4D SYSTEMS

Modules On The Go Series

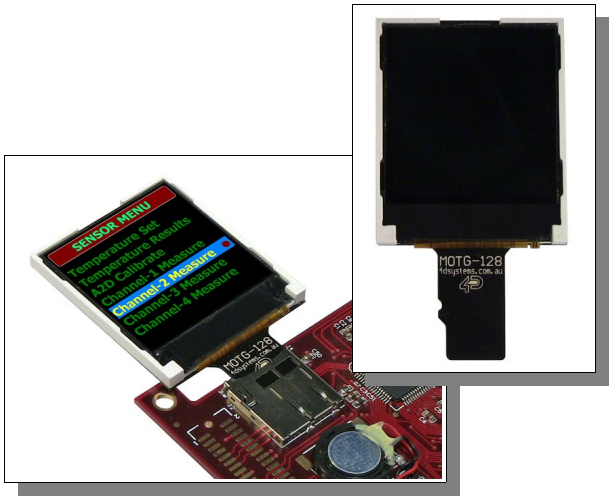
MOTG-128

Pluggable µSD LCD Display Module

Data Sheet

Document Date: 21st November 2011
Document Revision: 2.0

Description



MOTG-128 is a small pluggable module ideally suited as a diagnostic display for embedded systems. It incorporates a 1.44" 128x128 resolution full colour LCD-TFT and has a wide range of uses in educational, experimental as well as during any development phase where a quick display solution is required. It comes in a micro-SD memory card format which is designed to interface to any embedded platform that has an on board micro-SD (or SD) card adaptor.

Powerful graphics, text, image, animation and countless more features are built inside the MOTG-128. The display module utilises the standard SPI signals and acts as a slave device to the host micro. It offers a simple yet effective command set to any host platform that can communicate via a SPI port and all screen related functions are sent using a simple protocol.

MOTG-128 device simply plugs into a standard µSD/SD socket for quick assembly and maintenance, no need to design for specialised connectors or spend time waiting for them to arrive from exotic suppliers, just add a standard µSD socket and off you go. Don't need the device permanently, or want to make it optional? Simple, just unplug. Need some way of debugging your latest brainchild without top end equipment? Just plug a MOTG-128 into its µSD socket and use it as an outboard diagnostic display.

The MOTG-128 is a novel concept from 4D Systems that belong to a broader range of **Modules-On-The-Go** series. Some of the other modules on offer are:

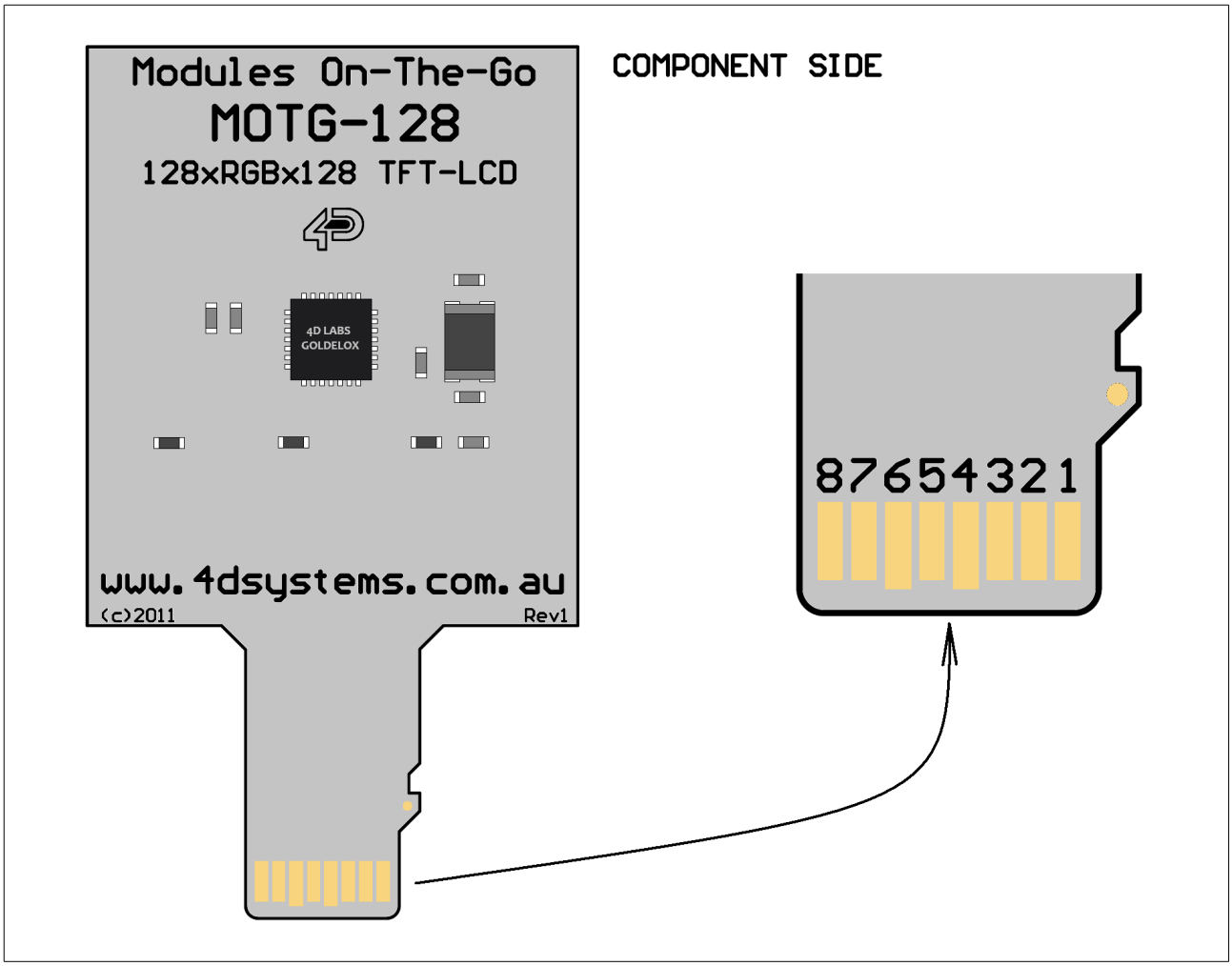
- **MOTG-96:** Pluggable 0.96" 96x64 OLED display module in a micro-SD card format.
- **MOTG-GPS:** Pluggable GPS module with a tiny built in ceramic antenna in a micro-SD card format.

Features

- Low-cost pluggable LCD display module.
- 128xRGBx128 resolution, 65K true to life colours, LCD-TFT screen.
- 1.44” diagonal size with an Active Display Area: 25.5mm x 26.5mm.
- LED back lighting with greater than 150° very wide viewing angle.
- Easy micro-SD card form factor that will plug into any embedded host platform that has a micro-SD or SD card adaptor.
- SPI interface to the host via the micro-SD or SD adaptor.
- Comprehensive set of built in high level graphics functions and algorithms that can draw lines, circles, text, and much more.
- RoHS Compliant.

Table of Contents

1. Pin Configuration and Description



Pin	Symbol	I/O	Description
1	NC	--	Not Connected.
2	CS	I	MOTG SPI Chip Select. The host asserts this signal LOW when accessing the display module.
3	SDI	I	MOTG SPI Data In. This pin connects to the Data Out of the host SPI.
4	3.3V	P	Voltage supply input. Range is 3.0V to 3.6V, nominal 3.3V.
5	SCK	I	MOTG SPI Clock In. This pin connects to the host SPI Clock output.
6	GND	P	Ground.
7	SDO	O	MOTG SPI Data Out. This pin connects to the host SPI Data input.
8	RESET	I	MOTG Reset Input (active LOW). This pin is only used by the Programming module for updating the MOTG with PmmC files. Not used during normal operation.

