GT32L32M0180
标准汉字字库芯片

规格书
DATASHEET

- 字符集：GB18030
- 兼容 Unicode
- 条形码（2 套）：EAN13 条码、CODE128 条码
- 字号：12x12、16x16、24x24、32x32 点阵
- 排置方式：横置横排
- 总线接口：SPI 串行总线
- 封装类型：SOP8-B

VER 1.0 I_A

2012-08

版本修订记录

版本号	修改内容	日期	备注
VER1.0I_A	字库芯片说明书的制定	2012-08	完整版

目　　录

1. 概述

GT32L32M0180是一款拥有12x12,16x16,24x24、32x32点阵字库芯片，支持GB18030国标汉字(含有国家信标委合法授权)、ASCII字符及条形码图库。排列格式为横置横排。用户通过字符内码，利用用户手册提供的方法计算出该字符点阵在芯片中的地址，可从该地址连续读出字符点阵信息。

GT32L32M0180除含有上述字库以外，还提供256个扇区，每个扇区4K字节或16页，每页256字节，可自由读写空间地址范围为：0x000000~0x0FFFFF。

1.1 芯片特点

- 数据总线：SPI 串行总线接口
- 点阵排列方式：横置横排
- 时钟频率：60MHz @3.3V
- 工作电压：2.7V~3.6V
- 电流：
  - 工作电流：12mA
  - 待机电流：10uA
- 工作温度：-40℃~85℃
- 封装：SOP8-B

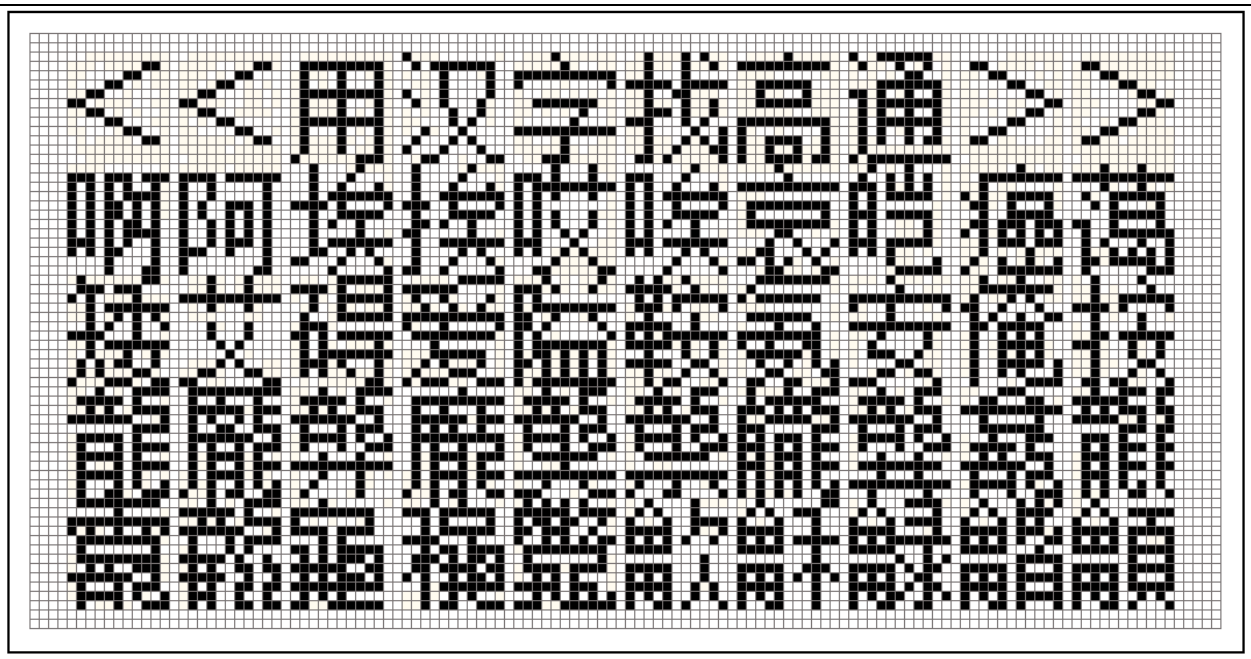
1.2 芯片内容

分类	字库	字号	字符数	字体	排列方式	备注
ASCII 字符集	ASCII	5x7	96	标准	W-横置横排	
	ASCII	7x8	96	标准	W-横置横排	
	ASCII	7x8	96	粗体	W-横置横排	
	ASCII	6x12	96	标准	W-横置横排	
	ASCII	8x16	128	标准	W-横置横排	
	ASCII	8x16	96	粗体	W-横置横排	
	ASCII	12x24	96	标准	W-横置横排	
	ASCII	16x32	96	标准	W-横置横排	
	ASCII	16x32	96	粗体	W-横置横排	
	ASCII	12 点阵不等宽	96	Arial（方头）	W-横置横排	
	ASCII	16 点阵不等宽	96	Arial（方头）	W-横置横排	
	ASCII	24 点阵不等宽	96	Arial（方头）	W-横置横排	
	ASCII	32 点阵不等宽	96	Arial（方头）	W-横置横排	
	ASCII	12 点阵不等宽	96	Time New Roman（白正）	W-横置横排	
	ASCII	16 点阵不等宽	96	Time New Roman（白正）	W-横置横排	
	ASCII	24 点阵不等宽	96	Time New Roman（白正）	W-横置横排	
	ASCII	32 点阵不等宽	96	Time New Roman（白正）	W-横置横排	
数字符号字符集	数字符号字符	14x28	15	黑体半角	W-横置横排	
	数字符号字符	20x40	12	黑体半角	W-横置横排	
	数字符号字符	28 点阵不等宽	15	Arial	W-横置横排	
	数字符号字符	40 点阵不等宽	13	Arial	W-横置横排	
GB18030 字符集	GB18030 汉字	12x12	20902	宋体	W-横置横排	
		16x16	27484	宋体	W-横置横排	
		24x24	27484	宋体	W-横置横排	
		32x32	27484	宋体	W-横置横排	
	GB18030 字符	12x12	1038	宋体	W-横置横排	
		16x16	1038	宋体	W-横置横排	
		24x24	1038	宋体	W-横置横排	
		32x32	1038	宋体	W-横置横排	
Unicode -> GBK 转码表			20902+1038			
其它图符集	条形码字符 EAN13	12x27	60	标准	W-横置横排	
	条形码字符 CODE128	16x20	107	标准	W-横置横排	
	天线符号	12x12	5		W-横置横排	
	电池符号	12x12	4		W-横置横排	
BIG5-> GBK 转码表			13468			

