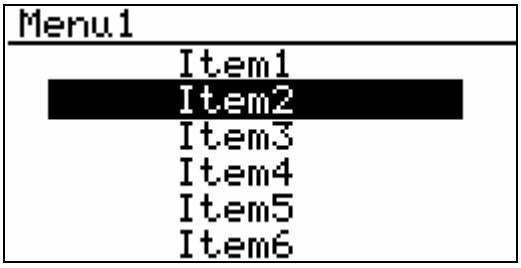
APPLICATION NOTE

LCD6402B Implementing List Menus

1 INTRODUCTION

This application note describes how to implement list menus on the LCD6402B controlled by a PIC16F873 host.

In many man machine interfaces the user is required to select an item from a list similar to the one shown below.



1.1 IMPLEMENTATION

There are two parts to the list menu, the menu graphics and the driver code. The menu graphics are created using LCDLAB's drag and drop editor which are then programmed to the LCD6402B. The driver code is written in 'C' which runs on the host controller and recalls the graphics via the I2C bus.

This division permits the menu titles to be stored on the LCD6402B and hence alterations are be made by LCDLAB without having to modify the driver code.

1.2 GRAPHICS

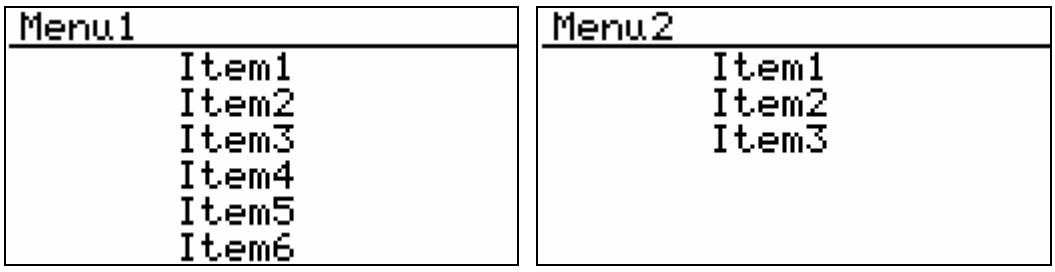
The main requirements for the list menu graphics are:

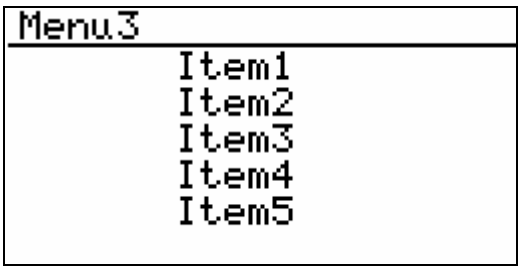
i) The first menu item in each list menu has the same origin.

ii) Subsequent menu items have the same pitch from one another.

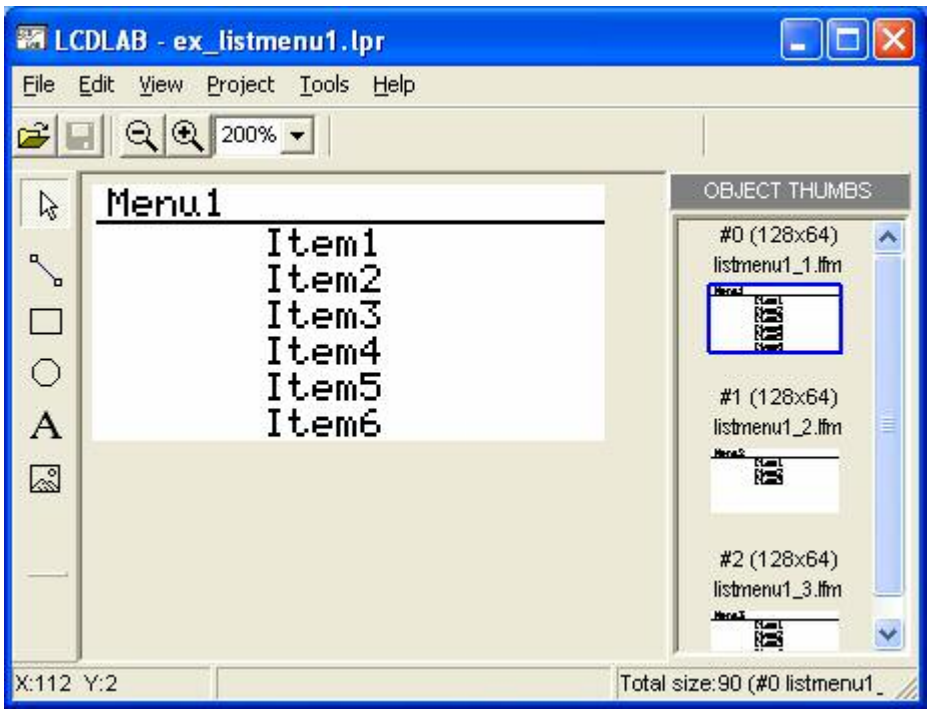
Thus when the host controller moves the inverted cursor up or down by a fixed number of pixels it will always align correctly to the item text.

