# 彩色 TFTLCD 显示模块

## 产品说明书

*V2.0 – 2008.09*

# 1 产品简介

## 1.1 主要功能与基本参数

是一块高画质的 TFT 真彩 LCD 模块，具有丰富多样的接口、编程方便、易于扩展等良好性能。 内置专用驱动和控制 IC（SPFD5408），并且驱动 IC 自己集成显示缓存，无需外部显示缓存。

彩色 TFT LCD 显示模块的基本参数如下表：

| 项目 | 规格 | 单位 | 注备 |
|---|---|---|---|
| 显示点阵数 | 240×RGB×320 | Dots | |
| LCD 尺寸 | 2.4（对角线） | 英寸 | |
| LCM 外形尺寸 | 42.72×59.46×3.0 | mm | 不包括转接 PCB 以及 FPC |
| 动态显示区 | 36.72×48.96 | mm | |
| 象素尺寸 | 0.147×0.147 | mm | |
| 象素成份 | a-Si TFT | | |
| LCD 模式 | 65K TFT | | |
| 总线 | 8 位 Intel 80 总线 | | |
| 背光 | 白色 LED | | |
| 驱动 IC | SPFD5408 | | |

## 1.2 模块结构

模块实际上就是将 的 TFT-LCD 显示器连接在 PCB 电路板上，并加在 PCB 电路板上加入背光限流电阻，将显示器不便于与开发板连接的软 PCB 连接接口引出，并以 DIP 的双排插针（板上保留有 FPC20/1.0 间距的 FPC 座）引出模块以便于用户连接。

为了方便用户的扩展使用， 模块将显示器主供电源和显示器背光电源分开供电，这样，如果用户有需要的话，可以单独给背光提供合适的电源以获取合适的背光亮度。一般情况下，可以将显示器主供电源和背光电源都接上 3V电压，但需要注意，在使用模组时，这两个电源是都要接的。下面介绍模块的接口，如 图 1.1 所示：

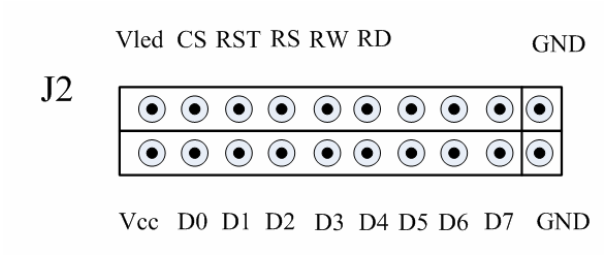




图 1.1 模块接口示意图

表 1.1 模块接口引脚说明

| 接口引脚 | 说 明 |
|---|---|
| Vcc | TFT-LCD 显示板电源（推荐 3.0V） |
| Vled | 背光灯电源 |
| D0~D7 | 8 位数据总线 |
| CS | 片选（低电平有效） |
| RST | Reset 复位（低电平复位） |
| RS | 控制寄存器/数据寄存器选择（低电平选择控制寄存器） |
| RW | 写信号（低电平有效） |
| RD | 读信号（低电平有效） |
| GND | 接地 |

## 1.3 系统环境

模块的极限电气特性如下表所示：

表 1.2 TFT LCD 面板极限电压范围

| 参数 | 符号 | 最小 | 最大 | 单位 | 备注 |
|---|---|---|---|---|---|
| 输入电源电压 | Vcc | -0.3 | 3.5 | V | 逻辑电压 |
| | Vio | -0.3 | 3.5 | V | |

**注意**：上表中的极限电压范围并非是模块的工作电压，而是指模块可以承受的电压范围；如果用户在接入电压时，超过了极限电压范围，则模块内部的 LCD 控制逻辑电路会受到破坏，并使模块的性能受到严重的影响；所以强烈推荐用户在使用模块时，使用正常的工作电压：3.0V。

表 1.3 极限环境参数

| 参数 | 符号 | 最小 | 最大 | 单位 | 备注 |
|---|---|---|---|---|---|
| 操作温度范围 | Top | -20 | 70 | ℃ | 周围环境 |
| 储存温度范围 | Tst | -30 | 80 | ℃ | 周围环境 |

**注意**：（1）禁止在含有腐蚀性气体的环境中保存、操作；

（2）TFT LCD 会根据具体的环境情况（主要是温度）的变化而在显示时存在细微的变化，这样的变化是可逆的。

而模块的工作电气参数，如表 1.4 所示：

表 1.4 TFT-LCD 模块操作电压（25℃）

| 项目 | 符号 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|
| 整机逻辑供电压 | Vdd-Vss | 2.80 | 3.0 | 3.20 | V |
| 输入高电平电压 | $V_{IH}$ | 0.8 $V_{dd}$ | - | $V_{dd}$ | V |
| 输入低电平电压 | $V_{IL}$ | Vss | - | 0.2$V_{dd}$ | V |
| 输出高电平电压 | $V_{OH}$ | 0.8$V_{dd}$ | - | $V_{dd}$ | V |
| 输出低电平电压 | $V_{OL}$ | Vss | - | 0.2$V_{dd}$ | V |
| 整机逻辑耗电流 | $I_{DD}$ | 2.5 | 3.0 | 3.5 | mA |

表 1.5 背光单元工作电气参数

| 项目 | 符号 | 额定值 | 单位 |
|---|---|---|---|
| 工作电流 | If | 60 | mA |
| 工作电压 | Vf | 3.3V | V |
| 功耗 | Po | 180 | mW |
| 工作温度 | Top | -20 to 70 | ℃ |
| 储存温度 | Tst | -30 to 80 | ℃ |

**注意**：背光 LED 为并联。

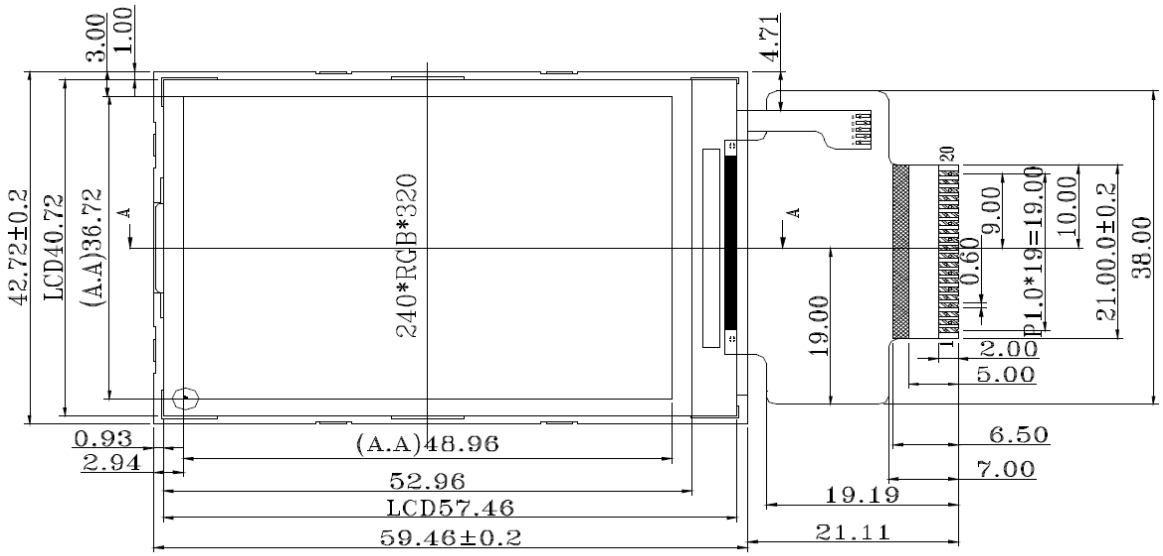
# 2 液晶显示器介绍

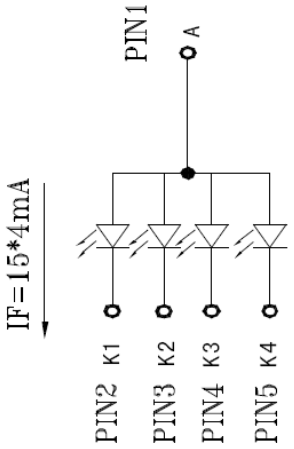
彩色 TFTLCD 显示模块的 LCD 驱动控制 IC 为 SPFD5408，用户在对 模块进行操作时，实际上是对 SPFD5408 进行相关的控制寄存器、显示数学据存储器进行操作的，所以，接下来重点对 的驱动控制特性进行详细的介绍。

