CHAPTER 8

Liquid Crystal Displays

Chapter Outline

8.1 Display Technologies

Light-emitting diodes (LEDs), particularly individual LEDs and seven-segment displays, have already been encountered in previous chapters. On their own, LEDs can only inform us of a few different states, for example to indicate whether something is on or off, or whether a variable is cleared or set. It is possible to be innovative in the way LEDs are used, for example, different flashing speeds can be used to represent different states, but clearly there is

Fast and Effective Embedded Systems Design. DOI: 10.1016/B978-0-08-097768-3.00008-8
Copyright © 2012 Elsevier Ltd. All rights reserved.

a limit to how much information a single LED can communicate. With seven-segment displays it is possible to display numbers and a few characters from the alphabet, but there is a problem with seven-segment displays in that in simple use they require a microcontroller output for each LED segment. The multiplexing technique described in Section 3.6.3 can be used, but a limitation is still seen. Equally, LEDs are relatively power hungry, which is an issue for power-conscious designs. Embedded systems designers need to be clever in the way they use the available input and output capabilities of a microcontroller, but obviously the use of LEDs as a display has its limitations. If text-based messages with a number of characters are required to be communicated with a user, then a more advanced display type is needed, such as the liquid crystal displays (LCDs) that are commonly used in consumer electronics.

8.1.1 Introducing Liquid Crystal Technology

Liquid crystal displays are of great importance these days, both in the electronic world in general, and in the embedded system environment. Their main advantages are their extremely low power requirements, light weight and high flexibility of application. They have been one of the enabling technologies for battery-powered products such as the digital watch, the laptop computer and the mobile telephone, and are available in a huge range of indicators and displays. However, LCDs do have some disadvantages; these include limited viewing angle and contrast in some implementations, sensitivity to temperature extremes, and very high cost for the more sophisticated graphical displays.

Although LCDs do not emit light, they can reflect incident light, or transmit or block backlight. The principle of an LCD is illustrated in Figure 8.1. 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 polarization that 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 polarizing light filters, light is either blocked or transmitted by the display cell.

The electrodes can be made in any pattern desired. These may include single digits or symbols, or the standard patterns of bargraph, seven-segment, dot matrix, starburst and so on. Alternatively, they may be extended to complex graphical displays, with addressable pixels.

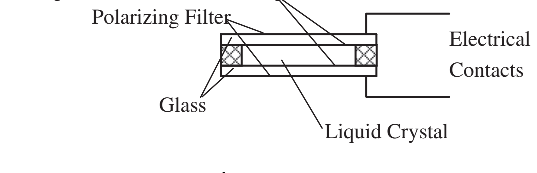


Figure 8.1:
A simple liquid crystal structure

8.1.2 Liquid Crystal Character Displays

A popular, indeed ubiquitous, form of LCD is the character display, as seen in Figure 8.2. 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 and DVD players. Driving this complex array of tiny LCD dots is far from simple, so such displays always contain a hidden microcontroller, customized 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. It is important to know its main features, in order to design with it.

The HD44780 contains an 80-byte random access memory (RAM) to hold the display data, and a read only memory (ROM) for generating the characters. It has a simple instruction set, including instructions for initialization, cursor control (moving, blanking, blinking) and clearing the display. Communication with the controller is made via an 8-bit data bus, three control lines and an enable/strobe line (E). These are itemized in Figure 8.3a.

Data written to the controller is interpreted either as instruction or as display data (Figure 8.3b), depending on the state of the RS (Register Select) line. An important use of reading data back from the LCD is to check the controller status via the Busy flag. As some instructions take a finite time to implement (e.g. a minimum of 40 μs is required to receive one character code), it is sometimes useful to be able to read the Busy flag and wait until the LCD controller is ready to receive further data.

