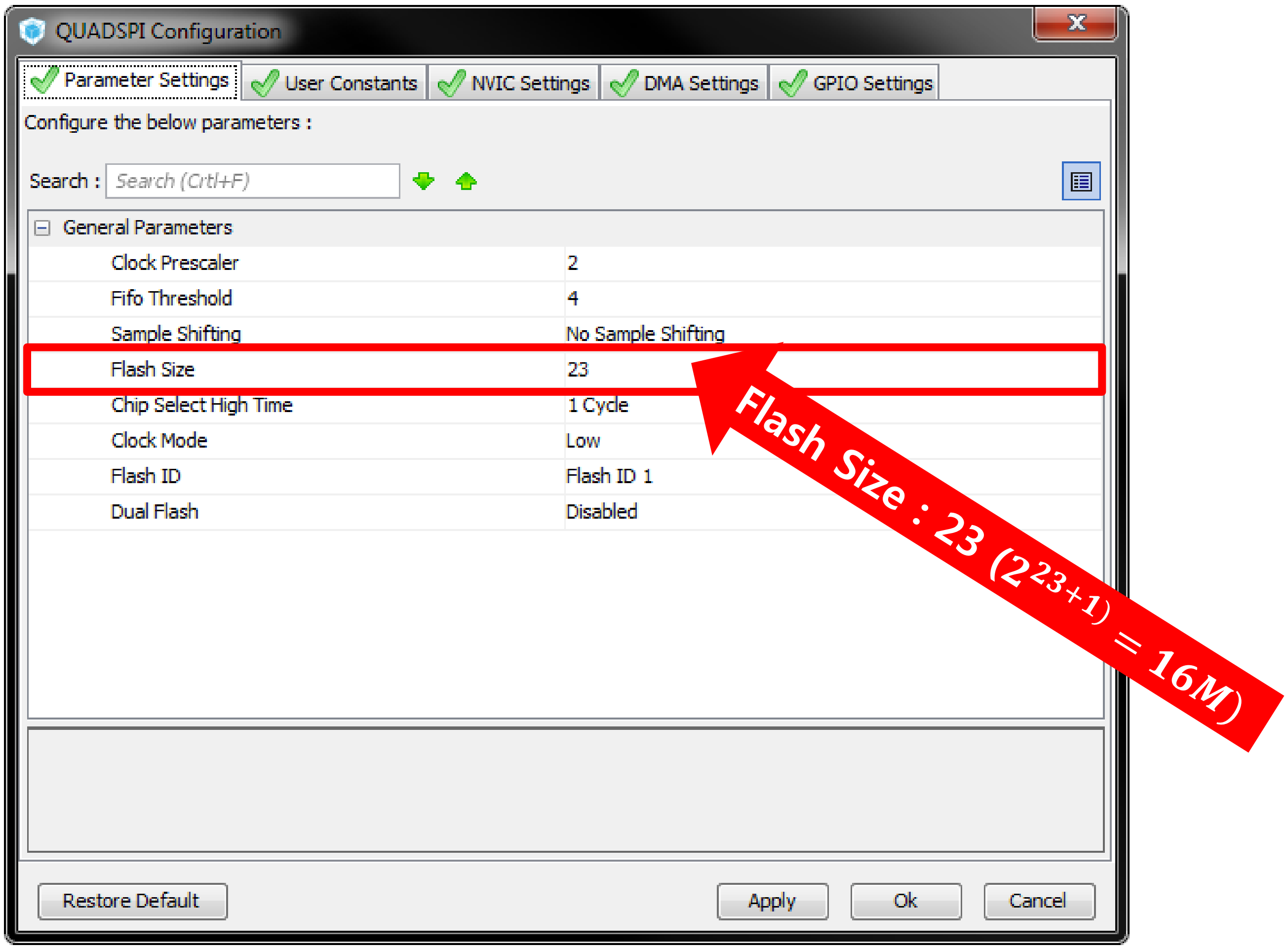
CubeMX QUADSPI Configuration



CubeMX QUADSPI Configuration

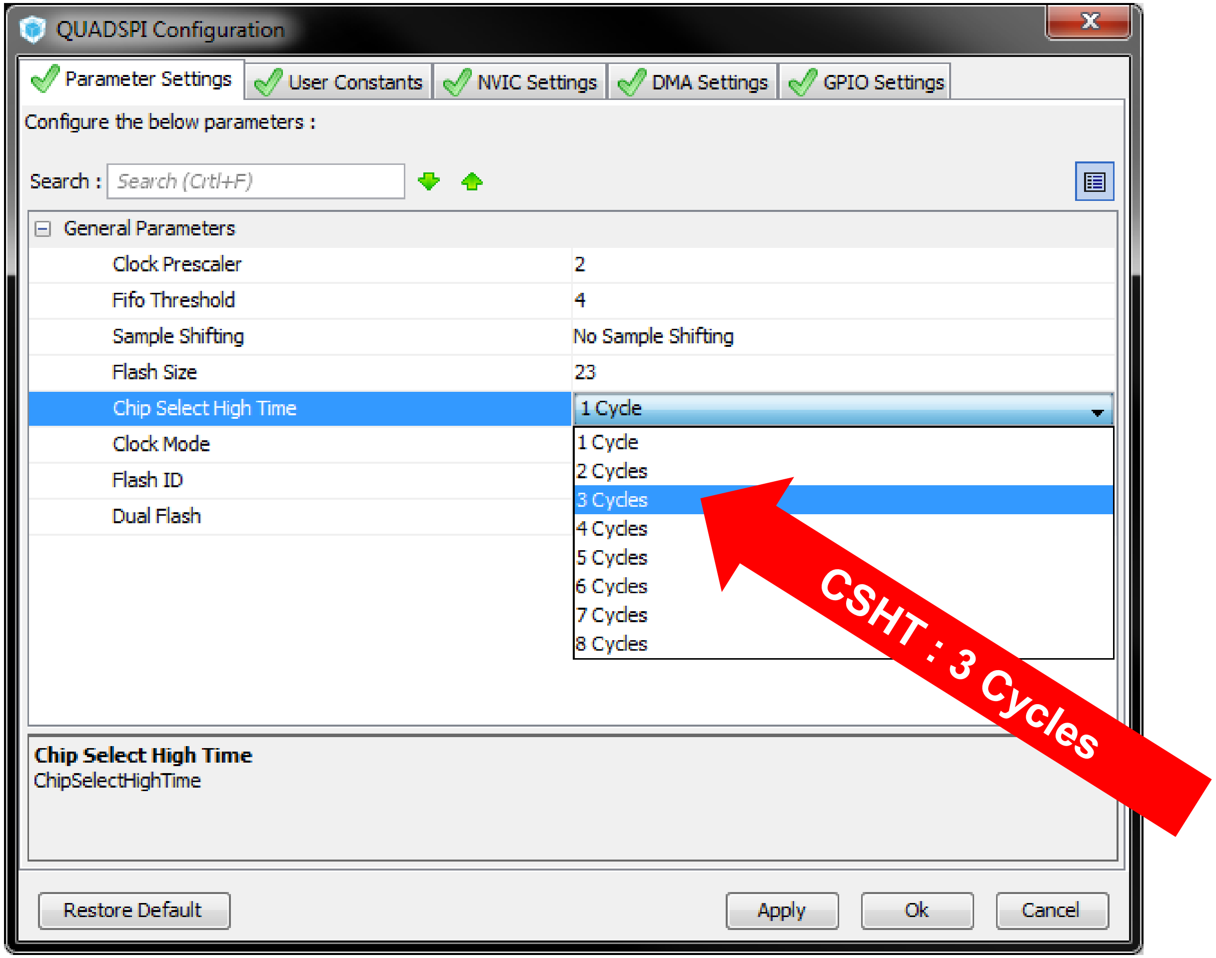


CubeMX QUADSPI Configuration

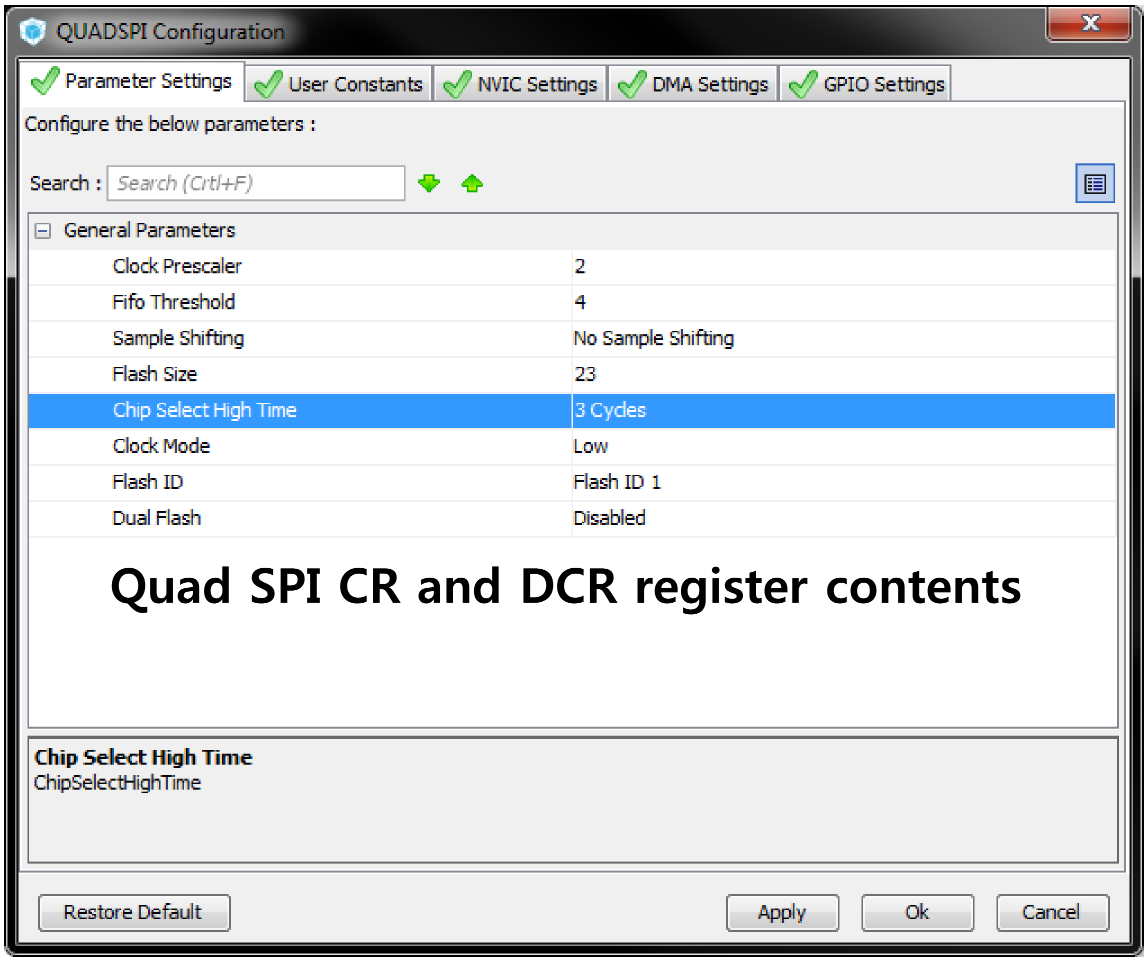


life.augmented

CubeMX QUADSPI Configuration



CubeMX QUADSPI Configuration



QSPI Memory Specification

Read/Write Array Commands (4 Byte Address Command Set)

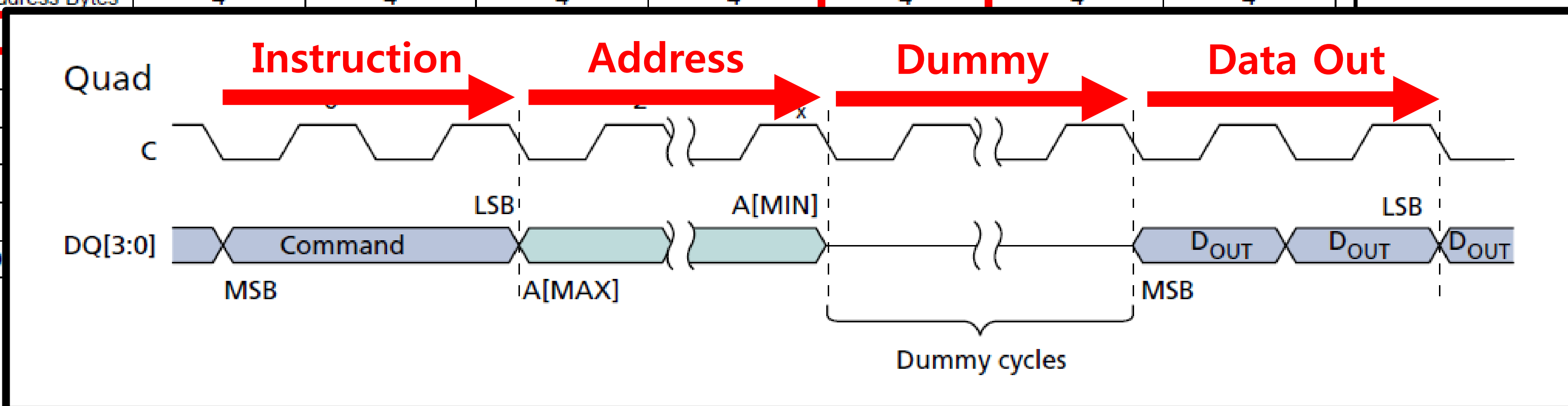