Legend: I = Input, O = Output, P = Power

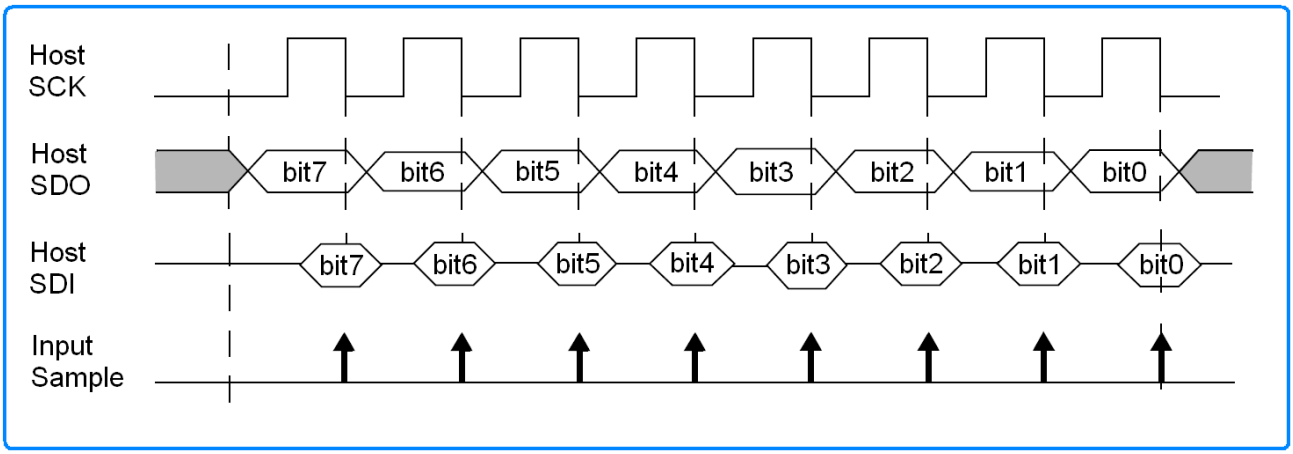
2. micro-SD SPI Hardware Interface

The MOTG-128 is designed to plug into standard micro-SD card sockets employed in most embedded platforms. It can also plug into standard SD slots with the aid of a micro-SD to SD adaptor. It is important to note that the communication interface is via standard SPI signalling and most embedded platforms employ the SPI mode for their on board memory card sockets. The following timing diagrams provide detailed information about the required SPI signalling.

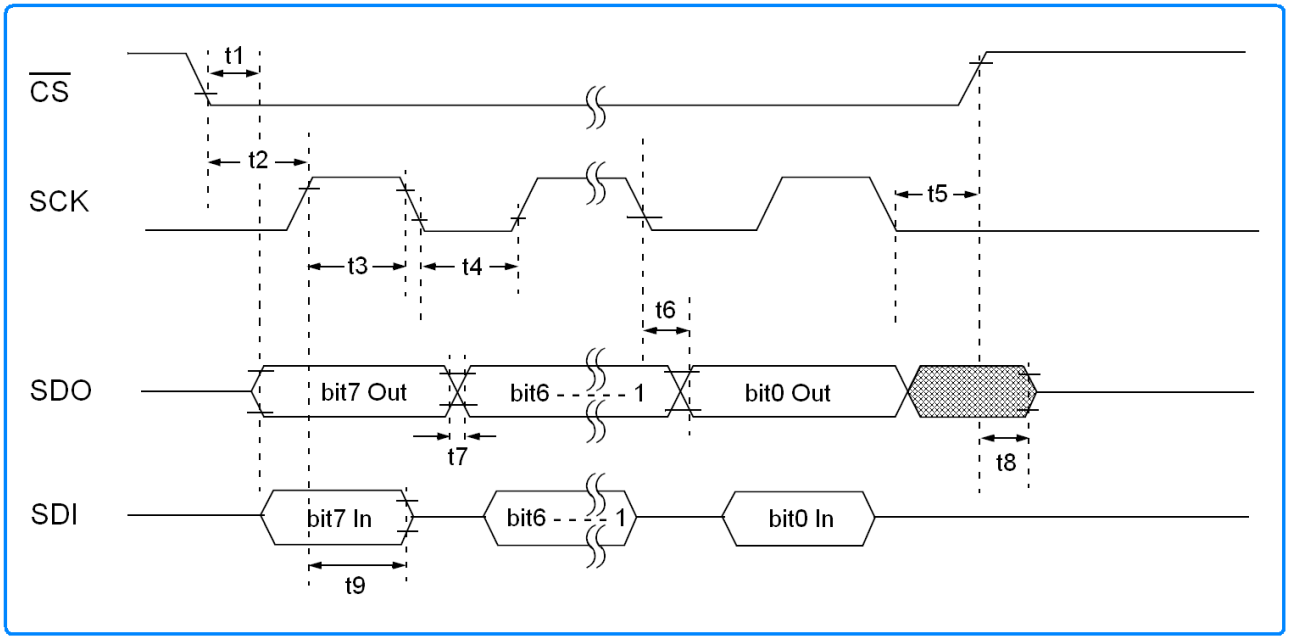
☞ **Note1:** MOTG-128 supports a maximum clock rate of 12Mhz.

☞ **Note2:** MOTG-128 will not work with external or built in PC card readers. These devices do not utilise the SPI signalling as required by the MOTG devices.

2.1 Generic Host Master SPI Timing



2.2 MOTG-128 Slave SPI Timing Diagram



Item	Symbol	Min.	Typ.	Max.	Unit
SDO Data Output Valid after CS↓ Edge	t1	--	--	50	ns
CS↓ to SCK↑ Input	t2	100	--	--	ns
SCK Input High Period	t3	40	--	--	ns
SCK Input Low Period	t4	40	--	--	ns
CS↑ after SCK Edge	t5	190	--	--	ns
SDO Data Output Valid after SCK Edge	t6	--	--	50	ns
SDO Data Output Rise and Fall Period	t7	--	--	25	ns
CS↑ to SDO Output High-Impedance	t8	10	--	50	ns
Hold Time of SDI Data Input to SCK Edge	t9	100	--	--	ns

3. Software Interface

The MOTG-128 display module is a slave peripheral device and it provides bidirectional communications to a host controller via its SPI interface. All communications between the host and the device occur over this SPI interface. The protocol is simple and easy to implement.

☞ **Note:** The host must initialise its SPI port as the master and must meet the MOTG SPI specifications outlined in the previous section.

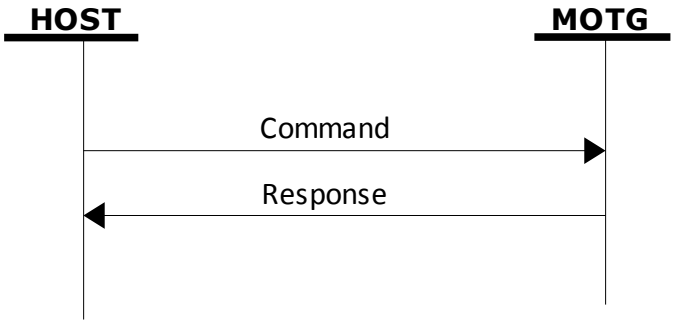
3.1 Command Protocol – Flow Control

The MOTG modules are slave devices and all communication and events must be initiated by the host. Each command is made up of a sequence of data bytes. When a command is sent to the device and the operation is completed, it will always return a response. For a command that has no specific response the device will send back a single acknowledge byte called the **ACK (06hex)**, in the case of success, or **NAK (15hex)**, in the case of failure.

Commands having specific responses may send back varying numbers of bytes, depending upon the command and response. It will take the device a certain amount of time to respond, depending on the command type and the operation that has to be performed. If the MOTG device receives a command that it does not understand it will reply back with a negative acknowledge called the NAK (15hex). Since a command is only identified by its position in the sequence of data bytes sending incorrect data can result in wildly incorrect operation.

