JLX240-003-PC

带字库 IC 的编程说明书

目 录

1．概述

JLX240-003-PC 型液晶显示模块既可以当成普通的图像型液晶显示模块使用（即显示普通图像型的单色图片功能），又含有 JLX-GB2312-3205 字库 IC，可以从字库 IC 中读出内置的字库的点阵数据写入到 LCD 驱动 IC 中，以达到显示汉字的目的。

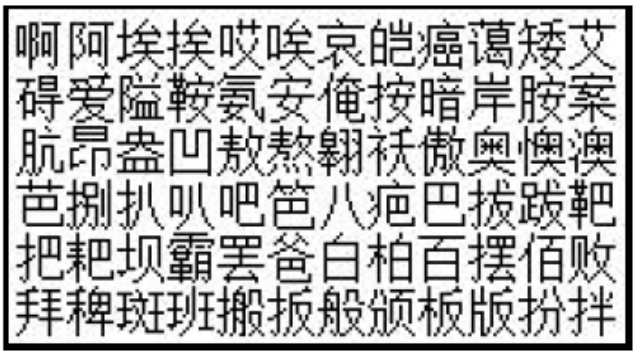
此字库 IC 存储内容如下表所述：

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

2．字型样张：

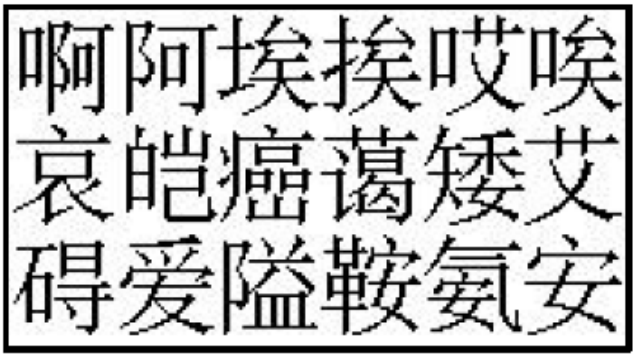
11X12 点 GB2312 汉字

15X16 点 GB2312 汉字



24X24 点 GB2312 汉字

32X32 点 GB2312 汉字



5x7 点 ASCII 字符

7x8 点 ASCII 字符

6x12 点 ASCII 字符

8x16 点 ASCII 字符

12 点阵不等宽 ASCII 方头

16 点阵不等宽 ASCII 方头

3. 外形尺寸及接口引脚功能

3.1 外形图：

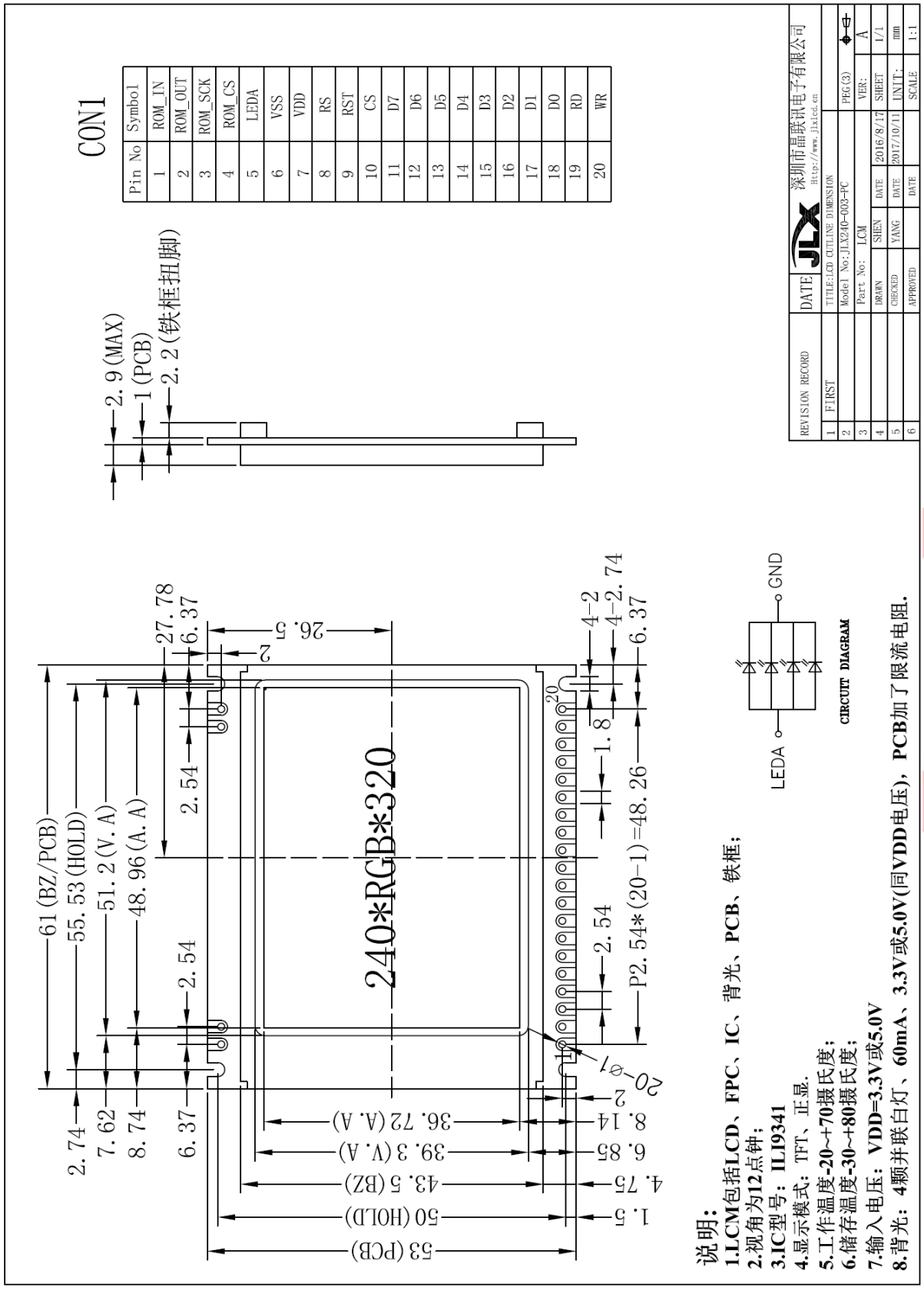


图 1. 外形尺寸

3.2 模块的接口引脚功能

表 1： 模块的 CON1 接口引脚功能

引 线 号	符 号	名 称	功 能
1	ROM_IN	字库 IC 接口	字库串行数据输入，不用字库时空
2	ROM_OUT	字库 IC 接口	字库串行数据输出，不用字库时空
3	ROM_SCK	字库 IC 接口	字库串行时钟，不用字库时空
4	ROM_CS	字库 IC 接口	字库片选输入，不用字库时空
5	LEDA	背光电源	背光电源正极，同 VDD 电压（5V 或 3.3V）
6	VSS	接地	0V
7	VDD	电路电源	5V 或 3.3V 可选
8	RS	寄存器选择信号	H:数据寄存器 0:指令寄存器（IC 资料上所写为”A0”）
9	RST	复位	低电平复位，复位完成后，回到高电平，液晶模块开始工作
10	CS	片选	低电平片选
11-18	D0-D7	I/O	数据总线 DB0～DB7
19	RD（E）	使能信号	并行时：使能信号
20	WR	读/写	并行时：H:读数据 0:写数据

4. 工作电路框图：

见图 2，模块由 LCD 驱动 IC ILI9341、字库 IC、背光、铁框组成。

5. 指令：

5.1 字库 IC（JLX-GB2312-32S4W）指令表

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

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

Read Data Bytes（一般读取）：

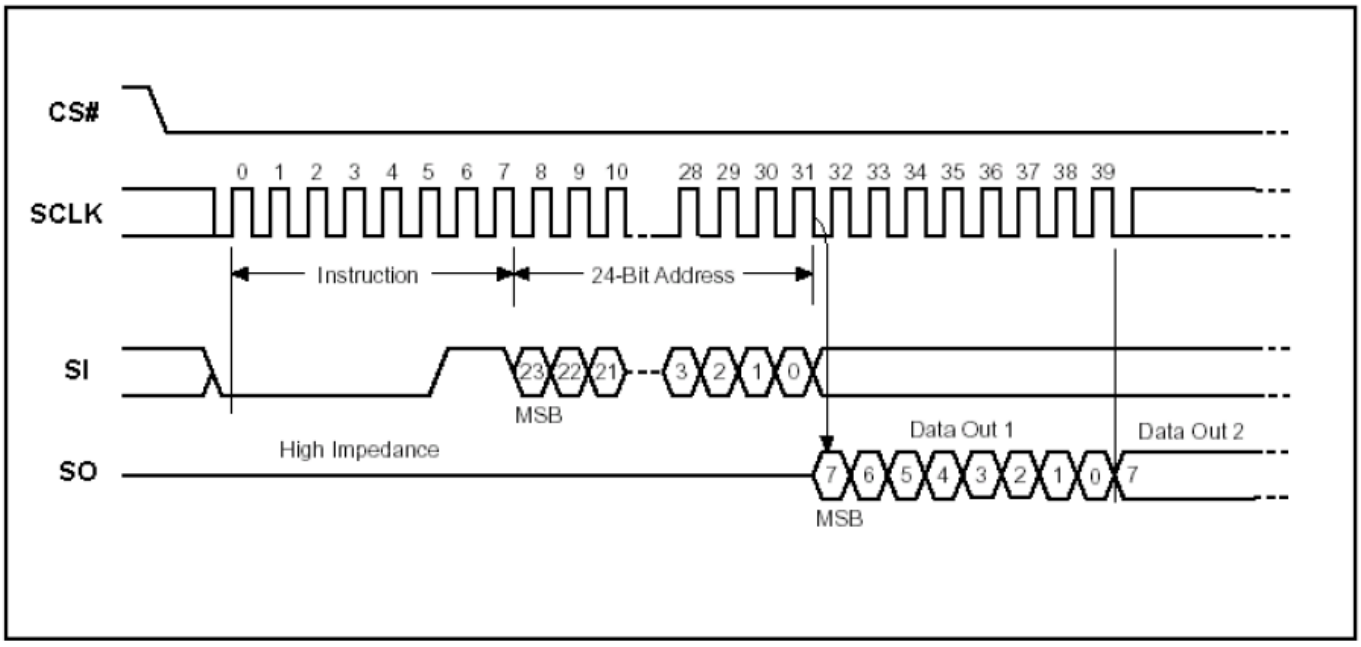
Read Data Bytes 需要用指令码来执行每一次操作。READ 指令的时序如下(图)：

