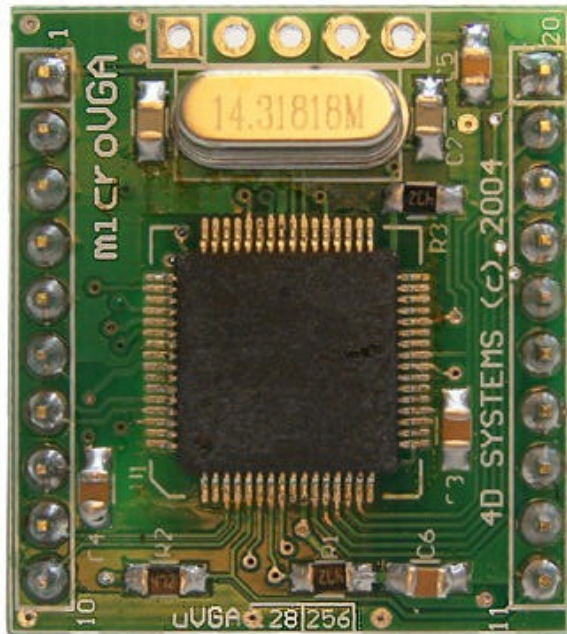




hiRes Core
USERS MANUAL
Revision 1.1



4D Systems



4D Systems

MicroVGA

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. 2000-2005



Table of contents

1. μ VGA Core description

- 1.1 Introduction
- 1.2 μ VGA core types
- 1.3 μ VGA core features

2. μ VGA Core command set

- 2.1 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 Place Text Character (unformatted)
 - 2.1.11 Place Ascii String Text (formatted)
 - 2.1.12 Add User Bitmapped Character
 - 2.1.13 Display User Bitmapped Character
 - 2.1.14 Display Image
 - 2.1.15 Paint Area
 - 2.1.16 Place Button Icon
 - 2.1.17 Plot Graph
 - 2.1.18 Block Copy
 - 2.1.19 Video Memory (Graphics RAM) Read/Write/Display
 - 2.1.20 Sync Lock
 - 2.1.21 Core Version Request
- 2.2 μ VGA Serial Interface
- 2.3 μ VGA USB Interface
- 2.4 Demonstration mode
- 2.5 Base Board Systems

3. Specifications

- 3.1 Core types and resolutions
- 3.2 μ VGA core pin-outs (all cores)
- 3.3 64 Colour Bitmap Organisation
- 3.4 256 Colour Bitmap Organisation
- 3.5 Power-Up Reset
- 3.6 Mechanical Details
- 3.7 Application examples



1 μ VGA Core Description

1.1 Introduction

The μ VGA core is a ready to go 'drop in' embedded graphics engine that will deliver 'stand-alone' VGA functionality to your project without the need to have a computer attached. 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 64 or 256 colours whilst freeing up the host processor from the 'processor hungry' screen control functions. This means a simple microcontroller with a standard serial interface can drive the module with ease. Figures 1-4 show some of the graphics capability of the μ VGA.

Fig1.

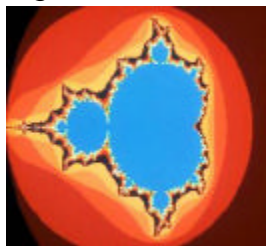


Fig2.

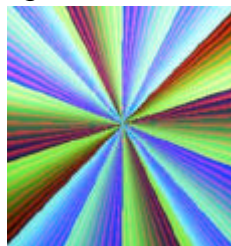


Fig3.

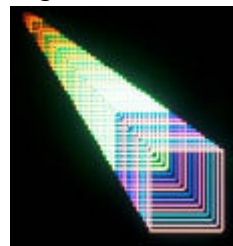
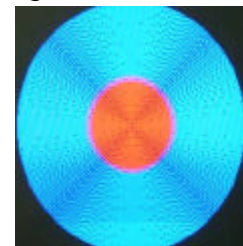


Fig4.



1.2 μ VGA hires core types

The μ VGA hires core comes in a range of types to suit any application in the following resolutions:

μ VGA-320	320 x 220 by 64 or 256 colours
μ VGA-420	420 x 350 by 64 or 256 colours
μ VGA-620	620 x 420 by 64 or 256 colours
μ VGA-640	640 x 480 by 64 or 256 colours
μ VGA-720	720 x 500 by 64 or 256 colours

The above mentioned cores all share the same 'foot-print' so that if a different resolution is required, it can be upgraded simply by replacing the μ VGA core module.



1.3 μVGA hiRes Core features

The μVGA hires core is aimed at being integrated into a number of different applications via a wealth of features designed to facilitate VGA functionality quickly and cheaply and thus reduce ‘time to market’. These features are as follows:

- ✍ 64 or 256 standard VGA colours.
- ✍ Three selectable font sizes (5x7, 8x8 and 8x12 – see Fig 5).
- ✍ User defined bitmapped characters, 64 by 8x8, 16 by 16x16 and 8 by 32x32.
- ✍ Vector drawing controls for circles, lines, squares.
- ✍ Standard ‘built in’ ASCII character set (See fig 5).
- ✍ Assorted pixel resolutions (please see ‘electrical specifications’).
- ✍ Logic level RS232 serial interface to μVGA core from host.
- ✍ Auto baud rate detection from 300 baud to 1Mbit/sec.
- ✍ USB Interface available (micro-USB).

Fig 5.





2 μ VGA hiRes Core Command Set

The heart of the μ VGA 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 μ VGA via its serial connection. **Please note that these signals are at TTL levels.** All cores have a 64 or 256 colour option.

2.1 Command set

(E) Erase Screen	1 byte	2.1.1	8
(B) Background Colour	2 bytes	2.1.2	9
(P) Put Pixel	6 bytes	2.1.3	10
(R) Read Pixel	5 bytes	2.1.4	11
(C) Draw Circle	7 bytes	2.1.5	12
(L) Draw Line	10 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 Text Character (formatted)	5 bytes	2.1.9	16
(t) Place text Character (unformatted)	9 bytes	2.1.10	17
(s) Place string ascii Text (formatted)	6 bytes + String	2.1.11	18
(A) Add User Bitmapped Character	11 to 131 bytes	2.1.12	19
(D) Display User Bitmapped Character	8 bytes	2.1.13	21
(I) Display Image	9 bytes + Image	2.1.14	23
(p) paint Area	10 bytes	2.1.15	24
(b) Place button Icon	11 bytes	2.1.16	25
(G) Plot Graph	12 bytes + Graph	2.1.17	26
(c) Block copy (Bitmap-Image Copy)	15 bytes	2.1.18	28
(v) video Memory (Graphics RAM)	3 bytes	2.1.19	30
(S) Sync Lock	1 byte	2.1.20	31
(V) Core Version Request	1 byte	2.1.21	32



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 resolution nature of the cores, a pixel horizontal and vertical address coordinates (x, y) will not fit into a single byte, a byte can only hold a maximum value of 255. Therefore each of the (x, y) pixel address data are represented as a 2 byte value, **x(msb)**, **x(lsb)** and **y(msb)**, **y(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 coordinate value of **013Fhex** the byte order can be shown as (**01hex**),(**3Fhex**).

NOTE: When transmitting the command and data bytes to the μ VGA, 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 VGA.

When a μ VGA command is sent, the μ VGA 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 VGA anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the μ VGA has to perform. If the μ VGA 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 7 bytes but only 6 bytes are sent, the command will not be executed and the μ VGA will wait until another byte is sent before trying to execute the command. There is no timeout on the μ VGA when incomplete commands are sent. The μ VGA 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.

IMPORTANT NOTE:

Some Cores may not utilise the full screen area of your monitor, therefore you may want to adjust your monitor settings to maximise the display area.



2.1.1 Erase Screen

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

Syntax : cmd, color

cmd : 42hex, Bascii

color : pixel colour value

64 colours to choose from for 64-Colour cores

Black = 00hex, 0dec

White = 3Fhex, 63dec, 00111111bin

256 colours to choose from for 256-Colour cores

Black = 00hex, 0dec

White = FFhex, 255dec, 11111111bin

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, FFhex

Set the background colour to value 255 (white) in a 256 colour core.



