基于状态机的 LCD 多级菜单设计

发布时间： 2013-01-23 11:41:14

来源：互联网

1 概述

液晶显示器（ Liquid Crystal Display ， LCD）由于其体积和功耗等因素，非常适合嵌入式环境的使用。 近年来， 随着微处理器性能的提高， 嵌入式系统实现的功能越来越强大， 产生的数据量也越来越大。 相对应地， 需要显示的数据量也随之增大。嵌入式环境下使用 LCD 显示器，由于条件限制，体积较小，且显示的内容有限。 而且，传统的 LCD 显示模式总是不加选择地显示所有监控的信息，在监控的信息量非常庞大时会导致不能及时显示用户所需求的信息。 多级菜单显示则是将信息分类显示的一种显示方式， 该方式根据用户的选择， 对显示信息加以筛选并分级显示， 这样既保证用户获取其所需的信息， 又能保障信息显示的实时性。

2 多级菜单的结构

设计多级菜单的目的在于将需要显示的信息分门归类， 方便用户筛选。 所以在设计菜单时需要根据整个系统的功能和要求来设定菜单的级数， 以及各级子菜单的个数。 整个多级菜单的拓扑结构为树型结构， 主菜单为根节点， 子菜单为枝节点，最后一级菜单为叶节点，如图 1 所示。

