



(μ LCD-128 & μ LCD-128-xMb)

USERS MANUAL

Revision 1.3



4D Systems



MicroLCD

PROPRIETARY INFORMATION

The information contained in this document is the property of 4D Systems Pty. Ltd., and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission. It should not be used for commercial purposes without prior agreement in writing.

4D Systems Pty. Ltd. Endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

Contact details are available from the company web site at www.4dsystems.com.au

All trademarks recognised and acknowledged.

Copyright 4D Systems Pty. Ltd. 2000-2006



Table of contents

1. μ LCD Description

- 1.1 Introduction
- 1.2 μ LCD features

2. μ LCD Command set

- 2.1 μ LCD-128 and μ LCD-128-xMb Command set
 - 2.1.1 Erase Screen
 - 2.1.2 Background Colour
 - 2.1.3 Put Pixel
 - 2.1.4 Read Pixel
 - 2.1.5 Draw Circle
 - 2.1.6 Draw Line
 - 2.1.7 Font Size
 - 2.1.8 Opaque or Transparent Text
 - 2.1.9 Place Text Character (formatted)
 - 2.1.10 LCD Display Control Functions
 - 2.1.11 Add User Bitmapped Character
 - 2.1.12 Display User Bitmapped Character
 - 2.1.13 Paint Area
- 2.2 μ LCD-128-xMb Additional Command set
 - 2.2.1 Draw Circle with Fill
 - 2.2.2 Place Text Character (unformatted)
 - 2.2.3 Place String of ASCII Text (formatted)
 - 2.2.4 Display Image
 - 2.2.5 Set Transparent Colour (for Images only)
 - 2.2.6 Enable/Disable Transparent colour
 - 2.2.7 Place Button Icon
 - 2.2.8 Block Copy & Paste (Screen Bitmap Copy)
 - 2.2.9 Display Object from Flash Memory
 - 2.2.10 Delay
 - 2.2.11 Loop Start
 - 2.2.12 Loop End
 - 2.2.13 Run
 - 2.2.14 Exit
 - 2.2.15 Restart
 - 2.2.16 Download Data to Flash Memory
 - 2.2.17 Read Data from Flash Memory
 - 2.2.18 Erase Flash Memory
- 2.3 μ LCD Serial Interface
- 2.4 μ LCD USB Interface



3. Specifications

- 3.1 μ LCD pin-outs
- 3.2 65,536 Colour Bitmap Organisation
- 3.3 Power-Up Reset

4. Appendix

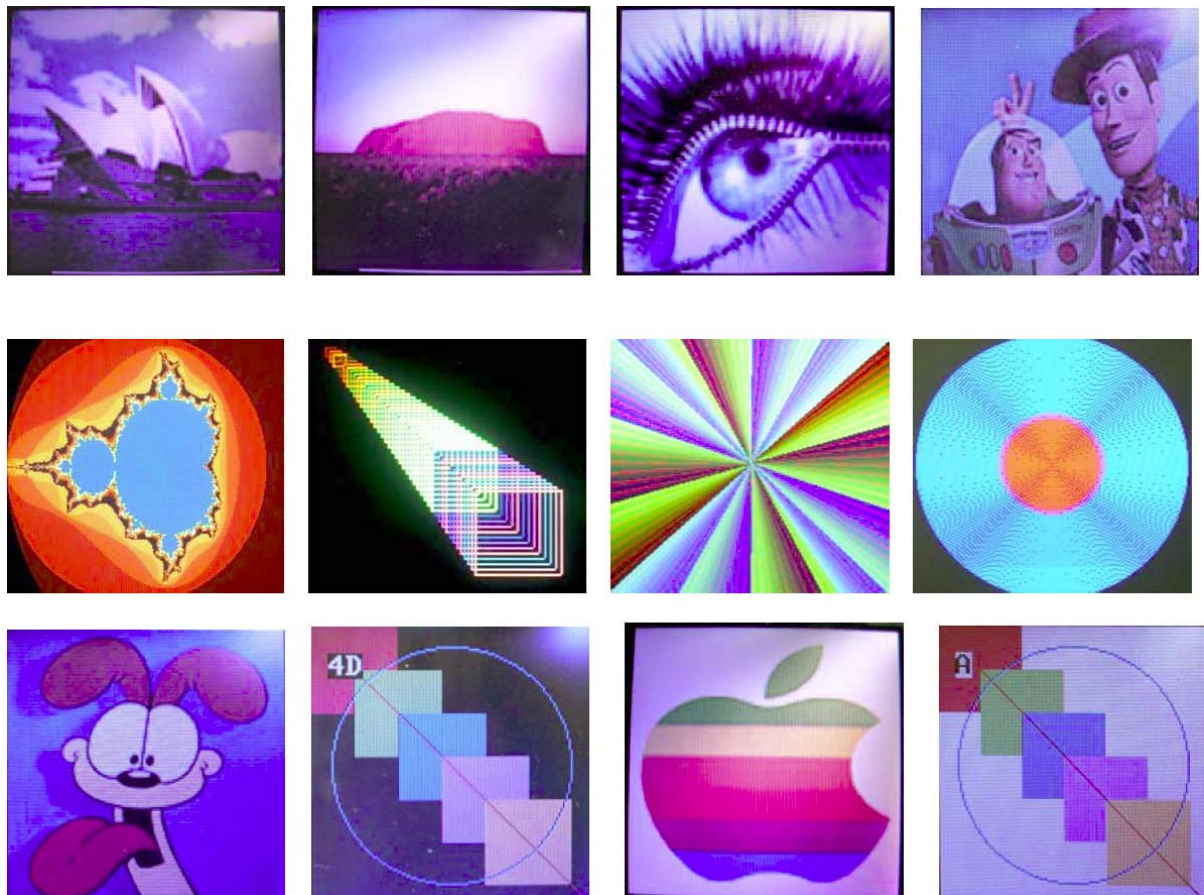
- 4.1 Available models
- 4.2 Related Products
- 4.3 Auto Demo/Slide Show
- 4.4 Precautions
- 4.5 Help and Other Information

1 μ LCD Description

1.1 Introduction

The μ LCD is a compact & cost effective all in one 'SMART' LCD Display with an embedded graphics controller that will deliver 'stand-alone' functionality to your project. The 'simple to use' embedded commands not only control background colour but can produce text in a variety of sizes as well as draw shapes (which can include user definable bitmapped characters such as logos) in 65,536 colours whilst freeing up the host processor from the 'processor hungry' screen control functions. This means a simple micro-controller with a standard serial or USB interface can drive the μ LCD module with total ease.

Figures below show some of the graphics capability of the μ LCD.



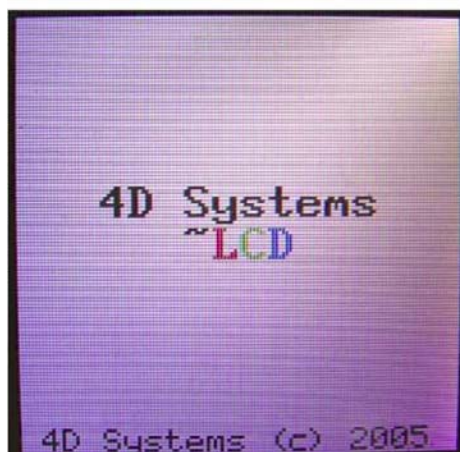


MicroLCD

1.2 μ LCD Features

The μ LCD is aimed at being integrated into a variety of different applications via a wealth of features designed to facilitate any given functionality quickly and cost effectively and thus reduce 'time to market'. These features are as follows:

- 65,536 colours.
- Three selectable font sizes (5x7, 8x8 and 8x12).
- User defined bitmapped characters, 64 by 8x8.
- Vector drawing controls for circles, lines, squares.
- Standard 'built in' ASCII character set .
- Logic level RS232 serial interface to μ LCD from host.
- Auto baud rate detection from 300 baud to 128Kbit/sec.
- USB Interface available (micro-USB).