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

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

■读取字节数据后，则把片选信号（CS#）变为高，结束本次操作。
如果片选信号（CS#）继续保持为低，则下一个地址的字节数据继续通过串行数据输出引脚（SO）移位输出。

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



Read Data Bytes at Higher speed（快速读取）：

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

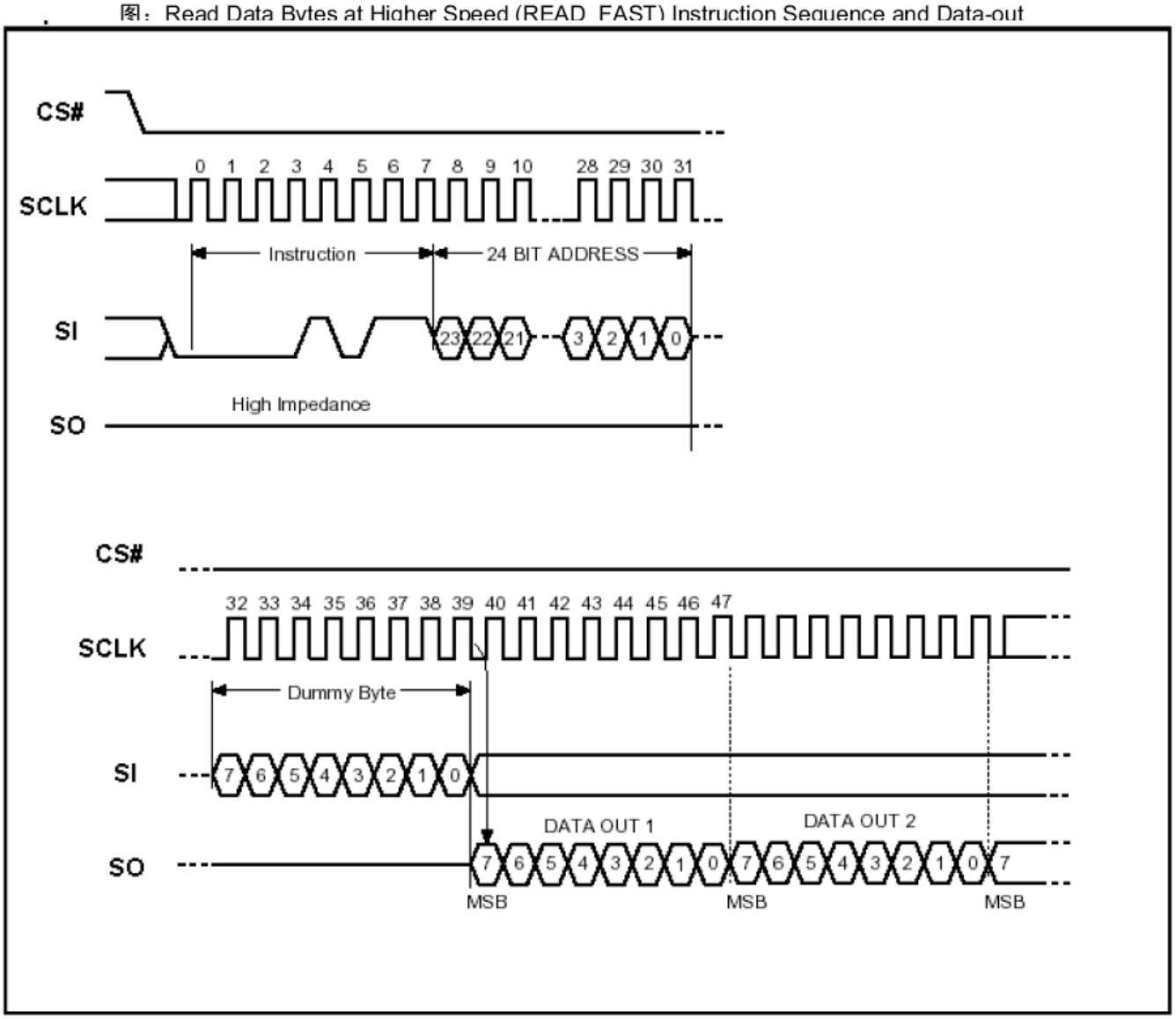
■首先把片选信号（CS#）变为低，紧跟着的是 1 个字节的命令字（0B h）和 3 个字节的地址以及一个字节 Dummy Byte 通过串行数据输入引脚（SI）移位输入，每一位在串行时钟（SCLK）上升沿被锁存。

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

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

如果不需要继续读取数据，则把片选信号（CS#）变为高，结束本次操作。

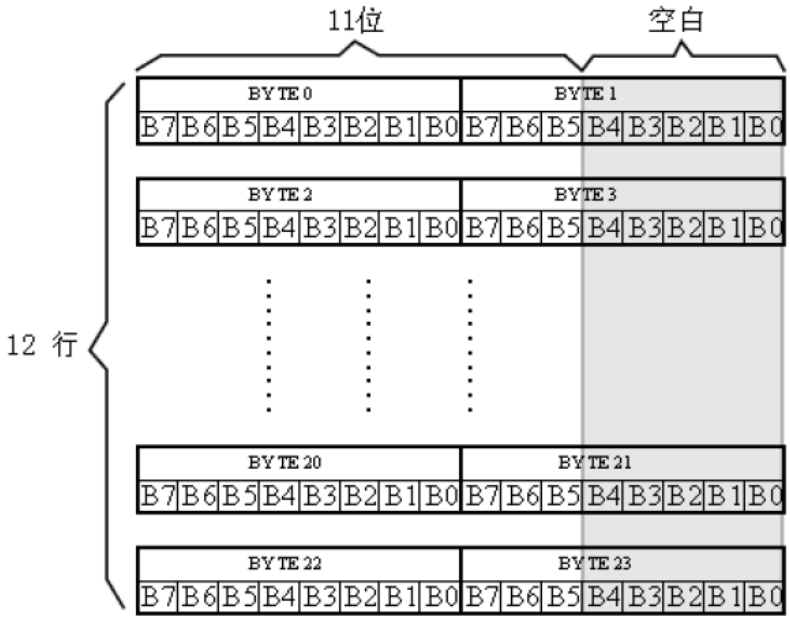
图：Read Data Bytes at Higher Speed (READ_FAST) Instruction Sequence and Data-out



