



Application Note

BRT_AN_088

FT81x/BT88x Series Programming Guide

Version 1.0

Issue Date: 13-09-2023

This application note describes the process and practice required to program 2nd generation **EVE** chips: FT81x(FT810, FT811, FT812, FT813) and BT88x(BT880, BT881, BT882, BT883).

Use of Bridgetek devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold Bridgetek harmless from any and all damages, claims, suits or expense resulting from such use.

Bridgetek Pte Ltd (BRT Chip)

1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464

Tel: +65 6547 4827

Web Site: <http://www.brtchip.com>

Copyright © Bridgetek Pte Ltd

Table of Contents

1	Introduction	9
1.1	Scope	9
1.2	Conventions	9
1.3	Intended Audience.....	9
1.4	API Reference Definitions	9
2	Programming Model	11
2.1	Address Space	11
2.2	Data Flow Diagram	12
2.3	Read Chip Identification Code (ID)	13
2.4	Initialization Sequence	13
2.5	PWM Control	15
2.6	RGB Color Signal	15
2.7	Touch Screen	15
2.8	Audio Routines	16
2.8.1	Sound Effect	16
2.8.2	Audio Playback	17
2.9	Graphics Routines	18
2.9.1	Getting Started.....	18
2.9.2	Coordinate Range and Pixel Precision	19
2.9.3	Screen Rotation	19
2.9.4	Drawing Pattern.....	21
2.9.5	Bitmap Transformation Matrix	23
2.9.6	Color and Transparency	23
2.9.7	Bitmap Rendering Performance.....	24
3	Register Description	25
3.1	Graphics Engine Registers	25
3.2	Audio Engine Registers	29
3.3	Touch Engine Registers.....	32

3.3.1	Overview	32
3.3.2	Common Registers.....	32
3.3.3	Resistive Touch Screen Engine (RTSE).....	34
3.3.4	Capacitive Touch Screen Engine (CTSE).....	37
3.3.5	Calibration	42
3.4	Coprocessor Engine Registers.....	42
3.5	Miscellaneous Registers.....	43
3.6	Special Registers	47
4	Display List Commands.....	50
4.1	Graphics State	50
4.2	Command Encoding	50
4.3	Command Groups	51
4.3.1	Setting Graphics State.....	51
4.3.2	Drawing Actions.....	51
4.3.3	Execution Control.....	51
4.4	ALPHA_FUNC	52
4.5	BEGIN	52
4.6	BITMAP_HANDLE	53
4.7	BITMAP_LAYOUT	54
4.8	BITMAP_LAYOUT_H	58
4.9	BITMAP_SIZE	58
4.10	BITMAP_SIZE_H	59
4.11	BITMAP_SOURCE	60
4.12	BITMAP_TRANSFORM_A	61
4.13	BITMAP_TRANSFORM_B	62
4.14	BITMAP_TRANSFORM_C	62
4.15	BITMAP_TRANSFORM_D	63
4.16	BITMAP_TRANSFORM_E	63
4.17	BITMAP_TRANSFORM_F	64

4.18	BLEND_FUNC	65
4.19	CALL	66
4.20	CELL.....	67
4.21	CLEAR	67
4.22	CLEAR_COLOR_A	69
4.23	CLEAR_COLOR_RGB	69
4.24	CLEAR_STENCIL.....	70
4.25	CLEAR_TAG.....	71
4.26	COLOR_A	71
4.27	COLOR_MASK.....	72
4.28	COLOR_RGB	73
4.29	DISPLAY	74
4.30	END.....	74
4.31	JUMP.....	75
4.32	LINE_WIDTH	75
4.33	MACRO	76
4.34	NOP	76
4.35	PALETTE_SOURCE	77
4.36	POINT_SIZE.....	77
4.37	RESTORE_CONTEXT	78
4.38	RETURN	79
4.39	SAVE_CONTEXT	79
4.40	SCISSOR_SIZE	80
4.41	SCISSOR_XY	81
4.42	STENCIL_FUNC	82
4.43	STENCIL_MASK.....	82
4.44	STENCIL_OP	83
4.45	TAG.....	84

4.46	TAG_MASK	84
4.47	VERTEX2F	85
4.48	VERTEX2II	86
4.49	VERTEX_FORMAT	86
4.50	VERTEX_TRANSLATE_X.....	87
4.51	VERTEX_TRANSLATE_Y.....	88
5	Coprocessor Engine	89
5.1	Command FIFO	89
5.2	Widgets	90
5.2.1	Common Physical Dimensions	91
5.2.2	Color Settings.....	91
5.2.3	Caveat.....	91
5.3	Interaction with RAM_DL.....	91
5.3.1	Synchronization between host & Coprocessor Engine.....	92
5.4	ROM and RAM Fonts.....	92
5.4.1	Legacy Font Metrics Block	92
5.4.2	Example to find the width of character.....	93
5.4.3	ROM Fonts (Built-in Fonts)	93
5.4.4	Using Custom Fonts	94
5.5	Coprocessor Faults	94
5.6	Coprocessor Graphics State	94
5.7	Parameter OPTION	95
5.8	Resources Utilization	96
5.9	Command Groups	96
5.10	CMD_DLSTART	98
5.11	CMD_INTERRUPT	98
5.12	CMD_COLDSTART.....	99
5.13	CMD_SWAP.....	99
5.14	CMD_APPEND	99