2.1.3 Put Pixel

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

cmd : `50hex, Pascii`

x(msb) : high order byte of the horizontal pixel position. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : low order byte of horizontal pixel position. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : high order byte of the vertical pixel position. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : low order byte of vertical pixel position. Refer to Section 3.1 for ranges for y for different cores.

colour : pixel colour value

64 colours to choose from for 64-Colour cores

Black = `00hex, 0dec`

White = `3Fhex, 63dec, 00111111bin`

256 colours to choose from for 256-Colour cores

Black = `00hex, 0dec`

White = `FFhex, 255dec, 11111111bin`

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

Example : `50hex, 01hex, 7Chex, 01hex, 22hex, 3Fhex`

Puts a white (`3Fhex`) pixel at location x = `380dec (017Chex)` and y = `290dec (0122hex)` in a 64 colour core.



2.1.4 Read Pixel

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

cmd : `52hex, Rascii`

x(msb) : high order byte of the horizontal pixel position. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : low order byte of horizontal pixel position. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : high order byte of the vertical pixel position. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : low order byte of vertical pixel position. Refer to Section 3.1 for ranges for y for different cores.

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, 7Chex, 01hex, 22hex`

uVGA reply : `3Fhex`

Reads a white (`3Fhex`) pixel at location $x = 380_{dec}$ (`017Chex`) and $y = 290_{dec}$ (`0122hex`) in a 64 colour core.



2.1.5 Draw Circle

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

cmd : `43hex, Cascii`

x(msb) : msb byte of the horizontal position of the centre of circle. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : lsb byte of the horizontal position of the centre of circle. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : msb byte of the vertical position of the centre of circle. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : low order byte of vertical pixel position. Refer to Section 3.1 for ranges for y for different cores.

rad : radius size of the circle:
0 to 255 (`00hex` to `FFhex`) (for all cores)

color : circle colour value:
64 colours to choose from for 64-Colour cores
Black = `00hex`, `0dec`
White = `3Fhex`, `63dec`, `00111111bin`
256 colours to choose from for 256-Colour cores
Black = `00hex`, `0dec`
White = `FFhex`, `255dec`, `11111111bin`

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

Example : `43hex, 01hex, 7Chex, 01hex, 22hex, 0Ahex, FFhex`

Draws a white circle (`FFhex`) centred at x = `380dec` (`017Chex`) and y = `290dec` (`0122hex`) with a radius of `10dec` (`0Ahex`) in a 256 colour core.



2.1.6 Draw Line

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

cmd : `4Chex, Lascii`

x1(msb) : msb byte of the horizontal position of line start. Refer to Section 3.1 for ranges for x for different cores.

x1(lsb) : lsb byte of the horizontal position of line start. Refer to Section 3.1 for ranges for x for different cores.

y1(msb) : msb byte of the vertical position of line start. Refer to Section 3.1 for ranges for y for different cores.

y1(lsb) : lsb byte of the vertical position of line start. Refer to Section 3.1 for ranges for y for different cores.

x2(msb) : msb byte of the horizontal position of line end. Refer to Section 3.1 for ranges for x for different cores.

x2(lsb) : lsb byte of the horizontal position of line end. Refer to Section 3.1 for ranges for x for different cores.

y2(msb) : msb byte of the horizontal position of line end. Refer to Section 3.1 for ranges for y for different cores.

y2(lsb) : lsb byte of the horizontal position of line end. Refer to Section 3.1 for ranges for y for different cores.

color : line colour value:

64 colours to choose from for 64-Colour cores

Black = `00hex, 0dec`

White = `3Fhex, 63dec, 00111111bin`

256 colours to choose from for 256-Colour cores

Black = `00hex, 0dec`

White = `FFhex, 255dec, 11111111bin`

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

Example : `4Chex, 00hex, 00hex, 00hex, 00hex, 01hex, A3hex, 01hex, 5Dhex, FFhex`

Draws a white line from (x1=0, y1=0) to (x2=419, y2=349) in a 256 colour core.



2.1.7 Font Size

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

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)

Syntax : cmd, char, column, row, color

cmd : 54hex, Tascii

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

column : horizontal position of character, see range below:

0 - 52 for 5x7 font, 0 - 39 for 8x8 and 8x12 font (320x220 core)
0 - 69 for 5x7 font, 0 - 51 for 8x8 and 8x12 font (420x350 core)
0 - 102 for 5x7 font, 0 - 76 for 8x8 and 8x12 font (620x420 core)
0 - 105 for 5x7 font, 0 - 79 for 8x8 and 8x12 font (640x480 core)
0 - 119 for 5x7 font, 0 - 89 for 8x8 and 8x12 font (720x500 core)

row : vertical position of character, see range below:

0 - 26 for 5x7 and 8x8 font, 0 - 17 for 8x12 font (320x220 core)
0 - 42 for 5x7 and 8x8 font, 0 - 28 for 8x12 font (420x350 core)
0 - 51 for 5x7 and 8x8 font, 0 - 34 for 8x12 font (620x420 core)
0 - 59 for 5x7 and 8x8 font, 0 - 39 for 8x12 font (640x480 core)
0 - 61 for 5x7 and 8x8 font, 0 - 40 for 8x12 font (720x500 core)

color : character colour value:

64 colours to choose from for 64-Colour cores

Black = 00hex, 0dec

White = 3Fhex, 63dec, 00111111bin

256 colours to choose from for 256-Colour cores

Black = 00hex, 0dec

White = FFhex, 255dec, 11111111bin

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 for a given core.

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

Place character 'A' (41hex) at column = 0, row = 0, colour = white (255) in a 256 colour core.



2.1.10 Place Text Character (unformatted)

Syntax : `cmd, char, x(msb), x(lsb), y(msb), y(lsb), color, x_size, y_size`

cmd : `74hex, tascii`

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

x(msb) : msb byte of the horizontal position of text. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : lsb byte of the horizontal position of text. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : msb byte of the vertical position of text. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : lsb byte of the vertical position of line start. Refer to Section 3.1 for ranges for y for different cores.

color : character colour value:

64 colours to choose from for 64-Colour cores

Black = `00hex, 0dec`

White = `3Fhex, 63dec, 00111111bin`

256 colours to choose from for 256-Colour cores

Black = `00hex, 0dec`

White = `FFhex, 255dec, 11111111bin`

x_size : horizontal size (width) of the character.

`01hex` = normal width

`02hex` = double width

`nhex` = n x normal width (do not use value of 0)

y_size : vertical size (height) of the character.

`01hex` = normal height

`02hex` = double height

`nhex` = n x normal height (do not use value of 0)

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 **x_size** and **y_size**) to be placed at any position. The font of the character is determined by the setting used by the 'Font Size' command.

Example : `74hex, 41hex, 00hex, AAhex, 00hex, 0Ahex, FFhex, 02hex, 03hex`

Place character 'A' (`41hex`) at x = 170, y = 10, colour = 255 (white), **x_size** = 2, **y_size** = 3 in a 256 colour core.



2.1.11 Place String of Ascii Text (formatted)

Syntax : cmd, align, column, row, color, char1, .. charN, terminator

cmd : 73hex, s ascii

align : Aligns the wrap around text to start from the same column position on the next line.

00hex - Wrap around text not aligned

01hex - Wrap around text aligned

column : horizontal position of start of string, see range below:

0 - 52 for 5x7 font, **0 - 39** for 8x8 and 8x12 font (320x220 core)

0 - 69 for 5x7 font, **0 - 51** for 8x8 and 8x12 font (420x350 core)

0 - 102 for 5x7 font, **0 - 76** for 8x8 and 8x12 font (620x420 core)

0 - 105 for 5x7 font, **0 - 79** for 8x8 and 8x12 font (640x480 core)

0 - 119 for 5x7 font, **0 - 89** for 8x8 and 8x12 font (720x500 core)

row : vertical position of start of string, see range below:

0 - 26 for 5x7 and 8x8 font, **0 - 17** for 8x12 font (320x220 core)

0 - 42 for 5x7 and 8x8 font, **0 - 28** for 8x12 font (420x350 core)

0 - 51 for 5x7 and 8x8 font, **0 - 34** for 8x12 font (620x420 core)

0 - 59 for 5x7 and 8x8 font, **0 - 39** for 8x12 font (640x480 core)

0 - 61 for 5x7 and 8x8 font, **0 - 40** for 8x12 font (720x500 core)

color : String colour value:

64 colours to choose from for 64-Colour cores

Black = **00hex**, **0dec**

White = **3Fhex**, **63dec**, **00111111bin**

256 colours to choose from for 256-Colour cores

Black = **00hex**, **0dec**

White = **FFhex**, **255dec**, **11111111bin**

char1, .. charN : String of ASCII characters.

terminator : String terminator, must be terminated with zero (**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. If the **align** byte is set to **01hex**, then the wrapped around text on the next line will be formatted and will start at the same column position as the 1st line. Maximum string length is **512 bytes**.