2 **μLCD** Command Set

The heart of the **μLCD** is the easy to understand command set. This comprises of a handful of easy to learn instructions that can draw lines, circles, squares, etc, to provide a full text and graphical user interface. The commands are sent to the **μLCD** via its serial connection (4 pin header). **Please note that the Rx and the Tx signals are at 3.3V levels. If interfacing to a host system running at 5V levels, then 1K series resistors must be inserted between the Host Tx/Rx and the **μLCD** Rx/Tx signals...**

2.1 **μLCD**-128 and **μLCD**-128-xMb Command Set

	Command Length	Section	Page
(E) Erase Screen	1 byte	2.1.1	8
(B) Background Colour	3 bytes	2.1.2	9
(P) Put P ixel	5 bytes	2.1.3	10
(R) Read P ixel	3 bytes	2.1.4	11
(C) Draw C ircle	6 bytes	2.1.5	12
(L) Draw L ine	7 bytes	2.1.6	13
(F) Font Size	2 bytes	2.1.7	14
(O) Opaque or Transparent Text	2 bytes	2.1.8	15
(T) Place T ext Character (formatted)	6 bytes	2.1.9	16
(Y) LCD Display Control functions	3 bytes	2.1.10	17
(A) Add User Bitmapped Character	10 bytes	2.1.11	18
(D) Display User Bitmapped Character	6 bytes	2.1.12	19
(p) paint Area	7 bytes	2.1.13	20

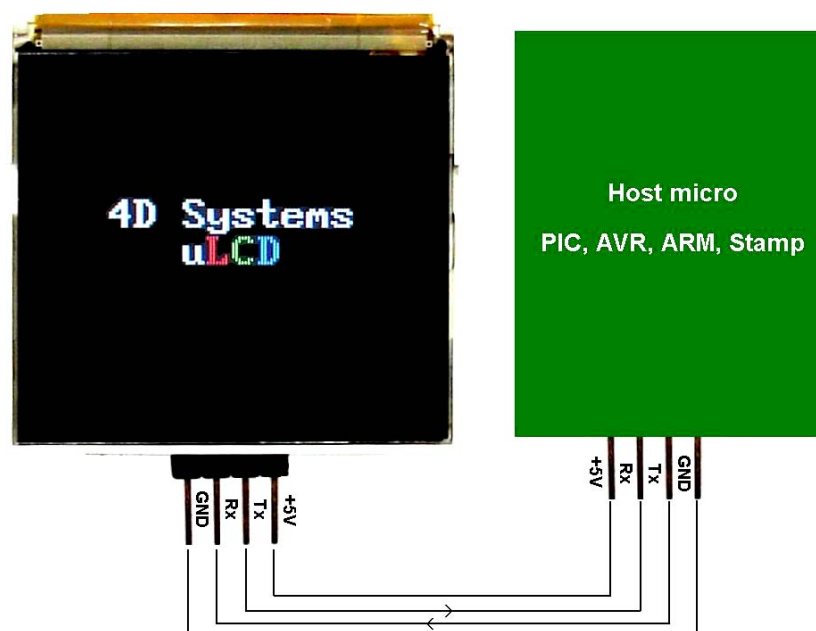
COMMAND PROTOCOL

The following are each of the commands with the correct syntax. Please note that all command examples listed below are in hex (**00hex**). Due to the high colour depth of the μ LCD, a pixel colour value will not fit into a single byte, a byte can only hold a maximum value of 255. Therefore the colour is represented as a 2 byte value, **colour(msb)**, **colour(lsb)**. The most significant byte (msb) is transmitted first followed by the least significant byte (lsb). This format is called the big endian. So for a 2 byte colour value of **013Fhex** the byte order can be shown as (**01hex**),(**3Fhex**).

NOTE: When transmitting the command and data bytes to the μ LCD, do not include any separators such as commas ',' or spaces ' ' or brackets '(' ')' between the bytes. The examples show these separators purely for legibility; these must not be included when transmitting data to the μ LCD.

When a μ LCD command is sent, the μ LCD will reply back with a single acknowledge byte called the **ACK (06hex)**. This tells the host that the command was understood and the operation is completed. It will take the μ LCD anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the μ LCD has to perform. If the μ LCD receives a command that it does not understand it will reply back with a negative acknowledge called the **NAK (15hex)**.

If a command that has 5 bytes but only 4 bytes are sent, the command will not be executed and the μ LCD will wait until another byte is sent before trying to execute the command. There is no timeout on the μ LCD when incomplete commands are sent. The μ LCD will reply back with a **NAK** for each invalid command it receives. For correct operation make sure the command bytes are sent in the correct sequence.





MicroLCD

2.1.1 Erase Screen (E)

Syntax : cmd

cmd : 45hex, Eascii

Description : This command clears the entire screen using the current background colour.

Example : 45hex
Clear the screen.



2.1.2 Background Colour (B)

Syntax : cmd, colour(msb), colour(lsb)

cmd : 42hex, Bascii

colour : pixel colour value: 2 bytes (16 bits) msb, lsb

65,536 colours to choose from

Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command sets the current background colour. Once this command is sent, only the background colour will change. Any other object on the screen with a different colour value will not be affected.

Example : 42hex, FFFFhex

Set the background colour to value 65,535 (white).



MicroLCD

2.1.3 Put Pixel (P)

Syntax : cmd, x, y, colour(msb), colour(lsb)

cmd : 50hex, Pascii

x : horizontal pixel position. 0dec to 127dec (00hex to 7Fhex).

y : vertical pixel position. 0dec to 127dec (00hex to 7Fhex).

colour : pixel colour value: 2 bytes (16 bits) msb, lsb

65,536 colours to choose from

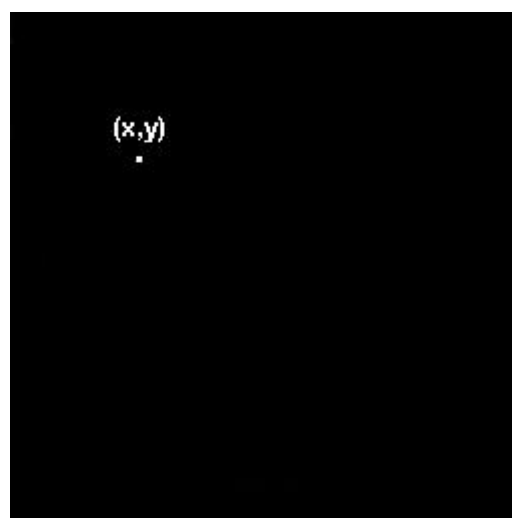
Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command will put a coloured pixel at location (x, y) on the screen.

Example : 50hex, 01hex, 0Ahex, 00hex, 00hex

Puts a black (0000hex) pixel at location x = 01dec (01hex) and y = 10dec (0Ahex).





2.1.4 Read Pixel (R)

Syntax : cmd, x, y

cmd : 52hex, Rascii

x : horizontal pixel position. 0dec to 127dec (00hex to 7Fhex).

y : vertical pixel position. 0dec to 127dec (00hex to 7Fhex).

Description : This command will read the colour value of pixel at location (x, y) on the screen and return it to the host. This is a useful command when for example a white pointer is moved across the screen and the host can read the colour on the screen and switch the colour of the pointer when it's on top of a light coloured area.

