

# JLX130-026-PN 使用说明书

## (不带字库 IC, 3.3V 供电)

### 目 录

序号	内 容 标 题	页 码
1	概述	2
2	特点	2
3	外形及接口引脚功能	3~4
4	基本原理	4
5	技术参数	4~5
6	时序特性	5~6
7	指令功能及硬件接口与编程案例	7 末页

## 1. 概述

晶联讯电子专注于液晶屏及液晶模块的研发、制造。所生产 JLX130-026-PN 型液晶模块由于使用方便、显示清晰，广泛应用于各种人机交流面板。

JLX130-026-PN 可以显示 240 列\*240 行点阵彩色图片，或显示 15 个/行\*15 行 16\*16 点阵的汉字，或显示 30 个/行\*30 行 8\*8 点阵的英文、数字、符号。

本产品可选择带中文字库 IC 与不带中文字库 IC 两种。

## 2. JLX130-026-PN 彩色图像型点阵液晶模块的特性

2.1 结构轻、薄、带背光。

2.2 IC 采用 ST7789V, IPS 全视角，功能强大，稳定性好

2.3 显示内容：

- 240\*240 点阵彩色图片；

- 可選用 32\*32 点阵或其他点阵的图片来自编汉字，按照 32\*32 点阵汉字来计算可显示 7 个字/行\*7 行。

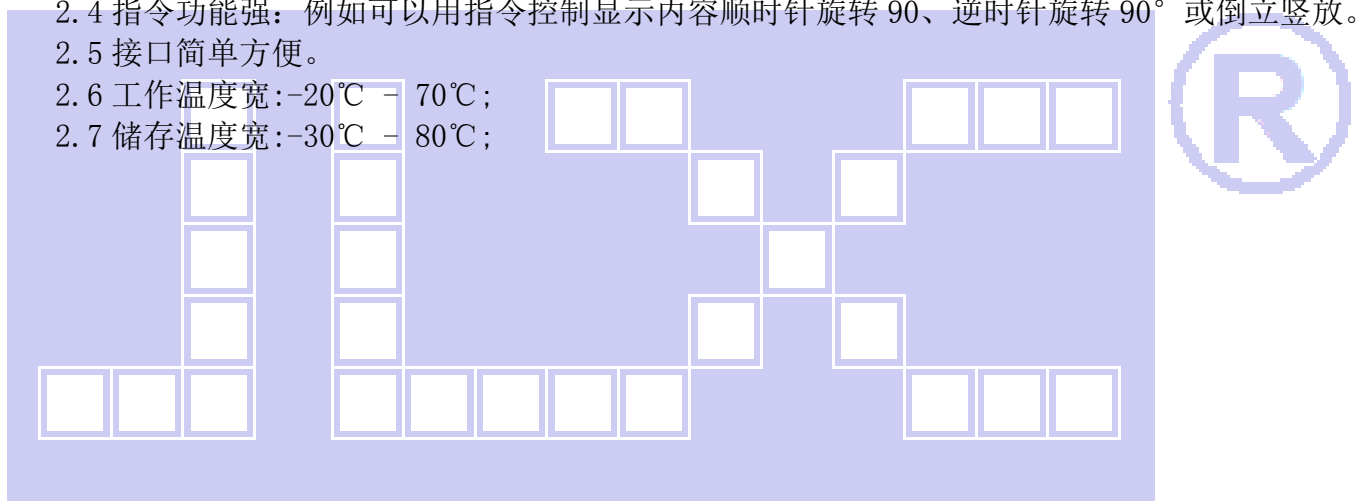
- 可選用 16\*16 点阵或其他点阵的图片来自编汉字，按照 16\*16 点阵汉字来计算可显示 15 个字/行\*15 行。

