一个占用内存极少的菜单系统的实现[1][2]

在各类仪器仪表的设计中，常常需要透过液晶和键盘来实现人机交互，当整个系统需要管理的内容较多的时候，如何透过键盘和液晶实现有效管理便是个问题。通常软件的设计者会设计一个菜单界面来达到这个目的。

本代码便是透过对身边常见的手机的界面做分析，总结，然后通过 C 代码实现类似的界面。

这套代码实现的菜单的特点：

1．支持 3 种菜单类型

2．支持多国语言

3．占用内存极小

4．全 C 代码，方便移植

5．支持数字按键快捷方式

一些手机界面的分析

常见的手机界面几乎都可以归类为这 3 种：**图标层**，**条项层**，**动态内容层**

* **图标层**

图标层基本特征是：使用图标和标题配合

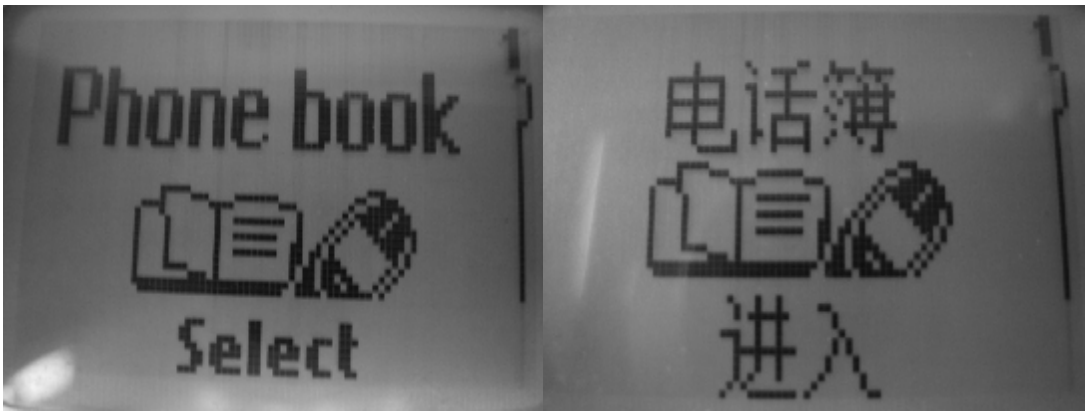


图 1 图标层菜单

图标层通常作为菜单系统的顶层，标题文字可以根据语言选择而更改，但图标通常不变。而且右边有滚动条提示当前位置。底部的一个词作为按键的功能提示。

也见过把滚动条横向放置在屏幕底部的，除了通过滚动条显示当前位置外，也有可能通过编号来提示当前位置。

[1] 作者：梁炎昌 Email:lycld@163.com

[2] 最后更新：2007-9-1

功能提示提示不是必须的，而且功能提示只有在按键安放在液晶附近才能使用，否则操作者根本不知道这个功能提示对应于那个按键。如果硬件设计时，功能提示的优点是显示了对应位置按键的实际功能，方便操作者。

在图 1 中，”Phone book””电话簿”是标题，”Select””进入”是按键的功能提示。

在大部分的手机中，图标不单单是静态的，通常在上下翻的时候，先是图标动画，然后才变成静态图标。

图 2 图标层菜单

在图 2 的左图中除了中间的大图标外，还有右边的一栏小图标，类似滚动条的作用，底部有 3 个按键的功能提示。而图 2 的右图中仅按键的功能提示有两个，图标在屏幕的左边。

当然了，形式可以多变，但本质依然是：**图标＋标题。**

* 条项层

条项层的基本特征是：一行行的文字

图 3 条项层菜单

条项层菜单通过显示一行行的文字作为菜单选择项。通过反显/高亮某行文字来提示该项是当前选中的菜单项。

右边依然可能有滚动条，提示选中的菜单项在整个菜单中位置。也依然可能有数字编号提示选中的菜单项在整个菜单中的位置。使用滚动条和数字编号的目的在于提示菜单项位置，但滚动条和数字编号均不是必须的。尤其在屏幕较小时，或者实现滚动条较复杂时，通常省略。

也依然有可能使用按键的功能提示，但也不是必须的。

在黑白液晶中，通常用反显整行文字来提示，但有时比较耗费 CPU，那么可以在需要提示的菜单项前显示个箭头，以箭头来提示选中的菜单项。移动一个箭头的位置，比取消反显行并反显新一行的代价会小很多，在 CPU 主频不高时可以使用。

图 4　条项层菜单

在彩色屏幕中，高亮显示选中的选择项，方法很多，可以通过改动背景颜色和文字颜色等方法实现。

反显/高亮，或者使用箭头，目的是要突出选中的菜单项。

* 动态内容层

动态内容层由标题和动态内容组成。

标题用来说明当前项的功能，而动态内容就是该功能当前状态。比如"设置时间"，动态内容就显示当前的时间。

标题是固定的，根据语言切换的，而动态内容可能需要调用一个函数得到，然后显示。

动态内容通常是一些设置内容的开关状态等。

依然有可能有滚动条和数字编号，以及按键的功能提示。

动态内容层菜单一样可以显示多个项，这点跟条项层菜单一样。当屏幕可以显示多个

项时，那么依然需要使用跟条项层菜单的方法来提示当前选中的菜单项。而当由于屏幕较小时，仅能显示一个动态内容项菜单，那么不需要提示选中，因为屏幕也就一个项。

图 5 中仅有一项菜单，无须通过反显提示当前菜单项。

图 6 中有 3 个菜单项，要通过高亮的方式提示当前选中的菜单项。

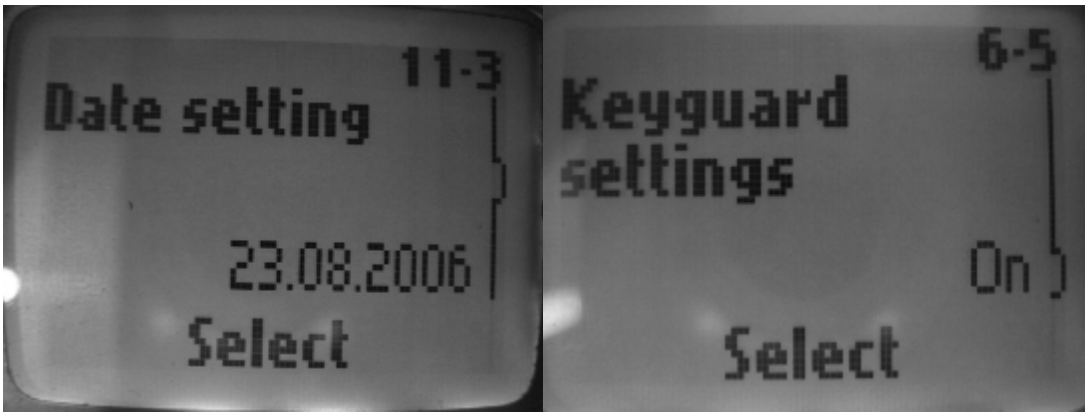


图 5 仅有一项菜单的动态内容层菜单

图 6 有多项的动态内容层菜单

在**条项层**和**动态内容**层也有可能会有本层菜单的标题，只要屏幕足够大。

在条项层和动态内容层菜单中，按键的功能提示可能会依据当前选中的菜单项而改变。

图 7 中由于屏幕够大，显示本层菜单标题。

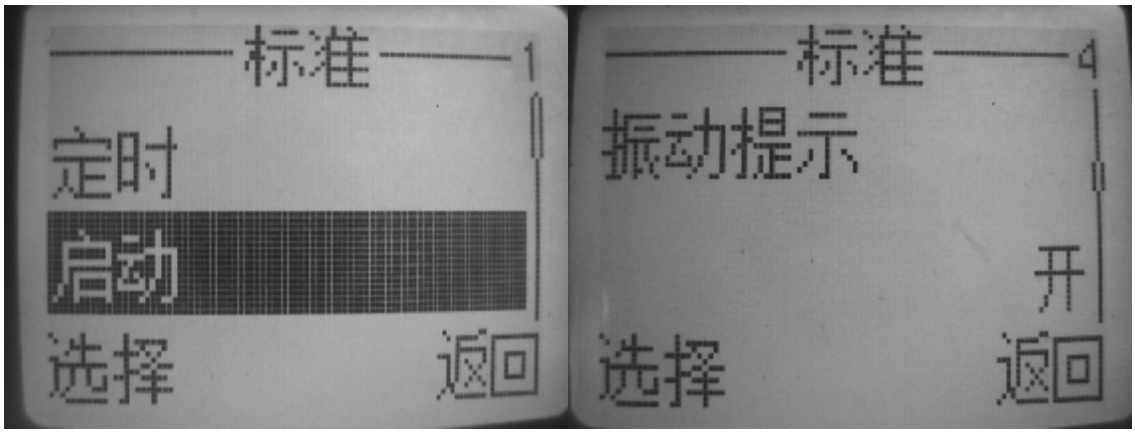


图 7 带标题的菜单

图标层代码实现分析

上下翻转是需要刷新图标和标题，偶尔也需要更改底部的按键功能提示。因此实现方法也就很简单了，根据上下翻转键刷新文字和图标。

滚动条通过当前位置和总的菜单项可以实现。

条项层代码实现分析

分两种情况：

1．要求显示的全部菜单项多于 LCD 可以显示的菜单项的数目。比如，要求显示的项有 6 项，但是 LCD 一次仅仅能显示 3 项，那么就涉及到如何刷新显示的问题了。

