目录

1、TFTLCD驱动原理-TFTLCD简介

TFTLCD即薄膜晶体管液晶显示器。它与无源TN-LCD、STN-LCD的简单矩阵不同，它在液晶显示屏的每一个象素上都设置有一个薄膜晶体管（TFT），可有效地克服非选通时的串扰，使显示液晶屏的静态特性与扫描线数无关，因此大大提高了图像质量。

TFTLCD具有：亮度好、对比度高、层次感强、颜色鲜艳等特点。是目前最主流的LCD显示器。广泛应用于电视、手机、电脑、平板等各种电子产品。

1、TFTLCD驱动原理-模块简介

◆ALINETEK TFT

ALIENTEK提供丰富的TFTLCD

1，ATK-2.8寸 TFTLCD模块

分辨率：240*320，驱动

2，ATK-3.5寸 TFTLCD模块

分辨率：320*480，驱动

3，ATK-4.3寸 TFTLCD模块

分辨率：480*800，驱动

4，ATK-7寸 TFTLCD模块(V1

分辨率：480*800，驱动

5，ATK-7寸 TFTLCD模块(V2

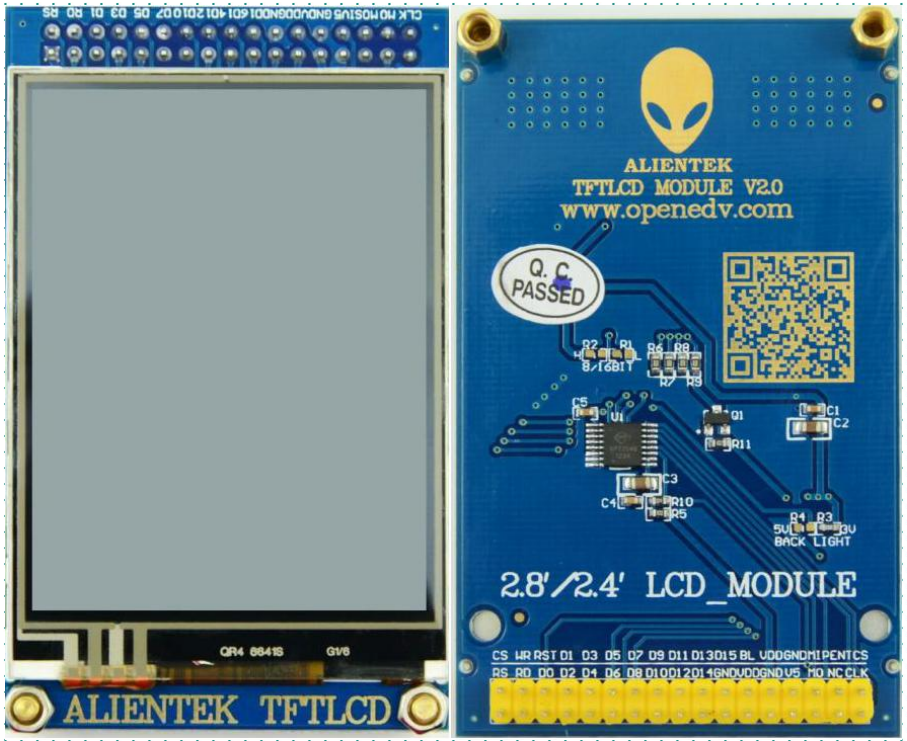
分辨率：480*800，驱动

1、TFTLCD驱动原理-模块简介

◆ALINETEK 2.8寸 TFTLCD模块特点

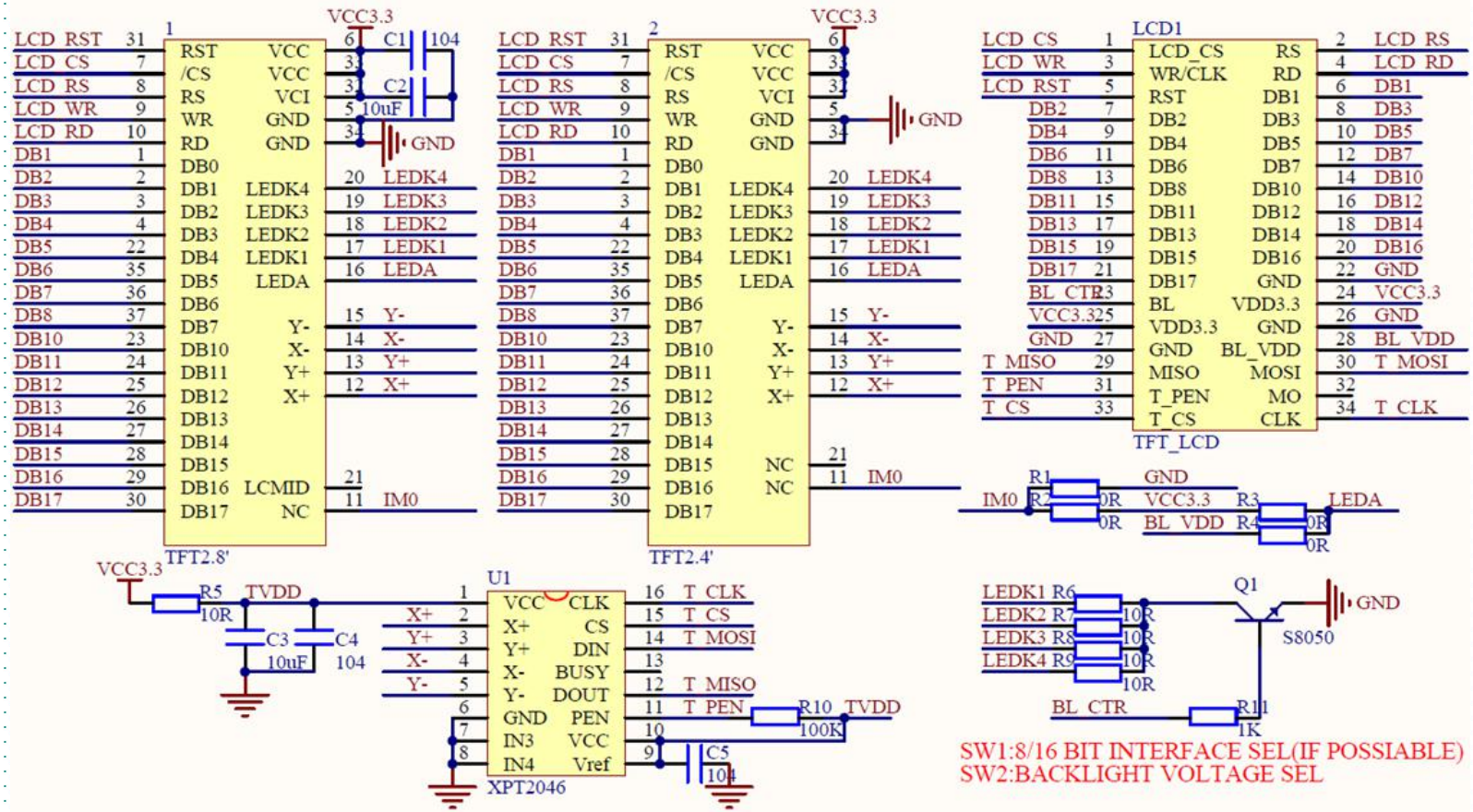
- 240*320分辨率
- 16位真彩显示（65536色）
- 自带电阻触摸屏
- 自带背光电路

