



拓普微 TFT 液晶显示模块系列 应用手册

深圳市拓普微科技开发有限公司

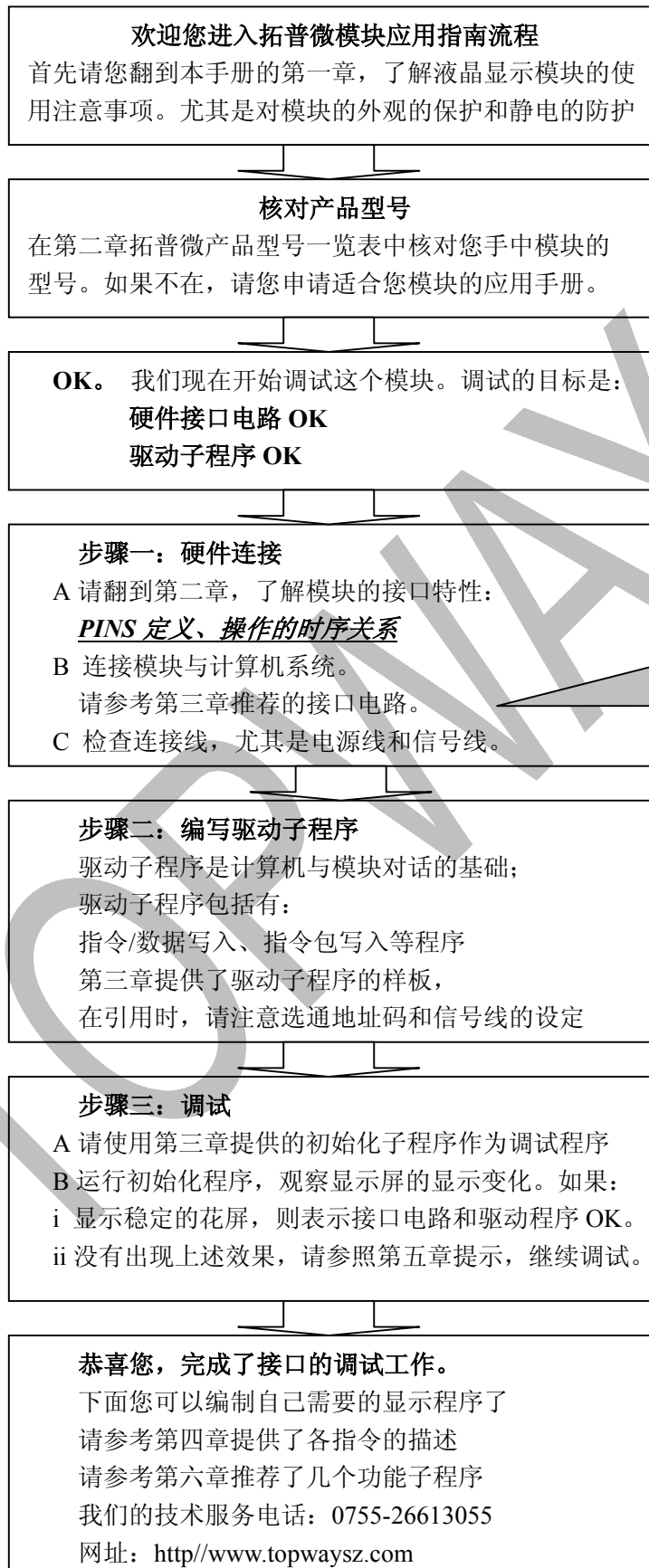
版本	描述	日期	编者
0.1	新版本	2007-07-05	郭强
0.2	更新第 6 页模块产品一览表	2007-7-12	郭强
0.3	更新第 11 页初始化程序	2007-7-26	郭强
0.4	增加串行接口操作方法	2007-9-14	郭强



目 录

应用指南.....	3
第一章 液晶显示模块使用须知.....	4
第二章 液晶显示模块产品汇总及接口特性.....	6
第三章 液晶显示模块接口技术.....	8
第四章 液晶显示模块指令系统.....	13
第五章 液晶显示模块的调试指导.....	15
第六章 液晶模块的功能子程序.....	16

应用指南



如果计算机操作时序快于或接近模块的接口时序要求，建议使用间接控制电路

第一章 液晶显示模块使用须知

液晶显示模块由易碎的玻璃盒、易划痕的偏光片和聚集集成电路的 PCB/FPC 板以及背光板等组成，模块属于精密器件，虽然拓普微在产品出厂时已经做了各项可能的保护，但在您使用之前还是希望能仔细阅读以下的注意事项，以免给您造成不必要的损失。

一、处理保护膜

在模块显示屏表面上贴有一层保护膜，以防止在调试、装配过程中沾污显示屏表面，在整机装配结束前不得揭去。

在剥离保护膜时，有时会产生静电，引起显示屏不正常的显示，这是正常的，模块在短时间上可以自行恢复。



二、加装衬垫

在整机装配时，建议在模块与前面板之间加装约 0.1 毫米厚的衬垫。面板与模块的接触面应保持平整，以免在装配后产生扭曲，并可提高其抗振性能。

三、严防静电

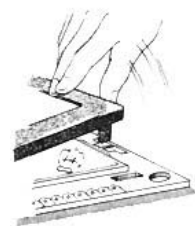
模块中的控制、驱动电路是低压、低功耗的 CMOS 电路，极易被静电击穿，静电击穿是一种不可修复的损坏，而人体有时会产生高达几十伏或上百伏的高压静电，尤其在干燥的环境中，所以，在操作、装配以及使用中都应极其小心，严防静电。为此：

1. 不要用手随意触摸外引线、电路板上的电路及金属框；
2. 如必须直接接触时，应使人体与模块保持在同一电位，或将人体良好接地；
3. 焊接使用的烙铁和操作用的电动工具必须良好接地，没有漏电；
4. 不得使用真空吸尘器进行清洁处理，因为它会产生很强的静电；
5. 空气干燥，也会产生静电，因此，工作间湿度应在 RH60% 左右；
6. 取出或放回包装袋或移动位置时，也需小心，防止产生静电。不要随意更换包装或舍弃原包装。



四、装配操作时的注意事项

1. 模块是经过精心设计组装而成的，请勿随意自行再加工、修整；
2. 金属框爪不得随意扭动、拆卸；
3. 不要随意修改加工 PCB 板外形、装配孔、线路及其部件；
4. 不得修改任何内部支架；
5. 不要碰、摔、折曲、扭动模块。
6. 安装时，尤其在使用螺丝固定 PCB 板时，不要使 PCB 板受力不均，

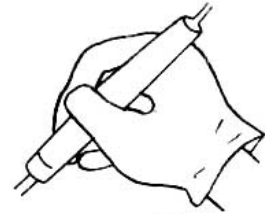


以免造成 PCB 板扭曲，拉起导电带，造成电路接触不良。

五、焊接

在焊接模块外引线、接口电路时，应按如下规程进行操作。

1. 烙铁头温度小于 320°C（无铅）280°C（有铅）；
2. 焊接时间小于 3~4s；
3. 不要使用酸性助焊剂；
4. 重复焊接不要超过 3 次，且每次重复需间隔 5 分钟。



