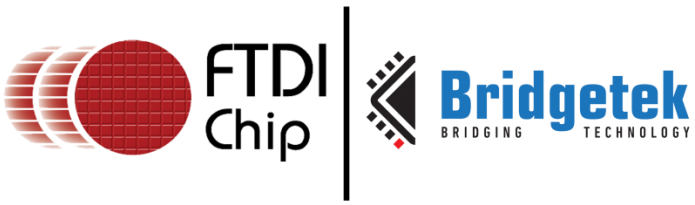


FT81x Series Programmers Guide

Version 1.1

Issue Date: 2016-09-19

This document is the programmers guide for the FT81X series chip (where x stands for any value of 0, 1, 2, 3). It describes the necessary information for programmers developing display, audio or touch applications with the FT81X (EVE) series devices.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

This document captures programming details for the FT81X series chips including graphics commands, widget commands and configurations to control FT81X series chips for smooth and vibrant screen effects.

The FT81X series chips are graphics controllers with add-on features such as audio playback and touch capabilities. They consist of a rich set of graphics objects (primitive and widgets) that can be used for displaying various menus and screen shots for a range of products including home appliances, toys, industrial machinery, home automation, elevators, and many more.

1.1 Overview

This document will be useful to understand the command set and demonstrate the ease of usage in the examples given for each specific instruction. In addition, it also covers various power modes, audio, and touch features as well as their usage.

Information on pin settings, hardware characteristic and hardware configurations can be found in the FT81X data sheet (DS_FT81X).

Within this document, the endianness of DL commands, co-processor engine commands, register values, data in RAM_G are in '*Little Endian*' format.

1.2 Scope

This document is targeted for software programmers and system designers to develop graphical user interface (GUI) applications on any system processor with an SPI master port.

1.3 API Reference Definitions

Functionality and nomenclature of the APIs used in this document.

wr8() – write 8 bits to intended address location

wr16() – write 16 bits to intended address location

wr32() – write 32 bits to intended address location

wr8s() – write 8 bits string to intended address location

rd8() – read 8 bits from intended address location

rd16() – read 16 bits from intended address location

rd32() – read 32 bits from intended address location

rd8s() – read 8 bits string from intended address location

cmd() – write 32 bits command to co-processor engine FIFO RAM_CMD

cmd_*() – Write 32 bits co-processor engine command with its necessary parameters to the co-processor engine FIFO (RAM_CMD).

dl() – Write 32 bits display list command to RAM_DL.

host_command() – send host command in host command protocol.

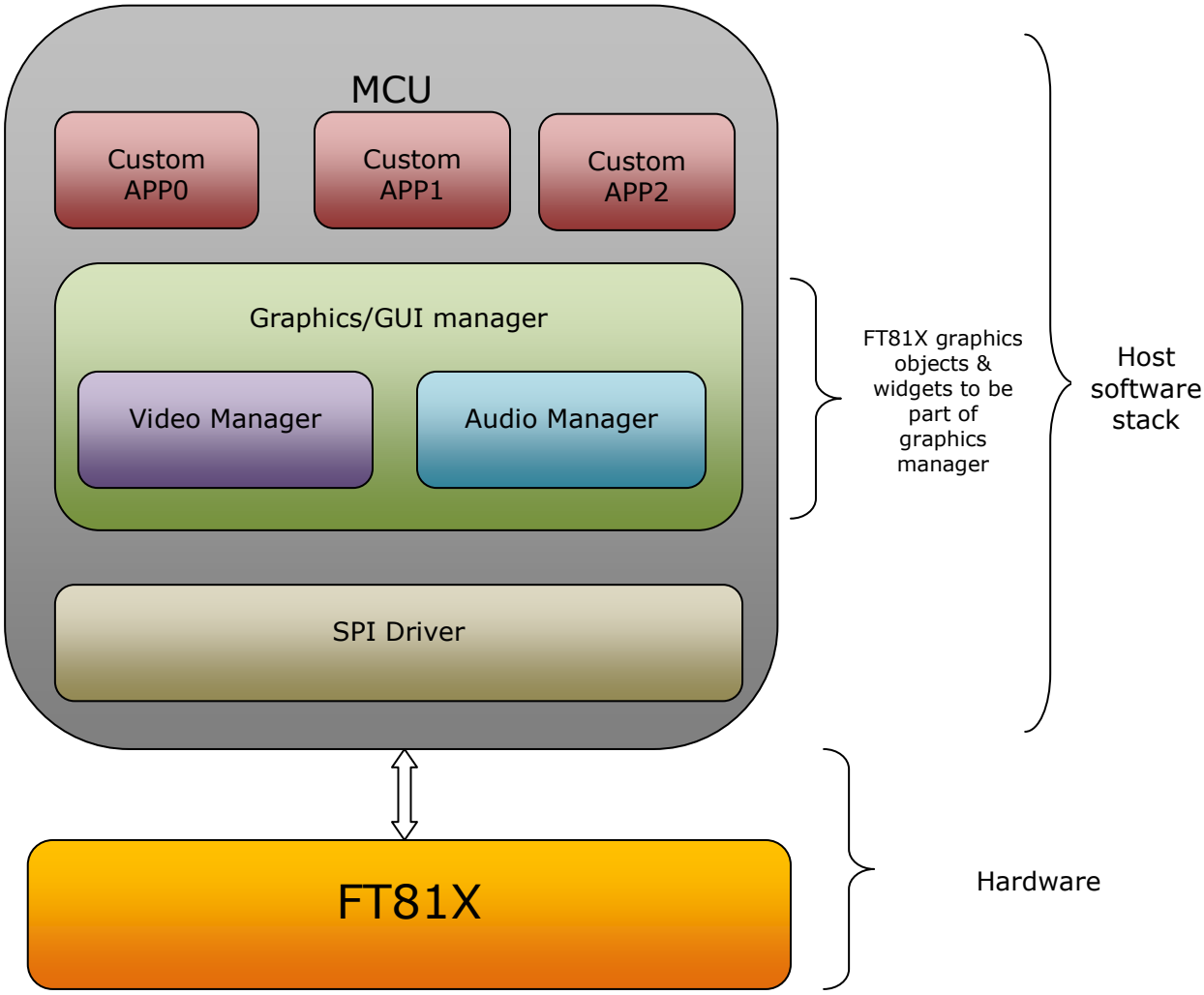
2 Programming Model

The FT81X appears to the host MCU as a memory-mapped SPI device. The host MCU sends commands and data over the serial protocol described in the data sheet.

2.1 General Software Architecture

The software architecture can be broadly classified into layers such as custom applications, graphics/GUI manager, video manger, audio manager, drivers etc. FT81X higher level graphics engine commands and co-processor engine widget commands are part of the graphics/GUI manager. Control & data paths of video and audio are part of video manager and audio manager. Communication between graphics/GUI manager and the hardware is via the SPI driver.

Typically the display screen shot is constructed by the custom application based on the framework exposed by the graphics/GUI manager.



2.2 Read Chip Identification Code

After reset or reboot, the chip ID can be read from address 0xC0000 to 0xC0003.

To read the chip identification code in the FT81X series chip, users are recommended to read 4 bytes of data from address 0xC0000 before the application overwrites this address, since it is located in RAM_G.

The following table describes data to be read:

Address	0xC0003	0xC0002	0xC0001	0xC0000
Data	0x00	0x01	0x10/0x12(FT810/FT812) 0x11/0x13(FT811/FT813)	0x08

2.3 Initialization Sequence

This section describes the initialization sequence in the different scenario.

- **Initialization Sequence during the boot up:**

1. Send Host command "CLKEXT" to FT81X, if an external clock is used.
2. Send Host command "ACTIVE" to enable the clock to the FT81X. FT81X starts its self-diagnosis process and may take up to 300ms. Alternatively, read REG_ID repeatedly until 0x7C is read.
3. Configure video timing registers, except REG_PCLK
4. Write first display list
5. Write REG_DLSWAP, FT81X swaps the display list immediately
6. Enable back light control for display
7. Write REG_PCLK, video output begins with the first display list
8. Use an MCU SPI clock of not more than 30MHz

```
MCU_SPI_CLK_Freq(<11MHz);//use the MCU SPI clock less than 11MHz

host_command(CLKEXT);//send command to "CLKEXT" to FT81X
host_command(ACTIVE);//send host command "ACTIVE" to FT81X


/* Configure display registers - demonstration for WQVGA resolution */
wr16(REG_HCYCLE, 548);
wr16(REG_HOFFSET, 43);
wr16(REG_HSYNC0, 0);
wr16(REG_HSYNC1, 41);
wr16(REG_VCYCLE, 292);
wr16(REG_VOFFSET, 12);
wr16(REG_VSYNC0, 0);
wr16(REG_VSYNC1, 10);
wr8(REG_SWIZZLE, 0);
wr8(REG_PCLK_POL, 1);
wr8(REG_CSPREAD, 1);
wr16(REG_HSIZE, 480);
wr16(REG_VSIZE, 272);

/* write first display list */
```

```
wr32(RAM_DL+0,CLEAR_COLOR_RGB(0,0,0));
wr32(RAM_DL+4,CLEAR(1,1,1));
wr32(RAM_DL+8,DISPLAY());

wr8(REG_DLSWAP,DLSWAP_FRAME);//display list swap

wr8(REG_GPIO_DIR,0x80 | rd8(REG_GPIO_DIR));
wr8(REG_GPIO,0x080 | rd8(REG_GPIO));//enable display bit

wr8(REG_PCLK,5);//after this display is visible on the LCD

MCU_SPI_CLK_Freq(<30Mhz);//use the MCU SPI clock upto 30MHz
```

Code snippet 1 Initialization sequence

- **Initialization Sequence from Power Down using PD_N pin:**

  1. Drive the PD_N pin high
  2. Wait for at least 20ms
  3. Execute ”Initialization Sequence during the Boot UP” from steps 1 to 9

- **Initialization Sequence from Sleep Mode:**

  1. Send the Host command “ACTIVE” to enable the clock to the FT81X
  2. Wait for at least 20ms
  3. Execute “Initialization Sequence during Boot Up” from steps 5 to 8

- **Initialization sequence from standby mode:**

  Execute all the steps mentioned in “Initialization Sequence from Sleep Mode” except waiting for at least 20ms in step 2.

  Note: Refer to the FT81X data sheet for more information.

2.4 Audio Routines

The FT81X audio engine has two functionalities: playback the audio data in RAM_G and synthesize the sound effect stored in ROM with selected pitches.

2.4.1 Sound Effect

The FT81X audio engine has various sound data built-in to work as a sound synthesizer.

Sample code to play C8 on the xylophone:

```
wr8(REG_VOL_SOUND,0xFF); //set the volume to maximum
wr16(REG_SOUND, (0x6C<< 8) | 0x41); // C8 MIDI note on xylophone
wr8(REG_PLAY, 1); // play the sound
```

Code snippet 2 Play C8 on the xylophone

Sample code to check the status of sound play:

```
Sound_status = rd8(REG_PLAY);//1-play is going on, 0-play has finished
```

Code snippet 3 Check the status of sound playing

Sample code to stop sound play:

```
wr16(REG_SOUND,0x0);//configure silence as sound to be played
wr8(REG_PLAY,1);//play sound
Sound_status = rd8(REG_PLAY);//1-play is going on, 0-play has finished
```

Code snippet 4 Stop playing sound

To avoid an audio pop sound on reset or power state change, trigger a "mute" sound, and wait for it to complete (completion of sound play is when REG_PLAY contains a value of 0). This sets the output value to 0 level. On reboot, the audio engine plays back the "unmute" sound to drive the output to the half way level.

Note: Refer to the FT81X data sheet for more information on the sound synthesizer and audio playback.

2.4.2 Audio Playback

The FT81X supports an audio playback feature. There are three types of audio format supported: 4 Bit IMA ADPCM, 8 Bit signed PCM, 8 Bit u-Law.

For IMA ADPCM format, please note the byte order: within one byte, the first sample (4 bits) shall be located from bit 0 to bit 3, while the second sample (4 bits) shall be located from bit 4 to bit 7.

For the audio data in the FT81X RAM_G to play, the FT81X requires the start address in REG_PLAYBACK_START to be 64 bit (8 Bytes) aligned. In addition, the length of audio data specified by REG_PLAYBACK_LENGTH is required to be 64 bit (8 Bytes) aligned.

To learn how to play back the audio data, please check the sample code below:

```
wr8(REG_VOL_PB,0xFF);//configure audio playback volume
wr32(REG_PLAYBACK_START,0);//configure audio buffer starting address
wr32(REG_PLAYBACK_LENGTH,100*1024);//configure audio buffer length
wr16(REG_PLAYBACK_FREQ,44100);//configure audio sampling frequency
wr8(REG_PLAYBACK_FORMAT,ULAW_SAMPLES);//configure audio format
wr8(REG_PLAYBACK_LOOP,0);//configure once or continuous playback
wr8(REG_PLAYBACK_PLAY,1);//start the audio playback
```

Code snippet 5 Audio playback

```
AudioPlay_Status = rd8(REG_PLAYBACK_PLAY);//1-audio playback is going on,
0-audio playback has finished
```

Code snippet 6 Check the status of audio playback

```
wr32(REG_PLAYBACK_LENGTH,0);//configure the playback length to 0
wr8(REG_PLAYBACK_PLAY,1);//start audio playback
```

Code snippet 7 Stop the audio playback

2.5 Graphics Routines

This section describes graphics features and captures a few examples.

2.5.1 Getting Started

This short example creates a screen with the text "FTDI" on it, with a red dot.

Figure 2: Getting Started Example

The code to draw the screen is:

```
wr32(RAM_DL + 0, CLEAR(1, 1, 1));            // clear screen
wr32(RAM_DL + 4, BEGIN(BITMAPS));            // start drawing bitmaps
wr32(RAM_DL + 8, VERTEX2II(220, 110, 31, 'F')); // ascii F in font 31
wr32(RAM_DL + 12, VERTEX2II(244, 110, 31, 'T')); // ascii T
wr32(RAM_DL + 16, VERTEX2II(270, 110, 31, 'D')); // ascii D
wr32(RAM_DL + 20, VERTEX2II(299, 110, 31, 'I')); // ascii I
wr32(RAM_DL + 24, END());
wr32(RAM_DL + 28, COLOR_RGB(160, 22, 22)); // change colour to red
wr32(RAM_DL + 32, POINT_SIZE(320)); // set point size to 20 pixels in
radius
wr32(RAM_DL + 36, BEGIN(POINTS)); // start drawing points
wr32(RAM_DL + 40, VERTEX2II(192, 133, 0, 0)); // red point
wr32(RAM_DL + 44, END());
wr32(RAM_DL + 48, DISPLAY()); // display the image
```

Code snippet 8 Getting Started

After the above drawing commands are loaded into display list RAM, register REG_DLSWAP is required to be set to 0x02 in order to make the new display list active on the next frame refresh.
Note:

- The display list always starts at address RAM_DL
- The address always increments by 4 bytes as each command is 32 bits wide.
- Command CLEAR is recommended to be used before any other drawing operation, in order to put the FT81X graphics engine in a known state.

- The end of the display list is always flagged with the command DISPLAY

2.5.2 Coordinate Plane

The valid X and Y coordinate ranges from -16384 to 16383 in units of single pixel precision.

The figure below illustrates the graphics coordinate plane and its visible area:

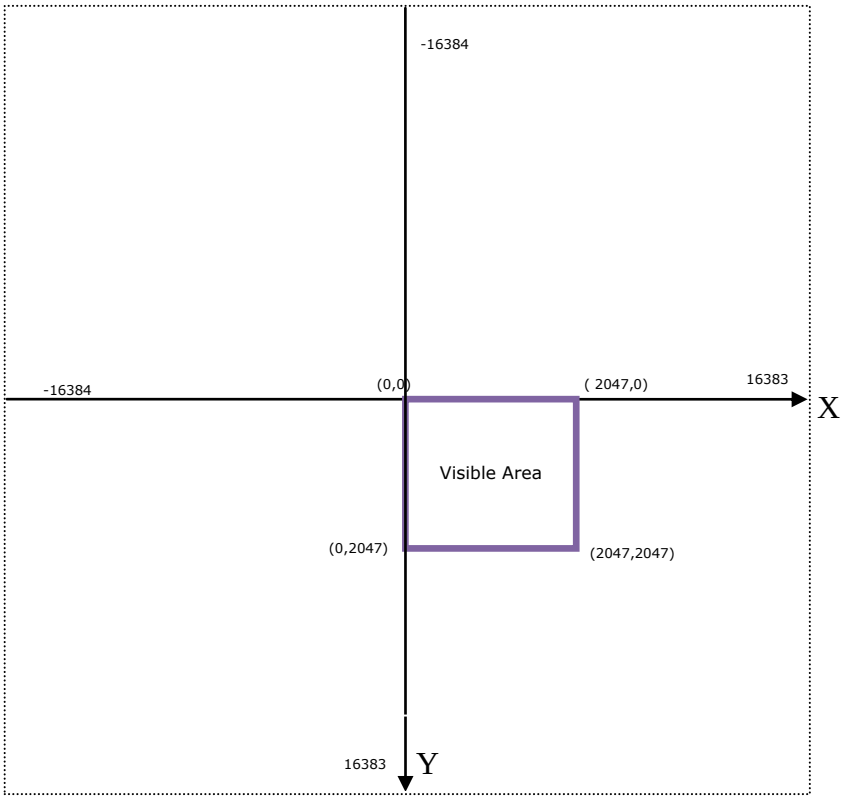


Figure 3: Coordinate plane in units of single pixel precision

The below figure shows the coordinate plane and visible area in units of 1/8 pixel precision:

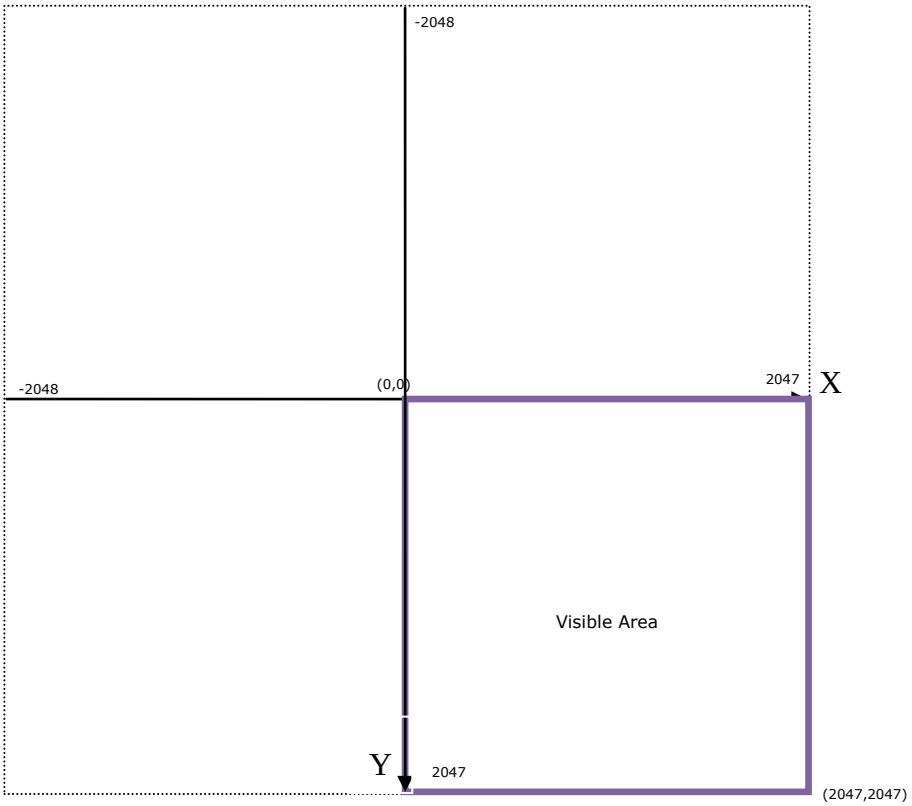


Figure 4: Coordinate plane in units of 1/8 pixel precision

VERTEX2F and **VERTEX_FORMAT** are the commands which enable the drawing operation to reach the coordinate plane.

2.5.3 Screen Rotation

REG_ROTATE controls the screen orientation. Changing the value of the register immediately causes the orientation of the screen to change. In addition, the coordinate system is also changed accordingly so that all the display commands and co-processor commands works in the rotated coordinate system.

NOTE: The touch transformation matrix is not affected by setting REG_ROTATE.

To adjust the touch screen accordingly, users are recommended to use CMD_SETROTATE as opposed to setting REG_ROTATE.

REG_ROTATE = 0 is the default landscape orientation:

REG_ROTATE = 1 is inverted landscape:

REG_ROTATE = 2 is portrait:

REG_ROTATE = 3 is inverted portrait:

REG_ROTATE = 4 is mirrored landscape:

REG_ROTATE = 5 is mirrored inverted landscape:

REG_ROTATE = 6 is mirrored portrait:

REG_ROTATE = 7 is mirrored inverted portrait:

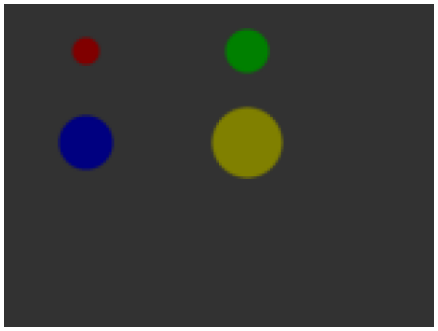
2.5.4 Drawing Pattern

The general pattern for drawing is driven by display list commands:

- **BEGIN** with one of the primitive types
- Input one or more vertices using "**VERTEX2II**" or "**VERTEX2F**", which specify the placement of the primitive on the screen
- **END** to mark the end of the primitive.

Examples

Draw points with varying radius from 5 pixels to 13 pixels with different colors:



```
dl( COLOR_RGB(128, 0, 0) );
dl( POINT_SIZE(5 * 16) );
dl( BEGIN(POINTS) );
dl( VERTEX2F(30 * 16,17 * 16) );
dl( COLOR_RGB(0, 128, 0) );
dl( POINT_SIZE(8 * 16) );
dl( VERTEX2F(90 * 16, 17 * 16) );
dl( COLOR_RGB(0, 0, 128) );
dl( POINT_SIZE(10 * 16) );
dl( VERTEX2F(30 * 16, 51 * 16) );
dl( COLOR_RGB(128, 128, 0) );
dl( POINT_SIZE(13 * 16) );
dl( VERTEX2F(90 * 16, 51 * 16) );
```

The VERTEX2F command gives the location of the circle center.

Draw lines with varying sizes from 2 pixels to 6 pixels with different colors (line width size is from center of the line to the boundary):

```
dl( COLOR_RGB(128, 0, 0) );
dl( LINE_WIDTH(2 * 16) );
dl( BEGIN(LINES) );
dl( VERTEX2F(30 * 16,38 * 16) );
dl( VERTEX2F(30 * 16,63 * 16) );
dl( COLOR_RGB(0, 128, 0) );
dl( LINE_WIDTH(4 * 16) );
dl( VERTEX2F(60 * 16,25 * 16) );
dl( VERTEX2F(60 * 16,63 * 16) );
dl( COLOR_RGB(128, 128, 0) );
dl( LINE_WIDTH(6 * 16) );
dl( VERTEX2F(90 * 16, 13 * 16) );
dl( VERTEX2F(90 * 16, 63 * 16) );
```

The VERTEX2F commands are in pairs to define the start and finish point of the line.

Draw rectangles with sizes of 5x25, 10x38 and 15x50 dimensions (line width size is used for corner curvature, LINE_WIDTH pixels are added in both directions in addition to the rectangle dimension):

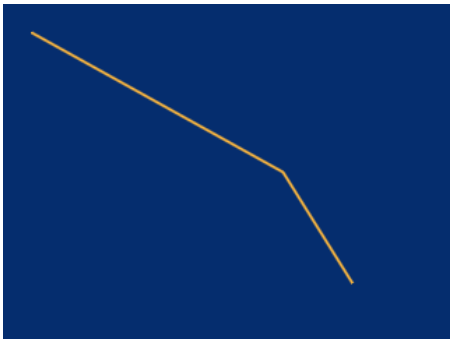
```
dl( COLOR_RGB(128, 0, 0) );
dl( LINE_WIDTH(1 * 16) );
```

```
dl( BEGIN(RECTS) );
dl( VERTEX2F(28 * 16,38 * 16) );
```

```
dl( VERTEX2F(33 * 16,63 * 16) );
dl( COLOR_RGB(0, 128, 0) );
dl( LINE_WIDTH(5 * 16) );
dl( VERTEX2F(50 * 16,25 * 16) );
dl( VERTEX2F(60 * 16,63 * 16) );
dl( COLOR_RGB(128, 128, 0) );
dl( LINE_WIDTH(10 * 16) );
dl( VERTEX2F(83 * 16, 13 * 16) );
dl( VERTEX2F(98 * 16, 63 * 16) );
```

The VERTEX2F commands are in pairs to define the top left and bottom right corners of the rectangle.

Draw line strips for sets of coordinates:



```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(LINE_STRIP) );
dl( VERTEX2F(5 * 16,5 * 16) );
dl( VERTEX2F(50 * 16,30 * 16) );
dl( VERTEX2F(63 * 16,50 * 16) );
```

Draw Edge strips for above:

```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(EDGE_STRIP_A) );
dl( VERTEX2F(5 * 16,5 * 16) );
dl( VERTEX2F(50 * 16,30 * 16) );
dl( VERTEX2F(63 * 16,50 * 16) );
```

Draw Edge strips for below:

```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(EDGE_STRIP_B) );
dl( VERTEX2F(5 * 16,5 * 16) );
dl( VERTEX2F(50 * 16,30 * 16) );
dl( VERTEX2F(63 * 16,50 * 16) );
```

Draw Edge strips for right:

```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(EDGE_STRIP_R) );
dl( VERTEX2F(5 * 16,5 * 16) );
dl( VERTEX2F(50 * 16,30 * 16) );
dl( VERTEX2F(63 * 16,50 * 16) );
```

Draw Edge strips for left:

```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(EDGE_STRIP_L) );
dl( VERTEX2F(5 * 16,5 * 16) );
dl( VERTEX2F(50 * 16,30 * 16) );
dl( VERTEX2F(63 * 16,50 * 16) );
```

2.5.5 Bitmap Transformation Matrix

To achieve the bitmap transformation, the bitmap transform matrix below is specified in the FT81X and denoted as *m*

$$m = \begin{bmatrix} BITMAP\_TRANSFORM\_A & BITMAP\_TRANSFORM\_B & BITMAP\_TRANSFORM\_C \\ BITMAP\_TRANSFORM\_D & BITMAP\_TRANSFORM\_E & BITMAP\_TRANSFORM\_F \end{bmatrix}$$

by default $m = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix}$, it is named as the **identity matrix**.

The coordinates x', y' after transforming is calculated in the following equation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = m \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

i.e.:

$$x' = x * A + y * B + C$$
$$y' = x * D + y * E + F$$

where A,B,C,E,D,E,F stands for the values assigned by commands BITMAP_TRANSFORM_A-F.

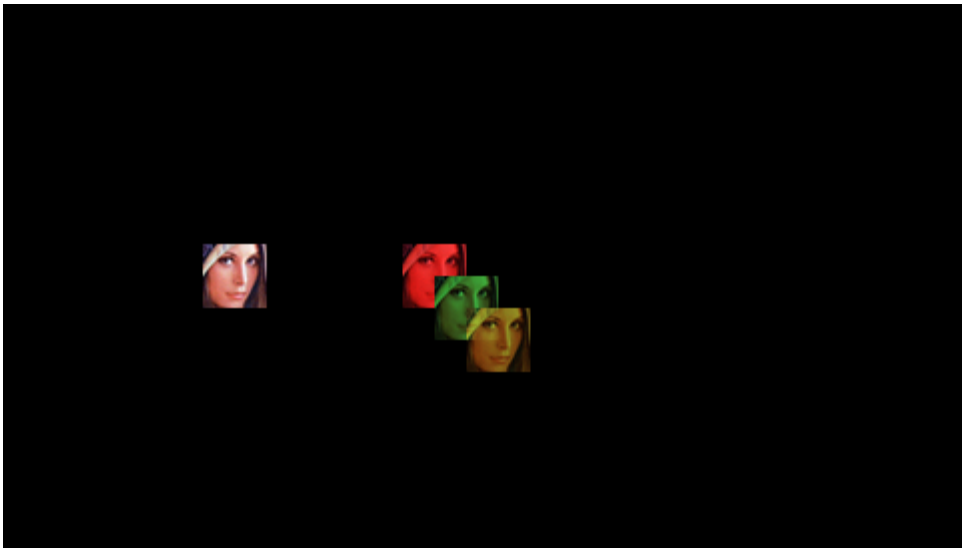
2.5.6 Color and Transparency

The same bitmap can be drawn in more places on the screen, in different colors and transparency:

```
dl(COLOR_RGB(255, 64, 64)); // red at (200, 120)
dl(VERTEX2II(200, 120, 0, 0));
dl(COLOR_RGB(64, 180, 64)); // green at (216, 136)
dl(VERTEX2II(216, 136, 0, 0));
dl(COLOR_RGB(255, 255, 64)); // transparent yellow at (232, 152)
dl(COLOR_A(150));
dl(VERTEX2II(232, 152, 0, 0));
```

Code snippet 9 color and transparency

The COLOR_RGB command changes the current drawing color, which colors the bitmap. The COLOR_A command changes the current drawing alpha, changing the transparency of the drawing: an alpha of 0 means fully transparent and an alpha of 255 is fully opaque. Here a value of 150 gives a partially transparent effect.



2.5.7 Performance

The graphics engine has no frame buffer: it uses dynamic compositing to build up each display line during scan out. Because of this, there is a finite amount of time available to draw each line. This time depends on the scan out parameters (decided by REG_PCLK and REG_HCYCLE) but is never less than 2048 internal clock cycles. FT81X's internal clock runs at 60MHz.

Some performance limits:

- The display list length must be less than 2048 instructions, because the graphics engine fetches display list commands one per clock.
- The usual performance of rending pixels is 16 pixels per clock
- For some bitmap formats, the drawing rate is 8 pixels per clock. These are TEXT8X8, TEXTVGA and PALETTED4444/565.
- For bilinear filtered pixels, the drawing rate is reduced to ¼ pixels per clock.

To summarize:

Table 1 Bitmap rendering performance

Filter Mode	Format	Rate
Nearest	TEXT8X8, TEXTVGA and PALETTED4444/565	8 pixel per clock
Nearest	all other formats	16 pixel per clock
BILINEAR	TEXT8X8, TEXTVGA and PALETTED4444/565	2 pixel per clock
BILINEAR	all other formats	4 pixel per clock

3 Register Description

In this chapter, all the registers in the FT81X are classified into 6 groups:

- Graphics Engine Registers,
- Audio Engine Registers,
- Touch Engine Registers,
- Co-processor Engine Registers,
- Special Registers
- Miscellaneous Registers.

The detailed definition for each register is listed in this chapter.

All the reserved bits shall be always zero. All the values prefixed with *0x* are hexadecimal.

All the offset of registers are based on the address RAM_REG (0x302000).

3.1 Graphics Engine Registers

Register Definition 1 REG_PCLK Definition

REG_PCLK Definition	
Reserved	R/W
31 8	7 0

Offset: 0x70 **Reset Value: 0x0**

Bit 7 - 0 : These bits are set to divide the main clock for PCLK. If the typical main clock was 60MHz and the value of these bits are 5, the PCLK will be 12 MHz. If the value of these bits are zero, there will be no PCLK output.
Bit 31 - 8: Reserved

Register Definition 2 REG_PCLK_POL Definition

REG_PCLK_POL Definition	
Reserved	R/W
31 0	

Address: 0x6C **Reset Value: 0x0**

Bit 0 : This bit controls the polarity of PCLK. If it is set to zero, PCLK polarity is on the rising edge. If it is set to one, PCLK polarity is on the falling edge.
Bit 31 - 1: Reserved

Register Definition 3 REG_CSPREAD Definition

REG_CSPREAD Definition	
Reserved	R/W
31 1 0	

Offset: 0x68 **Reset Value: 0x1**

Bit 0 : This bit controls the transition of RGB signals with PCLK active clock edge, which helps reduce the system noise . When it is zero, all the color signals are updated at the same time. When it is one, all the color signals timing are adjusted slightly so that fewer signal changes simultaneously.
Bit 31 - 1: Reserved.

Register Definition 4 REG_SWIZZLE Definition

REG_SWIZZLE Definition	
Reserved	R/W
31 4	3 0

Offset: 0x64 **Reset Value: 0x0**

Bit 3 - 0 : These bits are set to control the arrangement of output RGB pins, which may help support different LCD panel. Please check data sheet for the definition of values in this field.
Bit 31 - 4: Reserved

Register Definition 5 REG_DITHER Definition

REG_DITHER Definition	
Reserved	R/W
31 1	0

Offset: 0x60 **Reset Value: 0x1**

Bit 0 : Set to 1 to enable dithering feature on RGB signals output. Set to 0 to disable dithering feature. Reading 1 from this bit means dithering feature is enabled. Reading 0 from this bit means dithering feature is disabled.
Bit 31 - 1: Reserved
Note: The value is 0x01 after reset. Please refer to data sheet for details

Register Definition 6 REG_OUTBITS Definition

REG_OUTBITS Definition	
Reserved	R/W
31 9	8 0

Reset Value: 0x1B6(FT810/1)

Offset: 0x5C **0x0 (FT812/3)**

Bit 8 - 0: These 9 bits are split into 3 groups for Red, Green and Blue color output signals:

Bit 2 - 0: Blue color signal lines number. Reset value is 6.
Bit 5 - 3: Green Color signal lines number. Reset value is 6.
Bit 8 - 6: Red Color signal lines number. Reset value is 6.

Host can write these bits to control the numbers of output signals for each color.
Bit 31 - 9: Reserved
Note: Value 000 stands for 8 signal lines.

Register Definition 7 REG_ROTATE Definition

REG_ROTATE Definition	
Reserved	R/W
31 3	2 0

Offset: 0x58 **Reset Value: 0x00**

Bit 2~0: screen rotation control bits.
000: Default landscape orientation
001: Inverted landscape orientation
010: Portrait orientation
011: Inverted portrait orientation
100: Mirrored landscape orientation
101: Mirrored invert landscape orientation
110: Mirrored portrait orientation
111: Mirrored inverted portrait orientation
Bit 31 ~ 3: Reserved.

Note: Setting this register will NOT affect touch transform matrix.