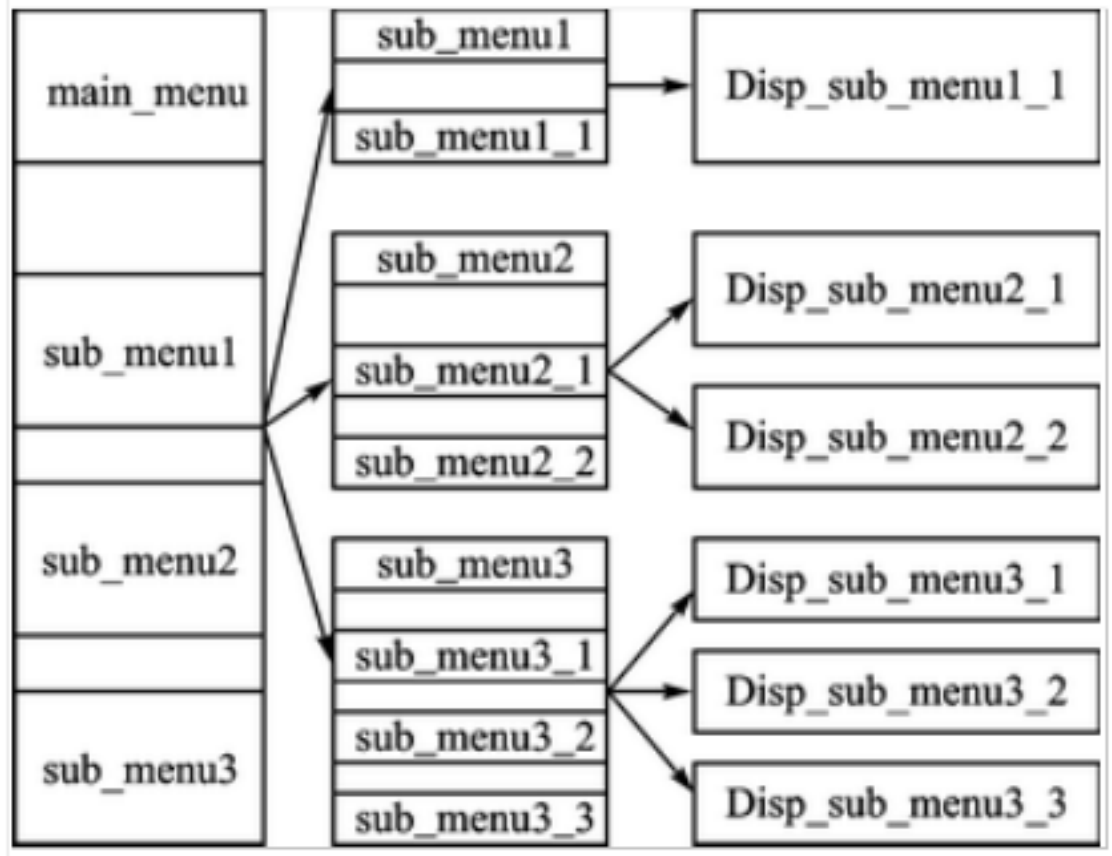

图 1 多级菜单的结构图

3 多级菜单的程序设计

3.1 循环方式

循环方式的设计思路： 预先定义一个包含 6 个结构元素的结构体、 5 个字符型和 1 个指针型。第 1 个字符变量存放当前界面的索引号；第 2 个字符变量存

放按下“ down(向下) ”键时需要跳转到的索引号；第 3 个字符变量存放按下 “ up（向上） ”键时需要跳转到的索引号；第 4 个字符变量存放按下 “ enter(进入) ”键时需要跳转的索引号；第 5 个字符变量存放按下 “ esc(退出) ”键时需要跳转的索引号； 第 6 个变量为函数指针变量, 存放当前索引号下需要执行的函数的入口地址。

将所有需要显示的界面其所对应的执行函数和按键索引号以结构体的形式列表存储。具体实现如下：

```
typedef struct{
uchar    index;              //第 1 个,存放当前界面的索引号
uchar    down_index;         //第 2 个,按下“ down(向下) ”键时需要跳转到的索引号；
uchar    up_index;           //第 3 个,按下“ up(向上) ”键时需要跳转到的索引号；
uchar    enter_index;        //第 4 个,按下“ enter(进入) ”键时需要跳转的索引号；
uchar    esc_index;          //第 5 个,按下“ esc(退出) ”键时需要跳转的索引号；
void (*operate)();           //第 6 个,为函数指针变量，存放当前索引号下需要执行
                                  的函数的入口地址。
}Key_index_struct;
```

假设菜单分 3 级，共 10 个界面，则有：

```
Key_index_struct const Key_tab[10]={
{0, 0, 0, 1, 0,(*main_menu)},
{1, 2, 3, 4, 0,(*sub_menu1)},
{2, 3, 1, 5, 0,(*sub_menu2)},
{3, 1, 2, 7, 0,(*sub_menu3)},