注意：模块是3.3V供电的，不支持5V电压的MCU，如果是5V MCU，必须在信号线串接120R电阻使用。



1、TFTLCD驱动原理-模块简介

◆ALINETEK 2.8寸 TFTLCD模块原理图



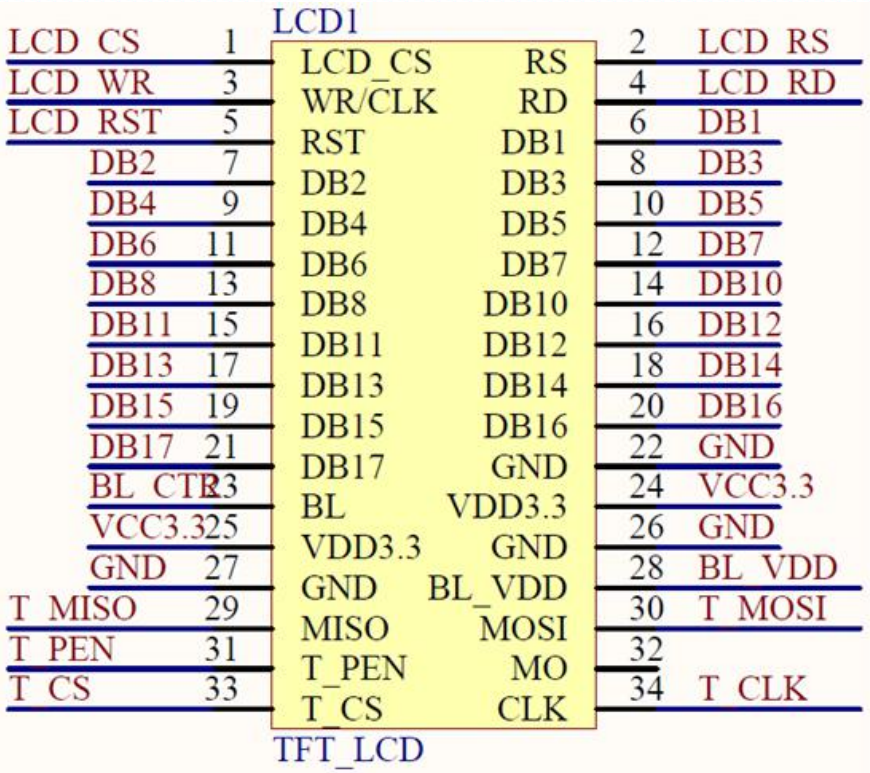
1、TFTLCD驱动原理-模块简介

◆ALINETEK 2.8寸 TFTLCD接口说明（16位80并口）

注意：DB1~DB8，DB10~DB17，总是按顺序连接MCU的D0~D15

- LCD_CS：LCD片选信号
- LCD_WR：LCD写信号
- LCD_RD：LCD读信号
- DB[17：1]：16位双向数据线。
- LCD_RST：硬复位LCD信号
- LCD_RS：命令/数据标志

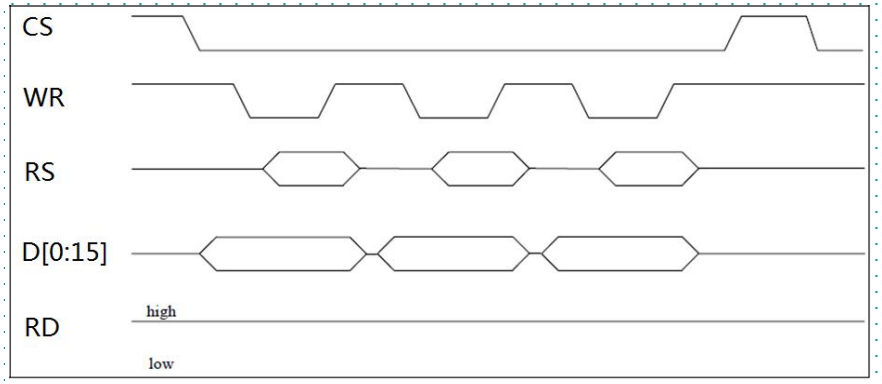
  (0:命令,1:数据)
- BL_CTR：背光控制信号
- T_MISO/T_MOSI/T_PEN/T_CS/T_CLK，触摸屏接口信号



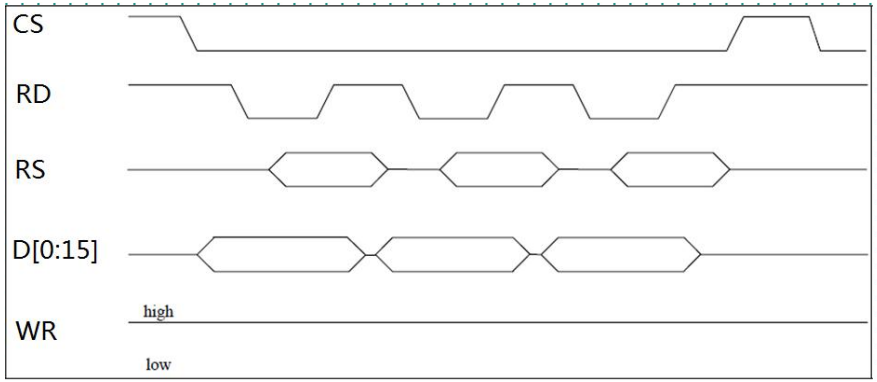
1、TFTLCD驱动原理-模块简介

◆ALINETEK 2.8寸 TFTLCD 16位80并口驱动简介

模块的8080并口读/写的过程为：
先根据要写入/读取的数据的类型，设置RS为高（数据）/低（命令），然后拉低片选，选中ILI9341，接着我们根据是读数据，还是要写数据置RD/WR为低，然后：
1.读数据：在RD的上升沿，读取数据线上的数据（D[15:0]）；
2.写数据：在WR的上升沿，使数据写入到ILI9341里面