Register Definition 8 REG_VSYNC1 Definition

REG_VSYNC1 Definition	
	R/W
31 12	11 0

Offset: 0x50 **Reset Value: 0x00A**

Bit 11 - 0: The value of these bits specifies how many lines for signal VSYNC takes at the start of new frame.
Bit 31 - 12: Reserved

Register Definition 9 REG_VSYNC0 Definition

REG_VSYNC0 Definition	
	R/W
31 12	11 0

Offset: 0x4C **Reset Value: 0x000**

Bit 11 - 0: The value of these bits specifies how many lines for the high state of signal VSYNC takes at the start of new frame.
Bit 31 - 12: Reserved

Register Definition 10 REG_VSIZE Definition

REG_VSIZE Definition	
Reserved	R/W
31 12	11 0

Offset: 0x48 **Reset Value: 0x110**

Bit 11 - 0: The value of these bits specifies how many lines of pixels in one frame.
The valid range is from 0 to 2048
Bit 31 - 12: Reserved

Register Definition 11 REG_VOFFSET Definition

REG_VOFFSET Definition	
Reserved	R/W
31 12	11 0

Offset: 0x44 **Reset Value: 0x00C**

Bit 11 - 0: The value of these bits specifies how many lines taken after the start of new frame.
Bit 31 - 12: Reserved

Register Definition 12 REG_VCYCLE Definition

REG_VCYCLE Definition	
Reserved	R/W
31 12	11 0

Offset: 0x40 **Reset Value: 0x124**

Bit 11 - 0: The value of these bits specifies how many lines in one frame.
Bit 31 - 12: Reserved

Register Definition 13 REG_HSYNC1 Definition

REG_HSYNC1 Definition	
Reserved	R/W
31 12	11 0

Offset: 0x3C **Reset Value: 0x029**

Bit 11 - 0: The value of these bits specifies how many PCLK cycles for HSYNC during start of line.
Bit 31 - 12: Reserved

Register Definition 14 REG_HSYNC0 Definition

REG_HSYNC0 Definition	
Reserved	R/W
31 12	11 0

Offset: 0x38 **Reset Value: 0x0**

Bit 11 - 0: The value of these bits specifies how many PCLK cycles of HSYNC high state during start of line.
Bit 31 - 12: Reserved

Register Definition 15 REG_HSIZE Definition

REG_HSIZE Definition	
Reserved	R/W
31 12	11 0

Offset: 0x34 **Reset Value: 0x1E0**

Bit 11 - 0: These bits are used to specify the numbers of PCLK cycles per horizonal line.
Bit 31 - 12: Reserved

Register Definition 16 REG_HOFFSET Definition

REG_HOFFSET Definition	
Reserved	R/W
31 12	11 0

Offset: 0x30 **Reset Value: 0x2B**

Bit 11 - 0: These bits are used to specify the numbers of PCLK cycles before pixels are scanned out.
Bit 31 - 12: Reserved

Register Definition 17 REG_HCYCLE

REG_HCYCLE Definition	
Reserved	R/W
31 12	11 0

Offset: 0x2C **Reset Value: 0x224**

Bit 11 - 0: These bits are the number of total PCLK cycles per horizontal line scan. The default value is 548 and supposed to support 480x272 screen resolution display. Please check the display panel specification for more details.
Bit 31 - 12: Reserved

Note: NONE

Register Definition 18 REG_DLSWAP Definition

REG_DLSWAP Definition	
Reserved	R/W
31 2	1 0

Offset: 0x54 **Reset Value: 0x00**

Bit 1 - 0: These bits can be set by the host to validate the display list buffer . The graphics engine will determine when to render the screen , depending on what values of these bits are set:

01: Graphics engine will render the screen immediately after current line is scanned out. It may cause tearing effect.

10: Graphics engine will render the screen immediately after current frame is scanned out. This is recommended in most of cases.

00: Do not write this value into this register.

11: Do not write this value into this register.

These bits can be also be read by the host to check the availability of the display list buffer. If the value is read as zero, the display list buffer is safe and ready to write. Otherwise, the host needs to wait till it becomes zero.

Bit 31 - 2: Reserved

Register Definition 19 REG_TAG Definition

REG_TAG Definition	
Reserved	R/O
31 8	7 0

Offset: 0x7C **Reset Value: 0x0**

Bit 7 - 0 : These bits are updated with tag value by FT81X graphics engine. The tag value here is corresponding to the touching point coordinator given in REG_TAG_X and REG_TAG_Y. Host can read this register to check which graphics object is touched.

Bit 31 - 8: Reserved

Note: Please note the difference between REG_TAG and REG_TOUCH_TAG. REG_TAG is updated based on the X,Y given by REG_TAG_X and REG_TAG_Y. However, REG_TOUCH_TAG is updated based on the current touching point given by FT81X touch engine.

Register Definition 20 REG_TAG_Y Definition

REG_TAG_Y Definition	
Reserved	R/W
31 11	10 0

Offset: 0x78 **Reset Value: 0x0**

Bit 10 - 0 : These bits are set by the host as Y coordinate of touching point, which will enable the host to query the tag value. This register shall be used together with REG_TAG_X and REG_TAG. Normally, in the case the host has already captured the touching point's coordinate, this register can be updated to query the tag value of respective touching point.
Bit 31 - 11: Reserved

Register Definition 21 REG_TAG_X Definition

REG_TAG_X Definition	
Reserved	R/W
31 11	10 0

Offset: 0x74 **Reset Value: 0x0**

Bit 10 - 0 : These bits are set by the host as X coordinate of touching point, which will enable host to query the tag value. This register shall be used together with REG_TAG_Y and REG_TAG. Normally, in the case the host has already captured the touching point's coordinate, this register can be updated to query the tag value of the respective touching point.
Bit 31 - 11: Reserved

3.2 Audio Engine Registers

Register Definition 22 REG_PLAY Definition

REG_PLAY Definition	
Reserved	R/W
31 1	0

Offset: 0x8C **Reset Value: 0x0**

Bit 0 : A write to this bit triggers the play of the synthesized sound effect specified in REG_SOUND.
Reading value 1 in this bit means the sound effect is playing. To stop the sound effect, the host needs to select the silence sound effect by setting up REG_SOUND and set this register to play.
Bit 31 - 1: Reserved

Register Definition 23 REG_SOUND Definition

REG_SOUND Definition	
Reserved	R/W
31 16	15 0

Offset: 0x88 **Reset Value: 0x0000**

Bit 0 - 15 : These bits are used to select the synthesized sound effect. They are split into two group Bit 7 - 0, Bit 15 - 8.

Bit 7 - 0 : These bits define the sound effect. Some of them are pitch adjustable and the pitch is defined in Bits 8 - 15. Some of them are not pitch adjustable and the Bits 8 - 15 will be ignored.

Bit 15 - 8: The MIDI note for the sound effect defined in Bits 0 - 7.

Note: Please refer to the datasheet sector "Sound Synthesizer" for the details of this register.

Register Definition 24 REG_VOL_SOUND Definition

REG_VOL_SOUND Definition	
Reserved	R/W

31 8 7 0

Offset: 0x84 **Reset Value: 0xFF**

Bit 7 - 0: These bits control the volume of the synthesizer sound. The default value 0xFF is highest volume. The value zero means mute.
Bit 31 - 8: Reserved

Register Definition 25 REG_VOL_PB Definition

REG_VOL_PB Definition	
Reserved	R/W

31 8 7 0

Offset: 0x80 **Reset Value: 0xFF**

Bit 7 - 0 : These bits control the volume of the audio file playback. The default value 0xFF is highest volume. The value zero means mute.
Bit 31 - 8 : Reserved

Register Definition 26 REG_PLAYBACK_PLAY Definition

REG_PLAYBCK_PLAY Definition	
Reserved	R/W

31 1 0

Offset: 0xCC **Reset Value: 0x0**

Bit 0 : A write to this bit triggers the start of audio playback, regardless of writing '0' or '1'. It will read back '1' when playback is ongoing, and '0' when playback completes.
Bit 31 - 1: Reserved
Note: Please refer to the datasheet section "Audio Playback" for the details of this register.

Register Definition 27 REG_PLAYBACK_LOOP Definition

REG_PLAYBACK_LOOP Definition	
Reserved	R/W

31 1 0

Offset: 0xC8 **Reset Value: 0x0**

Bit 0 : this bit controls the audio engine to play back the audio data in RAM_G from the start address once it consumes all the data. A value of 1 means LOOP is enabled, a value of 0 means LOOP is disabled.
Bit 31 - 1: Reserved
Note: Please refer to the datasheet section "Audio Playback" for the details of this register.

Register Definition 28 REG_PLAYBACK_FORMAT Definition

REG_PLAYBACK_FORMAT Definition	
Reserved	R/W

31 2 1 0

Offset: 0xC4 **Reset Value: 0x0**

Bit 1 - 0 : These bits define the format of the audio data in RAM_G. FT81X supports:

00: Linear Sample format
01: uLaw Sample format
10: 4 bit IMA ADPCM Sample format
11: Undefined.

Bit 31 - 2: Reserved

Note: Please read the datasheet section "Audio Playback" for more details.

Register Definition 29 REG_PLAYBACK_FREQ Definition

REG_PLAYBACK_FREQ Definition	
Reserved	R/O
31 16	15 0

Offset: 0xC0 **Reset Value: 0x1F40 (8000)**

Bit 15 - 0 : These bits specify the sampling fequency of audio playback data. Units is in Hz.
Bit 31 - 16: Reserved

Note: Please read the datasheet for more details.

Register Definition 30 REG_PLAYBACK_READPTR Definition

REG_PLAYBACK_READPTR Definition	
Reserved	R/O
31 20	19 0

Offset: 0xBC **Reset Value: 0x00000**

Bit 19 - 0 : These bits are updated by audio engine while playing audio data from RAM_G. It is the current audio data address which is playing back. The host can read this register to check if the audio engine has consumed all the audio data.
Bit 31 - 20 : Reserved
Note: Please read the datasheet section "Audio Playback" for more details.

Register Definition 31 REG_PLAYBACK_LENGTH Definition

REG_PLAYBACK_LENGTH Definition	
Reserved	R/W
31 20	19 0

Offset: 0xB8 **Reset Value: 0x00000**

Bit 19 - 0: These bits specify the length of audio data in RAM_G to playback, starting from the address specified in REG_PLAYBACK_START register.
Bit 31 - 20: Reserved

Note: Please read the datasheet section "Audio Playback" for more details.

Register Definition 32 REG_PLAYBACK_START Definition

REG_PLAYBACK_START Definition	
Reserved	R/W
31 20	19 0

Offset: 0xB4 **Reset Value: 0x00000**

Bit 19 - 0 : These bits specify the start address of audio data in RAM_G to playback.
Bit 31 - 20: Reserved

Note: Please read the datasheet section "Audio Playback" for more details.

3.3 Touch Screen Engine Registers

3.3.1 Overview

FT81X series chips support both resistive touch (FT810 and FT812) and capacitive touch (FT811 and FT813) functionality by two newly-integrated touch screen engines, i.e. Resistive Touch Engine(RTE) and Capacitive Touch Engine(CTE). Readers need to refer to the corresponding chapters below for their chip touch control.

3.3.2 Common Registers

This chapter describes the common registers which are effective to both RTE and CTE.

Table 2 common registers summary

Address	Register Name	Description
0x302150 - 0x302164	REG_TOUCH_TRANSFORM_A~F	Transform coefficient matrix
0x302168	REG_TOUCH_CONFIG	Configuration register

Register Definition 33 REG_TOUCH_CONFIG Definition

REG_TOUCH_CONFIG Definition																
Reserved									R/W							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15
14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		

Offset: 0x168 **Reset Value: 0x8381 (RTE) or 0x0381 (CTE)**

Bit 15 : Working mode of touch engine.
0: capacitive 1: resistive
Bit 14 - 13: Reserved
Bit 12: ignore short-circuit
Bit 11: enable low-power mode
Bit 10 - 4: I2C address of touch screen module.
Bit 3: This bit determines the vendor of capacitive touch screen.
0: FocalTech 1: Azoteq
Bit 2: Suppress 300ms startup
Bit 1 - 0: sampler clocks

Bit 31 - 16: Reserved

Register Definition 34 REG_TOUCH_TRANSFORM_F Definition

REG_TOUCH_TRANSFORM_F Definition																
R/W																
31	30															0

Offset: 0x164 **Reset Value: 0x0**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +0.0 after reset.

Register Definition 35 REG_TOUCH_TRANSFORM_E Definition

REG_TOUCH_TRANSFORM_E Definition		
R/W		
31 30	16 15	0

Offset: 0x160 **Reset Value: 0x10000**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +1.0 after reset.

Register Definition 36 REG_TOUCH_TRANSFORM_D Definition

REG_TOUCH_TRANSFORM_D Definition			
R/W			
31 30		16 15	0

Offset: 0x15C **Reset Value: 0x0**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +0.0 after reset.

Register Definition 37 REG_TOUCH_TRANSFORM_C Definition

REG_TOUCH_TRANSFORM_C Definition			
R/W			
31	30	16 15	0

Offset: 0x158 **Reset Value: 0x0**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +0.0 after reset.

Register Definition 38 REG_TOUCH_TRANSFORM_B Definition

REG_TOUCH_TRANSFORM_B Definition			
R/W			
31	30	16 15	0

Offset: 0x154 **Reset Value: 0x0**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +0.0 after reset.

Register Definition 39 REG_TOUCH_TRANSFORM_A Definition

REG_TOUCH_TRANSFORM_A Definition		
R/W		
31 30		16 15 0

Offset: 0x150 **Reset Value: 0x10000**

Bit 15 - 0 : The value of these bits represents the fractional part of the fixed point number.

Bit 30 - 16 : The value of these bits represents the integer part of the fixed point number.

Bit 31 : The sign bit for fixed point number

Note: This register represents fixed point number and the default value is +1.0 after reset.

3.3.3 Resistive Touch Engine(FT810/2)

All the registers available in RTE are almost identical to FT800, except its address.

Table 3 RTE registers summary

Address	Register Name	Description
0x302104	REG_TOUCH_MODE	Touch screen sampling Mode
0x302108	REG_TOUCH_ADC_MODE	Select ADC working mode
0x30210C	REG_TOUCH_CHARGE	Touch screen charge time, unit of 6 clocks
0x302110	REG_TOUCH_SETTLE	Touch screen settle time, unit of 6 clocks
0x302114	REG_TOUCH_OVERSAMPLE	Touch screen oversample factor
0x302118	REG_TOUCH_RZTHRESH	Touch screen resistance threshold
0x30211C	REG_TOUCH_RAW_XY	Touch screen raw x,y(16,16)
0x302120	REG_TOUCH_RZ	Touch screen resistance
0x302124	REG_TOUCH_SCREEN_XY	Touch screen x,y(16,16)
0x302128	REG_TOUCH_TAG	Touch screen Tag result

Register Definition 40 REG_TOUCH_TAG Definition

REG_TOUCH_TAG Definition	
RESERVED	RO

31 8 7 0

Offset: 0x12C **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. These bits are updated once when all the lines of the current frame are scanned out to the screen.
Bit 31 - 8: These bits are reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default.

Register Definition 41 REG_TOUCH_TAG_XY Definition

REG_TOUCH_TAG_XY Definition	
RO	RO
31 16	15 0

Offset: 0x128 **Reset Value: 0**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: Host can read this register to check the coordinates used by the touch engine to update the tag register REG_TOUCH_TAG.

Register Definition 42 REG_TOUCH_SCREEN_XY Definition

REG_TOUCH_SCREEN_XY Definition	
RO	RO
31 16	15 0

Offset: 0x124 **Reset Value: 0x80008000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen. After doing calibration, it shall be within the height of the screen size. If the touch screen is not being touched, it shall be 0x8000.
Bit 31 - 16: The value of these bits are the X coordinates of the touch screen. After doing calibration, it shall be within the width of the screen size. If the touch screen is not being touched, it shall be 0x8000.

Note: This register is the final computation output of the touch engine.

Register Definition 43 REG_TOUCH_DIRECT_Z1Z2 Definition

REG_TOUCH_DIRECT_Z1Z2 Definition			
Reserved	RO	Reserved	RO
31 26	25 16	15 10	9 0

Offset: 0x190 **Reset Value: NA**

Bit 9 - 0 : The 10 bit ADC value for touch screen resistance Z2.
Bit 15 - 10: Reserved
Bit 25 - 16: The 10 bit ADC value for touch screen resistance Z1.
Bit 31 - 26: Reserved
Note: To know it is touched or not, please check the 31st bit of REG_TOUCH_DIRECT_XY. Touch engine will do the post-processing for these Z1 and Z2 values and update the result in REG_TOUCH_RZ.

Register Definition 44 REG_TOUCH_DIRECT_XY

REG_TOUCH_DIRECT_XY Definition				
RO	Reserved	RO	Reserved	RO
31	30 26	25 16	15 10	9 0

Offset: 0x18C **Reset Value: 0x0**

Bit 9 - 0 : The 10 bit ADC value for Y coordinate
Bit 15 - 10: Reserved
Bit 16 - 25: The 10 bit ADC value for X coordinate.
Bit 30 - 26: Reserved
Bit 31 : If this bit is zero, it means a touch is being sensed and the two fields above contain the sensed data. If this bit is one, it means no touch is being sensed and the data in the two fields above shall be ignored.

Register Definition 45 REG_TOUCH_RZ Definition

REG_TOUCH_RZ Definition	
Reserved	RO
31 16	15 0

Offset: 0x120 **Reset Value: 0x7FFF**

Bit 15 - 0 : These bits are the resistance of touching on the touch screen . The valid value is from 0 to 0x7FFF. The highest value(0x7FFF) means no touch and the lowest value (0) menas the maximum pressure.
Bit 31 - 16: Reserved

Register Definition 46 REG_TOUCH_RAW_XY Definition

REG_TOUCH_RAW_XY Definition	
RO	RO
31 16	15 0

Offset: 0x11C **Reset Value: 0xFFFFFFFF**

Bit 15 - 0 : These bits are the raw Y coordinates of the touch screen before going through a transformation matrix. The valid range is from 0 to 1023. If there is no touch on screen, the value shall be 0xFFFF.
Bit 31 - 16: These bits are the raw X coordinates going through a transformation matrix. The valid range is from 0 to 1023. If there is no touch on screen, the value shall be 0xFFFF.
Note: The coordinates in this register have not mapped into the screen coordinates. To get the screen coordinates, please refer to REG_TOUCH_SCREEN_XY .

Register Definition 47 REG_TOUCH_RZTHRESH Definition

REG_TOUCH_RZTHRESH Definition	
Reserved	R/W

31 16 15 0

Offset: 0x118 **Reset Value: 0xFFFF**

Bit 15 - 0 : These bits control the touch screen resistance threshold. The host can adjust the touch screen touching sensitivity by setting this register. The default value after reset is 0xFFFF and it means the lightest touch will be accepted by the RTE. The host can set this register by doing experiments. The typical value is 1200.
Bit 31 - 16: Reserved

Register Definition 48 REG_TOUCH_OVERSAMPLE Definition

REG_TOUCH_OVERSAMPLE Definition	
Reserved	R/W

31 4 3 0

Offset: 0x114 **Reset Value: 0x7**

Bit 3 -0 : These bits control the touch screen oversample factor. The higher value of this register causes more accuracy with more power consumption, but may not be necessary. The valid range is from 1 to 15.
Bit 31 - 4: Reserved

Register Definition 49 REG_TOUCH_SETTLE Definition

REG_TOUCH_SETTLE Definition	
Reserved	R/W

31 4 3 0

Offset: 0x110 **Reset Value: 0x3**

Bit 3 - 0 : These bits control the touch screen settle time , in the unit of 6 clocks. The default value is 3, meaning the settle time is 18 (3*6) system clock cycles.
Bit 31 - 4: Reserved

Register Definition 50 REG_TOUCH_CHARGE Definition

REG_TOUCH_CHARGE Definition	
Reserved	R/W

31 16 15 0

Offset: 0x10C **Reset Value: 0x1770**

Bit 15 - 0 : These bits control the touch-screen charge time, in the unit of 6 system clocks. The default value after reset is 6000, i.e. the charge time will be 6000*6 clock cycles.
Bit 31 - 16: Reserved

Register Definition 51 REG_TOUCH_ADC_MODE Definition

REG_TOUCH_ADC_MODE Definition	
Reserved	R/W
31 1	0

Offset: 0x108 **Reset Value: 0x1**

Bit 0 : The host can set this bit to control the ADC sampling mode of the FT800, as per:

0: Single Ended mode. It causes lower power consumption but with less accuracy.
1: Differential Mode. It causes higher power consumption but with more accuracy.

The default mode after reset.

Bit 31 - 1: Reserved

Register Definition 52 REG_TOUCH_MODE Definition

REG_TOUCH_MODE Definition	
Reserved	R/W
31 2	1 0

Offset: 0x104 **Reset Value: 0x3**

Bit 1 - 0 : The host can set these two bits to control the touch screen sampling mode of the FT800 touch engine, as per:

00: Off mode. No sampling happens.
01: Single mode. Cause one single sample to occur.
10: Frame mode. Cause a sample at the start of each frame.
11: Continuous mode. Up to 1000 times per seconds. Default mode after reset.

Bit 31 - 2: Reserved

3.3.4 Capacitive Touch Engine(FT811/3)

Capacitive Touch Engine(CTE) is built in with the following features:

- I^2C interface to Capacitive Touch Panel Module(CTPM)
- Detects up to 5 touch points at the same time
- Supports CTPM with Focaltech FT5x06 series or Azotech IQS5xx series drive chips
- Compatibility(single touch) mode and Extended mode(multi-touch)

After reset or boot up, CTE works in compatibility mode and only one touch point is detected. In extended mode, it can detect up to 5 touch points simultaneously.

CTE makes use of the same registers set REG_TOUCH_TRANSFORM_A~F to transform the raw coordinates to a calibrated screen coordinate, regardless of whether it is in compatibility mode or extended mode.

Note: The *calibration process of the touch screen should only be performed in compatibility mode.*

Table 4 CTE registers summary

Address	Register Name	Description
0x302104	REG_CTOUCH_MODE	Touch screen sampling Mode
0x302108	REG_CTOUCH_EXTENDED	Select ADC working mode
0x30211C	REG_CTOUCH_TOUCH1_XY	Coordinate of second touch point
0x302120	REG_CTOUCH_TOUCH4_Y	Y coordinate of fifth touch point
0x302124	REG_CTOUCH_TOUCH_XY	Coordinate of first touch point
0x302128	REG_CTOUCH_TAG_XY	coordinate used to calculate the tag of first touch point
0x30212C	REG_CTOUCH_TAG	Touch screen Tag result of fist touch point
0x302130	REG_CTOUCH_TAG1_XY	XY used to tag of second touch point
0x302134	REG_CTOUCH_TAG1	Tag result of second touch point
0x302138	REG_CTOUCH_TAG2_XY	XY used to tag of third touch point
0x30213C	REG_CTOUCH_TAG2	Tag result of third touch point
0x302140	REG_CTOUCH_TAG3_XY	XY used to tag of fourth

Address	Register Name	Description
		touch point
0x302144	REG_CTOUCH_TAG3	Tag result of fourth touch point
0x302148	REG_CTOUCH_TAG4_XY	XY used to tag of fifth touch point
0x30214C	REG_CTOUCH_TAG4	Tag result of fifth touch point
0x30216C	REG_CTOUCH_TOUCH4_X	X coordinate of fifth touch point
0x30218C	REG_CTOUCH_TOUCH2_XY	Third touch point coordinate
0x302190	REG_CTOUCH_TOUCH3_XY	Fourth touch point coordinate

The following tables define the registers that CTE provide:

Register Definition 53 REG_CTOUCH_MODE Definition

Reserved	R/W

31 2 1 0

Offset: 0x104 **Reset Value: 0x3**

Bit 1 - 0 : The host can set these two bits to control the touch screen sampling mode of the touch engine, as per:

00: Off mode. No sampling happens.
01: Not defined.
10: Not defined.
11: On Mode.

Bit 31 - 2: Reserved

Register Definition 54 REG_CTOUCH_EXTENDED Definition

REG_CTOUCH_EXTEND Definition	
Reserved	R/W
31 1	0

Offset: 0x108 **Reset Value: 0x1**

Bit 0 : This bit controls the detection mode of the touch engine, as per:
- 0: Extended mode, multi-touch detection mode, up to 5 touch points
- 1: Compability mode, single touch detection mode

Bit 31 - 1: Reserved

Register Definition 55 REG_CTOUCH_TOUCH_XY Definition

REG_CTOUCH_TOUCH_XY Definition	
RO	RO
31 16	15 0

Offset: 0x124 **Reset Value: 0x80008000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the first touch point.
Bit 31 - 16: The value of these bits are X coordinates of the first touch point.

Note: This register is applicable for extended mode and compability mode. For compability mode,it reflects the position of the only touch point.

Register Definition 56 REG_CTOUCH_TOUCH1_XY Definition

REG_CTOUCH_TOUCH1_XY Definition	
RO	RO
31 16	15 0

Offset: 0x11C **Reset Value: 0x80008000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the second touch point.
Bit 31 - 16: The value of these bits are X coordinates of the second touch point.
Note: This register is only applicable in the extended mode

Register Definition 57 REG_CTOUCH_TOUCH2_XY Definition

REG_CTOUCH_TOUCH2_XY Definition	
RO	RO
31 16	15 0

Address: 0x18C **Reset Value: 0x80008000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the third touch point.
Bit 31 - 16: The value of these bits are X coordinates of the third touch point.

Note: This register is only applicable in the extended mode

Register Definition 58 REG_CTOUCH_TOUCH3_XY Definition

REG_CTOUCH_TOUCH3_XY Definition	
RO	RO
31 16	15 0

Offset: 0x190 **Reset Value: 0x80008000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the fourth touch point.
Bit 31 - 16: The value of these bits are X coordinates of the fourth touch point.

Note: This register is only applicable in the extended mode

Register Definition 59 REG_CTOUCH_TOUCH4_X Definition

REG_CTOUCH_TOUCH4_X Definition	
RO	
15	0

Offset: 0x16C **Reset Value: 0x8000**

Bit 15 - 0 : The value of these bits are the X coordinates of the fifth touch point.

Note: This register is only applicable in the extended mode. This is a 16 bit register

Register Definition 60 REG_CTOUCH_TOUCH4_Y Definition

REG_CTOUCH_TOUCH4_Y Definition	
RO	
15	0

Offset: 0x120 **Reset Value: 0x8000**

Bit 15 - 0 : The value of these bits are the Y coordinates of the fifth touch point.

Note: This register is only applicable in the extended mode. This is a 16 bit register

Register Definition 61 REG_CTOUCH_RAW_XY Definition

REG_CTOUCH_RAW_XY Definition			
RO		RO	
31	16	15	0

Address: 0x11C **Reset Value: 0xFFFFFFFF**

Bit 15 - 0 : The value of these bits are the Y coordinates of a touch point before going through the transform matrix
Bit 31 - 16: The value of these bits are the X coordinates of a touch point before going through the transform matrix
Note: This register is only available in compatibility mode

Register Definition 62 REG_CTOUCH_TAG Definition

REG_CTOUCH_TAG Definition	
RESERVED	RO

31 8 7 0

Offset: 0x12C **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. These bits are updated once when all the lines of the current frame are scanned out to the screen.It works in both extended mode and compatibility mode. In extended mode, it is the tag of the first touch point , i.e., the tag value mapping to the coordinate in REG_CTOUCH_TAG_XY

Bit 31 - 8:Reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default. In extended mode, it refers to the first touch point

Register Definition 63 REG_CTOUCH_TAG1 Definition

REG_CTOUCH_TAG1 Definition	
RESERVED	RO

31 8 7 0

Offset: 0x134 **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the second touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.
Bit 31 - 8: Reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default.

Register Definition 64 REG_CTOUCH_TAG2 Definition

REG_CTOUCH_TAG2 Definition	
RESERVED	RO
31 8	7 0

Offset: 0x13C **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the third touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.
Bit 31 - 8: These bits are reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default.

Register Definition 65 REG_CTOUCH_TAG3 Definition

REG_CTOUCH_TAG3 Definition	
RESERVED	RO

31 8 7 0

Offset: 0x144 **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the fourth touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.
Bit 31 - 8: Reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default.

Register Definition 66 REG_CTOUCH_TAG4 Definition

REG_CTOUCH_TAG4 Definition	
RESERVED	RO
31 8	7 0

Offset: 0x14C **Reset Value: 0**

Bit 7 - 0 : These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the fifth touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.
Bit 31 - 8:Reserved.

Note: The valid tag value range is from 1 to 255 ,therefore the default value of this register is zero, meaning there is no touch by default.

Register Definition 67 REG_CTOUCH_TAG_XY Definition

REG_CTOUCH_TAG_XY Definition	
RO	RO
31 16	15 0

Offset: 0x128 **Reset Value: 0**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG.

Register Definition 68 REG_CTOUCH_TAG1_XY Definition

REG_CTOUCH_TAG1_XY Definition	
RO	RO
31 16	15 0

Offset: 0x130 **Reset Value: 0**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG1.

Register Definition 69 REG_CTOUCH_TAG2_XY Definition

REG_CTOUCH_TAG2_XY Definition	
RO	RO
31 16	15 0

Offset: 0x138 **Reset Value: 0**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG2.

Register Definition 70 REG_CTOUCH_TAG3_XY Definition

REG_CTOUCH_TAG3_XY Definition	
RO	RO
31 16	15 0

Offset: 0x140 **Reset Value: 0**

Bit 15 -0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG3.

Register Definition 71 REG_CTOUCH_TAG4_XY Definition

REG_CTOUCH_TAG4_XY Definition	
RO	RO
31 16	15 0

Offset: 0x148 **Reset Value: 0**

Bit 15 - 0 : The value of these bits are the Y coordinates of the touch screen, which were used by the touch engine to look up the tag result.
Bit 31 - 16: The value of these bits are X coordinates of the touch screen, which were used by the touch engine to look up the tag result.

Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG4.

3.3.5 Calibration

The calibration process is initiated by **CMD_CALIBRATE** and works with both the RTE and CTE, but is only available in the compatibility mode of the CTE. However, the results of the calibration process are applicable to both compatibility mode and extended mode. As such, users are recommended to finish the calibration process before entering into extended mode.

After the calibration process is complete, the registers **REG_TOUCH_TRANSFORM_A-F** will be updated accordingly.

3.4 Co-processor Engine Registers

Register Definition 72 REG_CMD_DL Definition

REG_CMD_DL Definition	
Reserved	R/W
31 13	12 0

Offset: 0x100 **Reset Value: 0x0000**

Bit 12 - 0 : These bits indicate the offset from RAM_DL of a display list command generated by the coprocessor engine. The coprocessor engine depends on these bits to determine the address in the display list buffer of generated display list commands. It will update this register as long as the display list commands are generated into the display list buffer.

By setting this register properly, the host can specify the starting address in the display list buffer for the coprocessor engine to generate display commands. The valid value range is from 0 to 8191(sizeof(RAM_DL)-1).

Bit 31 - 13: Reserved

Register Definition 73 REG_CMD_WRITE Definition

REG_CMD_WRITE Definition	
Reserved	R/W
31 12	11 0

Offset: 0xFC **Reset Value: 0x0**

Bit 11 - 0 : These bits are updated by the host to inform the co-processor engine of the ending address of valid data feeding into its FIFO. Typically, the host will update this register after it has downloaded the coprocessor commands into its FIFO. The valid range is from 0 to 4095, i.e. within the size of the FIFO.

Bit 31 - 12: Reserved

Note: The FIFO size of the command buffer is 4096 bytes and each co-processor instruction is of 4 bytes in size. The value to be written into this register must be 4 bytes aligned.

Register Definition 74 REG_CMD_READ Definition

REG_CMD_READ Definition	
Reserved	R/W
31 12	11 0

Offset: 0xF8 **Reset Value: 0x000**

Bit 11 - 0: These bits are updated by the coprocessor engine as long as the coprocessor engine fetched the command from its FIFO. The host can read this register to determine the FIFO fullness of the coprocessor engine. The valid value range is from 0 to 4095. In the case of an error, the co-processor engine writes 0xFFF to this register.
Bit 31 - 12: Reserved

Note: The host shall not write into this register unless in an error recovery case. The default value is zero after the coprocessor engine is reset.

Register Definition 75 REG_CMDB_SPACE Definition

REG_CMDB_SPACE Definition	
Reserved	R/W
31 12	11 0

Offset: 0x574 **Reset Value: 0xFFC**

Bit 11 - 0 : These bits are updated by the coprocessor engine to indicate the free space in RAM_CMD. The host can read this register to determine how many bytes are available to be written into RAM_CMD before writing to RAM_CMD.
Bit 31 - 12: Reserved

Register Definition 76 REG_CMDB_WRITE Definition

REG_CMDB_WRITE Definition	
WO	
31	0

Offset: 0x578 **Reset Value: 0x000**

Bit 31 - 0 : The data or command to be written into RAM_CMD. The Host can issue one write transfer with this register address to transfer data less than or equal to the amount of REG_CMDB_SPACE to make bulky data transfer possible.

3.5 Special Registers

The registers listed here are not located in **RAM_REG**. They are located in special memory address in the FT81x.

Register Definition 77 REG_TRACKER Definition

REG_TRACKER Definition		
Read Only		
Track Value		Tag Value
31	16 15	0

Offset: 0x7000 **Reset Value: 0x0**

Bit 15 - 0: These bits are set to indicate the tag value of a graphics object which is being touched.

Bit 31 - 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.

Register Definition 78 REG_TRACKER_1 Definition

REG_TRACKER_1 Definition	
Read Only	
Track Value	Tag Value
31 16	15 0

Offset: 0x7004 **Reset Value: 0x0**

Bit 15 - 0: These bits are set to indicate the tag value of a graphics object which is being touched as the second point.
Bit 31 - 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.

Register Definition 79 REG_TRACKER_2 Definition

REG_TRACKER_2 Definition	
Read Only	
Track Value	Tag Value
31 16	15 0

Offset: 0x7008 **Reset Value: 0x0**

Bit 15 - 0: These bits are set to indicate the tag value of a graphics object which is being touched as the third touch point.
Bit 31 - 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.

Register Definition 80 REG_TRACKER_3 Definition

REG_TRACKER_3 Definition	
Read Only	
Track Value	Tag Value
31 16	15 0

Offset: 0x700C **Reset Value: 0x0**

Bit 15 - 0: These bits are set to indicate the tag value of a graphics object which is being touched as the fourth touch point.
Bit 31 - 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.

Register Definition 81 REG_TRACKER_4 Definition

REG_TRACKER_4 Definition	
Read Only	
Track Value	Tag Value
31 16	15 0

Offset: 0x7010 **Reset Value: 0x0**

Bit 15 - 0: These bits are set to indicate the tag value of a graphics object which is being touched as the fifth touch point.
Bit 31 - 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.

