AN0106
应用笔记

AT32 MCU XMC入门指南

前言

XMC 是一个将 AHB 传输信号与外部存储器信号相互转换的外设。支持的外部存储器有静态随机存储器 SRAM、NOR FLASH、PSRAM、NAND FLASH、PC 卡和同步动态随机存储器 SDRAM。另外，XMC 接口还可以用于驱动 LCD 屏。本文介绍了 XMC 外设功能，以及上述几种存储器和 LCD 的驱动方式和相关代码。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本 BSP，需要注意使用上的区别。

支持型号列表：

支持型号	具备 XMC 的型号

目录

表目录

图目录

1 XMC 概述

1） SRAM/NOR FLASH/PSRAM 界面：

- 支持 4 个外部存储器的片选信号，拥有各自的控制寄存器；
- 支持的存储器件包括：

  --SRAM

  --NOR FLASH

  --PSRAM
- 支持 8 位与 16 位数据宽度存储器；
- 提供多种时序模式选择：

  --读写相同时序的 2 种模式（模式 1、模式 2）

  --读写不同时序的 4 种模式（模式 A、模式 B、模式 C、模式 D）

  --地址数据线复用的模式（复用模式）

  --有时钟的同步模式（同步模式）
- 具可编程的时序控制寄存器；
- 支持硬件自动将 AHB 数据宽度转换为外部存储器适用的数据宽度；
- 支持驱动部分型号的 LCD 屏。

2）NAND FLASH 界面：

- 支持 2 个外部存储器的片选信号；
- 支持 8 位与 16 位数据宽度 NAND FLASH；
- 区分两个存储空间，各自具备可编程的时序控制寄存器；
- 支持将 AHB 数据宽度转换为外部存储器适用的数据宽度；
- 支持 ECC 运算。

3）PC 卡界面：

- 支持 1 个外部存储器的片选信号；
- 支持 16 位数据宽度 PC 卡；
- 区分三个存储空间，各自具备可编程的时序控制寄存器；
- 支持将 AHB 数据宽度转换为外部存储器适用的数据宽度。

4）SDRAM 界面：

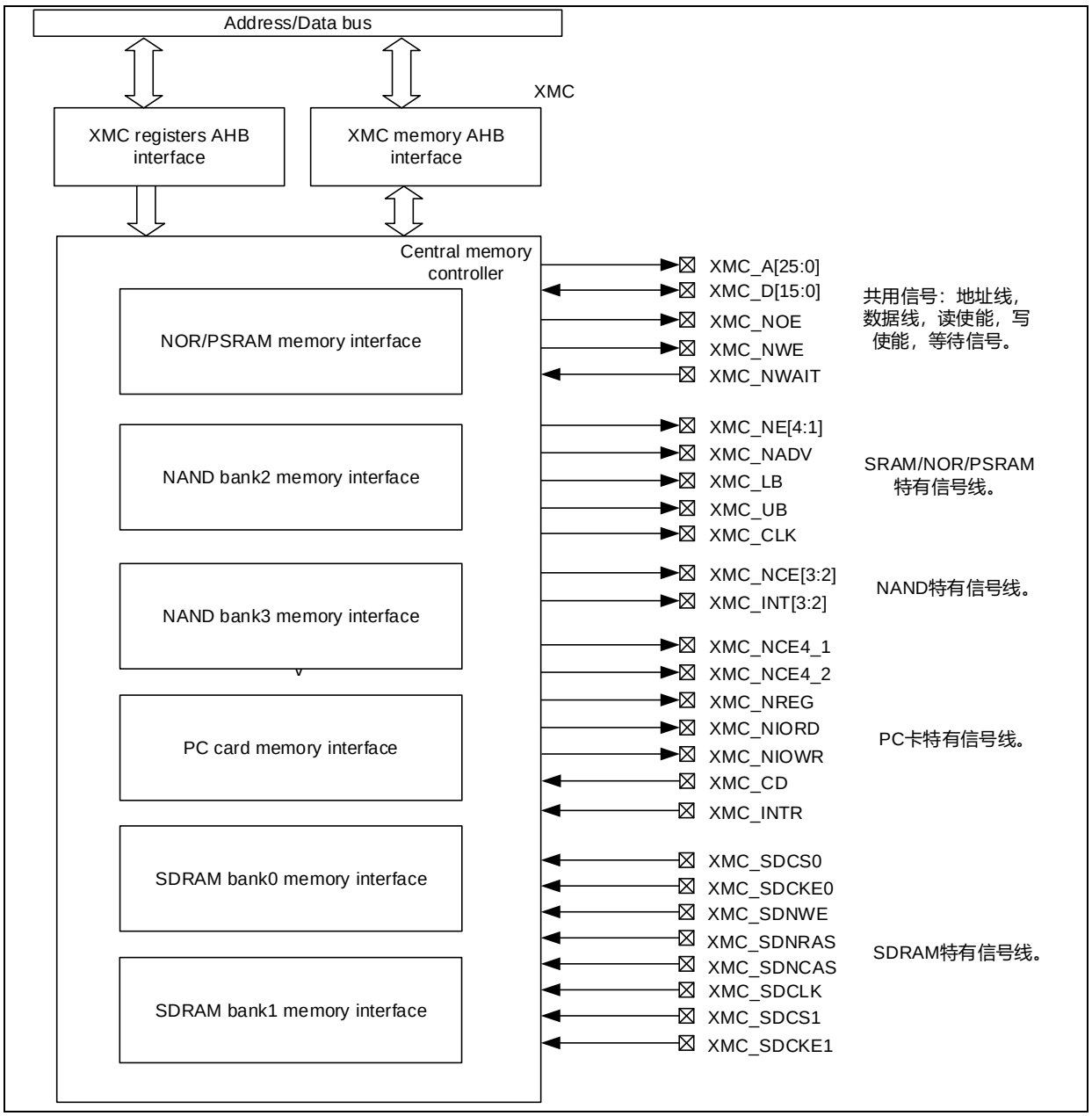
SDRAM 界面请参考“AN0089_AT32_MCU_SDRAM_Application_Note”，本文不再赘述。

2 XMC 结构

2.1 XMC 框图

如下图 1，XMC 接口分为 4 个界面，每个界面对应驱动不同的存储器类型，对应使用的引脚部分相同，部分不同。

图 1 XMC 框图



2.2 XMC 地址映射

以 AT32F435/F437 为例，XMC 地址映射如下图 2：

1）0x60000000~0x6FFFFFFF：

分为 4 个 64M 的 bank，每个 bank 各自对应单独的片选 NE[1]~NE[4]，每个 bank 各可以对应 SRAM/NOR FLASH/PSRAM 任一类型。

2）0x70000000~0x8FFFFFFF：

分为 2 个 bank，每个 bank 各自对应单独的片选 NCE[2]/NCE[3]，每个 bank 各能对应一个 NAND FLASH。

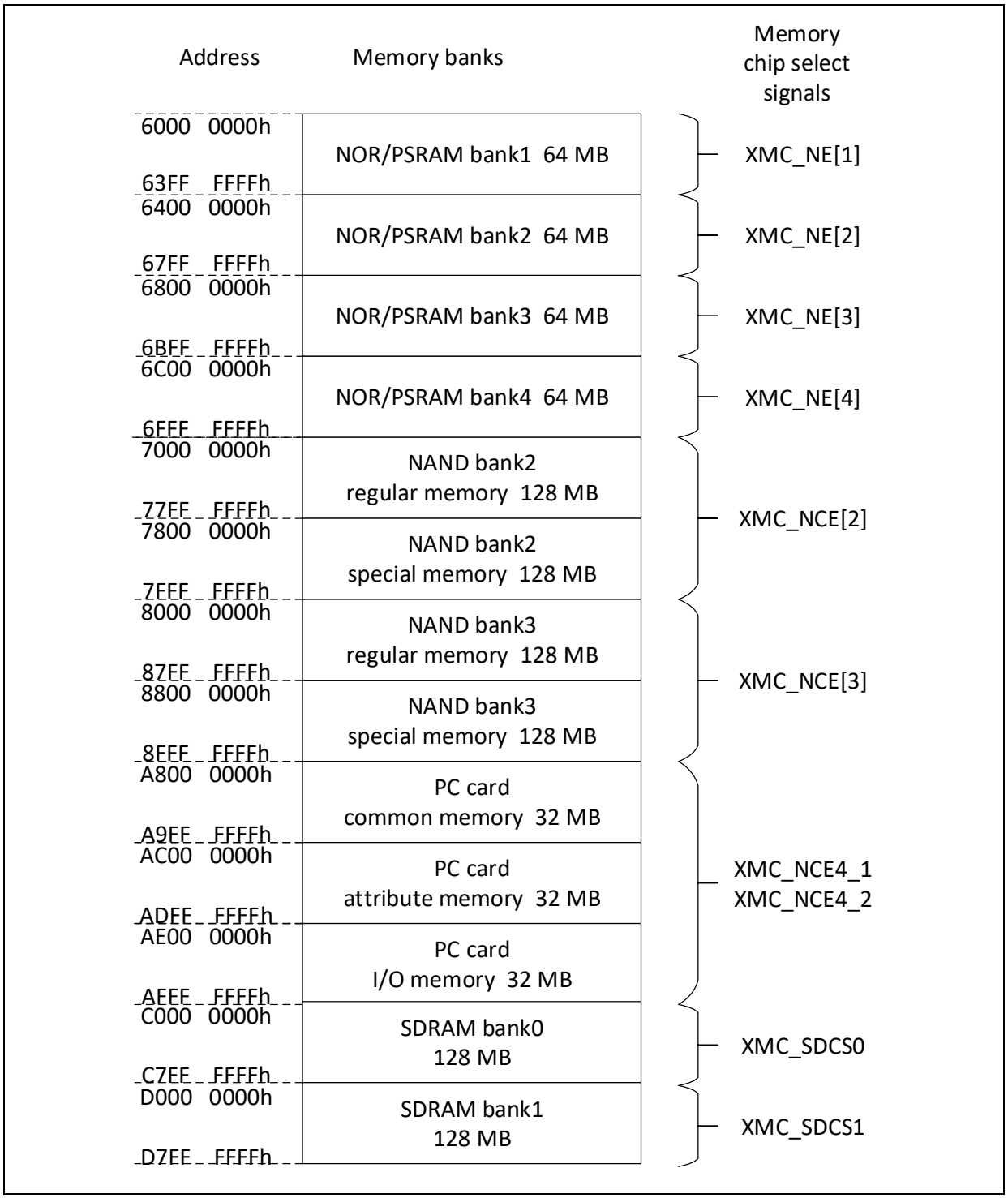
3）0xA8000000~0xAFFFFFFF：

包含 1 个 bank，对应片选信号 NCE4_1 和 NCE4_2，可以对应一个 PC 卡。

4）0xC0000000~0xC7FFFFFF/0xD0000000~0xD7FFFFFF：

包含两个 bank，对应片选信号 SDCS0 和 SDCS1，可以对应 2 个 SDRAM。

图 2 AT32F435/437 XMC 地址映射



注：上图为 AT32F435/F437 的地址映射，不同型号存在差异，请参考具体型号 RM。

2.3 XMC 操作宽度

AHB 接口为 XMC 访问外部存储器提供了通道。对 AHB 的操作会被转换成对外部存储器的操作。对 AHB 的操作有 8bit/16bit/32bit 几种操作宽度，当外部存储器的宽度是 8bit/16bit 时。对 AHB 的操作会被分割成几个 8bit/16bit 操作。如下表：

表 1 XMC 操作宽度

AHB操作宽度	外部存储器宽度	支持的操作	实际操作
32bit	8bit	读/写	对AHB的32bit操作被拆分成对存储器的4个8bit操作
32bit	16bit(2)	读/写	对AHB的32bit操作被拆分成对存储器的2个16bit操作
16bit	8bit	读/写	对AHB的16bit操作被拆分成对存储器的2个8bit操作
16bit	16bit(2)	读/写	对AHB的16bit操作对应1个对存储器的16bit操作
8bit	8bit	读/写	对AHB的8bit操作对应1个对存储器的8bit操作
8bit	16bit(2)	读/部分写(1)	对AHB的2个8bit操作对应1个对存储器的16bit操作

(1) -对于具有字节选择功能的存储器（SRAM、ROM、PSRAM 等）进行异步传输时，XMC 可以执行读/写操作并通过它的字节通道 XMC_LB，XMC_UB 可以访问正确的数据。因此支持 8bit-AHB 对 16bit 存储器的读/写操作。

-对于没有字节选择功能的存储器（NOR，NAND），仅支持 8bit-AHB 对 16bit 存储器的读操作（控制器读出完整的 16 位存储器数据，只使用需要的字节），而不支持写操作。

-对于有操作宽度指示信号的存储器（PC 卡/CF 卡），可以通过 XMC_NCE4_2 信号指示操作是 8bit 还是 16bit。支持 8bit-AHB 对 16bit 存储器的读/写操作。

(2) 对于 16 位宽度的外部存储器， XMC 将在内部使用 HADDR[22:1]对应外部存储器的地址 XMC_A[21:0]，而 HADDR[0]应保持为 0。

例如：向 16bit 的外部 NOR 存储器的地址 0x555 写入 0x70 时，XMC 应该操作如下：

*(__IO uint16_t *)(0x60000000+ (0x555<<1)) = 0x70;

即向 AHB 总线写的地址是 0x60000AAA；而实际 XMC 接口 XMC_A[21:0]=0x60000555。

3 XMC 功能

3.1 SRAM/NOR/PSRAM 界面

3.1.1 引脚定义

根据不同的存储器类型（SRAM/NOR/PSRAM）和使用的模式（异步/复用/同步），需要使用的引脚不同，典型的引脚定义如下：

表 2 SRAM/NOR/PSRAM 引脚定义

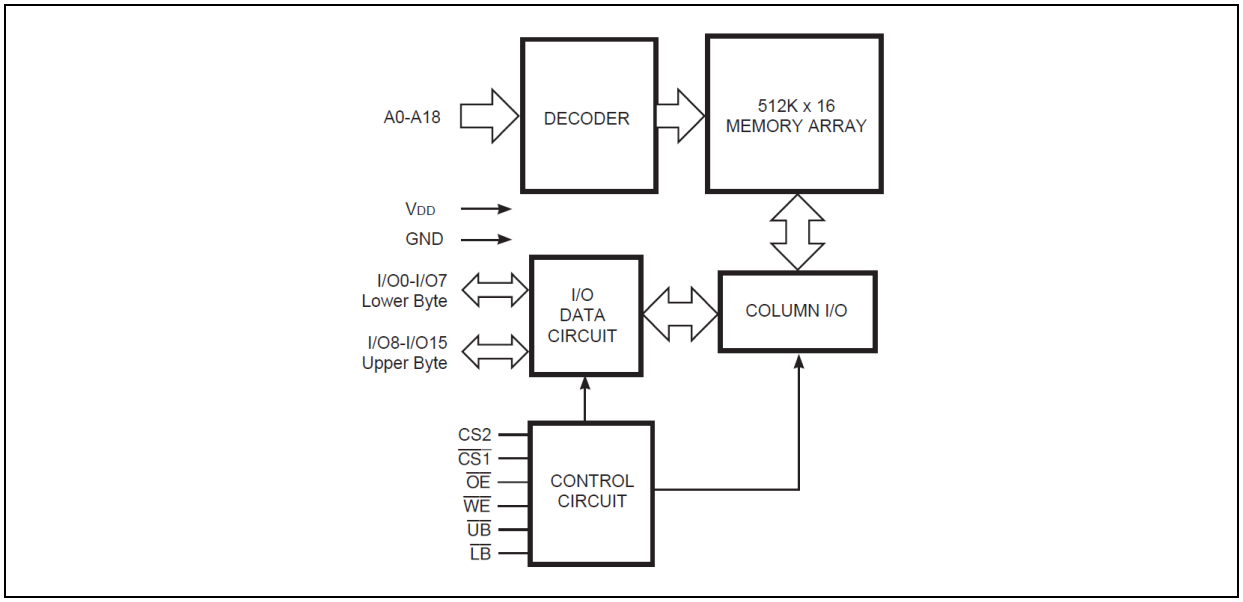
引脚信号	方向	含义
XMC_CLK	out	时钟（仅同步模式使用）
XMC_NE[x]	out	片选信号（x=1~4）
XMC_NADV	out	地址锁存/地址有效信号
XMC_A[25：0]	out	地址线
XMC_NOE	out	输出使能/读使能
XMC_NWE	out	写使能
XMC_UB	out	高字节片选--对应XMC_D[15]~[8]有效（使用NOR时无效）
XMC_LB	out	高字节片选--对应XMC_D[7]~[0]有效（使用NOR时无效）
XMC_D[15：0]	in/out	数据线
XMC_NWAIT	in	存储器要求的等待信号

3.1.2 SRAM/NOR/PSRAM 存储器介绍

3.1.2.1 SRAM

如下图 3，以异步非复用 SRAM IS62WV51216 为例：
容量为 512Kx16bit。512K=2^{19}，因此有效地址线数量为 19，即 A0~A18。
支持 16bit/8bit 读写，数据线数量为 16，即 I/O0~I/O15。16bit 读写时，$\overline{UB}$和$\overline{LB}$同时为低电平，使用数据线 I/O0~I/O15；8bit 读写时，$\overline{LB}$为低电平，$\overline{UB}$为高电平，使用数据线 I/O0~I/O7。
时序图请参考“非复用异步模式”一节，具体代码请参考“案例 1 SRAM 异步非复用读写”。

图 3 IS62WV51216 框图



3.1.2.2 PSRAM

以复用 PARAM W957D6HB 为例。

如下图 4，地址线 0~15 和数据线 0~15 复用，即 A/DQ[15:0]。另有高位数据线 A[max:16]，max 大小根据存储器容量定义，可参考上一节 SRAM 的计算方法。异步读/写操作和 SRAM 类似。

相较于 SRAM 较为特别的是，PSRAM 需要周期性的 refresh，以确保数据有效，因此不能长时间处于读/写状态，即片选信号 CS#不能长时间拉低，参考下图 5。其他时序要求请参考 W957D6HB 规格书。

时序图请参考“异步复用模式”一节，具体代码请参考“案例 2 PSRAM 异步复用读写”。

图 4 W957D6HB 框图

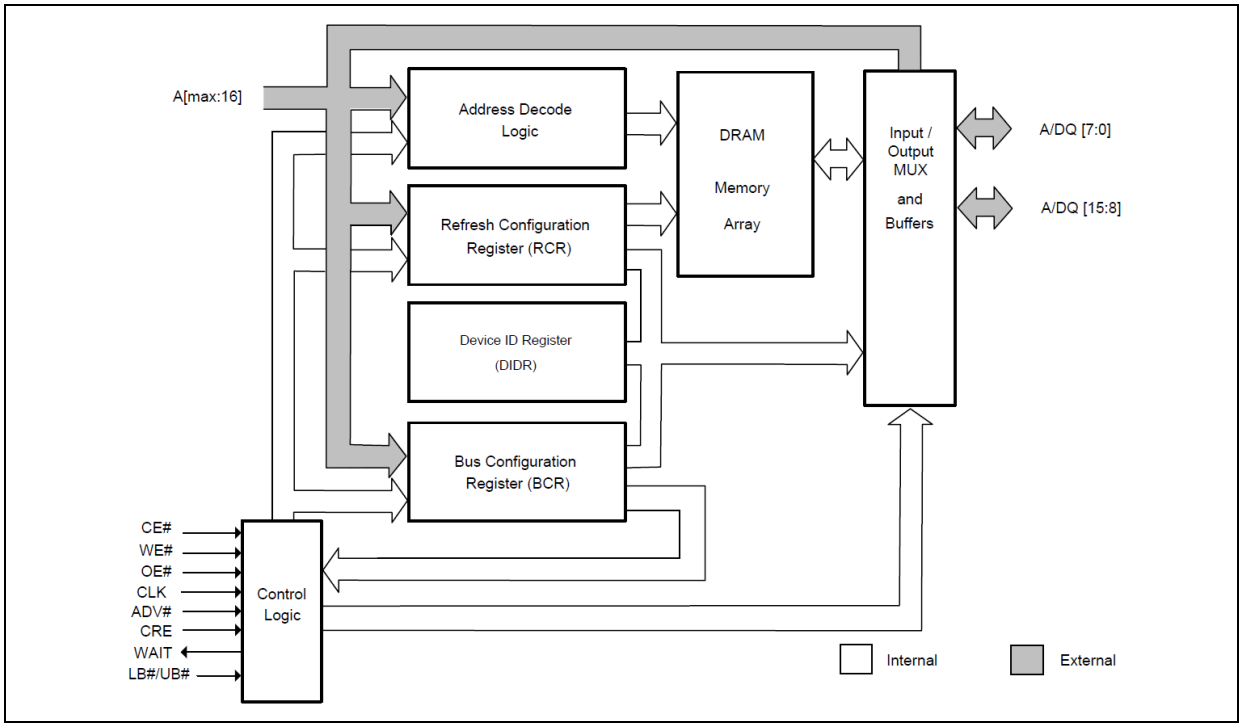


图 5 W957D6HB 时序要求

Parameter	Symbol	133MHz		104MHz		Unit	Note
		Min	Max	Min	Max		
Address access time (fixed latency)	tAA		70		70	ns	
ADV# access time (fixed latency)	tAADV		70		70	ns	
Burst to READ access time (variable latency)	tABA		34.75		34.75	ns	
CLK to output delay	tACLK		5.5		7	ns	
Address hold from ADV# HIGH (fixed latency)	tAVH	2		2		ns	
Burst OE# LOW to output delay	tBOE		20		20	ns	
CE# HIGH between subsequent burst or mixed-mode operations	tCBPH	5		5		ns	1
Maximum CE# pulse width	tCEM		4		4	µs	1

3.1.2.3 NOR FLASH

以异步非复用 NOR FLASH M29W128 为例：

M29W128 存储结构如下图 6，分为 128 个 block。1block=128K Byte=64K Word（此处 Word 为 half word）。

图 6 M29W128 存储结构

Block	Block Size	Address Range (x8)		Block Size	Address Range (x16)	
		Start	End		Start	End
127	128KB	0FE 0000h	0FF FFFFh	64KW	07F 0000h	07F FFFFh
⋮		⋮	⋮		⋮	⋮
63		07E 0000h	07F FFFFh		03F 0000h	03F FFFFh
⋮		⋮	⋮		⋮	⋮
0		000 0000h	001 FFFFh		000 0000h	000 FFFFh

Note: 1. The main memory array is divided into 128KB or 64KW uniform blocks.