并口写时序图

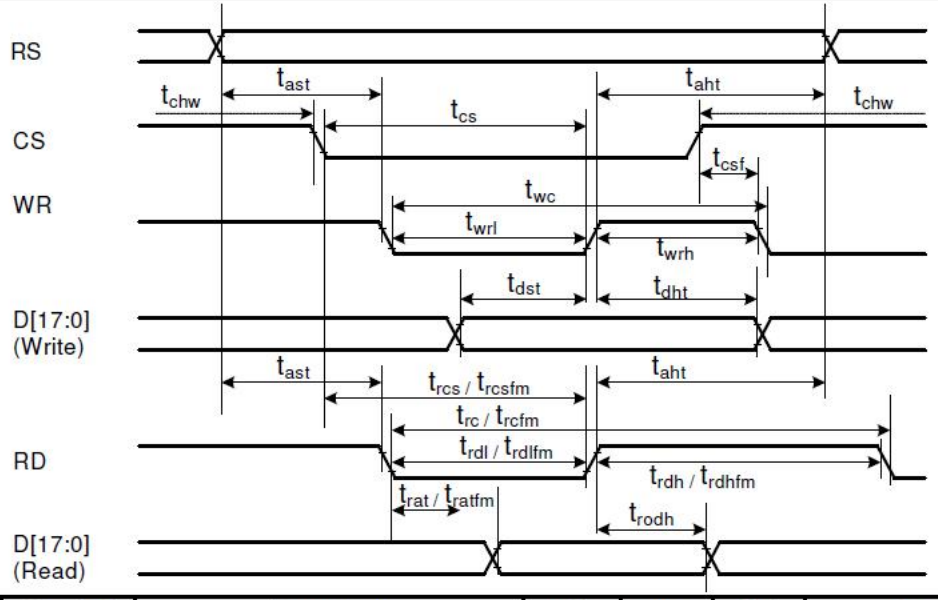


并口读时序图

1、TFTLCD驱动原理-模块简介

◆ILI9341 驱动时序

右图为：ILI9341 8080并口时序，
详见：ILI9341_DS.pdf，232页



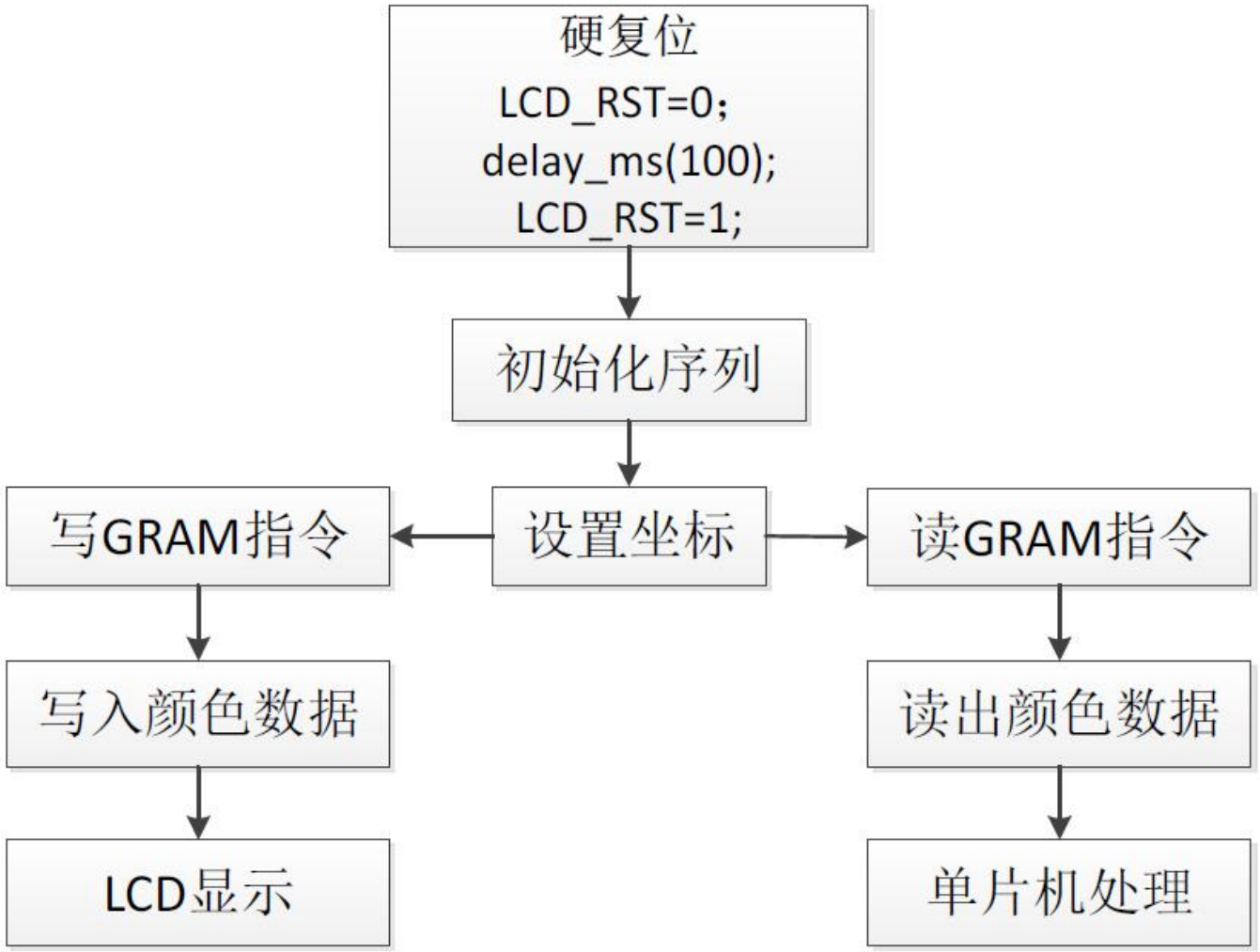
重点时序：

- 读ID低电平脉宽(trdl)
- 读ID高电平脉宽(trdh)
- 读FM低电平脉宽(trdlfm)
- 读FM高电平脉宽(trdhfm)
- 写控制低电平脉宽(twrl)
- 写控制高电平脉宽(twrh)

注意：ID指LCD的ID号
FM指帧缓存，即:GRAM

Signal	Symbol	Parameter	min	max	Unit	Description
RS	tast	Address setup time	0	-	ns	
	taht	Address hold time (Write/Read)	0	-	ns	
CSX	tchw	CSX "H" pulse width	0	-	ns	
	tcs	Chip Select setup time (Write)	15	-	ns	
	trcs	Chip Select setup time (Read ID)	45	-	ns	
	trcsfm	Chip Select setup time (Read FM)	355	-	ns	
	tcsf	Chip Select Wait time (Write/Read)	10	-	ns	
WRX	twc	Write cycle	66	-	ns	
	twrh	Write Control pulse H duration	15	-	ns	
	twrl	Write Control pulse L duration	15	-	ns	
RD(FM)	trcfm	Read Cycle (FM)	450	-	ns	
	trdhfm	Read Control H duration (FM)	90	-	ns	
	trdlfm	Read Control L duration (FM)	355	-	ns	
RD (ID)	trc	Read cycle (ID)	160	-	ns	
	trdh	Read Control pulse H duration	90	-	ns	
	trdl	Read Control pulse L duration	45	-	ns	

1、TFTLCD驱动原理-驱动流程



1、TFTLCD驱动原理-指令简介

◆RGB565格式说明

模块对外接口采用16位并口，颜色深度为16位，格式为RGB565，关系如下图：

数据线	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
LCD GRAM	R[4]	R[3]	R[2]	R[1]	R[0]	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[4]	B[3]	B[2]	B[1]	B[0]

◆ILI9341指令格式说明

ILI9341所有的指令都是8位的（高8位无效），且参数除了读写GRAM的时候是16位，其他操作参数，都是8位的。

ILI9341的指令很多，这里不一一介绍，仅介绍几个重要的指令，他们是：0XD3，0X36，0X2A，0X2B，0X2C，0X2E等6条指令。

1、TFTLCD驱动原理-指令简介

◆ 0XD3指令

该指令为读ID4指令，用于读取LCD控制器的ID 。因此，同一个代码，可以根据ID的不同，执行不同的LCD驱动初始化，以兼容不同的LCD屏幕。

顺序	控制			各位描述								HEX
	RS	RD	WR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX	1	1	0	1	0	0	1	1	D3H
参数 1	1	↑	1	XX	X	X	X	X	X	X	X	X	X
参数 2	1	↑	1	XX	0	0	0	0	0	0	0	0	00H
参数 3	1	↑	1	XX	1	0	0	1	0	0	1	1	93H
参数 4	1	↑	1	XX	0	1	0	0	0	0	0	1	41H