{4, 4, 4, 4, 1,(*sub_menu1_1)},
{5, 6, 6, 5, 2,(*sub_menu2_1)},
{6, 5, 5, 5, 2,(*sub_menu2_2)},
{7, 8, 9, 7, 3,(*sub_menu3_1)},
{8, 9, 7, 8, 3,(*sub_menu3_2)},
{9, 7, 8, 9, 3,(*sub_menu3_3)},
};

void Lcd_display(void)

{
switch(Key_status)

{
 case enter:
 Key_fun=Key_tab[Key_fun].enter_index;
 break;
 case down:
 Key_fun=Key_tab[Key_fun].down_index;
 break;
 case up:
```

```
Key_fun=Key_tab[Key_fun].up_index;
break;
case esc:
Key_fun=Key_tab[Key_fun].esc_index;
break;
default:
return;
break;
}
Key_fun_Pt=Key_tab[Key_fun].operate;
(*Key_fun_Pt)();//  执行当前按键的操作
}
```

当微处理器扫描键盘检测到有按键按下时，　根据按键按下的类型，　返回在当前界面下其所对应的跳转索引号，并执行相应的函数。

由于每个界面的绘制都是由一个独立函数实现的，　从循环方式的实现过程中发现，每发生一次按键按下操作都需要重新绘制整个屏幕。　如果核心处理器是低速主频的处理器，　在界面切换的时候会闪烁。　而且，每一个界面都有固定不变的索引号，在增加或删除界面的时候需要重新修改整个列表，　降低了程序的可移植性。

3.2　查询方式

查询方式是通过结构体对自身的递归调用实现菜单的多级嵌套。

结构体通过对自身的两次调用构建双向列表。一个菜单界面即为一个节点，节点的前驱和后继分别存放其父节点和子节点的入口地址。

菜单参数的结构体定义如下：

```
typedef struct Lcd_menu_content{
uchar *lpIcon;//  显示图标
uchar *lpText;//  显示文本信息
uchar nTextCount; //  菜单对应的文本信息的个数
}Lcd_menu_content;
```

