Future Technology Devices International Ltd.

Programmer Guide

Document Reference No.: FT_000793

Version 0.8

Issue Date: 2013-08-29

This document is a Programmer Guide for the FT800 chip. This guide details the chip features and procedures for use.

Table of Content

List of Code Snippet

List of Figures

List of Tables

List of Registers

1 Introduction

This document captures programming details of FT800 including graphics commands, widget commands and configurations useful to control the FT800 for smooth and vibrant screen effects.

FT800 is a graphics chip with added features such as audio playback and touch capabilities. FT800 graphics 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 white goods, toys, industrial machinery, simple phones, elevators, and many more.

1.1 Overview

The FT800 Programmer Guide will be useful to understand the FT800 command set and demonstrate the ease of command usage in the examples given for each specific instruction. The Programmer Guide will also be useful to understand various power modes, audio, and touch features and their usage.

Information on pin settings, hardware model and hardware configuration can be found in the FT800 data sheet (DS_FT800_Embedded_Video_Engine).

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 either an SPI or I^2C 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 co-processor engine FIFO (RAM_CMD).

dl() – Write the specified 32 bits display list command to RAM_DL. Please see the 2.5.4 Writing display lists for more information.

host_command() – send host command to FT800. Refer FT800 data sheet for more information.

1.4 Definitions of co-processor widgets OPTION parametersParameters option name definitions

The following table defines the co-processor widgets OPTION parameters' names mentioned in this document: (In this table, co-processor widget refers to graphics objects drawn by corresponding co-processor engine command)

Table 1 Parameter option name definitions

Name	Value	Description	Commands
OPT_3D	0	Co-processor widget is drawn in 3D effect. The default option.	CMD_BUTTON,CMD_CLOCK,CMD_KEYS, CMD_GAUGE,CMD_SLIDER, CMD_DIAL, CMD_TOGGLE,CMD_PROGRESS, CMD_SCROLLBAR
OPT_RGB565	0	Co-processor option to decode the JPEG image to RGB565 format	CMD_IMAGE
OPT_MONO	1	Co-processor option to decode the JPEG image to L8 format, i.e., monochrome	CMD_IMAGE
OPT_NODL	2	No display list commands generated for bitmap decoded from JPEG image	CMD_IMAGE
OPT_FLAT	256	Co-processor widget is drawn without 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 32 bit signed integer	CMD_NUMBER

Name	Value	Description	Commands
OPT_CENTERX	512	Co-processor widget centers horizontally	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTERY	1024	Co-processor widget centers vertically	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_CENTER	1536	Co-processor widget centers horizontally and vertically.	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_RIGHTX	2048	The label on the Co-processor widget is right justified	CMD_KEYS,CMD_TEXT, CMD_NUMBER
OPT_NOBACK	4096	Co-processor widget has no background drawn	CMD_CLOCK, CMD_GAUGE
OPT_NOTICKS	8192	Co-processor clock widget is drawn without hour ticks. Gauge widget is drawn without major and minor ticks	CMD_CLOCK, CMD_GAUGE
OPT_NOHM	16384	Co-processor clock widget is drawn without hour and minutes hands, only seconds hand is drawn	CMD_CLOCK
OPT_NOPOINTER	16384	The Co-processor gauge has no pointer	CMD_GAUGE
OPT_NOSECS	32768	Co-processor clock widget is drawn without seconds hand	CMD_CLOCK
OPT_NOHANDS	49152	Co-processor clock widget is drawn without hour, minutes and seconds hands	CMD_CLOCK

2 Programming Model

The FT800 appears to the host MCU as a memory-mapped SPI or I^2C device. The host communicates with the FT800 using Read or Write to an 8 megabyte address space.

The graphics engine and all other engines are '*Little Endian*' format. Endianness of DL commands, co-processor engine commands, register values read/write, input RGB bitmap data and ADPCM input data are in '*Little Endian*' format.

The host reads and writes the FT800 address space using SPI transactions. These transactions are either read or write commands. Serial data is sent with the most significant bit first. For I^2C transactions the same bytes sequence is encapsulated in the I^2C protocol.

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. FT800 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 or I^2C driver.

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

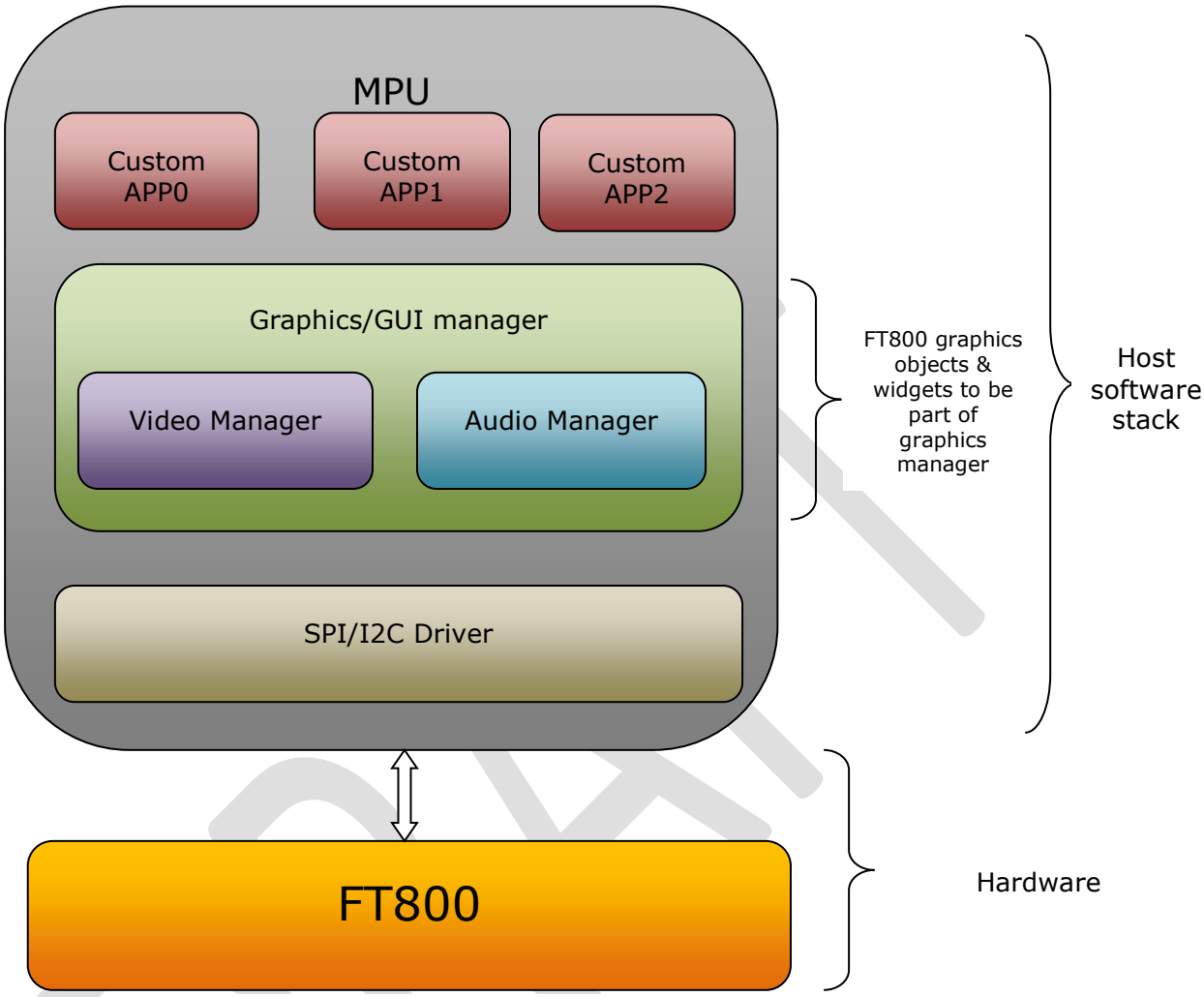


Figure 1: Software Architecture

2.2 Display configuration and initialization

To configure the display, load the timing control registers with values for the particular display. These registers control horizontal timing:

- REG_PCLK
- REG_PCLK_POL
- REG_HCYCLE
- REG_HOFFSET

- REG_HSIZE
- REG_HSYNC0
- REG_HSYNC1

These registers control vertical timing:

- REG_VCYCLE
- REG_VOFFSET
- REG_VSIZE
- REG_VSYNC0
- REG_VSYNC1

And the REG_CSPREAD register changes color clock timing to reduce system noise.

GPIO bit 7 is used for the display enable pin of the LCD module. By setting the direction of the GPIO bit to out direction, the display can be enabled by writing value of 1 into GPIO bit 7 or the display can be disabled by writing a value of 0 into GPIO bit 7. By default GPIO bit 7 direction is output and the value is 0.

Note: Refer to FT800 data sheet for information on display register set.

2.2.1 Horizontal timing

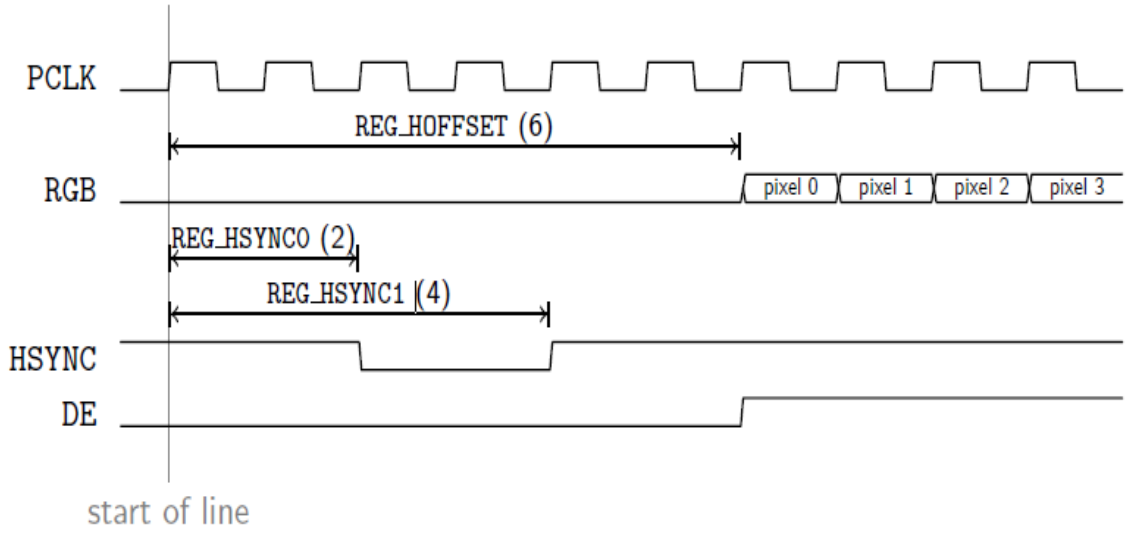


Figure 2: Horizontal Timing

REG_PCLK controls the frequency of PCLK. The register specifies a divisor for the main 48 MHz clock, so a value of 4 gives a 12 MHz PCLK. If REG_PCLK is zero, then all display output is suspended. REG_PCLK_POL controls the polarity of PCLK. Zero means that display data is clocked out on the rising edge of PCLK. One means data is clocked on the falling edge.

The total number of PCLKs in a horizontal line is REG_HCYCLE. Within this horizontal line are the scanned out pixels, REG_HSIZE in total. They start after REG_HOFFSET cycles. Signal DE is high while pixels are being scanned out.

Horizontal sync timing on signal HSYNC is controlled by REG_HSYNC0 and REG_HSYNC1. They specify the time at which HSYNC falls and rises respectively.

2.2.2 Vertical timing

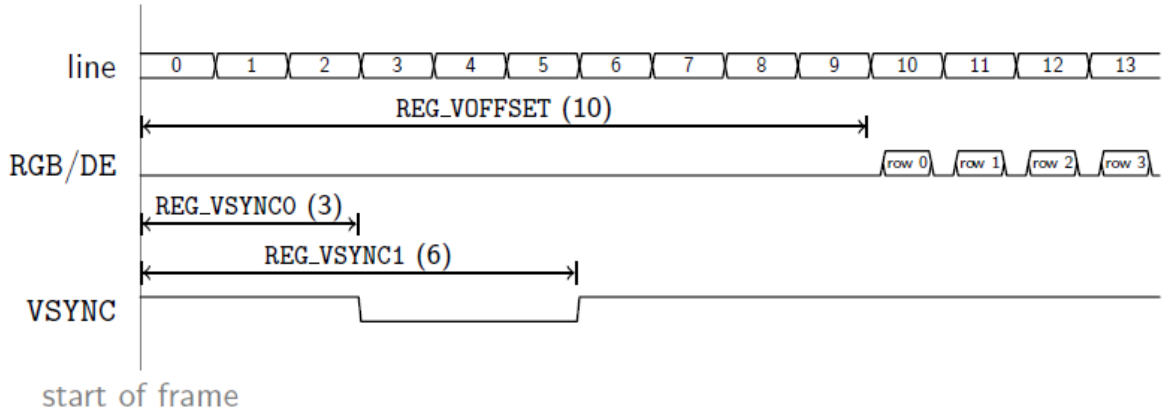


Figure 3: Vertical Timing

Vertical timing is specified in number of lines. The total number of lines in a frame is REG_VCYCLE. There are REG_VSIZE rows of pixels in total. They start after REG_VOFFSET cycles.

Vertical sync timing on signal VSYNC is controlled by REG_VSYNC0 and REG_VSYNC1. They specify the lines at which VSYNC falls and rises respectively.

2.2.3 Signals updating timing control

With REG_CSPREAD disabled, all color signals are updated at the same time:

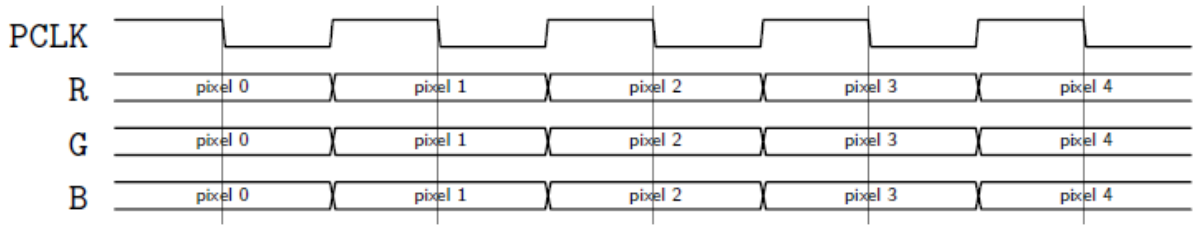


Figure 4: Pixel clocking with no CSPREAD

But with REG_CSPREAD enabled, the color signal timings are adjusted slightly so that fewer signals change simultaneously:

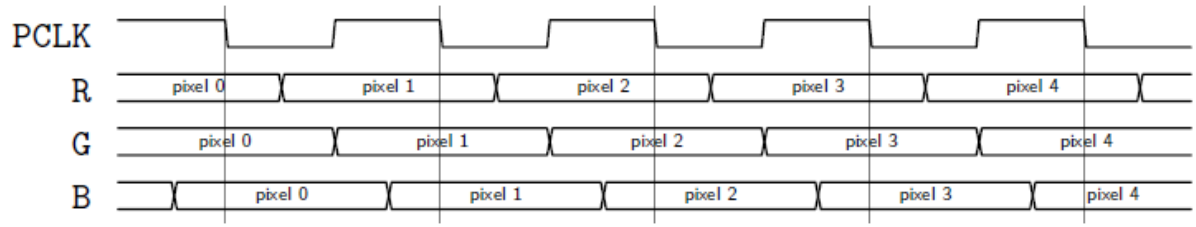


Figure 5: Pixel clocking with CSPREAD

2.2.4 Timing example: 480x272 at 60Hz

For a display updating at 60Hz, there are 48000000/60= 800000 fast clocks per frame. Setting the PCLK divisor REG_PCLK to 5 gives a PCLK frequency of 9.6 MHz and

800000/5= 160000PCLKs per frame.

For a 480 x 272 display, the typical horizontal period is 525 clocks, and vertical period is 286 lines. A little searching shows that a 548 x 292 size gives a period of 160016 clocks, very close to the target. So with a REG_HCYCLE=548 and REG_VCYCLE=292 the display frequency is almost exactly 60Hz. The other register settings can be set directly from the display panel datasheet.

2.2.5 Initialization Sequence

When REG_PCLK is zero, FT800 generates no video signal. The MCU can start the FT800 cleanly in this order:

Initialization Sequence during the Boot Up:

1. Use MCU SPI clock not more than 11MHz
2. Send Host command "CLKEXT" to FT800
3. Send Host command "ACTIVE" to enable clock to FT800.
4. Configure video timing registers, except REG_PCLK
5. Write first display list
6. Write REG_DLSWAP, FT800 swaps display list immediately
7. Enable back light control for display
8. Write REG_PCLK, video output begins with the first display list
9. Use MCU SPI clock not more than 30MHz

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

host_command(ACTIVE);//send host command "ACTIVE" to FT800

host_command(CLKEXT);//send command to "CLKEXT" to FT800

/* 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 | Ft_Gpu_Hal_Rd8(phost,REG_GPIO_DIR));
wr8(REG_GPIO,0x080 | Ft_Gpu_Hal_Rd8(phost,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

Execute ”Initialization Sequence during the Boot UP” from steps 1 to 9

Initialization Sequence from Sleep Mode:

1. Send Host command “ACTIVE” to enable clock to FT800
2. Wait for at least 20ms

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 FT800 data sheet for information on power modes. Follow section 2.3 for audio management during power down and reset operations.

2.3 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 sound synthesizer 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 sound synthesizer 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 sound synthesizer stop playing sound

To avoid a 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 FT800 data sheet for more information on sound synthesizer and audio playback.

2.4 Audio playback

FT800 supports 3 types of audio format: 4 Bit IMA ADPCM, 8 Bit signed PCM, 8 Bit u-Law. For IMA ADPCM format, please note the byte order: within one byte, first sample (4 bits) shall locate from bit 0 to bit 3, while the second sample (4 bits) shall locate from bit 4 to bit 7.

For the audio data in FT800 RAM to play, FT800 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

This section captures a few of the graphics screen shot examples.

2.5.1 Graphics Coordinate Plane

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

The valid X and Y coordinate ranges from -1024 to 1023 in pixel precision, i.e., from -16384 to 16383 in 1/16th pixel precision.