4. Command Set

The command interface between the MOTG device and the host is via the serial interface. A handful of easy to learn commands provide complete access to all the available functions. The simplified command set also means that very low overheads are imposed on the host controller. Commands and responses can be either single bytes or many bytes. All commands return a response, either an acknowledge or data.



The command set is grouped into following sections:

- General Commands
- Graphics Commands
- Text Commands

Each Command set is described in detail in the following sections.

☞ **Separation characters such as commas ',' or spaces ' ' or brackets'(' ')' between bytes that are shown in the command/response syntax descriptors are purely for legibility purposes and must not be considered as part of any transmitted/received data unless specifically stated.**

4.1 General Commands

Summary of Commands in this section:

- Version-Device Info Request – **56hex**
- Replace Background Colour – **42hex**
- Clear Screen – **45hex**
- Display Control Functions – **59hex**

4.1.1 Version-Device Info Request - 56Hex

Command	**Cmd, Output**	
	cmd	**56**(hex) or **V**(ascii) : Command header byte
	Output	**00hex** : Outputs the version and device info to the SPI port only. **01hex** : Outputs the version and device info to the SPI port as well as to the screen.
Response	**device_type, hardware_rev, firmware_rev, horizontal_res, vertical_res**	
	device_type	This response indicates the device type. **00hex** = micro-OLED. **01hex** = micro-LCD. **02hex** = micro-VGA. **03hex** = micro-DRIVE. **04hex** = MOTG-OLED. **05hex** = MOTG-LCD.
	hardware_rev	This response indicates the device hardware version
	firmware_rev	This response indicates the device firmware version.
	horizontal_res	This response indicates the horizontal resolution of the display. **22hex** : 220 pixels **28hex** : 128 pixels **32hex** : 320 pixels **60hex** : 160 pixels **64hex** : 64 pixels **76hex** : 176 pixels **96hex** : 96 pixels
	vertical_res	This response indicates the vertical resolution of the display. See horizontal_res above for resolution options. **22hex** : 220 pixels **28hex** : 128 pixels **32hex** : 320 pixels **60hex** : 160 pixels **64hex** : 64 pixels **76hex** : 176 pixels **96hex** : 96 pixels
Description	This command requests all the necessary information from the device about its characteristics and capability.	

4.1.2 Replace Background Colour – 42hex

Command	**cmd, colour**(msb:lsb)	
	cmd	**42**(hex) or **B**(ascii) : Command header byte
	colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command changes 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	**Command Data:** 42hex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	

4.1.3 Clear Screen – 45hex

Command	**cmd**	
	cmd	**45**(hex) or **E**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command clears the entire screen using the current background colour	
Example	**Command Data:** 45hex (Clear the screen).	

4.1.4 Display Control Functions – 59hex

Command	**cmd, mode, value**	
	cmd	**59**(hex) or **Y**(ascii) : Command header byte
	mode	**00hex : NA** **01hex : Display ON/OFF** DISPLAY OFF : when value = 00hex DISPLAY ON : when value = 01hex **02hex : Contrast Adjust** CONTRAST RANGE : when value = 00hex to 0Fhex **03hex : Display PowerUp-Shutdown (low power mode)** DISPLAY SHUTDOWN : when value = 00hex DISPLAY POWERUP : when value = 01hex
	value	See mode description above.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command changes some of the display settings such as contrast and low power mode.	

4.2 Graphics Commands

Summary of Commands in this section:

- Add User Bitmap Character – **41hex**
- Draw Circle – **43hex**
- Draw User Bitmap Character – **44hex**
- Draw Triangle – **47hex**
- Draw Image-Icon – **49hex**
- Set Background colour **– 4Bhex**
- Draw Line – **4Chex**
- Draw Pixel – **50hex**
- Read Pixel – **52hex**
- Screen Copy-Paste – **63hex**
- Draw Polygon – **67hex**
- Replace colour **– 6Bhex**
- Set Pen Size – **70hex**
- Draw Rectangle – **72hex**

4.2.1 Add User Bitmap Character - 41hex

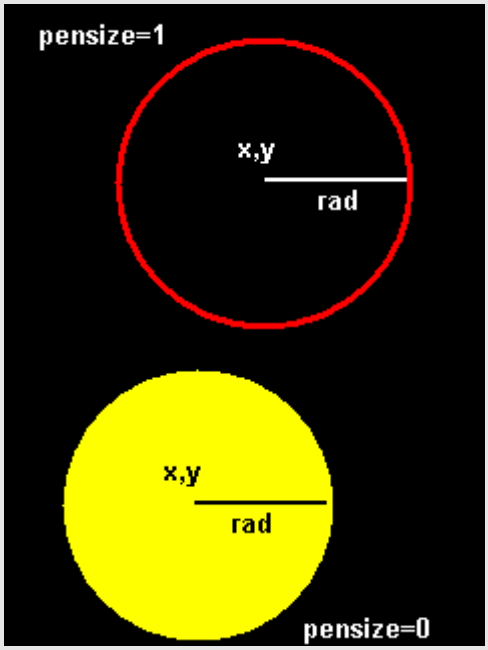
Command	**cmd, char_idx, data1, data2, .. , data8**	
	cmd	**41**(hex) or **A**(ascii) : Command header byte
	char_idx	Bitmap character index to add to memory. Range is 0 to 31 (**00**h to **1F**h), 32 characters of 8x8 format.
	data1..data8	8 data bytes that make up the composition of the bitmap character. The 8x8 bitmap composition is 1 byte wide (8 bits) by 8 bytes deep.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will add a user defined bitmap character into the internal memory.	

b7	b6	b5	b4	b3	b2	b1	b0		← Data Bits
			■	■					**data1 (18**hex**)**
		■			■				**data2 (24**hex**)**
	■					■			**data3 (42**hex**)**
■							■		**data4 (81**hex**)**
■							■		**data5 (81**hex**)**
	■					■			**data6 (42**hex**)**
		■			■				**data7 (24**hex**)**
			■	■					**data8 (18**hex**)**

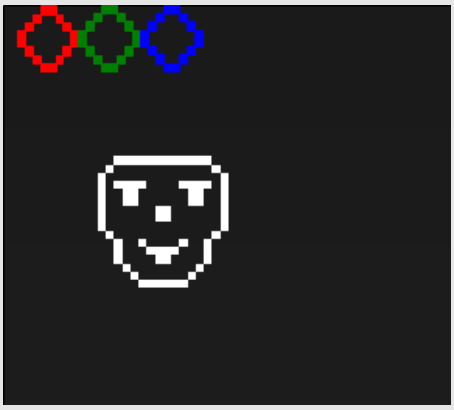
Example of 8x8 User defined bitmap

Example	**Command Data:** 41hex, 01hex, 18hex, 24hex, 42hex, 81hex, 81hex, 42hex, 24hex, 18hex This example adds and saves a user defined 8x8 bitmap as character index 1 into memory.