Command (byte)	READ4B	FAST READ4B	2READ4B	DREAD4B	4READ4B	QREAD4B	FRDTRD4B (fast DT read)
Mode	SPI	SPI	SPI	SPI	SPI/QPI	SPI	SPI
Address Bytes	4	4	4	4	4	4	4
1st byte	13 (hex)	0C (hex)	BC (hex)	3C (hex)	EC (hex)	6C (hex)	0E (hex)
2nd byte	ADD1	ADD1	ADD1	ADD1	ADD1	ADD1	ADD1
3rd byte	ADD2	ADD2	ADD2	ADD2	ADD2	ADD2	ADD2
4th byte	ADD3	ADD3	ADD3	ADD3	ADD3	ADD3	ADD3
5th byte	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4
6th byte		Dummy*	Dummy*	Dummy*	Dummy*	Dummy*	Dummy*
Data Cycles							
Action	read data byte by 4 byte address	re 4					

Command	Code	Extended	Dual I/O	Quad I/O	Data Bytes	Notes
READ Operations						
READ	03h	Yes	No	No	1 to ∞	4
FAST READ	0Bh	Yes	Yes	Yes		5
DUAL OUTPUT FAST READ	3Bh	Yes	Yes	No	1 to ∞	5
DUAL INPUT/OUTPUT FAST READ	0Bh 3Bh BBh	Yes	Yes	No		5, 6
QUAD OUTPUT FAST READ	6Bh	Yes	No	Yes	1 to ∞	5
QUAD INPUT/OUTPUT FAST READ	0Bh 6Bh EBh	Yes	No	Yes		5, 7

QSPI Memory Specification

Read/Write Array Commands (4 Byte Address Command Set)

