

Digole Serial:UART/I2C/SPI Character/Graphic LCD/OLED Display Module
Programmer Manual

(last updated: February 7, 2017)

This manual may be modified without notice

Upgrade able firmware	6
Set up the communication mode	6
Brief of Commands	6
Legend:	8
Deal with pixels 255 to 511	8
Known Bugs	8
Flash Memory Map	9
Port Connection	10
Command summery	12
Immigrate software to V3.3 of firmware	12
256 color code	13
Special Command in V4.0V firmware	13
Video Box (VIDEOxywh\x00/\x01....data)	13
Special Command in V4.0B/V4.0V firmware	14
Set image's background transparent ("TRANS 0/1")	14
Display characters	14
Display a string (TT.....\x00)	14
Move current position(TPxy)	15
Enhanced move current position(ETPxy)	15
Move position to last(ETB)	15
Move position offset(ETOxy)	15
Cursor(CSd)	15
Draw graphics	16
Set current graphic position(GPxy)	16
Draw a pixel(DPxy)	16
Draw line(LNx1y1x2y2)	16
Draw line to(LNxxy)	16
Draw rectangle(DRx1y1x2y2)	16
Draw filled rectangle(FRx1y1x2y2)	16

Draw circle(CCxyrf)	17
Draw image	17
Move area on the screen(MAxywhOxOy)	18
Drawing Decoration -----	19
Clear screen(CL).....	19
Set background color(BGC)	19
Set foreground color(SCc,ESCrgb).....	19
Set line pattern(SLPd).....	19
Set draw direction(SDd)	19
Set draw mode(DMd)	19
Set output/draw window(DWWINxywh)	20
Reset draw window(RSTDW)	20
Clear draw window(WINCL)	20
For Mono display only -----	20
Refresh screen instantly(FS0/1)	20
Set screen Normal/Inverse(INV0/1)	21
Fonts -----	22
Change current font(SFd)	22
Download standard user font(SUFnL...d...)	22
Download user font to flash chip(FLMWRal...d...)	22
Use user font in flash chip(SFFa)	23
Command Set -----	23
What is command set?	23
Write command set to flash(FLMWRal...d...)	23
Run command set(FLMCSa)	24
Read data from communication port-----	24
Use EEPROM(V3.3) -----	24
Write data to EEPROM(WREPal...d...)	24
Read data from EEPROM(RDEPal)	24
Use Flash-----	24

Use flash in MCU	24
Use flash in flash chip	24
Write data to flash(FLMWRal...d...)	25
Run command set(FLMCSa)	25
Read data in flash chip(FLMRDaI)	26
Erase flash memory in flash chip(FLMERaI)	26
Touch Panel -----	27
Calibrate touch screen(TUCHC)	27
Read touched coordinate(RPNXYW).....	27
Read a click event(RPNXYC)	27
Read touch panel instantly(RPNXYI), check screen pressed	27
Read voltage(RDBAT).....	27
Read analog(RDAUX).....	27
Read temperature(RDTMP)	28
Power management-----	28
Backlight brightness(BLd)	28
Turn screen on/off(SOOd)	28
Turn MCU off(DNMCU)	28
Turn module off(DNALL)	28
Setting and configuration -----	29
Start screen(welcome screen or splash screen)	29
Enable/disable start screen(DSSd)	29
Configuration show on/off(DCd).....	29
Download start screen to module(SSSI...d...)	29
Change I2C address(SI2CAa)	29
Set SPI mode(SPIMD0~3)	30
Change UART baud(SBrate)	30
Config universal character LCD adapter(STCRcr\x80\xC0\x94\xD4)	30
Config universal graphic LCD adapter(SLCDx...)	30
Adjust LCD contrast(CTx)	30
Others -----	31

Delay a period(DLYx)	31
Escape commands in V4.0 -----	31

Upgrade able firmware

The firmware on most of our display modules are upgrade able, please visit the firmware page for details:

www.digole.com/fw

Set up the communication mode

There are 3 different communication modes on all products: UART, I2C and SPI, what you need is just use solder to short the I2C/SPI jumper on adapter and make it works at I2C or SPI, if both jumpers are open, it works at UART, you can find a similar jumper like this: on board.



PROTOCOLS:






- UART : 8-N-1, 8bits, No parity bit, 1 stop bit.
- I2C: Slave Mode, 7-bit address, default address is Hex:27, change able. This mode may give you a headache due to more signal options in I2C, but we make it works as standard, you just need setup your I2C on master controller as Standard Master Mode.
- SPI: 8-bits, MSB first, data on raise edge of SCK sampled; this is Standard setting on SPI too.

Brief of Commands

TEXT	Position adjust: - "TP" set text position, - "ETP" enhanced set text position, - "ETO" set text offset, - "ETB" back to last text position Draw Text: - "TT" display text, - "TRT" text return	Appearance adjust: - "SF" set font, - "SFF" set font in flash, - "SUF" save user font, - "SC" set drawing color (8bit format), - "ESC" enhanced set drawing color (262K), - "SD" set direction, - "CS" cursor on/off, - "DM" drawing mode	Color screen V3.0: - "BGC" use current color as background, - "DWWIN" set drawing window, - "RSTDW" reset drawing window, - "WINCL" clear drawing window use background
Graphic	Position adjust: - "GP" set graphic position, Draw functions: - "DP" draw pixel, - "LN" draw line, - "LT" draw line to, - "DR" draw rectangle, - "FR" draw filled rectangle, - "CC" draw circle, - "MA" move an area, - "DIM" display mono image, - "EDIM1", "EDIM2", "EDIM3" display color image (256, 65K 262K color)	Appearance adjust: - "SD" set direction, - "SC" set color, - "ESC" enhanced set color, - "SLP" set line pattern, - "DM" drawing mode	Color screen V3.0: "BGC", "DWWIN", "RSTDW", "WINCL" same as above
Communication	"SI2CA"-Set I2C address, "SB"-set UART baud rate, "DC"-display module configuration		
Power Manage	"BL"-adjust backlight, "DNALL"-put module in sleep, "DNMCU"-put MCU sleep, "SOO"-turn screen on/off		
Screen	"SSS"-Save start screen, "DSS"-display start screen, "SOO"-turn screen on/off, "MCD"-send command to screen, "MDT"-send data to screen, "CL"-clear screen use background color, "CT"-set contrast, "SLCD"-config mono LCD (ST7920, ST7565, KS0108), "VIDEO"- Write Video data direct to screen, V4.0V		
Touch Panel	"RPNXY"[I/W/C]-read x,y, I=instant, W=wait touched, C=Click, "TUCHC"- calibrate touch panel, "RDBAT"-read battery voltage, "RDAUX"-read aux pin, "RDTMP"-read chip temperature,		
Flash Memory	"FLMER"-flash erase, "FLMRD"-flash read, "FLMWR"-flash write, "FLMCS"-run command set in flash, "SFF"-set font in flash		
EEPROM	"WREP"-write data to EEPROM, "RDEP"-read data from EEPROM		

Output Window	" D W W I N "-set output window," W I N C L "-clean the output window," R S T D W "-reset the output window to full screen
Other	" D L Y "-delay a period

Legend:

	-----	Text (character) Display
	-----	Monochrome Graphic Display
	-----	Color Graphic Display
	-----	All version of Firmware
	-----	Special version of Firmware

Deal with pixels 255 to 511

How do we recognize value from 0 to 511 used in position and length of pixel (x,y,w,h,r)?

In most small screen, the screen size usually less than 255, one byte of data can handle such screen, but in larger screen, the size may exceed 255, in order to make your code compatible for all size of screen, we use this technical: if the value is <255, just send one byte of the value, if >=255, send first byte of 255 then follow the rest of value, eg.: for value 120, just send one byte of value of 120, if the value is 310, send 255 first, then value of 55 as second byte, the value 255+55=310.

sample routine in C:

```
void writePosition(int p)
{
    if(p>255)
    {
        write(255);    //write a byte to COM port
        p-=255;
    }
    write(p);
}
```

Known Bugs

V3.3: SPI mode was set as mode 2 by accidentally, all other versions were mode 0, all modes list here:

https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

V3.0: draw window function not adjusted correctly after change draw direction, you need set draw direction first, then set a draw window.

Flash Memory Map

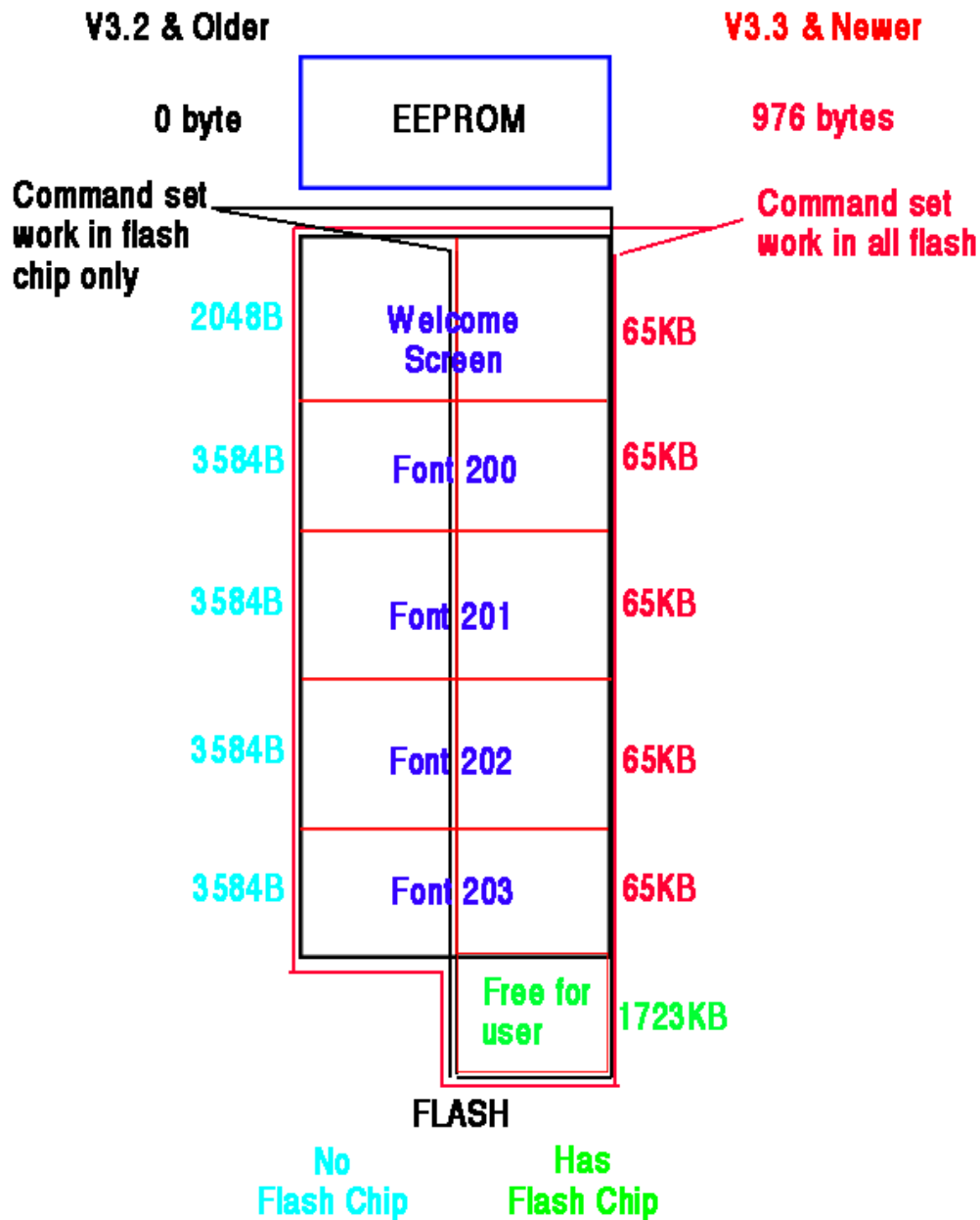
Flash memory size: 2MB~16MB if flash chip installed, 16KB if none flash chip installed.

