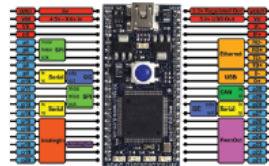




# FAST AND EFFECTIVE EMBEDDED SYSTEMS DESIGN

Applying the  
ARM mbed



Second Edition

Rob Toulson and Tim Wilmshurst



## Chapter 8: Liquid Crystal Displays

rt rev. 12.9.16

*If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:*

Toulson, R. & Wilmshurst, T. (2016). Fast and Effective Embedded Systems Design - Applying the ARM mbed (2<sup>nd</sup> edition), Newnes, Oxford, ISBN: 978-0-08-100880-5.

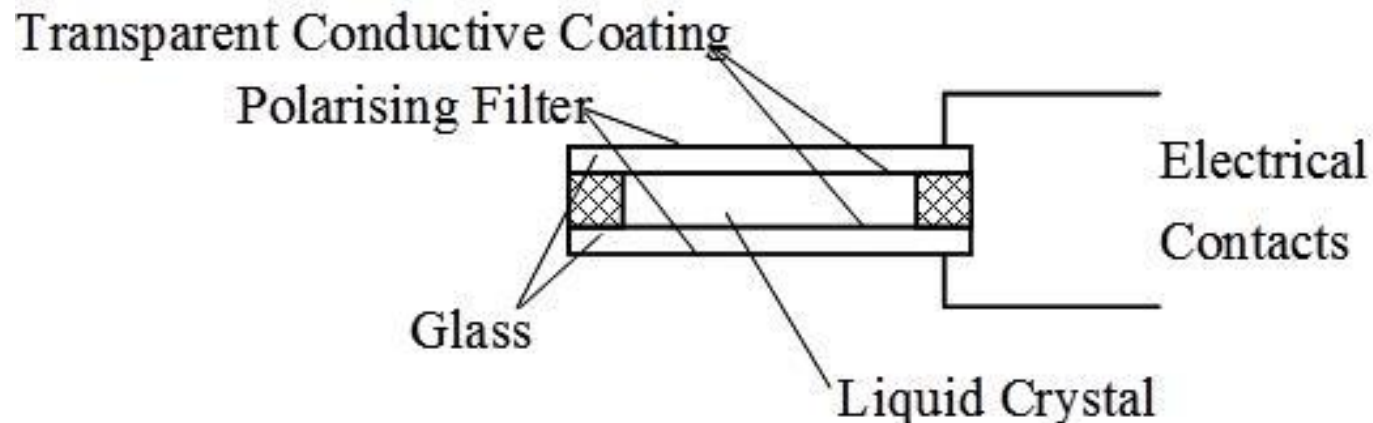
[www.embedded-knowhow.co.uk](http://www.embedded-knowhow.co.uk)

# Introducing liquid crystal technology

The liquid crystal is an organic compound which responds to an applied electric field by changing the alignment of its molecules, and hence the light polarisation which it introduces.

A small quantity of liquid crystal is contained between two parallel glass plates. A suitable field can be applied if transparent electrodes are located on the glass surface. In conjunction with the external polarising light filters, light is either blocked, or transmitted by the display cell.