六、模块的使用

1. 模块的外引线决不允许接错，在您想调试液晶模块时，请注意正确接线，尤其是正、负电源的接线不能有错，否则可能造成过流、过压、烧毁电路上的芯片等对液晶模块元器件有损的现象；
2. 模块在使用时，接入电源及断开电源，必须在正电源稳定接入以后，才能输入信号电平。不要在未接通电源或接入的电源电压未稳定时，在信号线上施加电压，这样有可能损坏模块中的 IC 及电路；
3. 因为液晶材料的物理特性，液晶的对比度会随着温度的变化而相应变化，所以，拓普微的产品都在接口 VOUT 端将负电源引出，提供给系统作为对比度的调节。您可做一个温度补偿电路，或者简单的安排一个电位器，然后返回到接口 V0 端；
4. 不应在规定工作温度范围以外使用，不应在存储极限温度范围外存储，如果模块的环境温度低于液晶材料的结晶温度，液晶体就会结晶，相反，如果温度过高，液晶材料将变成各向同性的液体，破坏了分子取向，这两种现象都将使模块丧失显示功能；
5. 显示屏受到轻微压力时，会产生异常显示。这时切断电源，稍待片刻，重新上电，即恢复正常；
6. 液晶显示器件或模块表面结雾时，不要通电工作，因为这将引起电极化学反应，产生断线；
7. COG 或 TAB 形式的 IC 对光比较敏感，在强光环境下，可能会造成 IC 的特性下降，甚至出现损坏。

七、模块的保养与存储

1. 只能使用异丙基酒精或乙荃酒精清洁模块，其他溶剂（比如水）都有可能损坏模块。
2. 若长期（如几年以上）存储，我们推荐以下方式：
 1. 装入聚乙烯口袋（最好有防静电涂层）并将口封住；
 2. 在 -10°C~ +35°C 之间存储；
 3. 放暗处，避强光；
 4. 决不能在表面压放任何物品；
 5. 严格避免在极限温/湿度条件下存放。



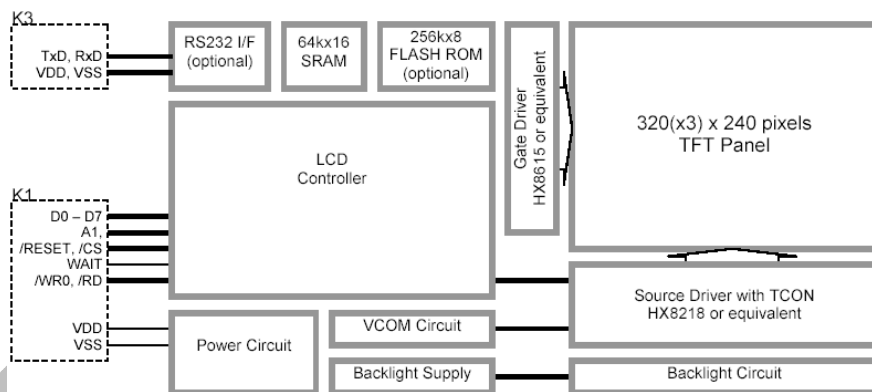
第二章 液晶显示模块产品汇总及接口特性

深圳市拓普微科技开发有限公司开发的 320*RGB*240 点阵 TFT 液晶显示模块系列。无论模块是内置 T8000 液晶显示控制器和中文字库，还是外配 TCB8000 控制板，都具有图形加速器功能，可通过简单的指令快速写字（中、西文字符），画点、线、矩形、圆，填充矩形面和圆面，以及可独立控制的游标等实用的绘图功能，支持并行接口及 RS232 串行接口，为使用者提供了简捷的与 MPU 系统接口，极适用于由单片机组成的控制系统，是自动化控制设备、仪器仪表、专业电控板等工业控制系统的彩色液晶显示模块之选。

320 x RGB x 240 点阵 TFT 液晶显示模块系列产品一览表

型号	显示尺寸	外形尺寸	控制器配置	背光色
LMT035DNAFWU	70.08*52.56	76.9*63.9*3.2	外接 TCB8000 控制板	LED-W
LMT035DNAFWU-NBN	70.08*52.56	92.7*72.0*8.8	内置 T8000 控制器	LED-W
LMT035DNAFWU-NNA**	70.08*52.56	76.9*63.9*7.7	外接 TCB8000 控制板	LED-W
LMT057DNAFWU (D)*	115.2*86.4	126.08*101.54*4.74	外接 TCB8000 控制板	LED-W
LMT057DNAFWU (D) - ANN**	115.2*86.4	155.2*109.0*10.5	外接 TCB8000 控制板	LED-W
LMT057DNAFWU (D) - AAN	115.2*86.4	155.2*109.0*13.9	内置 T8000 控制器	LED-W
LMT057DNAFWU (D) - AAA**	115.2*86.4	155.2*109.0*15.6	内置 T8000 控制器	LED-W
LMT057DNAFWU (D) - NNA**	115.2*86.4	126.08*101.54*6.44	外接 TCB8000 控制板	LED-W

* LMT057DNAFWU (D)表示 LMT057DNAFWU 和 LMT057DNAFWD; ** 产品是在 TFT 模块基础上增加触摸屏



图一、TFT 模块的电路原理图

无论模块是内置 T8000 控制器还是需要外接 TCB8000 控制板，模块与 MPU 的接口特性是一致的。见图一所示。接口包含有两种接口，一种是 I/O 并行接口 K1，适配 Intel8080 时序；一种是串行接口 K3，适配 RS232 标准协议。（注 TCB8000M 控制板还有一种总线接口）

1、并行接口

Pin No.	符号	I/O 状态	描述
1、2	VSS	Power Input	0V
3、4	VDD	Power Input	+5VDC
5	A1	Input	寄存器选择信号 A1=0 指令包入口, A1=1 状态寄存器
6	/CS	Input	片选信号, 低有效
7	/RESET	Input	复位信号, 低有效
8-15	D0-D7	I/O	数据总线, 最低位
16	/WAIT	Output	等待信号, 低电平有效
17	/RD	Input	读操作信号, 低有效
18	/WR0	Input	写操作信号, 低有效
19、20	NC	-	未用

2、串行接口

Pin No.	符号	I/O 状态	描述
1、2	Tx	Output	数据输出
3、4	Rx	Input	数据输入
5、6	GND	Power 0V	信号地
7、8	NC	--	未用
9、10	VDD	Power 5V	逻辑电源