Usage: 5 block of 65KB from address 0, used for welcome screen and 4 user fonts if flash chip installed. otherwise: 0~2047B for welcome screen, and 4x3584B for 4 user fonts.

User fonts: unlimited user fonts in flash chip, or 4 user fonts in 16KB flash.

Command sets: unlimited command set in flash chip, V3.3: unlimited command set in 16KB flash.

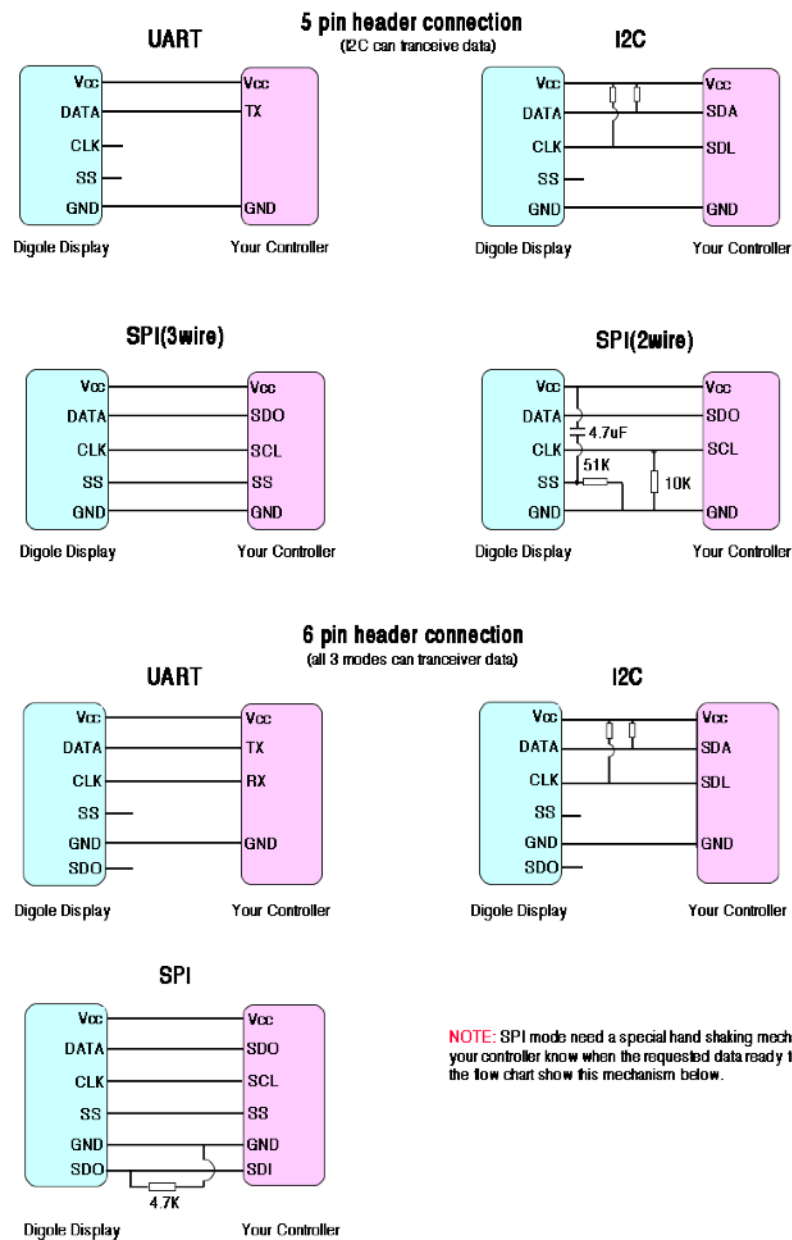
Data protection: data in 16KB flash can't be read out, but data in flash chip can.



Port Connection

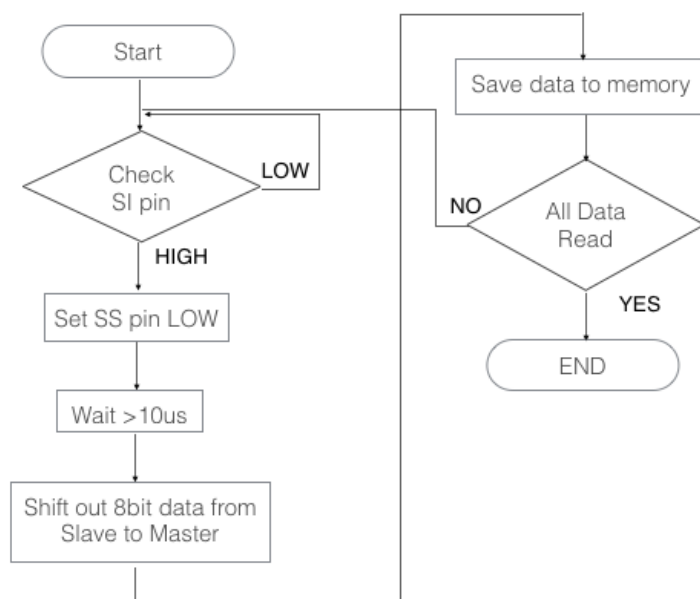
In the 5 pin header modules, only I2C mode can transmit and receive data, other mode only can transmit data to display module, if you need read data from EEPROM(V3.3), you only can use I2C mode to do that. All modes can transceive data in 6 pin header modules

On the 2 wire SPI mode, if the master circuit was setting slow, the C:4.7uF and R:51K on SS pin, build a RC delay circuit, it will disable the SPI port for about 200ms when power on, the 10K on CLK ground noise if SCL floating. The 2 pull resistors need over 10K on 5 pin header module, if they are too low, there is a problem when transfer data from slave to master.



NOTE: SPI mode need a special hand shaking mechanism to let your controller know when the requested data ready to clock out, the flow chart show this mechanism below.

SPI transceiver data flow chart:



Command summery



There are about 53 different commands usable in Digole's serial display modules, that include:

- 1) Display characters
- 2) Draw graphics
- 3) Drawing decoration
- 4) Fonts
- 5) Command set
- 6) Use EEPROM
- 7) Use flash memory
- 8) Power management
- 9) Operating touch screen panel
- 10) Setting and configuration
- 11) Others

Immigrate software to V3.3 of firmware

If you already developed the software before for our display module with older firmware version than V3.3, some modification need to be done in order to use the newest module:

- 1) value 0, 13 and 15 can be used to terminate the string in "TT" command, but in V3.3, only value of 0 used for that, value 13 used to move cursor to next line, and value 15 used to move cursor to the begin of line.
- 2) set background color command (BGC) changed. A byte of background color should follow to "BGC" in new version, and not affect the foreground color, but in old version, "BGC" only copy the color from foreground to background.

eg.: set a red background: "BGCxE0" in , "SCxE0BGC" in , sample code

```
void setBgColor(uint8_t color) //set current color as background
```

```
{
#ifdef V33
    write("BGC");
    write(color);
#else
    write("SC");
    write(color);
    write("BGC");
#endif
}
```

- 3) the sequence of 2 byte for font's length when downloading the user font is MSB, LSB in new version, but it was LSB, MSB in older version.

Sample code:

```
void downloadUserFont(int lon, const unsigned char *data, uint8_t font_sect) {
    uint8_t c;
    unsigned char b;
    write("SUF");
    write(font_sect);
    #if defined(V33)
        write((unsigned char) (lon / 256));
        write((unsigned char) (lon % 256));
    #else
```

```

write((unsigned char) (lon % 256));
write((unsigned char) (lon / 256));
#endif
b = 0;
for (int j = 0; j < lon; j++) {
    c = pgm_read_byte_near(data + j);
    write(c);
    if ((++b) == 64) {
        b = 0, delay(40);    //delay 40ms
    }
}
}

```

- 4) welcome screen and commands set, you only need add an extra value of 255 to indicate the end of data in new version, but in older version, you need put the data length at the beginning of data.
welcome screen changed to commands set format in V3.3 on monochrome screen too, if your welcome screen is bitmap, please use our online tool to convert it to commands set:
http://www.digole.com/tools/Mono_welcome_to_Commands_set.php
- 5) now all multi-bytes of value which indicate the length, address, position changed to MSB, LSB format, the user will not be confused any more

256 color code

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Special Command in V4.0V firmware

Video Box (VIDEOxywh\x00\x01....data)

This command let user to send raw image data to the LCD panel directly, after command "VIDEO", follow by 2 integer data x, y to indicate the top-left position counted as pixels where of the video box, the available value of x,y are from 0 to 65535 but not exceed the LCD panel size, then 2 bytes of value to indicate the box width and height, the available value are from 0 to 255.

