

T6963C 液晶驱动模块资料

240*128

资料收集整理：晓奇

Keil C51 源程序：Youth

T6963 控制指令码表

指令	编码	数据 1	数据 2	功能
寄存器设置	0010 0001	X 地址	Y 地址	设置光标位置
	0010 0010	数据	00H	设置起始寄存器
	0010 0100	地址低 8 位	地址高 8 位	设置地址指针
设置控制词	0100 0000	地址低 8 位	地址高 8 位	设置文本起始地址
	0100 0001	列	00H	设置文本区宽度
	0100 0010	地址低 8 位	地址高 8 位	设置图形起始地址
	0100 0011	列	00H	设置图形区宽度
模式设定	1000 x000	-	-	逻辑“或”模式
	1000 x001	-	-	逻辑“异或”模式
	1000 x010	-	-	逻辑“与”模式
	1000 x011	-	-	文本特征模式
	1000 0xxx	-	-	内部 CG ROM 模式
	1000 1xxx	-	-	外部 CG RAM 模式
显示模式	1001 0000	-	-	显示关闭
	1001 xx10	-	-	打开光标，黑色关闭
	1001 xx11	-	-	打开光标，黑色显示
	1001 01xx	-	-	打开文本方式，关闭图形方式
	1001 10xx	-	-	关闭文本方式，打开图形方式
	1001 11xx	-	-	图形文本混合方式
光标形式	1010 0000	-	-	1 条线
	1010 0001	-	-	2 条线
	1010 0010	-	-	3 条线
	1010 0011	-	-	4 条线
	1010 0100	-	-	5 条线
	1010 0101	-	-	6 条线
	1010 0110	-	-	7 条线
	1010 0111	-	-	8 条线
数据自动读写	1011 0000	-	-	数据自动写入设定
	1011 0001	-	-	数据自动读出设定
	1011 0010	-	-	自动复位
数据读写	1100 0000	--	--	数据写入，地址自动增量
	1100 0001	--	--	数据读出，地址自动增量
	1100 0010	--	--	数据写入，地址自动减量
	1100 0011	--	--	数据读出，地址自动减量
	1100 0100	--	--	数据写入，地址保持不变
	1100 0101	--	--	数据读出，地址保持不变
屏幕读取	1110 0000	-	-	
屏幕拷贝	1110 1000	-	-	
位设置 / 复位	1111 0xxx	-	-	位复位
	1111 1xxx	-	-	位设置
	1111 x000	-	-	位 0 (最低位)
	1111 x001	-	-	位 1
	1111 x010	-	-	位 2
	1111 x011	-	-	位 3
	1111 x100	-	-	位 4
	1111 x101	-	-	位 5
	1111 x110	-	-	位 6
	1111 x111	-	-	位 7

接口信号

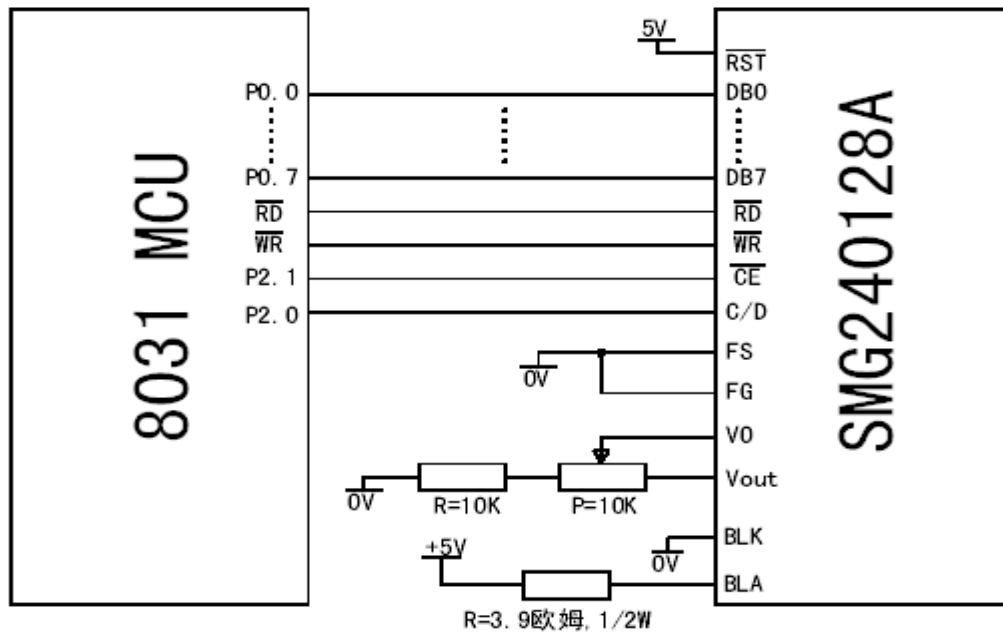
引脚号	标记号	说明	备注
1	FG	显示屏框夹外壳地	接地
2	Vss	电源地	
3	Vdd	电源+5V	
4	Vo	LCD 工作电源（对比度调节负电压输入）	
5	Wr	数据写入当 WR = L 时，将数据写入 T6963C	
6	Rd	数据读出，低电平有效	
7	CE	工作允许，当 CPU 和 T6563C 通讯时，E 必须在 L。	
8	C/D	WR = L C/D = H : 写命令 C/D = L : 读命令 RD = L C/D = H: 读状态 C/D = L : 读数据	
9	Reset	复位信号，H:正常(T6963C 有内部上拉电阻) L: 初始化 T6963C. Text 文本和图形的地址，文本和图形区域设定被保持。	
10	DB0	数据位 0	
11	DB1	数据位 1	
12	DB2	数据位 2	
13	DB3	数据位 3	
14	DB4	数据位 4	
15	DB5	数据位 5	
16	DB6	数据位 6	
17	DB7	数据位 7	
18	FS	字体选择: FS = H, 6*8 点的字体, FS = L, 8*8 点的字体	
19	Vout	DC-DC 负电源输出（液晶屏工作电压，作对比度调节）	
20	LED+	背光电源正端	
21	LED-	背光电源负端	