Command (byte)	READ4B	FAST READ4B	2READ4B	DREAD4B	4READ4B	QREAD4B	FRDTRD4B (fast DT read)
Mode	SPI	SPI	SPI	SPI	SPI/QPI	SPI	SPI
Address Bytes	4	4	4	4	4	4	4



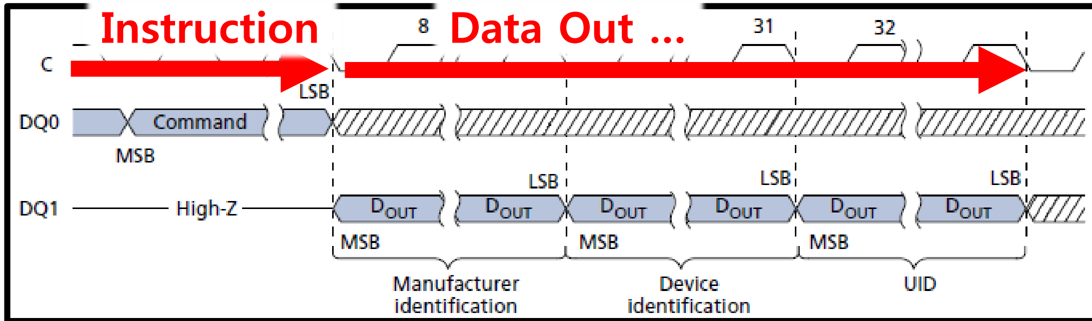
FAST READ	0Bh	Yes	Yes	Yes		5
DUAL OUTPUT FAST READ	3Bh	Yes	Yes	No	1 to ∞	5
DUAL INPUT/OUTPUT FAST READ	0Bh 3Bh BBh	Yes	Yes	No		5, 6
QUAD OUTPUT FAST READ	6Bh	Yes	No	Yes	1 to ∞	5
QUAD INPUT/OUTPUT FAST READ	0Bh 6Bh EBh	Yes	No	Yes		5, 7

life.augmented

Construct Framing using QSPI HAL


```
void QSPI_ReadID(QSPI_HandleTypeDef *hqspi, uint8_t *id)
{
  QSPI_CommandTypeDef cmd;

  cmd.InstructionMode    = QSPI_INSTRUCTION_1_LINE;
  cmd.AddressMode        = QSPI_ADDRESS_NONE;
  cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
  cmd.DataMode           = QSPI_DATA_1_LINE;
  cmd.DummyCycles        = 0;

  cmd.Instruction        = READ_ID_CMD;
  cmd.address            = 0; /* don't care */
  cmd.NbData             = 3;

  cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode           = QSPI_SIOO_INST_EVERY_CMD;

  if (HAL_QSPI_Command(hqspi, &cmd, HAL_QPSI_TMOUT) != HAL_OK) {
    Error_Handler();
  }

  if (HAL_QSPI_Receive_DMA(hqspi, id) != HAL_OK) {
    Error_Handler();
  }
}
```

Construct Framing using QSPI HAL

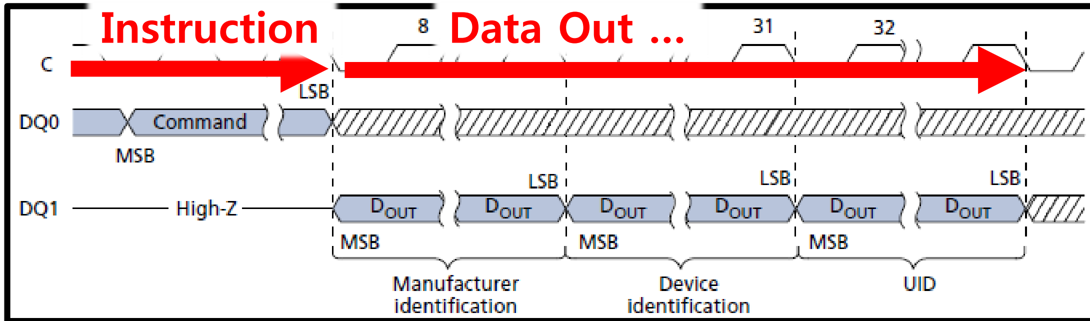



```
peDef *hqspi, uint8_t *id)

  cmd.InstructionMode    = QSPI_INSTRUCTION_1_LINE;
  cmd.AddressMode        = QSPI_ADDRESS_NONE;
  cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
  cmd.DataMode           = QSPI_DATA_1_LINE;
  cmd.DummyCycles        = 0;

  cmd.Instruction        = READ_ID_CMD;
  cmd.address            = 0; /* don't care */
  cmd.NbData             = 3;

  cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode           = QSPI_SIOO_INST_EVERY_CMD;

  if (HAL_QSPI_Command(hqspi, &cmd, HAL_QPSI_TMOUT) != HAL_OK) {
    Error_Handler();
  }

  if (HAL_QSPI_Receive_DMA(hqspi, id) != HAL_OK) {
    Error_Handler();
  }
}
```

