JLX384160G-973-PL 使用说明书

(带中文字库 IC)

目 录

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 JLX384160G-973 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。

JLX384160G-973 可以显示 384 列*160 行点阵单色或 4 灰度级的图片，或显示 12 个/行*5 行 32*32 点阵或显示 16 个/行*6 行 24*24 点阵的汉字，或显示 24 个/行*10 行 16*16 点阵的汉字，或显示 8*16 点阵的英文、数字、字符 48 个*10 行，或显示 5*8 点阵的英文、数字、字符 64 个*20 行。又含有 JLX-GB2312-3205 字库 IC，可以从字库 IC 中读出内置的字库的点阵数据写入到 LCD 驱动 IC 中，以达到显示汉字的目的。

2．JLX384160G-973 图像型点阵液晶模块的特性

2.1 结构牢。

2.2 IC 采用矽创公司 ST7586S,功能强大，稳定性好

2.3 功耗低。

2.4 接口简单方便:可采用 4 线 SPI 串行接口，或选择并行接口。

2.5 工作温度宽:-20℃ － 70℃;

2.6 显示内容:

- 可 384*160 点阵单色或 4 灰度级图片;
- 可显示 12 个×5 行 32*32 点阵的汉字;
- 可显示 16 个×6 行 24*24 点阵的汉字;
- 可显示 24 个×10 行 16*16 点阵的汉字;
- 可显示 32 个×13 行 12*12 点阵的汉字;
- 可显示 48 个*10 行 8*16 点阵的英文、数字、字符;
- 可显示 64 个*20 行 5*8 点阵的英文、数字、字符;
- 可显示其他的 ASCII 码等;

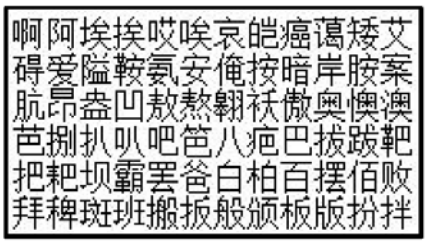
2.7 字库 IC(IC 型号：JLX-GB2312-3205，此 IC 为可选的配件)自带字库内容：

分类	字库内容	编码体系（字符集）	字符数
汉字字符	11X12 点 GB2312 标准点阵字库	GB2312	6763+846
	15X16 点 GB2312 标准点阵字库	GB2312	6763+846
	24X24 点 GB2312 标准点阵字库	GB2312	6763+846
	32X32 点 GB2312 标准点阵字库	GB2312	6763+846
	6X12 点国标扩展字符	GB2312	126
	8X16 点国标扩展字符	GB2312	126
	12X24 点国标扩展字符	GB2312	126
	16X32 点国标扩展字符	GB2312	126
ASCII 字符	5X7 点 ASCII 字符	ASCII	96
	7X8 点 ASCII 字符	ASCII	96
	6X12 点 ASCII 字符	ASCII	96
	8X16 点 ASCII 字符	ASCII	96
	8X16 点粗体 ASCII 字符	ASCII	96
	12 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96
	12 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96
	16 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96
	16 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96
	24 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96
	24 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96
	32 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96
	32 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96
输入法码表	全拼输入法码表	GB2312	

字型样张

11X12 点 GB2312 汉字

15X16 点 GB2312 汉字



24X24 点 GB2312 汉字

32X32 点 GB2312 汉字

5x7 点 ASCII 字符

7x8 点 ASCII 字符

6x12 点 ASCII 字符

8x16 点 ASCII 字符

12 点阵不等宽 ASCII 方头

16 点阵不等宽 ASCII 方头

3. 外形尺寸及接口引脚功能：

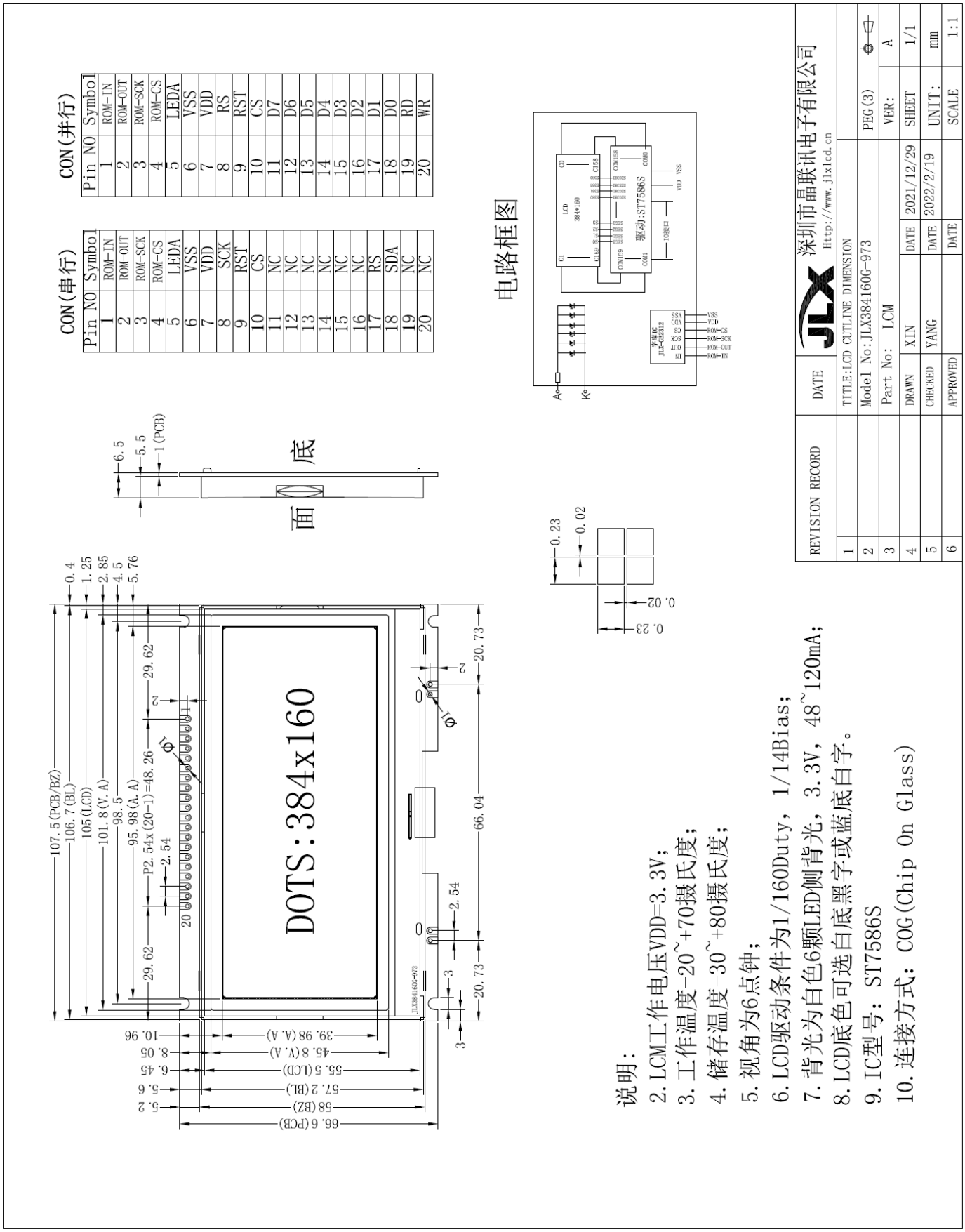


图 1. 液晶模块外形尺寸

模块的接口既可以当成并口用，也可以当成串口用（PCB 内部跳线选择串口/并口）：

◆当并行时，CON1 功能如下：

引 线 号	符 号	名 称	功 能
1	ROM_IN	字库 IC 接口 SI	字库 IC:串行数据输入。不带字库时：空脚
2	ROM_OUT	字库 IC 接口 SO	字库 IC:串行数据输出。不带字库时：空脚
3	ROM_SCK	字库 IC 接口 SCLK	字库 IC:串行时钟。不带字库时：空脚
4	ROM_CS	字库 IC 接口 CS#	字库 IC:片选输入。不带字库时：空脚
5	LEDA	背光电源	背光电源正极，同 VDD 电压（5V 或 3.3V）
6	VSS	接地	0V
7	VDD	电路电源	5V 或 3.3V
8	A0	寄存器选择信号	H:数据寄存器 0:指令寄存器
9	RESET	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
10	CS	片选	低电平片选
11	D7	I/O	数据总线 DB7-DB0
12	D6		
13	D5		
14	D4		
15	D3		
16	D2		
17	D1		
18	D0		
19	E(RD)	使能信号(读)	6800 时序时：E:使能信号 8080 时序时：读信号
20	R/W(WR)	读/写（写）	6800 时序时：RW：H:读信号 L:写信号 8080 时序时：写信号

表 1：并行接口功能

◆当 4 线 SPI 串行时，接口功能如下：

引 线 号	符 号	名 称	功 能
1	ROM_IN	字库 IC 接口 SI	字库 IC:串行数据输入。不带字库时：空脚
2	ROM_OUT	字库 IC 接口 SO	字库 IC:串行数据输出。不带字库时：空脚
3	ROM_SCK	字库 IC 接口 SCLK	字库 IC:串行时钟。不带字库时：空脚
4	ROM_CS	字库 IC 接口 CS#	字库 IC:片选输入。不带字库时：空脚
5	LEDA	背光电源	背光电源正极，同 VDD 电压（5V 或 3.3V）
6	VSS	接地	0V
7	VDD	电路电源	5V 或 3.3V
8	SCL(A0)	串行时钟	串行时钟
9	RESET	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
10	CS	片选信号	低电平片选
11	NC(D7)	空脚	
12	NC(D6)	空脚	
13	NC(D5)	空脚	
14	NC(D4)	空脚	

