

Chapter 8: Liquid Crystal Displays

rt rev. 12.03.2024

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

Wilmshurst, T., Spink, T. & Toulson, R. (2024). Fast and Effective Embedded Systems Design – From Bits and Bytes to IoT, with the ARM Mbed (3rd edition), Newnes, Oxford, ISBN: 978-0323951975.

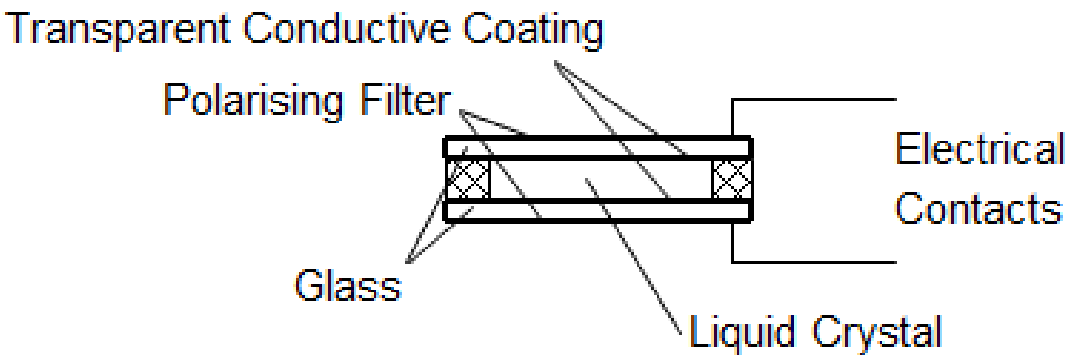
www.embedded-knowhow.co.uk

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 While this has been superseded by others, they have kept the interface and internal structure of the Hitachi device.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.

Signal Name	Signal Function
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

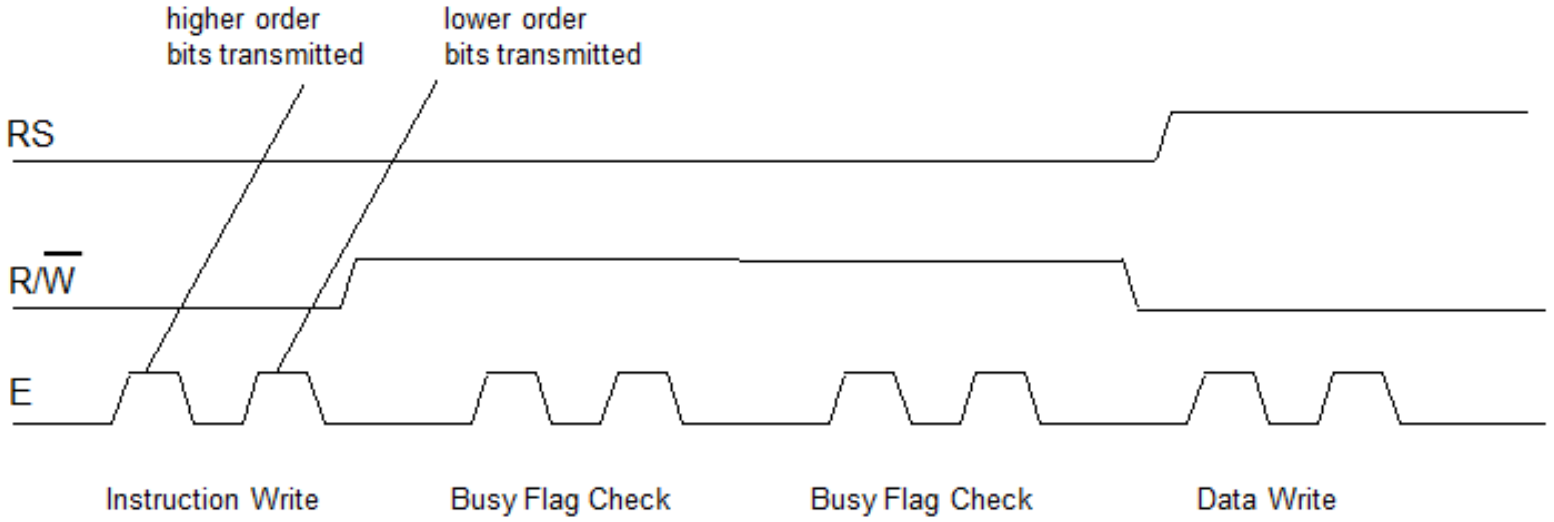
Data written to the HD44780 controller is interpreted either as instruction or as display data, depending on the state of the RS (Register Select) line.