cmd: frame information structure

life.augmented

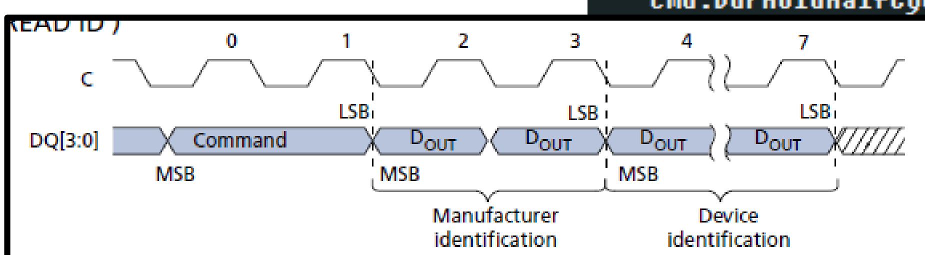
Construct Framing using QSPI HAL



INDIRECT MODE

```
peDef *hqspi, uint8_t *id)

cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
cmd.AddressMode        = QSPI_ADDRESS_NONE;
cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
cmd.DataMode           = QSPI_DATA_4_LINES;
cmd.DummyCycles        = 0;

cmd.Instruction        = MULTIPLE_IO_READ_ID_CMD;
cmd.address            = 0; /* don't care */
cmd.NbData             = 3;

cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
                       = QSPI_SIOO_INST_EVERY_CMD;

nd(hqspi, &cmd, HAL_QPSI_TMOUT) != HAL_OK) {

ve_DMA(hqspi, id) != HAL_OK) {

  }
}
```

cmd: frame information structure

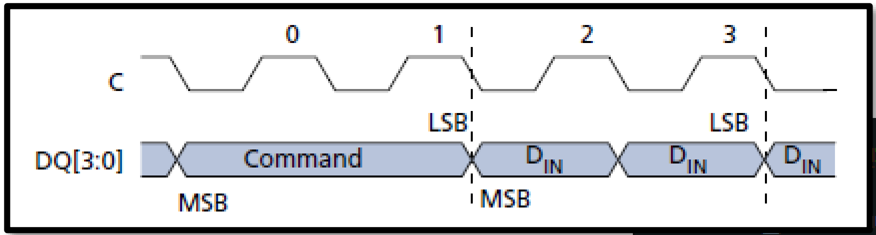


life.augmented

Automatic Status Polling using QSPI HAL

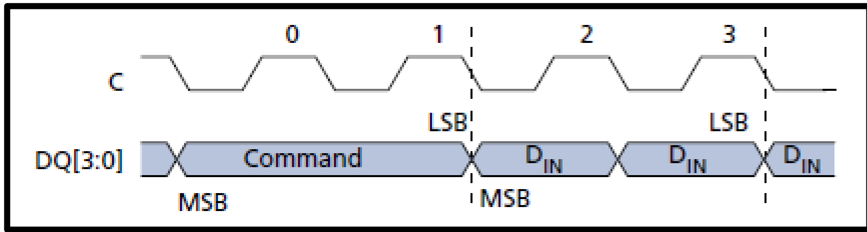

```
static void QSPI_AutoPollingMemReady(QSPI_HandleTypeDef *hqspi)
{
  QSPI_CommandTypeDef     cmd;
  QSPI_AutoPollingTypeDef cfg;

  cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
  cmd.AddressMode        = QSPI_ADDRESS_NONE;
  cmd.AlternateByteMode = QSPI_ALTERNATE_BYTES_NONE;
  cmd.DataMode           = QSPI_DATA_4_LINES;
  cmd.DummyCycles        = 0;

  cmd.Instruction        = READ_STATUS_REG_CMD;

  cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode           = QSPI_SIOO_INST_EVERY_CMD;

  cfg.Match            = 0x00; /* 1st-bit: write in progress */
  cfg.Mask             = 0x01;
  cfg.MatchMode        = QSPI_MATCH_MODE_AND;
  cfg.StatusBytesSize = 1;
  cfg.Interval         = 0x10;
  cfg.AutomaticStop    = QSPI_AUTOMATIC_STOP_ENABLE;

  if (HAL_QSPI_AutoPolling_IT(hqspi, &cmd, &cfg) != HAL_OK) {
    Error_Handler();
  }
}
```