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

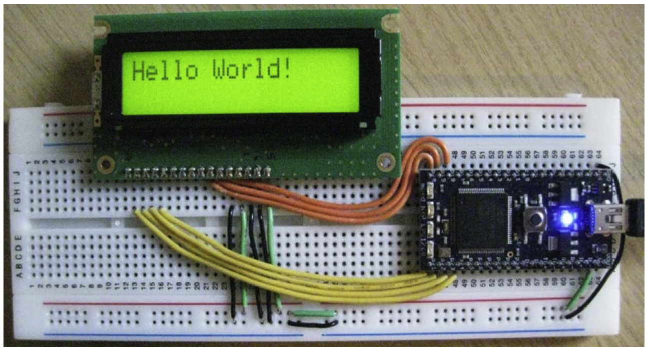
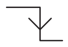
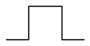
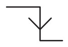
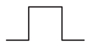


Figure 8.2:
The mbed driving an LCD

(a)

RS	Register Select:	0 = Instruction register	1 = Data Register
R/$\overline{\text{W}}$	Selects read or write		
E	Synchronizes 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		

(b)

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

(c)

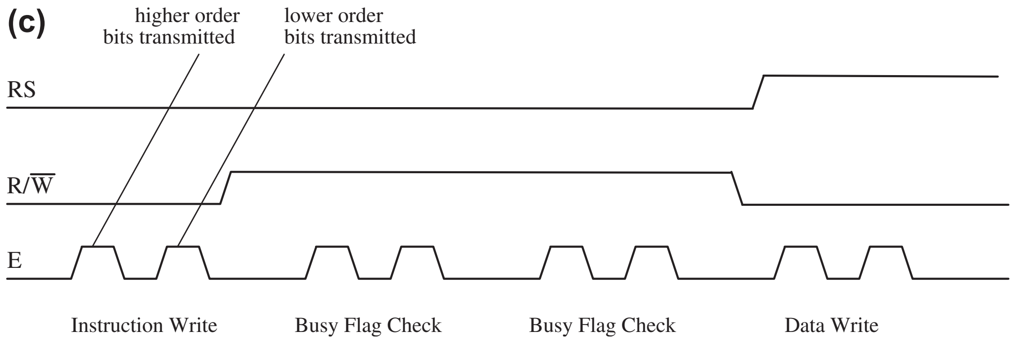


Figure 8.3:
(a) HD44780 user interface lines; (b) HD44780 data and instruction transfers; (c) HD44780 timing for 4-bit interface

8.2 Using the PC1602F LCD

The mbed processor can be interfaced 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: the LCD will need to be connected 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.
- Initializing the LCD: a specific sequence of control signals must be sent to the LCD to initialize it.
- Outputting data: we will need to understand how the LCD converts control data into legible display data.

Here, we will use the 2 × 16 character Powertip PC1602F LCD, although a number of similar LCDs can be found with the same hardware configuration and functionality.

8.2.1 Introducing the PC1602F Display

The PC1602F display is a 2 × 16 character display with an onboard data controller chip and an integrated backlight, as seen in Figure 8.2. The LCD has 16 connections, defined in Table 8.1.

In this example 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

Table 8.1: PC1602F pin descriptions

Pin number	Pin name	Function
1	V_{SS}	Power supply (GND)
2	V_{DD}	Power supply (5 V)
3	V_0	Contrast adjust
4	RS	Register select signal
5	$R/\overline{W}$	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 backlight (5 V)
16	K	Power supply for LED backlight (GND)

lines. As a result, the LCD can be controlled with only seven lines, rather than the 11 lines 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, as seen in Figure 8.3. The display is initialized 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 initialized, 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.

8.2.2 Connecting the PC1602F to the mbed

An mbed pin should be attached to each of the LCD data pins that are used, configured as digital output. 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 in Table 8.2. Note that, in simple applications, the LCD can be used only in write mode, so $R/\overline{W}$ can be tied permanently to ground (mbed pin 1). If rapid control of the LCD is required, i.e. if it needs updating faster than approximately every 1 ms, then the $R/\overline{W}$ input can be activated to enable reading from the LCD's Busy flag. The Busy flag changes state when a display request has been completed, so by testing it the LCD can be programmed to operate as quickly as possible. If the $R/\overline{W}$ line is tied to ground, then a 1 ms delay between data transfers is adequate to ensure that all internal processes can complete before the next transfer.