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 (as below).

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

An Mbed development board can be interfaced to an external LCD. Here the 2x16 character Powertip PC1602F LCD is used as example.

This LCD has 16 connections, applied as follows:

PC1602F Pin Number	Pin Name	Function	Mbed LPC1768 Pin number used	Nucleo F401RE Pin number used
1	V_{SS}	Power supply (GND)	1	GND
2	V_{DD}	Power supply (5V)	39	5V
3	V_0	Contrast adjust (nc on recent devices)	-	-
4	RS	Register select	19	D9
5	R/IW	Data read / write	1	GND
6	E	Enable	20	D8
7	DB0	Data bus line bit 0	nc	nc
8	DB1	Data bus line bit 1	nc	nc
9	DB2	Data bus line bit 2	nc	nc
10	DB3	Data bus line bit 3	nc	nc
11	DB4	Data bus line bit 4	21	D4
12	DB5	Data bus line bit 5	22	D5
13	DB6	Data bus line bit 6	23	D6
14	DB7	Data bus line bit 7/ busy flag	24	D7
15	A	Power supply for LED back light (5V)	39	5V, 3V3 or nc
16	K	Power supply for LED back light (GND)	1	GND or nc if 15 is nc

Using the PC1602F LCD

In the examples which follow the LCD is used in 4-bit mode, so only the upper 4 bits of the data bus (DB4-DB7) are connected.

Two simplifications are also applied – the busy flag is not accessed, and R/|W is tied permanently to ground; a 1 ms delay between data transfers is used to ensure that all internal processes can complete before the next action.

The display is initialised by sending control instructions to the configuration registers in the LCD. This is done by setting RS and R/|W low. Once the LCD has been initialised, display data can be sent by setting the RS bit high. For all transfers, the E bit must be pulsed for every nibble of data sent.

Using Modular Coding to Control the LCD

Three files are used for this application:

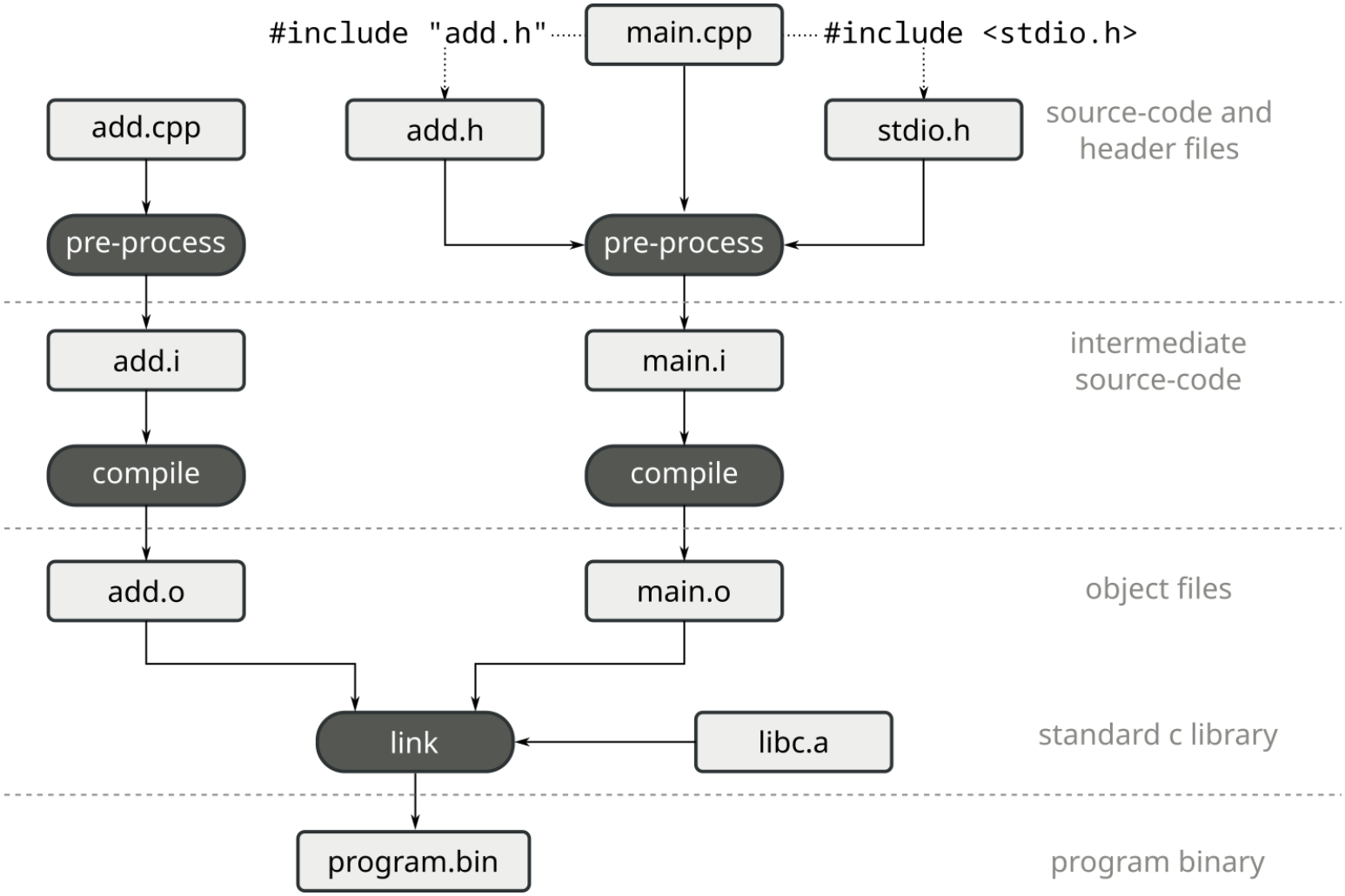
- a main code file (**main.cpp**) which can call functions defined in the LCD definition file,
- an LCD definition file (**LCD.cpp**) which will include all the functions for initialising and sending data to the LCD,
- an LCD header file (**LCD.h**) which will be used to declare data and function prototypes.

The following functions will be declared in the LCD header file:

- **toggle_enable( )**: a function to pulse the E bit,
- **LCD_init( )**: a function to initialise the LCD,
- **display_to_LCD( )**: a function to display characters on the LCD.

Modular Coding Background (from Appendix B)

In this programming example we explore aspects of this compilation diagram, described further in Section B11, Appendix B.



The **LCD.h** Header File

This **LCD.h** header file defines the function prototypes:

```
/* Program Example 8.1: LCD.h header file
  Runs on either LPC1768 or F401RE with pins shown in Table 8.1
                                                    */
#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.

In order to initialise the display we then need to send configuration data to the:

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

Initialising the Display: Function Mode

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
0	0

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	BW	N	F	X	X

BW = 0 → 4 bit mode
BW = 1 → 8 bit mode

N = 0 → 1 line mode
N = 1 → 2 line mode

F = 0 → 5×7 pixels
F = 1 → 5×10 pixels

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

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 and Clear

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 **LCD.cpp** File (1)

The LCD definition file (**LCD.cpp**) contains the three C functions, **toggle_enable( ), LCD_init( )** and **display_to_LCD( )**. Note that the **toggle_enable( )** function has two 1 ms delays, removing the need to monitor the busy flag.