Automatic Status Polling using QSPI HAL

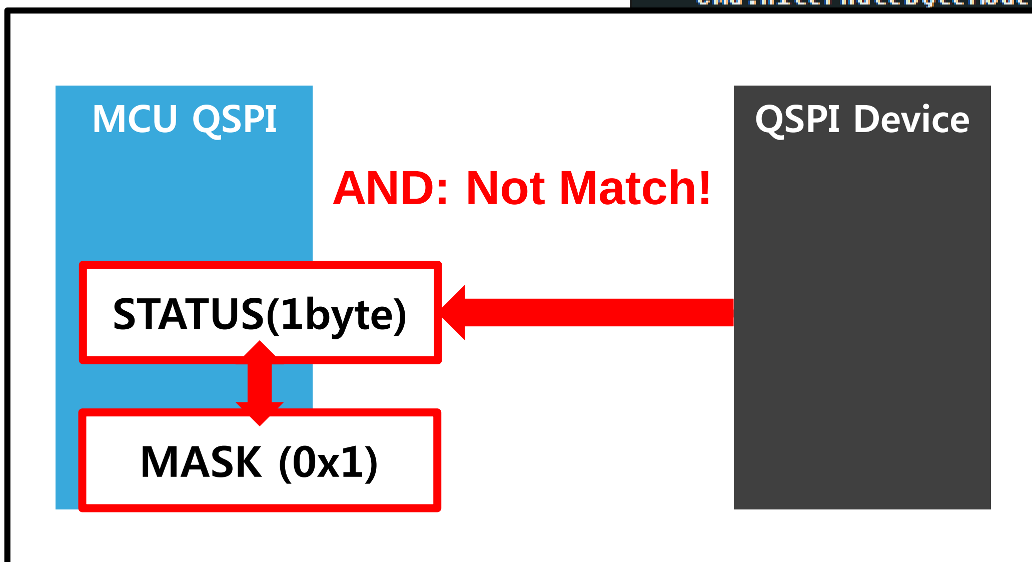



```
SPI_AutoPollingMemReady(QSPI_HandleTypeDef *hqspi)

dTypeDef      cmd;
  QSPI_AutoPollingTypeDef cfg;

  cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
  cmd.AddressMode        = QSPI_ADDRESS_NONE;
  cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
  cmd.DataMode           = QSPI_DATA_4_LINES;
  cmd.DummyCycles        = 0;

  cmd.Instruction        = READ_STATUS_REG_CMD;

  cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode           = QSPI_SIOO_INST_EVERY_CMD;

  cfg.Match             = 0x00; /* 1st-bit: write in progress */
  cfg.Mask              = 0x01;
  cfg.MatchMode         = QSPI_MATCH_MODE_AND;
  cfg.StatusBytesSize   = 1;
  cfg.Interval          = 0x10;
  cfg.AutomaticStop     = QSPI_AUTOMATIC_STOP_ENABLE;

  if (HAL_QSPI_AutoPolling_IT(hqspi, &cmd, &cfg) != HAL_OK) {
    Error_Handler();
  }
}
```

Automatic Status Polling using QSPI HAL

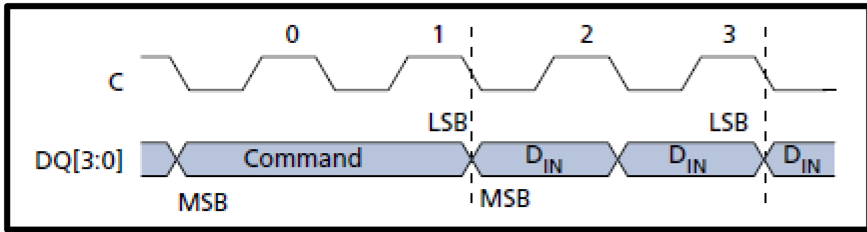



```
SPI_AutoPollingMemReady(QSPI_HandleTypeDef *hqspi)

dTypeDef     cmd;
QSPI_AutoPollingTypeDef cfg;

cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
cmd.AddressMode        = QSPI_ADDRESS_NONE;
cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
                       = QSPI_DATA_4_LINES;
                       = 0;

                       = READ_STATUS_REG_CMD;

                       = QSPI_DDR_MODE_DISABLE;
                       = QSPI_DDR_HHC_ANALOG_DELAY;
                       = QSPI_SIOO_INST_EVERY_CMD;

0x00; /* 1st-bit: write in progress */
0x01;
QSPI_MATCH_MODE_AND;
1;
0x10;
QSPI_AUTOMATIC_STOP_ENABLE;

ing_IT(hqspi, &cmd, &cfg) != HAL_OK) {
  Error_Handler();
 }
}
```