基本特性:

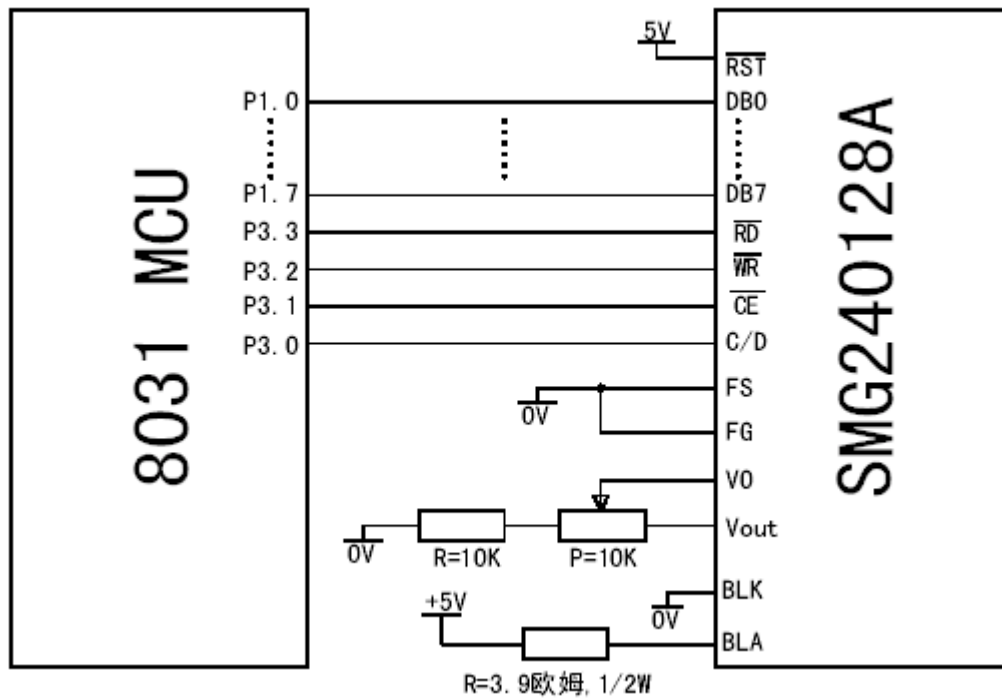
- 显示格式: 240*128 点阵
- STN 蓝灰底色
- 方便同 8 位 MPU 连接
- 低功耗
- 视角: 6 O'clock
- Multiplex level: 1/128 占空比, 1/12 偏压
- LCD 驱动电压 (上片): 20V
- LCD 驱动 IC
LCD 控制电路: T6963C
行驱动器 : T6A40
列驱动器 : T6A39
- 字体 : (pin-selectable)
水平点 : 6, 8
垂直点 : 8 (固定)
- 标准模块内置一个 128-字符的字库 ROM (代码 0101) T6963C-0101 。
- 外部显示内存: 32KB
字符区域, 图形区域, 外部字符字库的显示内存的地址由软件来决定。
- 从 CPU 中进行 读或写操作不影响显示。