Table 8.2: Wiring table for connecting the PC1602F to the mbed

mbed pin number	LCD pin number	LCD pin name	Power connection
1	1	V_{SS}	0 V
39	2	V_{DD}	5 V
1	3	V_0	0 V
19	4	RS	
1	5	$R/\overline{W}$	0 V
20	6	E	
21	11	DB4	
22	12	DB5	
23	13	DB6	
24	14	DB7	
39	15	A	5 V
1	16	K	0 V

A: anode; K: cathode of the backlight LED.

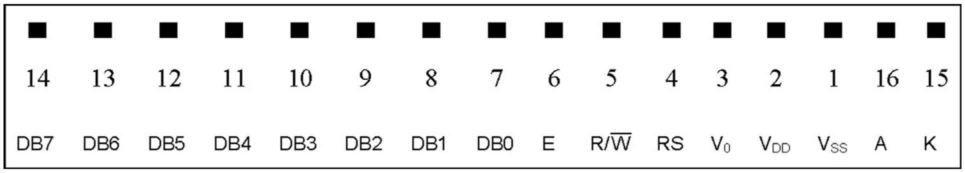

14 13 12 11 10 9 8 7 6 5 4 3 2 1 16 15
DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 E R/$\overline{W}$ RS V_0 V_{DD} V_{SS} A K

Figure 8.4:
PC1602F physical pin layout

The LCD pins DB0, DB1, DB2 and DB3 are left unconnected as these are not required when operating in 4-bit data mode. The PC1602F has the pin layout shown in Figure 8.4.

8.2.3 Using Modular Coding to Interface the LCD

Initializing 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 initializing 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 the LCD header file:

- **toggle_enable( )**: a function to toggle/pulse the enable bit
- **LCD_init( )**: a function to initialize the LCD
- **display_to_LCD( )**: a function to display characters on the LCD.

The **LCD.h** header file should therefore define the function prototypes as shown in Program Example 8.1.

```
/* 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 initialize the LCD
void display_to_LCD(char value); //function to display characters
#endif
```

Program Example 8.1 LCD header file

In the LCD definition file (**LCD.cpp**), we need to define the mbed objects required to control the LCD. Here, one digital output each can be used for RS and E, and an mbed **BusOut** object for the 4-bit data. The suggested mbed object definitions need to conform to the pin connections listed in Table 8.2.

To control and initialize the LCD, we send a number of control bytes, each sent as two 4-bit nibbles. As each nibble is asserted, the LCD requires the E bit to be pulsed. This is done by the **toggle_enable( )** function, detailed in Program Example 8.2.

8.2.4 Initializing the Display

A specific initialization procedure must be programmed in order for the PC1602F display to operate correctly. Full details are provided in the Powertip PC1602F datasheet (Reference 8.1) and also in Reference 8.2.

Reading this data, we see that 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, Display Mode and Clear Display registers in order to initialize the display. These are now introduced.

Function Mode

To set the LCD function mode, the RS, R/$\overline{\text{W}}$ and DB0-7 bits should be set as shown in Figure 8.5. Data bus values are sent as two nibbles.

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. The C/C++ code Function Mode register commands are shown in the **LCD_init( )** function detailed in Program Example 8.2.

Display Mode

The Display Mode control register must also be set up during initialization. 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 in Figure 8.6.

RS	R/$\overline{\text{W}}$
0	0

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

BW = 0 → 4 bit mode	N = 0 → 1 line mode	F = 0 → 5×7 pixels
BW = 1 → 8 bit mode	N = 1 → 2 line mode	F = 1 → 5×10 pixels

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

Figure 8.5:
Function Mode control register

RS	R/$\overline{W}$
0	0

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

P = 0 → display off	C = 0 → cursor off	B = 0 → cursor no blink
P = 1 → display on	C = 1 → cursor on	B = 1 → cursor blinking

Figure 8.6:
Display Mode control register

In order to switch the display on with a blinking cursor, the value 0x0F (in two 4-bit nibbles) is required. The C/C++ code Display Mode register commands are shown in the **LCD_init( )** function detailed in Program Example 8.2.

