第5章 GUI 应用基础

GUI 为 Graphics User Interface 的简写，即图形用户界面，这是用于提高人机交互友好性、易操作性的计算机程序，它是建立在计算机图形学基础上的产物。图形用户界面是当今计算机技术的重大成就之一，它极大地方便了非专业用户的使用，人们不再需要死记硬背大量的命令，而是通过窗口、菜单方便地进行操作。

随着嵌入式系统的日益发展，32 位嵌入式处理器及图形显示设备的广泛应用，目标产品对 GUI 的需求越来越多。由于嵌入式系统的资源有限，所以对 GUI 的要求是可载剪的，高速度的。本章先介绍点阵式图形液晶显示屏及常用的驱动控制器，并列出几款点阵图形液晶模块及在 EasyARM2200 开发板上的连接及驱动程序的编写，接着介绍嵌入式系统简易的图形用户界面 ZLG/GUI 的应用。

5.1 点阵式图形液晶显示屏

点阵式图形液晶显示屏是平板显示器件中的一种，具有低工作电压、微功耗、无辐射、体积小等特点，被广泛应用于各种各样嵌入式产品中，如手机、PDA、数码相机、电子游戏机和便携式仪表等等。随着 STN 和 TFT 液晶显示屏技术的成熟发展及制造成本的不断降低，点阵式图形液晶显示屏也就成为了嵌入式系统中最主要的图形显示设备。

5.1.1 点阵式图形液晶显示屏的特点

液晶，是一种在一定温度范围内呈现既不同于固态、液态，又不同于气态的特殊物质态，它既具有各向异性的晶体所特有的双折射性，又具有液体的流动性。液晶显示器件(英文的简写为 LCD)就是利用液晶态物质的液晶分子排列状态在电场中改变而调制外界光的被动型显示器件。

点阵式图形液晶显示屏是 LCD 的一种，能够动态显示图形、汉字以及各种符号信息，为各种电子产品提供了友好的人机界面。点阵式图形液晶显示屏的主要特点如下(这些特点也就是 LCD 的特点)：

1. 工作电压低

一般 LCD 的驱动电压为 1.5V～3V 左右，可以直接与大规模集成电路直接匹配，驱动极为方便。相比之下，CRT 显示器则需要几十千伏调制电压。在便携式产品中及其它嵌入式系统中，系统电源电压一般都不高，所以使用 LCD 非常合适。

2. 微功耗

图形液晶显示屏的功耗约为 1μW/cm^2，是所有显示器件中功耗最小的。在液晶显示器件中，功耗主要消耗在驱动 IC 及其外围电路上。由于其功耗极低，所以在电池供电等产品上得到广泛应用。

3. 体积小

液晶显示屏为平板形结构，体积相对阴极射线电子束管(CRT)显示器、投影显示器要小得多，重量也非常轻。这一特点对于嵌入式产品来说是非常重要的，所以大量的便携式产品中均使用 LCD 进行信息的显示。

4. 可视面积大

根据实际需求，液晶显示屏可以把尺寸做得很大，也可以做得很小，而且对于相同尺寸的显示器，液晶显示器的可视面积比其它类型显示器要更大一些。

5. 无电磁辐射

液晶显示器在防止辐射方面具有先天的优势，因为它根本就不存在辐射。在电磁波的防范方面，液晶显示器也有自己独特的优势，可以采用严格的密封技术将来自驱动电路的少量电磁波封闭在显示器中，而普通显示器为了散发热量的需要，必须尽可能地让内部的电路与空气接触，这样内部电路产生的电磁波也就大量地向外“泄漏”了。

6. 数字接口

点阵式液晶显示屏均是数字接口，不需要转换成模拟信号，理论上，这会使像素的色彩和定位都更加准确完美。

7. 寿命长

液晶显示器件本身几乎没有什么劣化问题，在允许的工作温度范围内工作时，通常 LCD 的寿命长达 50000 小时以上。

5.1.2 点阵式图形液晶显示屏的种类

1. 按显示原理分类

在日常生活中，我们可能接触过各种各样的点阵式图形液晶屏，按显示原理分通常有以下几种类型：

- **TN(Twist Nematic)扭曲向列型**

将涂有 ITO 透明导电层的两片玻璃基板间夹上一层正介电各向异性液晶，液晶分子沿玻璃表面平行排列，排列方向在上下玻璃之间连续扭转 90°。然后上下各加一偏光片，底面加上反光片，基本就构成了 TN 型液晶显示屏。

由于 TN 型液晶显示器件的电光响应慢，电光曲线陡度不够陡峭，所以在进行动态、多路驱动上有一定限制。使用 TN 型液晶显示技术，最多只能做到 64 路点阵的驱动，这种类型的图形液晶显示屏只能做单色小规模点像素的液晶屏。

- **STN(Super Twist Nematic)超扭曲向列型**

STN 型跟 TN 型结构大体相同，只不过液晶分子扭曲 180°，还可以扭曲 210°或 270°等，其特点是电光响应曲线更好，可以适应更多的行列驱动。彩色和单色模式的 STN 技术可以做到 2000 路点阵以上的驱动。

STN 液晶只可以实现伪彩色(一般人眼可以分辨 18bit 色，即 256K 色，所以达到 18bit 色和超过 18bit 色的被称之为真彩色，否则称之为伪彩色)显示，可以实现 VGA(640×480)、SVGA(800×600)等一些较高的分辨率，但由于构成它们的矩阵方式是无源矩阵，每个像素实际上是个无极电容，容易出现串扰现象，从而不能显示真正的活动图像。

TN 或 STN 型液晶显示器件，一般是对液晶盒施加电压，达到一定电压值，对行和列进行选择，出现“显示”现象，所以行列数越多，要求驱动电压越高。因此往往 TN 或 STN 型点阵式图形液晶显示器件要求有较高的正极性驱动电压或较低的负极性电压，也因为如此，TN 和 STN 型液晶难以做到高分辨率。

目前，大部份点阵式图形液晶显示屏均采用 STN 技术。

- **TFT(Thin Film Transistor)薄膜晶体管型**

TFT 为薄膜晶体管有源矩阵液晶显示器件，在液晶显示屏的每个像素点上设计一个薄膜晶体管来直接驱动，从而可以大大提高液晶显示器的对比度，响应时间，色彩还原能力。由于每个节点都相对独立，并可以连续控制，不仅提高了显示屏的反应速度，同时可以精确控制显示色阶，这样就容易实现真彩色、高分辨率的液晶显示器件。现在的 TFT 型液晶一般都实现了 18bit 以上的彩色(256K 色)，在分辨率上，已经实现 VGA(640×480)、SVGA(800×600)、XGA(1024×768)、SXGA(1280×1024)甚至 UXGA(1600×1200)。

TFT 液晶显示器也存在着耗电较大和成本较高的不足，多应用于中、高档电子产品中。

另外，液晶显示器还有宾主型(GH)、相变(PC)、电控双折射型(ECB)和铁电型(FE)等等，由于它们的应用远远没有 TN、STN 和 TFT 这么广泛，所以这里就不一一介绍了。

2. 按点像素分类

按点像素来分，我们常见点阵式图形液晶屏有以下几种：

- **单色屏**

只能显示两种颜色，比如黑白点阵式图形液晶显示屏(黑色字/白底色)，显示的点为黑色，不显示的点为白色。有些单色点阵式图形液晶显示屏的颜色为黑色字/黄绿底色、白色字/暗蓝底色、蓝色字/白底色等等。单色点阵式图形液晶均采用 TN/STN 技术，分辨率一般小于 640×480，因为价格较低而得到广泛应用。

- 4 级灰度屏

将黑白色分为 4 级。实质上是由控制器支持。

- 8 级灰度屏

将黑白色分为 8 级。实质上是由控制器支持。

- 16 级灰度屏

将黑白色分为 16 级。实质上是由控制器支持。

- 64 级灰度屏

将黑白色分为 64 级。实质上是由控制器支持。

- 256 级灰度屏

将黑白色分为 256 级。实质上是由控制器支持。

- 16 色屏

能显示标准 16 色，一般采用 STN 技术。

- 256 色伪彩色屏

能显示 RGB 256 色伪彩色，一般采用 STN 技术，分辨率一般小于 640×480。

- TFT 真彩色屏

采用 TFT 技术，一般都实现了 18bit 以上的彩色(256K 色)，分辨率高。

3. 触摸屏

对于点阵式图形液晶显示屏，为了提高人机交互的友好性，常常在显示屏上粘上一层透明的薄膜体层，用于检测屏幕触摸输入信号，形成触摸屏。所以我们可以将点阵式图形液晶显示屏分为触摸屏和非触摸屏两种。以下是几种常见的触摸屏的类型：

- **电阻式触摸屏**

(1) **四线电阻式触摸屏**

四线电阻式触摸屏(简称四线式触摸屏)包含两个透明的阻性层。其中一层在屏幕的左右边缘各有一条垂直总线，另一层在屏幕的底部和顶部各有一条水平总线，如图 5. 1 所示。四线式触摸屏是最常用的触摸屏之一，所以这里将对其作重点介绍。

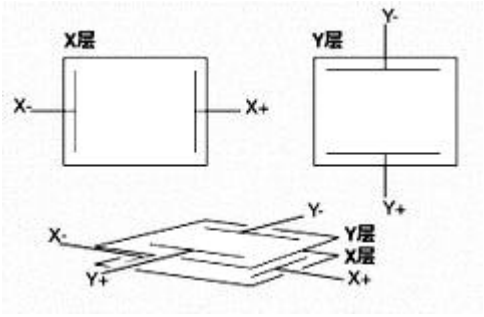

图 5. 1 四线式触摸屏结构示意图

触摸屏的两个金属导电层分别用来测量 X 轴和 Y 轴方向的坐标。用于 X 坐标测量的导电层从左右两端引出两个电极，记为 X+和 X-。用于 Y 坐标测量的导电层从上下两端引出两个电极，记为 Y+和 Y-。这就是四线电阻式触摸屏的引线构成。

当在一对电极上施加电压时，在该导电层上就会形成均匀连续的电压分布。若在 X 方向的电极对上施加一确定的电压，而 Y 方向电极对上不加电压时，在 X 平行电压场中，触点处的电压值可以在 Y+(或 Y-)电极上反映出来，通过测量 Y+电极对地的电压大小，便可得知触点的 X 坐标值。同理，当在 Y 电极对上加电压，而 X 电极对上不加电压时，通过测量 X+(或 X-)电极的电压，便可得知触点的 Y 坐标值。测量原理如图 5. 2 所示。

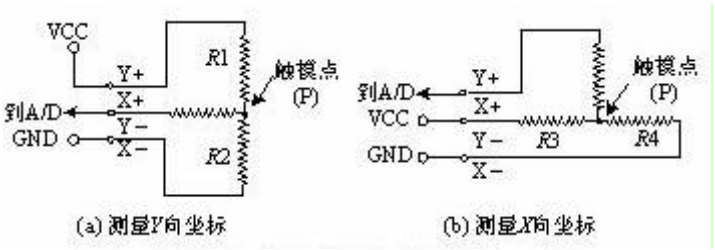

图 5. 2 四线式触摸屏测量原理

在使用触摸屏时，需要一个 ADC 转换器将模拟信号转换成数字信号，通常直接使用触摸屏控制器完成这一功能，也可以使用微处理器内部的 ADC 转换器实现。触摸屏控制器的主要功能是在微处理器的控制下向触摸屏的两个方向分时施加电压，并将相应的电压信号传送给自身 A/D 转换器，在微处理器 SPI 口提供的同步时钟作用下将数字信号输出到微处理器。常见的触摸屏控制器如：ADS7843/7846、MK715 等等。

(2) **五线电阻式触摸屏**

五线电阻式触摸屏(简称五线式触摸屏)与四线式有所不同，主要区别在于五线式触摸屏

将其中一导电层的四端均引出来作为四个电极，另一导电层仅仅作为测量的导体输出 X 方向和 Y 方向的电压，测量时要交替在 X 方向电极和 Y 方向电极上施加电压。五线式触摸屏控制器可以使用 TI 公司的 ADS7845。

- **表面声波触摸屏**

表面声波触摸屏的触摸部分是一块强化玻璃板，安装在显示屏幕的前面。玻璃屏的左上角和右下角各固定了竖直和水平方向的超声波发射换能器，右上角则固定了两个相应的超声波接收换能器。玻璃屏的四个周边则刻有 45°角由疏到密间隔非常精密的反射条纹。工作原理以右上角的 X 轴发射换能器为例：发射换能器把控制器通过触摸屏电缆送来的电信号转化为声波能量向左方表面传递，然后由玻璃板下边的一组精密反射条纹把声波能量反射成向上的均匀面传递，声波能量经过屏体表面，再由上边的反射条纹聚成向右的线传播给 X 轴的接收换能器，接收换能器将返回的表面声波能量变为电信号。当手指或其他能够吸收或阻挡声波能量的物体触摸屏幕时，X 轴途经手指部位向上走的声波能量被部分吸收，控制器再根据相应的数据计算出手指的位置。除了能响应 X、Y 坐标外，它还响应 Z 轴坐标，也就是能感知用户触摸压力大小。

表面声波触摸屏的主要特点是：性能稳定、反应速度快、清晰美观、抗暴、超高透光率，触摸准确，但容易受灰尘和水的干扰。主要代表产品有国产 TPS 触摸屏和进口 ELO 触摸屏两种。

- **电容式触摸屏**

电容式触摸屏是在玻璃屏幕上镀一层透明的薄膜体层，再在导体层外加上一块保护玻璃，双玻璃设计能彻底保护导体层及感应器。此外，在附加的触摸屏四边均镀上狭长的电极，在导电体内形成一个低电压交流电场。用户触摸屏幕时，由于人体电场，手指与导体层间会形成一个“耦合电容”，四边电极发出的电流会流向触点，而电流的强弱与手指及电极间的距离成正比，通过计算电流的比例及强弱，就能准确算出触摸点的位置。

电容式触摸屏的特点为触摸准确度较高，抗干扰能力较强（但怕静电干扰）。主要代表产品为美国 MicroTouch 公司的电容式触摸屏。

- **红外线触摸屏**

该技术是指通过红外线发射与接收感应元件在透明介质上形成红外线探测网，利用触摸体阻隔红外线的工作方式进行坐标点测定工作的技术。现在的红外触摸屏在技术性能参数上已基本达到其他触摸屏的水平。红外线式触摸屏安装简单，只需为显示器加上光点距架框，无需在屏幕表面加上涂层或接驳控制器。光点距架框的四边排列了红外线发射管及接收管，在屏幕表面形成一个红外线网。用户以手指触摸屏幕某一点，便会挡住经过该位置的横竖两条红外线，计算机便可即时算出触摸点位置。外界光线变化，如阳光或室内射灯等均会影响红外线触摸屏的准确度，且它不防水及污物，甚至细小的外来物也会导致误差，不适宜放置于户外。

知名的触摸屏制造商有美国的 EloTouch、MicroTouch、Keytec 公司，英国的 Intasolve 公司，日本的 Minato 和 Carrolltouch，中国台湾的 Ingenious Technology 公司等，主要面向电阻、电容和表面声波触摸屏。四线式电阻触摸屏较流行的是中国台湾生产的四线式触摸屏和内地南方地区生产的四线式触摸屏。五线式电阻触摸屏较为流行的是美国 EloTouch 和 Microtouch 品牌。

5.2 常用点阵式图形液晶控制器

在嵌入式系统应用中，如果微控制器本身带有液晶驱动控制功能，则可以直接对点阵式液晶显示屏进行连接控制；如果微控制器本身没有液晶驱动控制功能，则需要外扩液晶驱动板来连接液晶显示屏，或者使用点阵式图形液晶显示模块。

目前较为流行的点阵式图形液晶控制器有 EPSON 公司的 SED 系列产品，Hitachi 公司的 HD 系列产品和东芝公司的部份产品，这里将一些常用的控制器及其参数列举如下。

5.2.1 SED 系列点阵式图形液晶控制器

- **SED1335F**

SED1335F 是 EPSON 主推的点阵式图形液晶控制器芯片之一，适用于中等规模单色点阵图形液晶(通过外部增加的色彩生成电路和 M 信号生成电路，可实现 4 级灰度 LCD 驱动)，支持 64K 字节显示存储器，具有片内字符发生器 CGROM，器件逻辑框图如图 5. 3 所示，器件引脚图如图 5. 4 所示。SED1335F 具有功能较强的 I/O 缓冲器、指令丰富、可完成多种文本图形的显示、刷新功能。相关参数如下：

宽工作电压：2.7～5.5V

最大驱动液晶点阵：单色 640×256

支持存储器大小：64K 字节 SRAM

MPU 接口：适配 6800 系列或 8080 系列处理器时序接口

屏幕卷动：水平和垂直方向卷动

显示方式：最多 2 层字符和图形混合，最多 3 层图形混合

驱动 LCD 占空比：1/2～1/256

低功耗：工作电流典型值 3.5mA；空闲状态典型值 0.05uA。

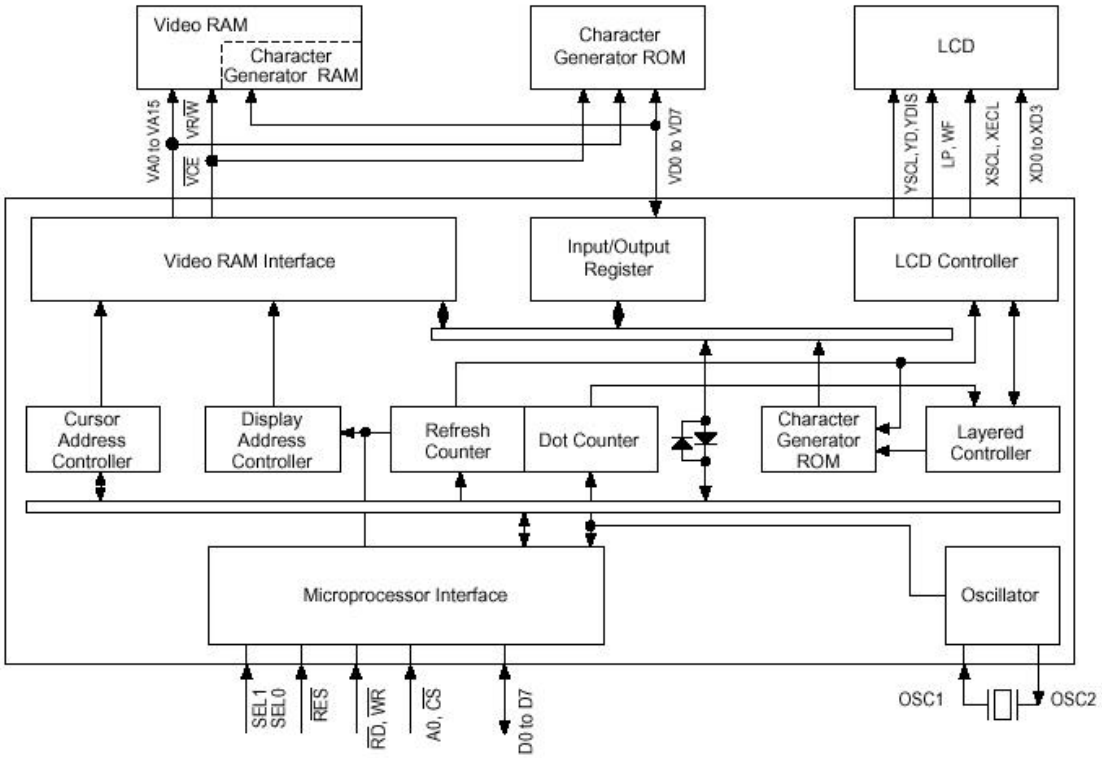

图 5. 3 SED1335F 逻辑框图

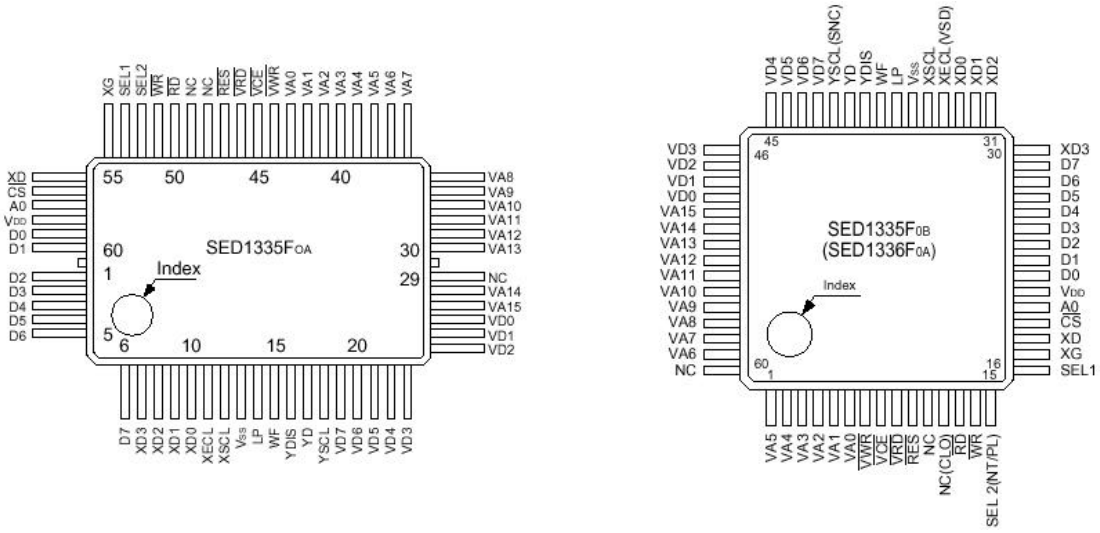

图 5. 4 SED1335F 引脚图

- **SED1351F**

SED1351F **是高占空比的点阵式图形液晶控制器，支持** 4 **级灰度液晶显示，支持虚拟屏，支持** 8 **位或** 16 **位总线接口，具有循环侵占方式的数据和地址的电路，使** MPU **存取** VRAM **时显示不受干扰**。SED1351F **是一款低功耗** CMOS **控制器，器件逻辑框图如图** 5. 5 **所示，**SED1351F0A **器件引脚图如图** 5. 6 **所示，**SED1351FLB **器件引脚图如图** 5. 7 **所示，相关参数如下：**

工作电压：5V(SED1351F0A)**或** 3V(SED1351FLB)

最大驱动液晶点阵：单色 1024×512，**灰度** 512×512

VRAM 存储器大小： 64K **字节** SRAM

MPU 接口：8 **位或** 16 **位总线接口**

LCD 显示方式：黑白色显示或 4 **级灰度显示**

屏幕卷动：水平和垂直方向卷动

驱动 LCD 占空比：最高 1/1024

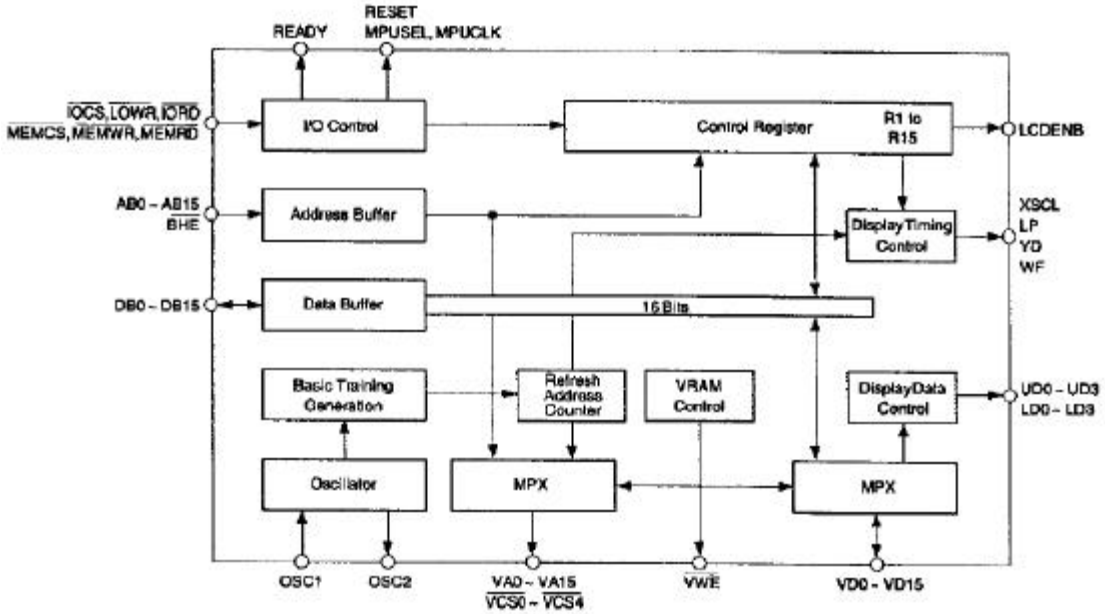

图 5. 5 SED1351F 逻辑框图

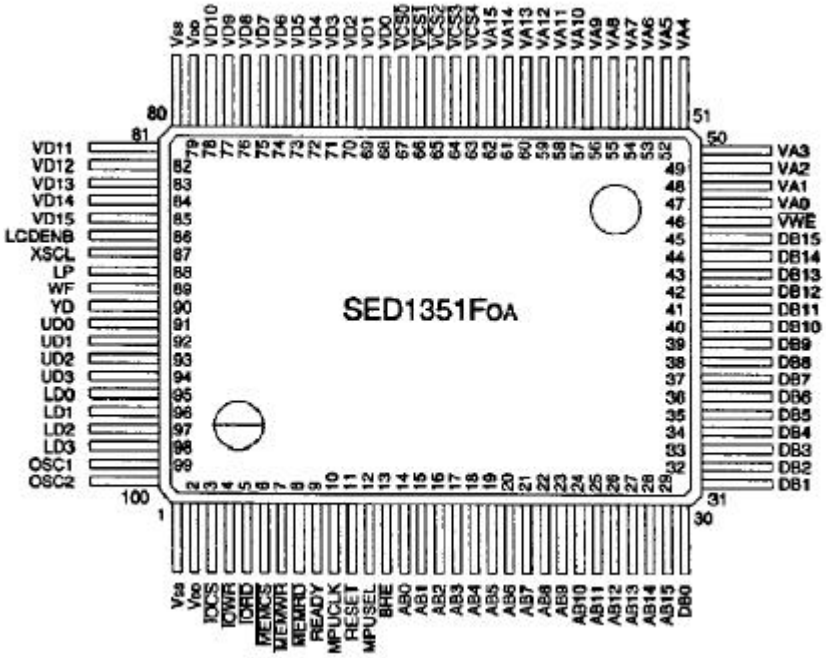

图 5. 6 SED1351F0A **引脚图**

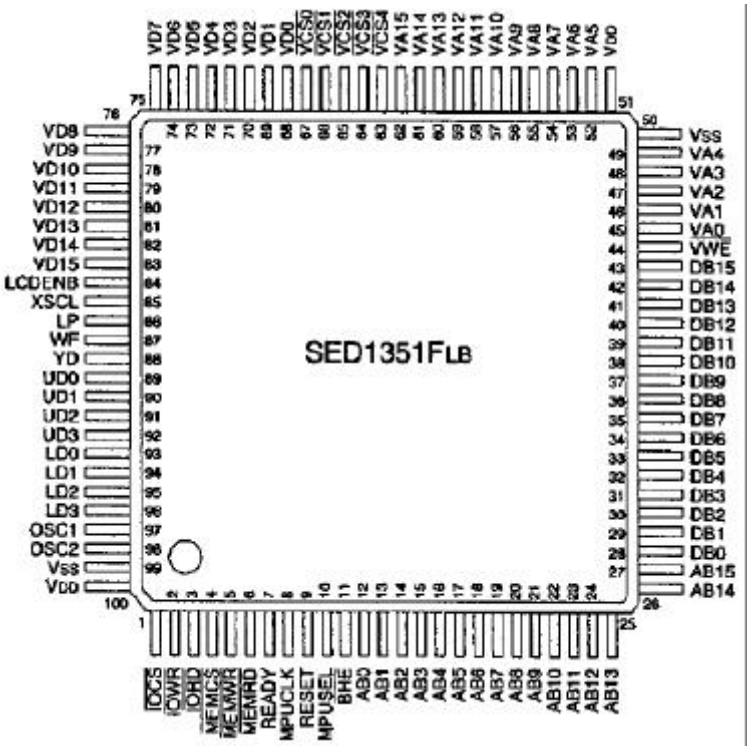

图 5. 7 SED1351FLB **引脚图**

- **SED1520 系列**

SED1520 **系列液晶驱动器是显示字符和图形的单片点阵式液晶驱动控制芯片，适用于小规模单色点阵图形液晶，其内部包含有显示数据** RAM，**具有较宽的工作电压范围，且功耗极低**。SED1520 **原理逻辑框图如图** 5. 8 **所示，**SED1520 **引脚图如图** 5. 9 **所示**。