定义功能只能用于文本格式, 不能被用于图形或字符混合格式。

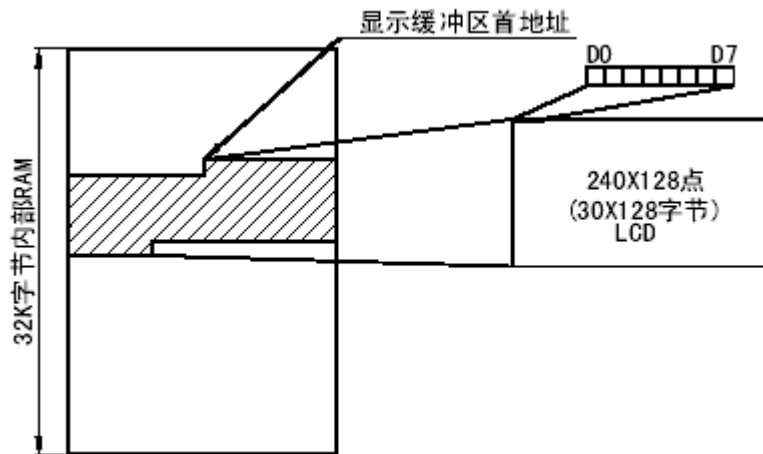
与 CPU 接口方法 1：总线方式：



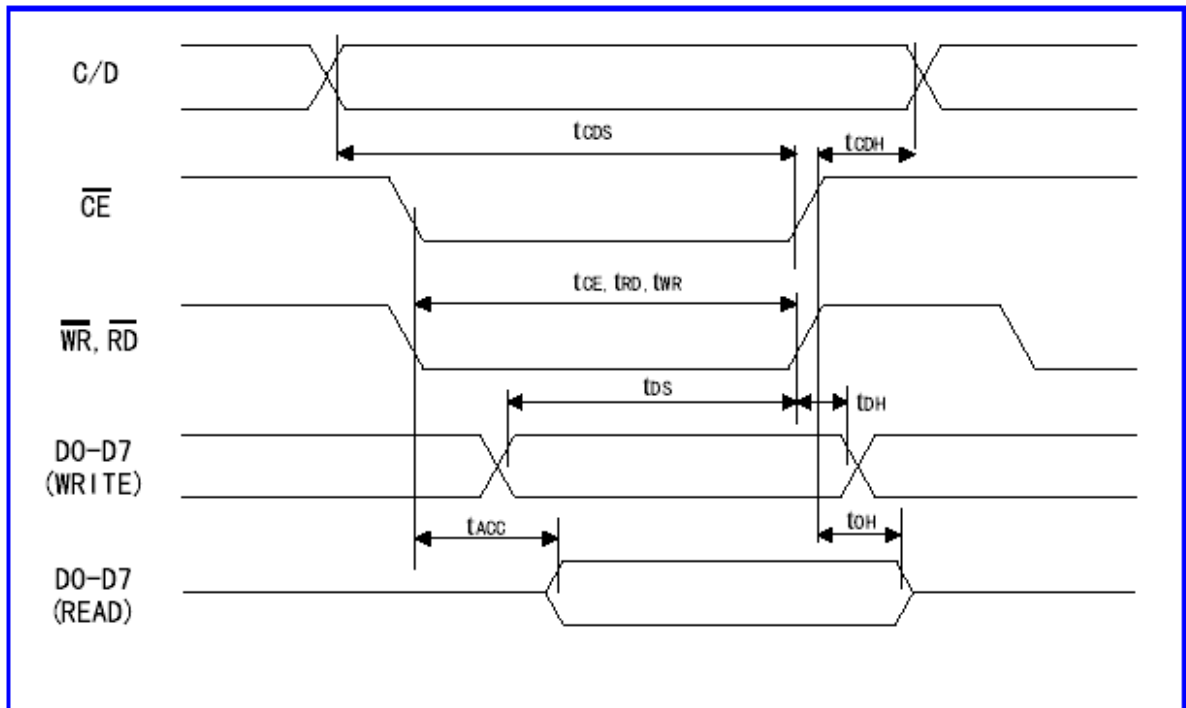
接口方式 2：模拟方式



RAM 与显示屏的关系



1. 读写操作时序



2. 时序参数

时序参数	符号	极限值			单位	测试条件
		最小值	典型值	最大值		
C/D 建立时间	t_{cDS}	100	-	-	ns	引脚 C/D
C/D 保持时间	t_{cDH}	10	-	-	ns	
片选、读、写脉冲宽度	t_{CE}, t_{RD}, t_{WR}	80	-	-	ns	-
数据建立时间(写操作)	t_{DS}	80	-	-	ns	引脚 DB0~DB7
数据保持时间(写操作)	t_{DH}	40	-	-	ns	
数据建立时间(读操作)	t_{ACC}	-	-	150	ns	
数据保持时间(读操作)	t_{OH}	10	-	50	ns	

- 6 初始化过程（复位过程）
- 6.1 写指令 80H：设置显示模式为 OR 模式。
- 6.2 写指令 98H：开显示。
- 6.3 写指令 43H：设置显示每行字节数。
- 6.4 写指令 42H：设置显示显示缓冲区首地址。