```
/* Program Example 8.2: Declaration of objects and functions in LCD.cpp file
Runs on either LPC1768 or F401RE with pins shown below
                                                                        */
  #include "LCD.h"
  DigitalOut RS(p19);                 //Use D9 for F401RE
  DigitalOut E(p20);                  //Use D8 for F401RE
  BusOut data(p21, p22, p23, p24);     //Use D4,D5,D6,D7 for F401RE

  void toggle_enable(void){
    E = 1;
    wait_us(1000);
    E = 0;
    wait_us(1000);
  }

  //initialise LCD function
  void LCD_init(void){
    wait_us(20000);           // pause for 20 ms
    RS = 0;                     // set low to write control data
    E = 0;                      // set low

  //continued over
```

The **LCD.cpp** File (2)

```
//function mode
  data = 0x2;              // 4 bit mode (data packet 1, DB4-DB7)
  toggle_enable();
  data = 0x8;              // 2-line, 7 dot char (data packet 2, DB0-DB3)
  toggle_enable();
//display mode
  data = 0x0;               // 4 bit mode (data 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();
}

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

Calling the LCD Functions

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

This 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
Runs on either LPC1768 or F401RE with pins shown in Example 8.2
                                                                    */

#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
  }
}
```

Adding Data to a Specified Location (1)

The display has a memory which is mapped so that each display digit has a unique memory address, shown below. A display pointer can be set before data is transferred; the data sent will then appear in the position specified.

For example, if the display pointer is set to address 0x40, data will be displayed at the first position on the second line. To change the pointer address, the 6-bit address value is sent in a control byte with bit 7 also set, shown further below.

Display Position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Display Pointer Address	**Row 1**	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	Row 2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

RS	R/	W		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0		1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

AC6-AC0 contain the 6-bit display pointer address

Adding Data to a Specified Location (2)

We can create a new function to set the value of the display pointer, called here **set_location( )**. Notice that bit DB7 is set by ORing the location value with 0x80.

```
/* Program Example 8.4 function to set the display location. Parameter
"location" holds address of display unit to be selected
                                                                    */
void set_location(char location){
    RS = 0;
    data = (location|0x80) >> 4;             // upper nibble
    toggle_enable();
    data = location&0x0F;                  // lower nibble
    toggle_enable();
}
```

Using the Mbed TextLCD Library

There are a number of software libraries available that make an alphanumeric LCD much simpler and quicker to program. For example, the Mbed **TextLCD** library is more advanced than the simple functions we have just created.

Simple **printf( )** statements are used to display characters and strings on the LCD, so a function call is not required for each character. A summary is given here.

Functions	Usage
Constructor	
TextLCD *name*(int *rs*, int *e*, int *d0*, int *d1*, int *d2*, int d3)	Create object, choosing name, specifying pins as indicated, and initialise.
Member Functions	
void **character**(int *column*, int *row*, int c)	Write character c, to specified column and row.
void **locate**(int column, int *row*)	Locate cursor at specified column and row.
void **cls**()	Clear screen, and set cursor to 0.
void **writeByte**(int *value*)	Write value with unchanged setting of RS.
void **writeCommand**(int *command*)	Set RS to 0 and write command.
void **writeData**(int *data*)	Set RS to 1 and write data.

Using the Mbed TextLCD library

This is a simple “Hello World” example using the **TextLCD** library.

```
/*Program Example 8.5:  TextLCD library example
  Runs on either LPC1768 or F401RE with pins shown below
                                                          */
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0,d1,d2,d3
         //Use D9,D8,D4,D5,D6,D7 or own choice for F401RE

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
 Runs on either LPC1768 or F401RE with pins shown below
                                                    */
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p19, p20, p21, p22, p23, p24); // rs,e,d0,d1,d2,d3
         //Use D9,D8,D4,D5,D6,D7 or own choice for F401RE
int x = 0;

int main() {
  lcd.printf("LCD Counter");
  while (true) {
    lcd.locate(5,1);     //locate the cursor on the 2nd row, 6th digit
    lcd.printf("%i",x);  //print the value of x as an integer
    wait_us(1000000);    //wait 1 second
    x++;
  }
}
```

Displaying Analog Input Data on the LCD

We can display analog input data on the LCD, as shown in this example. The program loops, and every time clears the screen before a new value is displayed. As floating point values are to be printed, a **mbed_app.json** file must be created.