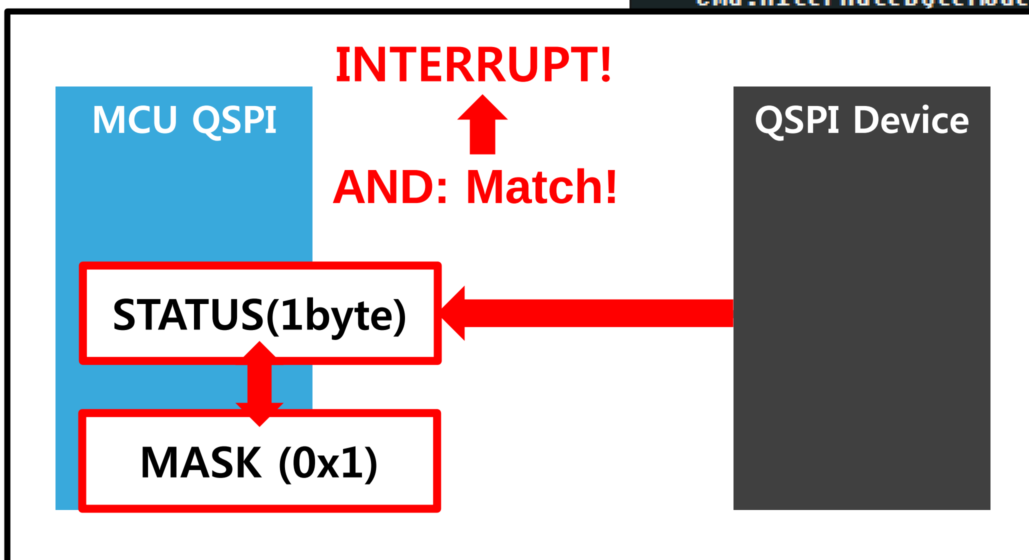
life.augmented

Automatic Status Polling using QSPI HAL



STATUS POLLING MODE

```
SPI_AutoPollingMemReady(QSPI_HandleTypeDef *hqspi)

dTypeDef      cmd;
QSPI_AutoPollingTypeDef cfg;

cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
cmd.AddressMode        = QSPI_ADDRESS_NONE;
cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
                       = QSPI_DATA_4_LINES;
                       = 0;

                       = READ_STATUS_REG_CMD;

                       = QSPI_DDR_MODE_DISABLE;
                       = QSPI_DDR_HHC_ANALOG_DELAY;
                       = QSPI_SIOO_INST_EVERY_CMD;
0x00; /* 1st-bit: write in progress */
0x01;
QSPI_MATCH_MODE_AND;
1;
0x10;
QSPI_AUTOMATIC_STOP_ENABLE;
ing_IT(hqspi, &cmd, &cfg) != HAL_OK) {
    Error_Handler();
  }
}
```



life.augmented

QSPI Memory Mapped Mode

- Prefetch for XiP
- External flash seen as internal with wait states
  - Read operations are automatically generated on AHB access
  - Frame & opcode defined during IP configuration as for indirect mode
- Base address mapped to 0x9000 0000 (* difference each MCU)
- Up to 256Mbytes spaces
- Read operation ONLY

QSPI Memory Mapped Mode


```
  QSPI_Reset(&hqspi);

#if (USE_SPI_MODE == USE_SPI_MODE_QUAD)
  QSPI_SetQuadSPIMode(&hqspi, QSPI_SPI_QUAD_MODE);
#elif (USE_SPI_MODE == USE_SPI_MODE_EXTENDED)
  QSPI_SetQuadSPIMode(&hqspi, QSPI_SPI_EXTENDED_MODE);
#endif
  QSPI_ReadID(&hqspi, id);
  QSPI_DummyCyclesCfg(&hqspi);

  QSPI_Erase(&hqspi, 0);
  QSPI_Read(&hqspi, 0, verify, 16);

  sprintf((char *)txbuf, "Hello, world!\n");
  QSPI_Write(&hqspi, 0, txbuf, 16);
  QSPI_Read(&hqspi, 0, rxbuf, 32);

  QSPI_TurnToMemoryMap(&hqspi);
```

QSPI Memory Mapped Mode

Turn back to the default interface

QSPI RESET

QSPI IFCE MODE EXCHANGE

```
  QSPI_Reset(&hqspi);

#if (USE_SPI_MODE == USE_SPI_MODE_QUAD)
  QSPI_SetQuadSPIMode(&hqspi, QSPI_SPI_QUAD_MODE);
#elif (USE_SPI_MODE == USE_SPI_MODE_EXTENDED)
  QSPI_SetQuadSPIMode(&hqspi, QSPI_SPI_EXTENDED_MODE);
#endif
  QSPI_ReadID(&hqspi, id);
  QSPI_DummyCyclesCfg(&hqspi);

  QSPI_Erase(&hqspi, 0);
  QSPI_Read(&hqspi, 0, verify, 16);

  sprintf((char *)txbuf, "Hello, world!\n");
  QSPI_Write(&hqspi, 0, txbuf, 16);
  QSPI_Read(&hqspi, 0, rxbuf, 32);

  QSPI_TurnToMemoryMap(&hqspi);
```

life.augmented

QSPI Memory Mapped Mode

ace

```
void QSPI_TurnToMemoryMap(QSPI_HandleTypeDef *hqspi)
{
  QSPI_CommandTypeDef cmd;
  QSPI_MemoryMappedTypeDef memmap;

  cmd.InstructionMode    = QSPI_INSTRUCTION_4_LINES;
  cmd.AddressMode        = QSPI_ADDRESS_4_LINES;
  cmd.AddressSize        = QSPI_ADDRESS_24_BITS;
  cmd.AlternateByteMode  = QSPI_ALTERNATE_BYTES_NONE;
  cmd.DataMode           = QSPI_DATA_4_LINES;
  cmd.DummyCycles        = DUMMY_CLOCK_CYCLES_READ_QUAD;

  cmd.Instruction        = QUAD_INOUT_FAST_READ_CMD;

  cmd.DdrMode            = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle   = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode           = QSPI_SIOO_INST_EVERY_CMD;

  memmap.TimeOutActivation = QSPI_TIMEOUT_COUNTER_DISABLE;

  QSPI_FIFOFlush(hqspi);

  if (HAL_QSPI_MemoryMapped(hqspi, &cmd, &memmap) != HAL_OK) {
    Error_Handler();
  }
}
```