对照图 8，可分成如下 3 类情况：

A．顶部上移，当前反显项在 LCD 顶部，上翻。那么就是刷新整个 3 项的显示了，因为 3 项的内容都更改了。

B．底部下移，当前反显项在 LCD 底部，下翻。那么就是刷新整个 3 项的显示了，因为 3 项的内容都更改了。

C．除了 AB 两种之外的都是当前反显项在 LCD 内部移动，取消原来的反显项，根据上下翻来反显当前项。

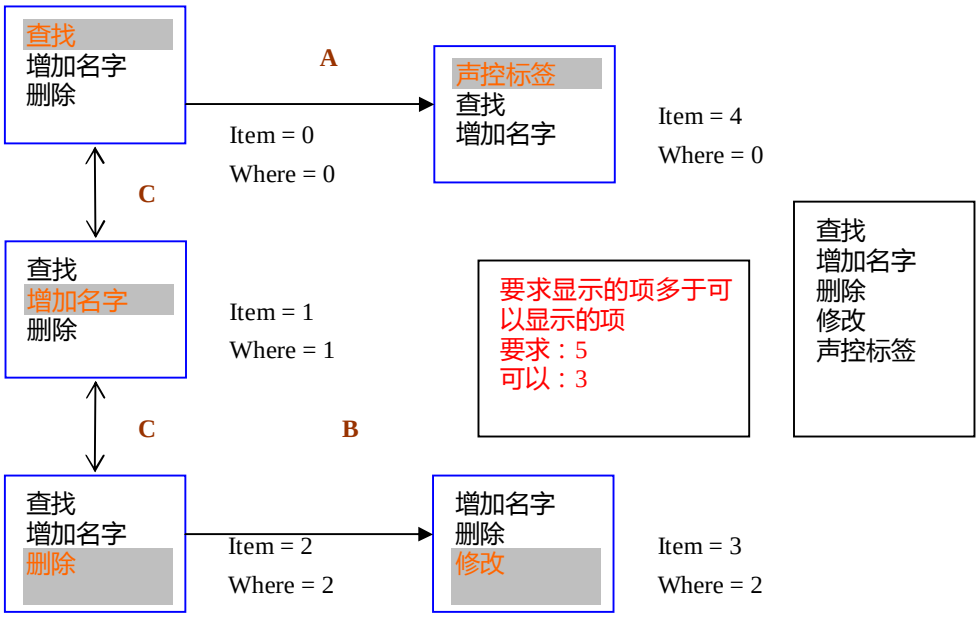


图 8 条项层菜单上下翻的分析

图中 Item 表示当前显示项的编号，Where 表示当前显示项在液晶上的位置。可见上图中 Item 的取值范围是 0 - 4，where 的取值范围是 0 - 2。

2．要求显示的项不多于 LCD 可以显示的项的数目。

对照图 9，有 3 种情况：

A．顶部上移，取消顶部反显，反显底部。

B．底部下移，取消底部反显，反显顶部。

C．除了 AB 两种之外的都是当前反显项在 LCD 内部移动，取消原来的反显项，根据上下翻来反显当前项。

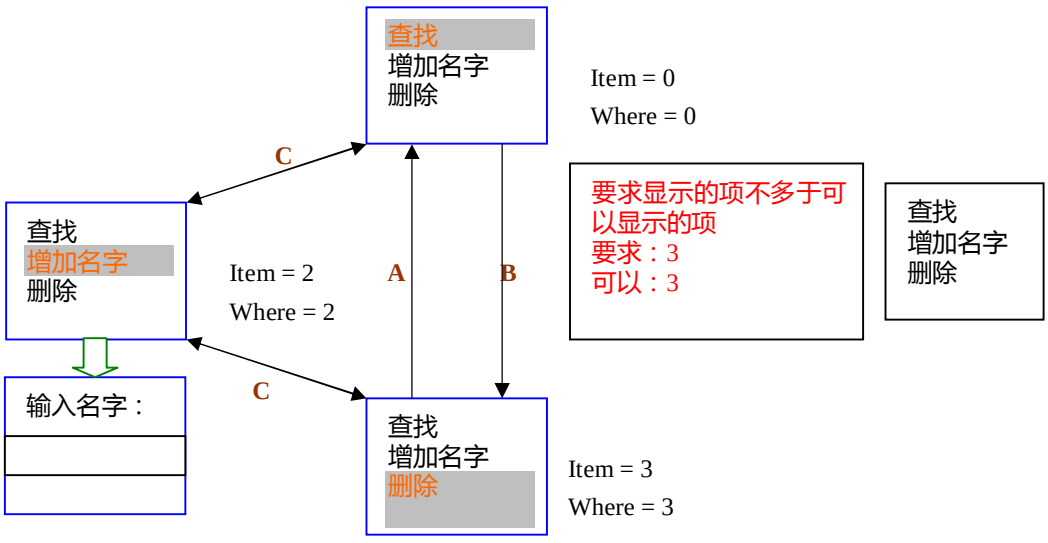


图 9 条项层菜单上下翻的分析

从硬件行为上来分析，在某一层的菜单中，可以有：上翻，下翻，确定，退出这 4 种情况，上翻和下翻是在本层菜单内浏览菜单项，确定是进入选中菜单项的子菜单，退出是退出当前菜单，回到上层菜单。

对于条项层菜单，每一层条项型菜单用 5 个函数来实现，分别是：

表格 1 条项菜单函数

MenuNameInit	初始化本层菜单
MenuNameUp	向上按键处理函数，向上按键默认作为确定键，用来进入子菜单或确定输入。
MenuNameDown	向下按键处理函数，向下按键默认作为退出键，用来返回父菜单或取消输入。
MenuNameLeft	向左按键处理函数，选中上一项菜单项
MenuNameRight	向右按键处理函数，选中下一项菜单项

* ***MenuName*** 是菜单名，层菜单均有一个菜单名。

当有按键输入时，这些函数会被调用，代码中设计了一个 FSM 来管理多层菜单，每一层菜单在 FSM 的一个索引表格中占用 5 个索引号，分别对应于表格 1 的 5 个函数。代码运行时，保存着当前的索引号，当按键按下的时候，代码会在这个索引表格中找到下一个索引号，以此作为当前索引号，同时会调用这个索引号对应的条项菜单函数。

表格 2 索引表中函数和索引号的对应关系

函数	索引号
MenuNameInit	***MenuName***Init_ID
MenuNameUp	***MenuName***Init_ID+1
MenuNameDown	***MenuName***Init_ID+2
MenuNameLeft	***MenuName***Init_ID+3
MenuNameRight	***MenuName***Init_ID+4

当当前状态为表格中的 ***MenuName***Init_ID+1 至 ***MenuName***Init_ID+4 时，是临时状态，所谓"临时状态"是指与这 4 个索引号对应函数在被调用的时候，依然会修改当前索引，***MenuName***Left 和 ***MenuName***Right 会把索引号修改为本层菜单的 ***MenuName***Init_ID，而 ***MenuName***Up 则会修改成某个子菜单对应 ***MenuName***Init_ID，***MenuName***Down 怎会修改成父菜单的 ***MenuName***Init_ID。唯有 ***MenuName***Init_ID 是个恒久的状态。

从上层菜单进入本层菜单后，索引号首先变为本菜单的 ***MenuName***Init_ID，同时会调用 ***MenuName***Init 函数，该函数首先把一些指针修改到本层菜单的资源中，包括条项的文字字符串开始位置，每个字符串长度，本层菜单条项的数目等，然后初始化本层菜单的第一次显示。

当向左按键被按下，根据当前索引号 ***MenuName***Init_ID 索引表格中查找，查处该按键对应的索引号是 ***MenuName***Init_ID+3,于是索引号被修改为 ***MenuName***Init_ID+3，同时调用 ***MenuName***Left 函数，***MenuName***Left 函数首先维护条项的显示：把上一个菜单项作为选中项，代码处理的方法是图 8 图 9 的方式。在 ***MenuName***Left 函数的最后，把索引号修改成 ***MenuName***Init_ID。

向右按键的处理跟向左按键的处理基本一致。

当向上按键被按下，根据当前索引号 ***MenuName***Init_ID 索引表格中查找，查处该按键对应的索引号是 ***MenuName***Init_ID+1,于是索引号被修改为 ***MenuName***Init_ID+1，同时调用 ***MenuName***Up 函数，***MenuName***Up 函数首先调用 PUSH 函数把本层菜单的特征参数保存到一个模拟堆栈中，目的是为了当从子菜单返回本层菜单时，可以把模拟堆栈中保存的特征参数恢复屏幕显示。***MenuName***Up 然后根据当前选中菜单项，修改索引号为该子菜单对应的索引号，并把一个刷新标志 Flash 设置为 1，在下一次键盘扫描中，虽然没有按键按下，但依然会因为 Flash 为 1，而调用该子菜单的 ***MenuName***Init 函数，该函数初始化本层菜单的显示。

对于一个常见的条项型菜单，其实大部分情况每个条项显示内容在编写程序的时候是已经知道的，因此这些显示内容可以放在程序代码段中，并可以通过指针寻址。

因此条项型菜单某个时刻的状态能跟如下这几个参数有关。

DispItem	指向显示内容数组开始位置的指针
Item	当前显示条项在显示内容数组中的偏置
Where	当前显示条项在液晶上的位置

恢复一个菜单原有显示，在得到上面 3 个参数后，便能在屏幕重画出菜单来。所以，在进入子菜单前，为了能从子菜单返回时，恢复本菜单显示，需要对上 3 个参数做备份，而这就是 PUSH 函数的内容。

在某个条项型菜单中，向下按键被按下，***MenuName***Down 函数被调用，该函数调用 Pop 函数，Pop 函数从模拟堆栈中恢复 DispItem、Item、Where 这 3 个参数，并把刷新标志 ReFlash 设置为 1，在下一次键盘扫描中，虽然没有按键按下，但依然会因为 ReFlash 为 1，而调用该子菜单的 ***MenuName***Init 函数，该函数根据恢复的 3 个参数显示本层菜单。

MenuTop.C 中有三个公共函数与之相关：

void BarMenuInit()	实现从父菜单进入本层菜单时的初始化和从子菜单退回到本层菜单的初始化
void BarMenuLeft()	维护按下向左按键时的条项显示
void BarMenuRight()	维护按下向右按键时的条项显示

BarMenuInit 在两个地方被调用：刚刚从父菜单进入到本层菜单时、从子菜单返回到

本层时。刚刚从父菜单进入本层时，Item=0 Where=0 ，代码比较容易写；而从子菜单返回时，那么 Where 就跟上次进入这个子菜单时的情形有关系了（因为可以翻动，因此 Item 项显示在液晶上的位置是可以改变的），既然 where 不太确定，那么我们也只有推算出 where=0 地方的 Item 值了，然后才开始显示，总之是需要处理 Item 的循环显示。