15	NC(D3)	空脚	
16	NC(D2)	空脚	
17	A0(D1)	寄存器选择	H:数据寄存器 0:指令寄存器
18	SDA(D0)	串行数据	串行数据
19	NC	空脚	
20	NC	空脚	

表 2：串行接口功能

4. 电路框图

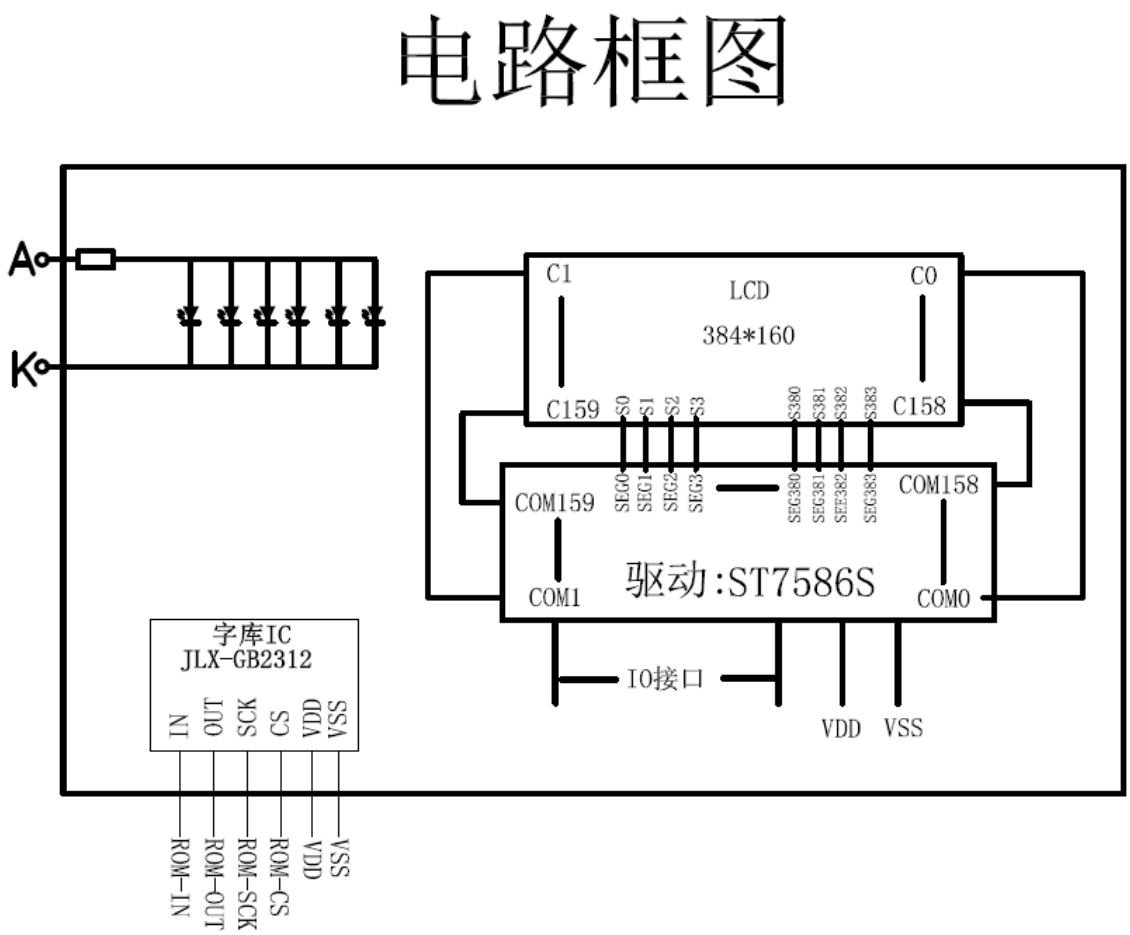


图 2:JLX384160G-973 图像点阵型液晶模块的电路框图

5. 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;
背光颜色：白色。
正常工作电流为：(8∽20)*6=48~120mA（LED 灯数共 6 颗);
工作电压：5V 或 3.3V，由你选择的 VDD 电源电压（5V 或 3.3V）决定;
正常工作条件下，LED 可连续点亮 5 万小时;

6．指令表及硬件接口、编程案例

液晶模块中有两个 IC，一个 IC 在 LCD 玻璃上，叫 ST7586S,是液晶屏的驱动 IC,另一个 IC 在 PCB 上，叫 JLX-GB2312-3205,是标准的汉字库存储芯片，是不可改写的存储器(ROM).

不带字库 IC 时，单片机（MCU）也可以通过控制驱动 IC（ST7586S）使液晶屏显示。

带字库 IC 时，单片机（MCU）可以从字库 IC 中读取汉字的点阵数据，再将点阵数据写入驱动 IC（ST7586S）使液晶屏显示。

6.1 LCD 驱动 IC（ST7586S）的指令表：

INSTRUCTION TABLE

INSTRUCTION	A0	R/W	COMMAND BYTE								DESCRIPTION
			D7	D6	D5	D4	D3	D2	D1	D0	
NOP	0	0	0	0	0	0	0	0	0	0	No operation
RESET	0	0	0	0	0	0	0	0	0	1	Software reset
Power Save	0	0	0	0	0	1	0	0	0	SLP	Set power save mode SLP=0: Sleep in mode SLP=1: Sleep out mode
Partial Mode	0	0	0	0	0	1	0	0	1	PTL	Set partial mode PTL=0: Partial mode on PTL=1: Partial mode off
Inverse Display	0	0	0	0	1	0	0	0	0	INV	Set inverse display mode INV=0: Normal display INV=1: Inverse display
All Pixel ON/OFF	0	0	0	0	1	0	0	0	1	AP	Set all pixel on mode AP=0: All pixel off mode AP=1: All pixel on mode
Display ON/OFF	0	0	0	0	1	0	1	0	0	DSP	Set LCD display DSP=0: Display off DSP=1: Display on
Set Column Address	0	0	0	0	1	0	1	0	1	0	Set column address Starting column address: 00h≦XS≦7Fh Ending column address: XS≦XE≦7Fh
	1	0	XS15	XS14	XS13	XS12	XS11	XS10	XS9	XS8	
	1	0	XS7	XS6	XS5	XS4	XS3	XS2	XS1	XS0	
	1	0	XE15	XE14	XE13	XE12	XE11	XE10	XE9	XE8	
	1	0	XE7	XE6	XE5	XE4	XE3	XE2	XE1	XE0	
Set Row Address	0	0	0	0	1	0	1	0	1	1	Set row address Starting row address: 00h≦YS≦9Fh Ending row address: YS≦YE≦9FH
	1	0	YS15	YS14	YS13	YS12	YS11	YS10	YS9	YS8	
	1	0	YS7	YS6	YS5	YS4	YS3	YS2	YS1	YS0	
	1	0	YE15	YE14	YE13	YE12	YE11	YE10	YE9	YE8	
	1	0	YE7	YE6	YE5	YE4	YE3	YE2	YE1	YE0	
Write Display Data	0	0	0	0	1	0	1	1	0	0	Write display data to DDRAM
	1	0	D7	D6	D5	D4	D3	D2	D1	D0	
Read Display Data	0	0	0	0	1	0	1	1	1	0	Read display data from DDRAM
	1	1	D7	D6	D5	D4	D3	D2	D1	D0	
Partial Display Area	0	0	0	0	1	1	0	0	0	0	Set partial area Partial display address start: 00h≦PTS≦9Fh Partial display address end: 00h≦PTE≦9Fh Display Area: 64≦Duty≦160
	1	0	PTS15	PTS14	PTS13	PTS12	PTS11	PTS10	PTS9	PTS8	
	1	0	PTS7	PTS6	PTS5	PTS4	PTS3	PTS2	PTS1	PTS0	
	1	0	PTE15	PTE14	PTE13	PTE12	PTE11	PTE10	PTE9	PTE8	
	1	0	PTE7	PTE6	PTE5	PTE4	PTE3	PTE2	PTE1	PTE0	
Scroll Area	0	0	0	0	1	1	0	0	1	1	Set scroll area Top Area: TA=00h~A0h Scrolling Area: SA=00h~A0h Bottom Area: BA=00h~A0h TA+SA+BA=160
	1	0	TA7	TA6	TA5	TA4	TA3	TA2	TA1	TA0	
	1	0	SA7	SA6	SA5	SA4	SA3	SA2	SA1	SA0	
	1	0	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	
Display Control	0	0	0	0	1	1	0	1	1	0	Set scan direction of COM and SEG MY=0: COM0→COM159 MY=1: COM159→COOM0 MX=0: SEG0→SEG383 MX=1: SEG383→SEG0
	1	0	MY	MX	0	0	0	0	0	0	
Start Line	0	0	0	0	1	1	0	1	1	1	Set display start line S=00h~9Fh
	1	0	S7	S6	S5	S4	S3	S2	S1	S0	

INSTRUCTION	A0	R/W	COMMAND BYTE								DESCRIPTION
			D7	D6	D5	D4	D3	D2	D1	D0	
Display Mode	0	0	0	0	1	1	1	0	0	M	Set display mode M=0: Gray mode M=1: Monochrome mode
Enable DDRAM Interface	0	0	0	0	1	1	1	0	1	0	Enable DDRAM interface
	1	0	0	0	0	0	0	0	1	0	
Display Duty	0	0	1	0	1	1	0	0	0	0	Set display duty DT=03h~9Fh
	1	0	DT7	DT6	DT5	DT4	DT3	DT2	DT1	DT0	
First Output COM	0	0	1	0	1	1	0	0	0	1	Set first output COM FC=00h~9Fh
	1	0	FC7	FC6	FC5	FC4	FC3	FC2	FC1	FC0	
FOSC Divider	0	0	1	0	1	1	0	0	1	1	Set FOSC dividing ratio
	1	0	0	0	0	0	0	0	FOD1	FOD0	
Partial Display	0	0	1	0	1	1	0	1	0	0	Set partial display mode
	1	0	1	0	1	0	0	0	0	0	
N-Line Inversion	0	0	1	0	1	1	0	1	0	1	Set N-Line inversion
	1	0	M	0	0	NL4	NL3	NL2	NL1	NL0	
Read Modify Write	0	0	1	0	1	1	1	0	0	RMW	Read modify write control RMW=0: Enable read modify write RMW=1: Disable read modify write
Set Vop	0	0	1	1	0	0	0	0	0	0	Set Vop
	1	0	Vop7	Vop6	Vop5	Vop4	Vop3	Vop2	Vop1	Vop0	
	1	0	-	-	-	-	-	-	-	Vop8	
Vop Increase	0	0	1	1	0	0	0	0	0	1	Vop increase one step
Vop Decrease	0	0	1	1	0	0	0	0	1	0	Vop decrease one step
BIAS System	0	0	1	1	0	0	0	0	1	1	Set BIAS system
	1	0	-	-	-	-	-	BS2	BS1	BS0	
Booster Level	0	0	1	1	0	0	0	1	0	0	Set booster level
	1	0	-	-	-	-	-	BST2	BST1	BST0	
Vop Offset	0	0	1	1	0	0	0	1	1	1	Set Vop offset
	1	0	0	VOF6	VOF5	VOF4	VOF3	VOF2	VOF1	VOF0	
Analog Control	0	0	1	1	0	1	0	0	0	0	Enable analog circuit
	1	0	0	0	0	1	1	1	0	1	
Auto Read Control	0	0	1	1	0	1	0	1	1	1	Auto read control XARD=0: Enable auto read XARD=1: Disable auto read
	1	0	1	0	0	XARD	1	1	1	1	
OTP WR/RD Control	0	0	1	1	1	0	0	0	0	0	OTP WR/RD control WR/RD=0: Enable OTP read WR/RD=1: Enable OTP write
	1	0	0	0	WR /RD	0	0	0	0	0	
OTP Control Out	0	0	1	1	1	0	0	0	0	1	OTP control out
OTP Write	0	0	1	1	1	0	0	0	1	0	OTP programming procedure
OTP Read	0	0	1	1	1	0	0	0	1	1	OTP up-load procedure
OTP Selection Control	0	0	1	1	1	0	0	1	0	0	OTP selection control Ctrl=0: Disable OTP Ctrl=1: Enable OTP
	1	0	0	Ctrl	0	1	1	0	0	1	
OTP Programming Setting	0	0	1	1	1	0	0	1	0	1	OTP programming setting
	1	0	0	0	0	0	1	1	1	1	