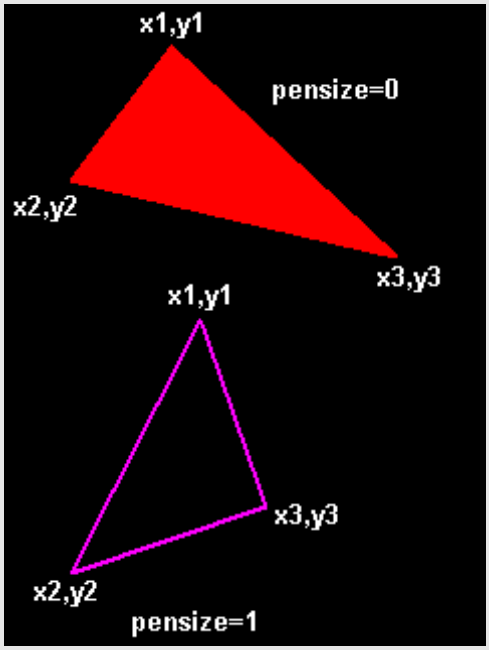
4.2.2 Draw Circle - 43hex

Command	**cmd, x, y, radius, colour**(msb:lsb)	
	cmd	**43**(hex) or **C**(ascii) : Command header byte
	x	Horizontal position of the circle centre.
	y	Vertical position of the circle centre.
	radius	Radius of the circle.
	colour	2 bytes define the circle colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured circle centred at **(x, y)** with a radius determined by the value set in the **'radius'** byte. The circle can be either solid or wire frame (empty) depending on the value of the Pen Size (see **Set Pen Size** command). when Pen Size = 0 : circle is solid when Pen Size = 1 : circle is wire frame 	
Example	**Command Data:** 43hex, 3Fhex, 3Fhex, 22hex, 00hex, 1Fhex Draws a RED circle (**001F**hex) centred at x = **63**dec (**3F**hex) and y = **63**dec (**3F**hex) with a radius of **34**dec (**22**hex).	

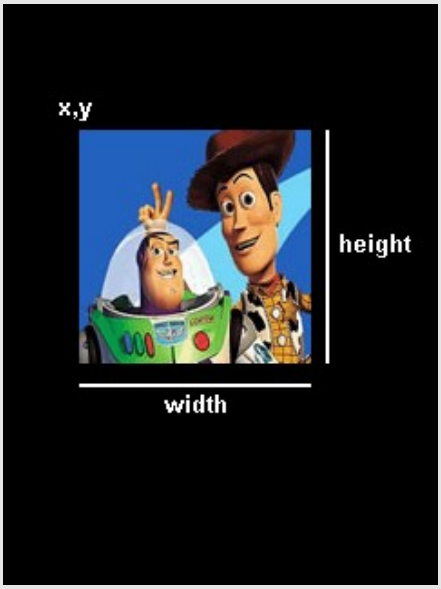
4.2.3 Draw User Bitmap Character - 44hex

Command	**cmd, char_idx, x, y, colour**(msb:lsb)	
	cmd	**44**(hex) or **D**(ascii) : Command header byte
	char_idx	Bitmap character index to draw from the previously added bitmap characters into memory. Range is 0 to 31 (**00**h to **1F**h), 32 characters of 8x8 format.
	x	Horizontal display position of the bitmap character.
	y	Vertical display position of the bitmap character.
	colour	2 bytes bitmap colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command draws the previously defined user bitmap character at location **(x, y)** on the screen. User defined bitmaps allow drawing & displaying unlimited graphic patterns quickly & effectively. 	
Examples	**Command Data:** 44hex, 01hex , 00hex, 00hex, F8hex, 00hex (Display 8x8 bitmap character index 1 at x = 0, y = 0, colour = RED). **Command Data:** 44hex, 02hex, 08hex, 00hex, 07hex, E0hex (Display 8x8 bitmap character index 2 at x = 8, y = 0, colour = GREEN). **Command Data:** 44hex, 03hex, 10hex, 08hex, 00hex, 1Fhex (Display 8x8 bitmap character index 3 at x = 16, y = 8, colour = BLUE).	

4.2.4 Draw Triangle - 47hex

Command	**cmd, x1, y1, x2, y2, x3, y3, colour**(msb:lsb)	
	cmd	**47**(hex) or **G**(ascii) : Command header byte
	x1, y1, x2, y2, x3, y3	3 vertices of the triangle. These must be specified in an anti-clockwise fashion.
	colour	2 bytes (big endian) triangle colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command draws a Solid/Wire-Frame triangle. The vertices must be specified in an anti-clock wise manner, i.e. **x2 < x1 : x3 > x2 : y2 > y1 : y3 > y1** A solid or a wire frame triangle is determined by the value of the Pen Size setting. when Pen Size = 0 : triangle is solid when Pen Size = 1 : triangle is wire frame 	

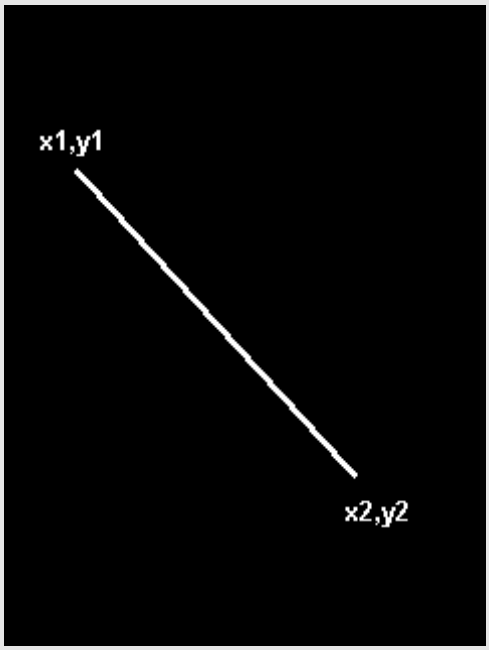
4.2.5 Draw Image-Icon - 49hex

Command	**cmd, x, y, width, height, colourMode** - (command block) **block_size, pixel1, .. pixel254** - (image part block) ... **block_size, pixeln, .. pixelN** - (image final block)	
	cmd	**49**(hex) or **I**(ascii) : Command header byte
	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.
	colourMode	**08**(hex) : 256 colour mode, 8bits/1byte per pixel. **10**(hex) : 65K colour mode, 16bits/2bytes per pixel .
	block_size	Total number of bytes for image block (max 254 pixels).
	pixel1..pixelN	Image pixel data where N is the total number of pixels. N = width x height (when colourMode = 08hex) N = 2 x width x height (when colourMode = 10hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command displays a bitmap image on to the screen with the top left corner specified by **(x, y)** and the 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. This command must be sent in a series of blocks. Each block consists of a maximum of 255 bytes. The block must be preceded with a 1 byte length of the block. After the MOTG has processed each block it will sent an ACK, either to indicate it is ready for the next block or that the image is complete. 	

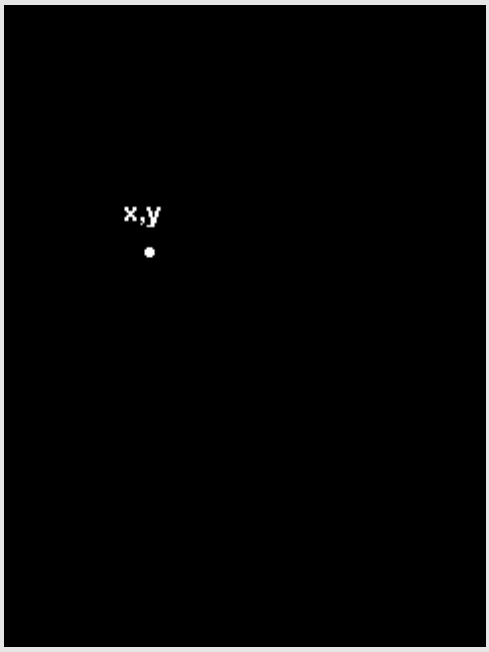
4.2.6 Set Background colour - 4Bhex

Command	**cmd, colour**(msb:lsb)	
	cmd	**4B**(hex) or **K**(ascii) : Command header byte
	colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command sets the background colour for the next erase and draw(refers to opaque mode text in Set Transparent-Opaque Text – 4Fhex) commands to be sent. Once this command is sent, the background colour will only change when it is rewritten. Nothing on the screen will be affected.	
Example	**Command Data:** 4Bhex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	

4.2.7 Draw Line – 4Chex

Command	**cmd, x1, y1, x2, y2, colour**(msb:lsb)	
	cmd	**4C**(hex) or **L**(ascii) : Command header byte
	x1	Top left horizontal start position of line.
	y1	Top left vertical start position of line.
	x2	Bottom right horizontal end position of line.
	y2	Bottom right vertical end position of line.
	colour	2 bytes define the Line colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured line from point **(x1, y1)** to point **(x2, y2)** on the screen.  	
Example	**Command Data:** 4Chex, 00hex, 00hex, 7Fhex, 7Fhex, Ffhex, FFhex Draws a WHITE line (FFFFhex) from (x1 = 00hex, y1 = 00hex) to (x2 = 7Fhex, y2 = 7Fhex).	