BarMenuLeft 和 BarMenuRight 依照上面的图和分析也是可以实现的。

菜单框架的代码实现

以下内容将分析本框架代码中的有限状态机的周转机理、图标层的实现、条项层的实现。MenuFSM.C 中有一个菜单状态的数据结构。

定义了当前状态索引号、4 个按键状态索引号、执行函数。

```
void (*KeyFuncPtr)(); //按键功能指针
typedef struct{
        U8 KeyStateIndex; //当前状态索引号
        U8 KeyUpState; //按下"向上"键时转向的状态索引号
        U8 KeyDnState; //按下"向下"键时转向的状态索引号
        U8 KeyLState;  //按下"向左"键时转向的状态索引号
        U8 KeyRState;  //按下"向右"键时转向的状态索引号
        void (*CurrentOperate)(); //当前状态应该执行的功能操作
} KbdTabStruct;
```

把这个数据结构例化成一个数组，我们看看这个数组中一些关键的代码：

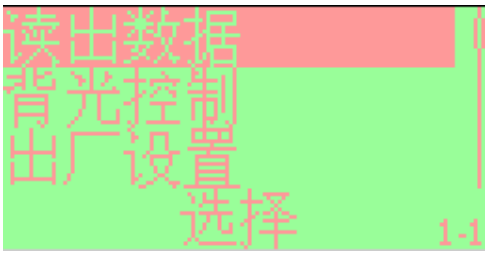
```
_CONST_ KbdTabStruct KeyTab[]={
//      |-----------> Index
//      |  Up
//      |  |  Down
//      |  |  |  Left
//      |  |  |  |  Right   --->功能函数
//      |  |  |  |  |      |
      { 0, 1, 2, 3, 4,(*DispMenuTop)},// 待机画面
      { 1, 0, 0, 0, 0,(*DispMenuTopUp)},
      { 2, 1, 1, 1, 0,(*DispMenuTopDown)},
      { 3, 1, 1, 1, 0,(*DispMenuTopLeft)},
      { 4, 1, 1, 1, 0,(*DispMenuTopRight)},

      { 5, 6, 7, 8, 9,(*DispIcoMenuInit)}, //图标层菜单
      { 6, 0, 0, 0, 0,(*DispIcoMenuUp)},   //
      { 7, 0, 0, 0, 0,(*DispIcoMenuDown)}, //
      { 8, 0, 0, 0, 0,(*DispIcoMenuLeft)}, //
      { 9, 0, 0, 0, 0,(*DispIcoMenuRight)},//

      {10,11,12,13,14,(*DispMenuFuncInit)}, //"功能"
      {11, 0, 0, 0, 0,(*DispMenuFuncUp)},   //
      {12, 0, 0, 0, 0,(*DispMenuFuncDown)}, //
      {13, 0, 0, 0, 0,(*DispMenuFuncLeft)}, //
      {14, 0, 0, 0, 0,(*DispMenuFuncRight)},//

};
```

上面的代码是给出了：待机界面、图标层菜单、条项层菜单"功能"，实际界面如下图：



以以上的代码段和图片，开始实际的分析：

有 4 个按键作为菜单周转按键：向上键、向下键、向左键、向右键。在检查按键输入时根据按键输入查找出下一个状态号和以及这个状态号的执行函数。

Main 程序在一系列初始化后，一致在执行 CheckKey 函数，该函数不断检查按键输入，有按键输入则根据按键分发处理，在退出该函数前检查两个全局变量 ReFlash、Flash，这两个全局变量定义在 Menu.C 中，在 Menu.H 中有声明。

```
U8 ReFlash = 0;            //子项菜单返回 刷新标志
U8 Flash = 0;              //跳转进入子菜单 刷新标志
```

分别作为子菜单返回时，请求刷新的标志和进出子菜单时，请求刷新的标志，一旦这两个标志均不为 0，那么 MenuFSM.C 中的 CheckKey 函数根据当前索引号查询到执行函数并执行之，然后清除这两个标志。

```
   if(Flash|ReFlash){
      KeyFuncPtr=KeyTab[KeyFuncIndex].CurrentOperate;
      (*KeyFuncPtr)();//执行当前按键的操作
      Flash = 0;
      ReFlash = 0;
    }
```

在待机界面下，当前状态索引号是 KeyFuncIndex = 0，CheckKey 函数检查到当按键输入是向上键(Up)时，在状态表中查出向上按键的状态号是 1，更新了当前状态索引号，然后找出状态号 1 的执行函数是 DispMenuTopUp()，于是执行这个函数，然后返回，继续检查按键，继续根据按键输入实行状态周转。

```
//MenuFSM.C  CheckKey()函数代码片段
```

```
    switch(KeyScan()){
        case Key_Up:{ //向上键,找出新的菜单状态编号
            KeyFuncIndex=KeyTab[KeyFuncIndex].KeyUpState;
            KeyFuncPtr=KeyTab[KeyFuncIndex].CurrentOperate;
            (*KeyFuncPtr)();//执行当前按键的操作
            break;
        //-------
        case Key_Right:{ //向右键,找出新的菜单状态编号
            KeyFuncIndex=KeyTab[KeyFuncIndex].KeyRState;
            KeyFuncPtr=KeyTab[KeyFuncIndex].CurrentOperate;
            (*KeyFuncPtr)();//执行当前按键的操作
            break;
        }
        default: //按键错误的处理
            IsKey = 0;
            break;
    }
//两个当中有任意一个不为零，那么执行一次，我们应该不会出现 Flash 和 ReFlash 都是 1 的
情况
    if(Flash|ReFlash){
        KeyFuncPtr=KeyTab[KeyFuncIndex].CurrentOperate;
        (*KeyFuncPtr)();//执行当前按键的操作
        Flash = 0;
        ReFlash = 0;
     }
    return IsKey;
```

上面描述到程序会执行 DispMenuTopUp()函数，那么我看看这个函数里到底做了什么事情。

```
//待机界面下 Up 键的处理
void DispMenuTopUp(void)
{
//-------------------
    KeyPressCount = 0;
//-------------------
    GUI_Clear();
//-------------------
//这里是顶层菜单,因此在进入子菜单前需要初始化变量
    InitMenuVal();
//--------------------------
//状态跳转,进入子菜单
    //jump to Menu index
    Jump2Menu(MenuIndex,FlashMode_AutoInit);
    return;
}
```

该函数清屏后调用了 InitMenuVal 函数，然后由 Jump2Menu 修改当前状态索引号为 MenuIndex，MenuIndex 是宏定义，值是 5，置 Flash 为 1，最后返回。InitMenuVal()函数初始化了一些全局变量，然后备份了待机界面的状态索引号。

```
//初始化全局变量
void InitMenuVal(void)
{
    WhereBackup_i = 0;
    ItemBackup_i = 0;
    Layer = 0;
    FatherIndex[Layer++] = MenuTopIndex;  //push index  待机界面作为起点
}
```

DispMenuTopUp()在 CheckKey()中被调用之后，Flash 变为了 1，那么如上所述 CheckKey()函数发现 Flash 为 1，会根据当前所引号找出执行函数来执行的，当前索引号在 DispMenuTopUp 调用后变成了 5，而 5 的执行函数就是 DispIcoMenuInit()，图标层的初始化函数。继续追踪代码：

```
void DispIcoMenuInit()
{
   ItemNum = IcoMenu[language].TextNum;//多少数据项
   DispItem = IcoMenu[language].Text;  //数据
   GUI_Clear();
//-------------------
   if(Flash == FlashMode_AutoInit){//从 0 进入,初始化 Item 等值
      PUSH();
      Item = 0;
      Where = 0;
   }
   //菜单名字
   GUI_DispStringAtBar(*(DispItem+Item),0,ICO_PosY+ICO_YSize,126,GUI_TA_HCENTER);
   GUI_DrawIcon(icos[Item],ICO_PosX,ICO_PosY);           //图标
   //显示按键对应的文字提示,1-->"进入"
   GUI_DispStringAtBar(*(Tip[language].Text+1),0,Enter_PosY,126,GUI_TA_HCENTER);
   Bar(Item,ItemNum,MenuBarPosX,MenuBarNumPosX);               //维护滚动条
   return;
}
```

发现该函数显示了菜单名字和图标等。界面到了下图：

需要留心的是这个函数中执行了一个 PSUH()的函数，该函数的功能是备份当前菜单的相关参数，为从子菜单返回父菜单时，能还原原有显示而服务。

在图标层下当前索引号是 MenuIndex=5，图标层翻动时是切换不同的文字显示和图标的，并且维护 Item 变量。函数 DispIcoMenuLeft()和函数 DispIcoMenuRight()的最后都是调用 Jump2Menu(SelfMenuIndex,FlashMode_NoAction)；FlashMode_NoAction 是通知代码不要做刷新屏幕的动作，而宏 SelfMenuIndex 的定义在 Menu.H 中如下：

```
   #define SelfMenuIndex FatherIndex[Layer-1]
```

也就是把当前状态索引号修正为备份数组 FatherIndex 里的最后一个。

```
void DispIcoMenuLeft()
{
   if(Item==0){
      Item = ItemNum-1;
   }else{
      Item--;
   }
   GUI_DispStringAtBar(*(DispItem+Item),0,ICO_PosY+ICO_YSize,126,GUI_TA_HCENTER);
   //菜单名字
   GUI_DrawIcon(icos[Item],ICO_PosX,ICO_PosY);           //图标
   Bar(Item,ItemNum,MenuBarPosX,MenuBarNumPosX);           //维护滚动条
```

```
    Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
}
```

在这里 Jump2Menu(SelfMenuIndex)的调用使得 KeyFuncIndex = 5，因为 PUSH（）函数的调用把 MenuIndex=5 备份了。也就是说不管如何翻动，当前状态索引号依然是 KeyFuncIndex=5，CheckKey()函数一直根据状态 5 来查询，分发。

```
      { 5, 6, 7, 8, 9,(*DispIcoMenuInit)}, //图标层菜单
      { 6, 0, 0, 0, 0,(*DispIcoMenuUp)},    //
      { 7, 0, 0, 0, 0,(*DispIcoMenuDown)}, //
      { 8, 0, 0, 0, 0,(*DispIcoMenuLeft)}, //
      { 9, 0, 0, 0, 0,(*DispIcoMenuRight)},//
```