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

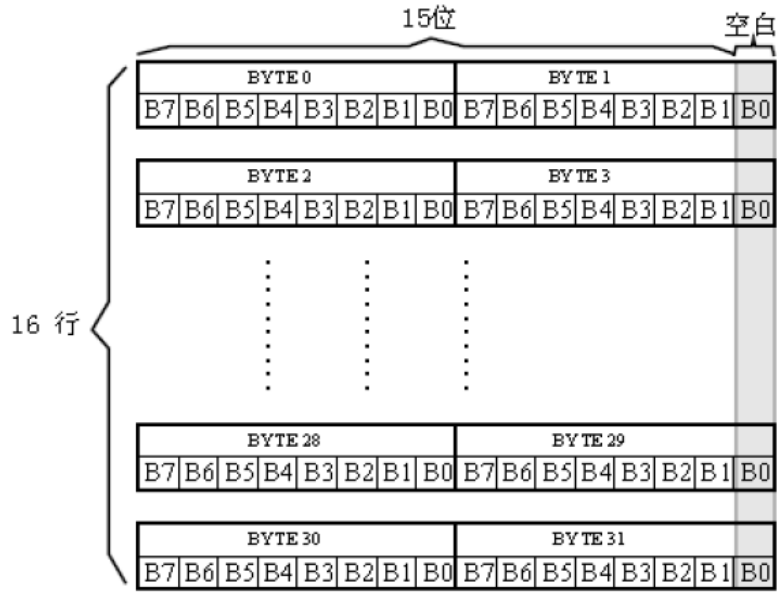
6.1.1 11X12 点汉字排列格式

11X12 点汉字的信息需要 24 个字节（BYTE 0 – BYTE 23）来表示。该 11X12 点汉字的点阵数据是横置横排的，其具体排列结构如下图：

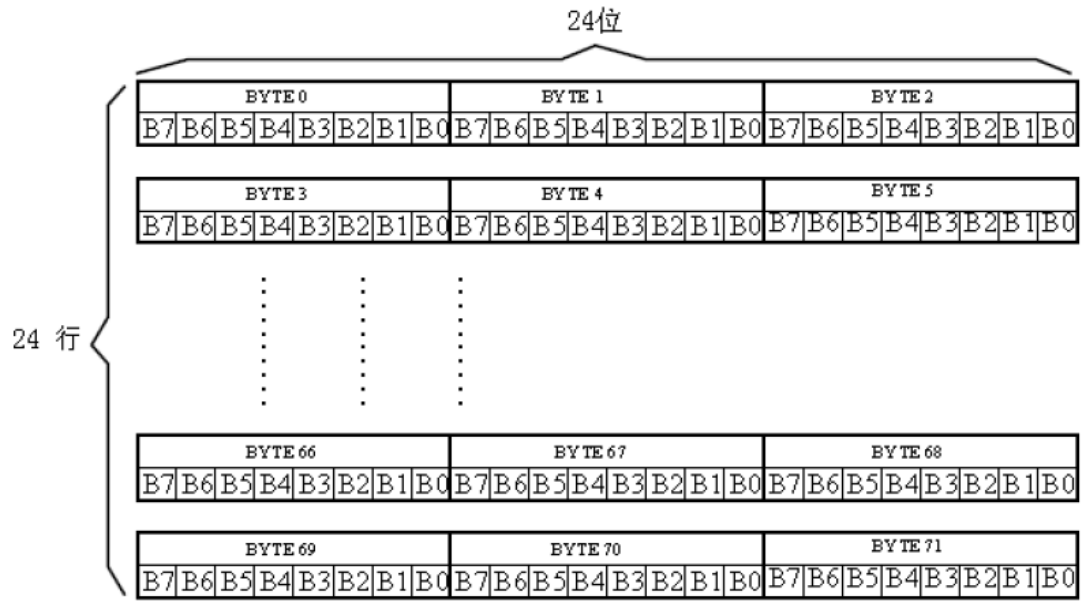


6.1.2 15X16 点汉字排列格式

15X16 点汉字的信息需要 32 个字节（BYTE 0 – BYTE 31）来表示。该 15X16 点汉字的点阵数据是横置横排的，其具体排列结构如下图：

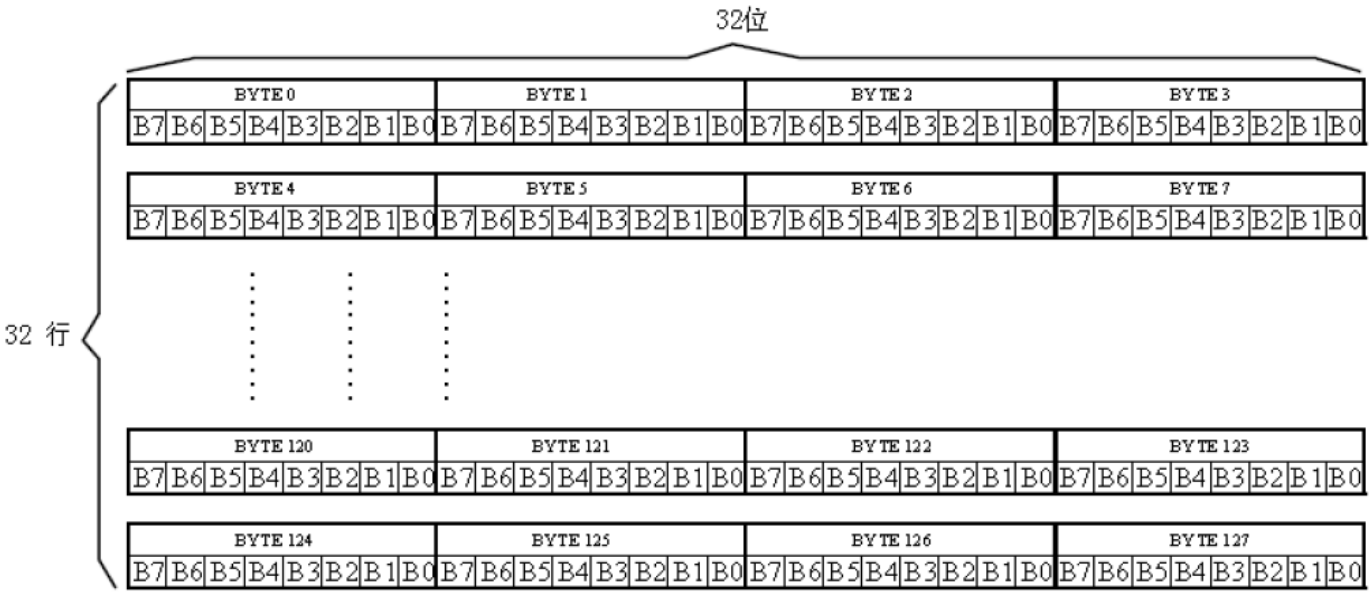


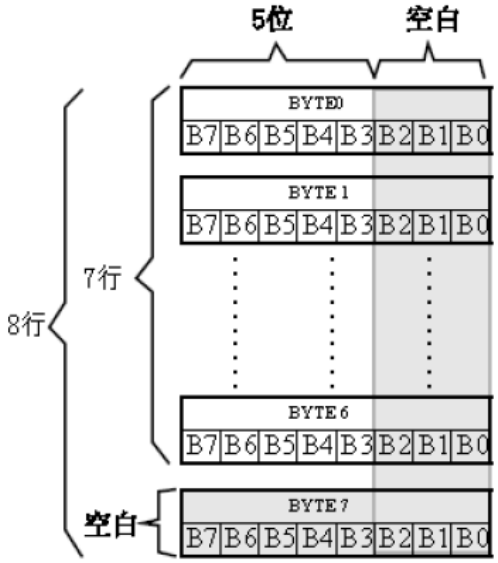
24X24 点汉字的信息需要 72 个字节（BYTE 0 – BYTE 71）来表示。该 24X24 点汉字的点阵数据是横置横排的，其具体排列结构如下图：



6.1.4 32X32 点汉字排列格式

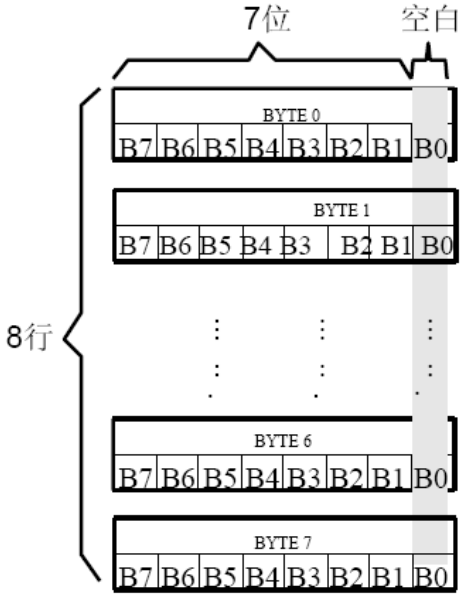
32X32 点汉字的信息需要 128 个字节（BYTE 0 – BYTE 127）来表示。该 32X32 点汉字的点阵数据是横置横排的，其具体排列结构如下图：





6.1.6 7X8 点 ASCII 字符排列格式

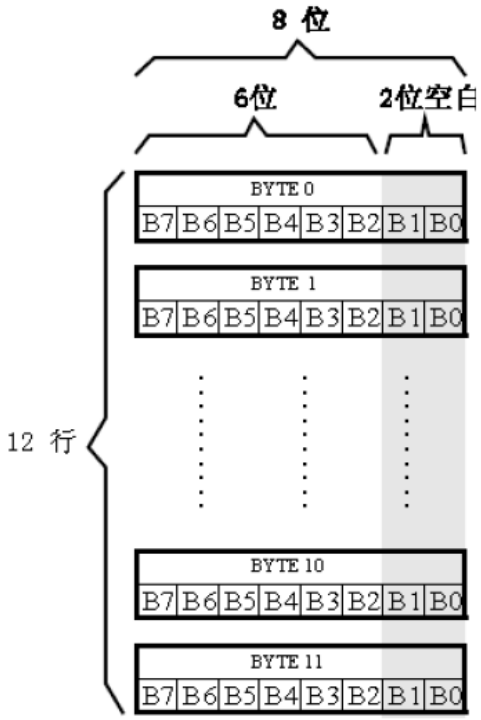
7X8 点 ASCII 的信息需要 8 个字节（BYTE 0 – BYTE7）来表示。该 ASCII 点阵数据是横置横排的其具体排列结构如下图：



6.1.7 6X12 点字符排列格式

适用于此种排列格式的字体有：
6X12 点 ASCII 字符
6X12 点国标扩展字符

6X12 点字符的信息需要 12 个字节（BYTE 0 – BYTE11）来表示。该点阵数据是横置横排的，其具体排列结构如下图：



6.1.8 8X16 点字符排列格式

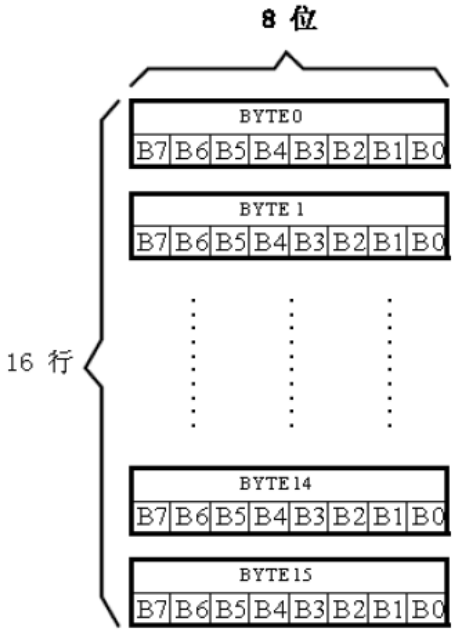
适用于此种排列格式的字符有：

8X16 点 ASCII 字符

8X16 点国标扩展字符

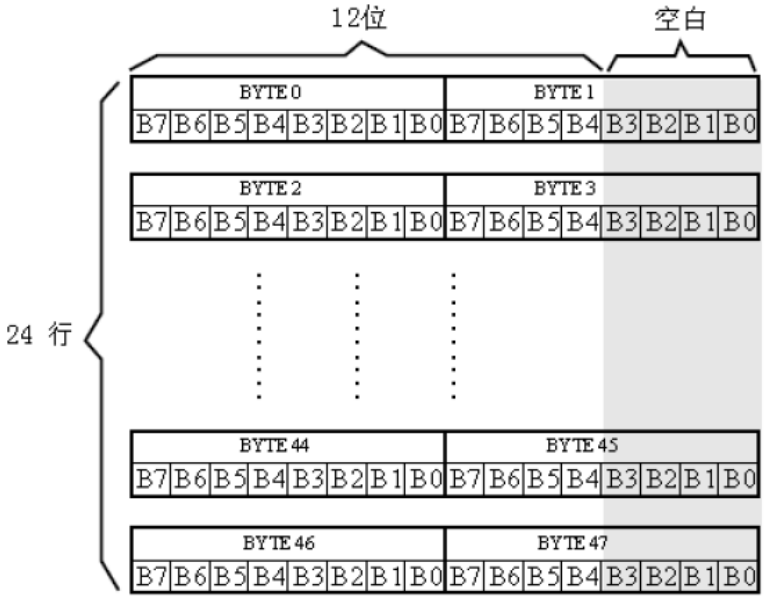
8X16 点特殊字符

8X16 点字符的信息需要 16 个字节（BYTE 0 – BYTE15）来表示。该点阵数据是横置横排的，其具体排列结构如下图：



适用于此种排列格式的字符有：
12X24 点 ASCII 字符
12X24 点国标扩展字符

12X24 点字符的信息需要 72 个字节（BYTE 0 – BYTE 71）来表示。但是由于该字符为标准的 24X24 点格式，而存储空间是按照 12X24 点 BYTE 取整进行存储的（即 72 BYTES）,注意在排版时作相应的调整。



