模组用户手册

我

SPL10A2 LCD&KEY 模组

使用说明书

V1.0 – 2004.9.13

凌阳大学计划推广中心
北京海淀上地信息产业基地中黎科技园 1 号楼 6 层
TEL: 86-10-62981668 FAX: 86-10-62985972 E-mail:unsp@sunplus.com.cn http://www:unsp.com.cn

版权声明

目　　录

1 前言

SPL10A-221 为 SUNPLUS 公司开发的 8 位单片机，其内置 32Seg*4Com LCD 驱动、4*3 键盘扫描、一个 Real Time Clock(RTC)、一个 1KHz/2KHz/4KHz 可选蜂鸣器驱动。勿需单独编程，仅通过数据接口，以串行通讯方式，可方便与主 CPU 实现数据交换。

基于 SPL10A-221 设计的 LCD&KEY 模组，集成了必要的 LCD、KEY、RTC、BEEP 功能，这些功能是任何一款仪器仪表的必备功能，因此，以 SPL10A-221 LCD&KEY 模组做为仪器仪表的配套表头，将有效减少主 CPU 的外扩需求。

2 系统简介

2.1 基本特性与参数指标

模块集成 LCD、KEY、RTC、BEEP 功能，可做各种仪器仪表的配套表头，有效减少了主 CPU 的外扩需求。

模块采用 DC3.3V 供电。

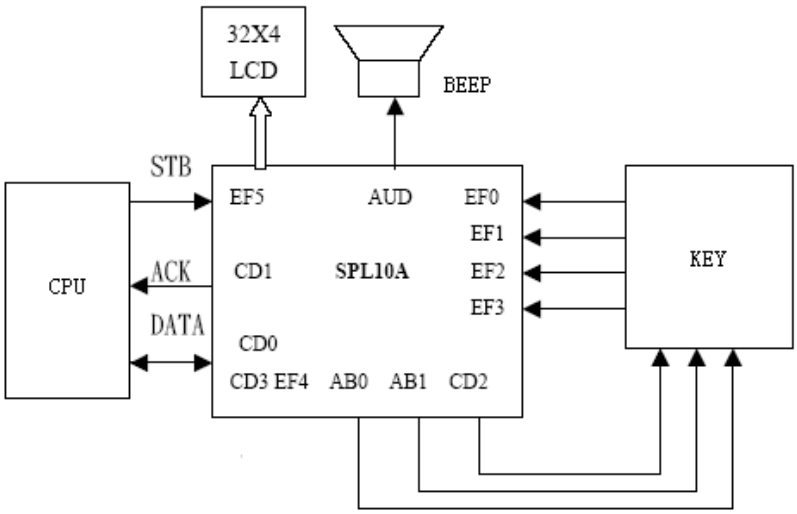
2.2 主要功能

模块使用的 LCD 最多可显示 99999.999 位数据，同时可显示“错误、常数、告警、异常、累计、余量、购量”提示、电池欠压提示、阀门开启提示、余量/增量条形提示，外接 8 个按键，一个复位键，预留电源、数据、蜂鸣器接口，可方便与任何一款 CPU 进行软硬件接口设计。

LCD 显示界面如下所示：

模块提供与 SPCE061A 的接口驱动源代码，软件接口简单。用户只需修改显示缓冲区内相关的内容，然后调用 LCD 刷新程序即可。RTC、KEY、BEEP 操作均由相关程序调用完成。