2.4 指令功能强：例如可以用指令控制显示内容顺时针旋转 90°、逆时针旋转 90° 或倒立竖放。

2.5 接口简单方便。

2.6 工作温度宽：-20℃ - 70℃；

2.7 储存温度宽：-30℃ - 80℃；



3. 外形尺寸及接口引脚功能

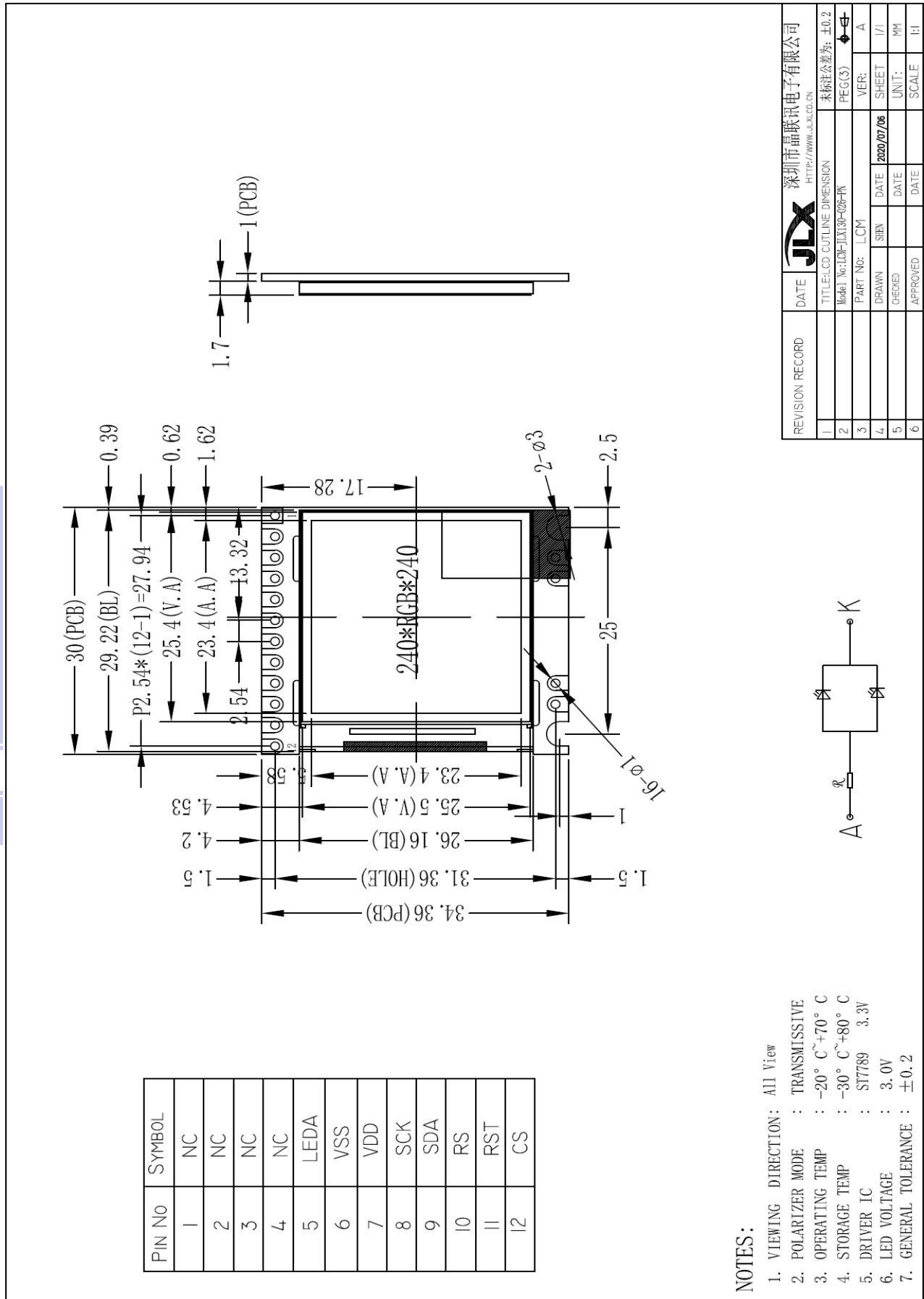


图 1. 外形尺寸

## 模块的接口引脚功能

表 1: 模块的接口引脚功能

引线号	符号	名称	功能
1	NC	空脚	空脚
2	NC	空脚	空脚
3	NC	空脚	空脚
4	NC	空脚	空脚
5	LDEA	背光电源正极	背光电源正极, (同 VDD 电压 3.3V)。
6	VSS	接地	0V
7	VDD	电路电源	3.3V
8	SCK	I/O	串行时钟
9	SDA	I/O	串行数据
10	A0 (RS)	寄存器选择信号	H: 数据寄存器 0: 指令寄存器 (IC 资料上所写为 "A0")
11	RST	复位	低电平复位, 复位完成后, 回到高电平, 液晶模块开始工作
12	CS	片选	低电平片选

## 4. 基本原理

### 4.1 液晶屏 (LCD)

在 LCD 上排列着 240×240 点阵, 240 个列信号与驱动 IC 相连, 240 个行信号也与驱动 IC 相连, IC 邦定在 LCD 玻璃上 (这种加工工艺叫 COG)。

### 4.3 背光参数

该型号液晶模块带 LED 背光源。它的性能参数如下:

工作温度: -20~+70° C;

存储温度: -30~+80° C;

背光板是白色。

正常工作电流为: 16~30mA (LED 灯数共 2 颗, 每颗灯是 8~15 mA)

工作电压: 同 VDD 电压 (LED 灯本身的电压是 3.0V, 但是因为在 PCB 上已加了限流电阻, 所以可以同 VDD 电压);

## 5. 技术参数

### 5.1 最大极限参数 (超过极限参数则会损坏液晶模块)

名称	符号	标准值			单位
		最小	典型	最大	
电路电源	VDD	-0.3	3.0	4.6	V
工作温度		-20		+70	°C
储存温度		-30		+80	°C

表 2: 最大极限参数

### 5.2 直流 (DC) 参数

名称	符号	测试条件	标准值			单位
			最小	典型值	最大	
工作电压	VDD		2.8	3.0	4.6	V
背光工作电压	VLED		2.9	3.0	3.1	V
背光工作电流	ILED	VLED=3.0V,	16	30	40	mA

		共 2 颗 LED 灯 并联			
--	--	----------------	--	--	--

表 3: 直流 (DC) 参数

## 6. 读写时序特性

详见 IC 资料 “ST7789V ”, 请找相关销售人员索要。

### 6.1 4\_SPI 时序

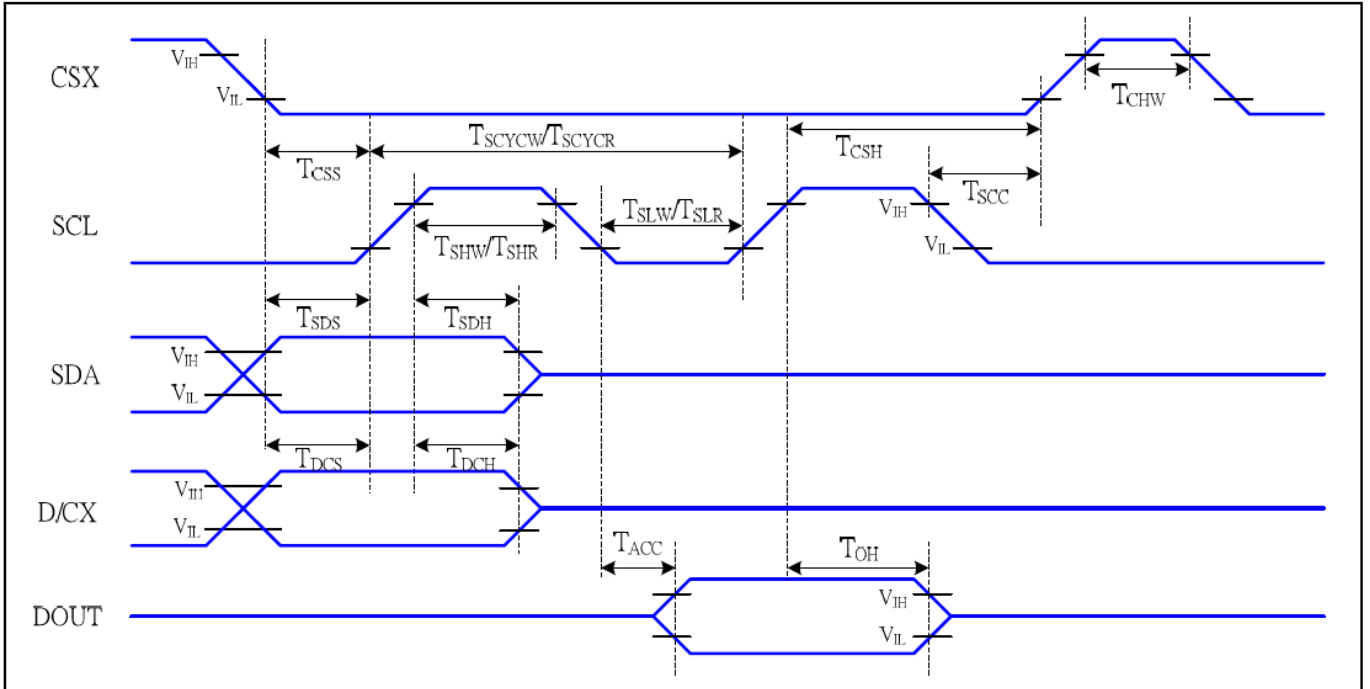
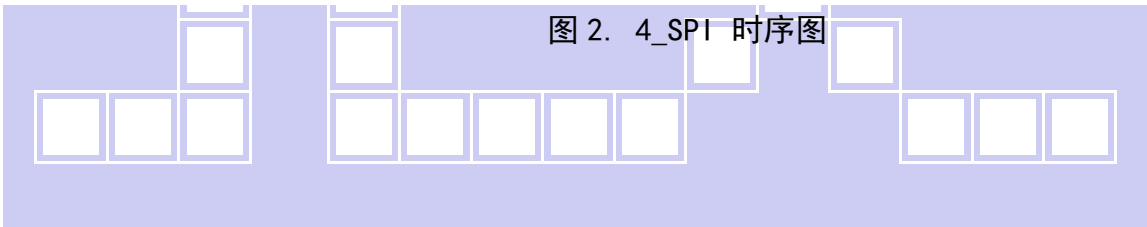