4.2.8 Draw Pixel - 50hex

Command	**cmd, x, y, colour**(msb:lsb)	
	cmd	**50**(hex) or **P**(ascii) : Command header byte
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
	colour	2 bytes (16 bits) define the pixel colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured pixel at location **(x, y)** on the screen.  	
Example	**Command Data:** 50hex, 01hex, 0Ahex, FFhex, FFhex Draw a WHITE pixel (FFFFhex) at location (x = 01hex, y = 0Ahex).	

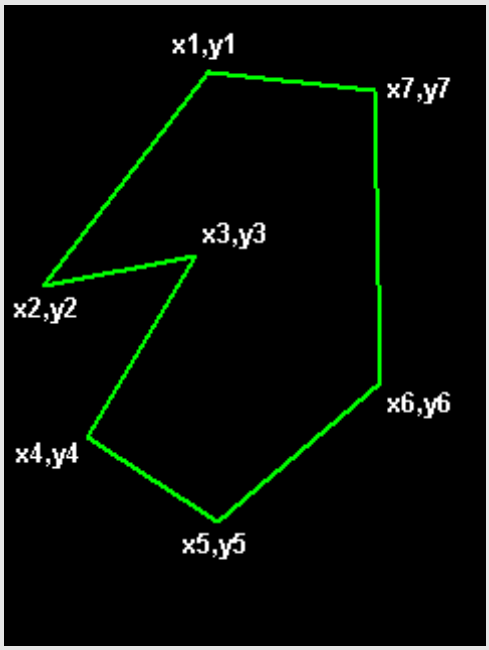
4.2.9 Read Pixel - 52hex

Command	**cmd, x, y**	
	cmd	**52**(hex) or **R**(ascii) : Command header byte
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
Response	**acknowledge, colour(**msb:lsb)	
	colour	Returns back 2 bytes (16 bits) pixel colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 (msb is 1st byte) lsb : G2G1G0B4B3B2B1B0 (lsb is 2nd byte)
Description	This command will read the colour value of a 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. **Note:** This command will always return 3 bytes, the first byte will always be the ACK/NAK.	
Example	**Command Data:** 52hex, 01hex, 0Ahex **MOTG Response:** 06hex, 00hex, 1Fhex Reads a BLUE pixel (001Fhex) at location (x = 01hex, y = 0Ahex).	

4.2.10 Screen Copy-Paste - 63hex

Command	**cmd, xs, ys, xd, yd, width, height**	
	cmd	**63**(hex) or **c**(ascii) : Command header byte
	xs	Top left horizontal start position of screen area to be copied (source).
	ys	Top left vertical start position of screen area to be copied (source).
	xd	Top left horizontal start position of where copied area is to be pasted (destination).
	yd	Top left vertical start position of where copied area is to be pasted (destination).
	width	Width of screen area to be copied (source).
	height	Height of screen area to be copied (source).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command copies a specified area of the screen as a bitmap block. The start location of the block to be copied is represented by **xs, ys** (top left corner) and the size of the area to be copied is represented by **width** and **height** parameters. The start location of where the block is to be pasted (destination) is represented by **xd, yd** (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.	

4.2.11 Draw Polygon - 67hex

Command	**cmd, vertices, x1, y1, .. , xn, yn, colour**(msb:lsb)	
	cmd	**67**(hex) or **g**(ascii) : Command header byte
	vertices	Number of vertices from 3 to 7. This byte specifies the number of vertices of the polygon.
	x1,y1,..xn, yn	Vertices of the triangle. These can be specified in any fashion.
	colour	2 bytes triangle colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command draws an Empty/Wire-Frame polygon. Up to 7 vertices can be specified in any manner. Currently only a wire frame polygon is supported.  	

4.2.12 Replace Colour - 6Bhex

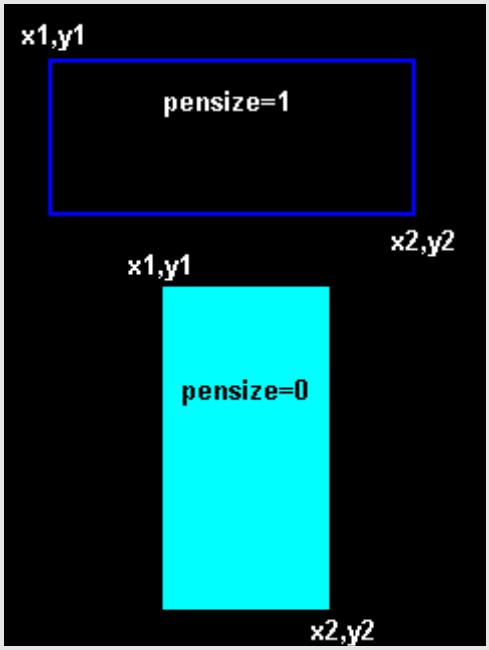
Command	**cmd, x1, y1, x2, y2, old colour**(msb:lsb)**, new colour**(msb:lsb)	
	cmd	**6B**(hex) or **k**(ascii) : Command header byte
	x1	Top left horizontal start position.
	y1	Top left vertical start position.
	x2	Bottom right horizontal end position.
	y2	Bottom right vertical end position.
	old colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
	new colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command replaces the old colour of the selected rectangular region to the new specified colour..	

4.2.13 Set Pen Size - 70hex

Command	**cmd, size**	
	cmd	**70**(hex) or **p**(ascii) : Command header byte
	size	Selects one of the 2 options: **00**hex : All graphics objects are drawn solid **01**hex : All graphics objects are drawn wire-frame Note: Does not apply to polygon command.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command determines if certain graphics objects are drawn in solid or wire frame fashion.	
Examples	**Command Data:** 70hex, 00hex (All objects will be drawn solid). **Command Data:** 70hex, 01hex (All objects will be drawn wire-frame).	

4.2.14 Draw Rectangle - 72hex

Command	**cmd, x1, y1, x2, y2, colour**(msb:lsb)	
	cmd	**72**(hex) or **r**(ascii) : Command header byte
	x1	Top left horizontal start position of rectangle.
	y1	Top left vertical start position of rectangle.
	x2	Bottom right horizontal end position of rectangle.
	y2	Bottom right vertical end position of rectangle.
	colour	2 bytes define the rectangle colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured rectangle from point **(x1, y1)** to point **(x2, y2)** on the screen. If colour is chosen to be that of the background then the effect will be erasure. If Pen Size value was previously set to 0, the rectangle will be solid, otherwise it will be wire-frame if value was 1.	