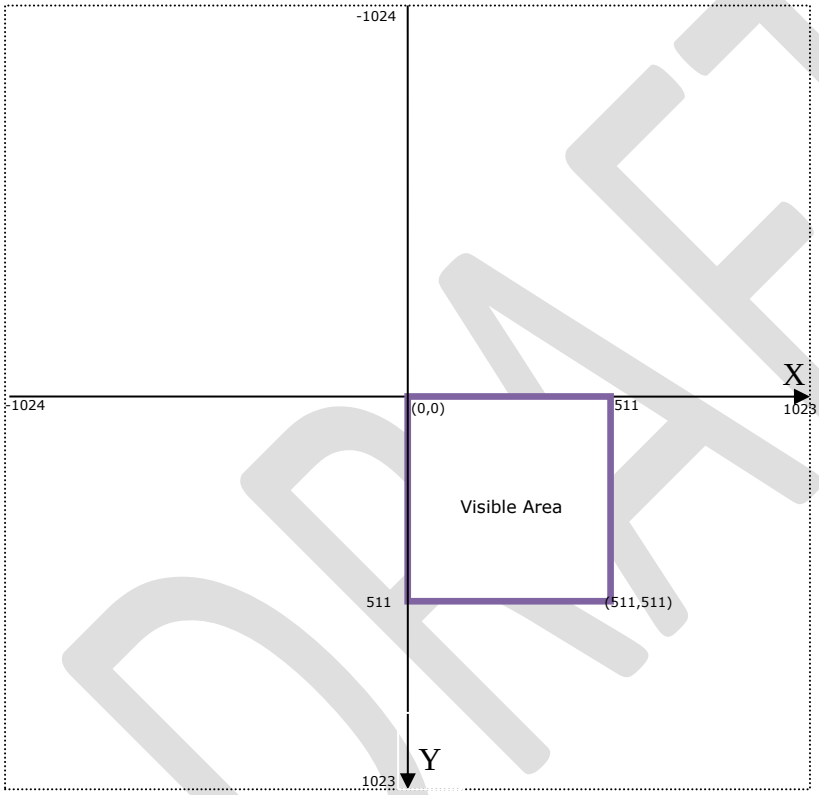


Figure 6 FT800 graphics coordinates plane in pixel precision

2.5.2 Getting started

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

Figure 7: Getting Start Example Image

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 color 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, 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 bit width.
- Command CLEAR is recommended to be used before any other drawing operation, in order to put FT800 graphics engine in a known state.
- The end of the display list is always flagged with the command DISPLAY

2.5.3 BEGIN and END

The general pattern for drawing is:

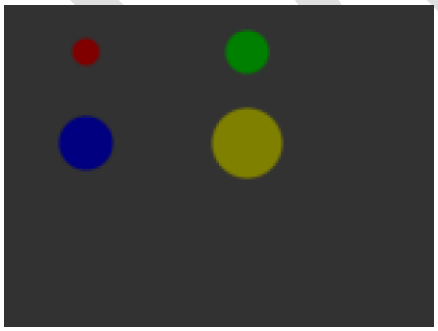
- BEGIN with one of the primitive types
- Input one or more vertices, which specify the placement of the primitive on the screen
- END to mark the end of the primitive

The primitive types that the graphics engine can draw are:

- BITMAPS - rectangular pixel arrays, in various color formats
- POINTS - anti-aliased points, point radius is 1-256 pixels
- LINES - anti-aliased lines, with width from 0 to 4095 1/16th of pixel units. (width is from center of the line to boundary)
- LINE_STRIP - anti-aliased lines, connected head-to-tail
- RECTS - round-cornered rectangles, curvature of the corners can be adjusted using LINE_WIDTH.
- EDGE_STRIP_A/B/L/R - edge strips

Examples

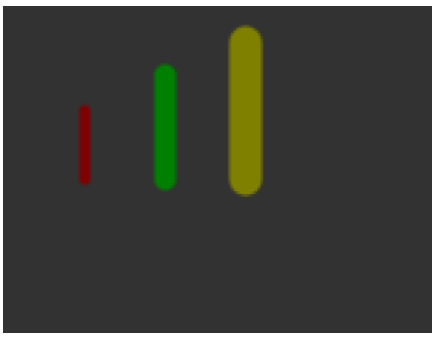
Draw points with varying radius from 5 pixels to 13 pixels with different colors:



```
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 till 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) );
```

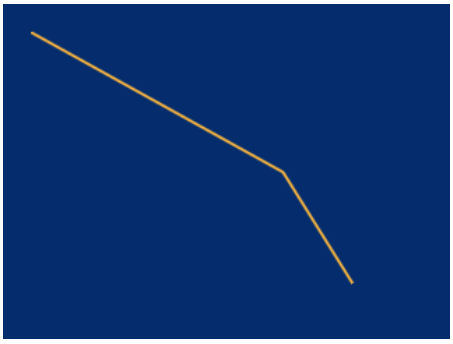
The VERTEX2F commands are in pairs to define the start and finish point of the line.

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

2.5.4 Writing display lists

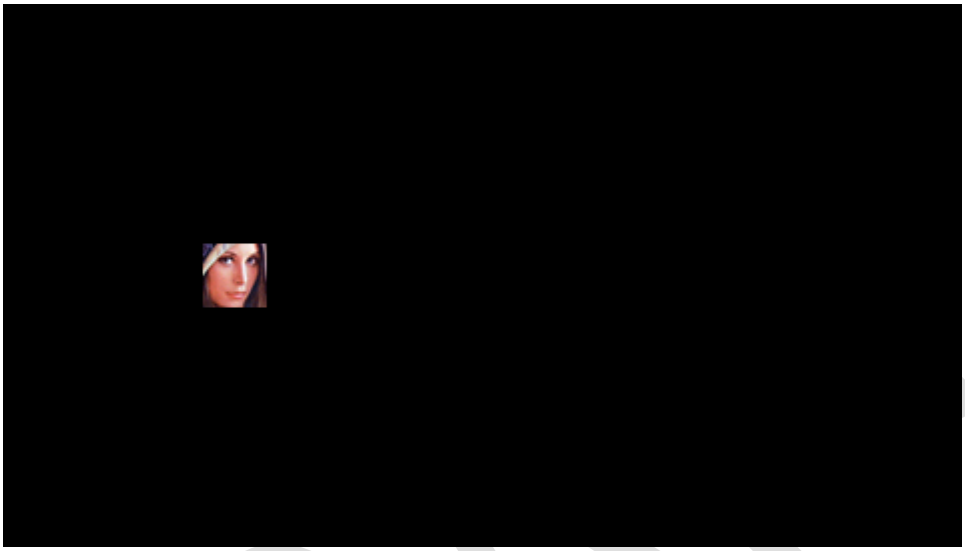
Writing display list entries with wr32() is time-consuming and error-prone, so instead a function might be used:

```
static size_t dli;
static void dl(unsigned long cmd)
{
wr32(RAM_DL + dli, cmd);
dli += 4;
}
...
dli = 0; // start writing the display list
dl(CLEAR(1, 1, 1)); // clear screen
dl(BEGIN(BITMAPS)); // start drawing bitmaps
dl(VERTEX2II(220, 110, 31, 'F')); // ascii F in font 31
dl(VERTEX2II(244, 110, 31, 'T')); // ascii T
dl(VERTEX2II(270, 110, 31, 'D')); // ascii D
dl(VERTEX2II(299, 110, 31, 'I')); // ascii I
dl(END());
dl(COLOR_RGB(160, 22, 22)); // change color to red
dl(POINT_SIZE(320)); // set point size
dl(BEGIN(POINTS)); // start drawing points
dl(VERTEX2II(192, 133, 0, 0)); // red point
dl(END());
dl(DISPLAY()); // display the image
```

Code snippet 9 dl function definition

2.5.5 Bitmap objects

The BITMAP_SOURCE command tells the graphics engine where to fetch the bitmap's pixel from in RAM. BITMAP_LAYOUT specifies the memory layout of the bitmap. This includes the pixel format - here it is RGB565 - and the width of the bitmap and its height. BITMAP_SIZE tells the graphics engine how to draw the bitmap on the screen, including its width and height in pixels, here 10 x 10.



```
dl(CLEAR(1, 1, 1)); // clear screen
dl(BITMAP_SOURCE(0));
dl(BITMAP_LAYOUT(RGB565, 20, 10));
dl(BITMAP_SIZE(NEAREST, BORDER, BORDER, 10, 10));
dl(BEGIN(BITMAPS)); // start drawing bitmaps
dl(VERTEX2II(100, 120, 0, 0)); // draw the image at (100,120)
dl(END());
dl(DISPLAY()); // display the image
```

Code snippet 10 Bitmap object

2.5.6 Bitmap transformation matrix

To achieve the bitmap transformation, the bitmap transform matrix below is specified in the FT800 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 identity matrix.

The coordinates x' , y' after transforming is calculated in 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.

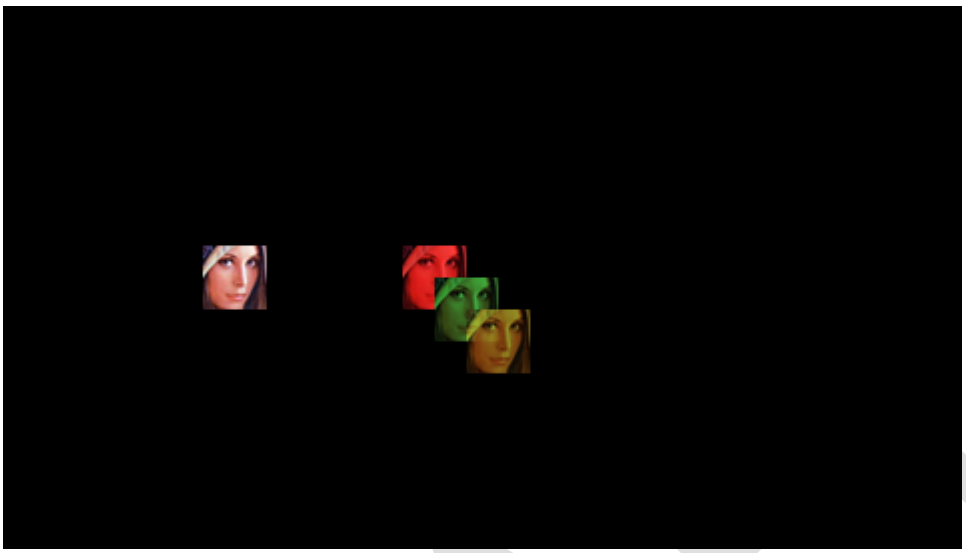
2.5.7 Color and transparency

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

```
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 11 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.



2.5.8 Screen coordinates using VERTEX2II and VERTEX2F

The VERTEX2II command used above only allows positive screen coordinates. If the bitmap is partially off screen, for example during a screen scroll, then it is necessary to specify negative screen coordinates. The VERTEX2F command allows negative coordinates. It also allows fractional coordinates, because it specifies screen (x,y) in units of 1/16 of a pixel.

For example, drawing the same bitmap at screen position (-10,-10) using VERTEX2F:


```
dl(BEGIN(BITMAPS));
dl(VERTEX2F(-160, -160));
dl(END());
```

Code snippet 12 negative screen coordinates example

2.5.9 Clearing a rectangular area using scissor

Graphic drawing operations are all limited by a scissor clip rectangle that controls the pixels that all drawing operations can modify. By default the scissor clip rectangle is the same size as the screen, allowing all pixels to be drawn.

Changing the size of the scissor rectangle using commands SCISSOR_SIZE and SCISSOR_XY, restricts drawing to the specified rectangle. In this example, changing the scissor to a square area, then clearing the screen to yellow results in a yellow square.