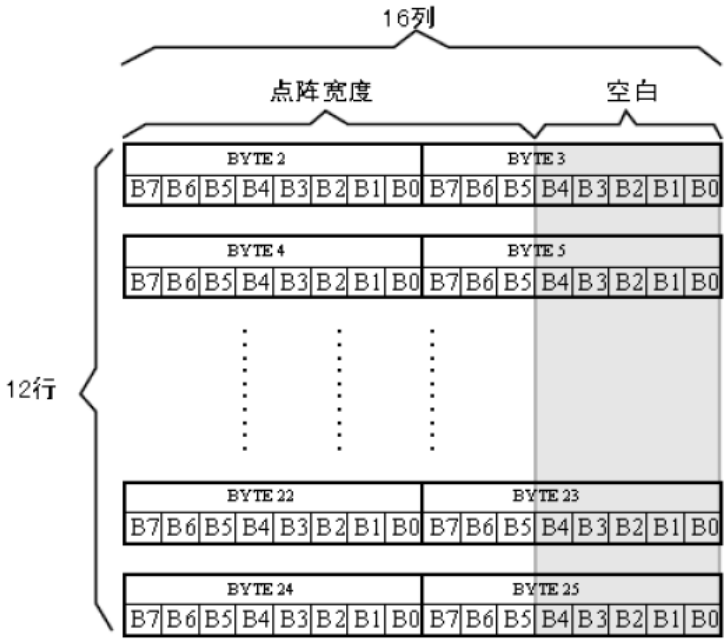
6.1.10 12 点阵不等宽字符排列格式

适用于此种排列格式的字体有：
12 点阵不等宽 ASCII 方头（Arial）字符
12 点阵不等宽 ASCII 白正（Times New Roman）字符

12 点阵不等宽字符的信息需要 26 个字节（BYTE 0 – BYTE25）来表示。

由于字符是不等宽的，因此在存储格式中 BYTE0~ BYTE1 存放点阵宽度数据，BYTE2-25 存放横置横排点阵数据。

不等宽字符的点阵存储宽度是以 BYTE 为单位取整的，根据不同字符宽度会出现相应的空白区。根 BYTE0~ BYTE1 所存放点阵的实际宽度数据，可以对还原下一个字的显示或排版留作参考。



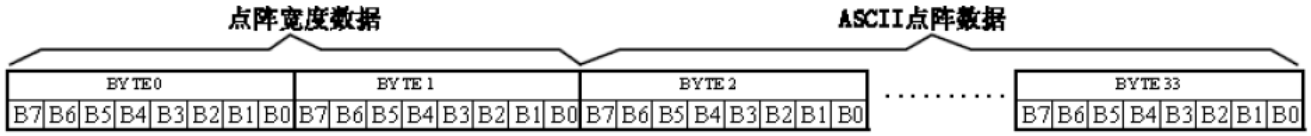
6.1.11 16 点阵不等宽字符排列格式

适用于此种排列格式的字体有：
16 点阵不等宽 ASCII 方头（Arial）字符
16 点阵不等宽 ASCII 白正（Times New Roman）字符

16 点阵不等宽字符的信息需要 34 个字节（BYTE 0 – BYTE33）来表示。

存储格式

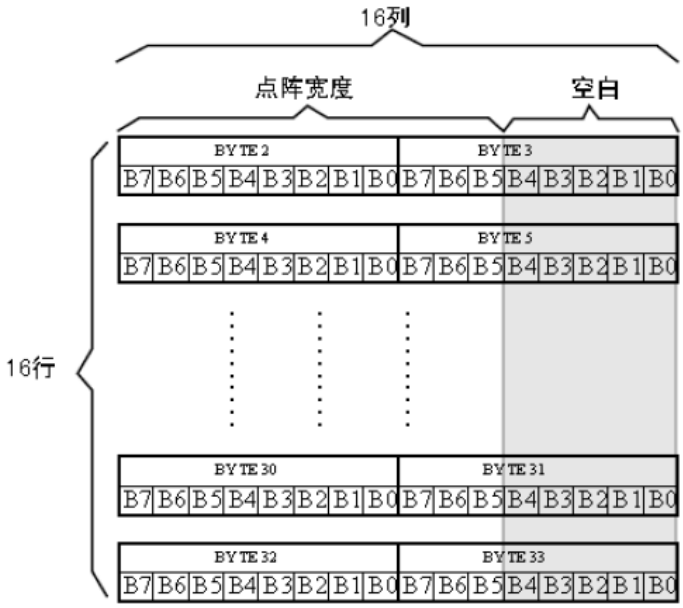
由于字符是不等宽的，因此在存储格式中 BYTE0~ BYTE1 存放点阵宽度数据，BYTE2-33 存放横置横排点阵数据。具体格式见下图：



存储结构

不等宽字符的点阵存储宽度是以 BYTE 为单位取整的，根据不同字符宽度会出现相应的空白区。根

BYTE0~ BYTE1 所存放点阵的实际宽度数据，可以对还原下一个字的显示或排版留作参考。



0-33BYTE 的点阵数据是: 00 0C 00 00 00 00 00 00 7F 80 7F C0 60 C0 60 C0 60 C0 7F 80 7F C0 60 E0 60 60 60 60 7F C0 7F 80 00 00

其中：

BYTE0~ BYTE1: 00 0C 为 ASCII 方头字符 B 的点阵宽度数据，即：12 位宽度。字符后面有 4 位空白区，可以在排版下一个字时考虑到这一点，将下一个字的起始位置前移。

BYTE2-33: 00 00 00 00 00 00 7F 80 7F C0 60 C0 60 C0 60 C0 7F 80 7F C0 60 E0 60 60 60 60 7F C0 7F 80 00 00 为 ASCII 方头字符 B 的点阵数据。

6.1.12 24 点阵不等宽字符排列格式

适用于此种排列格式的字体有：

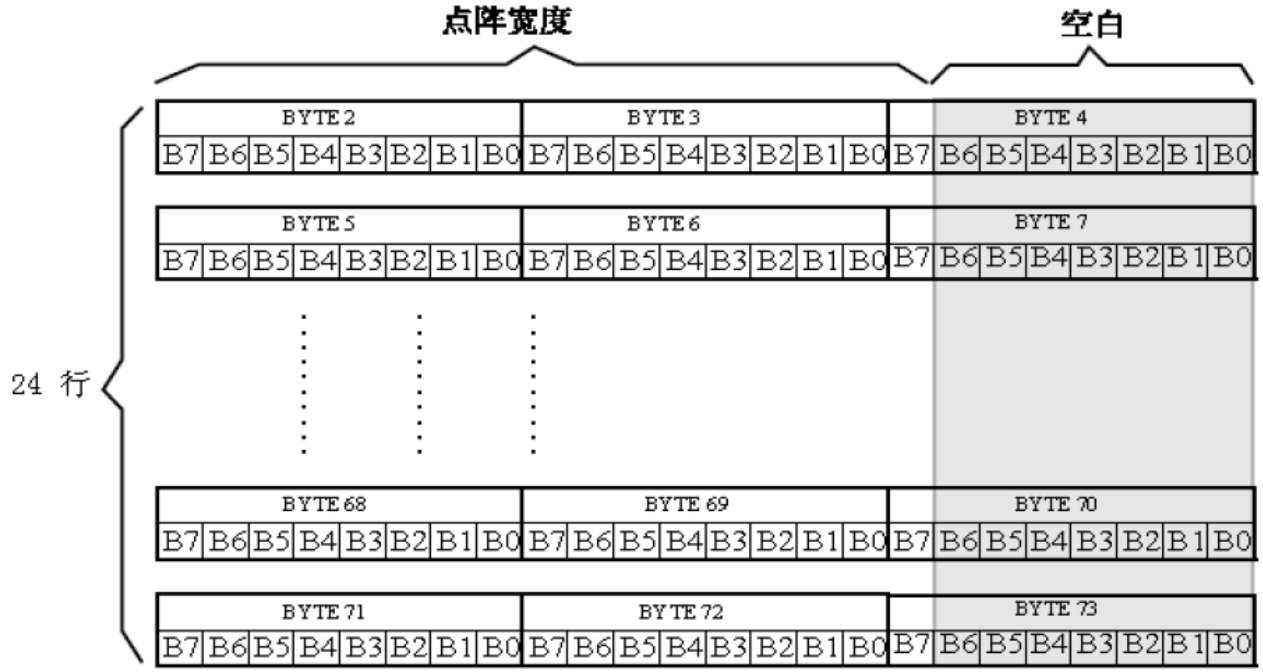
24 点阵不等宽 ASCII 方头（Arial）字符

24 点阵不等宽 ASCII 白正（Times New Roman）字符

24 点阵不等宽字符的信息需要 74 个字节（BYTE 0 – BYTE73）来表示。

由于字符是不等宽的，因此在存储格式中 BYTE0~ BYTE1 存放点阵宽度数据，BYTE2-73 存放点阵数据。

不等宽字符的点阵存储宽度是以 BYTE 为单位取整的，根据不同字符宽度会出现相应的空白区。根 BYTE0~ BYTE1 所存放点阵的实际宽度数据，可以对还原下一个字的显示或排版留作参考。