SED1520 **系列液晶控制器具有以下基本功能：**

低功耗 CMOS **工艺；**

高速 8 **位** MPU **接口**，6800 **系列或** 8080 **系列处理器接口时序**；
内部显示数据 RAM **为** 2560 **位**；
丰富的显示指令集；
支持主从操作，以驱动更多点阵路数；
LCD **电压** VDD－V5=3.5～13V；
驱动 LCD **占空比最高** 1/32；
工作电压为 2.4～7V。

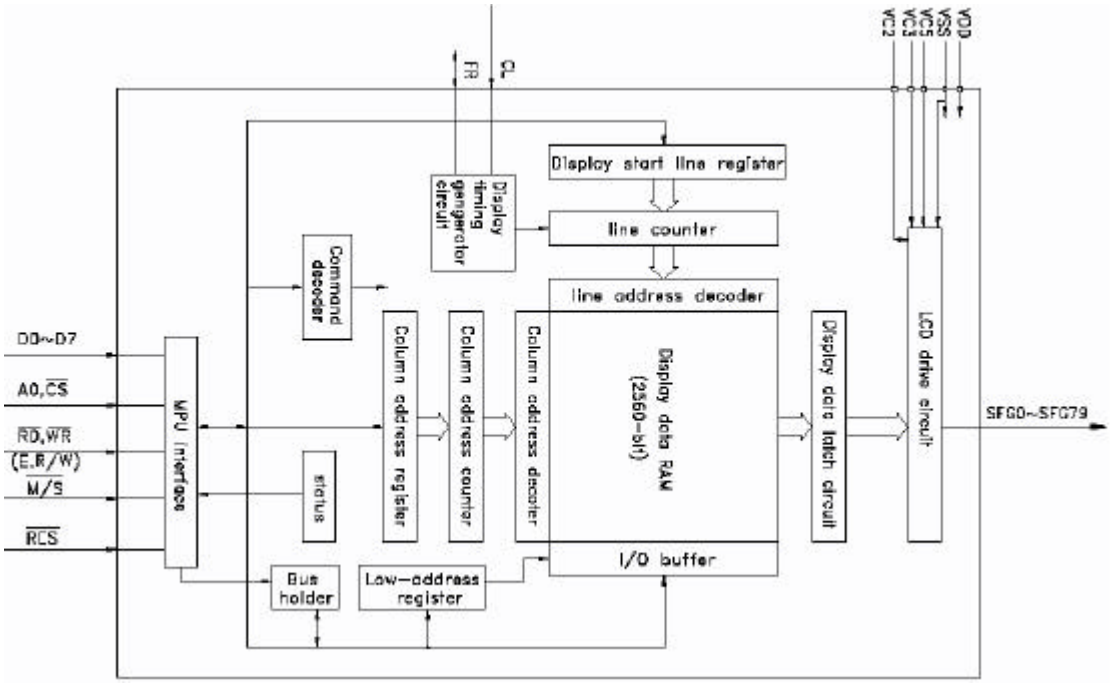

图 5. 8　SED1520 **原理框图**

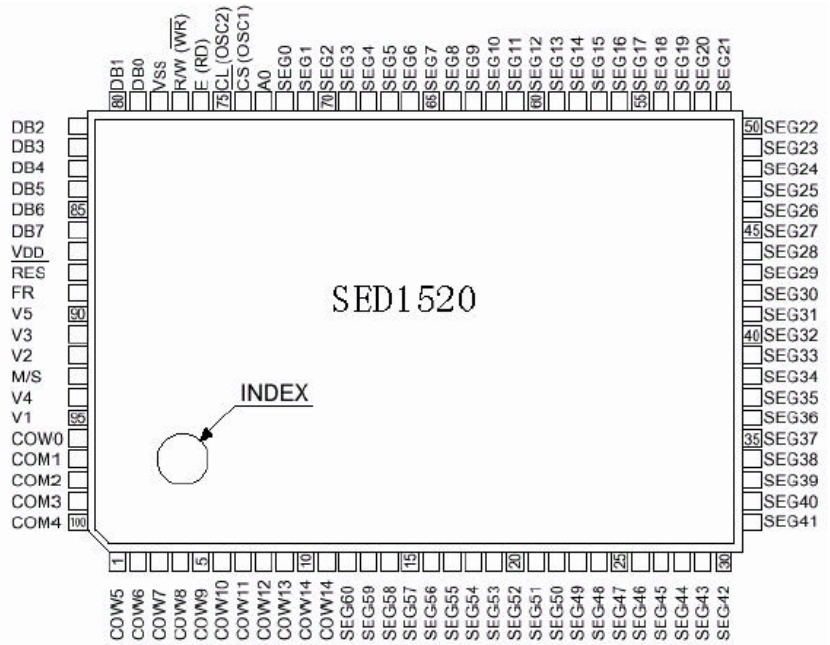

图 5. 9　SED1520 **引脚图**

- **SED1565 系列**

SED1565 系列液晶驱动器是显示字符和图形的单片点阵式液晶驱动控制芯片，适用于小规模单色点阵图形液晶，其内部包含有 65×132 位显示数据 RAM，具有较宽的工作电压范围，且功耗极低。MPU 通过 8 位并行方式或串行方式，将数据送入芯片内部的显示 RAM，芯片自行产生 LCD 的驱动信号。SED1565 原理逻辑框图如图 5. 10 所示，其封装形式有裸片封装和 TCP 封装。

SED1565 系列液晶控制器具有以下基本功能：

低功耗 CMOS 工艺；

8 位并行接口或串行接口；

内部显示数据 RAM 为 65×132=8580 位；

丰富的显示指令集；

支持主从操作，以驱动更多点阵路数；

LCD 电压 V5=－4.5～－16V；

驱动 LCD 占空比最高 1/65；

工作电压 1.8～5.5V。

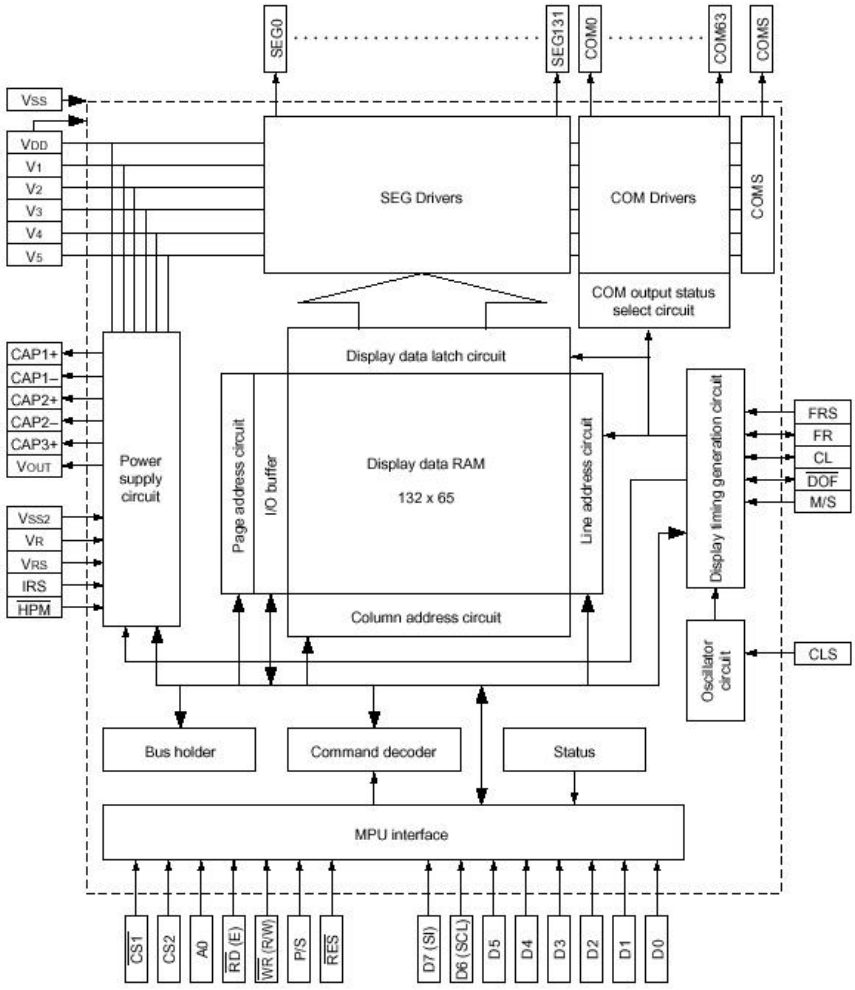

图 5. 10 SED1565 原理框图

- **SED1353F**

SED1353F 是一款最大能支持 1024×1024 点阵液晶的控制器，支持最大 4/16/256 色伪彩色液晶显示，支持黑白色、4/16 级灰度单色液晶显示，支持 128K 字节显示存储

器，支持虚拟屏，具有两种节电模式。器件逻辑框图如图 5. 11 所示，SED1353F0A 器件引脚图如图 5. 12 所示，SED1353F1A 器件引脚图如图 5. 13 所示，相关参数如下：

宽工作电压：2.7～5.5V

最大驱动液晶点阵：黑白色 1024×1024

4 级灰度/伪彩色 1024×512

16 级灰度/伪彩色 512×512

256 伪彩色 512×256

支持存储器大小：128K 字节 SRAM，支持 8 位/16 位宽度

MPU 接口：16 位 16MHZ 的 MC68xx MPU 接口

带有 READY(或 WAIT#)信号的 8/16 位 MPU 接口

显示方式：内置调色板

驱动 LCD 占空比：最高 1/1024

低功耗：5V 电压下，正常工作典型电流值 11mA

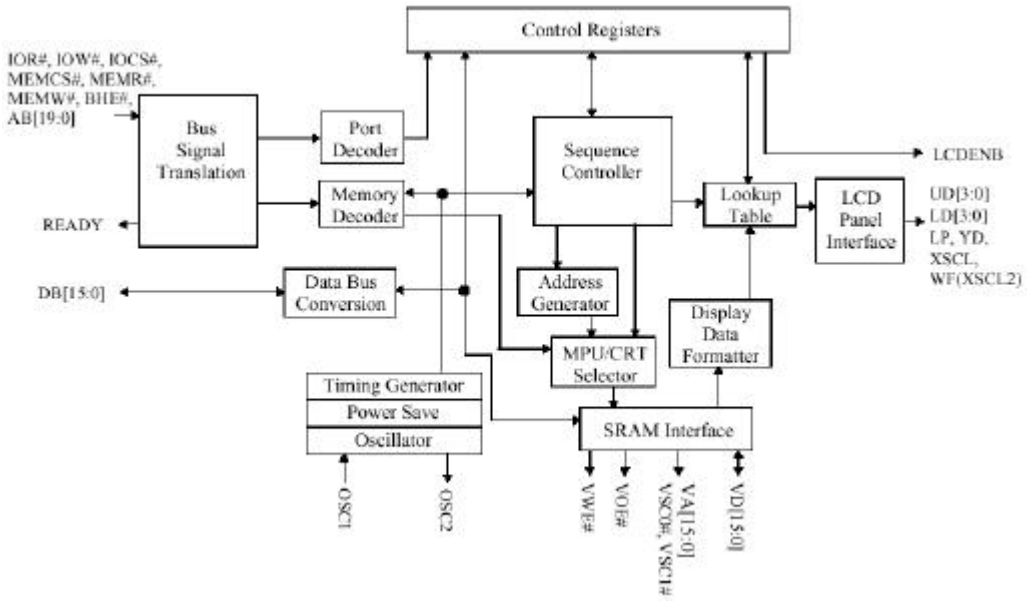

图 5. 11 SED1353F 原理框图

图 5. 12 SED1353F0A 引脚图

图 5. 13 SED1353F1A 引脚图

- **SED1354F**

SED1354F 是一款能支持 TFT 点阵液晶和 CRT 显示器的控制器，灵活方便的硬件配置和软件编程功能，支持最大 16/256/4096 色伪彩色液晶显示，支持 9/12 位、18 位 TFT 液晶显示，支持 16/256 级灰度单色液晶显示，支持 2M 字节 DRAM 显示存储器，支持虚拟屏，具有两种节电模式。器件逻辑框图如图 5. 14 所示，SED1354F 器件引脚图如图 5. 15 所示。SED1354F 适用于大规模点阵液晶，低成本，低功耗，相关参数如下：

宽工作电压：2.7～5.5V

最大驱动液晶点阵：16 级灰度/伪彩色 800×600
256 级灰度/伪彩色 800×600
4096 色伪彩色 800×600
9/12 位 TFT 液晶(64K 色) 800×600
18 位 TFT 液晶(64K 色) 800×600

支持存储器大小：2M 字节 EDO-DRAM 或 FPM-DRAM，支持 16 位宽度

MPU 接口：16 位总线接口，适应多种 CPU 的总线形式

显示方式：内置调色板

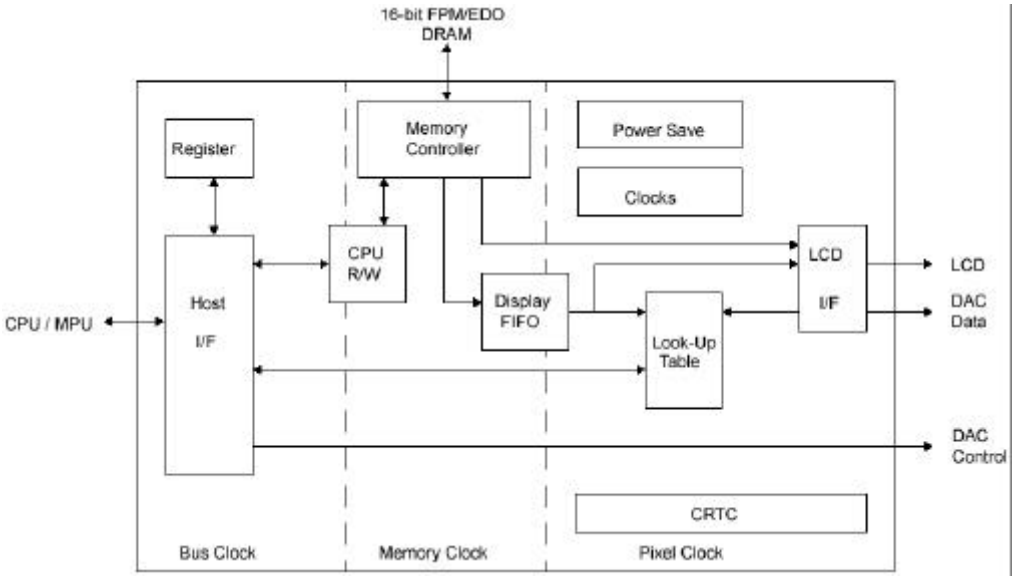

图 5. 14 SED1354 原理框图

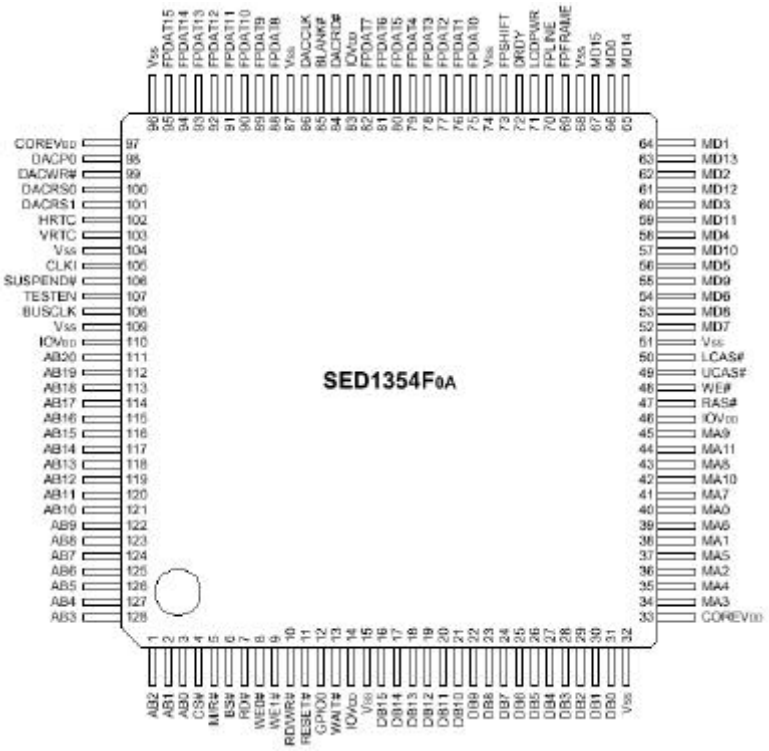

图 5. 15 SED1354 引脚图

5.2.2 HD 系列点阵式图形液晶控制器

- **HD61202/3**

HD61202/3 是日本日立公司的适用于小规模单色点阵液晶的液晶列/行驱动控制器，HD61202 和 HD61203 必须配套使用。使用两片或三片 HD61202 可组成 128 点列或 192 点列的列驱动器组。HD61202/3 内置 512 字节显示存储器，显示屏上各像素点的显示状态与显示存储器的各位数据一一对应。HD61202/3 是以 8 位总线方式与 MPU 接口，配备有显示存储器管理电路，MPU 通过这一电路访问显示存储器。HD61202 原理框图如图 5. 16 所示，芯片引脚图如图 5. 17 所示。

HD61202/3 的特点：

宽工作电压：2.7～5.5V

最大驱动液晶点阵：黑白色 64 列/行

支持存储器大小：64×64 / 8=512 字节

MPU 接口：8 位并行数据接口，适配 MC6800 系列时序

驱动 LCD 占空比：最高 1/64(HD61202) 1/128(HD61203)

低功耗：显示期间最大功耗 2mW

HD61202/3 广泛应用于中规模单色点阵液晶，但是目前这种驱动控制芯片已停产，市场上采用兼容产品 KS0107/8 代替。

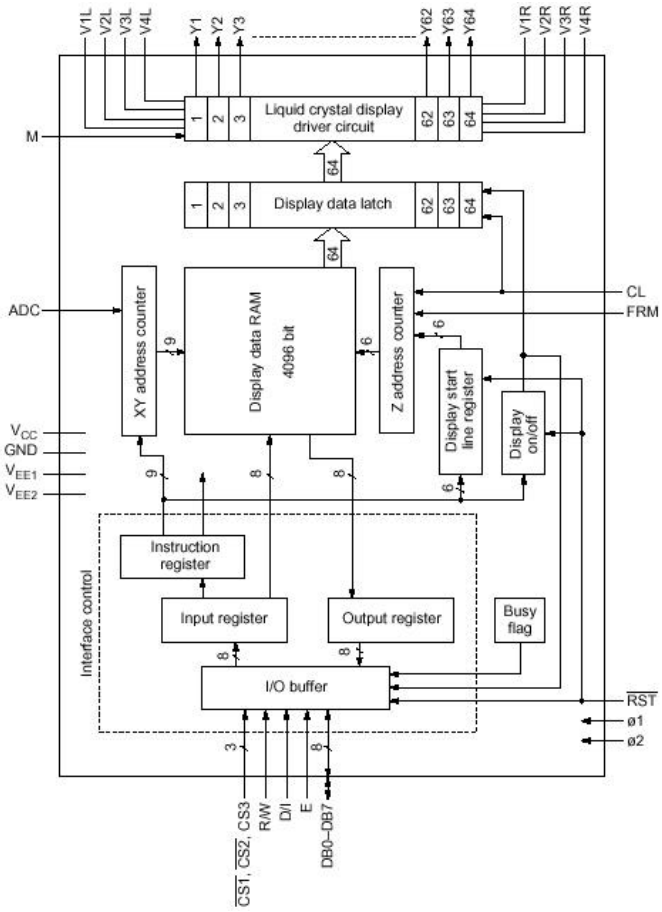

图 5. 16 HD61202 **原理框图**

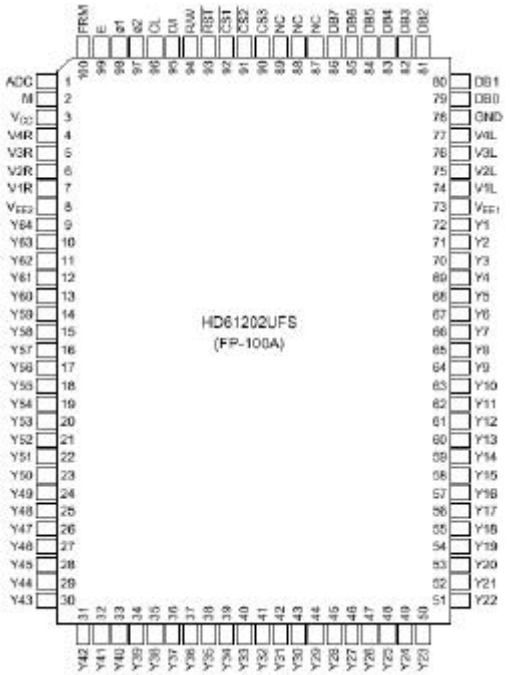

图 5. 17 HD61202 **引脚图**

- **HD61830**

HD61830 **适用于小规模的单色点阵图形液晶控制器，内置字符发生器** CGROM，**支持** 64K **字节显示存储器，具有片内字符发生器** CGROM，**器件逻辑框图如图** 5. 18 **所示，器件引脚图如图** 5. 19 **所示**。HD61830 **具支持主从并联使用，以驱动更多点阵路数。相关参数如下**：

工作电压：5.0V

最大驱动液晶点阵：单色 524288 **点**

支持存储器大小：64K **字节** SRAM

MPU 接口：8 **位并行数据接口，适配** M6800 **系列时序**

显示方式：图形方式/文本方式

字符发生器：160 **种** 5×7 **字符**(CGROM)

32 **种** 5×11 **字符**(CGROM)

可扩展 256 **字符外部** ROM

驱动 LCD 占空比： 最高 1/128

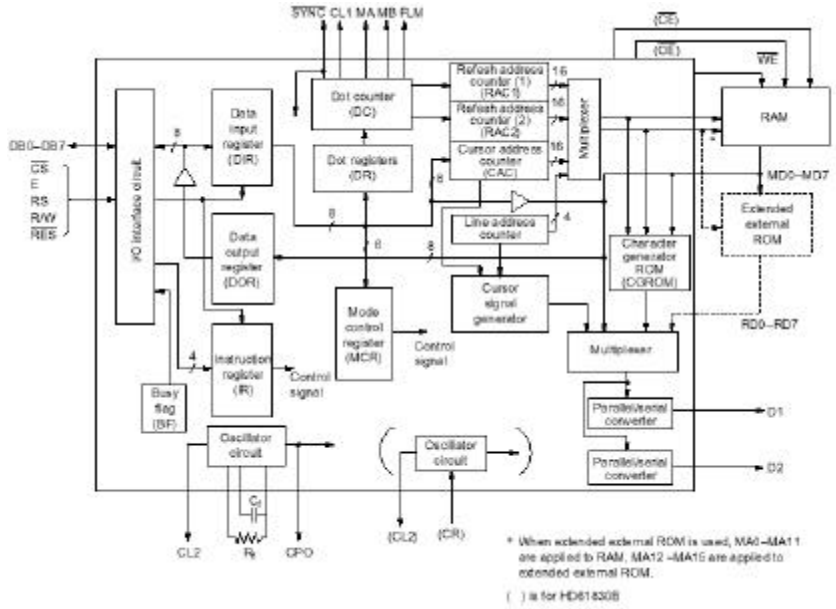

图 5. 18 HD61830 **原理框图**

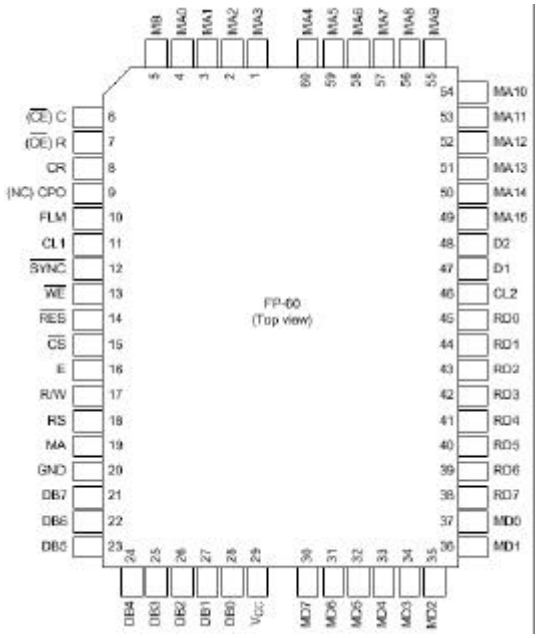

图 5. 19 HD61830 **引脚图**

5.2.3 东芝公司点阵式图形液晶控制器 T6963C

T6963C **常用于中规模的单色点阵图形液晶的显示控制器，其最大特点是具有独特的硬件初始化值设置功能，显示驱动所需的参数如占空比系数、驱动传输的字节数/行、字符的字体选择等均由引脚电平设置。器件逻辑框图如图** 5. 20 **所示，器件引脚图如图** 5. 21 **所示，相关参数如下：**

工作电压： 5.0V

最大驱动液晶点阵：单色 640×128(**单屏**)

支持存储器大小： 64K **字节** SRAM

MPU 接口： 8 **位并行数据接口**

显示方式：图形方式、文本方式、图形文本混合方式

字符发生器： 128 **种** 5×**8**、6×**8**、**7**×**8**、**8**×**8 字符**(CGROM)

可管理外部 2K **字节的** CGRAM(**8**×**8 字符**)

驱动 LCD 占空比： 1/16～1/128

低功耗：显示期间电流典型值 3.3mA

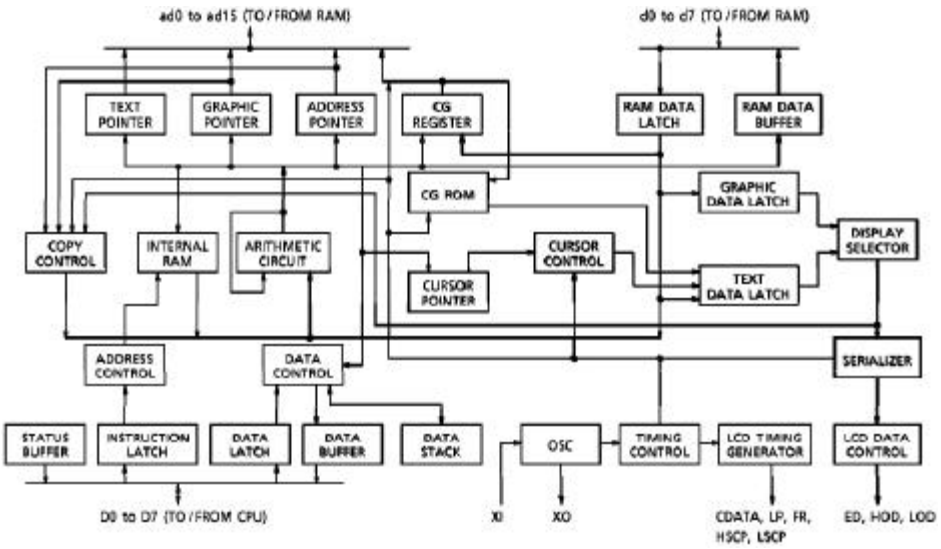

图 5. 20 T6963C **原理模框图**

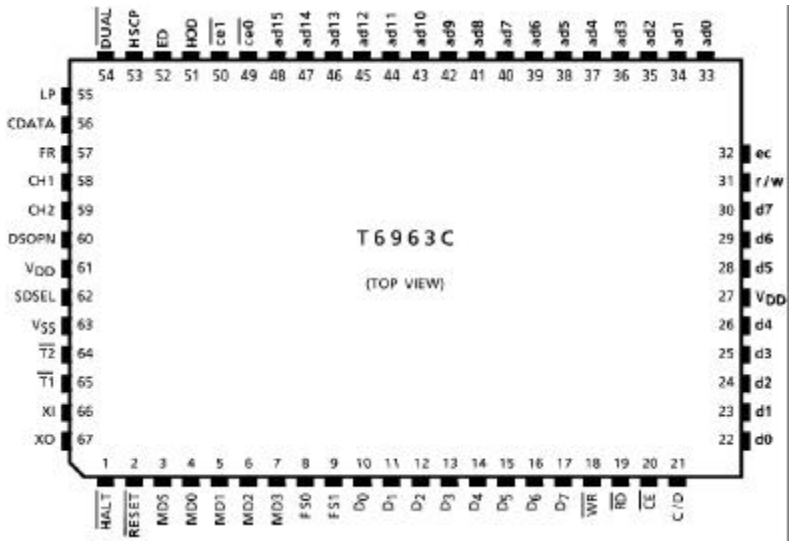

图 5. 21 T6963C **引脚图**

另外，针对国内用户对汉字显示的需求，还有一些内部集成有汉字库的点阵式图形液晶驱动控制器，比如 ST7920。

5.3 点阵式图形液晶模块及驱动程序

由于点阵式液晶显示屏的引脚较多，生产厂家通常会将液晶显示屏和驱动电路装配在一起，形成液晶显示模块，即 LCD Module，**简称** LCM。LCM **是液晶显示产品的另外一种商品形式，它是将液晶显示器件、连接件、驱动和控制集成电路(有些液晶模块需要外接液晶控制器)、**PCB **线路板、背光源、结构件装配在一起的组件。液晶显示模块在很大程度上方便了用户的使用，用户只要将其与微控制器连接，即可进行图形的显示输出控制。**