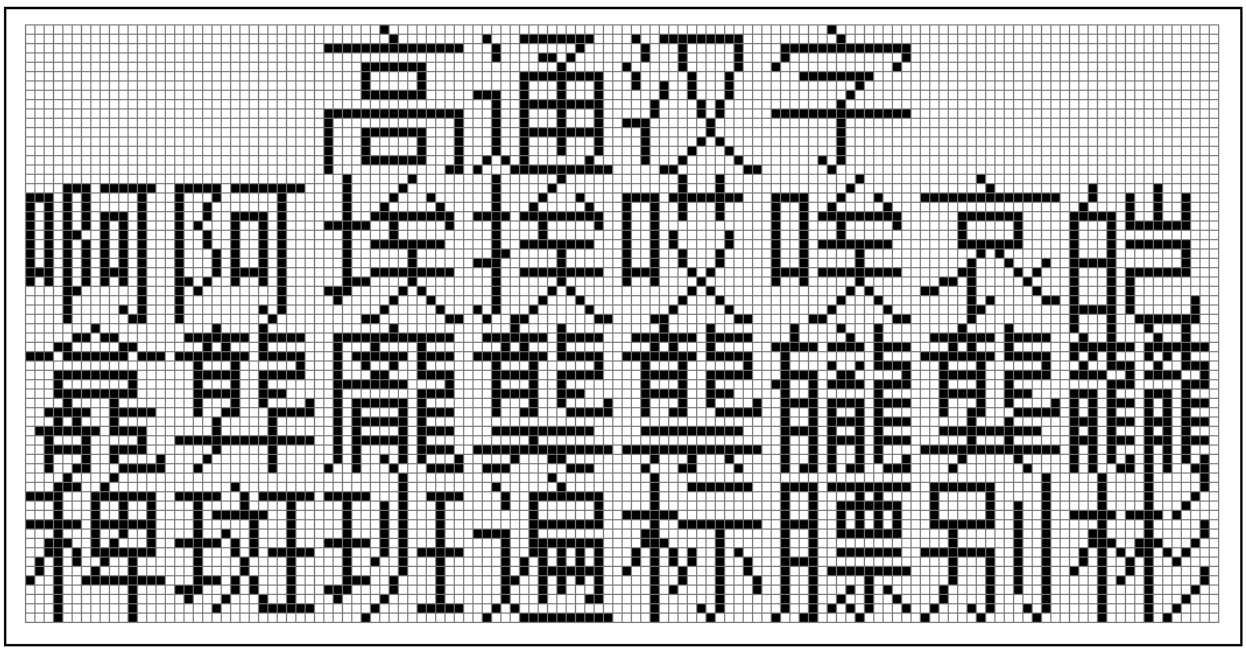
1.3 字型样张

1.3.1 汉字字符

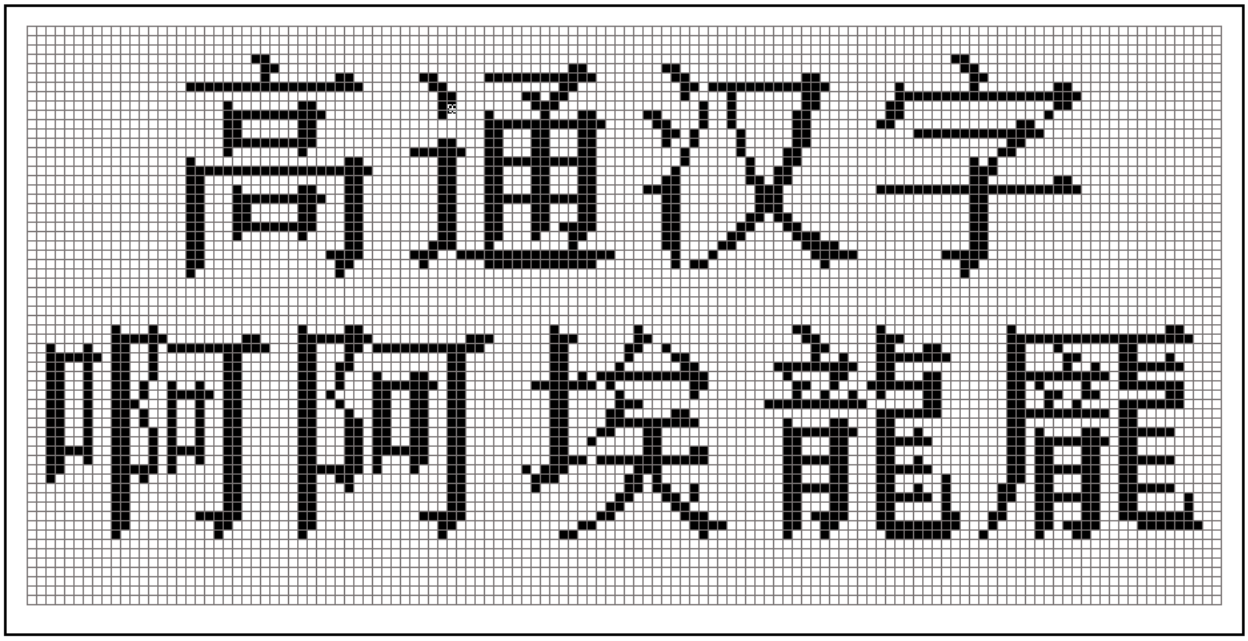
12x12 点阵 GB18030 汉字



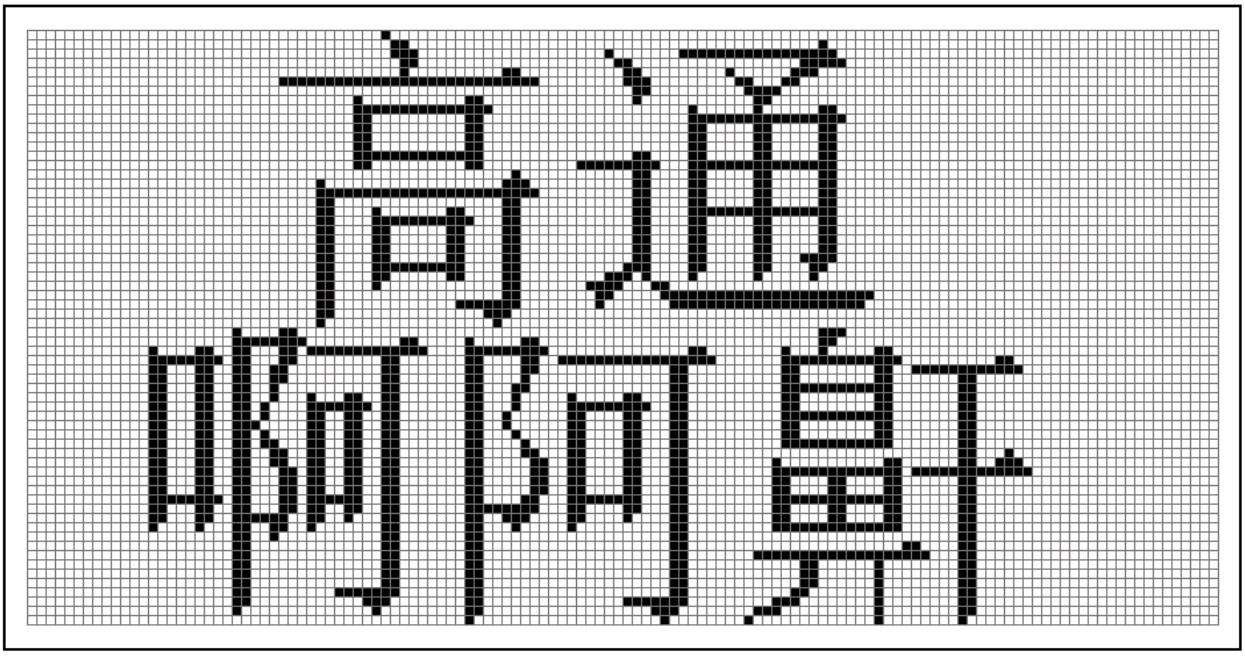
16x16 点阵 GB18030 汉字



24x24 点阵　GB18030 汉字

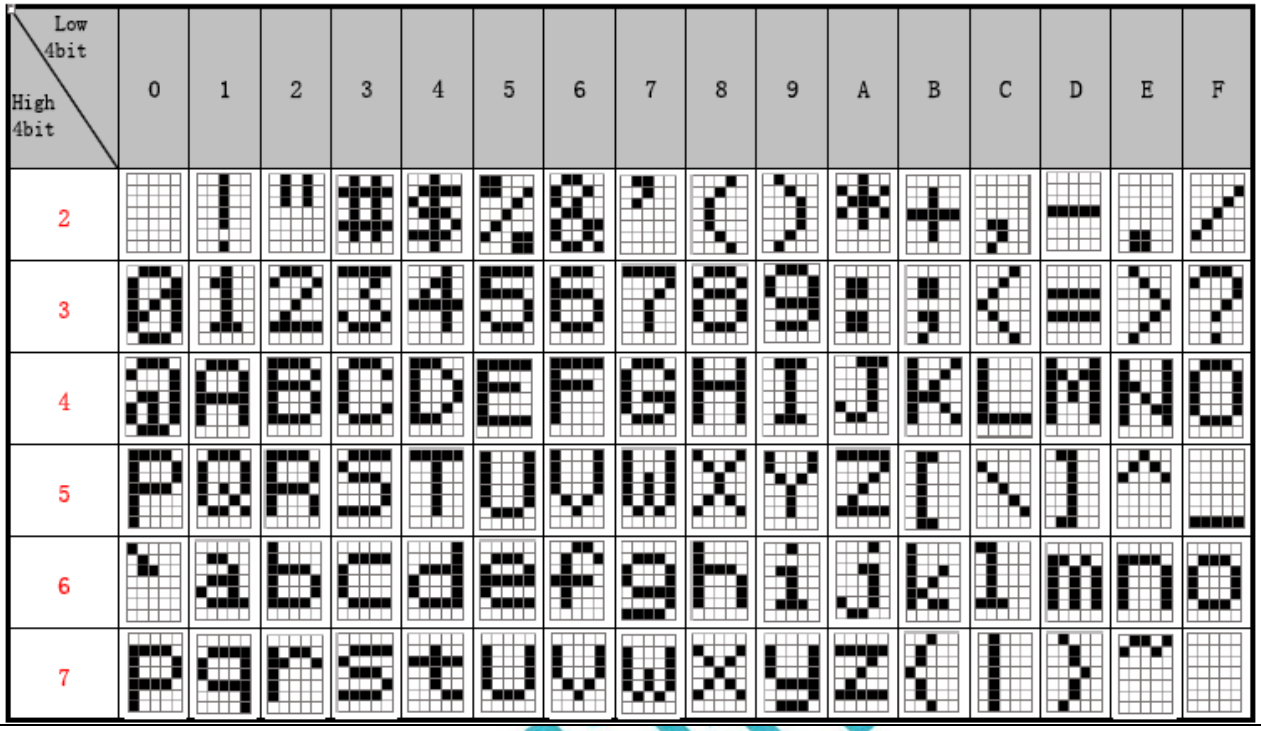


32x32 点阵　GB18030 汉字



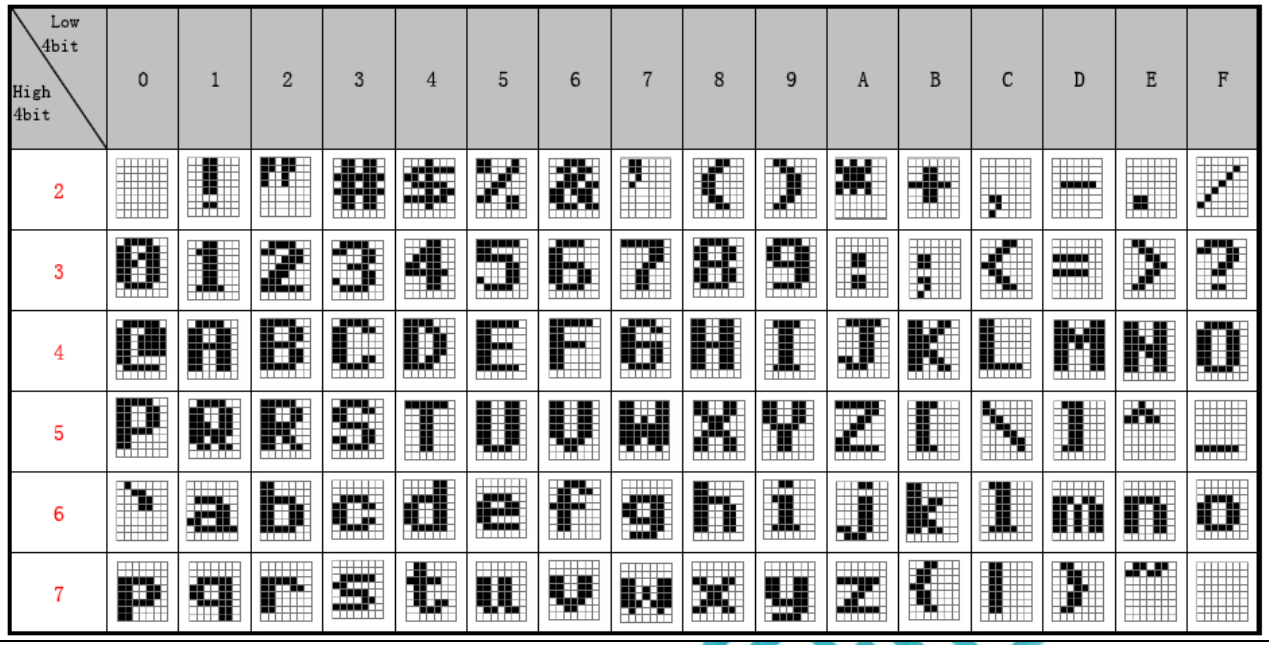
1.3.2 其它点阵字符

5x7 点阵 ASCII 标准字符

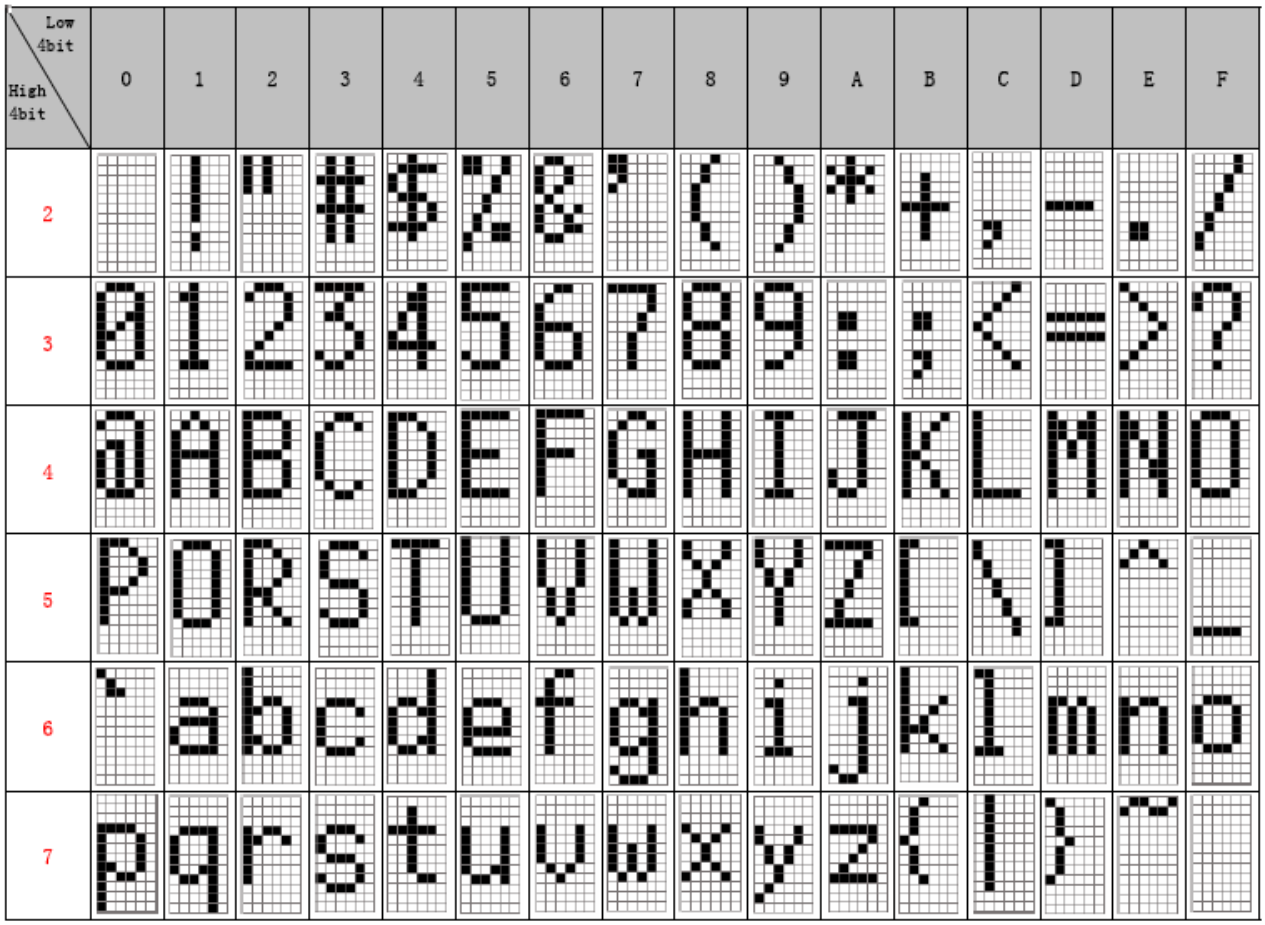


7x8 点阵 ASCII 标准字符

7x8 点阵 ASCII 粗体字符



6x12 点阵 ASCII 标准字符

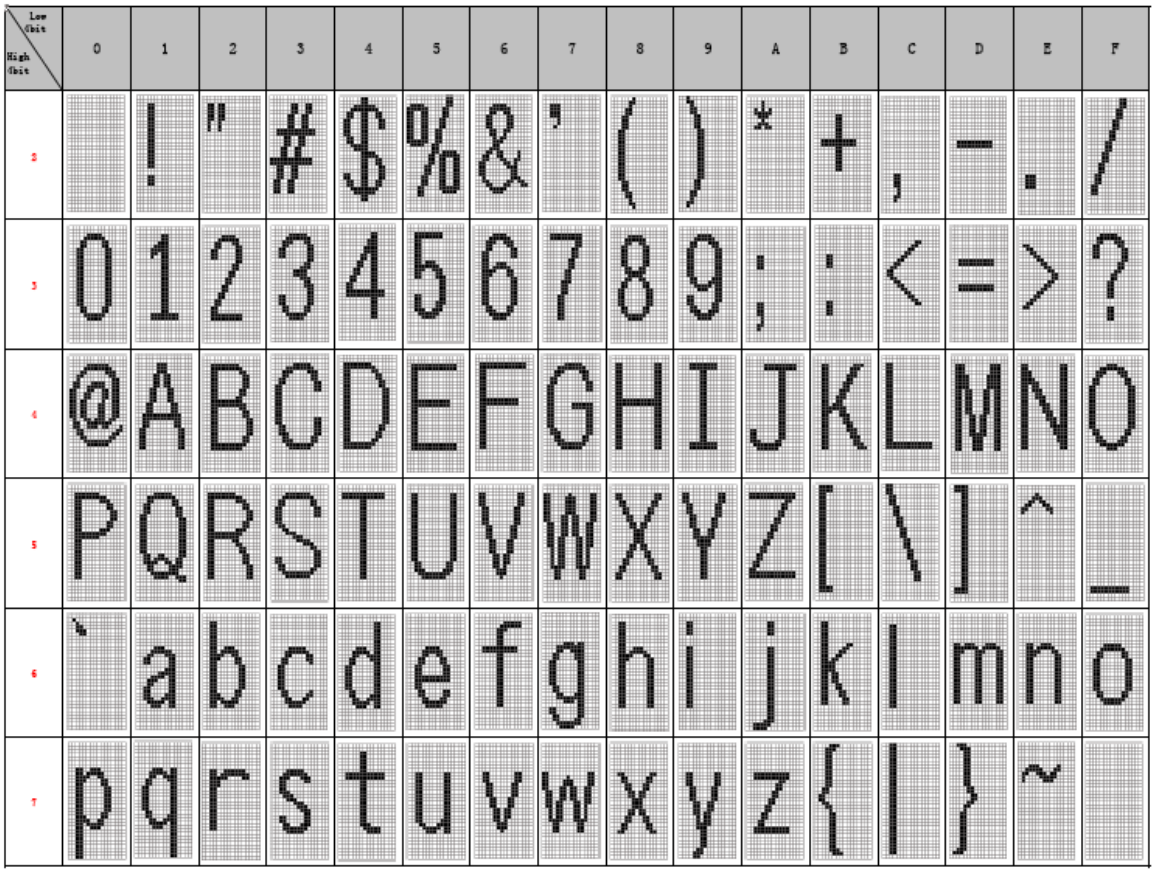


8x16 点阵 ASCII 标准字符

8x16 点阵 ASCII 粗体字符

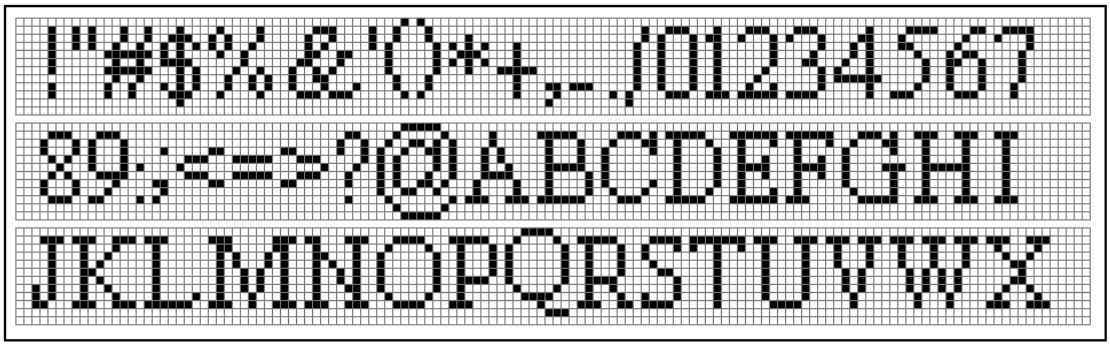
12x24 点阵 ASCII 标准字符

16x32 点阵 ASCII 标准字符

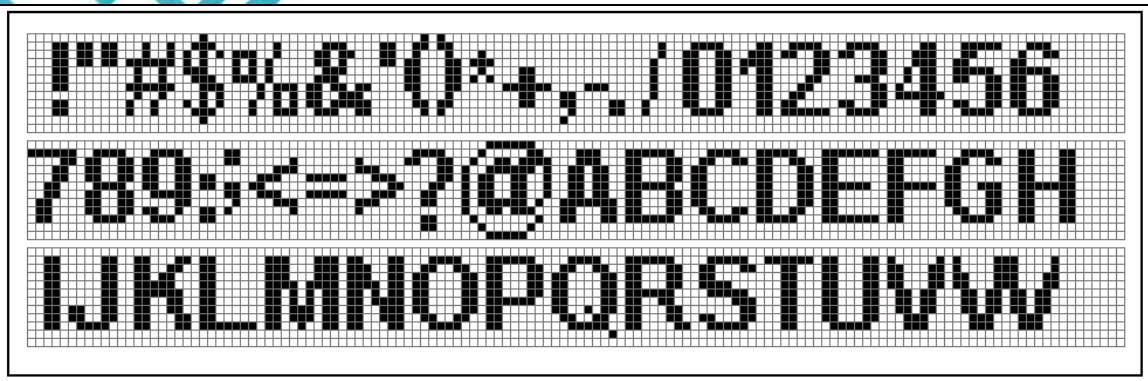


16x32 点阵 ASCII 粗体字符

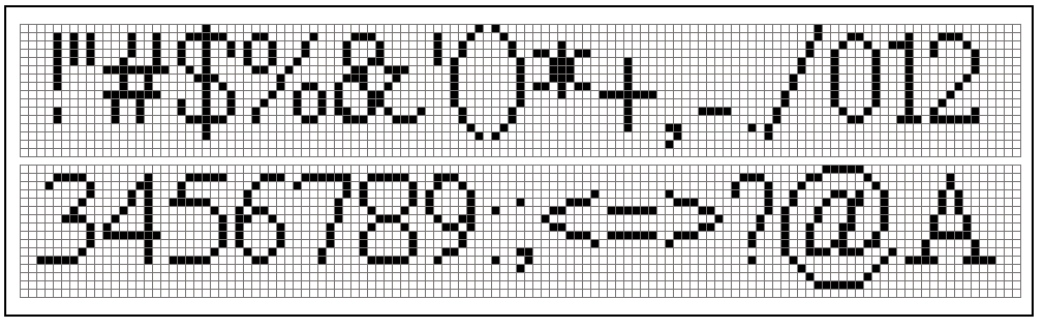
12 点阵不等宽 ASCII 白正(Time new Roman)



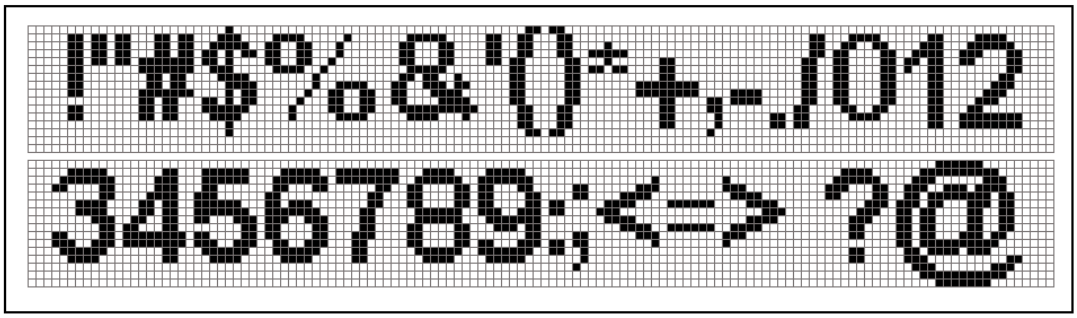
12 点阵不等宽 ASCII 方头（Arial）



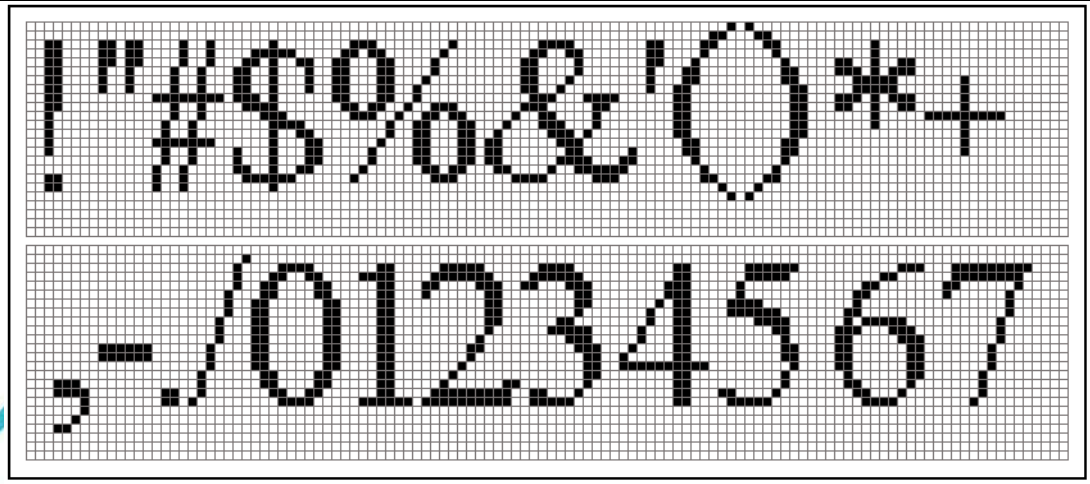
16 点阵不等宽 ASCII 白正(Time new Roman)