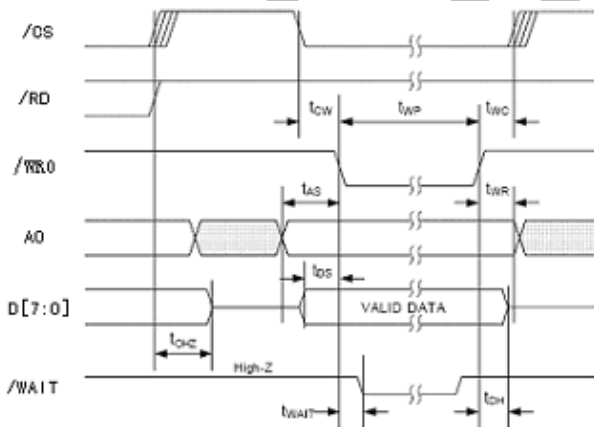
模块与 MPU 接口的时序关系适配 Intel8080 时序，其时序参数如下表：

符号	参数	MIN	MAX	单位
Tohz	输出无效到数据输出高阻态	0	8	ns
Twait	读、写有效到等待有效	0	9	ns
Tas	地址建立时间	9	-	ns
Trp、Twp	读、写脉宽	2*T	-	ns
Trc、Tcw	片选有效到读、写有效	0	-	ns
Trc、Twc	读、写结束到片选结束	0	-	ns
Twr	写恢复时间	T	-	ns
Tds	写数据到写有效建立	0	-	ns
Tdh	写数据到写无效保持	0	-	ns
Tah	地址保持时间到读无效	T	-	ns
Tac	取数时间	-	11	ns
Tolz	读到输出低阻	3	-	ns
Twds	在 WAIT 无效前无效数据	10	-	ns

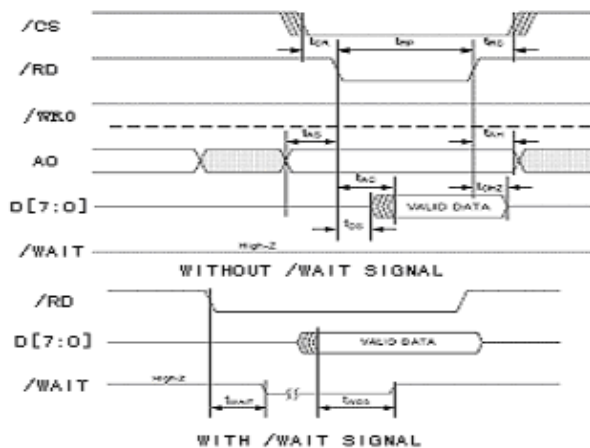
注：T 存储器时钟周期，为 24MHz 或 41.6ns

HVDD=3.0V+/-10%，LVDD=2.5V+/-10%，TOP=-40-85C

所有输入的 T_{RISE} 和 T_{FALL} 必须 $\leq 5ns(10\%-90\%)$ 所有接口的输出负载 =20pf



模块与 MPU 接口写时序关系图



模块与 MPU 接口读时序关系图

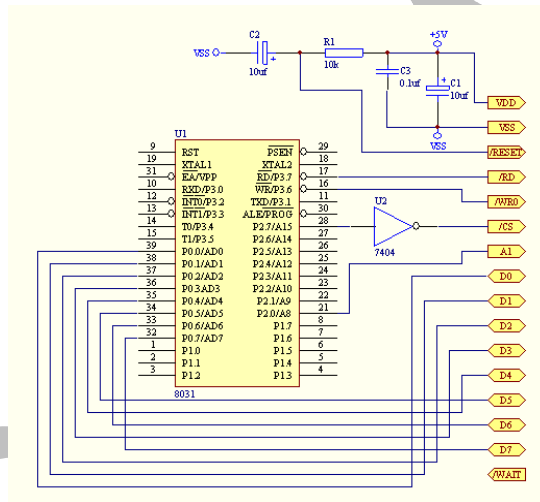
图二、模块与 MPU 接口时序关系图

第三章 液晶显示模块接口技术

TFT 液晶显示模块有两种接口提供给 MPU 使用，一种是并行接口；一种是串行接口。本章我们以单片机 8031 为控制系统，以 LMT057DNAFWU-AAN 为样本，介绍该系列模块与 MPU 接口的连接方法。时序搭配是液晶显示模块应用的基础，如果时序没有匹配好，液晶显示模块不会很好的工作，模块也是如此。一般来说，与 MPU 主机相比，液晶显示模块属于低速的外设，所以在与计算机连接时，双方的时序搭配尤为重要。这里推荐两种并行接口电路。

一、直接访问方式的接口电路及驱动程序

MPU 使用总线方式与 LMT057DNAFWU-AAN 直接连接，8031 数据口 P0 口与模块的数据口连接，由于 LMT057DNAFWU-AAN 接口操作适用于 Intel8080 时序，所以可以直接用 8031 的 /RD、/WR0 作为模块的读、写控制信号，模块的 /RESET 接 RC 复位电路。/CS 信号可由地址线译码产生，这里仅简单地通过反向器接至 A15。A1 信号由 8031 地址线 A8 提供，A8 = 0 为指令包入口地址；A8 = 1 为状态寄存器地址。由于模块内部对指令的执行速度比较快，我们可以不判断 WAIT 信号，见图三所示：



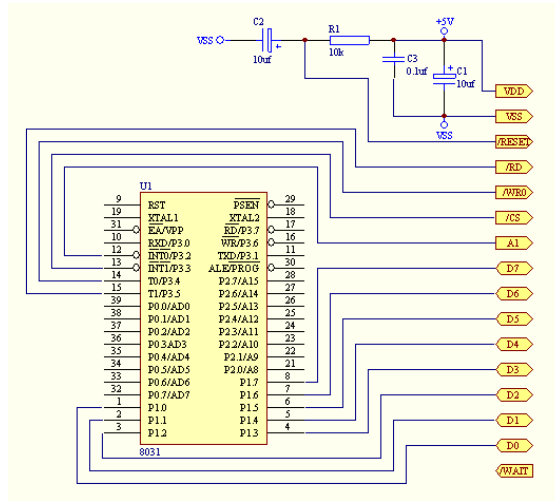
图三、直接访问方式示例电路图

直接访问方式下的驱动程序如下：

```
# define com_add XBYTE[0x8000]; //指令包入口地址设定
# define state_add XBYTE[0x8100]; //状态寄存器地址设定
// 一个指令代码或数据传送子程序
void SdCmd(uchar Command) //send a command
{
    com_add = Command;
}
// 指令包传送结束子程序
void CmdEnd() //send state bit
{
    state_add = 1;
}
// 一个指令包传送子程序
void WritePKG(uchar *pkg) // send a command package
{
    uchar i;
    for(i=*pkg;i-->0) //pkg 数组首地址存放指令包的数据量，包括指令代码
        SdCmd(*++pkg);
    CmdEnd();
}
```