如下图 7，是 M29W128 的常用命令。

读操作：

- READ/RESET，见下图中的“1”：

上电后默认状态，用于读取数据区域或复位。时序与 SRAM/PSRAM 相同，直接读对应地址即可返回数据。

- READ CFI （The common Flash interface），见下图中的“2”：

用于产品升级/厂商替换时，读取 FLASH 的各种参数（如擦除时间等），以进行程序设计。

- AUTO SELECT，见下图中的“3”：

用于读器件 ID 等信息，块保护状态等信息。

擦除操作：

- CHIP ERASE（整片擦除），见下图的“4”。
- BLOCK ERASE（块擦除），见下图的“5”。注：块是最小擦除单位。

写操作：

- PROGRAM（写操作），见下图的“6”：每次写入单个数据。“PA”即需要写入的数据的地址，“PD”即需要写入的数据。
- WRITE TO BUFFER PROGRAM（buffer 写操作），见下图的“7”：连续写入一个 buffer 的数据。“BAd”即需要写入的 block 的任一地址；“N”表示接下来要写入的数据个数；“PA”即需要写入的数据的地址，“PD”即需要写入的数据。注：此处“PA”和“PD”共“N”组。
- WRITE TO BUFFER PROGRAM CONFIRM（buffer 写确认）：如果使用了 buffer 写，即“7”的操作，那么在完成“7”的操作之后，还需要“8”的操作来确认写入。而“6”的单字节/字写入无需此步骤。

具体擦除/读/写操作代码请参考“案例 3 NOR FLASH 异步非复用读写”。

图 7 M29W128 常用命令

	Command and Code/Subcode	Bus Size	Address and Data Cycles												Notes
			1st		2nd		3rd		4th		5th		6th		
			A	D	A	D	A	D	A	D	A	D	A	D	
	READ and AUTO SELECT Operations														
1	READ/RESET (F0h)	x8	X	F0											
			AAA	AA	555	55	X	F0							
		x16	X	F0											
			555	AA	2AA	55	X	F0							
2	READ CFI (98h)	x8	AA	98											
		x16	55												
3	AUTO SELECT (90h)	x8	AAA	AA	555	55	AAA	90	Note 2	Note 2					2, 3, 4
		x16	555		2AA		555								
	BYPASS Operations														
	UNLOCK BYPASS (20h)	x8	AAA	AA	555	55	AAA	20							
		x16	555		2AA		555								
	UNLOCK BYPASS RESET (90h/00h)	x8	X	90	X	00									
		x16													
	ERASE Operations														
4	CHIP ERASE (80/10h)	x8	AAA	AA	555	55	AAA	80	AAA	AA	555	55	AAA	10	
		x16	555		2AA		555		555		2AA		555		
	UNLOCK BYPASS CHIP ERASE (80/10h)	x8	X	80	X	10									5
		x16													
5	BLOCK ERASE (80/30h)	x8	AAA	AA	555	55	AAA	80	AAA	AA	555	55	BAd	30	11
		x16	555		2AA		555		555		2AA				
	UNLOCK BYPASS BLOCK ERASE (80/30h)	x8	X	80	BAd	30									5
		x16													
	ERASE SUSPEND (B0h)	x8	X	B0											
		x16													
	ERASE RESUME (30h)	x8	X	30											
		x16													
	PROGRAM Operations														
6	PROGRAM (A0h)	x8	AAA	AA	555	55	AAA	A0	PA	PD					
		x16	555		2AA		555								
	UNLOCK BYPASS PROGRAM (A0h)	x8	X	A0	PA	PD									5
		x16													
7	WRITE TO BUFFER PROGRAM (25h)	x8	AAA	AA	555	55	BAd	25	BAd	N	PA	PD			6, 7, 8
		x16	555		2AA										
	UNLOCK BYPASS WRITE TO BUFFER PROGRAM (25h)	x8	BAd	25	BAd	N	PA	PD							5
		x16													
8	WRITE TO BUFFER PROGRAM CONFIRM (29h)	x8	BAd	29											
		x16													

3.1.3 读/写时序及控制寄存器配置

如下表 3，SRAM/NOR/PSRAM 界面读写时序分为异步/同步，复用/非复用，可以排列组合为 4 类。其中非复用异步又细分为 6 种模式。每种模式适用于特定的存储器类型，使用时需注意。

表 3 SRAM/NOR/PSRAM 界面时序种类

分类	特征	模式	适用存储器类型
异步非复用	读/写时序相同	mode1	SRAM/PSRAM
		mode2	NOR
	读/写时序不同	modeA	SRAM/PSRAM
		modeB	NOR
		modeC	NOR
		modeD	SRAM/PSRAM/NOR
复用模式	地址数据线复用	异步复用模式	PSRAM/NOR
同步模式	有时钟线	同步非复用模式	PSRAM/NOR
同步复用模式	地址数据线复用，且有时钟线	同步复用模式	PSRAM/NOR

3.1.3.1 非复用异步模式

如上表 3，非复用异步模式又分为 6 种模式。其中，mode1 和 mode2 是读写时序相同的模式；modeA、modeB、modeC、modeD 是读写时序不同的模式。以下分别列出每种模式的读/写时序和关键配置。

3.1.3.1.1 mode1

图 8 mode1 读时序

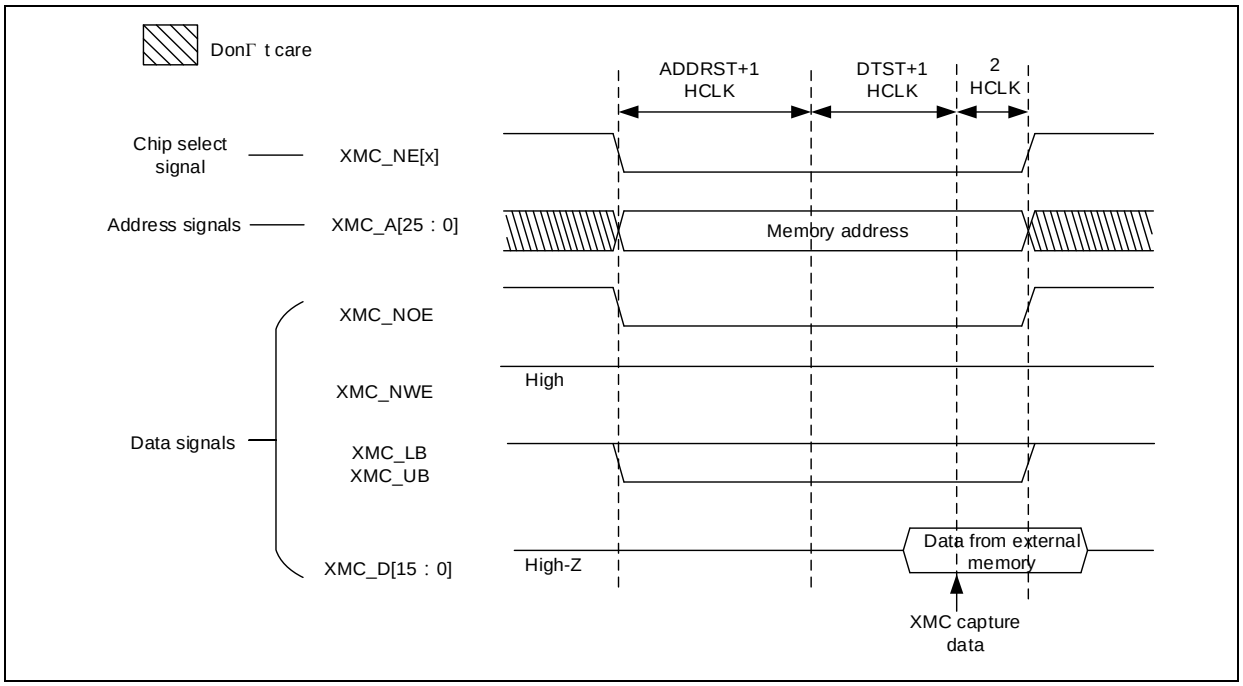


图 9 mode1 写时序

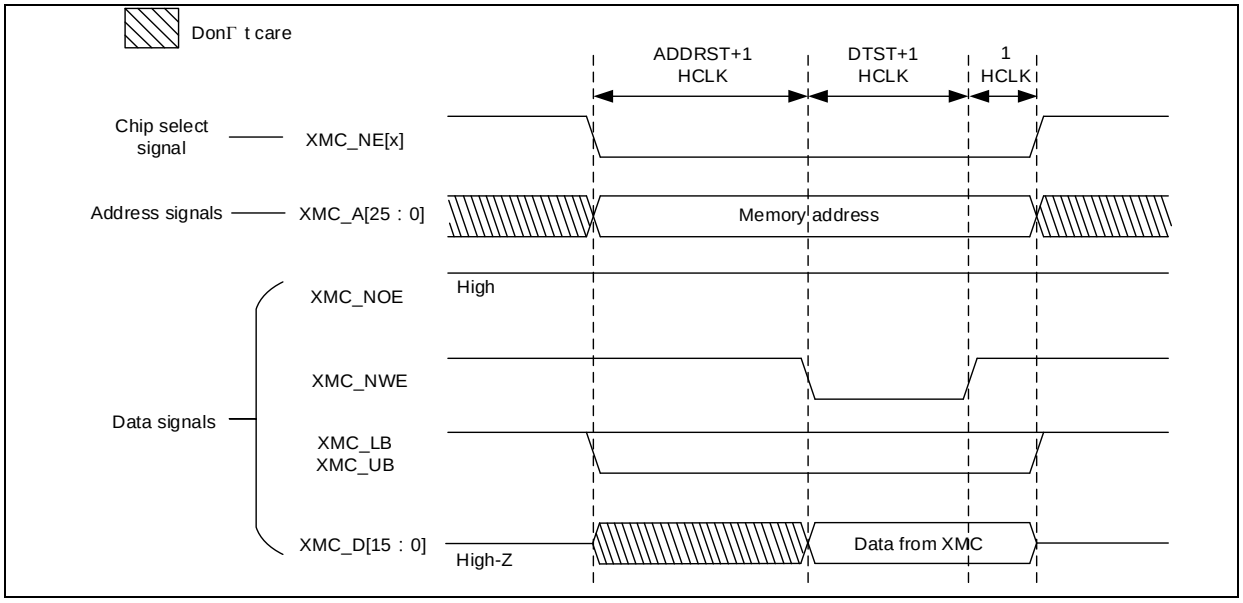


表 4 mode1 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义

RWTD	读写时序异同	0	读写时序相同 （均通过XMC_BK1TMGx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	00：SRAM 01：PSRAM	根据实际使用的存储器类型选择。 此处可选SRAM/PSRAM。
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需要通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 5 mode1 时序寄存器配置

XMC_BK1TMGx	名称	配置	含义
ASYNCM	异步访问模式选择	0	选择modeA/B/C/D。 mode1/2时，需保持默认值0。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图8，9
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图8，9

表中未列出的bit请保持默认值。

3.1.3.1.2 mode2

图 10 mode2 读时序

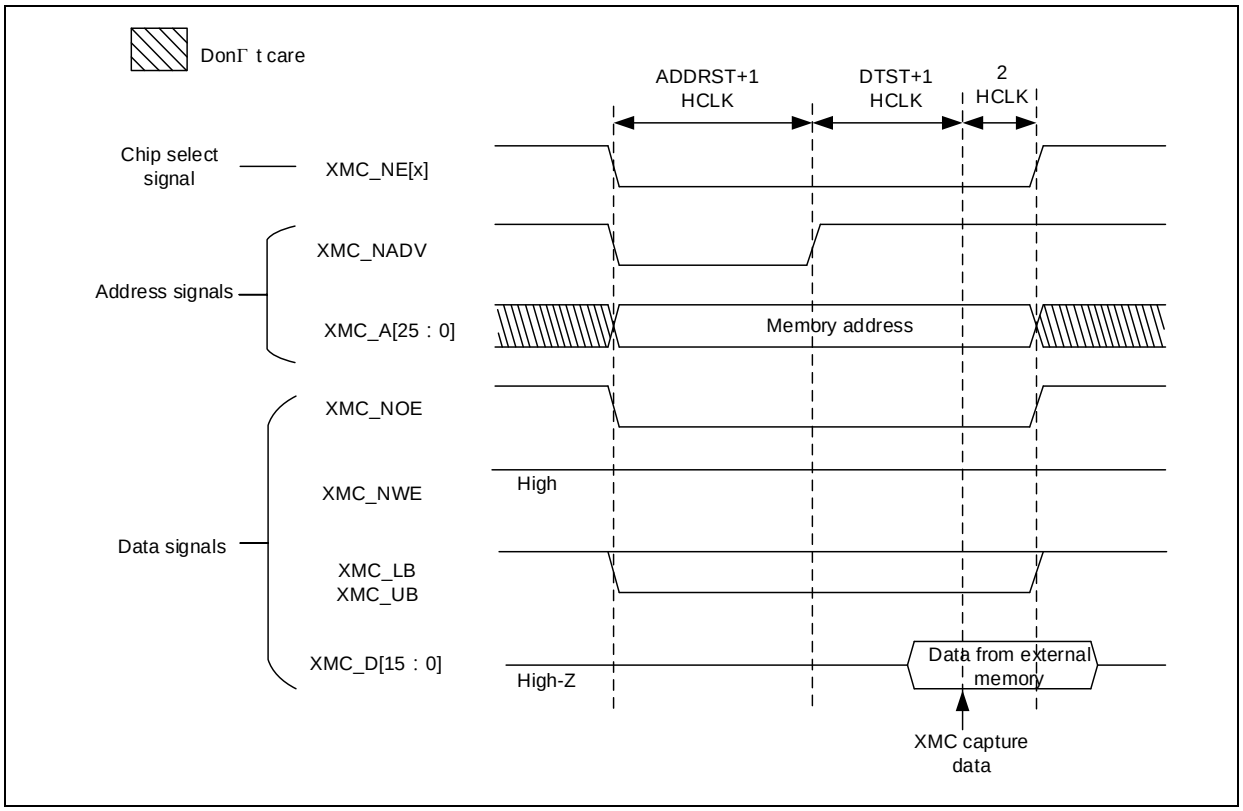


图 11 mode2 写时序

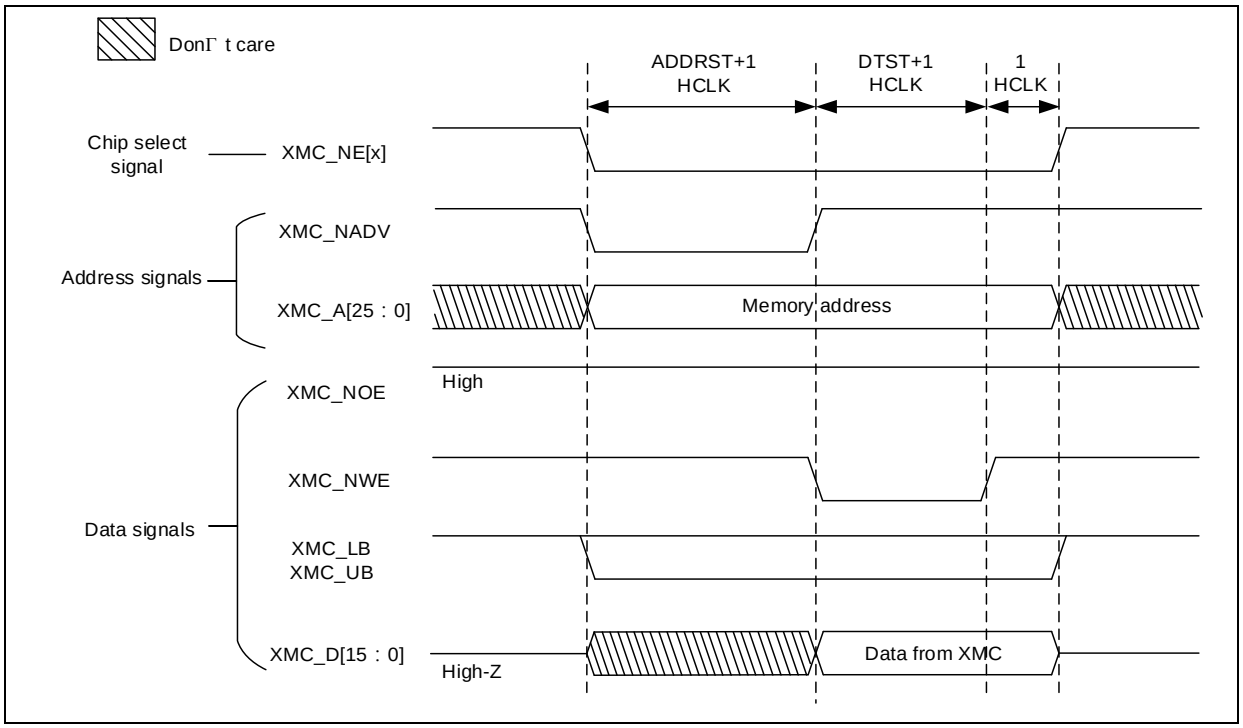


表 6 mode2 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义
RWTD	读写时序异同	0	读写时序相同 （均通过XMC_BK1TMGx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NOREN	NOR闪存访问使能	1	使用NOR时，需将此位置1
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	10：NOR	mode2仅支持NOR
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需要通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 7 mode2 时序寄存器配置

XMC_BK1TMGx	名称	配置	含义
ASYNCM	异步访问模式选择	0	选择modeA/B/C/D。 mode1/2时，需保持默认值0。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图10，11
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图10，11

表中未列出的bit请保持默认值。

3.1.3.1.3 modeA

图 12 modeA 读时序

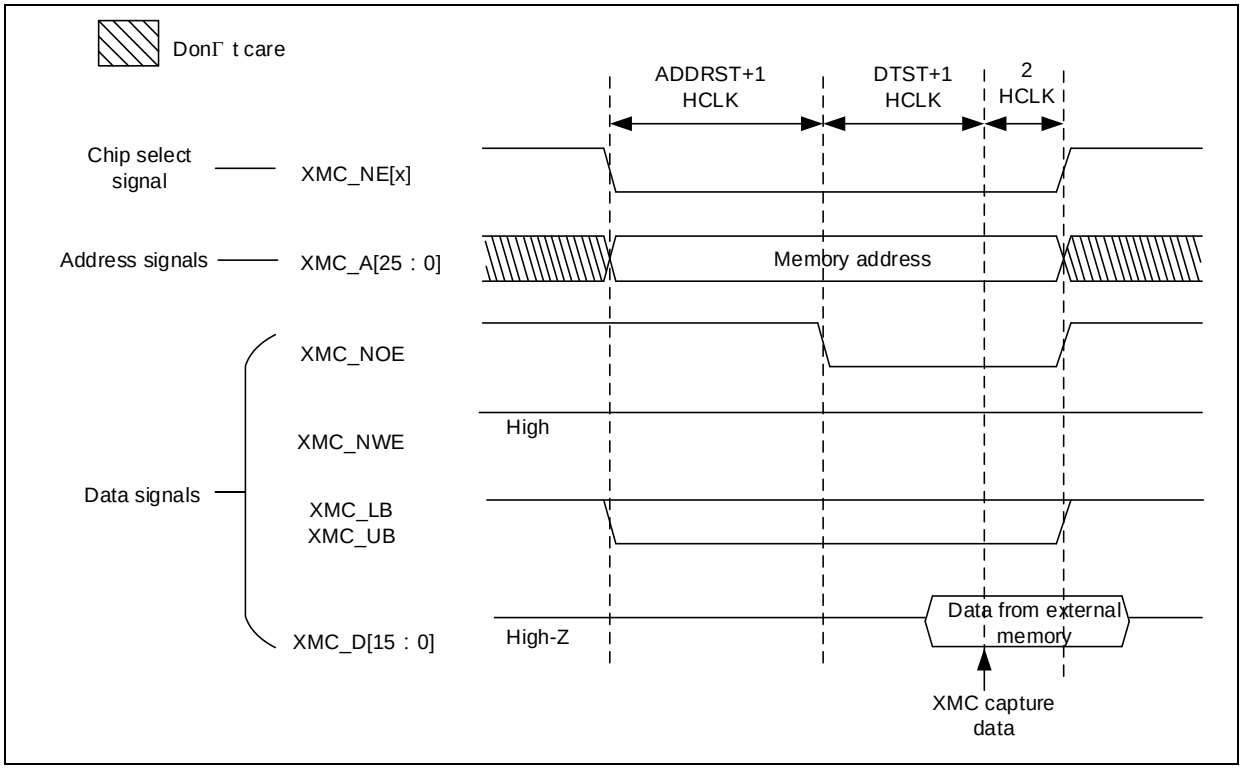


图 13 modeA 写时序

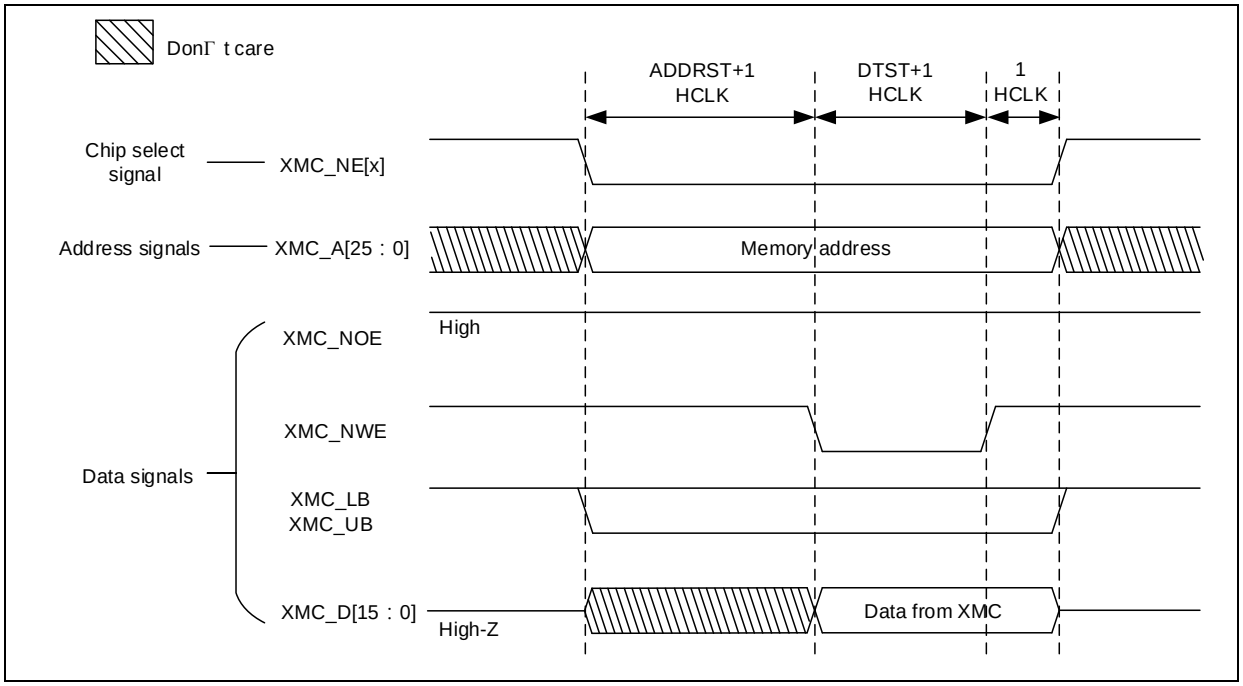


表 8 modeA 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义

RWTD	读写时序异同	1	读写时序不同 （读时序通过XMC_BK1TMGx配置， 写时序通过XMC_BK1TMGWRx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	00：SRAM 01：PSRAM	根据实际使用的存储器类型选择。 此处可选SRAM/PSRAM。
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 9 modeA 时序寄存器/写时序寄存器配置

XMC_BK1TMGx / XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	00：modeA	选择modeA/B/C/D。 此处选择modeA。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图12，13
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图12，13