In our example we shall create three menus; Menu1, Menu2 and Menu3 as follows:





Each of the above graphics are created using LCDLAB (see the screen shot below). This is done by first constructing Menu1 and using that as a template for Menu2 and Menu3. See LCDLAB Help for more information on placing and editing graphics.



The items have been captioned ‘Item1’, ‘Item2’ etc, but these can be anything you want and are easily changed by double clicking the desired caption and entering the text of your choice. The above is available as an example project that comes with LCDLAB.

See project:'ex_listmenu1\LCDLAB\ex_listmenu1.lpr' for more details.

1.3 DRIVER

The list menu driver code (shown below) has two main functions: a) to load the menu and set the cursor to its initial position, and b) to move the cursor to the next item.

Before loading a new list menu the driver needs to know the graphic to use, how many items the menu has and the initial item. This information is passed using the variables `LM1_Graphic`, `LM1_Count` and `LM1_Value`. Once

assigned `LM1_Load()` is called to load the menu. The job of `LM1_Load()` is to set the graphics mode, clear the screen, load the menu graphic and set the inverted cursor to the appropriate item. There after `LM_Modify()` is called in response to user input to move the cursor to the next item. When the cursor reaches the bottom of the list it moves it back to the first item. `LM_Modify()` also updates `LM1_Value` with the current selected item. Thus `LM1_Value` can be inspected at any time to determine the current item.

```
////////////////////////////////////////////////////////////////
// LISTMENU1.C
//
// Drive code for list menu.

// Position and pitch of menu items
#define lm1X 12
#define lm1Y 11
#define lm1PITCH 9

int LM1_Value;
int LM1_Count;
int LM1_Graphic;

void LM1_InvertCursor(int itemindex)
{
    int x, y;

    LCD64_RegWrite(rMODE, mdFG_XOR);
    x = lm1X;
    y = lm1Y+itemindex * lm1PITCH-1;
    LCD64_INSWrite(gRECTF);   // Draw rectangle
    LCD64_INSWrite(x   );     // x0
    LCD64_INSWrite(y   );     // y0
    LCD64_INSWrite(x+100 );   // x1
    LCD64_INSWrite(y+8 );     // y1
}

int LM1_Modify(void)
{
    LM1_InvertCursor(LM1_Value);        // Undraw current item
    LM1_Value++;
    if (LM1_Value == LM1_Count)
        LM1_Value = 0;
    LM1_InvertCursor(LM1_Value);        // Draw new item
    return LM1_Value;
}

void LM1_Load(void)
{
    LCD64_RegWrite(rMODE, 0);             // Normal mode
    LCD64_CMDWrite(cmdCLEAR_SCREEN);
    LCD64_CMDWrite(LM1_Graphic);          // Recall graphic
    LM1_InvertCursor(LM1_Value);
}
//
////////////////////////////////////////////////////////////////
```