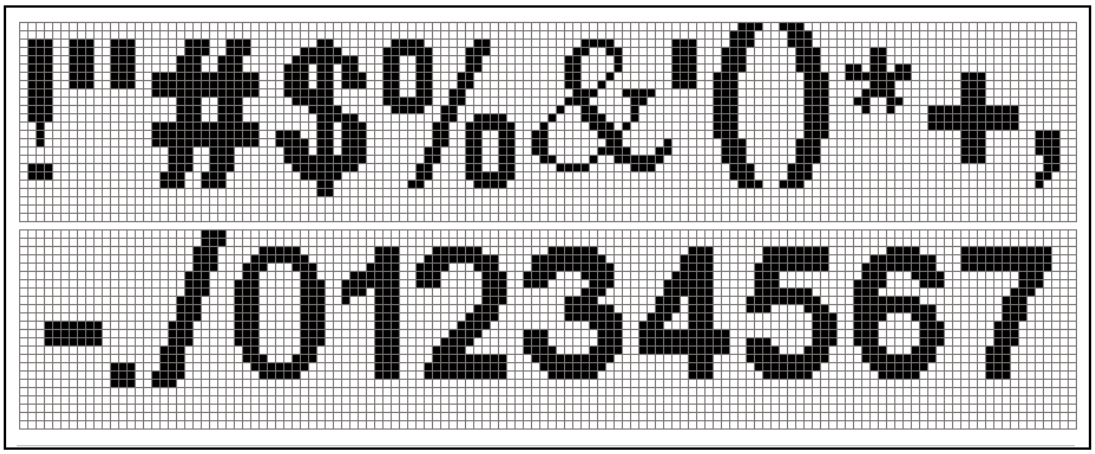
16 点阵不等宽 ASCII 方头（Arial）



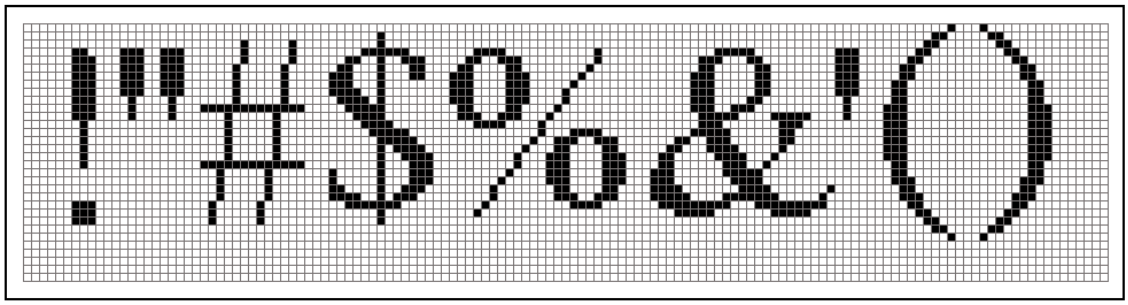
24 点阵不等宽 ASCII 白正(Time new Roman)



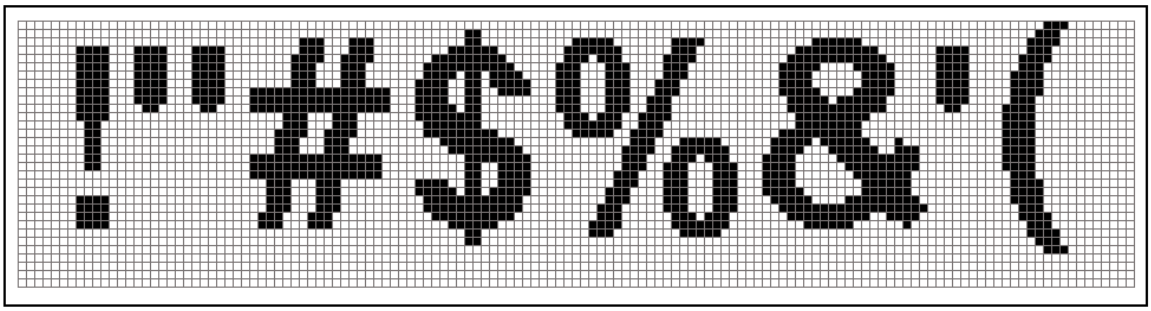
24 点阵不等宽 ASCII 方头 （Arial）



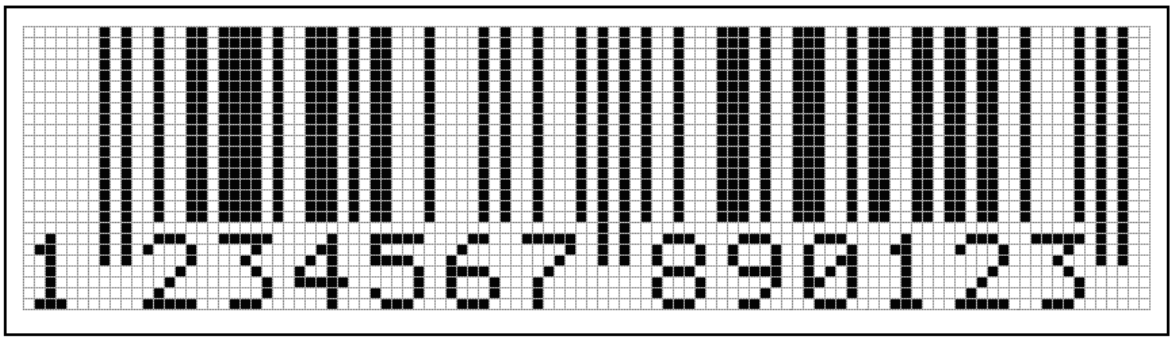
32 点阵不等宽 ASCII 白正(Time new Roman)



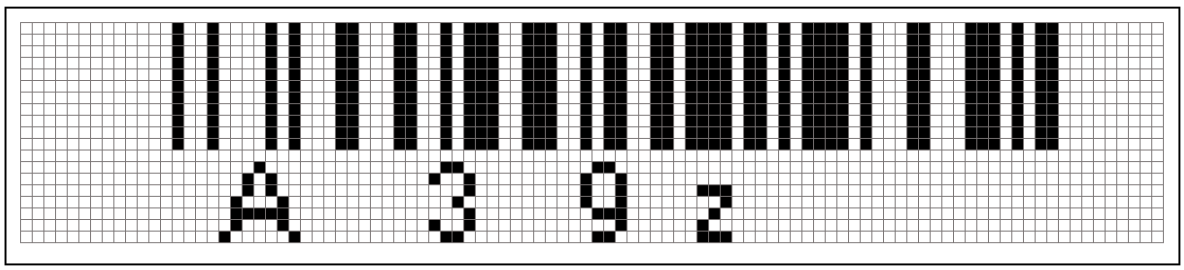
32 点阵不等宽 ASCII 方头（Arial）



条形码字符 EAN13



条形码字符 CODE128



2. 操作指令

2.1 Instruction Parameter(指令参数)

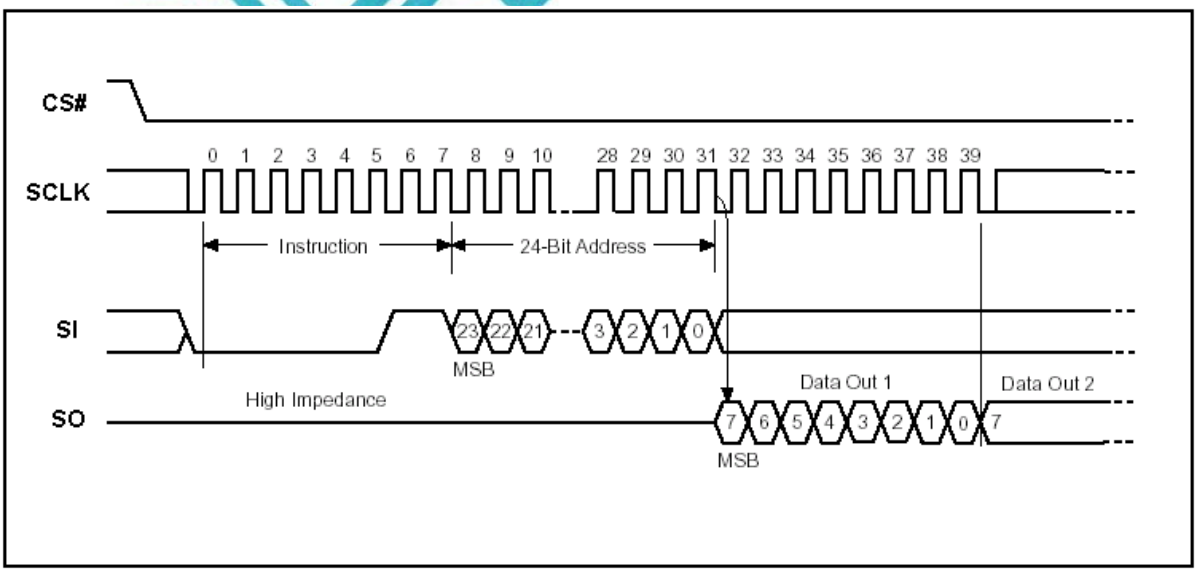
Instruction	Description	Instruction Code(One-Byte)		Address Bytes	Dummy Bytes	Data Bytes
Read	Read Data Bytes	0000 0011	03 h	3	—	1 to ∞
Fast Read	Read Data Bytes at Higher Speed	0000 1011	0B h	3	1	1 to ∞
WREN	Write Enalbe	0000 0110	06 h	—	—	—
WRDI	Write Disable	0000 0100	04 h			
PP	Page Program	0000 0010	02 h	3	—	1 to256
SE	Sector Erase	0010 0000	20 h	3	—	—
BE	Block Erase(64K)	1101 1000	D8 h	3	—	—
CE	Chip Erase	0110 0000/ 1100 0111	60 H/ C7 H	—	—	—