图 2. 4\_SPI 时序图

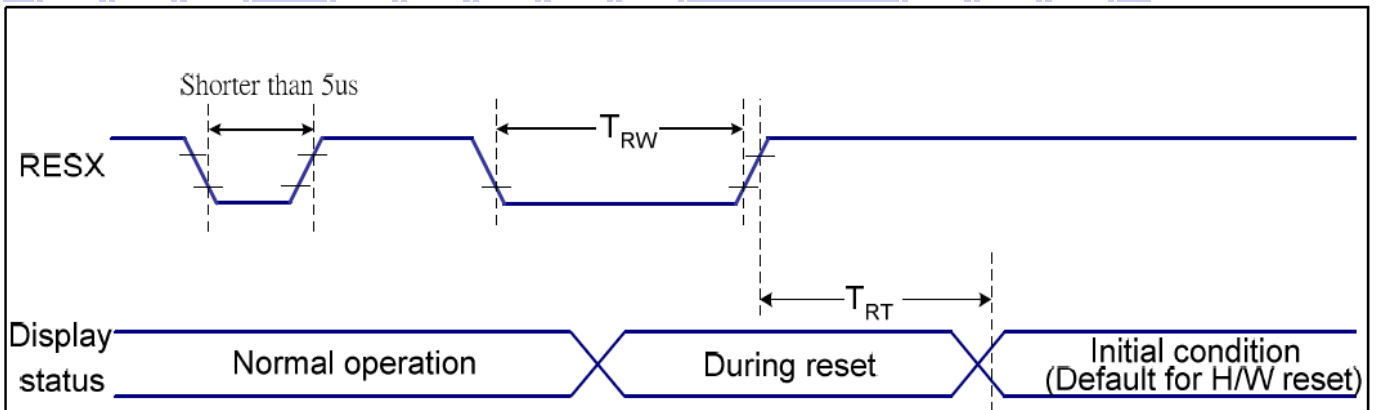


## 4\_SPI 时序要求 (AC 参数)

表 4

Signal	Symbol	Parameter	MIN	MAX	Unit	Description
CSX	$T_{CSS}$	Chip select setup time (write)	15		ns	
	$T_{CSH}$	Chip select hold time (write)	15		ns	
	$T_{CSS}$	Chip select setup time (read)	60		ns	
	$T_{SCC}$	Chip select hold time (read)	65		ns	
	$T_{CHW}$	Chip select "H" pulse width	40		ns	
SCL	$T_{SCYCW}$	Serial clock cycle (Write)	66		ns	-write command & data ram
	$T_{SHW}$	SCL "H" pulse width (Write)	15		ns	
	$T_{SLW}$	SCL "L" pulse width (Write)	15		ns	
	$T_{SCYCR}$	Serial clock cycle (Read)	150		ns	-read command & data ram
	$T_{SHR}$	SCL "H" pulse width (Read)	60		ns	
	$T_{SLR}$	SCL "L" pulse width (Read)	60		ns	
D/CX	$T_{DCS}$	D/CX setup time	10		ns	
	$T_{DCH}$	D/CX hold time	10		ns	
SDA (DIN)	$T_{SDS}$	Data setup time	10		ns	
	$T_{SDH}$	Data hold time	10		ns	
DOUT	$T_{ACC}$	Access time	10	50	ns	For maximum CL=30pF
	$T_{OH}$	Output disable time	15	50	ns	For minimum CL=8pF

## 6.2 电源启动后复位的时序要求 (RESET CONDITION AFTER POWER UP):


 图 3: 为电源启动后复位的时序  
 表 5: 电源启动后复位的时序要求

Related Pins	Symbol	Parameter	MIN	MAX	Unit
RESX	TRW	Reset pulse duration	10	-	us
	TRT	Reset cancel	-	5 (Note 1, 5)	ms
				120 (Note 1, 6, 7)	ms

## 7. 指令功能:

### 7.1 指令表

表6: 指令表

Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
NOP	0	↑	1	-	0	0	0	0	0	0	0	0	(00h)	No operation
SWRESET	0	↑	1	-	0	0	0	0	0	0	0	1	(01h)	Software reset
RDDID	0	↑	1	-	0	0	0	0	0	1	0	0	(04h)	Read display ID
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	ID17	ID16	ID15	ID14	ID13	ID12	ID11	ID10		ID1 read
	1	1	↑	-	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20		ID2 read
	1	1	↑	-	ID37	ID36	ID35	ID34	ID33	ID32	ID31	ID30		ID3 read
RDDST	0	↑	1	-	0	0	0	0	1	0	0	1	(09h)	Read display status
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	BSTON	MY	MX	MV	ML	RGB	MH	ST24		-
	1	1	↑	-	ST23	IFPF2	IFPF1	IFPF0	IDMON	PTLON	SLOUT	NORON		-
	1	1	↑	-	ST15	ST14	INVON	ST12	ST11	DISON	TEON	GCS2		-
	1	1	↑	-	GCS1	GCS0	TEM	ST4	ST3	ST2	ST1	ST0		-
RDDPM	0	↑	1	-	0	0	0	0	1	0	1	0	(0Ah)	Read display power
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	BSTON	IDMON	PTLON	SLPOUT	NORON	DISON	0	0		
RDD MADCTL	0	↑	1	-	0	0	0	0	1	0	1	1	(0Bh)	Read display
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	MY	MX	MV	ML	RGB	MH	0	0		-
RDD COLMOD	0	↑	1	-	0	0	0	0	1	1	0	0	(0Ch)	Read display pixel
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	0	D6	D5	D4	0	D2	D1	D0		-
RDDIM	0	↑	1	-	0	0	0	0	1	1	0	1	(0Dh)	Read display image
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	VSSON	0	INVON	0	0	GC2	GC1	GC0		-
RDDSM	0	↑	1	-	0	0	0	0	1	1	1	0	(0Eh)	Read display signal
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read



Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
	1	1	↑	-	TEON	TEM	0	0	0	0	0	0		-
RDDSDR	0	↑	1	-	0	0	0	0	1	1	1	1	(0Fh)	Read display self-diagnostic result
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	D7	D6	0	0	0	0	0	0		-
SLPIN	0	↑	1	-	0	0	0	1	0	0	0	0	(10h)	Sleep in
SLPOUT	0	↑	1	-	0	0	0	1	0	0	0	1	(11h)	Sleep out
PTLON	0	↑	1	-	0	0	0	1	0	0	1	0	(12h)	Partial mode on
NORON	0	↑	1	-	0	0	0	1	0	0	1	1	(13h)	Partial off (Normal)
INVOFF	0	↑	1	-	0	0	1	0	0	0	0	0	(20h)	Display inversion off
INVON	0	↑	1	-	0	0	1	0	0	0	0	1	(21h)	Display inversion on
GAMSET	0	↑	1	-	0	0	1	0	0	0	0	1	(26h)	Display inversion on
	1	↑	1	-	0	0	0	0	GC3	GC2	GC1	GC0		
DISPOFF	0	↑	1	-	0	0	1	0	1	0	0	0	(28h)	Display off
DISPON	0	↑	1	-	0	0	1	0	1	0	0	1	(29h)	Display on
CASET	0	↑	1	-	0	0	1	0	1	0	1	0	(2Ah)	Column address set
	1	↑	1	-	XS15	XS14	XS13	XS12	XS11	XS10	XS9	XS8		X address start:
	1	↑	1		XS7	XS6	XS5	XS4	XS3	XS2	XS1	XS0		$0 \leq XS \leq X$
	1	↑	1		XE15	XE14	XE13	XE12	XE11	XE10	XE9	XE8		X address start:
1	↑	1		XE7	XE6	XE5	XE4	XE3	XE2	XE1	XE0		$S \leq XE \leq X$	
RASET	0	↑	1	-	0	0	1	0	1	0	1	1	(2Bh)	Row address set
	1	↑	1	-	YS15	YS14	YS13	YS12	YS11	YS10	YS9	YS8		Y address start:
	1	↑	1		YS7	YS6	YS5	YS4	YS3	YS2	YS1	YS0		$0 \leq YS \leq Y$
	1	↑	1		YE15	YE14	YE13	YE12	YE11	YE10	YE9	YE8		Y address start:
1	↑	1		YE7	YE6	YE5	YE4	YE3	YE2	YE1	YE0		$S \leq YE \leq Y$	
RAMWR	0	↑	1	-	0	0	1	0	1	1	0	0	(2Ch)	Memory write
	1	↑	1	D1[17:8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]		Write data
	1	↑	1	Dx[17:8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]		
	1	↑	1	Dn[17:8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]		
RAMRD	0	↑	1	-	0	0	1	0	1	1	1	0	(2Eh)	Memory read





Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	D1[17:8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]		Read data
	1	1	↑	Dx[17:8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]		
	1	1	↑	Dn[17:8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]		
PTLAR	0	↑	1	-	0	0	1	1	0	0	0	0	(30h)	Partial start/end address set
	1	↑	1	-	PSL15	PSL14	PSL13	PSL12	PSL11	PSL10	PSL9	PSL8		Partial start address: (0, 1, 2, ...P)
	1	↑	1	-	PSL7	PSL6	PSL5	PSL4	PSL3	PSL2	PSL1	PSL0		
	1	↑	1	-	PEL15	PEL14	PEL13	PEL12	PEL11	PEL10	PEL9	PEL8		Partial end address (0, 1, 2, 3, ...P)
VSCRDEF	0	↑	1	-	0	0	1	1	0	0	1	1	(33h)	Vertical scrolling definition
	1	↑	1	-	TFA15	TFA14	TFA13	TFA12	TFA11	TFA10	TFA9	TFA8		
	1	↑	1	-	TFA7	TFA6	TFA5	TFA4	TFA3	TFA2	TFA1	TFA0		
	1	↑	1	-	VSA15	VSA14	VSA13	VSA12	VSA11	VSA10	VSA9	VSA8		
	1	↑	1	-	VSA7	VSA6	VSA5	VSA4	VSA3	VSA2	VSA1	VSA0		
	1	↑	1	-	BFA15	BFA14	BFA13	BFA12	BFA11	BFA10	BFA9	BFA8		
	1	↑	1	-	BFA7	BFA6	BFA5	BFA4	BFA3	BFA2	BFA1	BFA0		
TEOFF	0	↑	1	-	0	0	1	1	0	1	0	0	(34h)	Tearing effect line off
TEON	0	↑	1	-	0	0	1	1	0	1	0	1	(35h)	Tearing effect line on
	1	↑	1	-	-	-	-	-	-	-	-	TEM		
MADCTL	0	↑	1	-	0	0	1	1	0	1	1	0	(36h)	Memory data access control
	1	↑	1	-	MY	MX	MV	ML	RGB	0	0	0		-
VSCRSAADD	0	↑	1	-	0	0	1	1	0	1	1	1	(37h)	Vertical scrolling start address
	1	↑	1	-	VSP15	VSP14	VSP13	VSP12	VSP11	VSP10	VSP9	VSP8		
	1	↑	1	-	VSP7	VSP6	VSP5	VSP4	VSP3	VSP2	VSP1	VSP0		
IDMOFF	0	↑	1	-	0	0	1	1	1	0	0	0	(38h)	Idle mode off
IDMON	0	↑	1	-	0	0	1	1	1	0	0	1	(39h)	Idle mode on



Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
COLMOD	0	↑	1	-	0	0	1	1	1	0	1	0	(3Ah)	Interface pixel format
	1	↑	1	-	0	D6	D5	D4	0	D2	D1	D0		Interface format
RAMWRC	0	↑	1	-	0	0	1	1	1	1	0	0	(3Ch)	Memory write continue
	1	↑	1	D1[17:8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]		Write data
	1	↑	1	Dx[17:8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]		
	1	↑	1	Dn[17:8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]		
RAMRDC	0	↑	1	-	0	0	1	1	1	1	1	0	(3Eh)	Memory read continue
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy Read
	1	1	↑	D1[17:8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]		
	1	1	↑	Dx[17:8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]		
	1	1	↑	Dn[17:8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]		
TESCAN	0	↑	1	-	0	1	0	0	0	1	0	0	(44h)	Set tear scanline
	1	↑	1	-	N15	N14	N13	N12	N11	N10	N9	N8		
	1	↑	1	-	N7	N6	N5	N4	N3	N2	N1	N0		
RDTESCAN	0	↑	1	-	0	1	0	0	0	1	0	1	(45h)	Get scanline
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy Read
	1	1	↑	-	-	-	-	-	-	-	N9	N8		
	1	1	↑	-	N7	N6	N5	N4	N3	N2	N1	N0		
WRDISBV	0	↑	1	-	0	1	0	1	0	0	0	1	(51h)	Write display brightness
	1	↑	1	-	DBV7	DBV6	DBV5	DBV4	DBV3	DBV2	DBV1	DBV0		
RDDISBV	0	↑	1	-	0	1	0	1	0	0	1	0	(52h)	Read display brightness value
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	DBV7	DBV6	DBV5	DBV4	DBV3	DBV2	DBV1	DBV0		
WRCTRLD	0	↑	1	-	0	1	0	1	0	0	1	1	(53h)	Write CTRL display
	1	↑	1	-	0	0	BCTRL	0	DD	BL	0	0		
RDCTRLD	0	↑	1	-	0	1	0	1	0	1	0	0	(54h)	Read CTRL value display
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	0	0	BCTRL	0	DD	BL	0	0		