## 2.1 液晶显示器

### 2.1.1 液晶显示器结构

显示器内嵌TFT-LCD驱动控制芯片，采用先进的COG技术，将芯片嵌在LCD玻璃上，图 2.1 为此液晶显示器的尺寸图；而 图 2.2 为液晶显示器的实物图。



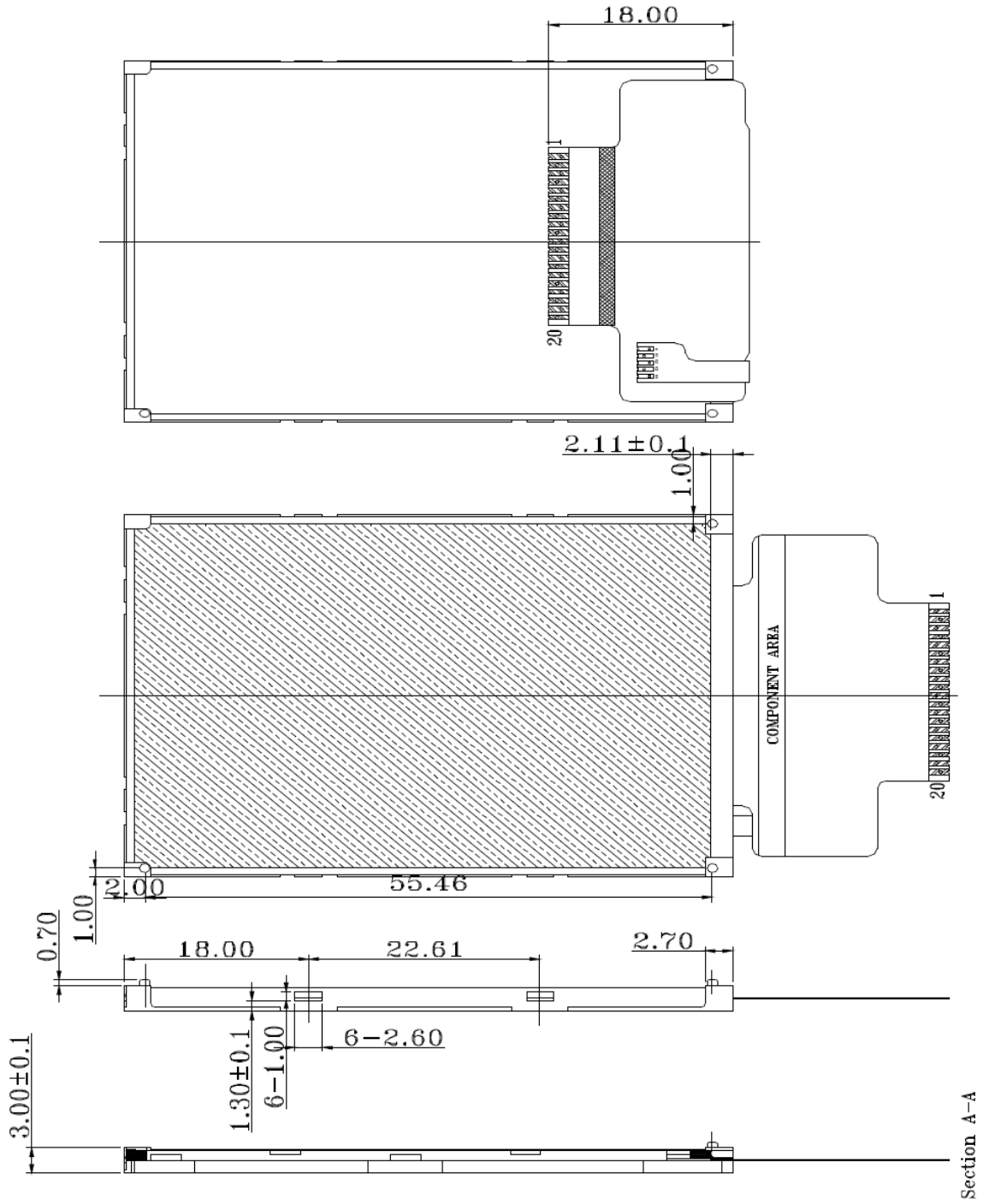



图 2.1 液晶显示器尺寸


图 2.2 液晶显示器实物图

图 2.3 显示 JPEG 图像效果图

### 2.1.2 显示 RAM 区映射情况

模块的2.4英寸TFT-LCD显示面板上，共分布着240×320个像素点，而模块内部的TFT-LCD驱动控制芯片内置有与这些像素点对应的显示数据RAM（简称显存）。模块中每个像素点需要16位的数据（即2字节长度）来表示该点的RGB颜色信息，所以模块内置的显存共有240 × 320 × 16bit的空间，通常我们以字节（byte）来描述其的大小。

模块的显示操作非常简便，需要改变某一个像素点的颜色时，只需要对该点所对应的2个字节的显存进行操作即可。而为了便于索引操作，模块将所有的显存地址分为X轴地址（X Address）和Y轴地址（Y Address），分别可以寻址的范围为X Address=0~239，Y Address = 0~319，X Address和Y Address交叉对应着一个显存单元（2byte）；这样只要索引到了某一个X、Y轴地址时，并对该地址的寄存器进行操作，便可对TFT-LCD显示器上对应的像素点进行操作了。

提示：以上的描述意味着，当我们对某一个地址上的显示进行操作时，需要对该地址进行连续两次的 8 位数据写入或读出的操作，方可完成对一个显存单元的操作。

模块的像素点与显存对应关系如 图 2.4 所示：

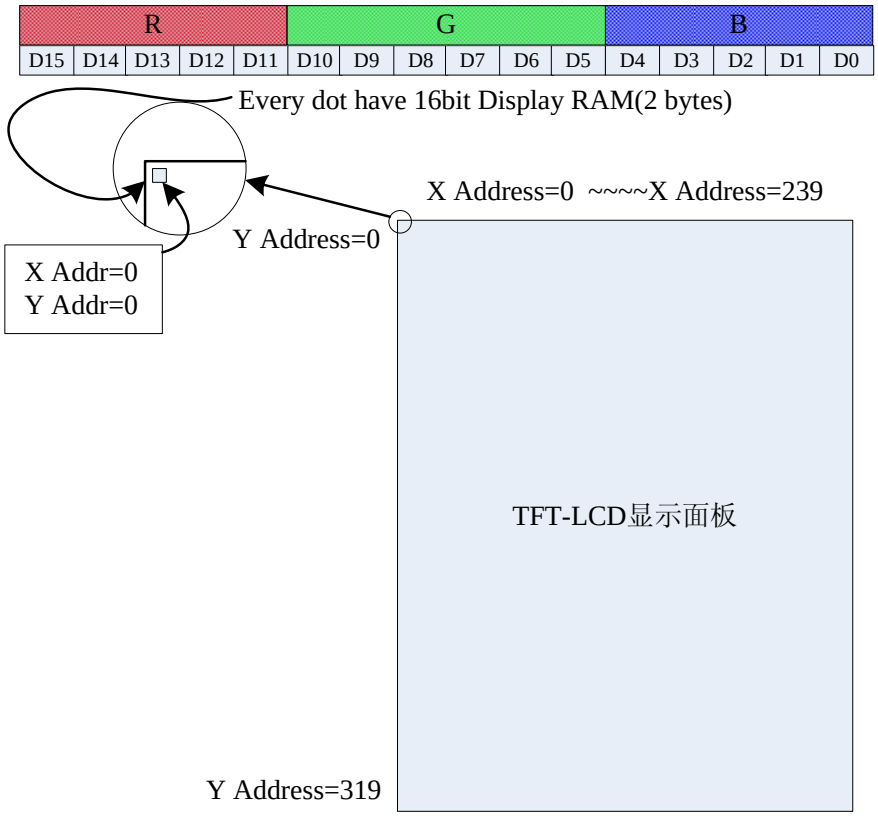



图 2.4 显存与像素点对应关系示意图

模块内部有一个显存地址累加器 AC，即用于在读写显存时对显存地址进行自动的累加，这在连续对屏幕显示数据操作时非常有用，特别是应用在图形显示、视频显示时。此外，AC 累加器可以设置为各种方向的累加方式，如通常情况下为对 X Address 累加方式，具体为当累加到一行的尽头时，会切换到下一行的开始累加；还可以为对 Y Address 累加方式，具体为当累加到一列（垂直方向）的尽头时，会切换到下一个 X Address 所对应的列开始累加，详细介绍请参见 2.3.2。

另外， 模块还提供了窗口操作的功能，可以对显示屏上的某一个矩形区域进行连续操作，详细介绍也请参见 2.3.3。

## 2.2 操作时序

模块支持标准 intel8080 总线，总线的最高速度可达 8MHz，也就是说，如果控制 MCU 速度足够快的话，是可以支持视频的显示的。图 2.4 为 模块的总线时序图：