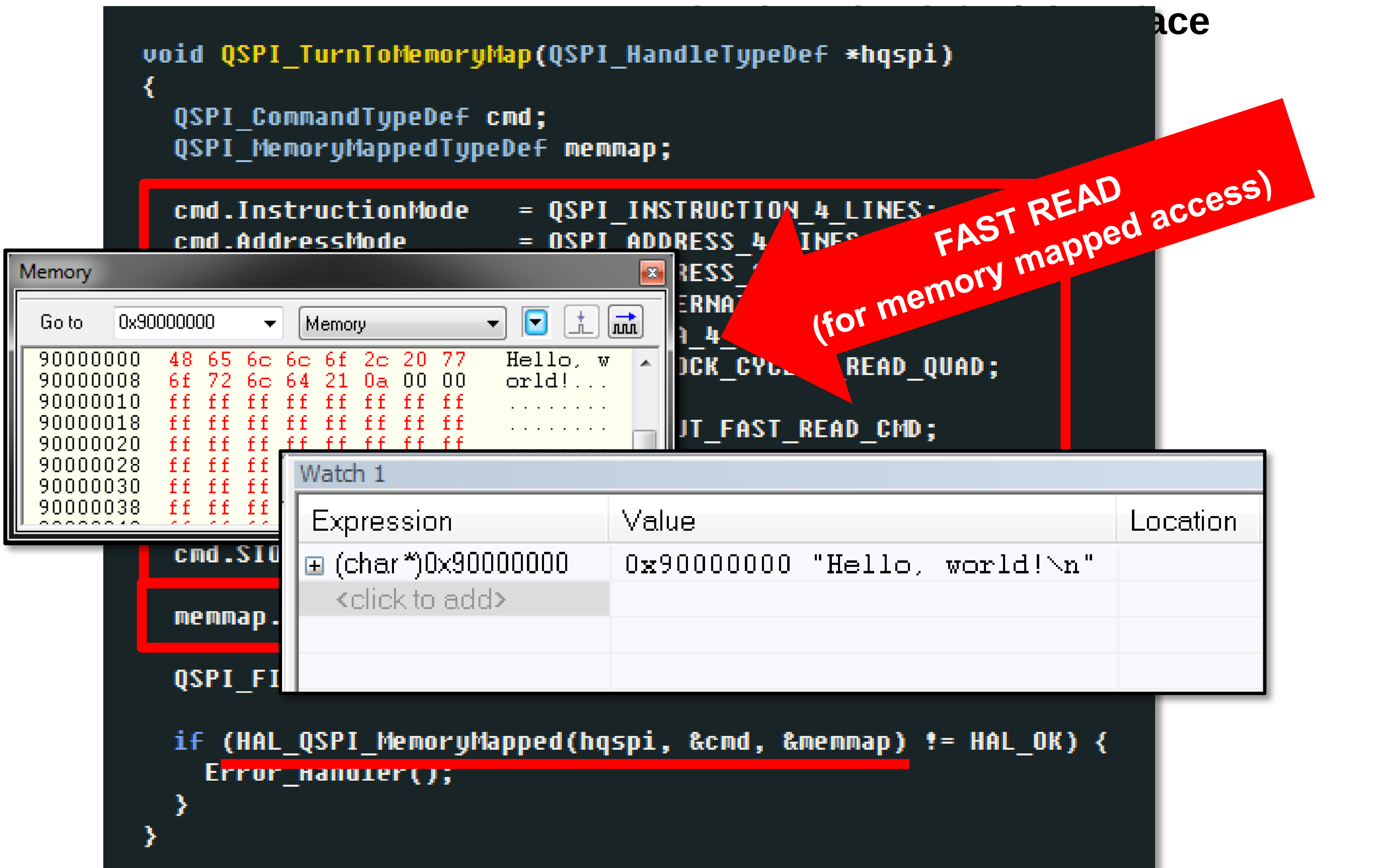
QSPI Memory Mapped Mode

ace

```
void QSPI_TurnToMemoryMap(QSPI_HandleTypeDef *hqspi)
{
  QSPI_CommandTypeDef cmd;
  QSPI_MemoryMappedTypeDef memmap;

  cmd.InstructionMode   = QSPI_INSTRUCTION_4_LINES;
  cmd.AddressMode       = QSPI_ADDRESS_4 INES
  cmd.AddressSize       = QSPI_ADDRESS_
  cmd.AlternateByteMode = QSPI_ALTERNA
  cmd.DataMode          = QSPI_DATA_4_
  cmd.DummyCycles       = DUMMY_CLOCK_CYC  READ_QUAD;

  cmd.Instruction       = QUAD_INOUT_FAST_READ_CMD;

  cmd.DdrMode           = QSPI_DDR_MODE_DISABLE;
  cmd.DdrHoldHalfCycle  = QSPI_DDR_HHC_ANALOG_DELAY;
  cmd.SIOOMode          = QSPI_SIOO_INST_EVERY_CMD;

  memmap.TimeOutActivation = QSPI_TIMEOUT_COUNTER_DISABLE;

  QSPI_FIFOFlush(hqspi);

  if (HAL_QSPI_MemoryMapped(hqspi, &cmd, &memmap) != HAL_OK) {
    Error_Handler();
  }
}
```

FAST READ
(for memory mapped access)

QSPI Memory Mapped Mode



END