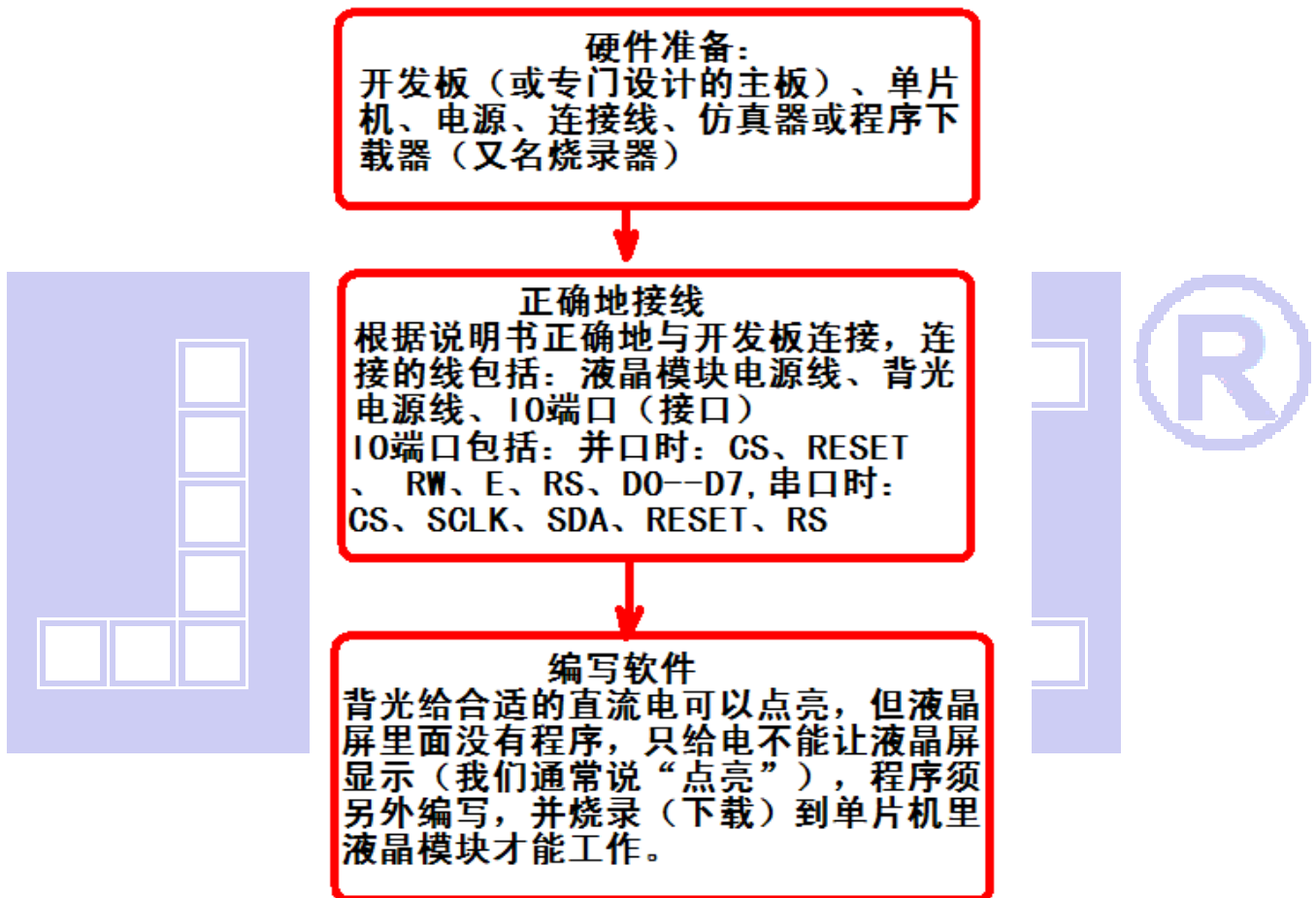
Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
WRCACE	0	↑	1	-	0	1	0	1	0	1	0	1	(55h)	Write content adaptive brightness control and Color enhancemnet
	1	↑	1	-	CECTRL	0	CE1	CE0	0	0	C1	C0		
RDCABC	0	↑	1	-	0	1	0	1	0	1	1	0	(56h)	Read content adaptive brightness control
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	0	CECTRL	0	0	0	0	C1	C0		
WRCABCMB	0	↑	1	-	0	1	0	1	1	1	1	0	(5Eh)	Write CABC minimum brightness
	1	↑	1	-	CMB7	CMB6	CMB5	CMB4	CMB3	CMB2	CMB1	CMB0		
RDCABCMB	0	↑	1	-	0	1	0	1	1	1	1	1	(5Fh)	Read CABC minimum brightness
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	CMB7	CMB6	CMB5	CMB4	CMB3	CMB2	CMB1	CMB0		
RDABCSDR	0	↑	1	-	0	1	1	0	1	0	0	0	(68h)	Read Automatic Brightness Control Self-Diagnostic Result
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	D7	D6	0	0	0	0	0	0		-
RDID1	0	↑	1	-	1	1	0	1	1	0	1	0	(DAh)	Read ID1
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	ID17	ID16	ID15	ID14	ID13	ID12	ID11	ID10		Read parameter
RDID2	0	↑	1	-	1	1	0	1	1	0	1	1	(DBh)	Read ID2
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑	-	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20		Read parameter
RDID3	0	↑	1	-	1	1	0	1	1	1	0	0	(DCh)	Read ID3

Instruction	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	Hex	Function
	1	1	↑	-	-	-	-	-	-	-	-	-		Dummy read
	1	1	↑		ID37	ID36	ID35	ID34	ID33	ID32	ID31	ID30		Read parameter

## 7.2 初始化方法

用户所编的显示程序, 开始必须进行初始化, 否则模块无法正常显示, 过程请参考程序

### 点亮液晶模块的步骤



## 7.3 串行接口程序

液晶模块与 MPU(以 8051 系列单片机为例) 接口图如下:

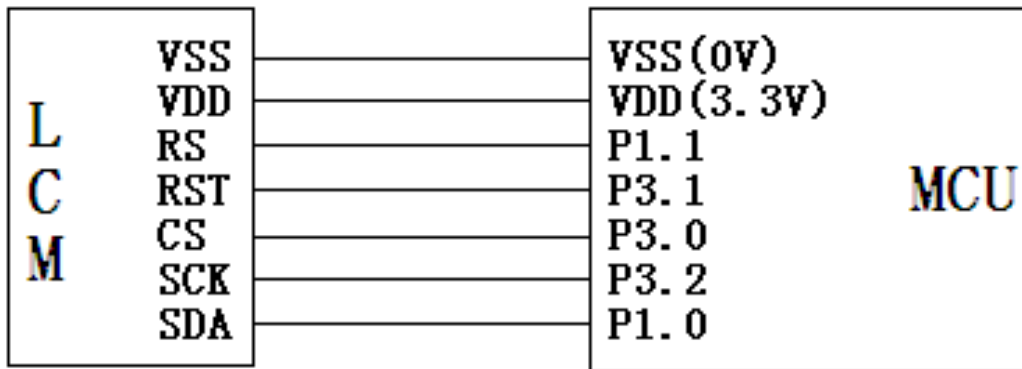


图 4. 串行接口

### 串行接口程序

```
//driver IC:ST7789V,
```

```
//#include <reg51.h>
#include <STC15F2K60S2.H>
#include <chinese_code.h>

//液晶屏 IC 所需要的信号线的接口定义
sbit CS=P3^0;
sbit RST=P3^1;
sbit RS=P1^1;
sbit SCK=P3^2;
sbit SDA=P1^0;
sbit key=P2^0; //P2.0 口与 GND 之间接一个按键
```

```
/*写指令到 LCD 模块*/
```

```
void transfer_command(int data1)
{
    char i;
    CS=0;
    RS=0;
    for(i=0;i<8;i++)
    {
        SCK=0;
        if(data1&0x80) SDA=1;
        else SDA=0;
        SCK=1;
        data1<<=1;
    }
}
```

```
/*写数据到 LCD 模块*/
```

```
void transfer_data(uchar data1)
```

```
{
    char i;
    CS=0;
    RS=1;
    for(i=0;i<8;i++)
    {
        SCK=0;
        if(data1&0x80) SDA=1;
        else SDA=0;
        SCK=1;
        data1<<=1;
    }
}
```

```
//===传 16 位指令，16 位指令一起赋值
```

```
void transfer_command_16(uint com_16bit)
```

```
{
    transfer_command(com_16bit >>8); //先传高 8 位
    transfer_command(com_16bit ); //再传低 8 位
}
```

```
//连写 2 个字节（即 16 位）数据到 LCD 模块
```

```
void transfer_data_16(uint data_16bit)
```

```
{
    transfer_data(data_16bit>>8);
    transfer_data(data_16bit);
}
```

```
//===发送 1 个字节的指令及 1 个字节的数据=====
```

```
void Lcd_Write_Com_Data(uint com,uint val)
```

```
{
    transfer_command_16(com); //先传指令
    transfer_data_16(val); //再传数据
}
```

```
void delay(long i)
```

```
{
    int j,k;
    for(j=0;j<i;j++)
        for(k=0;k<110;k++);
}
```

```
void delay_us(long i)
```

```
{
```

```

int j,k;
for(j=0;j<i;j++)
for(k=0;k<10;k++)
}

void Switch()
{
repeat:
if (key==1) goto repeat;
else delay(1000);
if (key) goto repeat;
else ;
}

void lcd_initial()
{
RST=0;
delay(200);
RST=1;
delay(200);
//***** Start Initial Sequence *****//
transfer_command(0x11);
delay(200);
//-----display and color format setting-----//
transfer_command(0x36); //行扫描顺序及 RGB, 列扫描顺序, 横放/竖放
transfer_data(0x00); //0x00 竖显示, 0xa0 横显示
transfer_command(0xB6); //显示功能设置: 列/行 显示顺序
transfer_data(0x0A);
transfer_data(0xa2); //改变 SOURCE 线的方向: 0xa2: 左到右, 0x82: 右到左

transfer_command(0x3a); //256K 16bit/pixel
transfer_data(0x05);
//-----ST7789V Frame rate setting-----//
transfer_command(0xb2);
transfer_data(0x0c);
transfer_data(0x0c);
transfer_data(0x00);
transfer_data(0x33);
transfer_data(0x33);

transfer_command(0xb7);
transfer_data(0x35);
//-----ST7789V Power setting-----//
transfer_command(0xbb);
transfer_data(0x19);

```



```
transfer_command(0xc0);
transfer_data(0x2c);
```

```
transfer_command(0xc2);
transfer_data(0x01);
```

```
transfer_command(0xc3);
transfer_data(0x12);
```

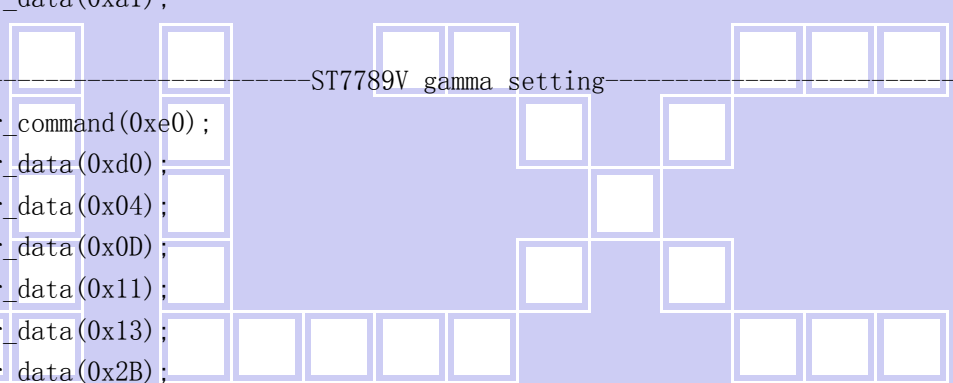
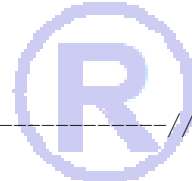
```
transfer_command(0xc4);
transfer_data(0x20);
```

```
transfer_command(0xc6);
transfer_data(0x0f);
```

```
transfer_command(0xd0);
transfer_data(0xa4);
transfer_data(0xa1);
```

//-----ST7789V gamma setting-----//

```
transfer_command(0xe0);
transfer_data(0xd0);
transfer_data(0x04);
transfer_data(0x0D);
transfer_data(0x11);
transfer_data(0x13);
transfer_data(0x2B);
transfer_data(0x3F);
transfer_data(0x54);
transfer_data(0x4C);
transfer_data(0x18);
transfer_data(0x0D);
transfer_data(0x0B);
transfer_data(0x1F);
transfer_data(0x23);
```

```
transfer_data(0x3F);
transfer_data(0x54);
transfer_data(0x4C);
transfer_data(0x18);
transfer_data(0x0D);
transfer_data(0x0B);
transfer_data(0x1F);
transfer_data(0x23);
```

```
transfer_command(0xe1);
transfer_data(0xd0);
transfer_data(0x04);
transfer_data(0x0C);
transfer_data(0x11);
transfer_data(0x13);
transfer_data(0x2c);
transfer_data(0x3f);
transfer_data(0x44);
```



```

transfer_data(0x51);
transfer_data(0x2f);
transfer_data(0x1f);
transfer_data(0x1f);
transfer_data(0x20);
transfer_data(0x23);

transfer_command(0x21);

transfer_command(0x29);    //打开显示
}
    
```

//定义窗口坐标: 开始坐标 (XS,YS)以及窗口大小 (x\_total,y\_total)

```
void lcd_address(int XS,int YS,int x_total,int y_total)
```

```

{
    int XE,YE;
    XE=XS+x_total-1;
    YE=YS+y_total-1;
    transfer_command(0x2a);    // 设置 X 开始及结束的地址
    transfer_data_16(XS);    // X 开始地址(16 位)
    transfer_data_16(XE);    // X 结束地址(16 位)

    transfer_command(0x2b);    // 设置 Y 开始及结束的地址
    transfer_data_16(YS);    // Y 开始地址(16 位)
    transfer_data_16(YE);    // Y 结束地址(16 位)

    transfer_command(0x2c);    // 写数据开始
}
    
```

```
void mono_transfer_data_16(int mono_data,int font_color,int back_color)
```

```

{
    int i;
    for(i=0;i<8;i++)
    {
        if(mono_data&0x80)
        {
            transfer_data_16(font_color);    //当数据是 1 时, 显示字体颜色
        }
        else
        {
            transfer_data_16(back_color);    //当数据是 0 时, 显示底色
        }
        mono_data<<=1;
    }
}
    
```