INSTRUCTION	A0	R/W	COMMAND BYTE								DESCRIPTION
			D7	D6	D5	D4	D3	D2	D1	D0	
Frame Rate	0	0	1	1	1	1	0	0	0	1	Frame rate setting in different temperature range
	1	0	-	-	-	FRA4	FRA3	FRA2	FRA1	FRA0	
	1	0	-	-	-	FRB4	FRB3	FRB2	FRB1	FRB0	
	1	0	-	-	-	FRC4	FRC3	FRC2	FRC1	FRC0	
	1	0	-	-	-	FRD4	FRD3	FRD2	FRD1	FRD0	
Temperature Range	0	0	1	1	1	1	0	0	1	0	Temperature range setting
	1	0	-	TA6	TA5	TA4	TA3	TA2	TA1	TA0	
	1	0	-	TB6	TB5	TB4	TB3	TB2	TB1	TB0	
	1	0	-	TC6	TC5	TC4	TC3	TC2	TC1	TC0	
Temperature Gradient Compensation	0	0	1	1	1	1	0	1	0	0	Set temperature gradient compensation coefficient
	1	0	MT13	MT12	MT11	MT10	MT03	MT02	MT01	MT00	
	1	0	MT33	MT32	MT31	MT30	MT23	MT22	MT21	MT20	
	1	0	MT53	MT52	MT51	MT50	MT43	MT42	MT41	MT40	
	1	0	MT73	MT72	MT71	MT70	MT63	MT62	MT61	MT60	
	1	0	MT93	MT92	MT91	MT90	MT83	MT82	MT81	MT80	
	1	0	MTB3	MTB2	MTB1	MTB0	MTA3	MTA2	MTA1	MTA0	
	1	0	MTD3	MTD2	MTD1	MTD0	MTC3	MTC2	MTC1	MTC0	
	1	0	MTF3	MTF2	MTF1	MTF0	MTE3	MTE2	MTE1	MTE0	

请详细参考 ST7586S 的 IC 资料.

6.2 字库 IC（JLX-GB2312-3205）的操作指令及点阵数据的调用方法：

6.2.1 字库 IC 的操作指令只有两条，两条只选一条进行使用，操作指令表如下：

Instruction Set

Instruction	Description	Instruction Code(One-Byte)		Address Bytes	Dummy Bytes	Data Bytes
READ	Read Data Bytes	0000 0011	03 h	3	—	1 to ∞
FAST_READ	at Higher Speed	0000 1011	0B h	3	1	1 to ∞

Read Data Bytes

所有对本芯片 SPI 接口的操作只有 2 个，那就是 Read Data Bytes (READ “一般读取”)和 Read Data Bytes at Higher Speed (FAST_READ “快速读取点阵数据”)。.

以下分别介绍一般读取和快速读取：

6.2.1.1 Read Data Bytes（一般读取）

Read Data Bytes 需要用指令码来执行每一次操作。READ 指令的时序如下(图)：

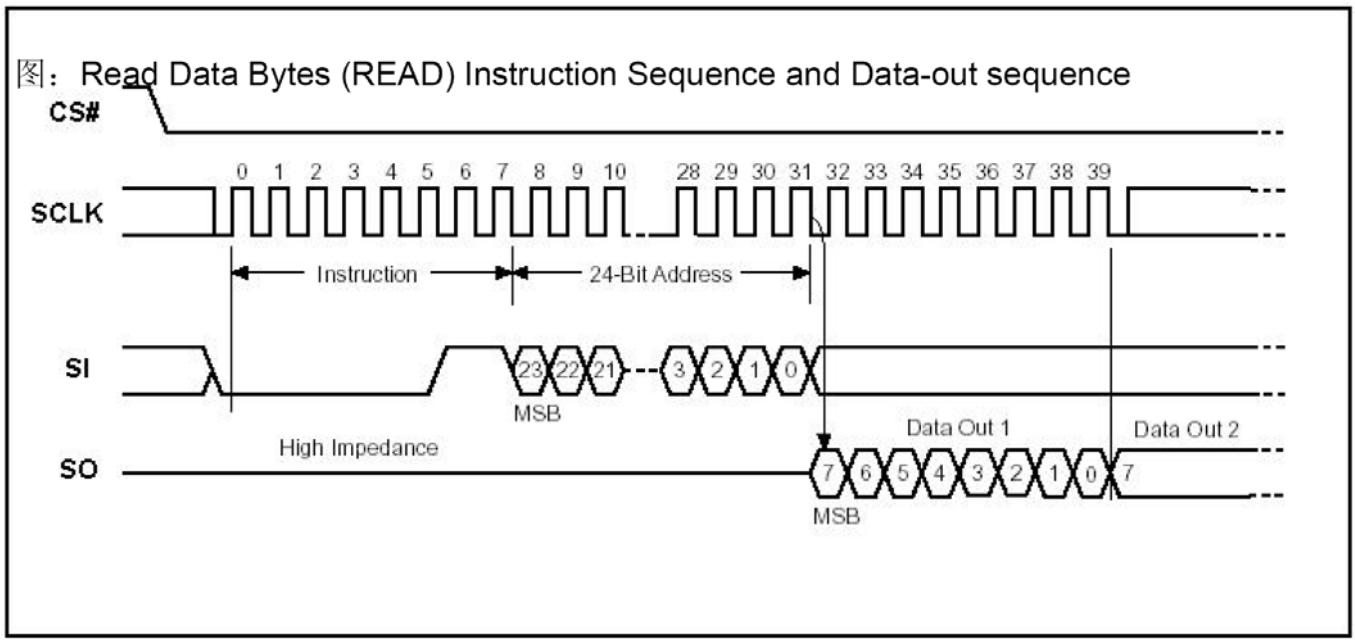
首先把片选信号（CS#）变为低，紧跟着的是 1 个字节的命令字（03 h）和 3 个字节的地址和通过串行数据输入引脚（SI）移位输入，每一位在串行时钟（SCLK）上升沿被锁存。

然后该地址的字节数据通过串行数据输出引脚（SO）移位输出，每一位在串行时钟（SCLK）下降沿被移出。

读取字节数据后，则把片选信号（CS#）变为高，结束本次操作。

如果片选信号（CS#）继续保持为底，则下一个地址的字节数据继续通过串行数据输出引脚（SO）移位输出。

图：Read Data Bytes (READ) Instruction Sequence and Data-out sequence



6.2.1.2 Read Data Bytes at Higher Speed（快速读取点阵数据）

Read Data Bytes at Higher Speed 需要用指令码来执行操作。READ_FAST 指令的时序如下(图)：

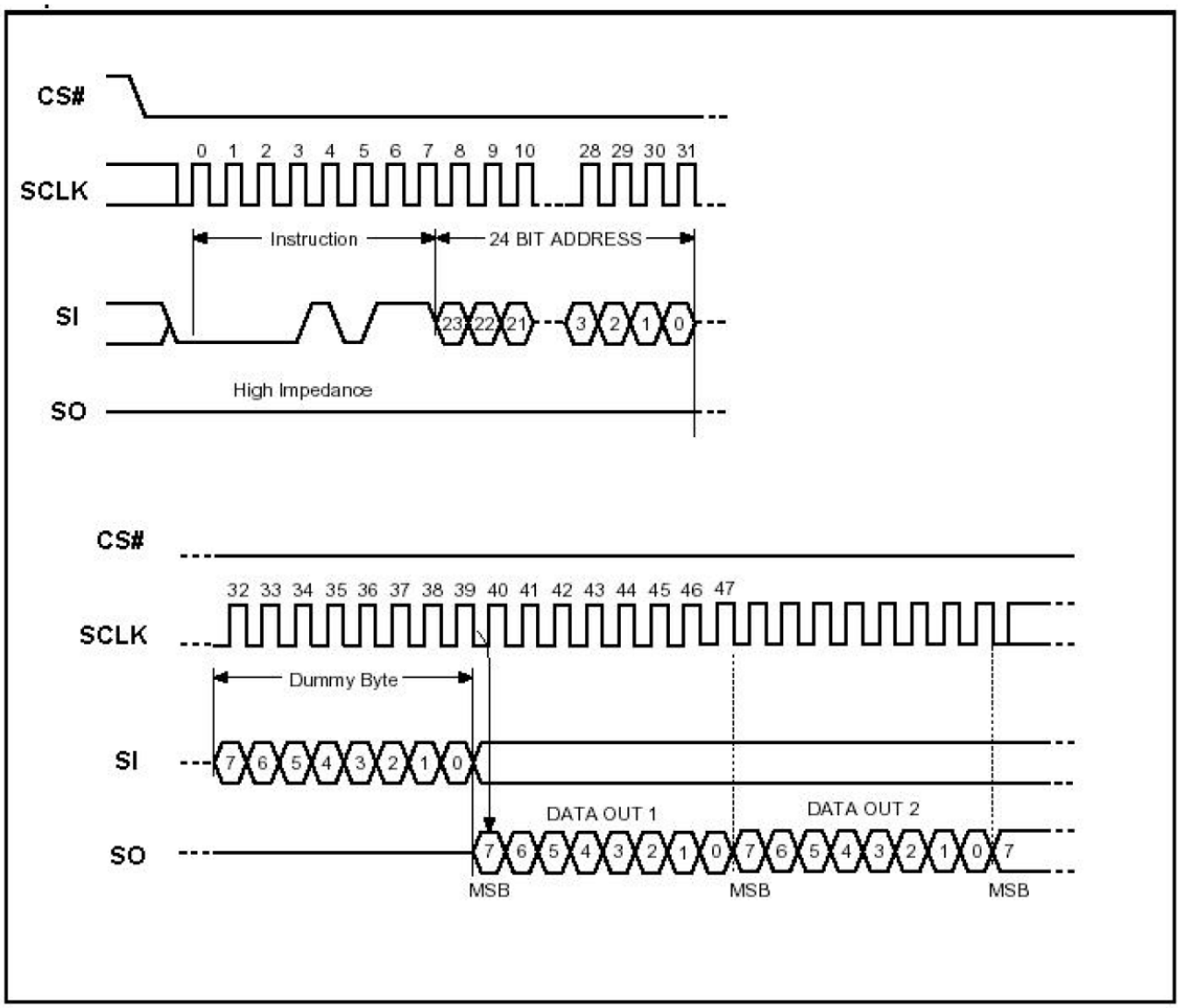
首先把片选信号（CS#）变为低，紧跟着的是 1 个字节的命令字（0B h）和 3 个字节的地址以及一个字节 Dummy Byte 通过串行数据输入引脚（SI）移位输入，每一位在串行时钟（SCLK）上升沿被锁存。