5.15	CMD_REGREAD	100
5.16	CMD_MEMWRITE	101
5.17	CMD_INFLATE.....	101
5.18	CMD_LOADIMAGE	102
5.19	CMD_MEDIAFIFO	103
5.20	CMD_PLAYVIDEO	104
5.21	CMD_VIDEOSTART	105
5.22	CMD_VIDEOFRAME	106
5.23	CMD_MEMCRC.....	106
5.24	CMD_MEMZERO.....	107
5.25	CMD_MEMSET	108
5.26	CMD_MEMCPY	108
5.27	CMD_BUTTON	109
5.28	CMD_CLOCK.....	110
5.29	CMD_FGCOLOR	113
5.30	CMD_BGCOLOR	114
5.31	CMD_GRADCOLOR.....	114
5.32	CMD_GAUGE	115
5.33	CMD_GRADIENT.....	119
5.34	CMD_GRADIENTA	120
5.35	CMD_KEYS	122
5.36	CMD_PROGRESS	124
5.37	CMD_SCROLLBAR.....	126
5.38	CMD_SLIDER.....	127
5.39	CMD_DIAL	129
5.40	CMD_TOGGLE.....	130
5.41	CMD_TEXT	132
5.42	CMD_SETBASE	134

5.43	CMD_NUMBER	135
5.44	CMD_LOADIDENTITY	136
5.45	CMD_SETMATRIX	137
5.46	CMD_GETMATRIX.....	137
5.47	CMD_GETPTR	138
5.48	CMD_GETPROPS.....	138
5.49	CMD_SCALE	139
5.50	CMD_ROTATE	140
5.51	CMD_TRANSLATE	141
5.52	CMD_CALIBRATE.....	142
5.53	CMD_SETROTATE	142
5.54	CMD_SPINNER	143
5.55	CMD_SCREENSAVER.....	144
5.56	CMD_SKETCH	145
5.57	CMD_STOP	146
5.58	CMD_SETFONT	147
5.59	CMD_SETFONT2	147
5.60	CMD_SETSCRATCH	148
5.61	CMD_ROMFONT.....	149
5.62	CMD_TRACK	150
5.63	CMD_SNAPSHOT	153
5.64	CMD_SNAPSHOT2	153
5.65	CMD_SETBITMAP	155
5.66	CMD_LOGO	156
5.67	CMD_SYNC.....	157
5.68	CMD_BITMAP_TRANSFORM	157
5.69	CMD_CSKETCH - Deprecated	159
6	Contact Information	160

Appendix A – References	161
Document References	161
Acronyms and Abbreviations.....	161
Appendix B – List of Tables/ Figures/ Registers/ Code Snippets.....	162
List of Tables.....	162
List of Figures	162
List of Registers	162
List of Code Snippets	165
Appendix C – Revision History	166

1 Introduction

This document captures the programming details for the 2nd generation **EVE** chips, which includes **FT81x** and **BT88x** Series chips.

This document supersedes the previous version of the **FT81x** programming guide and includes the essential information for the newly launched **BT88x**.

EVE chips serve as graphics controllers that come with additional functionalities such as touch capabilities and audio playback. They offer a diverse array of graphics objects that can be utilized to showcase a wide range of products, including home appliances, toys, industrial machinery, home automation systems, elevators, and more.

1.1 Scope

This document provides comprehensive information on programming **EVE**, including the programming model, memory map, registers, and commands. It also includes example routines that demonstrate the ease of using the command set. The document covers various power modes, audio, and touch features, as well as how to use them effectively.

The descriptions in this document are applicable to both **FT81x** and **BT88x**, unless specified otherwise.

Information on pin settings, hardware characteristics and hardware configurations, serial data protocol can be found in the datasheet.

1.2 Conventions

All values are in decimal by default.

The values with **0x** are in hexadecimal.

The values with **0b** are in binary.

The endianness of commands, register values, and data in **RAM_G** are in *little-endian* format.

Host refers to the **MCU/MPU** with SPI master interface connecting with **EVE**.

Host Commands refer to the **EVE** specific commands defined in the Serial Data Protocol section of the datasheet.

1.3 Intended Audience

The intended audience of this document are Software Programmers and System Designers who develop graphical user interface (**GUI**) applications on any host with an **SPI** master interface.