2.3 结构框图



2.4 系统环境

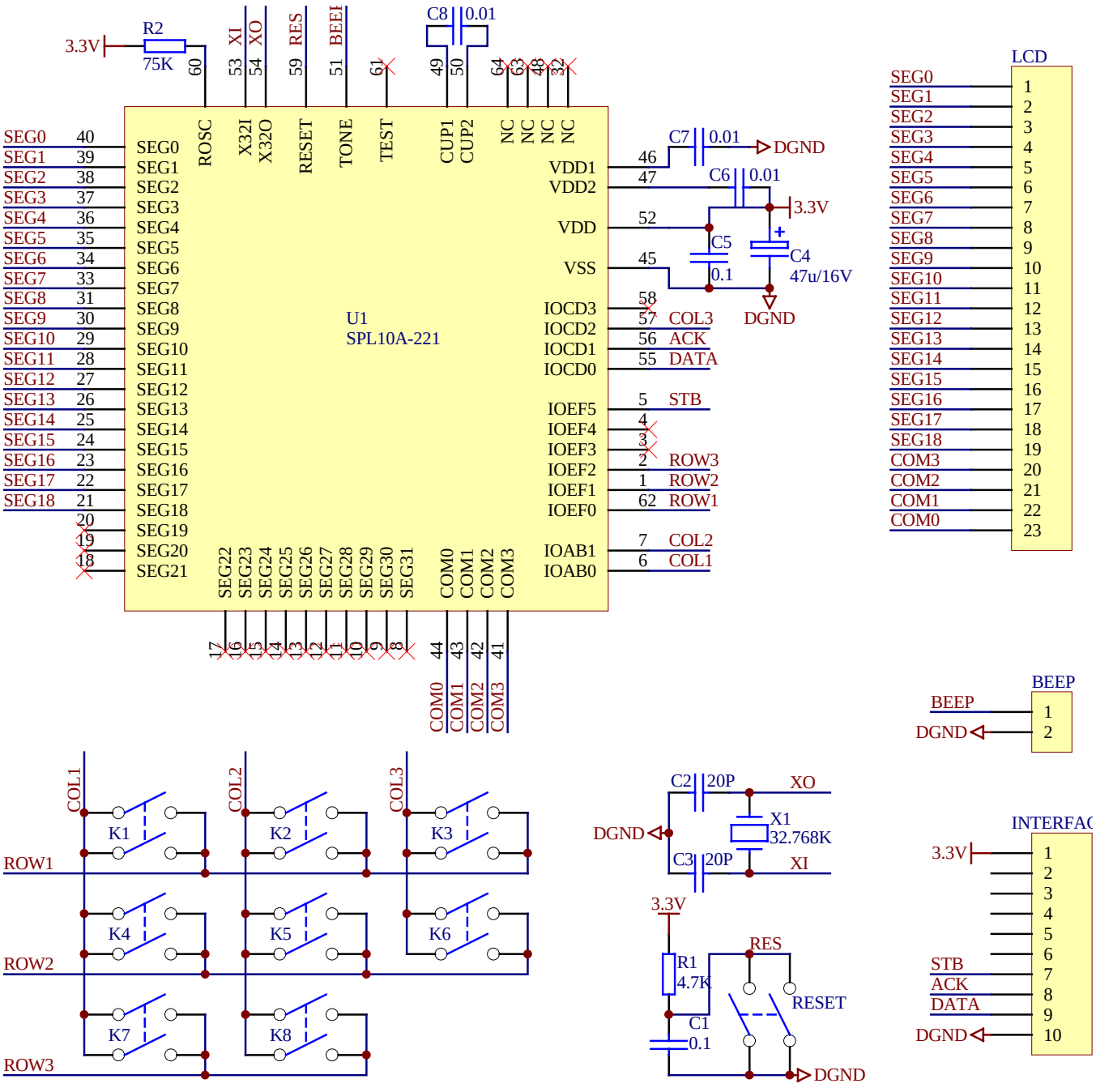
模块供电电压 DC3.3V。

2.5 注意事项

模块可与任何一款 CPU 通讯使用，应注意通讯时序。

3 硬件说明

3.1 电路原理图



3.2 主要元器件

SPL10A-221 功能概述

1．SPL10A-221 工作电压 2.4V~5.5V，最高工作频率 2MHz。

2．SPL10A-221 可直接驱动 32SEG*4COM LCD，主 CPU 可设 LCD 特性为 1/3Bias，1/4Duty。SPL10A-221 内部

有 16 字节的显示 RAM，由主 CPU 控制写入。显示 RAM 地址定义参见 LCD 地址。

LCD 地址表：

Address	Segment	Com0	Com1	Com2	Com3
0	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
1	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
2	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
3	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
4	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
5	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
6	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
7	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
8	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
9	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
10	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
11	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
12	0	D0	D1	D2	D3

	1	D4	D5	D6	D7
13	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
14	0	D0	D1	D2	D3
	1	D4	D5	D6	D7
15	0	D0	D1	D2	D3
	1	D4	D5	D6	D7

3．SPL10A-221 可外接最大 4*3 阵列键盘，键值参见键值表。

键值表

	EF0	EF1	EF2	EF3
IOAB0	1	5	9	13
IOAB1	2	6	10	14
IOCD2	3	7	11	15

4．SPL10A-221 可直接驱动蜂鸣器(Beeper)，并且有三种频率可选择，分别为 1KHz、2KHz、4KHz。蜂鸣器驱动需先设定输出频率(using Set Tone 1K, Set Tone 2K or Set Tone 4K command)，然后激活(Using Tone On command)，欲停止蜂鸣器则使用 Tone Off Command。蜂鸣器动作期间不可再输出其它的命令，否则会干扰到系统实时时钟，造成输出频率改变。

5．SPL10A-221 有内建的 Real Time Clock (RTC)的功能，可以提供实时时钟信息。Real Time Clock 功能需设定起始时间，否则系统会依据初始的设定开始计时。

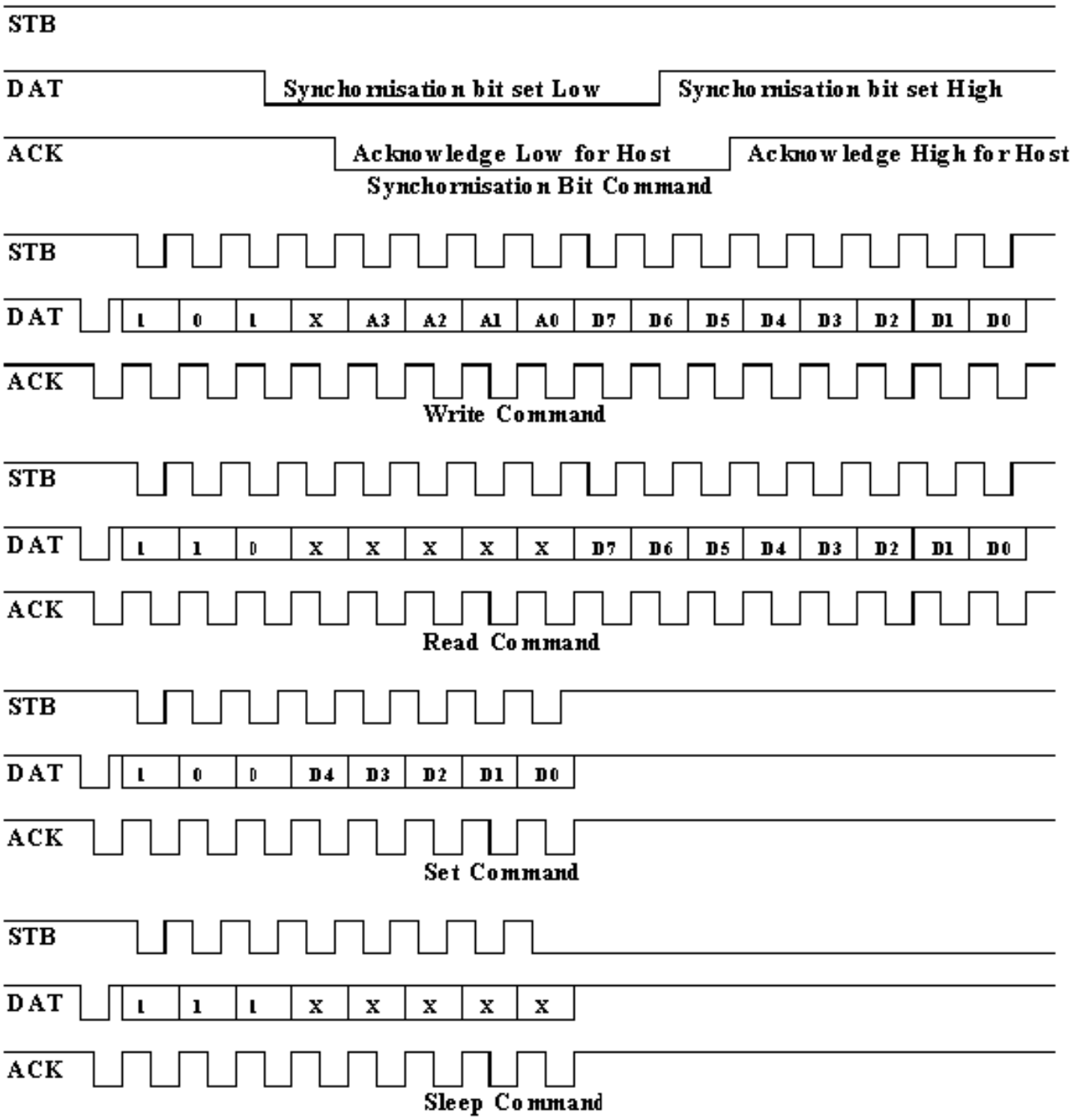
6．发送命令使 SPL10A 进入 Sleep Mode 时，命令发送完毕，必须把 STB 置低。