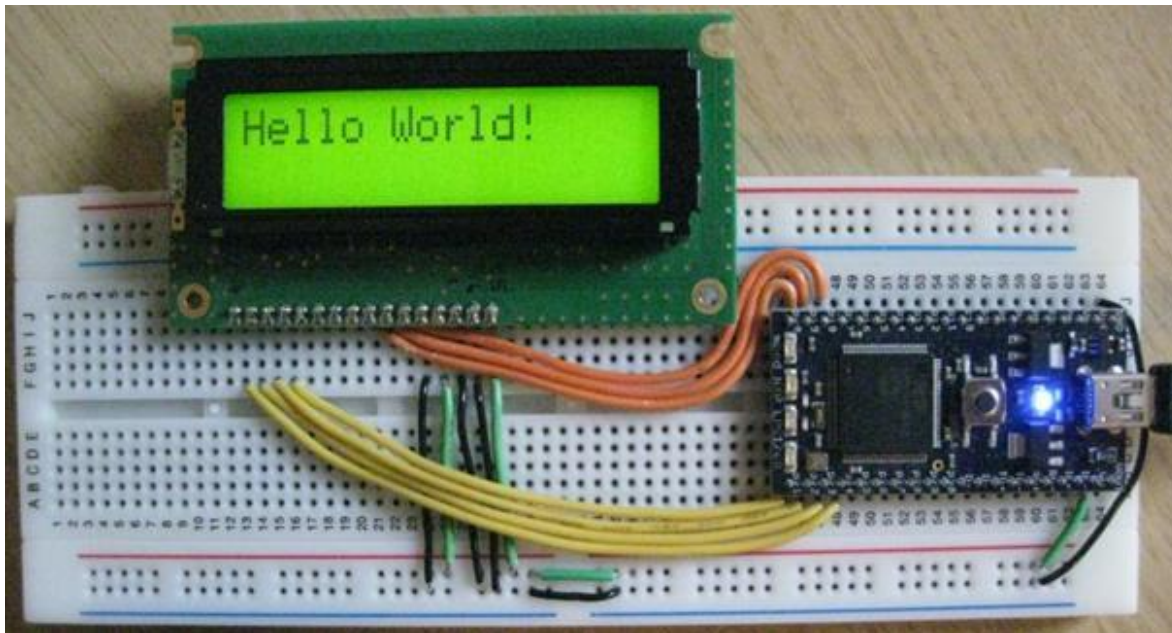
The electrodes can be made in any pattern desired, including single digits or symbols.



# Liquid crystal character displays

A popular, indeed ubiquitous, form of LCD is the character display. These are widely available from one line of characters to four or more, and are commonly seen on many domestic and office items, such as photocopiers, burglar alarms or DVD players.

Driving this complex array of tiny LCD dots is far from simple, so such displays always contain a hidden microcontroller, customised to drive the display. The first such controller to gain widespread acceptance was the Hitachi HD44780. While this has been superseded by others, they have kept the interface and internal structure of the Hitachi device.



# Liquid crystal character displays

The HD44780 contains an 80-byte RAM (Random Access Memory) to hold the display data, and a ROM (Read Only Memory) for generating the characters.

It has a simple instruction set, including instructions for initialisation, cursor control (moving, blanking, blinking), and clearing the display.

Communication with the controller is made via an 8-bit data bus, 3 control lines, and an enable/strobe line (E).

The HD44780 communication lines are shown below.

RS	Register Select:    0 = Instruction register    1 = Data Register
R/W	Selects read or write
E	Synchronises read and write operations
DB4 - DB7	Higher order bits of data bus; DB7 also used as Busy flag
DB0 - DB3	Lower order bits of data bus; not used for 4-bit operation

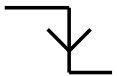
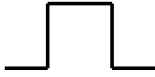
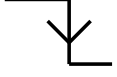

# Liquid crystal character displays

Data written to the HD44780 controller is interpreted either as instruction or as display data, depending on the state of the RS (Register Select) line.

The controller can be set up to operate in 8-bit or 4-bit mode.

In 4-bit mode only the four most significant bits of the bus are used, and two write cycles are required to send a single byte.

In both cases the most significant bit doubles as the Busy flag when a Read is undertaken.

RS	R/ $\overline{W}$	E	Action
0	0		Write instruction code
0	1		Read busy flag and address counter
1	0		Write data
1	1		Read data

# Using the PC1602F LCD display

We can interface the mbed processor to an external LCD, in order to display messages on the screen.

Interfacing an LCD requires a few involved steps to prepare the device and achieve the desired display. The following tasks must be considered in order to successfully interface the LCD:

- *Hardware integration:* we will need to connect the LCD to the correct mbed pins.
- *Modular coding:* as there are many processes that need to be completed, it makes sense to define LCD functions in modular files.
- *Initialising the LCD:* a specific sequence of control signals must be sent to the LCD in order to initialise it.
- *Outputting data:* we will need to understand how the LCD converts control data into legible display data.

We will use the 2x16 character Powertip PC1602F LCD, though a number of similar LCD displays can be found with the same hardware configuration and functionality.

# Using the PC1602F LCD display

The PC1602F display is a 2x16 character display with an on-board data controller chip and an integrated backlight.