这里列举几种常用的点阵式图形液晶模块及驱动程序。

5.3.1 SMG240128A 点阵图形液晶模块

SMG240128A **点阵图形液晶模块的点像素为** 240×128 **点，黑色字/白色底，**STN **液晶屏，**

视角为 6:00，内嵌控制器为东芝公司的 T6963C，外部显示存储器为 32K 字节，模块的电路原理框图如图 5. 22 所示。

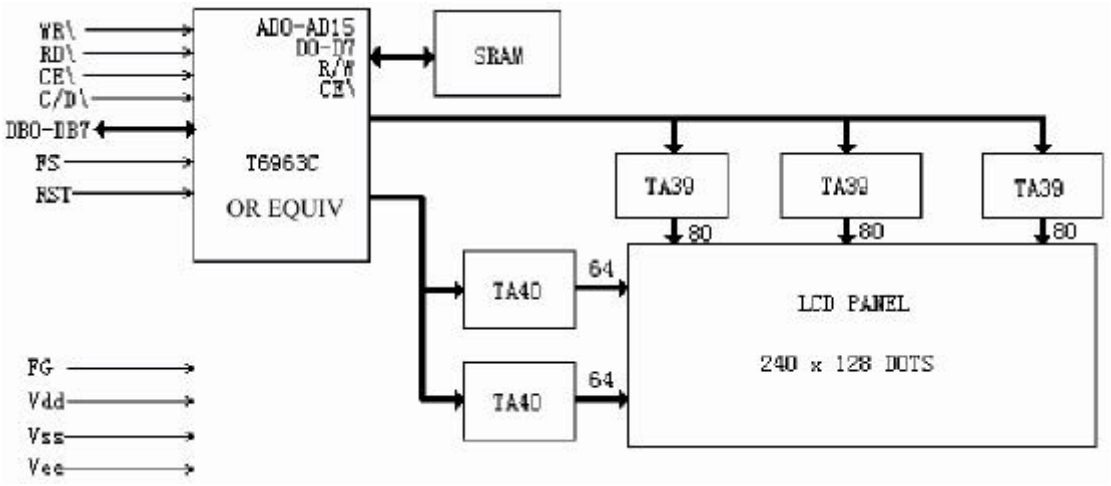

图 5. 22 SMG240128A 点阵图形液晶模块原理框图

EasyARM2200 开发板可以直接支持 SMG240128A 点阵图形液晶模块或相兼容的液晶模块，应用连接电路如图 5. 23 所示。采用 8 位总线方式连接，SMG240128A 点阵图形液晶模块的没有地址总线，显示地址和显示数据均通过 DB0～DB7 接口实现，由于模块工作电源是 5V，而 LPC2210 的 I/O 电压为 3.3V，所以在总线上串接 470Ω保护电阻。图形液晶模块的 C/D 与 A1 连接，用于控制模块处理数据/命令，将 C/D 与 A1 连接有一个好处，就是 LPC2210 可以使用 16 位总线方式操作该图形液晶模块(高 8 位数据被忽略)。模块的片选信号由 LPC2210 的 A22 和外部存储器 BANK3 片选 CS3 相“或”后得到，当 A22 和 nCS3 同时为 0 时，模块被选中，所以其数据操作地址为 0x83000000，命令操作地址为 0x83000002。

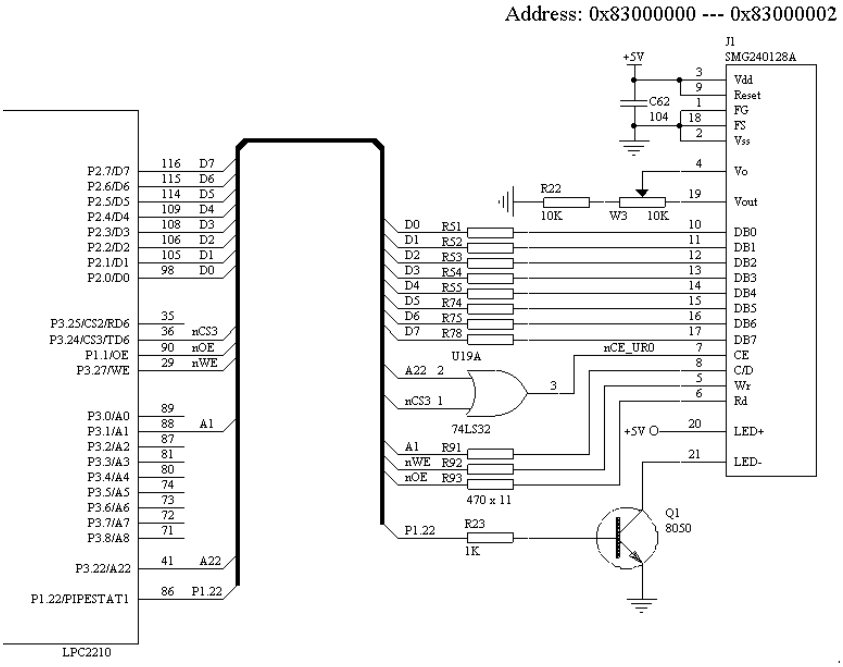

图 5. 23 SMG240128A 点阵图形液晶模块应用连接电路

使用 LPC2210 的总线对 SMG240128A 点阵图形液晶模块操作控制前，先要设置芯片的外部存储器控制器(EMC)，如程序清单 5. 1 所示。SMG240128A 点阵图形液晶模块驱动程序

如程序清单 5. 2 **所示，**ADS1.2 **集成开发环境、系统时钟** Fcclk=44.2368MHz **条件下通过。**

程序清单 5. 1 **存储器接口** BANK3 **总线配置—**SMG240128A

```
…
LDR     R0, =BCFG3                  ; 设置 BCFG3 寄存器
LDR     R1, =0x10000CA0
STR     R1, [R0]
…
```

程序清单 5. 2 SMG240128A **点阵图形液晶模块驱动程序**

```
/*****************************************************************************************
* 文件名：LCDDRIVE.C
* 功能： 图形液晶 240*128 驱动(型号为 SMG240128A)。32K 显示存，0000H-7FFFH 地址。
*        显示是横向字节，高位在前。
* 说明：图形液晶采用 T6963C 液晶控制芯片，单 5 伏供电，并行接口(使用 LPC2210 驱动)。
* 硬件连接： D0--D7   <===>   D0--D7
*            /WR      <===>   nWE
*            /RD      <===>   nOE
*            /CE      <===>   nCS3_1
*            C/D      <===>   A1
*
*            /RST     <===>   VCC
*****************************************************************************************/
#include   "config.h"

TCOLOR   gui_disp_buf[][GUI_LCM_XMAX/8];          // 声明 GUI 显示缓冲区

/* 定义 LCM 地址 */
#define   TG240128_COM        (*((volatile unsigned short *) 0x83000002))
#define   TG240128_DAT        (*((volatile unsigned short *) 0x83000000))

/*****************************************************************************
* 名称：LCD_WriteCommand()
* 功能：写命令子程序。(发送命令前，不检查液晶模块的状态)
* 入口参数：command     要写入 LCM 的命令字
* 出口参数：无
* 说明：函数会设置 LCM 数据总线为输出方式
*****************************************************************************/
#define    LCD_WriteCommand(command)   TG240128_COM = (uint16)command



/*****************************************************************************
* 名称：LCD_WriteData()
* 功能：写数据子程序。(发送数据前，不检查液晶模块的状态)
```

```
* 入口参数：dat       要写入 LCM 的数据
* 出口参数：无
* 说明：函数会设置 LCM 数据总线为输出方式
*******************************************************************************/
#define    LCD_WriteData(dat)    TG240128_DAT = (uint16)dat



/*******************************************************************************
* 名称：LCD_ReadState()
* 功能：读取状态字子程序。
* 入口参数：无
* 出口参数：返回值即为读出的状态字
* 说明：函数会设置 LCM 数据总线为输入方式
*******************************************************************************/
#define    LCD_ReadState()  TG240128_COM



/*******************************************************************************
* 名称：LCD_ReadData()
* 功能：读取数据子程序。
* 入口参数：无
* 出口参数：返回值即为读出的数据
* 说明：函数会设置 LCM 数据总线为输入方式
*******************************************************************************/
#define    LCD_ReadData()         TG240128_DAT



/* 以下为 LCM 的驱动层，主要负责发送 T6963 的各种命令，提供设置显示地址等功能，在发送命令前会检测其状态字。带参数命令模式：先参数，后命令；操作模式：先命令，后数据 */

/* T6963C 命令定义 */
#define    LCD_CUR_POS    0x21      /* 光标位置设置(只有设置到有效显示地址并打开显示时才看到) */
#define    LCD_CGR_POS    0x22      /* CGRAM 偏置地址设置(可以增加自已的符号) */
#define    LCD_ADR_POS    0x24      /* 地址指针位置(设置读写操作指针) */

#define    LCD_TXT_STP    0x40      /* 文本区首址(从此地址开始向屏幕左上角显示字符) */
#define    LCD_TXT_WID    0x41      /* 文本区宽度(设置显示宽度，N/6 或 N/8，其中 N 为 x 轴的点数) */
#define    LCD_GRH_STP    0x42      /* 图形区首址(从此地址开始向屏幕左上角显示点) */
#define    LCD_GRH_WID    0x43      /* 图形区宽度(设置显示宽度，N/6 或 N/8，其中 N 为 x 轴的点数) */

#define    LCD_MOD_OR         0x80      /* 显示方式：逻辑或 */
#define    LCD_MOD_XOR        0x81      /* 显示方式：逻辑异或 */
#define    LCD_MOD_AND        0x82      /* 显示方式：逻辑与 */
#define    LCD_MOD_TCH        0x83      /* 显示方式：文本特征 */
```

```
#define   LCD_DIS_SW          0x90      /* 显示开关：D0=1/0，光标闪烁启用/禁用 */
                                        /*     D1=1/0，光标显示启用/禁用 */
                                        /*     D2=1/0，文本显示启用/禁用(打开后再使用) */
                                        /*     D3=1/0，图形显示启用/禁用(打开后再使用) */

#define   LCD_CUR_SHP         0xA0      /* 光标形状选择：0xA0-0xA7 表示光标占的行数 */

#define   LCD_AUT_WR          0xB0      /* 自动写设置 */
#define   LCD_AUT_RD          0xB1      /* 自动读设置 */
#define   LCD_AUT_OVR         0xB2      /* 自动读/写结束 */

#define   LCD_INC_WR          0xC0      /* 数据一次写，地址加 1 */
#define   LCD_INC_RD          0xC1      /* 数据一次读，地址加 1 */
#define   LCD_DEC_WR          0xC2      /* 数据一次写，地址减 1 */
#define   LCD_DEC_RD          0xC3      /* 数据一次读，地址减 1 */
#define   LCD_NOC_WR          0xC4      /* 数据一次写，地址不变 */
#define   LCD_NOC_RD          0xC5      /* 数据一次读，地址不变 */

#define   LCD_SCN_RD          0xE0      /* 屏读 */

#define   LCD_SCN_CP          0xE8      /* 屏拷贝 */

#define   LCD_BIT_OP          0xF0      /* 位操作：D0-D2--定义 D0-D7 位，D3--1 为置位，0 为清除 */

/*****************************************************************************
* 名称：LCD_TestStaBit01()
* 功能：判断读写指令和读写数据是否允许。
* 入口参数：无
* 出口参数：返回 0 表示禁止，否则表示允许
*****************************************************************************/
uint8   LCD_TestStaBit01(void)
{   uint8 i;

    for(i=100; i>0; i--)
    {   if( (LCD_ReadState()&0x03)==0x03 ) break;
    }

    return(i);
}


/*****************************************************************************
* 名称：LCD_TestStaBit3()
```

```
* 功能：数据自动写状态是否允许。
* 入口参数：无
* 出口参数：返回 0 表示禁止，否则表示允许
*****************************************************************************/
uint8   LCD_TestStaBit3(void)
{   uint8 i;

    for(i=100; i>0; i--)
    {   if( (LCD_ReadState()&0x08)==0x08 ) break;
    }

    return(i);
}



/*****************************************************************************
* 名称：LCD_WriteTCommand1()
* 功能：写无参数命令子程序。会先判断 LCM 状态字。
* 入口参数：command          要写入 LCM 的命令字
* 出口参数：操作出错返回 0，否则返回 1
*****************************************************************************/
uint8   LCD_WriteTCommand1(uint8 command)
{   if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteCommand(command);          // 发送命令字

    return(1);
}



/*****************************************************************************
* 名称：LCD_WriteTCommand3()
* 功能：写双参数命令子程序。会先判断 LCM 状态字。
* 入口参数：command     要写入 LCM 的命令字
*           dat1        参数 1
*           dat2        参数 2
* 出口参数：操作出错返回 0，否则返回 1
* 说明：先发送两字节参数据数据，再发送命令字
*****************************************************************************/
uint8   LCD_WriteTCommand3(uint8 command, uint8 dat1, uint8 dat2)
{   if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteData(dat1);                    // 发送数据 1

    if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteData(dat2);                    // 发送数据 2
```

```
    if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteCommand(command);          // 发送命令字

    return(1);
}


/*****************************************************************************
* 名称：LCD_WriteTCommand2()
* 功能：写单参数命令子程序。会先判断 LCM 状态字。
* 入口参数：command      要写入 LCM 的命令字
*           dat1         参数 1
* 出口参数：操作出错返回 0，否则返回 1
* 说明：先发送参数据数据，再发送命令字
*****************************************************************************/
uint8   LCD_WriteTCommand2(uint8 command, uint8 dat1)
{   if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteData(dat1);                // 发送数据 1

    if( LCD_TestStaBit01()==0 ) return(0);
    LCD_WriteCommand(command);          // 发送命令字

    return(1);
}


/*****************************************************************************
* 名称：LCD_WriteTData1()
* 功能：写 1 字节数据子程序。会先判断状态字。
* 入口参数：dat          要写入 LCM 的数据
* 出口参数：操作出错返回 0，否则返回 1
*****************************************************************************/
uint8   LCD_WriteTData1(uint8 dat)
{   if( LCD_TestStaBit3()==0 ) return(0);
    LCD_WriteData(dat);                 // 发送命令字

    return(1);
}


/* 以下为 LCM 的用户接口层，主要负责解释用户命令，并发送到 LCM，为用户编程提供接口 */

/*****************************************************************************
```

```
* 名称：LCD_Initialize()
* 功能：LCM 初始化，将 LCM 初始化为纯图形模式，显示起始地址为 0x0000，。
* 入口参数：无
* 出口参数：无
* 说明：函数会设置 LCM 数据总线为输出方式
*****************************************************************************/
void    LCD_Initialize(void)
{   LCD_WriteTCommand3(LCD_TXT_STP, 0x00, 0x00);     // 设置文本方式 RAM 起始地址
    LCD_WriteTCommand3(LCD_TXT_WID, 30, 0x00);       // 设置文本模式的宽度，宽度为 N/6 或 N/8，
                                                     // N 为宽度点数，如 240
    LCD_WriteTCommand3(LCD_GRH_STP, 0x00, 0x00);     // 设置图形方式 RAM 起始地址
    LCD_WriteTCommand3(LCD_GRH_WID, 30, 0x00);       // 设置图形模式的宽度，宽度为 N/6 或 N/8，
                                                     // N 为宽度点数，如 240
    LCD_WriteTCommand1(LCD_MOD_OR);                  // 设置显示方式为"或"
    LCD_WriteTCommand1(LCD_DIS_SW|0x08);             // 设置纯图形显示模式
}


/*****************************************************************************
* 名称：LCD_FillAll()
* 功能：LCD 填充。以图形方式进行填充，起始地址为 0x0000。
* 入口参数：dat             要填充的数据
* 出口参数：无
*****************************************************************************/
void    LCD_FillAll(uint8 dat)
{   uint32    i;

    LCD_WriteTCommand3(LCD_ADR_POS, 0x00, 0x00);    // 置地址指针
    LCD_WriteTCommand1(LCD_AUT_WR);                 // 自动写
    for(i=0;i<128*30;i++)
    {   LCD_WriteTData1(dat);                       // 写数据
    }
    LCD_WriteTCommand1(LCD_AUT_OVR);                // 自动写结束
    LCD_WriteTCommand3(LCD_ADR_POS,0x00,0x00);      // 重置地址指针
}


/*****************************************************************************
* 名称：LCD_UpdatePoint()
* 功能：在指定位置上画点，刷新某一点。
* 入口参数：  x             指定点所在列的位置
*             y             指定点所在行的位置
* 出口参数：返回值为 1 时表示操作成功，为 0 时表示操作失败。
* 说明：操作失败原因是指定地址超出缓冲区范围。
```

```
*********************************************************************************/
void    LCD_UpdatePoint(uint32 x, uint32 y)
{   uint32    addr;

    /* 找出目标地址 */
    addr = y*(GUI_LCM_XMAX>>3) + (x>>3);
    LCD_WriteTCommand3(LCD_ADR_POS, addr&0xFF, addr>>8);      // 置地址指针

    /* 输出数据 */
    LCD_WriteTCommand2(LCD_INC_WR, gui_disp_buf[y][x>>3]);
}



/*********************************************************************************
* 名称：LCD_UpdateSCR()
* 功能：LCM 全屏刷新，即把全部显示缓冲区的数据输出到 LCM 的显示 RAM。
* 入口参数：无
* 出口参数：无
*********************************************************************************/
void    LCD_UpdateSCR(void)
{   uint32    i, j;

    /* 开始复制数据 */
    LCD_WriteTCommand3(LCD_ADR_POS, 0x00, 0x00);          // 置地址指针
    LCD_WriteTCommand1(LCD_AUT_WR);                       // 自动写
    for(i=0; i<GUI_LCM_YMAX; i++)                         // 历遍所有行
    {   for(j=0; j<GUI_LCM_XMAX/8; j++)                   // 历遍所有行
        {   LCD_WriteTData1(gui_disp_buf[i][j]);
         }
    }
    LCD_WriteTCommand1(LCD_AUT_OVR);                      // 自动写结束
}
```

5.3.2 MG12864 点阵式图形液晶模块

MG12864 点阵图形液晶模块的点像素为 128×64 点，黑色字/白色底，STN 液晶屏，视角为 6:00，内嵌控制器为 KS0107/ KS0108，模块的电路原理框图如图 5. 24 所示。

EasyARM2200 开发板可以通过外设 PACK 来支持 MG12864 点阵图形液晶模块或相兼容的液晶模块，应用连接电路如图 5. 25 所示。采用 8 位总线方式连接，MG12864 点阵图形液晶模块的没有地址总线，显示地址和显示数据均通过 D0～D7 接口实现，由于模块工作电源是 5V，而 LPC2210 的 I/O 电压为 3.3V，所以在总线上串接 470Ω 保护电阻。图形液晶模块的 $D/\overline{I}$ 引脚与 A1 连接，用于控制模块处理数据/命令，将 $D/\overline{I}$ 引脚与 A1 连接有一个好处，就是 LPC2210 可以使用 16 位总线方式操作该图形液晶模块(高 8 位数据被忽略)。模块的片选信号由 LPC2210 的 A23、A22、A21 和外部存储器 BANK2 通过 74HC138 设码后得

到的，由于模块使用了两片 KS0108 驱动控制器分别控制液晶的左右半屏，所以需要 LCM_CS1、LCM_CS2 两个选通信号；由于 KS0108 的片选是高电平有效，而 74HC138 的译码结果是低电平，所以需要使用 74HC04 将信号反相。当 A23、A22、A21 和 nCS2 同时为 0 时，LCM_CS1 变为高电平，模块左半屏的驱动控制器被选中，所以其数据操作地址为 0x82000000，命令操作地址为 0x82000002；当 A23、A22 和 nCS2 同时为 0，A21 为 1 时，LCM_CS2 变为高电平，模块右半屏的驱动控制器被选中，所以其数据操作地址为 0x82200000，命令操作地址为 0x82200002。

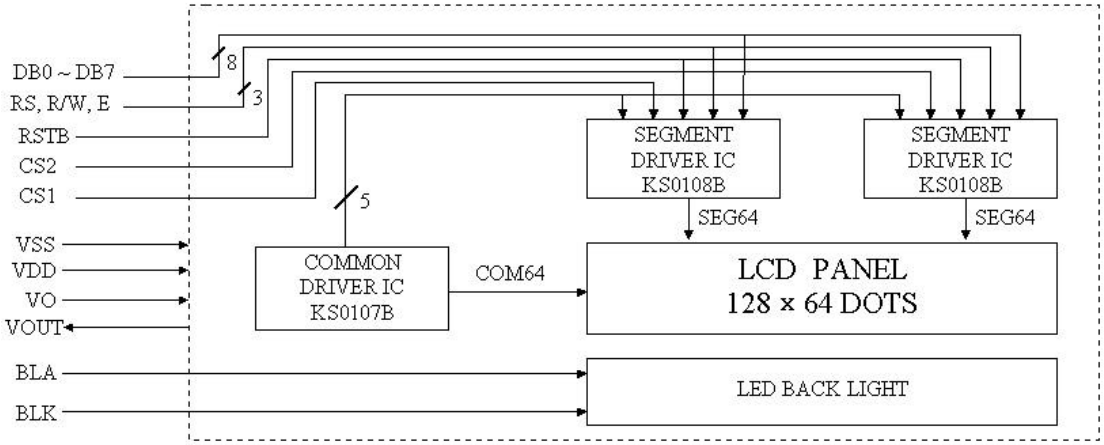

图 5. 24 MG12864 点阵图形液晶模块原理框图

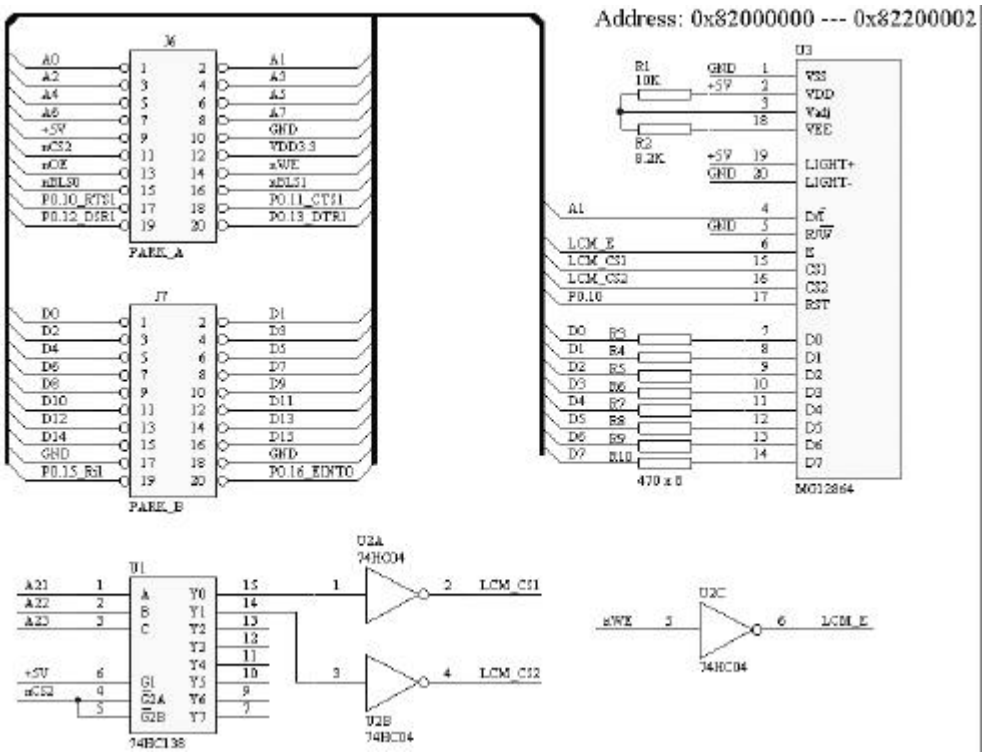

图 5. 25 MG12864 点阵图形液晶模块应用连接电路

使用 LPC2210 的总线对 MG12864 点阵图形液晶模块操作控制前，先要设置芯片的外部存储器控制器(EMC)，如程序清单 5. 3 所示，由于 MG12864 的速度较慢，所以把总线配置为最慢的时序。MG12864 点阵图形液晶模块驱动程序如程序清单 5. 4 所示，驱动程序对应

的头文件如程序清单 5. 5 **所示**，ADS1.2 **集成开发环境、系统时钟** Fcclk=22.1184MHz **条件下通过**。

程序清单 5. 3 **存储器接口** BANK2 **总线配置**—MG12864

```
…
LDR     R0, =BCFG2
LDR     R1, =0x0000ffef
STR     R1, [R0]
…
```