Example : 52hex, 01hex, 01hex

µLCD reply : 00hex, 1Fhex

Reads a blue (001Fhex) pixel at location x = 1dec (01hex) and y = 1dec (01hex).

2.1.5 Draw Circle (C)

Syntax : cmd, x, y, rad, colour(msb), colour(lsb)

cmd : 43hex, Cascii

x : horizontal circle centre position. 0dec to 127dec (00hex to 7Fhex).

y : vertical circle centre position. 0dec to 127dec (00hex to 7Fhex).

rad : radius size of the circle. 0dec to 127dec (00hex to 7Fhex).

colour : circle colour value: 2 bytes (16 bits) msb, lsb

65,536 colours to choose from

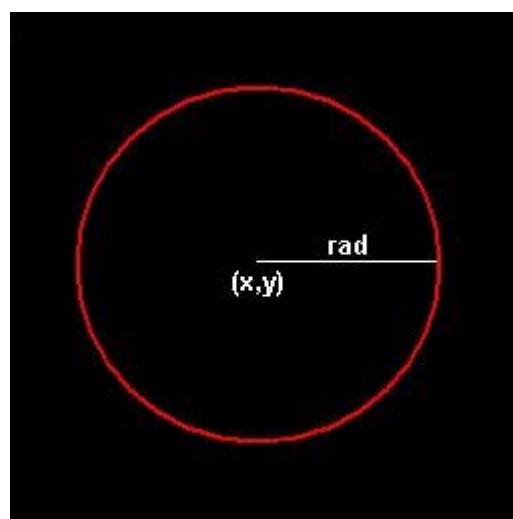
Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command will draw a coloured circle centred at (x, y) with a radius determined by the value of rad.

Example : 43hex, 3Fhex, 3Fhex, 22hex, FFhex, FFhex

Draws a white circle (FFFFhex) centred at x = 63dec (3Fhex) and y = 63dec (3Fhex) with a radius of 34dec (22hex).



2.1.6 Draw Line (L)

Syntax : `cmd, x1, y1, x2, y2, colour(msb), colour(lsb)`

cmd : 4Chex, Lascii

x1 : horizontal position of line start. 0dec to 127dec (00hex to 7Fhex).

y1 : vertical position of line start. 0dec to 127dec (00hex to 7Fhex).

x2 : horizontal position of line end. 0dec to 127dec (00hex to 7Fhex).

y2 : vertical position of line end. 0dec to 127dec (00hex to 7Fhex).

colour : line colour value: 2 bytes (16 bits) msb, lsb
65,536 colours to choose from

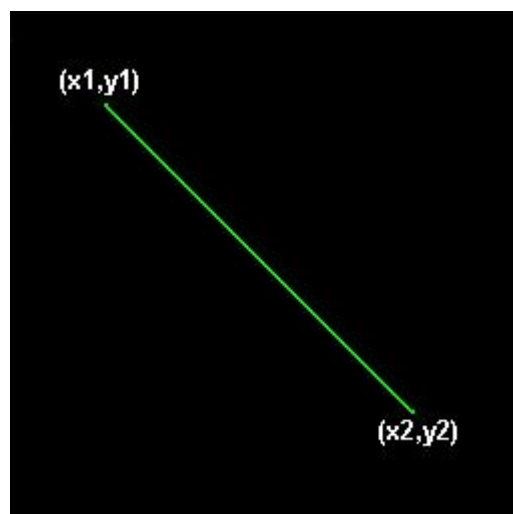
Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command will draw a coloured line from point (x1, y1) to point (x2, y2) on the screen.

Example : 4Chex, 00hex, 00hex, 7Fhex, 7Fhex, FFhex, FFhex

Draws a white line from (x1=0, y1=0) to (x2=127, y2=127).





2.1.7 Font Size (F)

Syntax : cmd, size

cmd : 46hex, Fascii

size : = 00hex : 5x7 small size font
= 01hex : 8x8 medium size font
= 02hex : 8x12 large size font

Description : This command will change the size of the font according to the value set by **size**. Changes take place after the command is sent. Any character on the screen with the old font size will remain as it was.

Example1: 46hex, 00hex Select small 5x7 fonts
Example1: 46hex, 01hex Select medium 8x8 fonts
Example1: 46hex, 02hex Select large 8x12 fonts



2.1.8 Opaque / Transparent Text (O)

Syntax : cmd, mode

cmd : 4Fhex, Oascii

mode : = 00hex : Transparent Text, objects behind the text can be seen.
= 01hex: Opaque Text, objects behind text is blocked by background

Description : This command will change the attribute of the text so that an object behind the text can either be blocked or transparent. Changes take place after the command is sent.

This command will change the attribute so that when a character is written, it will either write just the character alone (Transparent Mode) so any original character will be seen as well as the new, or overwrite any existing data with the new character.

Example1: 4Fhex, 00hex Transparent Mode

Example2: 4Fhex, 01hex Opaque Text



2.1.9 Place Text Character (formatted) (T)

Syntax : cmd, char, column, row, colour(msb), colour(lsb)

cmd : 54hex, Tascii

char : inbuilt standard ASCII character, 32dec to 127dec (20hex to 7Fhex)

column : horizontal position of character, see range below:
0 - 20 for 5x7 font, 0 - 15 for 8x8 and 8x12 font.

row : vertical position of character:
0 - 15 for 5x7 and 8x8 font, 0 - 9 for 8x12 font.

colour : character colour value: 2 bytes (16 bits) msb, lsb
65,536 colours to choose from
Black = 0000hex, 0dec
White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command will place a coloured ASCII character (from the ASCII chart) on the screen at a location specified by (**column, row**). The position of the character on the screen is determined by the predefined horizontal and vertical positions available, namely 0 to 20 columns by 0 to 15 rows.

Example : 54hex, 41hex, 00hex, 00hex, FFhex, FFhex

Place character 'A' (41hex) at column = 0, row = 0, colour = white (65,535).



2.1.10 LCD Display Control Functions (Y)

Syntax : cmd, mode, value

cmd : 59hex, Yascii

mode : = 00hex : **BACKLIGHT CONTROL.**

value = 0dec to 15dec : 0dec = Backlight off, 15dec = Brightest

for the previous versions (MkI & MkII)

value = 00hex: Backlight OFF
= 01hex: Medium Bright
= 02hex: Full Bright

mode : = 01hex : **DISPLAY ON/OFF.**

value = 00hex: Display OFF
= 01hex: Display ON

mode : = 02hex : **LCD CONTRAST.**

value = 0dec to 40dec : Contrast range (default = 23dec)

mode : = 03hex : **LCD POWER-UP/POWER-DOWN.**

value = 00hex: LCD Power-Down
= 01hex: LCD Power-Up

Note: It is important that the LCD be issued with the Power-Down command before switching off the power. This command switches off the internal voltage boosters and current amplifiers and they need to be turned off before main power is removed. If the power is removed without issuing this command, the LCD maybe damaged (over a period of time). This command also turns off the backlight. This command need not only be issued to shutdown but can be issued to conserve power by turning off the display and the backlight. The Power-Up command does not need to be executed when applying power. If a Power-Down command has been issued and Power is not switched off, the Power-Up command can be sent to Power the display back up again.

2.1.11 Add User Bitmapped Character (A)

Syntax : cmd, char#, data1, data2,, dataN

cmd : 41hex, Aascii

char# : bitmap character number to add to memory:
0 to 63 (00h to 3Fh), 64 characters of 8x8 format.

