TN-192

Bar Graph shows analog trends
on a dot-matrix character LCD module

Mark Durgin - SCIDYNE® Corporation

Introduction

Dot-Matrix character LCD (Liquid Crystal Display) modules are excellent at conveying alphanumeric information. Their low cost and ease-of-use make them a popular choice for products and devices requiring more than just basic On-Off LED indicators. However, interpreting analog trends with digits can be challenging. This Tech Note describes how to create a horizontal bar graph to visually display a numerical variable. It targets LCD modules which use the Hitachi HD44780 (or compatible) LCD controller chipset. Finally, the concepts are demonstrated using a 16 character by 2 row LCD module connected to an Arduino UNO. The C language code is easily adaptable to other microcontrollers.

LCD Module Basics

The lower-level details of how LCD modules display characters is generally not a concern for the average software programmer. However, understanding a little more of the underlying functionality will be helpful in creating and displaying the bar graph.

Every character written to the LCD module is represented in a matrix of dots (i.e.; pixels) arranged as 5 columns by 7 rows. An eighth row exists but is typically reserved for an optional underline / cursor effect. Depending on the character to be displayed some of the dots will be visible while others are not.

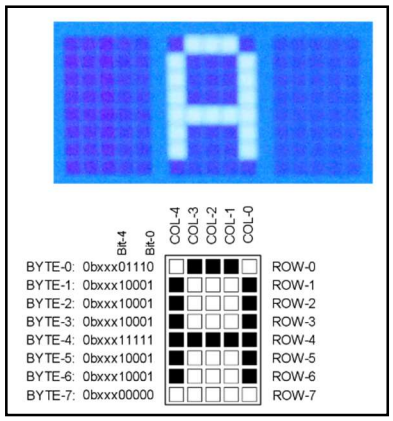


Figure 1 - 5x7 Matrix

Characters are formed from the binary pattern of eight bytes; one byte for each row and five bits from each byte. The upper three bits, b[7:5], are not used. A dot will be visible when its corresponding bit is logic "1". The relationship is shown in **Figure 1**. For standard characters the bytes and bits reside in a table within the LCD controller chip's ROM (Read-Only-Memory). The chip does all the work of translating the individual ASCII characters it receives to the multi-byte bit patterns needed to render the intended characters. In this example, the controller receiving an ASCII code 0x41 will look up the bytes to show the capital letter A.

Custom Characters

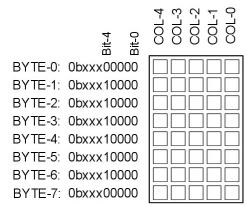
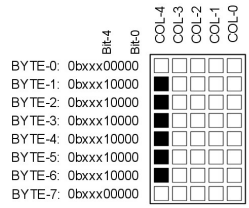
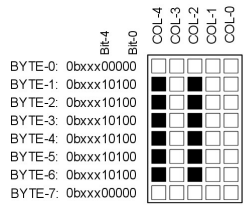
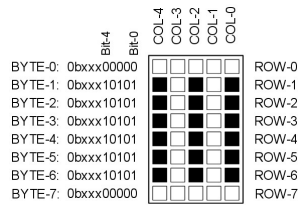
LCD modules support the standard ASCII character set representing Letters, Numbers, and common Punctuation marks. Sometimes the standard ASCII character set, or any special characters which might also be included within LCD controller, does not contain a particular character or symbol that is needed. For instance, the user may want to display the capital Greek letter omega (Ω) to signify Ohms.

Fortunately, the designers of the controller chip took this into consideration by allowing up to eight custom characters to be loaded into an area of the controllers RAM (Random-Access-Memory). Once programmed, the custom characters or symbols are accessed just like standard characters originating

from ROM. Because RAM is volatile memory power to the LCD module must be continually maintained. As such, custom characters must be loaded after each power interruption.

Custom characters get loaded into the LCD module using the Arduino **lcd.createChar(num, data)** function. The **num** parameter identifies which one of the eight custom characters (0-7) will be loaded and **data** references a multi-byte array containing the character dot pattern.