二、间接控制方式接口电路及驱动程序

间接控制方式是 MPU 通过 I/O 并行接口，按照模拟模块时序的方式，通过软件编程的方式间接实现对 LMT057DNAFWU-AAN 的时序操作。该方式能够很好的回避 MPU 和模块接口之间的时序差异。根据液晶显示模块的接口信号要求，需要占用 MPU 的 12 位并行接口，在上面给出的图例上，我们将 8031 的 P1 口作为数据总线。P3 口中 4 位作为 /RD、/WR0、A1、/CS 信号。接口电路参考图四电路。



图四、间接控制方式示例电路图

间接控制方式下的驱动程序如下：

```
# define com_add P1;           //数据端口设定
sbit A1 = P3^2;              // A0 信号端口设定
sbit CS = P3^3;              // CS 信号端口设定
sbit WR0 = P3^4;             // WR 信号端口设定
sbit RD = P3^5;              // RD 信号端口设定
// 一个指令代码或数据传送子程序
void SdCmd(uchar Command)
{
    A1=0;                     // 选择指令包入口地址
    com_add = Command;        // 数据送入数据端口
    CS=0;                      // 选通模块
    WR0=0;                     // 写信号有效
    WR0=1;                     // 写信号无效
    CS=1;                      // 封闭模块
}
// 指令包传送结束子程序
void CmdEnd()
{
    A1=1;                      // 选择状态寄存器地址
    com_add = 1;               // 数据送入数据端口
    CS=0;                      // 选通模块
    WR0=0;                     // 写信号有效
    WR0=1;                     // 写信号无效
    CS=1;                      // 封闭模块
}
// 一个指令包传送子程序
void WritePKG(uchar *pkg)
{
    uchar i;
    for(i=*pkg;i--;i--)
        SdCmd(++pkg);         // pkg 数组首地址存放指令包的数据量，包括指令代码
    CmdEnd();
}
```

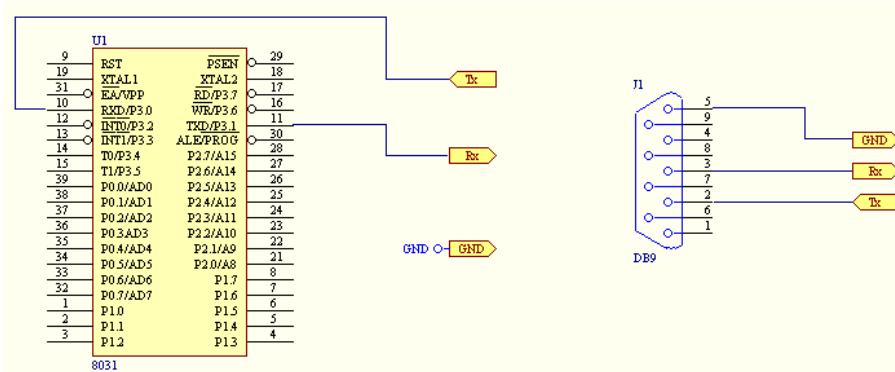
}

三、串行接口电路及通讯方式

MPU 系统还可以利用模块提供的串行接口与计算机系统实现远程通讯和最小线连接，占用计算机系统资源最小化。

模块的串行接口是标准的 RS232 接口形式，在模块内部电路专门为 RS232 接口配置了 MAX232 接口 IC，使得模块能够直接与计算机的 9 针标准串行通讯接口连接，见图五电路的右图。

模块的串行接口也可以直接与具有串行接口功能的 MPU 直接连接。见图五电路的左图，此时需要去掉模块上的 MAX232 芯片，如果使用者需要这样连接的话，请同销售工程师联系。



图五、串行接口电路示意图

模块出厂时设置为并行接口。当使用 RS232 串行接口时需要在模块后 PCB 做如下跳线处理。

接口形式	JP1	JP3	JP25
并行接口	OPEN	CLOSE	OPEN
串行接口	CLOSE	OPEN	CLOSE

模块的串行接口采用了 RS232 的通讯标准，设置为 8N1 通讯格式，即：1 位起始位，8 位数据位，无奇偶校验位以及 1 位停止位。上电/RESET 复位后，模块初始化设置 RS232 通讯波特率为 9600bps。在通讯连接成功后，计算机可以通过指令（60H）修改通讯的波特率：

指令码	操作	参数量	参数描述
60H	设置波特率	2	1: 115200bps 2: 57600bps 3: 38400bps 6: 19200bps 12: 9600 bps(上电初始值) 24: 4800bps 1047: 110bps

在串行通讯时，模块仍以指令包的形式实现的数据的传送。每个指令包起始于一个字节的“FF”，结束于字节“FE”，中间为指令代码+参数组。通讯数据格式如下：

1、串行通讯数据格式—指令包

发送序号	字节数	内容
1	1	0xFFH
2	1	指令码
3	1	参数/数据量
4	1-64	参数/数据 max64 字节
5	1	0xFEH

2、串行通讯数据格式—响应包

接收序号	字节数	内容
1	1	0x00H 或 0x08H

在计算机发送指令包后，将接收模块回复的响应包，响应包内容为模块工作的状态字。只要接收到模块回传的响应包，则表示通讯连接是正确的，连接正常运行。如果没有收到响应包，则表示通讯错误或者通讯失败，需要计算机检查通讯数据格式是否正确或者通讯连接是否还在保持。状态字 00H 表示内部电路已经设置为显示开启状态，08H 表示内部电路设置为显示关闭状态（上电初始值）。

3、串行通讯数据格式—带数据的响应包

接收序号	字节数	内容
1	1	寄存器数据
2	1	0x00H 或 0x08H

在串行通讯方式下，计算机可以使用 82H 指令（+寄存器地址）回读到寄存器的数据，模块将该数据包包含在响应包中回传给计算机。

4、串行通讯数据格式—再同步包

发送序号	字节数	内容
1	1	0xFFH
2	1	0xFFH
3	>65	0xFFH

当计算机发现与模块通讯中断或失败后，如果重新修改后发送仍没有回应的话，可以发送一组 FFH 数据，即再同步包，清除模块的接口数据缓冲器内容，重新恢复正常通讯的数据格式。

5、串行通讯数据格式—上电初始化

在刚一上电时，计算机需要对模块进行初始化设置，这些设置的步骤如下：

A、利用串行通讯电路实施对/RESET 的复位：

将计算机设置波特率为 110bps，8N1 方式，连续向模块发送 10 个 00H。

（可以通过示波器在/RESET 端检测到一组长达几十 ms 的低电平的出现）

B、进行初始化设置

按照指令包格式，传送初始化指令

（此时可以通过响应包内容判断通讯是否正确。如果没有任何回应，则表示通讯没有连接成功或者数据格式不对）

C、设置内部电路为显示开启状态

发送指令 FFH，00H，00H，FEH。

在该指令发送后，如果计算机已经设置寄存器 F08EH 数据为开显示模式，则显示屏将被点亮，如果没有设置 F08EH 寄存器，则接下来对 F08EH 的设置将点亮显示屏