The LCD display has 16 connections, defined as follows:

Pin Number	Pin Name	Function
1	V <sub>SS</sub>	Power supply (GND)
2	V <sub>DD</sub>	Power supply (5V)
3	V <sub>0</sub>	Contrast adjust
4	RS	Register select signal
5	R/IW	Data read / write
6	E	Enable signal
7	DB0	Data bus line bit 0
8	DB1	Data bus line bit 1
9	DB2	Data bus line bit 2
10	DB3	Data bus line bit 3
11	DB4	Data bus line bit 4
12	DB5	Data bus line bit 5
13	DB6	Data bus line bit 6
14	DB7	Data bus line bit 7
15	A	Power supply for LED back light (5V)
16	K	Power supply for LED back light (GND)

## Using the PC1602F LCD display

We will use the LCD in 4-bit mode. This means that only the upper 4 bits of the data bus (DB4-DB7) are connected.

The two halves of any byte are sent in turn on these lines. As a result the LCD can be controlled with only 7 lines, rather than the 11 lines which are required for 8-bit mode.

Every time a nibble (a 4-bit word is sometimes called a *nibble*) is sent, the E line must be pulsed or toggled on and off.

The display is initialised by sending control instructions to the configuration registers in the LCD.

This is done by setting RS and R/ $\overline{W}$  low, once the LCD has been initialised, display data can be sent by setting the RS bit high.

As before, the E bit must be pulsed for every nibble of display data sent.



# Connecting the PC1602F to the mbed

An mbed digital output should be attached to each of the LCD data pins that are used.

Four outputs are required to send the 4-bit instruction and display data and two outputs are required to manipulate the RS and E control lines.

The suggested interface configuration for connecting the mbed to the PC1602F is as shown opposite.

Note that, in simple applications, the LCD can be held in write mode, so R/|W can be tied permanently to ground (mbed pin 1).

If the R/|W line is tied to ground, then a 1ms delay between data transfers is adequate to ensure that all internal processes can complete before the next transfer.

mbed Pin number	LCD Pin Number	LCD Pin Name	Power Connections
1	1	V <sub>SS</sub>	0V
39	2	V <sub>DD</sub>	5V
1	3	V <sub>0</sub>	0V
19	4	RS	
1	5	R/ W	0V
20	6	E	
21	11	DB4	
22	12	DB5	
23	13	DB6	
24	14	DB7	
39	15	A	5V
1	16	K	0V

(Note: A = Anode, K = cathode of the back light LED)

# Using modular coding to interface the LCD display

Initialising and interfacing a peripheral device, such as an LCD, can be done effectively by using modular files to define various parts of the code.

We will therefore use three files for this application. The files required are:

- a main code file (**main.cpp**) which can call functions defined in the LCD feature file
- an LCD definition file (**LCD.cpp**) which will include all the functions for initialising and sending data to the LCD
- an LCD header file (**LCD.h**) which will be used to declare data and function prototypes

We will declare the following functions in our LCD header file

- **toggle\_enable( )**: a function to toggle/pulse the enable bit
- **LCD\_init( )**: a function to initialise the LCD
- **display\_to\_LCD( )**: a function to display characters on the LCD

# Using modular coding to interface the LCD display

Our **LCD.h** header file should therefore define the function prototypes as shown below

```
/* Program Example 8.1: LCD.h header file
*/

#ifndef LCD_H
#define LCD_H

#include "mbed.h"

void toggle_enable(void);           //function to toggle/pulse the enable bit
void LCD_init(void);               //function to initialise the LCD
void display_to_LCD(char value);   //function to display characters

#endif
```

# Initialising the display

A specific initialisation procedure must be programmed in order for the PC1602F display to operate correctly. Full details are provided in the Powertip PC1602F datasheet, but are summarised here.

We first need to wait a short period (approximately 20 ms), then set the RS and E lines to zero and then send a number of configuration messages to set up the LCD.

We then need to send configuration data to the

- Function Mode register
- Display Mode register
- Clear Display register

In order to initialise the display.

# Initialising the display

## Function Mode

To set the LCD function mode, the RS, R/W and DB0-7 bits should be set as shown below. Remember in 4-bit mode, data bus values are sent as two nibbles.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	BW	N	F	X	X

BW = 0 → 4 bit mode

BW = 1 → 8 bit mode

X = Don't care bits (can be 0 or 1)

N = 0 → 1 line mode

N = 1 → 2 line mode

F = 0 → 5×7 pixels

F = 1 → 5×10 pixels

If, for example, we send a binary value of 00101000 (0x28 hex) to the LCD data pins, this defines 4-bit mode, 2 line display and 5x7 dot characters.