For the style of bar graph being created in this Tech Note four custom characters are required. The table below shows the resulting dot patterns along with the C language multi-byte arrays used to produce them.

Filler - No Bars	**Partial** - 1 or 2 Bars		**Full** - 3 Bars
Custom Character 0	Custom Character 1	Custom Character 2	Custom Character 3
CHAR_0BAR_FILLER	CHAR_1BAR_PARTIAL	CHAR_2BAR_PARTIAL	CHAR_3BAR_FULL
			

```
byte _0vertBar[8] = {
  0b00000000, // Row-0
  0b00000000, // Row-1
  0b00000000, // Row-2
  0b00000000, // Row-3
  0b00000000, // Row-4
  0b00000000, // Row-5
  0b00000000, // Row-6
  0b00000000  // Row-7
};
```

```
byte _1vertBar[8] = {
  0b00000000, // Row-0
  0b00010000, // Row-1
  0b00010000, // Row-2
  0b00010000, // Row-3
  0b00010000, // Row-4
  0b00010000, // Row-5
  0b00010000, // Row-6
  0b00000000, // Row-7
};
```

```
byte _2vertBar[8] = {
  0b00000000, // Row-0
  0b00010100, // Row-1
  0b00010100, // Row-2
  0b00010100, // Row-3
  0b00010100, // Row-4
  0b00010100, // Row-5
  0b00010100, // Row-6
  0b00000000  // Row-7
};
```

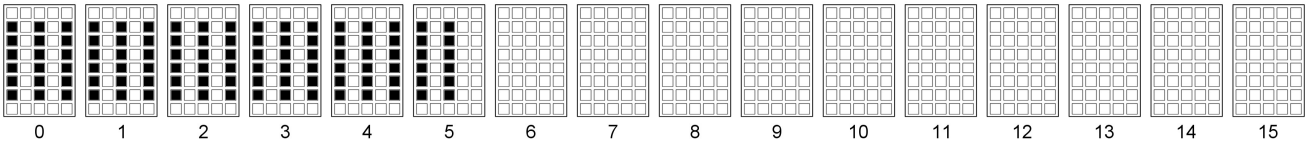
```
byte _3vertBar[8] = {
  0b00000000, // Row-0
  0b00010101, // Row-1
  0b00010101, // Row-2
  0b00010101, // Row-3
  0b00010101, // Row-4
  0b00010101, // Row-5
  0b00010101, // Row-6
  0b00000000  // Row-7
};
```

Bar Graph Anatomy

A bar graph is made up of Full, Partial, and Filler characters. For instance, a bar graph showing 0% would use only Filler characters. Likewise, a bar graph showing 100% would consist of only Full characters. Bar graphs in between will use a combination of Full, Partial, and Filler characters.

A typical LCD module is constructed with the spacing between alternating dots and those of adjacent characters being the same the distance. This characteristic is utilized by displaying a vertical bar in every other column. The result is a bar graph that looks seamless even though it may span multiple characters.

This bar graph shows seventeen bars illuminated. It is constructed from five Full characters [0:4], one Partial character consisting of two vertical bars [5], and ten Filler characters [6:15].



Bar Graph Coding

Producing the bar graph comes down to determining the proper combination of Full, Partial, and Filler characters needed to display the desired number of bars. Once the number of each type is determined they are simply written to the LCD module starting from the left in the order of Full, Partial, and Filler.

The snippet below shows the heart of the bar graph code. A complete listing for the accompanying demonstration program appears in Appendix-A. The code is available for downloading at the SCIDYNE website.

Create a few temporary variables

This section assures the bar graph remains in a displayable range. No attempt should be made to display a negative value or a value beyond the maximum number of bars the LCD can faithfully show.

Based on the value sent to this routine in barval, calculate the needed combination of Full, Partial, and Filler characters.

Locate where the bar graph will appear on the LCD module.