Register Definition 82 REG_MEDIAFIFO_READ Definition

REG_MEDIAFIFO_READ Definition	
R/W	
31	0

Offset: 0x7014 **Reset Value: 0x0**

Bit 31 - 0: The value specifies the read pointer pointing to the address in RAM_G of the media FIFO.

Register Definition 83 REG_MEDIAFIFO_WRITE Definition

REG_MEDIAFIFO_WRITE Definition	
R/W	
31	0

Offset: 0x7018 **Reset Value: 0x0**

Bit 31 - 0: The value specifies the write pointer pointing to the address in RAM_G of the media FIFO.

3.6 Miscellaneous Registers

In this chapter, the miscellaneous registers covers backlight control, interrupt, GPIO, and other functionality registers.

Register Definition 84 REG_CPURESET Definition

REG_CPURESET Definition	
Reserved	R/W
31 3	2 0

Offset: 0x20 **Reset Value: 0x0**

Bit 2 - 0 :
Bit 0 for coprocessor engine,
Bit 1 for touch engine,
Bit 2 for audio engine.
Write 1 to reset the corresponding engine. Write zero to go back normal working status.
Reading 1 means the engine is in reset status, and reading zero means the engine is in working status.

Bit 31 - 3: Reserved

Register Definition 85 REG_PWM_DUTY Definition

REG_PWM_DUTY Definition	
Reserved	R/W
31 8	7 0

Offset: 0xD4 **Reset Value: 0x80**

Bit 7 - 0 : These bits define the backlight PWM output duty cycle. The valid range is from 0 to 128. 0 means backlight completely off, 128 means backlight in max brightness.
Bit 31 - 8: Reserved

Register Definition 86 REG_PWM_HZ Definition

REG_PWM_HZ Definition	
Reserved	R/W
31 14	13 0

Offset: 0xD0 **Reset Value: 0xFA**

Bit 13 - 0 : These bits define the backlight PWM output frequency in HZ. The default is 250 Hz after reset. The valid frequency is from 250Hz to 10000Hz.
Bit 31 - 14: Reserved

Register Definition 87 REG_INT_MASK Definition

REG_INT_MASK Definition	
Reserved	R/W
31 8	7 0

Offset: 0xB0 **Reset Value: 0xFF**

Bit 7 - 0 : These bits are used to mask the corresponding interrupt. 1 means to enable the corresponding interrupt source, 0 means to disable the corresponding interrupt source. After reset , all the interrupt source are eligible to trigger an interrupt by default.
Bit 31 - 8: Reserved
Note: Refer to the datasheet section "Interrupts" for more details.

Register Definition 88 REG_INT_EN Definition

REG_INT_EN Definition	
Reserved	R/W
31 1	0

Offset: 0xAC **Reset Value: 0x0**

Bit 0 : The host can set this bit to 1 to enable the global interrupt. To disable the global interrupt, the host can set this bit to 0.
Bit 31 - 1: Reserved

Note: Refer to the datasheet section "Interrupts" for the details of this register.

Register Definition 89 REG_INT_FLAGS Definition

REG_INT_FLAGS Definition	
Reserved	R/C
31 8	7 0

Offset: 0xA8 **Reset Value: 0x00**

Bit 7 - 0 : These bits are interrupt flags set by the FT81X. The host can read these bits to determine which interrupt takes place. These bits are cleared automatically by reading. The host shall not write to this register.
Bit 31 - 8: Reserved
Note: Refer to the datasheet section "Interrupts" for more details.

Register Definition 90 REG_GPIO_DIR Definition

REG_GPIO_DIR Definition	
Reserved	R/W
31 8	7 0

Offset: 0x90 **Reset Value: 0x80**

Bit 7,1,0 : These bits configure the direction of GPIO pins of the FT81X. Bit 1,0 controls the direction of GPIO1/0 and Bit 7 controls the direction of pin DISP. Bit 7 shall NOT be set to zero for pin DISP functionality. (1=output,0=input)
Bit 31 - 8, 6,5,4,3,2: Reserved
Note: This register is a legacy register for backward comaptibility only

Register Definition 91 REG_GPIO Definition

REG_GPIO Definition	
Reserved	R/W
31 8	7 0

Offset: 0x94 **Reset Value: 0x00**

Bit 31 - 8: Reserved
Bit 7,1,0 : These bits control pins DISP, GPIO_1 and GPIO_0.
Bit 6-5: GPIO0/1, CTP_RST_N Drive Strength Setting (00:5mA - default, 01:10mA, 10:15mA, 11:20mA)
Bit 4: PCLK, DISP, V/HSYNC, DE, R,G,B, BACKLIGHT Drive Strength Setting (0:5mA - default, 1:10mA)
Bit 3-2: MISO, MOSI, INT_N Drive Strength Setting (00:5mA - default, 01:10mA, 10:15mA, 11:20mA)

Note: Refer to the datasheet for more details.
This register is a legacy register for backward compatibility only

Register Definition 92 REG_GPIOX_DIR Definition

REG_GPIOX_DIR Definition			
Reserved	R/W	**Reserved**	R/W
31 16	15	14 4	3 0

Offset: 0x98 **Reset Value: 0x8000**

Bit 31-16: Reserved

Bit 15 : Controlling the direction of pin DISP. For DISP functionality, this bit shall be kept intact.

Bit 14-4: Reserved

Bit 3-0: Controlling the direction of pin GPIO 3-0. (1 = output, 0 = input)

For FT810/811, only GPIO 1-0 are available. For FT812/813, GPIO 3-0 are available.

Register Definition 93 REG_GPIOX Definition

REG_GPIOX Definition			
Reserved	R/W	**Reserved**	R/W
31 16	15 9	8 4	3 0

Offset: 0x9C **Reset Value: 0x8000**

Bit 31-16: Reserved
Bit 15 : Setting or reading the level of pin DISP. 1 for high and 0 for low
Bit 14-13:GPIO[3:0], TOUCHWAKE Drive Strength Setting
(00:5mA - default, 01:10mA, 10:15mA, 11:20mA)
Bit 12:PCLK, DISP, V/HSYNC, DE, R,G,B, BACKLIGHT Drive Strength Setting
(0:5mA - default, 1:10mA)
Bit 11 - 10:MISO, MOSI, IO2, IO3, INT_N Drive Strength Setting
(00:5mA - default, 01:10mA, 10:15mA, 11:20mA)
Bit 9: INT_N Type
(0 : OD - default, 1 : Push-pull)
Bit 8-4: Reserved
Bit 3-0: Writing or reading the pin of GPIO 3-0. 1 for high and 0 for low.
For FT810/811, only GPIO 1-0 are available. For FT812/813, GPIO 3-0 are available.

Register Definition 94 REG_FREQUENCY Definition

REG_FREQUENCY Definition	
Read / Write	
31	0

Offset: 0xC **Reset Value: 0x3938700**

Bit 31 - 0: These bits are set 0x3938700 after reset, i.e. The main clock frequency is 60MHz by default. The value is in Hz. If the host selects the alternative frequency, this register must be updated accordingly.

Register Definition 95 REG_CLOCK Definition

REG_CLOCK Definition	
Read Only	
31	0

Offset: 8 **Reset Value: 0x00000000**

Bit 31 - 0: These bits are set to zero after reset. The register counts the number of FT81X main clock cycles since reset. If the FT81X main clock's frequency is 60Mhz, it will wrap around after about 71 seconds.

Register Definition 96 REG_FRAMES Definition

REG_FRAMES Definition	
Read Only	
31	0

Offset: 4 **Reset Value: 0x00000000**

Bit 31 - 0: These bits are set to zero after reset. The register counts the number of screen frames. If the refresh rate is 60Hz, it will wrap up till about 828 days after reset.

Register Definition 97 REG_ID Definition

REG_ID Definition	
Reserved	RO
31 8	7 0

Offset: 0 **Reset Value: 0x7C**

Bit 7 - 0: These bits are the built-in register ID. The host can read this to determine if the chip belongs to the EVE series. The value shall always be 0x7C.
Bit 31 - 8: Reserved

Register Definition 98 REG_TRIM Definition

REG_TRIM Definition		
Reserved		R/W
31	5	4 0

Address: 0x10256C **Reset Value: 0x0**

Bit 0 - 4: These bits are set to trim the internal clock.
Bit 5 - 31: Reserved

Register Definition 99 REG_SPI_WIDTH Definition

REG_SPI_WIDTH Definition			
Reserved			R/W
31	3	2 1	0

Address: 0x180 **Reset Value: 0x0**

Bit 2: Extra dummy on SPI read transfer. Writing 1 to enable one extra dummy byte on SPI read transfer.
Bit 1 - 0: SPI data bus width
00: 1 bit 01: 2 bit(Dual-SPI) 10: 4 bit (Quad-SPI) 11: undefined
Bit 31 - 3: Reserved

4 Display List Commands

The graphics engine of the FT81X takes the instructions from display list memory RAM_DL in the form of commands. Each command is 4 bytes long and one display list can be filled with up to 2048 commands as the size of RAM_DL is 8K bytes. The graphics engine performs the respective operation according to the definition of commands.

4.1 Graphics State

The graphics state which controls the effects of a drawing action is stored in the graphics context. Individual pieces of state can be changed by the appropriate display list commands (e.g. **COLOR_RGB)** and the entire current state can be saved and restored using the **SAVE_CONTEXT** and **RESTORE_CONTEXT** commands.

Note that the bitmap drawing state is special: Although the bitmap handle is part of the graphics context, the parameters for each bitmap handle are not part of the graphics context. They are neither saved nor restored by **SAVE_CONTEXT** and **RESTORE_CONTEXT**. These parameters are changed using the **BITMAP_SOURCE**, **BITMAP_LAYOUT**, and **BITMAP_SIZE** commands. Once these parameters are set up, they can be utilized at any display list until they were changed.

SAVE_CONTEXT and **RESTORE_CONTEXT** are comprised of a 4-level stack in addition to the current graphics context. The table below details the various parameters in the graphics context.

Table 5 Graphics Context

Parameters	Default values	Commands
func & ref	ALWAYS, 0	ALPHA_FUNC
func & ref	ALWAYS, 0	STENCIL_FUNC
Src & dst	SRC_ALPHA, ONE_MINUS_SRC_ALPHA	BLEND_FUNC
Cell value	0	CELL
Alpha value	0	COLOR_A
Red, Blue, Green colors	(255,255,255)	COLOR_RGB
Line width in 1/16 pixels	16	LINE_WIDTH
Point size in 1/16 pixels	16	POINT_SIZE
Width & height of scissor	HSIZE,2048	SCISSOR_SIZE
Starting coordinates of scissor	(x, y) = (0,0)	SCISSOR_XY
Current bitmap handle	0	BITMAP_HANDLE
Bitmap transform coefficients	+1.0,0,0,0,+1.0,0	BITMAP_TRANSFORM_A-F
Stencil clear value	0	CLEAR_STENCIL
Tag clear value	0	CLEAR_TAG

Parameters	Default values	Commands
Mask value of stencil	255	STENCIL_MASK
spass and sfail	KEEP,KEEP	STENCIL_OP
Tag buffer value	255	TAG
Tag mask value	1	TAG_MASK
Alpha clear value	0	CLEAR_COLOR_A
RGB clear color	(0,0,0)	CLEAR_COLOR_RGB
Palette source address	RAM_G	PALETTE_SOURCE
Units of pixel precision	1/16 pixel	VERTEX_FORMAT, VERTEX_2F

4.2 Command Encoding

Each display list command has a 32-bit encoding. The most significant bits of the code determine the command. Command parameters (if any) are present in the least significant bits. Any bits marked as "reserved" must be zero.

The graphics primitives supported by FT81X and their respective values are referenced in the **BEGIN** command.

4.3 Command Groups

4.3.1 Setting Graphics State

ALPHA_FUNC	set the alpha test function
BITMAP_HANDLE	set the bitmap handle
BITMAP_LAYOUT/ BITMAP_LAYOUT_H	set the source bitmap memory format and layout for the current handle
BITMAP_SIZE/ BITMAP_SIZE_H	set the screen drawing of bitmaps for the current handle
BITMAP_SOURCE	set the source address for bitmap graphics
BITMAP_TRANSFORM_A-F	set the components of the bitmap transform matrix
BLEND_FUNC	set pixel arithmetic function
CELL	set the bitmap cell number for the VERTEX2F command
CLEAR	clear buffers to preset values
CLEAR_COLOR_A	set clear value for the alpha channel
CLEAR_COLOR_RGB	set clear values for red, green and blue channels
CLEAR_STENCIL	set clear value for the stencil buffer
CLEAR_TAG	set clear value for the tag buffer
COLOR_A	set the current color alpha
COLOR_MASK	enable or disable writing of color components
COLOR_RGB	set the current color red, green and blue
LINE_WIDTH	set the line width
POINT_SIZE	set point size
RESTORE_CONTEXT	restore the current graphics context from the context stack
SAVE_CONTEXT	push the current graphics context on the context stack
SCISSOR_SIZE	set the size of the scissor clip rectangle
SCISSOR_XY	set the top left corner of the scissor clip rectangle
STENCIL_FUNC	set function and reference value for stencil testing
STENCIL_MASK	control the writing of individual bits in the stencil planes
STENCIL_OP	set stencil test actions
TAG	set the current tag value
TAG_MASK	control the writing of the tag buffer
VERTEX_FORMAT	set the precision of VERTEX2F coordinates
VERTEX_TRANSLATE_X	specify the vertex transformation’s X translation component
VERTEX_TRANSLATE_Y	specify the vertex transformation’s Y translation component
PALETTE_SOURCE	Specify the base address of the palette

4.3.2 Drawing Actions

BEGIN	start drawing a graphics primitive
END	finish drawing a graphics primitive
VERTEX2F	supply a vertex with fractional coordinates
VERTEX2II	supply a vertex with unsigned coordinates

4.3.3 Execution Control

NOP	No Operation
JUMP	execute commands at another location in the display list
MACRO	execute a single command from a macro register
CALL	execute a sequence of commands at another location in the display list
RETURN	return from a previous CALL command
DISPLAY	end the display list

4.4 ALPHA_FUNC

Specify the alpha test function

Encoding

31 24	23 11	10 8	7 6 5 4 3 2 1 0
0x09	Reserved	func	Ref

Parameters

func

Specifies the test function, one of NEVER, LESS, LEQUAL, GREATER, GEQUAL, EQUAL, NOTEQUAL, or ALWAYS. The initial value is ALWAYS (7)

NAME	VALUE
NEVER	0
LESS	1
LEQUAL	2
GREATER	3
GEQUAL	4
EQUAL	5
NOTEQUAL	6
ALWAYS	7

Figure 5: The constants of ALPHA_FUNC

ref

Specifies the reference value for the alpha test. The initial value is 0

Graphics context

The values of func and ref are part of the graphics context, as described in section 4.1

See also

None

4.5 BEGIN

Begin drawing a graphics primitive

Encoding

31 24	23 4	3	2	1	0
0x1F	reserved	**Prim**			

Parameters

prim

Graphics primitive. The valid value is defined as below:

Table 6 FT81X graphics primitive operation definition

NAME	VALUE	Description
BITMAPS	1	Bitmap drawing primitive
POINTS	2	Point drawing primitive
LINES	3	Line drawing primitive
LINE_STRIP	4	Line strip drawing primitive
EDGE_STRIP_R	5	Edge strip right side drawing primitive
EDGE_STRIP_L	6	Edge strip left side drawing primitive
EDGE_STRIP_A	7	Edge strip above drawing primitive
EDGE_STRIP_B	8	Edge strip below side drawing primitive
RECTS	9	Rectangle drawing primitive

Description

All primitives supported by the FT81X are defined in the table above. The primitive to be drawn is selected by the BEGIN command. Once the primitive is selected, it will be valid till the new primitive is selected by the BEGIN command.

Please note that the primitive drawing operation will not be performed until VERTEX2II or VERTEX2F is executed.

Examples

Drawing points, lines and bitmaps:


```
dl( BEGIN(POINTS) );
dl( VERTEX2II(50, 5, 0, 0) );
dl( VERTEX2II(110, 15, 0, 0) );
dl( BEGIN(LINES) );
dl( VERTEX2II(50, 45, 0, 0) );
dl( VERTEX2II(110, 55, 0, 0) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(50, 65, 31, 0x45) );
dl( VERTEX2II(110, 75, 31, 0x46) );
```

Graphics context

None

See also

END

4.6 BITMAP_HANDLE

Specify the bitmap handle

Encoding

31 24	23 5	4	3	2	1	0
0x05	reserved	handle				

Parameters

handle

Bitmap handle. The initial value is 0. The valid value range is from 0 to 31.

Description

By default, bitmap handles 16 to 31 are used for built-in font and 15 is used as scratch bitmap handle by co-processor engine commands CMD_GRADIENT, CMD_BUTTON and CMD_KEYS.

Graphics context

The value of handle is part of the graphics context, as described in section 4.1

See also

BITMAP_LAYOUT, BITMAP_SIZE

4.7 BITMAP_LAYOUT

Specify the source bitmap memory format and layout for the current handle.

Encoding

31 24	23 22 21 20 19	18 9	8 0
0x07	format	linestride	height

Parameters

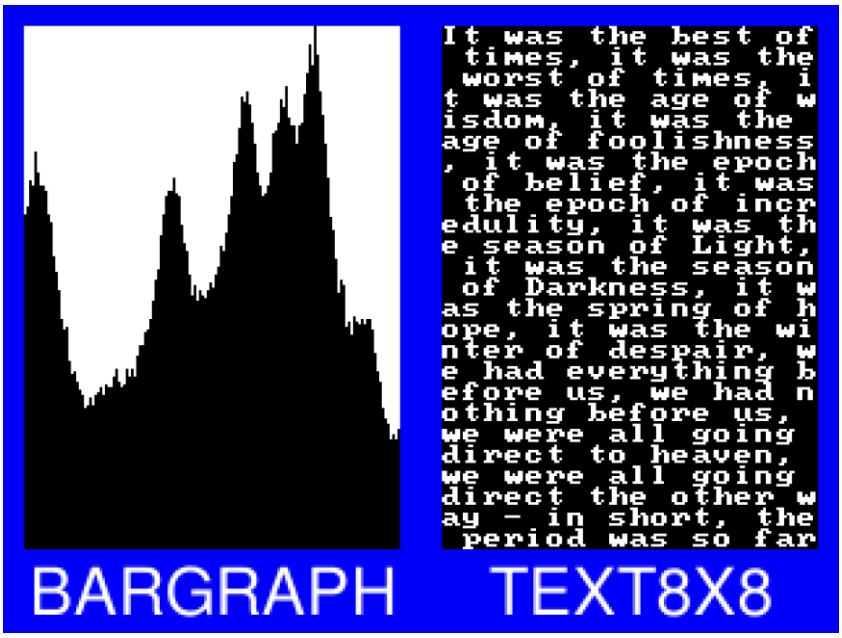
format

Bitmap pixel format. The valid range is from 0 to 11 and defined as per the table below.

Table 7 BITMAP_LAYOUT format list

Name	Value	Bits/pixel	Alpha bits	Red bits	Green bits	Blue bits
ARGB1555	0	16	1	5	5	5
L1	1	1	1	0	0	0
L4	2	4	4	0	0	0
L8	3	8	8	0	0	0
RGB332	4	8	0	3	3	2
ARGB2	5	8	2	2	2	2
ARGB4	6	16	4	4	4	4
RGB565	7	16	0	5	6	5
TEXT8X8	9	-	-	-	-	-
TEXTVGA	10	-	-	-	-	-
BARGRAPH	11	-	-	-	-	-
PALETTED565	14	8	0	5	6	5
PALETTED4444	15	8	4	4	4	4
PALETTED8	16	8	8	8	8	8
L2	17	2	2	0	0	0

Examples of various supported bitmap formats (except TXTVGA) are shown as below:



BARGRAPH - render data as a bar graph. Looks up the x coordinate in a byte array, then gives an opaque pixel if the byte value is less than y, otherwise a transparent pixel. The result is a bar graph of the bitmap data. A maximum of 256x256 size bitmap can be drawn using the BARGRAPH format. Orientation, width and height of the graph can be altered using the bitmap transform matrix.

TEXT8X8 - lookup in a fixed 8x8 font. The bitmap is a byte array present in the graphics ram and each byte indexes into an internal 8x8 CP437 [2] font (inbuilt font bitmap handles 16 & 17 are used for drawing TEXT8X8 format). The result is that the bitmap acts like a character grid. A single bitmap can be drawn which covers all or part of the display; each byte in the bitmap data corresponds to one 8x8 pixel character cell.

TEXTVGA – lookup in a fixed 8x16 font with TEXTVGA syntax. The bitmap is a TEXTVGA array present in the graphics ram, each element indexes into an internal 8x16 CP437 [2] font (inbuilt font bitmap handles 18 & 19 are used for drawing TEXTVGA format with control information such as background color, foreground color and cursor etc.). The result is that the bitmap acts like a TEXTVGA grid. A single bitmap can be drawn which covers all or part of the display; each TEXTVGA data type in the bitmap corresponds to one 8x16 pixel character cell.

linestride

Bitmap line strides, in bytes. It represents the amount of memory used for each line of bitmap pixels.

For L1, L2, L4 format, the necessary data has to be padded to make it byte aligned.

Normally, it can be calculated with the following formula:

linestride = width * byte/pixel

For example, if one bitmap is 64(width) x 32(height) pixels in L4 format, the line stride shall be (64*1/2 = 32).

height

Bitmap height, in lines

Description

For more details about memory layout according to pixel format, refer to the figures below:

L1 format layout		Byte Order
Pixel 0	Bit 7	Byte 0
Pixel 1	Bit 6	
.....		
Pixel 7	Bit 0	

L2 format layout		Byte Order
Pixel 0	Bit 7-6	Byte 0
Pixel 1	Bit 5-4	
Pixel 2	Bit 3-2	
Pixel 3	Bit 1-0	

L4 format layout		Byte Order
Pixel 0	Bit 7-4	Byte 0
Pixel 1	Bit 3-0	

L8 format layout		Byte Order
Pixel 0	Bit 7-0	Byte 0
pixel 1	Bit 15-8	Byte 1
pixel 2	Bit 23-16	Byte 2

Figure 6: L1/L2/L4/L8 Pixel Format

ARGB2 format layout			Byte Order
Pixel 0	A	Bit 7-6	Byte 0
	R	Bit 5-4	
	G	Bit 3-2	
	B	Bit 1-0	

ARGB1555 format layout			Byte Order
Pixel 0	A	Bit 15	Byte 1 Byte 0
	R	Bit 14-10	
	G	Bit 9- 5	
	B	Bit 4-0	

Figure 7: ARGB2/1555 Pixel Format

ARGB4/PALETTED4444			Byte Order
Pixel 0	A	Bit 15-12	Byte 1
	R	Bit 11-8	
	G	Bit 7-4	Byte 0
	B	Bit 3-0	

RGB332			Byte Order
Pixel 0	R	Bit 7-5	Byte 0
	G	Bit 4-2	
	B	Bit 1-0	

RGB565/PALETTED565			Byte Order
pixel 0	R	Bit 15-11	Byte 1 Byte 0
	G	Bit 10-5	
	B	Bit 4-0	

Figure 8: ARGB4/PALETTED4444, RGB332, RGB565/PALETTED565 Pixel Format

PALETTED8			Byte Order
Pixel 0	A	Bit 31-24	Byte 3
	R	Bit 23-16	Byte 2
	G	Bit 15-8	Byte 1
	B	Bit 7-0	Byte 0

Figure 9: PALETTED8 Pixel Format

Graphics context

None

Note

PALETTED8 format is supported indirectly in FT81X and it is different from PALETTED format in FT80X. To render Alpha, Red, Green and Blue channels, multi-pass drawing action is required.

The following display list snippet shows:

```
//addr_pal is the starting address of palette lookup table in
RAM_G
//bitmap source(palette indices) is starting from address 0

dl(BITMAP_HANDLE(0))
dl(BITMAP_LAYOUT(PALETTED8, width, height))
dl(BITMAP_SIZE(NEAREST, BORDER, BORDER, width, height))

dl(BITMAP_SOURCE(0)) //bitmap source(palette indices)

dl(BEGIN(BITMAPS))
dl(BLEND_FUNC(ONE, ZERO))

//Draw Alpha channel
dl(COLOR_MASK(0,0,0,1))
dl(PALETTE_SOURCE(addr_pal+3))
dl(VERTEX2II(0, 0, 0, 0))

//Draw Red channel
dl(BLEND_FUNC(DST_ALPHA, ONE_MINUS_DST_ALPHA))
dl(COLOR_MASK(1,0,0,0))
dl(PALETTE_SOURCE (addr_pal+2))
dl(VERTEX2II (0, 0, 0, 0))

//Draw Green channel
dl(COLOR_MASK(0,1,0,0))
dl(PALETTE_SOURCE(addr_pal + 1))
dl(VERTEX2II(0, 0, 0, 0))

//Draw Blue channel
dl(COLOR_MASK(0,0,1,0))
dl(PALETTE_SOURCE(addr_pal))
dl(VERTEX2II(0, 0, 0, 0))
```

Code Snippet 10 PALETTED8 drawing example

See also

BITMAP_HANDLE, BITMAP_SIZE, BITMAP_SOURCE,PALETTE_SOURCE

4.8 BITMAP_LAYOUT_H

Specify the 2 most significant bits of the source bitmap memory format and layout for the current handle.

Encoding

31 24	23 4	3 2	1 0
0x28	reserved	linestride	height

Parameters

linestride

The 2 most significant bits of the 12-bit linestride parameter value specified to BITMAP_LAYOUT.

height

The 2 most significant bits of the 11-bit height parameter value specified to BITMAP_LAYOUT.

Description

This command is the extension command of BITMAP_LAYOUT for large drawn bitmaps. This command is not needed if the specified linestride parameter value to BITMAP_LAYOUT is less than 1024 and the height parameter value is less than 512.

Examples

NA

See also

BITMAP_LAYOUT

4.9 BITMAP_SIZE

Specify the screen drawing of bitmaps for the current handle

Encoding

31 24	23 21	20	19	18	17 9	8 0
0x08	Reserved	filter	wrapx	wrapy	width	height

Parameters

filter

Bitmap filtering mode, one of NEAREST or BILINEAR

The value of NEAREST is 0 and the value of BILINEAR is 1.

wrapx

Bitmap x wrap mode, one of REPEAT or BORDER

The value of BORDER is 0 and the value of REPEAT is 1.

wrapy

Bitmap y wrap mode, one of REPEAT or BORDER

The value of BORDER is 0 and the value of REPEAT is 1.

width

Drawn bitmap width, in pixels. From 1 to 511. Zero has special meaning.

height

Drawn bitmap height, in pixels. From 1 to 511. Zero has special meaning.

Description

This command controls the drawing of bitmaps: the on-screen size of the bitmap, the behavior for wrapping, and the filtering function. Please note that if wrapx or wrapy is REPEAT then the corresponding memory layout dimension (BITMAP_LAYOUT line stride or height) must be power of two, otherwise the result is undefined.

For width and height, the value from 1 to 511 means the bitmap width and height in pixel. The value zero has the special meaning if there are no BITMAP_SIZE_H present before or a high bit in BITMAP_SIZE_H is zero: it means 2048 pixels, other than 0 pixels.

4.10 BITMAP_SIZE_H

Specify the 2 most significant bits of bitmaps dimension for the current handle.

Encoding

31 24	23 4	3 2	1 0
0x29	reserved	width	height

Parameters

width

2 most significant bits of bitmap width. The initial value is zero.

height

2 most significant bits of bitmap height. The initial value is zero.

Description

This command is the extension command of BITMAP_SIZE for bitmap larger than 511 x 511 pixels.

Graphics context

None

See also

BITMAP_HANDLE, BITMAP_LAYOUT, BITMAP_SOURCE, BITMAP_SIZE

4.11 BITMAP_SOURCE

Specify the source address of bitmap data in FT81X graphics memory RAM_G.

Encoding

31 24	23 22	21 0
0x01	reserved	addr

Parameters

addr

Bitmap address in RAM_G of FT81X, aligned with respect to the bitmap format.

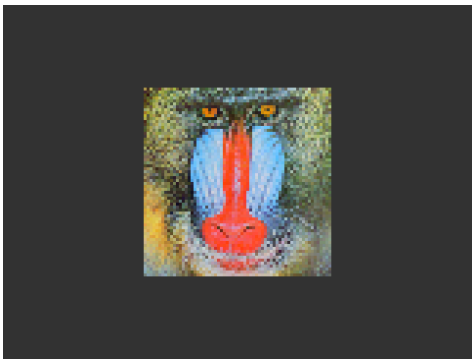
For example, if the bitmap format is RGB565/ARGB4/ARGB1555, the bitmap source shall be aligned to 2 bytes.

Description

The bitmap source address is normally the address in main memory where the bitmap graphic data is loaded.

Examples

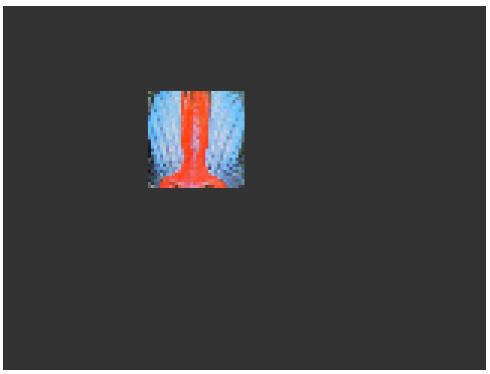
Drawing a 64 x 64 bitmap, loaded at address 0:



```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_SIZE(NEAREST, BORDER,
BORDER, 64, 64) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(48, 28, 0, 0) );
```

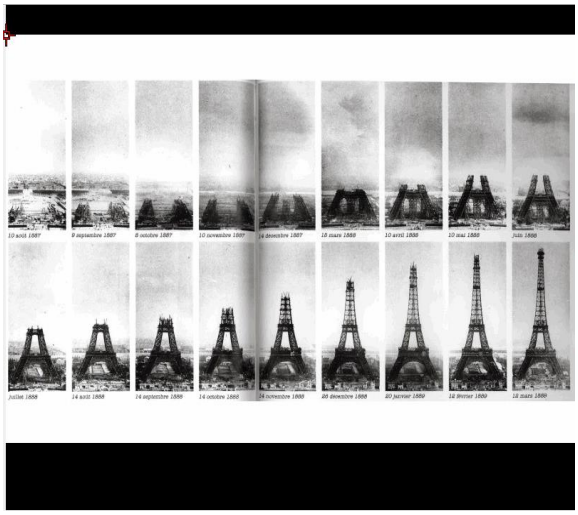
Using the same graphics data, but with source and size changed to show only a

32 x 32 detail:



```
dl( BITMAP_SOURCE(128 * 16 + 32) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_SIZE(NEAREST, BORDER,
BORDER, 32, 32) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(48, 28, 0, 0) );
```

Display one 800x480 image by using extended display list commands mentioned above:



```
dl(BITMAP_HANDLE(0));
dl(BITMAP_SOURCE(0));
dl(BITMAP_SIZE_H(1, 0));
dl(BITMAP_SIZE(NEAREST, BORDER,
BORDER, 288, 480));
dl(BITMAP_LAYOUT_H(1, 0));
dl(BITMAP_LAYOUT(ARGB1555, 576, 480));
dl(BEGIN(BITMAPS));
dl(VERTEX2II(76, 25, 0, 0));
dl(END());
```

Graphics context

None

See also

BITMAP_LAYOUT, BITMAP_SIZE

4.12 BITMAP_TRANSFORM_A

Specify the A coefficient of the bitmap transform matrix.

Encoding

31 24	23 17	16 0
0x15	Reserved	a

Parameters

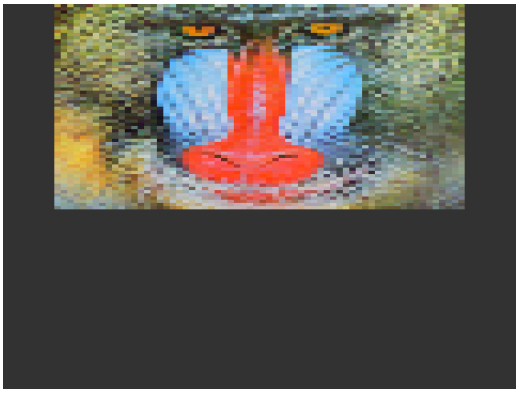
a Coefficient A of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 256.

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

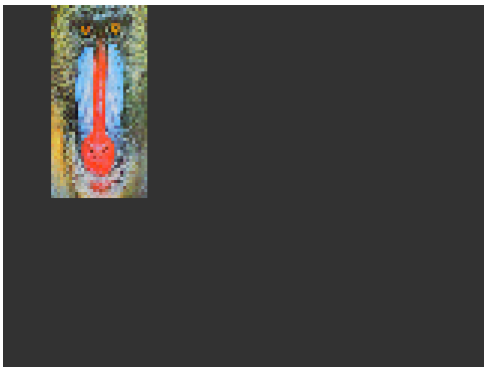
Examples

A value of 0.5 (128) causes the bitmap appear double width:



```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_TRANSFORM_A(128) );
dl( BITMAP_SIZE(NEAREST, BORDER, BORDER, 128, 128) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(16, 0, 0, 0) );
```

A value of 2.0 (512) gives a half-width bitmap:



```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_TRANSFORM_A(512) );
dl( BITMAP_SIZE(NEAREST, BORDER, BORDER, 128, 128) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(16, 0, 0, 0) );
```

Graphics context

The value of a is part of the graphics context, as described in section 4.1

See also

None

4.13 BITMAP_TRANSFORM_B

Specify the B coefficient of the bitmap transform matrix

Encoding

31 24	23 17	16 0
0x16	Reserved	b

Parameters

b

Coefficient B of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 0

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

Graphics context

The value of B is part of the graphics context, as described in section 4.1

See also

None

4.14 BITMAP_TRANSFORM_C

Specify the C coefficient of the bitmap transform matrix

Encoding

31 24	23 0
0x17	c

Parameters

c

Coefficient C of the bitmap transform matrix, in signed 15.8 bit fixed-point form. The initial value is 0

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

Graphics context

The value of c is part of the graphics context, as described in section 4.1

See also

None

4.15 BITMAP_TRANSFORM_D

Specify the D coefficient of the bitmap transform matrix

Encoding

31 24	23 17	16 0
0x18	Reserved	d

Parameters

d

Coefficient D of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 0

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

Graphics context

The value of d is part of the graphics context, as described in section 4.1

See also

None

4.16 BITMAP_TRANSFORM_E

Specify the E coefficient of the bitmap transform matrix

Encoding

31 24	23 17	16 0
0x19	Reserved	e

Parameters

e

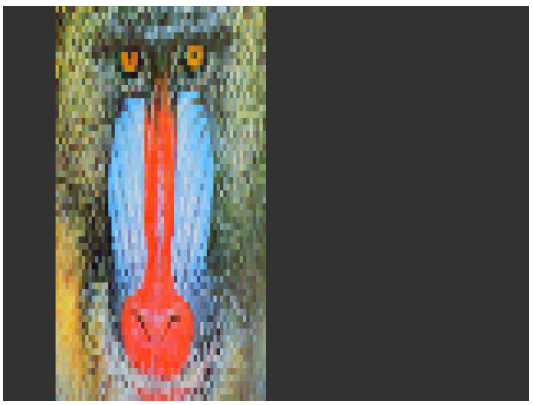
Coefficient E of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 256

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

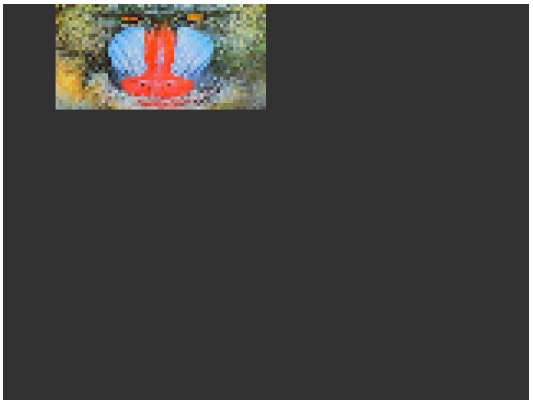
Examples

A value of 0.5 (128) causes the bitmap appear double height:



```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_TRANSFORM_E(128) );
dl( BITMAP_SIZE(NEAREST, BORDER, BORDER, 128, 128) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(16, 0, 0, 0) );
```

A value of 2.0 (512) gives a half-height bitmap:



```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_TRANSFORM_E(512) );
dl( BITMAP_SIZE(NEAREST, BORDER, BORDER, 128, 128) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(16, 0, 0, 0) );
```

Graphics context

The value of e is part of the graphics context, as described in section 4.1

See also

None

4.17 BITMAP_TRANSFORM_F

Specify the F coefficient of the bitmap transform matrix

Encoding

31 24	23 0
0x1A	f

Parameters

f

Coefficient F of the bitmap transform matrix, in signed 15.8 bit fixed-point form. The initial value is 0

Description

BITMAP_TRANSFORM_A-F coefficients are used to perform bitmap transform functionalities such as scaling, rotation and translation. These are similar to openGL transform functionality.