```
/*Program Example 8.7: Display analog input data
 Runs on either LPC1768 or F401RE with pins shown below
*/

#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(p19, p20, p21, p22, p23, p24); //rs,e,d0,d1,d2,d3
         //Use D9,D8,D4,D5,D6,D7 or own choice for F401RE
AnalogIn Ain(p17);  //Use A5 for F401RE
float percentage;

int main() {
  while(true){
    percentage = Ain*100;
    lcd.printf("%1.2f",percentage);
    ThisThread::sleep_for(20ms);
    lcd.cls();
  }
}
```

a) the main.cpp file

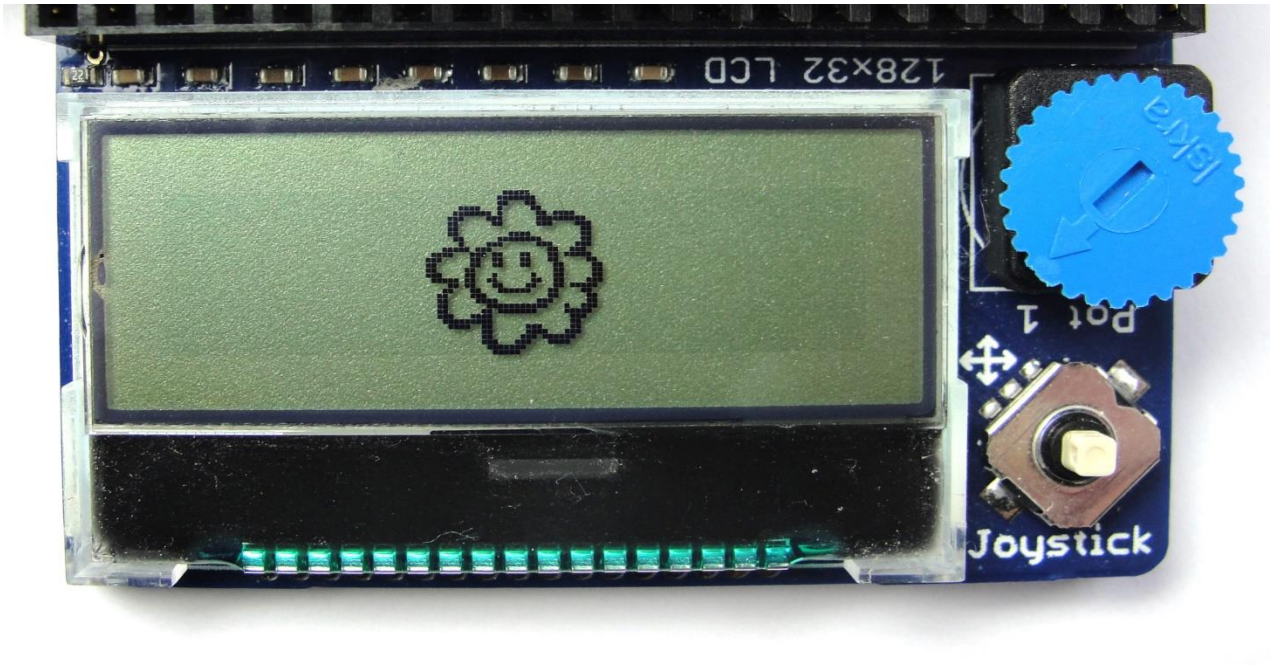
```
{
    "target_overrides":{
        "*": {
            "target.printf_lib": "std"
        }
    }
}
```

b) the mbed_app.json file

Pixel Graphics – Implementing the NHD-C12832 Display

A more advanced LCD, for example the NHD-C12832, allows the programmer to set or clear each individual pixel on the screen.

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

The C12832 can be controlled by importing and using the **C12832** library by developer Kevin Anderson. The main features are shown below.

Functions	Usage
`C12832 (mosi,sck,rst,a0,cs1)`	Create a C12832 LCD object with Mbed pin definitions. Pins are: data, clock, reset, register select (0=inst., 1=data), chip select.
`void setmode(int mode)`	mode: NORMAL = standard operation mode: XOR = write toggles pixel status
`void invert(unsigned int x)`	x=1 inverts the pixel "color", black to white etc. x=0 standard
`void cls(void)`	Clear the LCD
`void set_contrast(unsigned int x)`	x = contrast value (Value of 10 - 35 will be visible)
`printf(string)`	Prints formatted string to LCD display
`void locate(int x, int y)`	x, y sets display cursor position
`void set_font(unsigned char* font)`	font = Small_6, Small_7, Arial_9, Arial12x12, Arial24x23
`int putc(int c);`	Print the char c on the actual cursor position. Returns this value.
`void character(int x, int y, int c)`	Print the char c at position x, y.
`void line(int x0, int y0, int x1, int y1, int c)`	Draw a single pixel line from x0, y0 to x1, y1. (c = color 0 or 1)
`void rect(int x0, int y0, int x1, int y1, int c)`	Draw a rectangle from x0, y0 to x1, y1. (c = color 0 or 1)
`void fillrect(int x0, int y0, int x1, int y1, int c)`	Draw a filled rectangle from x0, y0 to x1, y1. (c = color 0 or 1)
`void circle(int x, int y, int r, int c)`	draw a circle with x, y center and radius r. (c = color 0 or 1)
`void fillcircle(int x, int y, int r, int c)`	draw a filled circle with x,y center and radius r. (c = color 0 or 1)
`void pixel(int x, int y,int c)`	set a single pixel at x, y. (c = color 0 or 1). Note, this function does not update the screen until copy_to_lcd is called.
`Void print_bm(Bitmap bm, int x, int y)`	Print bitmap object. Screen updated when copy_to_lcd() is called.
`void copy_to_lcd(void)`	Copy buffer contents to the screen

Pixel Graphics – Implementing the NHD-C12832 Display

This program example prints a simple formatted string to the display that continuously counts up. Notice the use of the **cls( )** function to clear the screen at the start of the program, and the **locate( )** function to set the text start position.