表中未列出的bit请保持默认值。

3.1.3.1.4 modeB

图 14 modeB 读时序

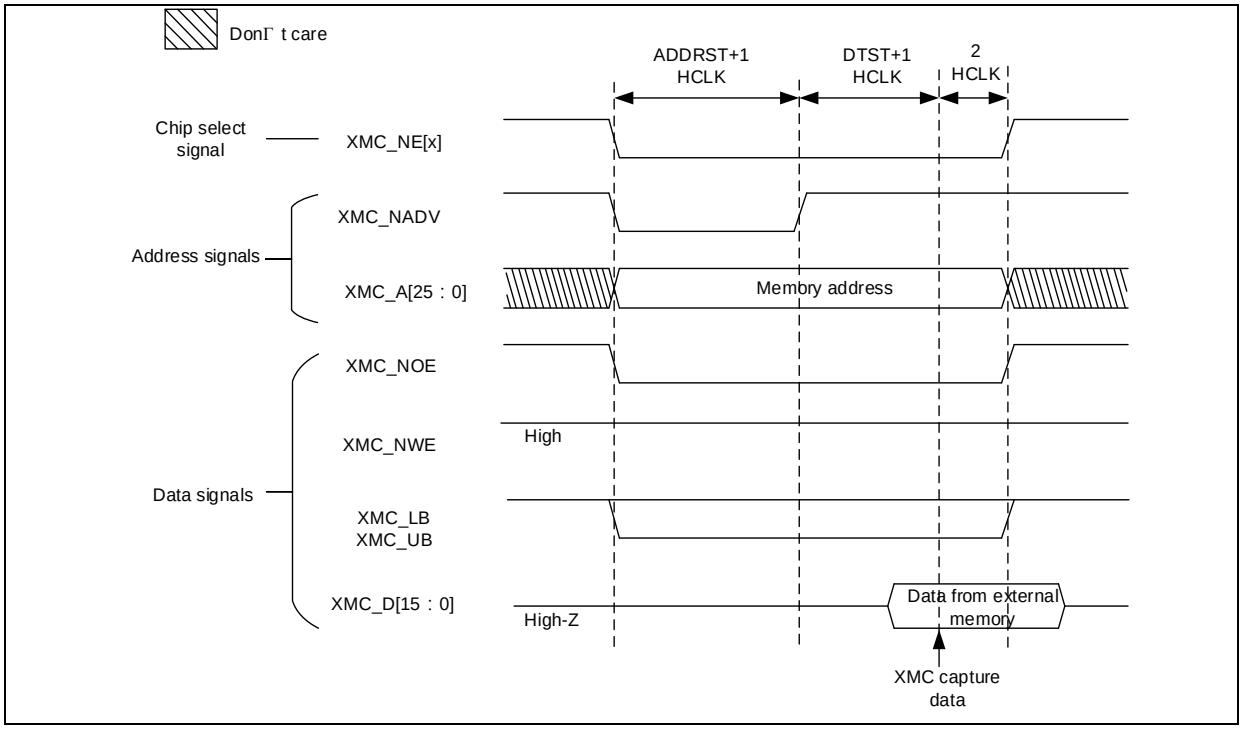


图 15 modeB 写时序

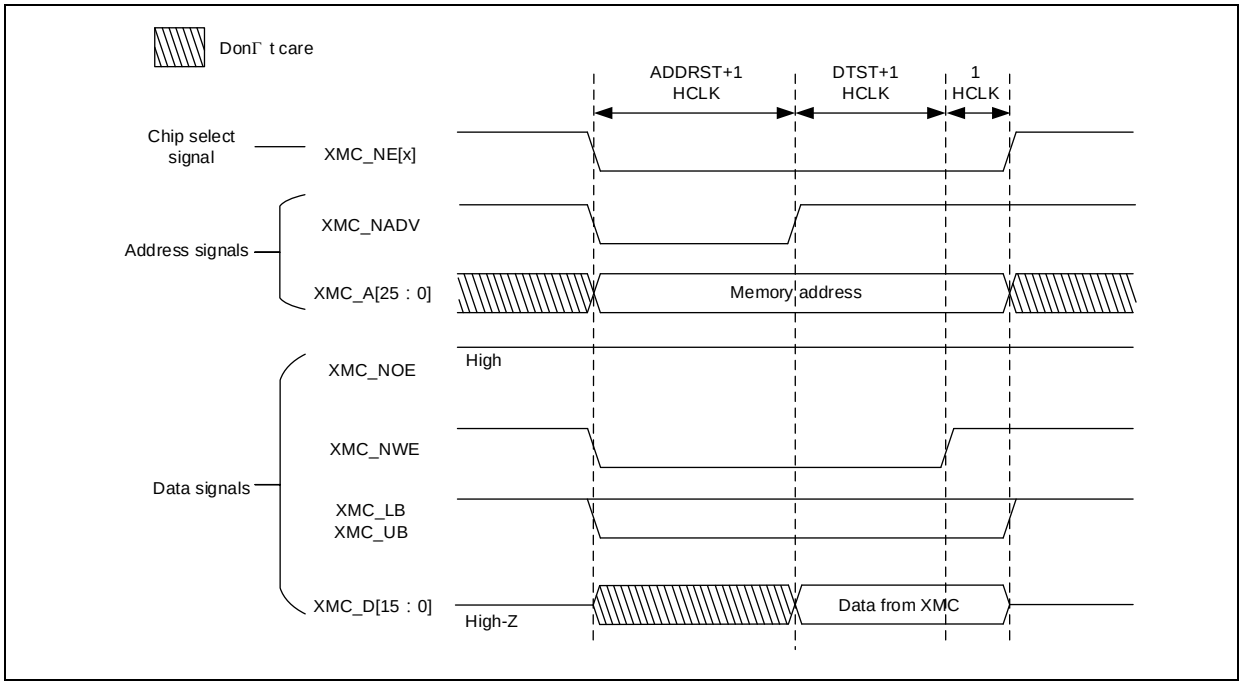


表 10 modeB 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义
RWTD	读写时序异同	1	读写时序不同 （读时序通过XMC_BK1TMGx配置， 写时序通过XMC_BK1TMGWRx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NOREN	NOR闪存访问使能	1	使用NOR时，需将此位置1
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	10：NOR	modeB仅支持NOR
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 11 modeB 时序寄存器/写时序寄存器配置

XMC_BK1TMGx/ XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	01：modeB	选择modeA/B/C/D。 此处选择modeB。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图14，15

ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图14，15

表中未列出的bit请保持默认值。

3.1.3.1.5 modeC

图 16 modeC 读时序

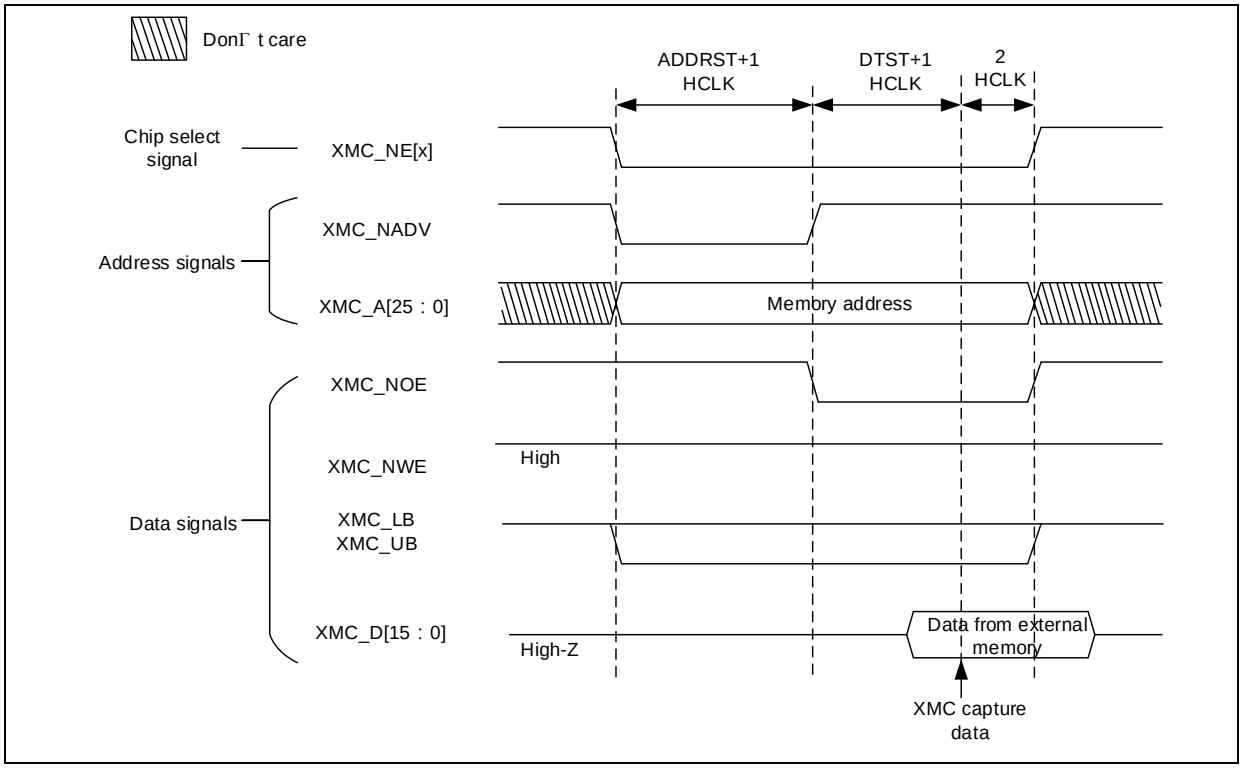


图 17 modeC 写时序

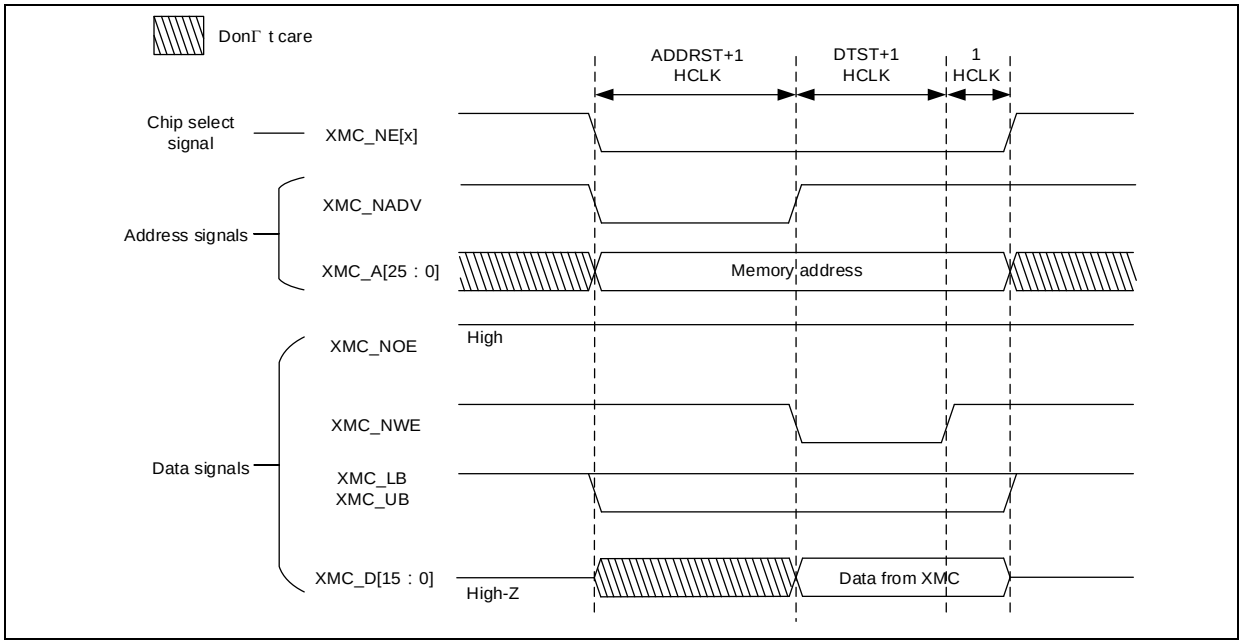


表 12 modeC 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义

RWTD	读写时序异同	1	读写时序不同 （读时序通过XMC_BK1TMGx配置， 写时序通过XMC_BK1TMGWRx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NOREN	NOR闪存访问使能	1	使用NOR时，需将此位置1
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	10：NOR	modeC仅支持NOR
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 13 modeC 时序寄存器/写时序寄存器配置

XMC_BK1TMGx/ XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	10：modeC	选择modeA/B/C/D。 此处选择modeC。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图16，17
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图16，17

表中未列出的bit请保持默认值。

3.1.3.1.6 modeD

图 18 modeD 读时序

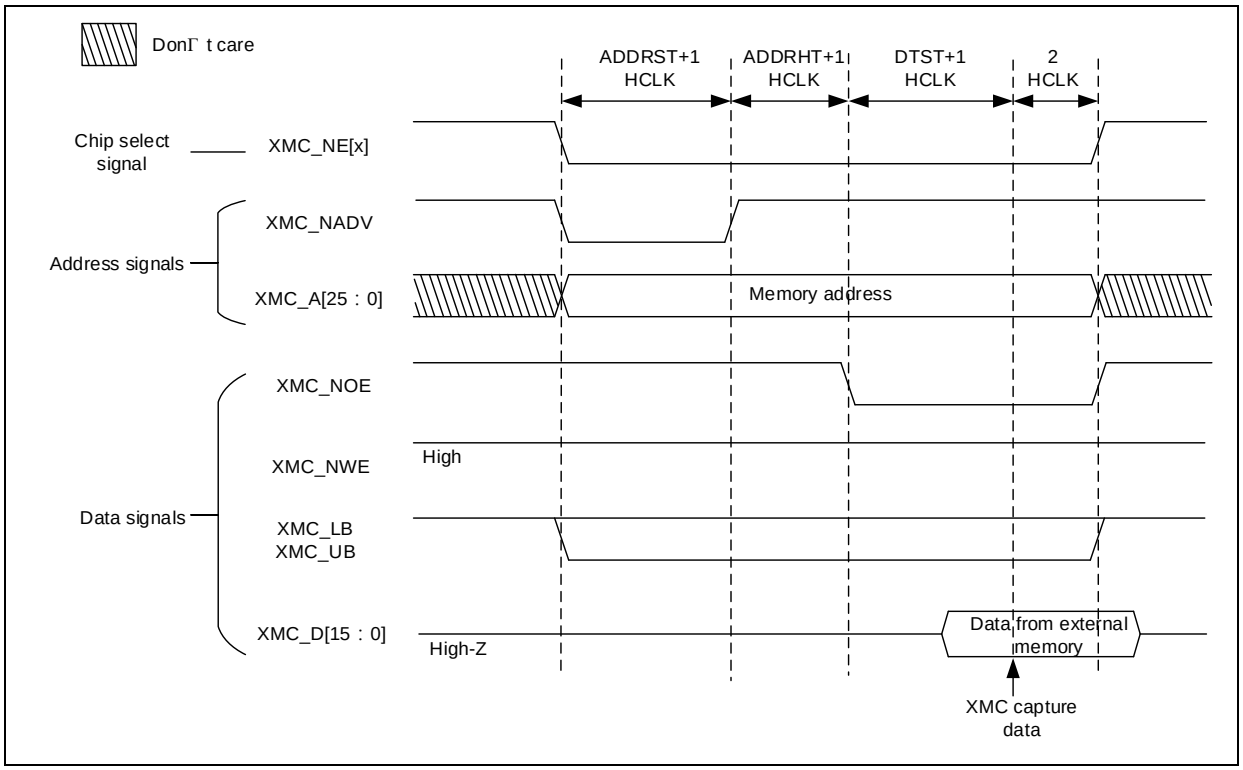


图 19 modeD 写时序

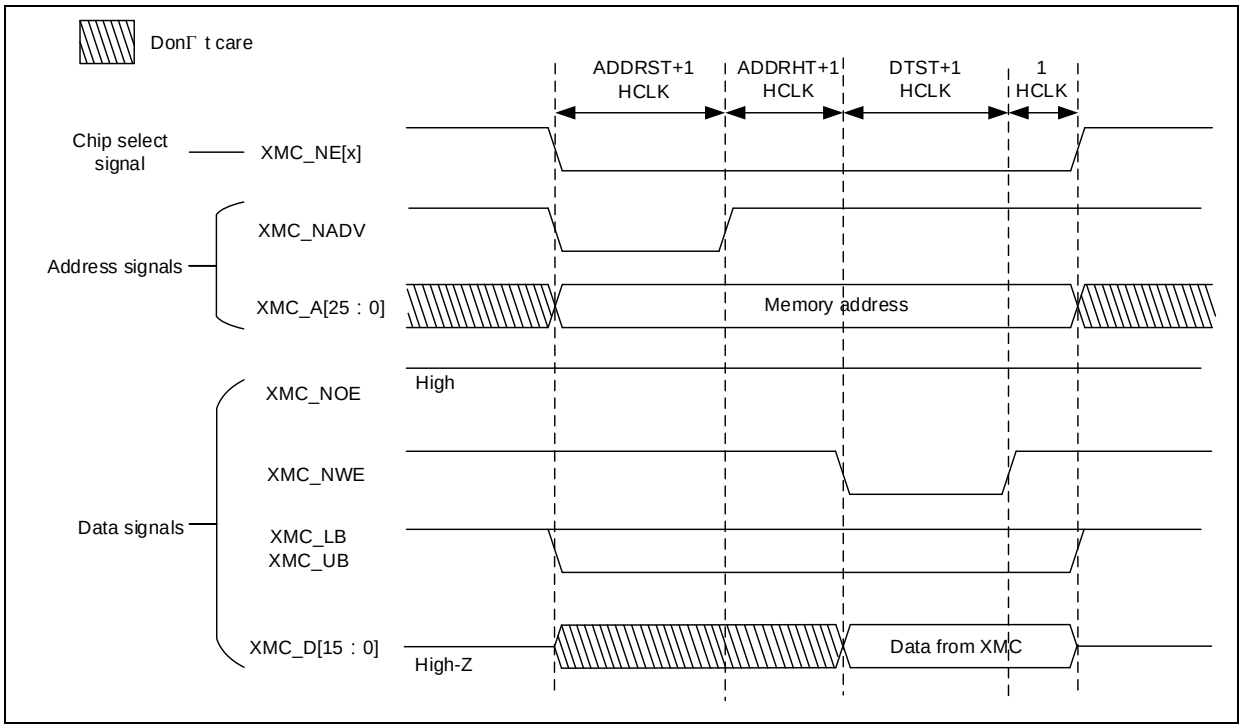


表 14 modeD 控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义

RWTD	读写时序异同	1	读写时序不同 （读时序通过XMC_BK1TMGx配置， 写时序通过XMC_BK1TMGWRx配置）
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NOREN	NOR闪存访问使能	根据存储器规格配置	使用NOR时，需将此位置1; 使用SRAM/PSRAM时，将此位清0.
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	00：SRAM 01：PSRAM 10：NOR	modeD支持SRAM/PSRAM/NOR。 根据实际使用的存储器选择。
ADMUXEN	复用使能	0	是否地址数据线复用。此处不使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 15 modeD 时序寄存器/写时序寄存器配置

XMC_BK1TMGx/ XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	11：modeD	选择modeA/B/C/D。 此处选择modeD。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图18，19
ADDRHT[3:0]	地址保持时间	根据需求与存储器规格配置	参考上图18，19
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图18，19

表中未列出的bit请保持默认值。

3.1.3.2 异步复用模式

图 20 异步复用读时序

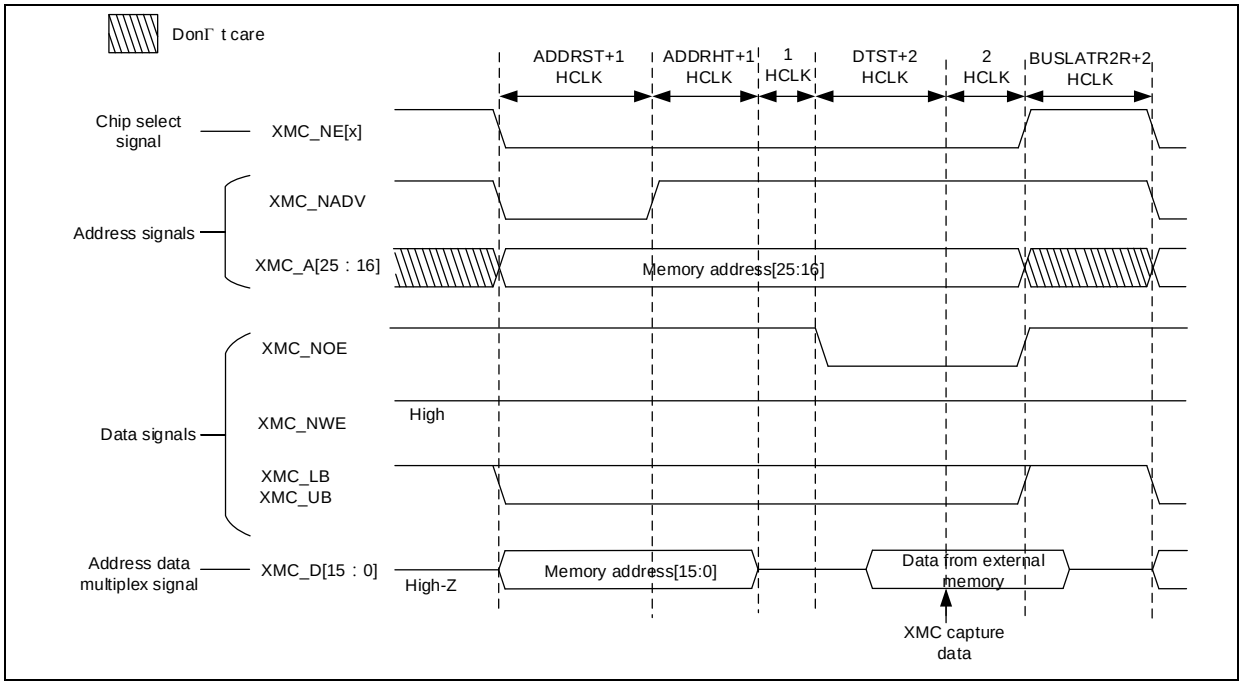


图 21 异步复用写时序

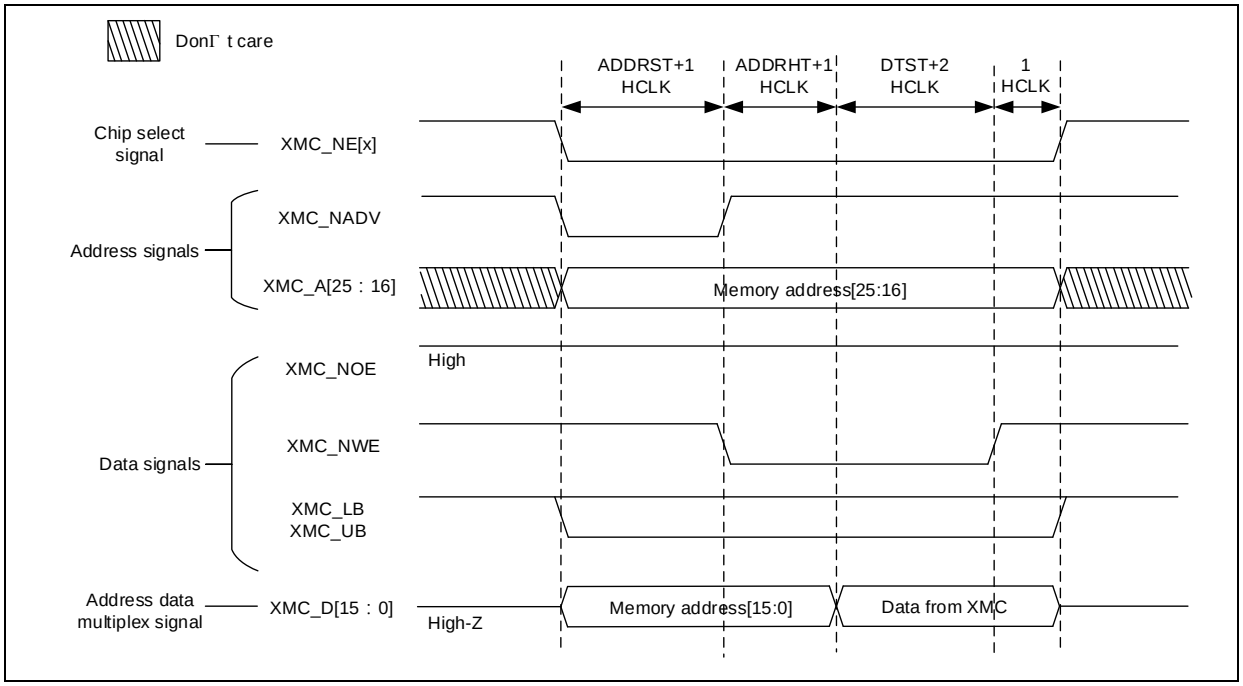


表 16 异步复用模式控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义
RWTD	读写时序异同	0	读写时序异同。 此处需配置为相同。
NWASEN	异步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0

NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NOREN	NOR闪存访问使能	根据存储器规格配置	使用NOR时，需将此位置1; 使用SRAM/PSRAM时，将此位清0.
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	01：PSRAM 10：NOR	复用模式支持PSRAM/NOR。 根据实际使用的存储器选择。
ADMUXEN	复用使能	1	是否地址数据线复用。此处需使能。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 17 异步复用模式时序寄存器/写时序寄存器配置

XMC_BK1TMGx/ XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	00	选择modeA/B/C/D。 此处需保持默认值00。
DTST[3:0]	数据建立时间	根据需求与存储器规格配置	参考上图20，21
ADDRHT[3:0]	地址保持时间	根据需求与存储器规格配置	参考上图20，21
ADDRST[3:0]	地址建立时间	根据需求与存储器规格配置	参考上图20，21

表中未列出的bit请保持默认值。

表 18 异步复用模式额外时序寄存器配置

XMC_EXTx	名称	配置	含义
BUSLATR2R[7:0]	连续读操作恢复时间	00	连续读操作间总线恢复时间为 BUSLATR2R[7:0]+1个HCLK。 参考上图20。