Graphics context

The value of f is part of the graphics context, as described in section 4.1

See also

None

4.18 BLEND_FUNC

Specify pixel arithmetic

Encoding

31 24	23 6	5 3	2 0
0x0B	reserved	src	dst

Parameters

src

Specifies how the source blending factor is computed. One of ZERO, ONE, SRC_ALPHA, DST_ALPHA, ONE_MINUS_SRC_ALPHA or ONE_MINUS_DST_ALPHA. The initial value is SRC_ALPHA (2).

dst

Specifies how the destination blending factor is computed, one of the same constants as src. The initial value is ONE_MINUS_SRC_ALPHA(4)

Table 8 BLEND_FUNC constant value definition

NAME	VALUE	Description
ZERO	0	Check openGL definition
ONE	1	Check openGL definition
SRC_ALPHA	2	Check openGL definition
DST_ALPHA	3	Check openGL definition
ONE_MINUS_SRC_ALPHA	4	Check openGL definition
ONE_MINUS_DST_ALPHA	5	Check openGL definition

Description

The blend function controls how new color values are combined with the values already in the color buffer. Given a pixel value source and a previous value in the color buffer destination, the computed color is:

$$source \times src + destination \times dst$$

for each color channel: red, green, blue and alpha.

Examples

The default blend function of (SRC_ALPHA, ONE_MINUS_SRC_ALPHA) causes drawing to overlay the destination using the alpha value:

```
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(50, 30, 31, 0x47) );
dl( COLOR_A( 128 ) );
dl( VERTEX2II(60, 40, 31, 0x47) );
```

A destination factor of zero means that destination pixels are not used:

```
dl( BEGIN(BITMAPS) );
dl( BLEND_FUNC(SRC_ALPHA, ZERO) );
dl( VERTEX2II(50, 30, 31, 0x47) );
dl( COLOR_A( 128 ) );
dl( VERTEX2II(60, 40, 31, 0x47) );
```

Using the source alpha to control how much of the destination to keep:

```
dl( BEGIN(BITMAPS) );
dl( BLEND_FUNC(ZERO, SRC_ALPHA) );
dl( VERTEX2II(50, 30, 31, 0x47) );
```

Graphics context

The values of src and dst are part of the graphics context, as described in section 4.1

See also

COLOR_A

4.19 CALL

Execute a sequence of commands at another location in the display list

Encoding

31 24	23 16	15 0
0x1D	reserved	dest

Parameters

dest

The offset of the destination address from RAM_DL which the display command is to be switched to. FT81X has the stack to store the return address. To come back to the next command of source address, the RETURN command can help.
The valid range is from 0 to 8191.

Description

CALL and RETURN have a 4 level stack in addition to the current pointer. Any additional CALL/RETURN done will lead to unexpected behavior.

Graphics context

None

See also

JUMP, RETURN

4.20 CELL

Specify the bitmap cell number for the VERTEX2F command.

Encoding

31 24	23 7	6 0
0x06	Reserved	Cell

Parameters

cell

bitmap cell number. The initial value is 0

Graphics context

The value of cell is part of the graphics context, as described in section 4.1

See also

None

4.21 CLEAR

Clear buffers to preset values

Encoding

31 24	23 3	2	1	0
0x26	Reserved	C	S	T

Parameters

c

Clear color buffer. Setting this bit to 1 will clear the color buffer of the FT81X to the preset value. Setting this bit to 0 will maintain the color buffer of the FT81X with an unchanged value. The preset value is defined in command CLEAR_COLOR_RGB for RGB channel and CLEAR_COLOR_A for alpha channel.

s

Clear stencil buffer. Setting this bit to 1 will clear the stencil buffer of the FT81X to the preset value. Setting this bit to 0 will maintain the stencil buffer of the FT81X with an unchanged value. The preset value is defined in command CLEAR_STENCIL.

t

Clear tag buffer. Setting this bit to 1 will clear the tag buffer of the FT81X to the preset value. Setting this bit to 0 will maintain the tag buffer of the FT81X with an unchanged value. The preset value is defined in command CLEAR_TAG.

Description

The scissor test and the buffer write masks affect the operation of the clear. Scissor limits the cleared rectangle, and the buffer write masks limit the affected buffers. The state of the alpha function, blend function, and stenciling do not affect the clear.

Examples

To clear the screen to bright blue:

```
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( CLEAR(1, 0, 0) );
```

To clear part of the screen to gray, part to blue using scissor rectangles:

```
dl( CLEAR_COLOR_RGB(100, 100, 100) );
dl( CLEAR(1, 1, 1) );
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( SCISSOR_SIZE(30, 120) );
dl( CLEAR(1, 1, 1) );
```

Graphics context

None

See also

CLEAR_COLOR_A, CLEAR_STENCIL, CLEAR_TAG, CLEAR_COLOR_RGB

4.22 CLEAR_COLOR_A

Specify clear value for the alpha channel

Encoding

32 24	23 8	7 0
0x0F	Reserved	Alpha

Parameters

alpha

Alpha value used when the color buffer is cleared. The initial value is 0

Graphics context

The value of alpha is part of the graphics context, as described in section 4.1

See also

CLEAR_COLOR_RGB, CLEAR

4.23 CLEAR_COLOR_RGB

Specify clear values for red, green and blue channels

Encoding

31 24	23 16	15 8	7 0
0x02	Red	Blue	Green

Parameters

red

Red value used when the color buffer is cleared. The initial value is 0

green

Green value used when the color buffer is cleared. The initial value is 0

blue

Blue value used when the color buffer is cleared. The initial value is 0

Description

Sets the color values used by a following CLEAR.

Examples

To clear the screen to bright blue:

```
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( CLEAR(1, 1, 1) );
```

To clear part of the screen to gray, part to blue using scissor rectangles:

```
dl( CLEAR_COLOR_RGB(100, 100, 100) );
dl( CLEAR(1, 1, 1) );
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( SCISSOR_SIZE(30, 120) );
dl( CLEAR(1, 1, 1) );
```

Graphics context

The values of red, green and blue are part of the graphics context, as described in section 4.1

See also

CLEAR_COLOR_A, CLEAR

4.24 CLEAR_STENCIL

Specify clear value for the stencil buffer

Encoding

31 24	23 8	7 0
0x11	Reserved	s

Parameters

s

Value used when the stencil buffer is cleared. The initial value is 0

Graphics context

The value of s is part of the graphics context, as described in section 4.1

See also

CLEAR

4.25 CLEAR_TAG

Specify clear value for the tag buffer

Encoding

31 24	23 8	7 0
0x12	Reserved	t

Parameters

t

Value used when the tag buffer is cleared. The initial value is 0.

Graphics context

The value of s is part of the graphics context, as described in section 4.1

See also

TAG, TAG_MASK, CLEAR

4.26 COLOR_A

Set the current color alpha

Encoding

31 24	23 8	7 0
0x10	Reserved	alpha

Parameters

alpha

Alpha for the current color. The initial value is 255

Description

Sets the alpha value applied to drawn elements - points, lines, and bitmaps. How the alpha value affects image pixels depends on BLEND_FUNC; the default behavior is a transparent blend.

Examples

Drawing three characters with transparency 255, 128, and 64:

```
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(50, 30, 31, 0x47) );
dl( COLOR_A( 128 ) );
dl( VERTEX2II(58, 38, 31, 0x47) );
dl( COLOR_A( 64 ) );
dl( VERTEX2II(66, 46, 31, 0x47) );
```

Graphics context

The value of alpha is part of the graphics context, as described in section 4.1

See also

COLOR_RGB, BLEND_FUNC

4.27 COLOR_MASK

Enable or disable writing of color components

Encoding

31 24	23 4	3	2	1	0
0x20	reserved	r	g	b	a

Parameters

r

Enable or disable the red channel update of the FT81X color buffer. The initial value is 1 and means enable.

g

Enable or disable the green channel update of the FT81X color buffer. The initial value is 1 and means enable.

b

Enable or disable the blue channel update of the FT81X color buffer. The initial value is 1 and means enable.

a

Enable or disable the alpha channel update of the FT81X color buffer. The initial value is 1 and means enable.

Description

The color mask controls whether the color values of a pixel are updated. Sometimes it is used to selectively update only the red, green, blue or alpha channels of the image. More often, it is used to completely disable color updates while updating the tag and stencil buffers.

Examples

Draw an '8' digit in the middle of the screen. Then paint an invisible 40-pixel circular touch area into the tag buffer:

```
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(68, 40, 31, 0x38) );
dl( POINT_SIZE(40 * 16) );
dl( COLOR_MASK(0, 0, 0, 0) );
dl( BEGIN(POINTS) );
dl( TAG( 0x38 ) );
dl( VERTEX2II(80, 60, 0, 0) );
```

Graphics context

The values of r, g, b and a are part of the graphics context, as described in section 4.1

See also

TAG_MASK

4.28 COLOR_RGB

Set the current color red, green and blue

Encoding

31 24	23 16	15 8	7 0
0x04	Red	Blue	Green

Parameters

red

Red value for the current color. The initial value is 255

green

Green value for the current color. The initial value is 255

blue

Blue value for the current color. The initial value is 255

Description

Sets the red, green and blue values of the FT81X color buffer which will be applied to the following draw operation.

Examples

Drawing three characters with different colors:


```
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(50, 38, 31, 0x47) );
dl( COLOR_RGB( 255, 100, 50 ) );
dl( VERTEX2II(80, 38, 31, 0x47) );
dl( COLOR_RGB( 50, 100, 255 ) );
dl( VERTEX2II(110, 38, 31, 0x47) );
```

Graphics context

The values of red, green and blue are part of the graphics context, as described in section 4.1

See also

COLOR_A

4.29 DISPLAY

End the display list. FT81X will ignore all the commands following this command.

Encoding

31 24	23 0
0x0	Reserved

Parameters

None

Graphics context

None

See also

None

4.30 END

End drawing a graphics primitive.

Encoding

31 24	23 0
0x21	Reserved

Parameters

None

Description

It is recommended to have an END for each BEGIN. However, advanced users may avoid the usage of END in order to save space for extra graphics instructions in RAM_DL.

Graphics context

None

See also

BEGIN

4.31 JUMP

Execute commands at another location in the display list

Encoding

31 24	23 16	15 0
0x1E	Reserved	dest

Parameters

dest

Display list address (offset from RAM_DL) to be jumped. The valid range is from 0 to 8191.

Graphics context

None

See also

CALL

4.32 LINE_WIDTH

Specify the width of lines to be drawn with primitive LINES in 1/16 pixel precision.

Encoding

31 24	23 12	11 0
0x0E	Reserved	width

Parameters

width

Line width in 1/16 pixel precision. The initial value is 16.

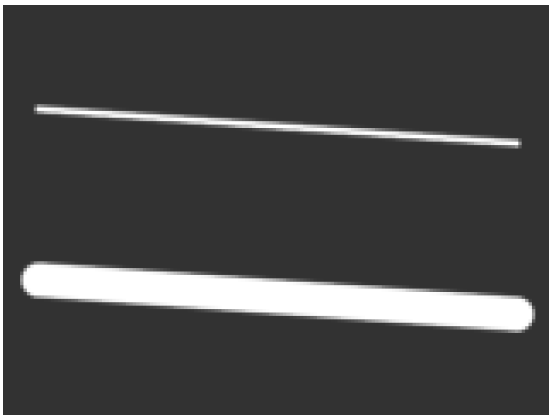
Description

Sets the width of drawn lines. The width is the distance from the center of the line to the outermost drawn pixel, in units of 1/16 pixel. The valid range is from zero to 4095. i.e. from zero to 255 pixels.

Please note the LINE_WIDTH command will affect the LINES, LINE_STRIP, RECTS, EDGE_STRIP_A/B/R/L primitives.

Examples

The second line is drawn with a width of 80, for a 5 pixel radius:



```
dl( BEGIN(LINES) );
dl( VERTEX2F(16 * 10, 16 * 30) );
dl( VERTEX2F(16 * 150, 16 * 40) );
dl( LINE_WIDTH(80) );
dl( VERTEX2F(16 * 10, 16 * 80) );
dl( VERTEX2F(16 * 150, 16 * 90) );
```

Graphics context

The value of width is part of the graphics context, as described in section 4.1

See also

None

4.33 MACRO

Execute a single command from a macro register.

Encoding

31 24	23 1	0
0x25	Reserved	m

Parameters

m

Macro registers to read. Value 0 means the FT81X will fetch the command from REG_MACRO_0 to execute. Value 1 means the FT81X will fetch the command from REG_MACRO_1 to execute. The content of REG_MACRO_0 or REG_MACRO_1 shall be a valid display list command, otherwise the behavior is undefined.

Graphics context

None

See also

None

4.34 NOP

No operation.

Encoding

31 24	23 0
0x2D	Reserved

Parameters

None

Description

Does nothing. May be used as a spacer in display lists, if required.

Graphics context

None

See also

None

4.35 PALETTE_SOURCE

Specify the base address of the palette.

Encoding

31 24	23 22	21 0
0x2A	r	addr

Parameters

r

Reserved

addr

Address of palette in RAM_G, 2-byte alignment is required if pixel format is PALETTE4444 or PALETTE565. The initial value is RAM_G

Description

Specify the base address in RAM_G for palette

Graphics context

The value of addr is part of the graphics context

See also

None

4.36 POINT_SIZE

Specify the radius of points

Encoding

31 24	23 13	12 0
0x0D	reserved	size

Parameters

size

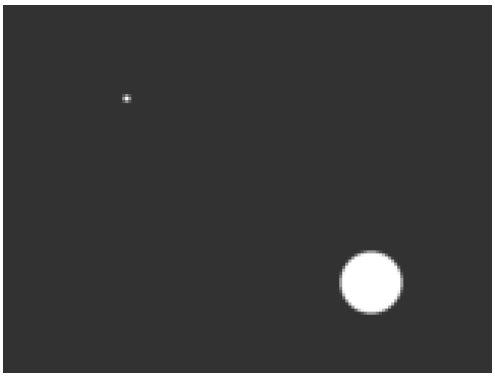
Point radius in 1/16 pixel precision. The initial value is 16. The valid range is from zero to 8191, i.e. from 0 to 511 pixels.

Description

Sets the size of drawn points. The width is the distance from the center of the point to the outermost drawn pixel, in units of 1/16 pixels.

Examples

The second point is drawn with a width of 160, for a 10 pixel radius:



```
dl( BEGIN(POINTS) );
dl( VERTEX2II(40, 30, 0, 0) );
dl( POINT_SIZE(160) );
dl( VERTEX2II(120, 90, 0, 0) );
```

Graphics context

The value of size is part of the graphics context, as described in section 4.1

See also

None

4.37 RESTORE_CONTEXT

Restore the current graphics context from the context stack

Encoding

31 24	23 0
0x23	Reserved

Parameters

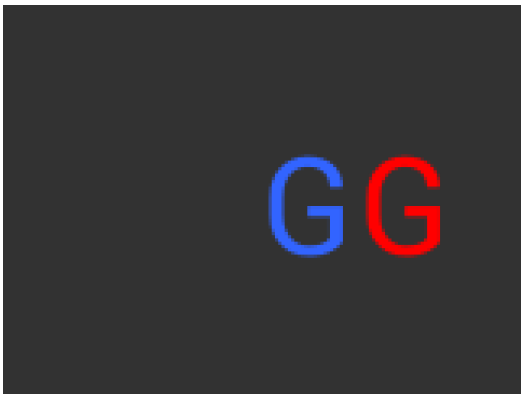
None

Description

Restores the current graphics context, as described in section 4.1. Four levels of SAVE and RESTORE stacks are available in the FT81X. Any extra RESTORE_CONTEXT will load the default values into the present context.

Examples

Saving and restoring context means that the second 'G' is drawn in red, instead of blue:



```
dl( BEGIN(BITMAPS) );
dl( COLOR_RGB( 255, 0, 0 ) );
dl( SAVE_CONTEXT() );
dl( COLOR_RGB( 50, 100, 255 ) );
dl( VERTEX2II(80, 38, 31, 0x47) );
dl( RESTORE_CONTEXT() );
dl( VERTEX2II(110, 38, 31, 0x47) );
```

Graphics context

None

See also

SAVE_CONTEXT

4.38 RETURN

Return from a previous CALL command.

Encoding

31 24	23 0
0x24	Reserved

Parameters

None

Description

CALL and RETURN have 4 levels of stack in addition to the current pointer. Any additional CALL/RETURN done will lead to unexpected behavior.

Graphics context

None

See also

CALL

4.39 SAVE_CONTEXT

Push the current graphics context on the context stack

Encoding

31 24	23 0
0x22	Reserved

Parameters

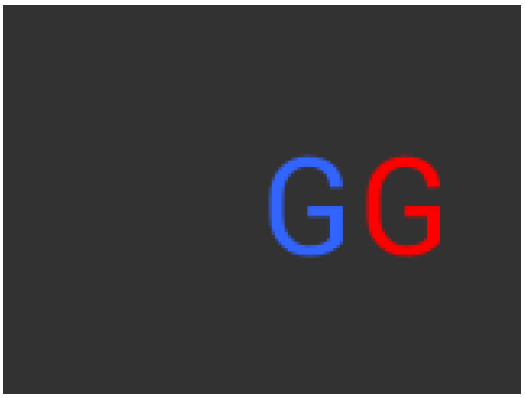
None

Description

Saves the current graphics context, as described in section 4.1. Any extra SAVE_CONTEXT will throw away the earliest saved context.

Examples

Saving and restoring context means that the second 'G' is drawn in red, instead of blue:




```
dl( BEGIN(BITMAPS) );
dl( COLOR_RGB( 255, 0, 0 ) );
dl( SAVE_CONTEXT() );
dl( COLOR_RGB( 50, 100, 255 ) );
dl( VERTEX2II(80, 38, 31, 0x47) );
dl( RESTORE_CONTEXT() );
dl( VERTEX2II(110, 38, 31, 0x47) );
```

Graphics context

None

See also

RESTORE_CONTEXT

4.40 SCISSOR_SIZE

Specify the size of the scissor clip rectangle

Encoding

31 24	23 12	11 0
0x1C	width	height

Parameters

width

The width of the scissor clip rectangle, in pixels. The initial value is 2048.

The value of zero will cause zero output on screen.

The valid range is from zero to 2048.

height

The height of the scissor clip rectangle, in pixels. The initial value is 2048.

The value of zero will cause zero output on screen.

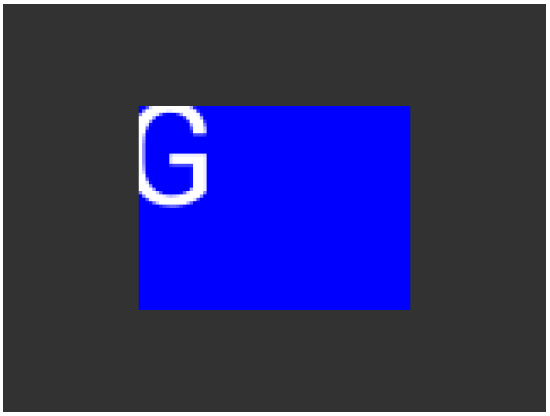
The valid range is from zero to 2048.

Description

Sets the width and height of the scissor clip rectangle, which limits the drawing area.

Examples

Setting a 40 x 30 scissor rectangle clips the clear and bitmap drawing:



```
dl( SCISSOR_XY(40, 30) );
dl( SCISSOR_SIZE(80, 60) );
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( CLEAR(1, 1, 1) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(35, 20, 31, 0x47) );
```

Graphics context

The values of width and height are part of the graphics context 4.1

See also

None

4.41 SCISSOR_XY

Specify the top left corner of the scissor clip rectangle

Encoding

31 24	23 22	21 11	10 0
0x1B	reserved	x	y

Parameters

x

The unsigned x coordinate of the scissor clip rectangle, in pixels. The initial value is 0. The valid range is from zero to 2047.

y

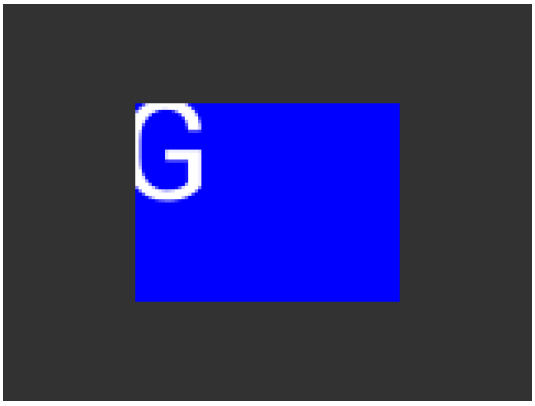
The unsigned y coordinates of the scissor clip rectangle, in pixels. The initial value is 0. The valid range is from zero to 2047.

Description

Sets the top-left position of the scissor clip rectangle, which limits the drawing area.

Examples

Setting a 40 x 30 scissor rectangle clips the clear and bitmap drawing:




```
dl( SCISSOR_XY(40, 30) );
dl( SCISSOR_SIZE(80, 60) );
dl( CLEAR_COLOR_RGB(0, 0, 255) );
dl( CLEAR(1, 1, 1) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(35, 20, 31, 0x47) );
```

Graphics context

The values of x and y are part of the graphics context 4.1

See also

None

4.42 STENCIL_FUNC

Set function and reference value for stencil testing

Encoding

31 24	23 20	19 16	15 8	7 0
0x0A	Reserved	func	ref	mask

Parameters

func

Specifies the test function, one of NEVER, LESS, LEQUAL, GREATER, GEQUAL, EQUAL, NOTEQUAL, or ALWAYS. The initial value is ALWAYS. About the value of these constants, please check Figure 5: The constants of ALPHA_FUNC

ref

Specifies the reference value for the stencil test. The initial value is 0

mask

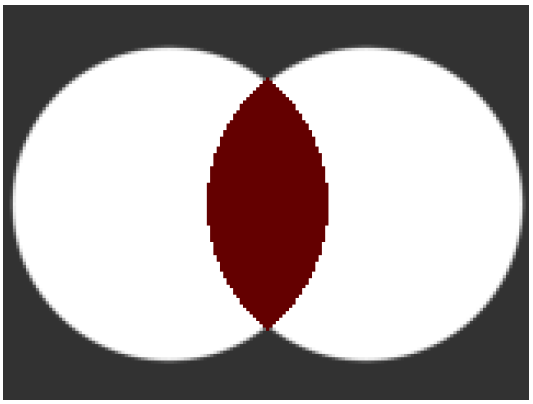
Specifies a mask that is ANDed with the reference value and the stored stencil value. The initial value is 255

Description

Stencil test rejects or accepts pixels depending on the result of the test function defined in func parameter, which operates on the current value in the stencil buffer against the reference value.

Examples

Draw two points, incrementing stencil at each pixel, then draw the pixels with value 2 in red:



```
dl( STENCIL_OP(INCR, INCR) );
dl( POINT_SIZE(760) );
dl( BEGIN(POINTS) );
dl( VERTEX2II(50, 60, 0, 0) );
dl( VERTEX2II(110, 60, 0, 0) );
dl( STENCIL_FUNC(EQUAL, 2, 255) );
dl( COLOR_RGB(100, 0, 0) );
dl( VERTEX2II(80, 60, 0, 0) );
```

Graphics context

The values of func, ref and mask are part of the graphics context, as described in section 4.1

See also

STENCIL_OP, STENCIL_MASK

4.43 STENCIL_MASK

Control the writing of individual bits in the stencil planes

Encoding

31 24	23 8	7 0
0x13	reserved	mask

Parameters

mask

The mask used to enable writing stencil bits. The initial value is 255

Graphics context

The value of mask is part of the graphics context, as described in section 4.1

See also

STENCIL_FUNC, STENCIL_OP, TAG_MASK

4.44 STENCIL_OP

Set stencil test actions

Encoding

31 24	23 6	5 3	2 0
0x0C	reserved	sfail	spass

Parameters

sfail

Specifies the action to take when the stencil test fails, one of KEEP, ZERO, REPLACE, INCR, DECR, and INVERT. The initial value is KEEP (1)

spass

Specifies the action to take when the stencil test passes, one of the same constants as sfail. The initial value is KEEP (1)

NAME	VALUE
ZERO	0
KEEP	1
REPLACE	2
INCR	3
DECR	4
INVERT	5

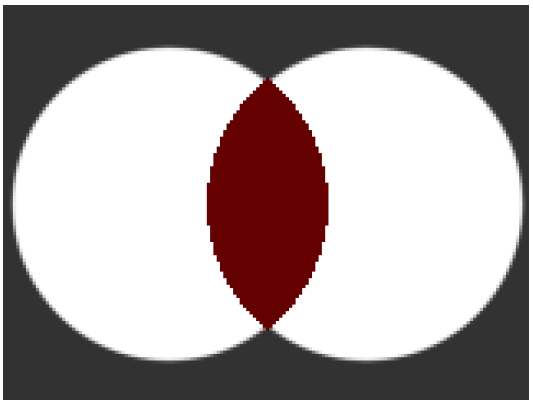
Figure 10: STENCIL_OP constants definition

Description

The stencil operation specifies how the stencil buffer is updated. The operation selected depends on whether the stencil test passes or not.

Examples

Draw two points, incrementing stencil at each pixel, then draw the pixels with value 2 in red:



```
dl( STENCIL_OP(INCR, INCR) );
dl( POINT_SIZE(760) );
dl( BEGIN(POINTS) );
dl( VERTEX2II(50, 60, 0, 0) );
dl( VERTEX2II(110, 60, 0, 0) );
dl( STENCIL_FUNC(EQUAL, 2, 255) );
dl( COLOR_RGB(100, 0, 0) );
dl( VERTEX2II(80, 60, 0, 0) );
```

Graphics context

The values of sfail and spass are part of the graphics context, as described in section 4.1

See also

STENCIL_FUNC, STENCIL_MASK

4.45 TAG

Attach the tag value for the following graphics objects drawn on the screen. The initial tag buffer value is 255.

Encoding

31 24	23 8	7 0
0x03	Reserved	s

Parameters

s

Tag value. Valid value range is from 1 to 255.

Description

The initial value of the tag buffer of the FT81X is specified by command **CLEAR_TAG** and takes effect by issuing command **CLEAR**. The TAG command can specify the value of the tag buffer of the FT81X that applies to the graphics objects when they are drawn on the screen. This TAG value will be assigned to all the following objects, unless the TAG_MASK command is used to disable it. Once the following graphics objects are drawn, they are attached with the tag value successfully. When the graphics objects attached with the tag value are touched, the register REG_TOUCH_TAG will be updated with the tag value of the graphics object being touched.

If there are no TAG commands in one display list, all the graphics objects rendered by the display list will report the tag value as 255 in REG_TOUCH_TAG when they are touched.

Graphics context

The value of s is part of the graphics context, as described in section 4.1

See also

CLEAR_TAG, TAG_MASK

4.46 TAG_MASK

Control the writing of the tag buffer

Encoding

31 24	23 1	0
0x14	Reserved	mask

Parameters

mask

Allow updates to the tag buffer. The initial value is one and it means the tag buffer of the FT81X is updated with the value given by the TAG command. Therefore, the following graphics objects will be attached to the tag value given by the TAG command.

The value zero means the tag buffer of the FT81X is set as the default value, rather than the value given by TAG command in the display list.

Description

Every graphics object drawn on screen is attached with the tag value which is defined in the FT81X tag buffer. The FT81X tag buffer can be updated by the TAG command.

The default value of the FT81X tag buffer is determined by CLEAR_TAG and CLEAR commands. If there is no CLEAR_TAG command present in the display list, the default value in tag buffer shall be 0.

TAG_MASK command decides whether the FT81X tag buffer takes the value from the default value of the FT81X tag buffer or the TAG command of the display list.

Graphics context

The value of mask is part of the graphics context, as described in section 4.1

See also

TAG, CLEAR_TAG, STENCIL_MASK, COLOR_MASK

4.47 VERTEX2F

Start the operation of graphics primitives at the specified screen coordinate, in the pixel precision defined by VERTEX_FORMAT.

Encoding

31 30	29 15	14 0
0x1	X	Y

Parameters

X

Signed x-coordinate in units of pixel precision defined in command VERTEX_FORMAT, which by default is 1/16 pixel precision.

Y

Signed y-coordinate in units of pixel precision defined in command VERTEX_FORMAT, which by default is 1/16 pixel precision.

Description

The pixel precision depends on the value of VERTEX_FORMAT. The maximum range of coordinates depends on pixel precision and is described in the VERTEX_FORMAT instruction.

Graphics context

None

See also

VERTEX_FORMAT

4.48 VERTEX2II

Start the operation of graphics primitive at the specified coordinates in pixel precision.

Encoding

31 30	29 21	20 12	11 7	6 0
0x2	X	Y	handle	cell

Parameters

x

x-coordinate in pixels, from 0 to 511.

y

y-coordinate in pixels, from 0 to 511.

handle

Bitmap handle. The valid range is from 0 to 31.

cell

Cell number. Cell number is the index of the bitmap with same bitmap layout and format. For example, for handle 31, the cell 65 means the character "A" in built in font 31.

Description

The range of coordinates is from -16384 to +16383 in terms of single pixel unit. The handle and cell parameters are ignored unless the graphics primitive is specified as bitmap by command **BEGIN**, prior to this command.

Graphics context

None

See also

BITMAP_HANDLE, CELL

4.49 VERTEX_FORMAT

Set the precision of VERTEX2F coordinates

Encoding

31 24	23 3	2 0
0x27	RESERVED	frac

Parameters

frac

Number of fractional bits in X,Y coordinates. Valid range is from 0 to 4. The initial value is 4.

Description

VERTEX2F uses 15 bit signed numbers for its (X,Y) coordinates. This command controls the interpretation of these numbers by specifying the number of fractional bits.

By varying the format, an application can trade range against precision.

Table 9 VERTEX_FORMAT and pixel precision

frac	Units in pixel precision	VERTEX2F range in pixels
0	1	-16384 to 16383
1	1/2	-8192 to 8191
2	1/4	-4096 to 4095
3	1/8	-2048 to 2047
4	1/16	-1024 to 1023

Graphics context

The value of **frac** is part of the graphics context

See also

VERTEX_2F

4.50 VERTEX_TRANSLATE_X

Specify the vertex transformation's X translation component

Encoding

31 24	23 17	16 0
0x2B	RESERVED	x

Parameters

x

signed x-coordinate in 1/16 pixel. The initial value is 0

Description

Specifies the offset added to vertex X coordinates. This command allows drawing to be shifted on the screen.

Graphics context

The value of x is part of the graphics context

See also

NONE

4.51 VERTEX_TRANSLATE_Y

Specify the vertex transformation's Y translation component

Encoding

31 24	23 17	16 0
0x2C	RESERVED	Y

Parameters

y

signed y-coordinate in 1/16 pixel. The initial value is 0

Description

Specifies the offset added to vertex Y coordinates. This command allows drawing to be shifted on the screen.

Graphics context

The value of *y* is part of the graphics context

See also

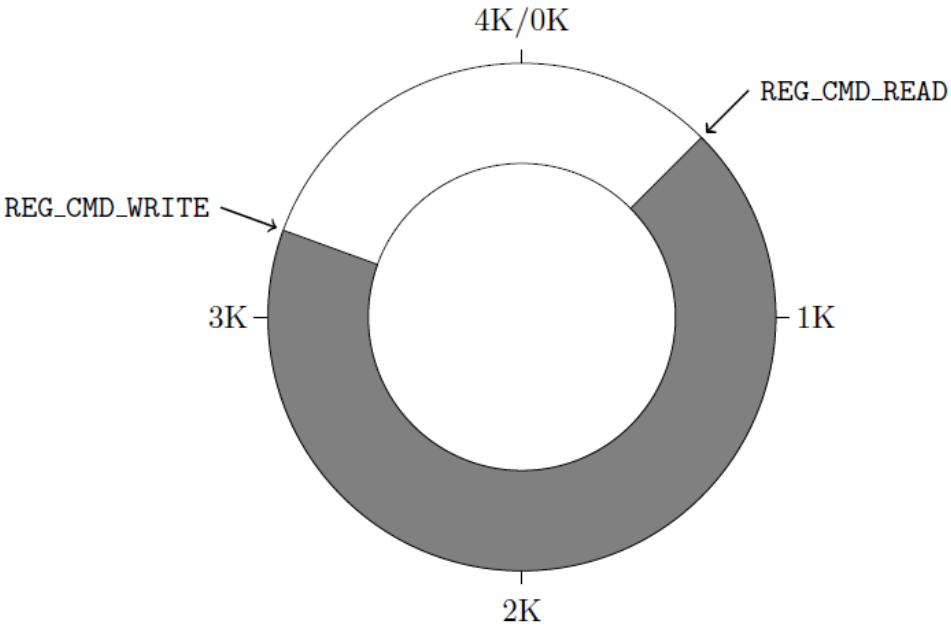
NONE

5 Co-Processor Engine

5.1 Command Interface

5.1.1 Circular Buffer

The co-processor engine is fed via a 4 Kbyte circular buffer in RAM_CMD. The host MCU writes co-processor commands or display list commands into the circular buffer, and the co-processor engine reads and executes the commands. The MCU updates the register **REG_CMD_WRITE** to indicate that there are new commands in the circular buffer, and the co-processor engine updates **REG_CMD_READ** after the commands have been executed. Therefore, when **REG_CMD_WRITE** is equal to **REG_CMD_READ**, it indicates the circular buffer is empty and all the commands are executed without error.