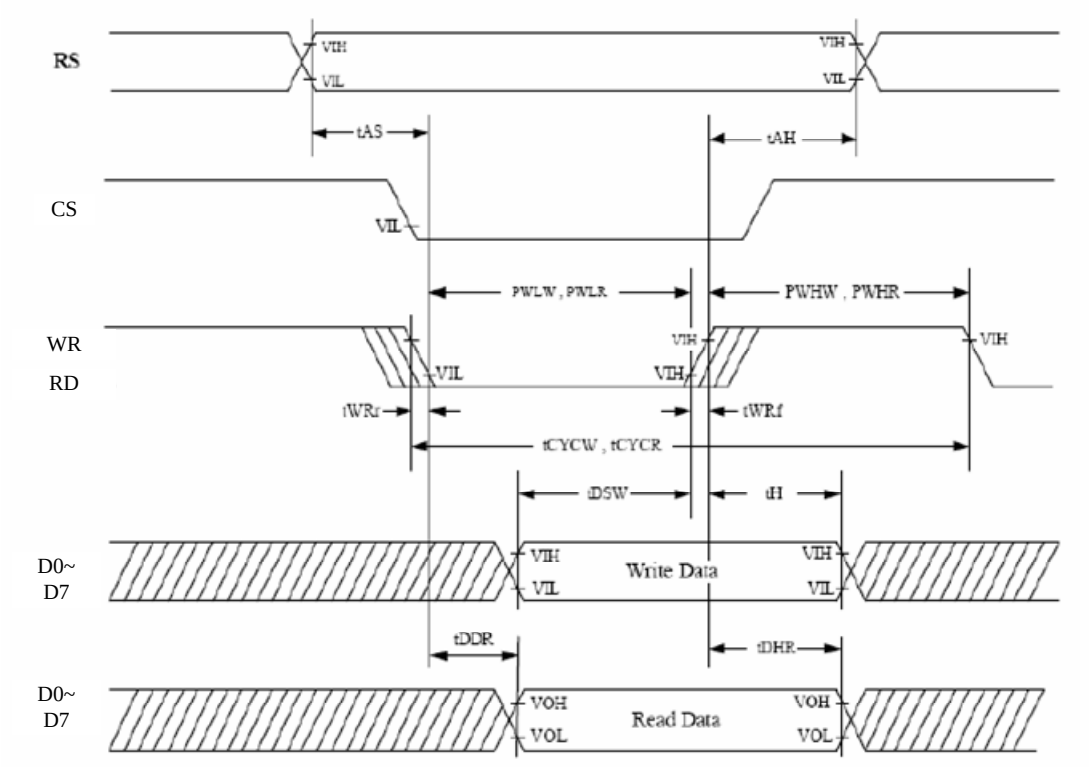



图 2.5 MzT24-1 模块总线时序图

**注意：** 模块的总线接口是 8 位的，也就意味着对显存的某一个地址操作时，需要连续进行两次操作方可完成，先传高字节再传低字节。

表 2.1 时序特性（IOVcc=2.4V～3.3V ， TA=25℃）

| 参数 | | 符号 | 单位 | 最小值 | 典型 | 最大值 |
|---|---|---|---|---|---|---|
| 总线周期时间 | 写周期 | $t_{CYCW}$ | ns | 125 | -- | -- |
| | 读周期 | $t_{CYCR}$ | ns | 450 | -- | -- |
| 控制低脉冲宽度（RW） | | $PW_{LW}$ | ns | 45 | -- | -- |
| 控制低脉冲宽度(RD) | | $PW_{LR}$ | ns | 170 | -- | -- |
| 控制高脉冲宽度(RW) | | $PW_{HW}$ | ns | 70 | -- | -- |
| 控制高脉冲宽度(RD) | | $PW_{HR}$ | ns | 250 | -- | -- |
| 读/写控制信号上升/下降时间 | | $t_{WRr}$， $t_{WRr}$ | ns | -- | -- | 25 |
| 建立时间 | 写（RS to CS,RW） | $t_{AS}$ | ns | 0 | -- | -- |
| | 读（RS to CS,RD） | | ns | 10 | -- | -- |
| 地址保持时间 | | $t_{AH}$ | ns | 2 | -- | -- |
| 写数据建立时间 | | $t_{DSW}$ | ns | 25 | -- | -- |
| 写数据保持时间 | | $t_{H}$ | ns | 10 | -- | -- |
| 读数据延迟时间 | | $t_{DDR}$ | ns | -- | -- | 200 |

| 参数 | 符号 | 单位 | 最小值 | 典型 | 最大值 |
|---|---|---|---|---|---|
| 读数据保持时间 | $t_{DDR}$ | ns | 5 | -- | -- |

复位时序的示意图如下图所示：

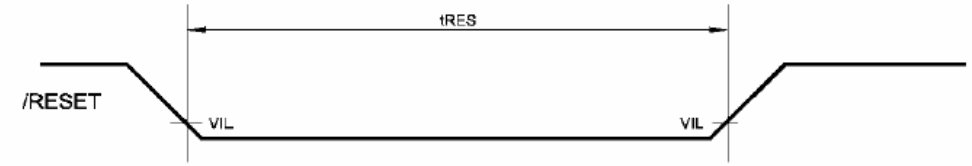



表 2.2 复位时序特性

| 参数 | 符号 | 单位 | 最小值 | 典型 | 最大值 |
|---|---|---|---|---|---|
| 复位低电平宽度 | $t_{RES}$ | ms | 1 | -- | -- |
| 复位上升沿时间 | $t_{rRES}$ | us | -- | -- | 10 |

## 2.3 控制方法及 LCD 显示特性

### 2.3.1 模块控制方法

对 模块的操作主要分为两种，一是对控制寄存器的读写操作，二是对显存的读写操作；而这两种操作实际上都是通过对 LCD 控制器（SPFD5408）的寄存器（register）进行操作完成的，SPFD5408 提供了一个索引寄存器（Index register），对该 Index register 寄存器的写入操作可以指定操作的寄存器索引，以便于完成控制寄存器、显存操作寄存器的读写操作。 提供了 RS（有些资料称 A0）控制线，并以此线的高低电平状态来区别这对 Index register 操作还是对所指向的寄存器进行操作：当 RS 为低电平时，表示当前的总线操作是对 Index register 进行操作，即指明接下去的寄存器操作是针对哪一个寄存器的；当 RS 为高电平时，表示为对寄存器操作。

模块内部有控制寄存器，用户在使用 之前以及对其进行操作过程当中，需要对一些寄存器进行写操作以完成对 LCD 的初始化，或者是完成某些功能的设置（如当前显存操作地址设置等）。

对控制寄存器进行操作前，需要先对索引寄存器（Index register）进行定入操作，以指明接下去的寄存器读写操作是针对哪一个寄存器的。操作的步骤如下：

1、 在 RS 为低电平的状态下，写入两个字节的数据，第一个字节为零，第二字节为寄存器索引值。

2、 然后在 RS 为主电平的状态下，写入两个字节数据，第一字节为高八位，第二字节为低八位；如要读出指定寄存器的数据，则需要连续三次读操作方能完成一次读出操作，第一个字节为无效数据，第二字节为高八位，第三字节为低八位。

注意： 的显存操作也是通过寄存器操作来完成的，即对 0x22 寄存器进行操作时，就是对当前位置点的显示进行读写操作。

模块的控制寄存器当中，最常被调用的是寄存器除了对显存操作的 0x22 寄存器外，还有当前显存地址的寄存器display RAM bus address counter (AC)，一共由两个的寄存器组成，分别存放有X Address和Y Adderss，表示当前对显存数据的读写操作是针对于该地址所指向的显存单元；而每一个显存单元在前面已经用图示意过，每个单元有 16 位，最高的 5 位为R（红）的分量，最低的 5 位为B（蓝）的

分量，中间 6 位为G（绿）分量。如 图 2.6 所示：


图 2.6 显存单元示意图

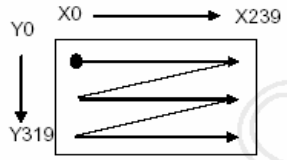
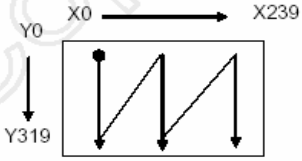
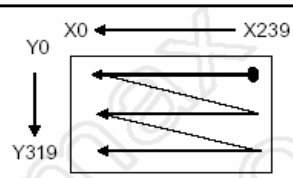
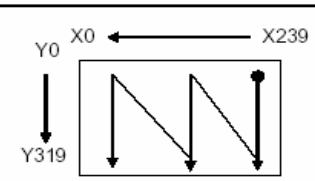
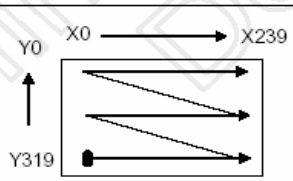
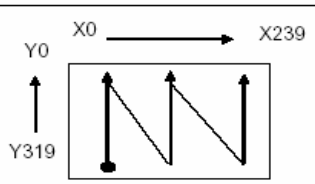
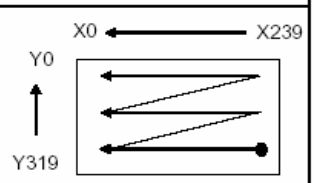
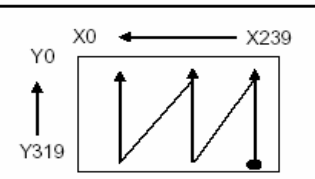
所以，当需要对 LCD 显示面板上某一个点（X，Y）进行操作时，需要先设置 AC，以指向需要操作的点所对应的显存地址，然后连续写入或者读出数据，才完成对该点的显存单元的数据操作。

而当对某一个显存单元完成写入数据操作后，AC 会自动的进行调整，或者是不进行调整（根据控制寄存器中的设置而决定）保持原来指向。AC 的这个特性对于　　模块来说非常有用，可以根据此特性设计出快速的 LCD 显示操作功能函数，以适应不同用户的需求。

### 2.3.2 显存地址指针

内部含有一个用于对显存单元地址自动索引的显存地址指针 display RAM bus address counter (AC)；AC 会根据当前用户操作的显存单元，在用户完成一次显存单元的写操作后进行调整，以指向下一个显存单元；可以通过对相关的寄存器当中的控制位的设置，来选择合适的 AC 调整特性。这些用于设置 AC 调整特性（实际上也就是显存操作地址的自动调整特性）的位分别是：AM（bit3 of R03h）、ID0（bit4 of R03h）、ID1（bit5 of R03h）；下面将说明这些控制位的特性。

而配合 AM 位的设置，可以得到多种 AC 调整方式，以适应不同用户的不同需要。可以通过下面的列表了解具体的设置对应的特性：

| AM | | | Description Figure | AM | | | Description Figure |
|---|---|---|---|---|---|---|---|
| 0 | | |  | 1 | | |  |
| | | |  | | | |  |
| | | |  | | | |  |
| | | |  | | | |  |

### 2.3.3 显存的窗口工作模式

除了一般的对全屏的工作模式外，还提供了一种局部的窗口工作模式，这样可以简化对局部显示区域的读写操作；窗口工作模式允许用户对显存操作时仅仅是对所设置的局部显示区域对应的显存进行读写操作。设置 ORG（bit7 of R03）位为 1 时，可以启动窗口工作模式，这时再对显存进行读写操作的话，AC 将只会在所设置的局部显示区域（简称窗口）进行调整；而设置的局部区域可以通过设置 R50 来确定最小的 X Address，设置 R51 来确定窗口的最大 X Address，设置 R52 来确定窗口的最小 Y Address，设置 R53 来确定窗口的最大 Y Address。