7．SPL10A-221 通讯时序

DATA 为主 CPU 与 SPL10A-221 之间的通讯数据线，同时为主 CPU 的同步信号线。STB 为时钟信号线，由主 CPU 送出。ACK 为 SPL10A-221 的应答线。

所有数据从高位开始传送。传送时序中没有对 STB、ACK 信号脉宽的要求，每一位数据的实际传送时间以 100ms 内收到 ACK 信号为准，收到 ACK 信号后即可进行下一位数据的传送。

在每帧数据通讯前，由 CPU 发出同步信号。在通讯过程中，若主 CPU 在 100ms 内未收到 ACK 信号，表示通讯失败，必须重新发同步信号和数据信号。在主 CPU 向 SPL10A-221 写操作过程中，DATA 方向由主 CPU 向 SPL10A-221。DATA 由主 CPU 准备就绪后，STB 下降沿通知 SPL10A-221 数据就绪，ACK 下降沿表示 SPL10A-221 准备接收数据，STB 上升沿表示数据传送完毕，ACK 上升沿表示 SPL10A-221 接收完毕。在主 CPU 向 SPL10A-221 读操作过程中，DATA 方向先由主 CPU 向 SPL10A-221，传送读操作命令，具体操作同写操作，随后 DATA 方向变更为由 SPL10A-221 向主 CPU。ACK 上升沿表示 SP10A-221 准备放置数据，STB 下降沿表示主 CPU 准备接收数据，ACK 下降沿表示 SPL10A-221 数据就绪，主 CPU 读取数据，STB 上升沿表示主 CPU 读取完毕。



8．命令总表

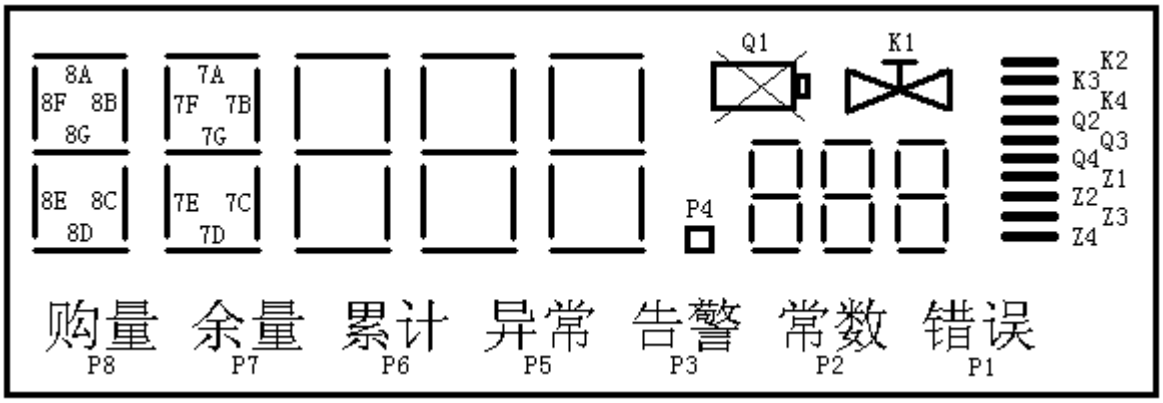
	Bit Stream Position MSB .. LSB															
Command Type	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Write LCD	1	0	1	0	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Write RTC	1	0	1	1	X	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Read Key	1	1	0	0	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0
Read RTC	1	1	0	1	0	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Set System	1	0	0	D4	D3	D2	D1	D0	Remark							
	1	0	0	0	0	0	0	1	Tone off							
	1	0	0	0	0	0	1	0	Tone on							
	1	0	0	0	0	1	0	0	Set Tone 1K							
	1	0	0	0	0	1	0	1	Set Tone 2K							
	1	0	0	0	0	1	1	0	Set Tone 4K							
	1	0	0	0	0	0	1	0	LCD 1/4Duty,1/3Bias							
	1	0	0	1	1	0	0	0	Key Continue refresh Mode							
	1	0	0	1	1	0	0	1	Key Hold Until Read Out							
	1	1	1	X	X	X	X	X	Go To Sleep							

9．RTC 寄存器

Register Name	**Range Data**	**Register Definition**								**Address A2~A0**
		D7	D6	D5	D4	D3	D2	D1	D0	
Seconds	00~59	0	10SEC			SEC				000
Minutes	00~59	0	10MIN			MIN				001
Hours	00~23	0	0	10HOUR		HOUR				010
Date	01~31	0	0	10DATE		DATE				011
Month	01~12	0	0	0	10M	MONTH				100
Week	01~07	0	0	0	0	WEEK				101
Year	00~19	0	0	0	10Y	YEAR				110

LCD 引脚及段定义



PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
COM0	K4	Q4	Z4	1E	1A	2E	2A	3E	3A	4E	4A	5E	5A	6E	6A	7E	7A	8E	8A				COM0
COM1	K3	Q3	Z3	1F	1B	2F	2B	3F	3B	4F	4B	5F	5B	6F	6B	7F	7B	8F	8B			COM1	
COM2	K2	Q2	Z2	1G	1C	2G	2C	3G	3C	4G	4C	5G	5C	6G	6C	7G	7C	8G	8C		COM2		
COM3	K1	Q1	Z1	P1	1D	P2	2D	P3	3D	P4	4D	P5	5D	P6	6D	P7	7D	P8	8D	COM3			

4 软件说明(非必需)

对模组之必须搭配应用的软件进行说明

5 应用举例

以下范例以 SPCE061A 指令集编写，可与凌阳 16 位单片机 SPCE061A 实验仪、35 板、61 板直接连接使用。注意连接时，VDDIO 选择 3.3V，使用 IOA5-IOA7 做 STB、ACK、DATA 输入输出接口。