每个界面对应一个节点，节点都定义成如下结构体的变量：

```
typedef struct Lcd_menu{
struct Lcd_menu*lpfather;//  父级
struct Lcd_menu*lpson;//  大儿子
uchar nSonCount;//  父级的儿子个数
Lcd_menu_content lpIconAndText;
uchar Flag_return;//  返回标志
void   (*operate)();// 处理函数入口地址
```

}Lcd_menu;

由图 1 可知，多级菜单的拓扑结构为树型拓扑结构，　即每一个节点只有一个父节点和若干个子节点。 所以，对整个叉树进行遍历即可准确地查找到菜单界面所在的节点。

结构体实现的链表如图　2 所示。

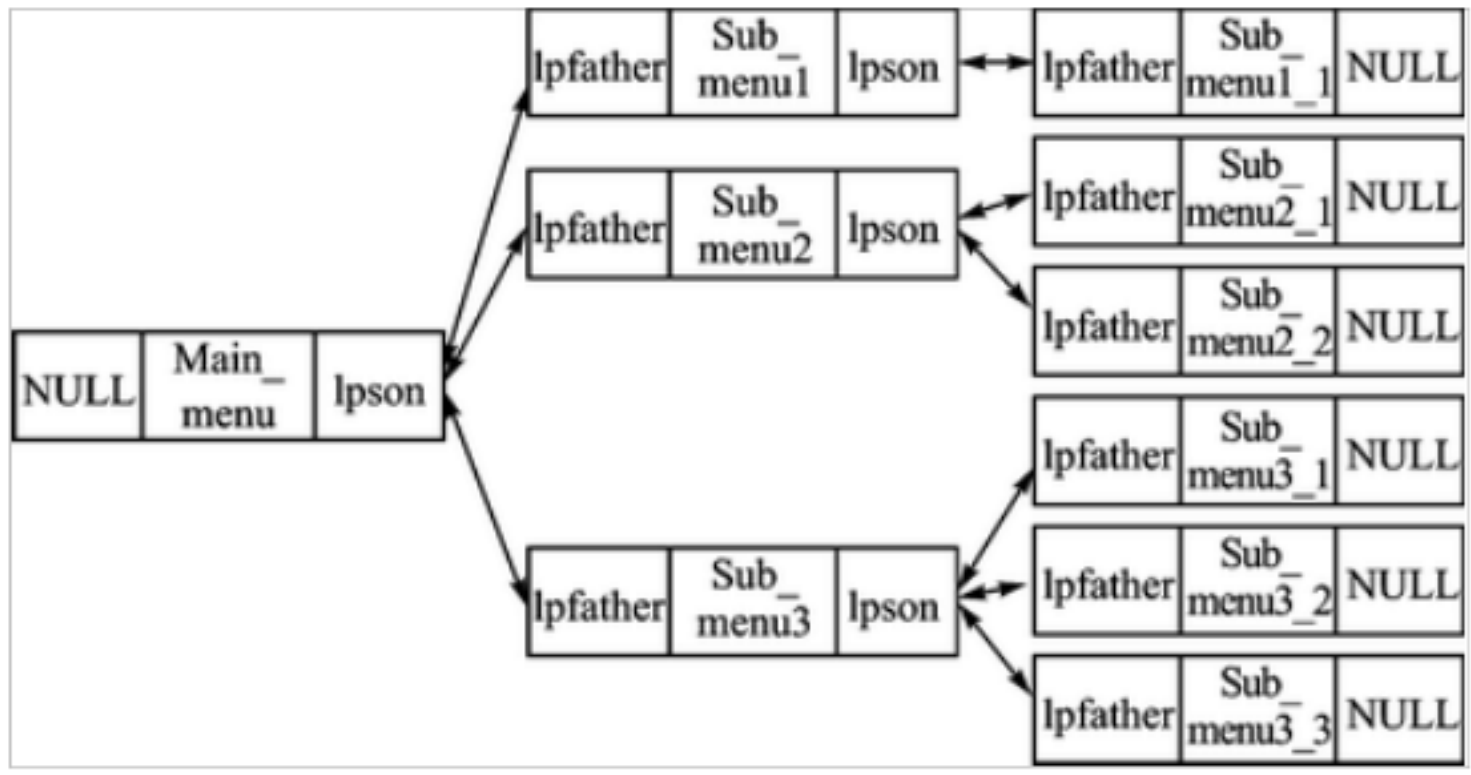

图 2　结构体链表

查询方式与循环方式相比，　由于减少了查表次数，　因而改善了 MPU 的效率；但查询方式占用　MPU 处理时间过长，不适应需要高速处理数据的应用。而且，在查询方式中增加或删除节点对程序改动较大，也不适合移植。

3.3　状态机方式

状态机是由事件驱动，　在各个状态之间跳转。 采用状态机方式时，　只需要提供驱动事件 （在此设计中驱动事件为有效的按键按下）　，然后根据按键扫描返回的键值，决定所要跳转的下一状态。

如图 3 所示，系统启动初始化是显示　Main_menu 界面，当按键检测有返回值(即有按键按下)时，根据按下的按键所代表的操作跳转到指定的状态。例如：按下 Up 或者是 down 键时，只是在 Main_menu 界面内高亮显示不同区域；按下 Enter 时，则要根据原来按下的　Up 和 down 键来选择需要跳转的方向，假设在按下 Enter 之前仅按下一次 down 键，则 key_v 的值为 2（key_v 的值默认为 1，即默认选中子菜单的第一项）　，就跳转为 Sub_menu2 界面；按下 Esc 键时，为从子菜单返回到上一级菜单，如果已经是主菜单了则返回的还是主菜单。