To compute the free space, the MCU can apply the following formula:

fullness = (REG_CMD_WRITE -REG_CMD_READ) *mod* 4096

free space = (4096 - 4) -*fullness*;

This calculation does not report 4096 bytes of free space, to prevent completely wrapping the circular buffer and making it appear empty.

If enough space is available in the FIFO, the MCU writes the commands at the appropriate location, and then updates REG_CMD_WRITE. To simplify the MCU code, the FT81X automatically wraps continuous writes from the top address (RAM_CMD + 4095) back to the bottom address (RAM_CMD + 0) if the starting address of a write transfer is within RAM_CMD.

FIFO entries are always 4 bytes wide - it is an error for either **REG_CMD_READ** or **REG_CMD_WRITE** to have a value that is not a multiple of 4 bytes. Each command issued to the co-processor engine may take 1 or more words: the length depends on the command itself, and any appended data. Some commands are followed by variable-length data, so the command size

may not be a multiple of 4 bytes. In this case the co-processor engine ignores the extra 1, 2 or 3 bytes and continues reading the next command at the following 4 byte boundary.

5.1.2 Auxiliary Registers

To offload work from the MCU for checking the free space in the circular buffer, the FT81X offers two auxiliary registers "**REG_CMDB_SPACE**" and "**REG_CMDB_WRITE**" for bulk transfers. It enables the MCU to write commands and data to the co-processor in a bulk transfer, without computing the free space in the circular buffer and increasing the address. As long as the amount of data to be transferred is less than the value in the register "**REG_CMDB_SPACE**", the MCU is able to safely write all the data to "**REG_CMDB_WRITE**" in one write transfer.

5.2 Widgets

The Co-Processor engine of FT81X provides pre-defined widgets for users to construct screen designs easily. The picture below illustrates the commands to render widgets and effects.

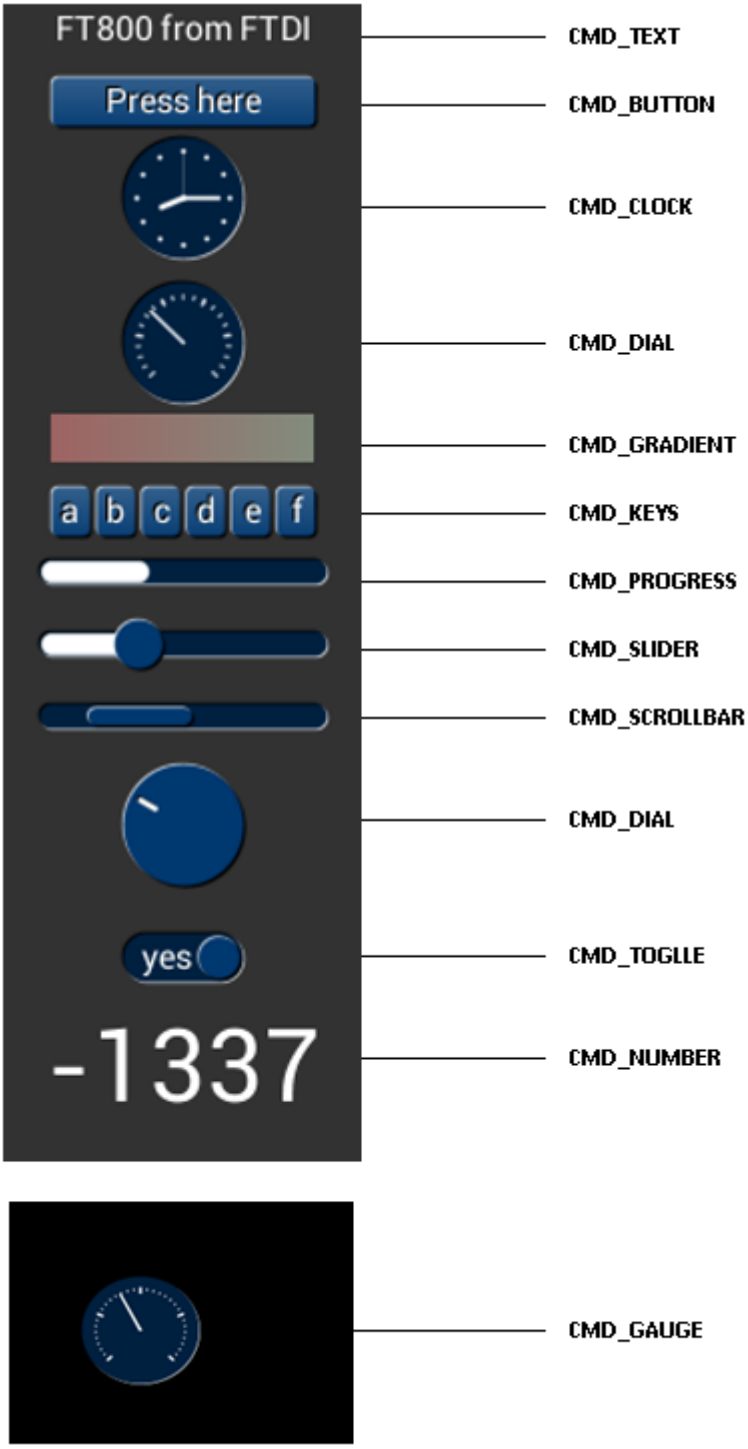


Figure 11: FT81X widget list

5.2.1 Common Physical Dimensions

This section contains the common physical dimensions of the widgets, unless it is specified in the widget introduction.

- All rounded corners have a radius that is computed from the font used for the widget (curvature of lowercase 'o' character).
  *radius = font height * 3 / 16*
- All 3D shadows are drawn with:
  (1) highlight offset 0.5 pixels above and left of the object
  (2) shadow offset 1.0 pixel below and right of the object.
- For widgets such as progress bar, scrollbar and slider, the output will be a vertical widget in the case where width and height parameters are of same value.

5.2.2 Color Settings

Co-processor engine widgets are drawn with the color designated by the precedent commands: **CMD_FGCOLOR**, **CMD_BGCOLOR** and **COLOR_RGB**. The co-processor engine will determine to render the different areas of the widgets in different colors according to these commands.

Usually, **CMD_FGCOLOR** affects the interaction area of co-processor engine widgets if they are designed for interactive UI elements, for example, **CMD_BUTTON**, **CMD_DIAL**. **CMD_BGCOLOR** applies the background color of widgets with the color specified. Please see the table below for more details.

Table 10 Widgets color setup table

Widget	CMD_FGCOLOR	CMD_BGCOLOR	COLOR_RGB
CMD_TEXT	NO	NO	YES
CMD_BUTTON	YES	NO	YES(label)
CMD_GAUGE	NO	YES	YES(needle and mark)
CMD_KEYS	YES	NO	YES(text)
CMD_PROGRESS	NO	YES	YES
CMD_SCROLLBAR	YES(Inner bar)	YES(Outer bar)	NO
CMD_SLIDER	YES(Knob)	YES(Right bar of knob)	YES(Left bar of knob)
CMD_DIAL	YES(Knob)	NO	YES(Marker)
CMD_TOGGLE	YES(Knob)	YES(Bar)	YES(Text)
CMD_NUMBER	NO	NO	YES
CMD_CALIBRATE	YES(Animating dot)	YES(Outer dot)	NO
CMD_SPINNER	NO	NO	YES

5.2.3 Caveat

The behavior of widgets is not defined if the input parameter values are outside the valid range.

5.3 Interaction with RAM_DL

If the co-processor command is to generate respective display list commands, the co-processor engine will write them to RAM_DL. The current write location in RAM_DL is held in the register **REG_CMD_DL**. Whenever the co-processor engine writes a word to the display list, it increments **REG_CMD_DL**. The special command **CMD_DLSTART** sets **REG_CMD_DL** to zero, for the start of a new display list.

All display list commands can also be written to the co-processor engine circular buffer. The co-processor engine has the intelligence to differentiate and copy them into the current display list location specified by **REG_CMD_DL**. For example, the following code snippet writes a small display list:

```
cmd(CMD_DLSTART); // start a new display list
cmd(CLEAR_COLOR_RGB(255, 100, 100)); // set clear color
cmd(CLEAR(1, 1, 1)); // clear screen

cmd(DISPLAY()); // display
```

Of course, this display list could have been written directly to RAM_DL. The advantage of this technique is that you can mix low-level operations and high level co-processor engine commands in a single stream:

```
cmd(CMD_DLSTART);                // start a new display list
cmd(CLEAR_COLOR_RGB(255, 100, 100)); // set clear color
cmd(CLEAR(1, 1, 1));                    // clear screen
cmd_button(20, 20,               // x, y
           60, 60,               // width, height in pixels
               30,               // font 30
           0,                    // default options
           "OK!");
```

5.4 Synchronization

At some points, it is necessary to wait until the co-processor engine has processed all outstanding commands. When the co-processor engine completes the last outstanding command in the command buffer, it raises the **INT_CMDEMPTY** interrupt. Another approach to detecting synchronization is that the MCU can poll **REG_CMD_READ** until it is equal to **REG_CMD_WRITE**.

One situation that requires synchronization is to read the value of **REG_CMD_DL**, when the MCU needs to do direct writes into the display list. In this situation the MCU should wait until the co-processor engine is idle before reading **REG_CMD_DL**.

5.5 ROM and RAM Fonts

In the FT81X, fonts are referring to bitmap-based fonts and used by the following co-processor commands, indexed by bitmap handles 0 to 31:

- CMD_BUTTON
- CMD_KEYS
- CMD_TOGGLE
- CMD_TEXT
- CMD_NUMBER

Each font supports not more than 128 characters, with index from 0 to 127.

5.5.1 Font Metrics Block

For each font, there is one 148-bytes font metrics block associated with it.

The format of the 148-bytes font metrics block is as below:

Table 11 FT81X Font metrics block format

Address	Size	Value
p + 0	128	width of each font character, in pixels
p + 128	4	font bitmap format, for example L1,L2, L4 or L8
p + 132	4	font line stride, in bytes
p + 136	4	font width, in pixels
p + 140	4	font height, in pixels
p + 144	4	pointer to font graphic data in memory

For ROM fonts, these blocks are located in ROM, pointed to by a 32 bit address stored in ROM_FONTROOT. In the FT81X, ROM_FONTROOT is defined as "0x2FFFFC", which stores value "0x201EE0".

For RAM fonts, these blocks shall be located in RAM_G. Users can call **CMD_SETFONT/CMD_SETFONT2** to indicate to the FT81X co-processor engine the address of these metrics blocks.

5.5.1.1 Example

To find the width of character 'g' (ASCII 0x67) in ROM font 34:

read 32-bit pointer *p* from ROM_FONTROOT

widths = *p* + (148 * (34 - 16)) (table starts at font 16)

read byte from memory at widths[0x67]

5.5.2 ROM Fonts (Built-in Fonts)

The FT81X has ROM to support built-in bitmap fonts. In total, there are 19 ROM fonts numbered from 16 to 34.

By default, ROM fonts 16 to 31 are attached to bitmap handles 16 to 31 and users may use these fonts by specifying bitmap handle from 16 to 31.

To use ROM font 32 to 34, the user needs to call CMD_ROMFONT to assign the bitmap handle with the ROM font number. Refer to CMD_ROMFONT for more details.

For ROM fonts 16 to 34(except 17 and 19), each font includes 95 printable ASCII characters from 0x20 to 0x7E inclusive. All these characters are indexed by its corresponding ASCII value.

For ROM fonts 17 and 19, each font includes 127 printable ASCII characters from 0x80 to 0xFF, inclusive. All these characters are indexed using value from 0x0 to 0x7F, i.e., code 0 maps to ASCII character 0x80 and code 0x7F maps to ASCII character 0xFF. Users are required to handle this mapping manually.

The picture below shows the ROM font effects in the FT81X.

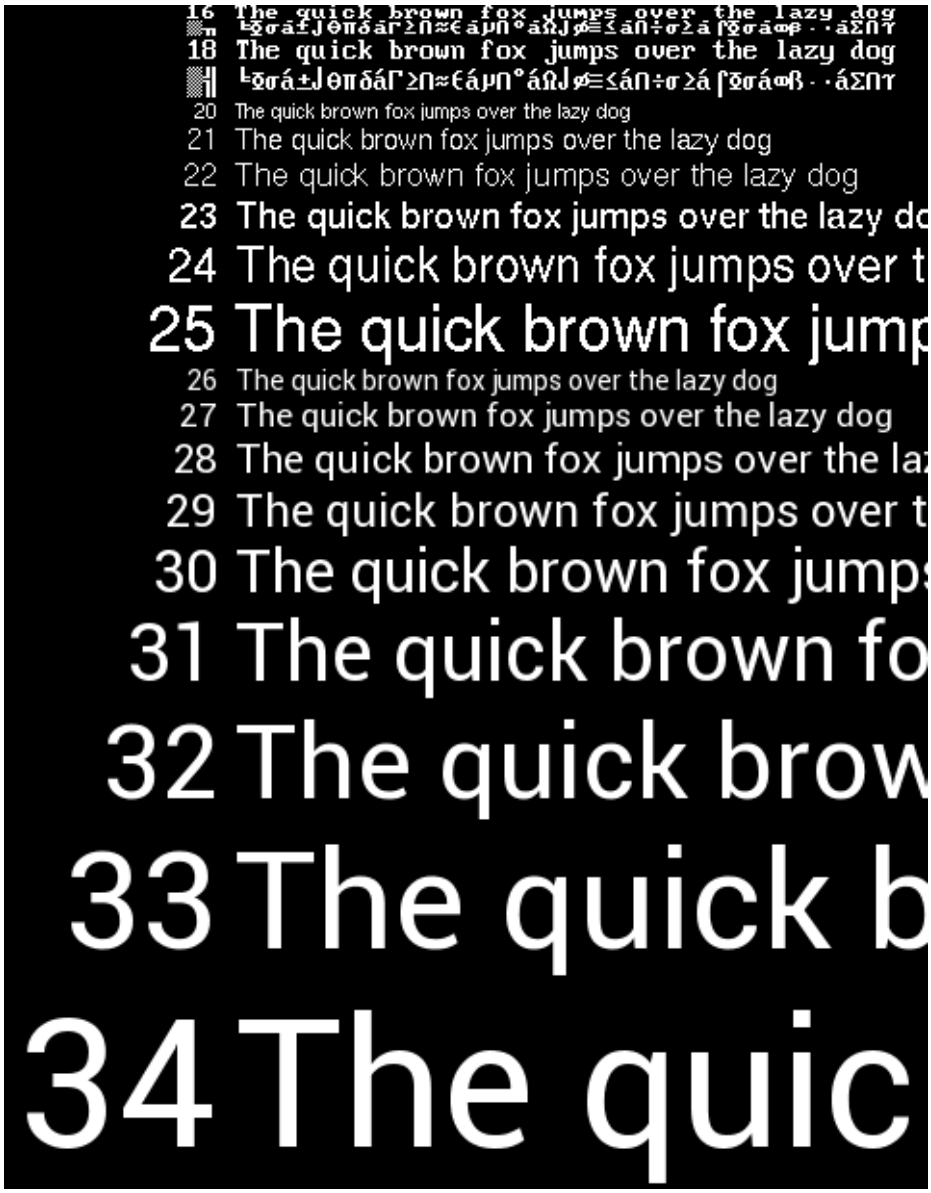


Figure 12: FT81X ROM Font List

5.5.3 RAM Fonts (Custom Fonts)

Users can define their own bitmap fonts in RAM_G by following the steps below:

- Download bitmap data into RAM_G

- Set up bitmap parameters using display list commands **BITMAP_SOURCE\BITMAP_LAYOUT\BITMAP_SIZE** or using the co-processor command **CMD_SETBITMAP**.
- Create and download the font metrics block in RAM_G. The address of the metrics block must be **4 bytes aligned**.
- Use the command **CMD_SETFONT** or **CMD_SETFONT2** to associate the new font with the selected bitmap handle.
- Use the selected bitmap handle in any co-processor command font argument.

5.6 Fault Scenarios

Some commands can cause co-processor engine faults. These faults arise because the co-processor engine cannot continue. For example:

- An invalid JPEG is supplied to **CMD_LOADIMAGE**
- An invalid data stream is supplied to **CMD_INFLATE**
- An attempt is made to write more than 2048 instructions into a display list

In the fault condition, the co-processor engine sets **REG_CMD_READ** to 0xfff (an illegal value because all command buffer data shall be 32-bit aligned), raises the **INT_CMDEMPTY** interrupt, and stops accepting new commands. When the host MCU recognizes the fault condition, it should recover as follows:

- Set **REG_CPURESET** to 1, to hold the co-processor engine in the reset condition
- Set **REG_CMD_READ** and **REG_CMD_WRITE** to zero
- Set **REG_CPURESET** to 0, to restart the co-processor engine

5.7 Graphics State

The co-processor engine maintains a small amount of internal states for graphics drawing. This state is set to the default at co-processor engine reset, and by **CMD_COLDSTART**. The state values are not affected by **CMD_DLSTART** or **CMD_SWAP**, so an application need only set them once at startup.

Table 12 Co-processor engine graphics state

State	Default	Commands
background color	*dark blue (0x002040)*	CMD_BGCOLOR
foreground color	*light blue (0x003870)*	CMD_FGCOLOR
gradient color	*white (0xffffff)*	CMD_GRADCOLOR
Spinner	*None*	CMD_SPINNER
object trackers	*all disabled*	CMD_TRACK
interrupt timer	*None*	CMD_INTERRUPT
Bitmap transform matrix: $\begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix}$	CMD_LOADIDENTITY,CMD_TRANSLATE, CMD_ROTATE.
Bitmap Handle	*15 or any handle set by CMD_SETSCRATCH*	CMD_GRADCOLOR, CMD_KEYS, CMD_BUTTON
base of number	*10*	CMD_SETBASE

Media FIFO	*Address is zero and length is zero*	CMD_MIDEAFIFO, CMD_PLAYVIDEO

5.8 Parameter "OPTION"

The following table defines the parameter "OPTION" mentioned in this chapter.

Table 13 Parameter OPTION definition

Name	Value	Description	Commands
OPT_3D	0	3D effect	CMD_BUTTON,CMD_CLOCK, CMD_KEYS, CMD_GAUGE,CMD_SLIDER, CMD_DIAL, CMD_TOGGLE,CMD_PROGRESS, CMD_SCROLLBAR
OPT_RGB565	0	Decode the source image to RGB565 format	CMD_LOADIMAGE
OPT_MONO	1	Decode the source JPEG image to L8 format, i.e., monochrome	CMD_LOADIMAGE
OPT_NODL	2	No display list commands generated	CMD_LOADIMAGE
OPT_FLAT	256	No 3D effect	CMD_BUTTON,CMD_CLOCK,CMD_KEYS, CMD_GAUGE,CMD_SLIDER, CMD_DIAL, CMD_TOGGLE,CMD_PROGRESS, CMD_SCROLLBAR
OPT_SIGNED	256	The number is treated as a 32 bit signed integer	CMD_NUMBER
OPT_CENTERX	512	Horizontally-centred style	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTERY	1024	Vertically centred style	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTER	1536	horizontally and vertically centred style	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_RIGHTX	2048	Right justified style	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_NOBACK	4096	No background drawn	CMD_CLOCK, CMD_GAUGE
OPT_NOTICKS	8192	No Ticks	CMD_CLOCK, CMD_GAUGE

Name	Value	Description	Commands
OPT_NOHM	16384	No hour and minute hands	CMD_CLOCK
OPT_NOPOINTER	16384	No pointer	CMD_GAUGE
OPT_NOSECS	32768	No second hands	CMD_CLOCK
OPT_NOHANDS	49152	No hands	CMD_CLOCK
OPT_NOTEAR	4	Synchronize video updates to the display blanking interval, avoiding horizontal ”tearing” artefacts.	CMD_PLAYVIDEO
OPT_FULLSCREEN	8	zoom the video so that it fills as much of the screen as possible.	CMD_PLAYVIDEO
OPT_MEDIAFIFO	16	source video data from the defined media FIFO	CMD_PLAYVIDEO
OPT_SOUND	32	Decode the audio data	CMD_PLAYVIDEO

5.9 Resources Utilization

The co-processor engine does not change the state of the graphics engine. That is, graphics states such as color and line width are not to be changed by the co-processor engine.

However, the widgets do reserve some hardware resources, which the user must take into account:

- Bitmap handle 15 is used by the 3D-effect buttons, keys and gradient, unless it is set to another bitmap handle using **CMD_SETSCRATCH**.
- One graphics context is used by objects, and the effective stack depth for **SAVE_CONTEXT** and **RESTORE_CONTEXT** commands is 3 levels.

5.10 Command Groups

These commands begin and finish the display list:

- **CMD_DLSTART** - start a new display list
- **CMD_SWAP** - swap the current display list

Commands to draw graphics objects:

- **CMD_TEXT** - draw text
- **CMD_BUTTON** - draw a button
- **CMD_CLOCK** - draw an analog clock
- **CMD_BGCOLOR** - set the background color
- **CMD_FGCOLOR** - set the foreground color
- **CMD_GRADCOLOR** – set up the highlight color used in 3D effects for **CMD_BUTTON** and **CMD_KEYS**
- **CMD_GAUGE** - draw a gauge

- **CMD_GRADIENT** - draw a smooth color gradient
- **CMD_KEYS** - draw a row of keys
- **CMD_PROGRESS** - draw a progress bar
- **CMD_SCROLLBAR** - draw a scroll bar
- **CMD_SLIDER** - draw a slider
- **CMD_DIAL** - draw a rotary dial control
- **CMD_TOGGLE** - draw a toggle switch
- **CMD_NUMBER** - draw a decimal number

Commands to operate on memory:

- **CMD_MEMCRC** - compute a CRC-32 for memory
- **CMD_MEMZERO** - write zero to a block of memory
- **CMD_MEMSET** - fill RAM_G with a byte value
- **CMD_MEMWRITE** - write bytes into RAM_G
- **CMD_MEMCPY** - copy a block of RAM_G
- **CMD_APPEND** - append more commands to display list

Commands for loading image data into FT81X RAM_G:

- **CMD_INFLATE** - decompress data into memory
- **CMD_LOADIMAGE** - load a JPEG/PNG image

Commands for setting the bitmap transform matrix:

- **CMD_LOADIDENTITY** - set the current matrix to identity
- **CMD_TRANSLATE** - apply a translation to the current matrix
- **CMD_SCALE** - apply a scale to the current matrix
- **CMD_ROTATE** - apply a rotation to the current matrix
- **CMD_SETMATRIX** - write the current matrix as a bitmap transform
- **CMD_GETMATRIX** - retrieves the current matrix coefficients

Other commands:

- **CMD_COLDSTART** - set co-processor engine state to default values
- **CMD_INTERRUPT** - trigger interrupt INT_CMDFLAG
- **CMD_REGREAD** - read a register value
- **CMD_CALIBRATE** - execute the touch screen calibration routine
- **CMD_ROMFONT** – load a ROM font into bitmap handle
- **CMD_SETROTATE** - Rotate the screen and set up transform matrix accordingly
- **CMD_SETBITMAP** – Set up display list commands for specified bitmap
- **CMD_SPINNER** - start an animated spinner
- **CMD_STOP** - stop any spinner, screensaver or sketch
- **CMD_SCREENSAVER** - start an animated screensaver
- **CMD_SKETCH** - start a continuous sketch update
- **CMD_SNAPSHOT** - take a snapshot of the current screen
- **CMD_SNAPSHOT2** - take a snapshot of part of the current screen with more format option
- **CMD_LOGO** - play device logo animation

5.11 CMD_DLSTART - start a new display list

When the co-processor engine executes this command, it waits until the current display list is scanned out, and then sets REG_CMD_DL to zero.

C prototype

void cmd_dlstart( );

Command layout

+0	CMD_DLSTART (0xffffff00)

Examples

```
cmd_dlstart();
//...
cmd_dlswap();
```

5.12 CMD_SWAP - swap the current display list

When the co-processor engine executes this command, it requests a display list swap immediately after the current display list is scanned out. Internally, the co-processor engine implements this command by writing to REG_DLSWAP. Refer to the REG_DLSWAP Definition.

This co-processor engine command will not generate any display list command into display list memory RAM_DL.

C prototype

void cmd_swap( );

Command layout

+0	CMD_DLSWAP(0xffffff01)

Examples

None

5.13 CMD_COLDSTART - set co-processor engine state to default values

This command sets the co-processor engine to default reset states.

C prototype

void cmd_coldstart( );

Command layout

+0	CMD_COLDSTART(0xffffff32)

Examples

Change to a custom color scheme, and then restore the default colors:

```
cmd_fgcolor(0x00c040);
cmd_gradcolor(0x000000);
cmd_button( 2, 32, 76, 56, 26,0, "custom" );
cmd_coldstart();
cmd_button( 82, 32, 76, 56, 26,0, "default");
```

5.14 CMD_INTERRUPT - trigger interrupt INT_CMDFLAG

When the co-processor engine executes this command, it triggers interrupt INT_CMDFLAG.

C prototype

void cmd_interrupt(uint32_t ms);

Parameters

ms

The delay before the interrupt triggers, in milliseconds. The interrupt is guaranteed not to fire before this delay. If ms are zero, the interrupt fires immediately.

Command layout

+0	CMD_INTERRUPT(0xffffff02)
+4	Ms

Examples

```
//To trigger an interrupt after a JPEG has
finished loading:
cmd_loadimage();
//...
cmd_interrupt(0); // previous load image
complete, trigger interrupt

//To trigger an interrupt in 0.5 seconds:
cmd_interrupt(500);
//...
```

5.15 CMD_APPEND - append more commands to current display list

Appends more commands resident in RAM_G to the current display list memory address where the offset is specified in REG_CMD_DL.

C prototype

```
void cmd_append( uint32_t ptr,
                 uint32_t num );
```

Parameters

ptr

Starting address of source commands in RAM_G

num

Number of bytes to copy. This must be a multiple of 4.

Command layout

+0	CMD_APPEND(0xffffff1e)
+4	ptr
+8	num

Description

After appending is done, the co-processor engine will increase the **REG_CMD_DL** by num to make sure the display list is in order.

Examples

```
cmd_dlstart();
cmd_append(0, 40); // copy 10 commands from main memory address 0
cmd(DISPLAY); // finish the display list

cmd_swap();
```

5.16 CMD_REGREAD - read a register value

C prototype

void cmd_regread(uint32_t ptr,
uint32_t result);

Parameters

ptr

Address of the register to be read

result

The register value to be read at ptr address.

Command layout

+0	CMD_REGREAD(0xffffff19)
+4	Ptr
+8	Result

Examples

```
//To capture the exact time when a command completes:
uint16_t x = rd16(REG_CMD_WRITE);
cmd_regread(REG_CLOCK, 0);
//...
printf("%08x\n", rd32(RAM_CMD + x + 8));
```

5.17 CMD_MEMWRITE - write bytes into memory

Writes the following bytes into the FT81X memory. This command can be used to set register values, or to update memory contents at specific times.

C prototype

```
void cmd_memwrite( uint32_t ptr,
                   uint32_t num );
```

Parameters

ptr

The memory address to be written

num

Number of bytes to be written.

Description

The data byte should immediately follow in the command buffer. If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command, these padding bytes can have any value. The completion of this function can be detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Caution: if using this command, it may corrupt the memory of the FT81X if used improperly.

Command layout

+0	CMD_MEMWRITE(0xffffff1a)
+4	ptr
+8	Num
+12	Byte0
+13	Byte1
..	..
+n	..

Examples

```
//To change the backlight brightness to 64 (half intensity) for a
particular screen shot:
//...
cmd_swap(); // finish the display list
cmd_dlstart(); // wait until after the swap
cmd_memwrite(REG_PWM_DUTY, 4); // write to the PWM_DUTY register
cmd(100);
```

5.18 CMD_INFLATE - decompress data into memory

Decompress the following compressed data into RAM_G. The data should have been compressed with the DEFLATE algorithm, e.g. with the ZLIB library. This is particularly useful for loading graphics data.

C prototype

void cmd_inflate(uint32_t ptr);

Parameters

ptr

Destination address. The data byte should immediately follow in the command buffer.

Description

If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command. These padding bytes can have any value

Command layout

+0	CMD_INFLATE(0xffffff22)
+4	Ptr
+8	byte0
+9	byte1
..	..
+n	..

Examples

To load graphics data to main memory address 0x8000:

```
cmd_inflate(0x8000);
// zlib-compressed data follows
```

5.19 CMD_LOADIMAGE - load a JPEG or PNG image

Decompress the following JPEG or PNG image data into an FT81X bitmap, in RAM_G. The image data should be in the following formats:

- Regular baseline JPEG (JFIF)
- PNG, except Adam-7 interlaced images
    - Only bit-depth 8 is supported, bit-depths 1, 2, 4, and 16 are not.

The PNG standard defines several image color formats. Each format is loaded as a bitmap as follows:

Grayscale loads as **L8**
Truecolor loads as **RGB565**
Indexed loads as **PALETTED565** or **PALETTED4444**
Grayscale and alpha not supported
Truecolor and alpha loads as **ARGB4**

When loading indexed images, **CMD_LOADIMAGE** generates a **PALETTE_SOURCE** instruction to the display list to specify the palette. **CMD_LOADIMAGE** uses **PALETTED4444** if the indexed image contains transparency, or **PALETTED565** otherwise.

For JPEG images, the bitmap is loaded as either a **RGB565** or **L8** format bitmap, depending on the original image. If OPT_MONO is given, L8 is used.

C prototype

```
void cmd_loadimage( uint32_t ptr,
                    uint32_t options );
```

Parameters

ptr

Destination address

Options

By default, the loaded bitmap is in RGB565 format. Option OPT_MONO causes the bitmap to be monochrome, in L8 format. Option OPT_FULLSCREEN causes the bitmap to be scaled so that it fills as much of the screen as possible. If option OPT_MEDIAFIFO is given, the media FIFO is used for the image data (see below). The command appends display list commands to set the source, layout and size of the resulting image. Option OPT_NODL prevents this - nothing is written to the display list.

If OPT_MEDIAFIFO is not given, then the byte data should immediate follow in the command buffer.

Description

The data byte should immediately follow in the command buffer if OPT_MEDIAFIFO is NOT set. If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command. These padding bytes can have any value.

The application on the host processor has to parse the JPEG/PNG header to get the properties of the JPEG/PNG image and decide to decode.

Behavior is unpredictable in cases of non-baseline JPEG images or the output data generated is more than the RAM_G size.

Command layout

+0	CMD_LOADIMAGE(0xffffff24)	Mandatory
+4	ptr	Mandatory
+8	options	Mandatory
+12	byte0	Option
+13	Byte1	Option
..	..	Option
+n	..	Option

Examples

To load a JPEG image at address 0 then draw the bitmap at (10,20) and (100,20):

```
cmd_loadimage(0, 0);
...                                          // JPEG file data follows
cmd(BEGIN(BITMAPS))
cmd(VERTEX2II(10, 20, 0, 0));                // draw bitmap at (10,20)
cmd(VERTEX2II(100, 20, 0, 0));               // draw bitmap at (100,20)
```

5.20 CMD_MEDIAFIFO – set up a streaming media FIFO in RAM_G

Sets up a streaming media FIFO in RAM_G.

C prototype

```
void cmd_mediafifo ( uint32_t ptr,
                     uint32_t size );
```

Parameters

ptr

starting address of memory block

size

number of bytes in the source memory block

Command layout

+0	CMD_MEDIAFIFO (0xffffff39)
+4	Ptr

+8	Size

Examples

To set up a 64-Kbyte FIFO at the top of RAM_G for JPEG streaming, and report the initial values of the read and write pointers:

```
cmd_mediafifo(0x100000 - 65536, 65536);//0x100000 is the top of RAM_G
printf("R=%08xW=%08x\n",rd32(REG_MEDIAFIFO_READ),rd32(REG_MEDIAFIFO_WRITE));
```

prints:

000f000 00f000

5.21 CMD_PLAYVIDEO – Video playback

Plays back MJPEG-encoded AVI video

C prototype

void cmd_playvideo (uint32_t opts);

Parameters

opts

OPT_FULLSCREEN: zoom the video so that it fills as much of the screen as possible.

OPT_MEDIAFIFO: instead of sourcing the AVI video data from the command buffer, source it from the media FIFO.

OPT_NOTEAR: Synchronize video updates to the display blanking interval, avoiding horizontal ”tearing” artifacts.

OPT_SOUND: Decode the audio data encoded in the data following, if any.

data

The video data to be played. Optional when opts has OPT_MEDIAFIFO enabled.

Command layout

+0	CMD_PLAYVIDEO (0xffffff3a)
+4	Opts
+8~ +n	Data

Data following parameter “opts” shall be padded to 4 bytes aligned with zero.

Note

For the audio data encoded into AVI video , three formats are supported:

4 Bit IMA ADPCM, 8 Bit signed PCM, 8 Bit u-Law

In addition, 16 Bit PCM is partially supported by dropping off less significant 8 bits in each audio sample.

Examples

To play back an AVI video, full-screen:

```
cmd_playvideo(OPT_FULLSCREEN | OPT_NOTEAR);
//... append AVI data ...
```

5.22 CMD_VIDEOSTART – initialize video frame decoder

Initializes the AVI video decoder. The video data should be supplied using the media FIFO. This command processes the video header information from the media FIFO, and completes when it has consumed it

C prototype

void cmd_videostart( );

Parameters

None

Command layout

+0	CMD_VIDEOSTART (0xffffff40)

Examples

To load frames of video at address 4:

```
videostart();
videoframe(4, 0);
```

5.23 CMD_VIDEOFRAME - load the next frame of video

Loads the next frame of video. The video data should be supplied in the media FIFO. This command extracts the next frame of video from the media FIFO, and completes when it has consumed it.

C prototype

void cmd_videoframe(uint32_t dst,
uint32_t ptr);

Parameters

dst

Main memory location to load the frame data

ptr

Completion pointer. The command writes the 32-bit word at this location. It is set to 1 if there is at least one more frame available in the video. 0 indicates that this is the last frame.