1、TFTLCD驱动原理-指令简介

◆0X36指令

该指令为存储访问控制指令，可以控制ILI9341存储器的读写方向，简单的说，就是在连续写GRAM的时候，可以控制GRAM指针的增长方向，从而控制显示方式（读GRAM也是一样）。

顺序	控制			各位描述								HEX
	RS	RD	WR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX	0	0	1	1	0	1	1	0	36H
参数	1	1	↑	XX	MY	MX	MV	ML	BGR	MH	0	0	0

控制位			效果
MY	MX	MV	LCD 扫描方向（GRAM 自增方式）
0	0	0	从左到右,从上到下
1	0	0	从左到右,从下到上
0	1	0	从右到左,从上到下
1	1	0	从右到左,从下到上
0	0	1	从上到下,从左到右
0	1	1	从上到下,从右到左
1	0	1	从下到上,从左到右
1	1	1	从下到上,从右到左

1、TFTLCD驱动原理-指令简介

◆0X2A指令

该指令是列地址设置指令，在从左到右，从上到下的扫描方式（默认）下面，该指令用于设置横坐标（x坐标）

顺序	控制			各位描述								HEX
	RS	RD	WR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX	0	0	1	0	1	0	1	0	2AH
参数 1	1	1	↑	XX	SC15	SC14	SC13	SC12	SC11	SC10	SC9	SC8	SC
参数 2	1	1	↑	XX	SC7	SC6	SC5	SC4	SC3	SC2	SC1	SC0	
参数 3	1	1	↑	XX	EC15	EC14	EC13	EC12	EC11	EC10	EC9	EC8	EC
参数 4	1	1	↑	XX	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0	

在默认扫描方式时，该指令用于设置x坐标，该指令带有4个参数，实际上是2个坐标值：SC和EC，即列地址的起始值和结束值，SC必须小于等于EC，且0≤SC/EC≤239。一般在设置x坐标的时候，我们只需要带2个参数即可，也就是设置SC即可，因为如果EC没有变化，我们只需要设置一次即可（在初始化ILI9341的时候设置），从而提高速度。

1、TFTLCD驱动原理-指令简介

◆0X2B指令

该指令是页地址设置指令，在从左到右，从上到下的扫描方式（默认）下面，该指令用于设置纵坐标（y坐标）

顺序	控制			各位描述								HEX
	RS	RD	WR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX	0	0	1	0	1	0	1	0	2BH
参数 1	1	1	↑	XX	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SP
参数 2	1	1	↑	XX	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	
参数 3	1	1	↑	XX	EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8	EP
参数 4	1	1	↑	XX	EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0	

在默认扫描方式时，该指令用于设置y坐标，该指令带有4个参数，实际上是2个坐标值：SP和EP，即页地址的起始值和结束值，SP必须小于等于EP，且0≤SP/EP≤319。一般在设置y坐标的时候，我们只需要带2个参数即可，也就是设置SP即可，因为如果EP没有变化，我们只需要设置一次即可（在初始化ILI9341的时候设置），从而提高速度。

1、TFTLCD驱动原理-指令简介

◆ 0X2C指令

该指令是写GRAM指令，在发送该指令之后，我们便可以往LCD的GRAM里面写入颜色数据了，该指令支持连续写 (地址自动递增)

顺序	控制			各位描述									HEX
	RS	RD	WR	D15~D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX	0	0	1	0	1	1	0	0	2CH
参数 1	1	1	↑	D1[15：0]									XX
……	1	1	↑	D2[15：0]									XX
参数 n	1	1	↑	Dn[15：0]									XX

在收到指令0X2C之后，数据有效位宽变为16位，我们可以连续写入LCD GRAM值，而GRAM的地址将根据MY/MX/MV设置的扫描方向进行自增。例如：假设设置的是从左到右，从上到下的扫描方式，那么设置好起始坐标（通过SC，SP设置）后，每写入一个颜色值，GRAM地址将会自动自增1（SC++），如果碰到EC，则回到SC，同时SP++，一直到坐标：EC，EP结束，其间无需再次设置的坐标，从而大大提高写入速度。

1、TFTLCD驱动原理-指令简介

◆0X2E指令

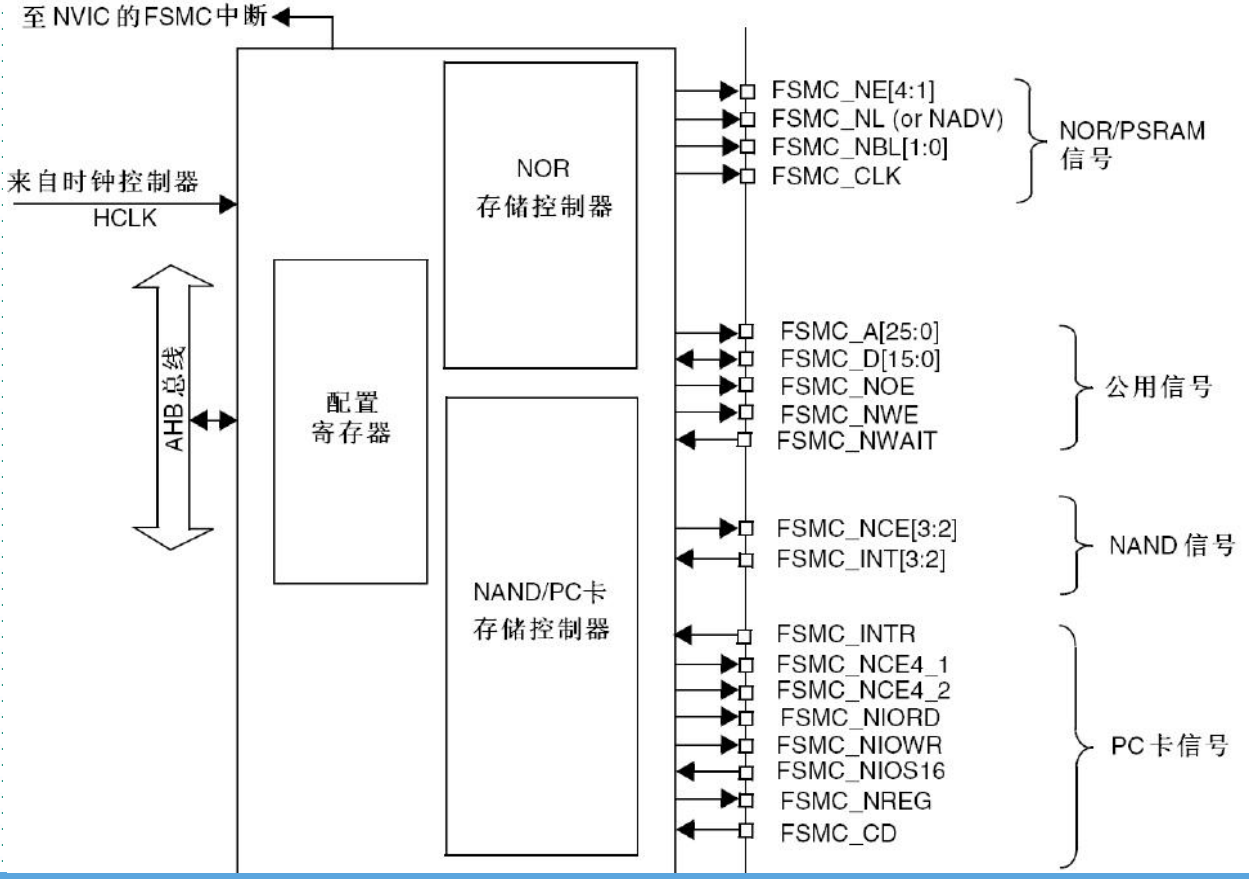
该指令是读GRAM指令，用于读取ILI9341的显存（GRAM），同0X2C指令，该指令支持连续读 (地址自动递增)