表中未列出的bit请保持默认值。

3.1.3.3 同步复用模式

图 22 同步复用读时序

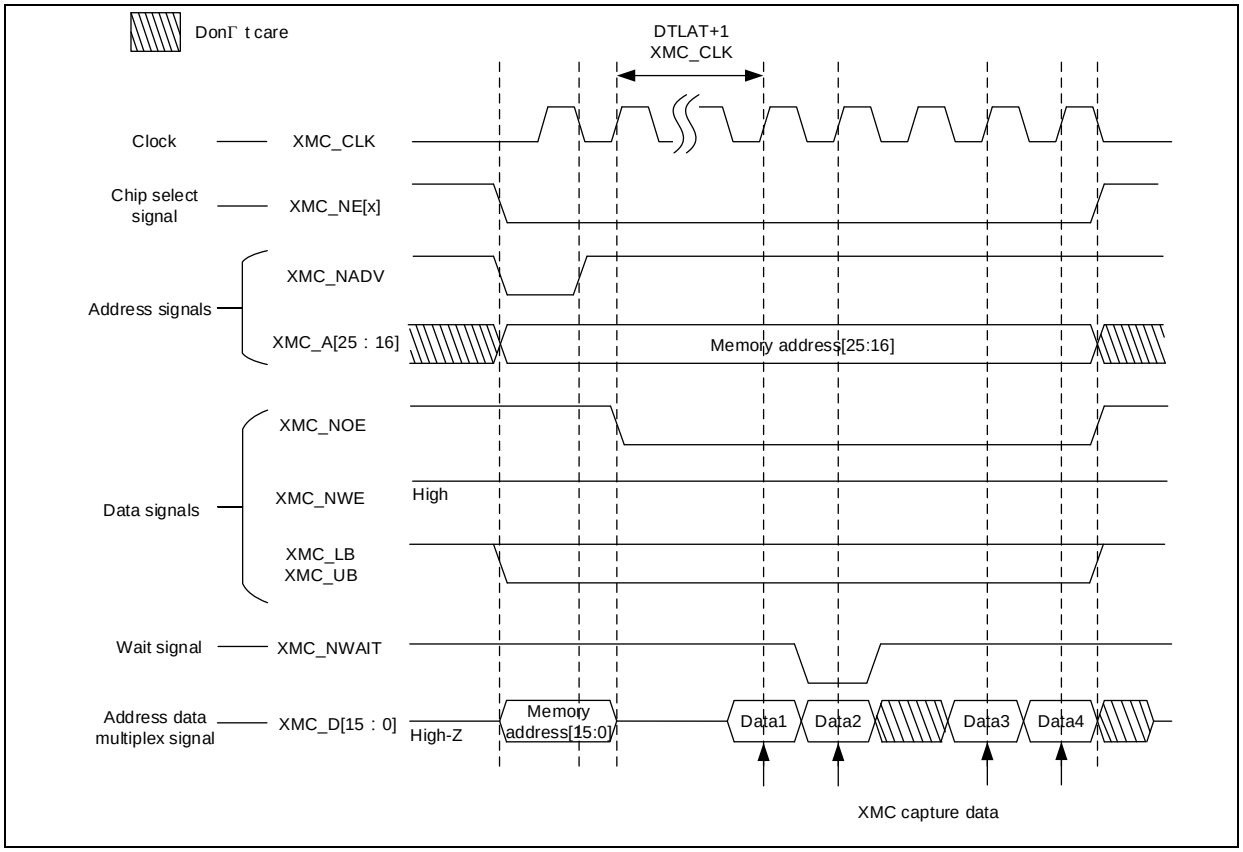


图 23 同步复用写时序

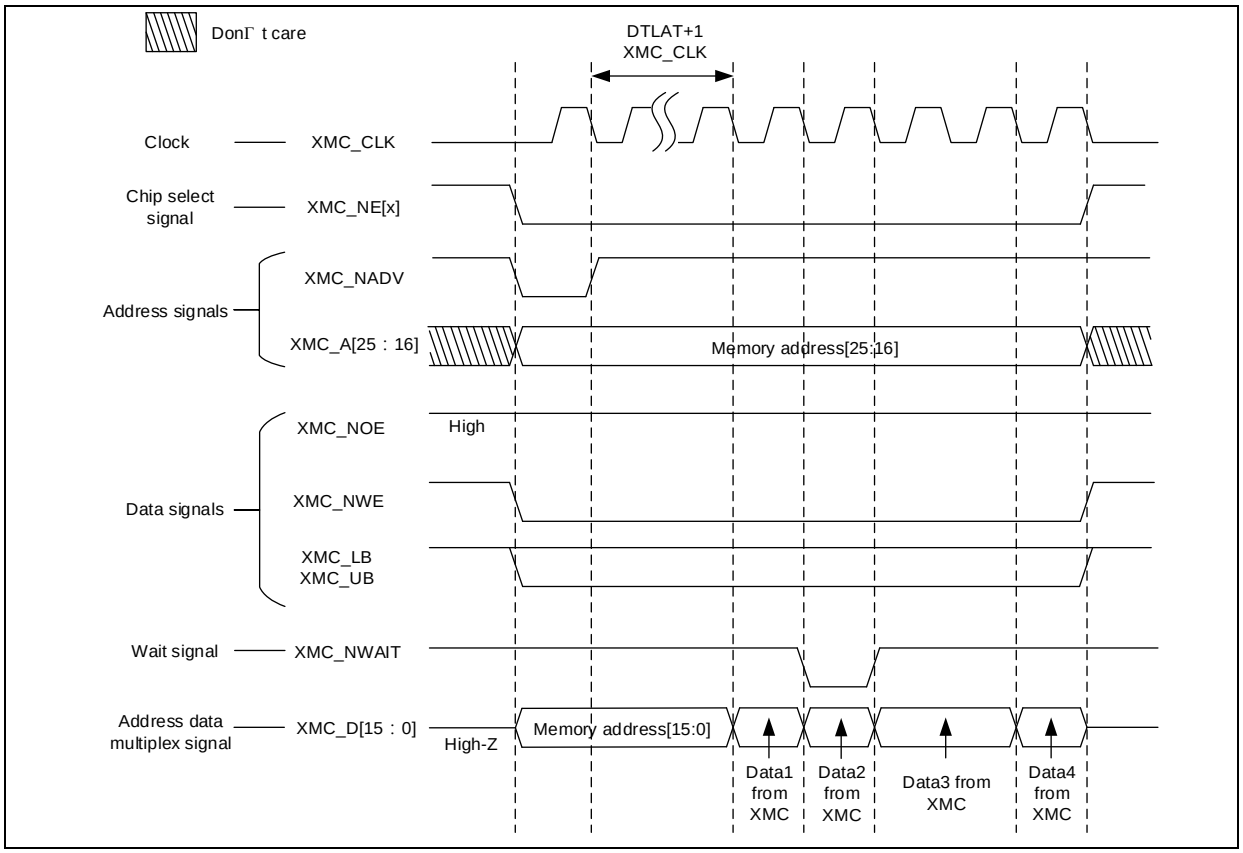


表 19 同步复用模式控制寄存器配置

XMC_BK1CTRLx	名称	配置	含义
MWMC	写操作同步使能	1	和写使能类似，写同步模式也有单独的控制位。 此处配置为1，开启写同步功能。 注：只有PSRAM支持同步写操作，NOR不支持。
CRPGS	CRAM页大小选择	根据存储器规格配置	同步模式配置这些位时，遇到跨页时，XMC会自动拆开访问。
RWTD	读写时序异同	0	读写时序异同。 此处需配置为相同。
NWSEN	同步等待信号使能	根据存储器规格配置	使用有等待信号的存储器时，此位写1； 使用没有等待信号的存储器时，此位写0
NWPOL	等待信号极性	根据存储器规格配置	和使用存储器的等待信号极性保持一致
WEN	写使能	1	可以进行写操作
NWTCFG	等待时序配置	根据存储器规格配置	0：等待信号在等待状态前的一个数据周期有效； 1：等待信号在等待状态期间有效。 上图22，23以NWTCFG=0为例。
WRAPEN	支持非对齐的成组模式	根据需求配置	同步模式时是否支持将非对齐的AHB成组操作拆成2次操作。
SYNCBEN	同步突发模式使能	1	同步模式使能。使能此位后，仅默认开启了读同步，如果需要开启写同步，还需将MWMC位置1。 注：PSRAM和NOR均支持同步读操作。
NOREN	NOR闪存访问使能	根据存储器规格配置	使用NOR时，需将此位置1; 使用PSRAM时，将此位清0.
EXTMDBW[1:0]	存储器宽度	00：8bit 01：16bit	根据实际使用的存储器可配置为8/16bit。
DEV[1:0]	存储器类型	01：PSRAM 10：NOR	选择存储器类型。
ADMUXEN	复用使能	根据存储器规格配置	是否地址数据线复用。 上图22，23以ADMUXEN =1为例。
EN	存储器块使能	1	使能存储器：使能存储器后默认开启了读使能，但写使能需通过对WEN写1开启。

表中未列出的bit请保持默认值。

表 20 同步复用模式时序寄存器/写时序寄存器配置

XMC_BK1TMGx/ XMC_BK1TMGWRx	名称	配置	含义
ASYNCM	异步访问模式选择	00	选择modeA/B/C/D。 此处需保持默认值00。
DTLAT [3:0]	数据延迟时间	根据需求与存储器规格配置	参考上图22，23
CLKPSC [3:0]	时钟分频系数	根据需求与存储器规格配置	XMC_CLK周期为HCLK周期*（CLKPSC+1）。

表中未列出的bit请保持默认值。

3.2 NAND FLASH 界面

3.2.1 引脚定义

NAND FLASH 界面典型的引脚定义如下表 21：

XMC 地址线 A[17]用作地址锁存信号 ALE；XMC 地址线 A[16]用作地址锁存信号 CLE。

① 当 ALE 为高电平时，表示 XMC_D 上传输的是地址；

② 当 CLE 为高电平时，表示 XMC_D 上传输的是命令；

③ 当 ALE 和 CLE 均为低电平时，表示 XMC_D 上传输的是数据。

表 21 NAND 引脚定义

引脚信号	方向	含义
XMC_NCE[x]	out	片选信号（x=2,3）
XMC_A[17]	out	地址锁存（ALE）信号
XMC_A[16]	out	命令锁存（CLE）信号
XMC_NOE	out	输出使能/读使能
XMC_NWE	out	写使能
XMC_D[15：0]	in/out	数据总线
XMC_NWAIT、XMC_INT[x], x=2,3	in	就绪/忙碌（R/B）信号

3.2.2 读/写时序

图 24 NAND FLASH 读/写时序

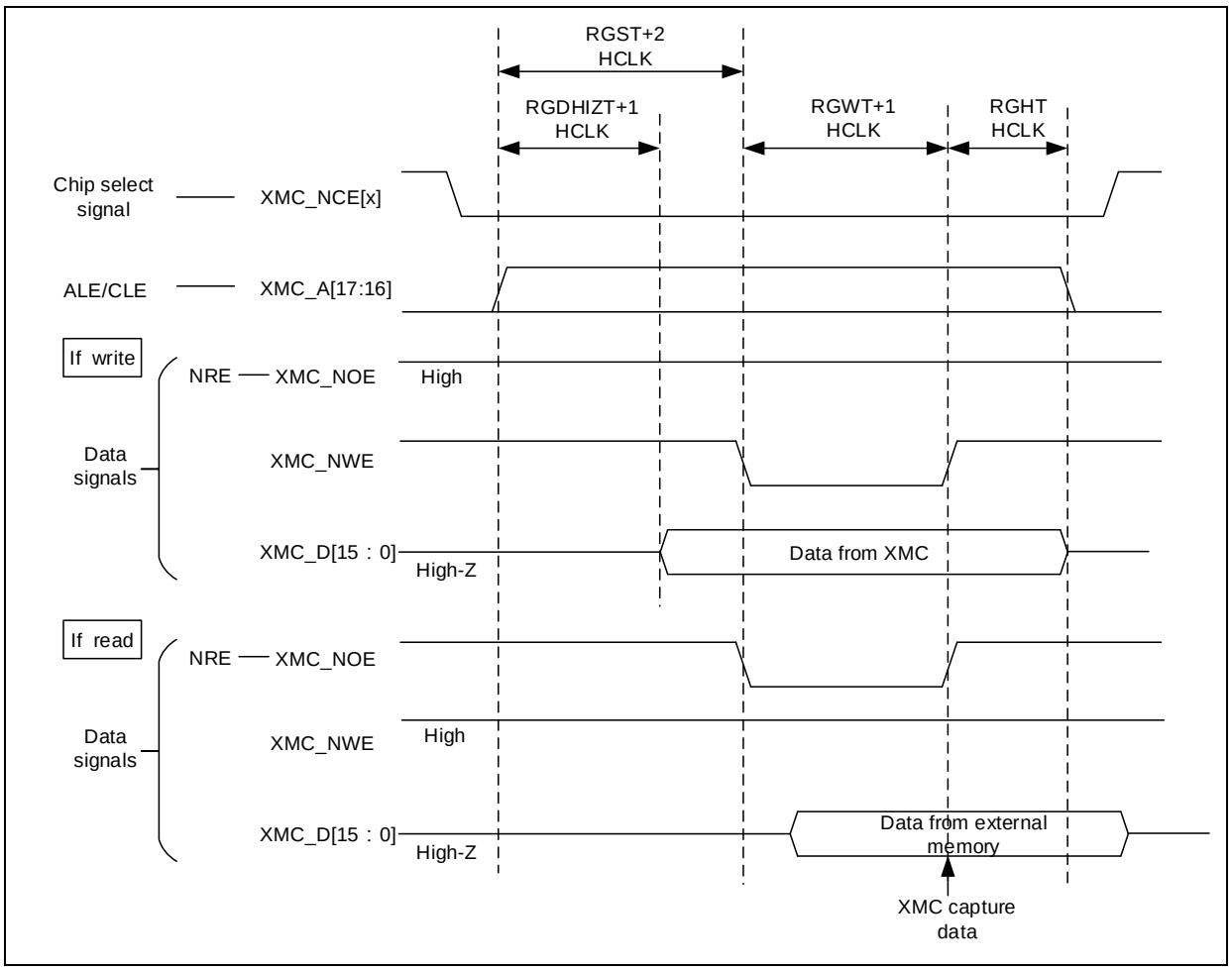


表 22 mode1 时序寄存器配置

XMC_BKxTMGRG/ XMC_BKxTMGSP	名称	访问模式	含义
RGDHIZT/SPDHIZT	存储器数据总线高阻时间	写	参考上图24
RGST/SPST	存储器建立时间	读写	参考上图24
RGWT/SPWT	存储器等待时间	读写	参考上图24
RGHT/SPHT	存储器保持时间	读写	参考上图24

表中未列出的bit请保持默认值。

3.2.3 NAND FLASH 存储器介绍

本节内容以 8bit NAND FLASH H27U1G8F2CTR 为例。具体代码请参考“案例 4 NAND FLASH 读写”。

3.2.3.1 存储结构

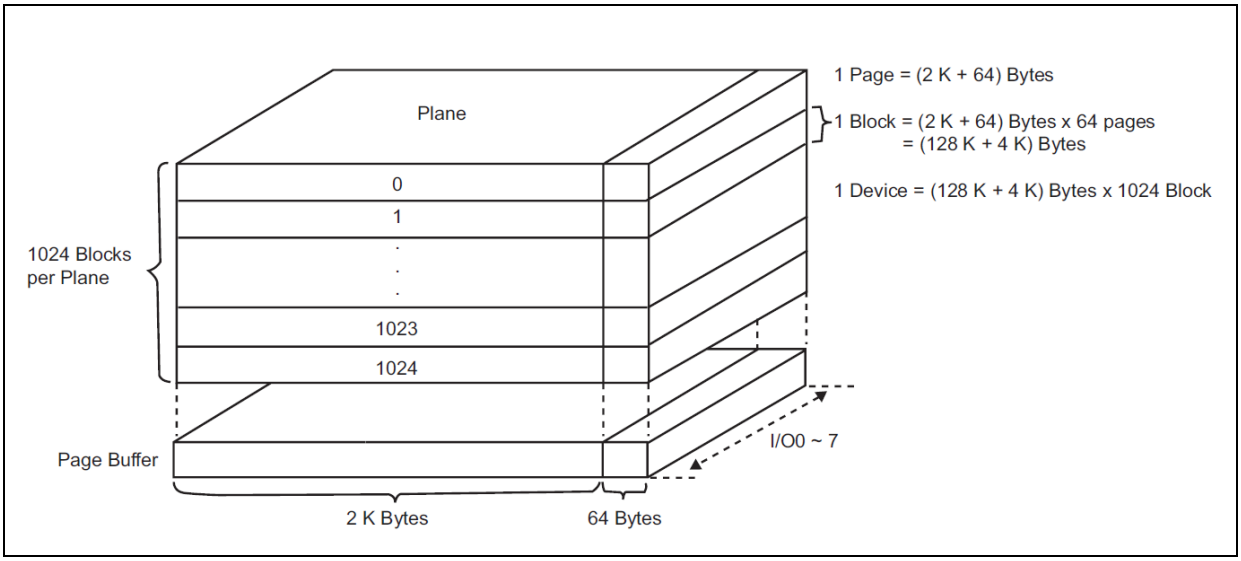
NAND FLASH 的存储结构分为几个层级：Byte -->Page--> Block--> Device。
如下图 25：

1 Page = (2 K + 64) Bytes。其中，前 2K Bytes 为数据区域，用于存放数据；后 64byte 为 spare 区域，用于存放坏块信息，ECC 码等信息。

1 Block = 64 Page。

1 Device = 1024 Block。

图 25 H27U1G8F2CTR 存储结构



3.2.3.2 地址周期

NAND FLASH 读写地址分为行地址和列地址。

列地址：也就是 Byte 地址，如下图 26 的 1st & 2nd cycle。对于 H27U1G8F2CTR 而言，1Page=(2K+64)Bytes，因此 A0~A11 有效。2nd cycle 的后 4bit 必须保持为 0。

行地址：也就是 Page 地址加 Block 地址，如下图 26 的 3rd to 4th cycle。对于 H27U1G8F2CTR 而言，1Block = 64Page，因此 A12~A17 为 Page 地址；1 Device = 1024Block，因此 A18~A27 为 Block 地址。

图 26 H27U1G8F2CTR 地址周期

	IO0	IO1	IO2	IO3	IO4	IO5	IO6	IO7
1st Cycle	A0	A1	A2	A3	A4	A5	A6	A7
2nd Cycle	A8	A9	A10	A11	L(1)	L(1)	L(1)	L(1)
3rd Cycle	A12	A13	A14	A15	A16	A17	A18	A19
4th Cycle	A20	A21	A22	A23	A24	A25	A26	A27

3.2.3.3 常用操作

NAND FLASH 读写只有异步复用模式，没有地址线。读/写均需要特定命令。写入数据前需要先擦除需要写入的 block。

注：擦除最小单位为 block。

H27U1G8F2CTR 常用操作命令如下图 27。

图 27 H27U1G8F2CTR 读/写命令

FUNCTION	1st	2nd	3rd	4th	Acceptable Command During Busy
PAGE READ	00h	30h	-	-	
READ FOR COPY-BACK	00h	35h	-	-	
READ ID	90h	-	-	-	
RESET	FFh	-	-	-	Yes
PAGE PROGRAM	80h	10h	-	-	
COPY BACK PGM	85h	10h	-	-	
BLOCK ERASE	60h	D0h	-	-	
READ STATUS REGISTER	70h	-	-	-	Yes
RANDOM DATA INPUT	85h	-	-	-	
RANDOM DATA OUTPUT	05h	E0h	-	-	
CACHE READ START	31h		-	-	
CACHE READ EXIT	3Fh	-	-	-	

3.2.3.3.1 读操作

读操作如下图 28：

① 写入“PAGE READ”命令的 1st：即写入“0x00”，命令锁存（CLE）输出高电平；

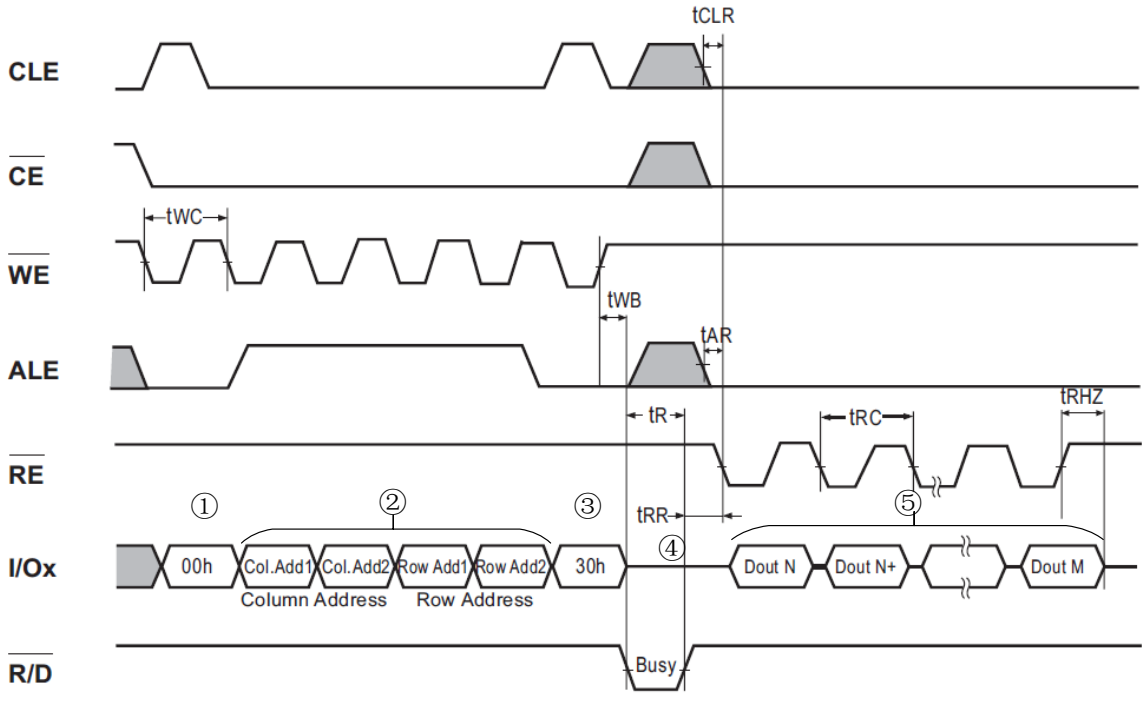
② 写需要读取的地址：分 4 次写入地址，即上图 26 的“1st Cycle”，“2nd Cycle”，“3rd Cycle”，“4th Cycle，地址锁存（ALE）输出高电平；

③ 写入“PAGE READ”命令的 2nd：即写入“0x30”，命令锁存（CLE）输出高电平；

④ 等待外部 NAND FLASH 准备就绪：等待 R/D 信号高电平；

⑤ 读取数据：此时命令锁存（CLE）和地址锁存（ALE）均为低电平。

图 28 H27U1G8F2CTR 读操作时序



3.2.3.3.2 擦除操作

擦除操作如下图 29：

① 写入“BLOCK ERASE”命令的 1st：即写入“0x60”，命令锁存（CLE）输出高电平；

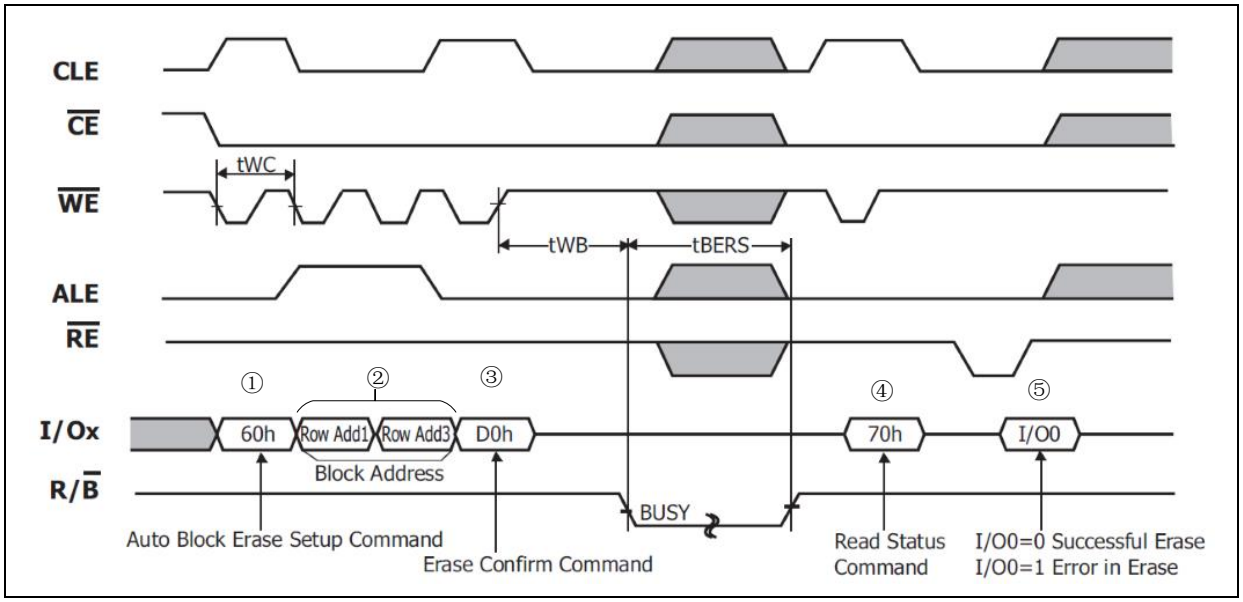
② 写需要擦除的 BLOCK 地址：擦除是以 BLOCK 为单位，因此只需要 BLOCK 地址。也就是只需要发送行地址，即上图 26 的“3rd Cycle”，“4th Cycle，其中 A18~A27（BLOCK 地址）有效，A12~A17（PAGE 地址）被 NAND FLASH 忽略。地址锁存（ALE）输出高电平。