范例分为三个部分：

一．主程序部分 main.c

主程序由用户自行编写，此处的主程序仅为示例，实现的功能是：若有键按下，将 8 个按键对应的 1-8 分别在 LCD 的相应位置显示。主程序调用键盘扫描子程序 CheckKeypad()，即可读取按键。主程序修改显示缓冲区内容，然后调用 LCD 刷新子程序 RefreshLCD()，即可完成 LCD 显示。

显示缓冲区有 11 个字单元，前 8 个字单元控制数字显示。如要显示“ 12345678 ”，则在显示缓冲区内顺序写入：“ 0x01，0x02，0x03，0x04，0x05，0x06，0x07，0x08 ”；如要显示“ 20040723 ”，则在显示缓冲区内顺序写入：“ 0x02，0x00，0x00，0x04，0x00，0x07，0x02，0x03 ”。第 9 个字单元的低 8 位自 D0 至 D7 分别控制 LCD 段定义中 P1 至 P8 的显示，0 不显示，1 显示。第 10 个字单元控制 LCD 段定义中 Z1-Z4、Q2-Q4、K2-K4 的显示，字单元内容 0x01 则 Z4 显示，0x02 则 Z4Z3 显示，0x03 则 Z4Z3Z2 显示，0x04 则 Z4Z3Z2Z1 显示，以此类推。第 11 个字单元内容的 D0 位为 1 则 LCD 段定义中 K1 显示，D1 位为 1 则 LCD 段定义中 Q1 显示。

```
#include "SPCE061V004.H"


extern InitIsr();
extern InitSPL10A();
extern RefreshLCD();
extern CheckKeypad();
extern DisplayBuffer;


unsigned int *p;
//主程序功能：扫描键盘，若有键按下，将键值送显示
main()
{
    p=&DisplayBuffer;           //指针 P 指向显示缓冲区
    InitIsr();                  //开放 1KHz 中断，用于 100ms 定时
    InitSPL10A();               //SPL10A 初始化
    while(1)  {
        switch(CheckKeypad())   { //扫描键盘
            case 1:
                *p=1;                   //将键值送入显示缓冲区
                RefreshLCD();
```

```
            break;
        case 2:
            *(p+1)=2;
            RefreshLCD();
            break;
        case 3:
            *(p+2)=3;
            RefreshLCD();
            break;
        case 4:
            *(p+3)=4;
            RefreshLCD();
            break;
        case 5:
            *(p+4)=5;
            RefreshLCD();
            break;
        case 6:
            *(p+5)=6;
            RefreshLCD();
            break;
        case 7:
            *(p+6)=7;
            RefreshLCD();
            break;
        case 8:
            *(p+7)=8;
            RefreshLCD();
            break;
        default:
            break;
    }
```

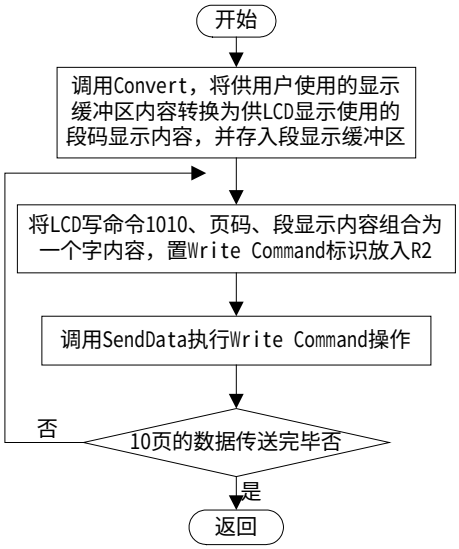
```
        *P_Watchdog_Clear=0x1;          //清狗
    }
}
```

二．SPL10A-221 驱动部分 spl10a_driver.asm

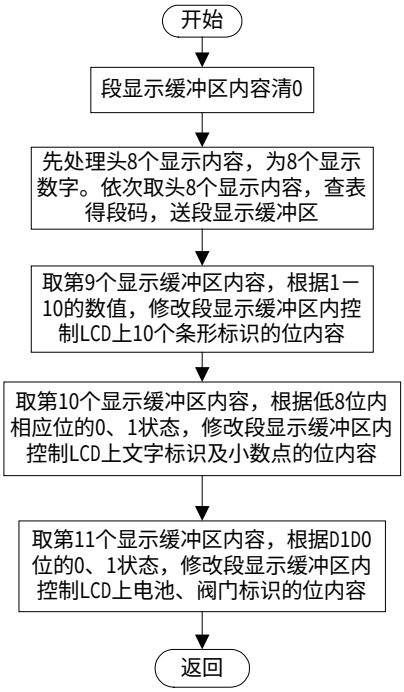
这部分程序包含若干功能子程序，完成包括 IO 口初始化功能、SPL10A-221 初始化功能、SPCE061 与 SPL10A-221 之间的通讯功能。

相关程序流程图

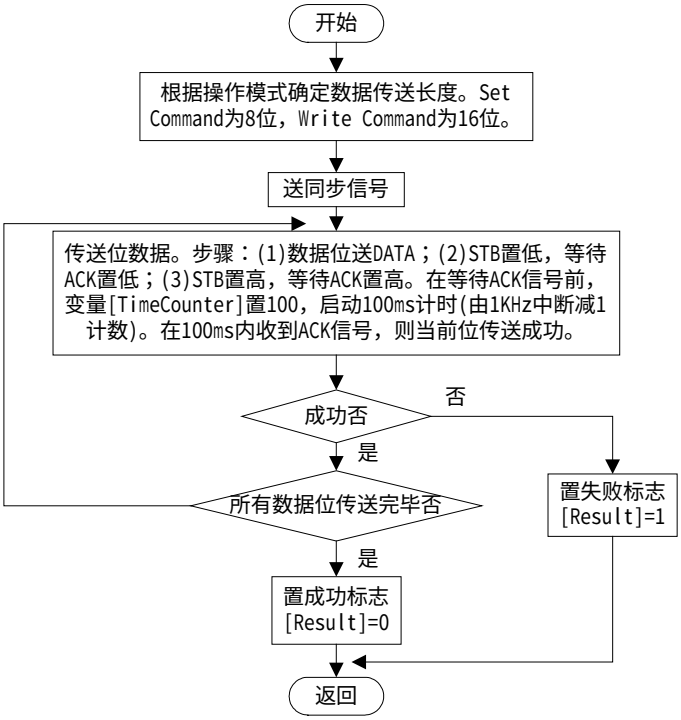
1． LCD 显示/刷新程序 RefreshLCD，完成用户修改显示缓冲区后，LCD 显示更新功能。