顺序	控制			各位描述												HEX
	RS	RD	WR	D15~D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
指令	0	1	↑	XX				0	0	1	0	1	1	1	0	2EH
参数 1	1	↑	1	XX												dummy
参数 2	1	↑	1	R1[4:0]		XX		G1[5:0]						XX		R1G1
参数 3	1	↑	1	B1[4:0]		XX		R2[4:0]					XX			B1R2
参数 4	1	↑	1	G2[5:0]			XX	B2[4:0]					XX			G2B2
参数 5	1	↑	1	R3[4:0]		XX		G3[5:0]						XX		R3G3
参数 N	1	↑	1	按以上规律输出												

ILI9341在收到该指令后，第一次输出的是dummy数据（无效），第二次开始，读取到的才是有效的GRAM数据（从坐标：SC，SP开始），输出规律为：每个颜色分量占8个位，一次输出2个颜色分量。比如：第一次输出是R1G1，随后的规律为：B1R2→G2B2→R3G3→B3R4→G4B4→R5G5... 以此类推

2、FSMC简介-FSMC介绍

FSMC，即灵活的静态存储控制器，能够与同步或异步存储器和16位PC存储器卡连接，STM32的FSMC接口支持包括SRAM、NAND FLASH、NOR FLASH和PSRAM等存储器。FSMC的框图如下图所示：



2、FSMC简介-FSMC驱动LCD原理

FSMC驱动外部SRAM时，外部SRAM的控制一般有：地址线（如A0~A25）、数据线（如D0~D15）、写信号（WE，即WR）、读信号（OE，即RD）、片选信号（CS），如果SRAM支持字节控制，那么还有UB/LB信号。

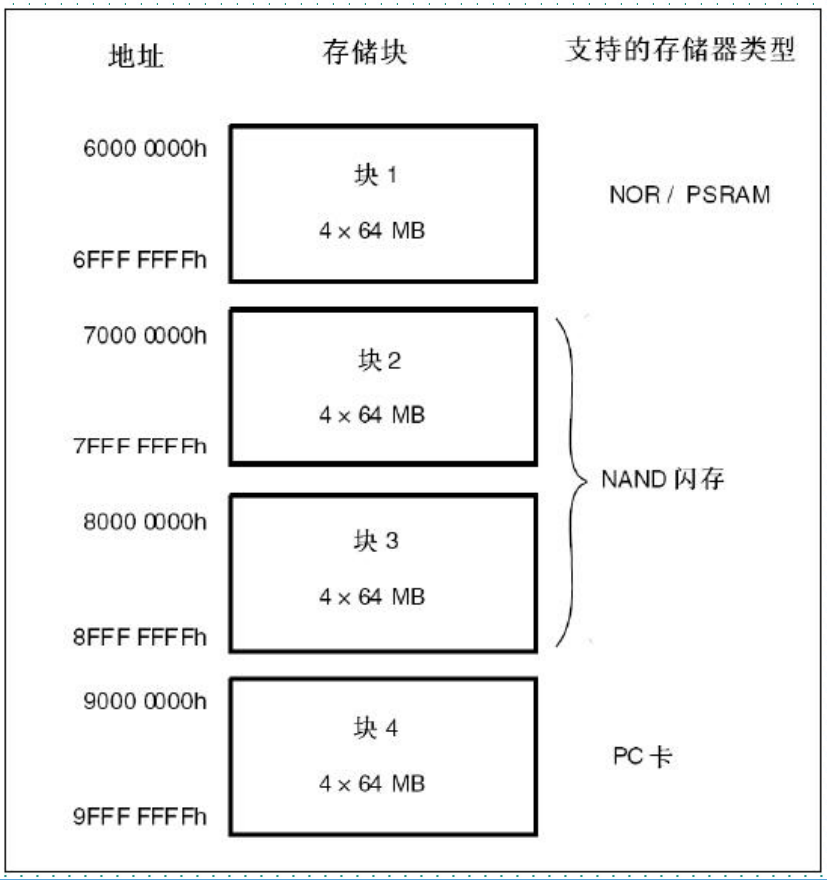
而TFTLCD的信号我们在前面介绍过，包括：RS、D0~D15、WR、RD、CS、RST和BL等，其中真正在操作LCD的时候需要用到的就只有：RS、D0~D15、WR、RD和CS。其操作时序和SRAM的控制完全类似，唯一不同就是TFTLCD有RS信号，但是没有地址信号。

TFTLCD通过RS信号来决定传送的数据是数据还是命令，本质上可以理解为一个地址信号，比如我们把RS接在A0上面，那么当FSMC控制器写地址0的时候，会使得A0变为0，对TFTLCD来说，就是写命令。而FSMC写地址1的时候，A0将会变为1，对TFTLCD来说，就是写数据了。这样，就把数据和命令区分开了，他们其实就是对应SRAM操作的两个连续地址。当然RS也可以接在其他地址线上，战舰V3和精英板开发板都是把RS连接在A10上面，而探索者STM32F4把RS接在A6上面。

因此，可以把TFTLCD当成一个SRAM来用，只不过这个SRAM有2个地址，这就是FSMC可以驱动LCD的原理。

2、FSMC简介-NOR PSRAM外设接口

STM32的FSMC支持8/16/32位数据宽度，我们这里用到的LCD是16位宽度的，所以在设置的时候，选择16位宽就OK了。FSMC的外部设备地址映像，STM32的FSMC将外部存储器划分为固定大小为256M字节的四个存储块



2、FSMC简介-存储块1 操作简介

STM32的FSMC存储块1（Bank1）用于驱动NOR FLASH/SRAM/PSRAM，被分为4个区，每个区管理64M字节空间，每个区都有独立的寄存器对所连接的存储器进行配置。Bank1的256M字节空间由28根地址线（HADDR[27:0]）寻址。

这里HADDR，是内部AHB地址总线，其中，HADDR[25:0]来自外部存储器地址FSMC_A[25:0]，而HADDR[26:27]对4个区进行寻址。如下表所示：