2.1.12 Add User Bitmapped Character

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

cmd : 41hex, Aascii

group : selects the appropriate bitmap format
00hex selects the 8x8 bitmap format
01hex selects the 16x16 bitmap format
02hex selects the 32x32 bitmap format

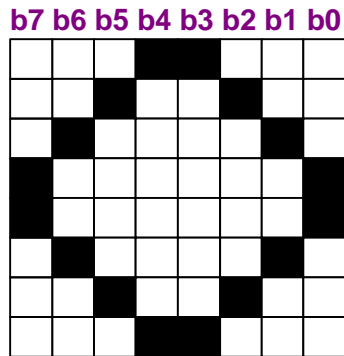
char# : bitmap character number to add to memory:
0 to 63 (00h to 3Fh), 64 characters of 8x8 format when group = 00h
0 to 15 (00h to 0Fh), 16 characters of 16x16 format when group = 01h
0 to 7 (00h to 07h), 8 characters of 32x32 format when group = 02h

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 = 1x8 = 8. The 16x16 bitmap composition is 2 bytes wide (16bits) by 16 bytes deep which makes N = 2x16 = 32. The 32x32 bitmap composition is 4 bytes wide (32bits) by 32 bytes deep hence N = 4x32 = 128.

Description : This command will add a user defined bitmapped character into the internal memory. There are 3 different size bitmaps available, 8x8 format, 16x16 format and 32x32 format. The desired format is selected by specifying the appropriate value in 'group'.

Example1: 41hex, 00hex, 01hex, 18hex, 24hex, 42hex, 81hex, 81hex, 42hex, 24hex, 18hex

This adds and saves user defined 8x8 bitmap (group 0) as character number 1 into memory as seen below.



- 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



Example2: 41hex , 01hex, 03hex, 3Fhex, FChex, 40hex, 02hex, 80hex, 01hex, BChex, 3Dhex, 98hex, 19hex, 98hex, 19hex, 81hex, 81hex, 81hex, 81hex, 80hex, 01hex, 40hex, 02hex, 24hex, 24hex, 23hex, C4hex, 21hex, 84hex, 10hex, 08hex, 08hex, 10hex, 07hex, E0hex

This adds and saves user defined 16x16 bitmap (group 1) as character number 3 into memory as seen below.

	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	
data1(hex=3Fh)																	data2(hex=FCh)
data3(hex=40h)																	data4(hex=02h)
data5(hex=80h)																	data6(hex=01h)
data7(hex=BC h)																	data8(hex=3Dh)
data9(hex=98h)																	data10(hex=19h)
data11(hex=98h)																	data12(hex=19h)
data13(hex=81h)																	data14(hex=81h)
data15(hex=81h)																	data16(hex=81h)
data17(hex=80h)																	data18(hex=01h)
data19(hex=40h)																	data20(hex=02h)
data21(hex=24h)																	data22(hex=24h)
data23(hex=23h)																	data24(hex=C4h)
data25(hex=21h)																	data26(hex=84h)
data27(hex=10h)																	data28(hex=08h)
data29(hex=08h)																	data30(hex=10h)
data31(hex=07h)																	data32(hex=E0h)

Example of a 16x16 user defined bitmap



2.1.13 Display User Bitmapped Character

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

cmd : 44hex, Dascii

group : selects the appropriate bitmap format:

00hex selects the 8x8 bitmap format

01hex selects the 16x16 bitmap format

02hex selects the 32x32 bitmap format

char# : which user defined character number to display from the selected group.

0 to 63 (00hex to 3Fhex), of 8x8 format when group = 00hex

0 to 15 (00hex to 0Fhex), of 16x16 format when group = 01hex

0 to 7 (00hex to 07hex), of 32x32 format when group = 02hex

x(msb) : msb byte of the horizontal display position of the character. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : lsb byte of the horizontal display position of the character. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : msb byte of the vertical display position of the character. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : lsb byte of the vertical display position of the character. Refer to Section 3.1 for ranges for y for different cores.

color : character colour value:

64 colours to choose from for 64-Colour cores

Black = 00hex, 0dec

White = 3Fhex, 63dec, 00111111bin

256 colours to choose from for 256-Colour cores

Black = 00hex, 0dec

White = FFhex, 255dec, 11111111bin

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.



4D Systems

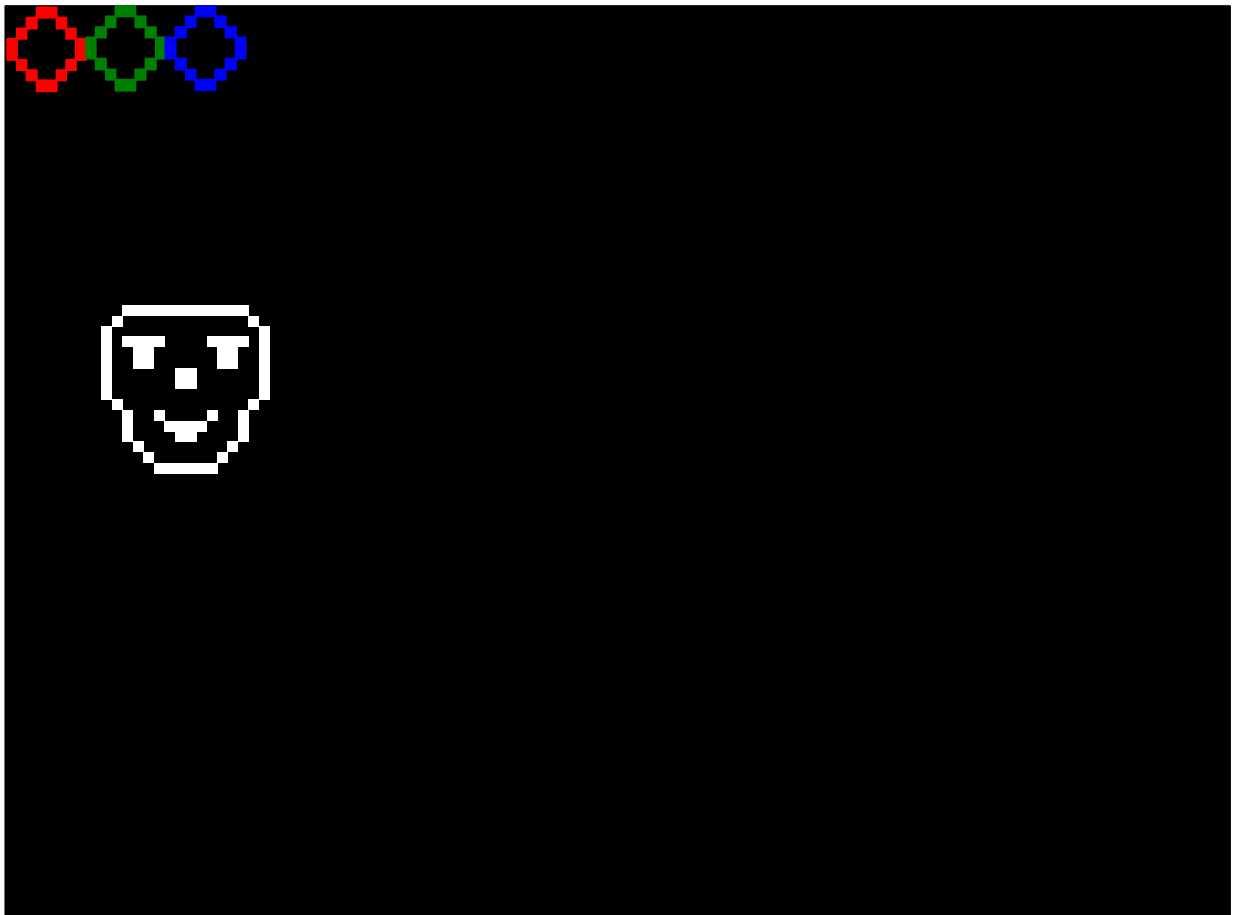
MicroVGA

Example 1: 44hex, 00hex, 01hex , 00hex, 00hex, 00hex, 00hex, 07hex
Display 8x8 bitmap character number 1 at x = 0, y = 0, colour = red