然后该地址的字节数据通过串行数据输出引脚（SO）移位输出，每一位在串行时钟（SCLK）下降沿被移出。

如果片选信号（CS#）继续保持为底，则下一个地址的字节数据继续通过串行数据输出引脚（SO）移位输出。例：读取一个 15x16 点阵汉字需要 32Byte，则连续 32 个字节读取后结束一个汉字的点阵数据读取操作。

如果不需要继续读取数据，则把片选信号（CS#）变为高，结束本次操作。

图：Read Data Bytes at Higher Speed (READ_FAST) Instruction Sequence and Data-out sequence



6.2.2字库调用方法：

6.2.2.1 汉字点阵排列格式

每个汉字在芯片中是以汉字点阵字模的形式存储的，每个点用一个二进制位表示，存 1 的点，当显示时可以在屏幕上显示亮点，存 0 的点，则在屏幕上不显示。点阵排列格式为横置横排：即一个字节的高位表示左面的点，低位表示右面的点（如果用户按 word mode 读取点阵数据，请注意高低字节的顺序），排满一行的点后再排下一行。这样把点阵信息用来直接在显示器上按上述规则显示，则将出现对应的汉字。

6.2.2.2 11X12 点、15X16点、24X24点、32X32点汉字及5X7 点、7X8 点、6X12点、12X24点字符、12 点阵不等宽字符、16点阵不等宽字符的排列格式：详见字库IC资料（JLX-GB2312-3205）的第19-26页。

6.2.2.3 汉字点阵字库地址表如下：

	字库内容	编码体系	码位范围	字符数	起始地址	参考算法
1	11X12 点 GB2312 标准点阵字库	GB2312	A1A1-F7FE	6763+846	0000	6.3.1.1
2	15X16 点 GB2312 标准点阵字库	GB2312	A1A1-F7FE	6763+846	2C9D0	6.3.1.2
3	24X24 点 GB2312 标准点阵字库	GB2312	A1A1-F7FE	6763+846	68190	6.3.1.3
4	32X32 点 GB2312 标准点阵字库	GB2312	A1A1-F7FE	6763+846	EDF00	6.3.1.4
5	6X12 点国标扩展字符	GB2312	A1A1-ABC0	126	1DBE0C	6.3.1.5
6	6X12 点 ASCII 字符	ASCII	20~7F 96		1DBE00	6.3.2.3
7	12 点阵不等宽 ASCII 方头（Arial）字符	ASCII	20~7F 96		1DC400	6.3.2.7
8	12 点阵不等宽 ASCII 白正字符	ASCII	20~7F	96	1DCDC0	6.3.2.8
9	8X16 点国标扩展字符	GB2312	A1A1-ABC0	126	1DD790	6.3.1.6
10	8X16 点 ASCII 字符	ASCII	20~7F 96		1DD780	6.3.2.4
11	5X7 点 ASCII 字符	ASCII	20~7F 96		1DDF80	6.3.2.1
12	7X8 点 ASCII 字符	ASCII	20~7F 96		1DE280	6.3.2.2
13	16 点阵不等宽 ASCII 方头（Arial）字符	ASCII	20~7F 96		1DE580	6.3.2.9
14	16 点阵不等宽 ASCIII 白正字符	ASCII	20~7F	96	1DF240	6.3.2.10
15	12X24 点国标扩展字符	GB2312	A1A1-ABC0	126	1DFF30	6.3.1.8
16	12X24 点 ASCII 字符	ASCII	20~7F 96		1DFF00	6.3.2.5
17	24 点阵不等宽 ASCII 方头（Arial）字符	ASCII	20~7F 96		1E22D0	6.3.2.11
18	24 点阵不等宽 ASCII 白正字符	ASCII	20~7F	96	1E3E90	6.3.2.12
19	16X32 点国标扩展字符	GB2312	A1A1-ABC0	126	1E5A90	6.3.1.9
20	16X32 点 ASCII 字符	ASCII	20~7F 96		1E5A50	6.3.2.6
21	32 点阵不等宽 ASCII 方头（Arial）字符	ASCII	20~7F 96		1E99D0	6.3.2.13
22	32 点阵不等宽 ASCII 白正字符	ASCII	20~7F	96	1ECA90	6.3.2.14
23	保留区				1EFB50	
29	输入法码表	GB2312			1F36F0	
32	保留区				1F7CC8	

6.2.2.4 字符在芯片中的地址计算方法：

用户只要知道字符的内码，就可以计算出该字符点阵在芯片中的地址，然后就可从该地址连续读出点阵信息用于显示。

举例说明:15X16 点 GB2312 标准点阵字库:

参数说明：

GBCode表示汉字内码。

MSB 表示汉字内码GBCode的高8bits。

LSB 表示汉字内码GBCode 的低8bits。

Address 表示汉字或ASCII字符点阵在芯片中的字节地址。

BaseAdd：说明点阵数据在字库芯片中的起始地址。

计算方法：

```
BaseAdd=0x2C9D0;
if(MSB >=0xA1 && MSB <= 0Xa9 && LSB >=0xA1)
    Address =( (MSB - 0xA1) * 94 + (LSB - 0xA1))*32+ BaseAdd;
else if(MSB >=0xB0 && MSB <= 0xF7 && LSB >=0xA1)
    Address = ((MSB - 0xB0) * 94 + (LSB - 0xA1)+ 846)*32+ BaseAdd;
```