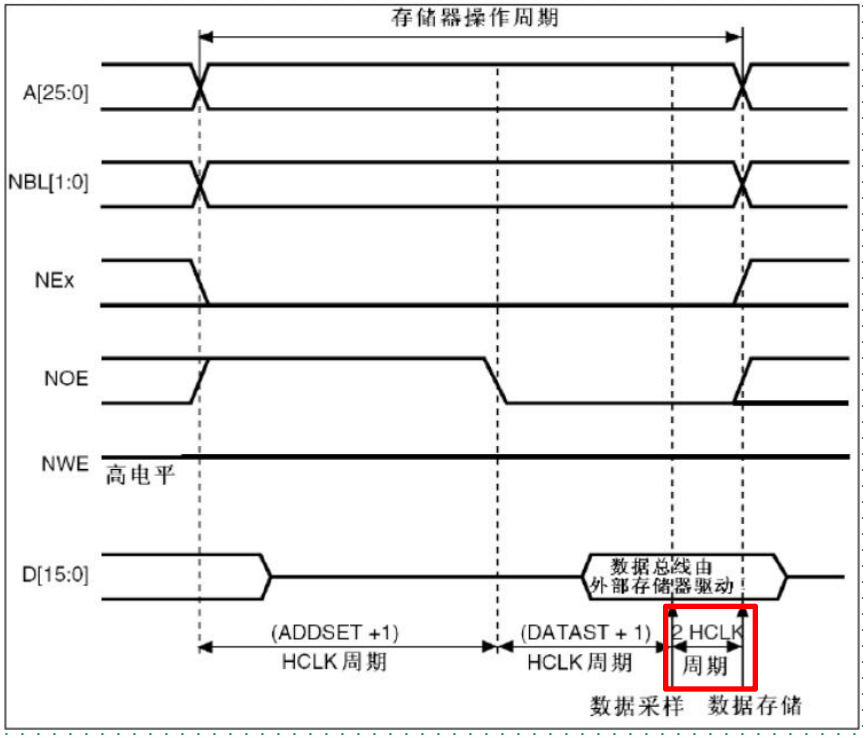
Bank1 所选区	片选信号	地址范围	HADDR	
			[27:26]	[25:0]
第 1 区	FSMC_NE1	0X6000, 0000~63FF, FFFF	00	FSMC_A[25:0]
第 2 区	FSMC_NE2	0X6400, 0000~67FF, FFFF	01	
第 3 区	FSMC_NE3	0X6800, 0000~6BFF, FFFF	10	
第 4 区	FSMC_NE4	0X6C00, 0000~6FFF, FFFF	11	

当Bank1接的是16位宽度存储器的时候：HADDR[25:1]→ FSMC_A[24:0]
当Bank1接的是8位宽度存储器的时候：HADDR[25:0]→ FSMC_A[25:0]

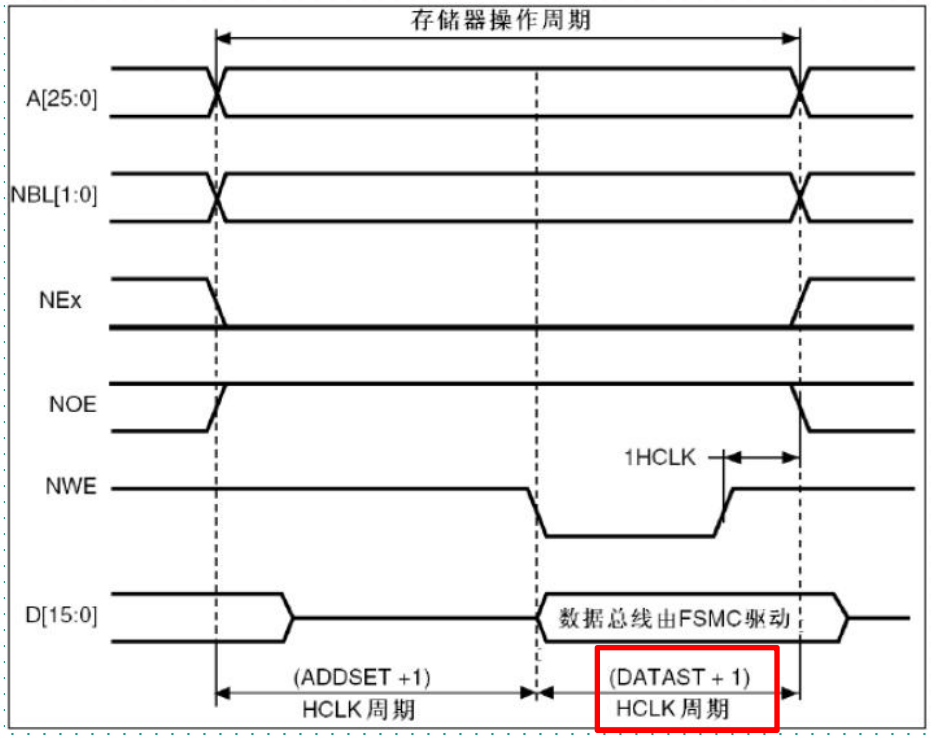
不论外部接8位/16位宽设备，FSMC_A[0]永远接在外部设备地址A[0]

2、FSMC简介-存储块1 操作简介

STM32的FSMC存储块1 支持的异步突发访问模式包括：模式1、模式A~D等多种时序模型，驱动SRAM时一般使用模式1或者模式 A，这里我们使用模式A来驱动LCD（当SRAM用），其他模式说明详见：STM32中文参考手册-FSMC章节。



模式A读时序图



模式A写时序图

模式A支持读写时序分开设置！ 对STM32F4仅写时序DATAST需要+1

2、FSMC简介-寄存器介绍

对于NOR FLASH/PSRAM控制器(存储块1)，通过FSMC_BCRx、FSMC_BTRx和FSMC_BWTRx寄存器设置（其中x=1~4，对应4个区）。通过这3个寄存器，可以设置FSMC访问外部存储器的时序参数，拓宽了可选用的外部存储器的速度范围。

◆ SRAM/NOR闪存片选控制寄存器（FSMC_BCRx）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												CBURSTRW	Reserved			ASCYCWAIT	EXTMOD	WAITEN	WREN	WAITCFG	WRAPMOD	WAITPOL	BURSTEN	Reserved	FACCEN	MWID		MTYP		MUXEN	MBKEN
												rw				rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

EXTMOD：扩展模式使能位，控制是否允许读写不同的时序，需设置为1

WREN：写使能位。我们需要向TFTLCD写数据，故该位必须设置为1

MWID[1:0]：存储器数据总线宽度。00，表示8位数据模式；01表示16位数据模式；10和11保留。我们的TFTLCD是16位数据线，所以设置WMID[1:0]=01。

MTYP[1:0]：存储器类型。00表示SRAM、ROM；01表示PSRAM；10表示NOR FLASH;11保留。我们把LCD当成SRAM用，所以需要设置MTYP[1:0]=00。

MBKEN：存储块使能位。需设置为1

2、FSMC简介-寄存器介绍

◆ SRAM/NOR闪存片选时序寄存器（FSMC_BTRx）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		ACCMOD		DATLAT				CLKDIV				BUSTURN				DATAST								ADDHLD				ADDSET			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ACCMOD[1:0]：访问模式。00:模式A；01:模式B；10:模式C；11:模式D。
DATAST[7:0]：数据保持时间，等于: DATAST**(+1)**个HCLK时钟周期，DATAST最大为255。对ILI9341来说，其实就是RD低电平持续时间，最大为355ns。对STM32F1，一个HCLK=13.8ns (1/72M)，设置为15；对STM32F4，一个HCLK=6ns(1/168M)，设置为60。
ADDSET[3:0]：地址建立时间。表示：ADDSET **(+1)**个HCLK周期，ADDSET最大为15。对ILI9341来说，这里相当于RD高电平持续时间，为90ns。STM32F1的FSMC性能存在问题，即便设置为0，RD也有190ns的高电平，我们这里设置为1。而对STM32F4，则设置为15。