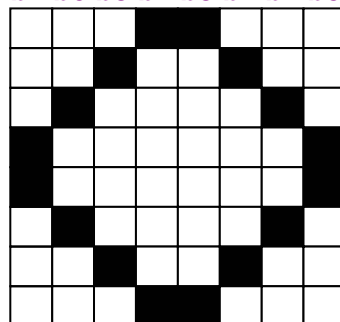
data1 to dataN : number of data bytes that make up the composition and format of the bitmapped character. The 8x8 bitmap composition is 1 byte wide (8bits) by 8 bytes deep which makes $N = 1 \times 8 = 8$.

Description : This command will add a user defined bitmapped character into the internal memory.

Example1: 41hex, 01hex, 18hex, 24hex, 42hex, 81hex, 81hex, 42hex, 24hex, 18hex

This adds and saves user defined 8x8 bitmap as character number 1 into memory as seen below.

b7 b6 b5 b4 b3 b2 b1 b0



data1 (hex = 18h)

data2 (hex = 24h)

data3 (hex = 42h)

data4 (hex = 81h)

data5 (hex = 81h)

data6 (hex = 42h)

data7 (hex = 24h)

data8 (hex = 18h)

Example of a 8x8 user defined bitmap

2.1.12 Display User Bitmapped Character (D)

Syntax : cmd, char#, x, y, colour(msb), colour(lsb)

cmd : 44hex, Dascii

char# : which user defined character number to display from the selected group. 0dec to 63dec (00hex to 3Fhex), of 8x8 format.

x : horizontal display position of the character. 0dec to 127dec (00hex to 7Fhex).

y : vertical display position of the character. 0dec to 127dec (00hex to 7Fhex).

colour : character colour value: 2 bytes (16 bits) msb, lsb

65,536 colours to choose from

Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command displays the previously defined user bitmapped character at location (x, y) on the screen. User defined bitmaps allow drawing & displaying unlimited graphic patterns quickly & effectively.

Example 1: 44hex, 01hex, 00hex, 00hex, F8hex, 00hex

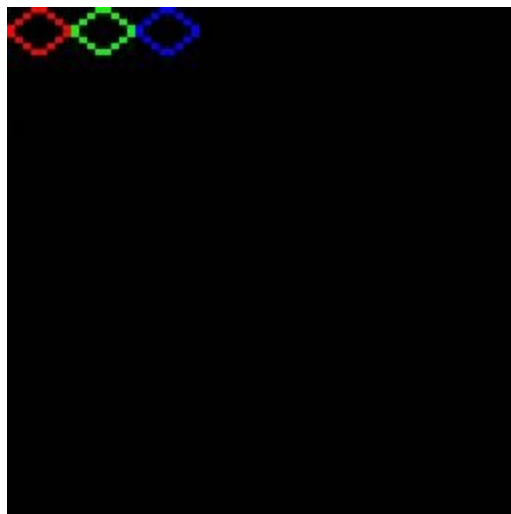
Display 8x8 bitmap character number 1 at x = 0, y = 0, colour = red

Example 2: 44hex, 01hex, 08hex, 00hex, 07hex, E0hex

Display 8x8 bitmap character number 1 at x = 8, y = 0, colour = green

Example 3: 44hex, 01hex, 10hex, 00hex, 00hex, 1Fhex

Display 8x8 bitmap character number 1 at x = 16, y = 0, colour = blue



2.1.13 paint Area (p)

Syntax : cmd, x1, y1, x2, y2, colour(msb), colour(lsb)

cmd : 70hex, pascii

x1 : top left horizontal start position of the Area. 0 to 127 (00hex to 7Fhex).

y1 : top left vertical start position of the Area. 0dec to 127 (00hex to 7Fhex).

x2 : bottom right horizontal end position of the Area. 0 to 127 (00hex to 7Fhex).

y2 : bottom right vertical end position of the Area. 0 to 127 (00hex to 7Fhex).

colour : Area colour value: 2 bytes (16 bits) msb, lsb

65,536 colours to choose from

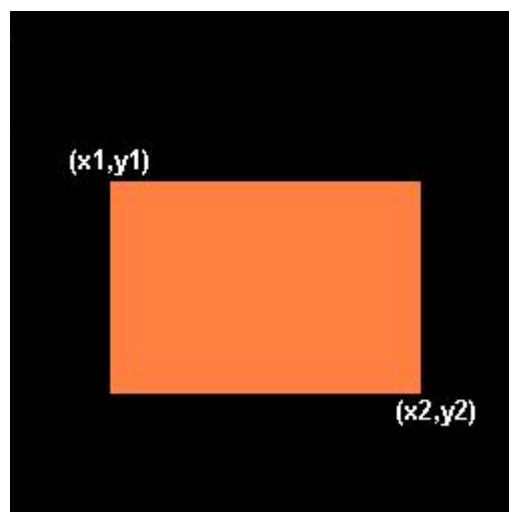
Black = 0000hex, 0dec

White = FFFFhex, 65,535dec, 1111111111111111bin

Description : This command will paint a specified area on the screen. x1, y1 refers to the top left corner of the area and x2, y2 refers to the bottom right hand corner of the area on the screen. If colour is chosen to be that of the background then the effect will be erasure.

Example : 70hex, 00hex, 00hex, 10hex, 10hex, 00hex, 00hex

Paint the area BLACK that has its top left corner at x1=0, y1=0 and its bottom RIGHT corner at x2=16, y2=16.





2.2 μ LCD-128-xMb Additional Command Set

The following are additional commands related to the μ LCD-128-xMb range of microLCD modules and they are described in this section. The μ LCD-128-xMb modules incorporate additional enhanced hardware and software features. Depending on the model, these devices have onboard flash memory ranging from 1Mb to 8Mb.

The μ LCD-128-xMb modules incorporate all of the μ LCD-128 commands, as described in the previous section, plus the additional commands outlined here. The features and commands set out in this section help differentiate μ LCD-128-xMb enhanced models from the standard μ LCD-128 model.

You will find references being made to “**Objects**” throughout this section. An object can be simply defined as those commands that reside inside the flash memory (programmed/downloaded) and can be displayed on the screen by the “**Display Object from Flash Memory**” command.

There are also some commands that can only reside inside the flash memory and must be executed from there. These commands will return a NAK if executed live from the serial link.

Tables below list all of the commands for both μ LCD-128 and the μ LCD-128-xMb models.

Command Set Summary (μ LCD-128 & μ LCD-128-xMb)

	Live	Object	Flash
Erase Screen	x		x
Background Colour	x		x
Put Pixel	x		
Read Pixel	x		
Draw Circle	x	X	x
Draw Line	x	X	x
Font Size	x		x
Opaque or Transparent Text	x		
Place Text Character (formatted)	x	X	x
LCD Display Control Functions	x		x
Add User Bitmapped Character	x		
Display User Bitmapped Character	x		
Paint Area	x	X	x



Command Set Summary (uLCD-128-xMb only)

	Live	Object	Flash
Draw Circle with Fill	x	X	x
Place Text Character (unformatted)	x	X	x
Place String of ASCII Text (formatted)	x	X	x
Display Image	x	X	x
Set Transparent Colour (for Images only)	x		x
Enable/Disable Transparent colour	x		x
Place Button Icon	x	X	x
Block Copy & Paste (Screen Bitmap Copy)	x		
Display Object from Flash Memory	x		
Delay			x
Loop Start			x
Loop End			x
Run	x		
Exit	x		x
Restart			x
Download Data to Flash Memory	x		
Read Data from Flash Memory	x		
Erase Flash Memory	x		

NOTES:

Live : Those commands that can be sent via the serial link and executed by the host.

Object : Those commands that can be recalled from the flash memory at any time by the host and displayed on the screen using the “Display Object from Flash Memory” command.

Flash : Those commands that can reside and be executed from inside the flash memory.