而当启动窗口工作模式后，需要确认对显存操作时，地址范围为：

“00”h ≦ MIN X address ≦ X address ≦ MAX X address ≦ “EF”h
“00”h ≦ MIN Y address ≦ Y address ≦ MAX Y address ≦ “13F”h

而前面所述的显存地址指针 AC 的调整特性，在窗口工作模式当中也是有效的，也就是说在一般显存操作模式（全屏范围显存）设置的 AC 调整特性，在工作在窗口模式时，也是有效的。下图为当 AM=0、ID0 和 ID1 都设置为 1 时的示意图：

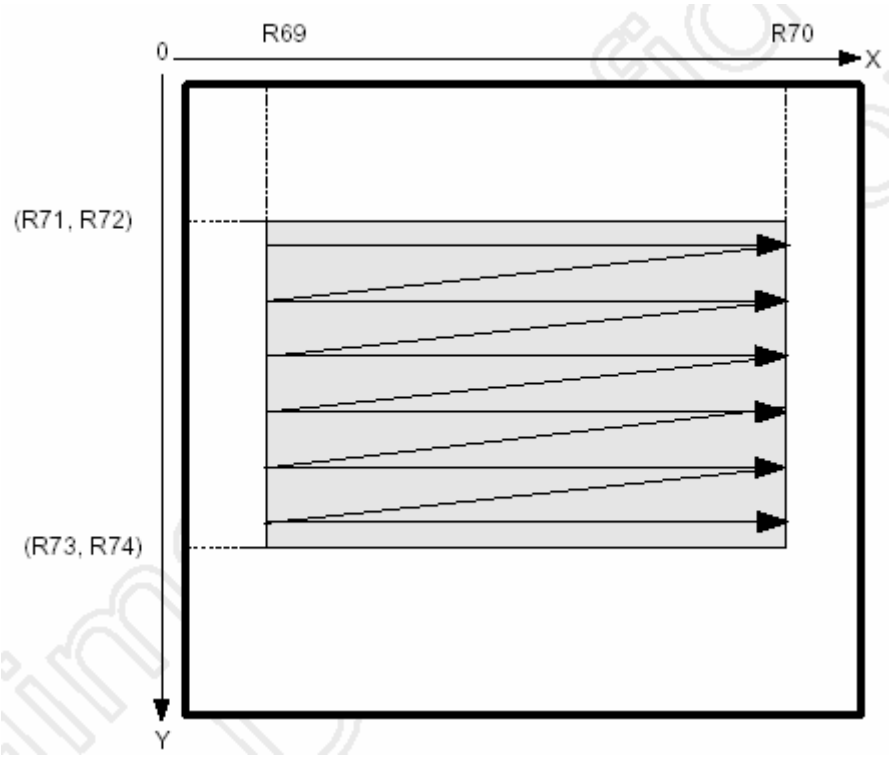



图 2.7 窗口工作模式

### 2.3.4 控制寄存器

为了广大用户快速地掌握　　模块的使用，这里列出部分常用的　　模块内部控制寄存器，以便快速的掌握　　的驱动控制方法。

表 2.3 常用控制寄存器说明

| Category. | Register No | Register | Upper 8-bit | | | | | | | | Lower 8-bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
| | 00h | ID Read | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 1 |
| | 01h | Driver Output Control | 0 | 0 | 0 | 0 | 0 | SM (0) | 0 | SS (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 02h | LCD AC Drive Control | 0 | 0 | 0 | 0 | 0 | 1 | B/C (0) | EOR (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | NW0 (0) |
| | 03h | Entry Mode | TRIREG (0) | DFM (0) | 0 | BGR (0) | 0 | 0 | HWM (0) | 0 | ORG (0) | 0 | I/D1 (1) | I/D0 (1) | AM (0) | 0 | 0 | 0 |
| | 04h | Resizing Control | 0 | 0 | 0 | 0 | 0 | 0 | RCV1 (0) | RCV0 (0) | 0 | 0 | RCH1 (0) | RCH0 (0) | 0 | 0 | RSZ1 (0) | RSZ0 (0) |
| | 07h | Display control (1) | 0 | 0 | PTDE1 (0) | PTDE0 (0) | 0 | 0 | 0 | BASEE (0) | 0 | VON (0) | GON (0) | DTE (0) | COL (0) | 0 | D1 (0) | D0 (0) |
| | 08h | Display control (2) | 0 | 0 | 0 | 0 | FP3 (1) | FP2 (0) | FP1 (0) | FP0 (0) | 0 | 0 | 0 | 0 | BP3 (1) | BP2 (0) | BP1 (0) | BP0 (0) |
| | 09h | Display control (3) | 0 | 0 | 0 | 0 | 0 | PTS2 (0) | PTS1 (0) | PTS0 (0) | 0 | 0 | PTG1 (0) | PTG0 (0) | ISC3 (0) | ISC2 (0) | ISC1 (0) | ISC0 (0) |
| | 0Ah | Display control (4) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FMARK OE (0) | FMI2 (0) | FMI1 (0) | FMI0 (0) |
| | 0Ch | External interface control (1) | 0 | ENC2 (0) | ENC1 (0) | ENC0 (0) | 0 | 0 | 0 | RM (0) | 0 | 0 | DM1 (0) | DM0 (0) | 0 | 0 | RIM1 (0) | RIM0 (0) |
| | 0Dh | Frame Maker Position | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FMP8 (0) | FMP7 (0) | FMP6 (0) | FMP5 (0) | FMP4 (0) | FMP3 (0) | FMP2 (0) | FMP1 (0) | FMP0 (0) |
| | 0Fh | External interface control (2) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VSPL (0) | HSPL (0) | 0 | EPL (0) | DPL (0) |
| | 10h | Power Control (1) | 0 | 0 | 0 | SAP (0) | BT3 (0) | BT2 (0) | BT1 (0) | BT0 (0) | APE (0) | 0 | AP1 (0) | AP0 (0) | 0 | DSTB (0) | SLP (0) | 0 |
| | 11h | Power Control (2) | 0 | 0 | 0 | 0 | 0 | DC12 (0) | DC11 (0) | DC10 (0) | 0 | DC02 (0) | DC01 (0) | DC00 (0) | 0 | VC2 (0) | VC1 (0) | VC0 (0) |
| | 12h | Power Control (3) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PON (0) | VRH3 (0) | VRH2 (0) | VRH1 (0) | VRH0 (0) |
| | 13h | Power Control (4) | 0 | 0 | 0 | VDV4 (0) | VDV3 (0) | VDV2 (0) | VDV1 (0) | VDV0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 17h | Power Control (5) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PSE (0) |
| | 20h | GRAM address Set Horizontal Address | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD7 (0) | AD6 (0) | AD5 (0) | AD4 (0) | AD3 (0) | AD2 (0) | AD1 (0) | AD0 (0) |
| | 21h | GRAM address Set Vertical Address | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD16 (0) | AD15 (0) | AD14 (0) | AD13 (0) | AD12 (0) | AD11 (0) | AD10 (0) | AD9 (0) | AD8 (0) |
| | 22h | GRAM data | | | | | | | | | | | | | | | | |
| | 28h | NVM read data (1) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | UID3 (0) | UID2 (0) | UID1 (0) | UID0 (0) |
| | 29h | NVM read data (2) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VCM14 (0) | VCM13 (0) | VCM12 (0) | VCM11 (0) | VCM10 (0) |
| | 2Ah | NVM read data (3) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VCMSEL(0) | 0 | 0 | VCM24 (0) | VCM23 (0) | VCM22 (0) | VCM21 (0) | VCM20 (0) |
| | 30h | $\gamma$ Control (1) | 0 | 0 | 0 | 0 | 0 | P0KP12 (0) | P0KP11 (0) | P0KP10 (0) | 0 | 0 | 0 | 0 | 0 | P0KP02 (0) | P0KP01 (0) | P0KP00 (0) |
| | 31h | $\gamma$ Control (2) | 0 | 0 | 0 | 0 | 0 | P0KP32 (0) | P0KP31 (0) | P0KP30 (0) | 0 | 0 | 0 | 0 | 0 | P0KP22 (0) | P0KP21 (0) | P0KP20 (0) |
| | 32h | $\gamma$ Control (3) | 0 | 0 | 0 | 0 | 0 | P0KP52 (0) | P0KP51 (0) | P0KP50 (0) | 0 | 0 | 0 | 0 | 0 | P0KP42 (0) | P0KP41 (0) | P0KP40 (0) |
| | 33h | $\gamma$ Control (4) | 0 | 0 | 0 | 0 | 0 | 0 | P0FP11 (0) | P0FP10 (0) | 0 | 0 | 0 | 0 | 0 | 0 | P0FP01 (0) | P0FP00 (0) |
| | 34h | $\gamma$ Control (5) | 0 | 0 | 0 | 0 | 0 | 0 | P0FP31 (0) | P0FP30 (0) | 0 | 0 | 0 | 0 | 0 | 0 | P0FP21 (0) | P0FP20 (0) |
| | 35h | $\gamma$ Control (6) | 0 | 0 | 0 | 0 | 0 | P0RP12 (0) | P0RP11 (0) | P0RP10 (0) | 0 | 0 | 0 | 0 | 0 | P0RP02 (0) | P0RP01 (0) | P0RP00 (0) |
| | 36h | $\gamma$ Control (7) | 0 | 0 | 0 | V0RP14 (0) | V0RP13 (0) | V0RP12 (0) | V0RP11 (0) | V0RP10 (0) | 0 | 0 | 0 | V0RP04 (0) | V0RP03 (0) | V0RP02 (0) | V0RP01 (0) | V0RP00 (0) |
| | 37h | $\gamma$ Control (8) | 0 | 0 | 0 | 0 | 0 | P0KN12 (0) | P0KN11 (0) | P0KN10 (0) | 0 | 0 | 0 | 0 | 0 | P0KN02 (0) | P0KN01 (0) | P0KN00 (0) |
| | 38h | $\gamma$ Control (9) | 0 | 0 | 0 | 0 | 0 | P0KN32 (0) | P0KN31 (0) | P0KN30 (0) | 0 | 0 | 0 | 0 | 0 | P0KN22 (0) | P0KN21 (0) | P0KN20 (0) |
| | 39h | $\gamma$ Control (10) | 0 | 0 | 0 | 0 | 0 | P0KN52 (0) | P0KN51 (0) | P0KN50 (0) | 0 | 0 | 0 | 0 | 0 | P0KN42 (0) | P0KN41 (0) | P0KN40 (0) |
| | 3Ah | $\gamma$ Control (11) | 0 | 0 | 0 | 0 | 0 | 0 | P0FN11 (0) | P0FN10 (0) | 0 | 0 | 0 | 0 | 0 | 0 | P0FN01 (0) | P0FN00 (0) |
| | 3Bh | $\gamma$ Control (12) | 0 | 0 | 0 | 0 | 0 | 0 | P0FN31 (0) | P0FN30 (0) | 0 | 0 | 0 | 0 | 0 | 0 | P0FN21 (0) | P0FN20 (0) |
| | 3Ch | $\gamma$ Control (13) | 0 | 0 | 0 | 0 | 0 | P0RN12 (0) | P0RN11 (0) | P0RN10 (0) | 0 | 0 | 0 | 0 | 0 | P0RN02 (0) | P0RN01 (0) | P0RN00 (0) |
| | 3Dh | $\gamma$ Control (14) | 0 | 0 | 0 | V0RN14 (0) | V0RN13 (0) | V0RN12 (0) | V0RN11 (0) | V0RN10 (0) | 0 | 0 | 0 | V0RN04 (0) | V0RN03 (0) | V0RN02 (0) | V0RN01 (0) | V0RN00 (0) |
| | 50h | Window Horzontal RAM address start | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HSA7 (0) | HSA6 (0) | HSA5 (0) | HSA4 (0) | HSA3 (0) | HSA2 (0) | HSA1 (0) | HSA0 (0) |
| | 51h | Window Horzontal RAM address start | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HEA7 (1) | HEA6 (1) | HEA5 (1) | HEA4 (0) | HEA3 (1) | HEA2 (1) | HEA1 (1) | HEA0 (1) |
| | 52h | Window Vertical RAM address start | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VSA8 (0) | VSA7 (0) | VSA6 (0) | VSA5 (0) | VSA4 (0) | VSA3 (0) | VSA2 (0) | VSA1 (0) | VSA0 (0) |
| | 53h | Window Vertical RAM address start | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VEA8 (1) | VEA7 (0) | VEA6 (0) | VEA5 (1) | VEA4 (1) | VEA3 (1) | VEA2 (1) | VEA1 (1) | VEA0 (1) |
| | 60h | Driver Output Control | GS (0) | 0 | NL5 (0) | NL4 (0) | NL3 (0) | NL2 (0) | NL1 (0) | NL0 (0) | 0 | 0 | SCN5 (0) | SCN4 (0) | SCN3 (0) | SCN2 (0) | SCN1 (0) | SCN0 (0) |
| | 61h | Image Display Control | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | NDL (0) | VLE (0) | REV (0) |