After defined the video box, the next byte of value indicate the color depth of each pixel, if value is 0, the color depth is 16BIT(2 bytes data) : **RRRRRRGGG** **GGGBBBBB**, if value is 1, the color depth is 18BIT (3 bytes data): **RRRRRR00**, **GGGGGG00**, **BBBBBB00**, except 2.6" IPS module, which only accept 18BIT (3 bytes data) with the format as: **0RRRRRR0**, **0GGGGGG0**, **0BBBBBB0**.

Then all the coming data from serial port will be sent to the video box directly.

How to exit the video mode? if user want to exit video mode and display other things on the screen, just stop send the data to the display, wait about 120ms, the module will exit video mode, and all following datas are treated as regular commands.

The maximum speed: UART mode-460800bps, I2C->400K bps, SPI-10MHz

Here is the example code for reference:

```

writeStr("VIDEO"); // "VIDEO" command on Digole Display, or use writeStr("VIDEO");
writeByte(0); //the X=0 position of top-left for Video window
writeByte(0);
writeByte(0); //the Y=30 position .....

```

```
writeByte(30);
if (k & 1)
writeByte(150); //the width of Video window, here is 150pixels
writeByte(100); //the height of Video window, here is 100pixels
writeByte(0); //the color format is 16bit, if the value is 1, the format is 18bit, doesn't matter on 2.6" IPS display
delay(300); //give the display enough time to init the Video box
for(long int n=0;n<150*100*2*20;n++) //send 20 frames data to display
    writeByte(data[n]);
delay(200); //this delay will cause the Digole display exit the "VIDEO" mode
printText("Thank you!"); //print "Thank you!" on the screen
```

Special Command in V4.0B/V4.0V firmware

Set image's background transparent ("TRANS 0/1")

When we show an image (256, 16K or 262K color) on the screen, the image occupy a rectangle area, this command can change the image shape on the screen, means the pixels in the rectangle area can be transparent, in order to do this, just sent the color to black (00) where want it to be transparent, then set "TRANS\x01" command, here is our logo shows on a blue background when TRANS=0 and TRANS=1:



Digole's Logo



TRANS=0



TRANS=1

Display characters

Display a string (TT.....\x00)

Command: **TT**. following with characters until value 0 received, the value 0 also is the terminator of a regular string. This command display a giving string on the current position, the position adjusted automatically, if the position reach the most right of screen, it move to the beginning of next line of character, the module can calculate the next character line according the current using font's size.

The value 10 and 13 (\n and \r in C, LF and CR in ASCII table) can move the current position, value 10 move to next line, value 13 move to the beginning of current line, use 10 and 13 move to the beginning of next line.

eg.:

"TTHello\nDigole" output on screen:

Hello

Digole

"TTHello\n\rDigole" output on screen:

Hello

Digole

Arduino lib function: drawStr(x,y,string),print(char),print(string),print(int).

Move current position(TPxy)

Command: **TP**. follow by x,y position, the x,y value is not refer to pixels, they are the column and row value that MCU calculated based on the font's size (usuall use space size to calculate current font's size).

The top-left position is: 0,0.

Arduino lib function: `setPrintPos(x,y,0); drawStr(x,y,string);`

Enhanced move current position(ETPxy)

Command: **ETP**. follow by x,y position in pixels, this function can adjust the text position as pixels on screen.

The top-left position is: 0,0.

Arduino lib function: `setPrintPos(x,y,0);`

Move position to last(ETB)

Command: **ETB**. after each character printed on screen, the current position is adjust to new value, the MCU also remembered the last character's position, if you want print few characters at same position, you can use this function. eg.: print a bold **K**, the command sequence is: "TTK\x00ETB ETO\x02\x00TTK", in here we use move position offset command, it move the last position 2 pixels right.

Arduino lib function: `setTextPosBack();`

Move position offset(ETOxy)

Command: **ETO**, follow by x,y value in pixels, the range of x,y value is -127~127, it adjust the current position with the relative value.

eg.: if current position is 46, 30, and the x=-10, y=5. after run this command, the new position is:36,35.

Arduino lib function: `setTextPosOffset(x,y);`

Cursor(CSd)

Command: **CS**, follow by a value 0 or 1, if d=1, the module will show a small cursor at the current position, otherwise, no cursor displayed.

Sorry, the cursor on function not always work properly for some reason, for example, when the cursor show up, you clear the area where cursor in, the cursor is disappeared, but the module still think it's show up, this will make it massed when module blinking the cursor.

Arduino lib function: `enableCursor(); disableCursor();`

Draw graphics

CGP — — — — Current Graphic Position

Set current graphic position(GPxy)

Command: **GP**, follow by x,y value in pixels, this function is same with “ETP” since firmware V3.0 and later, but, in older version before V3.0, the CGP and text position are separated.

Arduino lib function: `setPrintPos(x,y,0);`

Draw a pixel(DPxy)

Command: **DP**, follow by x,y value in pixels. This function draw a pixel at (x,y) position using foreground color (set by commands “SC” or “ESC”), the pixel also logic operate(draw mode, set by “DM” command) with existing pixel at same position.

This function doesn't change CGP.

Arduino lib function: `drawPixel(x,y);`

Draw line(LNx1y1x2y2)

Command: **LN**, follow by two coordinates which indicate where the line drawing from and to (x1,y1)(x2,y2) in pixels, the foreground color and draw mode affect this function, also affected by line pattern (set by command “SLP”).

The CGP also move to (x2,y2) after function executed.

eg.: draw a line from (30,40) to (200,300), the command sequence is this: “LN\x1E\x28\xC8\xFF\x2D”, because the value of 300 exceed one byte, you need use 2 bytes format.

Arduino lib function: `drawLine(x1,y1,x2,y2); drawHLine(x,y,w); drawVLine(x,y,h);`

Draw line to(LNxxy)

Command: **LT**, follow by the destination coordinate (x,y) in pixels, this function draw a line from current position to (x,y), everything else same as “draw line” function.

The CGP move to (x,y).

Arduino lib function: `drawLineTo(x,y);`

Draw rectangle(DRx1y1x2y2)

Command: **DR**, follow by the top-left coordinate (x,y), and the right-bottom coordinate, all in pixels, this function affect by foreground color, draw mode and line pattern.

The CGP move to (x2,y2).

Arduino lib function: `drawFrame(x,y,w,h);` //Arduino lib use width and height follow (x,y)

Draw filled rectangle(FRx1y1x2y2)

Command: **FR**, this function similar with “draw rectangle”, but will use current foreground color, draw mode and line pattern to fill this rectangle.

The CGP move to (x2,y2).

Arduino lib function: `drawBox(x,y,w,h);` //Arduino lib use width and height follow (x,y)

Draw circle(CCxyrf)

Command: **CC**, follow by the coordinate of circle center(x,y), then the radius, the “f” is indicate the circle is filled or not, if f=1, the circle is filled, this function affect by foreground color and draw mode, but **not affected** by line pattern. The CGP move to (x,y).

Arduino lib function: `drawCircle(x,y,r,f); drawDisc(x,y,r);`

Draw image

There are 4 different functions for drawing images, the difference are on the color depth which function can draw:

1) Draw black/white image (DIMxywh...d...):

Command: **DIM**, follow by the top-left coordinate (x,y) of the image, then image width (w), height (h), then follow the image data (...d...), each bit represent a pixel in the image.

Data in one byte can't cross different lines, that means, if the width of image is 12 pixels, you need 2 bytes for each line. MSB is on the left side.

This function affected by foreground color and draw mode, that means you can display different color of image use same command, but just set different foreground color before drawing.

CGP: not change.

Arduino lib function: `drawBitmap(x,y,w,h,*data);`

2) Draw 256 color of image(EDIM1xywh...d...):

Command: **EDIM1**, everything similar with “DIM”, except following:

- a) one byte represent a pixel, so, the color depth is 256, the color format is (MSB-LSB):**RRRGGGBB** (332)
- b) draw mode and foreground color will not affect it.

CGP: not change

Arduino lib function: `drawBitmap256(x,y,w,h,*data);`

2) Draw 64K color of image(EDIM2xywh...d...):

Command: **EDIM2**, everything similar with “EDIM1”, except this function use 2 bytes to represent one pixel, the color depth is 64K, the color format is: (MSB-LSB):**RRRRRGGG** , **GGGBBBBB**(565).

Arduino lib function: no

2) Draw 262K color of image(EDIM3xywh...d...):

Command: **EDIM3**, everything similar with “EDIM1”, except this function use 3 bytes to represent one pixel, but only the 6 MSB used for a color, the color depth is 262K, the color format is: (MSB-LSB):00**RRRRRR**, 00**GGGGGG**, 00**BBBBBB**.

CGP: not change

Arduino lib function: `drawBitmap262K(x,y,w,h,*data);`



Draw window introduced since V3.2, all coordinate (except draw image) were refereed to draw window.



NOTE: if you use draw image function in command set, you don't need set (x,y) coordinate, the module will use CGP as (x,y), this setting will let you display a same image (like button) in different position, and save your code size, what you do is: change the CGP before calling the command set which display the image(button?).

Move area on the screen(MAxywhOxOy)

Command: **MA**, follow by the top-left coordinate of the area, then the area width and height, all as pixels the Ox, and Oy are the offset refers to (x,y), it will move the area(x,y)-(x+width,y+height) to new top-left position of(x+Ox,y+Oy), the value of Ox, Oy are -127~127.

This function is useful to scroll screen in 4 directions.

Arduino lib function: `moveArea(x,y,w,h,Ox,Oy);`

Drawing Decoration


Clear screen(CL)


Command: **CL**, clear the screen panel.   : use current background color to clear the screen.

This function also reset current font to 0, screen rotation to 0, x position to 0, draw mode to 'C', draw window to full screen, line pattern to 0xff.