2.2.1 Draw Circle with Fill (i)

Syntax : `cmd, x, y, rad, colour(msb), colour(lsb)`

cmd : 69hex, iascii

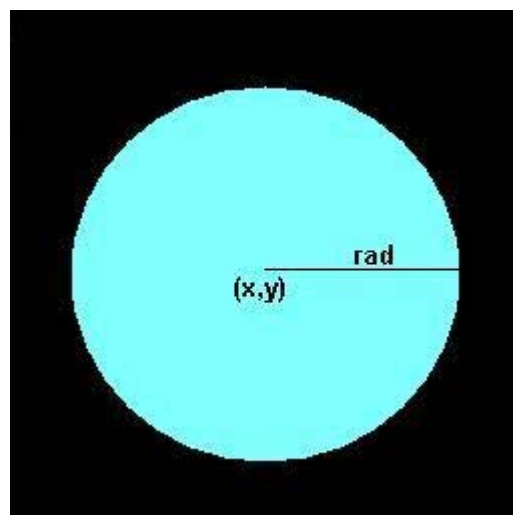
x : circle horizontal centre position. 0dec to 127dec (00hex to 7Fhex).

y : circle vertical centre position. 0dec to 127dec (00hex to 7Fhex).

rad : length of circle radius (in pixel units). 0dec to 127dec (00hex to 7Fhex).

colour(msb), colour(lsb) : 2 byte circle colour

Description : This command will draw a coloured fill circle centred at (x, y) with a radius determined by the value of **rad**.





2.2.2 Place text Character (unformatted) (t)

Syntax : cmd, char, x, y, colour(msb), colour(lsb), width, height

cmd : 74hex, tascii

char : inbuilt standard ASCII character, 32dec to 127dec (20hex to 7Fhex)

x : the horizontal position of character (in pixel units).

y : the vertical position of character (in pixel units).

colour(msb), colour(lsb) : 2 byte colour of the character

width : horizontal size of the character, n x normal size

height : vertical size of the character, m x normal size

Description : This command will place a coloured built in ASCII character anywhere on the screen at a location specified by (**x, y**). Unlike the 'T' command, this option allows text of any size (determined by **width** and **height**) to be placed at any position. The font of the character is determined by the '**Font Size**' command.



2.2.3 Place string of Ascii Text (formatted) (s)

Syntax : cmd, column, row, font_size, colour(msb), colour(lsb), char1, charN, terminator

cmd : 73hex, sascii

column : the horizontal start position of string

row : the vertical start position of string

font_size : 0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font. This has precedence over the Font command.

colour(msb), colour(lsb) : 2 byte colour of the string

char1..charN : string of ASCII characters (max 256 characters)

terminator : string terminator, must be 00hex

Description : This command allows the display of a string of ASCII characters. The horizontal start position of the string is specified by **column** and the vertical position is specified by **row**. The string must be **terminated** with 00hex. If the length of the string is longer than the maximum number of characters per line, then a wrap around will occur on to the next line. Maximum string length is **256 bytes**.

2.2.4 Display Image (I)

Syntax : cmd, x, y, width, height, colour_mode, pixel1, .. pixelN

cmd : 49hex, Iascii

x : Image horizontal start position (top left corner)

y : Image vertical start position (top left corner)

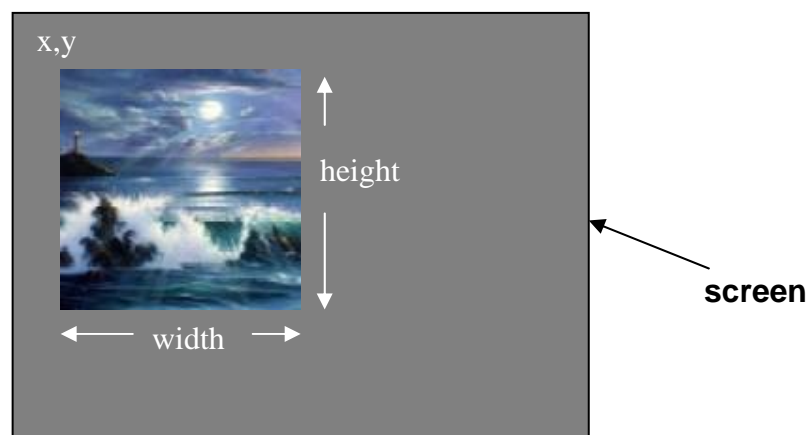
width : horizontal size of the image

height : vertical size of the image

colour_mode : 8dec = 256 colour mode, 8bits/1byte per pixel
16dec = 65K colour mode, 16bits/2bytes per pixel (msb, lsb)

pixel1..pixelN : image pixel data and N is the total number of pixels
N = height x width when colour_mode = 0
N = height x width x 2 when colour_mode = 1

Description : This command displays a bitmap image on to the screen with the top left corner specified by (x, y) and size of the image specified by **width** and **height** parameters. This command is more effective than using the "Put Pixel" command, where there are no overheads in specifying the x, y location of each pixel.



2.2.5 Set TraNsparent Colour (for Images only) (N)

Syntax : `cmd, colour(msb), colour(lsb)`

cmd : 4Ehex, Nascii

colour(msb), colour(lsb) : 2 byte transparency colour

Description: When a 2 byte transparent colour is specified, it inhibits those matching colour values of the bitmap image from being displayed or painted on to the screen. For example, a small icon image maybe surrounded by an area of a certain background colour. If we know the colour value of the background, we can stop it from being displayed on the screen and only display the image of interest.

For example, the smiley on the left has a red background colour and if we set our transparent colour value to match the same red colour value, the resultant display on the screen will be the smiley on the right.

The power up default value = 0xFFFF (White)





2.2.6 enable/disable Transparent colour (e)

Syntax : cmd, mode

cmd : 65hex, eascii

mode : 0 = Disable Transparent colour (default)
1 = Enable Transparent colour

Description : The Transparent Colour function can be enabled or disabled by executing this command. See “Set Transparent Colour” command for a detailed explanation.

2.2.7 Place button Icon (b)

Syntax : `cmd, state, x1, y1, x2, y2, colour(msb), colour(lsb)`

cmd : 62hex, bascii

state : Specifies whether the displayed button is drawn as UP (not pressed) or DOWN (pressed).

0 = Button Down (pressed)

1 = Button Up (not pressed)

x1 : top left horizontal start position of the button

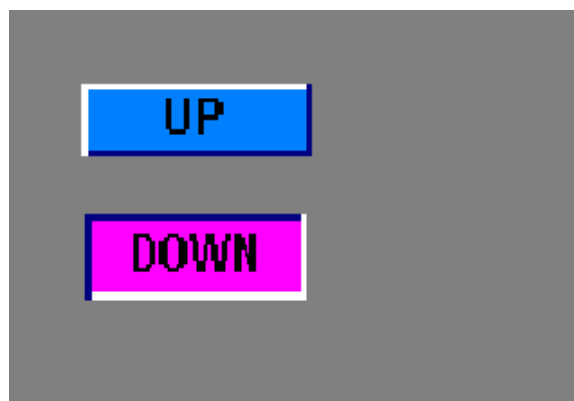
y1 : top left vertical start position of the button

x2 : bottom right horizontal end position of the button

y2 : bottom right vertical end position of the button

colour(msb), colour(lsb) : 2 byte button colour value

Description : This command will place a Button similar to the ones used in a PC Windows environment on the screen. **(x1, y1)** refers to the top left corner of the button and **(x2, y2)** refers to the bottom right hand corner of the button on the screen. The button can be displayed in an UP (button not pressed) or DOWN (button pressed) position by specifying the appropriate value in the **state** byte. Text can be placed inside the button, using the 't' "**Place Text Character**" (unformatted) command, describing the button function.