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 6Ah | Vertical Scoll ing Control | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VL8 (0) | VL7 (0) | VL6 (0) | VL5 (0) | VL4 (0) | VL3 (0) | VL2 (0) | VL1 (0) | VL0 (0) |
| | 80h | Display Position 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTDP 08 (0) | PTDP 07 (0) | PTDP 06 (0) | PTDP 05 (0) | PTDP 04 (0) | PTDP 03 (0) | PTDP 02 (0) | PTDP 01 (0) | PTDP 00 (0) |
| | 81h | GRAM start line address 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTSA 08 (0) | PTSA 07 (0) | PTSA 06 (0) | PTSA 05 (0) | PTSA 04 (0) | PTSA 03 (0) | PTSA 02 (0) | PTSA 01 (0) | PTSA 00 (0) |
| | 82h | GRAM end line address 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTEA 08 (0) | PTEA 07 (0) | PTEA 06 (0) | PTEA 05 (0) | PTEA 04 (0) | PTEA 03 (0) | PTEA 02 (0) | PTEA 01 (0) | PTEA 00 (0) |
| | 83h | Display Position 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTDP 18 (0) | PTDP 17 (0) | PTDP 16 (0) | PTDP 15 (0) | PTDP 14 (0) | PTDP 13 (0) | PTDP 12 (0) | PTDP 11 (0) | PTDP 10 (0) |
| | 84h | GRAM start line address 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTSA 18 (0) | PTSA 17 (0) | PTSA 16 (0) | PTSA 15 (0) | PTSA 14 (0) | PTSA 13 (0) | PTSA 12 (0) | PTSA 11 (0) | PTSA 10 (0) |
| | 85h | GRAM end line address 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTEA 18 (0) | PTEA 17 (0) | PTEA 16 (0) | PTEA 15 (0) | PTEA 14 (0) | PTEA 13 (0) | PTEA 12 (0) | PTEA 11 (0) | PTEA 10 (0) |
| | 90h | Panel interface Control 1 | 0 | 0 | 0 | 0 | 0 | 0 | DIVI1 (0) | DIVI0 (0) | 0 | 0 | 0 | RTNI4 (1) | RTNI3 (0) | RTNI2 (0) | RTNI1 (0) | RTNI0 (0) |
| | 92h | Panel Interface Control 2 | 0 | 0 | 0 | 0 | 0 | NOWI 2(0) | NOW1 1(0) | NOW1 0(0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 93h | Panel Interface Control 3 | 0 | 0 | 0 | 0 | 0 | 0 | VEQW 11(0) | VEQW 10(0) | 0 | 0 | 0 | 0 | 0 | MCPI2 (0) | MCPI1 (0) | MCPI0 (0) |
| | 95h | Panel Interface Control 4 | 0 | 0 | 0 | 0 | 0 | 0 | DIVE1 (0) | DIVE0 (0) | 0 | 0 | RTNE5 (0) | RTNE4 (1) | RTNE3 (1) | RTNE2 (1) | RTNE1 (1) | RTNE0 (0) |
| | 97h | Panel Interface Control 5 | 0 | 0 | 0 | 0 | NOW E3(0) | NOW E2(0) | NOW E1(0) | NOW E0(0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 98h | Panel Interface Control 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MCPE2 (0) | MCPE1 (0) | MCPE0 (0) |
| | A0h | NVM Conmtrol1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | TE (0) | 0 | EOP1 (0) | EOP0 (0) | 0 | 0 | EAD1 (0) | EAD0( 0) |
| | A1h | NVM Control 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ED7 (0) | 0 | 0 | ED4 (0) | ED3 (0) | ED2 (0) | ED1 (0) | ED0 (0) |
| | A4h | Calibration control | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CALB (0) |

### 2.3.5 常用寄存器设置说明

1）、索引寄存器 Index Register(IR)

| R/W | RS | CB15 | CCB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 0 | □ | □ | □ | □ | □ | □ | □ | □ | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |

当 RS 为低电平时，对写入数据即为对 IR 操作，即指定接下来的寄存器操作是针对哪一个寄存器；该设置共需设置 16 位，高八位为无用数据，低八位为指定的寄存器地址（ID0~ID7）。

2）、ID 读取寄存器 ID Read Register(SR)

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

当 RS 为零时，读取操作即可读取控制芯片的 ID 号。

3）、系统模式设置寄存器 Entry Mode(R03h)

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | TRIREG | DFM | 0 | BGR | 0 | 0 | HWM | 0 | ORG | 0 | I/D1 | I/D0 | AM | 0 | 0 | 0 |