//全屏显示一种颜色, 竖向显示

```
void display_color_h(int color_data)
{
    int i, j;
    transfer_command(0x36);    //行扫描顺序及 RGB, 列扫描顺序, 横放/竖放
    transfer_data(0x00);
    lcd_address(0, 0, 240, 320);
    for(i=0; i<240; i++)
    {
        for(j=0; j<320; j++)
        {
            transfer_data_16(color_data);
        }
    }
}
```

//全屏显示一种颜色, 横向显示

```
void display_color_v(int color_data)
{
    int i, j;
    transfer_command(0x36);    //行扫描顺序及 RGB, 列扫描顺序, 横放/竖放
    transfer_data(0x00);
    lcd_address(0, 0, 240, 240);
    for(i=0; i<240; i++)
    {
        for(j=0; j<240; j++)
        {
            transfer_data_16(color_data);
        }
    }
}
```



//指定地址显示颜色

```
void LCD_Fill(ulong sx, ulong sy, ulong ex, ulong ey, int color_data)
{
    ulong i, j;
    ulong width=ex-sx+1;
    ulong height=ey-sy+1;
    transfer_command(0x36);    //行扫描顺序及 RGB, 列扫描顺序, 横放/竖放
    transfer_data(0x00);
    lcd_address(sx, sy, ex, ey);
    for(i=0; i<height; i++)
    {
        for(j=0; j<width; j++)
        {
```

```

        transfer_data_16(color_data);
    }
}
lcd_address(0, 0, 240, 240);
}

```

```

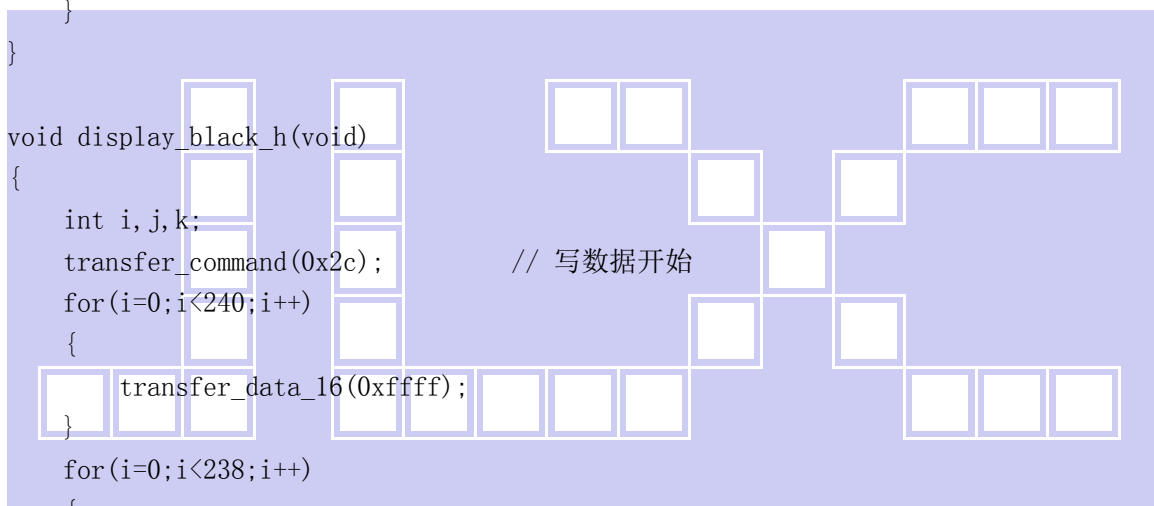
void display_white(void)
{
    int i, j;
    transfer_command(0x2c);
    for(i=0; i<240; i++)
    {
        for(j=0; j<240; j++)
        {
            transfer_data_16(0xffff);
        }
    }
}

```

```

void display_black_h(void)
{
    int i, j, k;
    transfer_command(0x2c); // 写数据开始
    for(i=0; i<240; i++)
    {
        transfer_data_16(0xffff);
    }
    for(i=0; i<238; i++)
    {
        for(k=0; k<1; k++)
        {
            transfer_data_16(0xffff);
        }
        for(j=0; j<238; j++)
        {
            transfer_data_16(0x0000);
        }
        for(k=0; k<1; k++)
        {
            transfer_data_16(0xffff);
        }
    }
    for(i=0; i<240; i++)
    {
        transfer_data_16(0xffff);
    }
}

```



```

    }
}

void display_black_v(void)
{
    int i, j, k;
    transfer_command(0x2c);          // 写数据开始
    for(i=0;i<240;i++)
    {
        transfer_data_16(0xffff);
    }
    for(i=0;i<238;i++)
    {
        for(k=0;k<1;k++)
        {
            transfer_data_16(0xffff);
        }
        for(j=0;j<238;j++)
        {
            transfer_data_16(0x0000);
        }
        for(k=0;k<1;k++)
        {
            transfer_data_16(0xffff);
        }
    }
    for(i=0;i<240;i++)
    {
        transfer_data_16(0xffff);
    }
}

```

```

//显示 8x16 点阵的字符串
void disp_string_8x16(int x, int y, char *text, int font_color, int back_color)
{
    int i=0, j, k;
    while(text[i]>0x00)
    {
        if((text[i]>=0x20)&&(text[i]<=0x7e))
        {
            j=text[i]-0x20;
            lcd_address(x, y, 8, 16);
            for(k=0;k<16;k++)
            {

```

```

        mono_transfer_data_16(ascii_table_8x16[j*16+k], font_color, back_color);
//?a??"ascii_table_8x16[]"?a??êy×é?ú"ASCII_TABLE_5X8_8X16_horizontal.h"à?
    }
    x+=8;
    i++;
}
else
i++;
}
}

```

```

void display_string_16x16(int x, int y, uchar *text, int font_color, int back_color)
{

```

```

    uchar i, j, k;

```

```

    uint address;

```

```

    j = 0;

```

```

    while(text[j] != '\0') // '\0' 字符串结束标志

```

```

    {

```

```

        i = 0;

```

```

        address = 1;

```

```

        while(Chinese_horizontal_text_16x16[i] > 0x7e) // >0x7f 即说明不是 ASCII 码字符

```

```

        {

```

```

            if(Chinese_horizontal_text_16x16[i] == text[j])

```

```

            {

```

```

                if(Chinese_horizontal_text_16x16[i + 1] == text[j + 1])

```

```

                {

```

```

                    address = i * 16;

```

```

                    break;

```

```

                }

```

```

            }

```

```

            i += 2;

```

```

        }

```

```

        if(y > 240)

```

```

        {

```

```

            y=0;

```

```

            x+=16;

```

```

        }

```

```

        if(address != 1) // 显示汉字

```

```

        {

```

```

            lcd_address(x, y, 16, 16);

```

```

            for(i=0; i<2; i++)

```

```

            {

```

```

                for(k = 0; k < 16; k++)

```

```

                {

```

```
mono_transfer_data_16(Chinese_horizontal_code_16x16[address], font_color, back_color);
    address++;
}
```

```
    }
    j+=2;
}
else //显示空白字符
{
```

```
    lcd_address(x, y, 16, 16);
    for(i = 0; i <2; i++)
    {
        for(k = 0; k < 16; k++)
        {
            mono_transfer_data_16(0x00, font_color, back_color);
        }
    }
    j+=2;
}
```

```
    }
    x=x+16;
}
```