Arduino lib function: `clearScreen();`

Set background color(BGC)


Command: **BGC**, in , this command only transfer current foreground color to background, so, you need set a foreground color first, then use "**BGC**", the background and foreground have same color now.


In , you need specify the one byte value of color (256 color depth, 332 format) follow this command. eg.: set a red

background: "**BGC**\xE0" in , "**SC**\xE0**BGC**" in .

Arduino lib function: `setBgColor(c);`

Set foreground color(SCc,ESCrgb)

 There are 2 commands to set foreground color: "**SC**" then follow by a byte to set 256 color depth, and "**ESC**" follow by 3 bytes to set 262K color depth, the color format refer to "EDIM1" and "EDIM3" commands.

 Only 2 different value for these display module: 0 and 1.

Arduino lib function: `setColor(c); setTrueColor(r,g,b);`

Set line pattern(SLPd)

Command: **SLP**, follow by a byte indicate which pixel should display or not, there are 8 bits in a byte, so when drawing line, the module will repeat every 8 bits according to the line pattern value.

eg.: "**SLP**\x55" command will let the draw line/rectangle function to draw a dotted line, because "\x55" equal: 0B01010101, if the bit is 0, that pixel will not displayed.

If the line pattern value is: 0B11010111, the drawing line is dashed.

Arduino lib function: `setLinePattern(d);`

Set draw direction(SDd)

Command: **SD**, then follow by the 0,1,2,3 direction you want, the original direction is 0, direction 1,2,3 represent 90,180 and 270 degree clockwise(also means turn display panel anti-clockwise). if the value out of {0~3}, the final direction is: $d\%4$, means, value 4 is 0 direction, and 5 is 1 direction...

Arduino lib function: `setRotation(d); undoRotation(); setRot90(); setRot180(); setRot270();`

Set draw mode(DMd)

Command: **DM**, follow by a byte of draw mode which only one letter can be used from {C,I,!,~,&^,O,o}. Draw mode is used to tell the module how to display the color of pixel using current foreground color operating with the existing pixel, there are 6 modes available:

'C'-Copy, doesn't matter the existing pixel on screen, this mode use current foreground color to over write the pixel, for "TT" command, it also clear the char box as back ground color, all other modes will not clear the char box.

'I'-Or (the "or" letter is not capital i, it's the shift value on key "\"), use current foreground color "OR" with the existing pixel.

'!' or '~'-Not, doesn't matter the current foreground color, it just "NOT" the existing pixel.

'&'-And, use current foreground color "AND" with existing pixel.

'^'-Xor, use current foreground color "XOR" with existing pixel.

and **all other letter**: 'O' or 'o' means "Over write", similar with 'C', but not clear the char box when using "TT". This mode will let you display text on a picture nicely.

eg.: at (20,20), the pixel color is red: 0B11100000, the foreground color is: 0B00011111, if the draw mode is OR, then when you draw a pixel at (20,20), the new pixel color is White, 0B00011111 OR 0B11100000 =0B11111111, this value is White color. But, if you set the draw mode is AND, the new color is Black.


Draw mode will affect all out put on screen except: display image, clear screen and clear draw window.


Arduino lib function: `setMode(d);`

Set output/draw window(DWWINxywh)

Draw window was embedded since firmware version 3.2 and later, instead of output to full screen, user can set a smaller rectangle area as draw window, then all following output will be showing in this window and the coordinate also refers to the top-left corner of draw window. This ability provide user a new way to relocate an area of content on the screen to different location easily, just change the draw window to the desired location, then done.

Command: **DWWIN**, follow by top-left coordinate value (x,y), then draw window's width (w) and height (h) all value in pixels.

 there are some bugs for this function: the value of draw window box will not be adjust when changing draw direction, if you need to use draw window in different draw direction, set draw direction first, then set draw window

later. in , the module will adjust the value of draw window automatically when draw direction changed.

Arduino lib function: `setDrawWindow(x,y,w,h);`

Reset draw window(RSTDW)

Command: **RSTDW**; remove the current draw window, what the module do is set the new draw window to full screen.

Arduino lib function: `resetDrawWindow();`

Clear draw window(WINCL)

Command: **WINCL**, clear the draw window use background color.

Arduino lib function: `cleanDrawWindow();`

For Mono display only

Refresh screen instantly(FS0/1)

Command: **FS**, follow by a byte of value 0 or 1. if value is 0, the module will not refresh the screen until it receive a fresh screen command such as "FS2", if the value is 1, the module will refresh the screen from internal screen buffer to screen when the module is idle (no more pending commands in receiving buffer) automatically.

This command only available on Black/White display module, the color module always refresh the screen instantly because no screen buffer used in onboard MCU.

If you need update the screen rapidly, disable the auto-refresh will help to avoid the screen flicking: draw all information to the screen buffer in MCU, the refresh the screen at once.

Arduino lib function: `flushScreen(d);`

Set screen Normal/Inverse(INV0/1)

Command: **INV**, follow a byte of value 0 or 1 to indicate the screen content normal or inverse, this command only available on some monochrome module, and the content is affected instantly.

Fonts

Only u8glib format font can be used in our module, you can find vary of font data from here: <https://github.com/olikraus/u8glib/tree/master/fntsrc>, the list of available font is here: <https://github.com/olikraus/u8glib/wiki/fontsize>. If none of standard u8glib font meet your requirement, we also provide an online tool to convert image file to custom u8glib compatible font data, the instruction is here: <http://www.digole.com/forum.php?topicID=330>

Change current font(SFd)

There are 7 fonts pre-installed in onboard MCU, the fonts' code and name are:

Font code	Font name	Font code	Font name
6	u8g_font_4x6	200	User font 1
10	u8g_font_6x10	201	User font 2
18	u8g_font_9x18B	202	User font 3
51	u8g_font_osr18	203	User font 4
120	u8g_font_gdr20		
123	u8g_font_osr35n		
0(default)	u8g_font_unifont		

The module also reserved flash memory space for each user fonts, the available user font space depends on the flash chip installed or not, if none flash chip on module, the user fonts will use MCU flash memory, and memory space for each font is 3584 bytes.

V3.3 If flash chip installed, the user fonts will stored in the chip, and the size for each font is 65K bytes.

All Ver When you want to change current font, use command: **SF**, follow by the font's code.

Arduino lib function: `setFont(d);`

Download standard user font(SUFnL...d...)

(You can use "SF"+200~203 command to use these fonts which downloaded by this command).
Command: **SUF**, follow by a byte of the index number of user font space, then 2 bytes of data length of font, then the

font data. Please note, in **V3.2** and early version, the font length send to module is: LSB, MSB formation, but since **V3.3**, the format changed to MSB, LSB.

eg.: save a user font to 201, the length is 1500 bytes, the command sequence should be:

V3.2 : "SUFx01\xDC\x05...data....", **V3.3** : "SUFx01\x05\xDC...data....".

40ms delay is needed after each 64 bytes of data send out, refer "Write data to flash" command.

Arduino lib function: `downloadUserFont(length, *data, index);` //if your module is: **V3.3**, put: `#define V33` at the top of your sketch

Download user font to flash chip(FLMWRal...d...)

There is no special command to download font data to flash chip, instead of it, use regular "download data to flash chip" command:**FLMWR** to any address in the flash chip, then use the next command to tell the module use this font.

There is no limitation on the numbers and size of font in flash chip, until the memory size of the chip full.

Following **FLMWR**, there **3** bytes to indicate the start address in chip, and **3** bytes indicate the length of data, all MSB first, then follow by all data byte. 3 bytes can access the address/length 0~16,777,215, and the onboard flash chip usually 2MB~16MB.

Note: You need to erase the flash memory space before writing data, please refer to the “erase flash memory command”.

40ms delay is needed after each 64 bytes of data send out, refer “Write data to flash” command.

Arduino lib function: `flashWrite(address,length,*data);`

Use user font in flash chip(SFFa)


Command: **SFF**, follow by 3 bytes of address which the font data in flash chip start from.


Arduino lib function: `setFlashFont(address);`

Command Set

What is command set?

The command set is a command sequence which contain one or more commands and data, that do complicated drawing functions on the screen. Save the command set in the flash chip or MCU flash, run this command set when

you needed later, using command set will save you lot of hardware and software resources.  A command set must

end with an extra byte of value 255,  but earlier version, you need put 2 bytes value on the beginning to indicate the data length of command set.

YES: Only setting and drawing functions (draw Text or graph) can be put in command set. **NO:** Functions to read data from module and write data to EEPROM and flash can't be embedded in command set.

CHANGE:

Few functions are little bit different when used in command set than regular:


Commands : **DIM**, **EDIM1**, **EDIM2**, **EDIM3**, these 4 command will show a picture on screen, in regular usage, the top-left coordinate will follow the command, then width and height. But when it in command set, the width and height will follow to the command directly, the top-left coordinate is the CGP, that means you need set CGP before these 4 commands in command set, this will give you the ability to display same images at different location on screen.

Command: **LT**. Line to command accept destination coordinate in regular usage, but it accept the coordinate offset when used in command set.

eg.: “GP\x0A\x10TTHello user\x00LN\x00\x12\x30\x12LT\xD0\x01LT\x30\x00\xFF”, this command set will display “Hello user” at (10,16) position, then draw line from (0,18) to (48,18) to (0,19) to (48,19). and end with “\xFF”. the hex value of \xFF is 255 in dec.

Write command set to flash(FLMWRal...d...)

Use regular write data to flash function to save command font in flash, if flash chip installed onboard, all 2MB memory

can be used for command set, also in , if no flash chip installed, the 16KB memory which used for welcome screen and user fonts can be used for command set also, the welcome screen is a real command set.

40ms delay is needed after each 64 bytes of data send out, refer “Write data to flash” command.

Arduino lib function: `flashWrite(address,length,*data);`

Run command set(FLMCSa)

Command: **FLMCS**, follow by 3 bytes of address which indicate the beginning of command set, 3 bytes address format allow the module access all 2MB memory in flash chip.

Read data from communication port

When you issued commands which need return data, the data can be read from the same communication port.

On UART mode, the returned data use same Baud rate and setting, if the data returned as bulk, the master need to poll the new data on the port continuously or use interrupt when new data received, there is no hand shake signal between master and slave side.