Keil C51 源程序：

```

/*****
/* LCM (MGLS-240128TA) 显示程序 */
/* MCU 型号: Winbond W78E58-24 */
/* 时钟频率: 22.1184 MHz */
/* 接口方式: 直接接口 (挂总线) */
/* 开发环境: Keil C51 V6.14 */
/* 开发日期: 2001.06.12- */
/* 程序编写: Youth */
*****/

#include <absacc.h>
#include <reg52.h>
#include <stdarg.h>
#include <stdio.h>

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

#define STX      0x02
#define ETX      0x03
#define EOT      0x04
#define ENQ      0x05
#define BS       0x08
#define CR       0x0D
#define LF       0x0A
#define DLE      0x10
#define ETB      0x17
#define SPACE    0x20
#define COMMA    0x2C

#define TRUE     1
#define FALSE    0

#define HIGH     1
#define LOW      0

// T6963C 端口定义

```

```

#define LCMDW      XBYTE[0x5000]    // 数据口
#define LCMCW      XBYTE[0x5002]    // 命令口

// T6963C 命令定义
#define LC_CUR_POS 0x21             // 光标位置设置
#define LC_CGR_POS 0x22             // CGRAM 偏置地址设置
#define LC_ADD_POS 0x24             // 地址指针位置
#define LC_TXT_STP 0x40             // 文本区首址
#define LC_TXT_WID 0x41             // 文本区宽度
#define LC_GRH_STP 0x42             // 图形区首址
#define LC_GRH_WID 0x43             // 图形区宽度
#define LC_MOD_OR  0x80             // 显示方式: 逻辑“或”
#define LC_MOD_XOR 0x81             // 显示方式: 逻辑“异或”
#define LC_MOD_AND 0x82             // 显示方式: 逻辑“与”
#define LC_MOD_TCH 0x83             // 显示方式: 文本特征
#define LC_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;
//                                     D3=1/0:图形显示启用/禁用;

#define LC_CUR_SHP 0xA0             // 光标形状选择: 0xA0-0xA7 表示光标占的行数
#define LC_AUT_WR  0xB0             // 自动写设置
#define LC_AUT_RD  0xB1             // 自动读设置
#define LC_AUT_OVR 0xB2             // 自动读/写结束
#define LC_INC_WR  0xC0             // 数据一次写, 地址加 1
#define LC_INC_RD  0xC1             // 数据一次读, 地址加 1
#define LC_DEC_WR  0xC2             // 数据一次写, 地址减 1
#define LC_DEC_RD  0xC3             // 数据一次读, 地址减 1
#define LC_NOC_WR  0xC4             // 数据一次写, 地址不变
#define LC_NOC_RD  0xC5             // 数据一次读, 地址不变
#define LC_SCN_RD  0xE0             // 屏读
#define LC_SCN_CP  0xE8             // 屏拷贝
#define LC_BIT_OP  0xF0             // 位操作: D0-D2: 定义 D0-D7 位; D3: 1 置位; 0: 清除

```

```
code uchar const uPowArr[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

```
// ASCII 字模宽度及高度定义
```

```
#define ASC_CHR_WIDTH      8
#define ASC_CHR_HEIGHT    12
```

```
// ASCII 字模, 显示为 8*16
```

```
char code ASC_MSK[96*12] = {
```

```
// Terminal9; 此字体下对应的点阵为: 宽 x 高=8x12
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,           // < 0x20 时,打印此字
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, // ''
```