6.1.13 32 点阵不等宽字符排列格式

适用于此种排列格式的字体有：

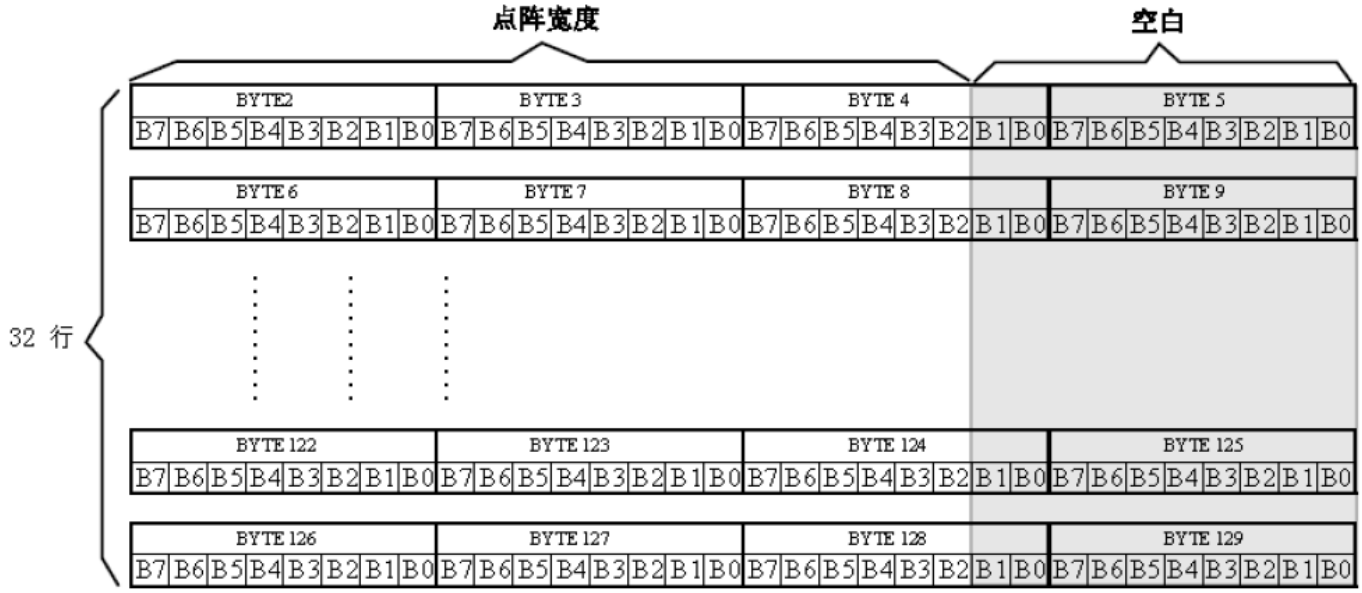
12 点阵不等宽 ASCII 方头（Arial）字符

12 点阵不等宽 ASCII 白正（Times New Roman）字符

32 点阵 ASCII 方头字符的信息需要 130 个字节（BYTE 0 – BYTE129）来表示。

由于字符是不等宽的，因此在存储格式中 BYTE0~ BYTE1 存放点阵宽度数据，BYTE2-129 存放点阵数据。

不等宽 ASCII 字符的存储结构是以宽度为 BYTE 取整的，根据不同字符宽度会出现相应的空白区。根据 BYTE0~ BYTE1 所存放点阵的宽度数据，可以对还原下一个字的显示或排版留作参考。



6.3 字符在芯片中的地址计算方法

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

6.3.1 汉字字符的地址计算

6.3.1.1 11X12 点 GB2312 标准点阵字库

参数说明：

GBCode表示汉字内码。

MSB 表示汉字内码GBCode的高8bits。

LSB 表示汉字内码GBCode 的低8bits。

Address 表示汉字或ASCII字符点阵在芯片中的字节地址。

BaseAdd：说明点阵数据在字库芯片中的起始地址。

计算方法：

```
BaseAdd=0x0;
if(MSB >=0xA1 && MSB <= 0Xa9 && LSB >=0xA1)
    Address =( (MSB - 0xA1) * 94 + (LSB - 0xA1))*24+ BaseAdd;
else if(MSB >=0xB0 && MSB <= 0xF7 && LSB >=0xA1)
    Address = ((MSB - 0xB0) * 94 + (LSB - 0xA1)+ 846)*24+ BaseAdd;
```

6.3.1.2 15X16 点 GB2312 标准点阵字库

参数说明：

GBCode表示汉字内码。

MSB 表示汉字内码GBCode的高8bits。

LSB 表示汉字内码GBCode 的低8bits。

Address 表示汉字或ASCII字符点阵在芯片中的字节地址。

BaseAdd：说明点阵数据在字库芯片中的起始地址。

计算方法：

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

6.3.1.3 24X24 点 GB2312 标准点阵字库

参数说明：

GBCode表示汉字内码。

MSB 表示汉字内码GBCode的高8bits。

LSB 表示汉字内码GBCode 的低8bits。

Address 表示汉字或ASCII字符点阵在芯片中的字节地址。

BaseAdd：说明点阵数据在字库芯片中的起始地址。

计算方法：

```
BaseAdd=0x68190;
if(MSB >=0xA1 && MSB <= 0Xa9 && LSB >=0xA1)
    Address =( (MSB - 0xA1) * 94 + (LSB - 0xA1))*72+ BaseAdd;
```

```
else if(MSB >=0xB0 && MSB <= 0xF7 && LSB >=0xA1)
    Address = ((MSB - 0xB0) * 94 + (LSB - 0xA1)+ 846)*72+ BaseAdd;
```

6.3.1.4 32X32 点 GB2312 标准点阵字库

参数说明：

GBCode表示汉字内码。

MSB 表示汉字内码GBCode的高8bits。

LSB 表示汉字内码GBCode 的低8bits。

Address 表示汉字或ASCII字符点阵在芯片中的字节地址。

BaseAdd：说明点阵数据在字库芯片中的起始地址。

计算方法：

```
BaseAdd=0XEDF00;
if(MSB >=0xA1 && MSB <= 0Xa9 && LSB >=0xA1)
    Address =( (MSB - 0xA1) * 94 + (LSB - 0xA1))*128+ BaseAdd;
else if(MSB >=0xB0 && MSB <= 0xF7 && LSB >=0xA1)
    Address = ((MSB - 0xB0) * 94 + (LSB - 0xA1)+ 846)*128+ BaseAdd;
```

6.3.1.5 6X12 点国标扩展字符

说明：

BaseAdd：说明本套字库在字库芯片中的起始字节地址。

FontCode：表示字符内码（16bits）

ByteAddress：表示字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DBE0C
if (FontCode>= 0xAAA1) and (FontCode<=0xAAFE ) then
    ByteAddress = (FontCode–0xAAA1 ) * 12+BaseAdd
Else if(FontCode>= 0xABA1) and (FontCode<=0xABC0 ) then
    ByteAddress = (FontCode–0xABA1 + 95) * 12+BaseAdd
```

1.6 8X16 点国标扩展字符

说明：

BaseAdd：说明本套字库在字库芯片中的起始字节地址。

FontCode：表示字符内码（16bits）

ByteAddress：表示字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DD790
if (FontCode>= 0xAAA1) and (FontCode<=0xAAFE ) then
    ByteAddress = (FontCode–0xAAA1 ) * 16+BaseAdd
Else if(FontCode>= 0xABA1) and (FontCode<=0xABC0 ) then
    ByteAddress = (FontCode–0xABA1 + 95) * 16+BaseAdd
```

6.3.1.7 8X16 点 GB2312 特殊字符

说明：

BaseAdd：说明本套字库在字库芯片中的起始字节地址。

FontCode：表示字符内码（16bits）

ByteAddress：表示字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1F2880
if (FontCode >= 0xACA1) and (FontCode <=0xACDF ) then
    ByteAddress = (FontCode–0xACA1 ) * 16+BaseAdd
```

6.3.1.8 12X24 点国标扩展字符

说明：

BaseAdd：说明本套字库在字库芯片中的起始字节地址。

FontCode：表示字符内码（16bits）

ByteAddress：表示字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DFF30
if (FontCode>= 0xAAA1) and (FontCode<=0xAAFE ) then
    ByteAddress = (FontCode–0xAAA1 ) * 48+BaseAdd
Else if(FontCode>= 0xABA1) and (FontCode<=0xABC0 ) then
    ByteAddress = (FontCode–0xABA1 + 95) * 48+BaseAdd
```

6.3.1.9 16X32 点国标扩展字符

说明：

BaseAdd：说明本套字库在字库芯片中的起始字节地址。

FontCode：表示字符内码（16bits）

ByteAddress：表示字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1E5A90
if (FontCode>= 0xAAA1) and (FontCode<=0xAAFE ) then
    ByteAddress = (FontCode–0xAAA1 ) * 64+BaseAdd
Else if(FontCode>= 0xABA1) and (FontCode<=0xABC0 ) then
    ByteAddress = (FontCode–0xABA1 + 95) * 64+BaseAdd
```

6.3.2 ASCII 字符的地址计算

6.3.2.1 5X7 点 ASCII 字符

参数说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DDF80
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 8+BaseAdd
```

6.3.2.2 7X8 点 ASCII 字符

参数说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x1DE280
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 8+BaseAdd

6.3.2.3 6X12 点 ASCII 字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x1DBE00
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 12+BaseAdd