AM、ID0 和 ID1 的设置在前面已有说明，这里不再重述。

ORG：窗口操作模式设置

ORG=1 时，进入窗口操作模式，即设置 R20 和 R21 寄存器时，需指定显存的操作地址在窗口范围之内（窗口范围由 R50~R53 设定）。

ORG=0，则显存操作范围为全屏点对应的显存地址，无论 R50~R53 设置的窗口范围如何。

HWM：显存高速操作模式

设置为 1 时生效。

BGR：RGB 三原色基数对应显存数据关系设置（以下说明以 65K 色为例）

该位可以设置 RGB 三原色在显存数据中的数据对应关系，当 BGR=0 时，显存数据当中三原色分量的分布情况如下图：


即为 RGB565 的格式。

如 BGR=1，则在上图的基础上，R 与 B 分量对调，即变成 BGR565 的格式。

注意：当 BGR=0 时，写入数据为 RGB565 格式，而读出的数据为 BGR 的，即写入与读出是不对应的；而当 BGR=1 时，写入和读出的数据是对应的。

DFM：与 TRIREG 位配合设置数据传输模式，详见后面有关数据传输模式的描述。

TRIREG：数据传输次数设置，即设置每一次显存数据传输时的模式。

（8 位总线时）

TRIREG=0　2 次传输完成 16 位显存数据传输；

TRIREG=1　3 次传输完成都 18 位显存数据传输；

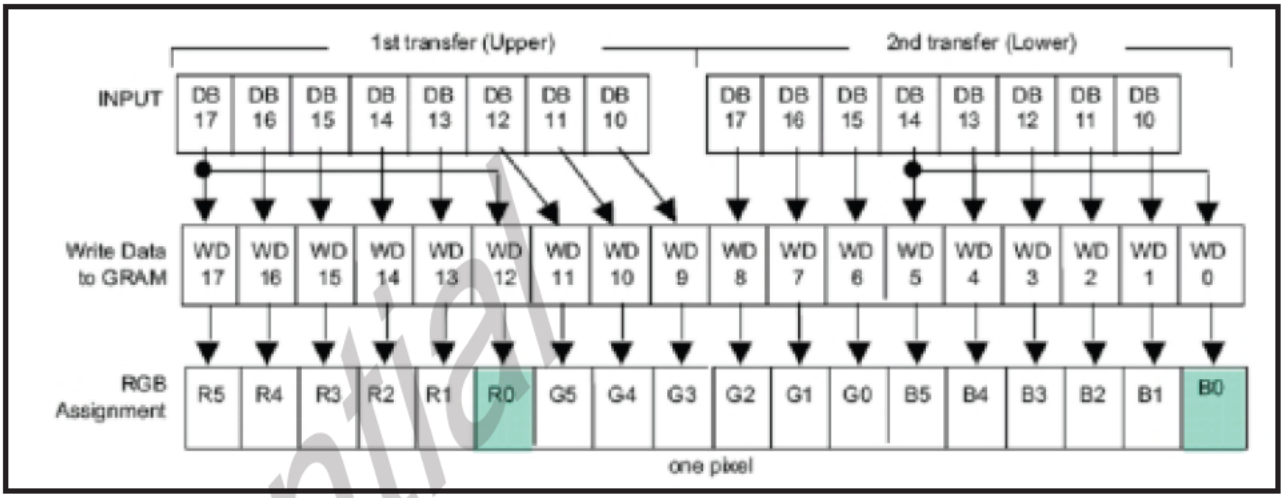



图 2.8 8 位总线，2 次传输模式，65K 色(TRIREG=0,DFM=0)

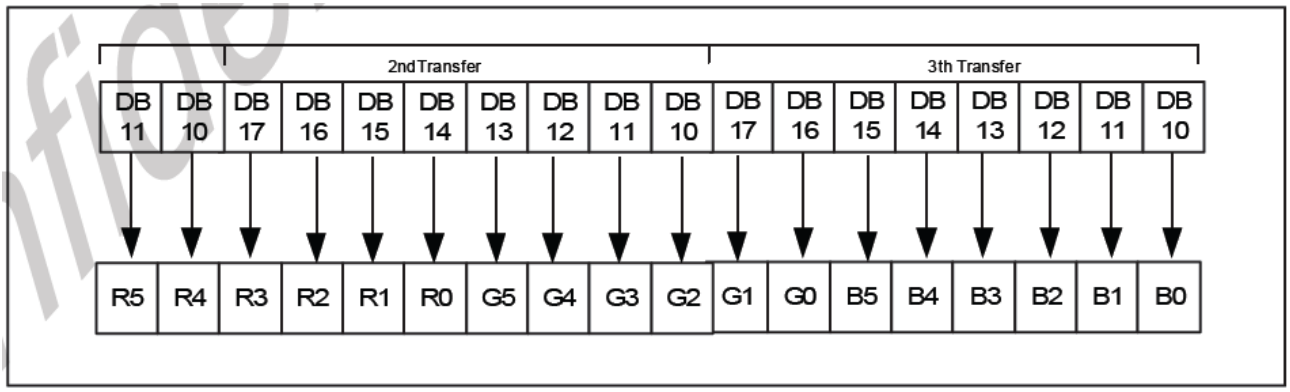



图 2.9 8 位总线，3 次传输模式，262K 色（TRIREG=1,DFM=0）

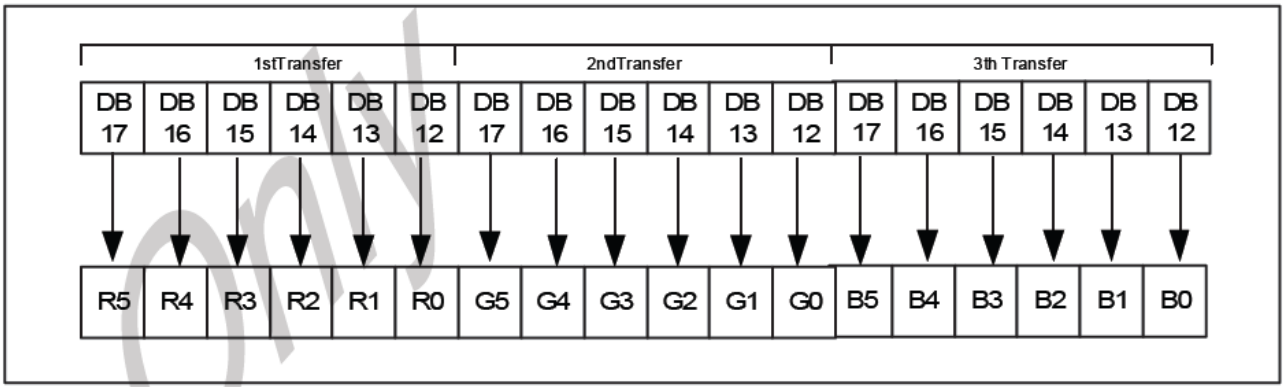



图 2.10 8 位总线，3 次传输模式，262K 色（TRIREG=1,DFM=1）

4）、显存地址设置，水平方向 GRAM Address Set Horizontal Address（R20）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD7 | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 |

5）、显存地址设置，垂直方向 GRAM Address Set Vertical Address（R21）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD16 | AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 |

6）、显存操作寄存器，对显存的读与写操作均通过该寄存器完成。

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | RAM write data (WD17-0) The DB17-0 pin assignment is different in different interface modes. | | | | | | | | | | | | | | | |

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | RAM Read data (RD17-0) The DB17-0 pin assignment is different in different interface modes. | | | | | | | | | | | | | | | |

写显存或者读显存数据时，也需要预先设置好 IR 寄存器，即指定索引寄存器指向 R22 寄存器，之后便可通过对 R22 的写或读的操作来完成显存的操作了，当然，可以配合 R20 和 R21 的设置去对指定显存地址的单元进行操作。

读操作时需要注意，将 IR 指向指 R22 后，需要有和次无效的读操作，然后才能读取到两字节的有效数据，也即第一次读数据的操作是无效的数据，接着两次读操作方是有效数据。

7）、窗口水平起始位置设置寄存器（R50）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HSA7 | HSA6 | HSA5 | HSA4 | HSA3 | HSA2 | HSA1 | HSA0 |

HSA7~HSA0 可设置窗口水平方向的起始位置。

8）、窗口水平结束位置设置寄存器（R51）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | HEA7 | HEA6 | HEA5 | HEA4 | HEA3 | HEA2 | HEA1 | HEA0 |

HEA7~HEA0 可设置窗口水平方向的结束位置。

9）、窗口垂直起始位置设置寄存器（R52）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VSA8 | VSA7 | VSA6 | VSA5 | VSA4 | VSA3 | VSA2 | VSA1 | VSA0 |

VSA8~VSA0 可设置窗口垂直方向的起始位置。

10）、窗口垂直结束位置设置寄存器（R53）

| R/W | RS | CB15 | CB14 | CB13 | CB12 | CB11 | CB10 | CB9 | CB8 | CB7 | CB6 | CB5 | CB4 | CB3 | CB2 | CB1 | CB0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | VEA8 | VEA7 | VEA6 | VEA5 | VEA4 | VEA3 | VEA2 | VEA1 | VEA0 |

VEA8~VEA0 可设置窗口垂直方向的结束位置。

## 2.4 模块寄存器初始化参考

在下面的范例代码中，使用到两个函数，其原型如下：

//延时程序

void TimeDelay(int Time)

函数: void LCD_RegWrite(unsigned char Reg_Addr,unsigned int Data)

描述: 写一个字节的数据至 LCD 中的控制寄存器当中

参数: Reg_Addr　　寄存器地址

　　Data　　写入的数据

返回: 无

```
LCD_RegWrite(0x0001,0x0100);                    //Driver Output Contral Register
```

```
LCD_RegWrite(0x0002,0x0700);      //0x0701    //LCD Driving Waveform Contral
LCD_RegWrite(0x0003,0x1030);                  //Entry Mode 设置
                                              //指针从左至右自上而下的自动增模式
                                              //Normal Mode(Window Mode disable)
                                              //RGB 格式
                                              //16 位数据 2 次传输的 8 总线设置
LCD_RegWrite(0x0004,0x0000);                  //Scalling Control register
LCD_RegWrite(0x0008,0x0207);                  //Display Control 2
LCD_RegWrite(0x0009,0x0000);                  //Display Control 3
LCD_RegWrite(0x000A,0x0000);                  //Frame Cycle Control
LCD_RegWrite(0x000C,0x0000);                  //External Display Interface Control 1
LCD_RegWrite(0x000D,0x0000);                  //Frame Maker Position
LCD_RegWrite(0x000F,0x0000);                  //External Display Interface Control 2
TimeDelay(100);
LCD_RegWrite(0x0007,0x0101);                  //Display Control
TimeDelay(100);
LCD_RegWrite(0x0010,0x16B0);     //0x14B0     //Power Control 1
LCD_RegWrite(0x0011,0x0001);     //0x0007     //Power Control 2
LCD_RegWrite(0x0017,0x0001);     //0x0000     //Power Control 3
LCD_RegWrite(0x0012,0x0138);     //0x013B     //Power Control 4
LCD_RegWrite(0x0013,0x0800);     //0x0800     //Power Control 5
LCD_RegWrite(0x0029,0x0009);                  //NVM read data 2
LCD_RegWrite(0x002a,0x0009);                  //NVM read data 3
LCD_RegWrite(0x00a4,0x0000);


LCD_RegWrite(0x0050,0x0000);                  //设置操作窗口的 X 轴开始列
LCD_RegWrite(0x0051,0x00EF);                  //设置操作窗口的 X 轴结束列
LCD_RegWrite(0x0052,0x0000);                  //设置操作窗口的 Y 轴开始行
LCD_RegWrite(0x0053,0x013F);                  //设置操作窗口的 Y 轴结束行
LCD_RegWrite(0x0060,0xA700);                  //Driver Output Control
                                              //设置屏幕的点数以及扫描的起始行
LCD_RegWrite(0x0061,0x0001);                  //Driver Output Control
```

```
LCD_RegWrite(0x006A,0x0000);          //Vertical Scroll Control


LCD_RegWrite(0x0080,0x0000);          //Display Position – Partial Display 1
LCD_RegWrite(0x0081,0x0000);          //RAM Address Start – Partial Display 1
LCD_RegWrite(0x0082,0x0000);          //RAM address End - Partial Display 1
LCD_RegWrite(0x0083,0x0000);          //Display Position – Partial Display 2
LCD_RegWrite(0x0084,0x0000);          //RAM Address Start – Partial Display 2
LCD_RegWrite(0x0085,0x0000);          //RAM address End – Partail Display2
LCD_RegWrite(0x0090,0x0013);          //Frame Cycle Control
LCD_RegWrite(0x0092,0x0000);          //Panel Interface Control 2
LCD_RegWrite(0x0093,0x0003);          //Panel Interface control 3
LCD_RegWrite(0x0095,0x0110);          //Frame Cycle Control
LCD_RegWrite(0x0007,0x0173);
```