③ 写入“BLOCK ERASE”命令的 2nd：即写入“0xD0”，命令锁存（CLE）输出高电平；

④ 读取 H27U1G8F2CTR 的状态寄存器，以判断擦除操作是否完成：即写入上图 27 的“READ STATUS REGISTER”命令“0x70”。命令锁存（CLE）输出高电平。

⑤ 等待擦除操作完成：重复④操作，直到读取的状态寄存器 bit0 为 0，表示擦除操作完成。

图 29 H27U1G8F2CTR 擦除操作时序



3.2.3.3.3 写操作

写操作如下图 30：

① 写入“PAGE PROGRAM”命令的 1st：即写入“0x80”，命令锁存（CLE）输出高电平；

② 写入需要写数据的地址：分 4 次写入地址，即上图 25 的“1st Cycle”，“2nd Cycle”，“3rd Cycle”，“4th Cycle，地址锁存（ALE）输出高电平；

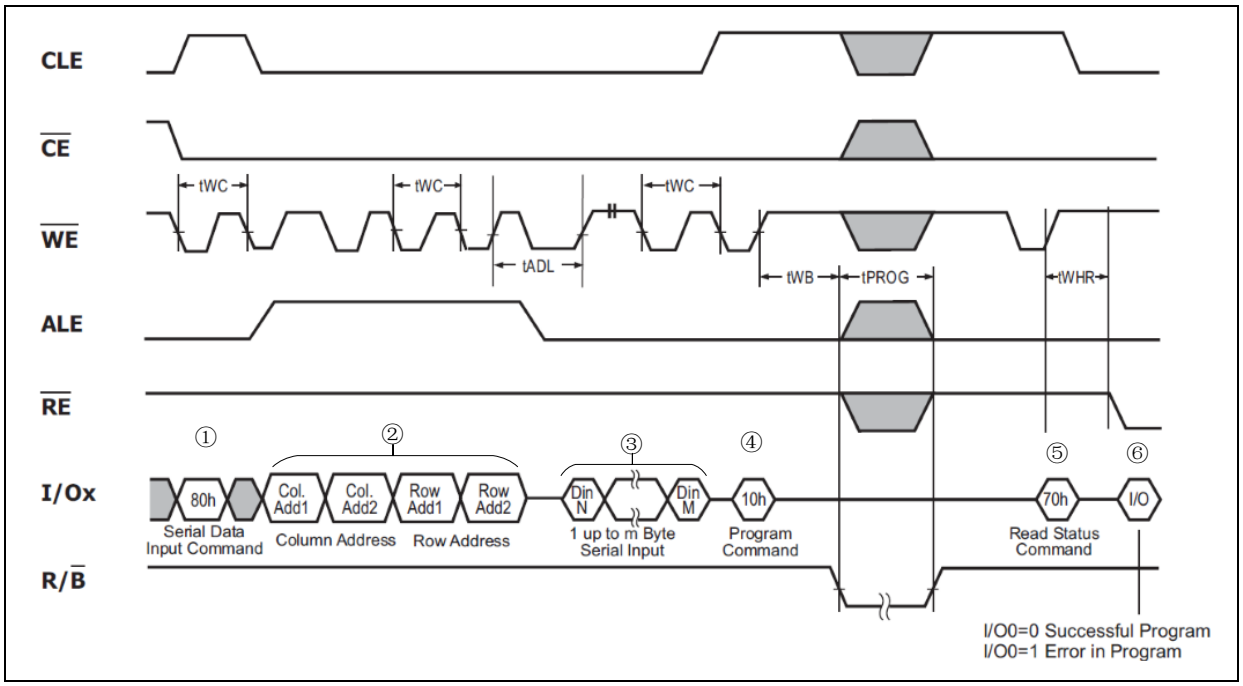
③ 写入待写入的数据：即下图 30 的 DinN~DinM。此时，命令锁存（CLE）和地址锁存（ALE）均保持低电平；

④ 写入“PAGE PROGRAM”命令的 2nd：即写入“0x10”，命令锁存（CLE）输出高电平

⑤ 读取 H27U1G8F2CTR 的状态寄存器，以判断数据写操作是否完成：即写入上图 25 的“READ STATUS REGISTER”命令“0x70”。命令锁存（CLE）输出高电平。

⑥ 等待数据写操作完成：重复④操作，直到读取的状态寄存器 bit0 为 0，表示数据写操作完成。

图 30 H27U1G8F2CTR 写操作时序



3.2.4 ECC 功能

NAND 界面包含了 ECC 运算模块，可在 NAND 界面读/写 NAND FLASH 时，对数据进行 ECC 的运算，计算得到 ECC 码存入 XMC_BK2ECC 寄存器。一次 ECC 计算可以矫正 1bit 的错误，可以检测到 2bit 的错误，更多数量的错误不保证能检测到。

用户可以在对 NAND FLASH 写数据之后把 XMC_BK2ECC 寄存器的 ECC 码写入 spare 区域；后续读取 NAND FLASH 时，将 spare 区域读出的 ECC 码与 XMC_BK2ECC 寄存器的 ECC 码进行比较和运行纠正算法，即可提高数据可靠性。

使用步骤：

① 配置 ECC page 大小 ECCPGS 以选择每次计算 ECC 的字节数目：256、512、1024、2048、4096 或 8192 个字节。此处 ECCPGS 的配置应该和实际使用的 NAND FLASH page 大小一致。

② 开启 ECC 使能位 ECCEN。

③ 进行数据区的读/写。

④ XMC 在收到/送出与 ECCPGS 设定的字节数后，将 ECC 码存入 XMC_BK2ECC 寄存器。

⑤ 软件读取最后一个字节/写入最后一个字节并且等待 FIFOE 标志位被置起。

⑥ 软件读取 XMC_BK2ECC 寄存器并进行对应的错误更正流程。

⑦ 软件清除 ECCEN 位，以备下次开启 ECCEN 进行 ECC 计算。重复 2~6 的步骤。

注：步骤④为 XMC 硬件操作，无需软件代码参与。ECC 功能的详细代码请参考“案例 5 NAND FLASH ECC 纠错”一节。

另外，根据 ECCPGS 配置的不同，ECC 结果寄存器的有效位数不同，详见下表：

表 23 ECC 有效位数

ECCPGS	000	001	010	011	100	101
Page大小	256	512	1024	2048	4096	8192
ECC码有效位	ECC[21：0]	ECC[23：0]	ECC[25：0]	ECC[27：0]	ECC[29：0]	ECC[31：0]

3.2.5 常规空间和特殊空间

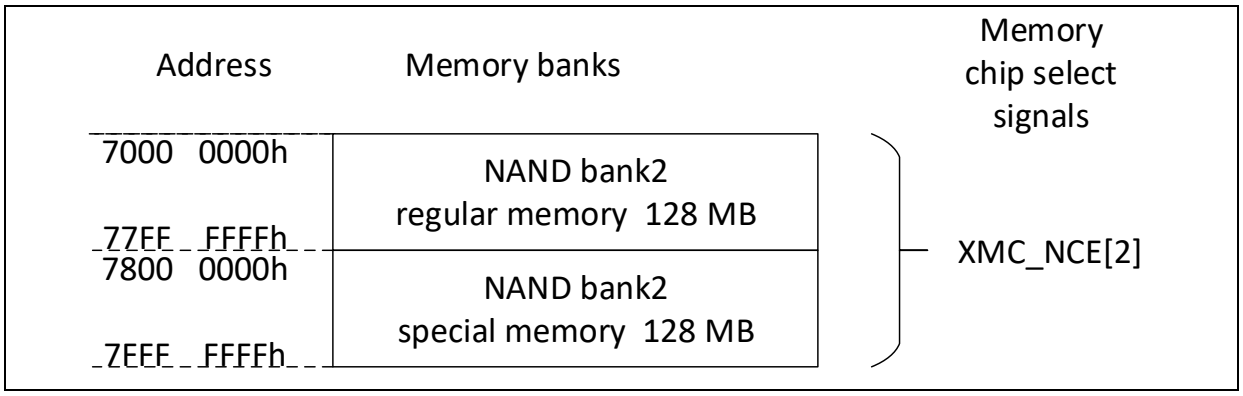
为了满足某些 NAND FLASH 在不同状态的不同时序需求，NAND 界面又分为常规空间和特殊空间。各有独立的时序寄存器：常规空间时序寄存器（XMC_BKxTMGRG）和特殊空间时序寄存器（XMC_BKxTMGSP）。

如下图 31，以 NAND bank2 为例：

需要使用常规空间时序时，对常规空间任意地址进行读/写操作即可（0x70000000~0x77FFFFFF）；

需要使用特殊空间时序时，对特殊空间任意地址进行读/写操作即可（0x78000000~0x7FFFFFFF）。

图 31 AT32F435/437 NAND bank2 地址映射



3.3 PC 卡界面

3.3.1 引脚定义

PC 界面卡典型的引脚定义如下表 24：

XMC_NCE4_2：用于指示对通用空间的操作宽度是 8bit/16bit。高电平为 8bit，低电平为 16bit。

XMC_NREG：用于指示对 I/O 空间的操作宽度是 8bit/16bit，作用类似通用空间的 XMC_NCE4_2。高电平为 8bit，低电平为 16bit。I/O 空间仅支持 16bit 操作，即 XMC_NREG 需保持低电平。

XMC_A[10:0]：地址线。此处的地址是扇区地址，而非 byte 地址。PC 卡的读/写/擦除的最小单位是扇区。

表 24 PC 卡引脚定义

引脚信号	方向	含义
XMC_NCE4_1	out	片选1（CE1）
XMC_NCE4_2	out	片选2（CE2）
XMC_A[10:0]	out	地址总线
XMC_NOE	out	通用及属性空间使用的输出使能信号
XMC_NWE	out	通用及属性空间使用的写使能信号
XMC_NIORD	out	I/O空间使用的输出使能信号
XMC_NIOWR	out	I/O空间使用的写使能信号
XMC_NREG	out	属性空间选择信号
XMC_D[15：0]	in/out	数据总线
XMC_CD	in	PC卡存在检测信号，高电平有效
XMC_NWAIT	in	就绪/忙碌（R/B）信号
XMC_INTR	in	PC卡中断信号

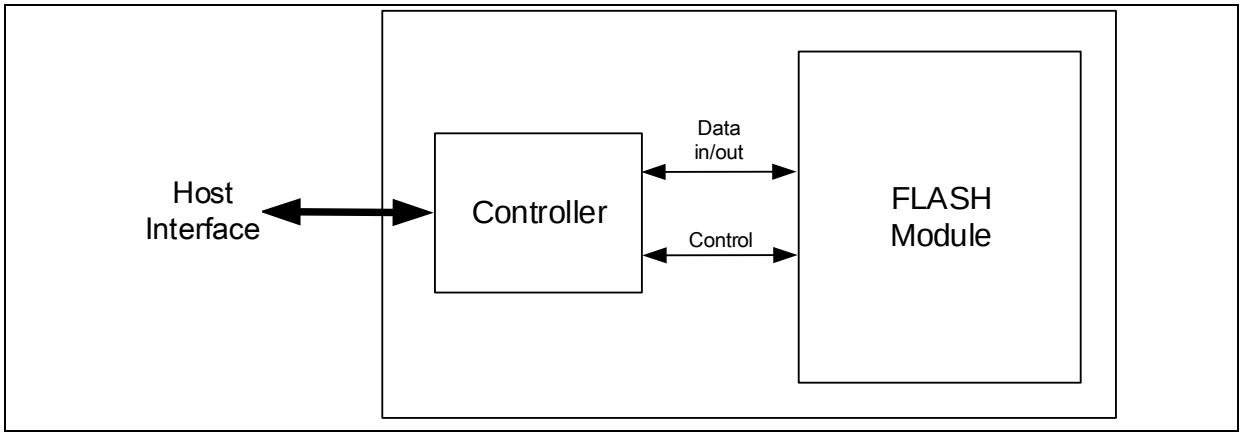
3.3.2 PC 卡/CF 卡结构介绍

如下图32，PC卡/CF卡主要分闪存模块（FLASH Module）和内部控制器（Controller）两部分。

闪存模块：通常是NAND FLASH，组织结构类似与硬盘，分柱面/扇区/磁头等。

内部控制器：XMC实际是通过内部控制器的寄存器来操作闪存模块。例如，XMC向PC卡写命令是向内部控制器的命令寄存器写命令；XMC读/写数据是从内部控制器数据寄存器读/写数据。

图 32 PC 卡/CF 卡框图



3.3.3 PC 卡读/写时序

图 33 PC Card 读/写时序

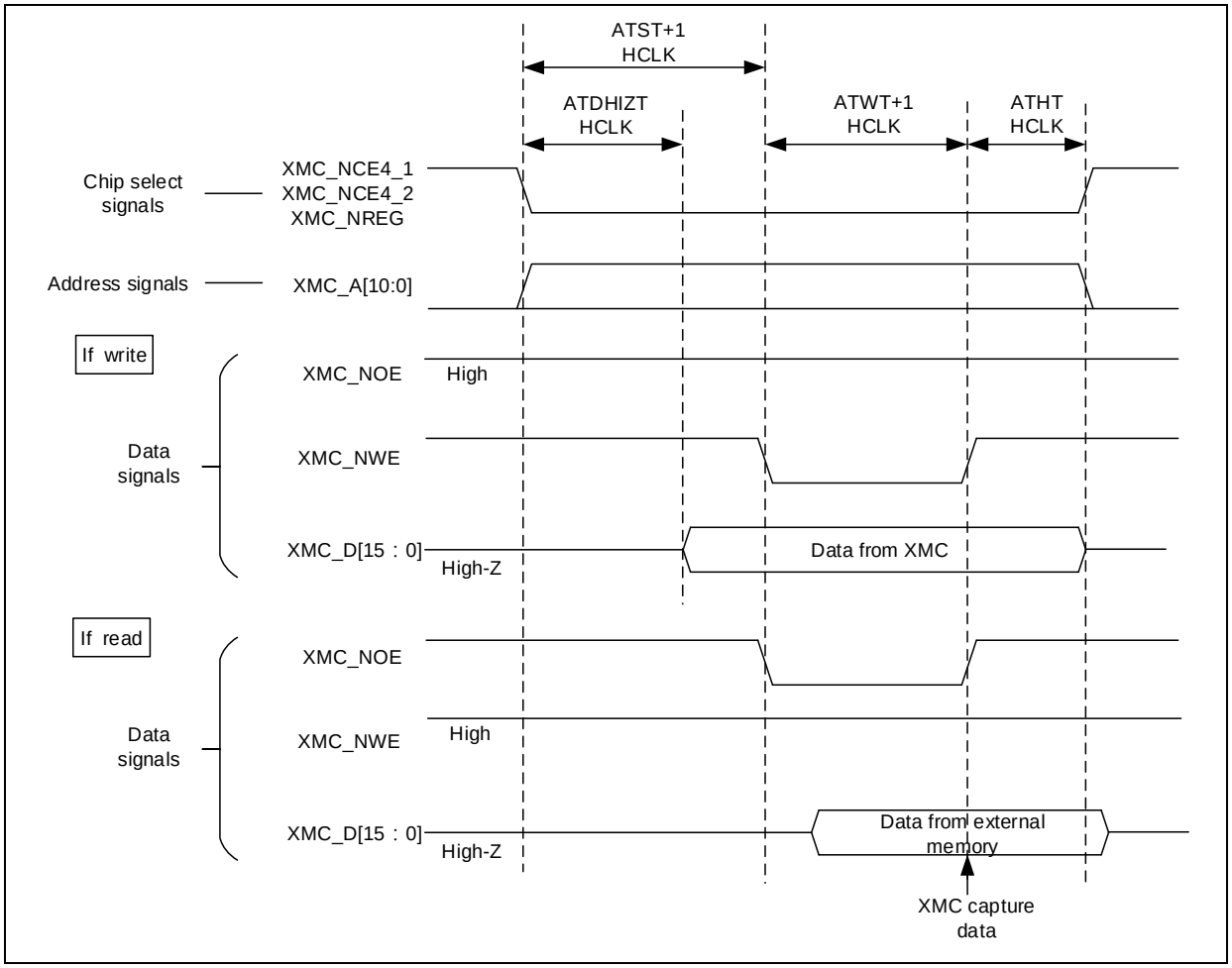


表 25 PC Card 时序寄存器配置

通用/属性/IO空间时序寄存器	名称	访问模式	含义
CMDHIZT/ATDHIZT/IOHIZT	存储器数据总线高阻时间	写	参考上图33
CMST/ATST/IOST	存储器建立时间	读写	参考上图33
CMWT/ATWT/IOWT	存储器等待时间	读写	参考上图33
CMHT/ATHT/IOHT	存储器保持时间	读写	参考上图33

表中未列出的bit请保持默认值。

3.3.4 通用空间/属性空间和 IO 空间

PC 卡/CF 卡通常包含 3 种操作模式：I/O Mapped，Memory Mapped，True IDE。AT32 的 XMC PC 卡界面支持其中的 I/O Mapped，Memory mapped 模式。

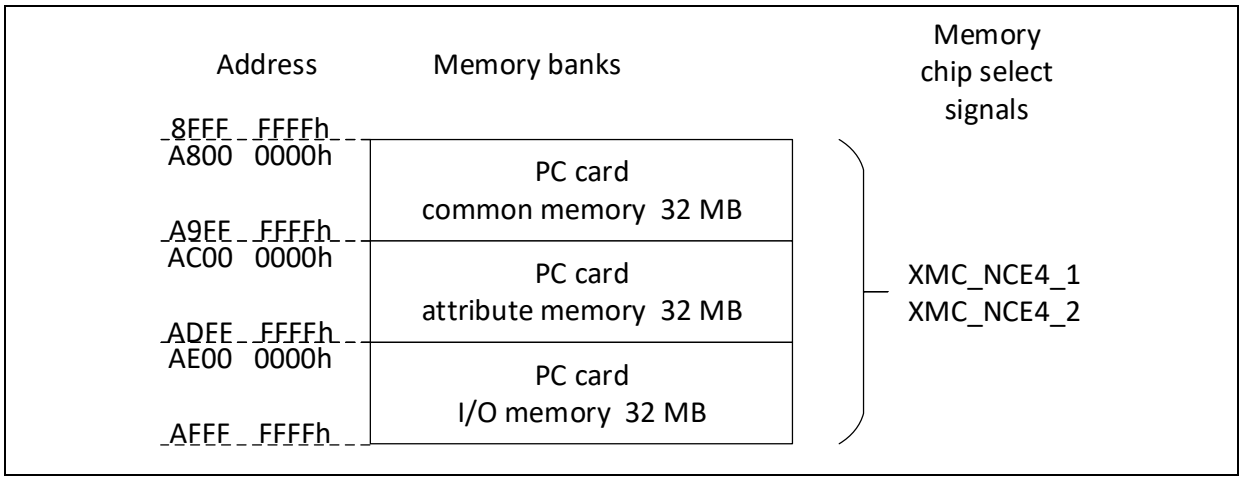
如下图 34，PC Card 界面分为 3 部分：通用空间、属性空间、I/O 空间。

通用空间：用于数据读/写，命令发送等操作。在 PC 卡的 Memory Mapped 模式下使用。

I/O 空间：用于数据读/写，命令发送等操作。在 PC 卡的 I/O 模式下使用。

属性空间：用于配置 PC 卡的模式（I/O Mapped，Memory Mapped，True IDE），reset PC 卡等操作。

图 34 AT32F435/437 PC 卡地址映射



3.4 驱动 LCD 屏

由于 LCD 屏的引脚和 XMC 接口有很大一部分的重合，因此 XMC 接口还可以用于驱动 LCD 屏。本文介绍两种 LCD 的驱动。

3.4.1 8bit LCD 屏驱动

3.4.1.1 8bit LCD 引脚定义

如下表 26，以 8bit 的 8080 接口 LCD 屏 KD024C 为例。具体代码请参考“案例 6 8bit LCD 屏驱动”。

表 26 8bit LCD 引脚定义

XMC	KD024C	含义
PB9	LCD_RESET	LCD reset信号。MCU端通过一个普通GPIO控制（本例中使用PB9）
XMC_NE1	LCD_CS	LCD片选
XMC_A0	LCD_D/C	LCD select data/command：数据/命令选择信号，用于指示数据总线上是数据还是命令。高电平表示数据和参数，低电平表示命令。 XMC的A0连接到LCD的D/C选择。 由于此处使用的是BANK1的第1个子bank，操作基地址为0x60000000。那么向LCD发送命令时，XMC写入地址应选择0x60000000；发送数据/参数时，XMC写入地址为0x60000001。
XMC_NOE	LCD_RD	LCD读使能
XMC_NWE	LCD_WR	LCD写使能
XMC_D[7：0]	DATA[7：0]	数据总线
--	IM[2:0]	接口选择：000（MCU 8bit bus interface）--直接硬件接地

3.4.1.2 8bit LCD 驱动步骤

通过 XMC 驱动 8bit LCD 的显示一张图片的步骤如下：

① XMC 初始化；

② reset LCD：PB9 输出低电平以 reset LCD，后再输出高电平退出 reset；

③ LCD 屏各项参数配置。以刷屏方向配置为例：1）刷屏方向配置命令(36h)；2）刷屏方向参数(00h)；
刷屏方向命令参考下图 35，更多命令请参考 KD024C 规格书；

④ 配置 LCD 开始显示，即 LCD 屏的 SRAM 数据即时显示到屏幕上：发送 display on 命令(29h)；

⑤ 定义并擦除需要显示的区域，如下图 36：选择阴影区域（X-START~X-END,Y-STARY~Y-END）为显示区域。本操作完成后，LCD 指针在初始位置（黑色圆点处）；

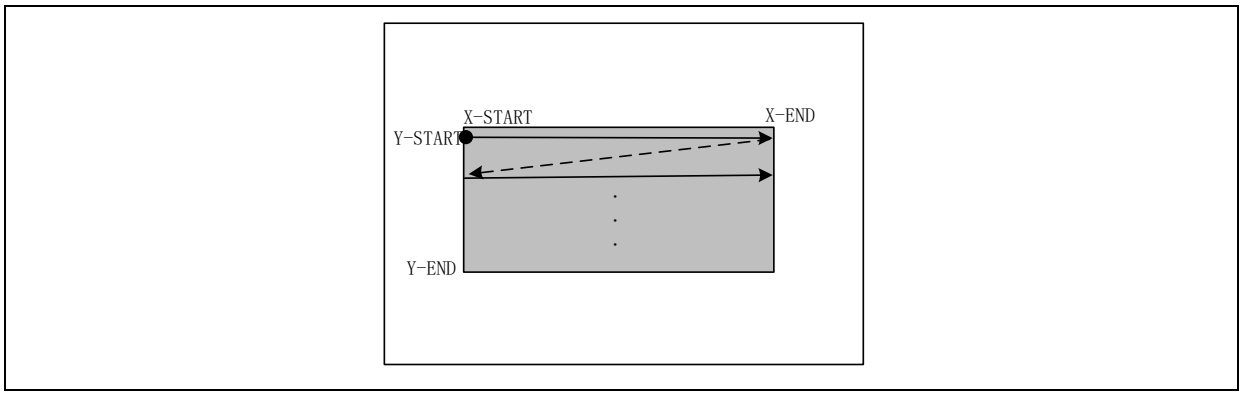
⑥ XMC 循环发送需要显示的图片数据（预先准备好的一组 buffer）：LCD 每接收到 1byte 数据，则指针加一，整个 buffer 接收完成后，即可在阴影区域显示一张完整图片；

⑦ 重复⑤⑥步骤即可刷新更换不同图片。

图 35 LCD 命令（以刷屏方向配置为例）

36h	MADCTL (Memory Access Control)												
	D/CX	RDX	WRX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Command	0	1	↑	XX	0	0	1	1	0	1	1	0	36h
Parameter	1	1	↑	XX	MY	MX	MV	ML	BGR	MH	0	0	00
	This command defines read/write scanning direction of frame memory.												