2.2 Read Data Bytes（一般读取）

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

- 首先把片选信号（CS#）变为低，紧跟着的是 1 个字节的命令字（03 h）和 3 个字节的地址和通过串行数据输入引脚（SI）移位输入，每一位在串行时钟（SCLK）上升沿被锁存。
- 然后该地址的字节数据通过串行数据输出引脚（SO）移位输出，每一位在串行时钟（SCLK）下降沿被移出。
- 读取字节数据后，则把片选信号（CS#）变为高，结束本次操作。
  如果片选信号（CS#）继续保持为底，则下一个地址的字节数据继续通过串行数据输出引脚（SO）移位输出。

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



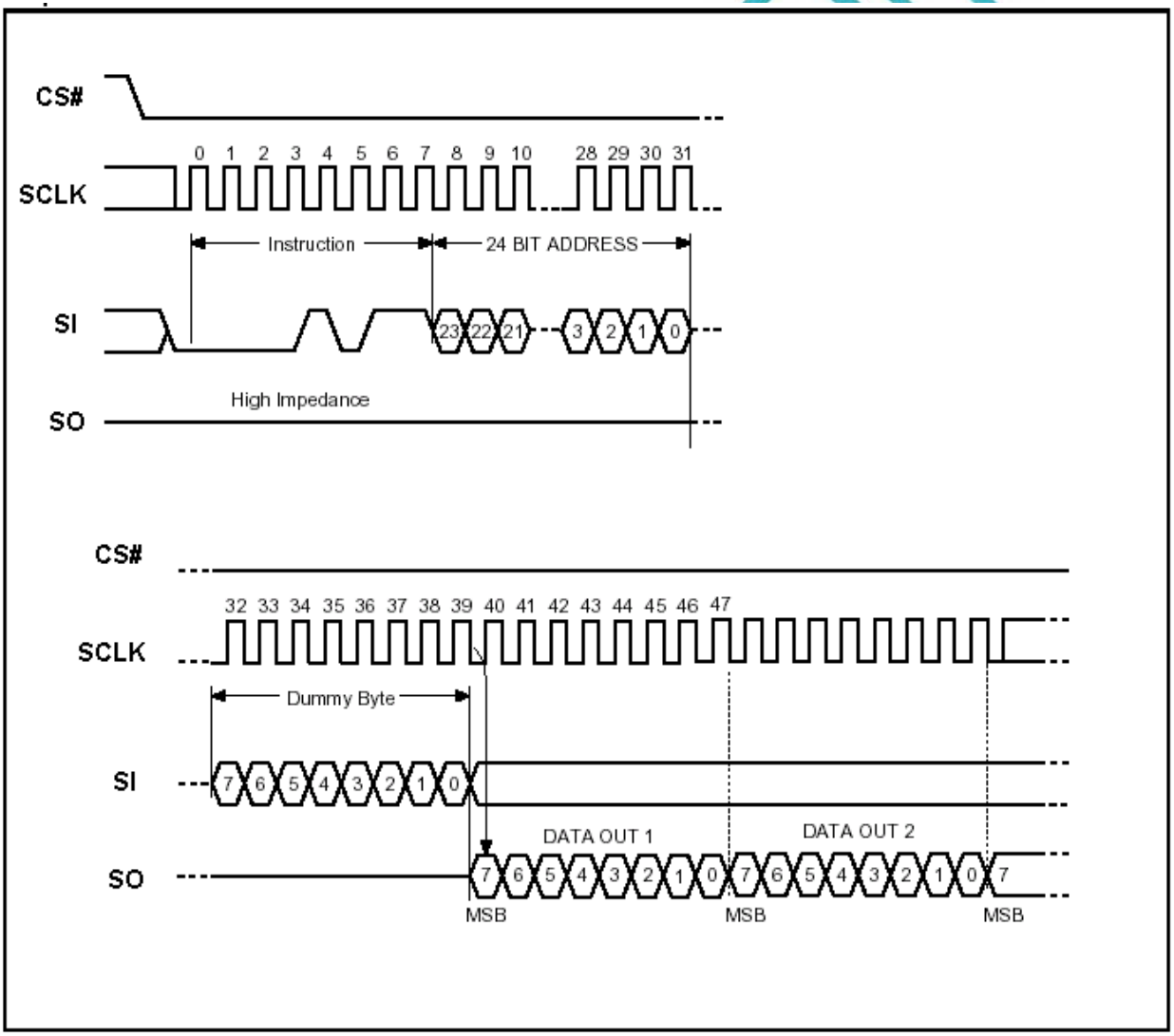
2.3 Read Data Bytes at Higher Speed（快速读取点阵数据）

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

- 首先把片选信号（CS#）变为低，紧跟着的是 1 个字节的命令字（0B h）和 3 个字节的地址以及一个字节 Dummy Byte 通过串行数据输入引脚（SI）移位输入，每一位在串行时钟（SCLK）上升沿被锁存。
- 然后该地址的字节数据通过串行数据输出引脚（SO）移位输出，每一位在串行时钟（SCLK）下降沿被移出。
- 如果片选信号（CS#）继续保持为底，则下一个地址的字节数据继续通过串行数据输出引脚（SO）移位输出。例：读取一个 15x16 点阵汉字需要 32Byte，则连续 32 个字节读取后结束一个汉字的点阵数据读取操作。

  如果不需要继续读取数据，则把片选信号（CS#）变为高，结束本次操作。

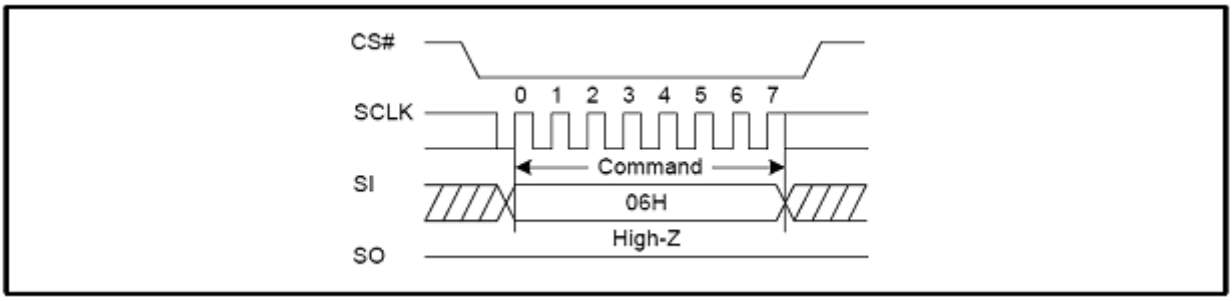
图：Read Data Bytes at Higher Speed (READ_FAST) Instruction Sequence and Data-out sequence:



2.4 Write Enable（写使能）

Write Enable 指令的时序如下(图)：

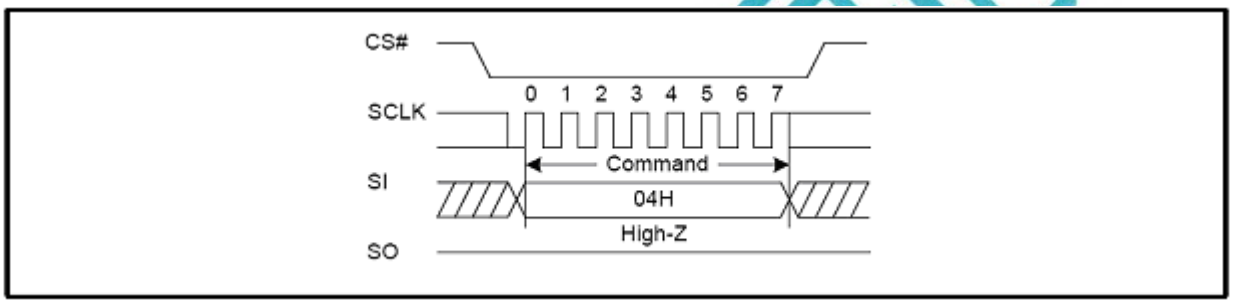
CS#变低-→发送 Write Enable 命令—>CS#变高



2.5 Write Disable（写非能）

Write Enable 指令的时序如下(图)：

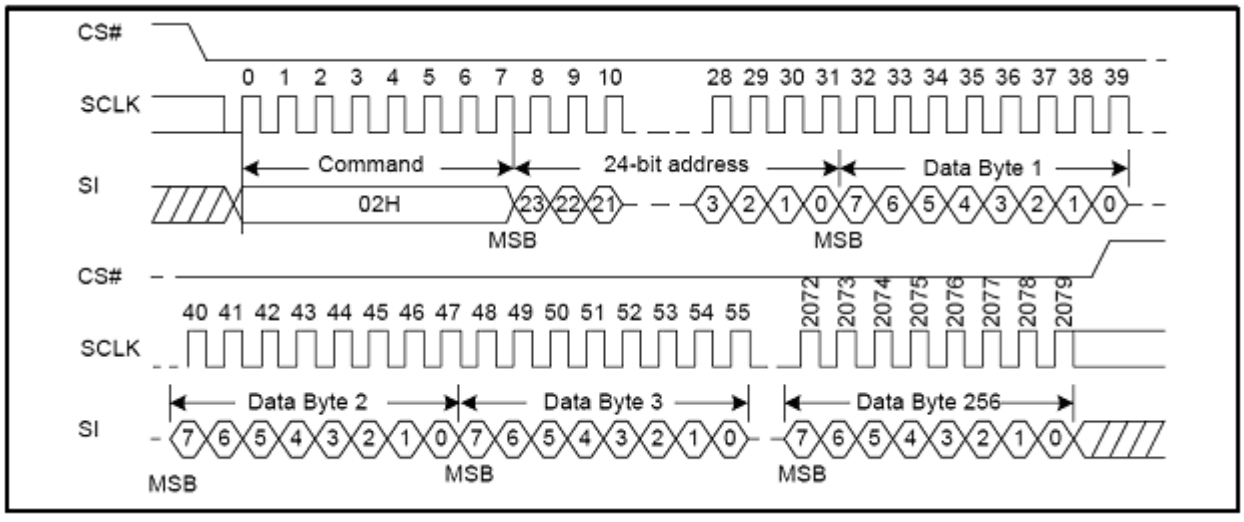
CS#变低-→发送 Write Disable 命令—>CS#变高



2.6 Page Program（页写入）

Page Program 指令的时序如下(图)：

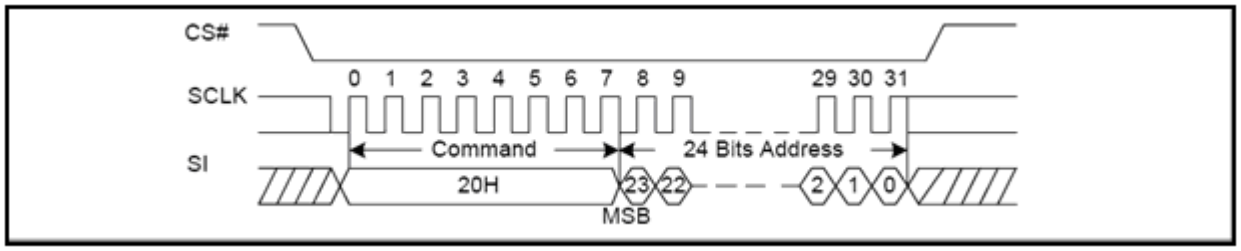
CS#变低-→发送 Page Program 命令→发送 3 字节地址—>发送数据—>CS#变高



2.7 Sector Erase（扇区擦除）

Sector Erase 指令的时序如下(图)：

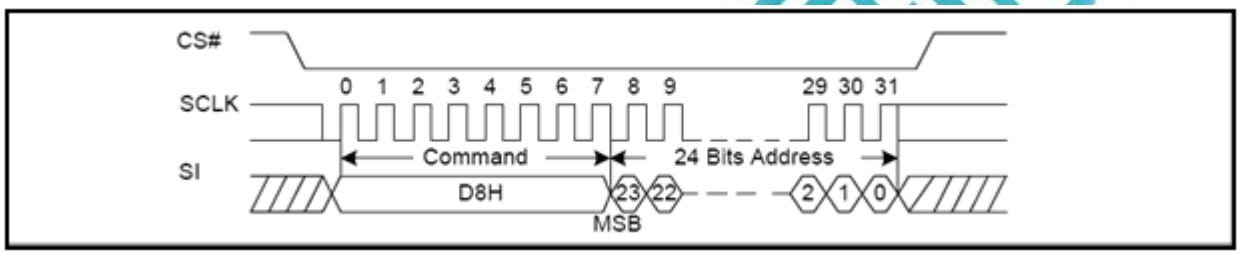
CS#变低->发送 Sector Erase 命令->发送 3 字节地址—>CS#变高



2.8 Block Erase(64K)（块擦除）

Block Erase 指令的时序如下(图)：

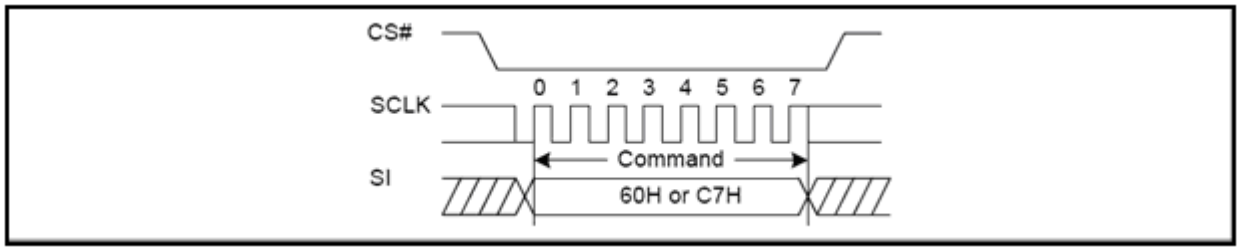
CS#变低->发送 64K Block Erase 命令->发送 3 字节地址—>CS#变高