# 3 模块驱动程序

## 3.1 驱动程序架构

动程序，包括文本显示、几何图形绘制等功能函数。通用的基础驱动程序采用全 C 的代码编程，可以方便的移值到各种拥有 C 编译器的 MCU 平台，当然用户也可以下载已经移植好的代码直接使用。

BMP 位图显示程序、JPEG 图像解码显示软件包等范例代码；用户可以从网站上获取更多的支持，同时基于此类范例的基础上设计出更绚目专业设计。这里，仅对基础驱动程序作出介绍，其它的应用范例请参考具体的代码以及相关说明文档。

模块的基础驱动程序架构如所示：

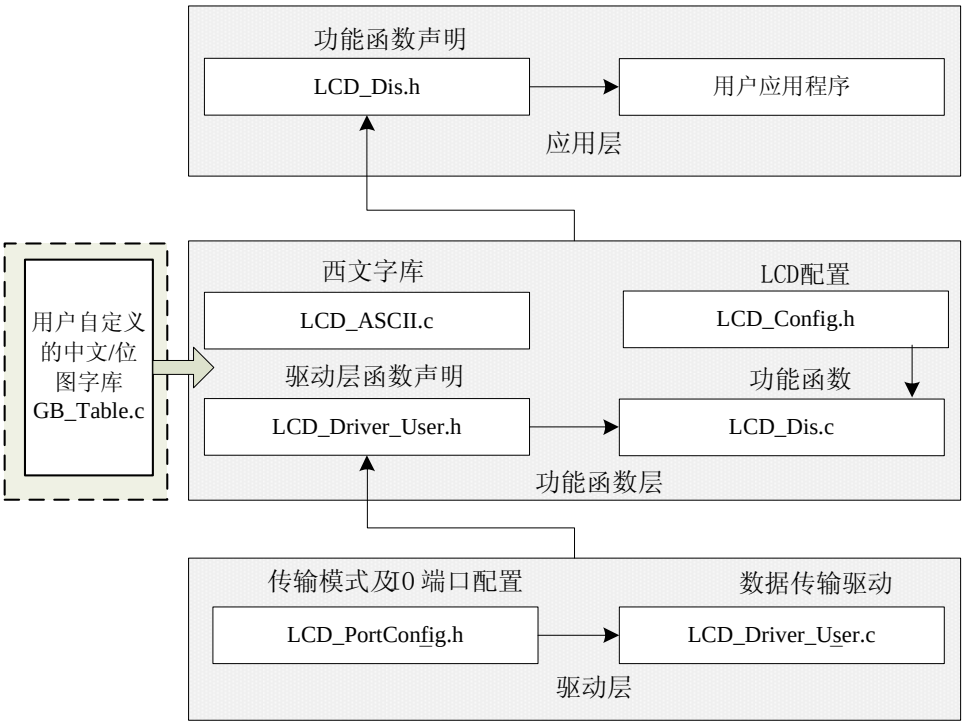



图 3.1 驱动程序架构

基础驱动程序由 8 个文件组成，分别为：底层驱动程序文件 LCD_PortConfig.h、LCD_Driver_User.c、用户 API 功能接口函数文件 LCD_Driver_User.h、LCD_Config.h、LCD_ASCII.c、LCD_Dis.c 以及 GB_Table.c、LCD_Dis.h。

**LCD_PortConfig.h**：该文件为底层驱动程序的头文件，主要对使用到的端口进行定义以及配置，用户在使用基础驱动程序时，要使端口的分配符合实际硬件的接线。

**LCD_Driver_User.c**：该文件为底层驱动程序，负责 MCU 与 模块进行数据传输的任务，主要包括初始化模块、写控制指令、写数据、读数据等函数；这些函数仅供给上一层的 LCD_Dis.c 调用，不建议用户在应用程序中调用这些函数。

**LCD_Driver_User.h**：该文件为 LCD_Driver_User.c 的对应头文件，里面对应 C 源文件当中的函数

进行外部声明。

**LCD_Config.h**：此文件为　　模块的功能配置文件，实际上此基础驱动程序是通用于各种不同的 LCD 模块的，如果需要将驱动程序应用于其它的 LCD 模块时，可以在该文件里面修改相关的配置；另外，功能配置文件里面提供了软件的坐标轴翻转功能配置（有些版本的代码中将此功能屏蔽了，仅为了让 MCU 对 LCD 的操作更快些）。

**LCD_Dis.c**：该文件中提供了供用户应用程序调用的　　驱动 API 功能函数，如绘点、绘线、绘矩形、绘圆等绘图函数，以及写字符、字符串等功能。

**LCD_ASCII.c**：为基础驱动程序的自带西文字库的数据文件。

**GB_Table.c**：为基础驱动程序的汉字/位图字库的数据文件，为用户自定义使用，另外如果定义了一些有别于基础程序所提供的范例汉定规格的字库，则用户需要自行修改 LCD_Dis.c 当中的 FontSet 函数。

**LCD_Dis.h**：该文件为 LCD_Driver_User.c 的对应头文件，里面对应 C 源文件当中的函数进行外部声明。

铭正同创所提供的通用版 LCD 基础驱动程序会不定时的推出版本的升级，但基本上程序的架构还是保持一样的，所以，可能您在网站上下载到的程序包会与这里的说明稍有不同，这点请谅解，也希望用户在使用这些程序包时，留意一下不同版本的相关说明文档。

## 3.2 常用功能函数介绍（用户 API）

LCD_Dis.c 文件中定义了常用的显示函数，包括：液晶显示控制函数、文本显示、图形显示等；下面介绍部分常用功能函数：

**液晶显示控制类：**

**1．LCD_Init 液晶初始化**

程序：void LCD_Init(void)

描述：液晶显示初始化函数

参数：无

返回：无

注意：在使用 LCD 前，首先应执行该函数，使 LCD 处于可以正常显示的状态；而该函数当中，如有必要，可以添加如端口初始化、LCD 复位等功能代码。该函数在 LCD_Driver_User.c 中定义。

**文本显示类函数：**

**2．FontSet 设置文本字体**

程序：void FontSet(int Font_NUM,unsigned int Color);

描述：选择显示字符的类型以及字符颜色

参数：Font_NUM　字符的类型选择为 1 时表示选择 15*29 点大小的西文字符显示类型，其它值为

用户自定义的汉字库/位图库使用。

Color　设置字符的颜色，16 位的字型数据，RGB 格式，与 LCD 点的 RGB 对应。

返回：无

注意：只针对显示驱动中包含的 ASCII 码显示。

**3．PutChar 显示单个字符**

程序：void PutChar(int x,int y,unsigned char a)

描述：在 x、y 的坐标上显示一个字符

参数：x 显示字符的起始列（0~239）　y 显示字符的起始行（0~319）

a　字符的编码，如当前选择的字符类型为驱动自带的西文字库，则该参数直接输入要显示的 ASCII 字符的 ASCII 码值即可；否则，该参数应输入用户自定义的字符在自定义字库中的编号。

返回：无

注意：无

**4．PutString 显示字符串**

程序：void PutString(int x,int y,char *p)

描述：从 x、y 的坐标上开始显示字符串

参数：x 显示字符的起始列（0~239）　y 显示字符的起始行（0~319）

a 字符串首地址

返回：无

注意：该函数仅当选择当前字符为驱动自带的西文字库时有效。

**图形显示类函数：**

**1．SetPaintMode 绘图模式设置**

程序：void SetPaintMode(int Mode,unsigned int Color)

描述：设置绘图模式，以及绘图的前景色

参数：Mode 无意义（暂时保留其功能）　Color 绘图时的前景色，16 位 RGB

返回：无

注意：无

**2．PutPixel 画点**

程序：void PutPixel(int x,int y)

描述：在 x、y 点上绘制一个前景色的点

参数：x 要画点的 x 坐标　y 要画的点的 y 坐标

返回：无

注意：无

**3．Line 画直线**

程序：void Line(int s_x,int s_y,int e_x,int e_y)