```
dl(Clear(1,1,1)); // Clear to black
dl(ScissorXY(40, 20)); // Scissor rectangle top left at (40, 20)
dl(ScissorSize(40, 40)); // Scissor rectangle is 40 x 40 pixels
dl(ClearColorRGB(255, 255, 0)); // Clear to yellow
dl(Clear(1, 1, 1));
dl(Display());
```

Code snippet 13 Clearing rectangular area using scissor

2.5.10 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 (REG_PCLK and REG_HCYCLE) but is never less than 2048 internal clock cycles.

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 graphics engine performance rending pixels is 4 pixels per clock, for any line with 2048 display commands the total pixels performance drawn must be less than 8192.
- For some bitmap formats, the drawing rate is 1 pixel per clock. These are TEXT8X8, TEXTVGA and PALETTED.
- For bilinear filtered pixels, the drawing rate is reduced to ¼ pixel per clock. Most bitmap formats draw at 1 pixel per clock, and the above formats (TEXT8X8, TEXTVGA and PALETTED) draw at 1 pixel every 4 clocks.

To summarize:

Table 2 Bitmap rending performance

Filter Mode	Format	Rate
Nearest	TEXT8X8, TEXTVGA and PALETTED	1 pixel per clock
Nearest	all other formats	4 pixel per clock
BILINEAR	TEXT8X8, TEXTVGA and PALETTED	1/4 pixel per clock
BILINEAR	all other formats	1 pixel per clock

3 FT800 Register Description

In this chapter, all the registers in the FT800 are classified into 5 groups: Graphics Engine Registers, Audio Engine Registers, Touch Engine Registers, and Co-processor Engine Registers as well as Miscellaneous Registers. This chapter gives the detailed definition for each FT800 register. To view the register summary of the FT800, there is one register summary table in the datasheet of FT800.

In addition, in this chapter, please note all the reserved bits are read-only and shall be zero. All the hexadecimal values are prefixed with 0x. Readers are encouraged to cross-reference the other chapters of this document for a better understanding.

3.1 Graphics Engine Registers

Register Definition 1 REG_PCLK Definition

REG_PCLK Definition	
Reserved	R/W

31 8 7 0

Address: 0x10246C **Reset Value: 0x0**

Bit 0 - 7 : These bits are set to divide the main clock for PCLK. If the typical main clock was 48MHz and the value of these bits are 5, the PCLK will be 9.6 MHz. If the value of these bits are zero, there will be no PCLK output.

Note: NONE

Register Definition 2 REG_PCLK_POL Definition

REG_PCLK_POL Definition	
Reserved	R/W

31 1 0

Address: 0x102468 **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.

Note: NONE

Register Definition 3 REG_CSPREAD Definition

Please check the sector 2.2.3 for more details.

REG_CSPREAD Definition	
Reserved	R/W

31 1 0

Address: 0x102464 **Reset Value: 0x1**

Bit 0 : This bit controls the transition of RGB signals with PCLK active clock edge. When REG_CSPREAD=0, R[7:2],G[7:2] and B[7:2] signals change following the active edge of PCLK. When REG_CSPREAD=1, R[7:2] changes a PCLK clock early and B[7:2] a PCLK clock later, which helps reduce the system noise .

Bit 1 - 31: Reserved.
Note: NONE

Register Definition 4 REG_SWIZZLE Definition

REG_SWIZZLE Definition	
Reserved	R/W

31 4 3 0

Address: 0x102460 **Reset Value: 0x0**

Bit 0 - 3 : These bits are set to control the arrangement of output RGB pins, which may help support different LCD panel. Please check the table above for details.

Note: NONE

Table 3 REG_SWIZZLE and RGB pins mapping table

REG_SWIZZLE				PINS			
b3	b2	b1	b0	R7, R6, R5, R4, R3, R2	G7, G6, G5, G4, G3, G2	B7, B6, B5, B4, B3, B2	
0	X	0	0	R[7:2]	G[7:2]	B[7:2]	Power on Default
0	X	0	1	R[2:7]	G[2:7]	B[2:7]	
0	X	1	0	B[7:2]	G[7:2]	R[7:2]	
0	X	1	1	B[2:7]	G[2:7]	R[2:7]	
1	0	0	0	G[7:2]	B[7:2]	R[7:2]	
1	0	0	1	G[2:7]	B[2:7]	R[2:7]	
1	0	1	0	G[7:2]	R[7:2]	B[7:2]	
1	0	1	1	G[2:7]	R[2:7]	B[2:7]	
1	1	0	0	B[7:2]	R[7:2]	G[7:2]	
1	1	0	1	B[2:7]	R[2:7]	G[2:7]	
1	1	1	0	R[7:2]	B[7:2]	G[7:2]	
1	1	1	1	R[2:7]	B[2:7]	G[2:7]	

Register Definition 5 REG_DITHER Definition

REG_DITHER Definition	
Reserved	R/W
31 1	0

Address: 0x10245C **Reset Value: 0x1B6**

Bit 0 : Set to 1 to enable dithering feature of output RGB signals. 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.

Note: Please refer to REG_SWIZZLE and RGB pins mapping table for details

Register Definition 6 REG_OUTBITS Definition

REG_OUTBITS Definition	
Reserved	R/W
31 9	8 0

Address: 0x102458 **Reset Value: 0x1B6**

Bit 0 - 8: These 9 bits are split into 3 groups for Red, Green and Blue color output signals:
Bit 0 - 2: Blue color signal lines number. Reset value is 6.
Bit 3 - 5: Green Color signal lines number. Reset value is 6.
Bit 6 - 8: Red Color signal lines number. Reset value is 6.
Host can write these bits to control the numbers of output signals for each color.

Note: NONE

Register Definition 7 REG_ROTATE Definition

REG_ROTATE Definition	
Reserved	R/W
31 1	0

Address: 0x102454 **Reset Value: 0x00**

Bit 0: 180 degree screen Rotation switch. Writing this bit to 0 will turn off the rotation functionality. Writing this bit to 1 will turn on the rotation functionality and 180 degree rotation will take place at the next frame rendered. Reading this bit will reflect the current rotation switch

Note: After rotation is turned on, please do the screen calibration again

Register Definition 8 REG_VSYNC1 Definition

REG_VSYNC1 Definition	
	R/W
31 10	9 0

Address: 0x10244C **Reset Value: 0x00A**

Bit0 - 9: The value of these bits specifies how many lines for signal VSYNC takes at the start of new frame.

Note: NONE

Register Definition 9 REG_VSYNC0 Definition

REG_VSYNC0 Definition	
	R/W
31 9	0

Address: 0x102448 **Reset Value: 0x000**

Bit0 - 9: The value of these bits specifies how many lines for the high state of signal VSYNC takes at the start of new frame.

Note: NONE

Register Definition 10 REG_VSIZE Definition

REG_VSIZE Definition	
Reserved	R/W
31 10 9	0

Address: 0x102444 **Reset Value: 0x110**

Bit0 - 9: The value of these bits specifies how many lines of pixels in one frame.

Note:

Register Definition 11 REG_VOFFSET Definition

REG_VOFFSET Definition	
Reserved	R/W
31 9	0

Address: 0x102440 **Reset Value: 0x00C**

Bit0 - 9: The value of these bits specifies how many lines takes after the start of new frame.

Note:

Register Definition 12 REG_VCYCLE Definition

REG_VCYCLE Definition	
Reserved	R/W
31 10 9	0

Address: 0x10243C **Reset Value: 0x124**

Bit0 - 9: The value of these bits specifies how many lines in one frame.

Note:

Register Definition 13 REG_HSYNC1 Definition

REG_HSYNC1 Definition	
Reserved	R/W
31 9	0

Address: 0x102438 **Reset Value: 0x029**

Bit0 - 9: The value of these bits specifies how many PCLK cycles for HSYNC during start of line.

Note: NONE

Register Definition 14 REG_HSYNC0 Definition

REG_HSYNC0 Definition	
Reserved	R/W
31 10 9	0

Address: 0x102434 **Reset Value: 0x0**

Bit0 - 9: The value of these bits specifies how many PCLK cycles of HSYNC high state during start of line.

Note: NONE

Register Definition 15 REG_HSIZE Definition

Please reference to section 2.2.1

REG_HSIZE Definition	
Reserved	R/W
31 10	9 0

Address: 0x102430 **Reset Value: 0x1E0**

Bit0 - 9: These bits are used to specify the numbers of PCLK cycles per horizonal line.

Note: NONE

Register Definition 16 REG_HOFFSET Definition

Please reference to section 2.2.1

REG_HOFFSET Definition	
Reserved	R/W
31 10	9 0

Address: 0x10242C **Reset Value: 0x2B**

Bit0 - 9: These bits are used to specify the numbers of PCLK cycles before pixels are scanned out.

Note: NONE

Register Definition 17 REG_HCYCLE

Please reference to section 2.2.1

REG_HCYCLE Definition	
Reserved	R/W
31 9	0

Address: 0x102428 **Reset Value: 0x224**

Bit0 - 9: 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.

Note: NONE

Register Definition 18 REG_TAP_MASK

REG_TAP_MASK Definition
R/W
31 0

Address: 0x102424 **Reset Value: 0xFFFFFFFF**

Bit0 - 31: These bits are used to mask the value of RGB output signals. The result will be used to caculate the CRC value which will be updated into REG_TAP_CRC.

Note: NONE

Register Definition 19 REG_TAP_CRC Definition

REG_TAP_CRC Definition	
Read Only	
31	0

Address: 0x102420 **Reset Value: 0x00000000**

Bit0 - 31: These bits are set by FT800 as the CRC value of RGB signals output. It updates once every time display list is rendered.

Note: NONE

Register Definition 20 REG_SNAPSHOT definition

REG_SNAPSHOT Definition	
Reserved	R/W
31	0

Address: 0x102418 **Reset Value: 0x00**

Bit 0: This bit will enable the graphics engine to capture the line of current screen shot and store each pixel data on the line into RAM_Composite.

Note: NONE

Register Definition 21 REG_SNAPY Definition

REG_SNAPY Definition	
Reserved	R/W
31 9	8 0

Address: 0x102414 **Reset Value: 0x00**

Bit 0 - 8: These bits specify which line will be rendered if render mode is in single line mode. It does not have an impact if render mode is in normal mode. The valid range is from 0 to 511 since FT800 can support screen resolution up to 512 by 512.

Note: NONE

Register Definition 22 REG_RENDERMODE Definition

REG_RENDERMODE Definition	
Reserved	R/W
31 1	0

Address: 0x102410 **Reset Value: 0x0**

Bit 0: This bit determines which mode is in use for the graphics engine to render the screen.

0 : Normal mode. The screen will be rendered frame by frame.

1 : Single Line mode. Graphics engine renders the screen with the line specified in

Note: NONE

Register Definition 23 REG_DLSWAP Definition

REG_DLSWAP Definition	
Reserved	R/W

31 2 1 0

Address: 0x102450 **Reset Value: 0x00**

Bit 0 - 1: These bits can be set by the host to validate the display list buffer of the FT800. The FT800 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 of the FT800. If the value is read as zero, the display list buffer of the FT800 is safe and ready to write. Otherwise, the host needs to wait till it becomes zero.

Note:

Register Definition 24 REG_TAG Definition

REG_TAG Definition	
Reserved	R/O
31 8	7 0

Address: 0x102478 **Reset Value: 0x0**

Bit 0 - 7 : These bits are updated with tag value by FT800 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.

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 FT800 touch engine.

Register Definition 25 REG_TAG_Y Definition

REG_TAG_Y Definition	
Reserved	R/W
31 9	8 0

Address: 0x102474 **Reset Value: 0x0**

Bit 0 - 8 : These bits are set by 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 coordinator, this register can be updated to query the tag value of respective touching point.

Note: NONE

Register Definition 26 REG_TAG_X Definition

REG_TAG_X Definition	
Reserved	R/W
31 9	8 0

Address: 0x102470 **Reset Value: 0x0**

Bit 0 - 8 : These bits are set by 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 coordinator, this register can be updated to query the tag value of the respective touching point.

Note: NONE

3.2 Touch Engine Registers

Register Definition 27 REG_TOUCH_DIRECT_Z1Z2 Definition

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

Address: 0x102578 **Reset Value: NA**

Bit 0 - 9 : The 10 bit ADC value for touch screen resistance Z2.
Bit 16-25: The 10 bit ADC value for touch screen resistance Z1.

Note: To know it is touched or not, please check the 31st bit of REG_TOUCH_DIRECT_XY. FT800 touch engine will do the post-processing for these Z1 and Z2 values and update the result in REG_TOUCH_RZ.

Register Definition 28 REG_TOUCH_DIRECT_XY

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

Address: 0x102574 **Reset Value: 0x0**

Bit 0 - 9 : The 10 bit ADC value for Y coordinate
Bit 16-25: The 10 bit ADC value for X coordinate.
Bit 31 : If this bit is zero, it means a touch is being sensed and the two fields above contains 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.

Note:

Register Definition 29 REG_TOUCH_TRANSFORM_F Definition

REG_TOUCH_TRANSFORM_F Definition		
R/W		
31 30	16 15	0

Address: 0x102530 **Reset Value: 0x0**

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

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

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

Register Definition 30 REG_TOUCH_TRANSFORM_E Definition

REG_TOUCH_TRANSFORM_E Definition			
R/W			
31	30	16 15	0

Address: 0x10252C **Reset Value: 0x10000**

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

Bit 16 - 30 : 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 31 REG_TOUCH_TRANSFORM_D Definition

REG_TOUCH_TRANSFORM_D Definition			
R/W			
31	30	16 15	0

Address: 0x102528 **Reset Value: 0x0**

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

Bit 16 - 30 : 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 32 REG_TOUCH_TRANSFORM_C Definition

REG_TOUCH_TRANSFORM_C Definition		
R/W		
31 30	16 15	0

Address: 0x102524 **Reset Value: 0x0**

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

Bit 16 - 30 : 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 33 REG_TOUCH_TRANSFORM_B Definition

REG_TOUCH_TRANSFORM_B Definition			
R/W			
31	30	16 15	0

Address: 0x102520 **Reset Value: 0x0**

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

Bit 16 - 30 : 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 34 REG_TOUCH_TRANSFORM_A Definition

REG_TOUCH_TRANSFORM_A Definition			
R/W			
31	30	16 15	0

Address: 0x10251C **Reset Value: 0x10000**

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

Bit 16 - 30 : 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 35 REG_TOUCH_TAG Definition

REG_TOUCH_TAG Definition	
RESERVED	RO

31 8 7 0

Address: 0x102518 **Reset Value: 0**

Bit 0 - 7 : 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 is scanned out to the screen.
Bit 8 - 31: 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 36 REG_TOUCH_TAG_XY Definition

REG_TOUCH_TAG_XY Definition	
RO	RO
31 16	15 0

Address: 0x102514 **Reset Value: 0**

Bit 0 - 15 : The value of these bits are the Y coordinates of the touch screen, which was used by the touch engine to look up the tag result.
Bit 16 - 31: The value of these bits are X coordinates of the touch screen, which was 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 37 REG_TOUCH_SCREEN_XY Definition

REG_TOUCH_SCREEN_XY Definition	
RO	RO
31 16	15 0

Address: 0x102510 **Reset Value: 0x80008000**

Bit 0 - 15 : 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 16 - 31: 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 of the FT800. It has been mapped into screen size.

Register Definition 38 REG_TOUCH_RZ Definition

REG_TOUCH_RZ Definition	
Reserved	RO
31 16	15 0

Address: 0x10250C **Reset Value: 0x7FFF**

Bit 0 - 15 : 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.

Note: .

Register Definition 39 REG_TOUCH_RAW_XY Definition

REG_TOUCH_RAW_XY Definition	
Reserved	RO
31 16	15 0

Address: 0x102508 **Reset Value: 0xFFFF**

Bit 0 - 15 : These bits are the raw Y coordinates of the touch screen. The valid range is from 0 to 1023. If there is no touch on screen, the value shall be 0xFFFF.
Bit 16 - 31: These bits are the raw X coordinates. 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 40 REG_TOUCH_RZTHRESH Definition

REG_TOUCH_RZTHRESH Definition	
Reserved	R/W
31 16	15 0

Address: 0x102504 **Reset Value: 0xFFFF**

Bit 0 - 15 : These bits control the touch screen resistance threshold. 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 touch engine of the FT800. The host can set this register by doing experiments. The typical value is 1200.

Note:

Register Definition 41 REG_TOUCH_OVERSAMPLE Definition

REG_TOUCH_OVERSAMPLE Definition	
Reserved	R/W
31 4	3 0

Address: 0x102500 **Reset Value: 0x7**

Bit 0 - 3 : 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.

Note: .

Register Definition 42 REG_TOUCH_SETTLE Definition

REG_TOUCH_SETTLE Definition	
Reserved	R/W
31 3	0

Address: 0x1024FC **Reset Value: 0x3**

Bit 0 - 3 : 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.

Note: .

Register Definition 43 REG_TOUCH_CHARGE Definition

REG_TOUCH_CHARGE Definition	
Reserved	R/W
31 16	15 0

Address: 0x1024F8 **Reset Value: 0x1770**

Bit 0 - 15 : 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.

Note: .

Register Definition 44 REG_TOUCH_ADC_MODE Definition

REG_TOUCH_ADC_MODE Definition	
Reserved	R/W

31 1 0

Address: 0x1024F4 **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.

Note: .

Register Definition 45 REG_TOUCH_MODE Definition

REG_TOUCH_MODE Definition	
Reserved	R/W

31 2 1 0

Address: 0x1024F0 **Reset Value: 0x3**

Bit 0 - 1 : 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.

Note: .

3.3 Audio Engine Registers

Register Definition 46 REG_PLAY Definition

31	0

Address: 0x102488 **Reset Value: 0x0**

Bit 0 : A write to this bit triggers the play of 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.

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

Register Definition 47 REG_SOUND Definition

REG_SOUND Definition	
Reserved	R/W
31 16	15 0

Address: 0x102484 **Reset Value: 0x0000**

Bit 0 - 15 : These bits are used to select the synthesized sound effect. They are split into two group Bit 0 - 7, Bit 8- 15.
Bit 0 - 7 : 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 8 - 15: 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 48 REG_VOL_SOUND Definition

REG_VOL_SOUND Definition	
Reserved	R/W

31 8 7 0

Address: 0x102480 **Reset Value: 0xFF**

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

Note:

Register Definition 49 REG_VOL_PB Definition

REG_VOL_PB Definition	
Reserved	R/W

31 8 7 0

Address: 0x10247C **Reset Value: 0xFF**

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

Note:

Register Definition 50 REG_PLAYBACK_PLAY Definition

REG_PLAYBCK_PLAY Definition	
Reserved	R/W
31 1	0

Address: 0x1024BC **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.

Note: Please refer to the datasheet section "Audio Playback" for the details of this register.

Register Definition 51 REG_PLAYBACK_LOOP Definition

REG_PLAYBACK_LOOP Definition	
Reserved	R/W

31 1 0

Address: 0x1024B8 **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.

Note: Please refer to the datasheet section "Audio Playback" for the details of this register.

Register Definition 52 REG_PLAYBACK_FORMAT Definition

Address: 0x1024B4 **Reset Value: 0x0**

Bit 0 - 1 : These bits define the format of the audio data in RAM_G. FT800 supports:

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

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

Register Definition 53 REG_PLAYBACK_FREQ Definition

REG_PLAYBACK_FREQ Definition	
Reserved	R/O

31 16 15 0

Address: 0x1024B0 **Reset Value: 0x1F40**

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

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

Register Definition 54 REG_PLAYBACK_READPTR Definition

REG_PLAYBACK_READPTR Definition	
Reserved	R/O

31 20 19 0

Address: 0x1024AC **Reset Value: 0x00000**

Bit 0 - 19 : These bits are updated by the FT800 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.

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

Register Definition 55 REG_PLAYBACK_LENGTH Definition

REG_PLAYBACK_LENGTH Definition	
Reserved	R/W
31 20	19 0

Address: 0x1024A8 **Reset Value: 0x00000**

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

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

Register Definition 56 REG_PLAYBACK_START Definition

REG_PLAYBACK_START Definition	
Reserved	R/W
31 20	19 0

Address: 0x1024A4 **Reset Value: 0x00000**

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

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

3.4 Co-processor Engine Registers

Register Definition 57 REG_CMD_DL Definition

REG_CMD_DL Definition	
Reserved	R/W

31 14 13 0

Address: 0x1024EC **Reset Value: 0x0000**

Bit 0 - 13 : 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. The coprocessor engine 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 8195.

Note: .

Register Definition 58 REG_CMD_WRITE Definition

Address: 0x1024E8 **Reset Value: 0x0**

Bit 0 - 11 : These bits are updated by the host MCU to inform the coprocessor 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.

Note: FIFO size of 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 59 REG_CMD_READ Definition

31	12 11 0

Address: 0x1024E4 **Reset Value: 0x000**

Bit 0 - 11 : 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 error, the coprocessor engine writes 0xFFF to this register.

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

Register Definition 60 REG_TRACKER Definition

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

Address: 0x109000 **Reset Value: 0x0**

Bit0 - 15: These bits are set to indicate the tag value of a graphics object which is being touched.

Bit 16 - 31: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor caculates how much the current touching points take within the predefined range. Please check the CMD_TRACK for more details.

Note: NONE

3.5 Miscellaneous Registers

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

Register Definition 61 REG_PWM_DUTY Definition

REG_PWM_DUTY Definition	
Reserved	R/W
31 8	7 0

Address: 0x1024C4 **Reset Value: 0x80**

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

Note:

Register Definition 62 REG_PWM_HZ Definition

REG_PWM_HZ Definition	
Reserved	R/W
31 14	13 0

Address: 0x1024C0 **Reset Value: 0xFA**

Bit 0 - 13 : 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.

Note:

Register Definition 63 REG_INT_MASK Definition

REG_INT_MASK Definition	
Reserved	R/W
31 8	7 0

Address: 0x1024A0 **Reset Value: 0xFF**

Bit 0 - 7 : 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 interrupt by default.

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

Register Definition 64 REG_INT_EN Definition

REG_INT_EN Definition	
Reserved	R/W
31 ... 1	0

Address: 0x10249C **Reset Value: 0x0**

Bit 0 : The host can set this bit to 1 to enable the global interrupt of FT800. To disable the global interrupt of FT800, the host can set this bit to 0.

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

Register Definition 65 REG_INT_FLAGS Definition

REG_INT_FLAGS Definition	
Reserved	R/C
31 8	7 0

Address: 0x102498 **Reset Value: 0x00**

Bit 0 - 7 : These bits are interrupt flags set by the FT800. The host can read these bits to determine which interrupt takes place. These bits are cleared automatically by reading. The host shall not write this register. After reset, there are no interrupts happen by default , therefore, it is 0x00.

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

Register Definition 66 REG_GPIO Definition

REG_GPIO Definition	
Reserved	R/W
31 8	7 0

Address: 0x102490 **Reset Value: 0x00**

Bit 0 - 7 : These bits are versatile. Bit 0 , 1, 7 are used to control GPIO pin values.
Bit 2 - 6 : These are used to configure the drive strength of the pins.

Note: Please read the datasheet section "General Purpose IO pins" for more details.

Register Definition 67 REG_GPIO_DIR Definition

REG_GPIO_DIR Definition	
Reserved	R/W
31 8	7 0

Address: 0x10248C **Reset Value: 0x80**

Bit 0 - 7 : These bits configure the direction of GPIO pins of the FT800. Bit 0 controls the direction of GPIO0 and Bit 7 controls the direction of GPIO7. The bit value 1 means the GPIO pin is set as an output, otherwise it means an input. After reset, only the GPIO7 is set to output by default.

Note:

Register Definition 68 REG_CPURESET Definition

REG_CPURESET Definition	
Reserved	RW
31 1	0

Address: 0x10241C **Reset Value: 0x00**

Bit 0: Write this bit to 1 will set the coprocessor engines of the FT800 into the reset state. Write this bit to 0 will resume from reset state to normal operational mode. If this bit is read as 1, the FT800 coprocessor engines are in reset state. Otherwise, FT800 corpocessor engines are in normal state.

Bit 1 - 31: Reserved.Read Only.
Note: NONE

Register Definition 69 REG_FREQUENCY Definition

REG_FREQUENCY Definition
Read / Write
31 0
Address: 0x10240C **Reset Value: 0x2DC6C00**
Bit0 - 31: These bits are set 0x2DC6C00 after reset, i.e. The main clock frequency is 48MHz by default. The value is in HZ. If the host selects the alternative frequency by using host command CLK36M, this register must be updated accordingly.
Note: NONE

Register Definition 70 REG_CLOCK Definition

REG_CLOCK Definition
Read Only
31 0
Address: 0x102408 **Reset Value: 0x00000000**
Bit0 - 31: These bits are set to zero after reset. The register counts the number of FT800 main clock cycles since reset. If the FT800 main clock's frequency is 48Mhz, it will wrap around after about 89 seconds.
Note: NONE

Register Definition 71 REG_FRAMES Definition

REG_FRAMES Definition	
Read Only	
31	0

Address: 0x102404 **Reset Value: 0x00000000**

Bit0 - 31: 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.

Note: NONE

Register Definition 72 REG_ID Definition

REG_ID Definition		
Reserved		RO
31	8 7	0

Address: 0x102400 **Reset Value: 0x7C**

Bit0 - 7: These bits are the built-in register ID. The host can read it to determine if the chip is FT800. The value shall always be 0x7C.

Note: NONE

4 Display list commands

The graphics engine of FT800 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 up to 2048 commands since the size of RAM_DL is 8K bytes. The graphics engine of the FT800 performs respective operation according to the definition of commands.

4.1 Graphics State

The graphics state which controls drawing is called the graphics context. Individual pieces of state can be changed by the appropriate display list commands (e.g. COLOR_RGB) and the entire 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 4 FT800 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	512,512	SCISSOR_SIZE
Starting coordinates of scissor	(x, y) = (0,0)	SCISSOR_XY
Current bitmap handle	0	BITMAP_HANDLE
Bitmap transform	+1.0,0,0,0,+1.0,0	BITMAP_TRANSFORM_A-F

Parameters	Default values	Commands
coefficients		
Stencil clear value	0	CLEAR_STENCIL
Tag clear value	0	CLEAR_TAG
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

Each display list command in this section lists any graphics context it sets.

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 reserved must be zero.

The graphics primitives supported by FT800 and their respective values are mentioned below

Table 5 FT800 Graphics Primitives list

Graphics Primitive	Primitive value
BITMAPS	1
POINTS	2
LINES	3
LINE_STRIP	4
EDGE_STRIP_R	5
EDGE_STRIP_L	6
EDGE_STRIP_A	7
EDGE_STRIP_B	8
RECTS	9

Various bitmap formats supported by FT800 and their respective values are mentioned below

Table 6 Graphics Bitmap Format table

Bitmap format	Bitmap format value
ARGB1555	0
L1	1
L4	2
L8	3
RGB332	4
ARGB2	5
ARGB4	6
RGB565	7
PALETTED	8
TEXT8X8	9
TEXTVGA	10
BARGRAPH	11

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	set the source bitmap memory format and layout for the current handle
BITMAP_SIZE	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
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

4.3.2 Drawing actions

BEGIN	begin drawing a graphics primitive
END	end drawing a graphics primitive
VERTEX2F	supply a vertex with fractional coordinates
VERTEX2II	supply a vertex with integer coordinates

4.3.3 Execution control

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 8: 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 7 FT800 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 FT800 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

Handles 16 to 31 are defined by the FT800 for built-in font and handle 15 is defined in the co-processor engine commands CMD_GRADIENT, CMD_BUTTON and CMD_KEYS. Users can define new bitmaps using handles from 0 to 14. If there is no co-processor engine command CMD_GRADIENT, CMD_BUTTON and CMD_KEYS in the current display list, users can even define a bitmap using handle 15.

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

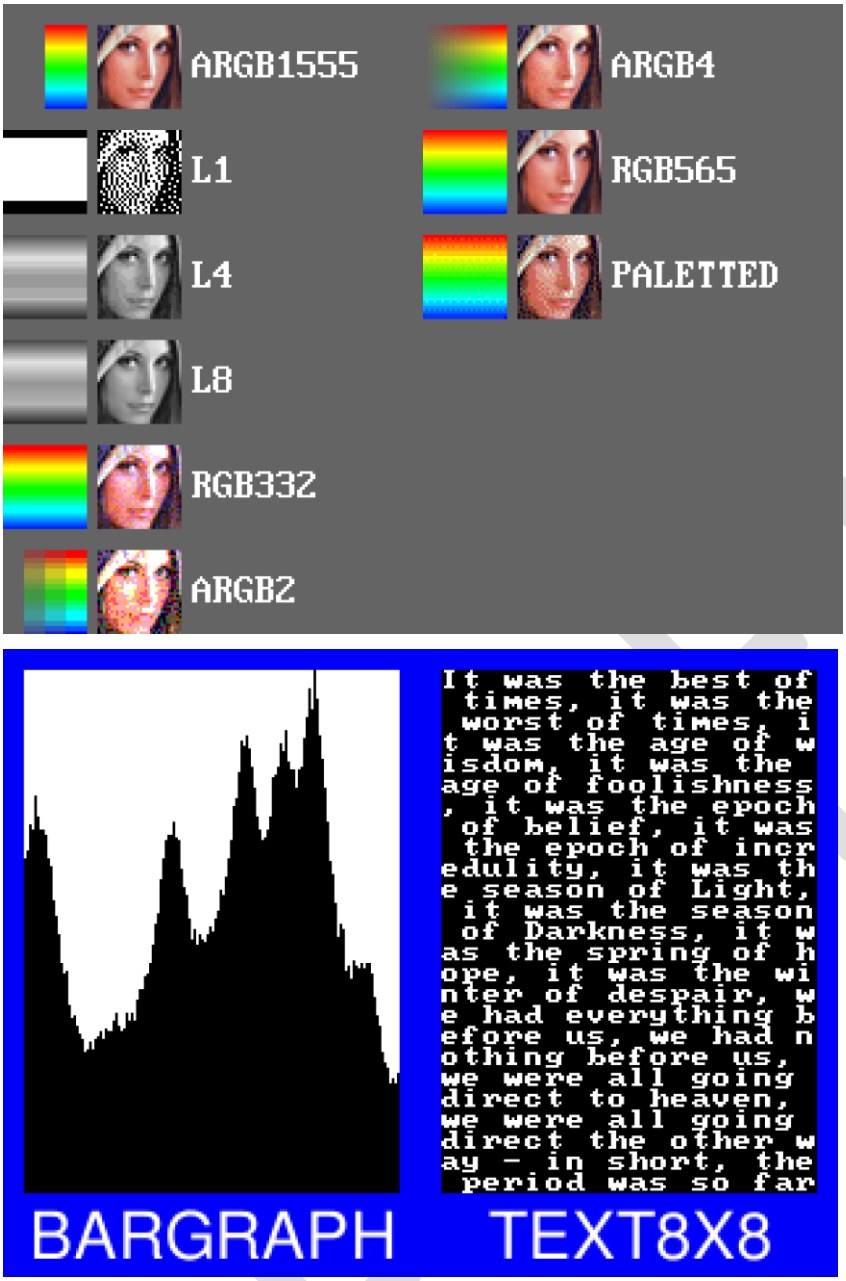
format

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

Table 8 BITMAP_LAYOUT format list

NAME	VALUE
ARGB1555	0
L1	1
L4	2
L8	3
RGB332	4
ARGB2	5
ARGB4	6
RGB565	7
PALETTED	8
TEXT8X8	9
TEXTVGA	10
BARGRAPH	11

Various bitmap formats supported are:



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 8x8 font with TEXTVGA syntax. The bitmap is a TEXTVGA array present in the graphics ram, each element indexes into an internal 8x8 CP437 [2] font (inbuilt font bitmap handles 16 & 17 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 8x8 pixel character cell.

PALETTED - bitmap bytes are indices into a palette table. By using a palette table - which contains 32-bit RGBA colors - a significant amount of memory can be saved. The 256 color palette is stored in a dedicated 1K (256x4) byte RAM_PAL.

linestride

Bitmap linestride, in bytes. Please note the alignment requirement which is described below.

height

Bitmap height, in lines

Description

The bitmap formats supported are L1, L4, L8, RGB332, ARGB2, ARGB4, ARGB1555, RGB565 and Palette.

For L1 format, the line stride must be a multiple of 8 bits; For L4 format the line stride must be multiple of 2 nibbles. (Aligned to byte)

For more details about alignment, please refer to the figures below:

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

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

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

Figure 9: Pixel representation in memory for bitmap format L1/L4/L8

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

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

Figure 10: Pixel representation in memory for Bitmap format ARGB2/1555

ARGB4 format layout		Byte Order
A	Bit 15-12	Byte 1
R	Bit 11-8	
G	Bit 7-4	Byte 0
B	Bit 3-0	

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

RGB565 format layout		Byte Order
R	Bit 15-11	Byte 1
G	Bit 10-5	
B	Bit 4-0	Byte 0

Palette format layout		Byte Order
A	Bit 7-0	Byte 3
R	Bit 15-8	Byte 2
G	Bit 23-16	Byte 1
B	Bit 31-24	Byte 0

Figure 11: Pixel representation for Bitmap Format ARGB4, RGB332, RGB565 and Palette

Graphics context

None

See also

BITMAP_HANDLE, BITMAP_SIZE, BITMAP_SOURCE

4.8 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

width

Drawn bitmap width, in pixels

height

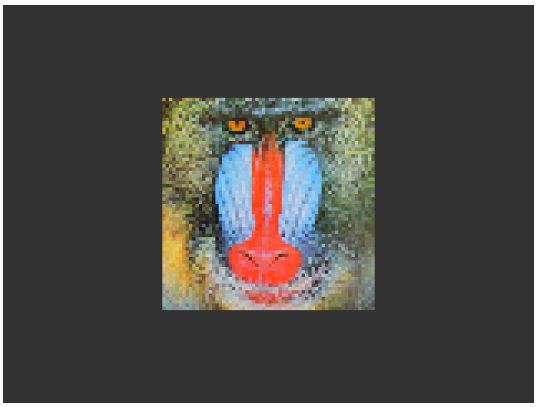
Drawn bitmap height, in pixels

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.

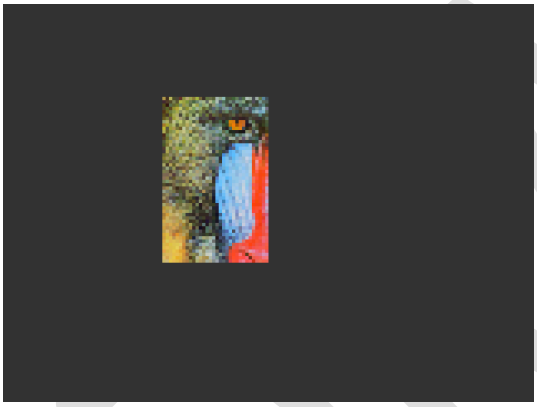
Examples

Drawing a 64 x 64 bitmap:



```
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) );
```

Reducing the size to 32 x 50:

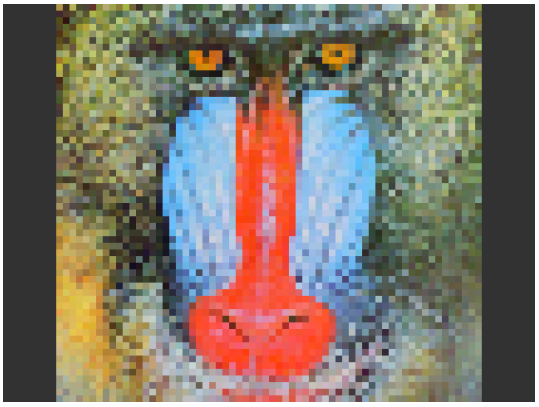


```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_SIZE(NEAREST, BORDER, BORDER, 32, 50) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(48, 28, 0, 0) );
```

Using the REPEAT wrap mode to tile the bitmap:

```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT(RGB565, 128, 64) );
dl( BITMAP_SIZE(NEAREST, REPEAT, REPEAT, 160, 120) );
dl( BEGIN(BITMAPS) );
dl( VERTEX2II(0, 0, 0, 0) );
```

4X zoom - 128 X 128 - using a bitmap transform:



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

Using a bilinear filter makes the zoomed image a little smoother:

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

Graphics context

None

See also

BITMAP_HANDLE, BITMAP_LAYOUT, BITMAP_SOURCE

4.9 BITMAP_SOURCE

Specify the source address of bitmap data in FT800 graphics memory RAM_G.

Encoding

31 24	23 20	19 0
0x01	Reserved	addr

Parameters

addr

Bitmap address in graphics SRAM FT800, aligned with respect to the bitmap format.

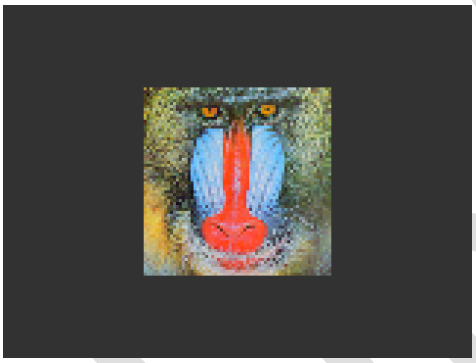
For example, if the bitmap format is RGB565/ARGB4/ARGB1555, the bitmap source shall be aligned to 2 bytes.

Description

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

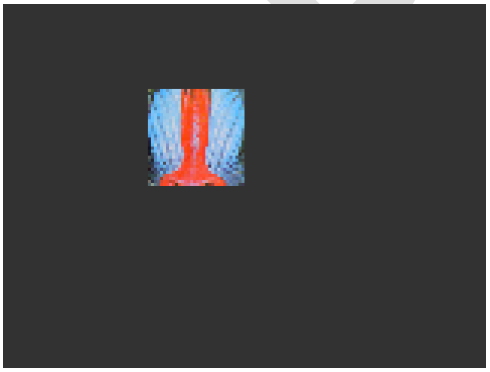
Examples

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) );
```

Graphics context

None

See also

BITMAP_LAYOUT, BITMAP_SIZE

4.10 BITMAP_TRANSFORM_A

Specify the A coefficient of the bitmap transform matrix.

Encoding

31 24	23 17	16 0
0x15	Reserved	a

Parameters

a

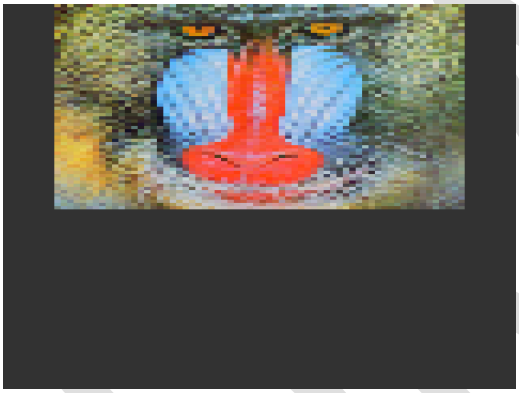
Coefficient A of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 256.

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.

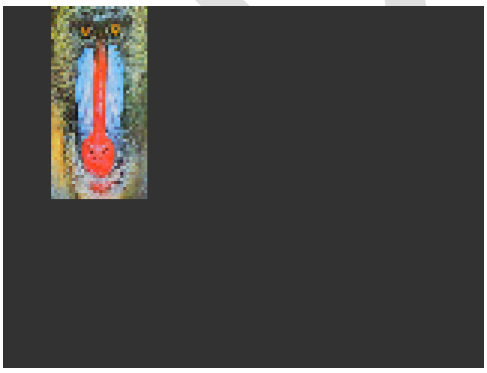
Examples

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

4.11 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

4.12 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

4.13 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

4.14 BITMAP_TRANSFORM_E

Specify the E coefficient of the bitmap transform matrix

Encoding

31 24	23 17	16 0
0x19	Reserved	e

Parameters

e

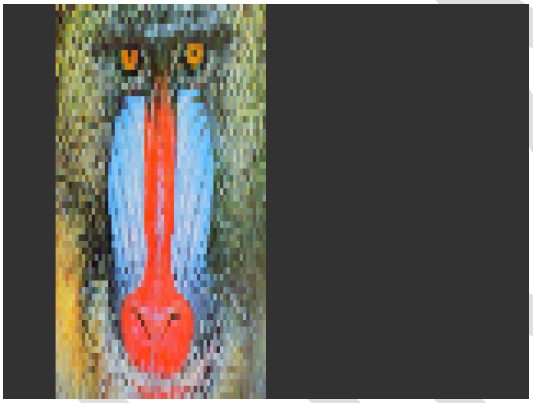
Coefficient E of the bitmap transform matrix, in signed 8.8 bit fixed-point form. The initial value is 256

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.

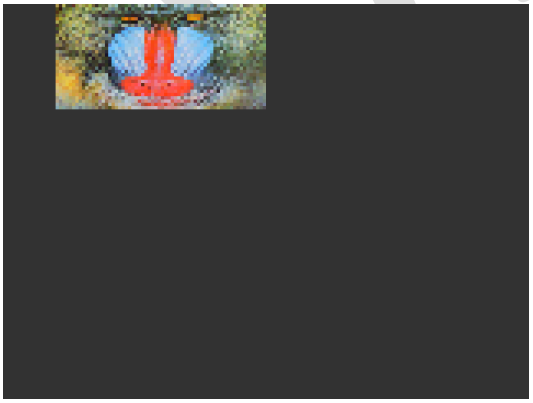
Examples

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

4.15 BITMAP_TRANSFORM_F

Specify the F coefficient of the bitmap transform matrix

Encoding

31 24	23 0
0x1F	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

4.16 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 9 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

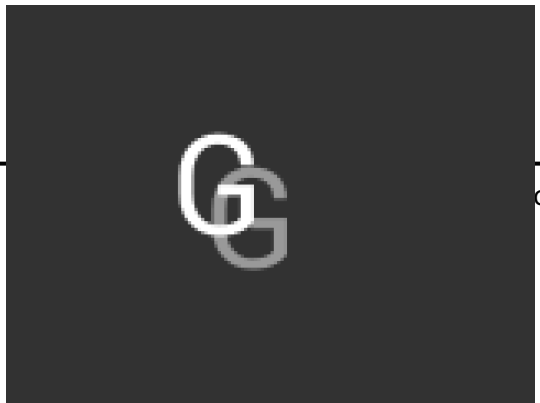
The blend function controls how new color values are combined with the values already in the color buffer. Given a pixel value source and a previous value in the color buffer destination, the computed color is:

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

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

4.17 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 destination address in RAM_DL which the display command is to be switched. FT800 has the stack to store the return address. To come back to the next command of source address, the RETURN command can help.

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

4.18 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

4.19 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 FT800 to the preset value. Setting this bit to 0 will maintain the color buffer of the FT800 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 FT800 to the preset value. Setting this bit to 0 will maintain the stencil buffer of the FT800 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 FT800 to the preset value. Setting this bit to 0 will maintain the tag buffer of the FT800 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

4.20 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

4.21 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

4.22 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

4.23 CLEAR_TAG

Specify clear value for the tag buffer

Encoding

31 24	23 8	7 0
0x06	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

4.24 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

4.25 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 FT800 color buffer. The initial value is 1 and means enable.

g

Enable or disable the green channel update of the FT800 color buffer. The initial value is 1 and means enable.

b

Enable or disable the blue channel update of the FT800 color buffer. The initial value is 1 and means enable.

a

Enable or disable the alpha channel update of the FT800 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 a '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

4.26 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 red, green and blue values of the FT800 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

4.27 DISPLAY

End the display list. FT800 will ignore all the commands following this command.

Encoding

31 24	23 0
0x0	Reserved

Parameters

None

Graphics context

None

See also

None

4.28 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. Whereas advanced users can avoid the usage of END in order to save extra graphics instructions in the display list RAM.

Graphics context

None

See also

BEGIN

4.29 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 to be jumped.

Graphics context

None

See also

CALL

4.30 LINE_WIDTH

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

Encoding

31 24	23 12	11 0
0x06	Reserved	width

Parameters

width

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

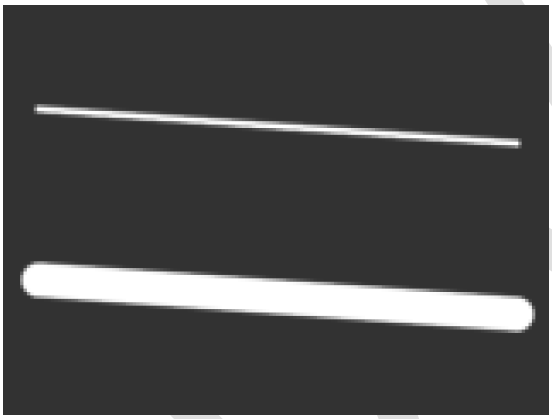
Description

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 16 to 4095 in terms of 1/16th pixel units.