On I2C module, the module will pull the clock line to low when data not ready, then release it, the master will clock the data out then. The hardware I2C port do this handshake automatically, if the master side use software I2C, check the clock line for release first.

On SPI mode, there is no hand shake in general, we use data out line (SDO) on the module (data in (SDI) on the master) as hand shake line, when the data is not ready on module, the module keep the SDO low, and pull the SDO to high when data ready, the master can check this line (SDI on master side), if the line high, pull the SS line to low, and wait at least 10us, then shift 8bit data out. See the chart flow on page 7.

Use EEPROM(V3.3)

There are 976 bytes of EEPROM memory could be accessed by user in firmware V3.3 and later, erase the EEPROM cell is not needed before writing.

Write data to EEPROM(WREPa...d...)

Command: **WREP**, follow by 2 bytes of address, 2 bytes of data length (MSB-LSB format), then the data.

Arduino lib function: `writeE2prom(address,length,*data);`


Read data from EEPROM(RDEPa)


Command: **RDEP**, follow by 2 bytes of address, 2 bytes of data length (MSB-LSB format), after these 8 bytes of command sent to the module, the master controller need to wait the data available on the communication port, read out all desired data from the port.

Arduino lib function: `readE2prom(address,length); read1();`

Use Flash

Use flash in MCU

If there is no flash chip installed onboard, you can use the 16KB flash, before , the 16KB internal flash only can be used for welcome screen and 4 user fonts, refer to “Flash Memory Map” on the beginning of this manual.

But since , you can use the 16KB flash memory for command set as well, user can't read out the data saved in the internal flash memory.

Use flash in flash chip

If the flash chip installed onboard, you can use the full 2MB~16MB flash chip to store welcome screen, user font, command set and user data, all data in flash chip can be read out. the flash in MCU become unusable.

Write data to flash(FLMWRal...d...)



This command applicable to internal 16KB flash or external flash chip.

Command: **FLMWR**, follow by 3 bytes of start address, 3 bytes of data length (MSB, LSB format), then the data. This command can write data to flash chip or internal flash memory.

Note: if write data to flash chip, you probably need to erase the desired memory first before writing, but you don't need to erase the memory in internal flash.

A delay is needed after each 64 bytes of data send to module, when writing to internal flash, the module is writing every 64 bytes as bulk, and during the writing time, all new coming data from the communication port is lost. When writing to flash chip, the delay also needed due to the flash chip is a serial device.

According to our test, a 40ms delay on each 64 bytes is working well for both internal and external flash memory writing.

Also Note: the module will send value 17 on the communication port when writing to flash chip done, the master controller can poll it and know when the writing done. But, writing to internal flash will not return this value.

Here is the sample code in C:

```
void flashWrite(unsigned long int addr, unsigned long int len, const unsigned char *data) {
    unsigned char c, b;
    unsigned long int i;
    write('F'); //write a byte to communication port
    write('L');
    write('M');
    write('W');
    write('R');
    write(addr >> 16);
    write(addr >> 8);
    write(addr);
    write(len >> 16);
    write(len >> 8);
    write(len);
    b = 0;
    for (i = 0; i < len; i++) {
        c = pgm_read_byte_near(data + i);
        write(c);
        if ((++b) == 64) {
            b = 0, delay(40); //delay 40ms
        }
    }
}

#ifdef FLASH_CHIP
    //check write memory done
    while (read1() != 17); //read a byte from communication port
#endif
}
```

Arduino lib function: flashWrite(address, length, *data);

Run command set(FLMCSa)



This command applicable to internal 16KB flash or external flash chip.

Command: **FLMCS**, follow by 3 bytes of address which indicate the beginning of command set, 3 bytes address format allow the module access all 2MB memory in flash chip.

Read data in flash chip(**FLMRDa**)

This command only applicable to external flash chip.

If the flash chip installed on the board, you can use it to save user data, and read the data when you need it.


Command: **FLMRD**, follow by 3 bytes of address, then 3 bytes of data length, all MSB format.


After this command issued, the master controller can read data from the communication port when data in module ready.

Arduino lib function: `flashReadStart(address,length); read1();`

Erase flash memory in flash chip(**FLMERa**)

This command only applicable to external flash chip.

Only writing data to flash chip need this command, this command can erase only specific range of address on all  color module. Because the erasing on the chip is operating as block, the module will save the useful data in the block to the RAM on screen panel, erase whole block, then restore the useful data back, so, you may see a block of screen at the left-bottom corner show some wild image, that is the data from the erased block.

In  module, there is not enough RAM to save data from flash chip block, so, all data in the block which the address fall into the desired address range will be erased.

Command: **FLMER**, follow by 3 bytes of starting address, then 3 bytes of length which want to be erased. MSB first.

Arduino lib function: `flashErase(address, length);`

Touch Panel

The touch screen controller onboard is TSC2046, which can control a resistive touch panel. The internal 12 bit A/D, also can be used to monitor a voltage (battery voltage), an analog input (0~2.5V) and the chip temperature.

Calibrate touch screen(TUCHC)

Even we already calibrated the touch screen before shipping out, you may still need to re-calibrate it after the module installed in chassis. Send command "TUCHC" to module will let it run calibrate function and save the alignment parameters in EEPROM.

Arduino lib function: `calibrateTouchScreen();`

Read touched coordinate(RPNXYW)

After module received this command, the module will waiting until the touch panel pressed down, and then send the touched position which mapped to screen pixel's coordinate back to master (x,y), always return 4 bytes, 2 integer value, x first then y.

Arduino lib function: `readTouchScreen(); read1();`

Read a click event(RPNXYC)

Similar with above function, but the module will return the coordinate data after touch panel released.

Arduino lib function: `readClick(); read1();`

Read touch panel instantly(RPNXYI), check screen pressed

The 2 of above function will drive the module frozen until the touch screen pressed, if you only want to check the touch screen pressed or not, this is the function for the software, it return a pair of out of range value of no press on touch screen.

You also can check a hardware signal on the module when screen pressed, there is a PENIRQ signal on the 9pin header, this signal will go low when screen pressed. This is the easiest way to quote the touch screen if there were a free I/O pin on your master controller.

Arduino lib function: none

Read voltage(RDBAT)

Connect a voltage on the Vbat pin on the 9pin header, then send command: **RDBAT** to module, the module will return 2 bytes of data of voltage on the Vbat pin, MSB format, the unit is mV, the range is 0~10,000. eg.: if the 2 bytes of value is: 18, 192, the voltage is: $18 \times 256 + 192 = 4800\text{mV}$, is 4.8V.

The input impedance is: 10kΩ

Hint: if the measured voltage is over 10V, a 2R voltage divider is needed.

Arduino lib function: `readBattery();`

Read analog(RDAUX)

Connect the analog to the AUX pin on the 9pin header, then use this command to read it, we didn't adjust the 2 bytes result here, the data range is 0~4095, and represent 0~2.5V.

Use this format to calculate the real voltage: $V = d \times 2.5 / 4096$. (d is the reading data)

Arduino lib function: `readAux();`

Read temperature(RDTMP)

This command read the temperature of the chip, the format to calculate the temperature is:


$T = (653 - (d * 2500 / 4096)) / 2.1$ °C, d is the reading data.

Note, the temperature on the chip may be affected by the backlight heat of LCD screen.


Arduino lib function: `readTemperature();`

Power management

Backlight brightness(BLd)

The backlight brightness on all color LCD modules  can be adjusted continuously by use command: BL, follow by a byte of value 0~100, 0 will turn backlight full off, and 100 will turn backlight full on.

The backlight on all OLED modules are not adjust-able.

The backlight on all monochrome LCD modules  can be turned on or off only currently.

Arduino lib function: `backLightBrightness(d); backLightOn(); backLightOff();`

Turn screen on/off(SOod)

Command: **SOO**, follow by a byte value 0/1, when d=0, the screen and the backlight will be turned off immediately, that will save much power on the module, this function work on all module.

On most of modules, the module only consume few mA after screen turned off.

The content on the screen unchanged if screen turn off then turn on later.

Arduino lib function: `screenOnOff(d);`

Turn MCU off(DNMCU)

Even the MCU will enter sleep mode when no pending commands in the receiver buffer, you may also want to turn the MCU into deep sleep mode manually.

Command: **DNMCU**, no following data needed, the module will check if there were more pending commands in buffer before entering sleep, if there were, the module will not enter sleep mode.

The module wake up automatically when new data received, but if the COM mode is I2C, some dummy data are needed to act as waking signal, so, use few `write(0)` then a delay 10ms is a good practice to wake up the MCU from deep sleep.

The screen will keep on, and all content on the screen unchanged when MCU off.

Arduino lib function: `cpuOff();`

Turn module off(DNALL)


This command put all power off: backlight off, screen off, MCU enter deep sleep, the module will only consume <0.05mA of current, the wake up sequence is same with wake up MCU, the module will restore backlight and put screen on also after wake up, the content on the screen unchanged.


Arduino lib function: `moduleOff();`

Setting and configuration

Start screen(welcome screen or splash screen)

Start screen is the first showing when power on, it is used to display your logo or information of the device, the start screen can be modified and disabled by user.

 On the firmware V3.0 and earlier, the start screen only the bitmap on all monochrome modules, if the screen is 128*64 pixels, the start screen only use $128 \times 64 / 8 = 1024$ bytes of in MCU flash memory.

But in the later, we developed color graphic OLED and LCD modules , the in MCU flash memory were not able to store whole color of start screen, at that time, we use command set to show start screen.

You can use our online tool to convert your picture to bitmap start screen here: http://www.digole.com/tools/PicturetoC_Hex_converter.php.

Enable/disable start screen(DSSd)

Command: **DSS**, if the following value is 0, the start screen is not show up on next power on.

Arduino lib function: `displayStartScreen(d);`

Configuration show on/off(DCd)

In default, the module will show start screen when power on, and also show the current COM mode after start screen showed up, that will tell you what is the Baud on UART mode or the slave address on I2C mode.

If you want to manage this configuration show on the screen, use command: **DC**, then follow by a byte value 0 or 1, if d=0, the configuration will not show on the screen on next power on.

Arduino lib function: `displayConfig(d);`

Download start screen to module(SSSI...d...)