如果未设置EXTMOD位，则读写共用这个时序寄存器！

2、FSMC简介-寄存器介绍

◆ SRAM/NOR闪存写时序寄存器（FSMC_BWTRx）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.		ACCMOD		DATLAT				CLKDIV				BUSTURN				DATAST								ADDHLD				ADDSET			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ACCMOD[1:0]：访问模式。00:模式A；01:模式B；10:模式C；11:模式D。
DATAST[7:0]：数据保持时间，等于: DATAST**(+1)**个HCLK时钟周期，DATAST最大为255。对ILI9341来说，其实就是WR低电平持续时间，为15ns，不过ILI9320等则需要50ns。考虑兼容性，对STM32F1，一个HCLK=13.8ns (1/72M)，设置为3；对STM32F4，一个HCLK=6ns(1/168M)，设置为9。
ADDSET[3:0]：地址建立时间。表示：ADDSET+1个HCLK周期，ADDSET最大为15。对ILI9341来说，这里相当于WR高电平持续时间，为15ns。同样考虑兼容ILI9320，对STM32F1，这里即便设置为1，WR也有100ns的高电平，我们这里设置为1。而对STM32F4，则设置为8。

2、FSMC简介-寄存器介绍

◆ 寄存器组合说明

在ST官方库提供的的寄存器定义里面，并没有定义FSMC_BCRx、FSMC_BTRx、FSMC_BWTRx等这个单独的寄存器，而是将他们进行了一些组合。规律如下：

FSMC_BCRx和FSMC_BTRx，组合成BTCR[8]寄存器组，他们的对应关系如下：
BTCR[0]对应FSMC_BCR1，BTCR[1]对应FSMC_BTR1
BTCR[2]对应FSMC_BCR2，BTCR[3]对应FSMC_BTR2
BTCR[4]对应FSMC_BCR3，BTCR[5]对应FSMC_BTR3
BTCR[6]对应FSMC_BCR4，BTCR[7]对应FSMC_BTR4

FSMC_BWTRx则组合成BWTR[7]，他们的对应关系如下：
BWTR[0]对应FSMC_BWTR1，BWTR[2]对应FSMC_BWTR2，
BWTR[4]对应FSMC_BWTR3，BWTR[6]对应FSMC_BWTR4，
BWTR[1]、BWTR[3]和BWTR[5]保留，没有用到。

3、源码讲解

1. 硬件连接
2. LCD&lcddev结构体讲解
3. 底层接口函数讲解
4. 初始化函数讲解
5. 坐标设置函数讲解
6. 画点函数讲解
7. 读点函数讲解
8. 字符显示函数讲解

3、源码讲解

1，MiniSTM32开发板硬件连接

2，精英STM32F103开发板硬件连接

3，战舰V3 STM32F103开发板硬件连接

4，探索者STM32F407开发板硬件连接

3、源码讲解-LCD结构体（Mini板没有）

```
//LCD地址结构体
typedef struct
{
        vu16 LCD_REG;
        vu16 LCD_RAM;
} LCD_TypeDef;
//使用NOR/SRAM的 Bank1.sector4,地址位HADDR[27,26]=11 A10作为数据命令区分线
//注意设置时STM32内部会右移一位对其!
#define LCD_BASE        ((u32)(0x6C000000 | 0x000007FE))
#define LCD             ((LCD_TypeDef *) LCD_BASE)
```

探索者**F4,RS接A6，则| 0x0000007E**

LCD_BASE，须根据外部电路的连接来确定，如Bank1.sector4就是从地址0X6C000000开始，而0X000007FE，则是A10的偏移量。以A10为例，7FE换成二进制为：111 1111 1110，而16位数据时，地址右移一位对齐，对应到地址引脚，就是：A10:A0=011 1111 1111，此时A10是0，但是如果16位地址再加1（对应到8位地址是加2，即7FE+0X02），那么：A10:A0=100 0000 0000，此时A10就是1了，即实现了对RS的0和1的控制。

我们将这个地址强制转换为LCD_TypeDef结构体地址，那么可以得到LCD->LCD_REG的地址就是0X6C00,07FE，对应A10的状态为0(即RS=0)，而LCD-> LCD_RAM的地址就是0X6C00,0800（结构体地址自增），对应A10的状态为1（即RS=1），从而实现对RS的控制。

3、源码讲解-lcddev结构体

```
//LCD重要参数集
typedef struct
{
	u16 width;			//LCD 宽度
	u16 height;			//LCD 高度
	u16 id;				//LCD ID
	u8  dir;			//横屏还是竖屏控制：0，竖屏；1，横屏。
	u16  wramcmd;		//开始写gram指令
	u16  setxcmd;		//设置x坐标指令
	u16  setycmd;		//设置y坐标指令
}_lcd_dev;
//LCD参数
extern _lcd_dev lcddev;	//管理LCD重要参数
```

lcddev结构体参数的赋值，基本上都是在LCD_Display_Dir函数完成

3、源码讲解-底层接口函数

7个底层接口函数：

1，写寄存器值函数 ： void LCD_WR_REG(u16 regval)

2，写数据函数：void LCD_WR_DATA(u16 data)

3，读数据函数：u16 LCD_RD_DATA(void)

4，写寄存器内容函数： void LCD_WriteReg(u16 LCD_Reg, u16 LCD_RegValue)

5，读寄存器内容函数： u16 LCD_ReadReg(u16 LCD_Reg)

6，开始写GRAM函数： void LCD_WriteRAM_Prepare(void)

7，写GRAM函数： void LCD_WriteRAM(u16 RGB_Code)

3、源码讲解-LCD初始化函数

LCD初始化函数伪代码：

```
//LCD初始化
void LCD_Init(void)
{
    初始化GPIO;
    初始化FSMC;                                  //Mini板不需要
    读取LCD ID;
    printf(“LCD ID:%x\r\n”,lcddev.id);//打印LCD ID，用到了串口1
                                                 //所以必须初始化串口1，否则黑屏
    根据不同的ID执行LCD初始化代码;
    LCD_Display_Dir(0);                          //默认为竖屏
    LCD_LED=1;                                   //点亮背光
    LCD_Clear(WHITE);                            //清屏
}
```

3、源码讲解-LCD坐标设置函数

```
//设置光标位置
//Xpos:横坐标
//Ypos:纵坐标
void LCD_SetCursor(u16 Xpos, u16 Ypos)
{
    if(lcddev.id==0X9341||lcddev.id==0X5310)
    {
        LCD_WR_REG(lcddev.setxcmd);
        LCD_WR_DATA(Xpos>>8);
        LCD_WR_DATA(Xpos&0XFF);
        LCD_WR_REG(lcddev.setycmd);
        LCD_WR_DATA(Ypos>>8);
        LCD_WR_DATA(Ypos&0XFF);
    }else if(lcddev.id==XXXX)       //根据不同的LCD型号，执行不同的代码
    {
        ……//省略部分代码
    }
}
```