程序清单 5. 4 MG12864 **点阵图形液晶模块驱动程序**

```
/*****************************************************************************
* 文件名：LCM_DRIVE.C
* 功能：图形液晶 MG12864 驱动程序。
* 说明：在 LCM_DRIVE.H 文件中定义了 LCM 操作地址。
*       左半屏的写命令操作地址为 0x82000000，写数据操作地址为 0x82000002；
*       右半屏的写命令操作地址为 0x82200000，写数据操作地址为 0x82200002。
*       由于 GRAPHICS.C 中使用了 disp_buf 作为作图缓冲区，所以 LCM_Write
*       Byte()、LCM_DispFill()均要更新 disp_buf。
*****************************************************************************/
#include   "config.h"

/* LCM 复位控制脚定义 */
#define          LCM_RST       (1<<10)              /* P0.10 用于控制复位 */

/* 定义显示缓冲 */
uint8   disp_buf[LCM_YMAX/8][LCM_XMAX];

/*****************************************************************************
* 名称：LCM_Wr1Command()
* 功能：写命令子程序，所选屏为左半屏(CS1)。
* 入口参数：command     要写入 LCM 的命令字
*****************************************************************************/
#define     LCM_Wr1Command(command)          LCMCS1W_COM = command

/*****************************************************************************
* 名称：LCM_Wr2Command()
* 功能：写命令子程序，所选屏为右半屏(CS2)。
* 入口参数：command     要写入 LCM 的命令字
*****************************************************************************/
#define     LCM_Wr2Command(command)          LCMCS2W_COM = command

/*****************************************************************************
* 名称：LCM_Wr1Data()
```

```
* 功能：写数据子程序，所选屏为左半屏(CS1)。
* 入口参数：wrdata          要写入 LCM 的数据
*********************************************************************************/
#define LCM_Wr1Data(wrdata)          LCMCS1W_DAT = wrdata

/*********************************************************************************
* 名称：LCM_Wr2Data()
* 功能：写数据子程序，所选屏为右半屏(CS2)。
* 入口参数：wrdata          要写入 LCM 的数据
*********************************************************************************/
#define LCM_Wr2Data(wrdata)          LCMCS2W_DAT = wrdata



/*********************************************************************************
* 名称：LCM_DispIni()
* 功能：LCM 显示初始化。使能显示，设置显示起始行为 0 并清屏。
* 入口参数：无
* 出口参数：无
*********************************************************************************/
void   LCM_DispIni(void)
{   uint32   i;

    IO0DIR = LCM_RST;

    IO0CLR = LCM_RST;                          // 复位驱动芯片
    for(i=0; i<5000; i++);
    IO0SET = LCM_RST;

    LCM_Wr1Command(LCM_DISPON);          // 打开显示
    LCM_Wr1Command(LCM_STARTROW);      // 设置显示起始行为 0
    LCM_Wr2Command(LCM_DISPON);
    LCM_Wr2Command(LCM_STARTROW);
    LCM_DispClr();                             // 清屏

    LCM_Wr1Command(LCM_ADDRSTRY+0);   // 设置页(行)地址
    LCM_Wr1Command(LCM_ADDRSTRX+0);   // 设置列地址，即列
    LCM_Wr2Command(LCM_ADDRSTRY+0);
    LCM_Wr2Command(LCM_ADDRSTRX+0);
}



/*********************************************************************************
* 名称：LCM_WriteByte()
* 功能：向指定点写数据(一字节)。
```

```
* 入口参数：x              x 坐标值(0-127)
*           y              y 坐标值(0-63)
*           wrdata         所要写的数据
* 出口参数：无
* 说明：会更新 disp_buf 相应存储单元
*******************************************************************************/
void   LCM_WriteByte(uint8 x, uint8 y, uint8 wrdata)
{   x = x&0x7f;                          // 参数过滤
    y = y&0x3f;

    y = y>>3;
    disp_buf[y][x] = wrdata;
    if(x<64)                             // 选择液晶控制芯片(即 CS1--控制前 64 个点，CS2--控制后 64 个点)
    {   LCM_Wr1Command(LCM_ADDRSTRX+x);     // 设置当前列地址，即 x 坐标
        LCM_Wr1Command(LCM_ADDRSTRY+y);     // 设置当前页地址，即 y 坐标
        for(x=0; x<10; x++);             // 短延时

        LCM_Wr1Data(wrdata);
    }
    else
    {   x = x-64;                        // 调整 x 变量值
        LCM_Wr2Command(LCM_ADDRSTRX+x);
        LCM_Wr2Command(LCM_ADDRSTRY+y);
        for(x=0; x<10; x++);

        LCM_Wr2Data(wrdata);
    }
}



/*******************************************************************************
* 名称：LCM_DispFill()
* 功能：向显示屏填充数据
* 入口参数：filldata        要写入 LCM 的填充数据
* 出口参数：无
* 说明：会更新 disp_buf 相应存储单元
*******************************************************************************/
void   LCM_DispFill(uint8 filldata)
{   uint8   x, y;
    uint8   i;

    LCM_Wr1Command(LCM_STARTROW);            // 设置显示起始行为 0
    LCM_Wr2Command(LCM_STARTROW);
```

```
    for(y=0; y<8; y++)
    {   LCM_Wr1Command(LCM_ADDRSTRY+y);      // 设置页(行)地址
        LCM_Wr1Command(LCM_ADDRSTRX);        // 设置列地址
        LCM_Wr2Command(LCM_ADDRSTRY+y);
        LCM_Wr2Command(LCM_ADDRSTRX);
        for(i=0; i<10; i++);

        for(x=0; x<64; x++)
        {   LCM_Wr1Data(filldata);
            LCM_Wr2Data(filldata);
            disp_buf[y][x] = filldata;
            disp_buf[y][x+64] = filldata;
        }
    }
}


/* ASCII 码对应的点阵数据表 */
uint8 const    ASCII_TAB20[80] = {
                            0x00, 0x00, 0x00, 0x00, 0x00,
                            0x00, 0x00, 0x9e, 0x00, 0x00,
                            0x00, 0x0e, 0x00, 0x0e, 0x00,
                             0x28, 0xfe, 0x28, 0xfe, 0x28,
                             0x48, 0x54, 0xfe, 0x54, 0x24,
                             0x46, 0x26, 0x10, 0xc8, 0xc4,
                             0x6c, 0x92, 0xaa, 0x44, 0xa0,
                             0x00, 0x0a, 0x06, 0x00, 0x00,
                            0x00, 0x38, 0x44, 0x82, 0x00,
                            0x00, 0x82, 0x44, 0x38, 0x00,
                            0x28, 0x10, 0x7c, 0x10, 0x28,
                            0x10, 0x10, 0x7c, 0x10, 0x10,
                            0x00, 0xa0, 0x60, 0x00, 0x00,
                            0x10, 0x10, 0x10, 0x10, 0x10,
                            0x00, 0xc0, 0xc0, 0x00, 0x00,
                            0x40, 0x20, 0x10, 0x08, 0x04
                                    };

uint8 const    ASCII_TAB30[80] = {
                            0x7C, 0xA2, 0x92, 0x8A, 0x7C,
                            0x00, 0x84, 0xFE, 0x80, 0x00,
                            0x84, 0xC2, 0xA2, 0x92, 0x8C,
                            0x42, 0x82, 0x8A, 0x96, 0x62,
                            0x30, 0x28, 0x24, 0xFE, 0x20,
                            0x4E, 0x8A, 0x8A, 0x8A, 0x72,
```

```
                        0x78, 0x94, 0x92, 0x92, 0x60,
                        0x02, 0xE2, 0x12, 0x0A, 0x06,
                        0x6C, 0x92, 0x92, 0x92, 0x6C,
                        0x0C, 0x92, 0x92, 0x52, 0x3C,
                        0x00, 0x6C, 0x6C, 0x00, 0x00,
                        0x00, 0xAC, 0x6C, 0x00, 0x00,
                        0x10, 0x28, 0x44, 0x82, 0x00,
                        0x28, 0x28, 0x28, 0x28, 0x28,
                        0x00, 0x82, 0x44, 0x28, 0x10,
                        0x04, 0x02, 0xA2, 0x12, 0x0C
                                };

uint8 const    ASCII_TAB40[80] = {
                        0x64, 0x92, 0xF2, 0x82, 0x7C,
                        0xFC, 0x22, 0x22, 0x22, 0xFC,
                        0xFE, 0x92, 0x92, 0x92, 0x6C,
                        0x7C, 0x82, 0x82, 0x82, 0x44,
                        0xFE, 0x82, 0x82, 0x44, 0x38,
                        0xFE, 0x92, 0x92, 0x92, 0x82,
                        0xFE, 0x12, 0x12, 0x12, 0x02,
                        0x7C, 0x82, 0x92, 0x92, 0xF4,
                        0xFE, 0x10, 0x10, 0x10, 0xFE,
                        0x00, 0x82, 0xFE, 0x82, 0x00,
                        0x40, 0x80, 0x82, 0x7E, 0x02,
                        0xFE, 0x10, 0x28, 0x44, 0x82,
                        0xFE, 0x80, 0x80, 0x80, 0x80,
                        0xFE, 0x04, 0x18, 0x04, 0xFE,
                        0xFE, 0x08, 0x10, 0x20, 0xFE,
                        0x7C, 0x82, 0x82, 0x82, 0x7C
                                };

uint8 const    ASCII_TAB50[80] = {
                        0xFE, 0x12, 0x12, 0x12, 0x0C,
                        0x7C, 0x82, 0xA2, 0x42, 0xBC,
                        0xFE, 0x12, 0x32, 0x52, 0x8C,
                        0x8C, 0x92, 0x92, 0x92, 0x62,
                        0x02, 0x02, 0xFE, 0x02, 0x02,
                        0x7E, 0x80, 0x80, 0x80, 0x7E,
                        0x3E, 0x40, 0x80, 0x40, 0x3E,
                        0x7E, 0x80, 0x70, 0x80, 0x7E,
                        0xC6, 0x28, 0x10, 0x28, 0xC6,
                        0x0E, 0x10, 0xE0, 0x10, 0x0E,
                        0xC2, 0xA2, 0x92, 0x8A, 0x86,
                        0x00, 0xFE, 0x82, 0x82, 0x00,
```

```
                        0x04, 0x08, 0x10, 0x20, 0x40,
                        0x00, 0x82, 0x82, 0xFE, 0x00,
                        0x08, 0x04, 0x02, 0x04, 0x08,
                        0x80, 0x80, 0x80, 0x80, 0x80
                                };

uint8 const    ASCII_TAB60[80] = {
                        0x00, 0x02, 0x04, 0x08, 0x00,
                        0x40, 0xA8, 0xA8, 0xA8, 0xF0,
                        0xFE, 0x90, 0x88, 0x88, 0x70,
                        0x70, 0x88, 0x88, 0x88, 0x40,
                        0x70, 0x88, 0x88, 0x90, 0xFE,
                        0x70, 0xA8, 0xA8, 0xA8, 0x30,
                        0x10, 0xFC, 0x12, 0x02, 0x04,
                        0x18, 0xA4, 0xA4, 0xA4, 0x7C,
                        0xFE, 0x10, 0x08, 0x08, 0xF0,
                        0x00, 0x88, 0xFA, 0x80, 0x00,
                        0x40, 0x80, 0x88, 0x7A, 0x00,
                        0xFE, 0x20, 0x50, 0x88, 0x00,
                        0x00, 0x82, 0xFE, 0x80, 0x00,
                        0xF8, 0x08, 0x30, 0x08, 0xF8,
                        0xF8, 0x10, 0x08, 0x08, 0xF0,
                        0x70, 0x88, 0x88, 0x88, 0x70
                                };

uint8 const    ASCII_TAB70[80] = {
                        0xF8, 0x28, 0x28, 0x28, 0x10,
                        0x10, 0x28, 0x28, 0x30, 0xF8,
                        0xF8, 0x10, 0x08, 0x08, 0x10,
                        0x90, 0xA8, 0xA8, 0xA8, 0x40,
                        0x08, 0x7E, 0x88, 0x80, 0x40,
                        0x78, 0x80, 0x80, 0x40, 0xF8,
                        0x38, 0x40, 0x80, 0x40, 0x38,
                        0x78, 0x80, 0x60, 0x80, 0x78,
                        0x88, 0x50, 0x20, 0x50, 0x88,
                        0x18, 0xA0, 0xA0, 0xA0, 0x78,
                        0x88, 0xC8, 0xA8, 0x98, 0x88,
                        0x00, 0x10, 0x6C, 0x82, 0x00,
                        0x00, 0x00, 0xFE, 0x00, 0x00,
                        0x00, 0x82, 0x6C, 0x10, 0x00,
                        0x10, 0x10, 0x54, 0x38, 0x10,
                        0x10, 0x38, 0x54, 0x10, 0x10
                                };
```

```
/*****************************************************************************
* 名称：LCM_DispChar()
* 功能：指定地址显示字符。
* 入口参数：disp_cy              显示行值(0-7)
*           disp_cx              显示列值(0-15)
*           dispdata             所要显示的字符(ASCII 码)
* 注：支持显示字符 0-9、A-Z、a-z 及空格，字符显示格式为 5*7，模为 8*8，所以
*     屏幕显示为 8*16(共 8 行，每行 16 个字符)。
*****************************************************************************/
void  LCM_DispChar(uint8 disp_cy, uint8 disp_cx, char dispdata)
{  uint8  const  *pchardata;
   uint8  i;

   disp_cy = disp_cy&0x07;              // 参数过滤
   disp_cx = disp_cx&0x0f;

  /* 先要找出显示数据的类型，并设置相应的点阵数据表，然后设置指针，以取得对应的点阵数据。 */
   switch(dispdata&0xf0)
   {  case 0x20:
            dispdata = (dispdata&0x0f)*5;
             pchardata = &ASCII_TAB20[dispdata];
             break;

      case 0x30:
            dispdata = (dispdata&0x0f)*5;
             pchardata = &ASCII_TAB30[dispdata];
             break;

      case 0x40:
            dispdata = (dispdata&0x0f)*5;
             pchardata = &ASCII_TAB40[dispdata];
             break;

      case 0x50:
            dispdata = (dispdata&0x0f)*5;
             pchardata = &ASCII_TAB50[dispdata];
             break;

      case 0x60:
            dispdata = (dispdata&0x0f)*5;
             pchardata = &ASCII_TAB60[dispdata];
             break;
```

```
        case 0x70:
                dispdata = (dispdata&0x0f)*5;
                 pchardata = &ASCII_TAB70[dispdata];
                 break;

        default:
                pchardata = &ASCII_TAB20[0];
                break;
    } // end of switch(dispdata&0xf0)...


    if( (disp_cx&0x08) == 0 )       // 选择液晶控制芯片(即 CS1--控制前 8 个字符，CS2--控制后 8 个字符)
    {   i = disp_cx<<3;
        LCM_Wr1Command(LCM_ADDRSTRX+i);                 // 设置当前列地址，即列
        LCM_Wr1Command(LCM_ADDRSTRY+disp_cy);           // 设置当前页地址，即行
        for(i=0; i<10; i++);

        LCM_Wr1Data(0x00);                              // 显示一列空格
        for(i=0; i<5; i++)
        {   LCM_Wr1Data(*pchardata);                    // 发送数据
            */
            pchardata++;
        }
        LCM_Wr1Data(0x00);
        LCM_Wr1Data(0x00);
    }
    else
    {   i = (disp_cx&0x07)<<3;    // 若 Y>7,则选取用 CS2 并且地址值要先减去 8，再乘以 8
        LCM_Wr2Command(LCM_ADDRSTRX+i);
        LCM_Wr2Command(LCM_ADDRSTRY+disp_cy);   // 设置当前页地址，即行
        for(i=0; i<10; i++);

        LCM_Wr2Data(0x00);                              // 显示一列空格
        for(i=0; i<5; i++)
        {   LCM_Wr2Data(*pchardata);                    // 发送数据
            pchardata++;
        }
        LCM_Wr2Data(0x00);
        LCM_Wr2Data(0x00);
    }
}


/*************************************************************************
```

```
* 名称：LCM_DispStr()
* 功能：字符串显示输出。
* 入口参数：disp_cy              显示起始行(0-7)
*           disp_cx              显示起始列(0-15)
*           disp_str             字串指针
* 出口参数：无
* 注：支持显示字符 0-9、A-Z、a-z 及空格，字符显示格式为 5*7，模为 8*8，所以
*     屏幕显示为 8*16(共 8 行，每行 16 个字符)。
*********************************************************************************/
void   LCM_DispStr(uint8 disp_cy, uint8 disp_cx, char *disp_str)
{   while( *disp_str != '\0')
    {   disp_cy = disp_cy&0x07;                          // 参数过滤
        disp_cx = disp_cx&0x0f;
        LCM_DispChar(disp_cy, disp_cx, *disp_str);       // 显示字符

        disp_str++;                                      // 指向下一字符数据
        disp_cx++;
        if(disp_cx>15) disp_cy++;                        // 指向下一显示行
    }
}
```

程序清单 5. 5　MG12864 **点阵图形液晶模块驱动程序头文件**

```
/*********************************************************************************
* 文件名：LCM_DRIVE.H
* 功能：图形液晶 MG12864 驱动程序。(头文件)
*********************************************************************************/
#ifndef     LCMDRIVE_H
#define     LCMDRIVE_H

/* 定义 LCM 像素数宏 */
#define     LCM_XMAX    128
#define     LCM_YMAX    64

 /* 定义 LCM 操作地址 */
#define     LCMCS1W_COM     (*((volatile uint8 *) 0x82000000))
#define     LCMCS1W_DAT     (*((volatile uint8 *) 0x82000002))
#define     LCMCS2W_COM     (*((volatile uint8 *) 0x82200000))
#define     LCMCS2W_DAT     (*((volatile uint8 *) 0x82200002))

/* 定义 LCM 操作的命令字 */
/* 打开 LCM 显示 */
#define     LCM_DISPON          0x3f
```

```
/* 显示起始行 0，可以用 LCM_STARTROW+x 设置起始行。(x<64) */
#define     LCM_STARTROW     0xc0

/* 页起始地址，可以用 LCM_ADDRSTRX+x 设置当前页(即行)。(x<8) */
#define     LCM_ADDRSTRY     0xb8

/* 列起始地址，可以用 LCM_ADDRSTRY+x 设置当前列(即更)。(x<64) */
#define     LCM_ADDRSTRX     0x40

/* 定义宏函数 */
#define     LCM_DispClr()          LCM_DispFill(0x00)          /* 清屏函数 */


/*****************************************************************************
* 名称：LCM_DispIni()
* 功能：LCM 显示初始化
* 入口参数：无
* 出口参数：无
* 注：初化显示后，清屏并设置显示起始行为 0。
*****************************************************************************/
extern void    LCM_DispIni(void);


/*****************************************************************************
* 名称：LCM_WriteByte()
* 功能：向指定点写数据(一字节)。
* 入口参数：x              x 坐标值(0-127)
*           y              y 坐标值(0-63)
*           wrdata         所要写的数据
* 出口参数：无
*****************************************************************************/
extern void    LCM_WriteByte(uint8 x, uint8 y, uint8 wrdata);


/*****************************************************************************
* 名称：LCM_DispFill()
* 功能：向显示屏填充数据
* 入口参数：filldata         要写入 LCM 的填充数据
* 出口参数：无
*****************************************************************************/
extern void    LCM_DispFill(uint8 filldata);


/*****************************************************************************
```

```
* 名称：LCM_DispChar()
* 功能：指定地址显示字符。
* 入口参数：disp_cy          显示行值(0-7)
*           disp_cx          显示列值(0-15)
*           dispdata         所要显示的字符(ASCII 码)
* 注：支持显示字符 0-9、A-Z、a-z 及空格，字符显示格式为 5*7，模为 8*8，所以
*     屏幕显示为 8*16(共 8 行，每行 16 个字符)。
*********************************************************************/
extern void   LCM_DispChar(uint8 disp_cy, uint8 disp_cx, char dispdata);



/*********************************************************************
* 名称：LCM_DispStr()
* 功能：字符串显示输出。
* 入口参数：disp_cy          显示起始行(0-7)
*           disp_cx          显示起始列(0-15)
*           disp_str         字串指针
* 出口参数：无
* 注：支持显示字符 0-9、A-Z、a-z 及空格，字符显示格式为 5*7，模为 8*8，所以
*     屏幕显示为 8*16(共 8 行，每行 16 个字符)。
*********************************************************************/
extern void   LCM_DispStr(uint8 disp_cy, uint8 disp_cx, char *disp_str);



#endif
```

5.3.3 SED1353F 控制伪彩液晶模块

LFUBK909XA 为伪彩类型的点阵图形液晶模块，点像素为 320×3(RGB)×240 点，屏幕尺寸为 5.2 英寸，为 ALPS 公司的产品，内嵌驱动器及电源电路，模块的电路原理框图如图 5. 26 所示。由于该模块没有集成液晶控制器，LPC2210 也没有点阵图形控制器功能，所以需要接一个液晶控制 SED1353F，SED1353F 能够支持 256 色伪彩液晶屏。

EasyARM2200 开发板可以通过外设 PACK 来支持以 SED1353F 为液晶控制器的点阵图形液晶模块，应用连接电路如图 5. 27 所示。采用 8 位总线方式连接，根据 SED1353F 的用户手册说明，DB8～DB15 必须接到 VDD。由于 SED1353F 工作电源是 5V，而 LPC2210 的 I/O 电压为 3.3V，所以在总线上串接 470Ω保护电阻。

在上电复位时，SED1353F 会读取 VD0～VD15 引脚上的状态值，然后根据读取到的值来设置芯片的工作模式，设置说明如下（VD0～VD15 具有内部下拉电阻，若外部不连接上拉电阻，读取得到的状态值为 0）：

VD0 为 0，选用 8 位总线接口；

VD1 为 1，以地址映射方式访问 I/O 寄存器

VD2 为 0，8080 系列 MPU 接口，带 READ 信号

VD3 为 0，不交换高、低字节数据 (16 位总线时)

VD12～VD4 均为 0，即设置 I/O 寄存器基地址(位 9～1)

VD15～VD13 均为 0，即设置选择存储器接口访问基地址(位 3～1)，所以电路上 AB19～

AB17 引脚要接地，以匹配存储器接口访问基地址。

SED1353F 的片选信号由 LPC2210 的 A23、A22、A21 和外部存储器 BANK2 通过 74HC138 设码后得到的，当 A23、A22、A21 和 nCS2 同时为 0 时，IO_CS 变为低电平，选中 SED1353F 控制寄存器，所以其寄存器地址为 0x82000000～0x8200000F；当 A23、A22 和 nCS2 同时为 0，A21 为 1 时，M_CS 变为低电平，选中显示存储器，所以其数据操作地址为 0x82200000～0x8221FFFF。

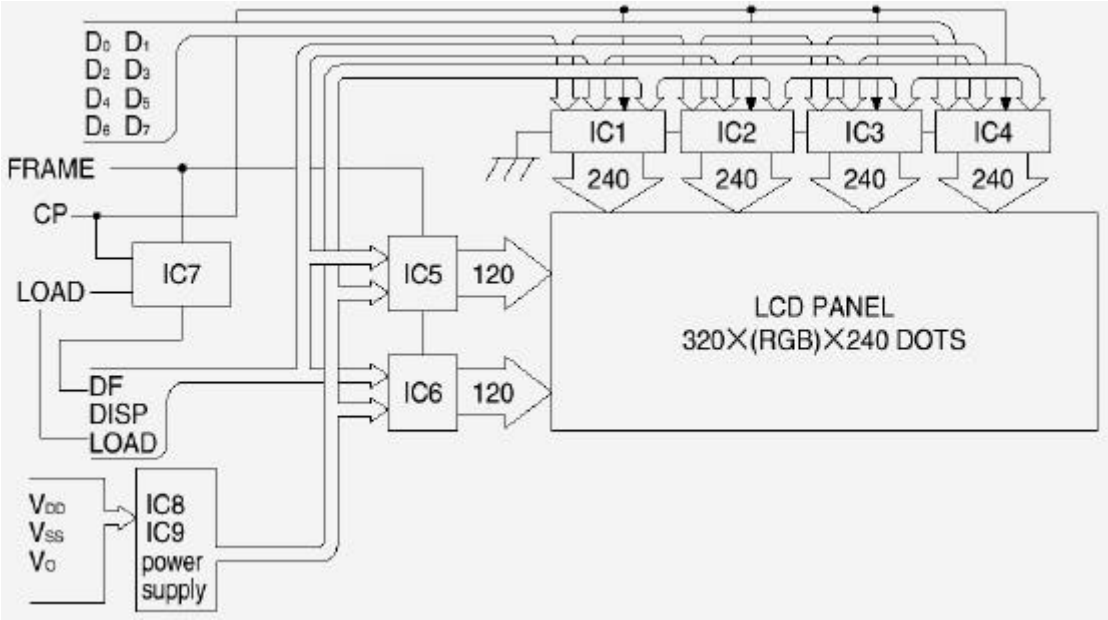

图 5. 26 LFUBK909XA 点阵图形液晶模块原理框图

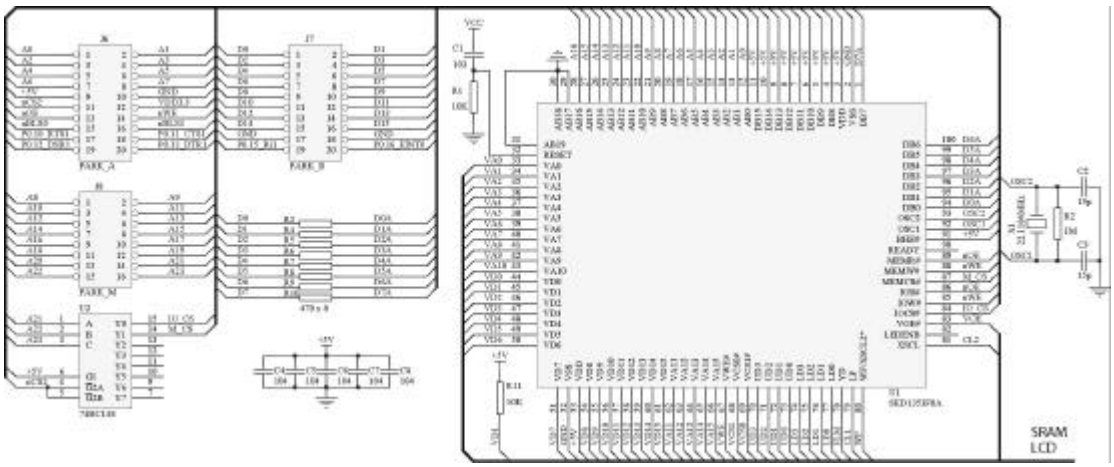

图 5. 27 SED1353F 控制器应用连接电路

使用 LPC2210 的总线对 SED1353F 操作控制前，先要设置芯片的外部存储器控制器(EMC)，如程序清单 5. 6 所示。基于 SED1353F 控制的点阵图形液晶模块 LFUBK909XA 驱动程序如程序清单 5. 7 所示，驱动程序对应的头文件如程序清单 5. 8 所示，ADS1.2 集成开发环境、系统时钟 Fcclk=44.2368MHz 条件下通过。

程序清单 5. 6 存储器接口 BANK2 总线配置—SED1353F

```
            …
            LDR         R0, =BCFG2
            LDR         R1, =0x00009629
            STR         R1, [R0]
            …
```