2.2.8 Block copy & Paste (Screen Bitmap Copy) (c)

Syntax : cmd, x_source, y_source, x_dest, y_dest, width, height

cmd : 63hex, cascii

x_source: top left horizontal start position of block to be copied

y_source: top left vertical start position of block to be copied

x_dest: top left horizontal start position of where copied block is to be pasted

y_dest: top left vertical start position of where the copied block is to be pasted

width : horizontal size of the block to be copied

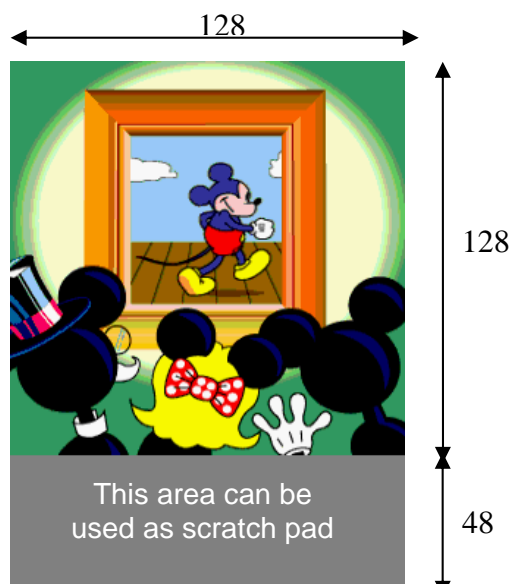
height : vertical size of the block to be copied

Description : This command copies an area of a bitmap block of specified size defined by the **width** and the **height** parameters. The start location of the block to be copied is represented by **x_source, y_source** (top left corner) and the start location of where the block is to be pasted to, is represented by **x_dest, y_dest** (top left corner).

This is a very powerful feature for animating objects, smooth scrolling, implementing a windowing system or copying patterns across the screen to make borders or tiles.

The Graphics RAM or the Screen Memory is made up of 128 x 176 pixels. Only the 128 x 128 portion is used for the display. Therefore there is a 128 x 48 area of memory that can be used as a temporary scratch pad to copy and paste blocks of bitmap data. This is ideal for animations, where the area under the object can be copied over to the scratch pad before it's displayed and then re-pasted back once the object has been moved to a different screen location. The following indicates how the graphics RAM is utilised:

<u>Graphics RAM</u>	<u>Comment</u>
x = 0, y = 0 to x = 127, y = 127	Used for the Screen Display
x = 0, y = 128 to x = 127, y = 175	Scratch Pad area





2.2.9 Display Object from flash Memory (f)

Syntax : cmd, address(hi), address(mid), address(lo)

cmd : 66hex, fascii

address : 24bit (3 bytes) address of the object in flash ROM

(hi) = Address high byte

(mid) = Address middle byte

(lo) = Address low byte

Description: Some of the commands can be stored as objects in the internal flash which can be later recalled by the host on demand and displayed or executed. The user must make sure the 24 bit address of each stored command/object is known before using this feature.

For example, a series of images can be stored as icons and later displayed as the application requires them. The following is a list of all the commands that can be stored as objects within the internal flash ROM:

Circle (empty), Circle (fill), Line, Text Character (formatted), Text Character (unformatted), Radio Button, Paint Area, Display String, Display Image.



2.2.10 Delay (must reside inside the flash) (04_{hex})

Syntax : cmd, value(msb), value(lsb)

cmd : 04_{hex}

value(msb, lsb) : A 2 byte delay value in milliseconds. Maximum value of 65,535 milliseconds or 65.5 seconds.

Description : When objects from the flash memory such as images are displayed sequentially, a delay can be inserted between subsequent objects. A delay basically has the same effect as a NOP (No Operation) which can be used to determine how long the object stays on the screen before the next object is displayed.

2.2.11 Loop Start (must reside inside the flash) (0D_{hex})

Syntax : cmd, counter

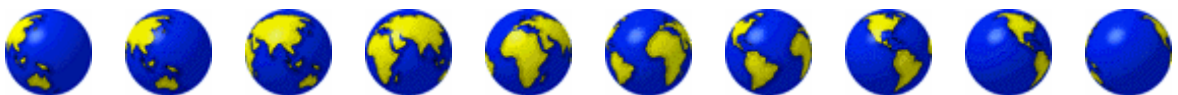
cmd : 0D_{hex}

counter : A 1 byte counter that determines how many times the loop occurs between the “Loop Start” and “Loop End” commands. Practical values should be between 2 and 255.

Description : A series of images that might be part of an animation may need to be redisplayed over and over to achieve a lengthy viewing. This command allows the user to determine exactly how many times the series of images are looped. This command must always be terminated with the “Loop End” command.

For example, we may want to animate the Globe rotating. Let’s say we have 10 image slides of the Globe at different rotated positions residing in the flash memory. When the images are displayed sequentially, the effective duration will only be the length of time it takes to display the 10 image frames. With this command, we can increase that length by looping thru the animation a number of times depending on the value set in the **counter** byte. When the display reaches the end of the last frame and encounters the “Loop End” command, the counter will be decremented and then the internal pointer will jump back to the object just after the “Loop Begin” command. This sequence will then repeat until the value in the counter reaches zero. The following demonstrates how this may be used:

```
Loop_Start, (counter = 25),
Image1,
Delay(10ms),
Image2,
Delay(10ms),
...,
...,
Image10,
Delay(10ms),
Loop_End (decrements counter then jumps to Image1)
```





2.2.12 Loop End (must reside inside the flash) (0C_{hex})

Syntax : cmd

cmd : 0C_{hex}

Description : Decrements the “Loop Start” counter and forces the internal flash pointer to jump to that command/object, which resides in the flash memory, just after the “Loop Start” command. When the “Loop Start” counter becomes zero, the “Loop End” command is bypassed and the command/object following it is executed. See “Loop Start” command for a detailed explanation.



2.2.13 Run (06_{hex})

Syntax : cmd

cmd : 06_{hex}

Description : The Run command forces the 24bit internal flash pointer to reset to zero (000000hex) and automatically start executing commands, from the flash memory, without any further interaction by the host processor. It will sequentially execute any valid flash related commands and display objects until it gets to the end of the flash memory. It is advisable to have the “Exit” or the “Restart” command at the end of the user composed slide show so that the pointer does not run off so to speak.



MicroLCD

2.2.14 Exit (07_{hex})

Syntax : cmd

cmd : 07_{hex}

Description : This command forces the program to stop executing from the internal flash memory and ready to accept and execute commands from the host via the serial interface. It can reside in the flash which will force the slide show to stop or it can be sent via the serial port while the program is running from the internal flash.



2.2.15 Restart (must reside inside the flash) (05_{hex})

Syntax : cmd

cmd : 05_{hex}

Description : The Restart command forces the 24bit internal flash pointer to reset to zero (000000hex). If a certain slide show is composed inside the flash memory and the “Restart” command is encountered at the end of it, the program will then jump back to the start of the flash and begin executing again.



2.2.16 Download Data to Flash Memory (08hex)

Syntax : `cmd, pageNum(msb), pageNum(lsb), data(1), .. , data(256),
chkSum(msb), chkSum(lsb)`

cmd : 08hex

pageNum(msb, lsb) : A 2 byte page number from 0 to 32,767. The internal flash memory has 4096 pages for every 1Mbyte and each page is 256 bytes long.

data(1 to 256) : 256 bytes of data. The data length must be 256 bytes long. If not all used then the rest must be padded.

chkSum(msb, lsb) : A 2 byte checksum value. The checksum is calculated by taking a binary sum of all bytes in the command frame and forming a 2 byte value, from (and including) the pageNum(msb) byte to the last data byte data(256). Then, the two's complement is taken (i.e., invert all bits of the result and then add 1) to yield the checksum.