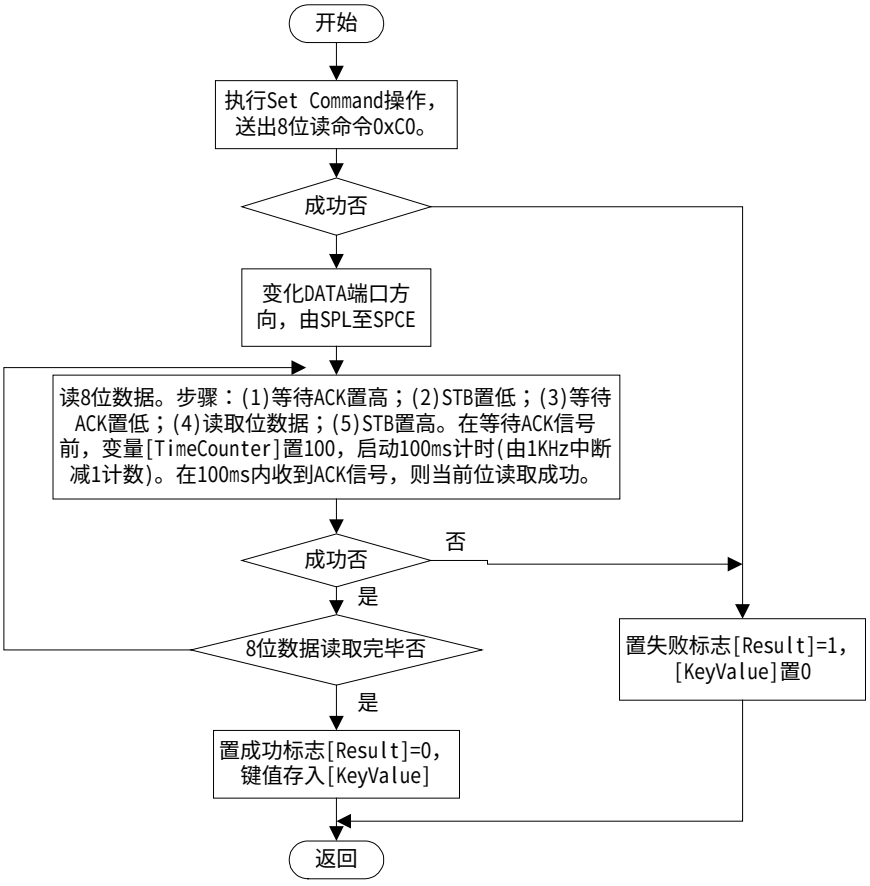
2．段码转换程序 Convert，完成将用户使用的显示缓冲区内容转换为 LCD 显示所需要的段码显示内容，并存入段显示缓冲区。