程序清单 5. 7 **基于** SED1353F **控制的** LFUBK909XA **液晶模块驱动程序**

```
/*********************************************************************************************
* 文件名：LCDDRIVE.C
* 功能：通过操作 SED1353，控制 256 色 RGB 伪彩液晶显示屏。
* 说明：使用 LPC2210 芯片控制，8 位总线接口。
*           SED1353 的寄存器地址为 0x8200000x，显示存储器的地址为 0x822xxxxx。
*********************************************************************************************/
#include    "config.h"

/* 定义显示缓冲区(可根据情况定义或直接使用 LCM 显示存储空间) */
TCOLOR    gui_disp_buf[GUI_LCM_YMAX][GUI_LCM_XMAX];

/* 定义 SED1353 控制地址 */
#define    SED1353_REG          0x82000000
#define    SED1353_DAT          0x82200000

uint8    const    INIT_TAB[14] = {
            0x00,       // 寄存器 0 -- 00000000B (正常工作模式，即非测试模式)
            0xBD,       // 寄存器 1 -- 10111101B (模式控制：打开显示，单屏，屏蔽 XSCL 水平非显示周期输
                        // 出，使能 LCDENB，彩色 8 位 LCD 数据--无 XCSL2，16 位显示存储数据总线方式)
            159,        // 寄存器 2 -- 159 (行字节控制)(每行显示占用的存储器字节数，即列参数)
            0x0E,       // 寄存器 3 -- 00001110B (非节功模式，256 彩色方式)
            239,        // 寄存器 4 -- 239 (总行数)
            0,          // 寄存器 5 -- 0 (WF 翻转周期,为 0 时表示每一帧 WF 输出翻转一次)
            0x00,       // 寄存器 6 -- 00H (第一屏显示 RAM 地址低 8 位)
            0x00,       // 寄存器 7 -- 00H (第一屏显示 RAM 地址高 8 位)
            0x00,       // 寄存器 8 -- 00H (第二屏显示 RAM 地址低 8 位)
            0x00,       // 寄存器 9 -- 00H (第二屏显示 RAM 地址高 8 位)
            239,        // 寄存器 A -- 239 (显示一区占用行数)
            0,          // 寄存器 B -- 0 (显示一区占用行数，高 2 位)
            10,         // 寄存器 C -- 10 (水平默认不显示周期)
            0           // 寄存器 D -- 0 (显示域冗余宽度，正常操作模式)
                                    };
/*********************************************************************************
* 名称：SED1353_Init()
* 功能：初始化 SED1353。320*240 256 色单屏彩屏。
* 入口参数：无
* 出口参数：无
```

```
* 说明：本子程序对 SED1353 寄存器 0--D 进行初始化(使用查表方法)
*******************************************************************************/
void    SED1353_Init(void)
{   volatile uint8   *REG_Point;
    uint8   i;

    REG_Point = (void *) SED1353_REG;
    for(i=0; i<14; i++)                 // 共初始化 14 个寄存器
    {   *REG_Point = INIT_TAB[i];       // 查表取出数据，然后赋值给 SED1353 相应的寄存器，实现初始化
        REG_Point++;                    // 指向下一寄存器
    }
}



uint8 const   LUT_RED_TAB[16] = {   0, 3, 5, 7, 9, 11, 13, 15,
                                    0, 3, 5, 7, 9, 11, 13, 15
                                };
uint8 const   LUT_GRN_TAB[16] = {   0, 3, 5, 7, 9, 11, 13, 15,
                                    0, 3, 5, 7, 9, 11, 13, 15
                                };
uint8 const   LUT_BLU_TAB[16] = {   0, 6, 10, 15,
                                    0, 6, 10, 15,
                                    0, 6, 10, 15,
                                    0, 6, 10, 15
                                };
/*******************************************************************************
* 名称：SED1353_LutInit()
* 功能：初始化 SED1353 的调色板，红、绿基色设置为 0、3、5、7、9、11、13、15，
*       蓝基色设置为 0、6、10、15。
* 入口参数：无
* 出口参数：无
* 说明：使用
*******************************************************************************/
void    SED1353_LutInit(void)
{   volatile uint8   *REG_Point1;
    volatile uint8   *REG_Point2;
    uint8   i;

    REG_Point1 = (void *) (SED1353_REG+0x0E);
    REG_Point2 = (void *) (SED1353_REG+0x0F);
    for(i=0; i<16; i++)
    {   *REG_Point1 = i;                    // 设置为自动更换存取方式，设置地址
        *REG_Point2 = LUT_RED_TAB[i];       // 设置红色调色板
        *REG_Point2 = LUT_GRN_TAB[i];       // 设置绿色调色板
```

```
        *REG_Point2 = LUT_BLU_TAB[i];          // 设置蓝色调色板
    }
}


/*****************************************************************************
* 名称：ScreenPoint()
* 功能：指定点显示。
* 入口参数：disp_adr              显示起始地址
*           dat                   要填充的数据
* 出口参数：无
*****************************************************************************/
void   ScreenPoint(uint32 disp_adr, uint8 dat)
{   volatile uint8   *DAT_Point;

    disp_adr &= 0x0001FFFF;                    // 地址过滤
    DAT_Point = (void *) (SED1353_DAT+disp_adr);
    *DAT_Point = dat;
}


/*****************************************************************************
* 名称：ScreenFill()
* 功能：屏幕填充。屏幕大小为 320*240，256 色。
* 入口参数：dat             要填充的数据
* 出口参数：无
* 说明：直接对显存进行操作。使用前要初始化好驱动器及调色板。
*****************************************************************************/
void   ScreenFill(TCOLOR dat)
{   volatile uint8   *DAT_Point;
    uint32   i;

    DAT_Point = (void *) SED1353_DAT;
    for(i=0; i<320*240; i++)                   // 填充字节数为 320*240
    {   *DAT_Point = dat;
        DAT_Point++;
    }
}


/*****************************************************************************
* 名称：LCD_Initialize()
* 功能：LCM 初始化。将 LCM 初始化为纯图形模式，显示起始地址为 0x0000。
* 入口参数：无
```

```
* 出口参数：无
* 说明：
*******************************************************************************/
void  LCD_Initialize(void)
{  SED1353_Init();                          // 初始化 LCM 工作模式
   SED1353_LutInit();                       // 初始化调色板
}


/*******************************************************************************
* 名称：LCD_FillAll()
* 功能：LCD 填充。以图形方式进行填充，起始地址为 0x0000。
* 入口参数：dat          要填充的颜色数据
* 出口参数：无
*******************************************************************************/
void  LCD_FillAll(TCOLOR dat)
{  volatile uint8  *DAT_Point;
   uint32  i, j;


   /* 开始复制填充数据 */
   DAT_Point = (void *) SED1353_DAT;        // 置地址指针
   for(i=0; i<GUI_LCM_YMAX; i++)            // 历遍所有行
   {  for(j=0; j<GUI_LCM_XMAX; j++)         // 历遍所有行
      {  *DAT_Point++ = dat;
       }
   }
}


/*******************************************************************************
* 名称：LCD_UpdatePoint()
* 功能：在指定位置上画点，刷新某一点。
* 入口参数：  x          指定点所在列的位置
*             y          指定点所在行的位置
* 出口参数：无
* 说明：操作失败原因是指定地址超出有效范围。
*******************************************************************************/
void  LCD_UpdatePoint(uint32 x, uint32 y)
{  volatile uint8  *DAT_Point;
   uint32  addr;

   /* 找出目标地址 */
   addr = y*GUI_LCM_XMAX + x;
```

```
    DAT_Point = (void *) (SED1353_DAT+addr);

    *DAT_Point = gui_disp_buf[y][x];                // 输出数据
}


/*******************************************************************************
*                              与 LCM 相关的 GUI 接口函数
*******************************************************************************/

/*******************************************************************************
* 名称：GUI_FillSCR()
* 功能：全屏填充。直接使用数据填充显示缓冲区。
* 入口参数：dat              填充的数据
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
void   GUI_FillSCR(TCOLOR dat)
{   uint32   i, j;

    /* 填充缓冲区 */
    for(i=0; i<GUI_LCM_YMAX; i++)               // 历遍所有行
    {   for(j=0; j<GUI_LCM_XMAX; j++)           // 历遍所有行
        {   gui_disp_buf[i][j] = dat;
        }
    }

    /* 填充 LCM */
    LCD_FillAll(dat);
}


/*******************************************************************************
* 名称：GUI_Initialize()
* 功能：初始化 GUI，包括初始化显示缓冲区，初始化 LCM 并清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
void   GUI_Initialize(void)
{   LCD_Initialize();                           // 初始化 LCM 模块工作模式，纯图形模式
    GUI_FillSCR(0x00);                          // 初始化缓冲区为 0x00，并输出屏幕(清屏)
}
```

```
/*****************************************************************************
* 名称：GUI_ClearSCR()
* 功能：清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
void    GUI_ClearSCR(void)
{    GUI_FillSCR(0x00);
}



/*****************************************************************************
* 名称：GUI_Point()
* 功能：在指定位置上画点。
* 入口参数：  x           指定点所在列的位置
*             y           指定点所在行的位置
*             color       显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：返回值为 1 时表示操作成功，为 0 时表示操作失败。(操作失败原因是指定
*           地址超出有效范围)
* 说明：用户根据 LCM 的实际情况编写此函数。对于单色，只有一个位有效，则要使用
*       左移的方法实现 point_dat = (point_dat&MASK_TAB [i]) | (color<<n)，其它位数的
*       一样处理。
*****************************************************************************/
uint8    GUI_Point(uint32 x, uint32 y, TCOLOR color)
{    /* 参数过滤 */
     if(x>=GUI_LCM_XMAX) return(0);
     if(y>=GUI_LCM_YMAX) return(0);

     /* 设置缓冲区相应的点 */
     gui_disp_buf[y][x] = color;

     /* 刷新显示 */
     LCD_UpdatePoint(x, y);
     return(1);
}



/*****************************************************************************
* 名称：GUI_ReadPoint()
* 功能：读取指定点的颜色。
* 入口参数：x             指定点所在列的位置
*           y             指定点所在行的位置
```

```
*                ret             保存颜色值的指针
* 出口参数：返回 0 时表示指定地址超出有效范围。
* 说明：对于单色，设置 ret 的 d0 位为 1 或 0，4 级灰度则为 d0、d1 有效，8 位 RGB 则
*          d0--d7 有效，RGB 结构则 R、G、B 变量有效。
*********************************************************************************/
int   GUI_ReadPoint(uint32 x, uint32 y, TCOLOR *ret)
{   /* 参数过滤 */
    if(x>=GUI_LCM_XMAX) return(0);
    if(y>=GUI_LCM_YMAX) return(0);

    /* 取得该点颜色(用户自行更改) */
    *ret = gui_disp_buf[y][x];

    return(1);
}



/*********************************************************************************
* 名称：GUI_HLine()
* 功能：画水平线。
* 入口参数：  x0          水平线起点所在列的位置
*             y0          水平线起点所在行的位置
*             x1          水平线终点所在列的位置
*             color       显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*          可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*********************************************************************************/
void   GUI_HLine(uint32 x0, uint32 y0, uint32 x1, TCOLOR color)
{   uint32   bak;

    if(x0>x1)                                   // 对 x0、x1 大小进行排列，以便画图
    {   bak = x1;
        x1 = x0;
        x0 = bak;
    }

    do
    {   GUI_Point(x0, y0, color);               // 逐点显示，描出垂直线
        x0++;
    }while(x1>=x0);
}
```

```
/*****************************************************************************
* 名称：GUI_RLine()
* 功能：画垂直线。
* 入口参数： x0          垂直线起点所在列的位置
*            y0          垂直线起点所在行的位置
*            y1          垂直线终点所在行的位置
*            color       显示颜色
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*       可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*****************************************************************************/
void  GUI_RLine(uint32 x0, uint32 y0, uint32 y1, TCOLOR color)
{  uint32  bak;

   if(y0>y1)                              // 对 y0、y1 大小进行排列，以便画图
   {  bak = y1;
      y1 = y0;
      y0 = bak;
   }
   do
   {  GUI_Point(x0, y0, color);           // 逐点显示，描出垂直线
      y0++;
   }while(y1>=y0);
}


/*****************************************************************************
* 名称：GUI_CmpColor()
* 功能：判断颜色值是否一致。
* 入口参数：color1       颜色值 1
*           color2       颜色值 2
* 出口参数：返回 1 表示相同，返回 0 表示不相同。
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写比较函数。
*****************************************************************************/
//int  GUI_CmpColor(TCOLOR color1, TCOLOR color2)
//{  if(color1==color2) return(1);
//     else  return(0);
//}


/*****************************************************************************
* 名称：GUI_CopyColor()
* 功能：颜色值复制。
* 入口参数：color1       目标颜色变量
```

```
*                color2          源颜色变量
* 出口参数：无
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写复制函数。
*******************************************************************************/
//void    GUI_CopyColor(TCOLOR *color1, TCOLOR color2)
//{    *color1 = color2;
//}
```

程序清单 5. 8　基于 SED1353F 控制的 LFUBK909XA 液晶模块驱动程序头文件

```
/*******************************************************************************************
* 文件名：LCDDRIVE.H
* 功能：LCD 驱动程序，包括底层驱动，刷新显示子程序。
* 说明：
*******************************************************************************************/
#ifndef    LCDDRIVE_H
#define    LCDDRIVE_H

/*  定义颜色数据类型(可以是数据结构) */
#define    TCOLOR                    uint8

/*  定义 LCM 像素数宏  */
#define    GUI_LCM_XMAX          320          /*  定义液晶 x 轴的像素数  */
#define    GUI_LCM_YMAX          240          /*  定义液晶 y 轴的像素数  */



/*******************************************************************************
* 名称：GUI_Initialize()
* 功能：初始化 GUI，包括初始化显示缓冲区，初始化 LCM 并清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
extern void    GUI_Initialize(void);



/*******************************************************************************
* 名称：GUI_FillSCR()
* 功能：全屏填充。直接使用数据填充显示缓冲区。
* 入口参数：dat            填充的数据
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
extern void    GUI_FillSCR(TCOLOR dat);
```

```
/*****************************************************************************
* 名称：GUI_ClearSCR()
* 功能：清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
extern void    GUI_ClearSCR(void);


/*****************************************************************************
* 名称：GUI_Point()
* 功能：在指定位置上画点。
* 入口参数：  x          指定点所在列的位置
*             y          指定点所在行的位置
*             color      显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：返回值为 1 时表示操作成功，为 0 时表示操作失败。(操作失败原因是指定
*           地址超出有效范围)
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
extern uint8    GUI_Point(uint32 x, uint32 y, TCOLOR color);


/*****************************************************************************
* 名称：GUI_ReadPoint()
* 功能：读取指定点的颜色。
* 入口参数：x          指定点所在列的位置
*           y          指定点所在行的位置
*           ret        保存颜色值的指针
* 出口参数：返回 0 时表示指定地址超出有效范围。
* 说明：对于单色，设置 ret 的 d0 位为 1 或 0，4 级灰度则为 d0、d1 有效，8 位 RGB 则
*       d0--d7 有效，RGB 结构则 R、G、B 变量有效。
*****************************************************************************/
extern int    GUI_ReadPoint(uint32 x, uint32 y, TCOLOR *ret);


/*****************************************************************************
* 名称：GUI_HLine()
* 功能：画水平线。
* 入口参数：  x0         水平线起点所在列的位置
*             y0         水平线起点所在行的位置
*             x1         水平线终点所在列的位置
*             color      显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
```

```
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，可以
*       一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*********************************************************************************/
extern void    GUI_HLine(uint32 x0, uint32 y0, uint32 x1, TCOLOR color);


/*********************************************************************************
* 名称：GUI_RLine()
* 功能：画垂直线。
* 入口参数：  x0          垂直线起点所在列的位置
*             y0          垂直线起点所在行的位置
*             y1          垂直线终点所在行的位置
*             color       显示颜色
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，可以
*       一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*********************************************************************************/
extern void    GUI_RLine(uint32 x0, uint32 y0, uint32 y1, TCOLOR color);


/*********************************************************************************
* 名称：GUI_CmpColor()
* 功能：判断颜色值是否一致。
* 入口参数：color1        颜色值 1
*           color2        颜色值 2
* 出口参数：返回 1 表示相同，返回 0 表示不相同。
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写比较函数。
*********************************************************************************/
//extern int    GUI_CmpColor(TCOLOR color1, TCOLOR color2);
#define    GUI_CmpColor(color1, color2)      (color1==color2)

/*********************************************************************************
* 名称：GUI_CopyColor()
* 功能：颜色值复制。
* 入口参数：color1        目标颜色变量
*           color2        源颜色变量
* 出口参数：无
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写复制函数。
*********************************************************************************/
//extern void    GUI_CopyColor(TCOLOR *color1, TCOLOR color2);
#define    GUI_CopyColor(color1, color2)      *color1 = color2

#endif
```

5.3.4 触模屏驱动程序

在嵌入式系统应用中，一般触模屏大多采用四线式电阻触摸屏，常用的控制芯片有ADS7843/7846、MK715，这里就介绍 ADS7843 触摸屏控制器。ADS7843 是 TI 公司生产的四线式电阻触摸屏控制器，AD 转换精度有 8 位和 12 位两种(即可以精确到 X 或 Y 轴方向上的 1/256 和 1/4096)，3 线/4 线同步串行接口，2.7V 至 5V 工作电源电压，具有低功耗、高速率等特点，其脚图如图 5. 28 所示，原理框图如图 5. 29 所示。ADS7843 应用原理图如图 5. 30 所示。

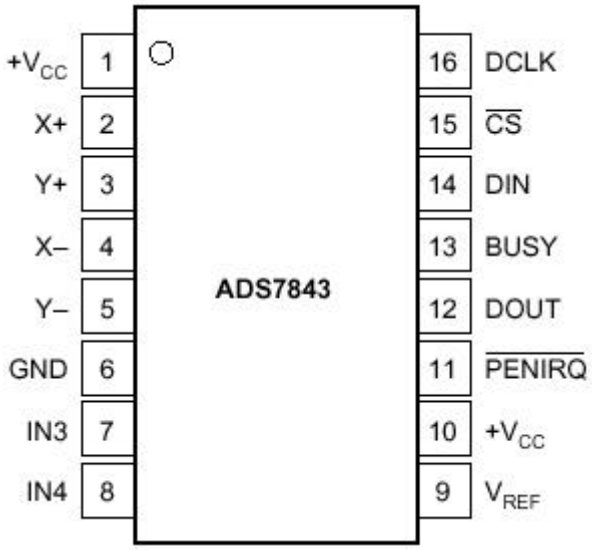

图 5. 28 ADS7843 引脚图

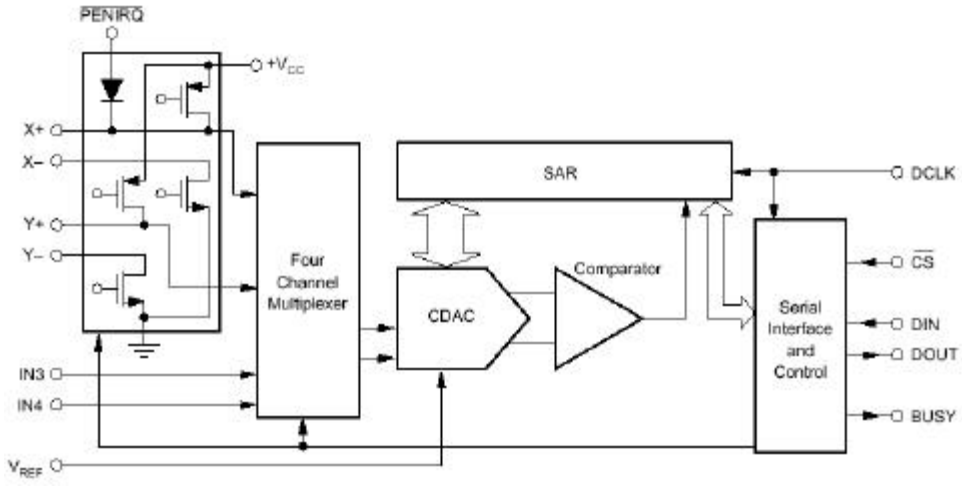

图 5. 29 ADS7843 原理框图

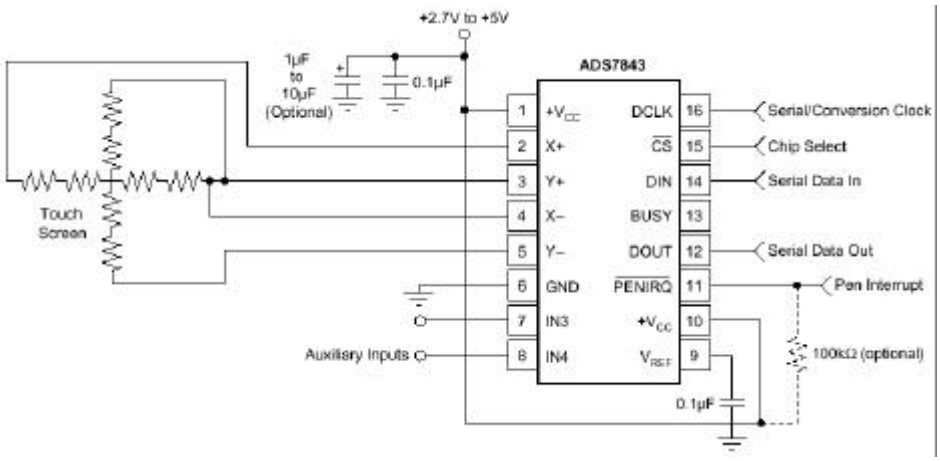

图 5. 30 ADS7843 **应用原理图**

ADS7843 **内置一个二极管用于中断通道和四个开关管，开关状态通过控制字设置，同步串行接口输入/输出数据，**AD7843 **的驱动程序参考程序清单** 5. 9。

程序清单 5. 9 ADS7843 **驱动程序**

```
/*****************************************************************************
* 文件名：ADS7843.C
* 功能：ADS7843 驱动程序。
*
* 说明：若系统时钟过快，需要调整 DELYA_200NS 的值。
*****************************************************************************/
#include    "config.h"

/* ADS7843 控制 I/O 口定义  */
#define       ADS7843_CS              (1<<10)
#define       ADS7843_DOUT            (1<<11)
#define       ADS7843_DIN             (1<<12)
#define       ADS7843_DCLK            (1<<13)

#define       ADS7843_CSS()           IO0SET = ADS7843_CS
#define       ADS7843_CSC()           IO0CLR = ADS7843_CS

#define       ADS7843_DOUTR()         (IO0PIN & ADS7843_DOUT)

#define       ADS7843_DINS()          IO0SET = ADS7843_DIN
#define       ADS7843_DINC()          IO0CLR = ADS7843_DIN

#define       ADS7843_DCLKS()         IO0SET = ADS7843_DCLK
#define       ADS7843_DCLKC()         IO0CLR = ADS7843_DCLK


/*  操作时序控制宏(即延时控制值) */
```

```
#define    DELAY_200NS          10

/*******************************************************************************
* 名称：DelayNo()
* 功能：短软件延时。
* 入口参数：dly              延时参数，值越大，延时越久
* 出口参数：无
*******************************************************************************/
void    DelayNo(uint32 i)
{    for(; i>0; i--);
}



/*******************************************************************************
* 名称：ADS7843_IOInit()
* 功能：初始化 ADS7843 的控制 I/O，CS=1，DCLK=0，DIN=0。
* 入口参数：无
* 出口参数：无
*******************************************************************************/
void    ADS7843_IOInit(void)
{    ADS7843_CSS();                         // CS = 1
     ADS7843_DCLKC();                       // DCLK = 0
     ADS7843_DINC();                        // DIN = 0
     DelayNo(DELAY_200NS);
}



/*******************************************************************************
* 名称：ADS7843_WriteRead()
* 功能：对 ADS7843 进行读写操作。操作按照 ADS7843 的规定，24Clocks，先写 8 位控制数据，然
*       后读取 12 位的转换结果。
* 入口参数：data             控制数据
* 出口参数：返回值为读出的数据
*******************************************************************************/
uint16    ADS7843_WriteRead(uint8 data)
{    uint8     i;
     uint16    ret_dat;

     data = data|0x80;                      //  设置 S 位
     ADS7843_IOInit();                      //  初始化 ADS7843 的控制 I/O

     ADS7843_CSC();                         // CS = 0
     for(i=0; i<8; i++)
     {    if( (data&0x80) != 0 ) ADS7843_DINS();    // DIN = 1
```

```
            else    ADS7843_DINC();                 // DIN = 0
        DelayNo(DELAY_200NS);
        ADS7843_DCLKS();                            // DCLK = 1
        DelayNo(DELAY_200NS);
        ADS7843_DCLKC();                            // DCLK = 0
        data = data<<1;
    }
    ADS7843_DINC();                                 // DIN = 0
    DelayNo(DELAY_200NS);

    ADS7843_DCLKS();                                // DCLK = 1
    DelayNo(DELAY_200NS);

    ret_dat = 0;
    for(i=0; i<12; i++)
    {   ret_dat = ret_dat<<1;
        ADS7843_DCLKC();                            // DCLK = 0
        DelayNo(DELAY_200NS);
        if( ADS7843_DOUTR() != 0 ) ret_dat = ret_dat|1;
        ADS7843_DCLKS();                            // DCLK = 1
        DelayNo(DELAY_200NS);
    }
    ADS7843_DCLKC();                                // DCLK = 0
    DelayNo(DELAY_200NS);

    for(i=0;i<3;i++)
    {   ADS7843_DCLKS();                            // DCLK = 1
        DelayNo(DELAY_200NS);
        ADS7843_DCLKC();                            // DCLK = 0
        DelayNo(DELAY_200NS);
    }

    ADS7843_CSS();                                  // CS = 1
    return(ret_dat);
}
```

5.4 ZLG/GUI 应用手册

5.4.1 ZLG/GUI 概述

ZLG/GUI 是占用资源小、使用方便的嵌入式系统简易的图形用户界面软件。ZLG/GUI 提供了最基本的画点、线、圆形、圆弧、椭圆形、矩形、正方形、填充等功能，较高级的接口功能有 ASCII 显示、汉字显示、图标显示、窗口、菜单等，支持单色、灰度、伪彩、真彩等图形显示设备。

5.4.2 基本画图原理

为了实现基本的图形操作，要在 RAM 中开辟一块或多块显示缓冲区，缓冲区大小一般与实际图形显示设备的点像素对应，如 128×64 点的单色图形 LCD，开辟显示缓冲区大小为 1024 字节(即 128×64/8，一字节数据对应 8 个点像素)。在 RAM 开辟显示缓冲区，进行图形操作时可以获得较高的速度，待操作完毕后再将显示缓冲区的数据发送到显示设备中；当然也可以直接使用显示设备的显示 RAM，如果直接使用显示设备中的显示 RAM 进行图形操作，由于其读/写操作速度较慢，会直接影响显示效果。一般地，在系统存储器 RAM 较宽松的情况下，对于单色、4 级灰度、16 级灰度或读/写操作速度较慢的显示设备，就使用系统存储器 RAM 建立显示缓冲区。

ZLG/GUI 的显示存储器映射(即显示缓冲区映射)如图 5. 31 与图 5. 32 所示。缓冲区为顺序排列的，即显示区(0,0)上的点对应缓冲区的第 1 个字节的显示数据。对于各种图形的运算结果先保存在显示缓冲区中。ZLG/GUI 接口函数的坐标参数采用绝对坐标值。

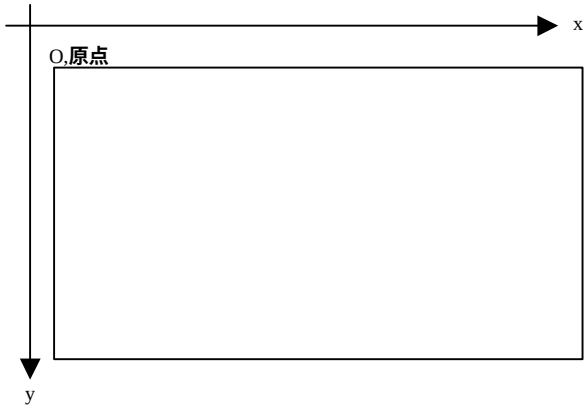

图 5. 31 显示设备的显示区

与图 5. 31 显示区对应的显示缓冲区如图 5. 32 所示。

D0	D1	D2	D3	D4	D5	D6	D7	D0	…
…									
Dn	Dn	Dn	Dn	Dn	Dn	Dn	Dn	Dn+1	…

图 5. 32 显示缓冲区

为了更好的理解及设计软件，可以将 ZLG/GUI 分为三个层次，如图 5. 33 所示。

第一层为硬件驱动层。这一层主要负责硬件驱动，将显示数据转换并发送给图形显示设

备。用户在该层上进行显示设备硬件及显示缓冲区的设置，处理画点、水平线、垂直线等操作。对于不同的图形显示设备，只需要对这一层进行更改即可使用 ZLG/GUI。

对于嵌入式系统，图形显示设备一般为点阵式图形液晶显示模块，即 LCM。对于不同的图形设备，可分为单色、灰度及彩色三种，它们的硬件控制操作、初始化操作、显示缓冲区的处理方法是不一样的。

第二层为基本图形层。这一层提供了其本的画线、圆形、圆弧、椭圆形、四方形、矩形、填充等功能。该层的主要特点是进行各种运算，实现图形显示。这一层的程序会直接调用硬件驱动层的函数实现显示更新。

第三层为高级接口层。主要为用户提供窗口、图标、菜单等图形接口，一般直接调用第二层的基本画图函数来实现。但为了加快刷新速度，可以调用第一层的驱动函数发送数据。该层的主要特点是进行各种数据结构处理，将它们转换为显示的具体参数，并控制画图操作。

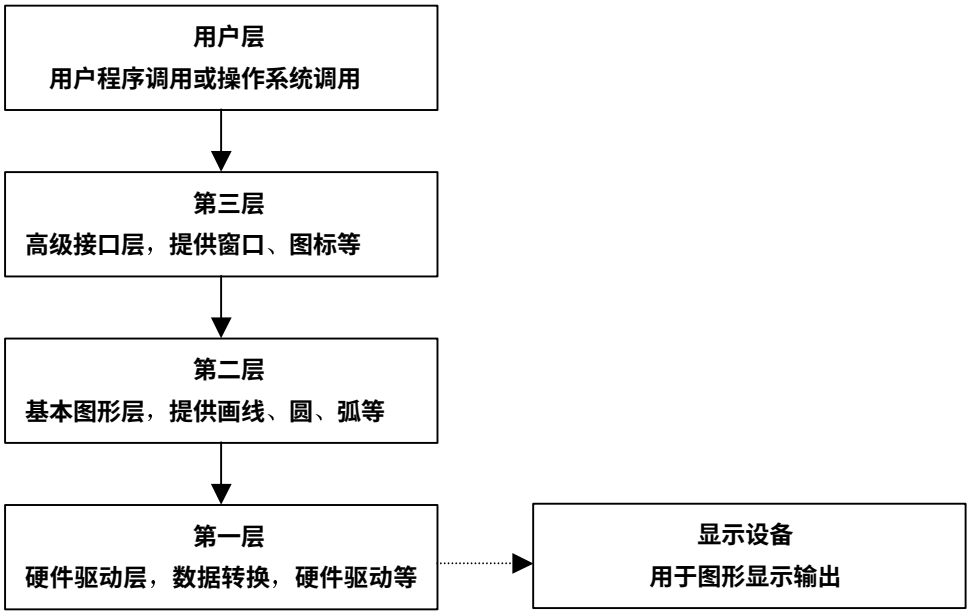

图 5. 33 ZLG/GUI 的层次关系图

在实际较大型 GUI 系统中，其接口设置会更复杂一些，一般是以消息驱动为基础，具有大量的图形资源、操作函数、数据结构，这样即可实现代码的最大重复利用，形成一个规范的接口方式，方便编程及扩展，具有很好的图形效果。

5.4.3 基本画图函数

ZLG/GUI 提供了最基本的画点、线、圆形、圆弧、椭圆形、矩形、正方形、填充等功能，对于这些基本画图函数，均采用实际的绝对坐标进行作图。在这一节里将详细介绍相关的函数功能及原理。