```
/*Program Example 8.8: Displaying a formatted string on the NHD-C12832
  Runs on either LPC1768 or F401RE with pins shown below
*/

#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11); //use D11,D13,D12,D10,D9 for F401RE
//For F401RE, connect also GND pins, and 3.3V output to app board pin 40

int main(){
  int j = 0;
  lcd.cls();                  // clear screen
  while(true){
    lcd.locate(10,10);            // set location to x=10, y=10
    lcd.printf("Counter : %d",j); // print counter value
    j++;                          // increment j
    ThisThread::sleep_for(500ms); // 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 C12832 LCD
 Runs on either LPC1768 or F401RE with pins shown below
*/

#include "mbed.h"
#include "C12832.h"

C12832 lcd(p5, p7, p6, p8, p11); //use D11,D13,D12,D10,D9 for F401RE
//For F401RE board, connect also GND pins, and 3.3V output to app board pin 40

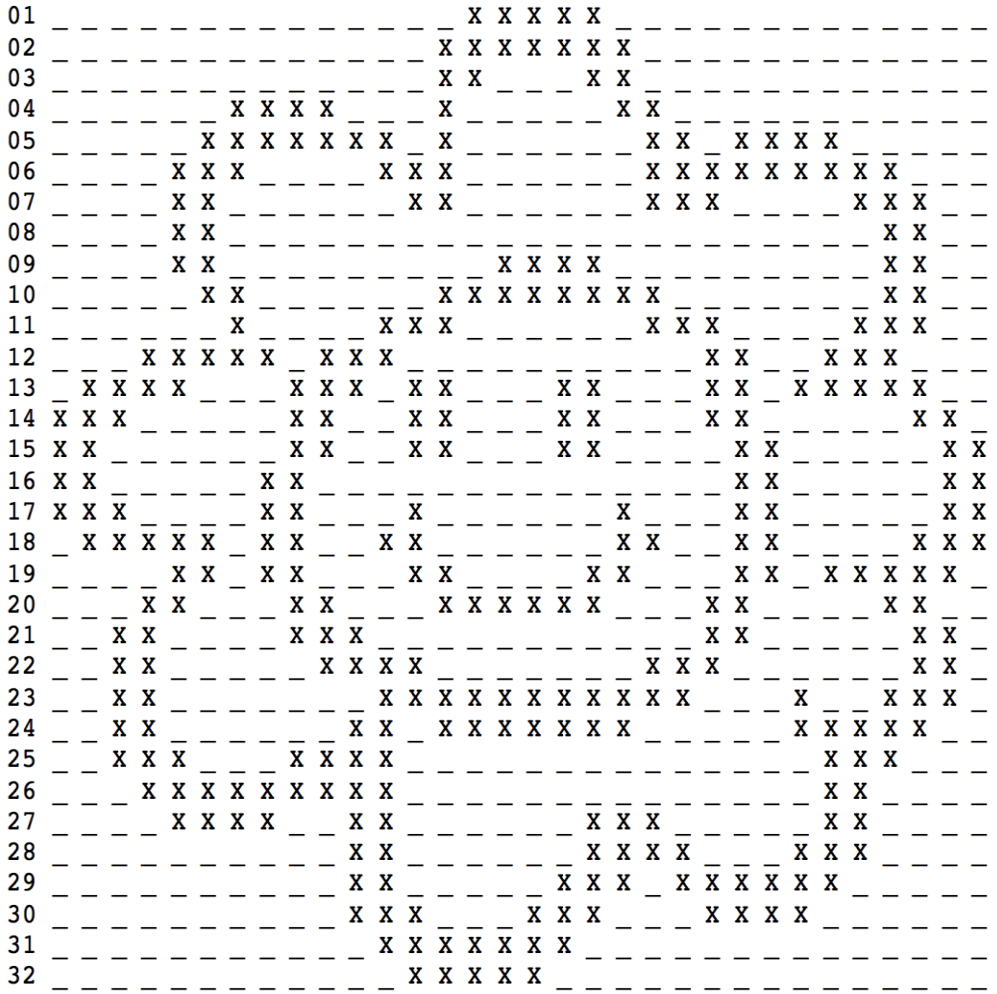
int main(){
  lcd.cls();           // clear screen
  lcd.pixel(10,9,1);   // set pixel 1
  lcd.pixel(10,10,1); // set pixel 2
  lcd.pixel(10,11,1); // set pixel 3
  lcd.pixel(9,10,1);   // set pixel 4
  lcd.pixel(11,10,1); // set pixel 5
  lcd.copy_to_lcd();  // Send pixel data to screen
}
```

Pixel Graphics – Implementing the NHD-C12832 Display

With the ability to set and clear individual pixels, it is possible to start displaying graphics and images.