If required, write any Full characters

If required write any Partial character

If required, write any Filler characters

```
//=========================================================================
// Show a horizontal bar graph
// Call with barval equal to number of bars to display and lcdrow
// to set which row on the lcd to place the bar graph.
//=========================================================================
void  lcd_bargraph (char barval, char lcdrow)
{
  // Create temporary variables on stack
  unsigned char fullbars, partbars, fillers;

  // Assure barval stays within displayable range
  if (barval <= 0)
    barval = 1;   // Assure at least one bar get shown, just looks nice

  if (barval > LCD_COLS * BARS_PER_CHAR)
   barval = LCD_COLS * BARS_PER_CHAR;     // Assure display will not be exceeded



  // Pre-calculate number of full bars "|||"
  if (barval <= (LCD_COLS * BARS_PER_CHAR))
    fullbars = barval / BARS_PER_CHAR;
  else
    fullbars = LCD_COLS;

  // Pre-calculate number of partial bars
  // partbars holds the index for the custom
  // partial bar characters: 0 = none, 1 = "|  ", or 2 = "|| "
  partbars =  barval % BARS_PER_CHAR;  // Modulus result will be 0, 1, or 2

  // Pre-calculate number of fillers needed
  fillers = LCD_COLS - fullbars - !!partbars;



  // Locate where the bargraph will appear
  lcd.setCursor(0, lcdrow);  // LCD COL, ROW

  // If needed write the full bar characters to display
  while(fullbars) {
    lcd.write(byte(CHAR_3BAR_FULL)); // Note: When calling lcd.write() a '0' must be cast as a byte
    fullbars--;
  }

  // If needed write the partial bar character to display
  if (partbars)
    lcd.write(byte(partbars));    // Note: when calling lcd.write() a '0' must be cast as a byte

  // If needed write the filler characters to display. This also erases any previous bar graph remnants
  while(fillers) {
    lcd.write(byte(CHAR_0BAR_FILLER));   // Note: when calling lcd.write() a '0' must be cast as a byte
    fillers--;
  }

}
```

Arduino Specific LCD functions

When studying the demonstration software be aware that certain functions are inherently part of the Arduino development environment. Their purpose is to make using LCD modules easier by hiding most of the lower-level details. When adapting the code for another microprocessor system it may be necessary to reproduce these functions if counterparts do not already exist. The specific functions are summarized below.

#include <LiquidCrystal.h>

This library allows an Arduino board to control Liquid Crystal displays (LCDs) using the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs. The library works in either 4 or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).
https://www.arduino.cc/en/Reference/LiquidCrystal

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters.
https://www.arduino.cc/en/Reference/LiquidCrystalConstructor

lcd.begin(cols, rows);

Initializes the interface to the LCD screen, and specifies the dimensions (number of columns and rows) of the display. begin() needs to be called before any other LCD library commands.
https://www.arduino.cc/en/Reference/LiquidCrystalBegin

lcd.createChar(num, data);

Create a custom character for use on the LCD. Up to eight characters of 5x8 pixels are supported (numbered 0 to 7). The appearance of each custom character is specified by an array of eight bytes, one for each row. The five least significant bits of each byte determine the pixels in that row. To display a custom character on the screen, write() its number.
https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar

lcd.setCursor(col, row);

Position the LCD cursor location at column and row where subsequent text written to the LCD will be displayed.
https://www.arduino.cc/en/Reference/LiquidCrystalSetCursor

lcd.print();

Prints text to the LCD.
https://www.arduino.cc/en/Reference/LiquidCrystalPrint

lcd.write(byte());

Write a character to the LCD. When calling lcd.write() a '0' value must be cast as a byte to avoid a compiler error.
https://www.arduino.cc/en/Reference/LiquidCrystalWrite

The Arduino environment also provides several more useful LCD oriented functions that were not mentioned in this Tech Note. Visit the official Arduino website https://www.arduino.cc for more details.