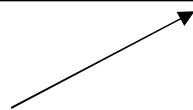
Example 2: 44hex, 00hex, 01hex, 00hex, 08hex, 00hex, 00hex, 38hex
Display 8x8 bitmap character number 1 at x = 8, y = 0, colour = green

Example 3: 44hex , 00hex, 01hex, 00hex, 10hex, 00hex, 00hex, C0hex
Display 8x8 bitmap character number 1 at x = 16, y = 0, colour = blue

Example 4: 44hex , 01hex, 03hex, 00hex, 10hex, 00hex, 20hex, FFhex
Display 16x16 bitmap character number 3 at x = 16, y = 32, colour = white



Screen





2.1.14 Display Image

Syntax : **cmd**, **x(msb)**, **x(lsb)**, **y(msb)**, **y(lsb)**, **width(msb)**, **width(lsb)**, **height(msb)**, **height(lsb)**, **data1**, .. **dataN**

cmd : 49hex, lascii

x(msb) : msb byte of the top left horizontal display position of the Image. Refer to Section 3.1 for ranges for x for different cores.

x(lsb) : lsb byte of the top left horizontal display position of the Image. Refer to Section 3.1 for ranges for x for different cores.

y(msb) : msb byte of the top left vertical display position of the Image. Refer to Section 3.1 for ranges for y for different cores.

y(lsb) : lsb byte of the top left vertical display position of the Image. Refer to Section 3.1 for ranges for y for different cores.

width(msb) : msb byte of Width of the Image.

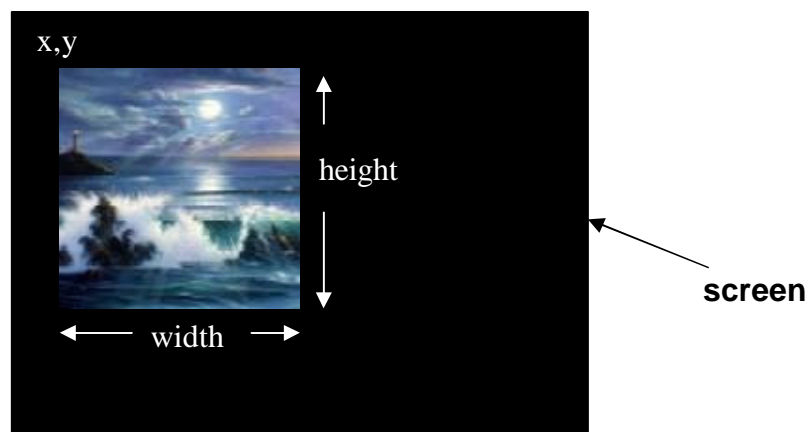
width(lsb) : lsb byte of Width of the Image.

height(msb) : msb byte of Height of the Image.

height(lsb) : lsb byte of the Height of the Image.

data1, .. **dataN** : Image data. These are 1 byte colour values corresponding to the image pixel information. The total size of the image data is determined by the width and the height. For example, if an image size of 128 x 64 (width=128, height=64) is to be displayed then a total of $128 \times 64 = 8,192$ bytes of data must be sent for data1 ,..., dataN.

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 feature is more effective than using the "Put Pixel" command where there are no overheads in specifying the **x, y** location of each pixel.





2.1.15 Paint Area

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

cmd : `70hex, pascii`

x1(msb) : msb byte of the top left horizontal start position of the Area. Refer to Section 3.1 for ranges for x for different cores.

x1(lsb) : lsb byte of the top left horizontal start position of the Area. Refer to Section 3.1 for ranges for x for different cores.

y1(msb) : msb byte of the top left vertical start position of the Area. Refer to Section 3.1 for ranges for y for different cores.

y1(lsb) : lsb byte of the top left vertical start position of the Area. Refer to Section 3.1 for ranges for y for different cores.

x2(msb) : msb byte of the bottom right horizontal end position of the Area. Refer to Section 3.1 for ranges for x for different cores.

x2(lsb) : lsb byte of the bottom right horizontal end position of the Area. Refer to Section 3.1 for ranges for x for different cores.

y2(msb) : msb byte of the bottom right vertical end position of the Area. Refer to Section 3.1 for ranges for y for different cores.

y2(lsb) : lsb byte of the bottom right vertical end position of the Area. Refer to Section 3.1 for ranges for y for different cores.

color : Area colour value:
64 colours to choose from for 64-Colour cores
Black = `00hex, 0dec`
White = `3Fhex, 63dec, 00111111bin`
256 colours to choose from for 256-Colour cores
Black = `00hex, 0dec`
White = `FFhex, 255dec, 11111111bin`

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, 10hex, 00hex, 20hex, 00hex, 64hex, 00hex, 64hex, 0Chex`

Paint the area GREEN that has its top left corner at x1=16, y1=32 and its bottom RIGHT corner at x2=100, y2=100.



2.1.16 Place Button Icon

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

cmd : 62hex, bascii

state : Specifies whether the displayed Button is drawn as UP (pressed) or DOWN (not pressed).
00hex = Button Down (pressed).
01hex = Button Up (not pressed).

x1(msb) : msb byte of the top left horizontal start position of the Button.

x1(lsb) : lsb byte of the top left horizontal start position of the Button.

y1(msb) : msb byte of the top left vertical start position of the Button.

y1(lsb) : lsb byte of the top left vertical start position of the Button.

x2(msb) : msb byte of the bottom right horizontal end position of the Button.

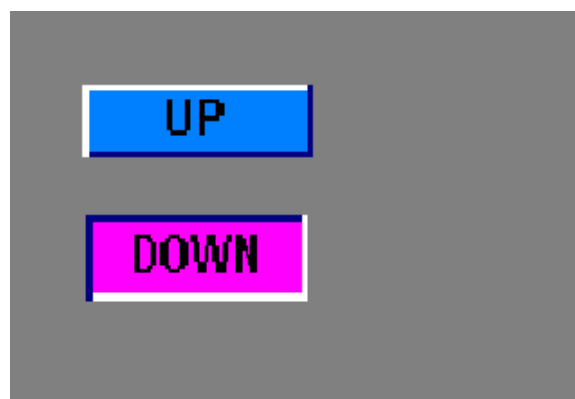
x2(lsb) : lsb byte of the bottom right horizontal end position of the Button.

y2(msb) : msb byte of the bottom right vertical end position of the Button.

y2(lsb) : lsb byte of the bottom right vertical end position of the Button.

color : Button colour value:
64 colours (00hex to 3Fhex) to choose from for 64-Colour cores.
256 colours (00hex to FFhex) to choose from for 256-Colour cores.

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.