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

4.31 MACRO

Execute a single command from a macro register.

Encoding

31 24	23 1	0
0x25	Reserved	m

Parameters

m

Macro register to read. Value 0 means the FT800 will fetch the command from REG_MACRO_0 to execute. Value 1 means the FT800 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

4.32 POINT_SIZE

Specify the radius of points

Encoding

31 24	23 17	16 0
0x0D	Reserved	Size

Parameters

size

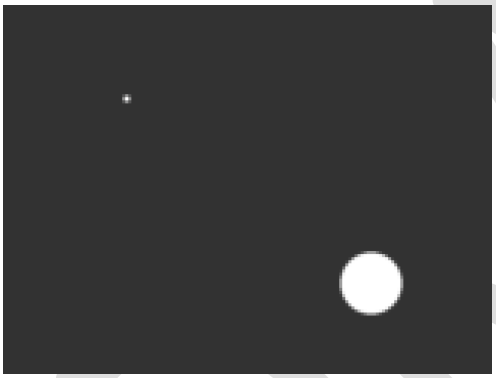
Point radius in 1/16 pixel. The initial value is 16.

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. The valid range is from 16 to 8191 with respect to 1/16th pixel unit.

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

4.33 RESTORE_CONTEXT

Restore the current graphics context from the context stack

Encoding

31 24	23 0
0x23	Reserved

Parameters

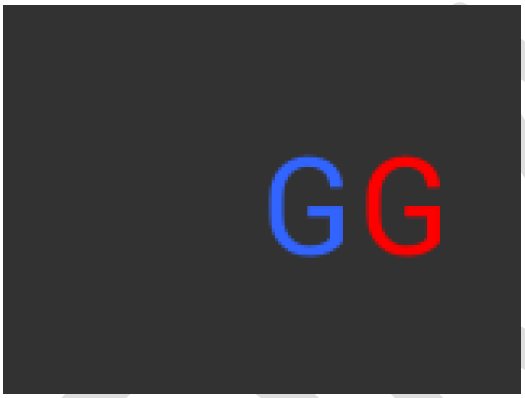
None

Description

Restores the current graphics context, as described in section 4.1. Four (4) levels of SAVE and RESTORE are available in the FT800. 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

4.34 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

4.35 SAVE CONTEXT

Push the current graphics context on the context stack

Encoding

31 24	23 0
0x22	Reserved

Parameters

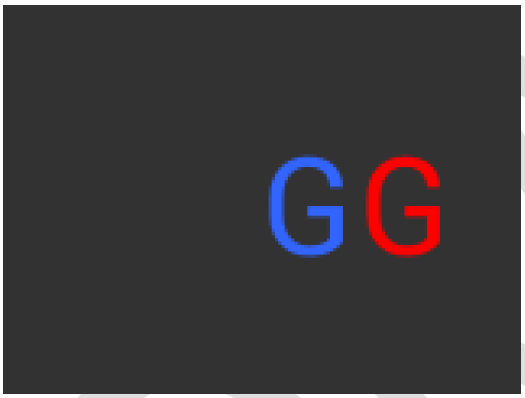
None

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

4.36 SCISSOR_SIZE

Specify the size of the scissor clip rectangle

Encoding

31 24	23 20	19 10	9 0
0x1C	Reserved	Width	Height

Parameters

width

The width of the scissor clip rectangle, in pixels. The initial value is 512.

The valid value range is from 0 to 512.

height

The height of the scissor clip rectangle, in pixels. The initial value is 512.

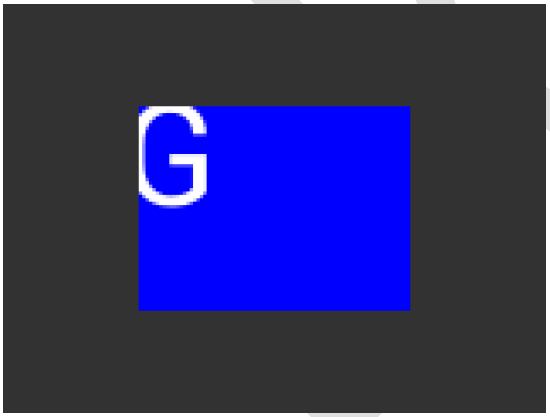
The valid value range is from 0 to 512.

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

4.37 SCISSOR_XY

Specify the top left corner of the scissor clip rectangle

Encoding

31 24	23 19	17 9	8 0
0x1B	Reserved	x	y

Parameters

x

The x coordinate of the scissor clip rectangle, in pixels. The initial value is 0

y

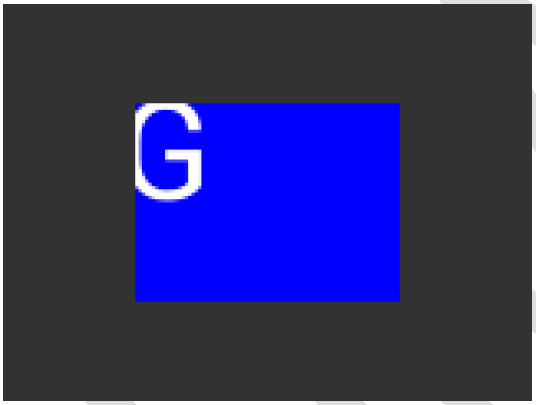
The y coordinate of the scissor clip rectangle, in pixels. The initial value is 0

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

4.38 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 8: The constants of ALPHA_FUNC

ref

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

mask

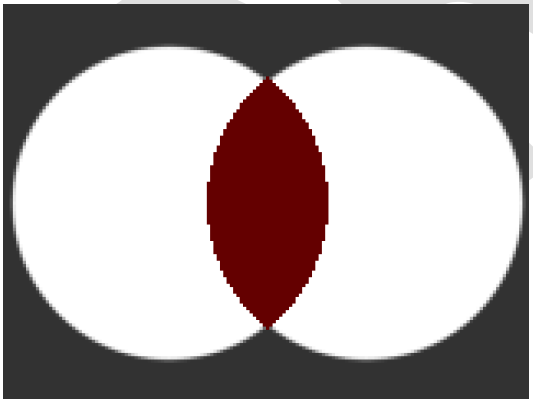
Specifies a mask that is ANDed with the reference value and the stored stencil value. The initial value is 255

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

4.39 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

4.40 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

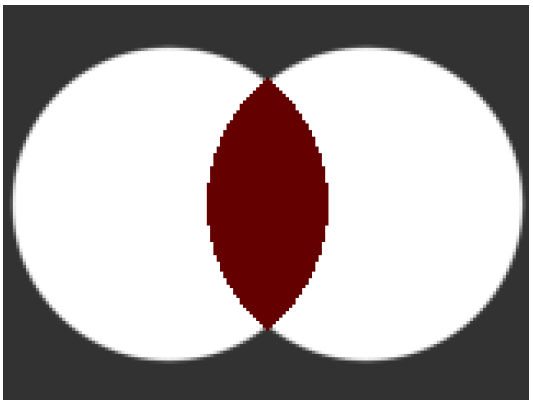
Figure 12: STENCIL_OP constants definition

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

4.41 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 FT800 is specified by command CLEAR_TAG and taken effect by command CLEAR. TAG command can specify the value of the tag buffer of the FT800 that applies to the graphics objects when they are drawn on the screen. Please note that 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 is no TAG commands in one display list, all the graphics objects rendered by the display list will report tag value as 255 in REG_TOUCH_TAG when they were 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

4.42 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 FT800 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 FT800 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 FT800 tag buffer. The FT800 tag buffer can be updated by TAG command.

The default value of the FT800 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 FT800 tag buffer takes the value from the default value of the FT800 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

4.43 VERTEX2F

Start the operation of graphics primitives at the specified screen coordinate, in 1/16th pixel precision.

Encoding

31 30	29 15	14 0
0x1	X	Y

Parameters

x

Signed x-coordinate in 1/16 pixel precision

y

Signed y-coordinate in 1/16 pixel precision

Description

The range of coordinates can be from -16384 to +16383 in terms of 1/16th pixel units. Please note the negative x coordinate value means the coordinate in the left virtual screen from (0, 0), while the negative y coordinate value means the coordinate in the upper virtual screen from (0, 0). If drawing on the negative coordinate position, the drawing operation will not be visible.

Graphics context

None

See also

None

4.44 VERTEX2II

Start the operation of graphics primitive at the specified coordinates. The handle and cell parameters will be ignored unless the graphics primitive is specified as bitmap by command BEGIN, prior to this command.

Encoding

31 30	29 21	20 12	11 7	6 0
0x02	X	Y	handle	cell

Parameters

x

x-coordinate in pixels

y

y-coordinate in pixels

handle

Bitmap handle. The valid range is from 0 to 31. From 16 to 31, the bitmap handle is dedicated to the FT800 built-in font.

cell

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

Graphics context

None

See also

None

5 Co-Processor Engine commands

CMD_TEXT - draw text

CMD_BUTTON - draw a button

CMD_CLOCK - draw an analog clock

CMD_GAUGE - draw a gauge

CMD_GRADIENT - draw a smooth color gradient

CMD_KEYS - draw a row of key

CMD_PROGRESS - draw a progress bar

CMD_SLIDER - draw a slider

CMD_SCROLLBAR - draw a scroll bar

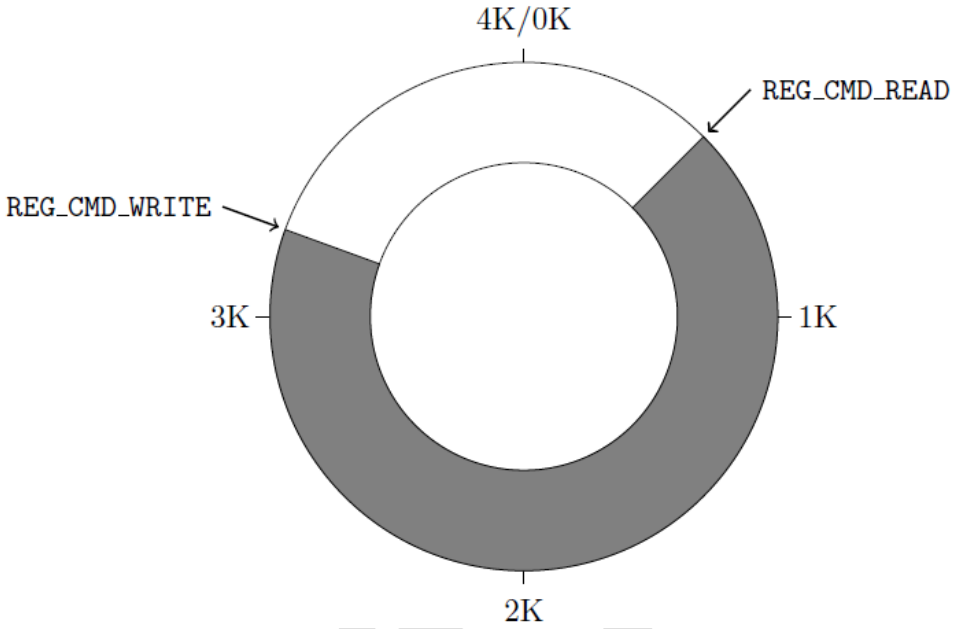
CMD_DIAL - draw a rotary dial control

CMD_TOGGLE - draw a toggle switch

CMD_NUMBER - draw a decimal number

The co-processor engine is fed via a 4 Kbyte FIFO in FT800 memory at RAM_CMD. The MCU writes commands into the FIFO, and the co-processor engine reads and executes

the commands. The MCU updates register REG_CMD_WRITE to indicate that there are new commands in the FIFO, and the co-processor engine updates REG_CMD_READ after commands have been executed.



So to compute the available free space in the FIFO, the MCU can compute:

fullness = (REG_CMD_WRITE -REG_CMD_READ) *mod* 4096

freespace = (4096 - 4) -*fullness*;

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

If enough space is available in the FIFO, the MCU writes the commands at the appropriate location in the FIFO RAM, then updates REG_CMD_WRITE. To simplify the MCU code, the FT800 hardware automatically wraps continuous writes from (RAM_CMD + 4095) back to (RAM_CMD + 0).

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 Co-processor handling of Display list commands

Most co-processor engine commands write to the current display list. The current write location in the display list is held in REG_CMD_DL. Whenever the co-processor engine writes a word to the display list, it does so at REG_CMD_DL then 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 be written as co-processor engine commands. The co-processor engine copies these commands into the current display list at REG_CMD_DL. For example, this series of co-processor engine commands 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!");
cmd(DISPLAY());                            // display
```

5.2 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 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.

5.3 ROM and RAM Fonts

The graphics engine hardware draws bitmap graphics, and it is useful for software to treat these graphics as fonts.

Font metrics - e.g. character height and width - are used by software when placing font characters. For the ROM character bitmaps, these font metrics are in ROM. The co-processor engine uses these metrics when drawing text in any of the 16 built-in ROM

fonts, numbered 16-31. Users can load similar font metrics into RAM, and hence create additional user-defined fonts in bitmap handles 0-14. Bitmap handle 15 is reserved for co-processor command CMD_Button/CMD_Keys/CMD_Gradient.

Each 148-byte font metric block has this format:

Table 10 FT800 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, 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 the ROM fonts, these blocks are also in ROM, in an array of length 16. The address of this array is held in ROM location 0xffffc. For example to find the width of character 'g' (ASCII 0x67) in font 31:

read 32-bit pointer p from 0xffffc

widths = p + (148 * (31 - 16)) (table starts at font 16)

read byte from memory at widths[0x67]

For the built-in ROM font of the FT800, the valid character range for one bitmap handle is printable ASCII code, i.e., from 32 to 127, both inclusive. For custom RAM font, the ASCII code range of valid characters is from 1 to 127.

To use a custom font in the user-interface objects:

- Select a bitmap handle from 0 to 14
- Load the font bitmap into memory
- Set the bitmap parameters using commands BITMAP_SOURCE, BITMAP_LAYOUT and BITMAP_SIZE.
- Create and download a font metric block in RAM. The address of metric block shall be **4 bytes aligned**.
- Use command CMD_SETFONT to register the new font with the selected handle.
- Use the selected handle in any co-processor command font argument.

5.4 Co-processor engine Limitations

For some of the widgets, if the input parameter values are more than 512 pixel resolution, the generated widgets may not be proper.

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 display region.

Only signed and unsigned integers are supported in CMD_NUMBER (fractional part is not supported).

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

5.5 Co-processor engine faults

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

5.6 Co-processor engine widgets physical dimensions

This section contains the common physical dimensions of the widgets.

- All rounded corners have a radius that is computed from the font used for the widget (curvature of lowercase 'o' character). The radius is computed as 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 widget will be a vertical widget in case width and height are of same value.

5.7 Co-processor engine widgets color setup

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

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

Table 11 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

5.8 Co-processor engine 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_TRACKER
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, etc.
Bitmap Handle	15	CMD_GRADCOLOR, CMD_KEYS, CMD_BUTTON

5.9 Co-processor engine resources

The co-processor engine object commands do not change hardware graphics state. That is, graphics states such as color and line width are not affected by any graphics object.

However, the graphics objects do reserve some hardware resources, which user programs may need to take into account:

- Bitmap handle 15 is used by the 3D-effect buttons, keys and gradient.
- One graphics context is used by objects, so 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 the 3D effects for CMD_BUTTON and CMD_KEYS highlight color
- 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 memory with a byte value
- CMD_MEMWRITE - write bytes into memory
- CMD_MEMCPY - copy a block of memory
- CMD_APPEND - append memory to display list

Commands for loading image data into FT800 memory:

- CMD_INFLATE - decompress data into memory
- CMD_LOADIMAGE - load a JPEG 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_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_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, 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 current display list is scanned out. Internally, the co-processor engine implements this command by writing to REG_DLSWAP. Please see 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 co-processor engine to reset default 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

Delay before interrupt triggers, in milliseconds. The interrupt is guaranteed not to fire before this delay. If ms is 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 memory to display list

Appends a block of memory 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

Start of source commands in main memory

num

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

Command layout

+0	CMD_APPEND(0xffffff02)
+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 register to 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 FT800 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 FT800 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 the FT800 memory, 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 immediate 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 image

Decompress the following JPEG image data into an FT800 bitmap, in main memory. The image data should be a regular baseline JPEG (JFIF) image.

C prototype

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

Parameters

ptr

Destination address

options

By default, option OPT_RGB565 means the loaded bitmap is in RGB565 format. Option OPT_MONO means the loaded bitmap to be monochrome, in L8 format. 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. OPT_NODL can be OR'ed with OPT_MONO or OPT_RGB565.

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 application on the host processor has to parse the JPEG header to get the properties of the JPEG 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)
+4	Ptr
+8	Options
+12	Byte0
+13	Byte1
..	..
+n	..

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_MEMCRC - compute a CRC-32 for memory

Computes a CRC-32 for a block of FT800 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 FT800 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));
```

5.21 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);

5.22 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);
```

5.23 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

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);
```

5.24 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 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. For built-in ROM font of FT800, the valid character inside of s is printable ASCII code, i.e., from 32 to 127, both inclusive. For custom RAM font, the ASCII code of valid character inside of s is from 1 to 127.