6.3 接口方式及程序：

6.3.1 液晶模块与 MPU(以 8051 系列单片机为例)接口图如下：

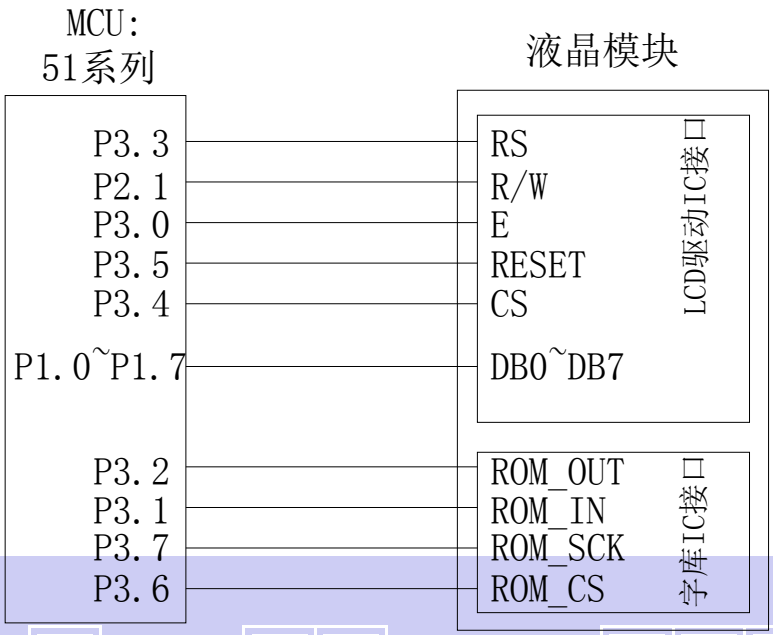


图 3：并行接口图

6.3.2 并行程序：

```
/*    液晶模块型号：JLX384160G-973-PL,
      并行接口,6800 时序,
      驱动 IC 是:ST7586S(or compatible),
      字库 IC:JLX-GB2312-3205
      可以调用字库 ROM 里的汉字：32*32 点阵，16*32 点阵，16*16 点阵，8*16 点阵，12*12 点阵，5*7 点阵等
      版权所有：晶联讯电子：网址  http://www.jlxlcd.cn;
        */
        #include <reg51.H>
        #include <intrins.h>
        #include <Ctype.h>

        #include <ASCII_TABLE_5X8_8X16_12x24_16x32_horizontal.h>

        sbit lcd_cs1 =  P3^4;
        sbit lcd_reset= P3^5;
        sbit lcd_rs  =  P3^3;
        sbit lcd_rw  =  P2^1;
        sbit lcd_e   =  P3^0;

        sbit Rom_IN   =  P3^1;  //字库 IC 接口定义:Rom_IN 就是字库 IC 的 SI
        sbit Rom_OUT  =  P3^2;  //字库 IC 接口定义:Rom_OUT 就是字库 IC 的 SO
        sbit Rom_SCK  =  P3^7;  //字库 IC 接口定义:Rom_SCK 就是字库 IC 的 SCK
        sbit Rom_CS   =  P3^6;  //字库 IC 接口定义 Rom_CS 就是字库 IC 的 CS#

        /*另外：DB0~DB7 与 P1.0~P1.7 相连*/
        sbit key  =  P2^0;  //按键：我的主板上是 P2.0 口与 GND 之间接一个按键


        #define uchar unsigned char
```

```
#define uint unsigned int
#define ulong unsigned long

uchar code bmp320160_1[];

//延时：1 毫秒的 i 倍
void delay(int i)
{
   int j,k;
   for(j=0;j<i;j++)
         for(k=0;k<110;k++);
}
//延时：1us 的 i 倍
void delay_us(int i)
{
   int j,k;
   for(j=0;j<i;j++)
         for(k=0;k<1;k++);
}

//等待一个按键，我的主板是用 P2.0 与 GND 之间接一个按键
void waitkey()
{
 repeat:
   if (key==1) goto repeat;
   else delay(2000);
}

//写指令到 LCD 模块
void transfer_command_lcd(int data1)
{

   lcd_cs1=0;
   lcd_rs=0;
   lcd_e=0;
   lcd_rw=0;
   P1=data1;
   lcd_e=1;
   delay_us(1);
   lcd_cs1=1;
   lcd_e=0;
}

//写数据到 LCD 模块
void transfer_data_lcd(int data1)
{
   lcd_cs1=0;
   lcd_rs=1;
   lcd_e=0;
   lcd_rw=0;
   P1=data1;
   lcd_e=1;
// delay_us(1);
   lcd_cs1=1;
   lcd_e=0;
}

/*LCD 模块初始化*/
void initial_lcd()
{
```

```
    lcd_reset=1;
    lcd_reset=0;                        //硬件复位
    delay(10);
    lcd_reset=1;                        //硬件复位完成后置高
    delay(10);

    transfer_command_lcd(0x11);     //退出睡眠模式

    transfer_command_lcd(0xC0);     // 设置 VOP
    transfer_data_lcd(0x2c);        // 设置 VOP 的值的低 8 位（总共 9 位）,每调一级是 0.03667V
    transfer_data_lcd(0x01);        // 设置 VOP 的值的第 9 位，也是最高一位
    transfer_command_lcd(0xC3);     // 设置 BIAS
    transfer_data_lcd(0x02);        // 00：BIAS = 1/14  02 = 1/12
    transfer_command_lcd(0xC4);     // 设置升压倍数
    transfer_data_lcd(0x07);         // 07：8 倍压

    transfer_command_lcd(0xD0);     // 允许模拟电路
    transfer_data_lcd(0x1D);         // 允许模拟电路

    transfer_command_lcd(0xB5);     // N-Line = 13
    transfer_data_lcd(0x00);         // 8d


    transfer_command_lcd(0x38);     // 0x38：设置为灰度模式； 0x39:  设置为黑白模式。
    transfer_command_lcd(0x3A);     // 允许 DDRAM 接口：单色模式、4 灰度级、16 灰度级；
    transfer_data_lcd(0x02);         // 0x03:16 灰度级；0x02:4 灰度级或单色模式。

// transfer_command_lcd(0x39);     // 39:  设置为黑白模式
// transfer_command_lcd(0x3A);     // 允许 DDRAM 接口
// transfer_data_lcd(0x02);         // 允许 DDRAM 接口
    transfer_command_lcd(0x36);     // 扫描顺序设置
    transfer_data_lcd(0x00);         // 扫描顺序设置:MX=1,MY=1: 从左到右，从上到下的扫描顺序
    transfer_command_lcd(0xB0);     // Duty 设置
    transfer_data_lcd(0x9f);        // Duty 设置:1/160
    transfer_command_lcd(0x20);     // 反显设置：OFF

    transfer_command_lcd(0xf1);      //温度补偿，温度变化改变帧频
    transfer_data_lcd(0x15);
    transfer_data_lcd(0x15);
    transfer_data_lcd(0x15);
    transfer_data_lcd(0x15);

    transfer_command_lcd(0xb1);     // 扫描起始行设置
    transfer_data_lcd(0x00);         // 扫描起始行设置：从 COM0 开始

    transfer_command_lcd(0x29);     // 打开显示：DISPLAY ON

}



/*写 LCD 行列地址：X 为起始的列地址，Y 为起始的行地址，x_total,y_total 分别为列地址及行地址的起点到终点的差值 */
void lcd_address(int x,int y,x_total,y_total)
{
    int x_end,y_end;

    x_end=x+(x_total-1)/3;
    y_end=y+y_total-1;

    transfer_command_lcd(0x2A);
```

```
    transfer_data_lcd((x>>8)&0x00ff);
    transfer_data_lcd(x&0x00ff);
    transfer_data_lcd(x_end>>8&0x00ff);
    transfer_data_lcd(x_end&0x00ff);
    transfer_command_lcd(0x2B);
    transfer_data_lcd((y>>8)&0x00ff);
    transfer_data_lcd(y&0x00ff);
    transfer_data_lcd(y_end>>8&0x00ff);
    transfer_data_lcd(y_end&0x00ff);
}


//传送同一个地址的 3 个点阵的黑白的数据：比如 SEG0、SEG1、SEG2（这 3 个点阵是同一个列地址，无法分开）
//送数据时左起第 1 列的数据是“D7 D6 D5 D4 D3 D2 D1 D0” 中的高 3 位---D7 D6 D5, 第 2 列是中 3 位---D4 D3 D2,第 3 列是低两位---D1 D0。
void transfer_mono_data_3pixel(uchar mono_data)
{
    uchar gray_data=0;

    if(mono_data&0x80)
    {
        gray_data=0xe0;   //二进制 11100000，就是给 D7、D6、D5 赋值
    }
    else
    {
        gray_data=0;
    }
    mono_data<<=1;
    if(mono_data&0x80)
    {
        gray_data+=0x1c;  //二进制 00011100，就是给 D4、D3、D2 赋值
    }
    else;
    mono_data<<=1;
    if(mono_data&0x80)
    {
        gray_data+=0x03; //二进制 00000011，就是给 D1、D0 赋值
    }
    else;
    transfer_data_lcd(gray_data);     //display 3 dots (seg_N,seg_N+1,seg_N+2)

}


//显示 6 个点阵
void transfer_mono_data_6pixel(uchar dat1)
{
    transfer_mono_data_3pixel(dat1);
    transfer_mono_data_3pixel(dat1<<3);
}

//显示 8 个点阵
void transfer_mono_data_8pixel(uchar dat1)
{
    transfer_mono_data_3pixel(dat1);//传送 dat1 的 D7\D6\D5 这 3 位，对应 3 个点阵(第 1、2、3 个）会显示出来；列地址是自动+1 的
    transfer_mono_data_3pixel(dat1<<3);//传送 dat1 的 D4\D3\D2 这 3 位，对应 3 个点阵(第 4、5、6 个）会显示出来；列地址是自动+1 的
    transfer_mono_data_3pixel(dat1<<6);//传送 dat1 的 D1\D0 这 2 位，对应 3 个点阵(第 7、8、9 个）会显示出来
    //这个液晶驱动 IC 的每个列地址管 3 个点阵，无法分开，所以第 7、8 个点阵会连累到第 9 个点阵，结果是每次显示 9 个点阵，只不过第 9 个点阵会补“0”
    //如果第 9 个点阵本来有显示内容，就会被无情地清掉
}
```

```
//显示 9 个点阵
void transfer_mono_data_9pixel(uchar dat1,uchar dat2)
{
   transfer_mono_data_6pixel(dat1);//先显示 6 个点阵
   transfer_mono_data_3pixel((dat1<<6)|(dat2>>2));   //显示 dat1 的 D1、D0 和 dat2 的 D7 位，对应 3 个点阵(第 7、7、9 个）会显示出来；列地址是自动+1 的
}

//显示 12 个点阵
void transfer_mono_data_12pixel(uchar dat1,uchar dat2)
{
   transfer_mono_data_9pixel(dat1,dat2);//先显示 9 个点阵
   transfer_mono_data_3pixel(dat2<<1);          //传送 dat2 的 D6\D5\D4 这 3 位，对应第 10、11、12 个个点阵会显示出来；列地址是自动+1 的
}

//显示 15 个点阵
void transfer_mono_data_15pixel(uchar dat1,uchar dat2)
{
   transfer_mono_data_12pixel(dat1,dat2);       //先显示 12 个点阵
   transfer_mono_data_3pixel(dat2<<4);          //传送 dat2 的 D3\D2\D1 这 3 位，对应第 13、14、15 个点阵会显示出来；列地址是自动+1 的
}

//显示 16 个点阵
 void transfer_mono_data_16pixel(uchar dat1,uchar dat2)
{
   transfer_mono_data_15pixel(dat1,dat2); //先显示 15 个点阵
   transfer_mono_data_3pixel(dat2<<7);          //显示第 16 个点阵,对应 dat2 的 D0 位。
   //这个液晶驱动 IC 的每个列地址管 3 个点阵，无法分开，所以第 16 个点阵会连累到第 17、18 个点阵，结果是每次显示 18 个点阵，只不过第 17、18 个点阵会补“0”
   //如果第 17、18 个点阵本来有显示内容，就会被无情地清掉
}

//显示 18 个点阵
void transfer_mono_data_18pixel(uchar dat1,uchar dat2,uchar dat3)
{
   transfer_mono_data_15pixel(dat1,dat2);            //先显示 15 个点阵
   transfer_mono_data_3pixel((dat2<<7)|(dat3>>1));   //传送 dat2 的 D0 和 dat3 的 D7、D6 这 3 位，对应第 16、17、18 个点阵会显示出来；列地址是自动+1 的
}

//显示 21 个点阵
void transfer_mono_data_21pixel(uchar dat1,uchar dat2,uchar dat3)
{
   transfer_mono_data_18pixel(dat1,dat2,dat3); //先显示 18 个点阵
   transfer_mono_data_3pixel(dat3<<2);          //传送 dat3 的 D5、D4、D3 这 3 位，对应第 19、20、21 个点阵会显示出来；列地址是自动+1 的
}

//显示 24 个点阵。方法一：
void transfer_mono_data_24pixel(uchar dat1,uchar dat2,uchar dat3)
{
   transfer_mono_data_21pixel(dat1,dat2,dat3); //先显示 21 个点阵
   transfer_mono_data_3pixel(dat3<<5);          //传送 dat3 的 D2、D1、D0 这 3 位，对应第 22、23、24 个点阵会显示出来；列地址是自动+1 的
```

```
    }


    //显示 24 个点阵。方法二：
    /*
    void transfer_mono_data_24pixel(uchar dat1,uchar dat2,uchar dat3)     //每个字节显示 8 个点阵，显示 8*3=24 个点阵
    {
        transfer_mono_data_3pixel(dat1);                  //传送 dat1 的 D7\D6\D5 这 3 位，对应第 1、2、3 个点阵会显示出来，列地址是自动+1 的