Going Further

With the fundamentals of creating a bar graph understood additional features and functionality can be imagined, such as:

- Create multiple bar graphs, one to display a set-point and another to display a process variable
- Flash the bar graph if above (or below) a preset limit
- Display negative and positive trends by having zero be referenced in the middle of the bar graph
- Add visual graduations to aid in interpreting the bar graph value
- Use a rolling-average filter to smooth rapidly changing bar graph values
- Reduce the number of columns the bar graph occupies so that text can also appear on the same row

References:

https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

https://www.arduino.cc

Seiko Instruments Liquid Display Module Specifications Catalog

Appendix - A

Bar Graph Demonstration Software

```
/*=========================================================================
  SCIDYNE Corporation   649 School Street   Pembroke, MA 02359-3649 USA
  Tel: (781) 293-3059   Fax: (781) 293-4034   URL: www.scidyne.com
  =========================================================================
  NOTICE -- This demonstration software is provided "As Is".  It can be
  freely used and modified as described under the terms of the
  Creative Commons Attribution License agreement.
    For details visit: https://creativecommons.org/licenses/by/4.0/
  =========================================================================
  Project    : Tech Note TN-192 Demonstration Software
  File       : TN192.ino
  Revision   : 1.00
  Date       : 01-05-19
  Author     : Mark Durgin
  Target     : Arduino UNO with LCD connected
  Compiler   : Arduino IDE 1.8.5
  =========================================================================
  Description and usage:

  Demo software to show a Bar graph on a 16-character by 2-row LCD module

  This code runs on an Arduino UNO (or MEGA 2560) connected to a character style
  LCD module with an on-board Hitachi HD44780 (or compatible) controller.

  The LCD module is connected as follows:
  (Based on the industry standard 14-16 pin inline LCD connector layout):

  LCD                      Connect to
  Pin   Function           Arduino Pin
  ---   ---------------    ---------------
  1     GND                GND
  2     +5V                +5V
  3     LCD Drive (contrast)  10K Pot wiper, connect POT ends to +5V and GND
  4     RS                 7
  5     R/*W               GND
  6     EN                 8
  7     DB0   Not Used     NC
  8     DB1   Not Used     NC
  9     DB2   Not Used     NC
  10    DB3   Not Used     NC
  11    DB4   DB0 / DB4    9
  12    DB5   DB1 / DB5    10
  13    DB6   DB2 / DB6    11
  14    DB7   DB3 / DB7    12
  15    Backlight +V       +5V
  16    Backlight -V        GND

  Also, a 10K pot is used to vary the voltage applied to Analog Input #0.
  Connect one end of the pot to +5V and the other end to GND. The pot wiper
  is connected to Arduino AI-0.
```

```
  ================================================================================
  History:
    01-15-19  R0.00   - Project Begins
  ================================================================================*/

// Include any necessary libraries
#include <LiquidCrystal.h>

// Program constants
#define LCD_D4  9          // Arduino pin for LCD 4-bit mode DO / D4
#define LCD_D5  10         // Arduino pin for LCD 4-bit mode D1 / D5
#define LCD_D6  11         // Arduino pin for LCD 4-bit mode D2 / D6
#define LCD_D7  12         // Arduino pin for LCD 4-bit mode D3 / D7
#define LCD_RS  7          // Arduino pin for LCD Register Select
#define LCD_EN  8          // Arduino pin for LCD Enable

#define LCD_COLS  16      // Number of columns on LCD
#define LCD_ROWS  2       // Number of rows on LCD

#define BARS_PER_CHAR 3        // Maximum number of bars per character
#define BARGRAPH_ROW  1        // Display Bar Graph on this LCD row

#define CHAR_0BAR_FILLER  0       // Index to custom character: zero bar   (Filler)   "   "
#define CHAR_1BAR_PARTIAL 1     // Index to custom character: single bar (Partial)  "|  "