图 36 LCD 显示示意



3.4.2 16bit LCD 触摸屏驱动

3.4.2.1 16bit LCD 引脚定义

如下表 27，以带触摸功能的 16bit 的 8080 接口 LCD 屏 MRB2801 为例。具体代码请参考“案例 7 16bit LCD 触摸屏驱动”。

表 27 16bit LCD 引脚定义

MCU	MRB2801	含义
SPI2_SCK	SPI_SCK	touch模块--SPI通信接口
SPI2_MISO	SPI_MISO	
SPI2_MOSI	SPI_MOSI	
PB12	T_CS	touch模块--片选使能
PB11	T_PEN	触摸检测
PC0	LCD_BL	背光控制
XMC_NE4	LCD_CS	LCD片选
XMC_A10	LCD_RS	LCD select data/command：数据/命令选择信号，用于指示数据总线上是数据还是命令。高电平表示数据和参数，低电平表示命令。XMC的A10连接到LCD的D/C选择。 由于此处使用的是BANK1的第1个子bank，操作基地址为0x60000000。那么向LCD发送命令时，XMC写入地址应选择0x60000000；发送数据/参数时，XMC写入地址为(0x60000000 | (1<<(10+1))),即0x60000800
XMC_NOE	LCD_RD	LCD读使能
XMC_NWE	LCD_WR	LCD写使能
XMC_D[15：0]	DATA[15：0]	数据总线

3.4.2.2 16bit LCD 驱动步骤

通过 XMC 驱动 16bit LCD 触摸屏，显示绘制内容：

① XMC 初始化；

② LCD 屏各项相关参数配置。（参考 8bit LCD 操作）；

③ 打开背光：LCD_BL 输出高电平；

④ 清空屏幕：将全屏所有像素点设置为同一个颜色；

⑤ touch 接口初始化：touch 相关的 SPI 和 GPIO 初始化；

⑥ 循环扫描触摸点，一旦扫描到触摸点（T_PEN 为低电平）；

⑦ 立即在对应坐标及其上下左右坐标的几个像素点设置为另一个颜色，实现绘制。

3.5 地址映射和交换

为了使 SDRAM 和 QSPI2 也可以执行代码，AT32F435/437 的 XMC 增加了 SWAP 功能。用户可以通过设定 SWAP_XMC[1:0]（XMC 存储器地址映射交换控制位）来交换部分存储区域地址，以使 SDRAM 和 QSPI2 可以执行代码。

如下表 28，阴影部分是代码可执行区域，非阴影部分是代码不可执行区域。粗字体部分是地址交换区域。

表 28 SWAP_XMC 地址交换示意

Start Address	End Address	SWAP_XMC[1:0]			
		00	01	10	11
0x60000000	0x7FFFFFFF	XMC_MEM（NOR/PSRAM/SRAM/NAND2）	**XMC_MEM（SDRAM）**	XMC_MEM（NOR/PSRAM/SRAM/NAND2）	**XMC_MEM（SDRAM）**
0x80000000	0x8FFFFFFF	XMC_MEM（NAND3）	XMC_MEM（NAND3）	**QSPI2 MEM**	**QSPI2 MEM**
0xB0000000	0xBFFFFFFF	QSPI2 MEM	QSPI2 MEM	**XMC_MEM（NAND3）**	**XMC_MEM（NAND3）**
0xC0000000	0xDFFFFFFF	XMC_MEM（SDRAM）	**XMC_MEM（NOR/PSRAM/SRAM/NAND2）**	XMC_MEM（SDRAM）	**XMC_MEM（NOR/PSRAM/SRAM/NAND2）**

4 XMC 应用案例

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

4.1 案例 1 SRAM 异步非复用读写

4.1.1 功能简介

实现 16 bit SRAM is61wv51216bll 的读写操作。

4.1.2 资源准备

1) 硬件环境：

包含 AT32F435/437 和 is61wv51216bll 的电路板。

接线如下：

XMC		SRAM
- xmc_a0~a25	--->	sram_a0~a25
- xmc_d0~d15	--->	sram_data[0]~ sram_data[15]
- xmc_ne3	--->	sram_cs
- xmc_noe	--->	sram_rd
- xmc_nwe	--->	sram_wr
- xmc_lb	--->	sram_nbl0
- xmc_ub	--->	sram_nbl1

2) 软件环境：

project\at_start_f437\examples\xmc\sram

4.1.3 软件设计

1) 操作流程

- 相关 GPIO 配置；
- SRAM 界面控制寄存器配置；
- SRAM 界面读/写时序配置；
- SRAM 界面存储块使能；
- 写和读操作。

2) 关键代码介绍

- SRAM 界面初始化

```
void sram_init(void)
{
  gpio_init_type    gpio_init_struct = {0};
  xmc_norsram_init_type    xmc_norsram_init_struct;
  xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;
```

```
  /* enable the gpio clock ---------------------------------------------------- */
  crm_periph_clock_enable(CRM_GPIOF_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOG_PERIPH_CLOCK, TRUE);
  /* enable the xmc clock ----------------------------------------------------- */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ------------------------------------------------------*/
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  …
  gpio_init_struct.gpio_pins = GPIO_PINS_10;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOG, &gpio_init_struct);

  /*-- xmc configuration -------------------------------------------------------*/
  xmc_norsram_default_para_init(&xmc_norsram_init_struct);
  xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM3;/*本例使用 XMC SRAM 界面（bank1）
的子 bank3*/
  xmc_norsram_init_struct.data_addr_multiplex = XMC_DATA_ADDR_MUX_DISABLE;/*数据线非复用*/
  xmc_norsram_init_struct.device = XMC_DEVICE_SRAM;/*存储器类型：SRAM*/
  xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_16_BITS; /*操作位宽：16bit*/
  xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE; /*同步模式：关*/
  xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE; /*异步等待信号使能：关*/
  xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW; /*等待信号有效电平：不使
用，保持默认值*/
  xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE; /*成组模式：关
*/
  xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE; /*等待时序：不使
用，保持默认值*/
  xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE; /*写使能：开*/
  xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE; /*同步等待信号使能：关
*/
  xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE; /*同步写模式：关*/
  xmc_nor_sram_init(&xmc_norsram_init_struct);

  /* timing configuration */
  xmc_norsram_timing_default_para_init(&rw_timing_struct, &w_timing_struct);
  rw_timing_struct.subbank = XMC_BANK1_NOR_SRAM3;
  rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  rw_timing_struct.addr_setup_time = 0x2; /*地址建立时间设置*/
  rw_timing_struct.addr_hold_time = 0x0; /*地址保持时间：modeA 不使用，保持默认值*/
```

```
  rw_timing_struct.data_setup_time = 0xc; /*数据建立时间*/
  rw_timing_struct.bus_latency_time = 0x0; /*总线延迟时间：异步非复用模式不使用，保持默认值*/
  rw_timing_struct.clk_psc = 0x0; /*时钟分频：异步模式不使用，保持默认值*/
  rw_timing_struct.data_latency_time = 0x0; /*数据延迟时间：异步模式不使用，保持默认值*/
  rw_timing_struct.mode = XMC_ACCESS_MODE_A; /*异步模式选择：使用 modeA*/
  w_timing_struct.subbank = XMC_BANK1_NOR_SRAM3; /*bank 选择：此处使用 bank1 的子 bank3*/
  w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  w_timing_struct.addr_setup_time = 0x2; /*写操作的地址建立时间设置*/
  w_timing_struct.addr_hold_time = 0x0; /*地址保持时间：modeA 不使用，保持默认值*/
  w_timing_struct.data_setup_time = 0xc; /*写操作的数据建立时间*/
  w_timing_struct.bus_latency_time = 0x0; /*总线延迟时间：异步非复用模式不使用，保持默认值*/
  w_timing_struct.clk_psc = 0x0; /*时钟分频：异步模式不使用，保持默认值*/
  w_timing_struct.data_latency_time = 0x0; /*数据延迟时间：异步模式不使用，保持默认值*/
  w_timing_struct.mode = XMC_ACCESS_MODE_A;   /*异步模式选择：使用 modeA*/
  xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

  /* bus turnaround phase for consecutive read duration and consecutive write duration */
  xmc_ext_timing_config(XMC_BANK1_NOR_SRAM3, 0x08, 0x08);/*连续读/写操作的恢复时间设置*/

  /* enable xmc_bank1_nor_sram3 */
  xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM3, TRUE);/*bank1 的子 bank3 使能*/
}
```

■ SRAM 读/写

```
/* --SRAM 写操作-----------------------------------------------------*/
void sram_writebuffer(uint16_t* pbuffer, uint32_t writeaddr, uint32_t numhalfwordtowrite)
{
  for(; numhalfwordtowrite != 0; numhalfwordtowrite--) /*!< while there is data to write */
  {
    *(__IO uint16_t *) (BANK1_SRAM3_ADDR + writeaddr) = *pbuffer++;/*以 16bit 宽度向 xmc bank1 的子
bank3 的 writeaddr 地址写入数据*/
    writeaddr += 2;/*每写入一个 16bit 数据后，地址加 2*/
  }
}

/* --SRAM 读操作-----------------------------------------------------*/
void sram_readbuffer(uint16_t* pbuffer, uint32_t readaddr, uint32_t numhalfwordtoread)
{
  for(; numhalfwordtoread != 0; numhalfwordtoread--) /*!< while there is data to read */
  {
    *pbuffer++ = *(__IO uint16_t*) (BANK1_SRAM3_ADDR + readaddr); /*以 16bit 宽度从 xmc bank1 的子
bank3 的 readaddr 地址读取数据*/
    readaddr += 2; /*每读取一个 16bit 数据后，地址加 2*/
  }
}
```

■ main 函数代码描述

```
int main(void)
{
  …
  sram_init();/*SRAM 界面及 GPIO 初始化*/
  fill_buffer(txbuffer, buffer_size, 0x3212);
  sram_writebuffer(txbuffer, write_read_addr, buffer_size);/*向 SRAM 写入一个 buffer*/
  sram_readbuffer(rxbuffer, write_read_addr, buffer_size);/*从 SRAM 读取一个 buffer*/
  /* read back sram memory and check content correctness */
  for(number = 0x00; (number < buffer_size) && (writereadstatus == 0); number++)
  {
    if(rxbuffer[number] != txbuffer[number])
    {
      writereadstatus = number + 1;
    }
  }
  if(writereadstatus == 0)
  {
    /* pass : printf "data is right" */
    printf("data is right\r\n");
  }
  else
  {
    /* fail: printf "data is error" */
    printf("data is error\r\n");
  }
  while(1)
  {
  }
}
```

4.2 案例 2 PSRAM 异步复用读写

4.2.1 功能简介

实现 16 bit 复用 PSRAM w957d6hb 的读写操作。

4.2.2 资源准备

1) 硬件环境：

包含 AT32F435/437 和 w957d6hb 的电路板。

接线如下：

XMC		PSRAM
- xmc_a16~a25	--->	psram_a16~a25
- xmc_ad0~ad15	--->	psram_data[0]~ psram_data[15]

- xmc_ne3 ---> psram_cs
- xmc_noe ---> psram_rd
- xmc_nwe ---> psram_wr
- xmc_lb ---> psram_nbl0
- xmc_ub ---> psram_nbl1
- xmc_nadv ---> psram_nadv

2) 软件环境：

project\at_start_f437\examples\xmc\psram

4.2.3 软件设计

1) 操作流程

- 相关 GPIO 配置；
- PSRAM 界面控制寄存器配置；
- PSRAM 界面读/写时序配置；
- PSRAM 界面存储块使能；
- 写和读操作。

2) 代码介绍

- PSRAM 界面初始化

```
void psram_init (void)
{
  gpio_init_type    gpio_init_struct = {0};
  xmc_norsram_init_type    xmc_norsram_init_struct;
  xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;

  /* enable the gpio clock ---------------------------------------------------- */
  crm_periph_clock_enable(CRM_GPIOF_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOG_PERIPH_CLOCK, TRUE);
  /* enable the xmc clock ----------------------------------------------------- */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOG, GPIO_PINS_SOURCE13, GPIO_MUX_12);
  …
  gpio_init_struct.gpio_pins = GPIO_PINS_7;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOB, &gpio_init_struct);

  /*-- xmc configuration -----------------------------------------------------*/
```

```
  xmc_norsram_default_para_init(&xmc_norsram_init_struct);
  xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM3;/*本例使用 XMC SRAM 界面（bank1）的子 bank3*/
  xmc_norsram_init_struct.data_addr_multiplex = XMC_DATA_ADDR_MUX_ENABLE;/*数据地址线复用*/
  xmc_norsram_init_struct.device = XMC_DEVICE_PSRAM;/*存储器类型：PSRAM*/
  xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_16_BITS; /*操作位宽：16bit*/
  xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE; /*同步模式：关*/
  xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE; /*异步等待信号使能：关*/
  xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW; /*等待信号有效电平：不使用，保持默认值*/
  xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE; /*成组模式：关*/
  xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE; /*等待时序：不使用，保持默认值*/
  xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE; /*写使能：开*/
  xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE; /*同步等待信号使能：关*/
  xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE; /*独立写时序：关*/
  xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE; /*同步写模式：关*/
  xmc_nor_sram_init(&xmc_norsram_init_struct);

  /* timing configuration */
  xmc_norsram_timing_default_para_init(&rw_timing_struct, &w_timing_struct);
  rw_timing_struct.subbank = XMC_BANK1_NOR_SRAM3;
  rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE; /*独立写时序：关*/
  rw_timing_struct.addr_hold_time      = 0x08; /*地址保持时间设置*/
  rw_timing_struct.addr_setup_time     = 0x09; /*地址建立时间设置*/
  rw_timing_struct.data_setup_time     = 0x0F; /*数据建立时间*/
  rw_timing_struct.data_latency_time   = 0x0; /*数据延迟时间：异步模式不使用，保持默认值*/
  rw_timing_struct.bus_latency_time    = 0x0; /*总线延迟时间：异步非复用模式不使用，保持默认值*/
  rw_timing_struct.clk_psc             = 0x0; /*时钟分频：异步模式不使用，保持默认值*/
  rw_timing_struct.mode = XMC_ACCESS_MODE_A; /*时序模式选择：modeA*/
  w_timing_struct.subbank = XMC_BANK1_NOR_SRAM3;
  w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE; /*独立写时序：关。写操作会使用以上设置的读写时序，以下写时序设置无效*/
  w_timing_struct.addr_hold_time       = 0x08;
  w_timing_struct.addr_setup_time      = 0x09;
  w_timing_struct.data_setup_time      = 0x0F;
  w_timing_struct.data_latency_time    = 0x0;
  w_timing_struct.bus_latency_time     = 0x0;
  w_timing_struct.clk_psc              = 0x0;
  w_timing_struct.mode = XMC_ACCESS_MODE_A;
  xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

  /* bus turnaround phase for consecutive read duration and consecutive write duration */
  xmc_ext_timing_config(XMC_BANK1_NOR_SRAM3, 0x08, 0x08);/*连续读/写操作的恢复时间设置*/
```

```
  /* enable xmc_bank1_nor_sram3 */
  xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM3, TRUE);/*bank1 的子 bank3 使能*/
}
```

■ PSRAM 读/写

```
/* --SRAM 写操作----------------------------------------------------*/
void sram_writebuffer(uint16_t* pbuffer, uint32_t writeaddr, uint32_t numhalfwordtowrite)
{
  for(; numhalfwordtowrite != 0; numhalfwordtowrite--) /*!< while there is data to write */
  {
    *(__IO uint16_t *) (BANK1_SRAM3_ADDR + writeaddr) = *pbuffer++;/*以 16bit 宽度向 xmc bank1 的子 bank3 的 writeaddr 地址写入数据*/
    writeaddr += 2;/*每写入一个 16bit 数据后，地址加 2*/
  }
}

/* --SRAM 读操作----------------------------------------------------*/
void sram_readbuffer(uint16_t* pbuffer, uint32_t readaddr, uint32_t numhalfwordtoread)
{
  for(; numhalfwordtoread != 0; numhalfwordtoread--) /*!< while there is data to read */
  {
    *pbuffer++ = *(__IO uint16_t*) (BANK1_SRAM3_ADDR + readaddr); /*以 16bit 宽度从 xmc bank1 的子 bank3 的 readaddr 地址读取数据*/
    readaddr += 2; /*每读取一个 16bit 数据后，地址加 2*/
  }
}
```

■ main 函数代码描述

```
int main(void)
{
  …
  sram_init();/*SRAM 界面及 GPIO 初始化*/
  fill_buffer(txbuffer, buffer_size, 0x3212);
  psram_writebuffer(txbuffer, write_read_addr, buffer_size);/*向 PSRAM 写入一个 buffer*/
  psram_readbuffer(rxbuffer, write_read_addr, buffer_size);/*从 PSRAM 读取一个 buffer*/
  /* read back sram memory and check content correctness */
  for(number = 0x00; (number < buffer_size) && (writereadstatus == 0); number++)
  {
    if(rxbuffer[number] != txbuffer[number])
    {
      writereadstatus = number + 1;
    }
  }
  if(writereadstatus == 0)
```

```
  {
    /* pass : printf "data is right" */
    printf("data is right\r\n");
  }
  else
  {
    /* fail: printf "data is error" */
    printf("data is error\r\n");
  }
  while(1)
  {
  }
}
```

4.3 案例 3 NOR FLASH 异步非复用读写

4.3.1 功能简介

实现 NOR FLASH m29w128 的 16bit 读写操作。

4.3.2 资源准备

1) 硬件环境：

包含 AT32F435/437 和 m29w128 的电路板。

接线如下：

XMC		NOR FLASH
- xmc_a16~a25	--->	norflash_a0~ norflash_a25
- xmc_d0~d15	--->	norflash_data[0]~ norflash_data[15]
- xmc_ne2	--->	norflash_cs
- xmc_noe	--->	norflash_rd
- xmc_nwe	--->	norflash_wr
- vdd	--->	norflash_byte(接高电平 VDD，选择 16bit 模式)
- xmc_nwait	--->	norflash_busy
- xmc_nadv	--->	norflash_nadv

2) 软件环境：

project\at_start_f437\examples\xmc\nor_flash

4.3.3 软件设计

1) 操作流程

- 相关 GPIO 配置；
- NOR FLASH 界面控制寄存器配置；
- NOR FLASH 界面读/写时序配置；
- NOR FLASH 界面存储块使能；
- 擦除，写和读操作。

2) 代码介绍

- NOR FLASH 界面初始化

```
void nor_init (void)
{
  gpio_init_type    gpio_init_struct = {0};
  xmc_norsram_init_type    xmc_norsram_init_struct;
  xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;

  /* enable the gpio clock ---------------------------------------------------- */
  crm_periph_clock_enable(CRM_GPIOF_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOG_PERIPH_CLOCK, TRUE);
  /* enable the xmc clock ----------------------------------------------------- */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOF, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  …
  gpio_init_struct.gpio_pins = GPIO_PINS_7;
  gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOB, &gpio_init_struct);

  /*-- xmc configuration -----------------------------------------------------*/
  xmc_norsram_default_para_init(&xmc_norsram_init_struct);
  xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM2;/*本例使用 XMC SRAM 界面（bank1）的子 bank2*/
  xmc_norsram_init_struct.data_addr_multiplex = XMC_DATA_ADDR_MUX_DISABLE;/*数据地址线非复用*/
  xmc_norsram_init_struct.device = XMC_DEVICE_NOR;/*存储器类型：NOR FLASH*/
  xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_16_BITS; /*操作位宽：16bit*/
  xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE; /*同步模式：关*/
  xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE; /*异步等待信号使能：关*/
  xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW; /*等待信号有效电平：不使用，保持默认值*/
  xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE; /*成组模式：关*/
  xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE; /*等待时序：不适应，保持默认值*/
  xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE; /*写使能：开*/
  xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE; /*同步等待信号使能：关*/
```

```
  xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_DISABLE; /*独立写时序：关*/
  xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE; /*同步写模式：关*/
  xmc_nor_sram_init(&xmc_norsram_init_struct);

  /* timing configuration */
  xmc_norsram_timing_default_para_init(&rw_timing_struct, &w_timing_struct);
  rw_timing_struct.subbank = XMC_BANK1_NOR_SRAM2;
  rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  rw_timing_struct.addr_setup_time      = 0x8; /*地址建立时间设置*/
  rw_timing_struct.addr_hold_time       = 0x0; /*地址保持时间设置*/
  rw_timing_struct.data_setup_time      = 0x10; /*数据建立时间*/
  rw_timing_struct.data_latency_time    = 0x0; /*数据延迟时间：异步模式不使用，保持默认值*/
  rw_timing_struct.bus_latency_time     = 0x0; /*总线延迟时间：异步非复用模式不使用，保持默认值*/
  rw_timing_struct.clk_psc              = 0x0; /*时钟分频：异步模式不使用，保持默认值*/
  rw_timing_struct.mode = XMC_ACCESS_MODE_B;/*时序模式选择：modeB*/
  w_timing_struct.subbank = XMC_BANK1_NOR_SRAM2;
  w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  w_timing_struct.addr_setup_time       = 0x8; /*写操作的地址建立时间设置*/
  w_timing_struct.addr_hold_time        = 0x0; /*写操作的地址保持时间设置*/
  w_timing_struct.data_setup_time       = 0x10; /*写操作的数据建立时间*/
  w_timing_struct.data_latency_time     = 0x0; /*写操作的数据延迟时间：异步模式不使用，保持默认值*/
  w_timing_struct.bus_latency_time      = 0x0; /*写操作的总线延迟时间：异步非复用模式不使用，保持默认
值*/
  w_timing_struct.clk_psc               = 0x0; /*写操作的时钟分频：异步模式不使用，保持默认值*/
  w_timing_struct.mode = XMC_ACCESS_MODE_B;
  xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

  /* enable xmc_bank1_nor_sram3 */
  xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM2, TRUE);/*bank1 的子 bank3 使能*/
}
```

■ NOR FLASH 读/写/擦除

```
/* --NOR FLASH 写操作----------------------------------------------------*/
/*NOR FLASH  写一个 buffer*/
nor_status nor_writebuffer(uint16_t* pbuffer, uint32_t writeaddr, uint32_t numhalfwordtowrite)
{
    nor_status status = NOR_ONGOING;
    do
    {
        /*!< Transfer data to the memory */
        status = nor_writehalfword(writeaddr, *pbuffer++);/*写入 halfword*/
        writeaddr = writeaddr + 2;/*地址加 2*/
        numhalfwordtowrite--;/*待写入数据长度减 1*/
    }
    while((status == NOR_SUCCESS) && (numhalfwordtowrite != 0));
```

```
    return (status);
}
/*NOR FLASH 写 halfword*/
nor_status nor_writehalfword(uint32_t writeaddr, uint16_t data)
{
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x00AA);/*写 halfword 命令 1*/
    NOR_WRITE(ADDR_SHIFT(0x02AA), 0x0055); /*写 halfword 命令 2/
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x00A0); /*写 halfword 命令 3*/
    NOR_WRITE((BANK1_NOR2_ADDR + writeaddr), data); /*写 halfword 的地址和数据*/

    return (nor_getstatus(PROGRAM_TIMEOUT));/*等待操作完成*/
}

/* --NOR FLASH 读操作------------------------------------------------------*/
void nor_readbuffer(uint16_t* pbuffer, uint32_t readaddr, uint32_t numhalfwordtoread)
{
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x00AA);/*读命令 1*/
    NOR_WRITE(ADDR_SHIFT(0x02AA), 0x0055); /*读命令 2*/
    NOR_WRITE((BANK1_NOR2_ADDR + readaddr), 0x00F0); /*写入读地址*/
    for(; numhalfwordtoread != 0x00; numhalfwordtoread--) /*!< while there is data to read */
    {
        /*!< read a halfword from the nor */
        *pbuffer++ = *(__IO uint16_t *)((BANK1_NOR2_ADDR + readaddr)); /*开始读数据*/
        readaddr = readaddr + 2; /*每读取一个 16bit 数据后，地址加 2*/
    }
}
/* --NOR FLASH 擦除操作------------------------------------------------------*/
nor_status nor_eraseblock(uint32_t blockaddr)
{
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x00AA); /*擦除命令 1*/
    NOR_WRITE(ADDR_SHIFT(0x02AA), 0x0055); /*擦除命令 2*/
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x0080); /*擦除命令 3*/
    NOR_WRITE(ADDR_SHIFT(0x0555), 0x00AA); /*擦除命令 4*/
    NOR_WRITE(ADDR_SHIFT(0x02AA), 0x0055); /*擦除命令 5*/
    NOR_WRITE((BANK1_NOR2_ADDR + blockaddr), 0x30); /*写入擦除地址*/
    return (nor_getstatus(BLOCKERASE_TIMEOUT)); /*等待操作完成*/
}
```