        transfer_mono_data_3pixel(dat1<<3);               //传送 dat1 的 D4\D3\D2 这 3 位，对应第 4、5、6 个点阵会显示出来，列地址是自动+1 的

        transfer_mono_data_3pixel((dat1<<6)|(dat2>>2));   //传送 dat1 的 D1\D0 和 dat2 的 D7 位，对应第 7、8、9 个点阵会显示出来，列地址是自动+1 的

        transfer_mono_data_3pixel(dat2<<1);               //传送 dat2 的 D6\D5\D4 这 3 位，对应第 10、11、12 个个点阵会显示出来；列地址是自动+1 的

        transfer_mono_data_3pixel(dat2<<4);               //传送 dat2 的 D3\D2\D1 这 3 位，对应第 13、14、15 个点阵会显示出来；列地址是自动+1 的

        transfer_mono_data_3pixel((dat2<<7)|(dat3>>1));   //传送 dat2 的 D0 和 dat3 的 D7、D6 这 3 位，对应第 16、17、18 个点阵会显示出来；列地址是自动+1 的

        transfer_mono_data_3pixel(dat3<<2);               //传送 dat3 的 D5、D4、D3 这 3 位，对应第 19、20、21 个点阵会显示出来；列地址是自动+1 的

        transfer_mono_data_3pixel(dat3<<5);               //传送 dat3 的 D2、D1、D0 这 3 位，对应第 22、23、24 个点阵会显示出来；列地址是自动+1 的

    }
    */

    //显示 27 个点阵
    void transfer_mono_data_27pixel(uchar dat1,uchar dat2,uchar dat3,uchar dat4)
    {
        transfer_mono_data_24pixel(dat1,dat2,dat3); //先显示 24 个点阵
        transfer_mono_data_3pixel(dat4);            //传送 dat4 的 D7、D6、D5 这 3 位，对应第 25、26、27 个点阵会显示出来；列地址是自动+1 的
    }
    //显示 30 个点阵
    void transfer_mono_data_30pixel(uchar dat1,uchar dat2,uchar dat3,uchar dat4)
    {
        transfer_mono_data_24pixel(dat1,dat2,dat3); //先显示 24 个点阵
        transfer_mono_data_6pixel(dat4);            //再显示 6 个点阵，24+6=30
    }
    //显示 32 个点阵
    void transfer_mono_data_32pixel(uchar dat1,uchar dat2,uchar dat3,uchar dat4)
    {
        transfer_mono_data_24pixel(dat1,dat2,dat3); //先显示 24 个点阵
        transfer_mono_data_8pixel(dat4);            //再显示 8 个点阵，24+8=32
        //这个液晶驱动 IC 的每个列地址管 3 个点阵，无法分开，所以第 31、32 个点阵会连累到第 33 个点阵，结果是每次显示 33 个点阵，只不过第 33 个点阵会补“0”
        //如果第 33 个点阵本来有显示内容，就会被无情地清掉
    }

    //显示 33 个点阵
    void transfer_mono_data_33pixel(uchar dat1,uchar dat2,uchar dat3,uchar dat4,uchar dat5)
    {
        transfer_mono_data_24pixel(dat1,dat2,dat3);       //先显示 24 个点阵
```

```
    transfer_mono_data_9pixel(dat4,dat5);              //再显示 9 个点阵
}

//显示 48 个点阵
void transfer_mono_data_48pixel(uchar dat1,uchar dat2,uchar dat3,uchar dat4,uchar dat5,uchar dat6)
{
    transfer_mono_data_24pixel(dat1,dat2,dat3);        //先显示 24 个点阵
    transfer_mono_data_24pixel(dat4,dat5,dat6);        //再显示 24 个点阵
}


//传送同一个地址的 3 个点阵的 4 灰度级的数据：比如 SEG0、SEG1、SEG2,这 3 个点阵是同一个列地址，无法分开
//送灰度数据(gray_data)时，SEG0 对应高 3 位（D7、D6、D5）,SEG1 对应中 3 位（D4、D3、D2),SEG2 对应低两位（D1、D0)。
void transfer_gray_data_3pixel(uchar dat1)
{
    uchar gray_data;
    gray_data=dat1&0xc0;;    //给 gray_data 的 D7、D6 赋值(=dat1 的 D7、D6)
    if((dat1&0xc0)==0xc0)
    {
        gray_data|=0x20; //给 gray_data 的 D5 赋值，当 dat1 的 D7、D6 都是 1 的时候，gray_data 的 D5=1,当 dat1 的 D7\D6 不都是 1 的时候，
gray_data 的 D5=0
    }
    gray_data|=((dat1>>1)&0x18);    //给 gray_data 的 D4、D3 赋值（=dat1 的 D5、D4)
    if((dat1&0x30)==0x30)
    {
        gray_data|=0x04; //给 gray_data 的 D2 赋值，当 dat1 的 D5、D4 都是 1 的时候，gray_data 的 D2=1,当 dat1 的 D7、D6 不都是 1 的时候，
gray_data 的 D2=0
    }
    gray_data|=((dat1>>2)&0x03);      //给 gray_data 的 D1、D0 赋值(=dat1 的 D3、D2)
    transfer_data_lcd(gray_data);     //传送 1 个字节灰度数据给液晶驱动 IC,对应的 3 个点阵会显示(seg_N,seg_N+1,seg_N+2)
}


//传送同一个地址的 12 个点阵的 4 灰度的数据：比如 SEG0、SEG1、SEG2.....SEG9、SEG10、SEG11（这 12 个点阵是 4 个列地址）
//每 2 位数据对应一个点阵，12 个点阵用：2*12=24 位，即 3 个字节:dat1、dat2、dat3
void transfer_gray_data_12pixel(uchar dat1,uchar dat2,uchar dat3)
{
    transfer_gray_data_3pixel(dat1);                   //显示 3 个点阵(seg_N,seg_N+1,SEG_N+2)
    transfer_gray_data_3pixel((dat1<<6)|(dat2>>2));    //显示 3 个点阵(seg_N+3,seg_N+4,SEG_N+5)
    transfer_gray_data_3pixel((dat2<<4)|(dat3>>4));    //显示 3 个点阵(seg_N+6,seg_N+7,SEG_N+8)
    transfer_gray_data_3pixel(dat3<<2);                      //显示 3 个点阵(seg_N+9,seg_N+10,SEG_N+11)
}



/*清屏*/
void clear_screen()
{
    int i,j;
    lcd_address(0,0,384,160);
     transfer_command_lcd(0x2c);
     for(i=0;i<160;i++)
    {
        for(j=0;j<24;j++)
          {
            transfer_mono_data_18pixel(0x00,0x00,0x00); //每个字节显示 8 个点阵，显示 8*3=24 个点阵
          }
    }
}
```

```
/*显示 8*16 点阵 ASCII 码字符或等同于 8*16 点阵的图像*/
void disp_8x16(int x,int y,uchar *dp)
{
   int i,j;
   uchar dat1;
   lcd_address(x,y,8,16);
   transfer_command_lcd(0x2c);
   for(i=0;i<16;i++)
   {
        for(j=0;j<1;j++)
        {
             dat1=*dp;dp++;
             transfer_mono_data_8pixel(dat1);
        }
   }
}


//括号里的参数分别为（列，行，数据指针）
void display_string_8x16(int x,int y,uchar *text)
{
   uint i=0,j,n,dat1;
   while(text[i]>0x00)
   {
        if((text[i]>=0x20)&&(text[i]<=0x7e))
        {
             j=text[i]-0x20;
             lcd_address(x,y,8,16);
             transfer_command_lcd(0x2c);
             for(n=0;n<16;n++)
             {
                  dat1=ascii_table_8x16[j][n];
                  transfer_mono_data_8pixel(dat1);
             }
             i++;
          x+=3;
        }
        else
        i++;
   }
}


//括号里的参数分别为（列，行，数据指针）
void display_string_12x24(int x,int y,uchar *text)
{
   uint i=0,j,n,dat1,dat2;
   while(text[i]>0x00)
   {
        if((text[i]>=0x20)&&(text[i]<=0x7e))
        {
             j=text[i]-0x20;
             lcd_address(x,y,12,24);
             transfer_command_lcd(0x2c);
             for(n=0;n<24;n++)
             {
                  dat1=ascii_table_12x24[j][2*n];
                  dat2=ascii_table_12x24[j][2*n+1];
```

```
                    transfer_mono_data_12pixel(dat1,dat2);
                }
                i++;
              x+=4;
          }
          else
          i++;
    }
}


//显示 12*12 点阵的图像
void disp_12x12(int x,int y,uchar *dp)
{
    int i,j;
    uchar dat1,dat2;

    lcd_address(x,y,12,12);

    transfer_command_lcd(0x2C);

    for(i=0;i<12;i++)
    {
        for(j=0;j<1;j++)//循环 1 次，每次显示 12 个点阵
          {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              transfer_mono_data_12pixel(dat1,dat2); //每个字节显示 8 个点阵，显示 8*2=16 个点阵
          }
    }
}


//显示 16*16 点阵的图像
void disp_16x16(int x,int y,uchar *dp)
{
    int i,j;
    uchar dat1,dat2;

    lcd_address(x,y,16,16);

    transfer_command_lcd(0x2C);

    for(i=0;i<16;i++)
    {
        for(j=0;j<1;j++)//循环 1 次，每次显示 18 个点阵
          {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              transfer_mono_data_16pixel(dat1,dat2); //每个字节显示 8 个点阵，显示 8*2=16 个点阵
          }
    }
}

//显示 18*18 点阵的图像
void disp_18x18(int x,int y,uchar *dp)
{
    int i,j;
    uchar dat1,dat2,dat3;
```

```
    lcd_address(x,y,18,18);

    transfer_command_lcd(0x2C);

    for(i=0;i<18;i++)
    {
        for(j=0;j<1;j++)//循环 1 次，每次显示 18 个点阵
          {
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                transfer_mono_data_18pixel(dat1,dat2,dat3); //每个字节显示 8 个点阵，显示 8*2=16 个点阵
          }
    }
}

//显示 21*21 点阵的图像
void disp_21x21(int x,int y,uchar *dp)
{
    int i,j;
    uchar dat1,dat2,dat3;

    lcd_address(x,y,21,21);

    transfer_command_lcd(0x2C);

    for(i=0;i<21;i++)
    {
        for(j=0;j<1;j++)//循环 1 次，每次显示 18 个点阵
          {
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                transfer_mono_data_21pixel(dat1,dat2,dat3); //每个字节显示 8 个点阵，显示 8*2=16 个点阵
          }
    }
}

//显示 24*24 点阵的图像
void disp_24x24(int x,int y,uchar *dp)
{
    int i,j;
    uchar dat1,dat2,dat3;

    lcd_address(x,y,24,24);

    transfer_command_lcd(0x2C);

    for(i=0;i<24;i++)
    {
        for(j=0;j<1;j++)//循环 1 次，每次显示 24 个点阵
          {
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                transfer_mono_data_24pixel(dat1,dat2,dat3); //每个字节显示 8 个点阵，显示 8*3=24 个点阵