描述：以绘图前景色在 s_x、s_y 为起始坐标，e_x、e_y 为结束坐标绘制一条直线

参数：x 要画线的 x 起点坐标　　y 要画的线的 y 起点坐标

e_x 要画线的 x 终点坐标　　e_y 要画的线的 y 起点坐标

返回：无

注意：无

**4．Circle 画圆**

程序：void Circle(int x,int y,int r,int mode)

描述：以 x,y 为圆心 R 为半径画一个圆(mode = 0) or 圆面(mode = 1)

参数：x 要画的圆心的 x 坐标　y 要画的圆心的 y 坐标　r 半径

Mode：圆模式……

Mode = 0 画圆框

Mode = 1 画实心圆

返回：无

注意：画实心圆需要用较长时间，用户需要做好清看门狗的操作。

**5．Rectangle 画矩形**

程序：void Rectangle(unsigned left, unsigned top, unsigned right, unsigned bottom, unsigned Mode)

描述：画矩形程序

参数：left - 矩形的左上角横坐标

top - 矩形的左上角纵坐标

right - 矩形的右下角横坐标

bottom - 矩形的右下角纵坐标

Mode - 绘制模式，可以是下列数值之一：

0: 矩形框（空心矩形）

1: 矩形面（实心矩形）

返回：无

注意：无

## 3.3 模块显示控制流程

利用资料中提供的　　基础驱动程序进行 LCD 显示编程时，一定要在调用文本显示或图形显示函数，以及控制类函数前调用 LCD 的初始化函数；在 LCD 初始化程序中，会进行端口初始化、LCD 初始

设置以及变量初始化等。下面为一般　　模块显示的控制例程片段：

```
#include "LCD_Dis.h"

int main(void)

{

        ……..//用户的其它用于初始化的代码

        LCD_Init();              //初始化端口、包括 LCD 的初始设置、以及显示初始（即传到函数的参数）

        SetPaintMode(1,0xf800);          //设置绘图前景色为红色

        Rectangle(0,0,60,60,0);          //画矩形框

        Rectangle(2,2,58,58,1);          //画实心矩形

        FontSet(1,0x001f);               //设置字符为自带西文字库,颜色为蓝色

        PutChar(0,16,'A');                       //显示 ASCII 字符’A’在坐标：0,16

        PutChar(16,0,'A');                //显示 ASCII 字符’A’在坐标：16,0

        PutChar(40,20,'A');               //显示 ASCII 字符’A’在坐标：40,20

        PutChar(56,20,'A');               //显示 ASCII 字符’A’在坐标：56,20

        PutString(20,100,”Mz Design”); //显示字符串: Mz Design，起始坐标：20,100

……                                       //后续代码已省略……

}
```

显示的效果应如 图 3.2 所示：

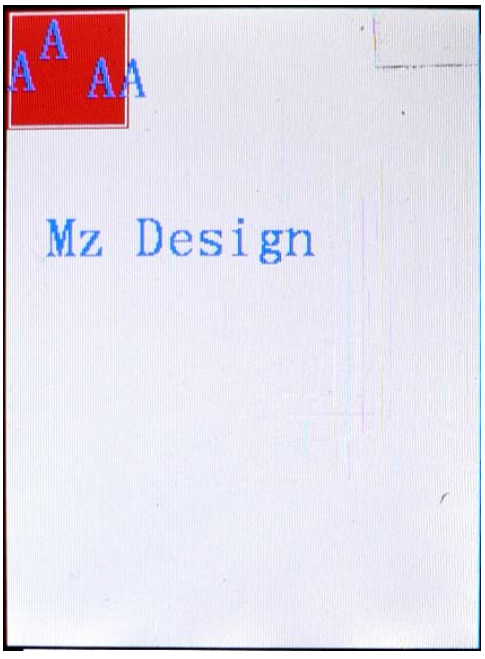



图 3.2 显示效果图

**注意**：在下载程序运行前，要保证硬件端口连线的正确，本例只是片段程序。

# 4 用户自定义汉字/位图注意事项

## 4.1 利用 模块基础驱动程序显示汉字/位图概述

模块的基础驱动程序当中，字符库的数据采用了与一般的单色点阵 LCD 的数据组成方式，即字库数据当中的一个位代表 LCD 显示中的一个像素点，取点方式为从左到右，自上到下的顺序。对于这点，驱动中自带的西文（ASCII 码）字库和用户可自定义的中文字库是一样的。

字库数据采用了以 Byte 为单位的位流结构，即当一行取点不为 8 的整数倍时，补齐数据至 8 位，无用位填零。

同一种字体大小（X 轴宽度和 Y 轴高度）的字库数据都放置在同一个数组中，除了自带的西文字库用户不需要改动，用户可以自行定义汉字字库，但需要在 FontSet 函数当中修改相应的代码，以声明自行定义的字库的 X 大小。当前面的工作都做完后，便可通过调用 FontSet 函数选择用户自定义的字库，并通过调用 PutChar 函数显示字库中的字符。

当然，根据不同的 MCU 的字长特性，用户在定义自己的字库时，需要选择合适的字模提取工具，并将字库数组定义好，这点请具体问题具体分析。

## 4.2 相关代码介绍

下面介绍一下 FontSet 函数的源码：

```
void FontSet(int Font_NUM,unsigned int Color)
{
    switch(Font_NUM)
    {
        case 1: Font_Wrod = 29;     //ASII 字符每个字符占用的存储单位
                X_Witch = 15;   //字符的宽度
                Y_Witch = 29;   //字符的高度
                Char_Color = Color;//设置字符颜色
                Char_TAB = (unsigned char *)(Asii16 - (32*29));//设置字库数组首地址
            break;
            case 2: Font_Wrod = 64;     //自定义字库的每个字符占用存储单位
                X_Witch = 32;           //字符的宽度
                Y_Witch = 32;           //字符的高度
                Char_Color = Color;
                Char_TAB = (unsigned char *)GB32; //设置字库数组首地址
            break;
```

```
    }
}
```

上面的代码中，当选择 Font_NUM 为 1 时，选用驱动自带的西文字库，这项里面的代码用户是不需要改动的。而代码当中演示了一个用户自定义的字库：GB32，该字库数组在 GB_Table.c 中定义，并在 LCD_Dis.c 的前面作了外部声明，我们也可以从代码中看出，我们需要做的是针对这个自定义的字库，设置好其宽度和高度的信息，并设置好每个字符所占用的存储单元数。

从前面的代码中可知，GB32 字库所定义的字符宽度为 32 点，高度为 32 点；而每个字符所占用的存储单元数为 64 个。下面再看一下示例的 GB32 字库数组：

```
const unsigned char GB32[] =
{
    //“科”字，字符数据流开始
    0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,
    0x0000,  0x0400,  0x0078,  0x0f00,  0x01fc,  0x0f00,  0x0fc0,  0x0f00,
    0x01c1,  0xcf00,  0x01c0,  0xef00,  0x01c8,  0x6f00,  0x01dc,  0x6f00,
    0x3ffe,  0x0f00,  0x01c0,  0x0f00,  0x03c1,  0x0f00,  0x03f1,  0xcf00,
    0x07f8,  0xef00,  0x07dc,  0xef00,  0x07cc,  0x6f70,  0x0dc0,  0x0ff8,
    0x09c0,  0x0fc0,  0x19c1,  0xff00,  0x31de,  0x0f00,  0x01c0,  0x0f00,
    0x01c0,  0x0f00,  0x01c0,  0x0f00,  0x01c0,  0x0f00,  0x01c0,  0x0f00,
    0x01c0,  0x0f00,  0x01c0,  0x0e00,  0x0000,  0x0000,  0x0000,  0x0000,//end
    //“技”字，字符数据流开始
    0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,  0x0000,
    0x0100,  0xe000,  0x0380,  0xf000,  0x0380,  0xf000,  0x0380,  0xf080,
    0x0380,  0xf1c0,  0x038f,  0xffe0,  0x03b0,  0xf000,  0x3ff8,  0xf000,
    0x0380,  0xf000,  0x0380,  0xf000,  0x0390,  0xf300,  0x03ff,  0xff80,
    0x03c2,  0x0300,  0x0f82,  0x0700,  0x3f83,  0x0600,  0x1b83,  0x0e00,
    0x0381,  0x8c00,  0x0381,  0x9c00,  0x0380,  0xf800,  0x0380,  0xf000,
    0x0380,  0xf800,  0x0381,  0xfc00,  0x0383,  0x9f80,  0x3f86,  0x0ff0,
    0x0f18,  0x03c0,  0x0760,  0x0180,  0x0000,  0x0000,  0x0000,  0x0000//end
};
```

可以看到，在 GB32 字库当中，定义了两个汉字的字符，每个字符占用了 64 个 16 位长度的 word 型空间，两个字符的数据流连在一起定义；不过由于在 FontSet 函数当中设置了 GB32 字库当中每个字符的占用存储单元数，所以是可以通过索引，分别找到两个字符的数据。

如果用户使用的 MCU 为 8 位或者更长的字长的话，在定义字库数组时需要考虑合适的数据类型，对

于这种情况，51 兼容 MCU 的驱动和基于凌阳 unsp 系列 MCU 的驱动，用户可以参考具体的驱动程序。

定义完成后，需要在 LCD_Dis.c 当中对该字库进行声明，如下：

```
extern const unsigned char GB32[];                    //32*32 自定义的汉字库
```

然后就可以在应用程序当中调用驱动所提供的 API 函数来显示这两个汉字了，如：

```
……
FontSet(2,0xf800);          //选择用户自定义的字库，并设置字符颜色
PutChar(10,10,0);           //在 10，10 点上显示“科”
PurtChar(42,10,1);          //在 42，10 点上显示“技”
…….
```

# 5 附录

## 5.1 实物图