The image below shows a 32 x 32 pixel image that represents a flower, with the lines numbered on the left hand side.

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

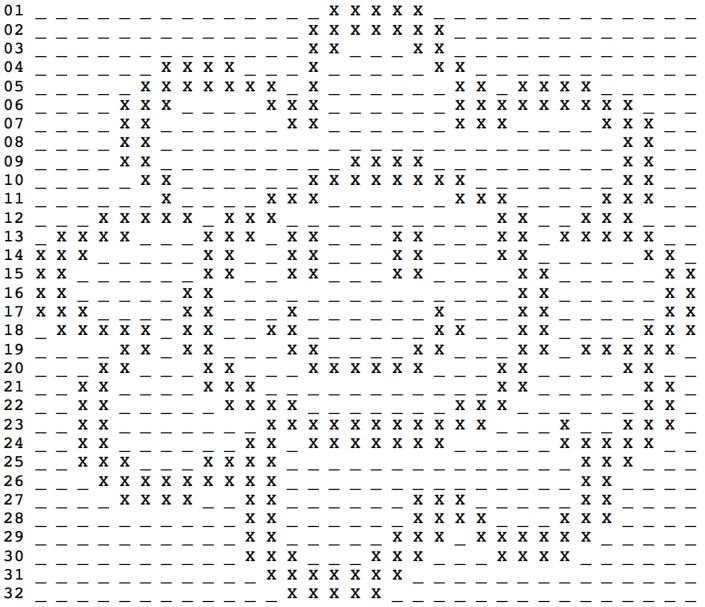


Pixel Graphics – Implementing the NHD-C12832 Display

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.

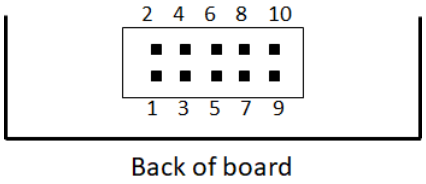
The pixel color is referred to by a 24-bit value where the highest 8-bits define the red shade, the middle 8 bits define the green shade and the lower 8 bits represent the blue shade, as shown in the table alongside

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

The uLCD-144-G2 LCD

This display is a compact 1.44” LCD color screen, with wide-ranging capability.

The display uses a UART serial interface to receive data from a microprocessor. To limit data transfer required, it has an optional SD card holder. Video clips, special fonts or images can be held on the card, and called up by the program.





Function	uLCD-144-G2 Pin	Mbed LPC1768 Pin	Nucleo F401RE Pin
Power (5V)	1	39	5V
TX / RX	3	10	D2
RX / TX	5	9	D8
Ground	7	1	GND
Reset	9	11	D9

The uLCD-144-G2 LCD

The **4DGl-uLCD-SE** library, by developer Jim Hamblen, can be used to interface the display. The library shares many similar API functions with the **C12832** library already used.

This program example prints formatted text in red, green and blue using the library's **color( )** and **printf( )** functions.