当在图标层的按键输入是 Up 时，执行的函数将有所不同，查看 DispIcoMenuUp()函数就了解细节了。

```
void DispIcoMenuUp()
{
   switch(Item){
      case 0:{
         GUI_Clear();
         //功能
         Jump2Menu(MenuFuncIndex,FlashMode_AutoInit);
         return;
      }
      case 1:{
         GUI_Clear();
         //参数
         Jump2Menu(MenuParaIndex,FlashMode_AutoInit);
         return;
      }
      case 2:{
         GUI_Clear();
         //测量
         Jump2Menu(MenuMeasureIndex,FlashMode_AutoInit);
         return;
      }
      case 3:{
         GUI_Clear();
         //语言
         Jump2Menu(MenuLanguageIndex,FlashMode_ManualInit);
         //在 MenuLanguage.C 中的 DispMenuLanguageInit 函数中修改 Item Where!
         return;
      }
      case 4:{
         Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
         return;
      }
      case 5:{
         Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
         return;
      }
      default:{
         Jump2Menu(MenuTopIndex,FlashMode_NoAction);
         return;
      }
   }
}
```

该函数根据 Item 的值由 Jump2Menu 函数修正当前状态索引号 ,并把 Flash 置为一个大于 0 的值，然后返回。我们已经知道 CheckKey()函数一旦发现 Flash 大于 0，便会根据当前索引号执行对应的功能函数的，因此，若是在"功能"下按了 Up 键，那么状态跳转到 MenuFuncIndex=10。检查 MenuFSM.C 中的表格：

```
      {10,11,12,13,14,(*DispMenuFuncInit)}, //"功能"

      {11, 0, 0, 0, 0,(*DispMenuFuncUp)},   //
      {12, 0, 0, 0, 0,(*DispMenuFuncDown)}, //
      {13, 0, 0, 0, 0,(*DispMenuFuncLeft)}, //
      {14, 0, 0, 0, 0,(*DispMenuFuncRight)},//
```

于是执行了 DispMenuFuncInit()函数。

```
//Bar 型菜单
void DispMenuFuncInit()
{
   GUI_Clear();
   BarMenu = &MenuFunc[language];
   ItemNum = (*BarMenu).TextNum;
   DispItem = (*BarMenu).Text;
   //显示按键对应的文字提示,1-->"选择"
   GUI_DispStringAtBar(*(Tip[language].Text+1),0,Enter_PosY,126,GUI_TA_HCENTER);
   //用户定义的初始化代码请放在这里
   BarMenuInit();//调用公共初始化的代码
}
```

该函数先清屏，依据当前语言修改菜单结构指针 BarMenu ;然后依据修改后的指针得到跟菜单显示有关的 2 个变量 ItemNum DispItem；显示按键提示词；最后调用了 BarMenuInit()初始化整个条项层菜单。然后就看到了"功能"的条项层菜单。

需要注意 BarMenuInit 函数开始的部分代码：

```
if(Flash == FlashMode_AutoInit){//常规进入 做备份
   PUSH();//在修改 Item Where 之前备份这些参数数据!!
   Item = 0;
   Where = 0;
} else if (Flash == FlashMode_ManualInit){
//非常规进入  在之前已经备份了，这里做边界检查
   if(Where > DispMin-1){//检查是否出界
      //Where = DispMin-1;//A.最底部的显示位置
      Where = 0;         //B.最顶部的显示位置
   }
   if(Item > ItemNum-1){//检查是否出界
      //Item = ItemNum-1;//A.最后的一个项
      Item = 0;         //B.最开始的一个项
   }
}
```

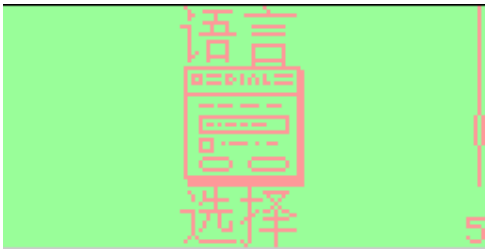
Flash 的值由父菜单的 Up 按键函数（这里是 DispIcoMenuUp 函数）设置，在本菜单中检查，通常我们只会把 Flash 设置为 `FlashMode_AutoInit`，因此调用 PUSH 函数

备份当前状态号（这里当前状态号是 10 ）。

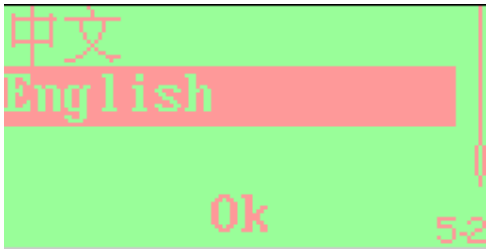
而如果父菜单的 Up 按键函数设置 flash 为 FlashMode_ManualInit。那么这个 Init 函数将不太一样。看语言选择的 Init 函数。

```
void DispMenuLanguageInit()
{
   GUI_Clear();
   //切换 Bar 显示内容
   BarMenu = &MenuLanguage[language];
   ItemNum = (*BarMenu).TextNum;
   DispItem = (*BarMenu).Text;
   //用户的初始化代码请放在这里
   //显示父菜单名字
   //显示按键提示         2-->OK
   GUI_DispStringAtBar(*(Tip[language].Text+2),0,Enter_PosY,126,GUI_TA_HCENTER);
   if(Flash == FlashMode_ManualInit){//首次进入本层菜单，调整 Item where
      PUSH();
      Item = language;
      Where = Item;
   }
   //公共初始化部分
   BarMenuInit();
}
```

在进入该函数前，函数 DispIcoMenuUp 已经把 flash 设置为 2，函数 DispMenuLanguageInit 将依据当前语言调整首次进入时，高亮项的位置，比如当前语言是中文，那么进入语言菜单时是这样：



而如果当前语言是英文，那么进入语言菜单时是这样：



FlashMode_ManualInit 的设置使得代码可以控制进入菜单时高亮在哪个项上，对于语言选择等一些方面有一定提示的作用。

回到原来的分析中，我们接下来开始分析在条项层菜单下，4 个按键的全部行为，条项层的显示内容基本上时固定的不变的，一个条项层的属性可以归结为:

```
DispItem  指向显示内容数组开始位置的指针
Item      当前显示条项在显示内容数组中的偏置
```

```
Where     当前显示条项在液晶上的位置
```

由 DispItem 和 Item 就知道了需要显示的字符串，根据 Where 我们可以显示该字符串在液晶上。

在条项层菜单中翻动时函数 ***MenuName***Left 和函数 ***MenuName***Right 分别调用 BarMenuLeft()和 BarMenuRight 来维护的 Item 和 Where。

```
void DispMenuFuncLeft()
{
   BarMenuLeft();
}
void DispMenuFuncRight()
{
   BarMenuRight();
}
```

在本层菜单中按向左键后，MenuFuncIndex 根据查表，由 10 变为临时的 13，执行 DispMenuFuncLeft 函数，该函数调用的 BarMenuLeft 在退出前，调用 Jump2Menu(SelfMenuIndex)，把当前状态号重新改回 10。

按向右键后，MenuFuncIndex 根据查表，由 10 变为临时的 14，执行 DispMenuFuncRight 函数，该函数退出前，该函数调用的 BarMenuRight 在退出前，调用 Jump2Menu(SelfMenuIndex)，把当前状态号重新改回 10。

也就是说不管是函数 BarMenuLeft 还是函数 BarMenuRight ，在其末尾均调用 Jump2Menu(SelfMenuIndex);来恢复到 DispMenuFuncInit 函数所在的索引号 10。

因此不管按向左向右键都会回到 MenuFuncIndex=10 来重新分发的。

当按键是向上时，调用 DispMenuFuncUp，根据 Item 号来修改当前状态索引号，进入一个子菜单中；或者根据 Item 号来执行实际的功能代码。代码跟函数 DispIcoMenuUp 相似。

当按键是向下时，调用 DispMenuFuncDown，调用 POP，得到原来已经压入堆栈的父菜单的状态索引号，并把 ReFlash 置为 1，通知 CheckKey()调用函数恢复父菜单的显示。

至此，图标层、条项型菜单分析已经很完整。

模版代码讲解

在 MenuPara.C 中有常见用法的模版代码：

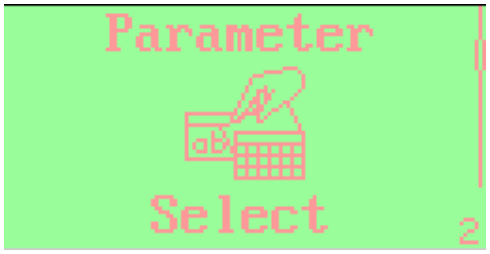
```
/*
*********************************************************************************
菜单使用例程
0.Jump2Menu + FlashMode_AutoInit/FlashMode_ManualInit
  跳转到任意菜单,FlashMode_AutoInit-->在 BarMenuInit 函数中执行 PUSH;
  FlashMode_ManualInit-->子菜单的 Init 函数中 PUSH,看例子:DispMenuLanguageInit
1.PUSH + POP
  执行一些功能,并使用 LCD 后,回到原有界面,回到原有界面--包括反显位置都恢复
2.Jump2Menu(SelfMenuIndex,FlashMode_ReFlash);
  执行一些功能,并使用 LCD 后,执行如上调用,回到原有界面--包括反显位置都恢复
```

```
3.POP
  执行一些功能,并使用 LCD 后,回到上层菜单,恢复上层界面
4.Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
  执行一定功能后,但不占用 LCD 的话,直接界面不动

*****************************************************************************
*/
void DispMenuParaUp()
{
   switch(Item){
      case 0:{//Jump2Menu
         GUI_Clear();
         GUI_DispStringAt("Do Something \nthen Back to \nTop Menu!",0,0);
         GUI_Delay(4000);
         Jump2Menu(MenuTopIndex,FlashMode_AutoInit);
         break;
      }
      case 1:{//PUSH  -->POP
         PUSH();
         GUI_Clear();
         GUI_DispStringAt("Do Something \nthen Back to \nThis Menu!",0,0);
         GUI_Delay(4000);
         POP();
         break;
      }
      case 2:{//ReFlash = 1
         GUI_Clear();
         GUI_DispStringAt("Do Something \nthen Back to \nThis Menu!",0,0);
         GUI_Delay(4000);
         Jump2Menu(SelfMenuIndex,FlashMode_ReFlash);
         break;
      }
      case 3:{//POP
         GUI_Clear();
         GUI_DispStringAt("Do Something \nthen Back to \nFather Menu!",0,0);
         GUI_Delay(4000);
         POP();
         break;
      }
      case 4:{//No ReFlash
         //No Action Here!
         //Or The Action no need the LCD Disp
         Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
         break;
      }
      default:{
         Jump2Menu(SelfMenuIndex,FlashMode_NoAction);
         return;
      }
   }
}
```

