

# Programming the *Dragon12* LCD module

The liquid crystal display (LCD) module on the *Dragon12* is a 16 character  $\times$  2 line alphanumeric display. The module is connected to one of the MC9S12DP256B MCU ports and the interface handshake is performed under software control.

Unlike the LED display, the LCD module requires no special handling to refresh the display. The module's internal electronics drive the transparent electrodes on the inside surfaces of the glass display. Characters written to the module's character memory will remain displayed until they are overwritten, the module is reset, or the power is turned off.

## 1 Theory

### 1.1 Liquid crystal display technology

Liquid crystal displays are an economical way to display short messages to users of embedded systems. They consume very little power. Recent advances in LCD technology have improved the display contrast and visual quality. The module integrates the glass display that holds the liquid crystal fluid as well as all the electronics to mediate between the bus interface and the display.

LCDs are the display of choice for battery powered devices because of their low current consumption.

### 1.2 Module interface

The module was designed to communicate with a CPU using a traditional CPU bus transfer cycle. In order to simplify external hardware requirements, the module data word can be narrowed down to 4 bits, transferring each byte in two nibbles (high followed by low). Since the MCU is in single-chip mode, it has no external bus interface therefore the bus cycle timing must be "faked" with software sequencing the signals on the parallel port.

### 1.2.1 Data and control signals

The module has 4 data lines  $DB[7 : 4]$  connected to the MCU port pins. There is one address line  $RS$  to select one of two internal registers, command and data. The  $EN$  input is pulsed to complete a write operation to the module.

The module's  $R/\overline{W}$  input is grounded, placing the module in write-only mode forever — the MCU can never read back data written to the display memory. Instead, the software must keep track of the characters written to the display buffer in dedicated variables.

### 1.2.2 Parallel port assignment

The LCD module is connected to port K as shown in table 1. Note that only 4 wires of the module's data bus are actually used. When the module is in 4-bit mode, only the upper 4 bits are active. The module powers up in 8-bit mode but the during initialisation sequence, the commands sent in 8-bit mode only use the upper 4 bits. The lower 4 bits are ignored.

Port K	Signal	LCD pin	Pin function
7	—	—	Not used
6	—	—	Not used
5	DB7	14	Data bus
4	DB6	13	Data bus
3	DB5	12	Data bus
2	DB4	11	Data bus
1	EN	6	Enable control signal
0	RS	4	Register select: 1=data, 0=instruction

Table 1: Port K pin assignments

The software shifts the data bits to line up with the port pin assignment. Bits representing the control signals  $EN$  and  $RS$  are merged with the data bits then written to the port data register.

## 2 Programming

### 2.1 Initialisation sequence

The module initialisation may seem a little obscure and irrational but it is dictated by a control interface developed in the 1970's and that has since become a de facto industry standard. The sequence must be followed exactly as specified in the documentation otherwise the module will not function correctly.

1. command byte=0x30  
Function set, 8-bit mode, db[7:4]=0011, RS=0, module treats lower 4 bits as "don't care"  
Write upper nibble only, then wait 4.1ms
2. command byte=0x30  
Function set, 8-bit mode, db[7:4]=0011, RS=0, module treats lower 4 bits as "don't care"  
Write upper nibble only, then wait 100us
3. command byte=0x30  
Function set, 8-bit mode, db[7:4]=0011, RS=0 module treats lower 4 bits as "don't care"  
Write upper nibble only, no wait
4. command byte=0x20  
Function set, 4-bit mode, db[7:4]=0010, RS=0 module treats lower 4 bits as "don't care" Write upper nibble only, no wait.  
*At this point the module bus is in 4-bit mode, so software must write both nibbles for the full 8 bits.*
5. command byte=0x28  
Function set: db[7:5]=001, 4-bit mode: db4=0, 2 lines: db3=1 (N=1), 5x7 chars: db2=0 (F=0), other bits: db0-1=00, RS=0  
Write upper nibble, then lower nibble, no wait
6. command byte=0x0c  
Display on/Off: db[7:3]=00001 (set the display on), display on: db2=1, cursor off: db1=0, blink off: db0=0, RS=0  
Write upper nibble, then lower nibble, no wait

7. command byte=0x01  
Display clear: db[7:0]=00000001, RS=0  
Write upper nibble, then lower nibble, no wait
8. command byte=0x06  
Entry mode set: db[7:2]=000001, I/D=1 (cursor increment mode), S=0  
(disable display auto shift), RS=0  
Write upper nibble, then lower nibble, no wait

## 2.2 Writing characters

The LCD module controller maintains an internal counter that points to the character location to be written. The initialisation sequence has set that counter to auto-increment after every character written. The strategy is to set the pointer to the first character in the line, and then write all the character in the line.

The characters are assembled in an array. Any location that is not supposed to show anything is written with a blank (ASCII 0x20).

Writing the characters is done in two steps:

1. Write the pointer:  
Command byte = 0x80 ORed with the address of the target location  
0x00 (line 1) or 0x40 (line 2);  
RS=0 (command)
2. Write 16 characters (ASCII) in succession, leftmost character first;  
RS=1 (data)

## 3 Application program interface

The complexity of displaying characters on the display can be encapsulated in a pair of functions (subroutines) that are easy to use. The interface presented by these functions is not very versatile, but it is simple.

A test program is provided in the file `lcdinit.c`.

### 3.1 High level functions

`void initLcd(void)`

Initialises the LCD, blanks the display and turns the cursor off. It accepts no parameters and returns `void`.

`void writeLine(char *string, int line)`

Write 16 characters to the display. `line=0` is the top line, `line=1` is the bottom line. If the string is shorter than 16 characters, it must be padded with blanks (ASCII 0x20) otherwise junk characters will be displayed. If the string is longer than 16 characters, the extra characters are not displayed.

### 3.2 Low level functions

Individual command and data bytes may be written using the low level functions that load the parallel port bits and pulse the EN control signal. Two functions are provided, to handle 8-bit and 4-bit bus width. The 8-bit mode function can only be used during the initialisation sequence.

`void writeLcd8(unsigned char data, unsigned char rs)`

Write the byte `data`, setting the RS control line according to the value of the least significant bit of `rs`. The lower 4 bits of `data` are not written because they are not connected to the module.

`void writeLcd4(unsigned char data, unsigned char rs)`

Write the byte `data` in 4 bit mode, high nibble then low nibble, setting the RS control line according to the value of the least significant bit of `rs`.

## 4 References

[han1] Hantronix, HDM16216L-5 data sheet, file `Document\Schematic\LCD16X2.PDF` in *Dragon12* documentation package.

[han2] Hantronix, character modules data sheet, file `Document\Schematic\LCD_SPEC.PDF` in *Dragon12* documentation package.

[chu04] Wayne Chu, Private correspondence, EVBPLUS/Wytec, March 2004