Note: If the command message is correct, adding the checksum word to the 2 byte sum of all the other bytes (excluding cmd) in the message will give a result of zero.

Description : This command allows downloading of objects such as images and other commands for storage that can be retrieved and used later on. For example, the μ LCD-128-1Mb has a 1Mbyte of flash memory which is divided into 4096 pages and each page is 256 bytes long. Downloads must always be limited to 256 bytes in length. For large objects such as images, the data must be broken up into multiple pages (chunks of 256 bytes) and this command then maybe used many times until all of the data is downloaded. There is no provision to download a single byte or few bytes and the data length must always be 256 bytes long. If only few bytes of data are to be downloaded, then make sure the rest of the remaining data are padded with 00hex or FFhex (it can be anything).

Once the command message is sent and the checksum is correct, the μ LCD will take a few milliseconds to write the data into its flash memory and at the end of which it will reply back with an **ACK**(06hex). If the checksum is incorrect a **NAK**(15hex) will be sent back without any write attempts.

Only **data(1)** to **data(256)** are stored in the flash. Other bytes in the command message such as page number and checksum are not stored.



2.2.17 Read Data from Flash Memory (09_{hex})

Syntax : cmd, pageNum(msb), pageNum(lsb)

cmd : 09_{hex}

pageNum(msb, lsb) : A 2 byte page number from 0 to 32,767. The internal flash memory has 4096 pages for every 1Mbyte and each page is 256 bytes long.

Description : This command provides a means of reading data back from the flash memory in lengths of 256 bytes. It maybe useful in validating the data that was stored previously using the download command. Once this command is sent, the μ LCD will return 256 bytes of data relating to that particular page.



2.2.18 Erase Flash Memory (0A_{hex})

Syntax : cmd

cmd : 0A_{hex}

Description : Before any data can be downloaded and stored in the flash memory, it must first be erased. There is no provision to erase individual bytes or pages of memory. Once this command is executed, all of the 1Mbytes (4096 pages) of memory will be erased. It will take around 10 seconds to completely erase the flash at the end of which an **ACK**(06_{hex}) is returned.



2.3 μ LCD Serial Interface (TTL)

The μ LCD needs to be connected via a serial link to a host system. The host uses this serial link to send commands to the μ LCD so that characters and graphics can be displayed on the screen. Use the signal pin-outs as well as the application example shown in the following section for correct connection to the host.

Please note that the serial connection (Rx/Tx) is at TTL levels (0 – 3.3V) and the logic levels are “high” = 1 = 3.3V, “low” = 0 = 0V. If interfacing to a host system running at 5V levels, then 1K series resistors must be inserted between the Host Tx/Rx and the μ LCD Rx/Tx signals.

Auto Baud Detect:

As previously mentioned, the μ LCD core has an auto-baud detect function which can operate from **300 baud to 128000 baud**. Prior to any graphical formatting and commands being sent to the core, it must first be initialized by sending the ASCII character ‘U’ (55h) after power-up. This will allow the core to determine and lock on to the baud rate of the host automatically without needing any further setup. This must be done every time the core is powered up.

If the host needs to change the baud rate, the μ LCD must be powered down and powered back up again. The “U” command cannot be used to change the baud rate during the middle of normal usage.

Serial Timing:

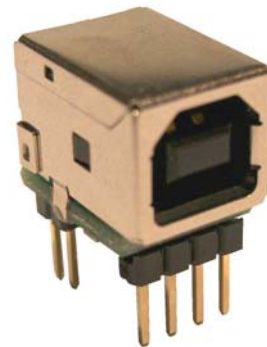
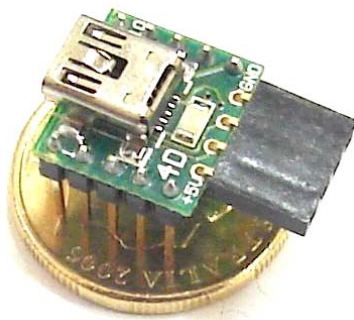
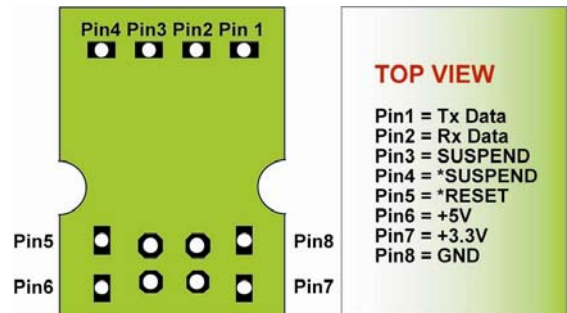
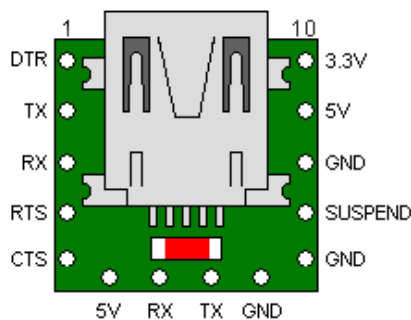
Each μ LCD command is made up of a sequence of data bytes. Some commands are a single byte and others are multiple bytes. When a command is sent to the μ LCD and the operation is completed, the μ LCD will reply back with a single acknowledge byte called the **ACK** (06h). This tells the host that the command was understood and the operation is completed. It will take the μ LCD anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the μ LCD has to perform. If the μ LCD receives a command that it does not understand it will reply back with a negative acknowledge called the **NAK** (15h).

For example, if a command has 5 bytes but only 4 bytes are sent, the command will not be executed and when the next following command bytes are sent the μ LCD will reply back with a **NAK** for each and every byte it receives. For correct operation make sure the command bytes are sent in the correct sequence.

Note: No termination character is to be sent at the end of the command sequence. i.e. don't send any CR, or Null, or any other end of command bytes.

2.4 μ LCD USB Interface

The μ LCD can be interfaced to a PC using a standard USB cable and any one of the 4D Systems microUSB modules (uUSB or the uUSB-MB) as shown below. The microUSB module (optional extra), simply connects to the μ LCD 4 pin header and captures the USB data and converts it into serial TTL data. The microUSB modules and drivers are available from your local 4D distributor. These are separate products and are not included with the μ LCD module.