以下给出执行效果图。

0：执行一些函数调用后返回待机界面（当然也可以跳转到任意一个菜单）。

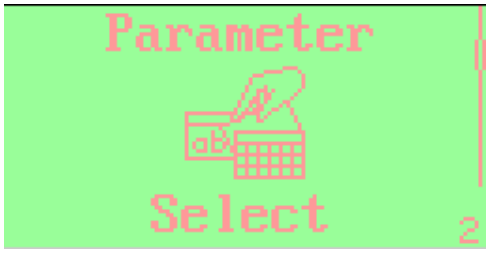


à


à


à


1 2:执行一些函数调用后返回本菜单，原有界面被恢复。

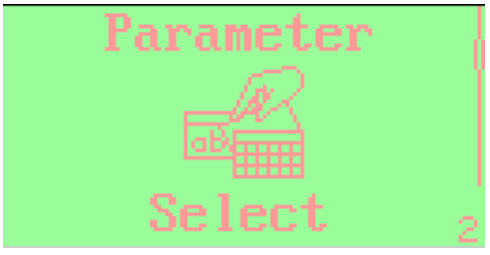


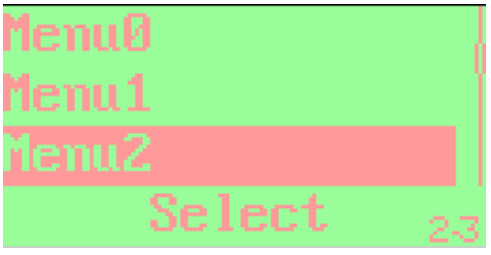
à

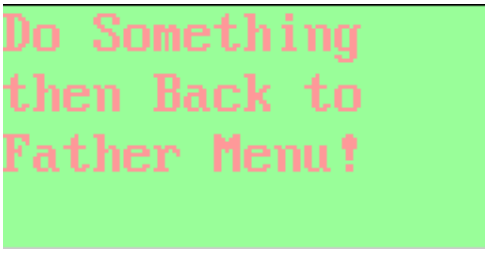

à


à


3．执行一些函数调用后，回到本菜单的父菜单。



à


à


à


动态内容层菜单

由于动态内容层代码不是很完善，因此不在这里分析，如果需要了解看MenuClock.C，里面有一个可用的原型。

à


à


简单讲讲思路：

函数 DispMenuClockLeft()和函数 DispMenuClockRight()在维护 Item 后，调用函数 FlashDisp()，该函数根据 Item 显示标题，并调用函数得到 Item 对应的动态内容，显示之。也要留意函数 DispMenuClockInit()里面是如何显示显示第一次的内容的，又是如何在进入子菜单后，刷新显示的。

动态生成条项菜单内容

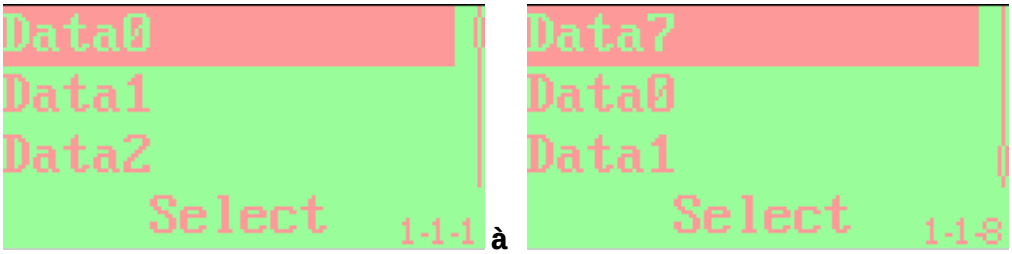
在搭建的代码框架中，条项菜单的显示内容通常是固定不变的，但也有些场合需要动态产生这些条项内容，比如说手机短信收件箱里，就是依照收到的短信的发送人名字生成条项的。

在 MenuFunc.C 中有一个动态生成条项的例子代码，由于内容的动态产生因此需要一些 RAM 来存放动态项。

```
#define DataItemSize 8
U8 DataItem[DispMax][DataItemSize]=
{
{"DataX  "},
{"DataX  "},
{"DataX  "},
};
U8 *_CONST_ DataItem_p[]=
{
DataItem[0],
DataItem[1],
DataItem[2]
```

```
};
```

该范例中，当上下翻转时，void DispMenuDataLeft()和 void DispMenuDataRight()动态修改 DataX，X 是一个数字。依照一样的原理，可以在这两个函数中完成得到动态显示内容，然后修改显示缓冲 DataItem。



动态生成条项菜单要占用的 RAM 大概就是 DataItem 数组占据的内存大小，这个大小跟 LCD 一屏可以显示多少个条项 **DispMax** 有关、跟一个条项内容的字符大小 **DataItemSize** 有关。

数字快捷方式的实现

数字快捷方式的意思是根据条项的编号，直接用数字键输入该编号，然后跳转到该子菜单中，在 Nokia 手机中这样的功能，只要添加如下几行代码到 MenuFSM.C 的 U8 CheckKey(void)函数中便能实现这种功能。

当然该功能需要 Key = KeyScan();能返回数字按键的键值。在 Menu.H 中有对应的宏开关来打开/关闭该功能。

```
/*
2006/09/08
数字键做快捷方式输入
1.判别数字键值是否小于 ItemNum-1 否则是无效快捷方式
2.有效快捷方式下，把键值给 Item，调用 Key_Up 的处理代码
3.需要添加超时处理，超时了就不能使用？（需要么？不需要么？）
*/
       case Key_1:
       case Key_2:
       case Key_3:
       case Key_4:
       case Key_5:
       case Key_6:
       case Key_7:
       case Key_8:
       case Key_9:
          Key = chang_code(Key);
          if((Key - '1') <= ItemNum-1){
             Item = Key - '1';
             //如果需要显示的项比可以显示的项少，那么修正 Where 否则默认 Where 为 0
             if(ItemNum <= DispMax){
                Where = Item;
             }else{
                Where = 0;
```

```
            }
            //-------------------------------------------------
            KeyFuncIndex=KeyTab[KeyFuncIndex].KeyUpState;
            KeyFuncPtr=KeyTab[KeyFuncIndex].CurrentOperate;
            (*KeyFuncPtr)();//执行当前按键的操作
            //-------------------------------------------------
        }else {
            IsKey = 0;
        }
        break;
```

菜单函数调用图

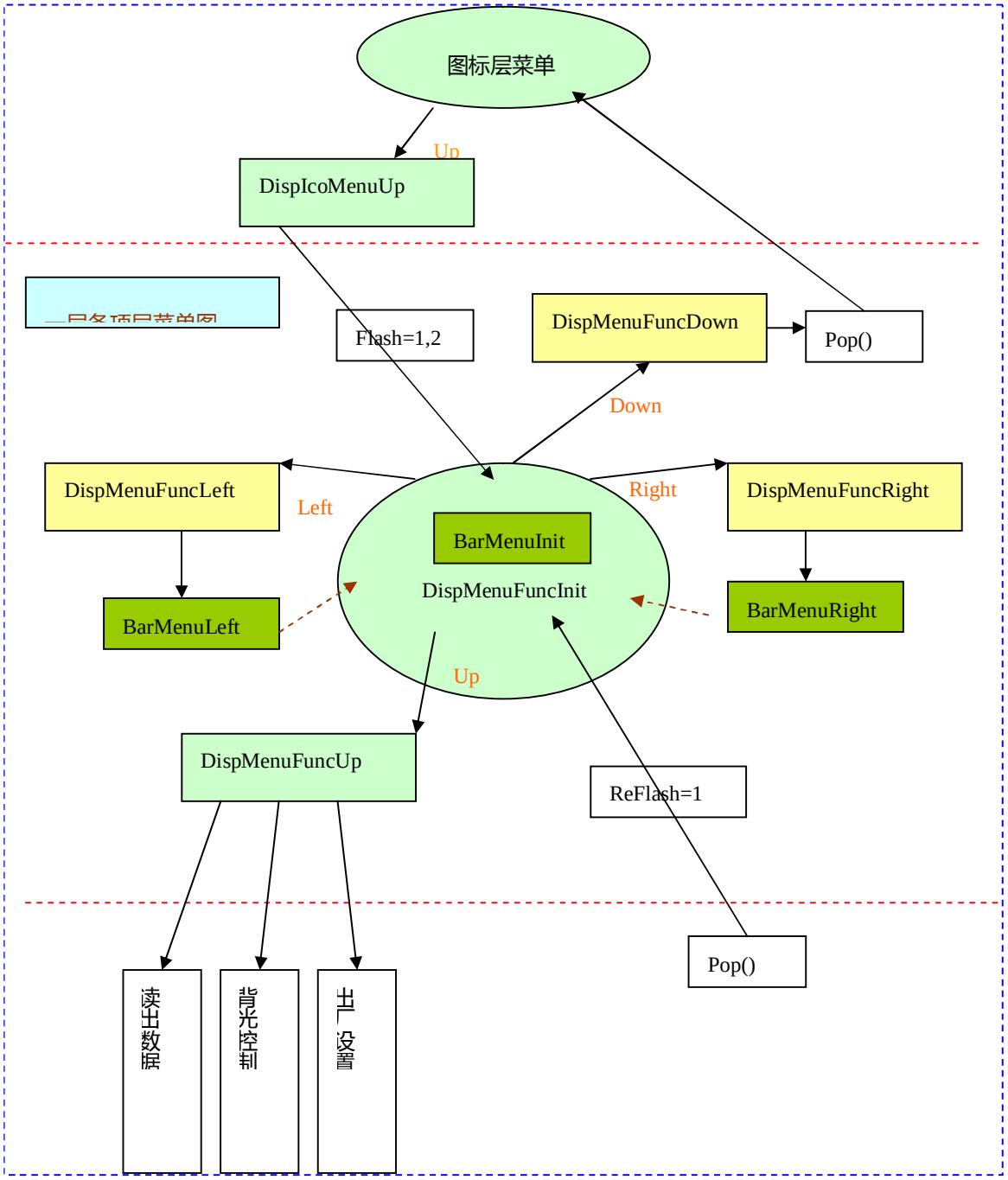


图 10 函数调用图

问答 FAQ

这个东西哪来的?