The following code snippet shows how to initialise Menu1:

```
// Load List Menu1
LM1_Graphic = 0;   // Menu1 graphic #0
LM1_Count = 6;     // Number of items in menu
LM1_Value = 0;     // Select the first item
LM1_Load();        // Load menu1
```

To move the cursor to the next item simply call `LM1_Modify()` as follows:

```
LM1_Modify();      // Move cursor
```

To determine which item the cursor is simply read `LM1_Value` as follows:

```
// If the third item is selected then...
if (LM1_Value == 2)
      ...
```

The above driver code makes use of the LCD64 driver functions which are listed in appendix A.

Note that all code has been written in 'C' and compiled using a CCS 'C' compiler.

1.4 APPLICATION EXAMPLE

To demonstrate our list menus we'll write an application that will navigate from one menu to the other using event driven code. In addition each menu is 'sticky' in that when the menu is called again the selected item will be the same as when the menu exited.

Program execution begins at `main()` which initialises the display, starts the menu system with `MyListMenu1()` and enters a loop which calls `DoEvents()`.

`MyListMenu1()` kicks things of by loading list menu #1 and assigning a key handler for the menu. It is not possible to use function pointers for the PIC series microcontroller so instead an enumerated value is used to reference the desired function. Since we are using event driven code there is no need to enter a loop scanning for key presses with in `MyListMenu1()` this done instead by `DoEvents()`.

When `DoEvents()` detects a key press it calls the handler `MyListMenu1_KeyDown()`. If key B is pressed then `LM1_Modiy()` is called which moves the inverted cursor down to the next item or returns it to the top if the cursor was on the last item. The value returned by `LM1_Modiy()`, i.e. which item the cursor is on, is saved in

`MyListMenu1_Value`. If key C is pressed then `MyListMenu2()` is called to load list menu #2 in the same way as list menu #1.