In the given example we would therefore send the value 0x2, pulse E, then send 0x8, then pulse E again.

# Initialising the display

## Display Mode

The Display Mode control register must also be set up during initialisation. Here we need to send a command to switch the display on, and to determine the cursor function. The Display Mode register is defined as shown below.

RS	R/W
0	0

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	P	C	B

P = 0 → display off  
P = 1 → display on

C = 0 → cursor off  
C = 1 → cursor on

B = 0 → cursor no blink  
B = 1 → cursor blinking

## Clear Display

Before data can be written to the display, the display must be cleared, and the cursor reset to the first character in the first row (or any other location that you wish to write data to). The Clear Display command is shown below.

RS	R/W
0	0

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1

# Sending display data to the LCD

Characters are displayed by setting the RS flag to 1 (data setting), then sending a data byte describing the ASCII character to be displayed.

When communicating with displays, by sending a single ASCII byte, the display is informed which particular character should be shown.

The complete ASCII table is included with the LCD datasheet, but for interest some common ASCII values for display on the LCD are shown in the Table opposite.

It can be seen for example, that if we send the data value 0x48 to the display, the character 'H' will be displayed.

		Less significant bits (lower nibble)															
		0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
More significant bits (upper nibble)	0x0																
	0x1																
	0x2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
	0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	0x6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0x7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

# The complete LCD.cpp definition

```
/* Program Example 8.2: Declaration of objects and
functions in LCD.cpp file

*/
#include "LCD.h"
DigitalOut RS(p19);
DigitalOut E(p20);
BusOut data(p21, p22, p23, p24);

//initialise LCD function
void LCD_init(void){
    wait(0.02);          // pause for 20 ms
    RS=0;                // set low to write control data
    E=0;                 // set low

    //function mode
    data=0x2;            // 4 bit mode (packet 1, DB4-DB7)
    toggle_enable();
    data=0x8;            // 2-line, 7 dot (packet 2, DB0-DB3)
    toggle_enable();

    //display mode
    data=0x0;            // 4 bit mode (packet 1, DB4-DB7)
    toggle_enable();
    data=0xF;            // display on, cursor on, blink on
    toggle_enable();

    //clear display
    data=0x0;            //
    toggle_enable();
    data=0x1;            // clear
    toggle_enable();
}

//... continued on right hand side
```

```
// ... continued

void toggle_enable(void){
    E=1;
    wait(0.001);
    E=0;
    wait(0.001);
}

//display function
void display_to_LCD(char value){
    RS=1;                // set high to write char data
    data=value>>4;        // shifted right 4 =upper nibble
    toggle_enable();
    data=value;           //bitmask with 0x0F =low nibble
    toggle_enable();
}
```



# Utilising the LCD functions

We can now develop a main control file (**main.cpp**) to utilise the LCD functions described above.

The following Program Example initialises the LCD, displays the word “HELLO” and then displays the numerical characters from 0 to 9.

```
/* Program Example 8.3 Utilising LCD functions in the main.cpp file */
#include "LCD.h"

int main() {
    LCD_init();           // call the initialise function
    display_to_LCD(0x48); // 'H'
    display_to_LCD(0x45); // 'E'
    display_to_LCD(0x4C); // 'L'
    display_to_LCD(0x4C); // 'L'
    display_to_LCD(0x4F); // 'O'
    for(char x=0x30;x<=0x39;x++){
        display_to_LCD(x); // display numbers 0-9
    }
}
```

**Read further:**  
Moving the display pointer to a specified location in Section 8.2.8

# Using the mbed TextLCD library

There is an mbed library which makes use of an alphanumeric LCD much simpler and quicker to program.

The mbed **TextLCD** library is also more advanced than the simple functions we have created, in particular the **TextLCD** library performs the laborious LCD setup routine for us.

The **TextLCD** definition also tells the LCD object which pins are used for which functions.

The object definition is defined in the following manner:

```
TextLCD lcd(int rs, int e, int d0, int d1, int d2, int d3);
```

We need to ensure that our pins are defined in the same order. For our particular hardware setup (described in Table 8.2) this will be:

```
TextLCD lcd(p19, p20, p21, p22, p23, p24);
```

# Using the mbed TextLCD library

Simple **printf( )** statements are used to display characters on the LCD screen.