- main 函数代码描述

```
int main(void)
{
  …
  /* nor flash interface confoig */
  nor_init();/*NOR FLASH 界面初始化*/
  nor_readid(&nor_id);/*读 NOR FLASH 的 ID*/
```

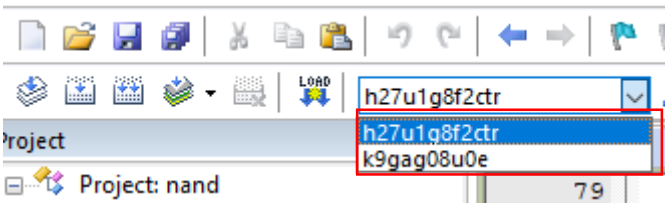
```
  nor_returntoreadmode();/*回到读模式*/
  nor_eraseblock(write_read_addr);/*擦除后续需要读写的 block*/
  fill_buffer(txbuffer, buffer_size, 0x3210);/*填充后续要写的 buffer*/
  nor_writebuffer(txbuffer, write_read_addr, buffer_size);/*向 NOR FLASH 写入一个 buffer 的数据*/
  nor_readbuffer(rxbuffer, write_read_addr, buffer_size);/*从 NOR FLASH 读出一个 buffer 的数据*/
  /* read back sram memory and check content correctness */
  for(number = 0x00; (number < buffer_size) && (writereadstatus == 0); number++)
  {
    if (rxbuffer[number] != txbuffer[number])
    {
      writereadstatus = number + 1;
    }
  }
  if(writereadstatus == 0)
  {
    /* pass : printf "data is right" */
    printf("data is right\r\n");
  }
  else
  {
    /* fail: printf "data is error" */
    printf("data is error\r\n");
  }
  while(1)
  {
  }
}
```

4.4 案例 4 NAND FLASH 读写

4.4.1 功能简介

实现 8bit NAND FLASH h27u1g8f2ctr(hynix)和 k9gag08u0e(samsung)的读写操作。两个 NAND 型号的切换参考下图 37 的红框处。

图 37 切换 NAND FLASH 型号



4.4.2 资源准备

1) 硬件环境：

包含 AT32F435/437 和 h27u1g8f2ctr 或 k9gag08u0e 的电路板。

接线如下：

XMC		NAND FLASH
- xmc_d0~d7	--->	io[0]~ io[7]
- xmc_a16	--->	cle
- xmc_a17	--->	ale
- xmc_nce2(xmc_ne1)	--->	ce#
- xmc_noe	--->	re#
- xmc_nwe	--->	we#
- xmc_wait	--->	r/b#

2) 软件环境：

project\at_start_f437\examples\xmc\nand

4.4.3 软件设计

1) 配置流程

- 相关 GPIO 配置；
- NAND FLASH 界面控制寄存器配置；
- NAND FLASH 界面读/写时序配置；
- NAND FLASH 界面存储块使能；
- 擦除，写和读操作。

2) 代码介绍

- NAND FLASH 界面初始化

```
void nand_init(void)
{
  gpio_init_type gpio_init_struct;
  xmc_nand_init_type nand_init_struct;
  xmc_nand_pccard_timinginit_type   regular_spacetiming_struct, special_spacetiming_struct;
  /* enable the xmc clock */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);
  /* enable gpiod/gpiob/gpioe clock */
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  …
  /* d4->d7 nand pin configuration   */
  gpio_init_struct.gpio_pins = GPIO_PINS_7 | GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10;
  gpio_init(GPIOE, &gpio_init_struct);

  /* -- xmc configuration ----------------------------------------------------- */
  xmc_nand_default_para_init(&nand_init_struct);
  nand_init_struct.nand_bank = XMC_BANK2_NAND;/*使用 XMC BANK2*/
```

```
  nand_init_struct.wait_enable = XMC_WAIT_OPERATION_DISABLE;/*等待信号使能：关*/
  nand_init_struct.bus_type = XMC_BUSTYPE_8_BITS;/*操作宽度：8bit*/
  nand_init_struct.ecc_enable = XMC_ECC_OPERATION_DISABLE;/*ECC 校验：关*/
#ifdef H27U1G8F2CTR
  nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_2048_BYTES;/*ECC page 大小：
H27U1G8F2CTR 为 2048 bytes*/
#elif defined K9GAG08U0E
  nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_8192_BYTES; /*ECC page 大小：
K9GAG08U0E 为 8192 bytes*/
#endif
  nand_init_struct.delay_time_cycle = 0x00;/* CLE 至 NRE 的延迟时间*/
  nand_init_struct.delay_time_ar = 0x00; /* ALE 至 NRE 的延迟时间*/
  xmc_nand_init(&nand_init_struct);
  xmc_nand_timing_default_para_init(&regular_spacetiming_struct, &special_spacetiming_struct);
  regular_spacetiming_struct.class_bank = XMC_BANK2_NAND;
  regular_spacetiming_struct.mem_setup_time = 254;/*设置常规空间的建立时间*/
  regular_spacetiming_struct.mem_hiz_time = 254;/*设置常规空间的数据总线高阻时间*/
  regular_spacetiming_struct.mem_hold_time = 254;/*设置常规空间的保持时间*/
  regular_spacetiming_struct.mem_waite_time = 254;/*设置常规空间的等待时间*/
  special_spacetiming_struct.class_bank = XMC_BANK2_NAND;
  special_spacetiming_struct.mem_setup_time = 254; /*设置特殊空间的建立时间*/
  special_spacetiming_struct.mem_hiz_time = 254; /*设置特殊空间的数据总线高阻时间*/
  special_spacetiming_struct.mem_hold_time = 254; /*设置特殊空间的保持时间*/
  special_spacetiming_struct.mem_waite_time = 254; /*设置特殊空间的等待时间*/
  xmc_nand_timing_config(&regular_spacetiming_struct, &special_spacetiming_struct);
  /* xmc nand bank cmd test */
  xmc_nand_enable(XMC_BANK2_NAND, TRUE);/* BANK2 使能*/
}
```

■ NAND FLASH 读/写/擦除

```
/* -- NAND FLASH page 写操作----------------------------------------------------*/
uint32_t nand_write_small_page(uint8_t *pbuffer, nand_address_type address_struct, uint32_t
num_page_to_write)
{
  uint32_t index = 0x00, num_page_written = 0x00, address_status = NAND_VALID_ADDRESS;
  uint32_t status = NAND_READY, size = 0x00;
  while((num_page_to_write != 0x00) && (address_status == NAND_VALID_ADDRESS) && (status ==
NAND_READY))
  {
    /* page write command and address */
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_WRITE0;/*page write 操作的命令
1*/
#ifdef H27U1G8F2CTR
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS);/*page write
的目标地址-- H27U1G8F2CTR 分 4 个 cycle 写入*/
```

```
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS); /*page write 的目标地址-- K9GAG08U0E 分 5 个 cycle  写入*/
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
    /* calculate the size */
    size = NAND_PAGE_SIZE + (NAND_PAGE_SIZE * num_page_written);/*计算待写入的数据个数*/
    /* write data */
    for(index = 0; index < size; index++)
    {
      *(__IO uint8_t *)(Bank_NAND_ADDR | DATA_AREA) = pbuffer[index];/*向 NAND FLASH 写入数据*/
    }
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_WRITE_TRUE1; /*page write 操作的命令 2*/
    /* check status for successful operation */
    status = nand_get_status();/*读 NAND FLASH 的状态寄存器，直到写操作完成*/
    if(status == NAND_READY)
    {
      num_page_written++;
      num_page_to_write--;
      /* calculate next small page address */
      address_status = nand_address_increment(&address_struct);
    }
  }
  return (status | address_status);
}

/* -- NAND FLASH page 读操作----------------------------------------------------*/
uint32_t nand_read_small_page(uint8_t *pbuffer, nand_address_type address_struct, uint32_t num_page_to_read)
{
  uint32_t index = 0x00, num_page_read = 0x00, address_status = NAND_VALID_ADDRESS;
  uint32_t status = NAND_READY, size = 0x00 ;

  while((num_page_to_read != 0x0) && (address_status == NAND_VALID_ADDRESS))
  {
    /* page read command and page address */
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_AREA_A; /*page read 操作的命令 1*/
```

```
#ifdef H27U1G8F2CTR
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS);/*page read 的目标地址-- H27U1G8F2CTR 分 4 个 cycle 写入 */
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS); /*page read 的目标地址-- K9GAG08U0E 分 5 个 cycle 写入 */
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_AREA_TRUE1;/*page read 操作的命令 2*/
    delay_ms(1);/*加一点延时，以等待 page read 命令写入后，NAND FLASH 准备好*/
    /* calculate the size */
    size = (NAND_PAGE_SIZE + NAND_SPARE_AREA_SIZE) + ((NAND_PAGE_SIZE + NAND_SPARE_AREA_SIZE)  * num_page_read);/*计算需要读取的数据长度*/
    /* get data into buffer */
    for(index = 0x00; index < size; index++)
    {
      pbuffer[index]= *(__IO uint8_t *)(Bank_NAND_ADDR | DATA_AREA);/*从 NAND FLASH 读取数据*/
    }
    num_page_read++;
    num_page_to_read--;
    /* calculate page address */
    address_status = nand_address_increment(&address_struct);
  }
  status = nand_get_status();
  return (status | address_status);
}

/* -- NAND FLASH 擦除操作----------------------------------------------------*/
uint32_t nand_erase_block(nand_address_type address_struct)
{
  *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_ERASE0; /*擦除操作的命令 1*/
#ifdef H27U1G8F2CTR
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);/*写入待擦除的 block 地址。此处写入行地址，NAND FLASH 仅取用其中有效的 block 地址。-- H27U1G8F2CTR 分 2 个 cycle 写入 */
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS); /*写入待擦除的 block 地址。此处写入行地址，NAND FLASH 仅取用其中有效的 block 地址。-- K9GAG08U0E 分 3 个
```

```
cycle 写入 */
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
  *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_ERASE1; /*擦除操作的命令 2*/
  return (nand_get_status());/*读取 NAND FLASH 状态寄存器，等待擦除操作完成*/
}
```

■ main 函数代码描述

```
int main(void)
{
  …
  /* xmc initialization */
  nand_init();/*NAND FLASH 界面初始化，GPIO 初始化*/
  /* nand reset command */
  nand_reset();/*复位 NAND FLASH*/
  delay_us(10);/*延时 10us，等待复位操作完成*/
  /* nand read id command */
  nand_read_id(&nand_id_struct);/*读取 ID*/

  /* verify the nand id */
  /* nand support:samsung:k9gag08u0e            hynix:h27u1g8f2ctr
                  id      :0xecd58472            id    :0xadf1801d
  */
  if((nand_id_struct.maker_id == NAND_AT_MakerID) && (nand_id_struct.device_id ==
NAND_AT_DeviceID))
  {
    /* nand memory address to write to */
    write_read_addr_struct.zone   = 0x00;/*设置即将写入数据的地址*/
    write_read_addr_struct.block = 0x00;
    write_read_addr_struct.page   = 0x00;
    write_read_addr_struct.byte   = 0x00;
    /* erase the nand first block */
    status = nand_erase_block(write_read_addr_struct);/*擦除即将写入的 block*/
    /* fill the buffer to send */
    fill_buffer(txbuffer, BUFFER_SIZE , 0x66);/*填充需要写入的 buffer*/
    /* write data to xmc nand memory */
    status = nand_write_small_page(txbuffer, write_read_addr_struct, pagenumber);/*page 写*/
    /* read back the written data */
    status = nand_read_small_page (rxbuffer, write_read_addr_struct, pagenumber);/*page 读*/
    /* verify the written data */
    for(j = 0; j < BUFFER_SIZE; j++)
    {
      if(txbuffer[j] != rxbuffer[j])
      {
```

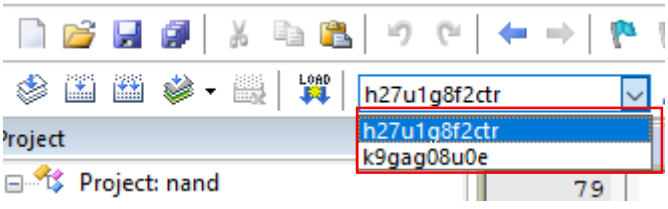
```
            writereadstatus++;
          }
        }
        if(writereadstatus == 0)
        {
          /* pass : printf "data is right" */
          printf("data is right\r\n");
        }
        else
        {
          /* fail: printf "data is error" */
          printf("data is error\r\n");
        }
      }
      /* printf to indicate that whether there is an device error (id not correct) */
      else
      {
        printf("the id is error\r\n");
      }
      while(1)
      {
      }
}
```

4.5 案例 5 NAND FLASH ECC 纠错

4.5.1 功能简介

实现 8bit NAND FLASH h27u1g8f2ctr(hynix)和 k9gag08u0e(samsung)的 ECC 纠错。手动造出 1bit 的错误，然后使用 ECC 功能和纠正算法进行纠正。
两个 NAND 型号的切换参考下图 38 的红框处。

图 38 切换 NAND FLASH 型号



4.5.2 资源准备

1) 硬件环境：

包含 AT32F435/437 和 h27u1g8f2ctr 或 k9gag08u0e 的电路板。

接线如下：

XMC		NAND FLASH
- xmc_d0~d7	--->	io[0]~ io[7]

- xmc_a16 ---> cle
- xmc_a17 ---> ale
- xmc_nce2(xmc_ne1) ---> ce#
- xmc_noe ---> re#
- xmc_nwe ---> we#
- xmc_wait ---> r/b#

2) 软件环境：

project\at_start_f437\examples\xmc\ecc

4.5.3 软件设计

1) 配置流程

- 相关 GPIO 配置；
- NAND FLASH 界面控制寄存器配置；
- NAND FLASH 界面读/写时序配置；
- NAND FLASH 界面存储块使能；
- 擦除，写操作，读操作和纠正程序。

2) 配置流程

- NAND FLASH 界面初始化

```
void nand_init(void)
{
  gpio_init_type gpio_init_struct;
  xmc_nand_init_type nand_init_struct;
  xmc_nand_pccard_timinginit_type  regular_spacetiming_struct, special_spacetiming_struct;
  /* enable the xmc clock */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);
  /* enable gpiod/gpiob/gpioe clock */
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE0, GPIO_MUX_12);
  …
  /* d4->d7 nand pin configuration  */
  gpio_init_struct.gpio_pins = GPIO_PINS_7 | GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10;
  gpio_init(GPIOE, &gpio_init_struct);

  /* -- xmc configuration ----------------------------------------------------- */
  xmc_nand_default_para_init(&nand_init_struct);
  nand_init_struct.nand_bank = XMC_BANK2_NAND;/*使用 XMC BANK2*/
  nand_init_struct.wait_enable = XMC_WAIT_OPERATION_DISABLE;/*等待信号使能：关*/
  nand_init_struct.bus_type = XMC_BUSTYPE_8_BITS;/*操作宽度：8bit*/
  nand_init_struct.ecc_enable = XMC_ECC_OPERATION_DISABLE;/*ECC 校验：关*/
#ifdef H27U1G8F2CTR
```

```
  nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_2048_BYTES;/*ECC page 大小：
H27U1G8F2CTR 为 2048 bytes*/
#elif defined K9GAG08U0E
  nand_init_struct.ecc_pagesize = XMC_ECC_PAGESIZE_8192_BYTES; /*ECC page 大小：
K9GAG08U0E 为 8192 bytes*/
#endif
  nand_init_struct.delay_time_cycle = 0x00;/* CLE 至 NRE 的延迟时间*/
  nand_init_struct.delay_time_ar = 0x00; /* ALE 至 NRE 的延迟时间*/
  xmc_nand_init(&nand_init_struct);
  xmc_nand_timing_default_para_init(&regular_spacetiming_struct, &special_spacetiming_struct);
  regular_spacetiming_struct.class_bank = XMC_BANK2_NAND;
  regular_spacetiming_struct.mem_setup_time = 254;/*设置常规空间的建立时间*/
  regular_spacetiming_struct.mem_hiz_time = 254;/*设置常规空间的数据总线高阻时间*/
  regular_spacetiming_struct.mem_hold_time = 254;/*设置常规空间的保持时间*/
  regular_spacetiming_struct.mem_waite_time = 254;/*设置常规空间的等待时间*/
  special_spacetiming_struct.class_bank = XMC_BANK2_NAND;
  special_spacetiming_struct.mem_setup_time = 254; /*设置特殊空间的建立时间*/
  special_spacetiming_struct.mem_hiz_time = 254; /*设置特殊空间的数据总线高阻时间*/
  special_spacetiming_struct.mem_hold_time = 254; /*设置特殊空间的保持时间*/
  special_spacetiming_struct.mem_waite_time = 254; /*设置特殊空间的等待时间*/
  xmc_nand_timing_config(&regular_spacetiming_struct, &special_spacetiming_struct);
  /* xmc nand bank cmd test */
  xmc_nand_enable(XMC_BANK2_NAND, TRUE);/* BANK2 使能*/
}
```

■ NAND FLASH 读/写/擦除

```
/* -- NAND FLASH page 写操作----------------------------------------------------*/
uint32_t nand_write_small_page(uint8_t *pbuffer, nand_address_type address_struct, uint32_t
num_page_to_write)
{
  uint32_t index = 0x00, num_page_written = 0x00, address_status = NAND_VALID_ADDRESS;
  uint32_t status = NAND_READY, size = 0x00;
  while((num_page_to_write != 0x00) && (address_status == NAND_VALID_ADDRESS) && (status ==
NAND_READY))
  {
    /* page write command and address */
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_WRITE0;/*page write 操作的命令
1*/
#ifdef H27U1G8F2CTR
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS);/*page write
的目标地址-- H27U1G8F2CTR 分 4 个 cycle 写入*/
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
```

```
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS); /*page write 的目标地址-- K9GAG08U0E 分 5 个 cycle  写入*/
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
    /* calculate the size */
    size = NAND_PAGE_SIZE + (NAND_PAGE_SIZE * num_page_written);/*计算待写入的数据个数*/
    /* write data */
    for(index = 0; index < size; index++)
    {
      *(__IO uint8_t *)(Bank_NAND_ADDR | DATA_AREA) = pbuffer[index];/*向 NAND FLASH 写入数据*/
    }
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_WRITE_TRUE1; /*page write 操作的命令 2*/
    while(xmc_flag_status_get(XMC_BANK2_NAND,XMC_FEMPT_FLAG) == RESET); /*等待 FIFO 空*/
    ecc_value_write = xmc_ecc_get(XMC_BANK2_NAND);/*获取本次写操作产生的 ECC 码*/
    xmc_nand_ecc_enable(XMC_BANK2_NAND, FALSE);/*关闭 ECC*/
    /* check status for successful operation */
    status = nand_get_status();/*读 NAND FLASH 的状态寄存器，直到写操作完成*/
    if(status == NAND_READY)
    {
      num_page_written++;
      num_page_to_write--;
      /* calculate next small page address */
      address_status = nand_address_increment(&address_struct);
    }
  }
  return (status | address_status);
}

/* -- NAND FLASH page 读操作----------------------------------------------------*/
uint32_t nand_read_small_page(uint8_t *pbuffer, nand_address_type address_struct, uint32_t num_page_to_read)
{
  uint32_t index = 0x00, num_page_read = 0x00, address_status = NAND_VALID_ADDRESS;
  uint32_t status = NAND_READY, size = 0x00 ;

  while((num_page_to_read != 0x0) && (address_status == NAND_VALID_ADDRESS))
  {
    /* page read command and page address */
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_AREA_A; /*page read 操作的命令 1*/
#ifdef H27U1G8F2CTR
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS);/*page read
```

```
的目标地址-- H27U1G8F2CTR 分 4 个 cycle 写入 */
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(COL_ADDRESS); /*page read
的目标地址-- K9GAG08U0E 分 5 个 cycle 写入 */
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(COL_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
    *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
    *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_AREA_TRUE1;/*page read 操作
的命令 2*/
    delay_ms(1);/*加一点延时，以等待 page read 命令写入后，NAND FLASH 准备好*/
    /* calculate the size */
    size = (NAND_PAGE_SIZE + NAND_SPARE_AREA_SIZE) + ((NAND_PAGE_SIZE +
NAND_SPARE_AREA_SIZE)  * num_page_read);/*计算需要读取的数据长度*/
    /* get data into buffer */
    for(index = 0x00; index < size; index++)
    {
      pbuffer[index]= *(__IO uint8_t *)(Bank_NAND_ADDR | DATA_AREA);/*从 NAND FLASH 读取数据*/
    }
    ecc_value_read = xmc_ecc_get(XMC_BANK2_NAND);/*获取本次读操作产生的 ECC 码*/
    xmc_nand_ecc_enable(XMC_BANK2_NAND, FALSE);/*关闭 ECC*/
    num_page_read++;
    num_page_to_read--;
    /* calculate page address */
    address_status = nand_address_increment(&address_struct);
  }
  status = nand_get_status();
  return (status | address_status);
}

/* -- NAND FLASH 擦除操作----------------------------------------------------*/
uint32_t nand_erase_block(nand_address_type address_struct)
{
  *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_ERASE0; /*擦除操作的命令 1*/
#ifdef H27U1G8F2CTR
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS);/*写入待擦除
的 block 地址。此处写入行地址，NAND FLASH 仅取用其中有效的 block 地址。-- H27U1G8F2CTR 分 2 个
cycle 写入 */
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
#elif defined K9GAG08U0E
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_1st_cycle(ROW_ADDRESS); /*写入待擦除
的 block 地址。此处写入行地址，NAND FLASH 仅读取其中有效的 block 地址。-- K9GAG08U0E 分 3 个
```

```
cycle 写入 */
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_2nd_cycle(ROW_ADDRESS);
  *(__IO uint8_t *)(Bank_NAND_ADDR | ADDR_AREA) = addr_3rd_cycle(ROW_ADDRESS);
#endif
  *(__IO uint8_t *)(Bank_NAND_ADDR | CMD_AREA) = NAND_CMD_ERASE1; /*擦除操作的命令 2*/
  return (nand_get_status());/*读取 NAND FLASH 状态寄存器，等待擦除操作完成*/
}
```

- ECC 纠正函数代码描述

```
void nand_ecc_correction(uint8_t *pbuffer,uint32_t tx_ecc_value ,uint32_t rx_ecc_value)
{
  uint32_t ecc_value=0, position=0 ,byte_position=0;
  uint8_t i,compare_data;

  /* check ecc value */
  if(tx_ecc_value!=rx_ecc_value)
  {
    /* 2048 byte -- 28bit ecc valid value; 8192 byte -- 32bit ecc valid value */
#ifdef H27U1G8F2CTR
    ecc_value =(tx_ecc_value^rx_ecc_value) & (uint32_t)0x0FFFFFFF;/* H27U1G8F2CTR page 大小
2048byte，对应 ECC 码有效位 28bit，此处取 ECC 码的低 28bit 有效位*/
#elif defined K9GAG08U0E
    ecc_value =(tx_ecc_value^rx_ecc_value) & (uint32_t)0xFFFFFFFF; /* K9GAG08U0E page 大小
8192byte，对应 ECC 码有效位 32bit，此处取 ECC 码的 32bit 有效位*/
#endif
    /* 1 bit error correction */
#ifdef H27U1G8F2CTR
    for(i=0;i<(28/2);i++)
#elif defined K9GAG08U0E
    for(i=0;i<(32/2);i++)
#endif
    {
      compare_data = (ecc_value>>(i*2))&0x3;
      /* find position */
      if(compare_data == 0x2)
      {
        position |= (1<<i);/*定位错误 bit 的位置*/
      }
      /* more than 1 bit erroc */
      else if(compare_data != 0x1)/*如果错误超过 1bit，将无法定位错误位置*/
      {
        return;
      }
    }
    /* correct receive value */
```

```
    byte_position = (uint32_t)(position/8);
    pbuffer[byte_position] ^= 1 << (position % 8);/*纠正错误的 bit*/
  }
}
```