2.1.17 Plot Graph

Syntax : **cmd**, **x(msb)**, **x(lsb)**, **y(msb)**, **y(lsb)**, **color**, **size(msb)**, **size(lsb)**, **delay**, **erase**, **erase_color**, **erase_height**, **data1**, .. **dataN**

cmd : **47**hex, **G**ascii

x(msb) : msb byte of the bottom left horizontal start position of the Graph.

x(lsb) : lsb byte of the bottom left horizontal start position of the Graph.

y(msb) : msb byte of the bottom left vertical start position of the Graph.

y(lsb) : lsb byte of the bottom left vertical start position of the Graph.

color : Graph colour value:
64 colours (**00**hex to **3F**hex) to choose from for 64-Colour cores.
256 colours (**00**hex to **FF**hex) to choose from for 256-Colour cores.

size(msb) : msb byte of the number of sample points that define **data1**, .. **dataN**

size(lsb) : lsb byte of the number of sample points that define **data1**, .. **dataN**

delay : 0 to 255 milliseconds (**00**hex – **FF**hex). This value allows a delay before the next sample point is plotted. A useful feature to have when plotting an ECG graph.

erase : This feature allows the current graph to overwrite a previously plotted graph or to be displayed on top of it.
00hex = Do not erase previous graph.
01hex = Erase previous graph with **erase_color**.

erase_color :
Don't care when **erase** = **00**hex.
Erase colour when **erase** = **01**hex
(**00**hex to **3F**hex) 64 colours to choose from for 64-Colour cores.
(**00**hex to **FF**hex) 256 to choose from for 256-Colour cores.

erase_height : When a previous graph is to be overwritten the **erase** byte should be set to **01**hex. As the new graph is plotted a point at a time the previous plot gets erased with the colour specified in **erase_color** byte. The **erase_height** value should therefore be set so that the highest point of the previous plot can be overwritten.



4D Systems

MicroVGA

data1, .. dataN : These are the sample points (1 byte each) for the graph. These sample points are referenced with respect to **y**. The total number of sample points must match the exact 2 byte value specified by **size**. Maximum number of sample points must not exceed 512 bytes.

Description : This command will allow a graph of any shape or form such as an E.C.G. to be plotted. The bottom left corner is specified by **x** and **y**. If a slow plotting graph effect is needed then an appropriate value can be set in the **delay** byte (maximum of 256 milliseconds). Each sample point **data1, .. dataN** is relatively referenced to **y** which means a sample point of value 0 will be plotted at **y** and a sample point of value 10 will be plotted at location **y+10**.



2.1.18 Block Copy (Bitmap-Image Copy)

Syntax : `cmd, x_source(msb), x_source(lsb), y_source(msb), y_source(lsb), x_dest(msb), x_dest(lsb), y_dest(msb), y_dest(lsb), width(msb), width(lsb), height(msb), height(lsb), source_page, dest_page`

cmd : `63hex, c ascii`

x_source(msb) : msb byte of the top left horizontal start position of block to be copied (source x).

x_source(lsb) : lsb byte of the top left horizontal start position of block to be copied (source x).

y_source(msb) : msb byte of the top left vertical start position of block to be copied (source y).

y_source(lsb) : lsb byte of the top left vertical start position of block to be copied (source y).

x_dest(msb) : msb byte of the top left horizontal start position of where the copied block is to be pasted (destination x).

x_dest(lsb) : lsb byte of the top left horizontal start position of where the copied block is to be pasted (destination x).

y_dest(msb) : msb byte of the top left vertical start position of where the copied block is to be pasted (destination y).

y_dest(lsb) : lsb byte of the top left vertical start position of where the copied block is to be pasted (destination y).

width(msb) : msb byte of Width of block to be copied.

width(lsb) : lsb byte of Width of block to be copied.

height(msb) : msb byte of Height of block to be copied.

height(lsb) : lsb byte of the Height of block to be copied.

source_page : Source Page, block copied from this page.

`00hex` selects page 0

`01hex` selects page 1

`02hex` selects page 2

`03hex` selects page 3

Note: All pages are not available for all cores, see description below.



dest_page : Destination Page, copied block to be pasted to this page.
00hex selects page 0
01hex selects page 1
02hex selects page 2
03hex selects page 3

Note: All pages are not available for all cores, see description below.

Description : This command copies a block of area of specified size from a source page to a destination page (both can be the same page or different). The start location of the block to be copied from, the source page, is represented by **x_source, y_source** (top left corner) and the start location of where the block is to be pasted to, the destination page, is represented by **x_dest, y_dest** (top left corner). The size of the block is specified by the **width** and the **height** parameters.

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. It will let you do cool things like store bitmaps in off-screen memory and quickly copy them over (useful for big sprites).

The μ VGA hiRes cores have 512K bytes of video or graphics memory and depending on the resolution of the core this memory is split into a certain number of pages. Due to the design of the internal architecture, not all pages are available for all cores. The following shows each core and available pages of memory.

<u>Core</u>	<u>Available Pages</u>
μ VGA-320 (320 x 220)	page 0, page 1, page 2, page 3
μ VGA-420 (420 x 350)	page 0, page 1
μ VGA-620 (620 x 420)	page 0, page 1
μ VGA-640 (640 x 480)	page 0
μ VGA-720 (720 x 500)	page 0

Each page is made up of banks of 64Kbyte boundaries. In certain cores this leads to a heap of memory that is left over and not used when displaying the page. The user can utilise these chunks of memory as temporary scratch pads to copy from and paste blocks of data to. The following shows each core and available size of scratch pad memory.

<u>Core</u>	<u>Available Scratch Pad Memory</u>
μ VGA-320 (320 x 220)	none available.
μ VGA-420 (420 x 350)	from (x=0, y=350) to (x=419, y=623) in each page. max. block width = 420, max. block height = 274.
μ VGA-620 (620 x 420)	none available.
μ VGA-640 (640 x 480)	from(x=0, y=480) to (x=639, y=815) only page 0. max. block width = 640, max. block height = 336.
μ VGA-720 (720 x 500)	from(x=0, y=500) to (x=719, y=727) only page 0. max. block width = 720, max. block height = 228.



2.1.19 Video Memory (Graphics RAM) Read, Write, Display Settings

Syntax : cmd, mode, page

cmd : 76hex, v ascii

mode : Selects the following modes of video memory operations:

- 00hex :** Display page select. Selects the page that is currently to be displayed on the screen. Default = page 0
- 01hex :** Read page select. Selects the page that all read operations are to be performed from. Default = page 0
- 02hex :** Write page select. Selects the page that all write operations are to be performed on. Default = page 0

page : Page select.

- 00hex :** all operations for the selected **mode** will be from page 0
- 01hex :** all operations for the selected **mode** will be from page 1
- 02hex :** all operations for the selected **mode** will be from page 2
- 03hex :** all operations for the selected **mode** will be from page 3

Note: All pages are not available for all cores, see description below.

Description : This command allows complete control over the reading, writing and displaying of the graphics RAM (video memory) pages. For example, while page 0 is displayed, page 1 can be written to in the background and then the display can be quickly changed to page 1. Many effects such as smooth animations and scrolling will utilise this command.

Notes:

- When a Display page is selected, the μ VGA will internally wait for the next Start of Frame before switching over the page. This provides a smooth change over. The μ VGA will then reply back to indicate this.
- The Read or Write settings of the video memory will not have any effect on the “**Block Copy**” command (63hex, c ascii). The “**Block Copy**” command can be used freely to copy from and paste to any page without having to set the read or the write modes.
- Not all pages are available in all cores. The following shows each core and available pages of memory:

<u>Core</u>	<u>Available Pages</u>
μ VGA-320 (320 x 220)	page 0, page 1, page 2, page 3
μ VGA-420 (420 x 350)	page 0, page 1
μ VGA-620 (620 x 420)	page 0, page 1
μ VGA-640 (640 x 480)	page 0
μ VGA-720 (720 x 500)	page 0



2.1.20 Sync Lock

Syntax : cmd

cmd : 53hex, Sascii

Description : This command allows the host to lock onto the start of the next video frame.

This is a very useful command when moving/animating objects on the screen where the object needs to be displayed and then erased so it can be redisplayed at a different location to give the effect of movement. Unless the host locks onto the start of frame, undesired blinking of the displayed object will occur. When the host sends this command it should wait for an 'ACK' from the μ VGA and once the ACK is received the host can then start issuing display commands. Use this command for dynamically displayed objects such as a bouncing ball, etc. For static objects that don't move or that do not need continuous display updates, this command is not necessary. Example of a moving object is given below:

```
while() {  
    send_cmd_uVGA('S'); // send the Sync Lock command  
    wait_uVGA_ACK(); //  $\mu$ VGA will respond at start of frame  
    send_cmd_uVGA('D', 1, x, y, BLACK); // erase object at old loc.  
    send_cmd_uVGA('D', 1, ++x, ++y, RED); // display at new loc.  
} // loop around while loop
```

Note: A word of caution: If the serial link is slow, that is, less than 9600baud, the response from the μ VGA will be too slow for this command to be effective. Try and make the serial link as fast as possible (9600 baud or above).