The **printf( )** function allows formatted print statements to be used. This means we are able to send text strings and formatted data to a display, meaning that a function call is not required for each individual character.

When using the **printf( )** function with the mbed **TextLCD** library, the display object's name is also required, so if the **TextLCD** object is defined with the object name **lcd**, then a "Hello World" string can be written as follows:

```
lcd.printf("Hello World!");
```

When using predefined mbed libraries, such as **TextLCD**, the library file needs importing to the mbed program. The mbed **TextLCD** library can be accessed from the following location:

<http://mbed.org/users/simon/libraries/TextLCD/livod0>

The library header file must also be included with the **#include** statement in our **main.cpp** file or relevant project header files.

# Using the mbed TextLCD library

```
/*Program Example 8.5:  TextLCD library Helo World example
                                                                    */
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0,d1,d2,d3

int main() {
    lcd.printf("Hello World!");
}
```

# Using the mbed TextLCD library

The Program Example below displays a count variable on the LCD display. The count variable increments every second.

```
/* Program Example 8.6: LCD Counter example */

#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p19, p20, p21, p22, p23, p24); // rs, e, d0, d1, d2, d3
int x=0;

int main() {
    lcd.printf("LCD Counter");
    while (1) {
        lcd.locate(5,1);
        lcd.printf("%i",x);
        wait(1);
        x++;
    }
}
```

# Displaying analog input data on the LCD

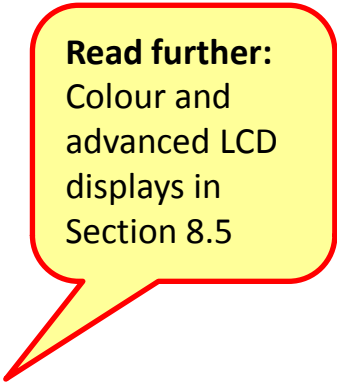
We can connect a potentiometer between 3.3V and 0V with the wiper connected to pin 18.

We can multiply the analog input value by 100 to display a percentage between 0 and 100%, as shown in the Program Example below.

An infinite loop can be used so that the screen updates automatically. To do this it is necessary to clear the screen and add a delay to set the update frequency.

```
/*Program Example 8.7: Display analog input data
*/
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0, d1,d2,d3
AnalogIn Ain(p17);
float percentage;

int main() {
    while(1){
        percentage=Ain*100;
        lcd.printf("%1.2f",percentage);
        wait(0.002);
        lcd.cls();
    }
}
```



**Read further:**  
Colour and  
advanced LCD  
displays in  
Section 8.5

# Voltmeter Exercise

Create a program to make the mbed and display act like a standard voltmeter, as shown below. Note the following:

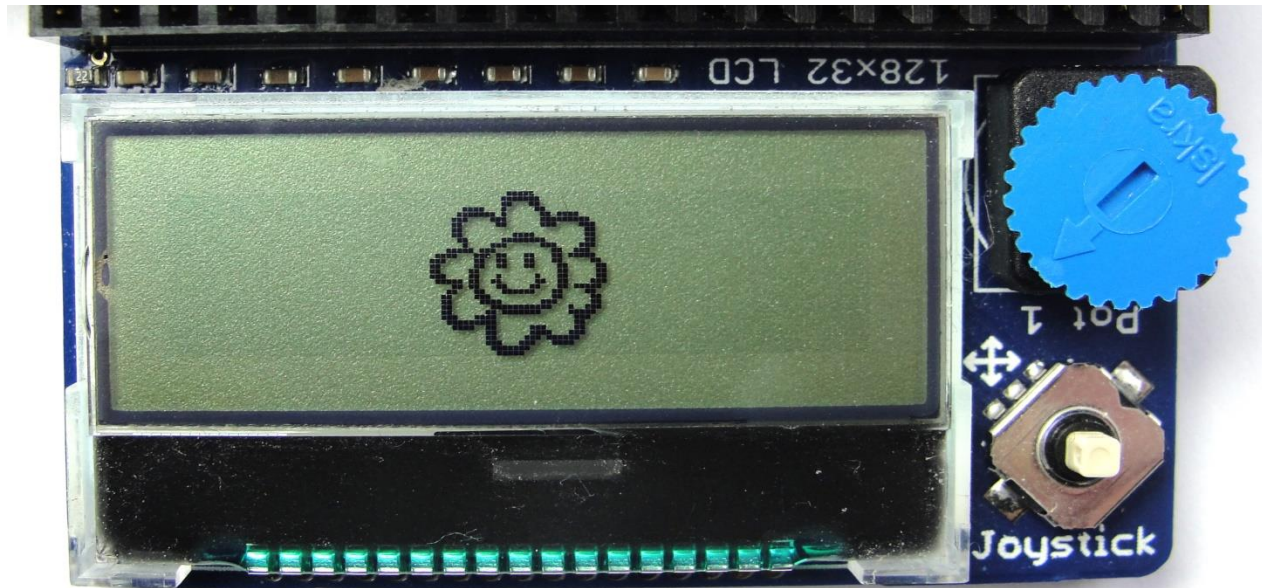
- You will need to convert the 0.0 - 1.0 analog input value to a value which represents 0 - 3.3 Volts.
- An infinite loop is required to allow the voltage value to continuously update as the potentiometer position changes.
- Check the display with the reading from an actual voltmeter – is it accurate?
- Increase the number of decimal places that the voltmeter reads too. Evaluate the noise and accuracy of the voltmeter readings with respect to the mbed's ADC resolution.