3．SPL10A-221 写操作程序 SendData。完成 Write Command、Set Command 操作。



4．键盘读取程序 CheckKeypad。实际完成一个 Read Command 操作。



程序范例

```
        .include hardware.inc

        .external _TimeCounter

        .define SETCOMMAND          0x00
        .define WRITECOMMAND  0x01

        .define STB                 0x20
        .define ACK                 0x40
        .define BITDATA             0x80
        .define TRUE            0x00
        .define FALSE           0x01

        .ram
        .public _Result,_KeyValue,_DisplayBuffer
        .var _Result,_KeyValue  //_Result=1 表示对 SPL10A 操作失败，_KeyValue 保存 1-8 键值(0 表示无键)
        .var Page
_DisplayBuffer:     .dw 11 dup(0) //显示缓冲区。第 0-7 单元为自左至右共 8 个数据。第 8 单元为增量条形标识。
                                  //第 9 单元为文字(含小数点)标识。第 10 单元为电池、阀门标识。
LCDBuffer:          .dw 10 dup(0) //段显示缓冲区，将显示缓冲区内的内容转变为段码后存入段显示缓冲区

        .data
//SPL10A 的总命令表
CommandTbl:
        .dw 0x8100          //00:Tone off
        .dw 0x8200          //01:Tone on
        .dw 0x8400          //02:Set Tone 1K
        .dw 0x8500          //03:Set Tone 2K
        .dw 0x8600          //04:Set Tone 4K
        .dw 0x8700          //05:Battery Level Detect
        .dw 0x8300          //06:LCD 1/4duty 1/3bias
```

```
        .dw 0x9000          //07:Key Mode0(5*4)       no shift
        .dw 0x9100          //08:Key Mode1(4*4)       shift1
        .dw 0x9200          //09:Key Mode2(3*4)       shift1,2
        .dw 0x9300          //0A:Key Mode3(3*4)       rolling F/B
        .dw 0x9800          //0B:Key continue refresh mode
        .dw 0x9900          //0C:Key hold until read out
        .dw 0xE000          //0D:Sleep mode

//数据长度表
//在 Set Command 操作中数据长度 8 位，在 Write Command 操作中数据长度为 16 位
SizeTbl:
        .dw 8,16

//数字显示段码
SegTbl:
        .dw 0x003F,0x0006,0x005B,0x004F,0x0066          //0，1，2，3，4
        .dw 0x006D,0x007D,0x0007,0x007F,0x006F              //5，6，7，8，9
        .dw 0x0000                                          //全灭

//增量条形标识段码
StickTbl:
        .dw 0x00,0x01,0x03,0x07,0x0F


        .code
        .public InitIO
//初始化 IO 口，其中 A7 接 DATA(输出)，A5 接 STB(输出)，A6 接 ACK(输入)
//入口：无
//出口：无
InitIO:  .proc
        R1=0x00A0
        [P_IOA_Dir]=R1          //A7(DATA)、A5(STB)输出,A6(ACK)输入
        [P_IOA_Attrib]=R1
```

```
        R1=0x00E0
        [P_IOA_Data]=R1
        RETF
            .endp

        .public _InitSPL10A
//SPL10A 初始化，包括初始化 IO 口、向 SPL10A 写初始化命令
//入口：无
//出口：无
_InitSPL10A: .proc
        CALL InitIO             //初始化 IO 口
        BP=_DisplayBuffer//显示缓冲区初始化
        R1=0x0A                 //前 8 个单元写入 0x0A 则不显示
        R2=8
CL1:    [BP++]=R1
        R2-=1
        JNZ CL1
        R1=0                    //后 3 个单元写入 0 则不显示
        R2=3
CL2:    [BP++]=R1
        R2-=1
        JNZ CL2
        R1=0x0B
        CALL SetCommand         //送 Key 初始化命令:Key continue refresh mode
        RETF
            .endp

        .public SetCommand
//向 SPL10A 送命令
//入口：R1。R1 内容不是真正的控制命令，而是实际命令在命令表中的存储偏移量，通过查表得出实际命令
//出口：无
SetCommand: .proc
```

```
        BP=CommandTbl       //BP 指向命令表首地址
        BP+=R1
        R1=[BP]             //取出实际命令
        R2=SETCOMMAND       //当前操作为 Set Command
        CALL SendData       //送出
        CALL Delay          //让 STB、DATA 信号保持一段时间的高电平
        RETF
            .endp

        .public SendData
//数据传送，完成 061A 向 SPL10A 的送数据过程。传送过程遵循 Set Command、Write Command 通讯时序
//入口：R1--数据，R2--操作模式(Set Command、Write Command)
//出口：无
SendData:   .proc
        BP=SizeTbl          //确定不同操作模式下数据长度
        BP+=R2
        R2=[BP]
        R3=[P_IOA_Data]     //STB、DATA 信号置高，准备发送同步信号
        R3|=STB+BITDATA
        [P_IOA_Data]=R3
        R3=[P_IOA_Data]     //DATA 置低，送同步信号
        R3^=BITDATA
        [P_IOA_Data]=R3
        R3=100              //启动 100ms 定时
        [_TimeCounter]=R3
Wait1:
        R3=[_TimeCounter]
        JNZ OK1
        GOTO Error          //100ms 内未收到 ACK 信号则传送失败
OK1:    R3=[P_IOA_Data]     //等待 ACK 信号为低
        R3&=ACK
        JNZ Wait1
```

```
        R3=[P_IOA_Data]      //DATA 置高
        R3|=BITDATA
        [P_IOA_Data]=R3
        R3=100               //启动 100ms 定时
        [_TimeCounter]=R3
Wait2:
        R3=[_TimeCounter]
        JNZ OK2
        GOTO Error
OK2:    R3=[P_IOA_Data]      //等待 ACK 信号为高
        R3&=ACK
        JZ Wait2             //只有 ACK 信号随 DATA 信号从高变低，又从低变高，则同步成功
        R4=0x8000            //数据从最高位开始传送
Next:   R3=R1
        R3&=R4               //判断传送位为高还是低
        JZ ResBit
        R3=[P_IOA_Data]      //传送位为高
        R3|=BITDATA
        JMP Send
ResBit: R3=[P_IOA_Data]      //传送位为低
        R3|=BITDATA
        R3^=BITDATA
Send:   [P_IOA_Data]=R3
        NOP
        NOP
        NOP
        NOP
        R3=[P_IOA_Data]      //STB 信号置低
        R3|=STB
        R3^=STB
        [P_IOA_Data]=R3
        R3=100               //启动 100ms 定时
```

```
        [_TimeCounter]=R3
Wait3:
        R3=[_TimeCounter]
        JZ Error
        R3=[P_IOA_Data]         //等待 ACK 信号置低
        R3&=ACK
        JNZ Wait3
        R3=[P_IOA_Data]         //STB 信号置高
        R3|=STB
        [P_IOA_Data]=R3
        R3=100                  //启动 100ms 定时
        [_TimeCounter]=R3
Wait4:
        R3=[_TimeCounter]
        JZ Error
        R3=[P_IOA_Data]         //等待 ACK 信号置高
        R3&=ACK
        JZ Wait4
        R3=0x01                 //清狗
        [P_Watchdog_Clear]=R3
        R4=R4 LSR 1             //R4 右移一位，以便判断下一位数据
        R2-=1
        JNZ Next            //继续，直至所有的数据位传送完毕
        R3=TRUE
        JMP Stop
Error:  R3=FALSE
Stop:   [_Result]=R3
        R3=[P_IOA_Data]         //传送完毕后 STB、DATA 信号置高
        R3|=STB+BITDATA
        [P_IOA_Data]=R3
        RETF