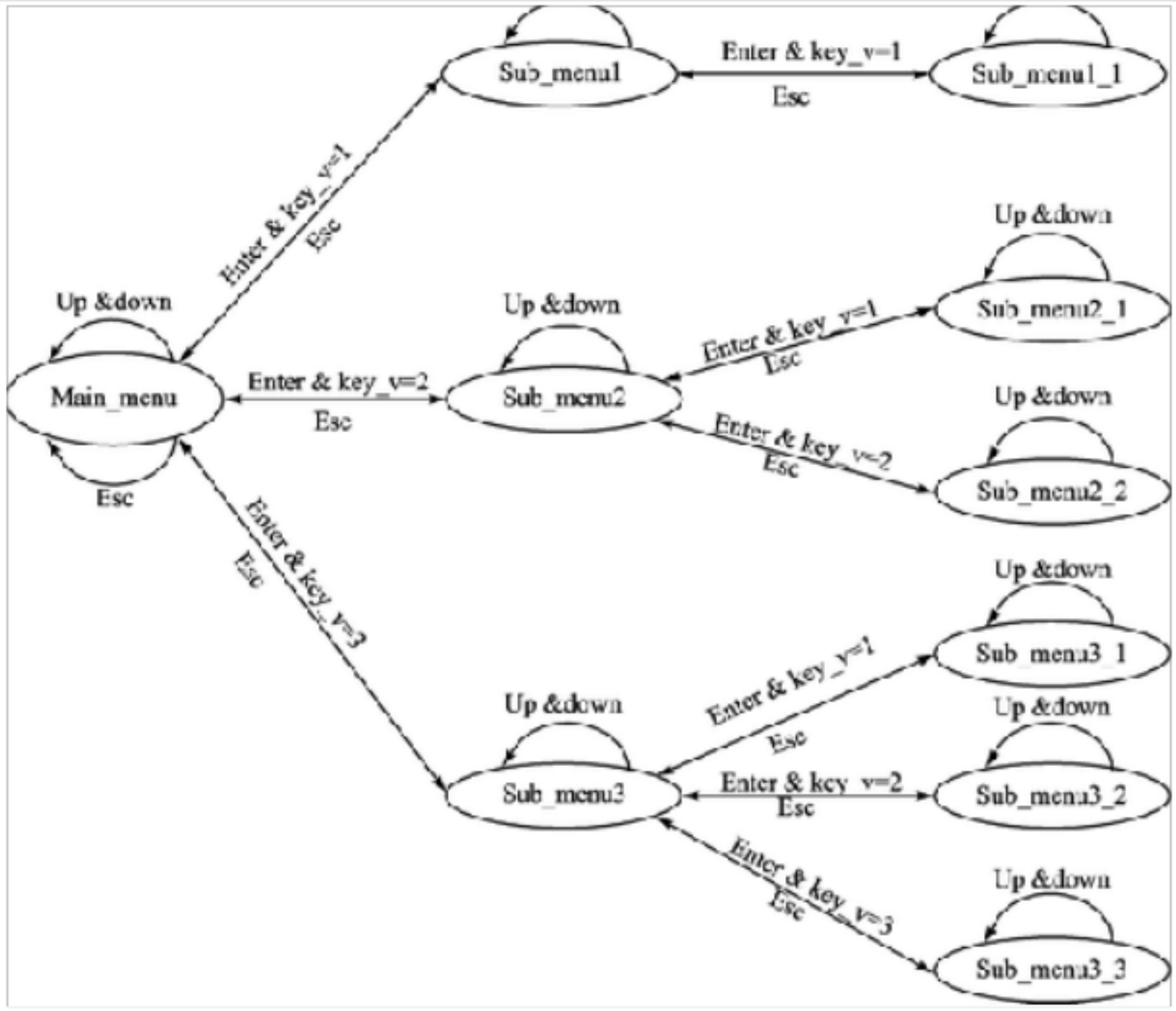

图 3　多级菜单的状态图

由于使用的是状态机的方式，　只有发生一次有效的按键，　状态才会发生一次跳转。而且，仅当　Enter 和 Esc 键按下时，才会切换界面。所以即便是在高速 MPU 应用中，也不会出现屏幕闪烁的效果。