//显示 16x32 点阵的单色的图像

```
void disp_16x32(int x, int y, char *dp, int font_color, int back_color)
```

```
{
    int i, j;
    lcd_address(x, y, 16, 32);
    for(i=0; i<16; i++)
    {
        for(j=0; j<4; j++)
        {
            mono_transfer_data_16(*dp, font_color, back_color);
            dp++;
        }
    }
}
```

//显示 32x32 点阵的单色的图像

```
void disp_32x32(int x, int y, char *dp, int font_color, int back_color)
```

```
{
    int i, j;
    lcd_address(x, y, 32, 32);
    for(i=0; i<32; i++)
    {
        for(j=0; j<4; j++)
        {
```



```

        mono_transfer_data_16(*dp, font_color, back_color);
        dp++;
    }
}
}

```

//显示 32x64 点阵的单色的图像

```

void disp_32x64(int x, int y, char *dp, int font_color, int back_color)
{
    int i, j;
    lcd_address(x, y, 32, 64);
    for(i=0; i<32; i++)
    {
        for(j=0; j<8; j++)
        {
            mono_transfer_data_16(*dp, font_color, back_color);
            dp++;
        }
    }
}

```

//显示一幅彩图

```

void display_image(int x, int y, uchar *dp)
{
    uchar i, j, k=0;
    transfer_command(0x36); //行扫描顺序及 RGB, 列扫描顺序, 横放/竖放
    transfer_data(0x00);
    lcd_address(x, y, 120, 120);
    for(i=0; i<120; i++)
    {
        for(j=0; j<120; j++)
        {
            transfer_data(*dp); //传一个像素的图片数据的高位
            dp++;
            transfer_data(*dp); //传一个像素的图片数据的低位
            dp++;
        }
    }
}

```

```

void main(void)
{

```

```

P1M1=0x00;
P1M0=0x00;
P2M1=0x00;
P2M0=0x00;
P3M1=0x00;
P3M0=0x00;
// CS=0;
lcd_initial();
while(1)
{
    display_color_v(blue);
    LCD_Fill(0, 0, 240, 56, red);
    disp_32x32(40+32*0, 8, jing_32x32, yellow, red);
    disp_32x32(40+32*1, 8, lian_32x32, yellow, red);
    disp_32x32(40+32*2, 8, xun_32x32, yellow, red);
    disp_32x32(40+32*3, 8, dian_32x32, yellow, red);
    disp_32x32(40+32*4, 8, zi_32x32, yellow, red);

```

```

LCD_Fill(0, 57, 240, 104, green);
disp_16x32(40+16*0, 64, J_16x32, purple, green);
disp_16x32(40+16*1, 64, X_16x32, purple, green);
disp_16x32(40+16*2, 64, X_16x32, purple, green);
disp_16x32(40+16*3, 64, yi_16x32, purple, green);
disp_16x32(40+16*4, 64, san_16x32, purple, green);
disp_16x32(40+16*5, 64, ling_16x32, purple, green);
disp_16x32(40+16*6, 64, gang_16x32, purple, green);
disp_16x32(40+16*7, 64, ling_16x32, purple, green);
disp_16x32(40+16*8, 64, er_16x32, purple, green);
disp_16x32(40+16*9, 64, liu_16x32, purple, green);

```

```

LCD_Fill(0, 105, 240, 152, blue);
disp_32x32(24+32*0, 112, den_32x32, white, blue);
disp_32x32(24+32*1, 112, jen_32x32, white, blue);
disp_16x32(88+16*0, 112, mon_16x32, white, blue);
disp_16x32(88+16*1, 112, er_16x32, white, blue);
disp_16x32(88+16*2, 112, si_16x32, white, blue);
disp_16x32(88+16*3, 112, ling_16x32, white, blue);
disp_16x32(88+16*4, 112, X_16x32, white, blue);
disp_16x32(88+16*5, 112, er_16x32, white, blue);
disp_16x32(88+16*6, 112, si_16x32, white, blue);
disp_16x32(88+16*7, 112, ling_16x32, white, blue);

```

```

LCD_Fill(0, 153, 240, 200, yellow);
disp_32x32(32+32*0, 160, shi2_32x32, red, yellow);
disp_32x32(32+32*1, 160, jiao_32x32, red, yellow);
disp_16x32(96+16*0, 160, mon_16x32, red, yellow);

```





```

disp_32x32(112+32*0, 160, quan_32x32, red, yellow);
disp_32x32(112+32*1, 160, shi2_32x32, red, yellow);
disp_32x32(112+32*2, 160, jiao_32x32, red, yellow);

LCD_Fill(0, 201, 240, 240, blue);
display_string_16x16(64, 213, "驱动", white, blue);
disp_string_8x16(96, 213, "IC:ST7789V", white, blue);
Switch();

// display_string_16x16(24, 56, "深圳市晶联讯电子有限公司", white, blue);
//
// disp_string_8x16(72, 88, "JLX130-026", white, blue);
// Switch();
display_color_v(light_blue);
LCD_Fill(2, 2, 236, 238, red);
disp_32x32(16+32*0, 26, wen_32x32, white, red);
disp_32x32(16+32*1, 26, du_32x32, white, red);
disp_32x32(32+32*5, 26, c_32x32, white, red);
disp_32x64(88+32*0, 66, liu_32x64, white, red);
disp_32x64(88+32*1, 66, ling_32x64, white, red);
disp_16x32(152+16*0, 90, den_16x32, white, red);
disp_32x64(88+16+32*2, 66, er_32x64, white, red);

disp_32x32(10+32*0, 138, she_32x32, white, red);
disp_32x32(10+32*1, 138, ding_32x32, white, red);
disp_32x32(10+32*2, 138, zhi_32x32, white, red);
disp_16x32(145+16*0, 138, liu_16x32, yellow, red);
disp_16x32(145+16*1, 138, ling_16x32, yellow, red);
// LCD_Fill(108, 138, 126, 260, light_black);

disp_32x32(10+32*0, 178, gong2_32x32, white, red);
disp_32x32(10+32*1, 178, ling_32x32, white, red);
disp_32x32(10+32*2, 178, zhi_32x32, white, red);
disp_16x32(145+16*0, 178, ji_16x32, yellow, red);
disp_16x32(145+16*1, 178, ji_16x32, yellow, red);
disp_16x32(145+16*2, 178, den_16x32, yellow, red);
disp_16x32(145+16*3, 178, ji_16x32, yellow, red);
disp_16x32(145+16*4, 178, ban_16x32, yellow, red);
Switch();

display_image(0, 0, pic1);
display_image(120, 0, pic1);
display_image(0, 120, pic1);
display_image(120, 120, pic1);
Switch();
    
```



```

display_color_v(0xf800);
Switch();
display_color_v(0x07e0);
Switch();
display_color_v(0x001f);
Switch();
display_black_v();
Switch();
display_color_v(0xffff);
Switch();
}
}

```

**-END-**