6.3.2.4 8X16 点 ASCII 字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x1DD780
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 16+BaseAdd

6.3.2.5 12X24 点 ASCII 字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x1DFF00
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 48+BaseAdd

6.3.2.6 16X32 点 ASCII 字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x1E5A50
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 64+BaseAdd

6.3.2.7 12 点阵不等宽 ASCII 方头（Arial）字符

说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DC400
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 26 + BaseAdd
```

6.3.2.8 12 点阵不等宽 ASCII 白正（Times New Roman）字符

说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DCDC0
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) *26 + BaseAdd
```

6.3.2.9 16 点阵不等宽 ASCII 方头（Arial）字符

说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DE580
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 34 + BaseAdd
```

6.3.2.10 16 点阵不等宽 ASCII 白正（Times New Roman）字符

说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。

计算方法：

```
BaseAdd=0x1DF240
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 34 + BaseAdd
```

6.3.2.11 24 点阵不等宽 ASCII 方头（Arial）字符

说明：

ASCIICode：表示 ASCII 码（8bits）

BaseAdd：说明该套字库在芯片中的起始地址。

Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：

```
BaseAdd=0x1E22D0
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 74 + BaseAdd
```

6.3.2.12 24 点阵不等宽 ASCII 白正（Times New Roman）字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：

```
BaseAdd=0x1E3E90
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 74+ BaseAdd
```

6.3.2.13 32 点阵不等宽 ASCII 方头（Arial）字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：

```
BaseAdd=0x1E99D0
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 130 + BaseAdd
```

6.3.2.14 32 点阵不等宽 ASCII 白正（Times New Roman）字符

说明：
ASCIICode：表示 ASCII 码（8bits）
BaseAdd：说明该套字库在芯片中的起始地址。
Address：ASCII 字符点阵在芯片中的字节地址。
计算方法：

```
BaseAdd=0x1ECA90
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20 ) * 130 + BaseAdd
```

6.4 附录

6.4.1 GB2312 1 区（376 字符）

GB2312 标准点阵字符 1 区对应码位的 A1A1~A9EF 共计 846 个字符；

GB2312 1 区

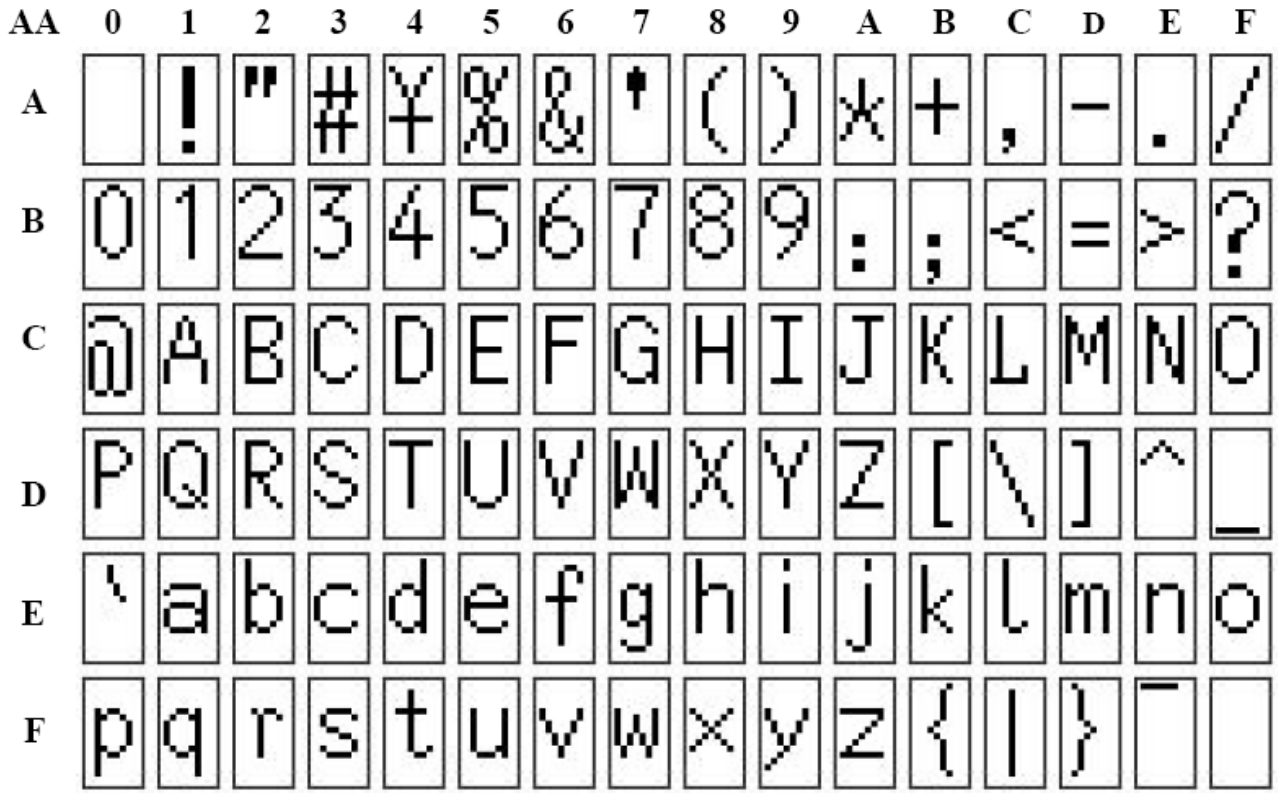
A1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A			、	。	·	ˉ	ˇ	¨	〃	々	—	～	‖	…	‘	’
B	“	”	〔	〕	〈	〉	《	》	「	」	『	』	〖	〗	【	】
C	±	×	÷	∶	∧	∨	∑	∏	∪	∩	∈	∷	√	⊥	∥	∠
D	⌒	⊙	∫	∮	≡	≌	≈	∽	∝	≠	≮	≯	≤	≥	∞	∵
E	∴	♂	♀	°	′	″	℃	$	¤	￠	￡	‰	§	№	☆	★
F	○	●	◎	◇	◆	□	■	△	▲	※	→	←	↑	↓	〓	

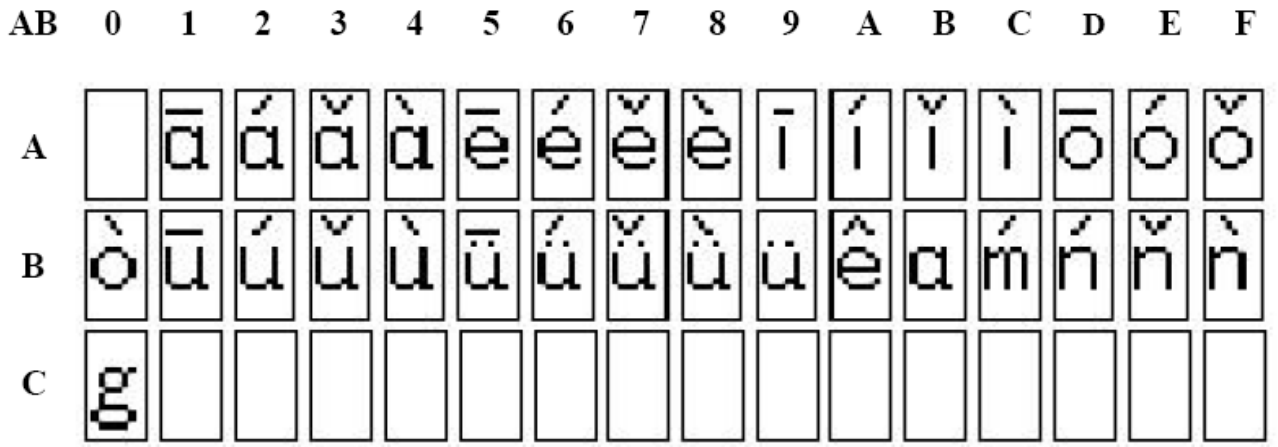
A2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A		ⅰ	ⅱ	ⅲ	ⅳ	ⅴ	ⅵ	ⅶ	ⅷ	ⅸ	ⅹ					
B		1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.
C	16.	17.	18.	19.	20.	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
D	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	①	②	③	④	⑤	⑥	⑦
E	⑧	⑨	⑩	€		㈠	㈡	㈢	㈣	㈤	㈥	㈦	㈧	㈨	㈩	
F		Ⅰ	Ⅱ	Ⅲ	Ⅳ	Ⅴ	Ⅵ	Ⅶ	Ⅷ	Ⅸ	Ⅹ	Ⅺ	Ⅻ			

A3	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A		！	＂	＃	￥	％	＆	＇	（	）	＊	＋	，	－	．	／
B	０	１	２	３	４	５	６	７	８	９	：	；	＜	＝	＞	？
C	＠	Ａ	Ｂ	Ｃ	Ｄ	Ｅ	Ｆ	Ｇ	Ｈ	Ｉ	Ｊ	Ｋ	Ｌ	Ｍ	Ｎ	Ｏ
D	Ｐ	Ｑ	Ｒ	Ｓ	Ｔ	Ｕ	Ｖ	Ｗ	Ｘ	Ｙ	Ｚ	［	＼	］	＾	＿
E	‘	ａ	ｂ	ｃ	ｄ	ｅ	ｆ	ｇ	ｈ	ｉ	ｊ	ｋ	ｌ	ｍ	ｎ	ｏ
F	ｐ	ｑ	ｒ	ｓ	ｔ	ｕ	ｖ	ｗ	ｘ	ｙ	ｚ	｛	｜	｝	￣	

6.4.2 8×16点国标扩展字符

内码组成为 AAA1~ABC0 共计 126 个字符

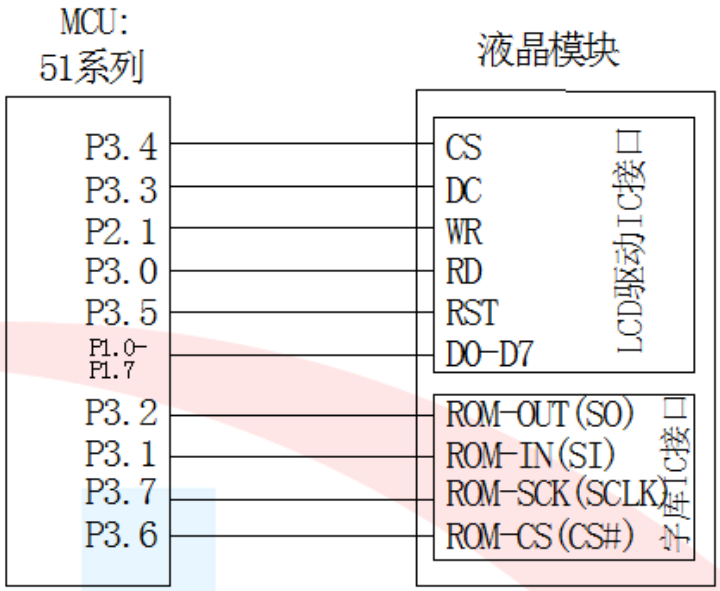




7. 硬件设计及例程：

7.1 当 LCD 驱动 IC 采用串行接口方式时的硬件设计及例程：

7.1.1 硬件接口：下图为串行方式的硬件接口：



7.1.2 例程：以下为并行方式显示汉字及 ASCII 字符的例程：