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

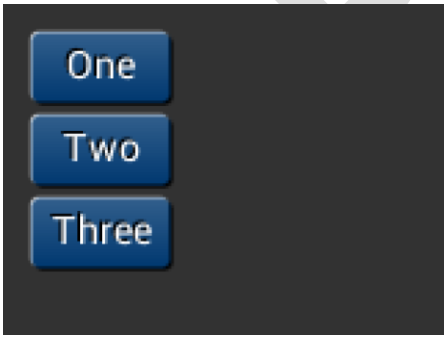
A 140x00 pixel button with large text:


```
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");
```

5.25 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 physical dimension 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);

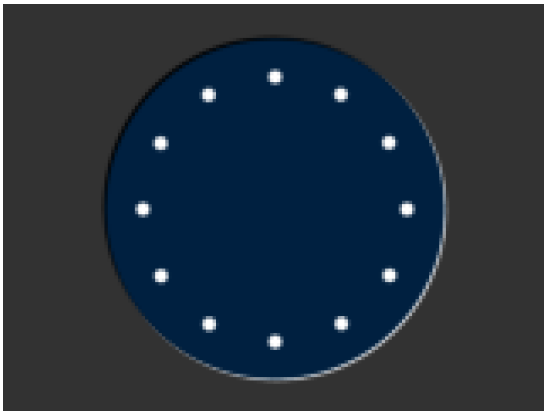
No background:

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

No ticks:

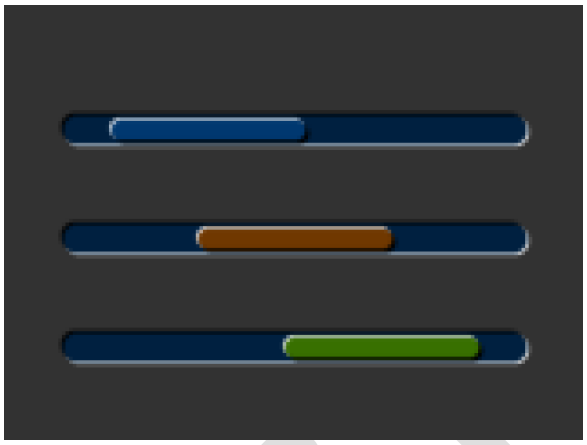
cmd_clock(80, 60, 50, OPT_NOTICKS, 8, 15, 0, 0);

No hands:



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

5.26 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

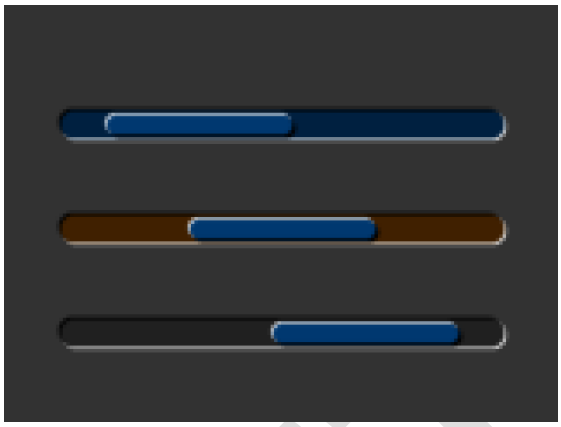
+0	CMD_FGCOLOR(0xffffff0a)
+4	C

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);
```

5.27 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.

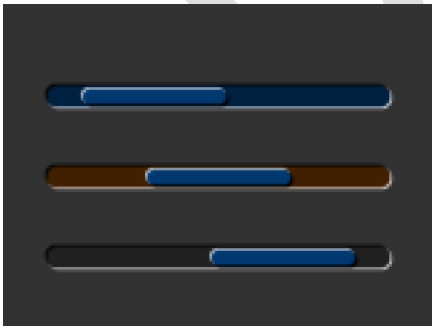
Background color is applicable for things that the user cannot move. Example behind gauges and sliders etc.

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);
```

5.28 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");
```

5.29 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, indicating 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);
```

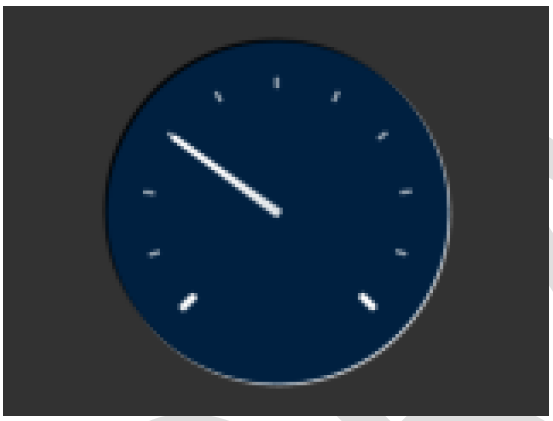
Ten major divisions with two minor divisions each:

```
cmd_gauge(80, 60, 50, 0, 10, 2, 30, 100);
```

Setting the minor divisions to 1 makes them disappear:

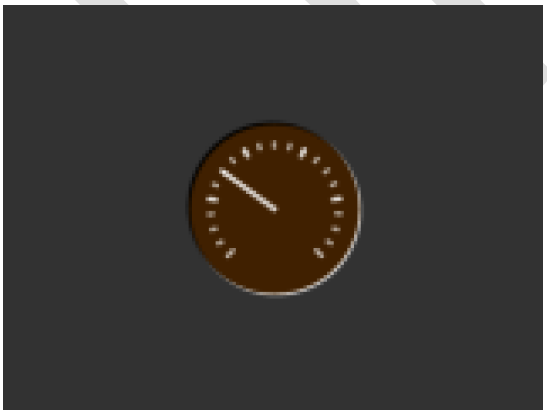
```
cmd_gauge(80, 60, 50, 0, 10, 1, 30, 100);
```

Setting the major divisions to 1 gives minor divisions 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);
```

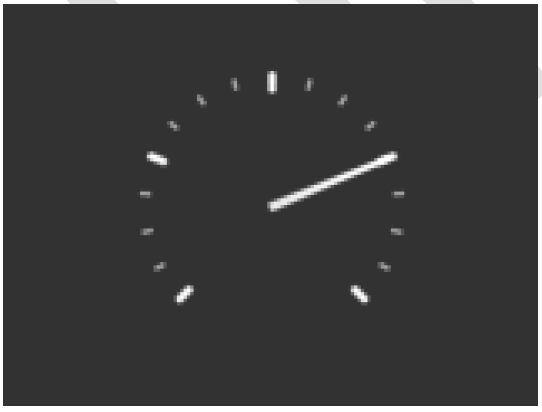
Scale 0-1000, indicating 1000:

```
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:

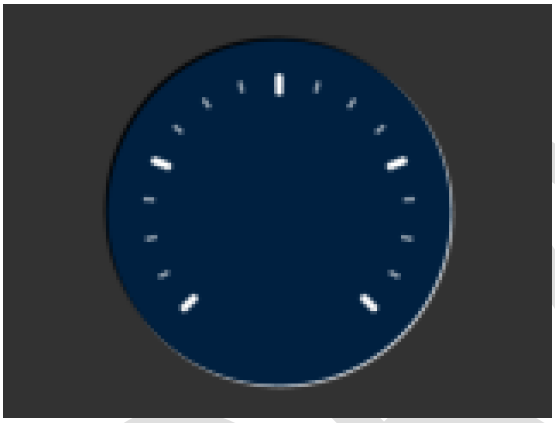


```
cmd_gauge(80, 60, 50, OPT_NOBACK, 4, 4, 49152, 65535);
```

No tick marks:

```
cmd_gauge(80, 60, 50, OPT_NOTICKS, 4, 4, 49152, 65535);
```

No pointer:

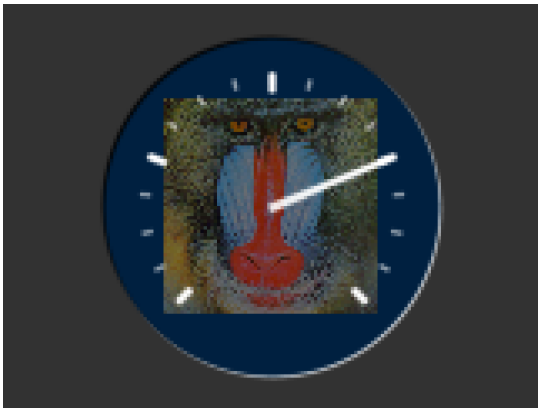


```
cmd_gauge(80, 60, 50, OPT_NOPOINTER, 4, 4, 49152, 65535);
```

Drawing the gauge in two passes, with bright red for the pointer:

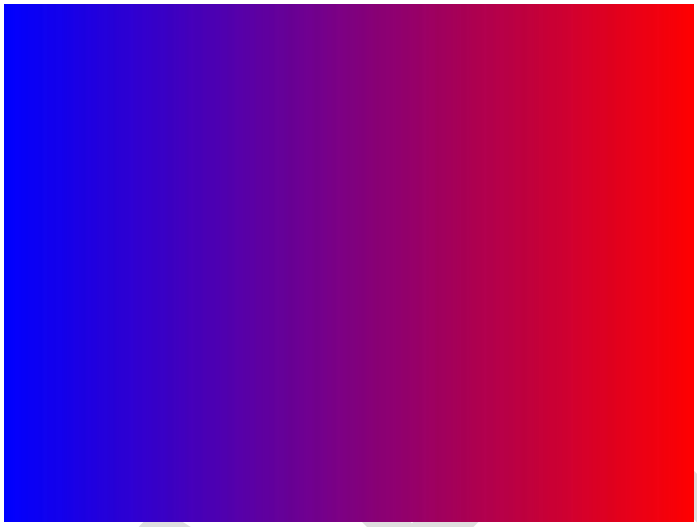
```
GAUGE_0 = OPT_NOPOINTER;
GAUGE_1 = OPT_NOBACK | OPT_NOTICKS;
cmd_gauge(80, 60, 50, GAUGE_0, 4, 4, 49152, 65535);
cmd(COLOR_RGB(255, 0, 0));
cmd_gauge(80, 60, 50, GAUGE_1, 4, 4, 49152, 65535);
```

Add a custom graphic to the gauge by drawing its background, a bitmap, 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);
```

5.30 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

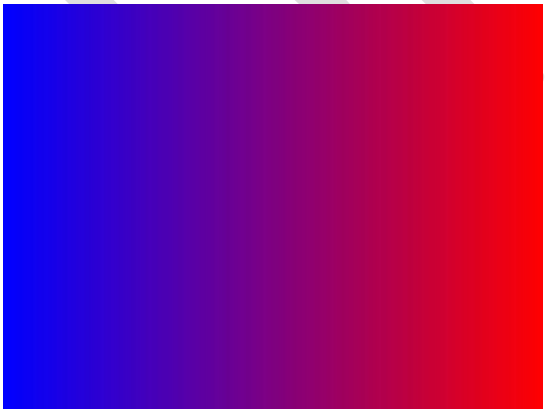
All the color's step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is R0 + t * (R1 - R0), where t is interpolated between 0 and 1. Gradient must be used with Scissor 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);
```

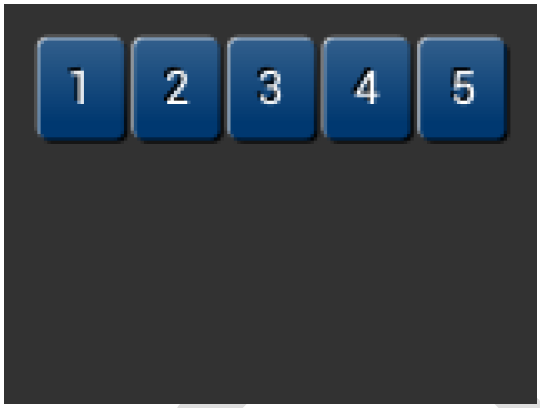
The same colors in a diagonal gradient

```
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");
```

5.31 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

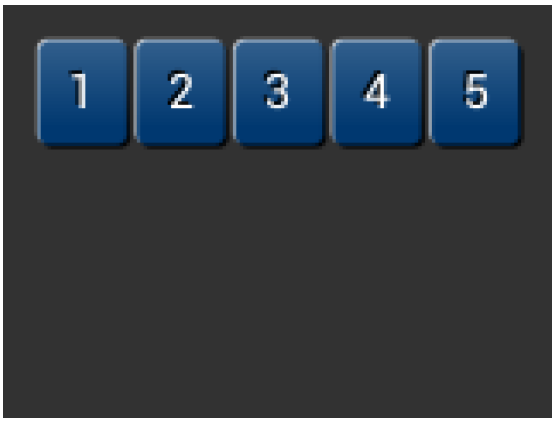
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

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

Examples

A row of keys:

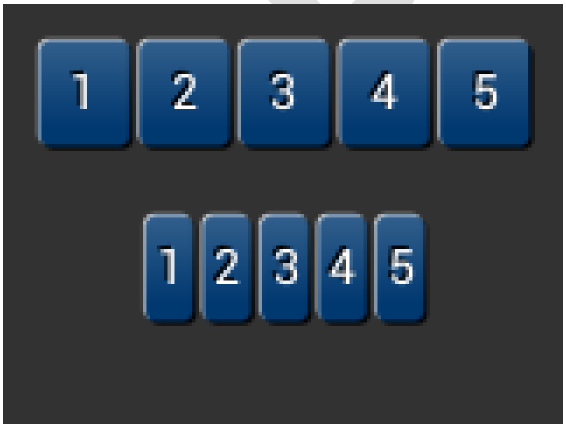



```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");
```

Without the 3D look:

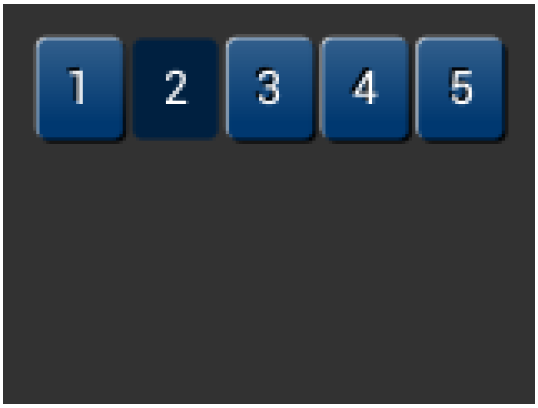

```
cmd_keys(10, 10, 140, 30, 26, OPT_FLAT, "12345");
```

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):

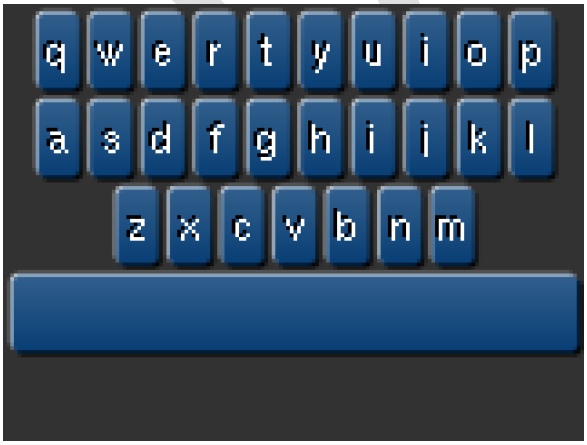



```
cmd_keys(10, 10, 140, 30, 26, 0x32, "12345");
```

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.");
```

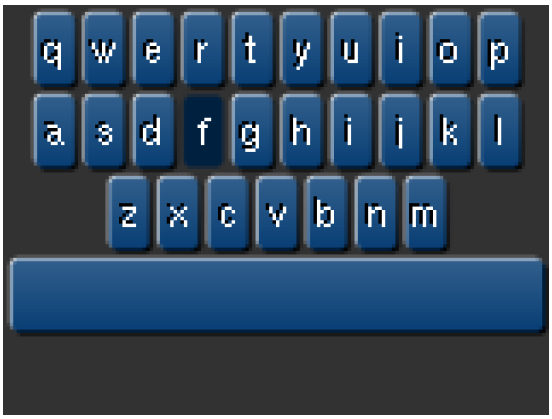
A compact keyboard drawn in font 20:




```
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, "");
```

5.32 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 removes 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(0xffffffof)
+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);
```

5.33 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 removes 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

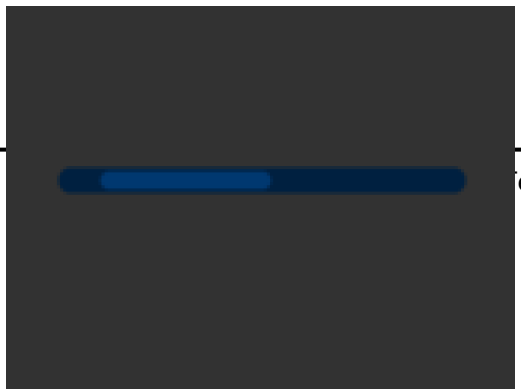
+0	CMD_SCROLLBAR(0xffffff11)
+4	X
+6	Y
+8	W
+10	H
+12	options
+14	val
+16	Size
+18	Range

Examples

A scroll bar indicating 10-50%:

```
cmd_scrollbar(20, 50, 120, 8, 0, 10, 40, 100);
```

Without the 3D look:

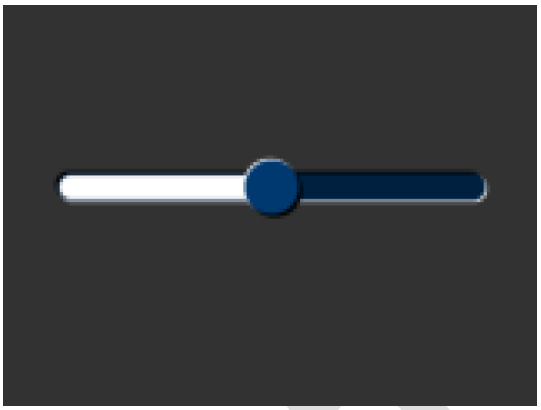


```
cmd_scrollbar(20, 50, 120, 8, OPT_FLAT,
10, 40, 100);
```

A brown-themed vertical scroll bar:

```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_scrollbar(140, 10, 8, 100, 0, 10, 40,
100);
```

5.34 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

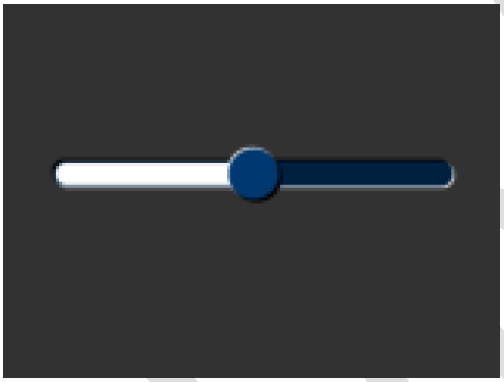
Refer to CMD_PROGRESS for more information on physical Dimension.