```
/*Program Example 8.13: Displaying color text on the uLCD-144-G2
     Works on both LPC1768 and F401RE, using pins shown.
*/

#include "mbed.h"
#include "uLCD_4DGL.h"

//Create the uLCD_4DGL object, defining respectively tx, rx, reset.
uLCD_4DGL uLCD(p9,p10,p11); //Use D8, D2, D9 for F401RE

int main(){
  uLCD.color(0xFF0000);                // set text color to red
  uLCD.printf("Text in RED\n");
  uLCD.color(0x00FF00);                // set text color to green
  uLCD.printf("Text in GREEN\n");
  uLCD.color(0x0000FF);                // set text color to blue
  uLCD.printf("Text in BLUE\n");
}
```

The uLCD-144-G2 LCD

This program example draws a series of concentric blue circles with increasing radii. The circles are drawn with a shared center at pixel (64,64), the center of the display. The circles increment in radius by 3 pixels on each iteration until the maximum radius of 64 is reached.

```
/*Program Example 8.14: drawing concentric color circles on the uLCD_4DGL
     Works on both LPC1768 and F401RE, using pins shown.                    */

#include "mbed.h"
#include "uLCD_4DGL.h"        // library also supports uLCD-144-G2 variant

//Create the uLCD_4DGL object, defining respectively tx, rx, reset.
uLCD_4DGL uLCD(p9,p10,p11); //Use D8, D2, D9 for F401RE

int main(){
  while(true) {
    for (int r = 0; r <= 64; r += 3) {    // increment r by 3 each time
      uLCD.circle(64, 64, r, 0x0000FF);   // draw blue circle of radius r
      ThisThread::sleep_for(100ms);
    }
    uLCD.cls();
  }
}
```

Chapter Review

- Liquid Crystal Displays (LCDs) use an organic crystal which can polarise and block light when subjected to an electric field.
- Many types of LCDs are available and, when interfaced with a microcontroller, they allow digital control of alphanumeric character displays, or pixel-oriented high-resolution monochrome and color displays.
- The PC1602F is a 16 column by 2 row character LCD which can be controlled by an Mbed development board.
- Character data can be defined using the 8-bit ASCII table.
- The Mbed **TextLCD** library can be used to simplify working with alphanumeric LCDs, it allows 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 frequently allocate each pixel a 24-bit color setting, with consequent high data transfer demands.
- The **uLCD-144-G2** display module is a color LCD screen that uses a UART serial communications interface to receive display data from a microcontroller, such as an Mbed development board.