          }
    }
}
```

```
//显示 27*27 点阵的图像
void disp_27x27(int x,int y,uchar *dp)
{
   int i,j;
   uchar dat1,dat2,dat3,dat4;

   lcd_address(x,y,27,27);

   transfer_command_lcd(0x2C);

   for(i=0;i<27;i++)
   {
       for(j=0;j<1;j++)//循环 1 次，每次显示 24 个点阵
         {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              dat3=*dp;dp++;
              dat4=*dp;dp++;
              transfer_mono_data_27pixel(dat1,dat2,dat3,dat4);  //每个字节显示 8 个点阵，显示 8*3=24 个点阵
         }
   }
}

//显示 30*30 点阵的图像
void disp_30x30(int x,int y,uchar *dp)
{
   int i,j;
   uchar dat1,dat2,dat3,dat4;

   lcd_address(x,y,30,30);

   transfer_command_lcd(0x2C);

   for(i=0;i<30;i++)
   {
       for(j=0;j<1;j++)//循环 1 次，每次显示 30 个点阵
         {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              dat3=*dp;dp++;
              dat4=*dp;dp++;
              transfer_mono_data_30pixel(dat1,dat2,dat3,dat4);  //每个字节显示 8 个点阵，显示 8*3=24 个点阵
         }
   }
}

//显示 32*32 点阵的图像
void disp_32x32(int x,int y,uchar *dp)
{
   int i,j;
   uchar dat1,dat2,dat3,dat4;

   lcd_address(x,y,32,32);

   transfer_command_lcd(0x2C);

   for(i=0;i<32;i++)
   {
       for(j=0;j<1;j++)//循环 1 次，每次显示 32 个点阵
         {
```

```
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                dat4=*dp;dp++;
                transfer_mono_data_32pixel(dat1,dat2,dat3,dat4);  //每个字节显示 8 个点阵，显示 8*4=32 个点阵
            }
        }
    }

    //显示 33*33 点阵的图像
    void disp_33x33(int x,int y,uchar *dp)
    {
        int i,j;
        uchar dat1,dat2,dat3,dat4,dat5;

        lcd_address(x,y,33,33);

        transfer_command_lcd(0x2C);

        for(i=0;i<33;i++)
        {
            for(j=0;j<1;j++)//循环 1 次，每次显示 24 个点阵
             {
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                dat4=*dp;dp++;
                dat5=*dp;dp++;
                transfer_mono_data_33pixel(dat1,dat2,dat3,dat4,dat5);  //每个字节显示 8 个点阵，显示 8*3=24 个点阵
            }
        }
    }

    //显示 48*48 点阵的图像
    void disp_48x48(int x,int y,uchar *dp)
    {
        int i,j;
        uchar dat1,dat2,dat3,dat4,dat5,dat6;

        lcd_address(x,y,48,48);

        transfer_command_lcd(0x2C);

        for(i=0;i<48;i++)
        {
            for(j=0;j<1;j++)//循环 1 次，每次显示 24 个点阵
             {
                dat1=*dp;dp++;
                dat2=*dp;dp++;
                dat3=*dp;dp++;
                dat4=*dp;dp++;
                dat5=*dp;dp++;
                dat6=*dp;dp++;
                transfer_mono_data_48pixel(dat1,dat2,dat3,dat4,dat5,dat6);  //每个字节显示 8 个点阵，显示 8*3=24 个点阵
            }
        }
    }

    //显示 384*160 点阵的图像
    void disp_384x160(uchar *dp)
```

```
{
   int i,j;
   uchar dat1,dat2,dat3;

   lcd_address(0,0,384,160);

   transfer_command_lcd(0x2C);

   for(i=0;i<160;i++)
   {
       for(j=0;j<16;j++)//循环 16 次，每次显示 24 个点阵，合计 384 个点阵
         {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              dat3=*dp;dp++;
              transfer_mono_data_24pixel(dat1,dat2,dat3); //每个字节显示 8 个点阵，显示 8*3=24 个点阵
         }
   }
}

//===显示测试画面：例如全显示，隔行显示，隔列显示，雪花显示=====
void test_display(uchar dat1,uchar dat2,uchar dat3)
{
   int i,j;

   lcd_address(0,0,384,160);

   transfer_command_lcd(0x2C);

   for(i=0;i<160;i++)
   {
       for(j=0;j<16;j++)//循环 16 次，每次显示 24 个点阵，合计 384 个点阵
         {

              transfer_mono_data_24pixel(dat1,dat2,dat3); //每个字节显示 8 个点阵，显示 8*3=24 个点阵
         }
   }
}

//显示 384*160 点阵的 4 灰度级图像
void disp_4gray_384x160(uchar *dp)
{
   uchar i,j;
   uchar dat1,dat2,dat3;
   lcd_address(0,0,384,160);  //
   transfer_command_lcd(0x2C);
   for(i=0;i<160;i++)
   {
       for(j=0;j<32;j++)//循环 26 次，每次显示 12 个点阵，合计 26*12=312 个点阵
         {
              dat1=*dp;dp++;
              dat2=*dp;dp++;
              dat3=*dp;dp++;
              transfer_gray_data_12pixel(dat1,dat2,dat3); //每个字节显示 4 个点阵，共显示 4*3=12 个点阵
         }

   }
}
/*显示 16*32 点阵 ASCII 码字符或等同于 8*16 点阵的图像*/
void disp_16x32(int x,int y,uchar *dp)
```

```
{
   int i,j;
   uchar dat1,dat2;

   lcd_address(x,y,16,132);

   transfer_command_lcd(0x2C);

   for(i=0;i<32;i++)
   {
       for(j=0;j<1;j++)//循环 1 次，每次显示 18 个点阵
        {
             dat1=*dp;dp++;
             dat2=*dp;dp++;
             transfer_mono_data_16pixel(dat1,dat2); //每个字节显示 8 个点阵，显示 8*2=16 个点阵
        }
   }
}

/****送指令到晶联讯字库 IC***/
void send_command_to_ROM( uchar datu )
{
   uchar i;
   for(i=0;i<8;i++ )
   {
        if(datu&0x80)
            Rom_IN = 1;
        else
            Rom_IN = 0;
            datu = datu<<1;
            Rom_SCK=0;
         delay_us(1);
            Rom_SCK=1;
//       delay_us(1);
   }
}

/****从晶联讯字库 IC 中取汉字或字符数据（1 个字节）***/
static uchar get_data_from_ROM( )
{

   uchar i;
   uchar ret_data=0;
   Rom_SCK=1;
   for(i=0;i<8;i++)
   {
        Rom_OUT=1;
        Rom_SCK=0;
        ret_data=ret_data<<1;
        if( Rom_OUT )
            ret_data=ret_data+1;
        else
            ret_data=ret_data+0;
        Rom_SCK=1;
   }
   return(ret_data);
}
```

```
/*从相关地址（addrHigh：地址高字节,addrMid：地址中字节,addrLow：地址低字节）中连续读出 DataLen 个字节的数据到 pBuff 的地址*/
/*连续读取*/
void get_n_bytes_data_from_ROM(uchar addrHigh,uchar addrMid,uchar addrLow,uchar *pBuff,uchar DataLen)
{
   uchar i;
   Rom_CS = 0;
   lcd_cs1=1;
   Rom_SCK=0;
   send_command_to_ROM(0x03);
   send_command_to_ROM(addrHigh);
   send_command_to_ROM(addrMid);
   send_command_to_ROM(addrLow);
   for(i = 0; i < DataLen; i++ )
        *(pBuff+i) =get_data_from_ROM();
   Rom_CS=1;
}

/***********/

ulong  fontaddr;
void disp_GB2312_32x32_string(uchar x,uchar y,uchar *text)
{
   uchar i= 0,j;
   uchar addrHigh,addrMid,addrLow ;
   uchar fontbuf[128];
   while((text[i]>0x00))
   {
        if( ((text[i]>=0xb0) && (text[i]<=0xf7) ) && (text[i+1]>=0xa1) )
        {
             fontaddr = (text[i]- 0xb0)*94;
             fontaddr += (text[i+1]-0xa1)+846;
             fontaddr = (ulong)(fontaddr*128);
             fontaddr = (ulong)(fontaddr+0Xedf00);

             addrHigh = (fontaddr&0xff0000)>>16;
             addrMid = (fontaddr&0xff00)>>8;
             addrLow = fontaddr&0xff;

             Rom_CS = 0;
             lcd_cs1=1;

             send_command_to_ROM(0x03);
             send_command_to_ROM(addrHigh);
             send_command_to_ROM(addrMid);
             send_command_to_ROM(addrLow);

             for(j = 0; j < 128; j++ )
             {
                  fontbuf[j] =get_data_from_ROM();
             }
             Rom_CS = 1;
             disp_32x32(x,y,fontbuf);
             i+=2;
             x+=11;
        }

        else if(( (text[i]>=0xa1) && (text[i]<=0xab) ) && (text[i+1]>=0xa1) )
        {

             fontaddr = (text[i]- 0xa1)*94;
```

```
                fontaddr += (text[i+1]-0xa1);
                fontaddr = (ulong)(fontaddr*128);
                fontaddr = (ulong)(fontaddr+0Xedf00);

                addrHigh = (fontaddr&0xff0000)>>16;
                addrMid = (fontaddr&0xff00)>>8;
                addrLow = fontaddr&0xff;

                Rom_CS = 0;
                lcd_cs1=1;
                Rom_SCK=0;

                send_command_to_ROM(0x03);
                send_command_to_ROM(addrHigh);
                send_command_to_ROM(addrMid);
                send_command_to_ROM(addrLow);

                for(j = 0; j < 128; j++ )
                {
                     fontbuf[j] =get_data_from_ROM();
                }
                Rom_CS = 1;

                disp_32x32(x,y+11,fontbuf);
                i+=2;
                x+=11;
            }

            else if( (text[i]>=0x20) && (text[i]<=0x7e) )