```
0x00,0x0C,0x1E,0x1E,0x1E,0x0C,0x0C,0x00,0x0C,0x0C,0x00,0x00, // '!'
```

```
0x00,0x66,0x66,0x66,0x24,0x00,0x00,0x00,0x00,0x00,0x00, // ""
```

```
0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00, // '#'
```

0x0C,0x0C,0x3E,0x03,0x03,0x1E,0x30,0x30,0x1F,0x0C,0x0C,0x00, // '\$'
0x00,0x00,0x00,0x23,0x33,0x18,0x0C,0x06,0x33,0x31,0x00,0x00, // '%'
0x00,0x0E,0x1B,0x1B,0x0E,0x5F,0x7B,0x33,0x3B,0x6E,0x00,0x00, // '&'
0x00,0x0C,0x0C,0x0C,0x06,0x00,0x00,0x00,0x00,0x00,0x00, // "'
0x00,0x30,0x18,0x0C,0x06,0x06,0x06,0x0C,0x18,0x30,0x00,0x00, // '('
0x00,0x06,0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x06,0x00,0x00, // ')''
0x00,0x00,0x00,0x66,0x3C,0xFF,0x3C,0x66,0x00,0x00,0x00,0x00, // '*'
0x00,0x00,0x00,0x18,0x18,0x7E,0x18,0x18,0x00,0x00,0x00,0x00, // '+'
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x1C,0x06,0x00, // ','
0x00,0x00,0x00,0x00,0x00,0x7F,0x00,0x00,0x00,0x00,0x00, // '-'
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x1C,0x00,0x00, // '.'
0x00,0x00,0x40,0x60,0x30,0x18,0x0C,0x06,0x03,0x01,0x00,0x00, // '/'
0x00,0x3E,0x63,0x73,0x7B,0x6B,0x6F,0x67,0x63,0x3E,0x00,0x00, // '0'
0x00,0x08,0x0C,0x0F,0x0C,0x0C,0x0C,0x0C,0x0C,0x3F,0x00,0x00, // '1'
0x00,0x1E,0x33,0x33,0x30,0x18,0x0C,0x06,0x33,0x3F,0x00,0x00, // '2'
0x00,0x1E,0x33,0x30,0x30,0x1C,0x30,0x30,0x33,0x1E,0x00,0x00, // '3'
0x00,0x30,0x38,0x3C,0x36,0x33,0x7F,0x30,0x30,0x78,0x00,0x00, // '4'
0x00,0x3F,0x03,0x03,0x03,0x1F,0x30,0x30,0x33,0x1E,0x00,0x00, // '5'
0x00,0x1C,0x06,0x03,0x03,0x1F,0x33,0x33,0x33,0x1E,0x00,0x00, // '6'
0x00,0x7F,0x63,0x63,0x60,0x30,0x18,0x0C,0x0C,0x0C,0x00,0x00, // '7'
0x00,0x1E,0x33,0x33,0x37,0x1E,0x3B,0x33,0x33,0x1E,0x00,0x00, // '8'
0x00,0x1E,0x33,0x33,0x33,0x3E,0x18,0x18,0x0C,0x0E,0x00,0x00, // '9'
0x00,0x00,0x00,0x1C,0x1C,0x00,0x00,0x1C,0x1C,0x00,0x00,0x00, // ':'
0x00,0x00,0x00,0x1C,0x1C,0x00,0x00,0x1C,0x1C,0x18,0x0C,0x00, // ';'
0x00,0x30,0x18,0x0C,0x06,0x03,0x06,0x0C,0x18,0x30,0x00,0x00, // '<'
0x00,0x00,0x00,0x00,0x7E,0x00,0x7E,0x00,0x00,0x00,0x00, // '='
0x00,0x06,0x0C,0x18,0x30,0x60,0x30,0x18,0x0C,0x06,0x00,0x00, // '>'
0x00,0x1E,0x33,0x30,0x18,0x0C,0x0C,0x00,0x0C,0x0C,0x00,0x00, // '?'
0x00,0x3E,0x63,0x63,0x7B,0x7B,0x7B,0x03,0x03,0x3E,0x00,0x00, // '@'
0x00,0x0C,0x1E,0x33,0x33,0x33,0x3F,0x33,0x33,0x33,0x00,0x00, // 'A'
0x00,0x3F,0x66,0x66,0x66,0x3E,0x66,0x66,0x66,0x3F,0x00,0x00, // 'B'
0x00,0x3C,0x66,0x63,0x03,0x03,0x03,0x63,0x66,0x3C,0x00,0x00, // 'C'
0x00,0x1F,0x36,0x66,0x66,0x66,0x66,0x66,0x36,0x1F,0x00,0x00, // 'D'
0x00,0x7F,0x46,0x06,0x26,0x3E,0x26,0x06,0x46,0x7F,0x00,0x00, // 'E'
0x00,0x7F,0x66,0x46,0x26,0x3E,0x26,0x06,0x06,0x0F,0x00,0x00, // 'F'
0x00,0x3C,0x66,0x63,0x03,0x03,0x73,0x63,0x66,0x7C,0x00,0x00, // 'G'
0x00,0x33,0x33,0x33,0x33,0x3F,0x33,0x33,0x33,0x33,0x00,0x00, // 'H'
0x00,0x1E,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00, // 'I'
0x00,0x78,0x30,0x30,0x30,0x30,0x33,0x33,0x33,0x1E,0x00,0x00, // 'J'
0x00,0x67,0x66,0x36,0x36,0x1E,0x36,0x36,0x66,0x67,0x00,0x00, // 'K'
0x00,0x0F,0x06,0x06,0x06,0x06,0x46,0x66,0x66,0x7F,0x00,0x00, // 'L'
0x00,0x63,0x77,0x7F,0x7F,0x6B,0x63,0x63,0x63,0x63,0x00,0x00, // 'M'
0x00,0x63,0x63,0x67,0x6F,0x7F,0x7B,0x73,0x63,0x63,0x00,0x00, // 'N'
0x00,0x1C,0x36,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x00,0x00, // 'O'
0x00,0x3F,0x66,0x66,0x66,0x3E,0x06,0x06,0x06,0x0F,0x00,0x00, // 'P'
0x00,0x1C,0x36,0x63,0x63,0x63,0x73,0x7B,0x3E,0x30,0x78,0x00, // 'Q'