2.9 Chip Erase（芯片擦除）

Chip Erase 指令的时序如下(图)：

CS#变低->发送 Chip Erase 命令—>CS#变高



3. 字符点阵字库地址表

NO.	字库内容	编码体系（字符集）	字符数	起始地址	参考算法
1	5x7 点阵 ASCII 标准字符	ASCII	96	0x100000	4.2.1
2	7x8 点阵 ASCII 标准字符	ASCII	96	0x100300	4.2.2
3	7x8 点阵 ASCII 粗体字符	ASCII	96	0x100600	4.2.3
4	6x12 点阵 ASCII 字符	ASCII	96	0x100900	4.2.4
5	8x16 点阵 ASCII 标准字符	ASCII	128	0x100D80	4.2.5
6	8x16 点 ASCII 粗体字符	ASCII	96	0x101580	4.2.6
7	12x24 点阵 ASCII 字符	ASCII	96	0x101B80	4.2.7
8	16x32 点阵 ASCII 标准字符	ASCII	96	0x102D80	4.2.8
9	16x32 点阵 ASCII 粗体字符	ASCII	96	0x104580	4.2.9
10	12 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96	0x105D80	4.2.10
11	16 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96	0x106740	4.2.11
12	24 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96	0x107400	4.2.12
13	32 点阵不等宽 ASCII 方头（Arial）字符	ASCII	96	0x108FC0	4.2.13
14	12 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96	0x10C080	4.2.14
15	16 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96	0x10CA50	4.2.15
16	24 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96	0x10D740	4.2.16
17	32 点阵不等宽 ASCII 白正（Times New Roman）字符	ASCII	96	0x10F340	4.2.17
18	14x28 数字符号字符	数字符号字符	15	0x112400	4.2.18
19	20x40 数字符号字符	数字符号字符	12	0x112748	4.2.19
20	28 点阵不等宽数字符号字符	数字符号字符	15	0x112CE8	4.2.20
21	40 点阵不等宽数字符号字符	数字符号字符	12	0x113396	4.2.21
22	12x12 点阵 GB18030 汉字	GB18030	20902	0x113D0E	4.1.1
23	16x16 点阵 GB18030 汉字	GB18030	27484	0x194FDE	4.1.2
24	24x24 点阵 GB18030 汉字	GB18030	27484	0x2743DE	4.1.3
25	12x12 点阵 GB18030 字符	GB18030	1038	0x113D0E	4.1.1
26	16x16 点阵 GB18030 字符	GB18030	1038	0x194FDE	4.1.2
27	24x24 点阵 GB18030 字符	GB18030	1038	0x2743DE	4.1.3
28	Unicode->GBK 转码表		20902+1038	0x46A90E	4.3.1/4.3.2
29	12x27 条形码字符 EAN13		60	0x478FD2	4.2.22
30	16x20 条形码字符 CODE128		107	0x479C7A	4.2.23

31	12x12 天线符号		5	0x47AD32	4.2.24
32	12x12 电池符号		4	0x47ADAA	4.2.25
33	32x32 点阵 GB18030 汉字	GB18030	27484	0x47AE10	4.1.4
34	32x32 点阵 GB18030 字符	GB18030	1038	0x47AE10	4.1.4
35	BIG5->GBK 转码表		20902+ 1038	0x7F1E10	4.3.3
36	保留区	0x7F8760~0x7FFFFF			

4. 字符点阵数据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。

4.1 汉字字符点阵数据的地址计算

4.1.1 12x12 点阵 GB18030 汉字&字符

12x12 点阵字库起始地址：BaseAdd＝0x113D0E，
Address：对应字符点阵在芯片中的字节地址。
地址的计算由下面的函数实现（ANSI C 语言编写）

```
/*****************************************************************************************************
函数：unsigned long gt(unsigned char c1, unsigned char c2, unsigned char c3, unsigned

char c4)
功能：计算汉字点阵在芯片中的地址
参数：汉字内码通过参数 c1,c2 传入，c3=0;c4=0. 注：12x12 为 GBK 字符集，无四字节区。
返回：汉字点阵的字节地址(byte address)。如果用户是按 word mode 读取点阵数据，则其地址(word address)为字节地址除以 2，即：word address = byte address / 2 .
例如：BaseAdd: 说明汉字点阵数据在字库芯片中的起始地址，即 BaseAdd＝0x093D0E，
“啊”字的内码为 0xb0a1,则 byte address = gt(0xb0,0xa1,0x00,0x00) *24+BaseAdd
                                    word address = byte address / 2
*****************************************************************************************************/
unsigned long gt (unsigned char c1, unsigned char c2, unsigned char c3, unsigned char

c4)
{
    unsigned long h=0;
    if(c2==0x7f) return (h);
    if(c1>=0xA1 && c1 <= 0xa9 && c2>=0xa1)        //Section 1
```

```
        h= (c1 - 0xA1) * 94 + (c2 - 0xA1);
    else if(c1>=0xa8 && c1 <= 0xa9 && c2<0xa1)      //Section 5
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xa8)*96 + (c2-0x40)+846;
    }
    if(c1>=0xb0 && c1 <= 0xf7 && c2>=0xa1)          //Section 2
        h= (c1 - 0xB0) * 94 + (c2 - 0xA1)+1038;
    else if(c1<0xa1 && c1>=0x81 && c2>=0x40 )       //Section 3
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0x81)*190 + (c2-0x40) + 1038 +6768;
    }
    else if(c1>=0xaa && c2<0xa1)                    //Section 4
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xaa)*96 + (c2-0x40) + 1038 +12848;
    }
    return(h);
}
Address = gt(c1,c2,0x00,0x00) *24+BaseAdd;
```

4.1.2 16x16 点阵 GB18030 汉字&字符

16x16 点阵字库起始地址：BaseAdd＝0x194FDE，
Address：对应字符点阵在芯片中的字节地址。
地址的计算由下面的函数实现（ANSI C 语言编写）

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

函数：unsigned long gt(unsigned char c1, unsigned char c2, unsigned char c3, unsigned char c4)
功能：计算汉字点阵在芯片中的地址
参数：c1,c2,c3,c4：4 字节汉字内码通过参数 c1,c2,c3,c4 传入，双字节内码通过参数 c1,c2 传入，c3=0,c4=0
返回：汉字点阵的字节地址(byte address)。如果用户是按 word mode 读取点阵数据，则其地址(word address)为字节地址除以 2，即：word address = byte address / 2 .
例如：BaseAdd: 说明汉字点阵数据在字库芯片中的起始地址，即 BaseAdd＝0x114FDE;
“啊”字的内码为 0xb0a1,则 byte address = gt(0xb0,0xa1,0x00,0x00) *32+BaseAdd
word address = byte address / 2
“ ”字的内码为 0x8139ee39,则 byte address = gt(0x81,0x39,0xee,0x39) *32+ BaseAdd
word address = byte address / 2

```
**********************************************************************************************/
unsigned long gt (unsigned char c1, unsigned char c2, unsigned char c3, unsigned char

c4)
{
    unsigned long h=0;
    if(c2==0x7f) return (h);
    if(c1>=0xA1 && c1 <= 0xA0 && c2>=0xa1)          //Section 1
        h= (c1 - 0xA1) * 94 + (c2 - 0xA1);
    else if(c1>=0xa8 && c1 <= 0xa9 && c2<0xa1)      //Section 5
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xa8)*96 + (c2-0x40)+846;
    }
    if(c1>=0xb0 && c1 <= 0xf7 && c2>=0xa1)          //Section 2
        h= (c1 - 0xB0) * 94 + (c2 - 0xA1)+1038;
    else if(c1<0xa1 && c1>=0x81 && c2>=0x40 )    //Section 3
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0x81)*190 + (c2-0x40) + 1038 +6768;
    }
    else if(c1>=0xaa && c2<0xa1)                    //Section 4
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xaa)*96 + (c2-0x40) + 1038 +12848;
    }
else    if(c1==0x81 && c2>=0x39) //四字节区 1
{
   h =1038 + 21008+(c3-0xEE)*10+c4-0x39;
}
else if(c1==0x82)//四字节区 2
{
    h =1038 + 21008+161+(c2-0x30)*1260+(c3-0x81)*10+c4-0x30;
    }
    return(h);
}
Address = gt(c1,c2,c3,c4) *32+BaseAdd;
```

4.1.3 24x24 点阵 GB18030 汉字&字符

24x24 点阵字库起始地址：BaseAdd＝0x2743DE，

Address：对应字符点阵在芯片中的字节地址。

地址的计算由下面的函数实现（ANSI C 语言编写）

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

函数：unsigned long gt(unsigned char c1, unsigned char c2, unsigned char c3, unsigned char c4)

功能：计算汉字点阵在芯片中的地址

参数：c1,c2,c3,c4：4 字节汉字内码通过参数 c1,c2,c3,c4 传入，双字节内码通过参数 c1,c2 传入，c3=0,c4=0

返回：汉字点阵的字节地址(byte address)。如果用户是按 word mode 读取点阵数据，则其地址(word address)为字节地址除以 2，即：word address = byte address / 2 .

例如：BaseAdd: 说明汉字点阵数据在字库芯片中的起始地址，即 BaseAdd＝0x1F43DE;

“啊”字的内码为 0xb0a1,则 byte address = gt(0xb0,0xa1,0x00,0x00) *72+BaseAdd

word address = byte address / 2

“ ”字的内码为 0x8139ee39,则 byte address = gt(0x81, 0x39,0xee,0x39) *72+ BaseAdd

word address = byte address / 2

```
*****************************************************************************************************/
unsigned long gt (unsigned char c1, unsigned char c2, unsigned char c3, unsigned char c4)
{
    unsigned long h=0;
    if(c2==0x7f) return (h);
    if(c1>=0xA1 && c1 <= 0xAB && c2>=0xa1)          //Section 1
        h= (c1 - 0xA1) * 94 + (c2 - 0xA1);
    else if(c1>=0xa8 && c1 <= 0xa9 && c2<0xa1)      //Section 5
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xa8)*96 + (c2-0x40)+846;
    }
    if(c1>=0xb0 && c1 <= 0xf7 && c2>=0xa1)          //Section 2
        h= (c1 - 0xB0) * 94 + (c2 - 0xA1)+1038;
    else if(c1<0xa1 && c1>=0x81 && c2>=0x40)        //Section 3
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0x81)*190 + (c2-0x40) + 1038 +6768;
    }
    else if(c1>=0xaa && c2<0xa1)                    //Section 4
    {
        if(c2>0x7f)
```

```
            c2--;
        h=(c1-0xaa)*96 + (c2-0x40) + 1038 +12848;
    }
    else   if(c1==0x81 && c2>=0x39) //四字节区 1
    {
            h =1038 + 21008+(c3-0xEE)*10+c4-0x39;
    }
    else if(c1==0x82)//四字节区 2
    {
        h =1038 + 21008+161+(c2-0x30)*1260+(c3-0x81)*10+c4-0x30;
    }
    return(h);
}
Address = gt(c1,c2,c3,c4) *72+BaseAdd;
```

4.1.4 32x32 点阵 GB18030 汉字&字符

32x32 点阵字库起始地址：BaseAdd＝0x47AE10，
Address：对应字符点阵在芯片中的字节地址。
地址的计算由下面的函数实现（ANSI C 语言编写）

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

函数：unsigned long gt(unsigned char c1, unsigned char c2, unsigned char c3, unsigned char c4)
功能：计算汉字点阵在芯片中的地址
参数：c1,c2,c3,c4：4 字节汉字内码通过参数 c1,c2,c3,c4 传入，双字节内码通过参数 c1,c2 传入，c3=0,c4=0

返回：汉字点阵的字节地址(byte address)。如果用户是按 word mode 读取点阵数据，则其地址(word address)为字节地址除以 2，即：word address = byte address / 2 .

例如：BaseAdd: 说明汉字点阵数据在字库芯片中的起始地址，即 BaseAdd＝0x1F43DE;
“啊”字的内码为 0xb0a1,则 byte address = gt(0xb0,0xa1,0x00,0x00) *72+BaseAdd
word address = byte address / 2
“㐂”字的内码为 0x8139ee39,则 byte address = gt(0x81, 0x39,0xee,0x39) *72+ BaseAdd
word address = byte address / 2

```
*****************************************************************************************************/
unsigned long gt (unsigned char c1, unsigned char c2, unsigned char c3, unsigned char c4)
{
    unsigned long h=0;
    if(c2==0x7f) return (h);
    if(c1>=0xA1 && c1 <= 0xAB && c2>=0xa1)          //Section 1
```

```
        h= (c1 - 0xA1) * 94 + (c2 - 0xA1);
    else if(c1>=0xa8 && c1 <= 0xa9 && c2<0xa1)       //Section 5
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xa8)*96 + (c2-0x40)+846;
    }
    if(c1>=0xb0 && c1 <= 0xf7 && c2>=0xa1)          //Section 2
        h= (c1 - 0xB0) * 94 + (c2 - 0xA1)+1038;
    else if(c1<0xa1 && c1>=0x81 && c2>=0x40)       //Section 3
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0x81)*190 + (c2-0x40) + 1038 +6768;
    }
    else if(c1>=0xaa && c2<0xa1)                    //Section 4
    {
        if(c2>0x7f)
            c2--;
        h=(c1-0xaa)*96 + (c2-0x40) + 1038 +12848;
    }
    else    if(c1==0x81 && c2>=0x39) //四字节区 1
    {
            h =1038 + 21008+(c3-0xEE)*10+c4-0x39;
    }
    else if(c1==0x82)//四字节区 2
    {
         h =1038 + 21008+161+(c2-0x30)*1260+(c3-0x81)*10+c4-0x30;
    }
    return(h);
}
Address = gt(c1,c2,c3,c4) *128+BaseAdd;
```

4.2 其它字符点阵数据的地址计算

4.2.1 5x7 点阵 ASCII 标准字符

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=0x100000

if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then

Address = (ASCIICode –0x20) * 8+BaseAdd

4.2.2 7x8 点阵 ASCII 标准字符

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=0x100300
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 8+BaseAdd

4.2.3 7x8 点阵 ASCII 粗体字符

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=0x100600
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 8+BaseAdd

4.2.4 6x12 点阵 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=0x100900
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 12+BaseAdd

4.2.5 8x16 点阵 ASCII 标准字符

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=0x100D80
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 16+BaseAdd

4.2.6 8x16 点阵 ASCII 粗体字符

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=0x101580
if((ASCIICode>=0x20)&&(ASCIICode<=0x7F))
Address = (ASCIICode-0x20)*16+BaseAdd

4.2.7 12x24 点阵 ASCII 标准字符

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=0x101B80
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 48+BaseAdd

4.2.8 16x32 点阵 ASCII 标准字符

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=0x102D80
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 64+BaseAdd