Command layout

+0	CMD_VIDEOFRAME (0xffffff41)
+4	Dst
+8	Ptr

Examples

To load frames of video at address 4:

```
videostart();
do {
   videoframe(4, 0);
   //... display frame ...
} while (rd32(0) != 0);
```

5.24 CMD_MEMCRC - compute a CRC-32 for memory

Computes a CRC-32 for a block of FT81X memory

C prototype

void cmd_memcrc(uint32_t ptr,
uint32_t num,
uint32_t result);

Parameters

ptr

Starting address of the memory block

num

Number of bytes in the source memory block

result

Output parameter; written with the CRC-32 after command execution. The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Command layout

+0	CMD_MEMCRC(0xffffff18)
+4	Ptr
+8	Num

+12	Result

Examples

To compute the CRC-32 of the first 1K byte of FT81X memory, first record the value of REG_CMD_WRITE, execute the command, wait for completion, then read the 32-bit value at result:

```
uint16_t x = rd16(REG_CMD_WRITE);
cmd_crc(0, 1024, 0);
...
printf("%08x\n", rd32(RAM_CMD + x + 12));
```

5.25 CMD_MEMZERO - write zero to a block of memory

C prototype

```
void cmd_memzero( uint32_t ptr,
                  uint32_t num );
```

Parameters

ptr

Starting address of the memory block

num

Number of bytes in the memory block

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Command layout

+0	CMD_MEMZERO(0xffffff1c)
+4	Ptr
+8	Num

Examples

To erase the first 1K of main memory:

```
cmd_memzero(0, 1024);
```

5.26 CMD_MEMSET - fill memory with a byte value

C prototype

```
void cmd_memset( uint32_t ptr,
                 uint32_t value,
                 uint32_t num );
```

Parameters

ptr

Starting address of the memory block

value

Value to be written to memory

num

Number of bytes in the memory block

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Command layout

+0	CMD_MEMSET(0xffffff1b)
+4	Ptr
+8	Value
+12	Num

Examples

To write 0xff the first 1K of main memory:

```
cmd_memset(0, 0xff, 1024);
```

5.27 CMD_MEMCPY - copy a block of memory

C prototype

```
void cmd_memcpy(  uint32_t dest,
                  uint32_t src,
                  uint32_t num );
```

Parameters

dest

address of the destination memory block

src

address of the source memory block

num

number of bytes to copy

Description

The completion of this function is detected when the value of REG_CMD_READ is equal to REG_CMD_WRITE.

Command layout

+0	CMD_MEMCPY(0xffffff1d)
+4	dst
+8	src
+12	num

Examples

To copy 1K byte of memory from 0 to 0x8000:

```
cmd_memcpy(0x8000, 0, 1024);
```

5.28 CMD_BUTTON - draw a button

C prototype

```
void cmd_button( int16_t x,
                 int16_t y,
                 int16_t w,
                 int16_t h,
                 int16_t font,
                 uint16_t options,
                 const char* s );
```

Parameters

x

x-coordinate of button top-left, in pixels

y

y-coordinate of button top-left, in pixels

font

bitmap handle to specify the font used in the button label. See ROM and RAM Fonts.

options

By default, the button is drawn with a 3D effect and the value is zero. OPT_FLAT removes the 3D effect. The value of OPT_FLAT is 256.

s

button label. It must be one string terminated with null character, i.e. '\0' in C language.

Description

Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_BUTTON(0xffffff0d)
+4	X
+6	Y
+8	W
+10	H
+12	Font
+14	Options
+16	S
+17	..
..	..
+n	0

Examples

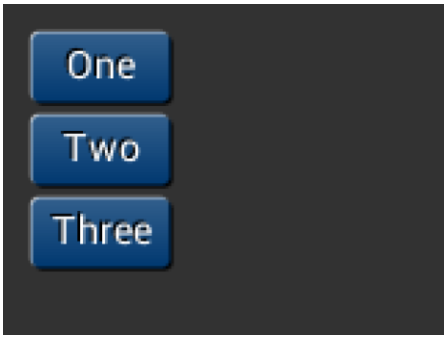
A 140x100 pixel button with large text:


```
cmd_button(10, 10, 140, 100, 31, 0, "Press!");
```

Without the 3D look:


```
cmd_button(10, 10, 140, 100, 31, OPT_FLAT, "Press!");
```

Several smaller buttons:




```
cmd_button(10, 10, 50, 25, 26, 0, "One");
cmd_button(10, 40, 50, 25, 26, 0, "Two");
cmd_button(10, 70, 50, 25, 26, 0, "Three");
```

Changing button color


```
cmd_fgcolor(0xb9b900),
cmd_button(10, 10, 50, 25, 26, 0, "Banana");
cmd_fgcolor(0xb97300),
cmd_button(10, 40, 50, 25, 26, 0, "Orange");
cmd_fgcolor(0xb90007),
cmd_button(10, 70, 50, 25, 26, 0, "Cherry");
```

5.29 CMD_CLOCK - draw an analog clock

C prototype

```
void cmd_clock( int16_t x,
                int16_t y,
                int16_t r,
                uint16_t options,
                uint16_t h,
                uint16_t m,
                uint16_t s,
                uint16_t ms );
```

Parameters

x

x-coordinate of clock center, in pixels

y

y-coordinate of clock center, in pixels

options

By default the clock dial is drawn with a 3D effect and the name of this option is OPT_3D. Option OPT_FLAT removes the 3D effect. With option OPT_NOBACK, the background is not drawn. With option OPT_NOTICKS, the twelve hour ticks are not drawn. With option OPT_NOSECS, the seconds hand is not drawn. With option OPT_NOHANDS, no hands are drawn. With option OPT_NOHM, no hour and minutes hands are drawn.

h

hours

m

minutes

s

seconds

ms

milliseconds

Description

The details of the physical dimensions are:

- The 12 tick marks are placed on a circle of radius r*(200/256).
- Each tick is a point of radius r*(10/256)
- The seconds hand has length r*(200/256) and width r*(3/256)
- The minutes hand has length r*(150/256) and width r*(9/256)
- The hours hand has length r*(100/256) and width r*(12/256)

Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_CLOCK(0xffffff14)
+4	X
+6	Y
+8	R
+10	Options
+12	H
+14	M
+16	S
+18	Ms

Examples

A clock with radius 50 pixels, showing a time of 8.15:

cmd_clock(80, 60, 50, 0, 8, 15, 0, 0);

Setting the background color

```
cmd_bgcolor(0x401010);
cmd_clock(80, 60, 50, 0, 8, 15, 0, 0);
```

Without the 3D look:

```
cmd_clock(80, 60, 50, OPT_FLAT, 8, 15, 0, 0);
```

The time fields can have large values. Here the hours are (7 x 3600s) and minutes are (38 x 60s), and seconds is 59. Creating a clock face showing the time as 7.38.59:

```
cmd_clock(
80, 60, 50, 0,
0, 0, (7 * 3600) + (38 * 60) + 59, 0);
```

No seconds hand:

cmd_clock(80, 60, 50, OPT_NOSECS, 8, 15, 0, 0);

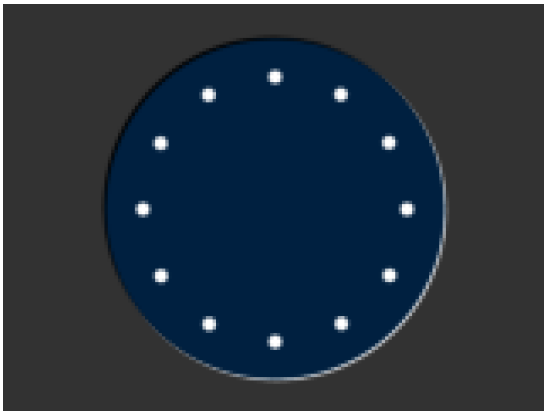
No background:

cmd_clock(80, 60, 50, OPT_NOBACK, 8, 15, 0, 0);

No ticks:

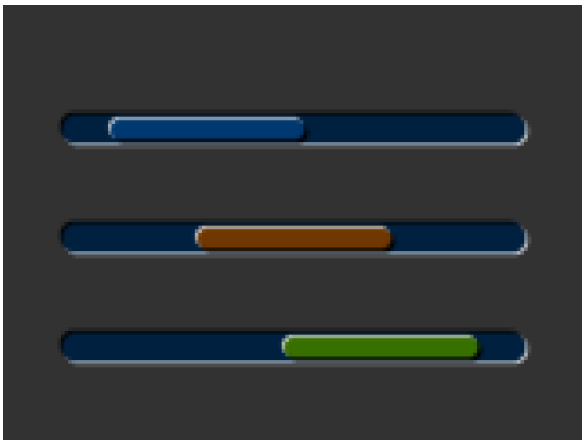
cmd_clock(80, 60, 50, OPT_NOTICKS, 8, 15, 0, 0);

No hands:



cmd_clock(80, 60, 50, OPT_NOHANDS, 8, 15, 0, 0);

5.30 CMD_FGCOLOR - set the foreground color



C prototype

void cmd_fgcolor(uint32_t c);

Parameters

c

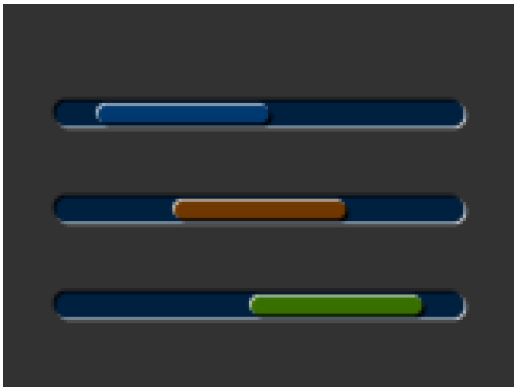
New foreground color, as a 24-bit RGB number. Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red. Foreground color is applicable for things that the user can move such as handles and buttons ("affordances").

Command layout

+0	CMD_FGCOLOR(0xffffff0a)
+4	C

Examples

The top scrollbar uses the default foreground color, the others with a changed color:



```
cmd_scrollbar(20, 30, 120, 8, 0, 10, 40, 100);
cmd_fgcolor(0x703800);
cmd_scrollbar(20, 60, 120, 8, 0, 30, 40, 100);
cmd_fgcolor(0x387000);
cmd_scrollbar(20, 90, 120, 8, 0, 50, 40, 100);
```

5.31 CMD_BGCOLOR - set the background color

C prototype

void cmd_bgcolor(uint32_t c);

Parameters

c

New background color, as a 24-bit RGB number. Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red.

Background color is applicable for things that the user cannot move E.g. behind gauges and sliders etc.

Command layout

+0	CMD_BGCOLOR(0xffffff09)
+4	C

Examples

The top scrollbar uses the default background color, the others with a changed color:

```
cmd_scrollbar(20, 30, 120, 8, 0, 10, 40, 100);
cmd_bgcolor(0x402000);
cmd_scrollbar(20, 60, 120, 8, 0, 30, 40, 100);
cmd_bgcolor(0x202020);
cmd_scrollbar(20, 90, 120, 8, 0, 50, 40, 100);
```

5.32 CMD_GRADCOLOR - set the 3D button highlight color

C prototype

void cmd_gradcolor(uint32_t c);

Parameters

c

New highlight gradient color, as a 24-bit RGB number. Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red.

Gradient is supported only for Button and Keys widgets.

Command layout

+0	CMD_GRADCOLOR(0xffffff34)
+4	C

Examples

Changing the gradient color: white (the default), red, green and blue


```
cmd_fgcolor(0x101010);
cmd_button( 2, 2, 76, 56, 31, 0, "W");
cmd_gradcolor(0xff0000);
cmd_button( 82, 2, 76, 56, 31, 0, "R");
cmd_gradcolor(0x00ff00);
cmd_button( 2, 62, 76, 56, 31, 0, "G");
cmd_gradcolor(0x0000ff);
cmd_button( 82, 62, 76, 56, 31, 0, "B");
```

The gradient color is also used for keys:


```
cmd_fgcolor(0x101010);
cmd_keys(10, 10, 140, 30, 26, 0, "abcde");
cmd_gradcolor(0xff0000);
cmd_keys(10, 50, 140, 30, 26, 0, "fghij");
```

5.33 CMD_GAUGE - draw a gauge

C prototype

```
void cmd_gauge( int16_t x,
                int16_t y,
                int16_t r,
                uint16_t options,
                uint16_t major,
                uint16_t minor,
                uint16_t val,
                uint16_t range );
```

Parameters

x

X-coordinate of gauge center, in pixels

y

Y-coordinate of gauge center, in pixels

r

Radius of the gauge, in pixels

options

By default the gauge dial is drawn with a 3D effect and the value of options is zero. OPT_FLAT removes the 3D effect. With option OPT_NOBACK, the background is not drawn. With option OPT_NOTICKS, the tick marks are not drawn. With option OPT_NOPOINTER, the pointer is not drawn.

major

Number of major subdivisions on the dial, 1-10

minor

Number of minor subdivisions on the dial, 1-10

val

Gauge indicated value, between 0 and range, inclusive

range

Maximum value

Description

The details of physical dimension are:

- The tick marks are placed on a 270 degree arc, clockwise starting at south-west position
- Minor ticks are lines of width r*(2/256), major r*(6/256)
- Ticks are drawn at a distance of r*(190/256) to r*(200/256)
- The pointer is drawn with lines of width r*(4/256), to a point r*(190/256)from the center
- The other ends of the lines are each positioned 90 degrees perpendicular to the pointer direction, at a distance r*(3/256) from the center

Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_GAUGE(0xffffff13)
+4	X
+6	Y
+8	R
+10	Options
+12	Major
+14	Minor
+16	Value
+18	Range

Examples

A gauge with radius 50 pixels, five divisions of four ticks each, indicates 30%:

```
cmd_gauge(80, 60, 50, 0, 5, 4, 30, 100);
```

Without the 3D look:

```
cmd_gauge(80, 60, 50, OPT_FLAT, 5, 4, 30, 100);
```

Ten major divisions with two minor divisions each:

```
cmd_gauge(80, 60, 50, 0, 10, 2, 30, 100);
```

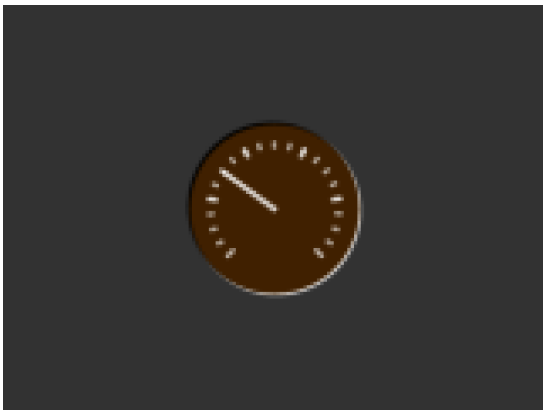
Setting the minor divisions to 1 makes them disappear:

```
cmd_gauge(80, 60, 50, 0, 10, 1, 30, 100);
```

Setting the major divisions to 1 gives minor division only:

```
cmd_gauge(80, 60, 50, 0, 1, 10, 30, 100);
```

A smaller gauge with a brown background:



```
cmd_bgcolor(0x402000);
cmd_gauge(80, 60, 25, 0, 5, 4, 30, 100);
```

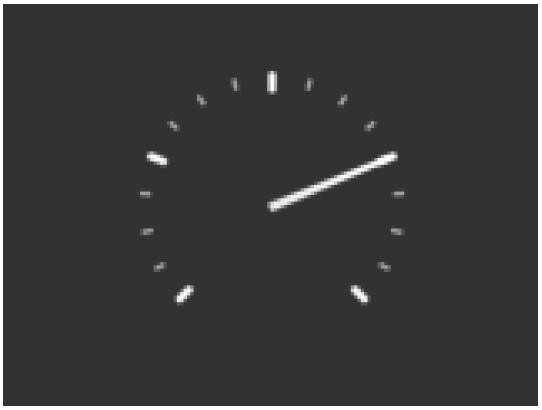
Scale 0-1000, indicating 1000:

```
cmd_gauge(80, 60, 50, 0, 5, 2, 1000, 1000);
```

Scaled 0-65535, indicating 49152:

```
cmd_gauge(80, 60, 50, 0, 4, 4, 49152, 65535);
```

No background:

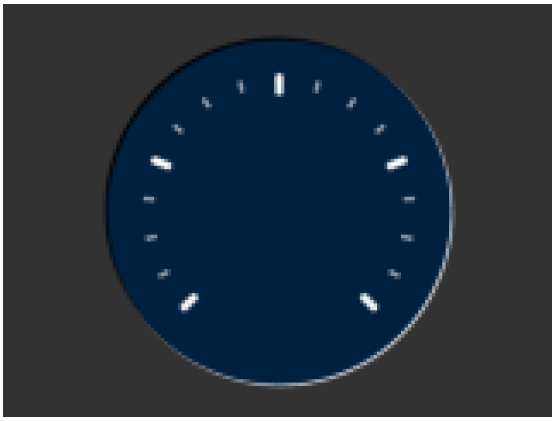


```
cmd_gauge(80, 60, 50, OPT_NOBACK, 4, 4, 49152, 65535);
```

No tick marks:

```
cmd_gauge(80, 60, 50, OPT_NOTICKS, 4, 4, 49152, 65535);
```

No pointer:

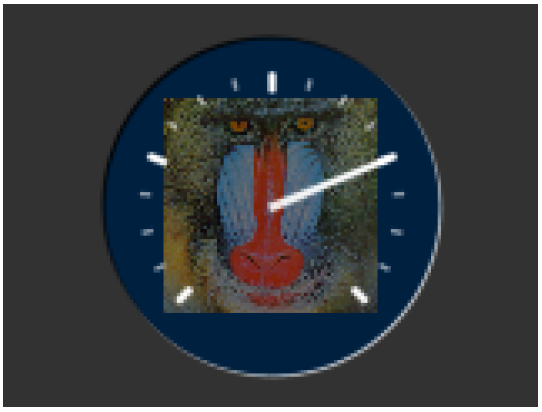


```
cmd_gauge(80, 60, 50, OPT_NOPOINTER, 4, 4, 49152, 65535);
```

Drawing the gauge in two passes, with bright red for the pointer:

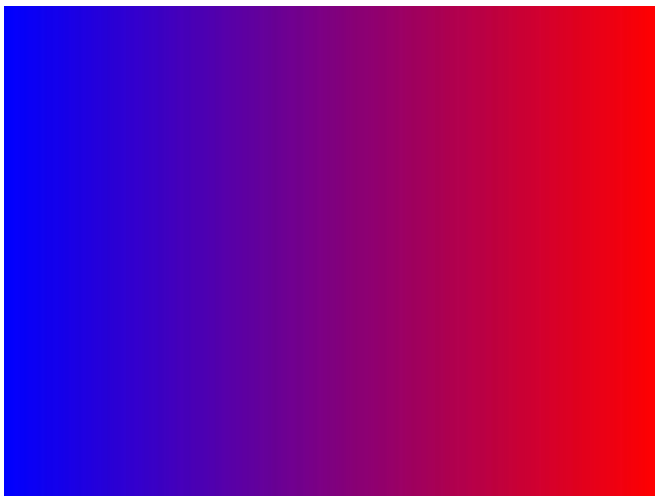
```
GAUGE_0 = OPT_NOPOINTER;
GAUGE_1 = OPT_NOBACK | OPT_NOTICKS;
cmd_gauge(80, 60, 50, GAUGE_0, 4, 4, 49152, 65535);
cmd(COLOR_RGB(255, 0, 0));
cmd_gauge(80, 60, 50, GAUGE_1, 4, 4, 49152, 65535);
```

Add a custom graphic to the gauge by drawing its background, a bitmap, and then its foreground:



```
GAUGE_0 = OPT_NOPOINTER | OPT_NOTICKS;
GAUGE_1 = OPT_NOBACK;
cmd_gauge(80, 60, 50, GAUGE_0, 4, 4, 49152, 65535);
cmd(COLOR_RGB(130, 130, 130));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(80 - 32, 60 -32, 0, 0));
cmd(COLOR_RGB(255, 255, 255));
cmd_gauge(80, 60, 50, GAUGE_1, 4, 4, 49152, 65535);
```

5.34 CMD_GRADIENT - draw a smooth color gradient



C prototype

```
void cmd_gradient( int16_t x0,
                   int16_t y0,
                   uint32_t rgb0,
                   int16_t x1,
                   int16_t y1,
                   uint32_t rgb1 );
```

Parameters

x0

x-coordinate of point 0, in pixels

y0

y-coordinate of point 0, in pixels

rgb0

Color of point 0, as a 24-bit RGB number. R is the most significant8 bits, B is the least. So 0xff0000 is bright red.

x1

x-coordinate of point 1, in pixels

y1

y-coordinate of point 1, in pixels

rgb1

Color of point 1

Description

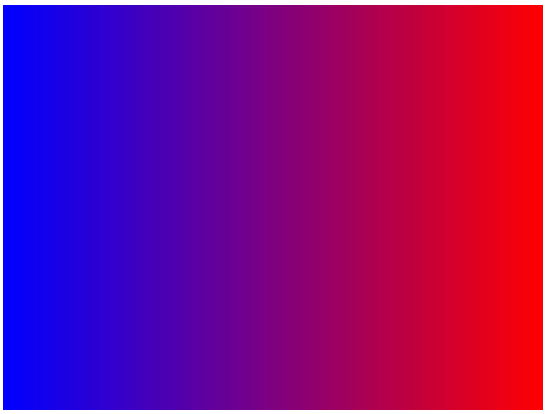
All the color's step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is R0 + t * (R1 - R0), where it is interpolated between 0 and 1. Gradient must be used with Scissor function to get the intended gradient display.

Command layout

+0	CMD_GRAGIENT(0xffffff0b)
+4	X0
+6	Yo
+8	RGB0
+12	X1
+14	Y1
+16	RGB1

Examples

A horizontal gradient from blue to red



```
cmd_gradient(0, 0, 0x0000ff, 160, 0, 0xff0000);
```

A vertical gradient

```
cmd_gradient(0, 0, 0x808080, 0, 120, 0x80ff40);
```

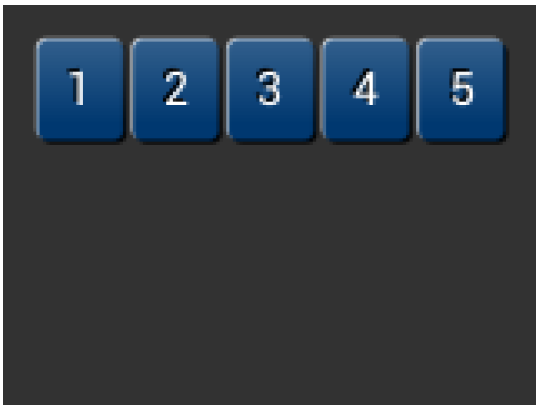
The same colors in a diagonal gradient

```
cmd_gradient(0, 0, 0x808080, 160, 120, 0x80ff40);
```

Using a scissor rectangle to draw a gradient stripe as a background for a title:


```
cmd(SCISSOR_XY(20, 40));
cmd(SCISSOR_SIZE(120, 32));
cmd_gradient(20, 0, 0x606060, 140, 0, 0x404080);
cmd_text(23, 40, 29, 0, "Heading 1");
```

5.35 CMD_KEYS - draw a row of keys



C prototype

```
void cmd_keys( int16_t x,
               int16_t y,
               int16_t w,
               int16_t h,
               int16_t font,
               uint16_t options,
               const char* s );
```

Parameters

x

x-coordinate of keys top-left, in pixels

y

y-coordinate of keys top-left, in pixels

font

Bitmap handle to specify the font used in key label. The valid range is from 0 to 31

options

By default the keys are drawn with a 3D effect and the value of option is zero. OPT_FLAT removes the 3D effect. If OPT_CENTER is given the keys are drawn at minimum size centered within the w x h rectangle. Otherwise the keys are expanded so that they completely fill the available space. If an ASCII code is specified, that key is drawn 'pressed' - i.e. in background color with any 3D effect removed.

w

The width of the keys

h

The height of the keys

s

key labels, one character per key. The TAG value is set to the ASCII value of each key, so that key presses can be detected using the REG_TOUCH_TAG register.

Description

The details of physical dimension are:

- The gap between keys is 3 pixels
- For OPT_CENTERX case, the keys are (font width + 1.5) pixels wide, otherwise keys are sized to fill available width

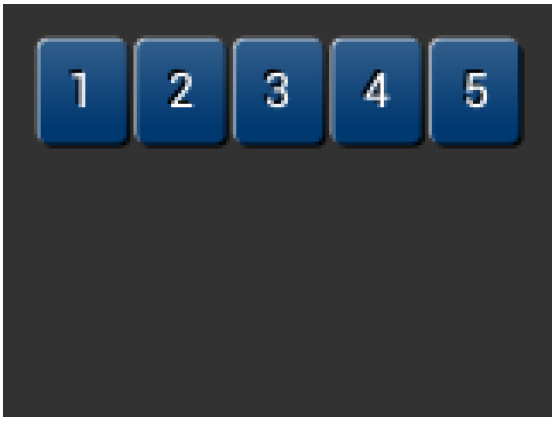
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_KEYS(0xffffff0e)
+4	X
+6	Y
+8	W
+10	H
+12	Font
+14	Options
+16	S
..	..
+n	0

Examples

A row of keys:

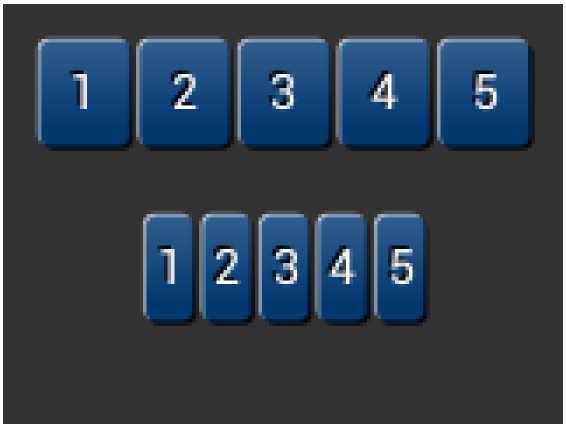



```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");
```

Without the 3D look:

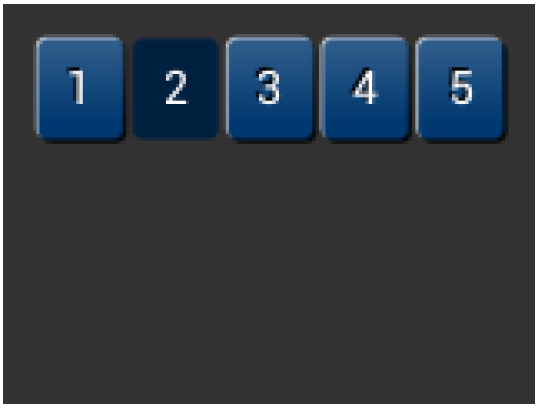

```
cmd_keys(10, 10, 140, 30, 26, OPT_FLAT, "12345");
```

Default vs. centered:




```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");
cmd_keys(10, 60, 140, 30, 26, OPT_CENTER, "12345");
```

Setting the options to show '2' key pressed ('2' is ASCII code 0x32):

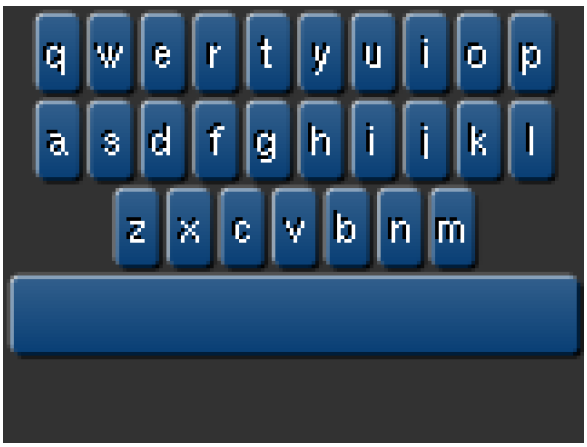



```
cmd_keys(10, 10, 140, 30, 26, 0x32, "12345");
```

A calculator-style keyboard using font 29:

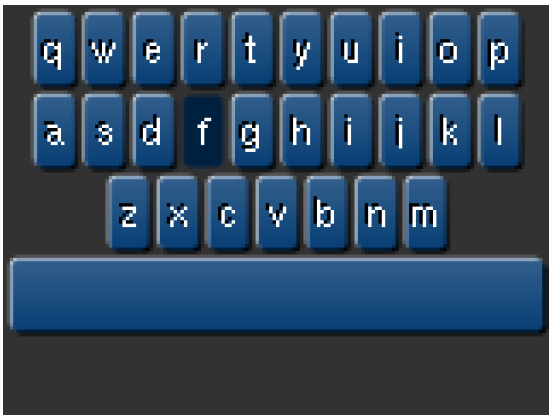

```
cmd_keys(22, 1, 116, 28, 29, 0, "789");
cmd_keys(22, 31, 116, 28, 29, 0, "456");
cmd_keys(22, 61, 116, 28, 29, 0, "123");
cmd_keys(22, 91, 116, 28, 29, 0, "0.");
```

A compact keyboard drawn in font 20:




```
cmd_keys(2, 2, 156, 21, 20, OPT_CENTER, "qwertyuiop");
cmd_keys(2, 26, 156, 21, 20, OPT_CENTER, "asdfghijkl");
cmd_keys(2, 50, 156, 21, 20, OPT_CENTER, "zxcvbnm");
cmd_button(2, 74, 156, 21, 20, 0, "");
```

Showing the f (ASCII 0x66) key pressed:




```
k = 0x66;
cmd_keys(2, 2, 156, 21, 20, k | OPT_CENTER, "qwertyuiop");
cmd_keys(2, 26, 156, 21, 20, k | OPT_CENTER, "asdfghijkl");
cmd_keys(2, 50, 156, 21, 20, k | OPT_CENTER, "zxcvbnm");
cmd_button(2, 74, 156, 21, 20, 0, "");
```

5.36 CMD_PROGRESS - draw a progress bar

C prototype

```
void cmd_progress( int16_t x,
                   int16_t y,
                   int16_t w,
                   int16_t h,
                   uint16_t options,
                   uint16_t val,
                   uint16_t range );
```

Parameters

x

x-coordinate of progress bar top-left, in pixels

y

y-coordinate of progress bar top-left, in pixels

w

width of progress bar, in pixels

h

height of progress bar, in pixels

options

By default the progress bar is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

val

Displayed value of progress bar, between 0 and range inclusive

range

Maximum value

Description

The details of physical dimensions are

- x,y,w,h give outer dimensions of progress bar. Radius of bar (r) is min(w,h)/2
- Radius of inner progress line is r*(7/8)

Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_PROGRESS(0xffffff0f)
+4	X
+6	Y
+8	W
+10	H
+12	options
+14	val
+16	range

Examples

A progress bar showing 50% completion:

```
cmd_progress(20, 50, 120, 12, 0, 50, 100);
```

Without the 3D look:

```
cmd_progress(20, 50, 120, 12, OPT_FLAT, 50, 100);
```

A 4 pixel high bar, range 0-65535, with a brown background:

```
cmd_bgcolor(0x402000);
cmd_progress(20, 50, 120, 4, 0, 9000, 65535);
```

5.37 CMD_SCROLLBAR – draw a scroll bar

C prototype

```
void cmd_scrollbar( int16_t x,
                    int16_t y,
                    int16_t w,
                    int16_t h,
                    uint16_t options,
                    uint16_t val,
                    uint16_t size,
                    uint16_t range );
```

Parameters

x

x-coordinate of scroll bar top-left, in pixels

y

y-coordinate of scroll bar top-left, in pixels

w

Width of scroll bar, in pixels. If width is greater than height, the scroll bar is drawn horizontally

h

Height of scroll bar, in pixels. If height is greater than width, the scroll bar is drawn vertically

options

By default the scroll bar is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

val

Displayed value of scroll bar, between 0 and range inclusive

range

Maximum value

Description

Refer to CMD_PROGRESS for more information on physical dimension.

Command layout

+0	CMD_SCROLLBAR(0xffffff11)
+4	X
+6	Y
+8	W
+10	H
+12	options
+14	val
+16	Size
+18	Range

Examples

A scroll bar indicating 10-50%:

cmd_scrollbar(20, 50, 120, 8, 0, 10, 40, 100);

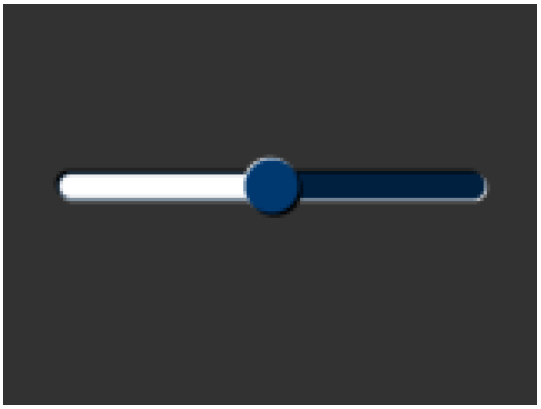
Without the 3D look:

cmd_scrollbar(20, 50, 120, 8, OPT_FLAT, 10, 40, 100);

A brown-themed vertical scroll bar:

```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_scrollbar(140, 10, 8, 100, 0, 10, 40, 100);
```

5.38 CMD_SLIDER – draw a slider



C prototype

```
void cmd_slider( int16_t x,
                 int16_t y,
                 int16_t w,
                 int16_t h,
                 uint16_t options,
                 uint16_t val,
                 uint16_t range );
```

Parameters

x

x-coordinate of slider top-left, in pixels

y

y-coordinate of slider top-left, in pixels

w

width of slider, in pixels. If width is greater than height, the scroll bar is drawn horizontally

h

height of slider, in pixels. If height is greater than width, the scroll bar is drawn vertically

options

By default the slider is drawn with a 3D effect. OPT_FLAT removes the 3D effect

val

Displayed value of slider, between 0 and range inclusive

range

Maximum value

Description

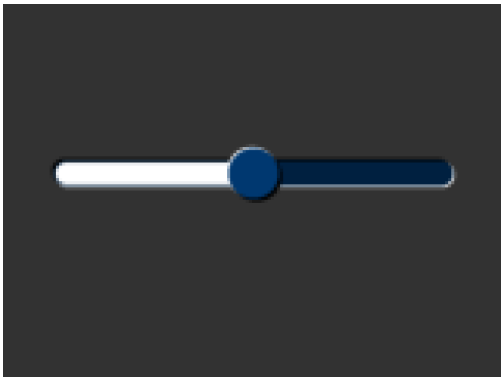
Refer to CMD_PROGRESS for more information on physical Dimension.