```

0x00,0x3F,0x66,0x66,0x66,0x3E,0x36,0x66,0x66,0x67,0x00,0x00, // 'R'
0x00,0x1E,0x33,0x33,0x03,0x0E,0x18,0x33,0x33,0x1E,0x00,0x00, // 'S'
0x00,0x3F,0x2D,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00, // 'T'
0x00,0x33,0x33,0x33,0x33,0x33,0x33,0x33,0x33,0x1E,0x00,0x00, // 'U'
0x00,0x33,0x33,0x33,0x33,0x33,0x33,0x33,0x1E,0x0C,0x00,0x00, // 'V'
0x00,0x63,0x63,0x63,0x63,0x6B,0x6B,0x36,0x36,0x36,0x00,0x00, // 'W'
0x00,0x33,0x33,0x33,0x1E,0x0C,0x1E,0x33,0x33,0x33,0x00,0x00, // 'X'
0x00,0x33,0x33,0x33,0x33,0x1E,0x0C,0x0C,0x0C,0x1E,0x00,0x00, // 'Y'
0x00,0x7F,0x73,0x19,0x18,0x0C,0x06,0x46,0x63,0x7F,0x00,0x00, // 'Z'
0x00,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00, // '['
0x00,0x00,0x01,0x03,0x06,0x0C,0x18,0x30,0x60,0x40,0x00,0x00, // '\'
0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x00, // ']'
0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00, // '^'
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00, // '_'
0x0C,0x0C,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, // ``
0x00,0x00,0x00,0x00,0x1E,0x30,0x3E,0x33,0x33,0x6E,0x00,0x00, // 'a'
0x00,0x07,0x06,0x06,0x3E,0x66,0x66,0x66,0x66,0x3B,0x00,0x00, // 'b'
0x00,0x00,0x00,0x00,0x1E,0x33,0x03,0x03,0x33,0x1E,0x00,0x00, // 'c'
0x00,0x38,0x30,0x30,0x3E,0x33,0x33,0x33,0x33,0x6E,0x00,0x00, // 'd'
0x00,0x00,0x00,0x00,0x1E,0x33,0x3F,0x03,0x33,0x1E,0x00,0x00, // 'e'
0x00,0x1C,0x36,0x06,0x06,0x1F,0x06,0x06,0x06,0x0F,0x00,0x00, // 'f'
0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x3E,0x30,0x33,0x1E, // 'g'
0x00,0x07,0x06,0x06,0x36,0x6E,0x66,0x66,0x66,0x67,0x00,0x00, // 'h'
0x00,0x18,0x18,0x00,0x1E,0x18,0x18,0x18,0x18,0x7E,0x00,0x00, // 'i'
0x00,0x30,0x30,0x00,0x3C,0x30,0x30,0x30,0x30,0x33,0x33,0x1E, // 'j'
0x00,0x07,0x06,0x06,0x66,0x36,0x1E,0x36,0x66,0x67,0x00,0x00, // 'k'
0x00,0x1E,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x7E,0x00,0x00, // 'l'
0x00,0x00,0x00,0x00,0x3F,0x6B,0x6B,0x6B,0x6B,0x63,0x00,0x00, // 'm'
0x00,0x00,0x00,0x00,0x1F,0x33,0x33,0x33,0x33,0x33,0x00,0x00, // 'n'
0x00,0x00,0x00,0x00,0x1E,0x33,0x33,0x33,0x33,0x1E,0x00,0x00, // 'o'
0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x0F, // 'p'
0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x3E,0x30,0x78, // 'q'
0x00,0x00,0x00,0x00,0x37,0x76,0x6E,0x06,0x06,0x0F,0x00,0x00, // 'r'
0x00,0x00,0x00,0x00,0x1E,0x33,0x06,0x18,0x33,0x1E,0x00,0x00, // 's'
0x00,0x00,0x04,0x06,0x3F,0x06,0x06,0x06,0x36,0x1C,0x00,0x00, // 't'
0x00,0x00,0x00,0x00,0x33,0x33,0x33,0x33,0x33,0x6E,0x00,0x00, // 'u'
0x00,0x00,0x00,0x00,0x33,0x33,0x33,0x33,0x1E,0x0C,0x00,0x00, // 'v'
0x00,0x00,0x00,0x00,0x63,0x63,0x6B,0x6B,0x36,0x36,0x00,0x00, // 'w'
0x00,0x00,0x00,0x00,0x63,0x36,0x1C,0x1C,0x36,0x63,0x00,0x00, // 'x'
0x00,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x3C,0x30,0x18,0x0F, // 'y'
0x00,0x00,0x00,0x00,0x3F,0x31,0x18,0x06,0x23,0x3F,0x00,0x00, // 'z'
0x00,0x38,0x0C,0x0C,0x06,0x03,0x06,0x0C,0x0C,0x38,0x00,0x00, // '{'
0x00,0x18,0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x18,0x00,0x00, // '|'
0x00,0x07,0x0C,0x0C,0x18,0x30,0x18,0x0C,0x0C,0x07,0x00,0x00, // '}'
0x00,0xCE,0x5B,0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x00, // '~'
};

```