从图 3 中可以看出，当要发生状态跳转时，　目的状态只能是当前状态几个分支预测中的一个，从而不需要遍历整个列表，能够适应高速数据处理的场合。

多级菜单的程序流程如图　4 所示。系统上电初始化后显示主菜单，　键盘扫描可以通过主程序中循环查询或者中断扫描来实现，　最终根据键盘返回的键值选择下一状态。

图 5 为基于状态机的多级菜单的实现。

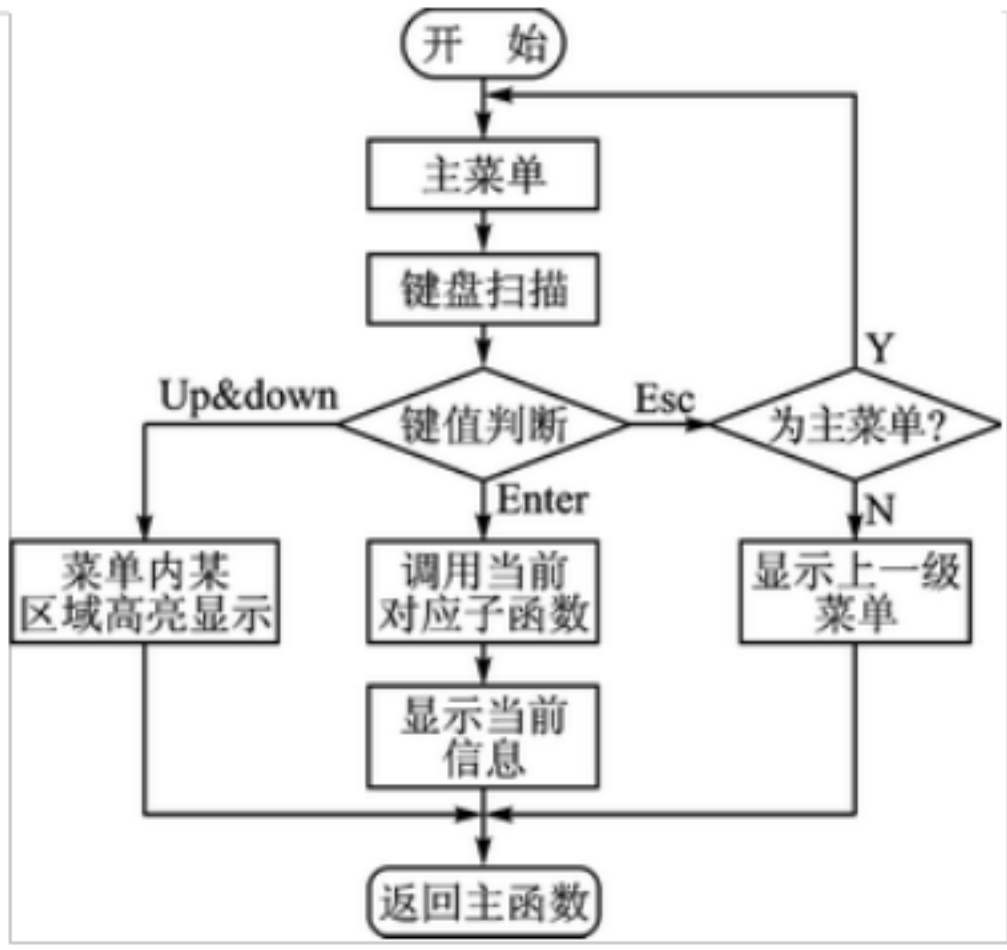

图 4　多级菜单的程序流程图

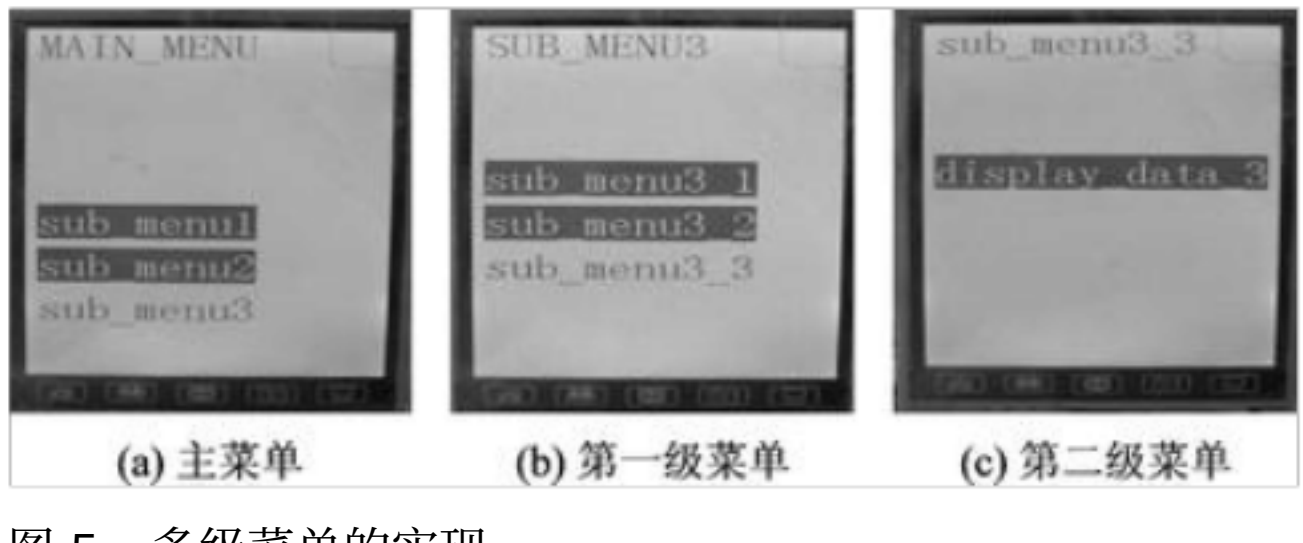

图 5　多级菜单的实现

结语

以上三种多级菜单的实现方式均具有很强的实用性。循环方式和查询方式下，程序结构简单易用，　而状态机方式下程序可移植性强。　状态机程序设计不仅可以用在人机接口设计中，还可用于嵌入式通信协议等其他场合中。