Each menu calls the next in turn looping back to menu #1. This example can be expanded to have more menus or the menu drive could be modified to create a new style of list menu.


```
////////////////////////////////////////////////////////////////////
// EX_LISTMENU1.C
// Example application code for three list menus.


//////////////////////////////////////////////////////
//                                                  //
//                  DIRECTIVES                      //
//                                                  //
//////////////////////////////////////////////////////
#include <16F876A.h>
#fuses hs,nowdt,noprotect,put,nowrt,nolvp,nodebug,nobrownout


#case
#zero_ram

#use delay(clock=19660800, restart_wdt)


//////////////////////////////////////////////////////
//                                                  //
//                  PROTOTYPES                      //
//                                                  //
//////////////////////////////////////////////////////
void MyListMenu1(void);
void MyListMenu2(void);
void MyListMenu3(void);

//////////////////////////////////////////////////////
//                                                  //
//                  DEFINES / VARIABLES             //
//                                                  //
//////////////////////////////////////////////////////

// #defines genertated by LCDLAB
#define listmenu1_lfm 0
#define listmenu2_lfm 1
#define listmenu3_lfm 2

// Keys along right side of LCD
#define KEY_A 13            // top
#define KEY_B 14            // middle
#define KEY_C 15            // bottom


// Dispatcher function ID's
enum TFuncID
    {
    fnNull,
    fnMyListMenu1_KeyDown,
    fnMyListMenu2_KeyDown,
    fnMyListMenu3_KeyDown
    };


typedef struct {
    TFuncID OnKeyDown;
```



```
    int Value;
} TKey;
TKey Key;



////////////////////////////////////////////////////////
//                                                    //
//                  LCD6402 INTERFACE                 //
//                                                    //
////////////////////////////////////////////////////////

#include "lcd64xx.c"

////////////////////////////////////////////////////////
//                                                    //
//                  GUI FUNCTIONS                     //
//                                                    //
////////////////////////////////////////////////////////

#include "listmenu1.c"

////////////////////////////////////////////////////////
//                                                    //
//                  APPLICATION CODE                  //
//                                                    //
////////////////////////////////////////////////////////

//*************************************
// LIST MENU 1
//
//*************************************
int MyListMenu1_Value;

void MyListMenu1_KeyDown(void)
{
    if (Key.Value == KEY_B)
        MyListMenu1_Value = LM1_Modify();
    else if (Key.Value == KEY_C)
        MyListMenu2();
}

void MyListMenu1(void)
{
    LM1_Value = MyListMenu1_Value;
    LM1_Count = 6;
    LM1_Graphic = listmenu1_lfm;
    LM1_Load();
    Key.OnKeyDown = fnMyListMenu1_KeyDown;  // Assign key handler
}

//*************************************
// LIST MENU 2
//
//*************************************
int MyListMenu2_Value;

void MyListMenu2_KeyDown(void)
{
```



```
    if (Key.Value == KEY_B)
        MyListMenu2_Value = LM1_Modify();
    else if (Key.Value == KEY_C)
        MyListMenu3();
}

void MyListMenu2(void)
{
    LM1_Value = MyListMenu2_Value;
    LM1_Count = 3;
    LM1_Graphic = listmenu2_lfm;
    LM1_Load();
    Key.OnKeyDown = fnMyListMenu2_KeyDown;  // Assign key handler
}


//*************************************
// LIST MENU 3
//
//*************************************
int MyListMenu3_Value;

void MyListMenu3_KeyDown(void)
{
    if (Key.Value == KEY_B) {
        MyListMenu3_Value = LM1_Modify();
    } else if (Key.Value == KEY_C )
        MyListMenu1();
}

void MyListMenu3(void)
{
    LM1_Value = MyListMenu3_Value;
    LM1_Count = 5;
    LM1_Graphic = listmenu3_lfm;
    LM1_Load();
    Key.OnKeyDown = fnMyListMenu3_KeyDown;  // Assign key handler
}



void DoEvents(void)
{
    TFuncID dispatch;


    Key.Value = LCD64_Read(rNEWKEY);
    if ((Key.Value !=0) && (Key.Value !=0xff)) {
        dispatch = Key.OnKeyDown;
    }

    // Dispatcher
    switch (dispatch) {
        case fnNull : break;
        case fnMyListMenu1_KeyDown : MyListMenu1_KeyDown(); break;
        case fnMyListMenu2_KeyDown : MyListMenu2_KeyDown(); break;
        case fnMyListMenu3_KeyDown : MyListMenu3_KeyDown(); break;
    }
    dispatch = fnNull;
```



```
}

void main() {
    int key;

    // Initialise
    LCD64_CMDWrite(cmdCLEAR_SCREEN);
    LCD64_CMDWrite(cmdBACKLIGHT_ON);


    MyListMenu1();
    while(1) {
        DoEvents();
    }
}
//
////////////////////////////////////////////////////////////////////
```