A microUSB example interface, with uUSB-MB module



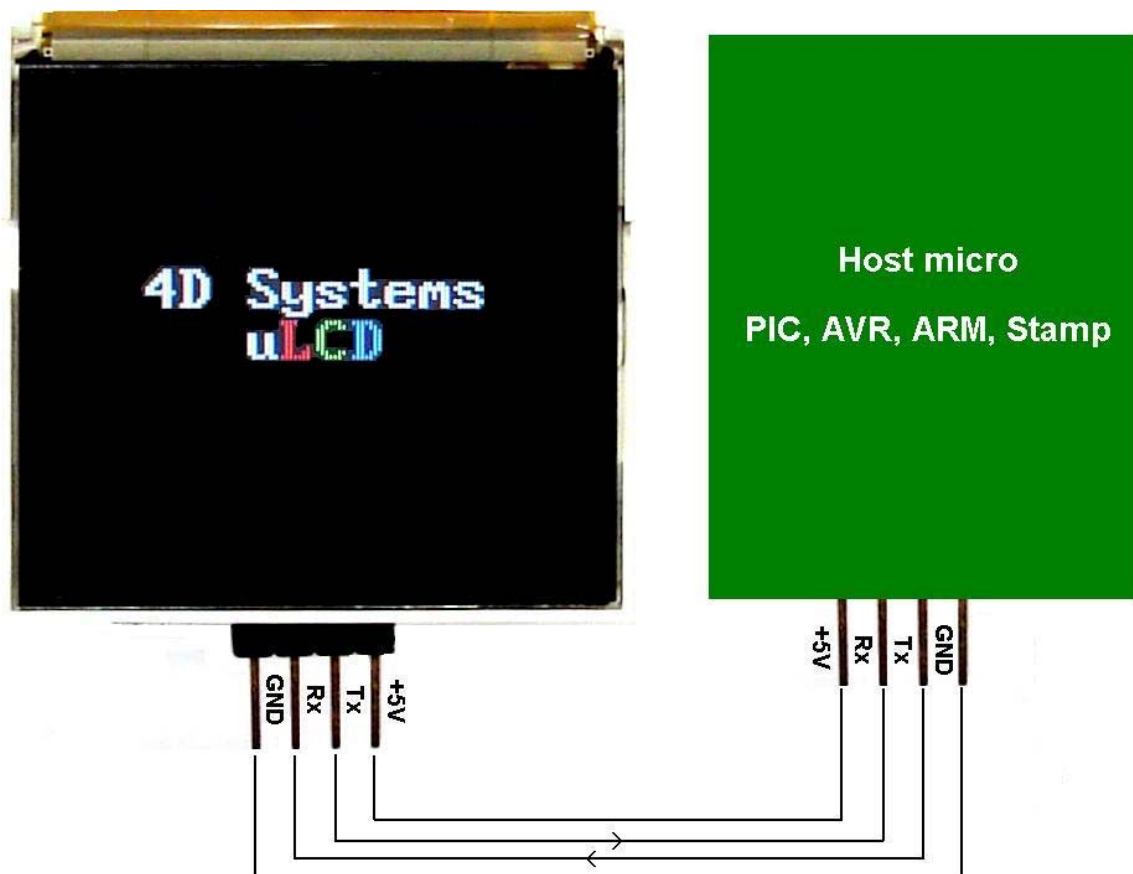
A microUSB example interface, with uUSB module

3. Specifications

The μ LCD has the following electrical specifications which must be adhered to at all times to prevent damage to the device. The μ LCD module footprint is 38mm x 38mm.

Symbol	Characteristic	Min	Typ	Max	Units
Vdd	Supply voltage	4.5V	5V	5.5V	V
I	Current	70mA	80mA	90mA	mA
Deg C	Operating temp	0	30	70	C
Tpu	Power-up delay	800		1000	mS

3.1 μ LCD Host Interface pin-outs



3.2 65,536 Colour Bitmap Organisation

The μ LCD 65K colour byte is organised as 5 bits for Red(D11, D12, D13, D14, D15), 6 bits for Green(D5, D6, D7, D8, D9, D10) and 5 bits for Blue(D0, D1, D2, D3, D4). This will give a combination of $32 \times 64 \times 32 = 65,536$ colours. Each colour is not limited to 4 shades. For example a lighter shade of Red can be obtained by adding a little bit of the Green and a little bit of the Blue. Full Red and full Green will result in Yellow. Some experimentation will be needed to obtain the desired colour.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B
E	E	E	E	E	R	R	R	R	R	R	L	L	L	L	L
D	D	D	D	D	E	E	E	E	E	E	U	U	U	U	U
					N	N	N	N	N	N	E	E	E	E	E
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

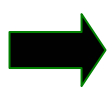
Example: To Obtain the Colour Yellow

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B
E	E	E	E	E	R	R	R	R	R	R	L	L	L	L	L
D	D	D	D	D	E	E	E	E	E	E	U	U	U	U	U
					N	N	N	N	N	N	E	E	E	E	E
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0



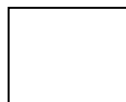
Example: To Obtain the Colour Magenta

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B
E	E	E	E	E	E	E	E	E	E	E	L	L	L	L	L
D	D	D	D	D	N	N	N	N	N	N	U	U	U	U	U
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1



Example: To Obtain the Colour White

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B
E	E	E	E	E	E	E	E	E	E	E	L	L	L	L	L
D	D	D	D	D	N	N	N	N	N	N	U	U	U	U	U
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1





3.3 Power-Up Reset

When the μ LCD comes out of a power up reset it initialises the video RAM and the internal Display registers. Allow up to 800ms to 1000ms before attempting to communicate with the μ LCD. The power up sequence of events should be as follows:

- Allow 800ms to 1000ms after power-up for μ LCD to settle. Do not attempt to communicate with the μ LCD during this period. The μ LCD may send garbage on its Tx Data line during this period, the host should disable its Rx Data reception.
- The host transmits the ASCII 'U' (capital U, 55hex) as the first command so the μ LCD can lock onto the hosts serial baud rate. The μ LCD will respond with an 'ACK' (06h). See section 2.3
- The μ LCD is now ready to accept screen function commands from the host.



4. Appendix

4.1 Available Models:

- **uLCD-128** (Standard module)
- **uLCD-128-1Mb** (Enhanced module with additional features and 1Mb Flash)
- **uLCD-128-4Mb** (Enhanced module with additional features and 4Mb Flash)
- **uLCD-128-8Mb** (Enhanced module with additional features and 8Mb Flash)

Please check stock availability with your local supplier.

4.2 Related Products:

- **uUSB**
 - microUSB module, USB to Serial Bridge
 - Standard USB B connector
 - 8 pin header provides the following signals:
 - 5V, 3.3V, GND, Tx, Rx, Reset, Suspend, -Suspend
 - 5 Volts supply @ 500mA, 3.3 Volts supply @ 100mA
- **uUSB-MB**
 - microUSB module, USB to Serial Bridge
 - Standard USB miniB connector
 - 10 pin header provides the following signals:
 - 5V, 3.3V, GND, Tx, Rx, Suspend, DTR, CTS, RTS, GND
 - 5 Volts supply @ 500mA, 3.3 Volts supply @ 100mA
 - Additional flow control signals, DTR, CTS, RTS
 - Available with an additional 4pin header for uLCD interface
- **USB-5V**
 - USB to 5 Volts supply module upto 500mA
 - Also an adaptor board for the uUSB module
 - Provides the 4pin header for the uLCD interface
- **Graphics Composer (free download)**
 - PC based software utility for Windows
 - Download images/text/animations into the uLCD-128-xMb flash memory.
 - For software and user guide downloads please visit the **uLCD** web-page of your local distributor.



4.3 Auto Demo/Slide Show:

The **uLCD-128-xMb** modules come with a Slide Show that is factory loaded into the onboard Flash Memory. There is a **2 pin jumper** at the back of the unit (on the component side). Upon power-up, if the shunt is inserted and there are preloaded objects in the flash memory(images/text/animations), the uLCD-128-xMb module will automatically play/display these from the onboard flash.

For normal usage this jumper must be **removed**. There is no jumper present on the **uLCD-128** module.

For the earlier model **uLCD-MkII** there was no demo jumper available. Instead the Demo functioned if the “IN1” pin (on the 10 way pins) was connected to GND prior to powering up the device.

4.4 Precautions:

- Avoid having to display the same image/object on the screen for lengthy periods of time. This will cause a burn-in which is a common problem with all types display technologies. Blank the screen after a while or dim it very low by adjusting the contrast. This can be achieved via the “**uLCD Display Control Functions**” command (section 2.1.10). Better still, implement a screen saver feature.
- Observe the Power-Down procedure (section 2.1.10). The uLCD module automatically takes care of the proper Power-Up sequence.

4.5 Help and Other Information:

- Assistance with latest information and downloads visit the **μLCD** product web-page of your distributor.
- Questions and technical support please email support@4dsystems.com.au