Command layout

+0	CMD_SLIDER(0xffffff10)
+4	X
+6	Y
+8	W
+10	H
+12	options
+14	val
+16	Range

Examples

A slider set to 50%:

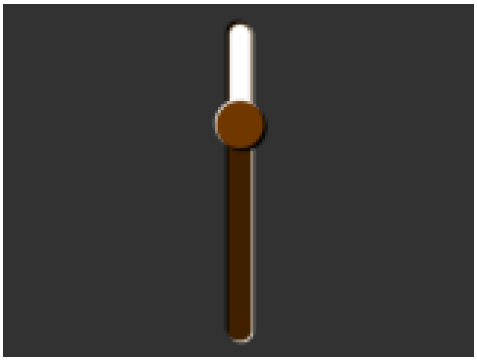


cmd_slider(20, 50, 120, 8, 0, 50, 100);

Without the 3D look:

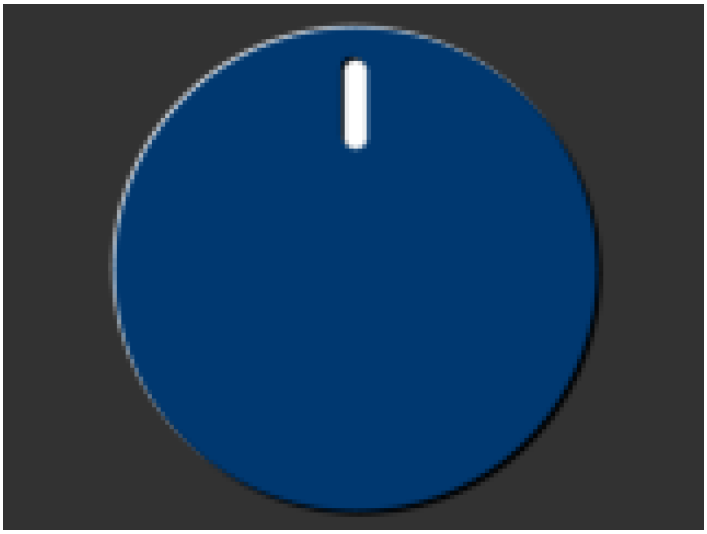
```
cmd_slider(20, 50, 120, 8, OPT_FLAT, 50, 100);
```

A brown-themed vertical slider with range 0-65535:



```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_slider(76, 10, 8, 100, 0, 20000, 65535);
```

5.39 CMD_DIAL – draw a rotary dial control



C prototype

```
void cmd_dial( int16_t x,
```

int16_t y,

int16_t r,

uint16_t options,

uint16_t val);

Parameters

x

x-coordinate of dial center, in pixels

y

y-coordinate of dial center, in pixels

r

radius of dial, in pixels.

Options

By default the dial is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

val

Specify the position of dial points by setting value between 0 and 65535 inclusive. 0 means that the dial points straight down, 0x4000 left, 0x8000 up, and0xc000 right.

Description

The details of physical dimension are

- The marker is a line of width r*(12/256), drawn at a distance r*(140/256)to r*(210/256) from the center

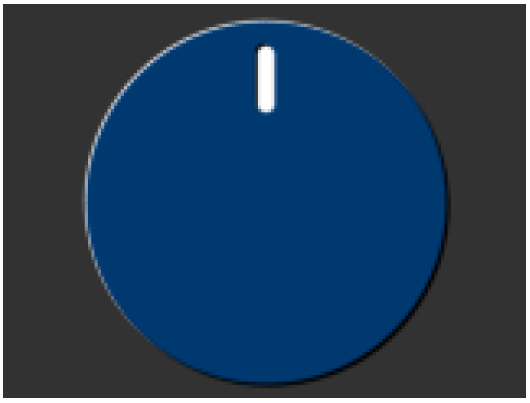
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_DIAL(0xffffff2d)
+4	X
+6	Y
+8	r
+10	options
+12	val

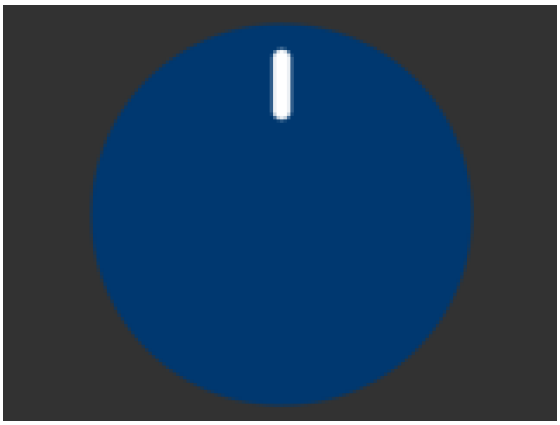
Examples

A dial set to 50%:



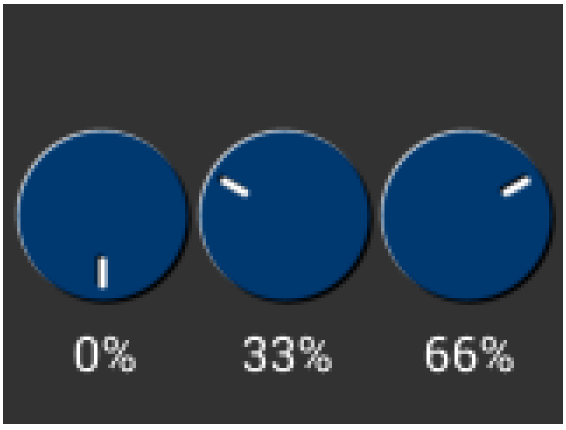
```
cmd_dial(80, 60, 55, 0, 0x8000);
```

Without the 3D look:



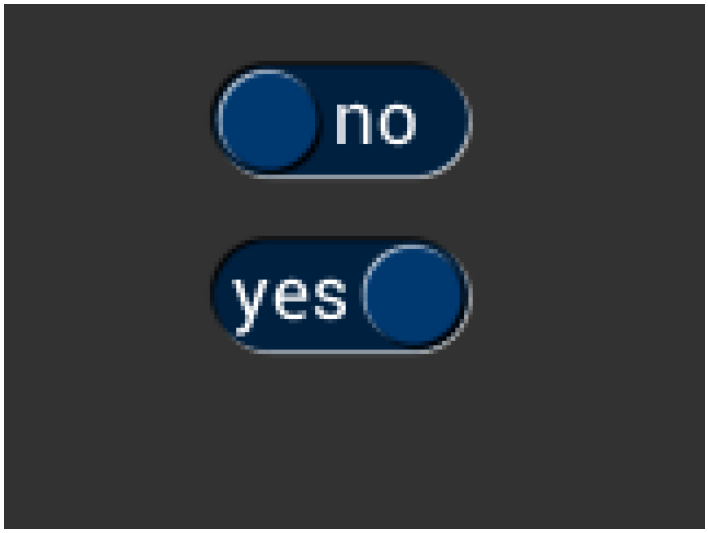
```
cmd_dial(80, 60, 55, OPT_FLAT, 0x8000);
```

Dials set to 0%, 33% and 66%:




```
cmd_dial(28, 60, 24, 0, 0x0000);
cmd_text(28, 100, 26, OPT_CENTER, "0%");
cmd_dial(80, 60, 24, 0, 0x5555);
cmd_text(80, 100, 26, OPT_CENTER, "33%");
cmd_dial(132, 60, 24, 0, 0xaaaa);
cmd_text(132, 100, 26, OPT_CENTER, "66%");
```

5.40 CMD_TOGGLE – draw a toggle switch



C prototype

```
void cmd_toggle( int16_t x,
                 int16_t y,
                 int16_t w,
                 int16_t font,
                 uint16_t options,
                 uint16_t state,
                 const char* s );
```

Parameters

x

x-coordinate of top-left of toggle, in pixels

y

y-coordinate of top-left of toggle, in pixels

w

width of toggle, in pixels

font

font to use for text, 0-31. See ROM and RAM Fonts

options

By default the toggle is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

state

state of the toggle: 0 is off, 65535 is on.

S

String label for toggle. A character value of 255 (in C it can be written as \xff) separates the two labels.

Description

The details of physical dimension are:

- Widget height (h) is font height * 20/16 pixel
- Outer bar radius (r) is font height * (10/16)
- Knob radius is r-1.5 pixel, where r is the outer bar radius above.
- The center of outer bar 's left round head is at (x, y + r/2) coordinate.

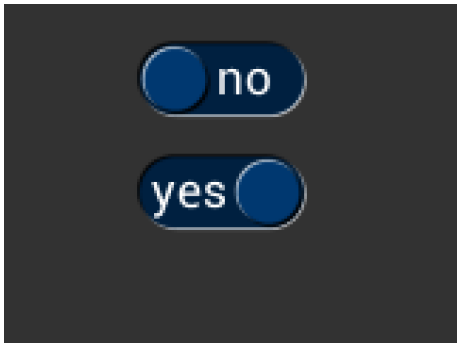
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_TOGGLE(0xffffff12)
+4	X
+6	Y
+8	W
+10	Font
+12	Options
+14	State
+16	S
..	..
..	0

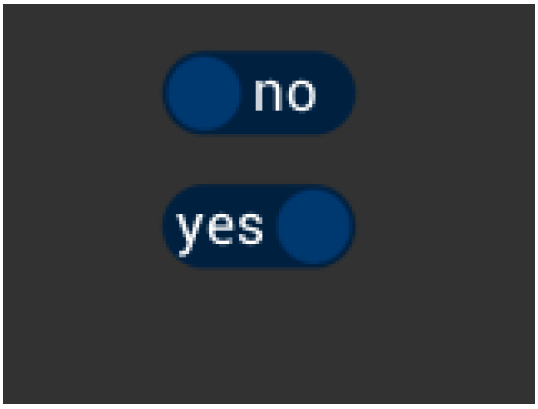
Examples

Using a medium font, in the two states



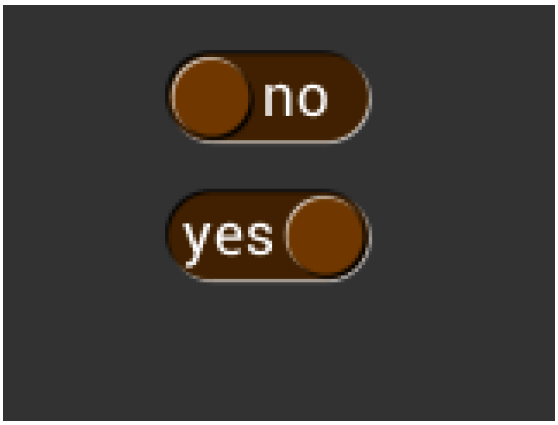

```
cmd_toggle(60, 20, 33, 27, 0, 0, "no" " \xff" "yes");
cmd_toggle(60, 60, 33, 27, 0, 65535, "no" "\xff" "yes");
```

Without the 3D look




```
cmd_toggle(60, 20, 33, 27, OPT_FLAT, 0, "no" "\xff" "yes");
cmd_toggle(60, 60, 33, 27, OPT_FLAT, 65535, "no" "\xff" "yes");
```

With different background and foreground colors:




```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_toggle(60, 20, 33, 27, 0, 0, "no" "\xff" "yes");
cmd_toggle(60, 60, 33, 27, 0, 65535, "no" "\xff" "yes");
```

5.41 CMD_TEXT - draw text

C prototype

```
void cmd_text(  int16_t x,
                int16_t y,
                int16_t font,
                uint16_t options,
                const char* s );
```

Parameters

x

x-coordinate of text base, in pixels

y

y-coordinate of text base, in pixels

font

Font to use for text, 0-31. See ROM and RAM Fonts

options

By default (x,y) is the top-left pixel of the text and the value of options is zero.

OPT_CENTERX centers the text horizontally, OPT_CENTERY centers it vertically.

OPT_CENTER centers the text in both directions.

OPT_RIGHTX right-justifies the text, so that the x is the rightmost pixel.

Text string

The text string itself which should be terminated by a null character (ASCII code 0x0)

Command layout

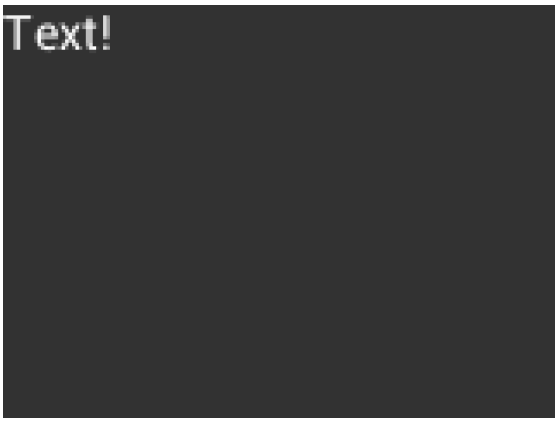
+0	CMD_TEXT(0xffffff0c)
+4	X
+6	Y
+8	Font
+10	Options
+12	S
..	..
..	0 (null character to terminate string)

Examples

Plain text at (0,0) in the largest font:

cmd_text(0, 0, 31, 0, “Text!”);

Using a smaller font:



cmd_text(0, 0, 26, 0, “Text!”);

Centered horizontally:

cmd_text(80, 60, 31, OPT_CENTERX, "Text!");

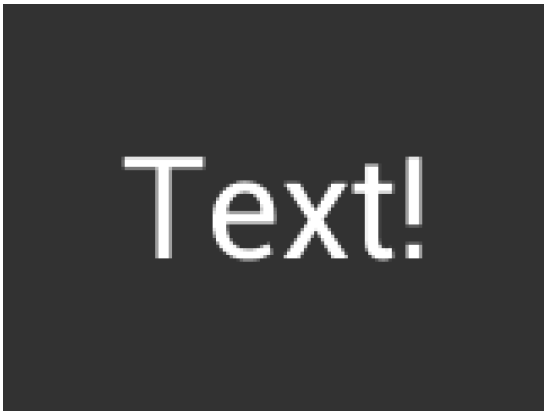
Right-justified:

cmd_text(80, 60, 31, OPT_RIGHTX, "Text!");

Centered vertically:

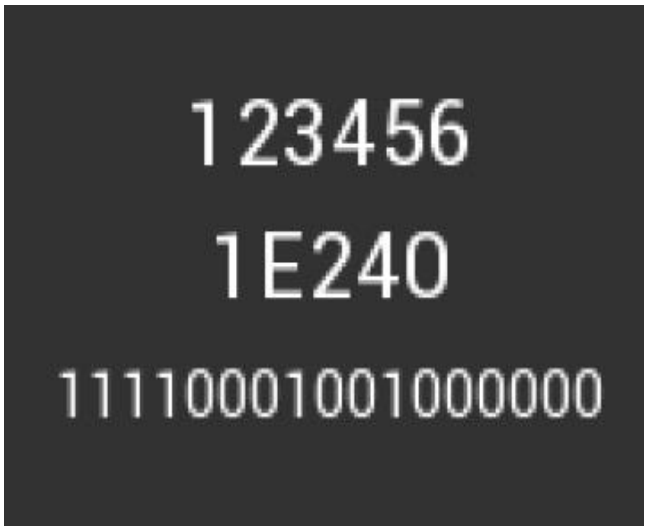
cmd_text(80, 60, 31, OPT_CENTERY, "Text!");

Centered both horizontally and vertically:



cmd_text(80, 60, 31, OPT_CENTER, “Text!”);

5.42 CMD_SETBASE – Set the base for number output



C prototype

void cmd_setbase(uint32_t b);

Parameters

b

Numeric base, valid values are from 2 to 36, examples are : 2 for binary, 8 for octal, 10 for decimal, 16 for hexadecimal.

Description

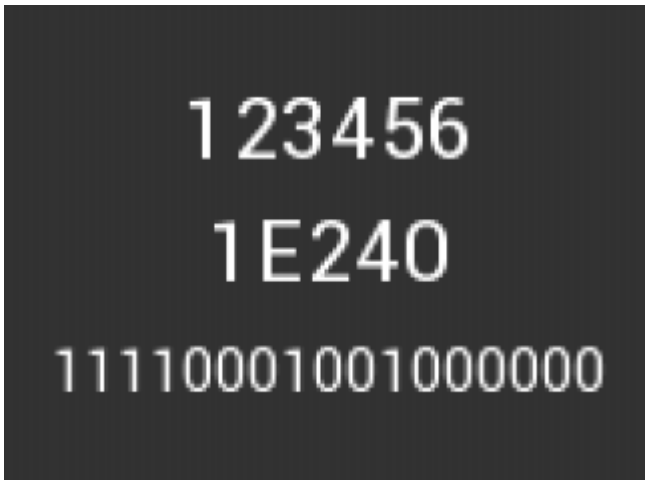
Set up numeric base for CMD_NUMBER

Command layout

+0	CMD_SETBASE(0xffffff38)
+4	b

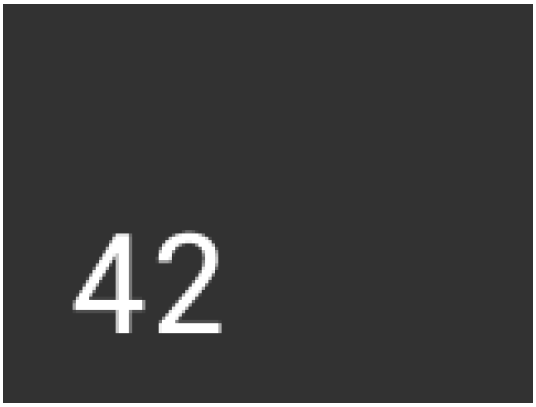
Examples

The number 123456 displayed in decimal, hexadecimal and binary:




```
cmd_number(80, 30, 28, OPT_CENTER, 123456);
cmd_setbase(16);
cmd_number(80, 60, 28, OPT_CENTER, 123456);
cmd_setbase(2);
cmd_number(80, 90, 26, OPT_CENTER, 123456);
```

5.43 CMD_NUMBER - draw number



C prototype

```
void cmd_number( int16_t x,
                 int16_t y,
                 int16_t font,
                 uint16_t options,
                 int32_t n );
```

Parameters

x

x-coordinate of text base, in pixels

y

y-coordinate of text base, in pixels

font

font to use for text, 0-31. See ROM and RAM Fonts

options

By default (x,y) is the top-left pixel of the text. OPT_CENTERX centers the text horizontally, OPT_CENTERY centers it vertically. OPT_CENTER centers the text in both directions. OPT_RIGHTX right-justifies the text, so that the x is the rightmost pixel. By default the number is displayed with no leading zeroes, but if a width 1-9 is specified in the options, then the number is padded if necessary with leading zeroes so that it has the given width. If OPT_SIGNED is given, the number is treated as signed, and prefixed by a minus sign if negative.

n

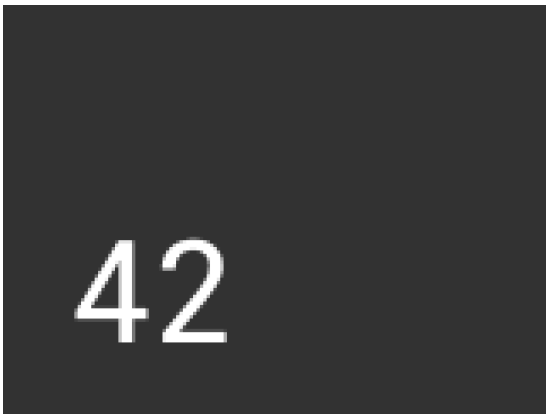
The number to display, is either unsigned or signed 32-bit, in the base specified in the preceding CMD_SETBASE. If no CMD_SETBASE appears before CMD_NUMBER, it will be in decimal base.

Command layout

+0	CMD_NUMBER(0xffffff2e)
+4	X
+6	Y
+8	Font
+10	Options
+12	n

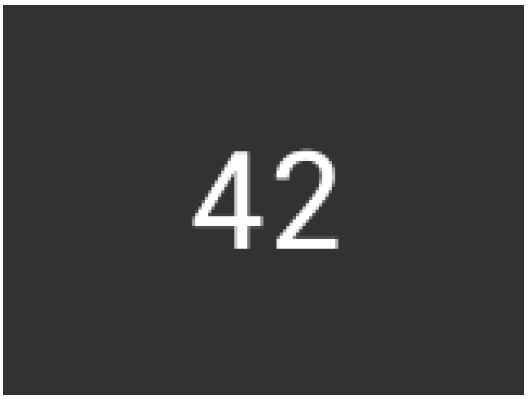
Examples

A number:




```
cmd_number(20, 60, 31, 0, 42);
```

Centered:

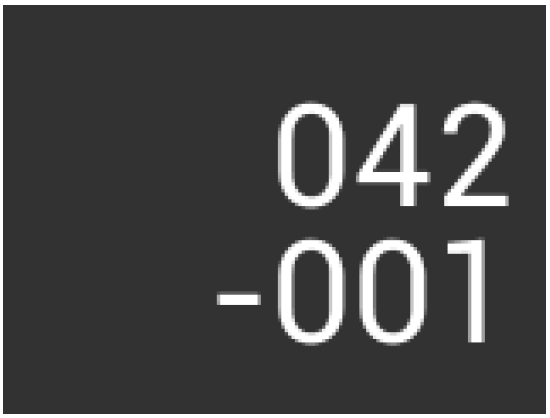



```
cmd_number(80, 60, 31, OPT_CENTER, 42);
```

Signed output of positive and negative numbers:


```
cmd_number(20, 20, 31, OPT_SIGNED, 42);
cmd_number(20, 60, 31, OPT_SIGNED, -42);
```

Forcing width to 3 digits, right-justified



cmd_number(150, 20, 31, OPT_RIGHTX | 3, 42);

cmd_number(150, 60, 31, OPT_SIGNED | OPT_RIGHTX | 3, -1);

5.44 CMD_LOADIDENTIY - Set the current matrix to the identity matrix

This command instructs the co-processor engine of the FT81X to set the current matrix to the identity matrix, so that the co-processor engine is able to form the new matrix as requested by CMD_SCALE, CMD_ROTATE,CMD_TRANSLATE command.

For more information on the identity matrix, refer to the Bitmap Transformation Matrix section.

C prototype

void cmd_loadidentity( );

Command layout

+0	CMD_LOADIDENTITY(0xffffff26)

5.45 CMD_SETMATRIX - write the current matrix to the display list

The co-processor engine assigns the value of the current matrix to the bitmap transform matrix of the graphics engine by generating display list commands, i.e., BITMAP_TRANSFORM_A-F. After this command, the following bitmap rendering operation will be affected by the new transform matrix.

C prototype

void cmd_setmatrix( );

Command layout

+0	CMD_SETMATRIX(0xffffff2a)

Parameter

None

5.46 CMD_GETMATRIX - retrieves the current matrix coefficients

Retrieves the current matrix within the context of the co-processor engine. Note the matrix within the context of the co-processor engine will not apply to the bitmap transformation until it is passed to the graphics engine through CMD_SETMATRIX.

C prototype

```
void cmd_getmatrix( int32_t a,
                    int32_t b,
                    int32_t c,
                    int32_t d,
                    int32_t e,
                    int32_t f );
```

Parameters

a

output parameter; written with matrix coefficient a. See the parameter of the command BITMAP_TRANSFORM_A for formatting.

b

output parameter; written with matrix coefficient b. See the parameter b of the command BITMAP_TRANSFORM_B for formatting.

c

output parameter; written with matrix coefficient c. See the parameter c of the command BITMAP_TRANSFORM_C for formatting.

d

output parameter; written with matrix coefficient d. See the parameter d of the command BITMAP_TRANSFORM_D for formatting.

e

output parameter; written with matrix coefficient e. See the parameter e of the command BITMAP_TRANSFORM_E for formatting.

f

output parameter; written with matrix coefficient f. See the parameter f of the command BITMAP_TRANSFORM_F for formatting.

Command layout

+0	CMD_GETMATRIX(0xffffff33)
+4	A
+8	B
+12	C
+16	D

+20	E
+24	F

5.47 CMD_GETPTR - get the end memory address of data inflated by CMD_INFLATE

C prototype

void cmd_getptr(uint32_t result

);

Parameters

result

The end address of decompressed data done by **CMD_INFLATE**.

The starting address of decompressed data was specified by **CMD_INFLATE**, while the end address of decompressed data can be retrieved by this command.

It is one out parameter and requires a single dummy parameter input of any value to run. This input value is then replaced with the actual result by the FT81x device in the same FIFO location. After execution of this command the user must read the FIFO location to obtain the result.

Command layout

+0	CMD_GETPTR (0xffffff23)
+4	result

Examples

```
cmd_inflate(1000); //Decompress the data into RAM_G + 1000
......             //Following the zlib compressed data
While(rd16(REG_CMD_WRITE)   != rd16(REG_CMD_READ)); //Wait till the compression was
done

uint16_t x = rd16(REG_CMD_WRITE);
uint32_t ending_address = 0;
cmd_getptr(0);
ending_address = rd32(RAM_CMD +  x + 4);
```

Code snippet 11 CMD_GETPTR command example

5.48 CMD_GETPROPS - get the image properties decompressed by CMD_LOADIMAGE

C prototype

void cmd_getprops(uint32_t ptr, uint32_t width, uint32_t height);

Parameters

ptr

The address of the image in RAM_G which was decompressed by the last **CMD_LOADIMAGE** before this command.

It is an output parameter.

width

The width of the image which was decompressed by the last **CMD_LOADIMAGE** before this command.

It is an output parameter.

height

The height of the image which was decompressed by the last **CMD_LOADIMAGE** before this command.

It is an output parameter.

Command layout

+0	CMD_GETPROPS (0xffffff25)
+4	ptr
+8	width
+12	height

Description

This command is used to retrieve the properties of the image which is decompressed by **CMD_LOADIMAGE**. Respective image properties are updated by the coprocessor after this command is executed successfully.

Examples

Please refer to the **CMD_GETPTR**

5.49 CMD_SCALE - apply a scale to the current matrix

C prototype

void cmd_scale(int32_t sx,
int32_t sy);

Parameters

sx

x scale factor, in signed 16. 16 bit fixed-point form.

sy

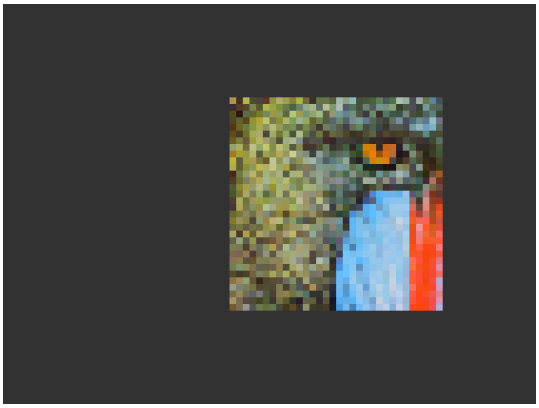
y scale factor, in signed 16. 16 bit fixed-point form.

Command layout

+0	CMD_SCALE(0xffffff28)
+4	sx
+8	sy

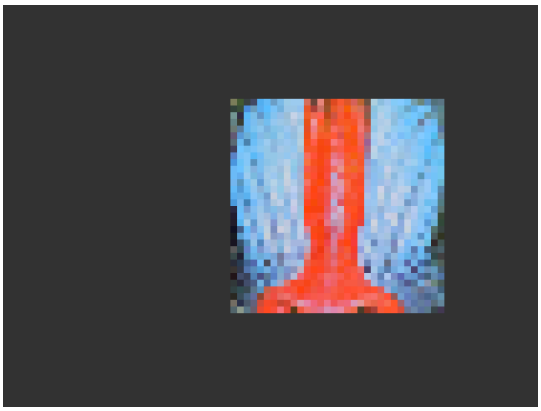
Examples

To zoom a bitmap 2X:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_scale(2 * 65536, 2 * 65536);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To zoom a bitmap 2X around its center:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(65536 * 32, 65536 * 32);
cmd_scale(2 * 65536, 2 * 65536);
cmd_translate(65536 * -32, 65536 * -32);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

5.50 CMD_ROTATE - apply a rotation to the current matrix

C prototype

void cmd_rotate(int32_t a);

Parameters

a

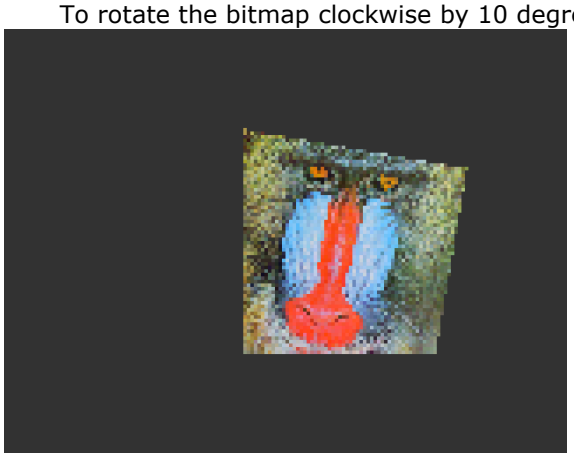
Clockwise rotation angle, in units of 1/65536 of a circle

Command layout

+0	CMD_ROTATE(0xffffff29)
+4	a

Examples

To rotate the bitmap clockwise by 10 degrees with respect to the top left of the bitmap:

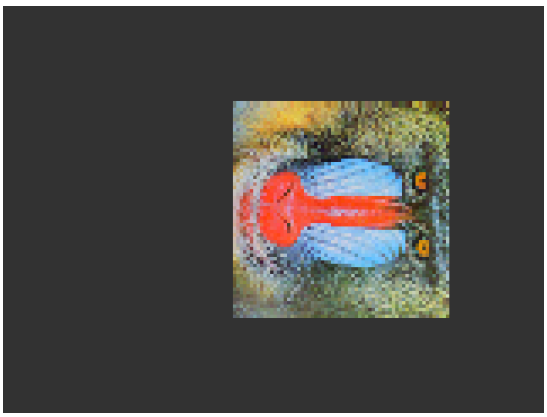


```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotate(10 * 65536 / 360);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To rotate the bitmap counter clockwise by 33 degrees wrt top left of the bitmap:

```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotate(-33 * 65536 / 360);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

Rotating a 64 x 64 bitmap around its center:



```
cmd(BEGIN(BITMAPS));
 cmd_loadidentity();
 cmd_translate(65536 * 32, 65536 * 32);
 cmd_rotate(90 * 65536 / 360);
 cmd_translate(65536 * -32, 65536 * -32);
 cmd_setmatrix();
 cmd(VERTEX2II(68, 28, 0, 0));
```

5.51 CMD_TRANSLATE - apply a translation to the current matrix

C prototype

```
void cmd_translate( int32_t tx,
                    int32_t ty );
```

Parameters

tx

x translate factor, in signed 16.16 bit fixed-point form.

ty

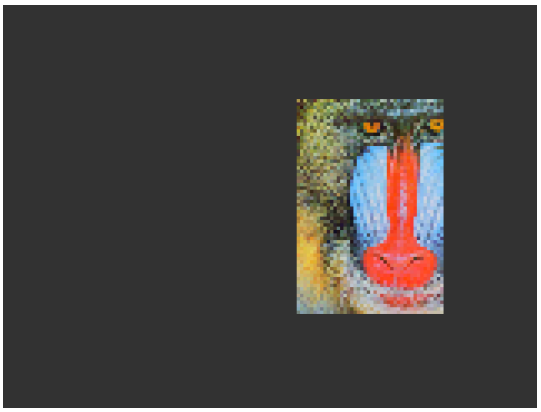
y translate factor, in signed 16.16 bit fixed-point form.

Command layout

+0	CMD_TRANSLATE(0xffffff27)
+4	Tx
+8	Ty

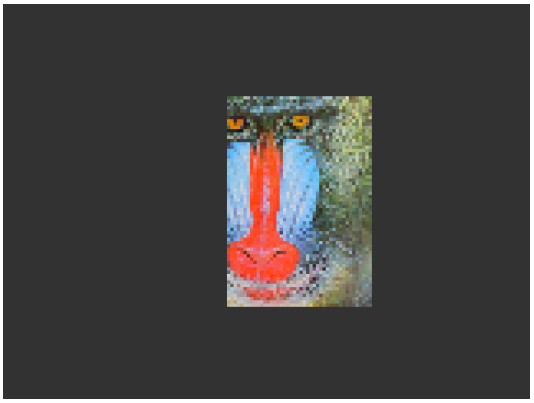
Examples

To translate the bitmap 20 pixels to the right:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(20 * 65536, 0);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To translate the bitmap 20 pixels to the left:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(-20 * 65536, 0);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

5.52 CMD_CALIBRATE - execute the touch screen calibration routine

The calibration procedure collects three touches from the touch screen, then computes and loads an appropriate matrix into **REG_TOUCH_TRANSFORM_A-F**. To use the function, create a display list and include **CMD_CALIBRATE**. The co-processor engine overlays the touch targets on the current display list, gathers the calibration input and updates **REG_TOUCH_TRANSFORM_A-F**.

Please note that this command only applies to RTE and compatibility mode of CTE.

C prototype

void cmd_calibrate(uint32_t result);

Parameters

result

output parameter; written with 0 on failure of calibration.

Description

The completion of this function is detected when the value of **REG_CMD_READ** is equal to **REG_CMD_WRITE**.