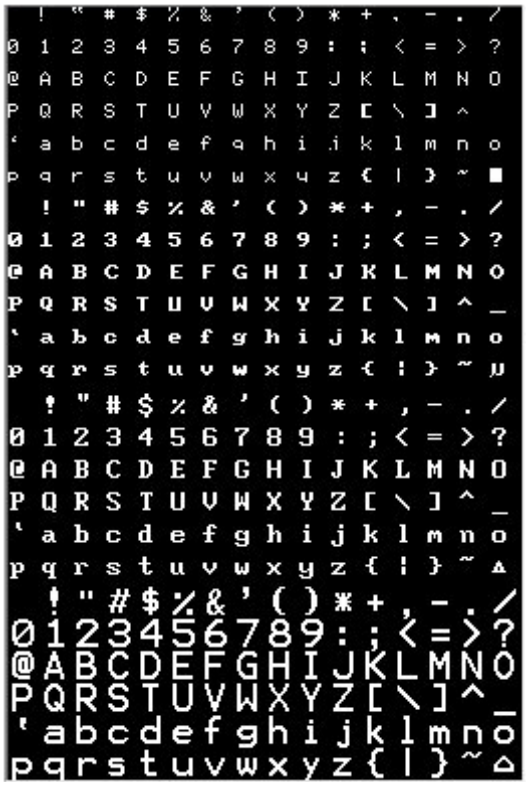


4.3 Text Commands

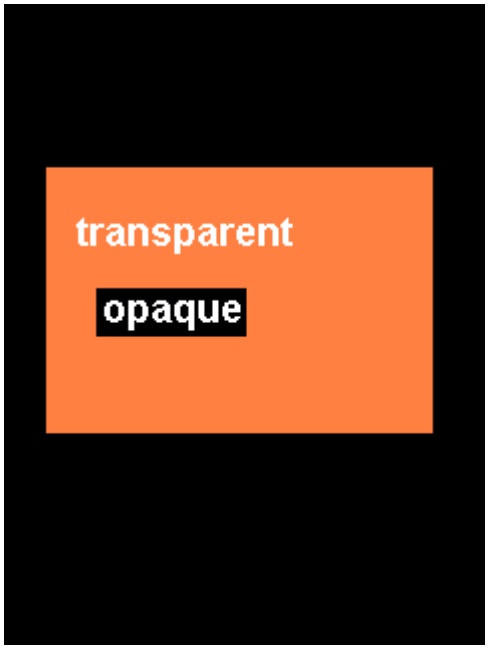
Summary of Commands in this section:

- Set Font – **46hex**
- Set Transparent-Opaque Text – **4Fhex**
- Draw "String" of ASCII Text (graphics format) – **53hex**
- Draw ASCII Character (text format) – **54hex**
- Draw Text Button – **62hex**
- Draw "String" of ASCII Text (text format) – **73hex**
- Draw ASCII Character (graphics format) – **74hex**

4.3.1 Set Font – 46hex

Command	**cmd, fontSet**	
	cmd	**46**(hex) or **F**(ascii) : Command header byte
	fontSet	Selects one of internal fonts. The supplied 3 fonts are: **00**hex : 5x7 small size font set **01**hex : 8x8 medium size font set **02**hex : 8x12 large size font set These fonts can be altered and other fonts can be added.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command selects one of the available internal fonts. Changes take place after the command is sent. Any character on the screen with the previous font set will remain as it was.	
Examples	**Command Data:** 46hex, 00hex (Select small 5x7 font). **Command Data:** 46hex, 00hex (Select medium 8x8 font). **Command Data:** 46hex, 00hex (Select large 8x12 font).	

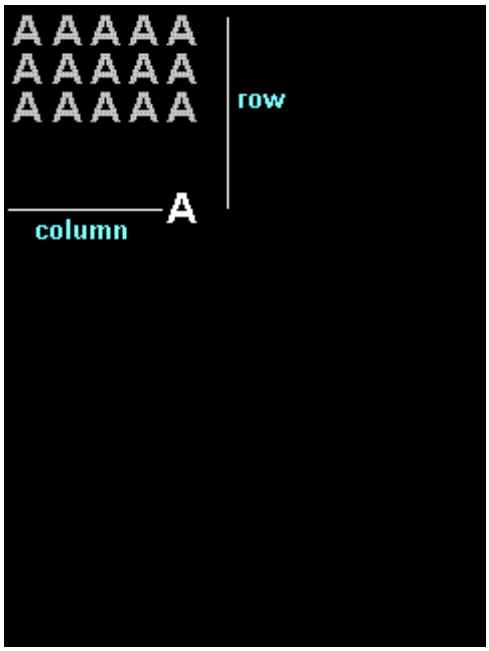
4.3.2 Set Transparent-Opaque Text - 4Fhex

Command	**cmd, mode**	
	cmd	**4F**(hex) or **O**(ascii) : Command header byte
	mode	Select one of the following options for text appearance: **00**hex : Transparent, objects behind text are visible. **01**hex : Opaque, objects behind text blocked by background.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
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.	
Examples	**Command Data:** 4Fhex, 00hex (Transparent text mode). **Command Data:** 4Fhex, 01hex (Opaque text mode).	

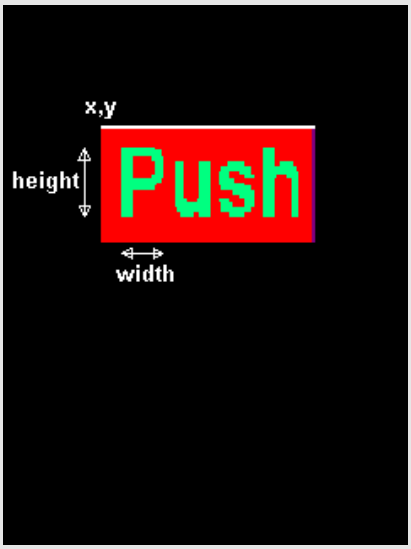
4.3.3 Draw "String" of ASCII Text (graphics format) - 53hex

Command	**cmd, x, y, font, stringColour**(msb:lsb)**, width, height, "string", terminator**	
	cmd	**53**(hex) or **S**(ascii) : Command header byte
	x	Top left horizontal start position of the string (pixel units).
	y	Top left vertical start position of the string (pixel units).
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added. **OR**ing the fonts with 0x10 will cause the string to be displayed in a proportional manner (eg 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text colour.
	width	This byte defines the width or horizontal size multiplier of the character in the string. Effects the total width of the string.
	height	This byte defines the height or vertical size multiplier of the character in the string. Effects the total height of the string.
	"string"	String of ASCII characters to be displayed (max. 256 characters).
	terminator	The string must be terminated with **00**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw/display a string of ASCII text anywhere on the screen in pixel coordinates specified by **x** and **y** parameters. The horizontal start position of the string is specified by **x** and the vertical position is specified by **y**. The string must be **terminated** with **00**hex. The size of the characters are determined by the **width** and **height** parameters. If the length of the string is longer than the maximum number of characters per line, a wrap around will occur on to the next line. Maximum string length is **256 bytes**.	

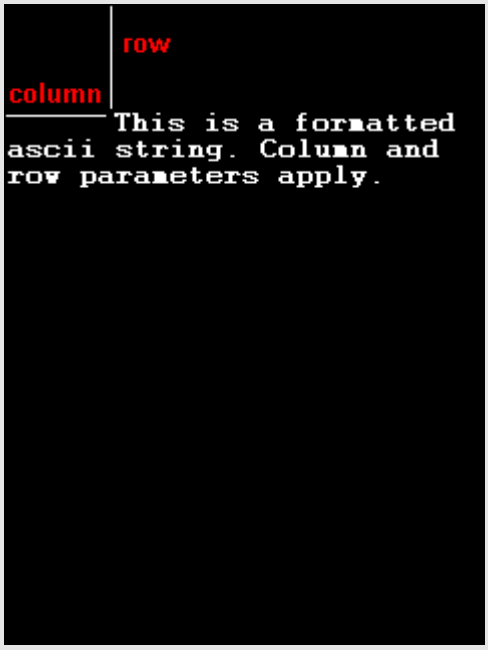
4.3.4 Draw ASCII Character (text format) - 54hex

Command	**cmd, char, column, row, charColour**(msb:lsb)	
	cmd	**54**(hex) or **T**(ascii) : Command header byte
	char	Inbuilt standard ASCII character. range : 32dec – 127dec (20hex - 7Fhex).
	column	Horizontal position of the character (character units). range : **0 - 20** for 5x7 font. range : **0 - 15** for 8x8 and 8x12 fonts.
	row	Vertical position of the character (character units). range : **0 - 15** for 5x7 and 8x8 fonts. range : **0 - 9** for 8x12 font.
	charColour	2 bytes define the character colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw/display an ASCII character anywhere on the screen in character unit coordinates. The horizontal position of the character is specified by the **column** and the vertical position is specified by the **row** parameters.	
Example	**Command Data:** 54hex, 41hex, 00hex, 00hex, FFhex, FFhex Draw/Display character 'A' (41hex) at column = 0, row = 0, colour = white (FFFFhex). 	