作图的基本思想。如果在 RAM 中建立显示缓冲区，画图操作先对显示缓冲区相应点的数据进行设置，然后控制数据输出更新 LCM 显示，保证 LCM 显示及缓冲区数据同步(注意，显示缓冲区只有硬件驱动层使用)；若没有在 RAM 中建立显示缓冲区，则在画点时可能需要先读取 LCM 上的数据(如单色 LCM、4 级/16 级灰度 LCM 等)，进行与/或操作，然后再将数据输出 LCM 更新显示，保证作图时不破坏原显示图形。

颜色的处理。对于不同的液晶显示模块，可定义不同的颜色数据类型 TCOLOR，如无符号 8 位、无符号 16 位、无符号 32 位，甚至数据结构。一般地，单色、4 级灰度、16 级灰度等颜色值不满一个字节的 LCM 定义 TCOLOR 为 uint8，画图操作使用低位来传递颜色参

数。ZLG/GUI 有时需要进行颜色变量值比较和复制颜色值，由于不同的颜色数据类型处理代码不完全一致，所以需要用户编写接口函数进行处理(由 ZLG/GUI 调用)，如后面介绍的 GUI_CmpColor()、GUI_CopyColor()。

显示缓冲区定义。首先在图形液晶模块驱动程序的头文件中定义两个与 LCD 像素大小相关的两个宏(如程序清单 5. 10 所示)，以便于系统定义缓冲区，文件名如 LCDDRIVE.H。另外，ZLG/GUI 也会根据这两个宏来判断操作是否超出范围，所以不管是否在 RAM 中建立显示缓冲区，这两个宏必须定义。

程序清单 5. 10 颜色数据类型及 LCD 像素大小定义

```
#define   TCOLOR              uint8
#define   GUI_LCM_XMAX        240          /* 定义液晶 x 轴的点数 */
#define   GUI_LCM_YMAX        128          /* 定义液晶 y 轴的点数 */
```

然后在程序 LCD_DRIVE.C 中将定义显示缓冲区，缓冲区定义成一个二维数组。对于单色 LCM，由于每一个字节对应 8 个点像素，所以缓冲区的定义为 gui_disp_buf [GUI_LCM_YMAX][GUI_LCM_XMAX/8]，倘若 GUI_LCM_XMAX 不是 8 的倍数时，显示缓冲区的定义改为 gui_disp_buf [GUI_LCM_YMAX][GUI_LCM_XMAX/8+1]，如程序清单 5. 11 所示。对于 256 色伪彩 LCM，每一字节对应一个点像素，所以缓冲区的定义为 gui_disp_buf [GUI_LCM_YMAX][GUI_LCM_XMAX]，如程序清单 5. 12 所示。

程序清单 5. 11 显示缓冲区定义—单色 LCM

```
/* 定义显示缓冲区 */
TCOLOR    gui_disp_buf[GUI_LCM_YMAX][GUI_LCM_XMAX/8];
```

程序清单 5. 12 显示缓冲区定义—256 色 LCM

```
/* 定义显示缓冲区 */
TCOLOR    gui_disp_buf[GUI_LCM_YMAX][GUI_LCM_XMAX];
```

硬件驱动层接口函数。为了实现对 LCM 的显示驱动控制，用户需提供如程序清单 5. 13 所示的九个 LCM 驱动接口函数。ZLG/GUI 的所有图形显示操作均通过调用这九个函数实现，这些函数的执行效率直接影响图形绘制的效率。

程序清单 5. 13 LCM 驱动接口函数

```
/*****************************************************************************
* 名称：GUI_FillSCR()
* 功能：全屏填充。直接使用数据填充显示缓冲区。
* 入口参数：dat              填充的数据
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
extern void    GUI_FillSCR(TCOLOR dat);

/*****************************************************************************
* 名称：GUI_Initialize()
```

```
* 功能：初始化 GUI，包括初始化显示缓冲区，初始化 LCM 并清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
extern void    GUI_Initialize(void);

/*******************************************************************************
* 名称：GUI_ClearSCR()
* 功能：清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
extern void    GUI_ClearSCR(void);

/*******************************************************************************
* 名称：GUI_Point()
* 功能：在指定位置上画点。
* 入口参数：  x            指定点所在列的位置
*             y            指定点所在行的位置
*             color        显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：返回值为 1 时表示操作成功，为 0 时表示操作失败。(操作失败原因是指定
*           地址超出有效范围)
* 说明：用户根据 LCM 的实际情况编写此函数。
*******************************************************************************/
extern uint8    GUI_Point(uint32 x, uint32 y, TCOLOR color);

/*******************************************************************************
* 名称：GUI_ReadPoint()
* 功能：读取指定点的颜色。
* 入口参数：x              指定点所在列的位置
*           y              指定点所在行的位置
*           ret            保存颜色值的指针
* 出口参数：返回 0 时表示指定地址超出缓冲区范围。
* 说明：对于单色，设置 ret 的 d0 位为 1 或 0，4 级灰度则为 d0、d1 有效，8 位 RGB 则
*       d0--d7 有效，RGB 结构则 R、G、B 变量有效。
*******************************************************************************/
extern int    GUI_ReadPoint(uint32 x, uint32 y, TCOLOR *ret);

/*******************************************************************************
* 名称：GUI_HLine()
* 功能：画水平线。
* 入口参数：  x0           水平线起点所在列的位置
```

```
*                y0          水平线起点所在行的位置
*                x1          水平线终点所在列的位置
*                color       显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*       可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*********************************************************************************/
extern void    GUI_HLine(uint32 x0, uint32 y0, uint32 x1, TCOLOR color);

/*********************************************************************************
* 名称：GUI_RLine()
* 功能：画垂直线。
* 入口参数：  x0          垂直线起点所在列的位置
*             y0          垂直线起点所在行的位置
*             y1          垂直线终点所在行的位置
*             color       显示颜色
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*       可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*********************************************************************************/
extern void    GUI_RLine(uint32 x0, uint32 y0, uint32 y1, TCOLOR color);

/*********************************************************************************
* 名称：GUI_CmpColor()
* 功能：判断颜色值是否一致。
* 入口参数：color1        颜色值 1
*           color2        颜色值 2
* 出口参数：返回 1 表示相同，返回 0 表示不相同。
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写比较函数。
*********************************************************************************/
extern int    GUI_CmpColor(TCOLOR color1, TCOLOR color2);

/*********************************************************************************
* 名称：GUI_CopyColor()
* 功能：颜色值复制。
* 入口参数：color1        目标颜色变量
*           color2        源颜色变量
* 出口参数：无
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写复制函数。
*********************************************************************************/
extern void    GUI_CopyColor(TCOLOR *color1, TCOLOR color2);
```

接口函数编程实例。GUI_Initialize()**函数如程序清单** 5. 14 **所示，程序调用了**

LCD_Initialize()初始化 LCM，将其设置为图形工作模式并打开显示；然后 GUI_FillSCR()进行全屏填充，实现清屏操作。另外，用户也可以加入背光灯的控制操作，点亮背光灯。

程序清单 5. 14 LCM 驱动接口函数--GUI_Initialize()

```
void   GUI_Initialize(void)
{   LCD_Initialize();                          // 初始化 LCM 模块工作模式，纯图形模式
    GUI_FillSCR(0x00);                         // 初始化缓冲区为 0x00，并输出屏幕(清屏)
}
```

GUI_FillSCR()函数如程序清单 5. 15 所示，程序调用了 LCD_FillAll()进行 LCM 全屏填充。当使用显示缓冲区 gui_disp_buf 时，需要同时填充缓冲区。

程序清单 5. 15 LCM 驱动接口函数--GUI_FillSCR()

```
void   GUI_FillSCR(TCOLOR dat)
{   uint32   i, j;

    /* 填充缓冲区 */
    for(i=0; i<GUI_LCM_YMAX; i++)              // 历遍所有行
    {   for(j=0; j<GUI_LCM_XMAX; j++)          // 历遍所有列
        {   gui_disp_buf[i][j] = dat;
        }
    }

    /* 填充 LCM */
    LCD_FillAll(dat);
}
```

GUI_ClearSCR()函数如程序清单 5. 16 所示，可直接调用 GUI_FillSCR 函数实现清屏。此函数可以定义成宏函数，以提高程序效率。

程序清单 5. 16 LCM 驱动接口函数--GUI_ClearSCR()

```
void   GUI_ClearSCR(void)
{   GUI_FillSCR(0x00);
}
//  也可以定义宏函数
//  #define   GUI_ClearSCR()                   GUI_FillSCR(0x00)
```

GUI_CmpColor()函数如程序清单 5. 17 所示，比较两个颜色值是否相等。对于单色 LCM，则只需比较 d0 位即可；对于 256 色 LCM，则需要比较一字节数据。此函数可以定义成宏函数，以提高程序效率。

程序清单 5. 17 LCM 驱动接口函数--GUI_CmpColor()

```
int   GUI_CmpColor(TCOLOR color1, TCOLOR color2)
{   if(color1==color2) return(1);
```

```
    else    return(0);
}
//  也可以定义宏函数
//  #define    GUI_CmpColor(color1, color2)         (color1==color2)
```

GUI_CopyColor()函数如程序清单 5. 18 所示，复制颜色值，即给颜色变量赋值。对于单色 LCM，只复制最低位 d0，其它位清 0；对于 256 色 LCM，则直接使用“=”赋值即可。此函数可以定义成宏函数，以提高程序效率。

程序清单 5. 18 LCM 驱动接口函数--GUI_CopyColor()

```
void    GUI_CopyColor(TCOLOR *color1, TCOLOR color2)
{   *color1 = color2;
}
//  也可以定义宏函数
//  #define    GUI_CopyColor(color1, color2)        *color1=color2
```

点坐标转换计算。进行图形操作最基本的运算就是点坐标转换计算，即将指定点的坐标值(x, y)转换成实际 LCM 的显示存储单元地址，以便于画点或读点操作(读点，即读取指定点上的当前显示颜色值)。进行转换计算要依据 LCM 的规定，列出 addr = ky +x 关系式，但对于单色 LCM、4 级灰度、16 级灰度或显示存储单元排列较为特殊的液晶模块，它们的计算公式有所不同。

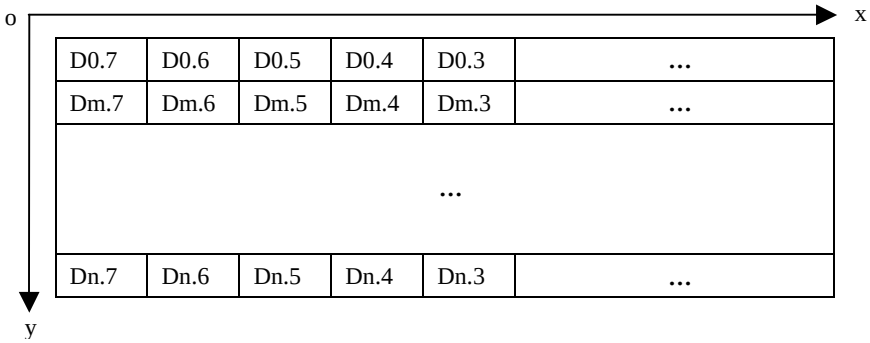

图 5. 34 单色 LCM 点像素与显示存储单元关系图--T6963/SED1335

如图 5. 34 所示为 T6963/SED1335 控制的单色 LCM 的点像素与显示存储单元关系，显示存储单元地址计算 addr = ky + x/8，其中 k 为一行点像素所使用的字节个数。取得要操作点所在的显示存储单元地址后，由于是单色 LCM，每一个点只占该存储字节单元中的一个位，所以还需要针对显示位进行操作。如程序清单 5. 19、程序清单 5. 20 所示，先定义一个 DEC_HEX_TAB 数据转换表，然后通过查表对所要操作位进行置位或复位操作。对于 4 级灰度、16 级灰度 LCM，可以使用 point_dat = (point_dat&MASK_TAB[i]) | (color<<n)进行设置，MASK_TAB 为点像素数据屏蔽字表，主要目的是为了设置指定点的数据，且不破坏其它点原有的数据。

程序清单 5. 19 DEC_HEX_TAB 数据转换表--T6963/SED1335

```
/*  定义转换表 DEC->HEX(根据实际液晶情况设置) */
// uint8    const    DEC_HEX_TAB[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
uint8    const    DEC_HEX_TAB[8] = { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 };
```

程序清单 5. 20 **单色** LCM **画点操作描述**–T6963/SED1335

```
/* 读取指定点(x, y)上的显示存储单元数据到 point_dat */
point_dat = *addr;

/* 置位操作 */
if(显示) point_dat = point_dat | DEC_HEX_TAB[x&0x07];
/* 复位操作 */
   else    point_dat = point_dat &(~ DEC_HEX_TAB[x&0x07]);

/* 将 point_dat 数据写回 addr 地址，输出显示 */
*addr = point_dat;
```

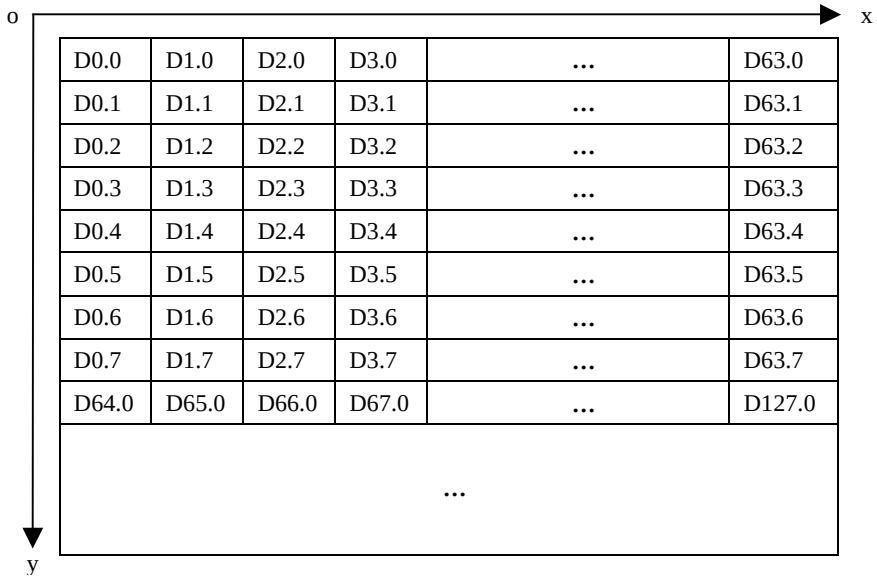

图 5. 35 **单色** LCM **点像素与显示存储单元关系图**–KS0107/8

对于使用 KS0107/8 **控制的单色** LCM，**点像素与显示存储单元关系比较特殊，如图** 5. 35 **所示**。KS0107/8 **对显示存储单元的访问是通过地址寄存器** X(**页寄存器**)**和地址寄存器** Y **来实现的，如图** 5. 35 **中的** D0～D63 **为第** 0 **页，即** X=0，D64～D127 **为第** 1 **页，即** X=1；**而** D0 **的** Y **寄存器地址为** 0，D1 **的** Y **寄存器地址为** 1 **….**D64 **的** Y **寄存器地址为** 0。**所以指定点的坐标值**(x, y)**要转换为两部分地址，才能访问到正确的显示存储单元，如** Xaddr、Yaddr。**计算公式为** Xaddr = y/8，Yaddr = x，**其中** x **要小于** 64。**而** DEC_HEX_TAB **数据转换表定义如程序清单** 5. 21 **所示，画点操作程序的描述如程序清单** 5. 22 **所示**。

程序清单 5. 21 DEC_HEX_TAB **数据转换表**–KS0107/8

```
/* 定义转换表 DEC->HEX(根据实际液晶情况设置) */
uint8   const   DEC_HEX_TAB[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
//uint8   const   DEC_HEX_TAB[8] = { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 };
```

程序清单 5. 22 **单色** LCM **画点操作描述**–KS0107/8

```
/* 读取指定点(x, y)上的显示存储单元数据到 point_dat */
```

```
point_dat = *(Xaddr, Yaddr);

/* 置位操作 */
if(显示) point_dat = point_dat | DEC_HEX_TAB[y&0x07];
/* 复位操作 */
   else    point_dat = point_dat &(~ DEC_HEX_TAB[y&0x07]);

/* 将 point_dat 数据写回(Xaddr, Yaddr)地址，输出显示 */
*(Xaddr, Yaddr) = point_dat;
```

对于 256 色 LCM，每一点即对应一字节显示存储单元，如图 5. 36 所示。显示存储单元地址计算 addr = ky + x，其中 k 为一行所使用的字节个数，即一行的点像素个数。画点操作程序描述如程序清单 5. 23 所示。

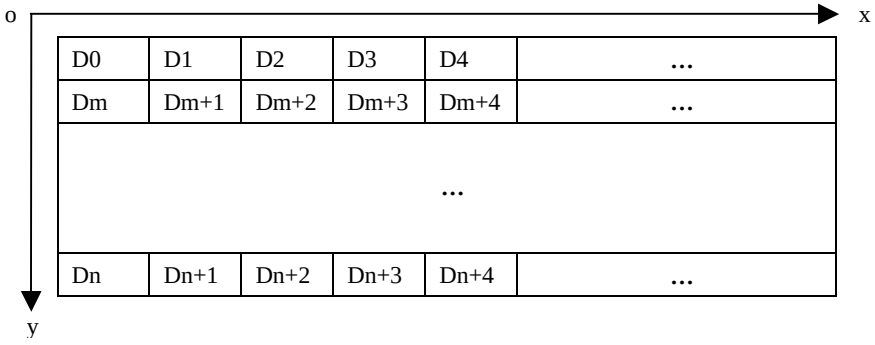

图 5. 36 256 色 LCM 点像素与显示存储单元关系图–SED1353

程序清单 5. 23 256 色 LCM 画点操作描述–SED1353

```
/* 将 point_dat 颜色值数据写入 addr 地址，输出显示 */
*addr = point_dat;
```

更新点函数。这是为 GUI_Point()提供的一个用于更新 LCM 指定点显示的函数，函数旬及参数用户均可自由设定，也可以直接在 GUI_Point()内编写相应的代码，这里使用 LCD_UpdatePoint()，入口参数为需更新点的坐标值(x, y)。对于使用 RAM 作显示出缓冲区时，更新的数据可在 gui_disp_buf 显示缓冲区中的取得，因为 gui_disp_buf 的内容即是 LCD 显示屏要显示的内容，如程序清单 5. 24 所示；对于没有在 RAM 中建立显示缓冲区的情况，LCD_UpdatePoint()还需要一个入口参数——更新的数据，程序直接把这一数据输出到 LCM 即可，如程序清单 5. 25 所示。

程序清单 5. 24 LCD_UpdataPoint()函数例子–有显示缓冲区

```
void   LCD_UpdatePoint(uint32 x, uint32 y)
{   volatile uint8   *DAT_Point;
    uint32   addr;

    /* 找出目标地址 */
    addr = y*GUI_LCM_XMAX + x;
    DAT_Point = (void *) (SED1353_DAT+addr);
```

```
    *DAT_Point = gui_disp_buf[y][x];                    // 输出数据
}
```

程序清单 5. 25 LCD_UpdataPoint()**函数例子–无显示缓冲区**

```
void  LCD_UpdatePoint(uint32 x, uint32 y, uint8 dat)
{  volatile uint8  *DAT_Point;
   uint32  addr;

   /* 找出目标地址 */
   addr = y*GUI_LCM_XMAX + x;
   DAT_Point = (void *) (SED1353_DAT+addr);

   *DAT_Point = dat;                                    // 输出数据
}
```

5.4.4 ZLG/GUI 参考手册

在这节里将统一介绍 ZLG/GUI **的接口函数及可用资源。函数是按其功能分类的，并且分别编写到不同的文件中，如下所示:**

基本图形操作函数	**——**	GUI_BASE.C
显示颜色管理函数	**——**	GUI_STOCKC.C
颜色转换操作函数	**——**	CONVERTCOLOR.C
5×7ASCII **码字库及显示函数**	**——**	FONT5_7.C
8×8ASCII **码字库及显示函数**	**——**	FONT8_8.C
24×32 **数字库及显示函数**	**——**	FONT24_32.C
单色图形及汉字显示函数	**——**	LOADBIT.C
图标菜单、下拉菜单操作函数	**——**	MENU.C
窗口操作函数	**——**	WINDOW.C

其它重要文件说明如下:

CONFIG.H——**用于声明常用宏，包含所有项目所用的头文件。(方便项目的管理)**

GUI_CONFIG.H——**用于配置** ZLG/GUI。**(用于裁剪** ZLG/GUI)

FONT_MACRO.H——**定义字节点阵宏。(用于定义字体点阵数据)**

功能配置说明如下:

- **GUI_LineWith_EN**

画有宽度的直线函数 GUI_LineWith()**使能控制，设置为** 1 **时函数有效，为** 0 **或其它值时函数禁止。**

- **GUI_CircleX_EN**

画圆函数 GUI_Circle()、GUI_CircleFill()**使能控制，设置为** 1 **时函数有效，为** 0 **或其它值时函数禁止。**

- **GUI_EllipseX_EN**

画椭圆函数 GUI_Ellipse()、GUI_EllipseFill()**使能控制，设置为** 1 **时函数有效，为** 0 **或**

其它值时函数禁止。

- **GUI_FloodFill_EN**

填充函数 GUI_FloodFill()使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。当使用填充函数时，可以定义 DOWNP_N、UPP_N 宏来设置向上及向下折点个数，这两个宏用于定义保存向上及向下折点数据的数组大小。

- **GUI_ArcX_EN**

画圆弧函数 GUI_Arc4()、GUI_Arc 使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。

- **GUI_Pieslice_EN**

扇形函数 GUI_Pieslice()使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。

- **GUI_WINDOW_EN**

窗口管理函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。窗口管理函数如 GUI_WindowsDraw()、GUI_WindowsHide()、GUI_WindowsClr()等。由于窗口管理函数使用到 5×7 ASCII 字体显示，所以必需同时设置 FONT5x7_EN 为 1。

- **GUI_MenuIco_EN**

图标菜单操作函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。图标菜单操作函数如 GUI_MenuIcoDraw() 、 GUI_Button49x14() 、 GUI_Button_OK() 、GUI_Button_Cancle()、GUI_Button_OK1()、GUI_Button_Cancle1()等。

- **GUI_MenuDown_EN**

下拉菜单操作函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。下拉菜单操作函数如 GUI_MMenuDraw() 、 GUI_MMenuSelect() 、 GUI_MMenuNSelect() 、GUI_SMenuDraw ()、GUI_SMenuSelect ()、GUI_SMenuHide ()等。

- **FONT5x7_EN**

5×7 ASCII 码字库及显示函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。5×7 ASCII 码字符显示函数如 GUI_PutChar()、GUI_PutString()、GUI_PutNoStr()等。

- **FONT8x8_EN**

8×8 ASCII 码字库及显示函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。8×8 ASCII 码字符显示函数如 GUI_PutChar8_8 ()、GUI_PutString8_8 ()、GUI_PutNoStr8_8 ()等。

- **FONT24x32_EN**

24×32 数字库及显示函数使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。24×32 数字字符显示函数如 GUI_PutChar24_32()。

- **GUI_PutHZ_EN**

汉字显示函数 GUI_PutHZ()使能控制，设置为 1 时函数有效，为 0 或其它值时函数禁止。

- **GUI_LoadPic_EN**

单色图形显示函数使能控制，设置为 1 **时函数有效，为** 0 **或其它值时函数禁止。单色图形显示函数如** GUI_LoadPic()、GUI_LoadPic1()。

- **CONVERTCOLOR_EN**

颜色转换操作函数使能控制，设置为 1 **时函数有效，为** 0 **或其它值时函数禁止。颜色转换操作函数如** GUI_Color2Index_565()、GUI_Color2Index_555()、GUI_Color2Index_444()、GUI_Color2Index_332 **等等**。

ZLG/GUI **函数表如表** 5. 1～**表** 5. 11 **所列。以下函数原型中，**uint8 **已定义为** unsigned char，uint32 **已定义为** unsigned int。

表 5. 1 **硬件驱动层接口函数**(LCDDRIVE.C)

函数原型	参数	功能
void GUI_Initialize(void)	**无**	**初始化** GUI(**缓冲区及** LCM)
void GUI_FillSCR(TCOLOR dat)	dat **填充的颜色值数据**	**全屏填充**
Void GUI_ClearSCR(void)	**无**	**清屏**
uint8 GUI_Point(uint32 x, uint32 y, TCOLOR color)	x，y **点的坐标** color **显示颜色**	**画点(返回值为** 1 **时表示操作成功，为** 0 **表示失败)**
int GUI_ReadPoint(uint32 x, uint32 y, TCOLOR *ret);	x，y **点的坐标** ret **保存变量的指针**	**读取指定点的颜色(返回值为** 0 **时表示操作失败)**
void GUI_HLine(uint32 x0, uint32 y0, uint32 x1, TCOLOR color)	x0，y0 **起点坐标** x1 **水平线终点** x **值** color **显示颜色**	**画水平线**
void GUI_RLine(uint32 x0, uint32 y0, uint32 y1, TCOLOR color)	x0，y0 **起点坐标** y1 **垂直线终点** y **值** color **显示颜色**	**画垂直线**
int GUI_CmpColor(TCOLOR color1, TCOLOR color2)	color1 **颜色值** 1 color2 **颜色值** 2	**比较两个颜色值是否相等，相等返回** 1，**否则返回** 0
void GUI_CopyColor(TCOLOR *color1, TCOLOR color2)	color1 **目标变量指针** color2 **原颜色值**	**颜色值复制，即***color1= color2

可用资源：LCDDRIVE.H **中定义宏** TCOLOR、GUI_LCM_XMAX **及** GUI_LCM_YMAX，TCOLOR **为颜色数据类型**，GUI_LCM_XMAX **和** GUI_LCM_YMAX **为** LCM **点像素数**。

表 5. 2 **基本图形操作函数**(GUI_BASE.C)

函数原型	参数	功能
void GUI_Line(uint32 x0, uint32 y0, uint32 x1, uint32 y1, TCOLOR color)	x0，y0 **起点坐标** x1，y1 **终点坐标** color **显示颜色**	**画任意两点之间的直线**
void GUI_LineWith(uint32 x0, uint32 y0, uint32 x1, uint32 y1, uint8 with, TCOLOR color)	x0，y0 **起点坐标** x1，y1 **终点坐标** with **线宽大小** color **显示颜色**	**画具有线宽的任意两点之间的直线**(with **值最大为** 50)

续表 5.2

函数原型	参数	功能
void GUI_LineS(uint32 const *points, uint8 no, TCOLOR color)	*points 顶点坐标值指针 no 顶点数 color 显示颜色	画多边形(即多个顶点之间的连续连线)
void GUI_Rectangle(uint32 x0, uint32 y0, uint32 x1, uint32 y1, TCOLOR color)	x0，y0 矩形左上角坐标 x1，y1 矩形右下角坐标 color 显示颜色	画矩形
void GUI_RectangleFill(uint32 x0, uint32 y0, uint32 x1, uint32 y1, TCOLOR color)	x0，y0 矩形左上角坐标 x1，y1 矩形右下角坐标 color 填充颜色	画填充矩形
void GUI_Square(uint32 x0, uint32 y0, uint32 with, TCOLOR color)	x0，y0 正方形左上角坐标 with 边长 color 显示颜色	画正方形
void GUI_Circle(uint32 x0, uint32 y0, uint32 r, TCOLOR color)	x0，y0 圆形的圆心坐标 r 圆形的半径 color 显示颜色	画圆形
void GUI_CircleFill(uint32 x0, uint32 y0, uint32 r, TCOLOR color)	x0，y0 圆形的圆心坐标 r 圆形的半径 color 填充颜色	画填充圆形
void GUI_Ellipse(uint32 x0, uint32 x1, uint32 y0, uint32 y1, TCOLOR color)	x0，x1 椭圆形的 x 坐标 y0，y1 椭圆形的 y 坐标 color 显示颜色	画椭圆形(x0、x1 分别为椭圆最左和最右的点的 x 坐标；y0、y1 分别为椭圆最上和最下的点的 y 坐标)
void GUI_EllipseFill(uint32 x0, uint32 x1, uint32 y0, uint32 y1, TCOLOR color)	x0，x1 椭圆形的 x 坐标 y0，y1 椭圆形的 y 坐标 color 填充颜色	画填充椭圆形(x0、x1 分别为椭圆最左和最右的点的 x 坐标；y0、y1 分别为椭圆最上和最下的点的 y 坐标)
void GUI_FloodFill(uint32 x0, uint32 y0, TCOLOR color)	x0，y0 指定填充点坐标 color 填充颜色	图形填充(在 GUI_CONFIG.H 中配置向上及向下折点个数)
void GUI_Arc4(uint32 x, uint32 y, uint32 r, uint8 angle, TCOLOR color)	x，y 圆弧圆心坐标 r 圆弧半径 angle 圆弧角度 color 显示颜色	画 1/4 圆弧(angle 为 1～4，即 0～90 度、90～180 度、180～270 度、270～360 度)
void GUI_Arc(uint32 x, uint32 y, uint32 r, uint32 stangle, uint32 endangle, TCOLOR color)	x，y 圆弧圆心坐标 r 圆弧半径 stangle 圆弧起始角度 endangle 圆弧终止角度 color 显示颜色	画任意角度圆弧(stangle、endangle 为 0～359 度)
void GUI_Pieslice(uint32 x, uint32 y, uint32 r, uint32 stangle, uint32 endangle, TCOLOR color)	x0，y0 扇形圆心坐标 r 扇形半径 stangle 扇形起始角度 endangle 扇形终止角度 color 显示颜色	画扇形(stangle、endangle 为 0～359 度)