2.1.21 Version Request

Syntax : cmd

cmd : 56hex, Vascii

Description : This command returns a 1 byte value from the μ VGA. This is useful if the μ VGA core is updated and the host can configure it-self appropriately to utilise the varying resolutions and functions of different Cores.

Example: 56 hex (host sends version request)

μ VGA reply : 32hex = μ VGA-320 (320x220 core present)
42hex = μ VGA-420 (420x350 core present)
62hex = μ VGA-620 (620x420 core present)
64hex = μ VGA-640 (640x480 core present)
72hex = μ VGA-720 (720x500 core present)



2.2 μ VGA Serial Interface (TTL)

The μ VGA needs to be connected via a serial link to a host system. The host uses this serial link to send commands to the μ VGA 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 is at TTL levels (0 – 5V) and the logic levels are “high” = 1 = 5V, “low” = 0 = 0V.**

Auto Baud Detect:

As previously mentioned, the μ VGA core has an auto-baud detect function which can operate from **300 baud to 1,000k 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 VGA 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 VGA 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 VGA and the operation is completed, the μ VGA 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 VGA anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the μ VGA has to perform. If the μ VGA 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 VGA 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 flags.



2.3 μ VGA USB Interface

Using a standard USB interface on a computer, you can control the μ VGA-USB device by sending commands to it using a program to send hex data to the μ VGA-USB. The USB device on the μ VGA-USB, simply captures the USB data and converts it into serial TTL data for the μ VGA. USB Drivers are available from:

the 4D website: www.4dsystems.com.au

or from the Dontronics website: <http://www.dontronics.com/micro-usb.html>

2.4 Demonstration Mode

The μ VGA core can be put into a demo mode by pulling the “Demo” signal (pin12) to logic 0 (GND) and powering up the unit. This will display some of the core capabilities and give the user an idea of the graphics functionality and it’s a quick method of operating the unit without having to connect to a host controller. The demo lasts several minutes.

To put the μ VGA back into the normal mode of operation this pin (Demo) must be pulled high with a resistor to Vcc 5V (4.7K to 10K nominal), otherwise every time the unit is powered up it will go thru the demo.

2.5 Base Board Systems

The μ VGA core can be purchased as a single ‘drop-in’ component ready to be plugged into your host application design, or if a ready-to-go solution is required, a “base-board” system can be purchased which contains all the necessary interface circuitry with which to connect directly to a monitor via a standard 15pin VGA connector. There are 2 base-board systems available:

- ✍ microVGA Universal Base μ VGA-UB1(64 colours) and the μ VGA-UB2 (256 colours) which contains a base PCB with 15pin VGA connector, RGB DAC circuitry, 20pin socket for the core and a 4pin user header (5VDC, Rx, Tx, GND). All that is required is a power supply input of 5VDC and a host microcontroller board with which to interface the system via the serial connection (Rx, Tx on the 4pin header). See Figures 6 and 7.
- ✍ microVGA USB Board the μ VGA-USB (256 colours only) which contains a base PCB with 15pin VGA connector, RGB DAC circuitry, 20pin socket for the core, USB connector (See Figures 8 and 9). The device is powered from the USB connector on the computer. The USB device on the board receives the USB data and converts it to serial TTL level signals for the microVGA.



4D Systems

MicroVGA

Figure 6. microVGA Universal Base μ VGA-UB2 (256 colours)

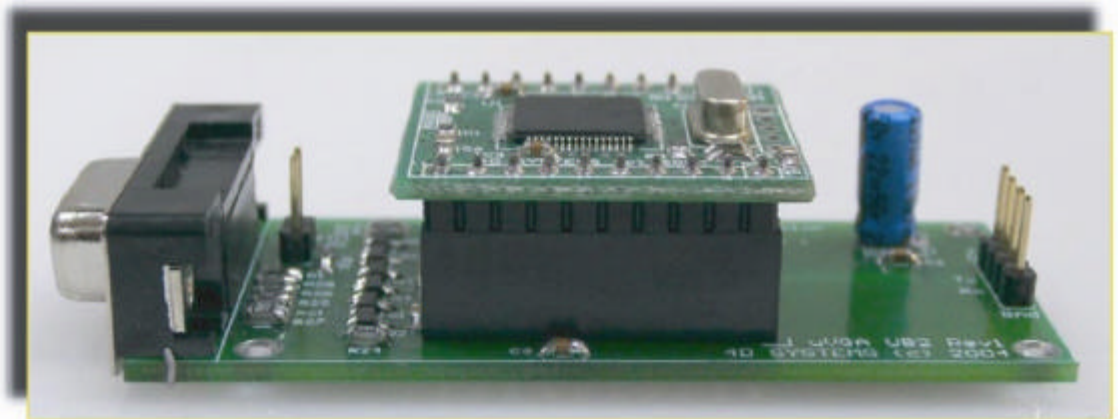
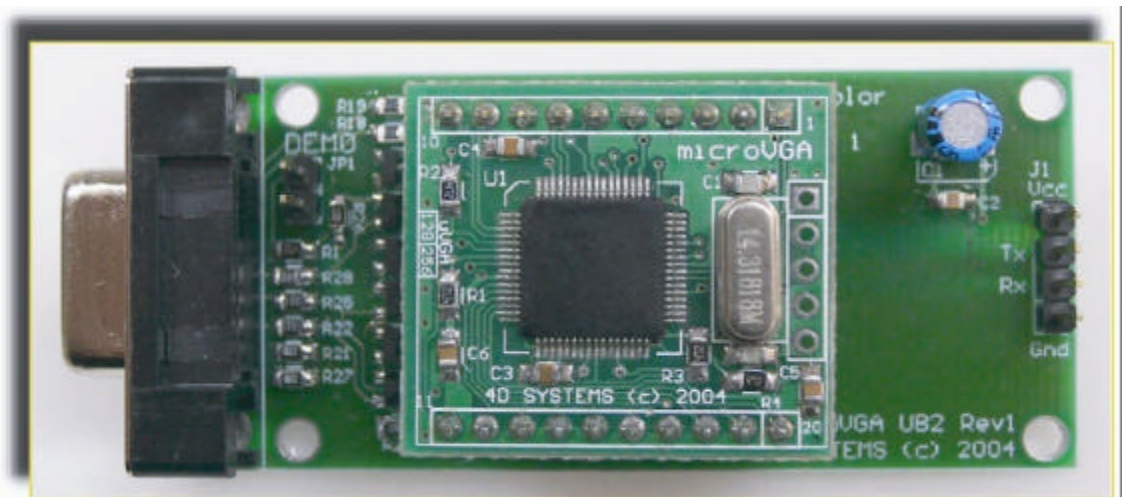


Figure 7. μ VGA-UB2 Base + μ VGA-xxx Core. Note the orientation of the Core pins. The serial inputs are true TTL signals, and not RS232 signals.