```
//driver IC:ili9341

#include <reg52.h>
#include <intrins.h>
sbit DC0 = P3^3;
sbit WR0 = P2^1;
sbit RD0 = P3^0;
sbit LCD_CS0 = P3^4;
sbit RST = P3^5;
sbit LEDA= P2^5;

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

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

```
#define red 0xf800     //定义红色
#define blue 0x001f    //定义蓝色
#define green 0x07e0 //定义绿色

#define background_color_blue blue          //背景颜色

#define font_color 0xffff                   //字体颜色
#define background_color_red red            //背景颜色

#define font_color_black 0x0000                 //字体颜色
#define background_color_green green        //背景颜色


unsigned char code pic1[];
unsigned                                                                    char
Graphic16[]={0x00,0XFF,0x00,0Xff,0x00,0Xff,0x00,0Xff,0x00,0Xff,0x00,0xff,0x00,0Xff,0x00,0Xff,0XFF,0x00,0XFf,0x00,0XFf,0x00,0XFf,0x00,0X
ff,0x00,0Xff,0x00,0Xff,0x00,0Xff,0x00};


/*延时*/
void delayms(long i)
{
    int j,k;
    for(j=0;j<i;j++)
          for(k=0;k<110;k++);
}

/*等待一个按键，我的主板是用 P2.0 与 GND 之间接一个按键*/
void waitkey()
{
 repeat:
    if (P2&0x01) goto repeat;
    else delayms(6);
    if (P2&0x01) goto repeat;
    else
    delayms(40);;
}


void data_out(uchar data1)
{
    //8080 8bit interface
    LCD_CS0 = 0;
    DC0 = 1;
    RD0 = 1;
    P1=data1;
```

```
    WR0 = 0;
    WR0 = 1;
    LCD_CS0 = 1;

}

void comm_out(uchar com)
{
    //8080 8bit interface
    DC0 = 0;
    LCD_CS0 = 0;
    RD0 = 1;
    P1 = com;
    WR0 = 0;
    WR0 = 1;
    LCD_CS0 = 1;
}

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

/****从晶联讯字库 IC 中取汉字或字符数据（1 个字节）***/
static uchar get_data_from_ROM( )
{
    uchar i;
    uchar ret_data=0;
    Rom_SCK=1;
    for(i=0;i<8;i++)
    {
        Rom_OUT=1;
        Rom_SCK=0;
        //delay(1);
        ret_data=ret_data<<1;
        if( Rom_OUT )
```

```
            ret_data=ret_data+1;
        else
            ret_data=ret_data+0;
        Rom_SCK=1;
        //delay(1);
    }
    return(ret_data);
}


//LCD 初始化
void LCD_initial()
{
    RST=1;
    delayms(10);
    RST=0;
    delayms(100);
    RST=1;
    delayms(1200);

//************** Start Initial Sequence **********//
    comm_out(0xCF);
    data_out (0x00);
    data_out (0xD9);
    data_out (0X30);

    comm_out(0xED);
    data_out (0x64);
    data_out (0x03);
    data_out (0X12);
    data_out (0X81);

    comm_out(0xE8);
    data_out (0x85);
    data_out (0x00);
    data_out (0x78);

    comm_out(0xCB);
    data_out (0x39);
    data_out (0x2C);
    data_out (0x00);
    data_out (0x34);
    data_out (0x02);

    comm_out(0xF7);
    data_out (0x20);
```

```
comm_out(0xEA);
data_out (0x00);
data_out (0x00);

comm_out(0xC0);       //Power control
data_out (0x1B);      //VRH[5:0]

comm_out(0xC1);       //Power control
data_out (0x12);      //SAP[2:0];BT[3:0]

comm_out(0xC5);       //VCM control
data_out (0x32);
data_out (0x3C);

comm_out(0xC7);       //VCM control2
data_out (0X9D);

comm_out(0x36);       // Memory Access Control
data_out (0x08);      //0x28：横屏/0x08：竖屏

comm_out(0x3A);
data_out (0x55);

comm_out(0xB1);
data_out (0x00);
data_out (0x1B);

comm_out(0xB6);       // Display Function Control
data_out (0x0A);
data_out (0xA2);

comm_out(0xF6);
data_out (0x01);
data_out (0x30);

comm_out(0xF2);       // 3Gamma Function Disable
data_out (0x00);

comm_out(0x26);       //Gamma curve selected
data_out (0x01);

comm_out(0xE0);       //Set Gamma
data_out (0x0F);
data_out (0x24);
data_out (0x1F);
data_out (0x0B);
data_out (0x0F);
```

```
    data_out (0x05);
    data_out (0x4A);
    data_out (0X96);
    data_out (0x39);
    data_out (0x07);
    data_out (0x11);
    data_out (0x03);
    data_out (0x11);
    data_out (0x0D);
    data_out (0x04);

    comm_out(0XE1);       //Set Gamma
    data_out (0x00);
    data_out (0x1B);
    data_out (0x20);
    data_out (0x04);
    data_out (0x10);
    data_out (0x02);
    data_out (0x35);
    data_out (0x23);
    data_out (0x46);
    data_out (0x04);
    data_out (0x0E);
    data_out (0x0C);
    data_out (0x2E);
    data_out (0x32);
    data_out (0x05);

    comm_out(0x11);       //Exit Sleep
    delayms(120);
    comm_out(0x29);       //Display on
}


//显示黑底加一个白色外框

void display_black(void)
{
    int i,j,k;

    comm_out(0x2c);
    for(i=0;i<240;i++)
    {
        data_out(0xff);
        data_out(0xff);
    }
    for(i=0;i<318;i++)
```

```
    {
        for(k=0;k<1;k++)
        {
            data_out(0xff);
            data_out(0xff);
        }

        for(j=0;j<238;j++)
        {
            data_out(0x00);
            data_out(0x00);
        }
        for(k=0;k<1;k++)
        {
            data_out(0xff);
            data_out(0xff);
        }
    }
    for(i=0;i<240;i++)
    {
        data_out(0xff);
        data_out(0xff);
    }
}


//全屏显示一种颜色

void display_color(uchar color_H,uchar color_L)
{
    int i,j;
    comm_out(0x2c);
    for(i=0;i<320;i++)
    {
        for(j=0;j<240;j++)
        {
            data_out(color_H);
            data_out(color_L);
        }
    }
}


//显示图片

    void display_image1(void)
```

```
{
    unsigned int i,j,t;
    comm_out(0x00);
    comm_out(0x2c);
    for(t=0;t<4;t++)
    {
        unsigned int   k=0;
        for(i=0;i<80;i++)
        {
            for(j=0;j<240;j++)
            {
                data_out(pic1[k++]);
                data_out(pic1[k++]);
            }
        }
    }
}


//===设置显示窗口的 X\Y 开始坐标及结束座标==================================================
void lcd_address(int xs,int ys,x_total,y_total)
{
    int xe,ye;
    xe=xs+x_total-1;
    ye=ys+y_total-1;
    comm_out(0x2a);          //设置列（x）起始和结束地址：分为开始地址的高 8 位、低 8 位，结束地址的高 8 位、低 8 位
    data_out(xs/256);
    data_out(xs%256);
    data_out(xe/256);
    data_out(xe%256);
    comm_out(0x2b);          //设置行（y）起始和结束地址：分为开始地址的高 8 位、低 8 位，结束地址的高 8 位、低 8 位
    data_out(ys/256);
    data_out(ys%256);
    data_out(ye/256);
    data_out(ye%256);
}

//将单色的 8 位的数据（代表 8 个像素点）转换成彩色的数据传输给液晶屏
void mono_data_out(char mono_data)
{
    char i,color_data_h,color_data_l;
    for(i=0;i<8;i++)
    {
        if(mono_data&0x80)
        {
            color_data_h=font_color/256;
```

```
            color_data_l=font_color%256;
        }
        else
        {
            color_data_h=background_color_red/256;
            color_data_l=background_color_red%256;
        }
        data_out(color_data_h);
        data_out(color_data_l);
        mono_data=mono_data<<1;
    }
}

//显示 8x16 点阵的单色的图像
void disp_8x16(uint x,uint y,char *dp)
{
    int i,j;
    lcd_address(x,y,8,16);
    comm_out(0x00);
    comm_out(0x22);
    for(i=0;i<16;i++)
    {
        for(j=0;j<1;j++)
        {
            mono_data_out(*dp);
            dp++;
        }
    }
}

//显示 16x16 点阵的单色的图像
void disp_16x16(uint x,uint y,char *dp)
{
    int i,j;
    lcd_address(x,y,16,16);
    comm_out(0x00);
    comm_out(0x22);
    for(i=0;i<16;i++)
    {
        for(j=0;j<2;j++)
        {
            mono_data_out(*dp);
            dp++;
        }
    }
}
```

```
void disp_16x32(uint x,uint y,char *dp)
{
    int i,j;
    lcd_address(x,y,16,32);
    comm_out(0x00);
    comm_out(0x22);
    for(i=0;i<32;i++)
    {
        for(j=0;j<2;j++)
        {
            mono_data_out(*dp);
            dp++;
        }
    }
}

void disp_32x16_zk(uint x,uint y,uchar *dp)
{
    int i,j;
    lcd_address(x,y,32,16);
    comm_out(0x00);
    comm_out(0x22);
    for(i=0;i<16;i++)
    {
        for(j=0;j<4;j++)
        {
            mono_data_out(*dp);
            dp++;
        }
    }
}
```