4.2.9 16x32 点阵 ASCII 粗体字符

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=0x104580
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 64+BaseAdd

4.2.10 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=0x105D80
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 26 + BaseAdd

4.2.11 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=0x106740
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 34 + BaseAdd

4.2.12 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=0x107400
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 74 + BaseAdd

4.2.13 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=0x108FC0
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 130 + BaseAdd

4.2.14 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=0x10C080
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) *26 + BaseAdd

4.2.15 16 点阵不等宽 ASCII 白正（Times New Roman）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=0x10CA50
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 34 + BaseAdd

4.2.16 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=0x10D740
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
    Address = (ASCIICode –0x20) * 74+ BaseAdd

4.2.17 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=0x10F340
if (ASCIICode >= 0x20) and (ASCIICode <= 0x7E) then
Address = (ASCIICode –0x20) * 130 + BaseAdd

4.2.18 14x28 点阵数字符号字符

说明：此部分内容为 0 1 2 3 4 5 6 7 8 9 , . ￥ $ £
Squence：表示 字符顺序，从 0 开始计数。
BaseAdd：说明该套字库在芯片中的起始地址。
Address：对应字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x112400
Address = Squence * 56+ BaseAdd

4.2.19 20x40 点阵数字符号字符

说明：此部分内容为 0 1 2 3 4 5 6 7 8 9 . ,
Squence：表示 字符顺序，从 0 开始计数。
BaseAdd：说明该套字库在芯片中的起始地址。
Address：对应字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x112748
Address = (Squence) * 120+BaseAdd

4.2.20 28 点阵不等宽数字符号字符

说明：此部分内容为 0 1 2 3 4 5 6 7 8 9 , . ￥ $ £
Squence：表示 字符顺序，从 0 开始计数。
BaseAdd：说明该套字库在芯片中的起始地址。
Address：对应字符点阵在芯片中的字节地址。
注：前两个字节为宽度信息。
计算方法：
BaseAdd=0x112CE8
Address = Squence * 114+ BaseAdd

4.2.21 40 点阵不等宽数字符号字符

说明：此部分内容为 0 1 2 3 4 5 6 7 8 9 . ,
Squence：表示 字符顺序，从 0 开始计数。
BaseAdd：说明该套字库在芯片中的起始地址。
Address：对应字符点阵在芯片中的字节地址。
计算方法：
BaseAdd=0x113396
Address = Squence * 202+ BaseAdd

4.2.22 EAN13 条形码调用程序

函数：DWORD* BAR_CODE(int* BAR_NUM)
功能：将数组条形码转为对应条形码图形地址。
参数：int* BAR_NUM 条形码数字数组指针，BAR_NUM[13]数组包含 13 个数字。
返回：定义 DWORD BAR_PIC_ADDR[13];用于存放对应地址，返回此数组指针。

```
DWORD BAR_PIC_ADDR[13];
DWORD* BAR_CODE(int* BAR_NUM)
{
    DWORD i,BaseAddr=0x478FD2;
BAR_PIC_ADDR[0]=BAR_NUM[0]*54+540*0+ BaseAddr;
BAR_PIC_ADDR[1]=BAR_NUM[1]*54+540*1+ BaseAddr;

    switch(BAR_NUM[0])
    {
    case 0:
        for(i=2;i<=6;i++)
        {
            BAR_PIC_ADDR[i]=BAR_NUM[i]*54+540*1+ BaseAddr;
        }
        break;

    case 1:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*2+ BaseAddr;
        break;
    case 2:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*1+ BaseAddr;
```

```
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*2+ BaseAddr;
        break;
    case 3:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*1+ BaseAddr;
        break;


case 4:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*2+ BaseAddr;
        break;
    case 5:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*2+ BaseAddr;
        break;
    case 6:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*1+ BaseAddr;
        break;
    case 7:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*2+ BaseAddr;
        break;
    case 8:
        BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*1+ BaseAddr;
        BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*2+ BaseAddr;
        BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*2+ BaseAddr;
```

```
            BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*1+ BaseAddr;
            break;
        case 9:
            BAR_PIC_ADDR[2]=BAR_NUM[2]*54+540*2+ BaseAddr;
            BAR_PIC_ADDR[3]=BAR_NUM[3]*54+540*2+ BaseAddr;
            BAR_PIC_ADDR[4]=BAR_NUM[4]*54+540*1+ BaseAddr;
            BAR_PIC_ADDR[5]=BAR_NUM[5]*54+540*2+ BaseAddr;
            BAR_PIC_ADDR[6]=BAR_NUM[6]*54+540*1+ BaseAddr;
            break;

        }


BAR_PIC_ADDR[7]=BAR_NUM[7]*54+540*3+ BaseAddr;
        for(i=8;i<=11;i++)
        {
            BAR_PIC_ADDR[i]=BAR_NUM[i]*54+540*4+ BaseAddr;
        }
        BAR_PIC_ADDR[12]=BAR_NUM[12]*54+540*5+ BaseAddr;
return BAR_PIC_ADDR;
}
```

4.2.23 GB/T 18347-2001(CODE128)条形码调用程序

函数：DWORD* BAR_CODE(int* BAR_NUM)
功能：将数组条形码转为对应条形码图形地址
参数：int* BAR_NUM 条形码数字数组指针,BAR_NUM[4]数组包含 4 个条形码 ASCII 符(数组取值

为 0~94)。
返回：定义 DWORD BAR_PIC_ADDR[7] ；用于存放对应地址，返回数组指针。
设基地址：BaseAddr= 0x479C7A
起始符有 3 种模式
当 flag=1 时为 Code-128-A;
当 flag=2 时为 Code-128-B;
当 flag=3 时为 Code-128-C;

```
DWORD flag;
DWORD BAR_PIC_ADDR[7];
DWORD* BAR_CODE(int* BAR_NUM)
{
  int i;
  for(i=0;i<7;i++)
  {
     switch(flag)
```

```
case 1 :
if(i==0)
{
  BAR_PIC_ADDR[i]=103*40+BaseAddr;
}
else if(i==1||i==2||i=3||i==4)
{
  BAR_PIC_ADDR[i]=BAR_NUM[i-1]*40+BaseAddr;
}
else if(i==5)
{
  BAR_PIC_ADDR[i]=95*40+BaseAddr;
}
else if(i==6)
{
  BAR_PIC_ADDR[i]=106*40+BaseAddr;
}
break;

case 2 :
if(i==0)
{
  BAR_PIC_ADDR[i]=104*40+BaseAddr;
}
else if(i==1||i==2||i=3||i==4)
{
  BAR_PIC_ADDR[i]=BAR_NUM[i-1]*40+BaseAddr;
}
else if(i==5)
{
  BAR_PIC_ADDR[i]=95*40+BaseAddr;
}
else if(i==6)
{
  BAR_PIC_ADDR[i]=106*40+BaseAddr;
}
break;


case 3 :
if(i==0)
{
  BAR_PIC_ADDR[i]=105*40+BaseAddr;
}
else if(i==1||i==2||i=3||i==4)
```

```
        {
          BAR_PIC_ADDR[i]=BAR_NUM[i-1]*40+BaseAddr;
        }
        else if(i==5)
        {
          BAR_PIC_ADDR[i]=95*40+BaseAddr;
        }
        else if(i==6)
        {
          BAR_PIC_ADDR[i]=106*40+BaseAddr;
        }
        break;

        default:
        break;
    }
     return BAR_PIC_ADDR；

}
```

注：在屏上打点时要按照国家规范进行拼凑。

4.2.24 天线调用程序

函数：DWORD* Antenna_CODE(int* NUM)
功能：获取 12x12 天线 调用地址。
参数：NUM 0123 带表天线信号强度。
返回：数据地址

```
DWORD* Antenna_CODE(int* NUM)
{
    DWORD BaseAdd= 0x47AD32;
    return(BaseAdd+NUM*24);
}
```

4.2.25 电池调用程序

函数：DWORD* Battery_CODE(int* NUM)
功能：获取 12x12 电池 调用地址。
参数：NUM 0123 带表电池电量。
返回：数据地址

```
DWORD* Battery_CODE(int* NUM)
{
    DWORD BaseAdd= 0x47ADAA
```

```
    return(BaseAdd+NUM*24);
```

4.3 内码转换程序

4.3.1 UNICODE 转 GBK 码表映射算法,仅 1&3 字符区

UNICODE 转 GBK 码表映射算法,仅 1&3 字符区
函数：WORD U2G(WORD Unicode)
功能：将 UNICODE 码转换为 GB 码,注：仅应用于 1 区&3 区。
参数：WORD Unicode,表示输入的 UNICODE 码。
返回：对应的 GB 码在字库中存放的地址。(注:读取对应地址 2 字节数据即为 unicode 对应的 GB 码)。

```
WORD UG[]={
0x3000,0x3001,0x3002,0x00b7,0x02c9,0x02c7,0x00a8,0x3003,0x3005,0x2014,
0xff5e,0x2016,0x2026,0x2018,0x2019,0x201c,0x201d,0x3014,0x3015,0x3008,
0x3009,0x300a,0x300b,0x300c,0x300d,0x300e,0x300f,0x3016,0x3017,0x3010,
0x3011,0x00b1,0x00d7,0x00f7,0x2236,0x2227,0x2228,0x2211,0x220f,0x222a,
0x2229,0x2208,0x2237,0x221a,0x22a5,0x2225,0x2220,0x2312,0x2299,0x222b,
0x222e,0x2261,0x224c,0x2248,0x223d,0x221d,0x2260,0x226e,0x226f,0x2264,
0x2265,0x221e,0x2235,0x2234,0x2642,0x2640,0x00b0,0x2032,0x2033,0x2103,
0xff04,0x00a4,0xffe0,0xffe1,0x2030,0x00a7,0x2116,0x2606,0x2605,0x25cb,
0x25cf,0x25ce,0x25c7,0x25c6,0x25a1,0x25a0,0x25b3,0x25b2,0x203b,0x2192,
0x2190,0x2191,0x2193,0x3013
};
WORD U2G(WORD Unicode)
{
    WORD i;
    if(Unicode<=0xffe5&&Unicode>=0xff01)//section   3
    {
        if(Unicode==0xff04)
        {
            return   0xa1e7;
        }
        if(Unicode==0xff5e)
        {
            return 0xa1ab;
        }
        if(Unicode==0xffe0)
        {
            return 0xa1e9;
        }
        if(Unicode==0xffe1)
        {
            return 0xa1ea;
        }
```

```
        if(Unicode==0xffe3)
        {
            return   0xa3fe;
        }
        if(Unicode==0xffe5)
        {
            return 0xa3a4;
        }
        else
        return Unicode-0xff01+0xa3a1;
    }
    if(Unicode<=0x3013)
    {
        for(i=0;i<94;i++)
        {
            if(Unicode==UG[i])
                return(0xa1a1+i);
        }
    }
}
```

4.3.2 UNICODE 转 GBK 码表映射算法（不含符号区,转换全部双字节区）

GB_Address 转换之后的 GB 码的存放地址，
函数：WORD U2G(WORD Unicode)
功能：将 UNICODE 码转换为 GB 码，
参数：WORD Unicode,表示输入的 UNICODE 码。
返回：对应的 GB 码在字库中存放的地址。
（注:读取对应地址 2 字节数据即为 unicode 对应的 GB 码）。
转码表起始地址： BaseAdd＝0x7F1E10

```
WORD U2G(WORD Unicode)
{if(unicode<=0x0451&&unicode>=0x00a0)
    {
        U_Start_Addr=0;
        Address= U_Start_Addr +(unicode-0x00a0)*2;
    }
    else if(unicode<=0x2642&&unicode>=0x2010)
    {
        U_Start_Addr =1892;
        Address= U_Start_Addr +(unicode-0x2010)*2;
    }
    else if(unicode<=0x33d5&&unicode>=0x3000)
    {
        U_Start_Addr =5066;
```

```
        Address= U_Start_Addr +(unicode-0x3000)*2;
    }
    else if(unicode<=0x9fa5&&unicode>=0x4e00)
    {
        U_Start_Addr =7030;
        Address= U_Start_Addr +(unicode-0x4e00)*2;
    }
    else if(unicode<=0xfe6b&&unicode>=0xfe30)
    {
        U_Start_Addr =48834;
        Address= U_Start_Addr +(unicode-0xfe30)*2;
    }
    else if(unicode<=0xff5e&&unicode>=0xff01)
    {
        U_Start_Addr =48954;
        Address= U_Start_Addr +(unicode-0xff01)*2;
    }
    else if(unicode<=0xffe5&&unicode>=0xffe0)
    {
        U_Start_Addr =49142;
        Address= U_Start_Addr +(unicode-0xffe0)*2;
    }
    else if(unicode<=0xFA29&&unicode>=0xF92C)
    {
        U_Start_Addr =49312;
        Address= U_Start_Addr +(unicode-0xF92C)*2;
    }
    else if(unicode<=0xE864&&unicode>=0xE816)
    {
        U_Start_Addr =49820;
        Address= U_Start_Addr +(unicode-0xE816)*2;
    }
    else if(unicode<=0x2ECA&&unicode>=0x2E81)
    {
        U_Start_Addr =49978;
        Address= U_Start_Addr +(unicode-0x2E81)*2;
    }
    else if(unicode<=0x49B7&&unicode>=0x4947)
    {
        U_Start_Addr =50126;
        Address= U_Start_Addr +(unicode-0x4947)*2;
    }
    else if(unicode<=0x4DAE&&unicode>=0x4C77)
    {
        U_Start_Addr =50352;
```

```
            Address= U_Start_Addr +(unicode-0x4C77)*2;
        }
        else if(unicode<=0x3CE0&&unicode>=0x3447)
        {
            U_Start_Addr =50976;
            Address= U_Start_Addr +(unicode-0x3447)*2;
        }
        else if(unicode<=0x478D&&unicode>=0x4056)
        {
            U_Start_Addr =55380;
            Address= U_Start_Addr +(unicode-0x4056)*2;
        }
        return Address+743980;
}
GB_Address= U2G(Unicode)+ BaseAdd;
C1=ZK_Read_1_byte(GB_Address);   //从字库中读取 GB 编码的高字节（1 个字节）
C2=ZK_Read_1_byte(GB_Address+1);   //从字库中读取 GB 编码的低字节（1 个字节）
```

ZK_Read_1_byte(unsigned long addr)函数功能为从字库的地址 addr 读取一个字节的数据。需客户自己根据实际情况编写读写函数，或者参考集通科技显示例程编写。本规格书未提供。

验证数据：啊字的 UNICODE 编码：0x554A, GB 编码为 0xB0A1,

读取后转入 GBK，或者 GBK 编码计算公式计算 UNICODE 编码对应的汉字的实际地址。

4.3.3 BIG5 转 GBK 转换算法

地址的计算由下面的函数实现（ANSI C 语言编写）

函数：unsigned long BIG5_G(unsigned short GBcode)