#define CHAR_2BAR_PARTIAL 2     // Index to custom character: double bar (Partial)  "|| "
#define CHAR_3BAR_FULL    3         // Index to custom character: triple bar (Full)     "|||"

// Prototypes
void  lcd_bargraph(char barval, char lcdrow);

// Initialize the LCD library with the Arduino pins used
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

// Bar graph characters:
byte _0vertBar[8] = {  // Chracter[0]:  Filler (blank) Character
  0x00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000
};

byte _1vertBar[8] = { // Chracter[1]: | One Bar Partial Character
  0b00000000,
  0b00010000,
  0b00010000,
  0b00010000,
  0b00010000,
  0b00010000,
  0b00010000,
  0b00000000,
};
```

```
byte _2vertBar[8] = { // Chracter[2]: || Two Bar Partial Character
  0b00000000,
  0b00010100,
  0b00010100,
  0b00010100,
  0b00010100,
  0b00010100,
  0b00010100,
  0b00000000
};

byte _3vertBar[8] = {  // Chracter[3]: ||| Three Bar Full Character
  0b00000000,
  0b00010101,
  0b00010101,
  0b00010101,
  0b00010101,
  0b00010101,
  0b00010101,
  0b00000000
};


//=========================================================================
// Program setup
//=========================================================================
void setup() {
  // Initialize LCD and set up the number of columns and rows:
  lcd.begin(LCD_COLS, LCD_ROWS);

  // Create custom LCD characters:  character number, multi-byte array
  lcd.createChar(CHAR_0BAR_FILLER,   _0vertBar);
  lcd.createChar(CHAR_1BAR_PARTIAL, _1vertBar);
  lcd.createChar(CHAR_2BAR_PARTIAL, _2vertBar);
  lcd.createChar(CHAR_3BAR_FULL,     _3vertBar);
}

//=========================================================================
// Program main loop
//=========================================================================
void loop() {
  int sensorReading; // Hold Analog Input #0 value
  char buf[21];      // Message Display Buffer

  // Read the potentiometer on AI-0 and scale it
  sensorReading = analogRead(A0) / 16;

  // Print a message to the lcd
  sprintf(buf,"Bar Graph %2dbars",sensorReading);
  lcd.setCursor(0, 0);  // LCD COL, ROW
  lcd.print(buf);

  // Put bar graph on display
  lcd_bargraph(sensorReading, BARGRAPH_ROW);

}
```

```
//========================================================================
// Show a horizontal bar graph
// Call with barval equal to number of bars to display and lcdrow
// to set which row on the lcd to place the bar graph.
//========================================================================
void  lcd_bargraph (char barval, char lcdrow)
{
  // Create temporary variables on stack
  unsigned char fullbars, partbars, fillers;

  // Assure barval stays within displayable range
  if (barval <= 0)
    barval = 1;   // Assure at least one bar get shown, just looks nice

  if (barval > LCD_COLS * BARS_PER_CHAR)
   barval = LCD_COLS * BARS_PER_CHAR;     // Assure display will not be exceeded

  // Pre-calculate number of full bars "|||"
  if (barval <= (LCD_COLS * BARS_PER_CHAR))
    fullbars = barval / BARS_PER_CHAR;
  else
    fullbars = LCD_COLS;

  // Pre-calculate number of partial bars
  // partbars holds the index for the custom
  // partial bar characters: 0 = "   ", 1 = "|  ", or 2 = "|| "
  partbars =  barval % BARS_PER_CHAR;  // Modulus result will be 0, 1, or 2

  // Pre-calculate number of fillers needed
  fillers = LCD_COLS - fullbars - !!partbars;

  // Locate where the bargraph will appear
  lcd.setCursor(0,  lcdrow);  // LCD COL, ROW

  // If needed write the full bar characters to display
  while(fullbars) {
    lcd.write(byte(CHAR_3BAR_FULL)); // Note: When calling lcd.write() a '0' must be cast as a byte
    fullbars--;
  }

  // If needed write the partial bar character to display
  if (partbars)
    lcd.write(byte(partbars));    // Note: when calling lcd.write() a '0' must be cast as a byte

  // If needed write the filler characters to display. This also erases any previous bar graph remnants
  while(fillers) {
    lcd.write(byte(CHAR_0BAR_FILLER));   // Note: when calling lcd.write() a '0' must be cast as a byte
    fillers--;
  }

}
```