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%:



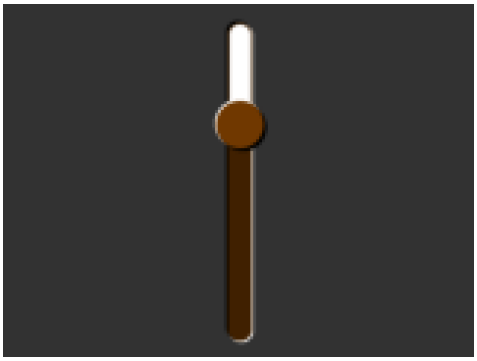
cmd_slider(20, 50, 120, 8, 0, 50, 100);

Without the 3D look:

cmd_slider(20, 50, 120, 8, OPT_FLAT, 50, 100);

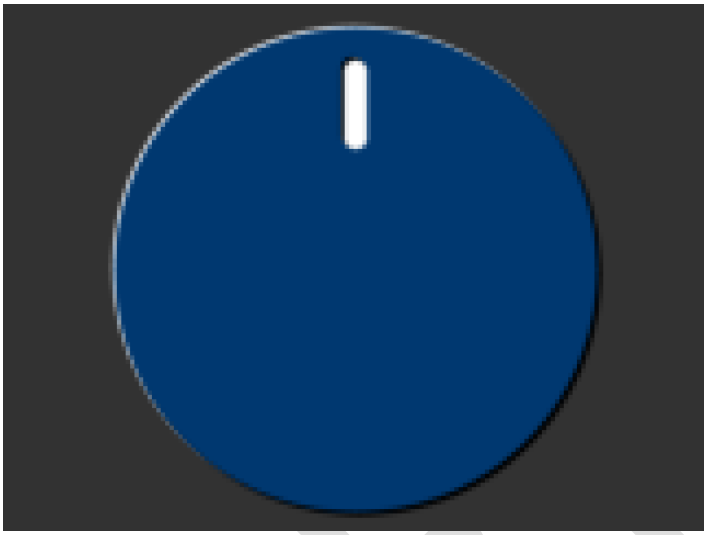
A brown-themed vertical slider with range 0-

65535:



```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_slider(76, 10, 8, 100, 0, 20000, 65535);
```

5.35 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 drawnwith a 3D effect and the value of options is zero. Options OPT_FLAT removes 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

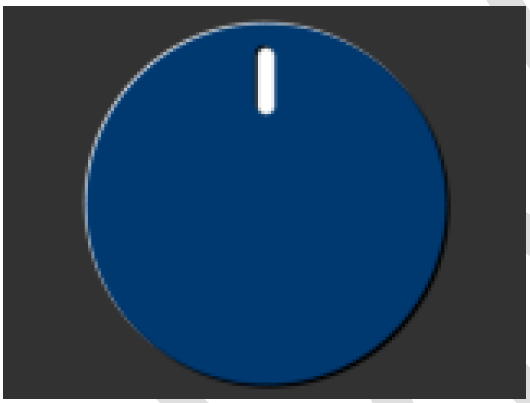
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_DIAL(0xffffff2d)
+4	X
+6	Y
+8	r
+10	options
+12	val

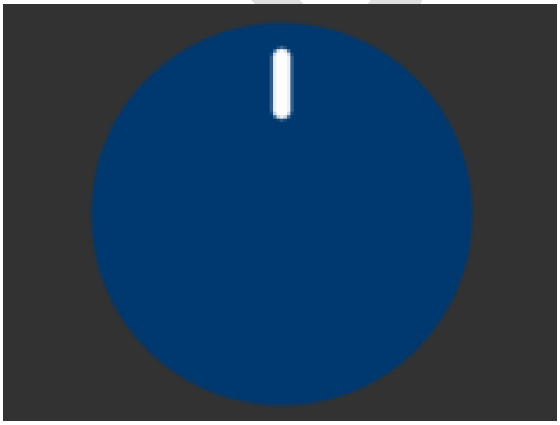
Examples

A dial set to 50%:



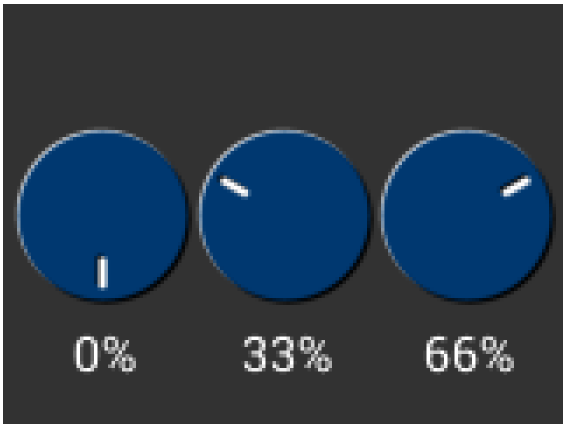
cmd_dial(80, 60, 55, 0, 0x8000);

Without the 3D look:



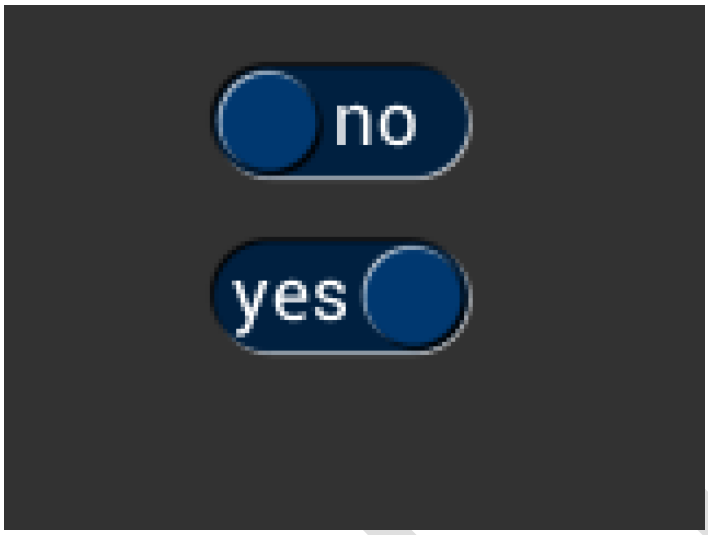
cmd_dial(80, 60, 55, OPT_FLAT, 0x8000);

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%");
```

5.36 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 removes 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

- Outer bar radius (r) is font height*(20/16)
- Knob radius is r-1.5

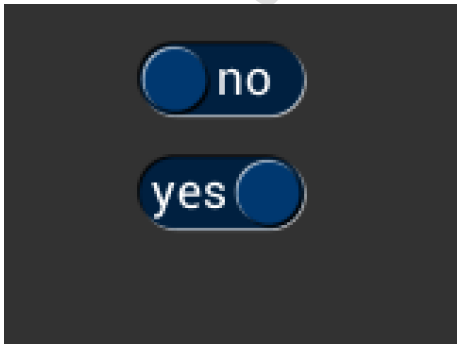
Refer to Co-processor engine widgets physical dimensions for more information.

Command layout

+0	CMD_TOGGLE(0xffffff12)
+4	X
+6	Y
+8	W
+10	Font
+12	Options
+14	State
+16	S
..	..
..	0

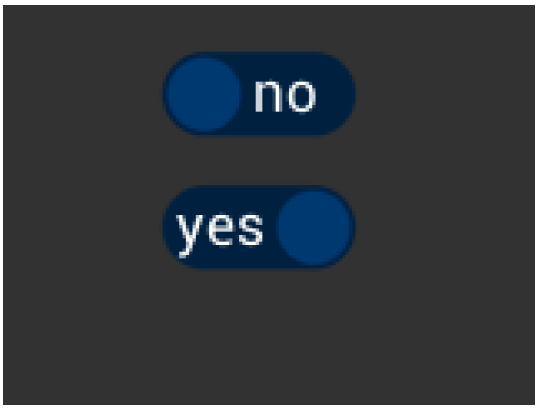
Examples

Using a medium font, in the two states



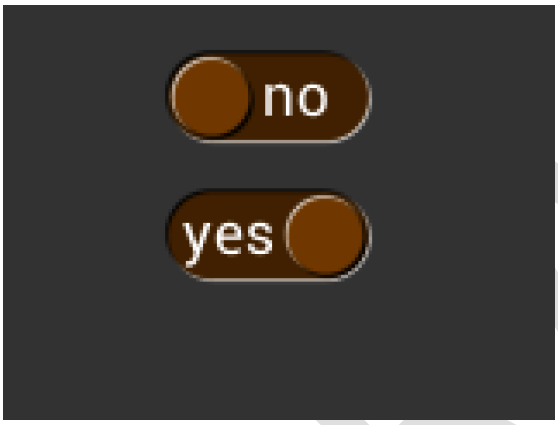

```
cmd_toggle(60, 20, 33, 27, 0, 0, "no" "\xff" "yes");
cmd_toggle(60, 60, 33, 27, 0, 65535, "no" "\xff" "yes");
```

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");
```

5.37 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. The value of OPT_RIGHTX is 2048.

Command layout

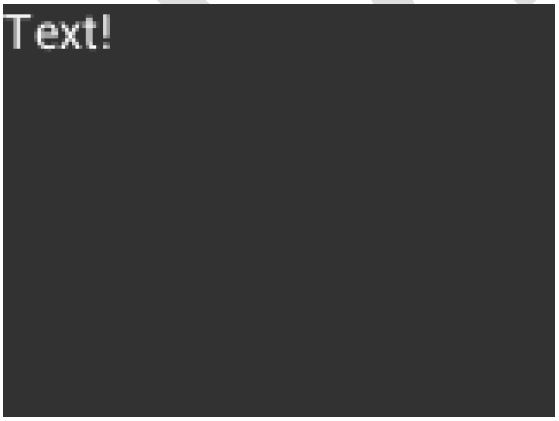
+0	CMD_TEXT(0xffffff0c)
+4	X
+6	Y
+8	Font
+10	Options
+12	S
..	..
..	0

Examples

Plain text at (0,0) in the largest font:

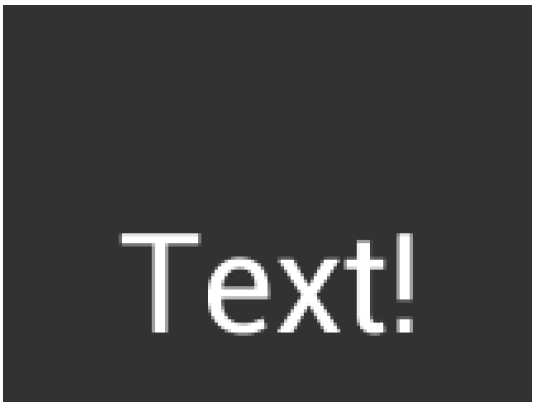
cmd_text(0, 0, 31, 0, "Text!");

Using a smaller font:



cmd_text(0, 0, 26, 0, "Text!");

Centered horizontally:



cmd_text(80, 60, 31, OPT_CENTERX, "Text!");

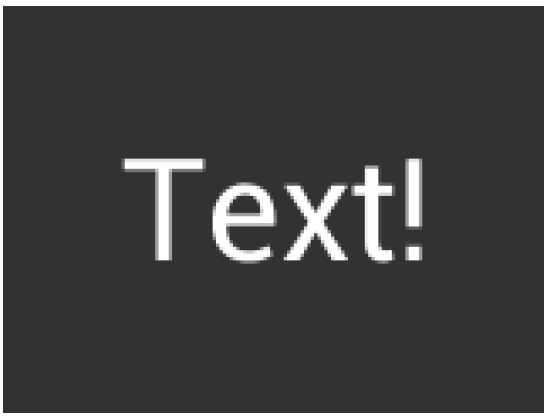
Right-justified:

cmd_text(80, 60, 31, OPT_RIGHTX, "Text!");

Centered vertically:

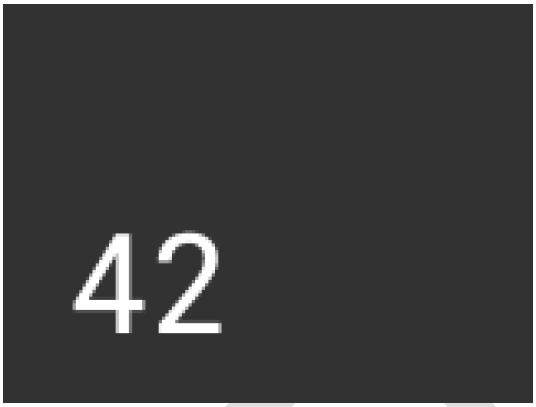
cmd_text(80, 60, 31, OPT_CENTERY, "Text!");

Centered both horizontally and vertically:




```
cmd_text(80, 60, 31, OPT_CENTER, "Text!");
```

5.38 CMD_NUMBER - draw a decimal 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

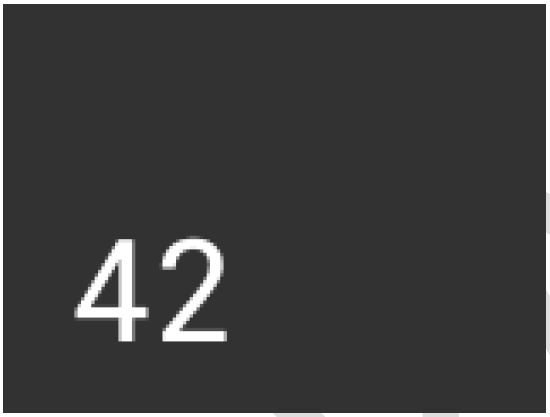
The number to display, either unsigned or signed 32-bit

Command layout

+0	CMD_NUMBER(0xffffff2e)
+4	X
+6	Y
+8	Font
+10	Options
+12	n

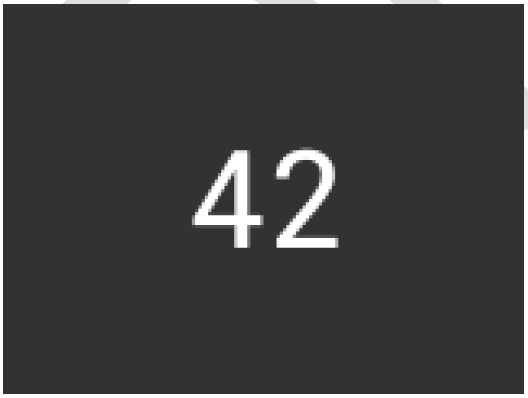
Examples

A number:



cmd_number(20, 60, 31, 0, 42);

Centered:

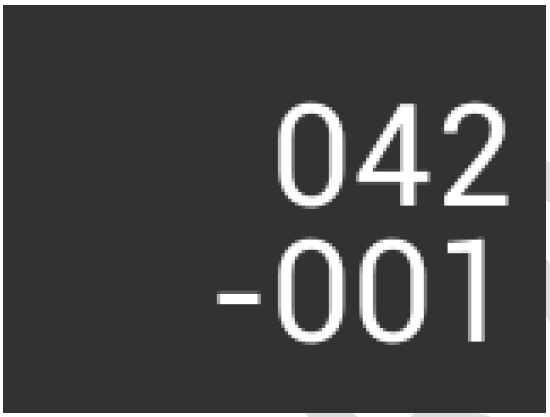


cmd_number(80, 60, 31, OPT_CENTER, 42);

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);
```

5.39 CMD_LOADIDENTITY - Set the current matrix to the identity matrix

This command instructs the co-processor engine of the FT800 to set the current matrix to the identity matrix, so that 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, please see Bitmap transformation matrix section.

C prototype

void cmd_loadidentity( );

Command layout

+0	CMD_LOADIDENTITY(0xffffff26)

5.40 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)

5.41 CMD_GETMATRIX - retrieves the current matrix coefficients

To retrieve the current matrix within the context of co-processor engine. Please note the matrix within the context of co-processor engine will not apply to the bitmap transformation until it is passed to 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 a 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

5.42 CMD_GETPTR - Get the end memory address of inflated data

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 as was specified by CMD_INFLATE, while the end address of decompressed data can be retrieved by this command.

It is one out parameter and can be passed in as any value with CMD_GETPTR to RAM_CMD.

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 14 CMD_GETPTR command example

5.43 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

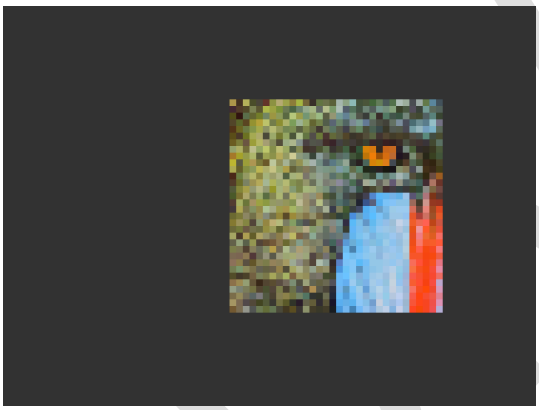
y scale factor, in signed 16. 16 bit fixed-point form.

Command layout

+0	CMD_SCALE(0xffffff28)
+4	sx
+8	sy

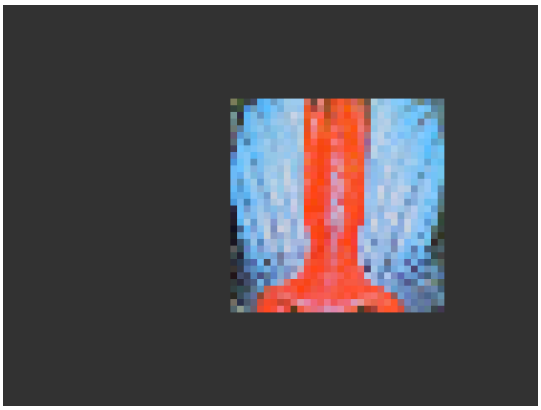
Examples

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));
```

5.44 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

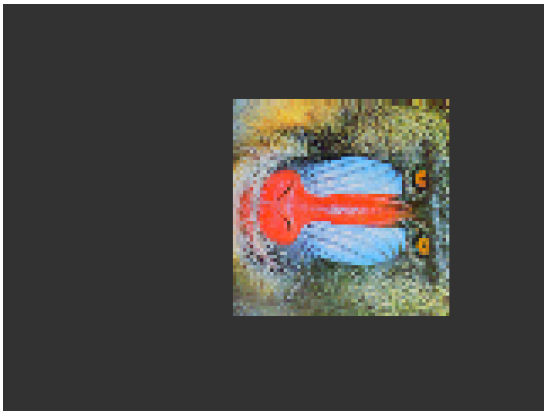
To rotate the bitmap clockwise by 10 degrees with respect to the top left of the bitmap:

```
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));
```

5.45 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

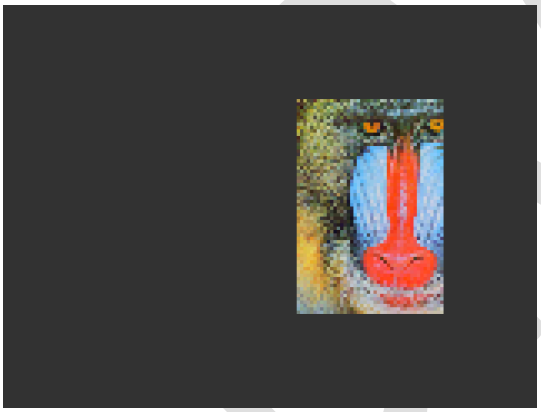
y translate factor, in signed 16.16 bit fixed-point form.