Command layout

+0	CMD_CALIBRATE(0xffffff15)
+4	Result

Examples

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80, 30, 27, OPT_CENTER, "Please tap on the dot");
cmd_calibrate();
```

Code snippet 12 CMD_CALIBRATE example

5.53 CMD_SETROTATE – Rotate the screen

C prototype

void cmd_setrotate(uint32_t r);

Parameters

r

The value from 0 to 7. The same definition as the value in REG_ROTATE. Refer to the section Rotation to understand more.

Description

CMD_SETROTATE sets REG_ROTATE to the given value, causing the screen to rotate. It also appropriately adjusts the touch transform matrix so that coordinates of touch points are adjusted to rotated coordinate system.

Command layout

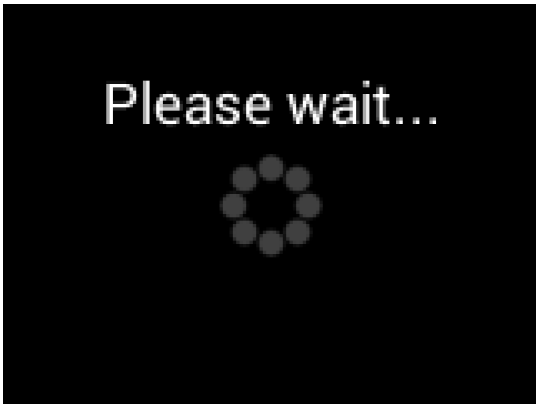
+0	CMD_SETROTATE (0xffffff36)
+4	R

Examples

```
cmd_setrotate(2); //Put the display in portrait mode
```

Code snippet 13 CMD_SETROTATE example

5.54 CMD_SPINNER - start an animated spinner



The spinner is an animated overlay that shows the user that some task is continuing. To trigger the spinner, create a display list and then use CMD_SPINNER. The co-processor engine overlays the spinner on the current display list, swaps the display list to make it visible, then continuously animates until it receives CMD_STOP. REG_MACRO_0 and REG_MACRO_1 registers are utilized to perform the animation kind of effect. The frequency of point's movement is with respect to the display frame rate configured.

Typically for 480x272 display panels the display rate is ~60fps. For style 0 and 60fps, the point repeats the sequence within 2 seconds. For style 1 and 60fps, the point repeats the sequence within 1.25 seconds. For style 2 and 60fps, the clock hand repeats the sequence within 2 seconds. For style 3 and 60fps, the moving dots repeat the sequence within 1 second.

Note that only one of **CMD_SKETCH, CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_spinner( int16_t x,
                  int16_t y,
                  uint16_t style,
                  uint16_t scale );
```

Command layout

+0	CMD_SPINNER(0xffffff16)
+4	X
+6	Y
+8	Style
+10	Scale

Parameters

X

The X coordinate of top left of spinner

Y

The Y coordinate of top left of spinner

Style

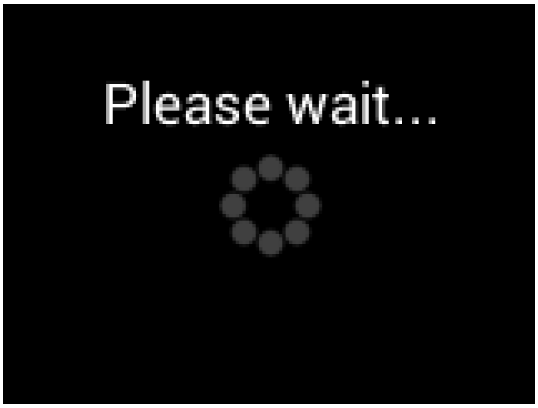
The style of spinner. Valid range is from 0 to 3.

Scale

The scaling coefficient of spinner. 0 means no scaling.

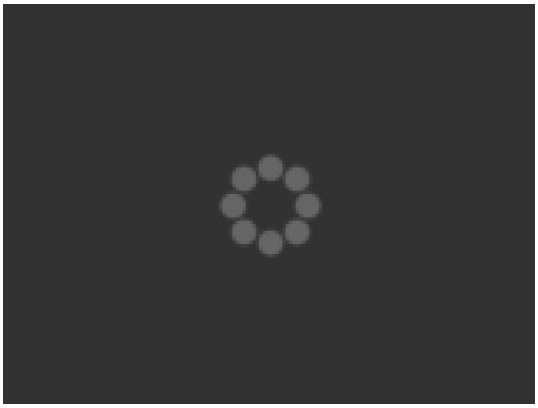
Examples

Create a display list, then start the spinner:



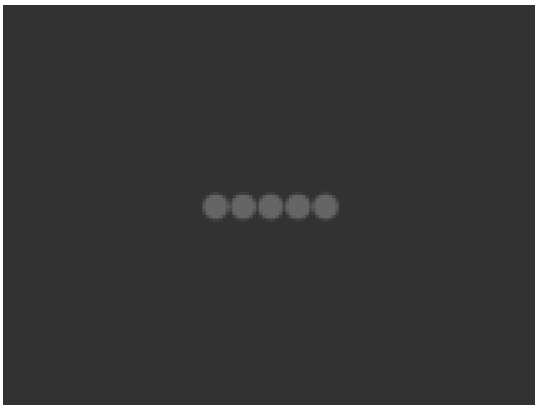

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80, 30, 27, OPT_CENTER, "Please wait...");
cmd_spinner(80, 60, 0, 0);
```

Spinner style 0, a circle of dots:



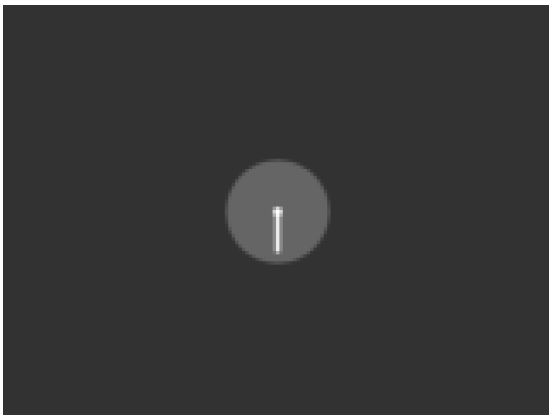
```
cmd_spinner(80, 60, 0, 0);
```

Style 1, a line of dots:



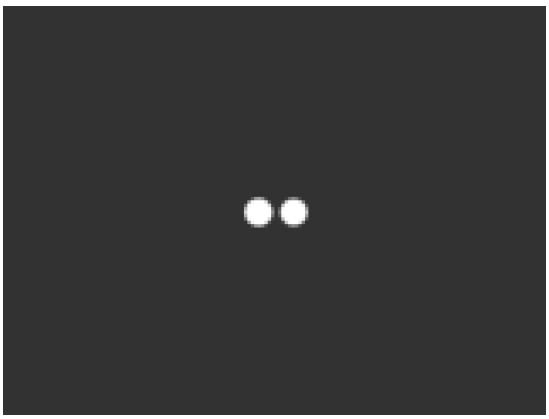
cmd_spinner(80, 60, 1, 0);

Style 2, a rotating clock hand:



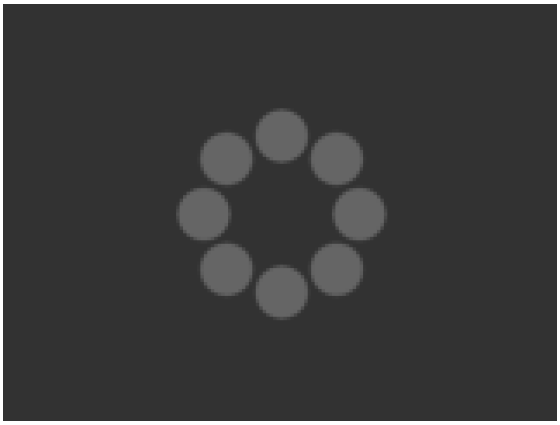
cmd_spinner(80, 60, 2, 0);

Style 3, two orbiting dots:



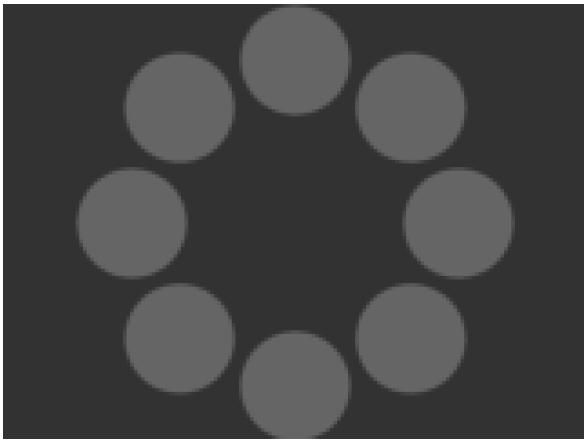
cmd_spinner(80, 60, 3, 0);

Half screen, scale 1:



cmd_spinner(80, 60, 0, 1);

Full screen, scale 2:



cmd_spinner(80, 60, 0, 2);

5.55 CMD_SCREENSAVER - start an animated screensaver

After the screensaver command, the co-processor engine continuously updates **REG_MACRO_0** with **VERTEX2F** with varying (x,y) coordinates. With an appropriate display list, this causes a bitmap to move around the screen without any MCU work. Command **CMD_STOP** stops the update process.

Note that only one of **CMD_SKETCH, CMD_SCREENSAVER, or CMD_SPINNER** can be active at one time.

C prototype

void cmd_screensaver( );

Description

REG_MACRO_0 is updated with respect to frame rate (depending on the display registers configuration). Typically for a 480x272 display the frame rate is around 60 frames per second.

Command layout

+0	CMD_SCREENSAVER(0xffffff2f)

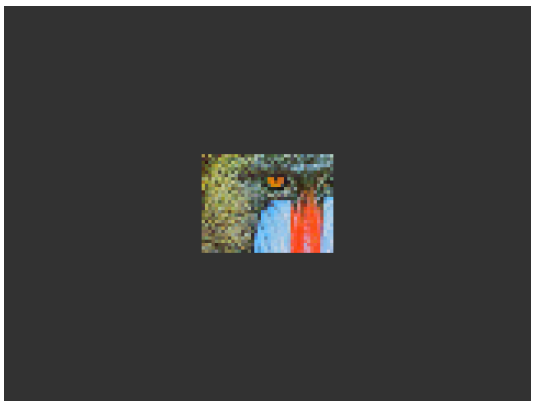
Examples

To start the screensaver, create a display list using a MACRO instruction – the co-processor engine will update it continuously:

```
cmd_screensaver();
cmd(BITMAP_SOURCE(0));
cmd(BITMAP_LAYOUT(RGB565, 128, 64));
cmd(BITMAP_SIZE(NEAREST,BORDER,BORDER, 40, 30));
cmd(BEGIN(BITMAPS));
cmd(MACRO(0));
cmd(DISPLAY());
```

Code snippet 14 CMD_SCREENSAVER example

Here is the result:



5.56 CMD_SKETCH - start a continuous sketch update

The Co-processor engine continuously samples the touch inputs and paints pixels into a bitmap, according to the given touch (x, y). This means that the user touch inputs are drawn into the bitmap without any need for MCU work. **CMD_STOP** is to be sent to stop the sketch process.

Note that only one of **CMD_SKETCH, CMD_SCREENSAVER,** or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_sketch( int16_t x,
                 int16_t y,
                 uint16_t w,
                 uint16_t h,
                 uint32_t ptr,
                 uint16_t format );
```

Parameters

x

x-coordinate of sketch area top-left, in pixels

y

y-coordinate of sketch area top-left, in pixels

w

Width of sketch area, in pixels

h

Height of sketch area, in pixels

ptr

Base address of sketch bitmap

format

Format of sketch bitmap, either L1 or L8

Description

Note that update frequency of bitmap data in graphics memory depends on the sampling frequency of the built-in ADC circuit of the FT81X, which is up to 1000 Hz.

Command layout

+0	CMD_SKETCH(0xffffff30)
+4	X
+6	Y
+8	W
+10	H
+12	Ptr
+16	Format

Examples

To start sketching into a 480x272 L1 bitmap:

```
cmd_memzero(0, 480 * 272 / 8);
cmd_sketch(0, 0, 480, 272, 0, L1);

//Then to display the bitmap
cmd(BITMAP_SOURCE(0));
cmd(BITMAP_LAYOUT(L1, 60, 272));
cmd(BITMAP_SIZE(NEAREST, BORDER, BORDER, 480, 272));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(0, 0, 0, 0));

//Finally, to stop sketch updates
cmd_stop();
```

Code snippet 15 CMD_SKETCH example

5.57 CMD_STOP - stop any of spinner, screensaver or sketch

This command is to inform the co-processor engine to stop the periodic operation, which is triggered by **CMD_SKETCH , CMD_SPINNER** or **CMD_SCREENSAVER**.

C prototype

void cmd_stop( );

Command layout

+0	CMD_STOP(0xffffff17)

Parameters

None

Description

For **CMD_SPINNER** and **CMD_SCREENSAVER**, **REG_MACRO_0** and **REG_MACRO_1** updating will be stopped.

For **CMD_SKETCH** or **CMD_CSKETCH**, the bitmap data in **RAM_G** updating will be stopped.

Examples

See **CMD_SKETCH,CMD_CSKETCH, CMD_SPINNER, CMD_SCREENSAVER**

5.58 CMD_SETFONT - set up a custom font

CMD_SETFONT is used to register one custom defined bitmap font into the co-processor engine. After registration, the co-processor engine is able to use the bitmap font with corresponding commands.

For further details about how to set up a custom font, refer to ROM and RAM Fonts.

C prototype

void cmd_setfont(uint32_t font,
uint32_t ptr);

Command layout

+0	CMD_SETFONT(0xffffff2b)
+4	Font
+8	Ptr

Parameters

font

The bitmap handle from 0 to 31

ptr

The metrics block address in RAM. 4 bytes aligned is required.

Examples

With a suitable font metrics block loaded in RAM at address 1000, to set it up for use with objects as font 7:

```
cmd_setfont(7, 1000);
cmd_button(20, 20,                          // x,y
           120, 40,                             // width,height in pixels
           7,                               // font 7, just loaded
           0,                                   // default options,3D style
           "custom font!");
```

Code snippet 16 CMD_SETFONT example

5.59 CMD_SETFONT2 - set up a custom font

To use a custom font with the co-processor objects, create the font definition data in RAM_G and issue CMD_SETFONT2, as described in ROM and RAM Fonts.

For details about how to set up a custom font, refer to ROM and RAM Fonts.

C prototype

void cmd_setfont2(uint32_t font,
uint32_t ptr,
uint32_t firstchar);

Command layout

+0	CMD_SETFONT2(0xffffff3b)
+4	Font
+8	Ptr
+12	firstchar

Parameters

font

The bitmap handle from 0 to 31

ptr

32 bit aligned memory address in RAM_G of font metrics block

firstchar

The ASCII value of first character in the font.

Examples

With a suitable font metrics block loaded in RAM_G at address 100000, first character's ASCII value 32, to use it for font 20:


```
cmd_setfont2(20, 100000, 32);
cmd_button(15, 30, 130, 20, 18, 0, "This is font 18");
cmd_button(15, 60, 130, 20, 20, 0, "This is font 20");
```

Code snippet 17 CMD_SETFONT2 example

5.60 CMD_SETSCRATCH - set the scratch bitmap for widget use

Graphical objects use a bitmap handle for rendering. By default this is bitmap handle 15. This command allows it to be set to any bitmap handle.

This command enables user to utilize bitmap handle 15 safely.

C prototype

void cmd_setscratch(uint32_t handle);

Parameters

handle

bitmap handle number , 0~31

Command layout

+0	CMD_SETSCRATCH (0xffffff3c)
+4	Handle

Examples

With the setscratch command, set the handle 31, handle 15 is available for application use, for example as a font:


```
cmd_setscratch(31);
cmd_setfont2(15, 100000, 32);
cmd_button(15, 30, 130, 20, 15, 0, "This is font 15");

//Restore bitmap handle 31 to ROM Font number 31.
cmd_romfont(31, 31);
```

Code snippet 18 CMD_SETSCRATCH example

5.61 CMD_ROMFONT – load a ROM font into bitmap handle

By default ROM fonts 16-31 are loaded into bitmap handles 16-31. This command allows any ROM font 16-34 to be loaded into any bitmap handle.

C prototype

void cmd_romfont(uint32_t font,

uint32_t romslot);

Parameters

font

bitmap handle number , 0~31

romslot

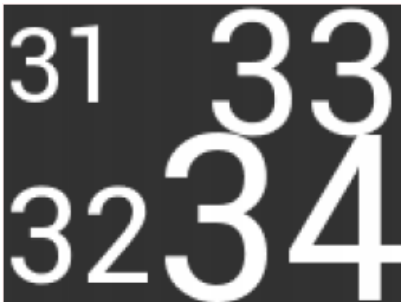
ROM font number, 16~34

Command layout

+0	CMD_ROMFONT (0xffffff3f)
+4	font
+8	romslot

Examples

Loading hardware fonts 31-34 into bitmap handle 1:




```
cmd_romfont(1, 31);
cmd_text( 0, 0, 1, 0, "31");
cmd_romfont(1, 32);
cmd_text( 0, 60, 1, 0, "32");
cmd_romfont(1, 33);
cmd_text(80,-14, 1, 0, "33");
cmd_romfont(1, 34);
cmd_text(60, 32, 1, 0, "34");
```

Code snippet 19 CMD_ROMFONT example

5.62 CMD_TRACK - track touches for a graphics object

The FT81X can assist the MCU in tracking touches on graphical objects. For example touches on dial objects can be reported as angles, saving computation load of the MCU. To do this, the MCU draws the object using a chosen tag value, and registers a track area for that tag.

Any touch on that object is then reported in REG_TRACKER as the first touch point, REG_TRACKER_1 as the second touch, REG_TRACKER_2 as the third touch, REG_TRACKER_3 as the fourth touch point, REG_TRACKER_4 as the fifth touch point.

NOTE: Multiple touch points are only available with capacitive displays (and FT811/FT813 controllers)

C prototype

void cmd_track(int16_t x,

int16_t y,

int16_t w,

int16_t h,

int16_t tag);

Parameters

x

For linear tracker functionality, x-coordinate of track area top-left, in pixels.

For rotary tracker functionality, x-coordinate of track area center, in pixels.

y

For linear tracker functionality, y-coordinate of track area top-left, in pixels.

For rotary tracker functionality, y-coordinate of track area center, in pixels.

w

Width of track area, in pixels.

h

Height of track area, in pixels.

Note:

A w and h of (1,1) means that the tracker is rotary, and reports an

angle value in REG_TRACKER. A w and h of (0,0) disables the track functionality of the co-processor engine. Other values mean that the tracker is linear, and reports values along its length from 0 to 65535 in REG_TRACKER

tag

tag of the graphics object to be tracked, 1-255

Command layout

+0	CMD_TRACK(0xffffff2c)
+4	X
+6	Y
+8	W
+10	h
+12	tag

Description

The Co-processor engine tracks the graphics object in rotary tracker mode and linear tracker mode:

- rotary tracker mode – Track the angle between the touch point and the center of the graphics object specified by the tag value. The value is in units of 1/65536 of a circle. 0

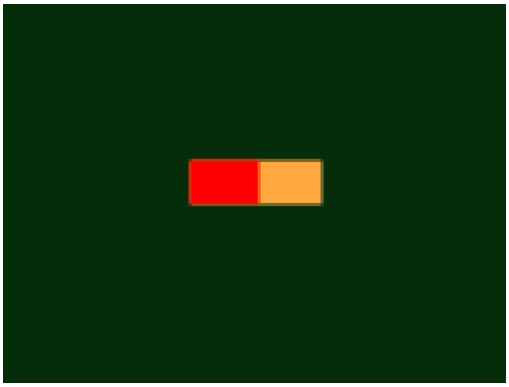
means that the angle is straight down, 0x4000 left, 0x8000 up, and 0xC000 right from the center.

- Linear tracker mode – If parameter w is greater than h, track the relative distance of the touch point to the width of the graphics object specified by the tag value. If parameter w is not greater than h, track the relative distance of touch points to the height of the graphics object specified by the tag value. The value is in units of 1/65536 of the width or height of the graphics object. The distance of the touch point refers to the distance from the top left pixel of graphics object to the coordinate of the touch point.

Please note that the behavior of CMD_TRACK is not defined if the center of the track object (in case of rotary track) or top left of the track object (in case of linear track) is outside the visible region in display panel.

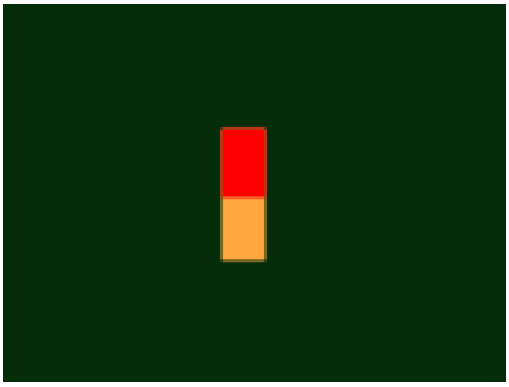
Examples

Horizontal track of rectangle dimension 40x12pixels and the present touch is at 50%:



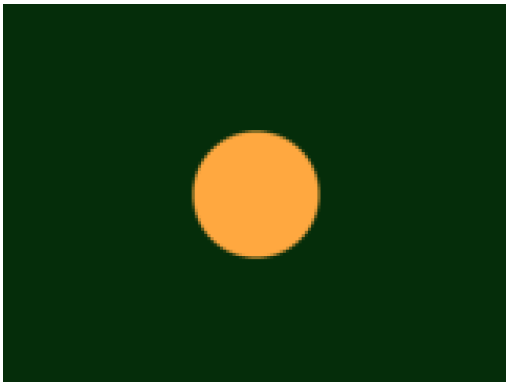
```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(RECTS) );
dl( VERTEX2F(60 * 16,50 * 16) );
dl( VERTEX2F(100 * 16,62 * 16) );
dl( COLOR_RGB(255, 0, 0) );
dl( VERTEX2F(60 * 16,50 * 16) );
dl( VERTEX2F(80 * 16,62 * 16) );
dl( COLOR_MASK(0 ,0 ,0 ,0) );
dl( TAG(1) );
dl( VERTEX2F(60 * 16,50 * 16) );
dl( VERTEX2F(100 * 16,62 * 16) );
cmd_track(60 * 16, 50 * 16, 40, 12, 1);
```

Vertical track of rectangle dimension 12x40 pixels and the present touch is at 50%:



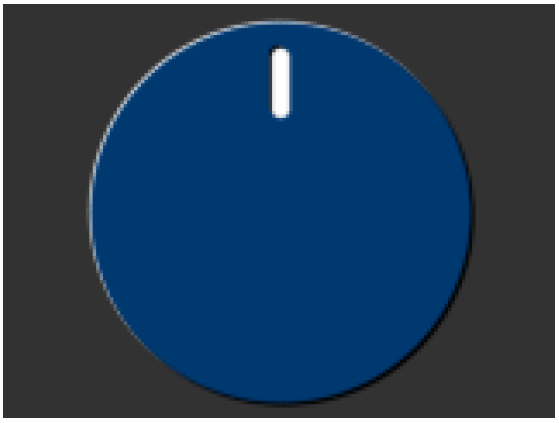
```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(RECTS) );
dl( VERTEX2F(70 * 16,40 * 16) );
dl( VERTEX2F(82 * 16,80 * 16) );
dl( COLOR_RGB(255, 0, 0) );
dl( VERTEX2F(70 * 16,40 * 16) );
dl( VERTEX2F(82 * 16,60 * 16) );
dl( COLOR_MASK(0 ,0 ,0 ,0) );
dl( TAG(1) );
dl( VERTEX2F(70 * 16,40 * 16) );
dl( VERTEX2F(82 * 16,80 * 16) );
cmd_track(70 * 16, 40 * 16, 12, 40, 1);
```

Circular track centered at (80,60) display location



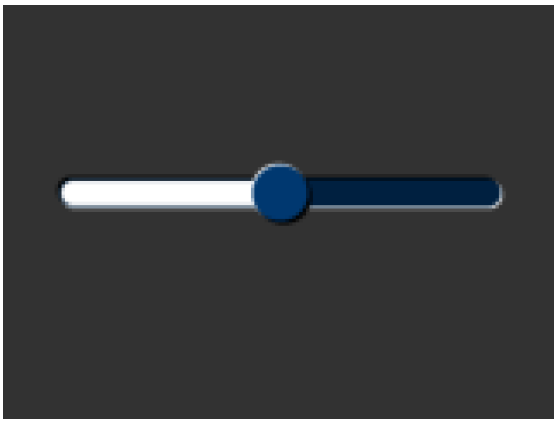
```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( TAG(1) );
dl( BEGIN(POINTS) );
dl( POINT_SIZE(20 * 16) );
dl( VERTEX2F(80 * 16, 60 * 16) );
cmd_track(80 * 16, 60 * 16, 1, 1, 1);
```

To draw a dial with tag 33 centered at (80, 60), adjustable by touch:



```
uint16_t angle = 0x8000;
cmd_track(80, 60, 1, 1, 33);
while (1) {
...
cmd(TAG(33));
cmd_dial(80, 60, 55, 0, angle);
...
uint32_t tracker = rd32(REG_TRACKER);
if ((tracker & 0xff) == 33)
angle = tracker >> 16;
...
}
```

To make an adjustable slider with tag 34:



```
uint16_t val = 0x8000;
cmd_track(20, 50, 120, 8, 34);
Ile (1) {
...
cmd(TAG(34));
cmd_slider(20, 50, 120, 8, val, 65535);
...
uint32_t tracker = rd32(REG_TRACKER);
if ((tracker & 0xff) == 33)
val = tracker >> 16;
...
}
```

5.63 CMD_SNAPSHOT - take a snapshot of the current screen

This command causes the co-processor engine to take a snapshot of the current screen, and write the result into RAM_G as an ARGB4 bitmap. The size of the bitmap is the size of the screen, given by the REG_HSIZE and REG_VSIZE registers.

During the snapshot process, the display should be disabled by setting REG_PCLK to 0 to avoid display glitch.

Because the co-processor engine needs to write the result into the destination address, the destination address must be never used or referenced by the graphics engine.

C prototype

void cmd_snapshot(uint32_t ptr);

Parameters

ptr

Snapshot destination address, in RAM_G

Command layout

+0	CMD_SNAPSHOT(0xffffff1f)
+4	ptr

Examples

To take a snapshot of the current 160 x 120 screens, then use it as a bitmap in the new display list:

```
wr(REG_PCLK,0);//Turn off the PCLK
wr16(REG_HSIZE,120);
wr16(REG_WSIZE,160);

cmd_snapshot(0);//Taking snapshot.

wr(REG_PCLK,5);//Turn on the PCLK
wr16(REG_HSIZE,272);
wr16(REG_WSIZE,480);

cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd(BITMAP_SOURCE(0));
cmd(BITMAP_LAYOUT(ARGB4, 2 * 160, 120));
cmd(BITMAP_SIZE(NEAREST, BORDER, BORDER, 160, 120));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 0, 0));
```

Code snippet 20 CMD_SNAPSHOT 160x120–screen

5.64 CMD_SNAPSHOT2 - take a snapshot of part of the current screen

The snapshot command causes the co-processor to take a snapshot of part of the current screen, and write it into graphics memory as a bitmap. The size, position and format of the bitmap may be specified. During the snapshot process, the display output process is suspended. LCD displays can easily tolerate variation in display timing, so there is no noticeable flicker.

C prototype

```
void cmd_snapshot2( uint32_t fmt,
                    uint32_t ptr,
                    int16_t  x,
                    int16_t  y,
                    int16_t  w,
                    int16_t  h);
```

Parameters

fmt

Output bitmap format, one of RGB565, ARGB4 or **0x20**. The value **0x20** produces an ARGB8 format snapshot.

See BITMAP_LAYOUT for format list.

ptr

Snapshot destination address, in RAM_G

x

x-coordinate of snapshot area top-left, in pixels

y

y-coordinate of snapshot area top-left, in pixels

w

width of snapshot area, in pixels. Note when *fmt* is **0x20**, i.e. in ARGB8 format, the value of width shall be doubled.

h

height of snapshot area, in pixels

Command layout

+0	CMD_SNAPSHOT2(0xffffff37)
+4	fmt
+8	ptr

+12	x
+14	y
+16	w
+18	h

Examples

To take a 32x32 snapshot of the top-left of the screen, then use it as a bitmap in the new display list:

```
cmd_snapshot2(RGB565, 0, 0, 0, 32, 32);
cmd_dlstart();
cmd_setbitmap(0, RGB565, 32, 32);
cmd(CLEAR(1,1,1));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 0, 0));
```

Code snippet 21 CMD_SNAPSHOT2 32x32 screen

Note:

For **ARGB8** format, pixel memory layout is as below:

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
A	R	G	B

5.65 CMD_SETBITMAP – set up display list for bitmap

This command will generate the corresponding display list commands (**BITMAP_SOURCE\BITMAP_LAYOUT\BITMAP_SIZE**) for given bitmap information, sparing the effort of writing display list manually.

The parameters filter/wrapx/wrapy in BITAMP_SIZE is always set to NEAREST/BORDER/BORDER value in the generated display list commands.

C prototype

```
void cmd_setbitmap( uint32_t   addr,
                    uint16_t   fmt,
                    uint16_t   width,
                    uint16_t   height  );
```

Parameters

addr

Address of bitmap data in RAM_G.

fmt

Bitmap format, see the definition in BITMAP_LAYOUT.

width

bitmap width, in pixels.

height

bitmap height, in pixels

Command layout

+0	CMD_SETBITMAP(0xffff ff43)
+4	addr
+8	fmt
+10	width
+12	height

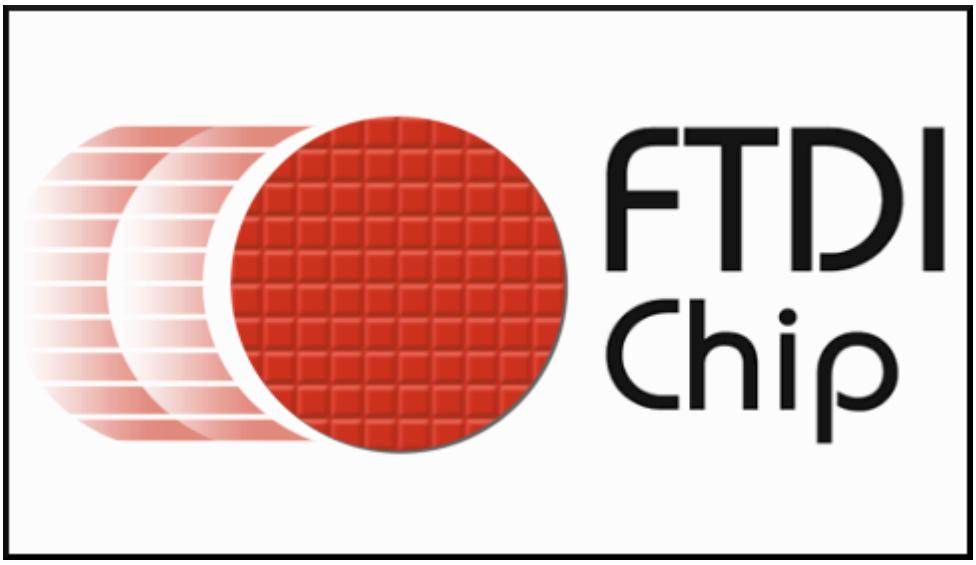
Examples

See CMD_SNAPSHOT2 - take a snapshot of part of the current screen.

Note

Two bytes needs to be appended after last parameter for 4 bytes alignment

5.66 CMD_LOGO - play FTDI logo animation



The logo command causes the co-processor engine to play back a short animation of the FTDI logo. During logo playback the MCU should not access any FT81X resources. After 2.5 seconds have elapsed, the co-processor engine writes zero to REG_CMD_READ and REG_CMD_WRITE, and starts waiting for commands. After this command is complete, the MCU shall write the next command to the starting address of RAM_CMD.

C prototype

void cmd_logo( );

Command layout

+0	CMD_LOGO(0xffffff31)

Examples

To play back the logo animation:

```
cmd_logo();
delay(3000);        // Optional to wait
While( (0 != rd16(REG_CMD_WRITE))  &&
          (rd16(REG_CMD_WRITE)   != rd16(REG_CMD_READ) ));//Wait till both read &
write pointer register are equal to zero
```

Code snippet 22 CMD_LOGO command example

5.67 CMD_CSKETCH – Deprecated

This command is the legacy command from the FT801 chip. Users are recommended to use "CMD_SKETCH" for FT81X since it works for both RTE and CTE.

C prototype

```
void cmd_csketch( int16_t x,
                  int16_t y,
                  uint16_t w,
                  uint16_t h,
                  uint32_t ptr,
                  uint16_t format,
                  uint16_t freq);
```

Command layout

+0	CMD_CSKETCH(0xffffff35)
+4	X
+6	Y
+8	W
+10	H
+12	Ptr
+16	Format
+18	Freq

Parameters

x

x-coordinate of sketch area top-left, in pixels

y

y-coordinate of sketch area top-left, in pixels

w

Width of sketch area, in pixels

h

Height of sketch area, in pixels

ptr

Base address of sketch bitmap

format

Format of sketch bitmap, either L1 or L8

Freq

Deprecated.

6 Contact Information

Head Quarters – Singapore

Bridgetek Pte Ltd
178 Paya Lebar Road, #07-03
Singapore 409030
Tel: +65 6547 4827
Fax: +65 6841 6071

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office – Taipei, Taiwan

Bridgetek Pte Ltd, Taiwan Branch
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District
Taipei 114
Taiwan, R.O.C.
Tel: +886 (2) 8797 5691
Fax: +886 (2) 8751 9737

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom

Bridgetek Pte. Ltd.
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales.emea@brtichip.com
E-mail (Support) support.emea@brtchip.com

Branch Office – Vietnam

Bridgetek VietNam Company Limited
Lutaco Tower Building, 5th Floor, 173A Nguyen Van Troi,
Ward 11, Phu Nhuan District,
Ho Chi Minh City, Vietnam
Tel : 08 38453222
Fax : 08 38455222

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Web Site

http://brtchip.com/

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Document References

FT81X Datasheet: DS_FT81X

SAMPLE PROJECTS: http://www.ftdichip.com/Support/SoftwareExamples/FT800_Projects.htm

OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.4

Acronyms and Abbreviations

Terms	Description
CS	Chip select
CTE	Capacitive Touch Engine
DL	Display list
EVE	Embedded Video Engine
FPS/fps	Frame Per Second
GPIO	General Purpose Input/output
Hz/KHz/MHz	Hertz/Kilo Hertz/Mega Hertz
I^2C	Inter-Integrated Circuit
LSB	least significant bit
MCU	Micro controller unit
MSB	most significant bit
OS	operating system
PWM	Pulse-width modulation
PWR	Power
RAM	Random access memory
RAM font	Custom font, resided in RAM_G
RGB	Red Blue Green
R/W	Read and Write
RO	Read only

ROM font	Built-in font, resided in ROM
RTE	Resistive Touch Engine
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
USB-IF	USB Implementers Forum
WO	Write Only

Memory Map

Start Address	End Address	Size	NAME	Description
00 0000h	0F FFFFh	1024 KB	RAM_G	General purpose RAM, also called "main memory"
30 0000h	30 1FFFh	8 KB	RAM_DL	Display List RAM
30 2000h	30 2FFFh	4 KB	RAM_REG	Registers
30 8000 h	30 8FFFh	4 KB	RAM_CMD	Co-processor command circular buffer

Note 1: The addresses beyond this table are reserved and shall not be read or written unless otherwise specified.

Note 2: ROM_FONTROOT is defined as *0x2FFFFC*

Appendix B – List of Figures/Tables/Code Snippets

List of Figures

List of Tables

List of Code Snippets

List of Registers

Appendix C – Revision History

Document Title: FT81x Series Programmers Guide

Document Reference No.: BRT_000031

Clearance No.: BRT#035

Product Page: http://brtchip.com/product

Document Feedback: Send Feedback

Revision	Changes	Date
0.1	Initial Draft Release	2015-02-17
1.0	First full release	2015-09-25
1.1	Updated Section 5.5.1 - Table 11 - FT81x Font Metrics block format- Included support for L2 Format in font metrics block Dual branding to reflect the migration of the product to the Bridgetek name – logo changed, copyright changed, contact information changed	2016-09-19