（以后发送指令后的响应包内容应该为 00H）

四、初始化程序

模块开始工作之前需要初始化内部功能寄存器，执行下面的程序即可，如同计算机的显示卡的 BIOS，在程序中出现的一些寄存器并没有出现在指令描述中，这是因为初始化后，这些寄存器将不再被修改，所以为了简化使用，我们不在这里解释，如果用户想深入了解模块中的 T8000 控制器的功能，可以参考 T8000 的使用资料。

//---寄存器设置-----

```
uchar code Set_F500[]={4,0x83,0x00,0xf5,0x00};
```

```
uchar code Set_F504[]={4,0x83,0x04,0xf5,0x04};
```

```
uchar code Set_F505[]={4,0x83,0x05,0xf5,0x80};
```

```
uchar code Set_F6C4[]={4,0x83,0xc4,0xf6,0x10};
```

```
uchar code Set_F080[]={4,0x83,0x80,0xf0,0xfc};
```



```
uchar code Set_F08E[]={4,0x83,0x8e,0xf0,0x32};
uchar code Set_F090[]={4,0x83,0x90,0xf0,0x14};
uchar code Set_F091[]={4,0x83,0x91,0xf0,0x25};
uchar code Set_F092[]={4,0x83,0x92,0xf0,0x1e};
uchar code Set_F094[]={4,0x83,0x94,0xf0,0x05};
uchar code Set_F095[]={4,0x83,0x95,0xf0,0x0e};
uchar code Set_F096[]={4,0x83,0x96,0xf0,0x03};
uchar code Set_8F[]={7,0x8f,0x69,0x45,0x61,0x67,0x6c,0x65};
```

//-----初始化子程序-----

void initLCDM(void)//初始化程序

```
{
// execute all the setting in above
WritePKG(Set_F500);
WritePKG(Set_F504);
WritePKG(Set_F505);
WritePKG(Set_F6C4);
WritePKG(Set_F080);
WritePKG(Set_F08E);
WritePKG(Set_F090);
WritePKG(Set_F091);
WritePKG(Set_F092);
WritePKG(Set_F094);
WritePKG(Set_F095);
WritePKG(Set_F096);
WritePKG(Set_8F);
}
```

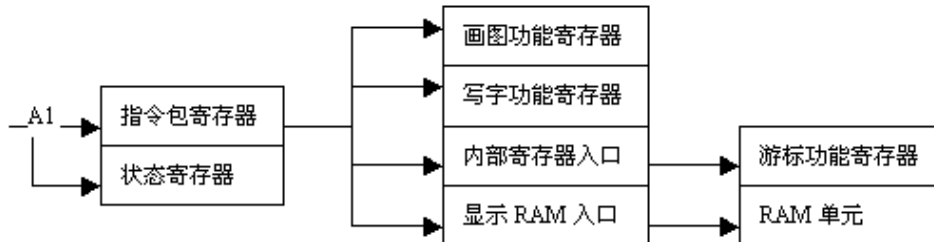
程序说明:

初始化程序将显示打开，但没有清显示 RAM 的操作，所以在调试时，我们可以把初始化子程序作为基础调试程序，在执行该程序后，正常显示效果是屏幕出现稳定的花屏。

深圳市拓普微科技开发有限公司制作

第四章 液晶显示模块指令系统

该模块在 T8000 控制器的电路上已经完成了初始工作需要的若干设置，所以指令系统以应用为主，如色彩设置，字符写入，图形绘制等。MPU 对模块的操作是通过接口指令包寄存器 (A1=0) 进行的，与指令包寄存器并行的还有一个状态寄存器 (A1=1)，其中有一位为指令包结束位，一旦被 MPU 设置为“1”后，模块将运行最新接收的指令包，实现所设置的功能。图六描述了指令的数据流图。



图六、模块的指令数据流图

1、指令包格式:

指令码 (1byte)	参数 (max64byte)	结束符
-------------	----------------	-----

计算机通过向指令包寄存器 (A1=0) 写入一个指令包数据，以实现对模块显示的图形、菜单、游标等功能进行设置。每个指令包包括一个字节的指令代码和该指令所规定的参数量。在完成指令包写入后，还需要将状态寄存器 (A1=1) 的 D0 位设置为“1”，以通知模块已完成一个指令包的传输，此时模块将启动指令译码器，完成对该指令包的功能设置。

2、指令参数排列顺序:

16 位数据 (如字符代码、坐标值、色度值、寄存器地址、显示数据等): 先低 8 位后高 8 位;

18 位数据 (如 RAM 地址): 先低 8 位再中 8 位后高 2 位;

坐标值写入顺序: 先 X 坐标 (16 位), 后 Y 坐标 (16 位)。

3、结束字: 向状态寄存器 (A1=1) 写入数据“01H”。以将状态寄存器的 D0 位设置为“1”。

计算机指令的操作指南

1、节电模式设置

模块状态寄存器 (A1=1) 有两个控制位，一个是上述已经描述的 D0 位，另一个为 D4 位，作为节电模式控制位。设置 D4 为“0”退出节电模式，开显示，进入正常运行模式，设置为“1”，进入节电模式，关显示，在节电模式下，模块将强制关闭 LCD 的驱动电源。

2、指令分类描述:

A、写字功能指令

指令码	操作	参数量	参数描述
10H	字库选择	1	00: 片内 8*8ASCII 字符 01: 片内 8*8 CGRAM (自创) 02: 片内 8*16 CGRAM (自创) 03: 片内 16*16 CGRAM (自创) 04: 外部 16*16 GB2312-80ROM
12H	显示位置	4	(X,Y)坐标值: 各 16 位
14H	字体色彩	2	5R-6G-5B (16 位)
15H	背景色彩	2	5R-6G-5B (16 位)
16H	单字写入	1-2	字符代码 (汉字代码为 16 位)
17H	字串写入	Max 64	第一参数为字符个数, 随后为字符代码
19H	自创字符	Max33	第一参数为字符代码, 随后为字模数据

B、画图功能指令

指令码	操作	参数量	参数描述
20H	图形色彩	2	5R-6G-5B (16位)
23H	画点	4	(X,Y)坐标值: 各16位
24H	画直线	8	起始(X,Y)坐标值和终止(X,Y)坐标值: 各16位
26H	画矩形线	8	起始角(X,Y)坐标值和对角(X,Y)坐标值: 各16位
27H	画矩形面	8	起始角(X,Y)坐标值和对角(X,Y)坐标值: 各16位
28H	画圆边线	5	圆心(X,Y)坐标值: 各16位, 半径(1字节)
29H	画圆面	5	圆心(X,Y)坐标值: 各16位, 半径(1字节)