            {
                fontaddr = (text[i]- 0x20);
                fontaddr = (ulong)(fontaddr*64);
                fontaddr = (ulong)(fontaddr+0x1e5a50);

                addrHigh = (fontaddr&0xff0000)>>16;
                addrMid = (fontaddr&0xff00)>>8;
                addrLow = fontaddr&0xff;

                get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,64);

                disp_16x32(x,y,fontbuf);
                i+=1;
                x+=6;
            }

            else
                i++;
    }
}

/*=====从字库读数据，显示 16*16 点阵的汉字或 8*16 点阵的数字========*/
void disp_GB2312_16x16_string(uchar x,uchar y,uchar *text)
{
    uchar i= 0;
    uchar addrHigh,addrMid,addrLow ;
    uchar fontbuf[64];
    ulong  fontaddr;
    while((text[i]>0x00))
    {
```

```
            if(((text[i]>=0xb0) &&(text[i]<=0xf7))&&(text[i+1]>=0xa1))
            {
                  fontaddr = (text[i]- 0xb0)*94;
                  fontaddr += (text[i+1]-0xa1)+846;
                  fontaddr = (ulong)(fontaddr*32);
                  fontaddr = (ulong)(fontaddr+0x2c9d0);

                  addrHigh = (fontaddr&0xff0000)>>16;
                  addrMid = (fontaddr&0xff00)>>8;
                  addrLow = fontaddr&0xff;
                  get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
                  disp_16x16(x,y,fontbuf);
                  i+=2;
                  x+=5;
            }
            else if(((text[i]>=0xa1) &&(text[i]<=0xab))&&(text[i+1]>=0xa1))
            {
                  fontaddr = (text[i]- 0xa1)*94;
                  fontaddr += (text[i+1]-0xa1);
                  fontaddr = (ulong)(fontaddr*32);
                  fontaddr = (ulong)(fontaddr+0x2c9d0);

                  addrHigh = (fontaddr&0xff0000)>>16;
                  addrMid = (fontaddr&0xff00)>>8;
                  addrLow = fontaddr&0xff;
                  get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
                  disp_16x16(x,y,fontbuf);
                  i+=2;
                  x+=5;
            }
            else if((text[i]>=0x20) &&(text[i]<=0x7e))
            {
                  uchar fontbuf[16];
                  fontaddr = (text[i]- 0x20);
                  fontaddr = (ulong)(fontaddr*16);
                  fontaddr = (ulong)(fontaddr+0x1dd780);
                  addrHigh = (fontaddr&0xff0000)>>16;
                  addrMid = (fontaddr&0xff00)>>8;
                  addrLow = fontaddr&0xff;

                  get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,16 );

                  disp_8x16(x,y,fontbuf);
                  i+=1;
                  x+=3;
            }
            else
                  i++;
      }

}

//------------------------------
void main ()
{

   while(1)
   {
         initial_lcd();
         clear_screen();//清屏
```

```
        disp_GB2312_32x32_string(0,32*0,"   深圳市晶联讯电子");
        disp_GB2312_32x32_string(0,32*1,"   JLX320160G-973");
        disp_GB2312_16x16_string(0,64*1,"自带 GB2312 国标汉字库,16X16、32X32 汉字库 JKLMNOP");
        disp_GB2312_16x16_string(0,80*1,"的 ASCII 码⑴①○●◎◇◆ 1.2.‖…ABCDEFGHIQRXYZWG");
        disp_GB2312_16x16_string(0,96*1,"鑫森淼焱晶磊众品鬟鬣麽麾縻麂麇麈麋麒麖麝磊琴斑替焱,");
        disp_GB2312_16x16_string(0,112*1,"abcdefghijklmnABCDEFGHIJKLMNOPQRSTU 款堪搭塔,");
        disp_GB2312_16x16_string(0,128*1,"麟黛黜黝黠黟黢黩黧黥黪黯鼢鼬鼯鼹鼷麝麟髓淼趁趋超越,");
        disp_GB2312_16x16_string(0,144*1,"△▽○●◆□☆♀♂※▼▲√×★■@#【】!?&±/¤§鑫淼");
        waitkey();
        clear_screen();//清屏
        disp_GB2312_32x32_string(0,32*0,"  新年作----刘长卿");
        disp_GB2312_32x32_string(0,32*1,"乡心新岁切，天畔独潸然.");
        disp_GB2312_32x32_string(0,32*2,"老至居人下，春归在客先.");
        disp_GB2312_32x32_string(0,32*3,"岭猿同旦暮，江柳共风烟.");
        disp_GB2312_32x32_string(0,32*4,"已似长沙傅，从今又几年.");
        waitkey();
        clear_screen();//清屏
        disp_384x160(bmp1);       //显示一个 320x160 点阵的图片
        waitkey();
//      clear_screen();//清屏
//      disp_384x160(bmp2);       //显示一个 320x160 点阵的图片
//      waitkey();
        clear_screen();//清屏
        disp_384x160(bmp4);       //显示一个 320x160 点阵的图片
        waitkey();
        clear_screen();//清屏
        disp_384x160(bmp3);       //显示一个 320x160 点阵的图片
        waitkey();
        test_display(0xff,0xff,0xff);
        waitkey();
        clear_screen();//清屏
        disp_4gray_384x160(bmp_4gray_2);      //显示一个 320x160 点阵的 4 灰度级的图片
        waitkey();

        clear_screen();//清屏
        disp_24x24(0,0,jing_24);  //在(0,0)位置显示一个 24x24 点阵的汉字或图片,三个参数分别是(x,y,24x24 点阵的指针)
        disp_24x24(7,0,lian_24);  //在(7,0)位置显示一个 24x24 点阵的汉字或图片,三个参数分别是(x,y,24x24 点阵的指针)
        disp_24x24(14,0,xun_24);  //在(14,0)位置显示一个 24x24 点阵的汉字或图片,三个参数分别是(x,y,24x24 点阵的指针)
        disp_16x16(40,0,jing_16);  //在(40,0)位置显示一个 16x16 点阵的汉字或图片,三个参数分别是(x,y,16x16 点阵的指针)
        disp_16x16(45,0,lian_16);  //在(45,0)位置显示一个 16x16 点阵的汉字或图片,三个参数分别是(x,y,16x16 点阵的指针)
        disp_16x16(50,0,xun_16);  //在(50,0)位置显示一个 16x16 点阵的汉字或图片,三个参数分别是(x,y,16x16 点阵的指针)
        disp_32x32(60,0,jing_32);  //在(60,0)位置显示一个 32x32 点阵的汉字或图片,三个参数分别是(x,y,32x32 点阵的指针)
        disp_32x32(70,0,lian_32);  //在(70,0)位置显示一个 32x32 点阵的汉字或图片,三个参数分别是(x,y,32x32 点阵的指针)
        disp_32x32(80,0,xun_32);  //在(80,0)位置显示一个 32x32 点阵的汉字或图片,三个参数分别是(x,y,32x32 点阵的指针)
        disp_12x12(92,0,jing_12);  //在(92,0)位置显示一个 12x12 点阵的汉字或图片,三个参数分别是(x,y,12x12 点阵的指针)
        disp_12x12(96,0,lian_12);  //在(96,0)位置显示一个 12x12 点阵的汉字或图片,三个参数分别是(x,y,12x12 点阵的指针)
        disp_12x12(100,0,xun_12);  //在(100,0)位置显示一个 12x12 点阵的汉字或图片,三个参数分别是(x,y,12x12 点阵的指针)
        disp_18x18(8,32,jing_18);
        disp_21x21(15,32,jing_21);
        disp_27x27(22,32,jing_27);
        disp_30x30(30,32,jing_30);
        disp_33x33(40,32,jing_33);  //
        disp_48x48(52,32,jing_48);
        disp_8x16(0,32,A_1);
        display_string_8x16(0,80,"ABCDEFG!@#$%^&123");
        display_string_12x24(0,96,"ABCDEFG!@#$%^&123");
        waitkey();

    }
}
```

```
/* 液晶模块型号：JLX384160G-973-PL-S，
   串行接口，
   驱动 IC 是:ST7586S(or compatible)，
   版权所有：晶联讯电子：网址  http://www.jlxlcd.cn;
*/
#include <reg51.H>
#include <intrins.h>
#include <Ctype.h>

#include <ASCII_TABLE_5X8_8X16_12x24_16x32_horizontal.h>
sbit key  =  P2^0;  //按键：我的主板上是 P2.0 口与 GND 之间接一个按键
sbit lcd_cs1=P3^4;        /*对应 LCD 的 CS 引脚*/
sbit lcd_reset=P3^5;      /*对应 LCD 的 RST 引脚*/
sbit lcd_rs=P1^1;           /*对应 LCD 的 RS 引脚*/
sbit lcd_sclk=P3^3;   //对应 LCD 的 SCK(D0)
sbit lcd_sid=P1^0;    //对应 LCD 的 SDA(D1)

sbit Rom_IN   =  P3^1;  //字库 IC 接口定义:Rom_IN 就是字库 IC 的 SI
sbit Rom_OUT  =  P3^2;  //字库 IC 接口定义:Rom_OUT 就是字库 IC 的 SO
sbit Rom_SCK  =  P3^7;  //字库 IC 接口定义:Rom_SCK 就是字库 IC 的 SCK
sbit Rom_CS   =  P3^6;  //字库 IC 接口定义 Rom_CS 就是字库 IC 的 CS#

#define uchar unsigned char
#define uint unsigned int
#define ulong unsigned long

uchar code bmp320160_1[];

//延时：1 毫秒的 i 倍
void delay(int i)
{
   int j,k;
   for(j=0;j<i;j++)
        for(k=0;k<110;k++);
}
//延时：1us 的 i 倍
void delay_us(int i)
{
   int j,k;
   for(j=0;j<i;j++)
        for(k=0;k<1;k++);
}

//等待一个按键，我的主板是用 P2.0 与 GND 之间接一个按键
void waitkey()
{
 repeat:
   if (key==1) goto repeat;
   else delay(2000);
}

////写指令到 LCD 模块
void transfer_command_lcd(int data1)
{
   char i;
   lcd_cs1=0;
   lcd_rs=0;
   for(i=0;i<8;i++)
   {
        lcd_sclk=0;
```

```
        if(data1&0x80) lcd_sid=1;
        else lcd_sid=0;
        lcd_sclk=1;
        data1=data1<<=1;
    }
    lcd_cs1=1;
}

//写数据到 LCD 模块
void transfer_data_lcd(int data1)
{
    char i;
    lcd_cs1=0;
    lcd_rs=1;
    for(i=0;i<8;i++)
    {
        lcd_sclk=0;
        if(data1&0x80) lcd_sid=1;
        else lcd_sid=0;
        lcd_sclk=1;
        data1=data1<<=1;
    }
    lcd_cs1=1;
}
```

-END-