2005 年 3 月份是从事一个手持仪器的开发，那时发觉系统需要管理的内容很多，为了有效管理，决定设计一个菜单程序，那时我的手机是 Nokia3315，于是开始研究这个手机的行为，思考实现方法，项目完工后，便有了这个第一个版本，后来把很多身边朋友的手机都认真的看了个遍，思考它们的异同、实现机理，然后再做了许多代码优化和修改，使得更易于移植，更有通用性。

对很多手机菜单,不单单手机的菜单，很多仪器菜单，留心观察后，觉得文章中的总结是基本上对的，而提出的实现方法也可行，可以适应多种情况。当然这只是我想到的一个实现的方法，相信也有其它可以实现的方法。

其中关于 FSM 的那部分代码 ,最早在 2003 年刚开始学习 C51 时在 C51BBS 上见过类似的，不过当时没有看懂，我这里也是借用了思路，我想我的贡献应该是实现条项函数和菜单周转时 PUSH、POP 的代码等。

能在 51 这种低速 CPU 上跑不?

这个程序的代码原本是在一个使用 SM89C58 的项目上使用的，内存仅仅 256Byte，Flash32K byte 汉字字模作为代码段一起烧在 Flash 中，整个系统内存耗损是 154byte，其中跟菜单代码有关的内存耗损是 50Byte，支持菜单深度 8 级。

所以绝对可以跑的。要提醒的时 C51 对于函数指针的，可能有一个 L13 的警告：

```
*** WARNING L13: RECURSIVE CALL TO SEGMENT
    SEGMENT: ?CO?MAIN
    CALLER:   ?PR?FNDATMNG?MAIN
```

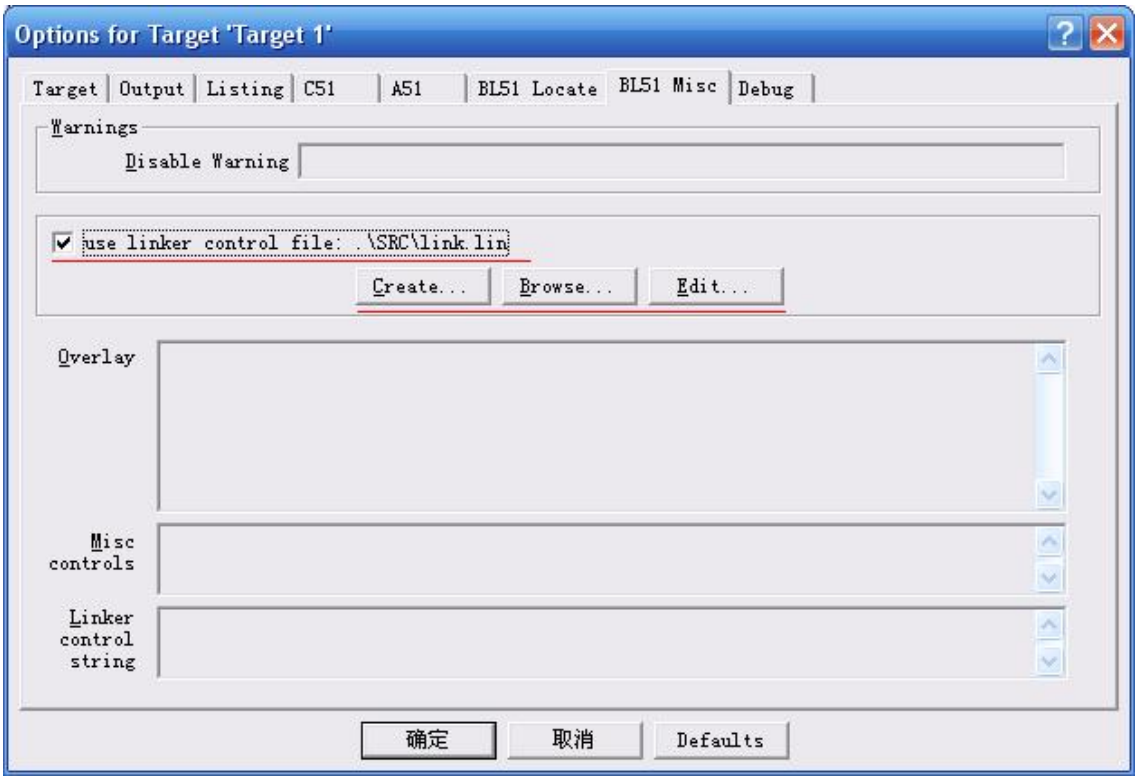
警告的内容大概是编译器无法正确分析函数调用关系。因此需要做手工 OVERLAY 调整。这个问题在 Keil C51 的 C51.PDF (在 keil\C51\HLP\下有)末尾一章 **Appendix F. Hints, Tips, and Techniques** 中有详细解释的，当然也有解决办法了。

1．创建 link.lin 文件，该文件是个文本文件。类似的内容如下：

```
PRINT(".\LST\Master.m51") RAMSIZE(256)
OVERLAY(
?CO?MAINMENU~(DispMenuTopUp),check_key !(DispMenuTopUp),
?CO?MAINMENU~(DispMenuTopDown),check_key !(DispMenuTopDown),
?CO?MAINMENU~(DispMenuTopLeft),check_key !(DispMenuTopLeft),
?CO?MAINMENU~(DispMenuTopRight),check_key !(DispMenuTopRight),
```

```
?CO?MAINMENU~(DispMenuInit),check_key !(DispMenuInit),
?CO?MAINMENU~(DispMenuUp),check_key !(DispMenuUp),
?CO?MAINMENU~(DispMenuDown),check_key !(DispMenuDown),
?CO?MAINMENU~(DispMenuLeft),check_key !(DispMenuLeft),
?CO?MAINMENU~(DispMenuRight),check_key !(DispMenuRight),
)
```

2．Keil 的 Option 下：BL51 Misc 中选上该 link.lin 文件

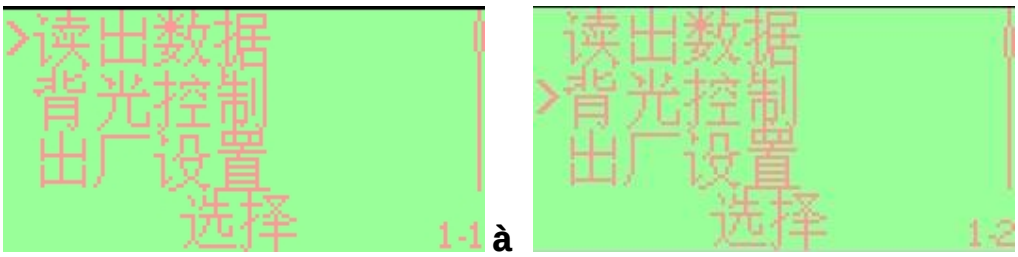


3．Link.lin 的具体内容怎么写，请看 C51.PDF 中的解释吧。

我担心我的 CPU 速度不够快，能用这个菜单不？

CPU 速度不快，导致的结果是刷新液晶慢，因此写代码的时候可以考虑优化写液晶的代码上，比如尽量减少使用清空全屏，仅仅清除使用了的部分（当然这个就要求你知道你已经使用了液晶的那些部分）

在条项层菜单显示中，由于反显的是整个行的文字，若是 CPU 太慢，可以采用如下一个变通的方式：在条项前显示一个箭头作为指示符号，箭头所在项为当前项，翻动的时候移动这个箭头，仅仅在出现"顶部上移"^"底部下移"才是刷新全屏。这样的做法可以减少刷新条项的工作量——毕竟刷新一个箭头总比刷新整行的文字要省事很多。



需要修改 void BarMenuLeft()和 void BarMenuRight()函数，请仔细阅读这两个函数的代码和注释。留意代码处理了大概两类情况：1。需要修改屏幕上所有菜单项显示的。 2。取消原有反显，反显新位置的。在 1 的情况下，几乎要更新全屏幕，没什么办法优化代码。在 2 的情况下就可以单单移动箭头了，毕竟工作量小很多。

- ü 代码已经实现，由一个宏 **LessCPUUseage** 来控制，如果需要使用该效果，Menu.H 中打开宏定义即可。

```
#define LessCPUUseage
```

这个菜单框架的内存耗损大不不大？

下面是内存具体列表：

变量名	RAM	值域
ReFlash	1	-->0/1
Flash	1	-->0/1
language	1	-->0/1/....
KeyFuncIndex	1	-->0/.../255
FatherIndex	9	-->N+1
Layer	1	0/..../1
ItemBackup	8	N
ItemBackup_i	1	0/..../1
WhereBackup	8	N
WhereBackup_i	1	0/..../1
OnlyYou	1	
BarDataMode	1	
Item	1	
ItemNum	1	
Where	1	
DispMax	1	
DispMin	1	
Menu_i	1	
Menu_j	1	
DispItem	2	
Size	1	
BarMenu	2	
TipBar	2	

```
                              -------------->Menu.C
    KeyFuncPtr                        2   --->MenuFSM.C
50
-----------------------------------------------------------

N 菜单深度

N=8
3N+1+X=50
--->X = 25
3N+25
```

菜单的内存耗损计算公式是：**3N+25**，N 为菜单深度，当菜单深度设置为 8 时，内存占用是 50byte。

做菜单极少会做很深的菜单，通常是展宽某一层菜单，展宽菜单并不额外占用 RAM 资源（当然你为子菜单额外写的函数就可能占用 RAM）。

也要注意这里计算的是这个菜单框架占用的 RAM 的资源大小，而没有提及一个体统中别的函数代码占用 RAM 情况，不要把 **3N+25** 当成你最后用到 RAM 的大小！

怎么移植?