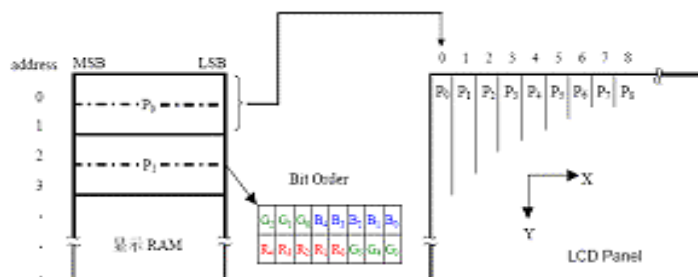
C、游标设置指令 = 指令码 (1 字节) + 寄存器地址 (2 字节) + 参数 (1 字节)

指令码	寄存器地址	单字节参数描述
83H	F100	D[7]: 游标功能 : 0 无效; 1 起用 D[6]: 游标图案透过显示: 0 无效 1 有效 (仅用于数据为 00 情况下) D[1:0]: 游标模式选择 01 2bpp, 余无效
	F102	LUT0L[7:0]设置 (对应游标图案数据 00)
	F103	LUT0H[7:0]设置
	F104	LUT1L[7:0]设置 (对应游标图案数据 01)
	F105	LUT1H[7:0]设置
	F106	LUT2L[7:0]设置 (对应游标图案数据 10)
	F107	LUT2H[7:0]设置
	F108	LUT3L[7:0]设置 (对应游标图案数据 11)
	F109	LUT3H[7:0]设置
	F10A	设置游标图案水平尺寸, 取值为 0~像素点列数-1, max 256
	F10B	设置游标图案垂直尺寸, 取值为 0~像素点行数-1, max 256
	F10C	设置游标图案显示起始位置 X 坐标值 (水平像素点列数) 低 8 位
	F10D	高 2 位
	F10E	设置游标图案显示起始位置 Y 坐标值 (垂直像素行数) 低 8 位
	F10F	高 1 位
F142	设置游标图案存储起始地址 A17-0, 低 8 位 A7-0	
F143	中 8 位 A15-8	
F144	高 2 位 A17-16	

E、显示 RAM 操作指令

指令码	操作	参数量	参数描述
81H	设置地址	3	A0-17 地址码
84H	写入数据	Max64	第一参数为数据个数, 随后为数据

该模块内置显示存储器为 184Kbyte。地址范围从 00000H 到 2DFFFH。显示的起始地址为 00000H。MPU 可以通过 81H 和 84H 组合向显示 RAM 写入数据。该模块显示色彩为 64K 色, 每个像素数据为 16 位“5R6G5B”形式, 数据存储从 00000H 单元开始, 每 2 个单元与屏上的一个显示像素 RGB 对应, 如 00H-01H 单元对应为显示屏 home 位置的像素, 以此类推, 偶数地址为一个像素数据的低 8 位, 紧接的奇数地址为高 8 位, 如图七表示:



第五章 液晶显示模块的调试指导

调试的工作主要是经验和实践的结合，我们要具备冷静、细心的态度。首先我们排除接口连线错误的因素，因为这需要读者自己的检查。内置中文字库液晶显示模块接口采用的是 Intel8080 时序，与我国较为常用的单片机时序相匹配，所以在硬件电路接口连接，时序的搭配实现比较容易，我们就我们的经验提示如下：

一、硬件方面

连接线接触是否实在

因为作为调试使用，系统板可能还没有专门为模块设计接口，或者直接连到仿真器上测试，这时可能会使用临时制作的连线。所以请检查连接线接头处的接触是否实在，包括检查焊接点是否虚焊，连接点是否隔离很好等。建议在模块的 PCB 板上接口使用插座。

模块的复位脚/RST 的连接

一定要接一个复位电路的。当产品使用环境比较好时，可以直接采用我们在管脚定义里提供的 RC 复位电路，但当产品使用环境比较恶劣时，也可以将/RST 接到 MPU 的口线上，这样不仅可以用品线复位，还可以进行定时刷新，预防一些其它干扰。

/RST 信号复位的原则是：

- i) 上电后应该比 MPU 提前复位，同时在程序上电执行液晶模块初始化程序之前先运行一段延时程序。
- ii) 上电后，/RST 保持低电平至少 5 个晶振时钟周期，低电平拉高后至少保持 2ms 再对模块进行软件初始化。

时序的搭配

如果选用的 MPU 为高速器件，或者它的总线读写周期小于 1us，建议您还是采用间接方式为稳妥。

二 软件问题

- 1 地址选择的正确性，无论是直接方式还是间接方式，地址的选择包含了 A1、/CS 信号的选择要正确。
- 2 主要寄存器设置的正确性。

由于模块上电时需要复位时间，一般在几百 MS，所以在调试时，需要在给模块上电后等待一会儿再运行程序，如果直接在 MPU 板上烧入程序，上电运行，那么，在启动时请多加延时程序。

在间接控制方法下，程序完成了时序关系建立，要记住的是/RD、/WR 信号的变化要独立进行，不要与其它信号端（如 RS、/CS 等）的设置共用，如果同时设置这几个信号，将会出现读写错误，而且不容易检查到。

第六章 液晶模块的功能子程序

模块指令的特征是以显示画面的像素坐标为显示位置数据，节省了使用者计算 RAM 地址以及考虑与显示画面的对应关系。尤其提供有画点、画线、画圆、填充图形等功能指令，大大简化了软件的编写和节省了系统运行时间。功能程序如下：

一、色彩设置程序

1、前景色设置子程序（适用于图形和字符写入）

```
void SetFgColor(uint color) //前景色设置程序
{
    uchar Buffer[4];
    Buffer[0]=3;
    Buffer[1]=0x20; // 或者 0x14 前景色设置指令代码
    Buffer[2]=color; // 色彩数据低 8 位
    Buffer[3]=color>>8; // 色彩数据高 8 位
    WritePKG(Buffer);
}
```

2、背景色设置子程序（仅用于字符）

```
void SetFontBgColor(uint color) //背景色设置程序
{
    uchar Buffer[4];
    Buffer[0]=3;
    Buffer[1]=0x15; //背景色设置指令代码
    Buffer[2]=color; //色彩数据低 8 位
    Buffer[3]=color>>8; //色彩数据高 8 位
    WritePKG(Buffer);
}
```

二、字符写入程序

3、ASCII 字符串写入子程序（最多一次写入 63 个字符）

```
void PrintASCII(uint X, Y, uchar *pstr)
{
    uchar Buffer[6], NoOfChar;
    Buffer [0]=2; // 指令包数据量设置
    Buffer [1]=0x10; // 字库选择指令代码
    Buffer [2]=0x00; // 字库选择，使用内部 8x8 ASCII 字库
    WritePKG(TempData);
    Buffer[0]=5; // 指令包数据量设置
    Buffer[1]=0x12; // 显示位置设置指令代码
    Buffer[2]=X; // X 坐标低 8 位
    Buffer[3]=X>>8; // X 坐标高 8 位
    Buffer[4]=Y; // Y 坐标低 8 位
    Buffer[5]=Y>>8; // Y 坐标高 8 位
    WritePKG(Buffer);
    NoOfChar=strlen(pstr); // 计算数组长度
    SdCmd(0x17); // 字符串写入指令代码
    SdCmd(NoOfChar); // 字符数量
    while(*pstr>0)
    {
        SdCmd(*pstr++); //写入字符代码
    }
    CmdEnd();
}
```