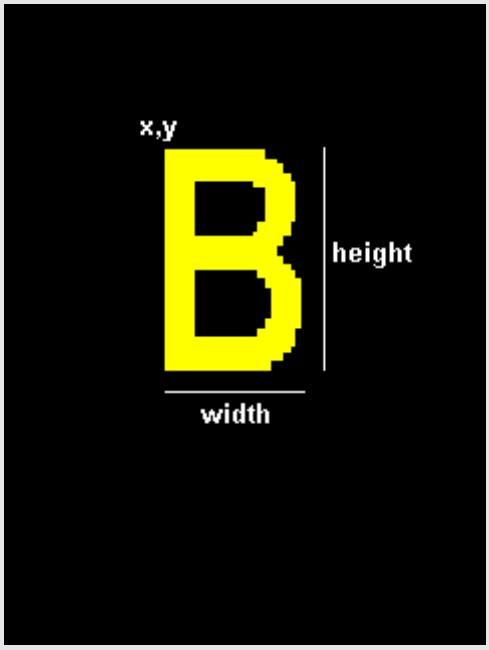
4.3.5 Draw Text Button - 62hex

Command	**cmd, state, x, y, buttonColour**(msb:lsb)**, font, stringColour**(msb:lsb)**, width, height, "string", terminator**	
	cmd	**62**(hex) or **b**(ascii) : Command header byte
	state	This byte specifies whether the displayed button is drawn **UP** (not pressed) or **DOWN** (pressed). **0** : Button Down (pressed) **1** : Button Up (not pressed)
	x	Top left horizontal start position of the button.
	y	Top left vertical start position of the button.
	buttonColour	2 bytes define the button colour.
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added.
	stringColour	2 bytes define the string text colour.
	width	This byte defines the width or horizontal size (x magnification) of the character in the string. Effects the total width of the string and button.
	height	This byte defines the height or vertical size (y magnification) of the character in the string. Effects the total height of the string and button.
	"string"	String of ASCII characters displayed inside the button. Limit the string to a single line width.
	terminator	The string must be terminated with **00**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will place a Text button similar to the ones used in a PC Windows environment. The **(x, y)** refers to the top left corner of the button and the size of the button is automatically calculated and drawn on the screen with the string text relatively justified inside the button. 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. Separate button and text colours provide many variations in appearance and format.	

4.3.6 Draw "String" of ASCII Text (text format) - 73hex

Command	**cmd, column, row, font, stringColour**(msb:lsb), **"string", terminator**	
	cmd	**73**(hex) or **s**(ascii) : Command header byte
	column	Horizontal start position of the string (character units). range : **0 - 20** for 5x7 font. range : **0 - 15** for 8x8 and 8x12 fonts.
	row	Vertical start position of the string (character units). range : **0 - 15** for 5x7 and 8x8 fonts. range : **0 - 9** for 8x12 font.
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added. **OR**ing the fonts with 0x10 will cause the string to be displayed in a proportional manner (eg 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text colour.
	"string"	String of ASCII characters to be displayed (max. 256 characters).
	terminator	The string must be terminated with **00**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw/display a string of ASCII text anywhere on the screen in character unit coordinates. The horizontal start position of the string is specified by the **column** and the vertical position is specified by the **row** parameters. The string must be **terminated** with **00**hex. If the length of the string is longer than the maximum number of characters per line, a wrap around will occur on to the next line. Maximum string length is **256 bytes**.	

4.3.7 Draw ASCII Character (graphics format) - 74hex

Command	**cmd, char, x, y, charColour**(msb:lsb)**, width, height**	
	cmd	**74**(hex) or **t**(ascii) : Command header byte
	char	Inbuilt standard ASCII character. range : 32dec – 127dec (20hex - 7Fhex).
	x	Horizontal position of the character (pixel units).
	y	Vertical position of the character (pixel units).
	charColour	2 bytes define the character colour.
	width	This byte defines the width or horizontal size (multiplier) of the character.
	height	This byte defines the height or vertical size (multiplier) of the character.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw/display an ASCII character anywhere on the screen in pixel coordinates specified by **x** and **y** parameters. Unlike the '**Draw ASCII Character (text format)**' 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 '**Set Font**' command.  	

5. LCD Screen 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 of display technologies. Blank the screen after a while or dim it very low by adjusting the contrast. Better still; implement a screen saver feature.

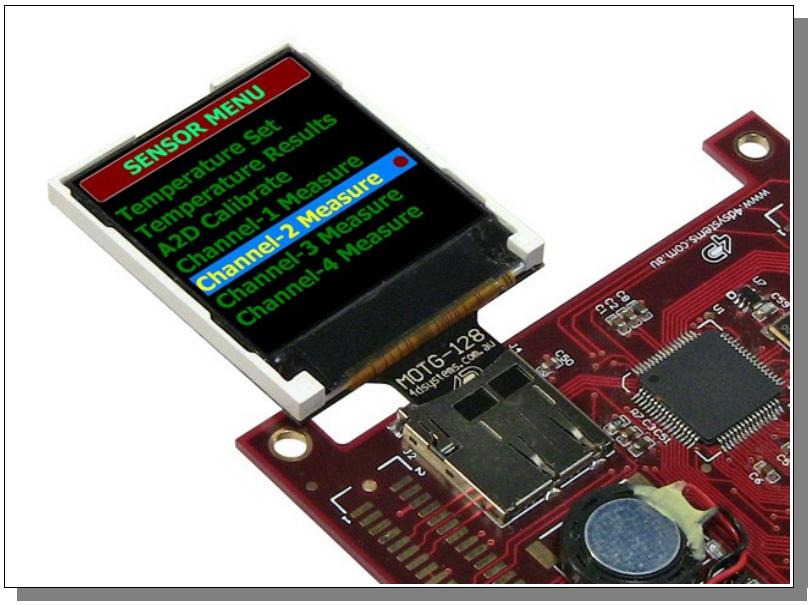
- The display can be easily scratched. The soft polarisation film on the glass surface may be damaged if rubbed by hard objects. Handle with care to avoid scratching the display.

- Moisture and water can damage the display. Moisture on the surface of a powered display will cause the electrodes to corrode. Wipe off any moisture gently or let the display dry before usage.

- Dirt from fingerprint oil and fat can easily stain the surface of the display. Gently wipe off any stains with a soft lint-free cloth.

- The performance of the display will degrade under high temperature and humidity. Avoid such conditions when storing.

- Displays are susceptible to mechanical shock and any force exerted on the module may result in deformed zebra strips and cracks.