4D Systems

MicroVGA

Figure 8. μ VGA-USB Base + μ VGA-xxx Core

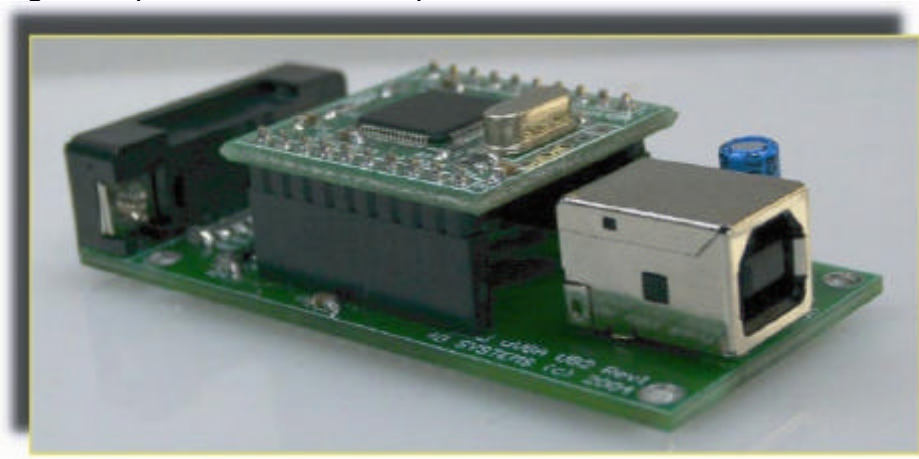
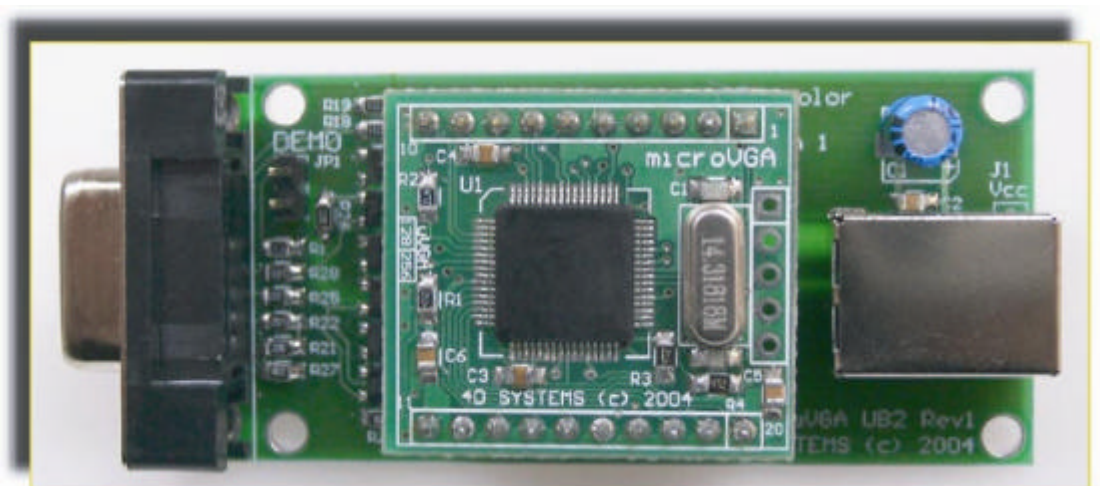


Figure 9. μ VGA-UB2 Base + μ VGA-xxx Core. Note the orientation of the Core pins.





3. Specifications

The μ VGA core has the following electrical specifications which must be adhered to at all times to prevent damage to the device. The μ VGA core module footprint is 25mm x 28mm.

Symbol	Characteristic	Min	Typ	Max	Units
Vdd	Supply voltage	4.5V	5V	5.5V	V
I	Current	280mA	300mA	320mA	mA
Deg C	Operating temp	0	30	70	C
Tpu	Power-up delay	800		1000	mS

3.1 Core types and resolution

Core type	X resolution	Y resolution
μ VGA-320	0 to 319 (320 total)	0 to 219 (220 total)
μ VGA-420	0 to 419 (420 total)	0 to 349 (350 total)
μ VGA-620	0 to 619 (620 total)	0 to 419 (420 total)
μ VGA-640	0 to 639 (640 total)	0 to 479 (480 total)
μ VGA-720	0 to 719 (720 total)	0 to 499 (500 total)

3.2 μ VGA hiRes Core pin-outs

Pin outs for 64 colour cores

Pin 1 - Green 0	Pin 20 - Green 1
Pin 2 - Red 1	Pin 19 - Blue 0
Pin 3 - Red 0	Pin 18 - Blue 1
Pin 4 - Blank Red	Pin 17 - Rx Data
Pin 5 - Ground	Pin 16 - Tx Data
Pin 6 - Blank Blue	Pin 15 - Vcc +5V
Pin 7 - Blank green	Pin 14 - Ground
Pin 8 - Ground	Pin 13 - Test
Pin 9 - H sync	Pin 12 - Demo
Pin 10- V sync	Pin 11 - Enable

Pin outs for 256 colour cores

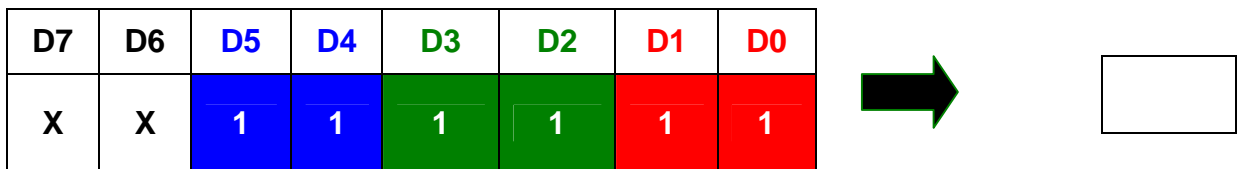
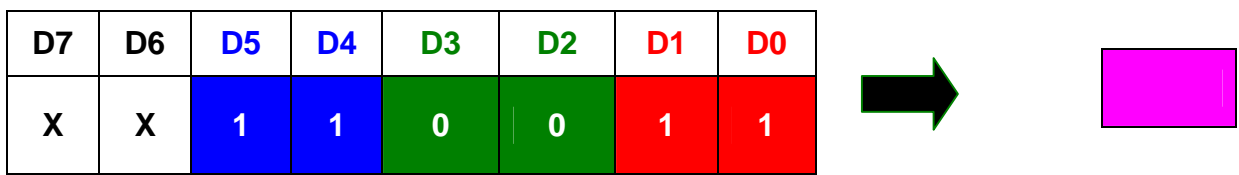
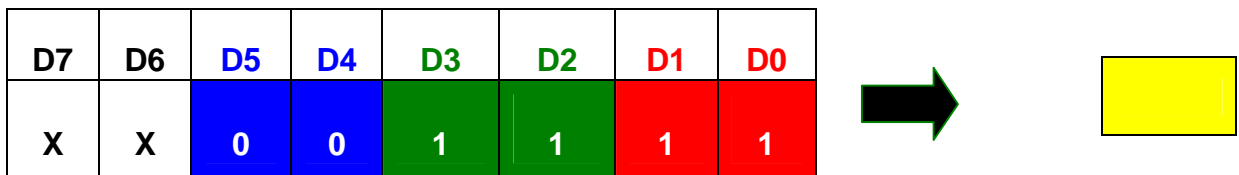
Pin 1 - Red 2	Pin 20 - Green 0
Pin 2 - Red 1	Pin 19 - Green 1
Pin 3 - Red 0	Pin 18 - Green 2
Pin 4 - Blank Red	Pin 17 - Rx Data
Pin 5 - Ground	Pin 16 - Tx Data
Pin 6 - Blank Blue	Pin 15 - Vcc +5V
Pin 7 - Blank green	Pin 14 - Ground
Pin 8 - Ground	Pin 13 - Blue 1
Pin 9 - H sync	Pin 12 - Demo
Pin 10- V sync	Pin 11 - Blue 2



3.3 64 Colour Bitmap Organisation

The μ VGA 64 colour byte is organised as 2 bits for Red (D0, D1), 2 bits for Green (D2, D3) and 2 bits for Blue (D4, D5). This will give a combination of $4 \times 4 \times 4 = 64$ colours. The upper 2 bits (D6, D7) are not used. 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.