```

typedef struct typFNT_GB16 // 汉字字模显示数据结构
{
    char Index[2];
    char Msk[32];
};

struct typFNT_GB16 xdata GB_16[] = { // 显示为 16*16
"中",0x01,0x00,0x01,0x00,0x21,0x08,0x3F,0xFC,0x21,0x08,0x21,0x08,0x21,0x08,0x21,0x08,
    0x21,0x08,0x3F,0xF8,0x21,0x08,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,
"文",0x02,0x00,0x01,0x00,0x01,0x00,0xFF,0xFE,0x08,0x20,0x08,0x20,0x08,0x20,0x04,0x40,
    0x04,0x40,0x02,0x80,0x01,0x00,0x02,0x80,0x04,0x60,0x18,0x1E,0xE0,0x08,0x00,0x00,
"测",0x40,0x02,0x27,0xC2,0x24,0x42,0x84,0x52,0x45,0x52,0x55,0x52,0x15,0x52,0x25,0x52,
    0x25,0x52,0x25,0x52,0xC5,0x52,0x41,0x02,0x42,0x82,0x42,0x42,0x44,0x4A,0x48,0x04,
"试",0x00,0x20,0x40,0x28,0x20,0x24,0x30,0x24,0x27,0xFE,0x00,0x20,0xE0,0x20,0x27,0xE0,
    0x21,0x20,0x21,0x10,0x21,0x10,0x21,0x0A,0x29,0xCA,0x36,0x06,0x20,0x02,0x00,0x00,
};

uchar gCurRow,gCurCol; // 当前行、列存储，行高 16 点，列宽 8 点

uchar fnGetRow(void)
{
    return gCurRow;
}

uchar fnGetCol(void)
{
    return gCurCol;
}

uchar fnST01(void) // 状态位 STA1,STA0 判断（读写指令和读写数据）
{
    uchar i;

    for(i=10;i>0;i--)
    {
        if((LCMCW & 0x03) == 0x03)
            break;
    }
    return i; // 若返回零，说明错误
}

uchar fnST2(void) // 状态位 ST2 判断（数据自动读状态）
{
    uchar i;

    for(i=10;i>0;i--)

```

```

    {
        if((LCMCW & 0x04) == 0x04)
            break;
    }
    return i;          // 若返回零，说明错误
}

uchar fnST3(void)    // 状态位 ST3 判断（数据自动写状态）
{
    uchar i;

    for(i=10;i>0;i--)
    {
        if((LCMCW & 0x08) == 0x08)
            break;
    }
    return i;        // 若返回零，说明错误
}

uchar fnST6(void)    // 状态位 ST6 判断（屏读/屏拷贝状态）
{
    uchar i;

    for(i=10;i>0;i--)
    {
        if((LCMCW & 0x40) == 0x40)
            break;
    }
    return i;        // 若返回零，说明错误
}

uchar fnPR1(uchar uCmd,uchar uPar1,uchar uPar2) // 写双参数的指令
{
    if(fnST01() == 0)
        return 1;
    LCMDW = uPar1;
    if(fnST01() == 0)
        return 2;
    LCMDW = uPar2;
    if(fnST01() == 0)
        return 3;
    LCMCW = uCmd;
    return 0;        // 返回 0 成功
}

uchar fnPR11(uchar uCmd,uchar uPar1) // 写单参数的指令

```

```

{
    if(fnST01() == 0)
        return 1;
    LCMDW = uPar1;
    if(fnST01() == 0)
        return 2;
    LCMCW = uCmd;
    return 0;                // 返回 0 成功
}

uchar fnPR12(uchar uCmd)    // 写无参数的指令
{
    if(fnST01() == 0)
        return 1;
    LCMCW = uCmd;
    return 0;                // 返回 0 成功
}

uchar fnPR13(uchar uData)  // 写数据
{
    if(fnST3() == 0)
        return 1;
    LCMDW = uData;
    return 0;                // 返回 0 成功
}

uchar fnPR2(void)          // 读数据
{
    if(fnST01() == 0)
        return 1;
    return LCMDW;
}

// 设置当前地址
void fnSetPos(uchar urow, uchar ucol)
{
    uint  iPos;

    iPos = urow * 30 + ucol;
    fnPR1(LC_ADD_POS,iPos & 0xFF,iPos / 256);
    gCurRow = urow;
    gCurCol = ucol;
}