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 in Figure 8.7.

RS	R/$\overline{W}$
0	0

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

Figure 8.7:
Clear Display command

8.2.5 Sending Display Data to the LCD

A function called (in this example) **display_to_LCD( )** displays characters on the LCD screen. Characters are displayed by setting the RS flag to 1 (data setting), then sending a data byte describing the ASCII character to be displayed.

The term ASCII (American Standard Code for Information Interchange) has been introduced previously in Chapter 6; it is a method for defining alphanumeric characters as 8-bit values. 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 Table 8.3.

The **display_to_LCD( )** function needs to accept an 8-bit value as a data input, so that it can display the desired character on the LCD screen. In C/C++ the **char** data type can be used to define an 8-bit number. It can be seen in the **LCD.h** header file (Program Example 8.1) that

Table 8.3: Common ASCII values

		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	{	|	}	~	

display_to_LCD( ) has already been defined as a function with a **char** input. Referring to Table 8.3, it can be seen, for example, that if we send the data value 0x48 to the display, the character ‘H’ will be displayed.

The **display_to_LCD( )** function is shown in Program Example 8.2. Note that, as we are using 4-bit mode, the most significant bits of the ASCII byte must be shifted right in order to be output on the 4-bit bus created through **BusOut**. The lower 4 bits can then be output directly.

8.2.6 The Complete LCD.cpp *Definition*

The LCD definition file (**LCD.cpp**) contains the three C functions — **toggle_enable( )**, **LCD_init( )** and **display_to_LCD( )** — as described above. The complete listing of **LCD.cpp** is shown in Program Example 8.2. Note that the **toggle_enable( )** function has two 1 ms delays, which remove the need to monitor the Busy flag; the downside to this is that we have introduced a timing delay into our program, the consequences of which will be discussed in more detail in Chapter 9.

```
/* 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);
void toggle_enable(void){
  E=1;
  wait(0.001);
  E=0;
  wait(0.001);
}
//initialize 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 (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();
}
```

Program Example 8.2 Declaration of objects and functions in LCD.cpp

8.2.7 Utilizing the LCD Functions

We can now develop a main control file (**main.cpp**) to utilize the LCD functions described above. Program Example 8.3 initializes the LCD, displays the word ‘HELLO’ and then displays the numerical characters from 0 to 9.

```
/* Program Example 8.3 Utilizing LCD functions in the main.cpp file
*/
#include "LCD.h"
int main() {
  LCD_init();                  // call the initialize 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
  }
}
```

Program Example 8.3 File main.cpp utilizing LCD functions

Exercise 8.1

Applying Table 8.2, connect a Powertip PC1602F LCD to an mbed and construct a new program with the files **main.cpp**, **LCD.cpp** and **LCD.h** as described in the program examples above. Compile and run the program.

1. Verify that the word 'HELLO' and the numerical characters 0–9 are correctly displayed with a flashing cursor.
2. Change the program so that your name appears on the display, after the word 'HELLO', instead of the numerical characters 0–9.

Exercise 8.2

If you look at the **toggle_enable( )** function in Program Example 8.2, you will notice that it sets the E line high and low, waiting 1 ms in each case. This saves us testing the Busy flag, as seen in Figure 8.3c, as the display will have exited from its Busy state during the 1 ms delay. For many embedded applications, however, this loss of 2 ms would be quite unacceptable.

Try applying the information so far supplied to write a revised program, so that the delays are removed, and the Busy flag is tested. You will now need to activate $R/\overline{W}$ and set it to 1 in order to read the Busy flag, which is indicated by data bit DB7. Once Busy is clear, the program should proceed. Estimate how much time is saved by your new program. You can test this with an oscilloscope by making your program write a digit continuously to the display, and measuring on the oscilloscope the time between the E pulses.