■ main 函数代码描述

```
int main(void)
{
  …
  /* xmc initialization */
  nand_init();/*NAND  界面初始化，GPIO 初始化*/
  /* nand reset command */
  nand_reset();/*复位 NAND FLASH*/
  delay_us(10);
  /* nand read id command */
  nand_read_id(&nand_id_struct);/*读 ID*/
  /* verify the nand id */
  /* nand support:samsung:k9gag08u0e            hynix:h27u1g8f2ctr
                  id      :0xecd58472            id    :0xadf1801d
  */
  if((nand_id_struct.maker_id == NAND_AT_MakerID) && (nand_id_struct.device_id ==
NAND_AT_DeviceID))
  {
    /* nand memory address to write to */
    write_read_addr_struct.zone   = 0x00;/*设置即将读写的地址*/
    write_read_addr_struct.block = 0x00;
    write_read_addr_struct.page   = 0x00;
    write_read_addr_struct.byte   = 0x00;

    /* erase the nand first block */
    status = nand_erase_block(write_read_addr_struct);/*擦除即将读写的 block*/
    /* write data to xmc nand memory */
    /* fill the buffer to send */
    fill_buffer(tx_buffer, BUFFER_SIZE , 0x0);/*填充即将写入的 buffer*/
    /* change the regular to make ecc value isn't 0 */
    tx_buffer[10]=0x03;/*设置 tx_buffer[10]的值*/
     /* calculate ecc value while transmiting */
    xmc_nand_ecc_enable(XMC_BANK2_NAND, TRUE);/*使能 ECC 功能*/
    status = nand_write_small_page(tx_buffer, write_read_addr_struct, page_number);/*写入一个 buffer 的
值，并获取一个写入时产生的 ECC 码-- ecc_value_write */
    /* save the right ecc_value (in this case ,we suppose it as the true value) */
    ecc_value_write_last = ecc_value_write;/*将本次 ECC 码赋值给 ecc_value_write_last */
    /* change the 85 data like there is a 1 bit error happened */
    status = nand_erase_block(write_read_addr_struct);/*擦除之前写入的 block*/
    tx_buffer[10]=0x23;/*将 tx_buffer[10]的值更改 1bit，即造成 1bit 的错误*/
```

```
    status = nand_write_small_page(tx_buffer, write_read_addr_struct, page_number); /*将有 1bit 错误的
buffer 再次写入 NAND FLASH*/
    /* calculate ecc value while transmiting */
    xmc_nand_ecc_enable(XMC_BANK2_NAND, TRUE);/*重新开启 ECC 功能*/
    /* read back the written data */
    status = nand_read_small_page (rx_buffer, write_read_addr_struct, page_number);/*读取上一步骤写入
的数据，并获取读这个 buffer 产生的 ECC 码-- ecc_value_read */
    /* ecc check and correct 1 bit error */
    nand_ecc_correction(rx_buffer ,ecc_value_write_last ,ecc_value_read);/*使用 ECC 纠正函数—根据第
一次写入正确 buffer 产生的 ECC 码和读取的有 1bit 错误的 buffer 产生的 ECC 码来校准读取的 buffer
rx_buffer */
    if(rx_buffer[10]==0x03)/*判断 ECC 纠正函数是否生效*/
    {
      /* pass : printf "ecc is work" */
      printf("ecc is work\r\n");
    }
    else
    {
      /* fail: printf "ecc is not work" */
      printf("ecc is not work\r\n");
    }
  }
  /* printf to indicate that whether there is an device error (id not correct) */
  else
  {
    printf("the id is error\r\n");
  }
  while(1)
  {
  }
}
```

4.6 案例 6 8bit LCD 屏驱动

4.6.1 功能简介

通过 XMC 驱动 8bit LCD 显示图片。

4.6.2 资源准备

1) 硬件环境：

包含 AT32F435/435 和 8bit 的 8080 接口 LCD 屏 KD024C 的电路板。

2) 软件环境：

project\at_start_f437\examples\xmc\lcd_8bit

4.6.3 软件设计

1) 配置流程

- XMC 初始化；
- 复位 LCD；
- LCD 屏各项参数配置；
- 配置 LCD 开始显示；
- 定义并擦除需要显示的区域；
- 显示一张图片：XMC 循环发送需要显示的图片数据（预先准备好的一组 buffer）；
- 重复上两个步骤，刷新更换不同图片。

2) 代码介绍

- XMC 初始化

```
void xmc_init(void)
{
  gpio_init_type    gpio_init_struct = {0};
  xmc_norsram_init_type    xmc_norsram_init_struct;
  xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;

  /* enable the gpio clock */
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOF_PERIPH_CLOCK, TRUE);

  /* enable the xmc clock */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOD, GPIO_PINS_SOURCE4, GPIO_MUX_12);
  …
  /* lcd reset lines configuration */
  gpio_init_struct.gpio_pins = GPIO_PINS_9;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOB, &gpio_init_struct);

  /*-- xmc configuration -----------------------------------------------------*/
  xmc_norsram_default_para_init(&xmc_norsram_init_struct);
  xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM1; /*使用 XMC BANK1 的子 bank1*/
  xmc_norsram_init_struct.data_addr_multiplex = XMC_DATA_ADDR_MUX_DISABLE; /*非复用*/
  xmc_norsram_init_struct.device = XMC_DEVICE_SRAM; /*器件类型选择默认的 SRAM*/
  xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_8_BITS; /*操作宽度：8bit*/
  xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE;/*同步模式：关*/
  xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW; /*无无需使用，保持默认*/
```

```
  xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE; /*写使能：开*/
  xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
  xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE; /*无需使用，保持默认*/
  xmc_nor_sram_init(&xmc_norsram_init_struct);

  /* timing configuration */
  xmc_norsram_timing_default_para_init(&rw_timing_struct, &w_timing_struct);
  rw_timing_struct.subbank = XMC_BANK1_NOR_SRAM1;
  rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE;
  rw_timing_struct.addr_setup_time = 0x2; /*设置读/写时序*/
  rw_timing_struct.addr_hold_time = 0x0;
  rw_timing_struct.data_setup_time = 0x2;
  rw_timing_struct.bus_latency_time = 0x0;
  rw_timing_struct.clk_psc = 0x0;
  rw_timing_struct.data_latency_time = 0x0;
  rw_timing_struct.mode = XMC_ACCESS_MODE_A; /*使用 modeA*/
  w_timing_struct.subbank = XMC_BANK1_NOR_SRAM1;
  w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE;
  w_timing_struct.addr_setup_time = 0x2; /*设置写时序*/
  w_timing_struct.addr_hold_time = 0x0;
  w_timing_struct.data_setup_time = 0x2;
  w_timing_struct.bus_latency_time = 0x0;
  w_timing_struct.clk_psc = 0x0;
  w_timing_struct.data_latency_time = 0x0;
  w_timing_struct.mode = XMC_ACCESS_MODE_A; /*使用 modeA*/
  xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

  /* bus turnaround phase for consecutive read duration and consecutive write duration */
  xmc_ext_timing_config(XMC_BANK1_NOR_SRAM1, 0x08, 0x08);/*额外时序—保持默认值*/

  /* enable xmc_bank1_nor_sram4 */
  xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM1, TRUE); /*使能 BANK1 的子 bank1*/
}
```

- lcd 初始化函数代码描述

```
void lcd_init(void)
{
  /* init xmc */
  xmc_init(); /*XMC 接口初始化*/
```

```
delay_ms(50);

/* reset lcd */
LCD_RESET_HIGH;
delay_ms(1);
LCD_RESET_LOW; /*复位 LCD*/
delay_ms(10);
LCD_RESET_HIGH; /*退出复位*/
delay_ms(120);
/*以下为 LCD 各项参数设置：例如扫描方向、显示区域、帧率等等。具体请参考 KD024C 规格书*/
lcd_wr_command(0x36);
lcd_wr_data(0x00);
lcd_wr_command(0x3a);
lcd_wr_data(0x55);

/* 320*240 */
lcd_wr_command(0x2a);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_data(0x01);
lcd_wr_data(0x3f);
lcd_wr_command(0x2b);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_data(0xef);

lcd_wr_command(0xb2);
lcd_wr_data(0x0c);
lcd_wr_data(0x0c);
lcd_wr_data(0x00);
lcd_wr_data(0x33);
lcd_wr_data(0x33);
lcd_wr_command(0xb7);
lcd_wr_data(0x35);
lcd_wr_command(0xbb);
lcd_wr_data(0x30);
lcd_wr_command(0xc3);
lcd_wr_data(0x1c);
lcd_wr_command(0xc4);
lcd_wr_data(0x18);
lcd_wr_command(0xc6);
lcd_wr_data(0x0f);
lcd_wr_command(0xd0);
lcd_wr_data(0xa4);
lcd_wr_data(0xa2);
```

```
  lcd_wr_command(0xe0);
  lcd_wr_data(0xf0);
  lcd_wr_data(0x00);
  lcd_wr_data(0x0a);
  lcd_wr_data(0x10);
  lcd_wr_data(0x12);
  lcd_wr_data(0x1b);
  lcd_wr_data(0x39);
  lcd_wr_data(0x44);
  lcd_wr_data(0x47);
  lcd_wr_data(0x28);
  lcd_wr_data(0x12);
  lcd_wr_data(0x10);
  lcd_wr_data(0x16);
  lcd_wr_data(0x1b);
  lcd_wr_command(0xe1);
  lcd_wr_data(0xf0);
  lcd_wr_data(0x00);
  lcd_wr_data(0x0a);
  lcd_wr_data(0x10);
  lcd_wr_data(0x11);
  lcd_wr_data(0x1a);
  lcd_wr_data(0x3b);
  lcd_wr_data(0x34);
  lcd_wr_data(0x4e);
  lcd_wr_data(0x3a);
  lcd_wr_data(0x17);
  lcd_wr_data(0x16);
  lcd_wr_data(0x21);
  lcd_wr_data(0x22);
  lcd_wr_command(0x3a);
  lcd_wr_data(0x55);
  lcd_wr_command(0x11);

  delay_ms(120);

  /* display on */
  lcd_wr_command(0x29); /*开始显示*/
}
```

■ lcd 显示函数代码描述

```
void lcd_display_number(void)
{
  int i;
  lcd_setblock(50, 50, 199, 219); /*选择显示区域*/
```

```
  for(i=0; i<sizeof(gimage_8); i++)
  {
    lcd_wr_data(gimage_8[i]); /*写入需要显示的图片对应的数据 buffer*/
  }
  delay_ms(1000);
  …(重复上面的步骤，更换 buffer(gimage_8)的数据，以显示不同的图片)
}
```

- mian 函数代码描述

```
int main(void)
{
  …
  /* configures xmc&lcd */
  lcd_init();/*LCD 初始化—包含 XMC 界面初始化*/
  /* clear the lcd */
  point_color = WHITE;
  lcd_clear(point_color);/*清除 LCD 屏幕，显示白色*/

  while(1)
  {
    /* display number on lcd */
    lcd_display_number();/*开始显示图片*/
  }
}
```

4.7 案例 7 16bit LCD 触摸屏驱动

4.7.1 功能简介

通过 XMC 驱动 16bit LCD 触摸屏，显示绘制内容。

4.7.2 资源准备

1) 硬件环境：

包含 AT32F435/435 和 16bit 的 8080 接口 LCD 触摸屏 MRB2801 的电路板。

2) 软件环境：

project\at_start_f437\examples\xmc\lcd_touch_16bit

4.7.3 软件设计

1) 配置流程

- XMC 初始化；
- LCD 屏各项参数配置；
- 开启背光；
- 清除屏幕：在全屏所有像素点输出同一个颜色；
- 初始化触摸模块：SPI 和 GPIO 初始化；

- 循环检测触摸点，并在触摸点所在坐标及附近几个坐标的像素点的输出另一个颜色，以绘图案。

2) 代码介绍

- xmc 接口初始化

```
void xmc_init(void)
{
  gpio_init_type    gpio_init_struct = {0};
  xmc_norsram_init_type    xmc_norsram_init_struct;
  xmc_norsram_timing_init_type rw_timing_struct, w_timing_struct;

  /* enable the gpio clock */
  crm_periph_clock_enable(CRM_GPIOC_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOE_PERIPH_CLOCK, TRUE);

  /* enable the xmc clock */
  crm_periph_clock_enable(CRM_XMC_PERIPH_CLOCK, TRUE);

  /*-- gpio configuration ----------------------------------------------------*/
  gpio_pin_mux_config(GPIOC, GPIO_PINS_SOURCE2, GPIO_MUX_14);
  …
  /* lcd bl lines configuration */
  gpio_init_struct.gpio_pins = GPIO_PINS_0;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init(GPIOC, &gpio_init_struct);

  /*-- xmc configuration -----------------------------------------------------*/
  xmc_norsram_default_para_init(&xmc_norsram_init_struct);
  xmc_norsram_init_struct.subbank = XMC_BANK1_NOR_SRAM4; /*使用 XMC BANK1 的子 bank4*/
  xmc_norsram_init_struct.data_addr_multiplex = XMC_DATA_ADDR_MUX_DISABLE; /*非复用*/
  xmc_norsram_init_struct.device = XMC_DEVICE_SRAM; /*器件类型选择默认的 SRAM*/
  xmc_norsram_init_struct.bus_type = XMC_BUSTYPE_16_BITS; /*操作宽度：16bit*/
  xmc_norsram_init_struct.burst_mode_enable = XMC_BURST_MODE_DISABLE;/*同步模式：关*/
  xmc_norsram_init_struct.asynwait_enable = XMC_ASYN_WAIT_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.wait_signal_lv = XMC_WAIT_SIGNAL_LEVEL_LOW; /*无无需使用，保持默认*/
  xmc_norsram_init_struct.wrapped_mode_enable = XMC_WRAPPED_MODE_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.wait_signal_config = XMC_WAIT_SIGNAL_SYN_BEFORE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.write_enable = XMC_WRITE_OPERATION_ENABLE; /*写使能：开*/
  xmc_norsram_init_struct.wait_signal_enable = XMC_WAIT_SIGNAL_DISABLE; /*无需使用，保持默认*/
  xmc_norsram_init_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE; /*独立写时序：开*/
```

```
  xmc_norsram_init_struct.write_burst_syn = XMC_WRITE_BURST_SYN_DISABLE; /*无需使用，保持默认*/
  xmc_nor_sram_init(&xmc_norsram_init_struct);

  /* timing configuration */
  xmc_norsram_timing_default_para_init(&rw_timing_struct, &w_timing_struct);
  rw_timing_struct.subbank = XMC_BANK1_NOR_SRAM4;
  rw_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE;
  rw_timing_struct.addr_setup_time = 0x2; /*设置读/写时序*/
  rw_timing_struct.addr_hold_time = 0x0;
  rw_timing_struct.data_setup_time = 0x2;
  rw_timing_struct.bus_latency_time = 0x0;
  rw_timing_struct.clk_psc = 0x0;
  rw_timing_struct.data_latency_time = 0x0;
  rw_timing_struct.mode = XMC_ACCESS_MODE_A; /*使用 modeA*/
  w_timing_struct.subbank = XMC_BANK1_NOR_SRAM4;
  w_timing_struct.write_timing_enable = XMC_WRITE_TIMING_ENABLE;
  w_timing_struct.addr_setup_time = 0x2; /*设置写时序*/
  w_timing_struct.addr_hold_time = 0x0;
  w_timing_struct.data_setup_time = 0x2;
  w_timing_struct.bus_latency_time = 0x0;
  w_timing_struct.clk_psc = 0x0;
  w_timing_struct.data_latency_time = 0x0;
  w_timing_struct.mode = XMC_ACCESS_MODE_A; /*使用 modeA*/
  xmc_nor_sram_timing_config(&rw_timing_struct, &w_timing_struct);

  /* bus turnaround phase for consecutive read duration and consecutive write duration */
  xmc_ext_timing_config(XMC_BANK1_NOR_SRAM4, 0x08, 0x08);/*额外时序—保持默认值*/

  /* enable xmc_bank1_nor_sram4 */
  xmc_nor_sram_enable(XMC_BANK1_NOR_SRAM4, TRUE); /*使能 BANK1 的子 bank1*/
}
```

■ lcd 初始化函数代码描述

```
void lcd_init(void)
{
  /* init xmc */
  lcd_struct->xmc_init();/*XMC 界面和 GPIO 初始化*/

  delay_ms(50);

  /* read id */
  lcd_wr_command(0x0000);
  delay_ms(5);
```

```
lcd_struct->lcd_id = lcd_rd_data();
lcd_wr_command(0xd3);

lcd_struct->lcd_id = lcd_rd_data();
lcd_struct->lcd_id = lcd_rd_data();
lcd_struct->lcd_id = lcd_rd_data();
lcd_struct->lcd_id = lcd_rd_data();
lcd_struct->lcd_id = lcd_struct->lcd_id << 8;
lcd_struct->lcd_id |= lcd_rd_data();

lcd_wr_command(0xcf);  /*LCD 各项参数配置，详情请参考 MRB2801 相关资料*/
lcd_wr_data(0x00);
lcd_wr_data(0xc1);
lcd_wr_data(0x30);
lcd_wr_command(0xed);
lcd_wr_data(0x64);
lcd_wr_data(0x03);
lcd_wr_data(0x12);
lcd_wr_data(0x81);
lcd_wr_command(0xe8);
lcd_wr_data(0x85);
lcd_wr_data(0x10);
lcd_wr_data(0x7a);
lcd_wr_command(0xcb);
lcd_wr_data(0x39);
lcd_wr_data(0x2c);
lcd_wr_data(0x00);
lcd_wr_data(0x34);
lcd_wr_data(0x02);
lcd_wr_command(0xf7);
lcd_wr_data(0x20);
lcd_wr_command(0xea);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_command(0xc0);
lcd_wr_data(0x1b);
lcd_wr_command(0xc1);
lcd_wr_data(0x01);
lcd_wr_command(0xc5);
lcd_wr_data(0x30);
lcd_wr_data(0x30);
lcd_wr_command(0xc7);
lcd_wr_data(0xb7);
lcd_wr_command(0x36);
lcd_wr_data(0x48);
lcd_wr_command(0x3a);
```

```
lcd_wr_data(0x55);
lcd_wr_command(0xb1);
lcd_wr_data(0x00);
lcd_wr_data(0x1a);
lcd_wr_command(0xb6);
lcd_wr_data(0x0a);
lcd_wr_data(0xa2);
lcd_wr_command(0xf2);
lcd_wr_data(0x00);
lcd_wr_command(0x26);
lcd_wr_data(0x01);
lcd_wr_command(0xe0);
lcd_wr_data(0x0f);
lcd_wr_data(0x2a);
lcd_wr_data(0x28);
lcd_wr_data(0x08);
lcd_wr_data(0x0e);
lcd_wr_data(0x08);
lcd_wr_data(0x54);
lcd_wr_data(0xa9);
lcd_wr_data(0x43);
lcd_wr_data(0x0a);
lcd_wr_data(0x0f);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
lcd_wr_command(0xe1);
lcd_wr_data(0x00);
lcd_wr_data(0x15);
lcd_wr_data(0x17);
lcd_wr_data(0x07);
lcd_wr_data(0x11);
lcd_wr_data(0x06);
lcd_wr_data(0x2b);
lcd_wr_data(0x56);
lcd_wr_data(0x3c);
lcd_wr_data(0x05);
lcd_wr_data(0x10);
lcd_wr_data(0x0f);
lcd_wr_data(0x3f);
lcd_wr_data(0x3f);
lcd_wr_data(0x0f);
lcd_wr_command(0x2b);
lcd_wr_data(0x00);
lcd_wr_data(0x00);
```

```
  lcd_wr_data(0x01);
  lcd_wr_data(0x3f);
  lcd_wr_command(0x2a);
  lcd_wr_data(0x00);
  lcd_wr_data(0x00);
  lcd_wr_data(0x00);
  lcd_wr_data(0xef);
  lcd_wr_command(0x11);
  delay_ms(120);
  lcd_wr_command(0x29);
  lcd_wr_command(0x36);
  lcd_wr_data(0x08);
  lcd_wr_command(0x2a);
  lcd_wr_data(0x00);
  lcd_wr_data(0x00);
  lcd_wr_data(0xef >> 8);
  lcd_wr_data(0xef & 0xff);
  lcd_wr_command(0x2b);
  lcd_wr_data(0x00);
  lcd_wr_data(0x00);
  lcd_wr_data(0x13f >> 8);
  lcd_wr_data(0x13f & 0xff);
  LCD_BL_HIGH;   /*打开背光*/
}
```

■ 触摸模块初始化

```
void touch_pin_init(void)
{
  gpio_init_type   gpio_init_struct = {0};
  spi_init_type    spi_init_struct;
  /* enable the gpio clock */
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOD_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_SPI4_PERIPH_CLOCK, TRUE);
  gpio_pin_mux_config(GPIOB, GPIO_PINS_SOURCE13, GPIO_MUX_6);/*GPIO 配置*/
  …
  gpio_bits_set(GPIOB, GPIO_PINS_11);
  spi_default_para_init(&spi_init_struct); /*SPI 初始化*/
  spi_init_struct.transmission_mode = SPI_TRANSMIT_FULL_DUPLEX;
  spi_init_struct.master_slave_mode = SPI_MODE_MASTER;
  spi_init_struct.mclk_freq_division = SPI_MCLK_DIV_256;
  spi_init_struct.first_bit_transmission = SPI_FIRST_BIT_MSB;
  spi_init_struct.frame_bit_num = SPI_FRAME_8BIT;
  spi_init_struct.clock_polarity = SPI_CLOCK_POLARITY_HIGH;
  spi_init_struct.clock_phase = SPI_CLOCK_PHASE_2EDGE;
```

```
  spi_init_struct.cs_mode_selection = SPI_CS_SOFTWARE_MODE;
  spi_init(SPI4, &spi_init_struct);
  spi_enable(SPI4, TRUE);
}
```

■ 触摸点检测并显示函数

```
void touch_scan(void)
{
  uint16_t x, y;
  if(gpio_input_data_bit_read(GPIOB, GPIO_PINS_11) == 0)/*检测是否有触摸点*/
  {
    touch_dev_struct.touch_read_xy(&touch_dev_struct.x_p[1], &touch_dev_struct.y_p[1]); /*读取触摸点坐标*/
    x = (240 * touch_dev_struct.x_p[1]) / (0xed0);
    y = (320 * touch_dev_struct.y_p[1]) / (0xe60);

    lcd_drawpoint(x, y, RED); /*在触摸点坐标对应的像素点及其周围几个像素点显示红色*/
    lcd_drawpoint(x - 1, y, RED);
    lcd_drawpoint(x + 1, y, RED);
    lcd_drawpoint(x, y + 1, RED);
    lcd_drawpoint(x, y - 1, RED);
  }
}
```

■ main 函数代码描述

```
int main(void)
{
  …
  lcd_struct->lcd_init();/*LCD 初始化--包括 XMC 初始化，GPIO 初始化，包括 LCD 参数设置*/
  point_color = GBLUE;
  lcd_struct->lcd_clear(point_color);/*清除屏幕显示—全屏输出 GBLUE  颜色*/

  touch_struct->init();/*触摸模块初始化—SPI 通信接口和 GPIO 初始化*/
  while(1)
  {
    touch_struct->touch_scan();/*检测触摸点，并读取坐标，并在该坐标及附近几个点显示红色*/
  }
}
```

5 文档版本历史

表 29 文档版本历史

日期	版本	变更
2022.04.01	2.0.0	最初版本
2022.07.01	2.0.1	增加驱动16位宽LCD的D/C线使用说明

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。