可用资源：GUI_BASIC.H **中定义数据结构** PointXY，**结构变量中包含点的坐标变量** x、y，**数据类型均为** uint32，**如程序清单** 5. 26 **所示**。

程序清单 5. 26 **点数据结构定义**

```
/* 定义坐标数据结构 */
typedef  struct
         {  uint32  x;                    // x 坐标变量
            uint32  y;                    // y 坐标变量
         }
         PointXY;
```

表 5. 3 **显示颜色管理函数**(GUI_STOCKC.C)

函数原型	参数	功能
void GUI_SetColor(TCOLOR color1, TCOLOR color2)	color1 前景色 color2 背景色	设置前景色及背景色
void GUI_GetBackColor(TCOLOR *bakc)	*bakc 保存变量的指针	读取背景色的值
void GUI_GetDispColor(TCOLOR *bakc)	* bakc 保存变量的指针	读取前景色的值
void GUI_ExchangeColor(void)	无	交换前景色与背景色

表 5. 4 **窗口管理函数**(WINDOWS.C)

函数原型	参数	功能
uint8 GUI_WindowsDraw(WINDOWS *win)	*win 窗口句柄	显示窗口
uint8 GUI_WindowsHide(WINDOWS *win)	*win 窗口句柄	消隐窗口
void GUI_WindowsClr(WINDOWS *win)	*win 窗口句柄	清屏窗口(即清屏窗口用户区域)

窗口数据结构 WINDOWS **如程序清单** 5. 27 **所示**。

程序清单 5. 27 **窗口数据结构**

```
/* 定义窗口数据结构 */
typedef  struct
{  uint32  x;              // 窗口位置(左上角的 x 坐标)
   uint32  y;              // 窗口位置(左上角的 y 坐标)

   uint32  with;           // 窗口宽度
   uint32  hight;          // 窗口高度

   uint8   *title;         // 定义标题栏指针 (标题字符为 ASCII 字符串，最大个数受窗口限制)
   uint8   *state;         // 定义状态栏指针 (若为空时则不显示状态栏)
} WINDOWS;
```

表 5. 5 **图标菜单操作函数**(MENU.C)

函数原型	参数	功能
void GUI_Button49x14(uint32 x, uint32 y, uint8 *dat)	x，y 显示按钮起始坐标 *dat 按钮的点阵数据	显示按钮(按钮大小为 49*14)

续表 5.5

函数原型	参数	功能
void GUI_Button_OK(uint32 x, uint32 y)	x，y 显示按钮起始坐标	显示 OK 按钮
void GUI_Button_OK1(uint32 x, uint32 y)	x，y 显示按钮起始坐标	显示选中状态的 OK 按钮
void GUI_Button_Cancle(uint32 x, uint32 y)	x，y 显示按钮起始坐标	显示 CANCLE 按钮
void GUI_Button_Cancle1(uint32 x, uint32 y)	x，y 显示按钮起始坐标	显示选中状态的 CANCLE 按钮
uint8 GUI_MenuIcoDraw(MENUICO *ico)	*ico 图标菜单句柄	显示图标菜单

图标菜单数据结构 MENUICO 如程序清单 5. 28 所示。

程序清单 5. 28 图标菜单数据结构

```
/* 定义图标菜单数据结构 */
typedef  struct
{  uint32  x;                    // 图标菜单位置(左上角的 x 坐标)
   uint32  y;                    // 图标菜单位置(左上角的 y 坐标)
   uint8   *icodat;              // 32*32 的 ICO 数据地址
   uint8   *title;               // 相关标题提示 (42*13)
   uint8   state;                // 图标菜单状态，为 0 时表示未选中，为 1 时表示已选中

   void    (*Function)(void);    // 对应的服务程序
} MENUICO;
```

表 5. 6 下拉菜单操作函数(MENU.C)

函数原型	参数	功能
uint8 GUI_MMenuDraw(MMENU *men)	*men 主菜单句柄	显示主菜单
void GUI_MMenuSelect(MMENU *men, uint8 no)	*men 主菜单句柄 no 选中的主菜单项	选择主菜单项(no 为 0～n)
void GUI_MMenuNSelect(MMENU *men, uint8 no)	*men 主菜单句柄 no 取消选中的主菜单项	取消主菜单项的选择(no 为 0～n)
uint8 GUI_SMenuDraw(SMENU *men)	*men 子菜单句柄	显示子菜单
void GUI_SMenuSelect(SMENU *men, uint8 old_no, uint8 new_no)	*men 子菜单句柄 old_no 原选中的子菜单项 new_no 新选中的子菜单项	选择子菜单项(选取消 old_no 项的选中状态，然后选择 new_no 项)
uint8 GUI_SMenuHide(SMENU *men)	*men 子菜单句柄	消隐子菜单

下拉式菜单数据结构 MMENU、SMENU 以及菜单规格如程序清单 5. 29 所示。

程序清单 5. 29 下拉式菜单数据结构

```
/* 定义主菜单宽度，及最大菜单个数 */
#define  MMENU_WIDTH              34
#define  MMENU_NO                 6

/* 定义菜单的宽度(下拉菜单)，及最大子菜单个数 */
#define  SMENU_WIDTH              66
#define  SMENU_NO                 8
```

```
/* 定义一子菜单项的数据结构 */
typedef  struct
{  WINDOWS  *win;                          // 所属窗口
   uint8    mmenu_no;                      // 对应的主菜单项号(0-n)

   uint8    no;                            // 子菜单项个数
   char     *str[SMENU_NO];                // 子菜单字符串
   uint8    state;                         // 所选择的子菜单

   void     (*Function[SMENU_NO])(void);   // 子菜单对应的服务程序
} SMENU;

/* 主菜单数据结构 */
typedef  struct
{  WINDOWS  *win;                          // 所属窗口

   uint8    no;                            // 主菜单个数
   char     *str[MMENU_NO];                // 主菜单字符串
} MMENU;
```

表 5. 7 图形及汉字操作函数(LOADBIT.C)

函数原型	参数	功能
void GUI_LoadPic (uint32 x, uint32 y, uint8 *dat, uint32 hno, uint32 lno)	x，y 显示的起始坐标 *dat 点阵数据指针 hno，lno 图形的行/列数	显示单色点阵图形
void GUI_LoadPic1 (uint32 x, uint32 y, uint8 *dat, uint32 hno, uint32 lno)	x，y 显示的起始坐标 *dat 点阵数据指针 hno，lno 图形的行/列数	显示单色点阵图形(反相显示)
void GUI_PutHZ(uint32 x, uint32 y, uint8 *dat, uint8 hno, uint8 lno)	x，y 显示的起始坐标 *dat 点阵数据指针 hno，lno 汉字的行/列数	汉字显示

表 5. 8 5×7 ASCII 码字符显示函数(FONT5_7.C)

函数原型	参数	功能
uint8 GUI_PutChar(uint32 x, uint32 y, uint8 ch)	x，y 字符显示的坐标 ch 字符的十六进制值	显示一个字符(返回值为 1 时表示操作成功，为 0 表示失败)
void GUI_PutString(uint32 x, uint32 y, char *str)	x，y 显示的起始坐标 *str 指向字符串的指针	显示一字符串(以'\0'结束，没有自动换行功能)
void GUI_PutNoStr(uint32 x, uint32 y, char *str, uint8 no)	x，y 显示的起始坐标 *str 指向字符串的指针 no 要显示字符的个数	显示字符串中指定个数的字符

表 5. 9 8×8 ASCII 码字符显示函数(FONT8_8.C)

函数原型	参数	功能
uint8 GUI_PutChar8_8(uint32 x, uint32 y, uint8 ch)	x，y 字符显示的坐标 ch 字符的十六进制值	显示一个字符(返回值为 1 时表示操作成功，为 0 表示失败)
void GUI_PutString8_8(uint32 x, uint32 y, char *str)	x，y 显示的起始坐标 *str 指向字符串的指针	显示一字符串(以'\0'结束，没有自动换行功能)
void GUI_PutNoStr8_8(uint32 x, uint32 y, char *str, uint8 no)	x，y 显示的起始坐标 *str 指向字符串的指针 no 要显示字符的个数	显示字符串中指定个数的字符

表 5. 10 24×32 ASCII 数字字符显示函数(FONT24_32.C)

函数原型	参数	功能
uint8 GUI_PutChar24_32(uint32 x, uint32 y, uint8 ch)	x，y 字符显示的坐标 ch 字符的十六进制值	显示一个字符(返回值为 1 时表示操作成功，为 0 表示失败)

表 5. 11 颜色转换操作函数(CONVERTCOLOR.C)

函数原型	参数	功能
uint16 GUI_Color2Index_565(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 64K 色索引值
uint32 GUI_Index2Color_565(uint16 index)	index 索引颜色值	64K 色索引值 → RGB 颜色值
uint16 GUI_Color2Index_555(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 32K 色索引值
uint32 GUI_Index2Color_555(uint16 index)	index 索引颜色值	32K 色索引值 → RGB 颜色值
uint16 GUI_Color2Index_444(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 4K 色索引值
uint32 GUI_Index2Color_444(uint16 index)	index 索引颜色值	4K 色索引值 → RGB 颜色值
uint8 GUI_Color2Index_332(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 256 色索引值
uint32 GUI_Index2Color_233(uint8 index)	index 索引颜色值	256 色索引值 → RGB 颜色值
uint8 GUI_Color2Index_222(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 64 色索引值
uint32 GUI_Index2Color_222(uint8 index)	index 索引颜色值	64 色索引值 → RGB 颜色值
uint8 GUI_Color2Index_111(uint32 colorRGB)	colorRGB RGB 颜色值	RGB 颜色值 → 8 色索引值
uint32 GUI_Index2Color_111(uint8 Index)	index 索引颜色值	8 色索引值 → RGB 颜色值

5.5 ZLG/GUI 应用例程

5.5.1 如何使用 ZLG/GUI

使用 ZLG/GUI 可按以下步骤进行。

- 使用 ADS 1.2 新建一个工程。使用 LPC2200 的工程模板来建立用户的工程，比如 D:\EasyARM2200\GUI_TEST1。
- 复制 ZLG/GUI 文件。把 ZLG_GUI 整个目录及文件复制工程目录下，比如 D:\EasyARM2200\GUI_TEST1\SRC(从光盘上复制文件后，需要将 ZLG_GUI 目录下所有文件的只读属性去掉)。

- 添加 ZLG/GUI 文件。在 ADS 1.2 的项目管理窗口中新建一个组 ZLG/GUI，然后在这个组内添加 D:\EasyARM2200\GUI_TEST1\SRC\ZLG_GUI 下的所有 C 源程序文件和 GUI_CONFIG.H 配置文件。
- 新建驱动程序，并添加到项目中。驱动程序文件名比如：LCDDRIVE.c、LCDDRIVE.H，可以保存在 D:\EasyARM2200\GUI_TEST1\SRC 目录下。
- 修改 CONFIG.H。修改 D:\EasyARM2200\GUI_TEST1\SRC 目录下的 CONFIG.H，增加包含 LCDDRIVE.H 头文件和 ZLG/GUI 的所有头文件。
- 修改 GUI_CONFIG.H。修改 D:\EasyARM2200\GUI_TEST1\SRC\ZLG_GUI 目录下的 GUI_CONFIG.H 文件，裁剪 ZLG/GUI。
- 新建并添加用户文件，并在起动文件中设置 CPU 初始化部份代码(如总线配置或 GPIO 配置)。在用户文件中加入 config.h 文件，先调用初始化函数 GUI_Initialize()，然后即可进行画图操作。

5.5.2 汉字数据的生成

ZLG/GUI 并没有集成汉字字库，这使得其代码尺寸大大缩小，但当用户使用汉字显示时，就需要提供所要显示汉字的点阵数据。当然，图标、图形显示同样需要相关的点阵数据。通常，用户可以使用软件生成的汉字、图形的点阵数据，只要正确配置软件的设置，即可生成适用的点阵数据。

这里介绍的是字模软件 V1.0，软件支持汉字取模(即取点阵数据)，操作简单项，灵活性较强。字模软件 V1.0 主界面如图 5. 37 所示。

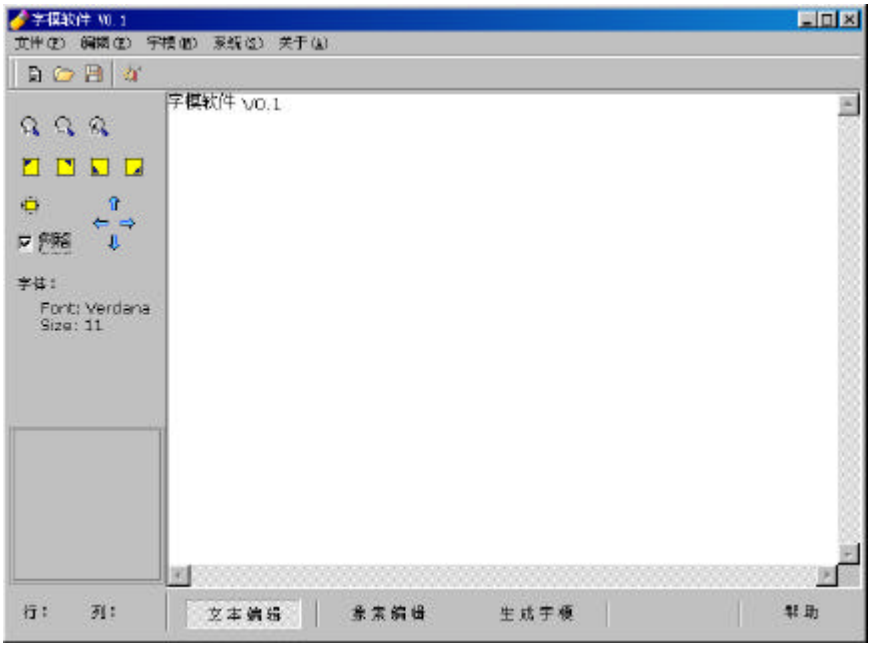

图 5. 37 字模软件主界面

输入汉字

输入汉字到字模软件有两种方法，一种是在文本编辑窗中直接输入文字，另一种是用【文件】->【打开】打开一个已编好的文本文件。文本编辑窗口如图 5. 38 所示。

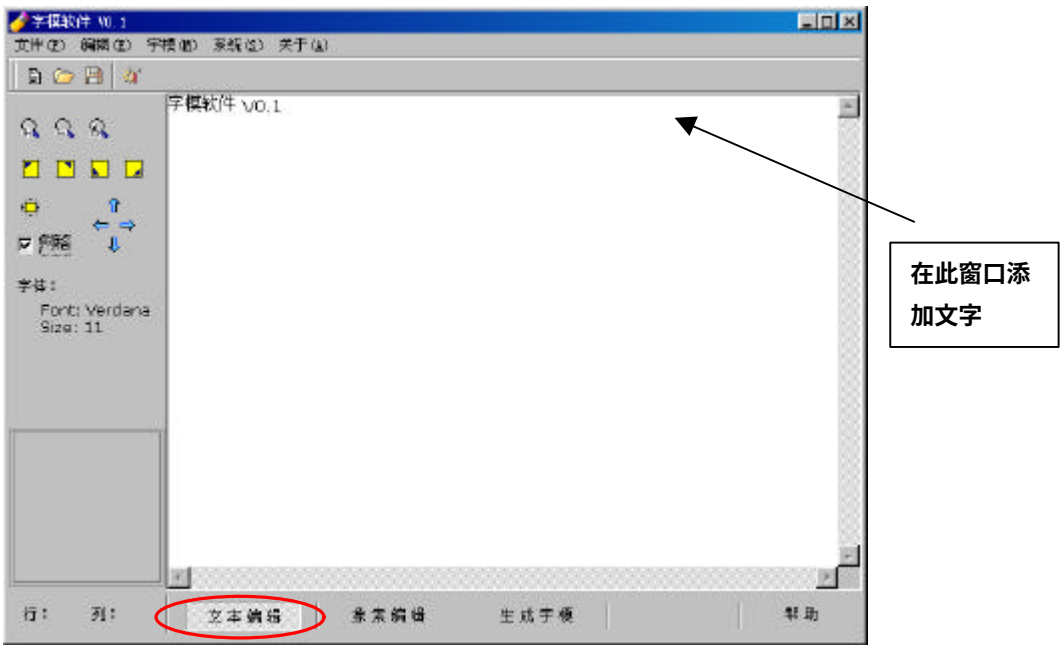

图 5. 38 字模软件文本编辑窗口

汉字字体设置

打开【系统】->【字体设置 ..】，弹出字体设置窗口，如图 5. 39 **所示。字体设置对编辑出窗口中的所有文字均有效。**

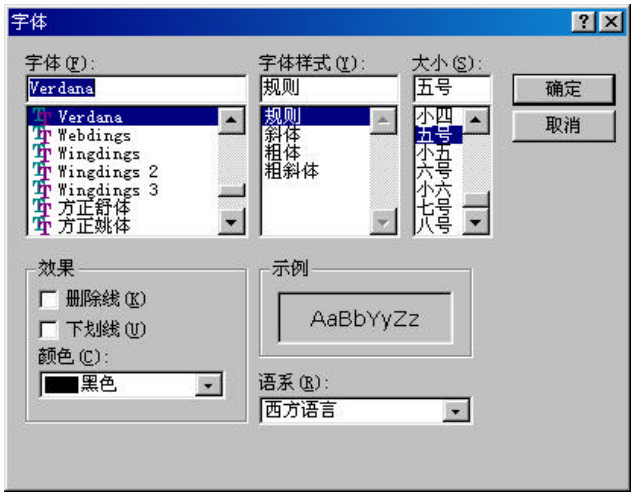

图 5. 39 字体设置窗口

字模生成设置

对于不同的液晶模块或不同的显示驱动程序，它们需要的字模格式有所不同，字模软件可以设置 4 **种取模格式，支持** C **语言格式及汇编格式。打开【系统】->【选项 ..】即弹出参数设置窗口，注释及数组命名设置窗口如图** 5. 40 **所示，字模格式设置窗口如图** 5. 41 **所示。**

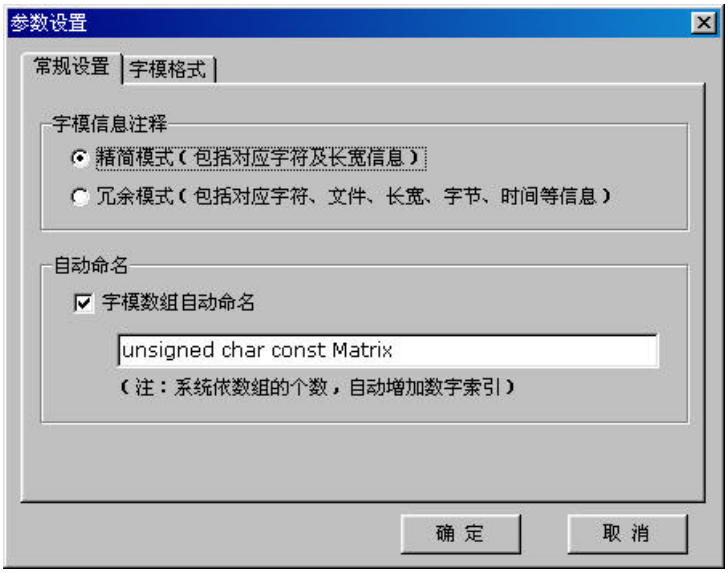

图 5. 40 参数设置窗口–注释及数组命名

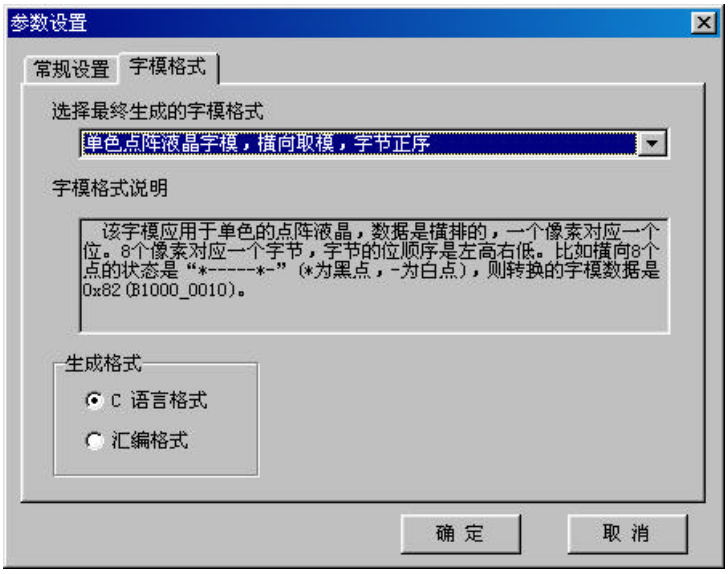

图 5. 41 参数设置窗口–字模格式

生成字模

在字模软件主窗口中，点击“生成字模”按钮，或打开【字模】->【后成字模】即可产生相应的字模数据，如图 5. 42 所示。此时可以将这数据选中，然后复制到用户的程序当中，还可以使用【文件】->【保存..】将转换结果保存。

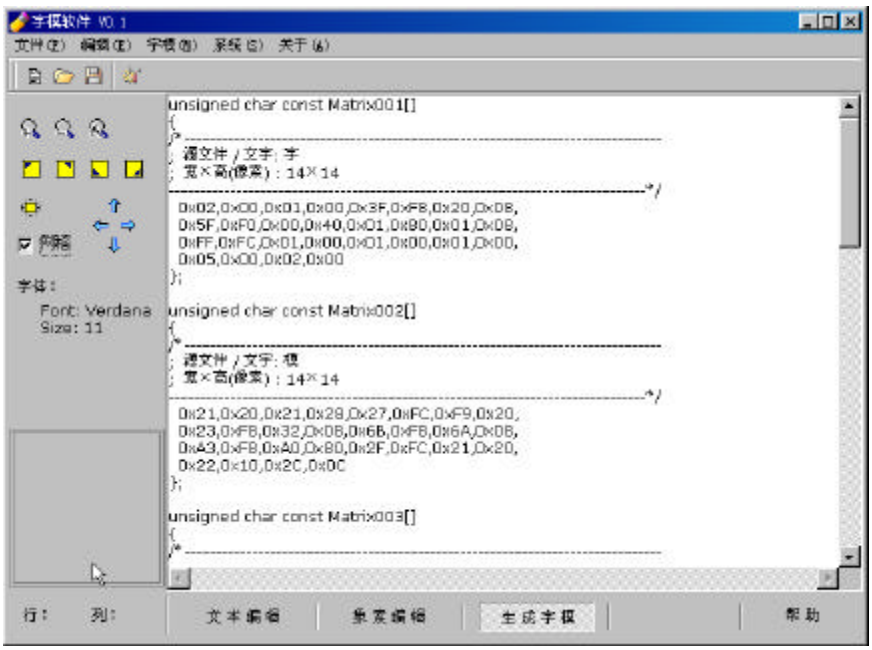

图 5. 42 生成字模窗口

按像素生成字模

由文本编辑窗口直接生成的字模数据是按汉字来分组的，即每一个汉字对应一个数组，使用时需要将每一个汉字的数据传递给显示驱动程序。字模软件提供了一个像素编辑功能，可用于编辑特殊文字或图形，用此功能生成的字模只有一个数组，方便程序编写。先在文本编辑窗口中输入相应文字，点击“象素编辑”即启动像素编辑窗口，如图 5. 43 所示。

用户可以使用编辑工具进行编辑，比如放大/缩小显示、左上角对齐等，还可以用鼠标单击相应的像素点，设置该点是否显示。另外，【编辑】菜单中有几个功能项，可以用于增加/删除一行像素菜。

图 5. 43 像素编辑窗口

点击“生成字模”，即可按像素生成字模，如图 5. 44 所示。

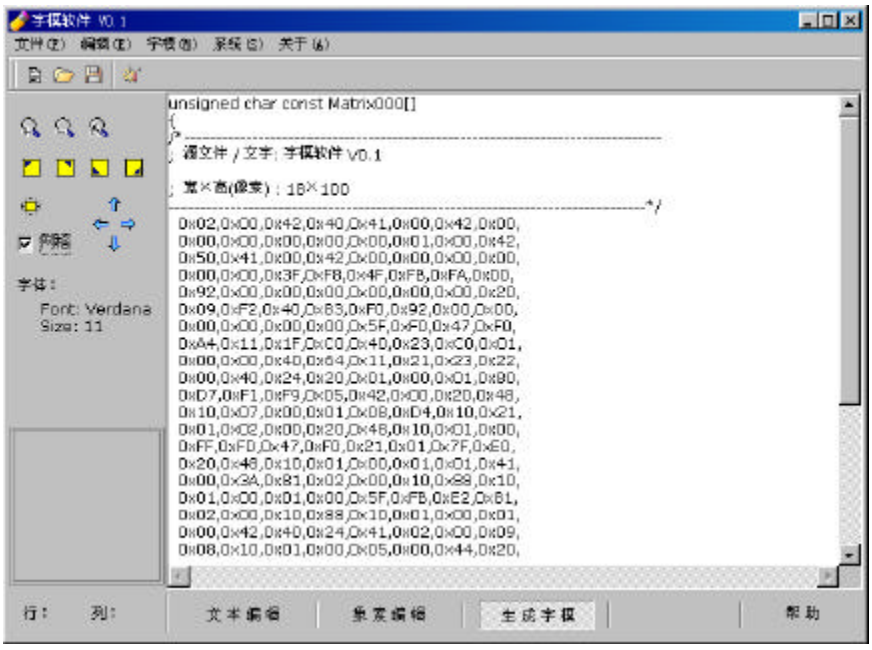

图 5. 44 按像素生成字模窗口

5.5.3 ZLG/GUI 应用实例

在使用 ZLG/GUI 时，需要一个包含其它头文件及系统配置的文件 CONFIG.H，这样做是为了方便项目的管理，源程序中只要包含此文件即可使用系统的函数或配置。CONFIG.H 文件代码如程序清单 5. 30 所示。

其中，GUI_CONFIG.H 为 ZLG/GUI 配置文件；LCDDRIVE.H 为液晶驱动程序(即硬件驱动层接口函数)的头文件；GUI_BASIC.H 为基本图形操作程序的头文件；GUI_STOCKC.H 为颜色转换操作程序的头文件；FONT_MACRO.H 定义了字节点阵宏；FONT5_7.H 为 5×7 ASCII 码字符显示程序的头文件；FONT8_8.H 为 8×8 ASCII 码字符显示程序的头文件；FONT24_32.H 为 24×32 数字字符显示程序的头文件；LOADBIT.H 为单色图形及汉字显示程序的头文件；WINDOWS.H 为窗口管理功能的头文件；MENU.H 为下拉式菜单、按钮及图标菜单功能的头文件。