8.2.8 Adding Data to a Specified Location

The display is mapped out with memory address locations so that each display unit has a unique address. The display pointer can therefore be set before outputting data, and the data will then appear in the position specified. The display address layout is shown in Figure 8.8.

If the program counter is set to address 0x40, data will be displayed at the first position on the second line. To change the pointer address, the desired 6-bit address value must be sent in a control byte with bit 7 also set, as shown in Figure 8.9.

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

Figure 8.8:
Screen display pointer address values

A new function can be created to set the location of the display pointer prior to writing. We will call this function **set_location( )**, as shown in Program Example 8.4.

```
/* 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();
}
```

Program Example 8.4 Function to change the display pointer position

Notice that bit DB7 is set by ORing the location value with 0x80.

Exercise 8.3

Add the **set_location( )** function shown in Program Example 8.4 to the LCD.cpp definition. You will also need to declare the function prototype in LCD.h. Now add **set_location( )** function calls to **main.cpp** so that the word 'HELLO' appears in the center of the first line and the numerical characters 0–9 appear in the center of the second line of the display.

Exercise 8.4

Modify the **LCD_init( )** function to disable the flashing cursor. Here, you will need to modify the value sent to the Display Mode register.

RS	R/$\overline{W}$
0	0

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

AC0-AC6 describe the 6-bit display pointer address

Figure 8.9:
Display pointer control

8.3 Using the mbed TextLCD Library

There is an mbed library which makes the 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);
```

C code feature

Simple **printf( )** statements are used to display characters on the LCD screen; this statement is also discussed in Section B.9 of Appendix B. The **printf( )** function allows formatted print statements to be used. This means that text strings and formatted data can be sent to a display, meaning that a function call is not required for each individual character (as was the case in Program Example 8.3).

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 as in the examples above (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 to be imported to the mbed program. The mbed **TextLCD** library can (at the time of writing) 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 the **main.cpp** file or relevant project header files. Program Example 8.5 is a simple Hello World example using the **TextLCD** library.

```
/*Program Example 8.5: TextLCD library 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!");
}
```

Program Example 8.5 TextLCD Hello World

The cursor can be moved to a chosen position to allow you to choose where to display data. This uses the **locate( )** function as follows. The display is laid out as two rows (0−1) of 16 columns (0–15). The locate function defines the column first followed by the row. For example, add the following statement to move the cursor to the second line and fourth column:

```
lcd.locate(3,1);
```

Any **printf( )** statements after this will be printed at the new cursor location.

Exercise 8.5

1. Create a new program and import the **TextLCD** library file (right click on the project and select 'import library').
2. Add Program Example 8.5 to the main.cpp file and compile and run. Verify that the program correctly displays the Hello World characters.
3. Experiment further with the locate function and verify that the Hello World string can be positioned to a desired location on the display.

The screen can also be cleared with the following command:

```
lcd.cls();
```

Program Example 8.6 displays a count variable on the LCD. 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++;
  }
}
```

Program Example 8.6 LCD counter

Exercise 8.6

1. Implement Program Example 8.6 as a new program. Do not forget to import the TextLCD library!
2. Increase the speed of the counter and investigate how the cursor position changes as the count value increases.