4、汉字字符串写入子程序（最多一次写入 31 个汉字）

```
void PrintGB(uint X, Y, uchar *pstr)
{
    uchar TempData[3], Buffer[6], NoOfChar;
```



```
TempData[0]=2;           // 指令包数据量设置
TempData[1]=0x10;        // 字库选择指令代码
TempData[2]=0x04;        // 字库选择, 使用模块内部 16*16GB2132 汉字库
WritePKG(TempData);
Buffer[0]=5;             // 指令包数据量设置
Buffer[1]=0x12;          // 显示位置设置指令代码
Buffer[2]=X;             // X 坐标低 8 位
Buffer[3]=X>>8;          // X 坐标高 8 位
Buffer[4]=Y;             // Y 坐标低 8 位
Buffer[5]=Y>>8;          // Y 坐标高 8 位
WritePKG(Buffer);
NoOfChar=strlen(pstr);   // 计算数组长度
SdCmd(0x17);             // 字符串写入指令代码
SdCmd(NoOfChar/2);       // 字符数量
while(*pstr>0)
{
    SdCmd(*pstr++);       // 写入字符代码
}
CmdEnd();
}
```

三、制图程序

5、画点子程序

```
void Draw_Dot(uint X, Y)
{
    uchar Buffer[6];
    Buffer[0]=5;           // 指令包数据量设置
    Buffer[1]=0x23;        // 画点指令代码
    Buffer[2]=X;           // 点坐标 X 低 8 位
    Buffer[3]=X>>8;        // 点坐标 X 高 8 位
    Buffer[4]=Y;           // 点坐标 Y 低 8 位
    Buffer[5]=Y>>8;        // 点坐标 Y 高 8 位
    WritePKG(Buffer);
}
```

6、画线子程序

```
void Draw_Line(uint x1, y1, x2, y2)
{
    uchar Buffer[10];
    Buffer[0]=9;           // 指令包数据量设置
    Buffer[1]=0x24;        // 画线指令代码
    Buffer[2]=x1;           // 起始坐标 X 低 8 位
    Buffer[3]=x1>>8;        // 起始坐标 X 高 8 位
    Buffer[4]=y1;           // 起始坐标 Y 低 8 位
    Buffer[5]=y1>>8;        // 起始坐标 Y 高 8 位
    Buffer[6]=x2;           // 结束坐标 X 低 8 位
    Buffer[7]=x2>>8;        // 结束坐标 X 高 8 位
    Buffer[8]=y2;           // 结束坐标 Y 低 8 位
    Buffer[9]=y2>>8;        // 结束坐标 Y 高 8 位
    WritePKG(Buffer);
    delays(5);
}
```

7、画矩形边子程序

```
void Draw_Rect(uint x1, y1, x2, y2)
{
    uchar Buffer[10];
    Buffer[0]=9;           // 指令包数据量设置
    Buffer[1]=0x26;        // 画矩形边线指令代码
    Buffer[2]=x1;           // 起始角坐标 X 低 8 位
    Buffer[3]=x1>>8;        // 起始角坐标 X 高 8 位
    Buffer[4]=y1;           // 起始角坐标 Y 低 8 位
```



```
Buffer[5]=y1>>8; // 起始角坐标 Y 高 8 位
Buffer[6]=x2; // 对角坐标 X 低 8 位
Buffer[7]=x2>>8; // 对角坐标 X 高 8 位
Buffer[8]=y2; // 对角坐标 Y 低 8 位
Buffer[9]=y2>>8; // 对角坐标 Y 高 8 位
WritePKG(Buffer);
delayms(20);
}
```

8、画圆边子程序

```
void Draw_Circle(uint X, Y, uchar R)
```

```
{
    uchar Buffer[7];
    Buffer[0]=6; // 指令包数据量设置
    Buffer[1]=0x28; // 画圆指令代码
    Buffer[2]=X; // 圆心坐标 X 低 8 位
    Buffer[3]=X>>8; // 圆心坐标 X 高 8 位
    Buffer[4]=Y; // 圆心坐标 Y 低 8 位
    Buffer[5]=Y>>8; // 圆心坐标 Y 高 8 位
    Buffer[6]=R; // 圆半径 R
    WritePKG(Buffer);
    delayms(20);
}
```

9、画矩形面子程序

```
void Fill_Rect(uint x1, y1, x2, y2)
```

```
{
    uchar Buffer[10];
    Buffer[0]=9; // 指令包数据量设置
    Buffer[1]=0x27; // 画矩形面指令代码
    Buffer[2]=x1; // 起始角坐标 X 低 8 位
    Buffer[3]=x1>>8; // 起始角坐标 X 高 8 位
    Buffer[4]=y1; // 起始角坐标 Y 低 8 位
    Buffer[5]=y1>>8; // 起始角坐标 Y 高 8 位
    Buffer[6]=x2; // 对角坐标 X 低 8 位
    Buffer[7]=x2>>8; // 对角坐标 X 高 8 位
    Buffer[8]=y2; // 对角坐标 Y 低 8 位
    Buffer[9]=y2>>8; // 对角坐标 Y 高 8 位
    WritePKG(Buffer);
    delayms(20);
}
```

10、画圆面子程序

```
void Fill_Circle(uint X, Y, uchar R)
```

```
{
    uchar Buffer[7];
    Buffer[0]=6; // 指令包数据量设置
    Buffer[1]=0x29; // 画圆面指令代码
    Buffer[2]=X; // 圆心坐标 X 低 8 位
    Buffer[3]=X>>8; // 圆心坐标 X 高 8 位
    Buffer[4]=Y; // 圆心坐标 Y 低 8 位
    Buffer[5]=Y>>8; // 圆心坐标 Y 高 8 位
    Buffer[6]=R; // 圆半径 R
    WritePKG(Buffer);
    delayms(20);
}
```