功能：计算 BIG5 码对应 GB 码存放的起始地址

参数：WORD UCODE 输入 BIG5 编码双字节

返回：对应 GB 码存放地址，对应 GB 码高位在前，低位在后。

起始地址：： BaseAdd＝0x7F1E10

```
unsigned long BIG5_G(unsigned short B5code)
{

    unsigned char TMP,B5_MSB,B5_LSB;
    B5_MSB= B5code >>8;
    B5_LSB= B5code &0x00ff;
    unsigned long part_addr,BaseAddr=0x    ;//转码表基地址;
    if(B5_MSB>=0xa1&&B5_MSB<=0xa3)
    {
        part_addr=0;
        TMP=0xa1;
    }

    else if(B5_MSB>=0xa4&&B5_MSB<=0xc6)
```

```
    {
        part_addr=816;
        TMP=0xa4;
     }

    else if(B5_MSB>=0xc9&&B5_MSB<=0xf9)
    {
        part_addr=11618;
        TMP=0xc9;
     }

     if(B5_LSB<=0x7e&&B5_LSB>=0x40)
     {
         return(part_addr+((B5_MSB-TMP)*157+B5_LSB-0x40)*2+BaseAddr);
     }

     else if(B5_LSB<=0xfe&&B5_LSB>=0xa1)

     {
         return(part_addr+((B5_MSB-TMP)*157+B5_LSB-0xa1+63)*2+BaseAddr);
     }
}
```

5. 自由可读写空间描述

5.1 存储组织

每设备	每块	每扇区	每页	
1M	64K	4K	256	字节
4K	256	16		页
256	16			扇区
16				块

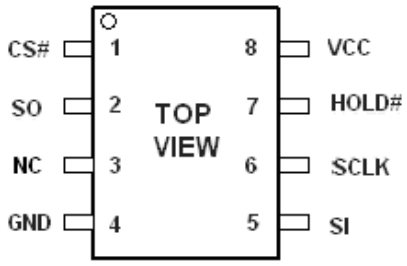
5.2 存储块、扇区结构

块	扇区	地址范围	
15	255	0x0FF000	0x0FFFFF
	.....	.....	.....
	240	0x0F0000	0x0F0FFF
14	239	0x0EF000	0x0EFFFF
	.....	.....	.....
	224	0x0E0000	0x0E0FFF
.....	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
.....	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
.....	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
2	47	0x02F000	0x02FFFF
	.....	.....	.....
	32	0x020000	0x020FFF
1	31	0x01F000	0x01FFFF
	.....	.....	.....
	16	0x010000	0x010FFF
0	15	0x00F000	0x00FFFF
	.....	.....	.....
	0	0x000000	0x000FFF

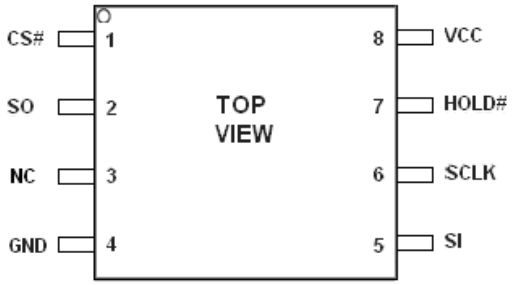
6. 引脚描述与电路连接

6.1 引脚配置

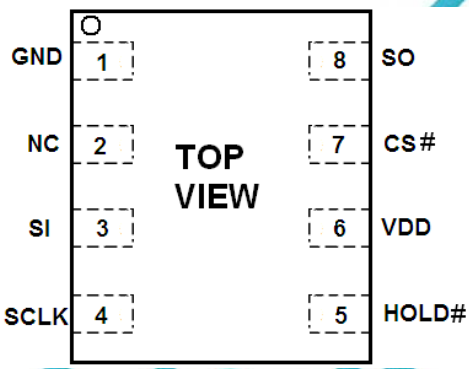
SOP8-A



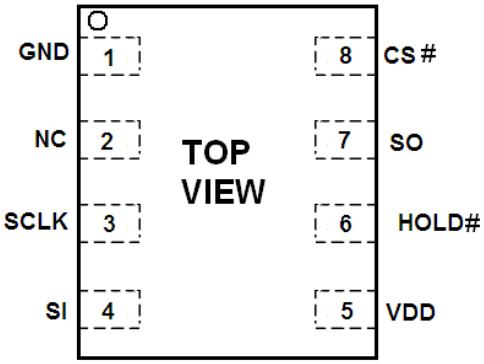
SOP8-B



DFN8-A



DFN8-B



6.2 引脚描述

SOP8-A/SOP8-B

NO.	名称	I/O	描述
1	CS#	I	片选输入（Chip enable input）
2	SO	O	串行数据输出 (Serial data output)
3	NC		悬空
4	GND		地(Ground)
5	SI	I	串行数据输入 (Serial data input)
6	SCLK	I	串行时钟输入（Serial clock input）
7	HOLD#	I	总线挂起（Hold, to pause the device without）
8	VCC		电源(+ 3.3V Power Supply)

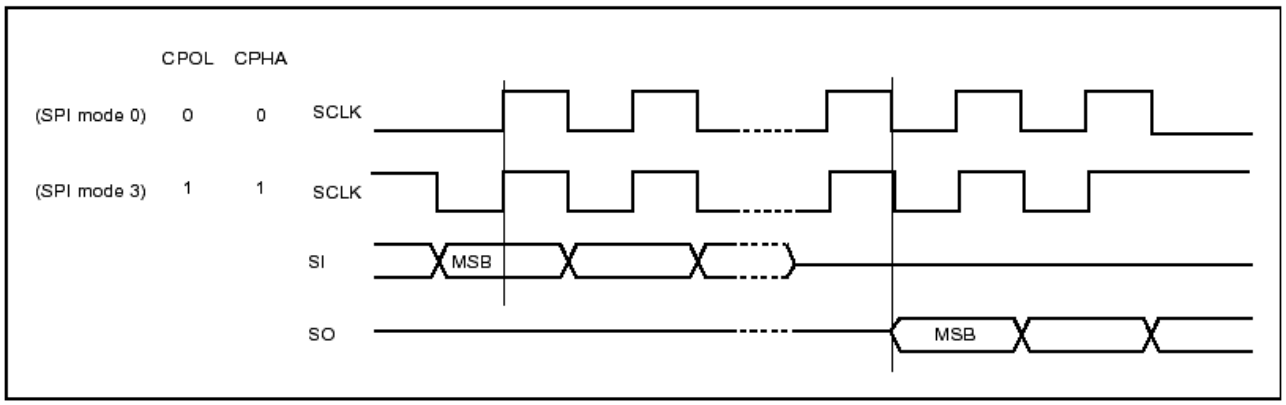
DFN8-A

NO.	名称	I/O	描述
1	GND		地(Ground)
2	NC		悬空
3	SI	I	串行数据输入 (Serial data input)
4	SCLK	I	串行时钟输入（Serial clock input）
5	HOLD#	I	总线挂起（Hold, to pause the device without）
6	VCC		电源(+ 3.3V Power Supply)
7	CS#	I	片选输入（Chip enable input）
8	SO	O	串行数据输出 (Serial data output)

DFN8-B

NO.	名称	I/O	描述
1	GND		地(Ground)
2	NC		悬空
3	SCLK	I	串行时钟输入（Serial clock input）
4	SI	I	串行数据输入 (Serial data input)
5	VCC		电源(+ 3.3V Power Supply)
6	HOLD#	I	总线挂起（Hold, to pause the device without）
7	SO	O	串行数据输出 (Serial data output)
8	CS#	I	片选输入（Chip enable input）

串行数据输出（SO）：该信号用来把数据从芯片串行输出，数据在时钟的下降沿移出。
串行数据输入（SI）：该信号用来把数据从串行输入芯片，数据在时钟的上升沿移入。
串行时钟输入（SCLK）：数据在时钟上升沿移入，在下降沿移出。
片选输入（CS#）：所有串行数据传输开始于CS#下降沿，CS#在传输期间必须保持为低电平，在两条指令之间保持为高电平。

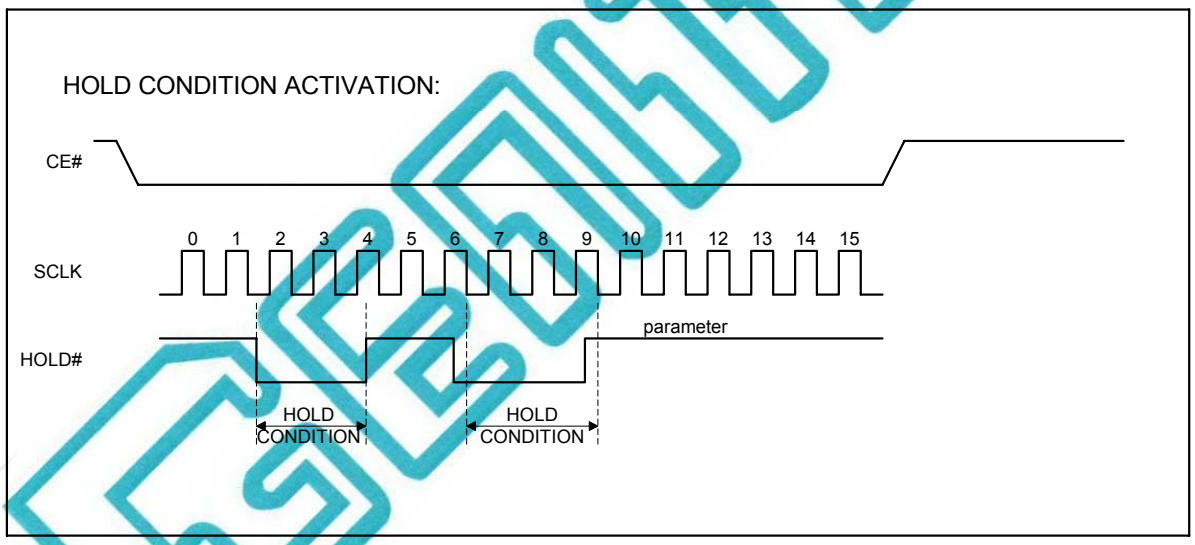


总线挂起输入（HOLD#）：

该信号用于片选信号有效期间暂停数据传输，在总线挂起期间，串行数据输出信号处于高阻态，芯片不对串行数据输入信号和串行时钟信号进行响应。

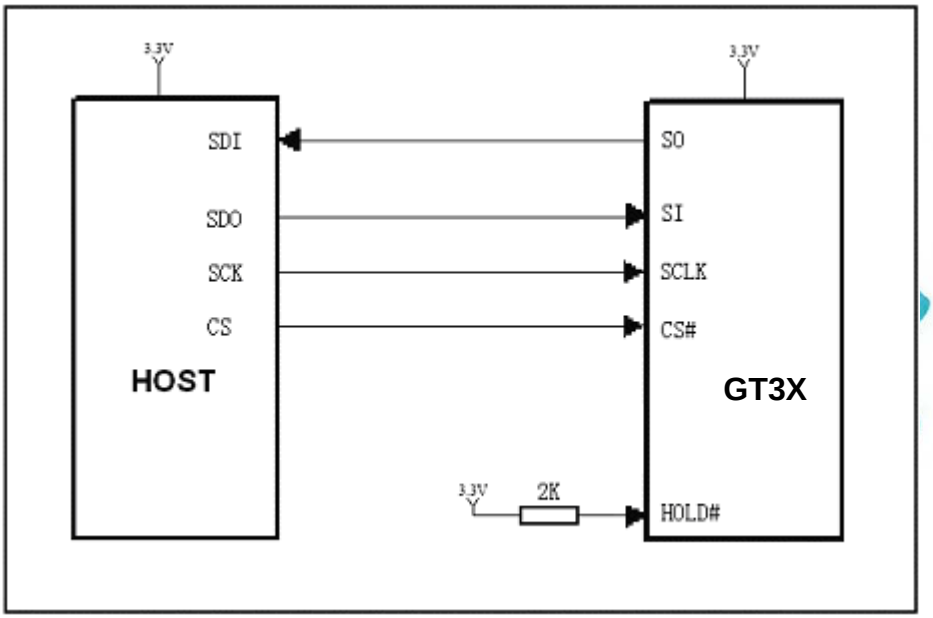
当HOLD#信号变为低并且串行时钟信号（SCLK）处于低电平时，进入总线挂起状态。

当HOLD#信号变为高并时串行时钟信号（SCLK）处于低电平时，结束总线挂起状态。



6.3 SPI 接口与主机接口参考电路示意图

SPI 与主机接口电路连接可以参考下图（#HOLD 管脚建议接 2K 电阻 3.3V 拉高）。



SPI 接口与主机接口参考电路示意图

7. 电气特性

7.1 绝对最大额定值

Symbol	Parameter	Min.	Max.	Unit	Condition
T_{OP}	Operating Temperature	-40	85	℃	
T_{STG}	Storage Temperature	-65	150	℃	
VCC	Supply Voltage	-0.3	3.6	V	
V_{IN}	Input Voltage	-0.3	VCC+0.3	V	
GND	Power Ground	-0.3	0.3	V	

7.2 DC 特性

Condition：T_{OP} =-20℃ to 70℃，GND=0V

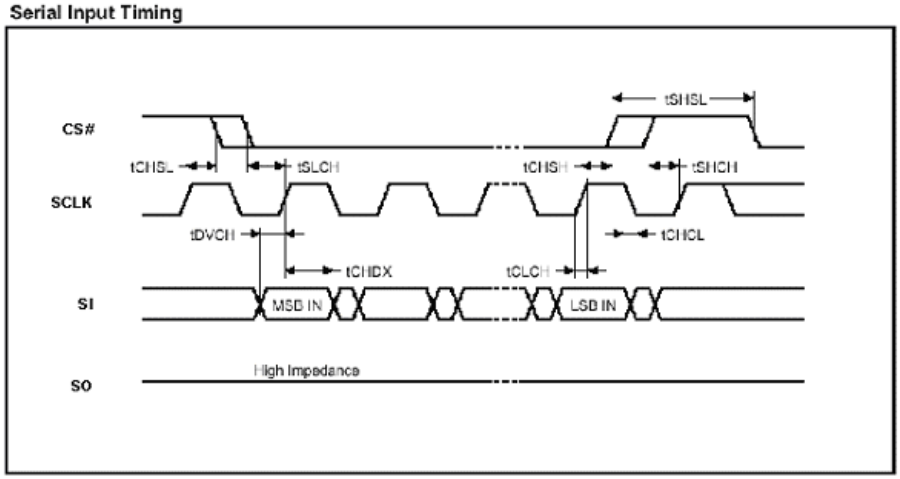
Symbol	Parameter	Min.	Max.	Unit	Condition
I_{DD}	VCC Supply Current(active)		12	mA	
I_{SB}	VCC Standby Current		10	uA	
V_{IL}	Input LOW Voltage	-0.3	0.3VCC	V	VCC=2.2~3.6V
V_{IH}	Input HIGH Voltage	0.7VCC	VCC+0.4	V	
V_{OL}	Output LOW Voltage		0.4 (I_{OL}=1.6mA)	V	
V_{OH}	Output HIGH Voltage	0.8VCC (I_{OH}=-100uA)		V	
I_{LI}	Input Leakage Current	0	2	uA	
I_{LO}	Output Leakage Current	0	2	uA	