8.4 Displaying Analog Input Data on the LCD

An analog input can be defined – before the **main( )** function – on pin 18 as follows:

```
AnalogIn Ain(p18);
```

We can now connect a potentiometer between 3.3 V (mbed pin 40) and 0 V (mbed pin 1) with the wiper connected to pin 18. The analog input variable has a floating point value between 0 and 1, where 0 is 0 V and 1 represents 3.3 V. We will multiply the analog input value by 100 to display a percentage between 0 and 100%, as shown in Program Example 8.7. 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();
  }
}
```

Program Example 8.7 Display analog input data

Exercise 8.7

1. Implement Program Example 8.7 and verify that the potentiometer modifies readings between 0 and 100%.
2. Modify the **wait( )** statement to be a larger value and evaluate the change in performance. It is interesting to make a mental note that a certain range of update rates appears irritating to view (too fast), while others may be perceived as too slow.

Exercise 8.8

Create a program to make the mbed and display act like a standard voltmeter, as shown in Figure 8.10. Potential difference should be measured between 0 and 3.3 V and displayed to the screen. Note the following:

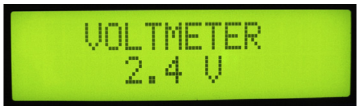


Figure 8.10:
Voltmeter display

- You will need to convert the 0.0–1.0 analog input value to a value that represents 0 –3.3 V.
- An infinite loop is required to allow the voltage value to update continuously 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. Evaluate the noise and accuracy of the voltmeter readings with respect to the mbed's ADC resolution.

■

8.5 More Advanced LCDs

8.5.1 Color LCDs

Nowadays, LCD technology is used in advanced displays for mobile phones, PC monitors and televisions. 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 therefore usually referred to by a 24-bit value, where the highest 8 bits define the red shade, the middle 8 bits the green shade and the lower 8 bits the blue shade, as shown in Table 8.4.

Given that each pixel needs to be assigned a 24-bit value and a 1280 × 1024 LCD computer display has over 1 million pixels (1280 * 1024 = 1 310 720), a lot of data needs to be sent to a color LCD. Standard color LCDs are set to refresh the display at a frequency of 60 Hz, so the digital input requirements are much greater than those associated with the alphanumeric LCD discussed previously in this chapter.

8.5.2 Controlling a SPI LCD Mobile Phone Display

A mobile phone screen, like the Nokia 6610, represents state-of-the-art, low-cost, high-volume display technology. Yet many have an interface based on the good old serial peripheral interface (SPI) standard, introduced in Chapter 7. It is hard to find formal published

Table 8.4: 24-bit color values

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

data on the Nokia display; Reference 8.3, however, gives some useful background. We can use the Nokia 6610 display with the mbed. Here, we will see how to connect it and how to type data on the screen from the PC keyboard. Instead of using the basic SPI library, we will use the predesigned mbed MobileLCD library. You will need to import this library from:

http://mbed.co.uk/projects/cookbook/svn/MobileLCD/tests/MobileLCD

The mobile screen has 10 pins, as shown in Figure 8.11. It can be connected to the mbed using the connections shown in Table 8.5.

It is possible to clear the screen, set the background color, fill blocks, draw pixel images, move the pointer and perform many other operations. Program Example 8.8 writes characters to the display from the computer keyboard, and uses the # key to clear the screen.

```
/*Program Example 8.8: Program which reads character from computer screen, and displays
on Nokia LCD display.
*/
#include "mbed.h"
#include "MobileLCD.h"                    // include the Nokia display library
MobileLCD lcd(p11, p12, p13, p15, p16); // mosi,miso,clk,cs,rst

Serial pc(USBTX, USBRX);                // host terminal comms setup
char c;                                 // char variable for keyboard input
void screen_setup(void);                // function prototype