菜单系统跟键盘有关系，跟 LCD 有关系，所以移植的难点集中在这两个硬件驱动上面。

unsigned char KeyScan(void)

键盘扫描函数，该函数在 ***MenuFSM.C*** 中被函数 **U8 CheckKey(void)**调用，**KeyScan** 函数检查一次键盘，如果有有效按键按下，需要返回按键的扫描键值 **Key_Up Key_Down Key_Left Key_Right。**

void GUI_Delay(unsigned int Period)

软件延时函数，该函数完成一定时间的延时功能。

void GUI_Init(void)

LCD 初始化函数，初始化 LCD 控制器等等。

void GUI_Clear(void)

液晶清屏函数，清空 LCD 整个屏幕的显示。

void GUI_DispIcon(unsigned char const *Ico,unsigned char X,unsigned char Y)

显示图标函数，该函数在指定位置显示一个图标。

*Ico　图标的字模指针

X　图标显示位置 X 坐标

Y　图标显示位置 Y 坐标

void GUI_DispStringAtBar(unsigned char const *s,unsigned char x0, unsigned char y0, unsigned char x1,unsigned char Mode)

条项显示函数，该函数在一个指定位置显示条项文字。

***S** 条项文字的字符串

x0 y0 条项显示的开始坐标点

x1 条项显示末尾的 x 坐标

Mode 对齐方式，左对齐：条项文字在 **x0**-**x1** 之间左对齐显示，并清空余下部分；右对齐显示：条项文字在 **x0**-**x1** 之间右对齐，并清空余下部分；居中：条项文字在 **x0**-**x1** 之间居中显示，并清空余下部分；

unsigned char GUI_SetTextMode(unsigned char TextMode)

设置文字显示模式函数，该函数设置当前系统文字的显示模式，仅仅支持文字正显/反显两种模式。

void GUI_DispStringAt(unsigned char const *s,unsigned char X,unsigned char Y)

字符串显示函数，在指定位置显示一个字符串。

不必含糊！事实上你需要的大概仅仅是如下这样的一个函数：

void GUI_DispStringAt(unsigned char const *s,unsigned char x0,unsigned char y0,unsigned char x1,unsigned char Mode,unsigned char Reverse)

该函数实现字符串显示，实际上时整合了 **GUI_DispStringAtBar** 和 **GUI_TextMode** 函数。可以指定一个显示位置，也可以指定文字对齐方式，还有是否反显显示文字。

文字对齐方式是为了更加美观而做的，一般在显示条项是都基本上是左对齐方式，而显示按键的提示词时却是居中显示的，当显示的项是当前项时，我们需要突出显示它，因此可以通过反显显示来实现。

该函数得到显示内容的字符串，根据字符内码显示到液晶上，实现多国语言的关键便是这个函数，函数需要根据字符内码得到该字符的液晶字模数据，对于使用 ASCII 码的语言来说，只有 26 个字母，需要的字模数据不多，实现起来比较简单；而对于象中文韩文等文字来说就比较麻烦了，由于汉字的数目很多，如果把整个汉字都取字模的话，数据量很可观的。如果做过液晶显示的一定知道如何处理中文显示的问题的，简单说仅仅对那些需要用到的汉字取字模，并做一个简单的检索表，这个表可以根据内码输入检索到该内码字模数据。如果你能做中文显示了，那么对于其他语言，如韩文日文，情况类似，也是根据内码得到字模数据，然后显示。当然如果要是用 Unicode 的话更好了。可以避免多种语言编码的冲突。

移植的时候也要留意 MenuTop.C 中的一个函数 BarDisp

```
/*
*************************************************************************
* BarDisp - Bar 型菜单显示
* DESCRIPTION: -
*
* @Para s:BAR 显示的文字内容
* @Para X:X 轴坐标
* @Para Y:Y 轴坐标
```

```
* @Para HighLight:1--->高亮显示当前项  0--->普通显示当前项
//HightLight = 1--->HightLight Disp
//HightLight = 0--->Normal Disp
* Return :
*
*********************************************************************
*/
void BarDisp(U8 code *s,U8 X,U8 Y,U8 HighLight)
```

代码中垂直滚动条的一个函数 Bar()，如果你的代码无需，可以不实现。

```
void Bar(U8 Item_,U8 ItemNum_,U8 BarPosX,U8 BarNumPosX)
```

该函数显示一个滚动条和滚动条下方的菜单位置索引号。

但是依然建议你能实现一个类似的函数来提示用户当前项的位置。如 Bar 函数中的部分代码实现了索引号的显示。

```
//显示历史索引号
   if(ItemBackup_i > 1){//大于１才是
      for(i = 0; i <ItemBackup_i-1;i++){//最后一个位于１的位置
         Item_ = ItemBackup[ItemBackup_i-1-i]+1;  //从备份数据中得到标号，然后加１显示
         U8_temp = (U8)(Item_%10);//
         GUI_DispCharAt(U8_temp+'0',BarNumPosX-8*(i+1),BarNumPosY);
         Display_Locate(0x10, BarNumPosX-8*(i+1)+8-1, BarNumPosY); //描分隔符
         Display_Locate(0x10, BarNumPosX-8*(i+1)+8-0, BarNumPosY);
      }
   }
```

关于移植的建议

1．移植者应当熟悉液晶的使用，先完成一个通用字符串显示函数，也就是：

```
void GUI_DispStringAt(unsigned char const *s,unsigned char X,unsigned char Y)
```

当然也包括 LCD 的初始化和清屏函数。

2．完成键盘扫描函数

```
unsigned char KeyScan(void)
```

该函数也需要测试，以保证能正确返回至少菜单用到的四个按键的键值。

3．完成一个延时函数

```
void GUI_Delay(unsigned int Period)
```

4．完成菜单主循环函数，该函数在 main 函数作好别的初始化后调用，该函数不返回！

```
/*
*********************************************************************
* MenuMainLoop - 菜单主循环
* DESCRIPTION: -
* Main 函数调用该函数，本函数并不会返回
* @Para void:
* Return :
*
*********************************************************************
*/
void MenuMainLoop(void)
{
   GUI_Init();
   GUI_Clear();
   DispMenuTop();  //显示待机界面
   while(1){
```

```
        if(CheckKey()){//检查菜单
           GUI_Delay(300);
        }
        if(!KeyFuncIndex){//KeyFuncIndex 为 0 时是待机界面
            //这里书写待机时需要完成的代码，比如刷新时钟显示
           DispRealTime(0,64);
         }
    }
}
```

杂项

- **如何在开机后直接进入功能函数，在按键后才进入菜单呢？这对于仪器界面来说更为有用，因为一个仪器开机就要开始执行仪器的功能。**
- 我们看一段代码：

```
void MenuMainLoop(void)
{
    GUI_Init();
    GUI_Clear();
    DispMenuTop();  //显示待机界面
    while(1){
        if(CheckKey()){//检查菜单
           GUI_Delay(300);
        }
        if(!KeyFuncIndex){//KeyFuncIndex 为 0 时是待机界面
            //这里书写待机时需要完成的代码，比如刷新时钟显示
           DispRealTime(0,64);
}
}
}
```

待机界面下，KeyFuncIndex= 0，我们根据 KeyFuncIndex 来判别当前 LCD 界面是在待机界面还是在子菜单界面下。这样在待机界面下我们就可以执行那些我们需要的函数了，比如说显示实时时间 DispTime 等等。

- **如何实现中断中使用 LCD?**
- 1.使用消息机制，中断 ISR 发送消息，在 main()的 While(1)循环当中不断检查消息，如果在我们已经进入某个子菜单后，中断到来，While(1)检查到消息，把当前菜单状态 PUSH 备份后根据消息显示内容。比如充电器插入后，发送了消息，那么即便是我们原本已经在了某层菜单，当我们检查到消息后，我们 PUSH 当前菜单，根据消息显示提示，而后 POP，回到原来菜单。但是若是我原来已经在消息编辑状态，它又是如何工作？似乎不行 2.有显示缓冲区，中断时去使用液晶不修改缓冲区，结束后重新显示缓冲　2006/08/20　依然认为应当的实现方式是 1 的方法，通常在有 OS 的环境下，可以通过消息传递，而显示仅仅在一个任务当中。

方法 1 有效的前提是系统依然在 While(1)循环中，否则如何知道消息呢？也就是说系统当中不能有独占 CPU 现象出现，似乎解决的方法是键盘也用消息机制，不管在哪里我

们都是检查消息的¯即便我们的目的是一直等待键盘，我们也检查其他消息！

- **如何实现 Nokia 的数字建快捷方式 比如修改界面为英文 :6-2-1-2 然后界面就修改成了英文。**
- 这个问题正在思索中 ,大概的思路是在检查键盘的在待机界面下 ,一旦是按键进入菜单，在 CheckKey 函数中启动一个定时器，在未曾超时的情况下把数字键的输入赋值给 Item，此时系统 FSM 在 MenuXXXInit 的状态中，修改按键键值为 Up，模拟一次确定键的输入，修改 Flash= 1 让后面的代码调用对应的 Up 处理函数，达到输入的目的。可以如此抽象这个过程：上下翻动按键智能顺序改变 Item 的值，而数字按键直接修改了 Item 的值，并模拟一次确定键输入，使得菜单直接进入对应 Item 的执行代码/子菜单。
  - ü **2006/09/08 该功能已经实现，MenuFSM.C 中 U8 CheckKey(void)函数添加了对于数字按键的处理。查看 Menu.H 的 DigiKeyLink 开关。**
- 一旦定时器超时了，禁止上面说到的在 CheckKey 中的代码。以后依照常规处理 4 个按键，若是其他按键，退出到拨号界面¯这是手机的处理方法。

- **如何在待机界面下实现数字键的快捷方式动态定义？**
- 在 CheckKey 函数中添加对数字按键的处理，检查是否已经设定为快捷方式的标志，若标志对 ,读取快捷方式跳转。标志和快捷方式均需要可擦写的 Flash EEPROM 等的支持。如果没有标志，进入拨号界面。

  如果并不需要动态定义，那么处理方法跟其它按键的快捷方式一致。
- **关于多国语言切换:**

既然显示内容均是通过切换内容指针来实现的，那么多少种语言都不是问题，而且可以做到不同种语言的菜单项数无须一样。但这个带来的小问题是在于 Up 函数中也需要根据语言来判别 Item——既然 Item 对应的项在不同语言下的翻译或者对应的功能可能是不一样的。

菜单层把需要显示的字符串传递给液晶显示函数，该函数根据字符内码显示内容到液晶，多国语言的实现关键在于液晶显示函数当中，跟菜单框架关系不大。

```
//DispMenuMeasureInit ()代码片段
BarMenu = &MenuMeasure[language]; //切换到功能菜单的显示内容
ItemNum = (*BarMenu).TextNum;    //获得需要显示的条项数目
DispItem = (*BarMenu).Text;        //指向字符串
```