Command: **SSS**, follow by 2 bytes of data length of start screen, then the data, as described before, the data structure are different for monochrome module and color module.

In V3.2 and earlier version on color module, the command set also need 2 bytes of data to indicate the command set length, when you downloading this format of start screen to module, 2 bytes of length follow to **SSS** to indicate the length of rest data, and in the rest of data, the first 2 bytes to indicate the length of command set, their relationship is: SSS (length+2) (length) (...data...).

Arduino lib function: `downloadStartScreen(length,*data);`

Change I2C address(SI2CAa)

When you connect multiple modules on a I2C bus, every slave modules MUST be assigned with different address, this function can change the default address of 0x27 to other value.

This command only work at I2C COM mode, you can't use it at UART or SPI mode.

Command: **SI2CA**, follow by a byte of new address. The module use the new address instantly, it also save this new address in internal memory, you don't need to change it on the next power recycle.

Arduino lib function: `setI2CAddress(a);`

Set SPI mode(SPIMD0~3)

There are 4 mode for SPI, based on the Clock polarity and phase, the default is mode 0 (except V3.3, was mode 2).
Command: **SPIMD**, follow a byte to indicate the new SPI mode, the module will use the new mode on next power up.
This command only available on firmware **V3.4 and later**.

Change UART baud(SBrate)

The module always on 9600 8N1 baud rate when power on if UART selected, and can be changed by using command: **SB**, the follow vary bytes of baud rate, the module accept 2 format of data:

- 1) unsigned long int: 4 fixed bytes, MSB first.
- 2) numbers: vary length.

eg.: set the baud rate to 115200: "SB\x00\x01\xC2\x00"—long int format, or "SB115200"—number format.

The module will not save the new baud rate in memory, so, you always need to start your mater circuit on 9600 rate first, then send the change baud rate command, then adjust the master's rate to new.

Arduino lib function: `DigoleSerialDisp mydisp(&Serial, rate);`

Config universal character LCD adapter(STCRcr\x80\xC0\x94\xD4)

Our universal character LCD adapter can work with different size of character LCDs: 0801,0802,0804, 1601, 1602, 1604, 2001, 2002, 2004, 4001 and 4002 if the LCD controller is KS0066U/F / HD44780 or compatible chip.

The default set is 1602, if your LCD is other than 1602, you need to use this command: **STCR**, follow by a byte for column and a byte for row, the rest of 4 bytes is fixed for KS0066U/F / HD44780 LCD controller.

eg.: for 2004 LCD: "STCR\x14\x04\x80\xC0\x94\xD4".

Arduino lib function: `setLCDColRow(c,r);`

Config universal graphic LCD adapter(SLCDx...)

In the new version of Universal GLCD adapter V2, the firmware can recognize the KS0108 and ST7920 by itself, you don't need to use the following command to set up now.

only for older version of PCB Our universal graphic LCD adapter only work with monochrome 128*64 GLCD with ST7920, KS0108 and ST7565 or compatible LCD controller.

Command: **SLCD**, follow by one or more bytes of data.

"SLCD0", set the adapter for ST7920, "SLCD2" for ST7565.

"SLCD1", "SLCD3" and "SLCD4xxx" are for KS0108, the reason of this is because the E and CS1, CS2 signal on KS0108 can be config High or Low active logical level on different GLCD.

"SLCD1" for E↓, CS1↓ and CS2↓, "SLCD3" for E↓, CS1↑ and CS2↑ KS0108 GLCD, and "SLCD4xxx" can config the E,CS1,CS2 independently, the "xxx" is for E,CS1,CS2 logical level, eg.: "SLCD4011" is same with "SLCD3" command.

The adapter will remember the last config even power off, you don't need to do this again after you seen the screen work with the adapter correctly.

Arduino lib function: `setLCDChip(x); //only for standard GLCD`

Adjust LCD contrast(CTx)

Command: **CT**, follow by a byte of value 0~100, this command only effective for 128*64 GLCD with ST7565 controller, The contrast on GLCD use KS0108 and ST7920 controller only be adjustable by a hardware pot.

Arduino lib function: `setContrast(x);`

Others

Delay a period(DLYx)

This command only available since V3.9, but this is a bug in V3.9: it will be halt if on I2C/SPI mode, fixed in V4.0.

Command: “**DLY**”, following with a byte of delay period, value 1 for about 0.25s.

Escape commands in V4.0

we embedded ESC commands set in the new version of firmware, that means, you can use letter commands and ESC commands in same time, use ESC commands may save the module some time to scan the received data, e.g.: Letter command: “TT” = ESC command: 27, 01, both are 2 bytes, but for longer letter commands, it will also save some program space for user.

All ESC command are 2 bytes, first byte must be value of 27, the second byte indicate the function, here is the cross reference, the number from TT is 1, and increased 1 after each commands, the last number is 59 for “DLY”:

"TT", "ETB", "ETP", "ETO", "ESC", "SI2CA", "SC", "SB", "SD", "SF",
"SSS", "SUF", "SOO", "SLP", "FR", "LN", "LT", "DC", "DIM", "DP",
"DR", "DSS", "DOUT", "TP", "BL", "TRT", "GP", "CL", "CC", "CS",
"CT", "MA", "MCD", "MDT", "DM", "EDIM", "FS", "WREP", "RDEP", "INV",
"DNALL", "DNMCU", "SLCD", "RPNXY", "TUCHC", "RDBAT", "RDAUX", "RDTMP", "FLMER", "FLMRD",
"FLMWR", "FLMCS", "BGC", "DWWIN", "RSTDW", "WINCL", "SPIMD", "FTOB", "DLY", "TRANS",
"VIDEO"

Old manual, for reference only

Digole Serial:UART/I2C/SPI Character/Graphic LCD/OLED Display Module User Manual

(last updated: Oct. 10th 2015)

Summery	33
What benefits you if using these products in you electronic projects?	33
FEATURES	33
How to set up the communication mode?	34
Protocols:	34
Set up universal graphic serial LCD adapter:	34
1) For ST7920 LCD controller:	34
2) For KS0108 controller:	34
3) For ST7565 controller:	34
Commands List	35
Character/Graphic Display Shared Command	35
Graph Display Command: for mono and color serial Graphic display only	36
Special Command to set universal 12864 adapter	39
New Command for firmware version 2.8 and up	39
Commands for touch screen panel	39
Commands for onboard flash memory	40
Communication Port	41
SD card and touch screen Port	41
FAQ	42

Summary

This manual will describe most common features for our Serial LCD/OLED displays and modules, each particular product may have different looks, size and material, but all interface to your master circuits and control commands are same, that means you can switch Digole Serial display in your application without any modification of your master circuit and software.

Our Serial display products are listed below, you can purchase them with lowest price at: <http://www.digole.com/index.php?categoryID=153>

What benefits you if using these products in your electronic projects?

- **Save lots of the I/O resources:** these products only need 1 to 3 I/O pins from your master controller that depends on the communication type you want.
- **Easy to use:** the commands sending to products are easy to remember and understand.

On Graphic serial products:

- **Save huge memory space** to store font and start screen on graphic display: in graphic product, there are 7 preloaded fonts ready for your application, and also have 16KB memory space for your user fonts, once you uploaded the start screen or user fonts, it will be stored in products.
- Using user fonts function, you can display any graphs or characters in any language
- These products already **integrated graphic functions** such as: draw line/rectangle/circle/image, send few bytes of instruction to products, it will do it for you, that also saves your lots of code space
- You can display contents in 4 different directions: 0°, 90°, 180°, 270°(clockwise) on same screen, the product will map the coordinate accordingly.

FEATURES

- | | |
|--|---|
| <ul style="list-style-type: none"> • Communication mode: UART/I2C/SPI, detect your setting automatically • Receiving buffer: 64/2048 bytes • Work with all microcontroller and microprocessor • Communication signal can work on 3.3V and 5.0V TTL • Default setting: UART baud 9600bps, I2C 0x27 address | <ul style="list-style-type: none"> • Low power consumption: less than 4mA (for adapter only, completed module may have higher depends on the backlight power consumption) • Simple command sets, easy to remember • Simple graphic engine integrated (Graphic Products) • 7 preloaded fonts, font's data structure full compatible with U8Glib(Graphic Products) • UART baud (bps): 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200 |
|--|---|

NOTE: for screen size ≥ 255 pixels, you need 2 bytes to present the position or size, first send value 255, then send the remainder, e.g. for position of:

200: send 200 only

255: send 255 then 0 ($255+0=255$)

400: send 255 then 145 ($255+145=400$)

the onboard MCU will check the first byte, if it is 255, it will wait the second byte, then add them together.

How to set up the communication mode?

There are 3 different communication modes on all products: UART, I2C and SPI, what you need is just use solder to short the I2C/SPI jumper on adapter and make it works at I2C or SPI, if both jumpers are open, it works at UART, you can find a similar jumper like this: on board.



PROTOCOLS:

- UART : 8-N-1, 8bits, No parity bit, 1 stop bit.
- I2C: Slave Mode, 7-bit address, default address is Hex:27, change able. This mode may give you a headache due to more signal options in I2C, but we make it works as standard, you just need setup your I2C on master controller as Standard Master Mode.
- SPI: 8-bits, MSB first, data on raise edge of SCK sampled; this is Standard setting on SPI too.

SET UP UNIVERSAL GRAPHIC SERIAL LCD ADAPTER:

1) For ST7920 LCD controller:

- Verify and compare the pinout on adapter and LCD panel, they should be same on order (refer to the picture bellow);
- Pull up PSB pin (usually pin# 15) to VCC to let the LCD panel work at parallel mode;
- Connect the adapter with the LCD panel, adjust the contrast pot, power up the adapter, you should see the welcome screen displayed.