3、源码讲解-LCD画点函数

```
//画点
//x,y:坐标
//POINT_COLOR:此点的颜色
void LCD_DrawPoint(u16 x,u16 y)
{
    LCD_SetCursor(x,y);              //设置光标位置
    LCD_WriteRAM_Prepare();          //开始写入GRAM
    LCD->LCD_RAM=POINT_COLOR;        //非Mini板的操作方式
}
```

MiniSTM32;

```
LCD_WR_DATA(POINT_COLOR);
```

3、源码讲解-LCD读点函数

LCD读点函数：u16 LCD_ReadPoint(u16 x,u16 y)

1，非Mini板的读点函数代码(FSMC方式，适合战舰、精英、探索者F4板)

2，Mini板的读点函数代码(GPIO方式，适合Mini板)

3、源码讲解-LCD字符显示函数

```
//在指定位置显示一个字符
//x,y:起始坐标
//num:要显示的字符:" "--->"~"
//size:字体大小 12/16/24
//mode:叠加方式(1)还是非叠加方式(0)
void LCD_ShowChar(u16 x,u16 y,u8 num,u8 size,u8 mode)
{
    u8 temp,t1,t;
    u16 y0=y;
    u8 csize=(size/8+((size%8)?1:0))*(size/2);    //得到字体一个字符对应点阵集所占的字节数
    num=num-' ';                    //得到偏移后的值（ASCII字库是从空格开始取模，所以-' '就是对应字符的字库）
    for(t=0;t<csize;t++)
    {
        if(size==12)temp=asc2_1206[num][t];                    //调用1206字体
        else if(size==16)temp=asc2_1608[num][t];               //调用1608字体
        else if(size==24)temp=asc2_2412[num][t];               //调用2412字体
        else return;                                           //没有的字库
        for(t1=0;t1<8;t1++)
        {
            if(temp&0x80)LCD_Fast_DrawPoint(x,y,POINT_COLOR);
            else if(mode==0)LCD_Fast_DrawPoint(x,y,BACK_COLOR);
             temp<<=1;
             y++;
             if(y>=lcddev.height)return;                       //超区域了
             if((y-y0)==size)
             {
                 y=y0;
                 x++;
                 if(x>=lcddev.width)return;                    //超区域了
                 break;
              }
         }
    }
}
```

3、源码讲解-LCD字符显示函数

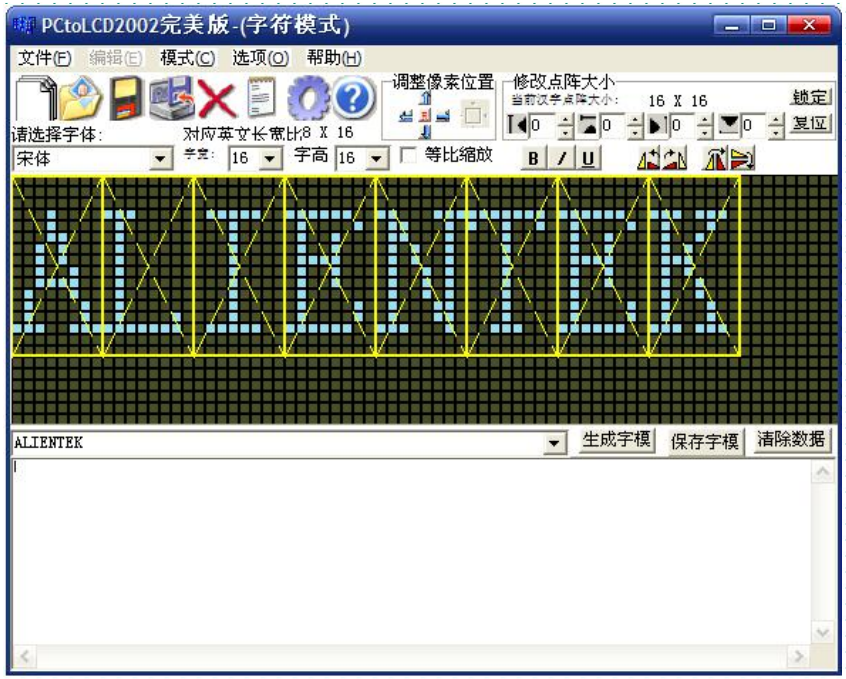
◆ 字符码表

```
const unsigned char oled_asc2_1206[95][12]={
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*" ",0*/
{0x00,0x00,0x00,0x00,0x3F,0x40,0x00,0x00,0x00,0x00,0x00,0x00},/*"!",1*/
……
{0x40,0x00,0x80,0x00,0x40,0x00,0x20,0x00,0x20,0x00,0x40,0x00},/*"~",94*/
};
```

```
const unsigned char oled_asc2_1608[95][16]={
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*" ",0*/
{0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0xCC,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00},/*"!",1*/
……
{0x00,0x00,0x60,0x00,0x80,0x00,0x80,0x00,0x40,0x00,0x40,0x00,0x20,0x00,0x20,0x00},/*"~",94*/
}
```

```
const unsigned char oled_asc2_2412[95][36]={
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*" ",0*/
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x80,0x38,0x0F,0xFE,0x38,0x0F,0x80,0x38,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*"!",1*/
……
{0x00,0x00,0x00,0x18,0x00,0x00,0x60,0x00,0x00,0x40,0x00,0x00,0x40,0x00,0x00,0x20,0x00,0x00,0x10,0x00,0x00,0x08,0x00,0x00,0x04,0x00,0x00,0x04,0x00,0x00,0x0C,0x00,0x00,0x10,0x00,0x00},/*"~",94*/
}
```

3、源码讲解-LCD字符显示函数



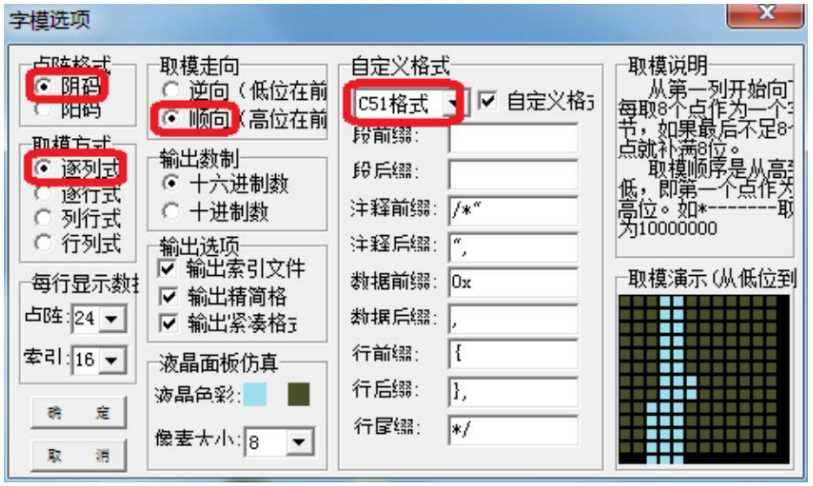


图 17.3.2 设置取模方式

上图设置的取模方式，在右上角的取模说明里面有，即：从第一列开始向下每取 8 个点作为一个字节，如果最后不足 8 个点就补满 8 位。取模顺序是从高到低，即第一个点作为最高位。如*-------取为 10000000。其实就是按如图 17.3.3 所示的这种方式：