/*从相关地址（addrHigh：地址高字节,addrMid：地址中字节,addrLow：地址低字节）中连续读出 DataLen 个字节的数据到 pBuff 的地址*/

```
/*连续读取*/
void get_n_bytes_data_from_ROM(uchar addrHigh,uchar addrMid,uchar addrLow,uchar *pBuff,uchar DataLen)
{
    uchar i;
    Rom_CS = 0;
    LCD_CS0=1;
    Rom_SCK=0;
    send_command_to_ROM(0x03);
    send_command_to_ROM(addrHigh);
```

```
    send_command_to_ROM(addrMid);
    send_command_to_ROM(addrLow);
    for(i = 0; i < DataLen; i++ )
        *(pBuff+i) =get_data_from_ROM();
    Rom_CS=1;
}

/***********/
/*
ulong   fontaddr;
void disp_GB2312_32x32_string(uint x,uint y,uchar *text)
{
    uchar i= 0,j;
    uchar addrHigh,addrMid,addrLow ;
    uchar fontbuf[64];
    while((text[i]>0x00))
    {
        if( ((text[i]>=0xb0) && (text[i]<=0xf7) ) && (text[i+1]>=0xa1) )
        {
            fontaddr = (text[i]- 0xb0)*94;
            fontaddr += (text[i+1]-0xa1)+846;
            fontaddr = (ulong)(fontaddr*128);
            fontaddr = (ulong)(fontaddr+0Xedf00);

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

            Rom_CS = 0;
            LCD_CS0=1;

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

            for(j = 0; j < 64; j++ )
            {
                fontbuf[j] =get_data_from_ROM();
            }
            disp_32x16_zk(x,y,fontbuf);
            for(j = 0; j < 64; j++ )
            {
                fontbuf[j] =get_data_from_ROM();
            }
            Rom_CS = 1;
            disp_32x16_zk(x,y+16,fontbuf);
```

```
        i+=2;
        x+=32;
}

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

        fontaddr = (text[i]- 0xa1)*94;
        fontaddr += (text[i+1]-0xa1);
        fontaddr = (ulong)(fontaddr*128);
        fontaddr = (ulong)(fontaddr+0Xedf00);

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

        Rom_CS = 0;
        LCD_CS0=1;
        Rom_SCK=0;

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

        for(j = 0; j < 64; j++ )
        {
              fontbuf[j] =get_data_from_ROM();
        }
        LCD_CS0=1;Rom_CS = 0;
        disp_32x16_zk(x,y,fontbuf);
        LCD_CS0=1;Rom_CS = 0;
        for(j = 0; j < 64; j++ )
        {
              fontbuf[j] =get_data_from_ROM();
        }
        Rom_CS = 1;

        disp_32x16_zk(x,y+16,fontbuf);
        i+=2;
        x+=32;
}

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

```
            fontaddr = (ulong)(fontaddr+0x1e5a50);

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

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

            disp_16x32(x,y,fontbuf);
            i+=1;
            x+=16;
        }

        else
            i++;
    }
}
*/


//=====从字库读数据，显示 16*16 点阵的汉字或 8*16 点阵的数字========
void disp_GB2312_16x16_string(uint x,uint y,uchar *text,font_color,back_color)
{
    uchar i= 0;
    uchar addrHigh,addrMid,addrLow ;
    uchar fontbuf[32];
    ulong    fontaddr;
    while((text[i]>0x00))
    {
        if(((text[i]>=0xb0) &&(text[i]<=0xf7))&&(text[i+1]>=0xa1))
        {
            fontaddr = (text[i]- 0xb0)*94;
            fontaddr += (text[i+1]-0xa1)+846;
            fontaddr = (ulong)(fontaddr*32);
            fontaddr = (ulong)(fontaddr+0x2c9d0);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;
            get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
            disp_16x16(x,y,fontbuf,font_color,back_color);
            i+=2;
            x+=16;
        }
        else if(((text[i]>=0xa1) &&(text[i]<=0xab))&&(text[i+1]>=0xa1))
        {
            fontaddr = (text[i]- 0xa1)*94;
            fontaddr += (text[i+1]-0xa1);
```

```
            fontaddr = (ulong)(fontaddr*32);
            fontaddr = (ulong)(fontaddr+0x2c9d0);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;
            get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
            disp_16x16(x,y,fontbuf,font_color,back_color);
            i+=2;
            x+=16;
        }
        else if((text[i]>=0x20) &&(text[i]<=0x7e))
        {
            uchar fontbuf[16];
            fontaddr = (text[i]- 0x20);
            fontaddr = (ulong)(fontaddr*16);
            fontaddr = (ulong)(fontaddr+0x1dd780);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;

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

            disp_8x16(x,y,fontbuf,font_color,back_color);
            i+=1;
            x+=8;
        }
        else
            i++;
    }
}
```

```
/*=====从字库读数据，显示 16*16 点阵的汉字或 8*16 点阵的数字========*/
/*
void disp_GB2312_16x16_string(uint x,uint y,uchar *text)
{
    uchar i= 0;
    uchar addrHigh,addrMid,addrLow ;
    uchar fontbuf[32];
    ulong    fontaddr;
    while((text[i]>0x00))
    {
        if(((text[i]>=0xb0) &&(text[i]<=0xf7))&&(text[i+1]>=0xa1))
        {
            fontaddr = (text[i]- 0xb0)*94;
```

```
            fontaddr += (text[i+1]-0xa1)+846;
            fontaddr = (ulong)(fontaddr*32);
            fontaddr = (ulong)(fontaddr+0x2c9d0);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;
            get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
            disp_16x16(x,y,fontbuf);
            i+=2;
            x+=16;
        }
        else if(((text[i]>=0xa1) &&(text[i]<=0xab))&&(text[i+1]>=0xa1))
        {
            fontaddr = (text[i]- 0xa1)*94;
            fontaddr += (text[i+1]-0xa1);
            fontaddr = (ulong)(fontaddr*32);
            fontaddr = (ulong)(fontaddr+0x2c9d0);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;
            get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,32 );
            disp_16x16(x,y,fontbuf);
            i+=2;
            x+=16;
        }
        else if((text[i]>=0x20) &&(text[i]<=0x7e))
        {
            uchar fontbuf[16];
            fontaddr = (text[i]- 0x20);
            fontaddr = (ulong)(fontaddr*16);
            fontaddr = (ulong)(fontaddr+0x1dd780);
            addrHigh = (fontaddr&0xff0000)>>16;
            addrMid = (fontaddr&0xff00)>>8;
            addrLow = fontaddr&0xff;
            get_n_bytes_data_from_ROM(addrHigh,addrMid,addrLow,fontbuf,16 );
            disp_8x16(x,y,fontbuf);
            i+=1;
            x+=8;
        }
        else
            i++;
    }

}
*/
```

```
//主程序
void main(void)
{
    LCD_initial();
    LEDA=0;
    while(1)
    {
        display_image1();
        waitkey();
        display_color(0xf8,0x00);
        waitkey();
        display_color(0x07,0xe0);
        waitkey();
        display_color(0x00,0x1f);
        waitkey();
        display_black();
//      disp_GB2312_32x32_string(0,32*0,"  晶联讯 TFT 彩屏  ");
//      disp_GB2312_32x32_string(0,32*1,"◆240x320 点阵◆");
        disp_GB2312_16x16_string(0,16*4,"自带 GB2312 国标汉字库，可显示：");
        disp_GB2312_16x16_string(0,16*5,"32x32，24x24，16x16，12x12 点阵");
        disp_GB2312_16x16_string(0,16*6,"可以显示半角全角,其他字符,例如");
        disp_GB2312_16x16_string(0,16*7,"⑴⑵⑶①②③○●◎◇◆ А Б В Г ");
        disp_GB2312_16x16_string(0,16*8,"ぁあぃいぅうぇえā á ǎ à ē é ě");
        disp_GB2312_16x16_string(0,16*9,"、。• ˉ ˇ ¨ 〃 々—～ ‖ … ‘’ 1. 2. 3.");
        disp_GB2312_16x16_string(0,16*10," !#$%&'()*+,-./0123456789:;<=>");
        disp_GB2312_16x16_string(0,16*11,"?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]");
        disp_GB2312_16x16_string(0,16*12,"^_`abcdefghijklmnopqrstuvwxyz{");
        disp_GB2312_16x16_string(0,16*13,"********************************");
        disp_GB2312_16x16_string(0,16*14,"深圳市晶联讯电子有限公司是集研");
        disp_GB2312_16x16_string(0,16*15,"发、生产、销售于一体的从事液晶");
        disp_GB2312_16x16_string(0,16*16,"显示屏及液晶模块的高科技公司。");
        disp_GB2312_16x16_string(0,16*17,"多年来，本公司凭借稳定、可靠的");
        disp_GB2312_16x16_string(0,16*18,"产品质量，优异的性价比，完善的");
        disp_GB2312_16x16_string(0,304,"售后服务，让广大用户都倍感放心");
        waitkey();

    }
}
```