D7	D6	D5	D4	D3	D2	D1	D0
X	X	BLUE1	BLUE0	GREEN1	GREEN0	RED1	RED0

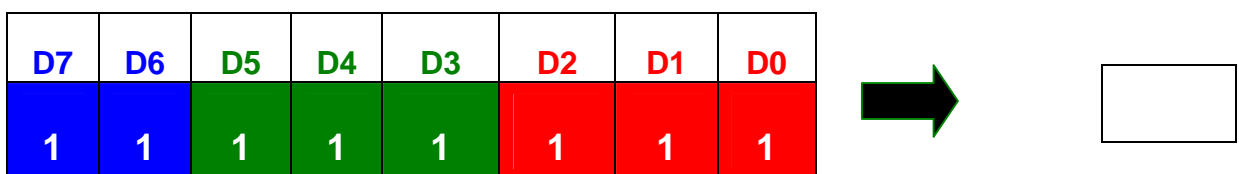
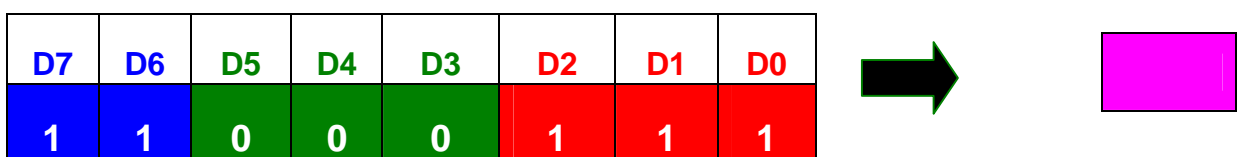
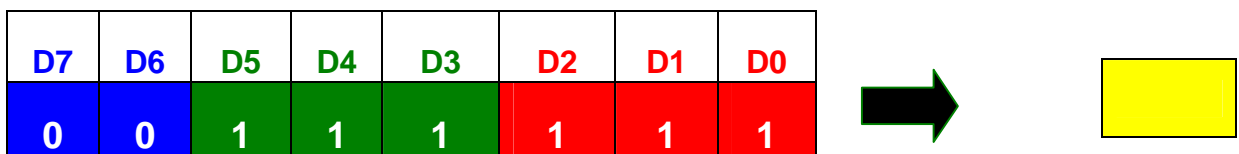




3.4 256 Colour Bitmap Organisation

The μ VGA 256 colour byte is organised as 3 bits for Red (D0, D1, D2), 3 bits for Green (D3, D4, D5) and 2 bits for Blue (D6, D7). This will give a combination of $8 \times 8 \times 4 = 256$ colours. The upper 2 bits (D6, D7) are not used. 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.

D7	D6	D5	D4	D3	D2	D1	D0
BLUE2	BLUE1	GREEN2	GREEN1	GREEN0	RED2	RED1	RED0



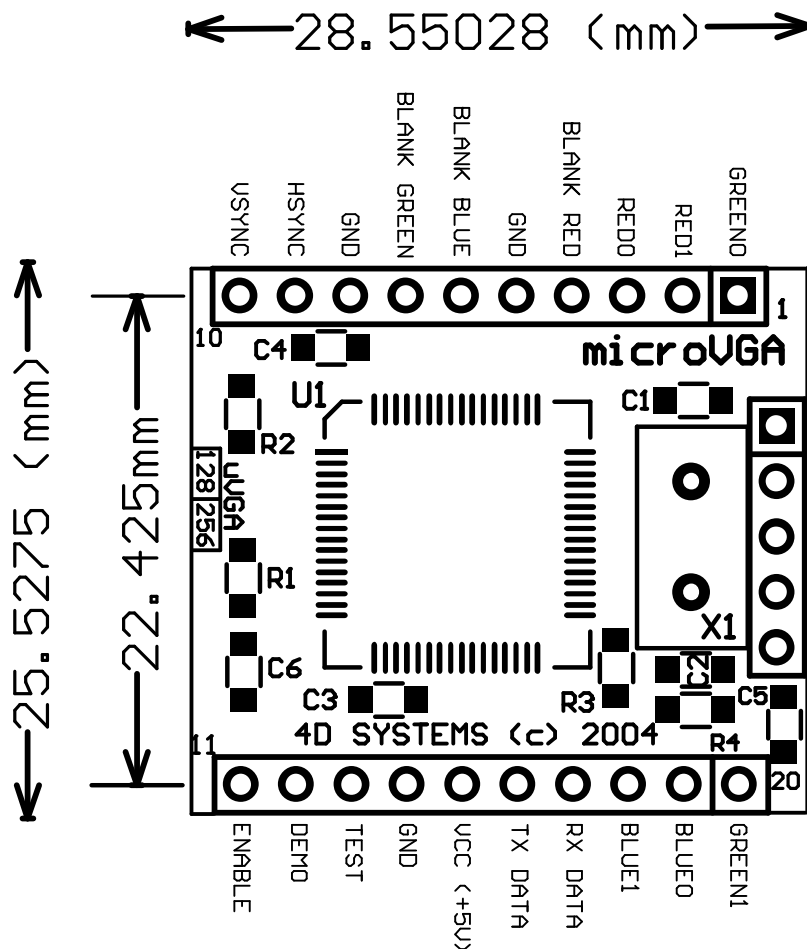


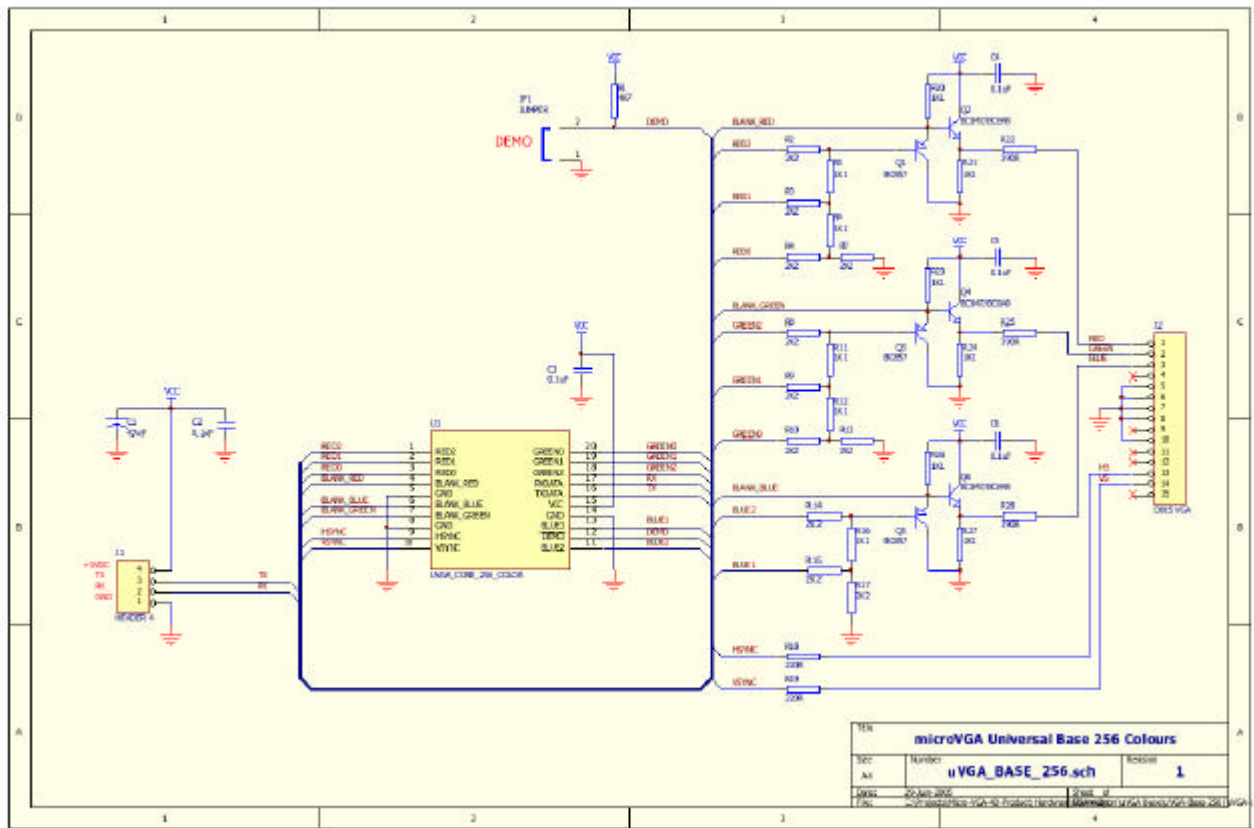
3.5 Power-Up Reset

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

- ✎ Allow 800ms to 1000ms after power-up for μ VGA to settle. Do not attempt to communicate with the μ VGA during this period. The μ VGA 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 VGA can lock onto the hosts serial baud rate. The μ VGA will respond with an 'ACK' (06h). See section 2.3
- ✎ The μ VGA is now ready to accept screen function commands from the host.

3.6 Mechanical Details





uVGA 256 Colour Circuit Diagram (uVGA-UB2)