# Pixel graphics – implementing the NHD-C12832 display

A more advanced LCD display allows the programmer to set or clear each individual pixel on the screen. For example the NHD-C12832 included on the mbed app board

The C12832 is an LCD matrix of 128x32 pixels, allowing more intricate images and text messages to be displayed.





# Pixel graphics – implementing the NHD-C12832 display



Formatted text can be printed to the screen by using the **printf( )** function as defined in Table 8.5 in the textbook. Program Example 8.8 prints a simple formatted string that continuously counts up. Note the use of the **cls( )** function to clear the screen at the start of the program, and the locate function to set the position for the text to be drawn at.

```
/*Program Example 8.8: Displaying a formatted string on the NHD-C12832
*/
#include "mbed.h"      // Basic Library required for onchip peripherals
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11); // Initialize lcd
int main(){
    int j=0;
    lcd.cls();           // clear screen
    while(1){
        lcd.locate(10,10); // set location to x=10, y=10
        lcd.printf("Counter : %d",j); // print counter value
        j++;              // increment j
        wait(0.5);        // wait 0.5 seconds
    }
}
```

# Pixel graphics – implementing the NHD-C12832 display



It is also possible to set pixels individually. Program Example 8.9 draws a small cross on the LCD display with a centre point at location **x=10, y=10**.

```
/*Program Example 8.9: Setting individual pixels on the NHD-C12832
                                                                    */
#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11); // Initialize lcd

int main(){
    lcd.cls();           // clear screen
    lcd.pixel(10,9,1);   // set pixel 1
    lcd.pixel(10,10,1);  // set pixel 2
    lcd.pixel(10,11,1);  // set pixel 3
    lcd.pixel(9,10,1);   // set pixel 4
    lcd.pixel(11,10,1);  // set pixel 5
    lcd.copy_to_lcd();   // Send pixel data to screen
}
```

App Board

When displaying this array of data on an LCD, pixels represented with an X can be set while pixels represented with a dash will be left clear

[illegible]

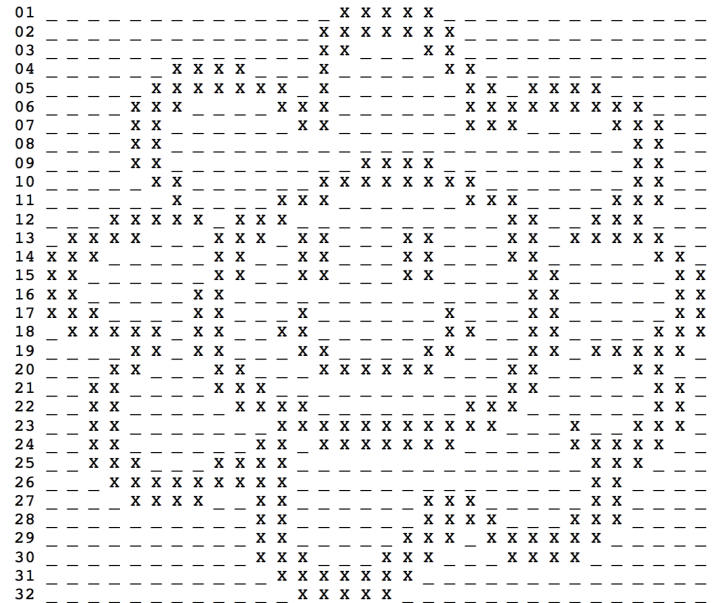
# Pixel graphics – implementing the NHD-C12832 display



The image shown can be defined by an array of binary values that represent the status of each pixel.