int main() {
  pc.printf("\n\rType something to be displayed:\n\r");
  screen_setup();                       // call the screen setup function
  while(1){
    c = pc.getc();                      // c = character input from computer keyboard
    wait(0.001);
    if (c=='#'){                        // perform the following if "#" is pressed
      screen_setup();                   // call the screen setup function
      lcd.locate(0,0);                    // move the cursor back to row 0 column 0
    }
    else{
      lcd.printf("%c",c);                 // print character on the LCD screen
      pc.printf("%c",c);                  // print character on the terminal screen
    }
   }
 }
//function definition for screen_setup
void screen_setup(void) {
  lcd.background(0x0000FF);               // set the background color
  lcd.cls();                              // clear the screen
}
```

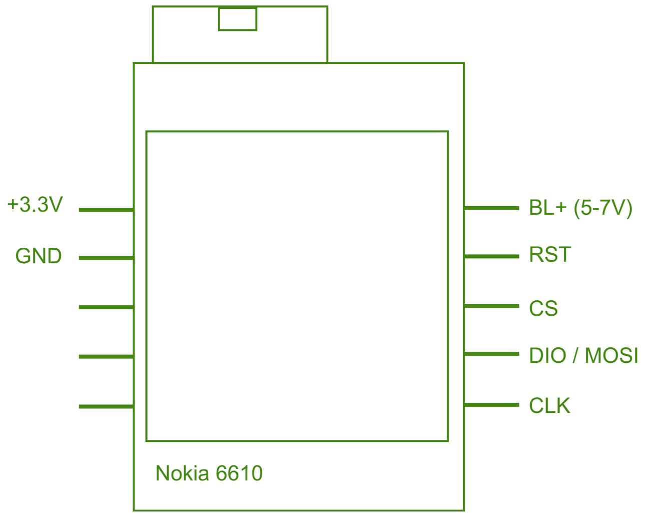


Figure 8.11:
Nokia 6610 display connections

Table 8.5: Connections between Nokia display and mbed

Nokia 6610 pin	mbed pin
+3.3 V	40
GND	1
BL+	39
RST	9
CS	8
MOSI	5
CLK	7

Program Example 8.8 Writing characters to the Nokia display

Having connected the circuit, create a new project, and enter and compile the example code. Once it is running on the mbed, and with the terminal application open, you should be able to type to the LCD screen.

Exercise 8.9

Add the following fill functions to the **screen_setup( )** function (after the lcd.cls command) in order to fill some areas of the LCD:

```
lcd.fill(2, 51, 128, 10, 0x00FF00);
lcd.fill(50, 1, 10, 128, 0xFF0000);
```

It is also possible to draw on the screen pixel by pixel. The following loop will create the function for a sine wave and print this to the screen. Add this to the setup function.

```
for(int i=0; i<130; i++) {
  lcd.pixel(i, 80 + sin((float)i / 5.0)*10, 0x000000);
}
```

We have seen that LCDs allow greater flexibility than simple LED-based displays, and that simple alphanumeric and advanced color LCD devices can be utilized with microcontrollers and embedded systems.

8.6 Mini-Project: Digital Spirit Level

Design, build and test a digital spirit level based on the mbed. Use an ADXL345 (Section 7.3) 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 color LCD to output the measured orientation data, in degrees.

To help you proceed, consider the following:

1. Design your display to show a pixel or an image moving around the LCD screen with respect to the orientation of the accelerometer.
2. Add the digital switch to allow simple calibration and zeroing of the data.
3. Improve your display output to give accurate measurements of the two-plane orientation angles in degrees from the horizontal (i.e. horizontal = 0°).

Chapter Review

- Liquid crystal displays (LCDs) use an organic crystal which can polarize 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 color 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 initialize 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.
- Color LCDs use a serial interface to display data, with each pixel given a 24-bit color setting.
- The mbed **MobileLCD** library can be used to interface a Nokia 6610 mobile phone-type LCD.

Quiz

1. What are the advantages and disadvantages associated with using an alphanumeric LCD 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 ASCII stand for?
6. What are the ASCII values associated with the numerical characters from 0 to 9?
7. What is the correct **printf( )** C/C++ code required to display the value of a floating point variable called 'ratio' to two decimal places ?
8. List five practical examples of a color LCD used in an embedded system.
9. 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?

References

8.1. Rapid Electronics. PC1602F datasheet. http://www.rapidonline.com/pdf/57-0913.pdf
8.2. HD44780 LCDisplays. http://www.a-netz.de/lcd.en.php
8.3. Nokia 6100 LCD Display Driver. http://www.sparkfun.com/tutorial/Nokia%206100%20LCD%20Display%20Driver.pdf

This page intentionally left blank