Command layout

+0	CMD_TRANSLATE(0xffffff27)
+4	Tx
+8	Ty

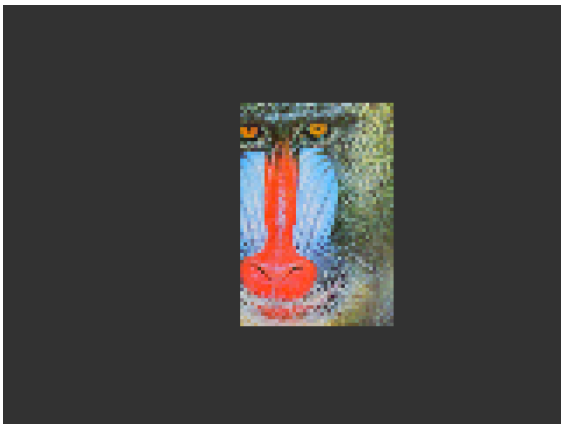
Examples

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));
```

5.46 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 it, create a display list and then use 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.

C prototype

void cmd_calibrate(uint32_t result);

Parameters

result

output parameter; written with 0 on failure of calibration.

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

Command layout

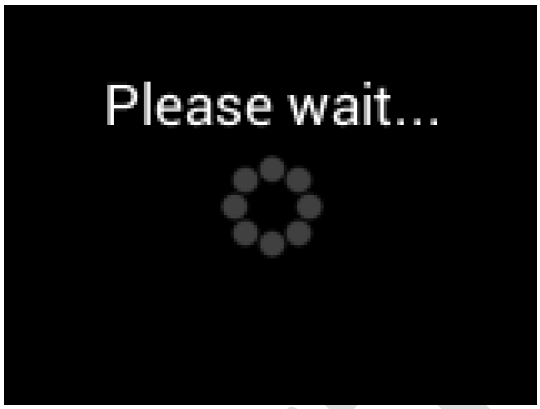
+0	CMD_CALIBRATE(0xffffff15)
+4	result

Examples

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80, 30, 27, OPT_CENTER, "Please tap on the dot");
cmd_calibrate();
```

Code snippet 15 CMD_CALIBRATE example

5.47 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 points 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 X coordinate of top left of spinner

Style

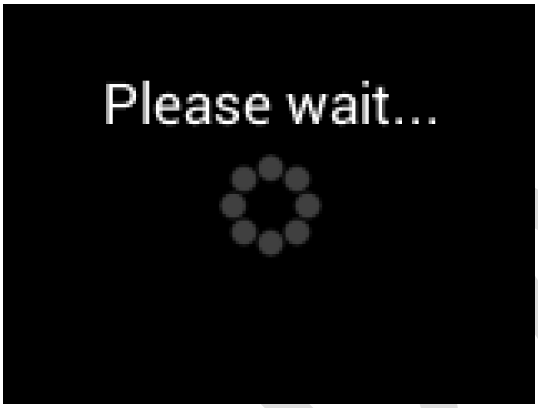
The style of spinner. Valid range is from 0 to 3.

Scale

The scaling coefficient of spinner. 0 means no scaling.

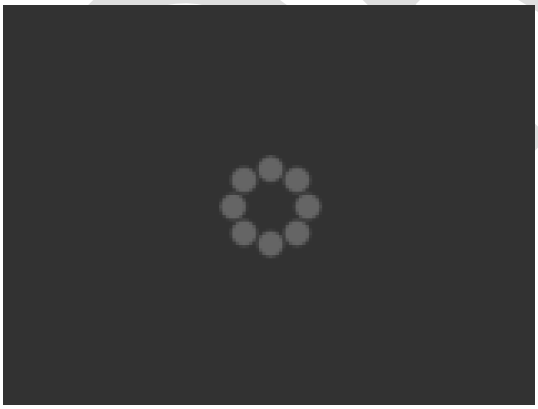
Examples

Create a display list, then start the spinner:



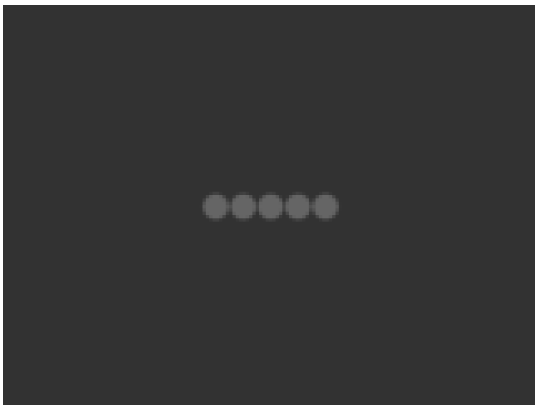

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80, 30, 27, OPT_CENTER, "Please wait...");
cmd_spinner(80, 60, 0, 0);
```

Spinner style 0, a circle of dots:



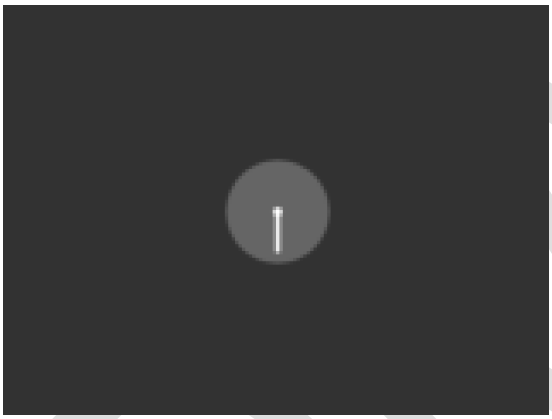
```
cmd_spinner(80, 60, 0, 0);
```

Style 1, a line of dots:



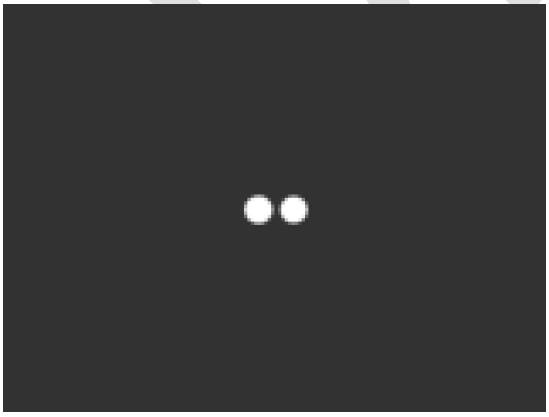
cmd_spinner(80, 60, 1, 0);

Style 2, a rotating clock hand:



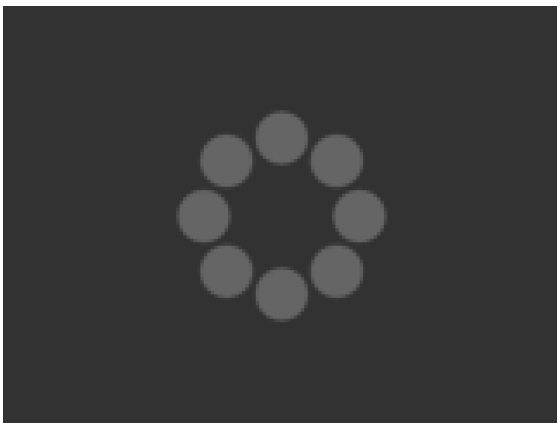
cmd_spinner(80, 60, 2, 0);

Style 3, two orbiting dots:



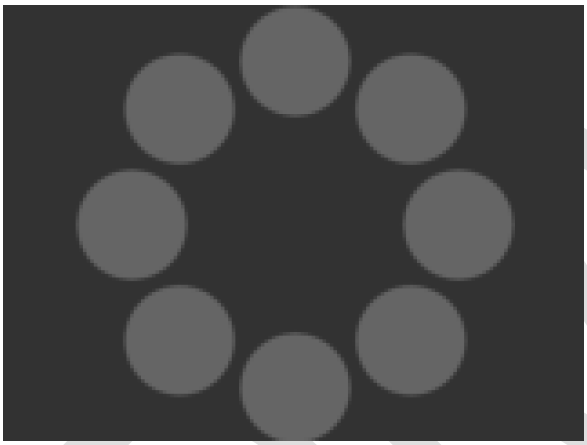
cmd_spinner(80, 60, 3, 0);

Half screen, scale 1:



cmd_spinner(80, 60, 0, 1);

Full screen, scale 2:



cmd_spinner(80, 60, 0, 2);

5.48 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 frequency of frames displayed (depending on the display registers configuration). Typically for 480x272 display the frame rate is around 60fps.

Command layout

+0	CMD_SCREENSAVER(0xffffff2f)

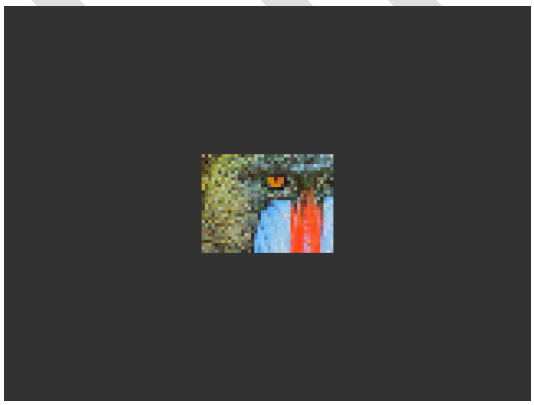
Examples

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 16 CMD_SCREENSAVER example

Here is the result:



5.49 CMD_SKETCH - start a continuous sketch update

After the sketch command, the co-processor engine continuously samples the touch inputs and paints pixels into a bitmap, according to the touch (x, y). This means that the user touch inputs are drawn into the bitmap without any need for MCU work. Command CMD_STOP stops 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

The details of physical dimension are

- Update frequency of sketch depends on touch frequency (up to max of 1000Hz can be achieved)

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

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 17 CMD_SKETCH example

5.50 CMD_STOP - stop any of spinner, screensaver or sketch

C prototype

void cmd_stop( );

Command layout

+0	CMD_STOP(0xffffff17)

Examples

See CMD_SKETCH

5.51 CMD_SETFONT - set up a custom font

CMD_SETFONT is used to register one custom defined bitmap font into the FT800 co-processor engine. After registration, the FT800 co-processor engine is able to use the bitmap font with its co-processor command.

About the details about how to set up custom font, please 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 14. Bitmap handle 15 is

ptr

The metric block address in RAM. 4 bytes aligned is required.

Examples

With a suitable font metric 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 18 CMD_SETFONT example

5.52 CMD_TRACK - track touches for a graphics object

This command will enable co-processor engine to track the touch on the particular graphics object with one valid tag value assigned. Then, co-processor engine will update the REG_TRACKER periodically with the frame rate of LCD display panel.

Co-processor engine tracks the graphics object in rotary tracker mode and linear tracker mode:

- rotary tracker mode – Track the angle between the touching point and the center of graphics object specified by 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 touching point to the width of graphics object specified by tag value. If parameter w is not greater than h, Track the relative distance of touching point to the height of graphics object specified by tag value. The value is in units of 1/65536 of the width or height of graphics object. The distance of touching point refers to the distance from the top left pixel of graphics object to the coordinate of touching point.

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.

Please 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 co-processor engine.

tag

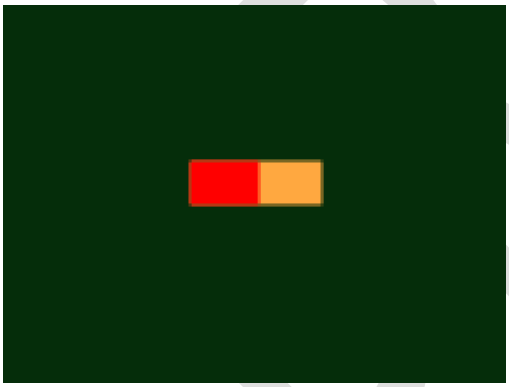
tag of the graphics object to be tracked, 1-255

Command layout

+0	CMD_TRACK(0xffffff2c)
+4	X
+6	Y
+8	W
+10	h
+12	tag

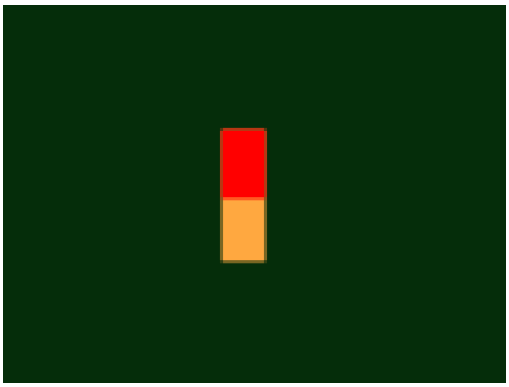
Examples

Horizontal track of rectangle dimension 40x12pixels and the present touch is at 50%:



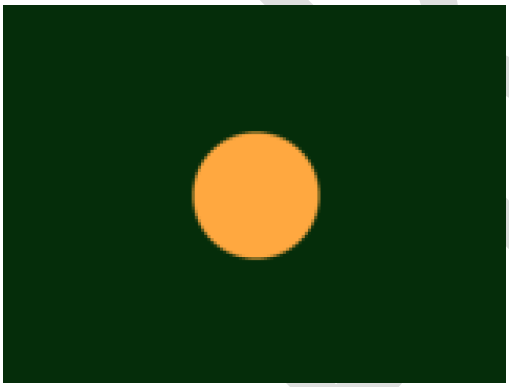
```
dl( CLEAR_COLOR_RGB(5, 45, 110) );
dl( COLOR_RGB(255, 168, 64) );
dl( CLEAR(1 ,1 ,1) );
dl( BEGIN(RECTS) );
dl( VERTEX2F(60 * 16,50 * 16) );
dl( VERTEX2F(100 * 16,62 * 16) );
dl( COLOR_RGB(255, 0, 0) );
dl( VERTEX2F(60 * 16,50 * 16) );
dl( VERTEX2F(80 * 16,62 * 16) );
dl( COLOR_MASK(0 ,0 ,0 ,0) );
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);
```

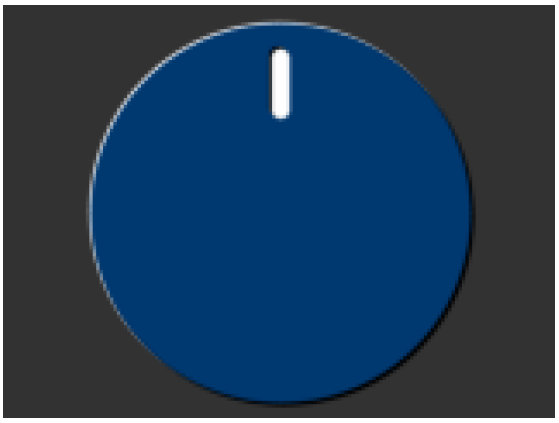
Circular track centered at (80,60) display location



```
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);
```

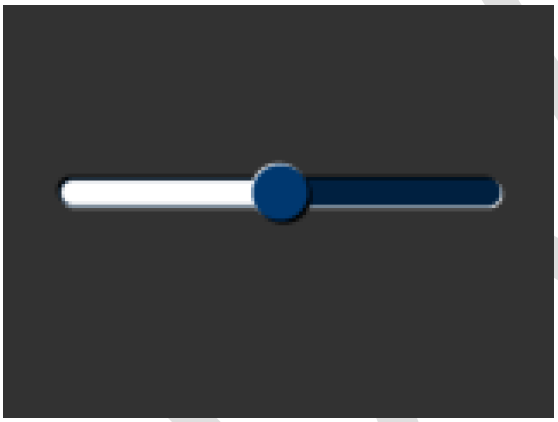
To draw a dial with tag 33 centered at (80, 60), adjustable by touch:

```
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);
while (1) {
...
cmd(TAG(34));
cmd_slider(20, 50, 120, 8, 0, val, 65535);
...
uint32_t tracker = rd32(REG_TRACKER);
if ((tracker & 0xff) == 33)
val = tracker >> 16;
...
}
```

5.53 CMD_SNAPSHOT - take a snapshot of the current screen

The snapshot command causes the co-processor engine to take a snapshot of the current screen, and write it into graphics memory as a 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.

If the current screen sources bitmap memory that is written by CMD_SNAPSHOT, then the results are undefined.

C prototype

void cmd_snapshot(uint32_t ptr);

Parameters

ptr

Snapshot destination address, in main memory

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

Command layout

+0	CMD_SNAPSHOT(0xffffff1f)
+4	ptr

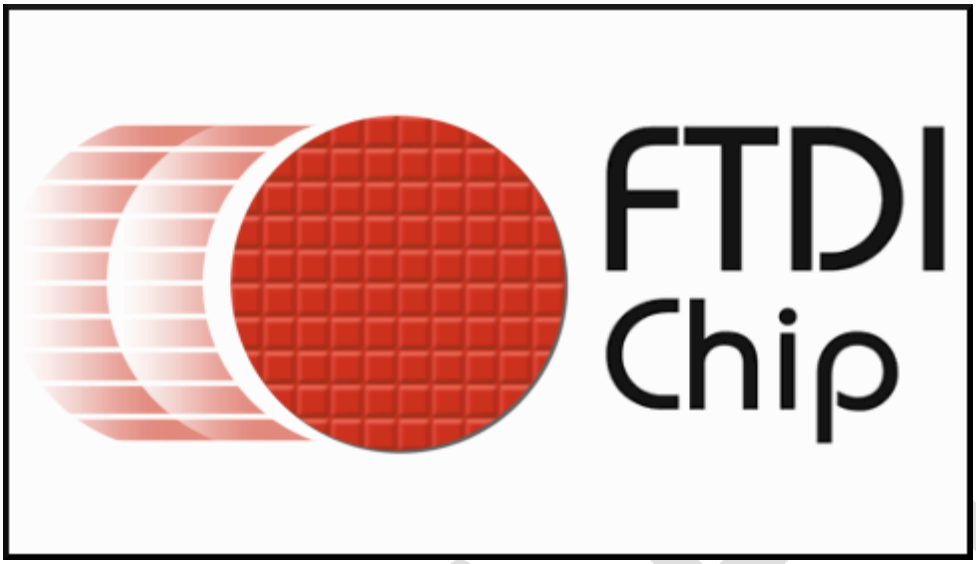
Examples

To take a snapshot of the current 160 x 120 screen, 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 19 CMD_SNAPSHOT 160x120 screen

5.54 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 FT800 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 20 CMD_LOGO command example

Appendix A – Document References

1) DS_FT800_Embedded_Video_Engine

Appendix B – Acronyms and Abbreviations

Terms	Description
CS	Chip select
DL/dl	Display list
EVE	Embedded Video Engine
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
RGB	Red Blue Green
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
USB-IF	USB Implementers Forum
RO	Read only

APPENIX C – Address Memory Map

Start Address	End Address	Size	NAME	Description
00 0000h	03 FFFFh	256 kB	RAM_G	Main graphics RAM
0C 0000h	0C 0003h	4 B	ROM_CHIPID	FT800 chip identification and revision information: Byte [0:1] Chip ID: "0800" Byte [2:3] Version ID: "0100"
0B B23Ch	0F FFFBh	275 kB	ROM_FONT	Font table and bitmap
0F FFFCh	0F FFFFh	4 B	ROM_FONT_ADDR	Font table pointer address
10 0000h	10 1FFFh	8 kB	RAM_DL	Display List RAM
10 2000h	10 23FFh	1 kB	RAM_PAL	Palette RAM
10 2400h	10 257Fh	380 B	REG_*	Registers
10 8000 h	10 8FFFh	4 kB	RAM_CMD	Graphics Engine Command Buffer

NOTE: The addresses beyond this table are reserved and shall not be read or written.

Appendix D – Revision History

Document Title: FT800 Programmer Guide
Document Reference No.: FT_000793
Clearance No.: FTDI# 349
Product Page: http://www.ftdichip.com/FTProducts.htm
Document Feedback: Send Feedback

Revision	Changes	Date
0.1	Initial Release	2012-08-01
0.8	Draft Release	2013-08-29