            .endp
```

```
        .public Delay
Delay:  .proc
        R3=0x8000
Wait:   R1=0x01                 //清狗
        [P_Watchdog_Clear]=R1
        R3-=1
        JNZ Wait
        RETF
        .endp

        .public _CheckKeypad
//键盘读取程序。过程遵循 Read Command 时序。
//入口：无
//出口：_KeyValue。0 表示无键按下，1-8 对应 8 个按键
_CheckKeypad:   .proc
        R1=0xC000           //取出实际命令
        R2=SETCOMMAND       //当前操作为 Set Command
        CALL SendData       //送出
        R1=[_Result]
        JZ ComeOn
        R1=0
        [_KeyValue]=R1
        RETF
ComeOn: R1=0x0020           //变化端口方向
        [P_IOA_Dir]=R1      //A7(DATA)输入、A5(STB)输出,A6(ACK)输入
        [P_IOA_Attrib]=R1
        R1=0x00E0
        [P_IOA_Data]=R1
        R1=8                //读入的数据是 8 位
        R2=0                //读入的数据暂存处
        R4=0x0080
```

```
RdNext:  R3=100                 //启动 100ms 定时
         [_TimeCounter]=R3
Wait5:   R3=[_TimeCounter]
         JZ RdFail
         R3=[P_IOA_Data]        //等待 ACK 置高，表示 SPL 准备放置数据
         R3&=ACK
         JZ Wait5
         R3=[P_IOA_Data]        //STB 置低表示 SPCE 准备接收数据
         R3|=STB
         R3^=STB
         [P_IOA_Data]=R3
         R3=100                 //启动 100ms 定时
         [_TimeCounter]=R3
Wait6:   R3=[_TimeCounter]
         JZ RdFail
         R3=[P_IOA_Data]        //等待 ACK 置低，表示 SPL 数据就绪
         R3&=ACK
         JNZ Wait6
         R3=[P_IOA_Data]        //读入位数据
         R3&=BITDATA
         JNZ SetBit
         R2|=R4
         R2^=R4                 //读入的位为低
         JMP Con
SetBit:  R2|=R4                 //读入的位为高
Con:     R4=R4 LSR 1
         R3=[P_IOA_Data]        //STB 置高，表示 SPCE 已完成读取
         R3|=STB
         [P_IOA_Data]=R3
         R1-=1
         JNZ RdNext
         R1=TRUE
```

```
        [_Result]=R1
        JMP RdSuc
RdFail: R1=FALSE
        [_Result]=R1
        R2=0
RdSuc:  [_KeyValue]=R2
        CALL InitIO             //恢复 IO 口原方向
        R3=[P_IOA_Data]         //读完毕后 STB、DATA 信号置高
        R3|=STB+BITDATA
        [P_IOA_Data]=R3
        CALL Delay
        R1=[_KeyValue]          //返回值
        RETF
            .endp

        .public Convert
//显示缓冲区内容到段显示缓冲区内容的转变，完成将显示缓冲区内的所有显示内容转换为要
//实际显示的段码送入段显示缓冲区功能
//入口：_DisplayBuffer
//出口：LCDBuffer
Convert:  .proc
        BP=LCDBuffer            //首先清除段显示缓冲区
        R1=0
        R2=10                   //段显示缓冲区有 10 个单元
Clr:    [BP++]=R1
        R2-=1
        JNZ Clr
        R1=9
        [Page]=R1               //先从第 9 页到第 0 页写数据
        R1=_DisplayBuffer//R1 指向显示缓冲区
        R2=8                    //先处理 8 个数字
Lp1:    BP=SegTbl               //BP 指向段码表
```

```
        R3=[R1++]           //取显示缓冲区内容
        BP+=R3
        R3=[BP]             //取段码
        PUSH R3 TO [SP]     //段码的高 4 位与低 4 位要分开处理
        R3&=0x0F
        BP=LCDBuffer        //BP 指向段显示缓冲区
        R4=[Page]
        BP+=R4
        R4=[BP]
        R4|=R3
        [BP]=R4             //存入低 4 位段码
        POP R3 FROM [SP]
        R3&=0xF0
        BP=LCDBuffer
        R4=[Page]
        R4-=1
        BP+=R4
        R4=[BP]
        R4|=R3
        [BP]=R4             //存入高 4 位段码
        R3=[Page]
        R3-=1
        [Page]=R3
        R2-=1
        JNZ Lp1             //8 个数字未转换完毕
        R3=[R1++]           //增量条形标识显示数据范围 1-10
        PUSH R3 TO [SP]
        CMP R3,5
        JNAE Stick1
        R3=4
Stick1: BP=StickTbl         //BP 指向条形标识段码
        BP+=R3
```

```
        R3=[BP]
        R4=1                    //第 1 页
        BP=LCDBuffer
        BP+=R4
        R4=[BP]
        R4|=R3
        [BP]=R4
        POP R3 FROM [SP]
        PUSH R3 TO [SP]
        CMP R3,5
        JAE C10
        POP R3 FROM [SP]
        JMP Word
C10:    CMP R3,8
        JNAE C1
        JMP C2
C1:     R3-=4
        JMP Stick2
C2:     R3=3
Stick2: BP=StickTbl
        BP+=R3
        R3=[BP]
        R3=R3 LSL 4
        BP=LCDBuffer            //第 0 页
        R4=[BP]
        R4|=R3
        [BP]=R4
        POP R3 FROM [SP]
        CMP R3,8
        JNAE Word
        CMP R3,11
        JNAE C3
```

```
        JMP C4
C3:     R3-=7
        JMP Stick3
C4:     R3=3
Stick3: BP=StickTbl
        BP+=R3
        R3=[BP]
        BP=LCDBuffer            //第 0 页
        R4=[BP]
        R4|=R3
        [BP]=R4
Word:   R2=1
        R3=1
        [Page]=R3
C6:     R4=[R1]                 //取出文字标识显示内容
        R4&=R3
        JZ C5
        BP=LCDBuffer
        BP+=R2
        R4=[BP]
        R4|=0x80
        [BP]=R4
C5:     R3=R3 LSL 1
        R2+=1
        CMP R2,9
        JNE C6
        R1+=1
        R2=[R1]
        BP=LCDBuffer
        R3=[BP]
        PUSH R2 TO [SP]
        R2&=0x01
```

```
        JNZ C7
        JMP C8
C7:     R3|=0x08
C8:     POP R2 FROM [SP]
        R2&=0x02
        JNZ C9
        JMP C11
C9:     R3|=0x80
C11:[BP]=R3
        RETF

            .endp

        .public _RefreshLCD
//LCD 刷新程序。主程序修改显示缓冲区后调用，完成 LCD 显示。
//入口：_DisplayBuffer
//出口：无
_RefreshLCD:.proc
        CALL Convert        //调转换程序，将显示缓冲区内容转换为段显示缓冲区内容
        R1=0
        [Page]=R1
        R4=10               //共有 10 页的数据要传送
        BP=LCDBuffer        //BP 指向段显示缓冲区
Lop:R1=0xA000           //LCD 写命令
        R2=[Page]           //将页码移至 D8-D11
        R2=R2 LSL 4
        R2=R2 LSL 4
        R1|=R2
        R2=[BP++]           //取出段显示内容
        R1|=R2              //这时完整包含 LCD 写命令、页码、段显示内容
        R2=WRITECOMMAND     //Write Command 操作标识
        PUSH BP TO [SP]
```

```
        PUSH R4 TO [SP]
        CALL SendData          //向 SPL10A 送数据
        POP R4 FROM [SP]
        POP BP FROM [SP]
        R2=[Page]              //准备传送下一页数据
        R2+=1
        [Page]=R2
        R4-=1
        JNE Lop
        RETF
            .endp
```