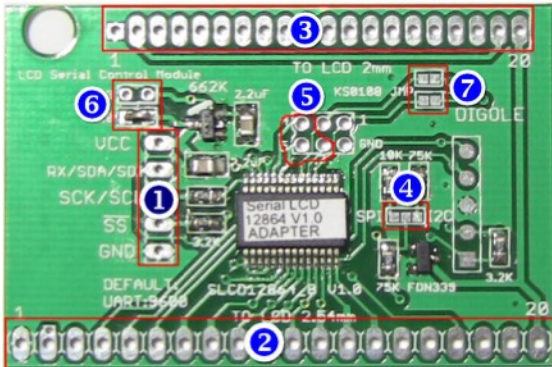
2) For KS0108 controller:

- Verify and compare the pinout on adapter and LCD panel, they should be same on order (refer to the picture bellow);
- Shorting the 2 KS0108 jumper: 7 on the picture;
- In most case you need add a 10K contrast pot by yourself;
- Connect the adapter with the LCD panel, power up the adapter, send chip configuration command (refer to above table) to adapter;
- Adjust the contrast pot, you should see the welcome screen displayed, if not, you probably need to try other value (1,3,4) in chip configuration command.

3) For ST7565 controller:

NOTE: There are vary of ST7565 LCD panels on market, and the pin out are vary from one module to others, this adapter only support SPI mode on the LCD panel, you usually need to connect pins on this adapter with LCD panel corresponding.

- Connect the pins on the adapter with LCD panel corresponding;
- Power up the adapter, then send chip configuration command to adapter ("SLCD2"), then you will see the welcome screen on the panel;
- You can use the set contrast command to adjust the contrast (ST7565 support software contrast adjustment).



1 Communication port to MCU
2 To LCD with 2.54mm
3 To LCD with 2.0mm
4 UART/I2C/SPI select jumper

Jumper:	I2c	SPI
UART	Open	Open
I2C	Short	Open
SPI	Open	Short

5 Digital out put @25mA
6 5V, 3.3V Power Supply to LCD jumper,
DON'T SHORT on BOTH
7 KS0108 Jumper
 Short both for KS0108

PIN CONFIGURATION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ST7920	Vss	Vdd	VO	RS	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	PSB	NC	RST	Vout	BLA	BLK
KS0108	GND	Vdd	VO	D/I	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	CS1	CS2	RST	Vout	A+	K-
ST7565	Vss	Vdd		D/C	CS	SCK	SD										RST		A+	K-

Commands List

CHARACTER/GRAPHIC DISPLAY SHARED COMMAND

(B-one byte, B...-Bytes, see note)

Command	Description	Arduino lib function	note	Char Display	Mono-Graph	Color-Graph
CL	C lear screen and set the display position to first Column and first Row (x=0.y=0), for graphic LCD, it also set the font to default and turn off the cursor. For color display with firmware V3.0 and later, this will fill the screen with background color.	clearScreen();	The module will not execute this command until other command received.	✓	✓	✓
CSB	set C ur S or on/off	enableCursor(); disableCursor();	B=0 off, B=1 on	✓	✓	✓
BLB	Set B ack L ight ON/OFF, B=0 or 1, 0: off, 1: on. TFT LCD module: B=0~100 set the brightness.	backLightOn(); backLightOff(); setBackLight(0~100);	unavailable on Character Adapter V1.x	✓	✓	✓
SOOB	Set Screen ON/OFF to save power B=0 or 1, 0:off, 1:on		For GLCD/OLED only	✓	✓	✓
DCB	D isplay C onfig on/off (Current communication setting). Module will display the config when next power on if COM mode changed even you turned it off.	displayConfig(0); displayConfig(1);	B=0 off, B=1 on	✓	✓	✓
SBB...	Set UART B aud, B are ASCII characters, the available values are: "300", "1200", "2400", "4800", "9600", "14400", "19200", "28800", "38400", "57600", "115200"	Set BAUD when initial the class	When adapter power up or reset, always start with 9600bps Baud rate	✓	✓	✓
SI2CAB	Set I2C Address, the default address is 0x27, the adapter will store the new address in memory	setI2CAddress(0x34);	Change address to 0x34	✓	✓	✓
STCRBB BBBB	Set T ext C olumns and R ows, this command will config your LCD if other than 1602 and the chip is other than KS0066U/F / HD44780	setLCDColRow(20,4);	The last 4 B should be "\x80\xC0\x94\xD4", it mapped the starting RAM address on LCD	✓		
TPBB	set T ext P osition for following display, BB are x and y	setPrintPos(x,y);	Only affect the following "TT" command	✓	✓	✓
TTB...	display T ex T string, the text will wrapped in next row if the current row fulfilled, the Text Position will be changed to the last char displayed, this command terminated by 0x00 received.	print(string); print(int); print(char); print(float); print(double); drawStr(x,y,string);	The print function in Arduino, can also print other data and format the out put.	✓	✓	✓
MCDB	M anual C omman D : send command B to display bypass the adapter	directCommand(0xaf);	Use it if you want to control the display directly	✓	✓	✓
MDTB	M anual D a T a: send data B to display bypass the adapter	directData(0x88);	Same as above	✓	✓	✓

GRAPH DISPLAY COMMAND: FOR MONO AND COLOR SERIAL GRAPHIC DISPLAY ONLY

Command	Description	Arduino lib function	note	Mono Graphic	Color Graphic
GPBB	set G raphic P osition for following draw line command, BB are x and y in byte	setPrintPos(x,y,1);	X,y=0 to 255	√	√
DMB	Set the D isplay M ode for on coming command, the available values for B are: 1) “C” as Copy, 2) “!” as NOT, 3) “ ” as OR, 4) “^” as XOR, 5) “&” as AND. These mean the next drawing pixel will logic operation with pixel already on screen.	setMode(‘!’);	Like the Bitwise Operator in C	√	√
DIMBBB BB...	D isplay I mage, 1st B is x position, 2nd is y, 3rd B is image width, 4th is height, then following data. Each byte present 8 pixels, if the image width not divide 8, the last byte of a row only contain few pixels, eg. For width of 9 to 16, you need 2 bytes for a row. <u>If this command is running from Flash, x,y value not needed, use current position instead.</u>	drawBitmap(x,y,width,height,*data);		√	√
SDB	S end graphic fuction D irection, the value of B is 0 to 3, represent 0 to 270 degree respectively.	setRotation(0); undoRotation(); setRot90(); setRot180(); setRot270();	The setRotation(); will accept 0 to 3 represent 0 to 270 degree respectively	√	√
CTB	Set display C on T rast, only for some models, Only for ST7565 LCD Controller	setContrast(30);	Only for ST7565 LCD Controller	√	
FRBBBB	Draw a F illed R ectangle, 4 B are: X,Y(left top), X,Y (right bottom). <u>If this command is running from Flash, x,y value not needed, will use current position instead, and follow by width and hight(not right-bottom position)</u>	drawBox(x,y,width,height);	In order to compatible with u8g, drawBox() in Arduino use width and height	√	√
DRBBBB	Draw a R ectangle, 4 B are: X,Y(left top), X,Y (right bottom). <u>If this command is running from Flash, x,y value not needed, use current position instead, and follow by width and hight(not right-bottom position)</u>	drawFrame(x,y,width,height);	drawFrame () in Arduino use width and height	√	√
CCBBBB	Draw a C ir C le, 4 B are: X,Y, radius, filled or not. <u>If this command is running from Flash, x,y value not needed, use current position instead.</u>	drawCircle(x,y,r,f); drawDisc(x,y,r);	f=1 means filled circel	√	√
DPBB	Draw a P ixel, 2 B are: x,y. The color was set up by commands of “SC” or “ESC”	drawPixel(x,y);		√	√
LNBBBB	Draw a Line from (x,y) to (x1,y1), 4 B are: x,y,x1,y1	drawLine(x,y,x1,y1); drawHLine(x,y,width); drawVLine(x,y,height);	drawHLine()-horizontal line drawVLine()-veritcal line	√	√
LTBB	Draw a L ine from T ast position to (x,y), 2 B are:x,y	drawLineTo(x,y);		√	√
TRT	Move text cursor to next line(call T ext R e T urn)	nextTextLine();	The y pixels moved depending on the font size current using	√	√

PIN	Description	PIN	Description
1	SDO: SPI mode only (Only for Touch and Flash Mem Modules)	2	GND (0V)
3	SS: SPI mode only chip select control in, low active	4	I2C and SPI mode: SCK/SCL: Clock in UART mode: TX(Only for Touch and Flash Mem Modules)
5	UART mode: RX I2C mode: SDA SPI mode: SDI	6	VCC: power supply 1.8V to 9V or 3.3V to 9V depends on the module

SFB or SFFBBB	Set Font, follow by the font number, preloaded font number is: 6,10,18,51,120,123,0(default), user font number is 200,201,202,203 maps to 4 user font memory sections, you can combine adjacent sections together is the font size >4kb(each section has 4kb in size). SFFBBB Set Font in Flash, only for Flash memory module, follow with 3 bytes of address in flash	setFont(0); setFlashFont((long int)0x20000);		✓	✓
SCB	Set Color for following display, this command affect all following drawing command, such as: text, line, circle, pixel, rectangle... 8 BIT color format: RRRGGGBB	setColor(1); setColor(0xE0);//Red	0 and 1 for black white screen 0 to 255 for color screen	✓	✓
BGC	set current color as background color. Only available on firmware V3.0 and later	setBgColor();			✓ (V3.0)
MABBB BBB	Move rectangle Area on screen to another place, the 6 B are represent: (x,y)(left- top),(w,h)(width-height), (xoffset,yoffset).	moveArea(x,y,w,h,xoff set,yoffset);		✓	✓
ETB	Enhanced set the current Text position Back to last char, this function will allow you display multiple chars at same position.	setTextPosBack();		✓	✓
ETOB	Enhanced set Text position Offset, the 2 B are xoffset then yoffset, it will adjust the text position in pixels	setTextPosOffset(xoffs et, yoffset);	0 to 255	✓	✓
ETPBB	Enhanced set Text Position as pixels on screen, the 2 B are x, y coordinate on screen	setTextPosAbs(x,y);	X,y=0 to 255	✓	✓
SSSBBB ...	Set Start Screen, 1st B is the High byte of data length, 2nd B is the Low byte of data length, following by data. For mono display, the start screen is bitmap data; For color display, the start screen is commands set, the first and second bytes are the commands length. N/A on Flash Memory module, use command set instead it.	uploadStartScreen(102 4, *data);	For mono display,the length of data should be: screen Width*High/ 8, eg. For 128x64 LCD, the length is 1024 bytes. For color module, the limitation is 2046 bytes	✓	✓