菜单的显示内容都在 Menu.C 当中定义。

```
U8 code *MeasureItemCN[]=
{
{"检测上拉电阻"},
```

```
{"清空出错记录"},
{"在线命令"},
{"输入口测试"},
{"模拟主轴测试"},
{"轴信号测试"},
{"继电器测试"},
};

U8 code *MeasureItemEN[]=
{
{"PushUp Res"},
{"ClearErrCount"},
{"OnlineCMD "},
{"SysInputTest"},
{"SVC TEST"},
{"AXis TEST"},
{"Relay TEST"},
};

//宏定义:
#define MenuWhat(n) {(U8 **)n,(sizeof(n)/sizeof(U8 code *)),0}

code struct Menu MenuMeasure[] =
{
MenuWhat(MeasureItemCN),
MenuWhat(MeasureItemEN),
};
```

需要留心 BarMenu = &MenuFunc[language]; 这个语句并没有边界保护能力，如果在 Menu.C 没有定义实质内容，或者 language 越过你定义的范围，那么情况不可预料！

关于内存耗损:

跟菜单深度有关系，可以统计 Menu.H 当中的全局变量的数量就知道了，大概是 25 + 3*N，N 是菜单深度 MenuSize。曾经用这个系统在仅有 256Byte 的 SM89C58 上实现过项目，当时菜单深度设置为 MenuSize =8，

Keil 编译结果：

```
Program Size: data=166.5 xdata=0 code=29114
```

由于实际上菜单深度没有到 8 仅仅到 4 - 5 的样子，因此修改菜单深度 MenuSize=5，重新编译，结果如下：

```
Program Size: data=157.5 xdata=0 code=29114
```

如果依然觉得内存耗损过多，那么检查 Menu.C 当中的全局变量的取值范围，然后想办法合并那些取值范围小的成为一个变量，然后修改代码中使用到这些变量的地方，屏蔽与之无关的位。

关于更多的改进:

框架如此，看你怎么改了，更多的细节都是要费比较多的时间来修改的。

目前代码比较完整的是条项层菜单的处理，而关于条项层菜单也有可以改进和扩展应用的方面。

比如显示内容动态生成的，在短消息的收件箱中就是如此实现的，根据实际的短消息条数和短消息的发送者来显示菜单。

而在一些地方期望进去条项层子菜单时并不是反显就在 Item=0 Where=0 的地方，而是根据动态修改 Item 和 Where 来决定显示的，比如在语言选择菜单中，进入菜单是是反显在当前语言的地方，而不一定是 Item=0 Where=0 的地方。（已经实现！）

还有在彩色液晶当中条项显示可以更改背景颜色什么的，可以在当前条项的位置前面加小图标来突出显示。

很多，你能想到，基本上可以实现。

关于移植:

需要实现字符串显示函数，理所当然也这个字符串函数应当自己实现多语言的显示，就是根据内码识别语言显示内容。

关于快捷方式:

在待机界面下 ,实际上不单单 Up 按键可以进入菜单 ,其它按键也是可以有功能定义的，比如把 Left 按键链接到测量菜单去，在菜单退出后，并直接回到待机界面下。

```
//待机界面下 Left 键的处理
void DispMenuTopLeft(void)
{
    KeyPressCount = 0;
/*
*   这里可以做这个按键的功能定义,因为这是在待机界面下
*   比如,做某个子菜单的快捷链接--按下该按键就直接跳转到某个子菜单
*/
//--------------------
//这里是顶层菜单,因此在进入子菜单前需要初始化变量
    InitMenuVal();
//--------------------------
//状态跳转,进入子菜单
    Jump2Menu(MenuMeasureIndex, FlashMode_AutoInit);
    return;
}
```

除了这样一种链接方式外，也可以是直接进入功能函数的，比如按 Right 按键直接进入测量函数。函数退出前把当前状态索引号恢复到待机界面的索引号，置 Flash 为 1，通知 CheckKey()刷新显示就好了。

```
//待机界面下 Up 键的处理
void DispMenuTopUp(void)
{
    GUI_Clear();
//--------------------
//这里是顶层菜单,因此在进入子菜单前需要初始化变量
    InitMenuVal();
//--------------------------
//状态跳转,进入子菜单      //jump to Menu index
    Jump2Menu(MenuIndex);
    Flash = 1;            //tell check_key() that we need reflash screen
```

```
    return;
}

//待机界面下 Left 键的处理
void DispMenuTopLeft(void)
{
    Jump2Menu(MenuTopIndex);//无功能定义，返回本索引号
    return;
}
```

关于如何实现在线帮助提示

在 Nokia 一些版本的手机中，用户一旦让界面停在某个菜单/菜单项上，之后将近一段时间后没有按键输入的话，界面将立刻跳转去显示当前菜单/菜单项的说明文档，期间可以通过上下翻页按键来浏览整个文档¯因为单屏可能不够显示整个说明文档。确定或者退出按键将返回原来的菜单/菜单项的显示。

思路：代码实现上需要对按键做超时累计，一旦超时了，根据索引号知道需要显示的帮助文档是什么，调用帮助显示函数显示文档。

如何快速创建一个条项层菜单?

1。复制 **MenuPara.C** 文件，修改文件名为 **MenuXXX.C**(XXX 为菜单名称)。

2。打开 **MenuXXX.C**,把

```
void DispMenuParaInit()
void DispMenuParaUp()
void DispMenuParaDown()
void DispMenuParaLeft()
void DispMenuParaRight()
```

五个函数的 Para 替代为 XXX。

3。在 **MenuFSM.C** 中修改 FSM 状态表 `const KbdTabStruct KeyTab[]` 一个条项层菜单需要 5 个条目。

4。根据函数 DispMenuParaInit()在 KeyTab 中的实际位置，在 Menu.H 中定义一个索引号。如同`#define MenuParaIndex 15` 那样。

5。在需要挂接该子菜单的层那里的 Up 函数的分发项，把指针修改到该子菜单索引号。否则这个子菜单是孤立的，没有任何的可能访问到该菜单。方法类似 MenuTop.C 中 void DispIcoMenuUp()函数中是如何挂接 Para 这个子菜单的那样。

6。void DispMenuXXXUp()中定义实质功能函数。

7。Menu.C 中定义条项标题内容，并修改 DispMenuParaInit 的指针指向该数组。

Ø 为什么没有在定义条项菜单的文字的时候使用函数指针的方法对应执行函数?

在条项菜单层的 Up 函数处理中，我们是通过 switch(Item)来做分支处理的，由于通

常 Item 的值不会太大，因此用 switch 的方法可行，如果当 Item 的值较大时，那么可以这样：建立一个跟条项文字对应的执行函数调用表。在 Up 函数中以查表的方式跳转。

实际上可能多语言的时候，可能不同语种的执行函数不一样，那么可以把给每个语种做一个表格，由语种和 Item 来查找。

实际上或者有更好的把条项的文字和对应的执行函数关联在一起，集中在 Menu.H 中体现，从而使得修改时，显示内容和执行函数能够较为直观的在一起被审阅。

在 FramBuffer 型 LCD 上直接使用这份代码

这份代码使用的 LCD 控制器是 KS0108，底层关于显示的驱动均是基于 KS0108 上实现，但并不限制于这个型号的 LCD。我曾经做过的移植包括：

a) SM89C58+128x64 的液晶，液晶控制器是 KS0108，CPU 频率：11.0592MHZ
b) STC89C58+192x64 的液晶，液晶控制器是 KS0108，CPU 频率：24MHZ
c) MSP430 + NOKIA 6610 132x132 256 色屏，CPU 频率：8MHZ
d) 带控制器的 ARM9 CPU，对液晶操作是直接读写显存
e) uC/GUI 在 PC 上的虚拟移植，也是对显存的直接读写（实际上本文中的部分图片就是这个版本上的截图）

这里我提供如何在极少对 LCD 函数改动情况下，如何在 FramBuffer 型 LCD 上使用 KS0108 的驱动函数。

方法很简单，直接架空最底层的函数：Display_Locate

```
void Display_Locate(unsigned char DisplayData, unsigned char X, unsigned
char Y)
{
   unsigned char i;
   Y = Y*8;
   for(i = 0; i < 8; i++){
      if (DisplayData&(1<<i)){
         Menu_SetColor(Menu_RED);//前景色
         Menu_DrawPixel(X, Y++);
      }else {
         Menu_SetColor(Menu_BLUE);//背景色
         Menu_DrawPixel(X, Y++);
      }
   }
}
```

移植时需要提供一个描点函数 `Menu_DrawPixel` 来完成。

为什么公布这份代码，使用的话需要授权什么的么？

我想有用的东西，自己一个人放着在电脑里，N 年后也可能变成占空间的垃圾文件，或者别人用的上这份代码呢，于是打算公布代码，期间断断续续续的写文档，从 2007 年年初到现在，期间代码几经修改，在后来的几个项目中一次一次移植运用。

这份代码你可以随便使用，随便修改，但作为反馈，在你使用使用这份代码，写封 Email 来告诉我你在用这个东西，当然这不是最终的目的，我是期望你能告诉我你做的改进和移植过程等等这些有利于我再次反馈到代码中去，以便我也有得着。

还有，如果你想对别人声称这份代码是你写的，呵呵，那么就不诚实了，用这份代码的时候，真的希望你能看懂代码，看懂后，这些知识就是你的了。那时，或者你会跟别人说："我找了份还不差的菜单的代码，我把它搞定了，我还发现了代码中的 BUG"。呵呵，记得把 BUG 反馈给我,包括这份文档里面的 BUG。

可以来 Email 探讨其中的问题，但不保证一定回复邮件。来 Email 的时候务必记得把问题写清楚，我看不懂问题，又怎么回复你呢？

- liandao @2007-9-2
- lycld@163.com

未解决问题:

如何结构化整个显示内容？使得显示的各种内容跟程序隔离？

图标层: 图标 字符串 执行函数

条项层: 字符串 执行函数

修改记录：

2006/8/16 开始动手写作这个文档。

2007/3/17 毕业前追写了部分文档，代码也有改进。

2007/5 移植到一个项目中使用，期间做了许多修改。

2007/8 决定完成全部文档，并改进代码。