三．中断定时部分 isr.asm

这部分程序为 1KHz 中断服务程序，结合计数变量 TimeCounter，完成 100ms 定时。100ms 定时应用于 ACK 信号的等待，若 100ms 内未收到 ACK 信号，则表示通讯失败。

```
        .include hardware.inc

        .define RUN_1KHz_TimeBase_INT 0x0010

        .ram
        .public _TimeCounter
        .var _TimeCounter

        .text
        .public    _IRQ4
_IRQ4:
        PUSH R1,R5 to [SP]
        R1=0x0010
        TEST R1,[P_INT_Ctrl]           //是否为 1KHz 中断
        JNZ   IRQ4_1K                  //1KHz 中断
        R1=0x0020
        TEST R1,[P_INT_Ctrl]           //是否为 2KHz 中断
        JNZ   IRQ4_2K                  //2KHz 中断
```

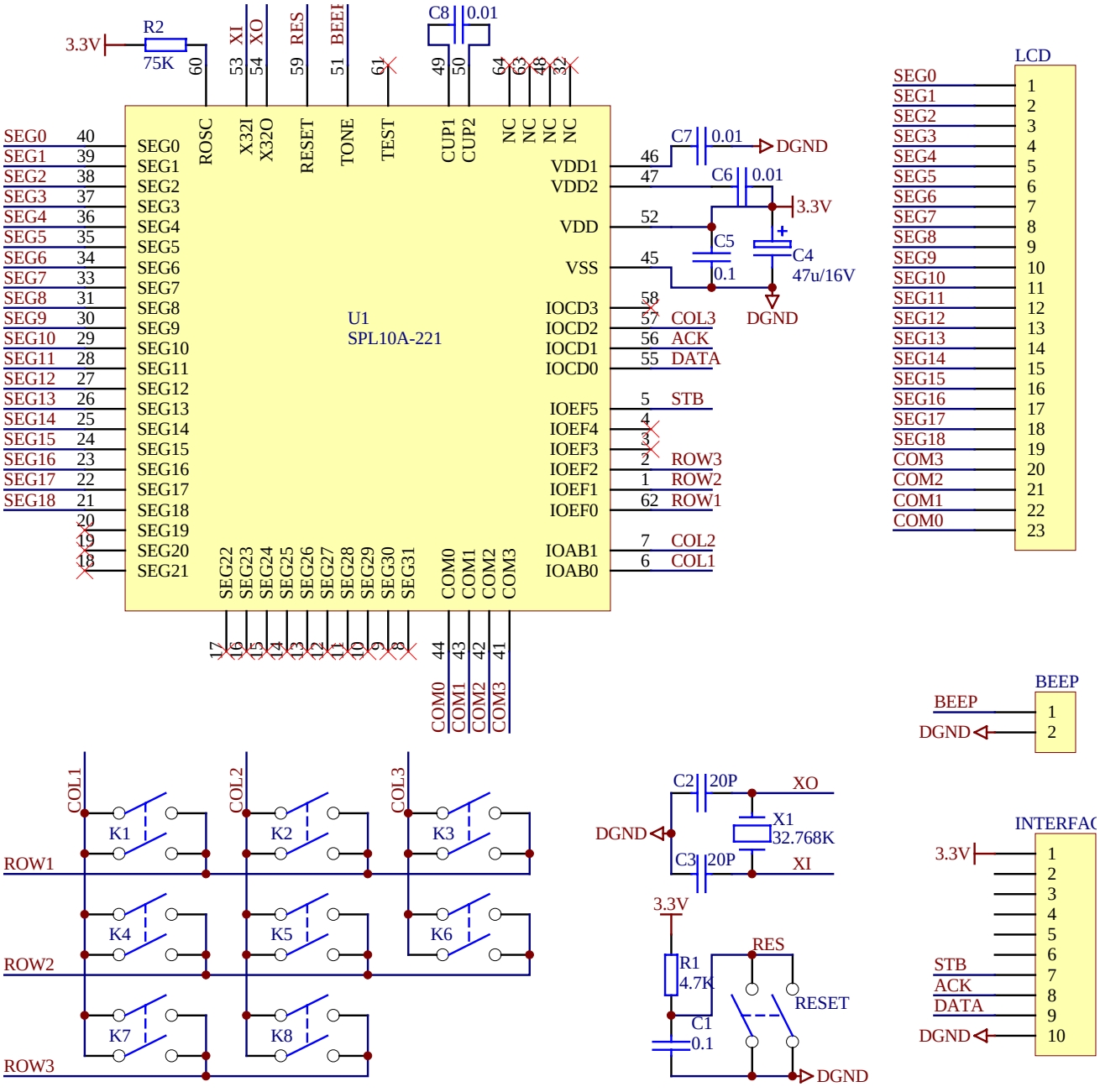
```
IRQ4_4K:
        R1=0x0040
        [P_INT_Clear]=R1
        POP R1,R5 FROM [SP]
        RETI
IRQ4_2K:
        R1=0x0020
        [P_INT_Clear]=R1
        POP R1,R5 FROM [SP]
        RETI
IRQ4_1K:
        R1=[_TimeCounter]
        JZ Back
        R1-=1                   //100ms 定时计数
        [_TimeCounter]=R1
Back:   R1=0x0010
        [P_INT_Clear]=R1
        POP R1,R5 FROM [SP]
        RETI

        .code
        .public _InitIsr
_InitIsr: .proc
        R1=0
        [_TimeCounter]=R1
        R1=RUN_1KHz_TimeBase_INT        //开放 1KHz 中断
        [P_INT_Ctrl]=R1
        int IRQ
        RETF
            .endp
```

6 常见问题解答

7 附录

7.1 电路原理图


U1
SPL10A-221
3.3V
R2
75K
C8 0.01
C7 0.01
C6 0.01
C5 0.1
C4 47u/16V
DGND
LCD
BEEP
INTERFAC
K1
K2
K3
K4
K5
K6
K7
K8
ROW1
ROW2
ROW3
COL1
COL2
COL3
C2 20P
C3 20P
X1 32.768K
XO
XI
R1 4.7K
C1 0.1
RES
RESET
STB
ACK
DATA
COM3
COM2
COM1
COM0

7.2 实物图

7.3 配件清单

Item	Quantity	Reference	Part
1	2	C1,C5,	104
2	3	C6,C7,C8	103
3	2	C2,C3	20p
4	1	C4	47u/16V
5	1	R1	4.7K
6	1	R2	75K
7	1	X1	32.768K
8	9	S_KEY	Key
9	1	U1	SPL10A-221