SUFBBB B...	Set User Font, 1st B is section of memory you want to upload, 2nd B is the lower byte of data length, 3rd B is the higher byte of data length, following by data	uploadUserFont(1,1434,*data);		✓	✓
DSSB	Display Start Screen stored in memory, also set up Automatic start screen display or not on next power up. N/A on Flash Memory module, use command set instead it.	displayStartScreen(1 or 0);	1= on, 0=off	✓	✓
DOU TB	Send a Byte to output head on board, the current driving ability for each pin is: 25mA (Sink/Source)	digitalOutput(0x1F);	The output head are vary from adapters	✓	✓
SLPB	Set Line Pattern when drawing line, only for new version firmware later than Jan. 2013. eg. B=0xAA is dot line, B=0xFA is dash line	setLinePattern(pattern);	Old version not support this function	✓	✓
EDIM1B BBBB...	Enhanced Display Image, 1 byte color format. The following 4 bytes are start position: x, y, image width, height, then following image data, each byte represent one pixel, the color format in a byte is:RRRGGGBB x,y value are not needed if running from flash memory, use current position instead	drawBitmap256(x,y,width,height, *data);			✓
EDIM2B BBBB...	Enhanced Display Image, 2 byte color format. The following 4 bytes are start position: x, y, image width, height, then following image data, each pixel occupy 2 bytes: B1:B0, the color structure: RRRRRGGG, GGGBBBBB, MSB first. x,y value are not needed if running from flash memory, use current position instead	drawBitmap65K(x,y,width,height, *data);	Available on firmware version 2.7 and higher		✓
EDIM3B BBBB...	Enhanced Display Image, 3 byte color format. The following 4 bytes are start position: x, y, image width, height, then following image data, 3 bytes represent one pixel: Red, Green, Blue, the validate value: 0 to 64 (6bits),that is:00RRRRRR, 00GGGGGG,00BBBBBB. x,y value are not needed if running from flash memory, use current position instead	drawBitmap262K(x,y,width,height, *data);			✓
ESCBBB	Enhanced Set Color for following display, the BBB are color Red, color Green and Color Blue, this command affect all following drawing command, such as: text, line, circle, pixel, rectangle...	setTrueColor(R,G,B);			✓
DWWIN BBBB	Define a Draw Window, follow by top-left position: X,Y, then the drawing window's Width and Height. NOT affect image display functions:DIM, EDIMx.	setDrawWindow(x,y,w,h);			✓ (V3.0)
RSTDW	ReSeT the Draw Window to full screen.	resetDrawWindow();			✓ (V3.0)
WINCL	Use current background color to clear the defined drawing window.	cleanDrawWindow();			✓ (V3.0)

SPECIAL COMMAND TO SET UNIVERSAL 12864 ADAPTER

Command	Description	Arduino lib function	note
SLCDB	Only for multi-chip driver adapter: B=0 or '0' for ST7920 B=1 or '1' for KS0108 ("E" Low, "CS1"&"CS2" Low) B=2 or '2' for ST7565 Since product after Apr. 20 2013: B=3 or '3' for KS0108 ("E" Low, "CS1"&"CS2" High) B=4 or '4' for KS0108 , follow by effective level for "E", "CS1" and "CS2", eg. "SLCD4011" is same as "SLCD3"	setLCDChip(chip_num);	For Universal Graphic Serial LCD Adapter only

NEW COMMAND FOR FIRMWARE VERSION 2.8 AND UP

Command	Description	Arduino lib function	note
DNMCU	Shut Down MCU to save energy, wake up by new commands received, in order to wake up MCU properly, send some dummy command (data 0) then wait few microseconds		
DNALL	Shut Down whole module (MCU & Display panel) to save energy, wake up by new commands received, in order to wake up MCU properly, send some dummy command (data 0) then wait few microseconds		

COMMANDS FOR TOUCH SCREEN PANEL

Command	Description	Arduino lib function	note
RPNXYI	Instant Read X, Y position when pen touching the panel, the output is 4 bytes data, first 2 bytes is X, second 2 bytes is Y, if pen is not pressed on touch screen , it will return 0xFFFF, 0xFFFF.		The return value of X,Y already mapped to pixels on screen according to the direction set (SD command)
RPNXYW	Waiting and Read X, Y position when pen touching the panel, the output is 4 bytes data, first 2 bytes is X, second 2 bytes is Y, the module will halt until pen pressed	readTouchScreen();	same as above
RPNXYC	Read a touchscreen click, it similar with "RPNXYW", but will wait till pen lift up	readClick();	same as above(V3.0)
TUHC	Calibrate the touch panel, the module will display 4 or5 cross line on the screen one by one, and waiting user the touch the dots, then map the touch panel to screen pixel on X, Y axes.	calibrateTouchScreen();	
RDBAT	Read battery voltage, the battery should connect on the Vbat pin on the module, out put is 2 byte integer. the result is millivolt. The maximum voltage measurable is 10V	readBattery();	
RDAUX	Using the internal 12bits A/D to read the voltage on AUX pin on the module, out put is 2 byte integer, data range is 000H~FFFH.	readAux();	

COMMANDS FOR ONBOARD FLASH MEMORY

The final commands set for flash memory might be changed in future.

The onboard flash memory could be 2M or 4M bytes, we organize whole space as block, the size of each block is 16K byte. You can store your data or Micro-command sets in that space.

Command	Description	Arduino lib function	note
FLMER BBB LLL	Erase the flash memory from BBB address till the length of LLL, this function will use LCD RAM to buffer the erased sector(4096 bytes) and then restore the datas un-want to be erased, you will see a band of massive pixels at the bottom of screen. If you just erase the whole sector(eg.: 0 to 4095), the module will not bother the LCD RAM	flashErase(address,length);	
FLMRD BB BLLL	Read LLL bytes of continious data from the beginning of address BBB in Flash Memory	flashReadStart(address,length) ;	
FLMWR BB BLLL	Write data to beginning address of BBB in Flash Memory with data length of LLL, once writing done, the module will send value 17 (XON) back to master controller. In order to make sure you don't drive the receiving buffer over flow(high speed at I2C/SPI mode, UART module always slower than onboard MCU), you better write the data length <2048 at once, and wait the XON response, otherwise, you need put some delay when sending data, if you need write more data, just repeat this command with different beginning address.	flashWriteL(address,length,*data); flashWrite(address,length,*data);	XON/XOFF is used for software flow control, the value of XON is 17, XOFF is 19
FLMCS BBB	Running the command set in the beginning of BBB in flash memory. Command set is a collector of most commands available on the module, but you can't put touch screen function, flash memory function and I2C address change function in it. In order to indicate the end of a command set, the last 3 bytes must be 255 (0xFF).	runCommandSet(address);	some commands running from flash are different than usual: the X/Y position not needed, use current position instead, this will give you more flexible to drawing same thing at different positions
SFF BBB	Use current font in flash, the BBB is the address in flash	setFlashFont(address);	

Communication Port

Pinout

SD CARD AND TOUCH SCREEN PORT

The SD card port is not available on all Digole Serial Display Module, please check for the special product for more details, The function of the pinout are already printed on the PCB, you can use this port to operate the SD card by yourself:

GND	Ground on the board	DO	Data Out from SD card
CLK	Clock signal to SD card	DI	Data In to SD card
CS	Chip Select signal to SD card, LOW activate	SD-Vcc	+ Power supplier to SD card
PENIRQ	Pen pressed down interrupt on touch screen, low activate	Vbat	Read battery voltage using onboard A/D
AUX	Convert the analog signal on this pin to digital using on board A/D (12bit)		

FAQ

Q: Some time, I can hear high pitch hum from module, why?

A: Your master controller has higher voltage than the display module, add 1K to 5.1K resistor series in the data line will reduce this noise (we already add series resistors in CLK and SS line).

Q: What is the maximum speed on communication port?

A: Depends on the mode you selected, UART baud: 115200, I2C: 200KHz, SPI:200KHz, also depends on the commands interpreted by the onboard MCU. Most of display module have 2K bytes of receiving buffer, just don't drive the buffer overflow.

Q: Can modules work with 5V TTL controller?

A: Yes, it can work with 3.3V and 5V TTL/CMOS controllers. Even with 1.8V system but not test comprehensively.

Q: What the difference between V2.9, V3.0 and V3.1 on color modules?

A: The V3.0 introduced 3 new commands to let you display contents in a rectangle area and background function: DWWIN—Define a **Dra**Wing **WIN**dow, RSTDW—**ReSeT** the **Dra**Wing window to full screen, WINCL—Use current background color to clear the defined drawing window.

In version 3.1, you can use LF (\n in C) and CR (\r in C) to control the text position.

Color display module Firmware version history:

V3.7B: Fixed “ETO” function which can't take negative value, and support WINBOND and SST flash chips

V3.6B: improved stability of touch screen function

V3.5: added B version which embedded a boot loader

V3.4:

Fixed SPI mode which was wrong in V3.3, and new “SPIMD” (set SPI mode) command introduced.

V3.3:

- 1) Change welcome screen to command set.
- 2) Use user font 200~203 when Flash chip on board is available.
- 3) Provide 976 bytes of EEPROM to user.
- 4) Set back ground color command changed.
- 5) Download user font command changed.

V3.2:

- 1) Touch screen calibrate function improved.
- 2) Drawing window automatically mapped according “set direction” function, user don't need send “set drawing window” after drawing direction changed.
- 3) Fixed a I2C bug when user reading data from module(touch screen and flash memory), now, the SCL pin will keep low when data is not ready in the module.(Clock Stretch)

V3.1:

- 1) The LF and CR characters are used to control text position now, not be used to terminate “TT” (print text) command now, user must send value of 0 (\x00) to terminate the “TT” command.

V3.0:

- 1) Introduced Drawing Window functions, user can set a partial screen area as drawing window, then all drawing (except draw picture) will out put in the drawing window, and the coordinate is refer to the drawing window, not to the whole screen.

Monochrome display module Firmware version history:

V2.8:

Added refresh screen command: FS, FS0 mean turn off instantly refresh screen, FS1 mean turn on it, FSx force to refresh screen, FSx should be used with FS0, because the screen will not be refreshed until FSx, if you sent FS0 previously.