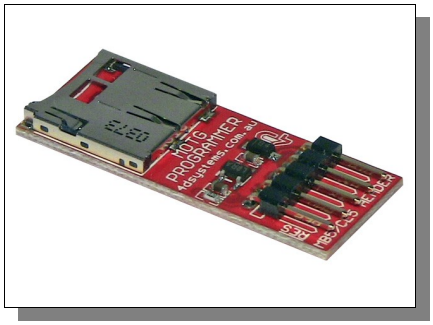
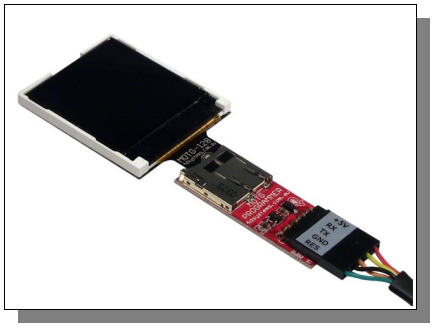


6. Development, Support and Test Tools

6.1 MOTG Programming Cable and Adaptor

The combination of the 4D-Programming-Cable and the MOTG-Program-Adaptor provides a convenient physical link between the PC and the MOTG module via the USB port. The MOTG module to PC link is required when:

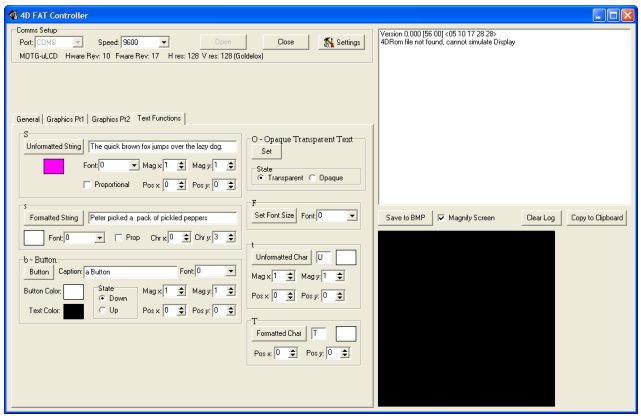
- Testing the MOTG module with FAT-Controller software tool
- Programming the MOTG module with PmmC file

☞ **Note:** The 4D-Programming-Cable and the MOTG-Program-Adaptor will need to be purchased separately.

6.2 4D FAT Controller – Software Test Tool

The 4D FAT-Controller is a free software tool to quickly test the functionality of the MOTG device using your PC, prior to writing any piece of embedded code for your platform to communicate with the MOTG module. The FAT-Controller simulates the embedded host controller and provides an easy means of exercising the available commands.



☞ **Note:** To use the MOTG module with the FAT-Controller, you'll need to program the module with a special test PmmC file. Refer to section 6.3 on how to program a PmmC file. Latest test PmmC can be found here: www.4dsystems.com.au/downloads/MOTG/MOTG-128/PmmC/Test/

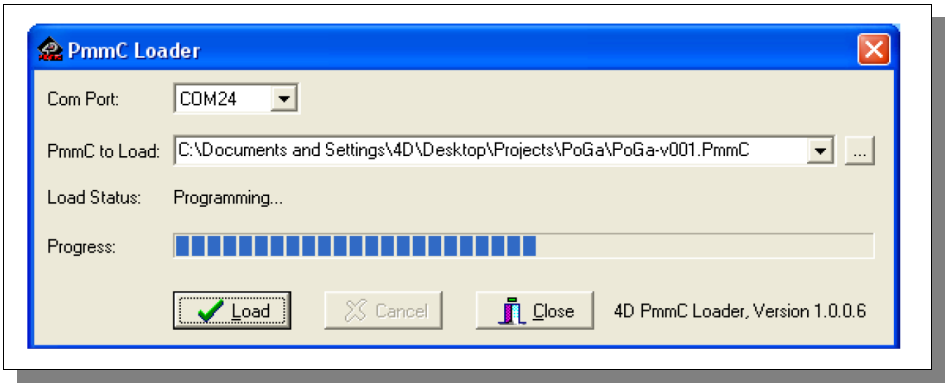
6.3 Programming the MOTG with a PmmC File

PmmC is an abbreviation of Personality-module-micro-Code. The GOLDLEOX chip used in the MOTG device is a custom controller and all functionality including the high level commands are built into the chip. This chip level configuration is available as a PmmC file and contains all of the low level micro-code information (analogy of that of a soft silicon) which define the characteristics and functionality of the MOTG device. The ability of programming the device with a PmmC file provides an extremely flexible method of customising as well as upgrading the MOTG with future enhancements.

As we make further improvements, we will release PmmC file updates and you should check regularly to benefit from these enhancements. You can download the latest version from here:
www.4dsystems.com.au/downloads/MOTG/MOTG-128/PmmC/Embedded/

To program the MOTG module with its PmmC file follow these steps:

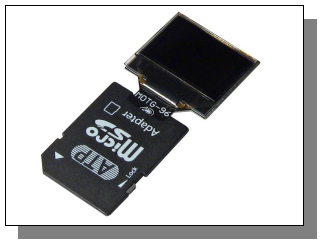
- Download the PmmC-Loader Software tool.
- Run the PmmC Loader.
- Select the COM port your MOTG is attached to (using the Programming Cable & Adaptor).
- Select the MOTG PmmC file you've just downloaded. There's a small browser button on the right hand side to help you locate the file.
- Click the 'Load' button. The progress bar will inform you when the programming is done.



Note: The MOTG module is always shipped, factory programmed, with the latest PmmC file. It is advisable that you check regularly for the latest updates here:
www.4dsystems.com.au/downloads/MOTG/MOTG-128/PmmC/Embedded/

6.4 micro-SD to SD Adaptor

If your embedded hardware platform uses an SD connector, you'll need a micro-SD to SD adaptor. These should be readily available from most suppliers and are also available from 4D Systems online shopping cart.



7. Specifications and Ratings

Absolute Maximum Ratings	
Operating ambient temperature	-35°C to +75°C
Storage temperature	-40°C +80°C
Voltage on any digital input pin with respect to GND	-0.3V to 6.0V
Voltage on SWITCH pin with respect to GND	-0.3V to 6.0V
Voltage on VCC with respect to GND	-0.3V to 6.0V
Maximum current out of GND pin	300mA
Maximum current into VCC pin	250mA
Maximum output current sunk/sourced by any pin	4.0mA
Total power dissipation	1.0W

NOTE: Stresses above those listed here may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the recommended operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Recommended Operating Conditions

Parameter	Conditions	Min	Typ	Max	Units
Supply Voltage (VCC)		3.0	3.3	3.6	V
Operating Temperature		-30	--	+70	°C
Input Low Voltage	SDI, CS pins	GND	--	0.8	V
Input High Voltage	SDI, CS pins	2.0	3.3	5.0	V
Reset Pulse	External Open Collector	2.0	--	--	µs
Operational Delay	Power-Up or External Reset	1000	--	--	ms

Global Characteristics based on Operating Conditions

Parameter	Conditions	Min	Typ	Max	Units
Supply Current (ICC)	VCC = 3.3V	14	50	70	mA
Output Low Voltage (VOL)	SDO pin, IOL = 3.4mA	--	--	0.4	V
Output High Voltage (VOH)	SDO pin, IOL = -2.0mA	2.4	--	3.3	V
Capacitive Loading	All pins	--	--	50	pF
Flash Memory Endurance	MOTG PmmC Programming	--	1000	--	E/W

Optical Characteristics							
Parameter		**Condition**	**Temp.**	**Min**	**Typ**	**Max**	**Unit**
Luminance		Vcc = 3.3V	--	--	250	--	Cd/m^2
Response Time	Rise Time(Tr)	$\theta = \Psi = 0^0$	-10^0C	--	--	--	msec
	Decay Time(Td)			--	--	--	
	Rise Time(Tr)		25^0C	--	--	240	
	Decay Time(Td)			--	--	240	
	Rise Time(Tr)		60^0C	--	--	--	
	Decay Time(Td)			--	--	--	
Viewing Angle	θ	$\Psi = 0^0$	25^0C	--	--	30	Deg
		$\Psi = 90^0$		--	--	30	
		$\Psi = 180^0$		--	--	30	
		$\Psi = 270^0$		--	--	30	
Contrast Ratio	CR	$\theta = \Psi = 0^0$	25^0C	300	450	--	--

Ordering Information

Order Code: MOTG-128
Package: 150mm x 95mm (ZIF Bag dimensions).
Packaging: Module sealed in antistatic padded ZIF bag.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either express or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage (‘High Risk Activities’). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems’ products and devices in 'High Risk Activities' and in any other application is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights..

Contact Information

For Technical Support : support@4dsystems.com.au

For Sales Support : sales@4dsystems.com.au

Website : www.4dsystems.com.au