This type of binary image data is often described as a *bitmap*.

The table below shows how the bitmap data is constructed in binary form, by considering line 18 as an example.



18	_ X X X X X _ X	X _ _ X X _ _ _	_ _ _ X X _ _ X	X _ _ _ _ X X X
18	0 x 7 D	0 x 9 8	0 x 1 9	0 x 8 7

It can be seen that each row of data is defined as a number of consecutive 8-bit values. The first 8 pixel values in row 18 represent the binary value b01111101 or 0x7D hex.

The code to implement this flower bitmap on the app board is given in the textbook.

# Color LCD displays

For color displays, each pixel is made up of three subpixels for red, green and blue.

Each subpixel can be set to 256 different shades of its color, so it is therefore possible for a single LCD pixel to display  $256 * 256 * 256 = 16.8$  million different colors.

The pixel color is referred to by a 24-bit value where the highest 8-bits define the red shade, the middle 8 bits define the green shade and the lower 8 bits represent the blue shade, as shown in the table alongside

Color	24-bit value
Red	0xFF0000
Green	0x00FF00
Blue	0x0000FF
Yellow	0xFFFF00
Orange	0xFF8000
Purple	0x800080
Black	0x000000
White	0xFFFFFF



## Read further:

Learn how to interface the uLCD-144-G2 color LCD display in Section 8.6 of the textbook.

# Mini-project – Digital spirit level

Design, build and test a digital spirit level based on the mbed.

Use an ADXL345 accelerometer to measure the angle of orientation in two planes, a digital push-to-make switch to allow calibration and zeroing of the orientation, and a colour LCD to output the measured orientation data, in degrees.

To help you proceed consider the following:

- Design your display to show a pixel or image moving around the LCD screen with respect to the orientation of the accelerometer.
- Add the digital switch to allow simple calibration and zeroing of the data.
- Improve your display output to give accurate measurements of the 2-plane orientation angles in degrees from the horizontal (i.e. horizontal = 0°).
- How accurate are your x and y readings? Set up a number of know angles and test your spirit level; continuously improve your code until accurate readings are achieved at all angles in both axes.

# Chapter quiz questions

1. What are the advantages and disadvantages associated with using an alphanumeric liquid crystal display in an embedded system?
2. What types of cursor control are commonly available on alphanumeric LCDs?
3. How does the mbed **BusOut** object help to simplify interfacing an alphanumeric display?
4. What is the function of the E input on an alphanumeric display such as the PC1602F?
5. What does the term ASCII refer to?
6. What are the ASCII values associated with the numerical characters from 0-9?
7. Referring to the **TextLCD** library, describe the C code required to display the value of a floating point variable called 'ratio' to 2 decimal places in the middle of the second row of a 2x16 character alphanumeric display?
8. What is a bitmap and how can it be used to display images on an LCD display?
9. List and describe five practical examples of a color LCD used in an embedded system.
10. If a color LCD is filled to a single background color, what colors will the following 24-bit codes give:
  - a. 0x00FFFF
  - b. 0x00007F
  - c. 0x7F7F7F

# Chapter Review

- Liquid Crystal Displays (LCDs) use an organic crystal which can polarise and block light when an electric field is introduced.
- Many types of LCDs are available and, when interfaced with a microcontroller, they allow digital control of alphanumeric character displays and high resolution colour displays.
- The PC1602F is a 16 column by 2 row character display which can be controlled by the mbed.
- Data can be sent to the LCD registers to initialise the device and to display character messages.
- Character data is defined using the 8-bit ASCII table.
- The mbed **TextLCD** library can be used to simplify working with LCDs and allow the display of formatted data using the **printf( )** function.
- The NHD-C12832 display, which is installed on the mbed application board, has 128x32 pixels, allowing graphics and images, as well as alphanumeric text, to be displayed.
- Color LCDs, such as the uLCD-144-G2, frequently transfer data through a serial interface, with each pixel given a 24-bit color setting.