// 设置当前显示行、列
void cursor(uchar uRow, uchar uCol)

```

```

{
    fnSetPos(uRow * 16, uCol);
}

// 清屏
void cls(void)
{
    uint i;

    fnPR1(LC_ADD_POS,0x00,0x00); // 置地址指针
    fnPR12(LC_AUT_WR); // 自动写
    for(i=0;i<240*30;i++)
    {
        fnST3();
        fnPR13(0x00); // 写数据
    }
    fnPR12(LC_AUT_OVR); // 自动写结束
    fnPR1(LC_ADD_POS,0x00,0x00); // 重置地址指针
    gCurRow = 0; // 置地址指针存储变量
    gCurCol = 0;
}

// LCM 初始化
char fnLCMInit(void)
{
    if(fnPR1(LC_TXT_STP,0x00,0x00) != 0) // 文本显示区首地址
        return -1;
    fnPR1(LC_TXT_WID,0x1E,0x00); // 文本显示区宽度
    fnPR1(LC_GRH_STP,0x00,0x00); // 图形显示区首地址
    fnPR1(LC_GRH_WID,0x1E,0x00); // 图形显示区宽度
    fnPR12(LC_CUR_SHP | 0x01); // 光标形状
    fnPR12(LC_MOD_OR); // 显示方式设置
    fnPR12(LC_DIS_SW | 0x08); // 显示开关设置

    return 0;
}

// ASCII(8*16) 及 汉字(16*16) 显示函数
uchar dprintf(char *fmt, ...)
{
    va_list arg_ptr;
    char c1,c2,cData;
    char tmpBuf[64]; // LCD 显示数据缓冲区
    uchar i=0,j,uLen,uRow,uCol;
    uint k;

```

```

va_start(arg_ptr, fmt);
uLen = (uchar)vsprintf(tmpBuf, fmt, arg_ptr);
va_end(arg_ptr);

while(i<uLen)
{
    c1 = tmpBuf[i];
    c2 = tmpBuf[i+1];
    uRow = fnGetRow();
    uCol = fnGetCol();
    if(c1 >= 0)
    { // ASCII
        if(c1 < 0x20)
        {
            switch(c1)
            {
                case CR:
                case LF: // 回车或换行
                    i++;
                    if(uRow < 112)
                        fnSetPos(uRow+16,0);
                    else
                        fnSetPos(0,0);
                    continue;
                case BS: // 退格
                    if(uCol > 0)
                        uCol--;
                    fnSetPos(uRow,uCol);
                    cData = 0x00;
                    break;
                default: // 其他
                    c1 = 0x1f;
            }
        }
        for(j=0;j<16;j++)
        {
            fnPR12(LC_AUT_WR); // 写数据
            if(c1 >= 0x1f)
            {
                if(j < (16-ASC_CHR_HEIGHT))
                    fnPR13(0x00);
                else
                    fnPR13(ASC_MSK[(c1-0x1f)*ASC_CHR_HEIGHT+j-(16-
ASC_CHR_HEIGHT)]);
            }
            else

```

```

        fnPR13(cData);
        fnPR12(LC_AUT_OVR);
        fnSetPos(uRow+j+1,uCol);
    }
    if(c1 != BS)                // 非退格
        uCol++;
}
else
{
    // 中文
    for(j=0;j<sizeof(GB_16)/sizeof(GB_16[0]);j++)
    {
        if(c1 == GB_16[j].Index[0] && c2 == GB_16[j].Index[1])
            break;
    }
    for(k=0;k<sizeof(GB_16[0].Msk)/2;k++)
    {
        fnSetPos(uRow+k,uCol);
        fnPR12(LC_AUT_WR);        // 写数据
        if(j < sizeof(GB_16)/sizeof(GB_16[0]))
        {
            fnPR13(GB_16[j].Msk[k*2]);
            fnPR13(GB_16[j].Msk[k*2+1]);
        }
        else                    // 未找到该字
        {
            if(k < sizeof(GB_16[0].Msk)/4)
            {
                fnPR13(0x00);
                fnPR13(0x00);
            }
            else
            {
                fnPR13(0xff);
                fnPR13(0xff);
            }
        }
        fnPR12(LC_AUT_OVR);
    }
    uCol += 2;
    i++;
}
if(uCol >= 30)                // 光标后移
{
    uRow += 16;
    if(uRow < 0x80)
        uCol -= 30;
}

```

```

        else
        {
            uRow = 0;
            uCol = 0;
        }
    }
    fnSetPos(uRow,uCol);
    i++;
}
return uLen;
}

void main(void) // 测试用
{
    fnLCMInit();
    cls();
    cursor(0,0);
    dprintf("%s","This is a test:中文测试");
}

```

编后说明:

不同生产厂所生产的模块在引脚排列上会有所不同，注意区别。但是 T6963 控制芯片的编程方法是一致的，所以这里所提供的程序可以作为大家的参考程序。多谢 Youth 贡献演示程序，这也是一个 C51 规范的编程实例，在编程规范和编程技巧方面都是我们学习的典范。

晓奇(xiaoqi)