2 APPENDIX A

```
////////////////////////////////////////////////////////////////////
// LCD64XX.C

//////////////////////////////////////////////////////
//                                                  //
//          LCD64 DEFINES                           //
//                                                  //
//////////////////////////////////////////////////////

#use i2c(sda=PIN_C4, scl=PIN_C3, FORCE_HW)

// LCD64 registers
#define rCMD                    0x20
#define rSTATUS                 0x21
#define rNEWKEY                 0x22
#define rKEY                    0x23
#define rCUX                    0x24
#define rCUY                    0x25
#define rORX                    0x26
#define rORY                    0x27
#define rNVML                   0x28
#define rNVMH                   0x29
#define rOPTION                 0x2A
#define rINS                    0x2B
#define rCONTRAST               0x2C
#define rFGCLR                  0x2D
#define rBGCLR                  0x2E
#define rMODE                   0x2F
#define rFONT                   0x30

// LCD64 rCMD command defines
#define cmdRESET_INTERPRETER    0x91
#define cmdBACKLIGHT_ON         0x92
#define cmdBACKLIGHT_OFF        0x93
#define cmdCLEAR_SCREEN         0x94
#define cmdSAVE_CONTEXT         0xa8
#define cmdRESTOR_CONTEXT       0xa9
#define cmdRESET                0xff

// LCD64 rINS graphic instruction defs  // Expected parameters
#define gRESET                  0x08    //
#define gCLS                    0x21    //
#define gORIGIN                 0x2A    // x, y
#define gCURSOR                 0x3A    // x, y
#define gFGCLR                  0x41    // clr
#define gBGCLR                  0x42    // clr
#define gMODE                   0x43    // mode
#define gPIX                    0x4A    // x, y, n
#define gLINETO                 0x52    // x, y
#define gRECT                   0x5C    // x0, y0, x1, y1
#define gRECTF                  0x64    // x0, y0, x1, y1
#define gFONT                   0x69    // f
#define gPUTC                   0x71    // c
#define gHRAST                  0x79    // rast
#define gPUTS                   0x80    // c0..cn, 0
```



```
#define gHBMP                   0x8A    // w/8, H, rdata,
#define gCIRCLE                 0x99    // r
#define gCIRCLEF                0x9A    // r
#define gVRAST                  0xA1    // r
#define gVBMP                   0xAA    // w, h/8, rdata,
#define gCALLI                  0xF1    // n

// LCD64 rFGCLR & rBGCLR register colour defines
#define clrBLACK                0
#define clrWHITE                0xFF

// LCD64 rMODE register defines
#define mdFG_OR                 0x00    // OR foreground colour
#define mdFG_XOR                0x01    // XOR foreground colour
#define mdFG_ON                 0x00    // Foreground colour ON
#define mdFG_OFF                0x02    // Foreground colour OFF
#define mdBG_OR                 0x00    // OR backgound colour
#define mdBG_XOR                0x10    // XOR backgound colour
#define mdBG_ON                 0x00    // Backgound colour ON
#define mdBG_OFF                0x20    // Backgound colour OFF


////////////////////////////////////////////////////////
//                                                    //
//          LCD64 INTERFACE FUNCTIONS                 //
//                                                    //
////////////////////////////////////////////////////////

#inline
int LCD64_Read(int addr)
{
    int r;

    i2c_start();
    i2c_write(0xa0);
    i2c_write(addr);
    i2c_start();
    i2c_write(0xa0 | 1);
    r=i2c_read(0);
    i2c_stop();
    return(r);
}

#inline
void LCD64_Write(int addr,int data)
{
    i2c_start();
    i2c_write(0xa0);
    i2c_write(addr);
    i2c_write(data);
    i2c_stop();
}

#inline
void LCD64_BusyWait( void)
{
    int status;

    do {
```



```
        delay_ms(1);
        status = LCD64_Read(0x21);
        status &= 0x80;
    } while ( status != 0);
}

void LCD64_RegWrite(int reg,int data)
{
    LCD64_BusyWait();
    LCD64_Write(reg, data);
}

void LCD64_CMDWrite( int cmd)
{
    LCD64_BusyWait();
    LCD64_Write( 0x20, cmd);
}

void LCD64_INSWrite( BYTE data)
{
    LCD64_BusyWait();
    LCD64_Write( 0x2b, data);
}

////////////////////////////////////////////////////////
//                                                    //
//          LCD WRAPPER FUNCTIONS                     //
//                                                    //
////////////////////////////////////////////////////////

void LCD_Cursor(int x,y)
{
    LCD64_BusyWait();
    LCD64_Write( 0x2b, 0x3a);
    LCD64_Write( 0x2b, x);
    LCD64_Write( 0x2b, y);
}

void LCD_Putc(int c )
{
    LCD64_BusyWait();
    LCD64_Write(0x2b,0x71);
    LCD64_Write(0x2b,c);
}

void LCD_Font( int f )
{
    LCD64_BusyWait();
    LCD64_Write(0x2b, 0x69);
    LCD64_Write(0x2b, f);
}

//
////////////////////////////////////////////////////////////////////
```