11、图画写入子程序

```
void ShowBMP(uint x,y,width,high,uint bmp[])
```

//width: 图形水平像素点数; high: 图形垂直像素行数; bmp[]: 图形数组名

```
{
    uchar Buffer[5];
    uint i,j,k;
    ulong p;
    ulong addr; // 设置显示 RAM 指针
}
```



```

addr=y*640+2*x;          // RAM 单元地址计算公式 addr= X*2 + Y*320*2
p=0;                    // 图形数据寻址指针
for(i=0;i<high;i++)    // 图形行数据写入循环
{
  Buffer[0]=4;
  Buffer[1]=0x81;        // 图形行写入首地址设置
  Buffer[2]=addr;
  Buffer[3]=addr>>8;
  Buffer[4]=addr>>16;
  WritePKG(Buffer);
  j=width;
  while(j>=16)          // 以下为图形行数据写入
  {
    SdCmd(0x84);        // 显示 RAM 写入指令
    SdCmd(32);          // 固定一次写入 32 字节图形数据
    for(k=0;k<16;k++)  // 即 16 个像素数据
    {
      SdCmd bmp[p];     // 写入像素数据低 8 位
      SdCmd bmp[p]>>8); // 写入像素数据高 8 位
      p++;
    }
    CmdEnd();
    j=j-16;
  }
  if (j>0)              //16 倍数余下数据写入
  {
    SdCmd(0x84);
    SdCmd(2*j);         // 设置写入数据字节数
    for(k=0;k<j;k++)   // 即 J 个像素数据
    {
      SdCmd bmp[p];     // 写入像素数据低 8 位
      SdCmd bmp[p]>>8); // 写入像素数据高 8 位
      p++;
    }
    CmdEnd();
  }
  addr=addr+640;       // 下一行单元首地址修正
}

```

12、游标建立子程序

```

void SET_SPRITE()
{
  uchar j,k;
  uchar Buffer[5];
  //下面是一个箭头（16*16 点阵）的游标图案数据，仅作为示例
  uchar code pic03[]={ 0xff, 0xff, 0xff, 0xf0, 0xea, 0xaa, 0xaa, 0xc0,      //1-2 行
    0xea, 0xaa, 0xab, 0x00, 0xea, 0xaa, 0xac, 0x00,      //3-4 行
    0xea, 0xaa, 0xb0, 0x00, 0xea, 0xaa, 0xb0, 0x00,      //5-6 行
    0xea, 0xaa, 0xac, 0x00, 0xea, 0xaa, 0xab, 0x00,      //7-8 行
    0xea, 0xaa, 0xaa, 0xc0, 0xea, 0xfa, 0xaa, 0xb0,      //9-10 行
    0xeb, 0x0e, 0xaa, 0xac, 0xec, 0x03, 0xaa, 0xab,      //11-12 行
    0xf0, 0x00, 0xea, 0xab, 0xc0, 0x00, 0x3a, 0xac,      //13-14 行
    0x00, 0x00, 0x0e, 0xb0, 0x00, 0x00, 0x03, 0xc0 }; //15-16 行
  uint WHITE=0xffff, BLACK=0x0000, BLUE=0x001f, GREEN=0x07e0, RED=0xf800;
  Buffer[0]=4;
  Buffer[1]=0x83;
  Buffer[2]=0x00;
  Buffer[3]=0xf1;
  Buffer[4]=0x01          //激活游标寄存器 D[1:0]=01 2bpp
  WritePKG(Buffer);
  Buffer[0]=4;
  Buffer[1]=0x83;
  Buffer[2]=0x02;

```



```
Buffer[3]=0xf1;
Buffer[4]=BLUE; //设置光标数据 00 的显示色彩（低 8 位）
WritePKG(Buffer);
Buffer[0]=4; //LUT 00 high
Buffer[1]=0x83;
Buffer[2]=0x03;
Buffer[3]=0xf1;
Buffer[4]=BLUE>>8; //设置光标数据 00 的显示色彩（高 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x04;
Buffer[3]=0xf1;
Buffer[4]=GREEN; //设置光标数据 01 的显示色彩（低 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x05;
Buffer[3]=0xf1;
Buffer[4]=GREEN>>8; //设置光标数据 01 的显示色彩（高 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x06;
Buffer[3]=0xf1;
Buffer[4]=WHITE; //设置光标数据 10 的显示色彩（低 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x07;
Buffer[3]=0xf1;
Buffer[4]=WHITE>>8; //设置光标数据 10 的显示色彩（高 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x08;
Buffer[3]=0xf1;
Buffer[4]=BLACK; //设置光标数据 11 的显示色彩（低 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x09;
Buffer[3]=0xf1;
Buffer[4]=BLACK>>8; //设置光标数据 11 的显示色彩（高 8 位）
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x0a;
Buffer[3]=0xf1;
Buffer[4]=0x0f; // 设置光标图案水平像素点数，示例为 16*16 点阵
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x0b;
Buffer[3]=0xf1;
Buffer[4]=0x0f; // 设置光标图案水蛭像素点数，示例为 16*16 点阵
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x42;
Buffer[3]=0xf1;
Buffer[4]=0x00; //设置光标图案存储单元起始地址（低 8 位） 示例为 2b000H
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
```



```

Buffer[2]=0x43;
Buffer[3]=0xf1;
Buffer[4]=0xb0; //设置游标图案存储单元起始地址（中 8 位） 示例为 2b000H
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x83;
Buffer[2]=0x44;
Buffer[3]=0xf1;
Buffer[4]=0x02; //设置游标图案存储单元起始地址（高 2 位） 示例为 2b000H
WritePKG(Buffer);
Buffer[0]=4;
Buffer[1]=0x81; //写游标图案存储单元起始地址 示例为 2b000H
Buffer[2]=0x00;
Buffer[3]=0xb0;
Buffer[4]=0x02;
WritePKG(Buffer);
for(j=0;j<2;j++) //写入游标图案数据给 2b000H 后单元
{
    SdCmd(0x84); //写入显示 RAM 指令码
    SdCmd(32); //写入数据量
    for(k=0;k<32;k++) //写入数据
    {
        SdCmd(pic03[k+j*32]);
    }
    CmdEnd();
}

```

13、游标显示子程序

```

void use_sprite(uint x,y)
{
    uchar Buffer[5];
    Buffer[0]=4;
    Buffer[1]=0x83;
    Buffer[2]=0x00;
    Buffer[3]=0xf1; // 启用游标设置
    Buffer[4]=0xc1; //D7 enable;D6 transparency edable; D[1:0]=01 2bpp
    WritePKG(Buffer);
    Buffer[0]=4;
    Buffer[1]=0x83;
    Buffer[2]=0x0c;
    Buffer[3]=0xf1;
    Buffer[4]=x; // 游标显示坐标 X 值低 8 位
    WritePKG(Buffer);
    Buffer[0]=4;
    Buffer[1]=0x83;
    Buffer[2]=0x0d;
    Buffer[3]=0xf1;
    Buffer[4]=x>>8; // 游标显示坐标 X 值高 8 位
    WritePKG(Buffer);
    Buffer[0]=4;
    Buffer[1]=0x83;
    Buffer[2]=0x0e;
    Buffer[3]=0xf1;
    Buffer[4]=y; // 游标显示坐标 Y 值低 8 位
    WritePKG(Buffer);
    Buffer[0]=4;
    Buffer[1]=0x83;
    Buffer[2]=0x0f;
    Buffer[3]=0xf1;
    Buffer[4]=y>>8; // 游标显示坐标 Y 值高 8 位
    WritePKG(Buffer);
}

```

深圳市拓普微科技开发有限公司制作