程序清单 5. 30 CONFIG.H 文件

```
#ifndef     TRUE
#define     TRUE      1
#endif

#ifndef     FALSE
#define     FALSE     0
#endif

#ifndef     NULL
#define     NULL      0
#endif
```

```
typedef     unsigned char     uint8;          /* 无符号 8 位整型变量                  */
typedef     signed char       int8;           /* 有符号 8 位整型变量                  */
typedef     unsigned short    uint16;         /* 无符号 16 位整型变量                 */
typedef     signed short      int16;          /* 有符号 16 位整型变量                 */
typedef     unsigned int      uint32;         /* 无符号 32 位整型变量                 */
typedef     signed int        int32;          /* 有符号 32 位整型变量                 */
typedef     float             fp32;           /* 单精度浮点数（32 位长度）            */
typedef     double            fp64;           /* 双精度浮点数（64 位长度）            */

/***********************************/
/*          ARM 的特殊代码                 */
/***********************************/
//这一段无需改动
#include      "LPC2294.h"

/***********************************/
/*        应用程序配置                     */
/***********************************/
//以下根据需要改动
#include      "GUI_CONFIG.H"
#include      "LCDDRIVE.H"
#include      "GUI_BASIC.H"
#include      "GUI_STOCKC.H"
#include      "FONT_MACRO.H"
#include      "FONT5_7.H"
#include      "FONT8_8.H"
#include      "FONT24_32.H"
#include      "LOADBIT.H"
#include      "WINDOWS.H"
#include      "MENU.H"
…
```

1. 驱动程序的编写

为了能够使用 ZLG/GUI 对不同的图形液晶操作，需要用户定义液晶大小并编写相关的驱动程序，并提供九个硬件驱动层接口函数。例子如程序清单 5. 31、程序清单 5. 32 所示，这是对 SED1353 控制的 256 色伪彩液晶，点像素为 320×240，使用 RAM 建立显示缓冲区 gui_disp_buf。

程序清单 5. 31 256 色伪彩液晶(320×240)参数定义—LCDDRIVE.H

```
/* 定义颜色数据类型(可以是数据结构) */
#define   TCOLOR                    uint8

/* 定义 LCM 像素数宏 */
```

```
#define   GUI_LCM_XMAX          320                    /* 定义液晶 x 轴的像素数 */
#define   GUI_LCM_YMAX          240                    /* 定义液晶 y 轴的像素数 */
…
```

程序清单 5. 32 **256 色伪彩液晶(320×240)硬件驱动层接口函数—LCDDRIVE.C**

```
#include "config.h"

/* 定义显示缓冲区(可根据情况定义或直接使用 LCM 显示存储空间) */
TCOLOR   gui_disp_buf[GUI_LCM_YMAX][GUI_LCM_XMAX];
…
/*****************************************************************************
* 名称：GUI_FillSCR()
* 功能：全屏填充。直接使用数据填充显示缓冲区。
* 入口参数：dat              填充的数据
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
void   GUI_FillSCR(TCOLOR dat)
{   uint32   i, j;

    /* 填充缓冲区 */
    for(i=0; i<GUI_LCM_YMAX; i++)          // 历遍所有行
    {   for(j=0; j<GUI_LCM_XMAX; j++)      // 历遍所有列
        {   gui_disp_buf[i][j] = dat;
        }
    }

    /* 填充 LCM */
    LCD_FillAll(dat);
}

/*****************************************************************************
* 名称：GUI_Initialize()
* 功能：初始化 GUI，包括初始化显示缓冲区，初始化 LCM 并清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*****************************************************************************/
void   GUI_Initialize(void)
{   LCD_Initialize();                          // 初始化 LCM 模块工作模式，纯图形模式
    GUI_FillSCR(0x00);                         // 初始化缓冲区为 0x00，并输出屏幕(清屏)
}

/*****************************************************************************
```

```
* 名称：GUI_ClearSCR()
* 功能：清屏。
* 入口参数：无
* 出口参数：无
* 说明：用户根据 LCM 的实际情况编写此函数。
*********************************************************************************/
void   GUI_ClearSCR(void)
{   GUI_FillSCR(0x00);
}

/*********************************************************************************
* 名称：GUI_Point()
* 功能：在指定位置上画点。
* 入口参数：  x          指定点所在列的位置
*             y          指定点所在行的位置
*             color      显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：返回值为 1 时表示操作成功，为 0 时表示操作失败。(操作失败原因是指定
*           地址超出有效范围)
* 说明：用户根据 LCM 的实际情况编写此函数。对于单色，只有一个位有效，则要使用
*       左移的方法实现 point_dat = (point_dat&MASK_TAB [i]) | (color<<n)，其它位数的
*       一样处理。
*********************************************************************************/
uint8   GUI_Point(uint32 x, uint32 y, TCOLOR color)
{   /* 参数过滤 */
    if(x>=GUI_LCM_XMAX) return(0);
    if(y>=GUI_LCM_YMAX) return(0);

    /* 设置缓冲区相应的点 */
    gui_disp_buf[y][x] = color;

    /* 刷新显示 */
    LCD_UpdatePoint(x, y);
    return(1);
}

/*********************************************************************************
* 名称：GUI_ReadPoint()
* 功能：读取指定点的颜色。
* 入口参数：x          指定点所在列的位置
*           y          指定点所在行的位置
*           ret        保存颜色值的指针
* 出口参数：返回 0 时表示指定地址超出有效范围。
* 说明：对于单色，设置 ret 的 d0 位为 1 或 0，4 级灰度则为 d0、d1 有效，8 位 RGB 则
*       d0--d7 有效，RGB 结构则 R、G、B 变量有效。
```

```
*******************************************************************************/
int    GUI_ReadPoint(uint32 x, uint32 y, TCOLOR *ret)
{   /* 参数过滤 */
    if(x>=GUI_LCM_XMAX) return(0);
    if(y>=GUI_LCM_YMAX) return(0);

    /* 取得该点颜色(用户自行更改) */
    *ret = gui_disp_buf[y][x];

    return(1);
}

/*******************************************************************************
* 名称：GUI_HLine()
* 功能：画水平线。
* 入口参数：  x0          水平线起点所在列的位置
*             y0          水平线起点所在行的位置
*             x1          水平线终点所在列的位置
*             color       显示颜色(对于黑白色 LCM，为 0 时灭，为 1 时显示)
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*       可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*******************************************************************************/
void   GUI_HLine(uint32 x0, uint32 y0, uint32 x1, TCOLOR color)
{   uint32   bak;

    if(x0>x1)                                   // 对 x0、x1 大小进行排列，以便画图
    {   bak = x1;
        x1 = x0;
        x0 = bak;
    }

    do
    {   GUI_Point(x0, y0, color);               // 逐点显示，描出垂直线
        x0++;
    }while(x1>=x0);
}

/*******************************************************************************
* 名称：GUI_RLine()
* 功能：画垂直线。
* 入口参数：  x0          垂直线起点所在列的位置
*             y0          垂直线起点所在行的位置
*             y1          垂直线终点所在行的位置
```

```
*                color    显示颜色
* 出口参数：无
* 说明：对于单色、4 级灰度的液晶，可通过修改此函数作图提高速度，如单色 LCM，
*          可以一次更新 8 个点，而不需要一个点一个点的写到 LCM 中。
*******************************************************************************/
void   GUI_RLine(uint32 x0, uint32 y0, uint32 y1, TCOLOR color)
{   uint32   bak;

    if(y0>y1)                                   // 对 y0、y1 大小进行排列，以便画图
    {   bak = y1;
        y1 = y0;
        y0 = bak;
    }
    do
    {   GUI_Point(x0, y0, color);               // 逐点显示，描出垂直线
        y0++;
    }while(y1>=y0);
}

/*******************************************************************************
* 名称：GUI_CmpColor()
* 功能：判断颜色值是否一致。
* 入口参数：color1        颜色值 1
*           color2        颜色值 2
* 出口参数：返回 1 表示相同，返回 0 表示不相同。
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写比较函数。
*******************************************************************************/
int   GUI_CmpColor(TCOLOR color1, TCOLOR color2)
{   if(color1==color2) return(1);
      else   return(0);
}

/*******************************************************************************
* 名称：GUI_CopyColor()
* 功能：颜色值复制。
* 入口参数：color1        目标颜色变量
*           color2        源颜色变量
* 出口参数：无
* 说明：由于颜色类型 TCOLOR 可以是结构类型，所以需要用户编写复制函数。
*******************************************************************************/
void   GUI_CopyColor(TCOLOR *color1, TCOLOR color2)
{   *color1 = color2;
}
```

如程序清单 5. 33 所示为 16 标准色颜色值定义，注释为 RGB 颜色的 R 值、G 值、B 值。

程序清单 5. 33 16 标准色颜色值定义

```
/* 设置颜色宏定义 */
#define   BLACK     0x00    /* 黑色:    0,   0,   0   */
#define   NAVY      0x02    /* 深蓝色:  0,   0,   128 */
#define   DGREEN    0x10    /* 深绿色:  0,   128, 0   */
#define   DCYAN     0x12    /* 深青色:  0,   128, 128 */
#define   MAROON    0x80    /* 深红色:  128, 0,   0   */
#define   PURPLE    0x82    /* 紫色:    128, 0,   128 */
#define   OLIVE     0x90    /* 橄榄绿:  128, 128, 0   */
#define   LGRAY     0xCA    /* 灰白色:  192, 192, 192 */
#define   DGRAY     0x92    /* 深灰色:  128, 128, 128 */
#define   BLUE      0x03    /* 蓝色:    0,   0,   255 */
#define   GREEN     0x1C    /* 绿色:    0,   255, 0   */
#define   CYAN      0x1F    /* 青色:    0,   255, 255 */
#define   RED       0xE0    /* 红色:    55,  0,   0   */
#define   MAGENTA   0xE3    /* 品红:    255, 0,   255 */
#define   YELLOW    0xFC    /* 黄色:    255, 255, 0   */
#define   WHITE     0xFF    /* 白色:    255, 255, 255 */
```

2. 基本作图

将 ZLG/GUI 的文件加入到项目中，然后在用户程序中加入包含 CONFIG.H 的语句，即可调用 ZLG/GUI 的 API 函数进行画图，以下以画直线、圆弧、多边形及多图形填充为例进行说明。

画直线可以使用 GUI_RLine、GUI_Hline 和 GUI_Line 三个函数，调用时要提供直线起点和终点的绝对坐标值，以及显示的颜色，对于画水平线和垂直线直接使用 GUI_RLine、GUI_Hline 会得到更好的效率。当要画具有线宽要求的直线时，使用 GUI_LineWith 函数。画直线使用的例子如程序清单 5. 34 所示。

程序清单 5. 34 画直线演示

```
/*****************************************************************************
* 名称：DemoLine()
* 功能：绘画直线演示。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  DemoLine(void)
{  WINDOWS  demow;

   /* 显示演示窗口 */
   demow.x = 45;
   demow.y = 25;
   demow.with = 150;
```

```
    demow.hight = 80;
    demow.title = (uint8 *) "Line for Demo";
    demow.state = (uint8 *) "Enter a key return.";
    GUI_WindowsDraw(&demow);

    /* 打钩 */
    GUI_Line(90, 60, 110, 90, GREEN);
    GUI_Line(110, 90, 190, 50, GREEN);
    GUI_LineWith(50, 60, 70, 90, 3, GREEN);
    GUI_LineWith(70, 90, 170, 40, 3, GREEN);

    /* 画垂直线及水平线 */
    GUI_RLine(70, 45, 70, RED);
    GUI_HLine(140, 85, 180, RED);

    WaitAKey();                                    // 等待一按键
    GUI_WindowsHide(&demow);
}
```

画任意圆弧是使用 GUI_Arc **函数，调用时使用的参数是圆弧的圆心、半径、起始角度及终止角度，企图使用** GUI_Arc **画一个圆是错误的。若要画** 1/4 **圆弧，使用** GUI_Arc4 **函数会获得更好的效率。画圆弧函数使用的例子如程序清单** 5. 35 **所示。**

程序清单 5. 35 **画圆弧演示**

```
/*****************************************************************************
* 名称：DemoCircle()
* 功能：绘画圆演示。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void    DemoCircle(void)
{   WINDOWS    demow;

    /* 显示演示窗口 */
    demow.x = 45;
    demow.y = 25;
    demow.with = 150;
    demow.hight = 80;
    demow.title = (uint8 *) "Circle for Demo";
    demow.state = (uint8 *) "Enter a key return.";
    GUI_WindowsDraw(&demow);

    /* 画两个圆 */
    GUI_CircleFill(75, 60, 15, GREEN);
```

```
    GUI_Circle(100, 70, 5, GREEN);

    /* 画两个交圆，并对交点进行填充 */
    GUI_Circle(135, 65, 20, YELLOW);
    GUI_Circle(155, 65, 20, GREEN);
    GUI_FloodFill(145, 65, RED);

    WaitAKey();                                    // 等待一按键
    GUI_WindowsHide(&demow);
}
```

画多边形时，要提供多边形的端点坐标数据（如程序清单 5. 36 **的** mline、poly5、poly4 **数组），然后调用** GUI_LineS **函数即可把多边形的各个端点连接起来，只有在起始端点和终止端点为同一点时，画出的图形才是完全封闭的。使用填充函数** GUI_FloodFill，**需要指出要填充区域内的任意一点，并提供填充颜色。画多边形和多边形填充的使用例子如程序清单** 5. 36 **所示。**

程序清单 5. 36 **画多边形及填充演示**

```
uint32   const mline[] = {120,40, 110,55, 90,60, 110,70, 120,90, 130,70, 150,60, 130,55, 120,40};
uint32   const poly5[] = {65,45, 50,60, 50,90, 80,90, 80,60, 65,45};
uint32   const poly4[] = {155,50, 190,50, 155,85, 190,85, 155,50};
/*******************************************************************************
* 名称：DemoPoly()
* 功能：绘画多边形演示。
* 入口参数：无
* 出口参数：无
*******************************************************************************/
void   DemoPoly(void)
{   WINDOWS   demow;

    /* 显示演示窗口 */
    demow.x = 45;
    demow.y = 25;
    demow.with = 150;
    demow.hight = 80;
    demow.title = (uint8 *) "Poly for Demo";
    demow.state = (uint8 *) "Enter a key return.";
    GUI_WindowsDraw(&demow);

    GUI_LineS(poly5, 6, GREEN);
    GUI_LineS(mline, 9, RED);
    GUI_FloodFill(115, 55, GREEN);
    GUI_LineS(poly4, 5, RED);
    GUI_FloodFill(160, 52, RED);
```

```
    WaitAKey();                                 // 等待一按键
    GUI_WindowsHide(&demow);
}
```

3. 画窗口

窗口的使用前要先定义一个窗口的数据结构，如程序清单 5. 37 **中的** mainwindows，**设置窗口的起始坐标、窗口的大小、窗口的标题等相关参数后，即可调用** GUI_WindowsDraw **进行显示输出。图标菜单也需要定义相关的数据结构，如程序清单** 5. 37 **中的** mainmenu，**将每一个图标菜单项的显示坐标地址、图标的数据指针、对应的服务函数等进行设置后，即可调用** GUI_MenuIcoDraw **实现显示输出。另外，主程序需要先调用** GUI_SetColor()**函数来设置前景色及背景色。画窗口和图标菜单应用的例子如程序清单** 5. 37 **所示。**

程序清单 5. 37 **窗口及图标菜单演示**

```
/*****************************************************************************
* 名称：RunDemo()
* 功能：进行 GUI 的演示，包括画线、画圆、圆弧、椭圆、矩形、多边形、bmp 图及
*       汉字、动画等。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  RunDemo(void)
{  WINDOWS  mainwindows;
   MENUICO  mainmenu[8];
   uint8  select;                               // 菜单选项变量
   uint8  key;
   uint8  i;

   /* 设置主窗口并显示输出 */
   mainwindows.x = 0;
   mainwindows.y = 0;
   mainwindows.with = 320;
   mainwindows.hight = 240;
   mainwindows.title = (uint8 *) "Graphics Function  Demo!";
   mainwindows.state = NULL;
   GUI_WindowsDraw(&mainwindows);               // 绘制主窗口

   /* 设置图标菜单 */
   for(i=0; i<8; i++)
   {  if(i<4)
      {  mainmenu[i].x = i*59+11;
         mainmenu[i].y = 20;
      }
      else
```

```
    {   mainmenu[i].x = (i-4)*59+11;
        mainmenu[i].y = 72;
    }
    mainmenu[i].state = 0;                          // 初始化为未选中状态
}

/* 连接相应的图标数据 */
mainmenu[0].icodat = (uint8 *) menuico1;
mainmenu[0].title = (uint8 *) menuchar1;
mainmenu[1].icodat = (uint8 *) menuico2;
mainmenu[1].title = (uint8 *) menuchar2;
mainmenu[2].icodat = (uint8 *) menuico3;
mainmenu[2].title = (uint8 *) menuchar3;
mainmenu[3].icodat = (uint8 *) menuico4;
mainmenu[3].title = (uint8 *) menuchar4;
mainmenu[4].icodat = (uint8 *) menuico5;
mainmenu[4].title = (uint8 *) menuchar5;
mainmenu[5].icodat = (uint8 *) menuico6;
mainmenu[5].title = (uint8 *) menuchar6;
mainmenu[6].icodat = (uint8 *) menuico7;
mainmenu[6].title = (uint8 *) menuchar7;
mainmenu[7].icodat = (uint8 *) menuico8;
mainmenu[7].title = (uint8 *) menuchar8;

/* 连接菜单功能函数 */
mainmenu[0].Function = (void(*)())DemoLine;
mainmenu[1].Function = (void(*)())DemoCircle;
mainmenu[2].Function = (void(*)())DemoArc;
mainmenu[3].Function = (void(*)())DemoEllipse;
mainmenu[4].Function = (void(*)())DemoRectang;
mainmenu[5].Function = (void(*)())DemoPoly;
mainmenu[6].Function = (void(*)())DemoBmp;
mainmenu[7].Function = (void(*)())MoveCircle;

/* 设置默认菜单 */
mainmenu[0].state = 1;
select = 0;

/* 进行主菜单显示及选择操作 */
while(1)
{   for(i=0; i<8; i++)                              // 显示图标菜单
    {   GUI_MenuIcoDraw(&mainmenu[i]);
    }
    /* 选择功能 */
```

```
        while(1)
        {   key = WaitKey();
            if(key==KEY_OK) break;                          // 点击 OK 键选择
            if(key==KEY_NEXT)
            {   mainmenu[select].state = 0;                 // 取消上一选择
                GUI_MenuIcoDraw(&mainmenu[select]);
                select++;                                   // 指向下一菜单
                if(select>7) select=0;
                  mainmenu[select].state = 1;
                GUI_MenuIcoDraw(&mainmenu[select]);
            }
            if(key==KEY_BACK)
            {   mainmenu[select].state = 0;                 // 取消上一选择
                GUI_MenuIcoDraw(&mainmenu[select]);
                if(select==0) select=7;
                  else  select--;                           // 指向下一菜单
                  mainmenu[select].state = 1;
                GUI_MenuIcoDraw(&mainmenu[select]);
            }
        }

        /* 执行相应功能 */
        (*mainmenu[select].Function)();
        if(select>=6) GUI_WindowsDraw(&mainwindows);    // 主窗口重绘(清屏)
    }

}
```

4. 图形、汉字显示

图形及汉字的显示操作前，要将相应的点阵数据放到一个数组中，然后调用 GUI_LoadBmp 及 GUI_PutHZ 即可，显示汉字时要指定汉字的行、列规格，并要自行处理字与字之间的间隙控制。使用图形及汉字的例子如程序清单 5. 38 所示。

程序清单 5. 38 图形及汉字显示演示

```
/*--  调入了一幅图像：C:\WINDOWS\Desktop\Critter2.ico  --*/
/*--  宽度 x 高度=32x32  --*/
uint8  const  critter[]=
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x03,0xF0,0x00,0x00,0x07,0xFC,0x00,0x00,0x0C,0x9F,0x00,0x00,0x1A,0xCF,0x80,0x00,
0x1C,0x8F,0x80,0x00,0x1F,0xFF,0xC0,0x00,0x1F,0xFF,0xE0,0x00,0x0F,0xFF,0xF0,0x00,
0x0F,0xFF,0xF8,0x00,0x07,0xFF,0xFC,0x00,0x03,0xFF,0xFF,0x00,0x01,0xFF,0xFF,0xC0,
0x00,0xFF,0xFF,0xC0,0x00,0xFF,0xFF,0xC0,0x1F,0xFF,0xFF,0xC0,0x3F,0xFF,0xFF,0xC0,
0x1F,0xFF,0xFF,0xC0,0x0F,0xFF,0xFF,0xC0,0x0F,0xFF,0xFF,0xC0,0x00,0x1F,0xFF,0xC0,
```

```
0x00,0x0F,0xFF,0x80,0x00,0x0F,0xFF,0x80,0x00,0x1F,0xFF,0x80,0x00,0x1F,0xFF,0x80,
0x00,0x3F,0xFF,0x80,0x00,0x7F,0x9F,0x00,0x00,0x7F,0x1F,0x00,0x00,0x7E,0x1F,0x00
};

/*******************************************************************************
* 名称：RunOpen()
* 功能：执行"Open"菜单命令，打开一个图标。
* 入口参数：无
* 出口参数：无
*******************************************************************************/
void    RunOpen(void)
{    GUI_LoadBmp(105, 70, (uint8 *)critter, 32, 32);
}

/*--  文字:  请  --*/
/*--  宋体 12;  此字体下对应的点阵为：宽 x 高=16x16    --*/
uint8   const   hzchar1[]=
{
0x00,0x40,0x47,0xFC,0x30,0x40,0x23,0xF8,0x00,0x40,0x07,0xFE,0xF0,0x00,0x13,0xF8,
0x12,0x08,0x13,0xF8,0x12,0x08,0x13,0xF8,0x16,0x08,0x1A,0x08,0x12,0x28,0x02,0x10
};

/*--  文字:  稍  --*/
/*--  宋体 12;  此字体下对应的点阵为：宽 x 高=16x16    --*/
uint8   const   hzchar2[]=
{
0x0E,0x20,0x79,0x22,0x08,0xA4,0x08,0xA8,0x7D,0xFC,0x19,0x04,0x1D,0x04,0x2B,0xFC,
0x29,0x04,0x49,0x04,0x49,0xFC,0x89,0x04,0x09,0x04,0x09,0x04,0x09,0x14,0x09,0x08
};

/*--  文字:  等  --*/
/*--  宋体 12;  此字体下对应的点阵为：宽 x 高=16x16    --*/
uint8   const   hzchar3[]=
{
0x20,0x80,0x3E,0xFC,0x28,0xA0,0x45,0x10,0x85,0x10,0x3F,0xF8,0x01,0x00,0xFF,0xFE,
0x00,0x20,0x00,0x20,0x7F,0xFC,0x04,0x20,0x02,0x20,0x02,0x20,0x00,0xA0,0x00,0x40
};


/*******************************************************************************
* 名称：DemoBmp()
* 功能：绘画 bmp 图及汉字显示演示。
* 入口参数：无
* 出口参数：无
```

```
*****************************************************************************/
void  DemoBmp(void)
{  GUI_ClearSCR();
   GUI_LoadBmp(0, 0, (uint8 *)windows, 168, 124);
   GUI_PutHZ(210, 20, (uint8 *)hzchar1, 16, 16);
   GUI_PutHZ(210, 50, (uint8 *)hzchar2, 16, 16);
   GUI_PutHZ(210, 80, (uint8 *)hzchar3, 16, 16);

   WaitAKey();                              // 等待一按键
}
```

5. 菜单操作

使用下拉菜单时，要先在 MENU.H 文件中设置主菜单、子菜单的宽度，然后定义主菜单的数据结构变量和子菜单的数据结构变量，进行主菜单、子菜单的显示字符及对应功能函数等相关设置，然后使用 GUI_MmenuSelect 先显示主菜单，当确定主菜单后即可调用 GUI_SmenuDraw 函数下拉显示子菜单，选择子菜单项时使用 GUI_SmenuSelect 进行反白显示。使用下拉菜单的例子如程序清单 5. 39 所示。

程序清单 5. 39 下拉菜单示演示

```
/*****************************************************************************
* 名称：RunMenuDemo()
* 功能：进行菜单的演示操作。
* 入口参数：无
* 出口参数：无
*****************************************************************************/
void  RunMenuDemo(void)
{  WINDOWS   mainwindows;
   MMENU      mainmenu;
   SMENU      submenu[5];
   uint8      mselect;
   uint8      select, bak;
   uint8      key;

   /* 设置主窗口并显示输出 */
   mainwindows.x = 0;
   mainwindows.y = 0;
   mainwindows.with = 240;
   mainwindows.hight = 128;
   mainwindows.title = (uint8 *) "Down Menu   Demo!";
   mainwindows.state = (uint8 *) "ready";
   GUI_WindowsDraw(&mainwindows);                    // 绘制主窗口

   mainmenu.win = &mainwindows;                      // 设置主菜单所属窗口
   mainmenu.no = 5;                                  // 主菜单项个数
```

```
mainmenu.str[0] = (char *) "File";                  // 主菜单项各项字符
mainmenu.str[1] = (char *) "Edit";
mainmenu.str[2] = (char *) "View";
mainmenu.str[3] = (char *) "Find";
mainmenu.str[4] = (char *) "Help";
GUI_MMenuDraw(&mainmenu);                           // 显示主菜单

/* 子菜单设置 */
for(bak=0; bak<5; bak++)
{   submenu[bak].win = &mainwindows;                // 设置菜单所属的窗口
    submenu[bak].mmenu_no = bak;
    submenu[bak].state = 0;
}

submenu[0].no = 3;
submenu[0].str[0] = (char *) "Open";
submenu[0].str[1] = (char *) "Close";
submenu[0].str[2] = (char *) "Exit";
submenu[0].Function[0] = (void(*)()) RunOpen;
submenu[0].Function[1] = (void(*)()) RunClose;
submenu[0].Function[2] = (void(*)()) RunBye;

submenu[1].no = 4;
submenu[1].str[0] = (char *) "Cut";
submenu[1].str[1] = (char *) "Copy";
submenu[1].str[2] = (char *) "Paste";
submenu[1].str[3] = (char *) "Delete";
submenu[1].Function[0] = (void(*)()) NotThing;
submenu[1].Function[1] = (void(*)()) NotThing;
submenu[1].Function[2] = (void(*)()) NotThing;
submenu[1].Function[3] = (void(*)()) NotThing;

submenu[2].no = 2;
submenu[2].str[0] = (char *) "Tools";
submenu[2].str[1] = (char *) "Font";
submenu[2].Function[0] = (void(*)()) NotThing;
submenu[2].Function[1] = (void(*)()) NotThing;

submenu[3].no = 3;
submenu[3].str[0] = (char *) "Find";
submenu[3].str[1] = (char *) "Replace";
submenu[3].str[2] = (char *) "Next";
submenu[3].Function[0] = (void(*)()) NotThing;
submenu[3].Function[1] = (void(*)()) NotThing;
```

```
submenu[3].Function[2] = (void(*)()) NotThing;
submenu[3].Function[3] = (void(*)()) NotThing;

submenu[4].no = 1;
submenu[4].str[0] = (char *) "About";
submenu[4].Function[0] = (void(*)()) RunAbout;


/* 进行子菜单 1 显示及选择操作 */
mselect = 0;
while(1)
{   /* 选择主菜单项 */
    while(1)
    {   GUI_MMenuSelect(&mainmenu, mselect);          // 显示当前主菜单项
        key    = WaitKey();                           // 等待按键操作
        if(key==KEY_OK) break;                        // 若是 OK 键，则退出主菜单选择
        if(key==KEY_BACK)                             // BACK 键操作
        {   if(mselect>0)
            {   GUI_MMenuNSelect(&mainmenu, mselect); // 取消当前主菜单项选择
                mselect--;                            // 指向下一主菜单项
            }
        }
        if(key==KEY_NEXT)                             // NEXT 键操作
        {   if( mselect<(mainmenu.no-1) )
            {   GUI_MMenuNSelect(&mainmenu, mselect); // 取消当前主菜单项选择
                mselect++;                            // 指向下一主菜单项
            }
        }
    }
    GUI_MMenuNSelect(&mainmenu, mselect);

    /* 下拉子菜单，选择功能 */
    select = submenu[mselect].state;
    bak = select;
    GUI_SMenuDraw(&submenu[mselect]);                 // 显示子菜单
    while(1)
    {   key = WaitKey();
        if(key==KEY_OK) break;                        // 点击 OK 键选择
        if(key==KEY_NEXT)
        {   bak = select;
            select++;
            if(select>=submenu[mselect].no) select = 0;
            GUI_SMenuSelect(&submenu[mselect], bak, select);
        }
```

```
            if(key==KEY_BACK)
            {   bak = select;
                if(select==0) select = submenu[mselect].no - 1;
                   else select--;
                GUI_SMenuSelect(&submenu[mselect], bak, select);
            }
        }
        GUI_SMenuHide(&submenu[mselect]);
        (*submenu[mselect].Function[select])();                    // 调用相应的服务程序
        if( (mselect==0) && (select==2) )
        {   break;
        }
    }
}
```