Note：I_{IL}：Input LOW Current，I_{IH}：Input HIGH Current，
I_{OL}：Output LOW Current，I_{OH}：Output HIGH Current，

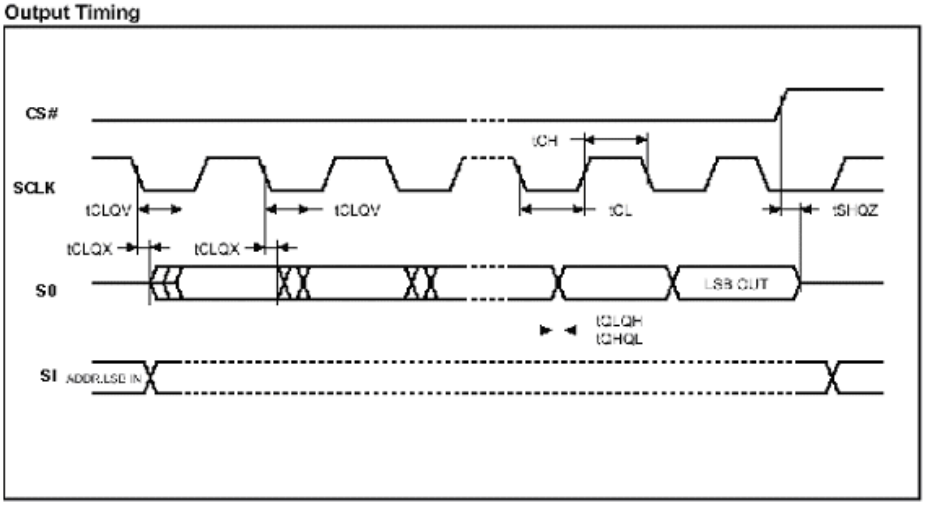
7.3 AC 特性

Symbol	Alt.	Parameter	Min.	Max.	Unit
Fc	Fc	Clock Frequency	D.C.	90	MHz
tCH	tCLH	Clock High Time	15		ns
tCL	tCLL	Clock Low Time	15		ns
tCLCH		Clock Rise Time(peak to peak)	0.1		V/ns
tCHCL		Clock Fall Time (peak to peak)	0.1		V/ns
tSLCH	tCSS	CS# Active Setup Time (relative to SCLK)	5		ns
tCHSL		CS# Not Active Hold Time (relative to SCLK)	5		ns
tDVCH	tDSU	Data In Setup Time	2		ns
tCHDX	tDH	Data In Hold Time	5		ns
t CHSH		CS# Active Hold Time (relative to SCLK)	5		ns
t SHCH		CS# Not Active Setup Time (relative to SCLK)	5		ns
t SHSL	tCSH	CS# Deselect Time	100		ns
t SHQZ	tDIS	Output Disable Time		9	ns
t CLQV	tV	Clock Low to Output Valid		9	ns
t CLQX	tHO	Output Hold Time	0		ns

Serial Input Timing



Output Timing

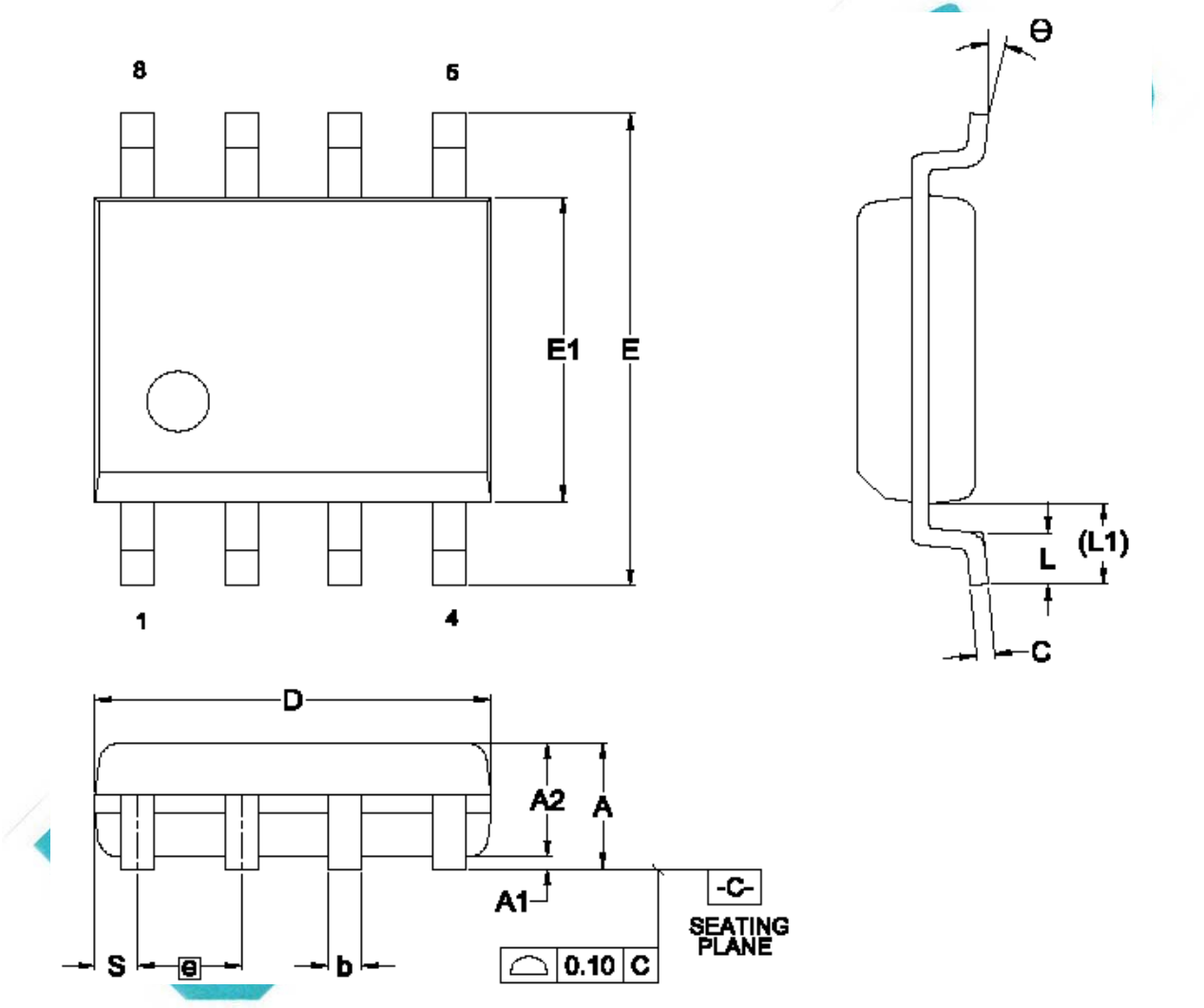


8. 封装尺寸

封 装 类 型	封装尺寸
SOP8-A	4.90mmX3.90mm（193milX154mil）
SOP8-B	5.28mmX7.90mm（206milX311mil）
DFN8-A	4.0mmx 4.0mm (158milX158mil)
DFN8-B	4.0mmx 4.0mm (158milX158mil)

Package

SOP8-A Unit :mm

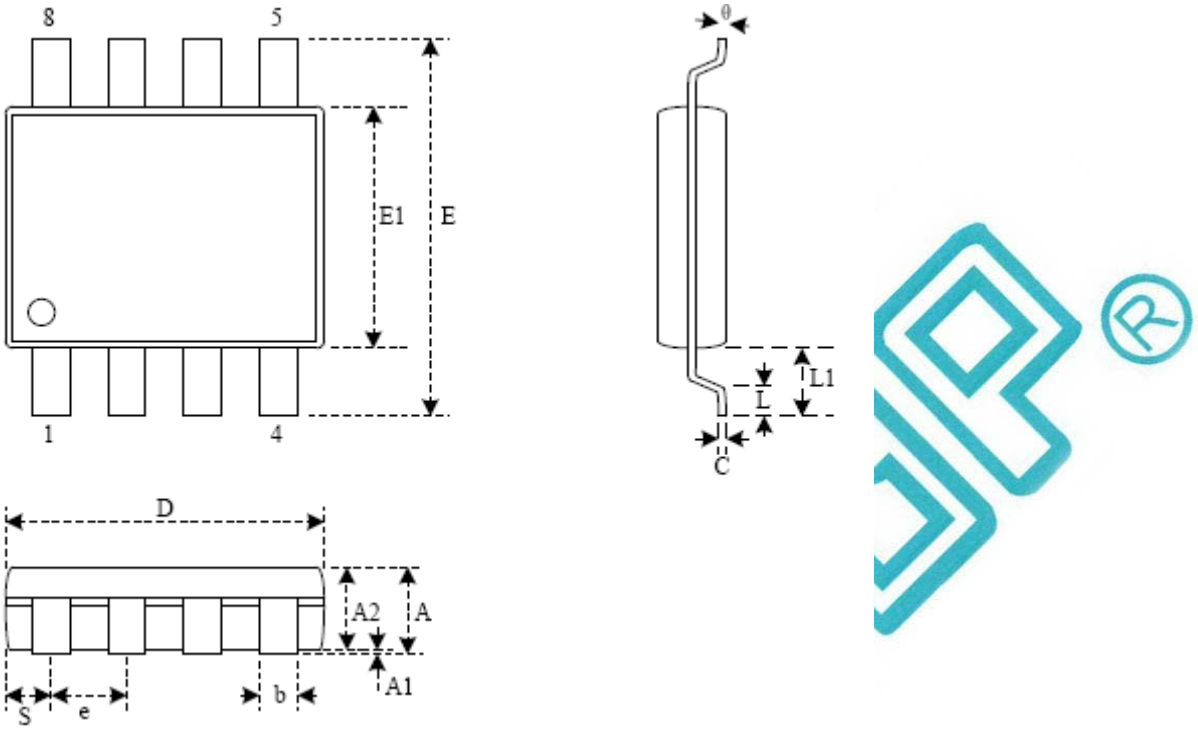


Dimensions(inch dimensions are derived from the original mm dimensions)

		A	A1	A2	b	C	D	E	E1	⊖	L	L1	S	θ
Mm	Min.	-	0.10	1.35	0.36	0.15	4.77	5.80	3.60		0.46	0.65	0.41	0
	Norm.	-	0.15	1.45	0.41	0.20	4.90	5.99	3.90	1.27	0.66	1.05	0.54	5
	Max.	1.75	0.20	1.55	0.51	0.25	5.03	6.20	4.00		0.86	1.25	0.67	8
inch	Min.	-	0.004	0.053	0.014	0.006	0.188	0.228	0.150		0.018	0.033	0.016	0
	Norm.	-	0.006	0.057	0.016	0.008	0.193	0.236	0.154	0.050	0.026	0.041	0.021	5

	Max.	0.069	0.008	0.061	0.020	0.010	0.198	0.244	0.156		0.034	0.049	0.026	8

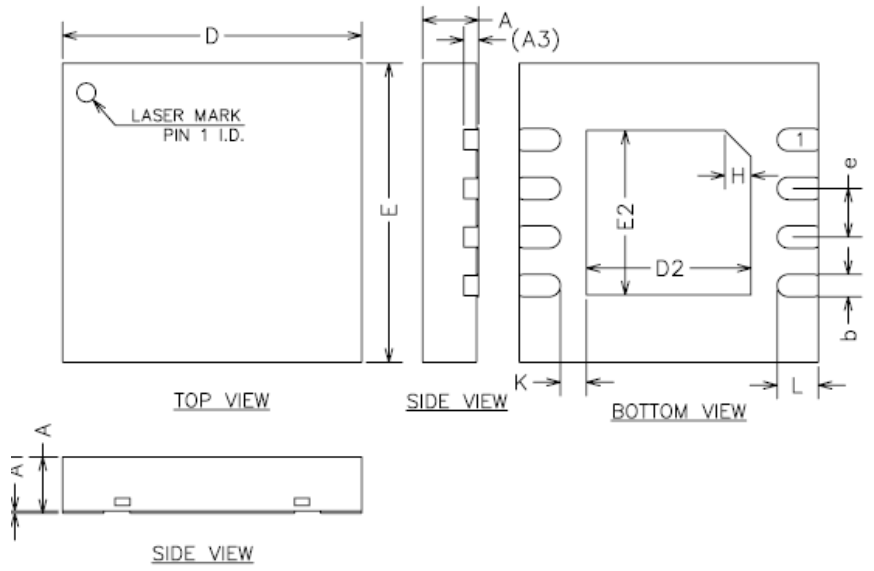
SOP8-B Unit :mm



Dimensions(inch dimensions are derived from the original mm dimensions)

		A	A1	A2	b	C	D	E	E1	⊖	L	S	θ
Mm	Min.	-	0.05	0.75	0.35	0.15	5.18	7.70	5.18		0.50	0.41	0
	Norm.	-	0.10	0.80	0.42	0.20	5.28	7.90	5.28	1.27	0.65	0.54	5
	Max.	1.0	0.15	0.85	0.48	0.25	5.38	8.10	5.38		0.80	0.67	10
inch	Min.	-	0.002	0.030	0.014	0.006	0.204	0.303	0.204		0.020	0.016	0
	Norm.	-	0.004	0.032	0.016	0.008	0.206	0.311	0.206	0.050	0.026	0.021	5
	Max.	0.04	0.006	0.034	0.020	0.010	0.210	0.319	0.210		0.031	0.026	10

DFN8-A/DFN8-B Unit :mm



COMMON DIMENSIONS
(UNITS OF MEASURE=MILLIMETER)

SYMBOL	MIN	NOM	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A3		0.20REF	
b	0.25	0.30	0.35
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.10	2.20	2.30
E2	2.10	2.20	2.30
e	0.55	0.65	0.75
H		0.35REF	
K		0.35REF	
L	0.45	0.55	0.65
R	0.13	–	–