Mosaic Industries

Custom Display Characters

APPLICATION NOTE
MI-AN-022a

Summary

The following software shows how custom characters can be defined and displayed on the character display. The code is programmed using C.

```
/*  Dispchar.c */

// Copyright 1997 Mosaic Industries, Inc.  All Rights Reserved.
//  Disclaimer: THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, WITHOUT ANY
//  WARRANTIES OR REPRESENTATIONS EXPRESS OR IMPLIED, INCLUDING, BUT NOT
//  LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS
//  FOR A PARTICULAR PURPOSE.

// This program demonstrates how to define custom characters for the
// 4x20 display. Up to 8 custom characters can be loaded into the
// display RAM at any time.  Read the comments in the code to learn how
// to access more than 8 custom characters.


#include <\mosaic\allqed.h>
// this include statement should appear at the top of each source code file.

#define BYTES_PER_CHAR  8      // 8 vertical pixels
             // there are 5 horizontal pixels: we'll define 5 bits per byte

#define MAX_CUSTOM_CHARS  8   // # available spots in display's CG RAM

// An easy way to define and visualize a bit-mapped character
// is to write it out in binary. The 1's make a pattern that is easily seen.
// The binary can be converted to hex and stored in a byte array or struct
// and then written to the CG RAM (character generator ram) in the display
// every time the system is powered up.
// Each character is 5 bits wide by 8 bits high.

// Here's an up-arrow expressed in binary and hex:
// 00000        0x00
// 00100        0x04
// 01110        0x0E
// 11111        0x1F
// 00100        0x04
// 00100        0x04
// 00100        0x04
// 00000        0x00
// Here's a down-arrow:
// 00000        0x00
// 00100        0x04
// 00100        0x04
// 00100        0x04
// 11111        0x1F
// 01110        0x0E
// 00100        0x04
// 00000        0x00

// Here's a white-on-black (inverse) up-arrow:
// 11111        0x1F
// 11011        0x1B
// 10001        0x11
// 00000        0x00
// 11011        0x1B
// 11011        0x1B
// 11011        0x1B
// 11111        0x1F

// Here's a white-on-black (inverse) down-arrow:
// 11111        0x1F
// 11011        0x1B
// 11011        0x1B
// 11011        0x1B
// 00000        0x00
// 10001        0x11
// 11011        0x1B
// 11111        0x1F

// We define and init a 2-dimensional byte-array named CustomChars[][]
// that specifies each  bit-mapped character.
// Each row contains the 8 bytes that define a single character.
// Column 0 contains the uppermost pixels in each char;
// column 7 contains the lowermost pixels in each char.


// put initializer in ROM using linker #pragma directive:
#pragma option init=.code    // use this for single page "hammer" compile
// #pragma option init=.doubleword     // for multi-page "bricks" compile

static char CustomChars[BYTES_PER_CHAR][MAX_CUSTOM_CHARS]  =
{   { 0x00, 0x04, 0x0E, 0x1F, 0x04, 0x04, 0x04, 0x00 },     // UP_ARROW
 { 0x00, 0x04, 0x04, 0x04, 0x1F, 0x0E, 0x04, 0x00 },     // DOWN_ARROW
      { 0x1F,0,0,0,0,0,0,0 },
      // these are test patterns; define your own chars here
{ 0x00, 0x1F,0,0,0,0,0,0},
{ 0x00, 0x00, 0x1F,0,0,0,0,0},
{ 0x00, 0x00, 0x00, 0x1F,0,0,0,0},
{ 0x1F, 0x1B, 0x11, 0x00, 0x1B, 0x1B, 0x1B, 0x1F },     // INV_UP_ARROW
{ 0x1F, 0x1B, 0x1B, 0x1B, 0x00, 0x11, 0x1B, 0x1F }      // INV_DOWN_ARROW
};
```

```
#pragma option init=.init     // restore initialization section to RAM

// define char_id = row# in CustomChars:
#define UP_ARROW        0
#define DOWN_ARROW      1

// fill in any definitions you want here!!

#define INV_UP_ARROW    6
#define INV_DOWN_ARROW  7

// Note: in multipage applications, make sure that the
// CustomChars data array is in the same source file as the function
// that writes the data to the display; otherwise page conflicts can occur.
// No such restriction applies if you are using the "hammer" icon
// for single-page compilation.

#define SET_CGRAM_ADDR_MASK  0x40   // bitwise OR with (char# * 8) to make cmd

_Q uchar MakeCGRamCommand(char char_id)
// this is a hardware dependent command; see the display
// data sheet at the back of the QED Hardware manual.
// Implements "Set CG RAM address" command.
{   return((8*char_id) | SET_CGRAM_ADDR_MASK );
}


_Q void Init1CustomChar(char char_id, char row_index)
// char_id is how we specify char; specifies location in display cg ram
// row_index refers to location of font definition in CustomChars[][]
{   uchar command, j;
   command = MakeCGRamCommand(char_id);   // set CG RAM -> start of char
   for(j=0; j < BYTES_PER_CHAR; j++)   // for each byte in character
   {   CommandToDisplay(command+j);    // set address in CG Ram
       CharToDisplay(CustomChars[row_index][j]);   // write bit pattern
  }
}

_Q void InitCustomChars(void)
{   char i;
   for(i=0; i < MAX_CUSTOM_CHARS; i++)         // for each character
       Init1CustomChar(i, i);  // assumes char_id == row_index
}
// NOTE: You can load font definitions into the CG RAM "on the fly".
// This allows you to keep a large font table in CustomChars[][]
// and selectively load 8 chars at a time into the display RAM.
// Init1CustomChar() allows you to define a char_id that is
// different than row_index; all you need to do is code new
// versions of InitCustomChars()  to gain access to more than
// 8 custom-defined fonts.
// Still, a maximum of 8 custom characters can be displayed at any one time.
```

```
_Q void Show(void)
// a test routine. puts 8 custom characters on display
// starting at current cursor position.
// Assumes that custom chars have been initialized.
{   char i ;
PutCursor(0,0);     // display at line#0, col#0
for(i=0; i < MAX_CUSTOM_CHARS; i++)         // for each character
      CharToDisplay(i);    // write to display
}


void main( void )
{ InitCustomChars();
Show();
}
```