1.4 API Reference Definitions

The following table provides the 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
rd8()	read 8 bits from intended address location

rd16()	read 16 bits from intended address location
rd32()	read 32 bits from intended address location
cmd()	write 32 bits data to command FIFO i.e., RAM_CMD
cmd_*	Write 32 bits command with its necessary parameters to command FIFO i.e., RAM_CMD .
dl()	Write 32 bits display list command to RAM_DL .
host_command()	send host command to EVE

Table 1 – API Reference Definitions

2 Programming Model

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

2.1 Address Space

The entire memory and registers of **EVE** are mapped into a 22-bit address space. The address space definitions are provided in the following table:

Name	Start Address	End Address	Size (bytes)	Description
RAM_G	0x000000	0x0FFFFF	1024 Ki	General purpose graphics RAM , also called main memory in this document. Applicable to FT81x .
RAM_G	0x000000	0x03FFFF	256 Ki	General purpose graphics RAM , also called main memory in this document. Applicable to BT88x .
ROM_FONTRoot	0x2FFFFC	0x2FFFFF	4	Font table pointer address
RAM_DL	0x300000	0x301FFF	8 Ki	Display list RAM
RAM_REG	0x302000	0x302FFF	4 Ki	Registers
RAM_CMD	0x308000	0x308FFF	4 Ki	Command FIFO

Table 2 – Memory Map

Note:

1. A serial data protocol has been defined for the host to adhere to when attempting to access these addresses. Please refer to the datasheet for more details.
2. The addresses beyond this table are reserved and shall not be read or written unless otherwise specified.

2.2 Data Flow Diagram

From the programmer's perspective, Figure 1 illustrates the data flow between the host and internal components of **EVE**. Please note that the direct write from host to **RAM_DL** requires extra careful actions to sync up the read/write pointers (specified by the respective registers of **EVE**) because the coprocessor engine may also write the generated display list commands into **RAM_DL**.

To save such effort, the better approach is to write the display list commands to **RAM_CMD** and make the coprocessor update the **RAM_DL** accordingly.

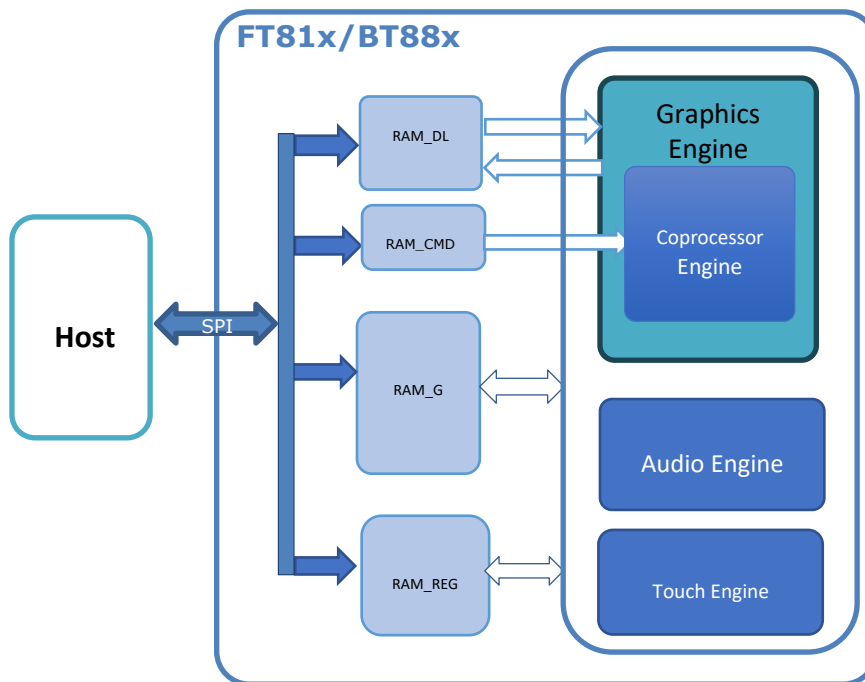


Figure 1 – FT81x/BT88x data flow

The data here refers to the following items:

- **Display list:** Instructions for graphics engine to render the screen
- **Coprocessor command:** Predefined commands by coprocessor engine
- **Bitmap data:** Pixel data in EVE defined formats: such as **RGB565**, **L8** etc.
- **JPEG/PNG stream:** Image data in **PNG/JPEG** format conforming to **EVE** requirement, for coprocessor engine to decode.
- **MJPEG stream:** The video data in **MJPEG** format conforming to **EVE** requirement for coprocessor engine to decode.
- **Audio stream:** The audio samples in **uLaw**, **ADPCM**, **PCM**
- **Register values:** The content of registers.

2.3 Read Chip Identification Code (ID)

Upon resetting or rebooting, the chip ID for **FT81x** can be accessed by reading 4 bytes of data from addresses 0xC0000 to 0xC0003, while for **BT88x**, it can be accessed by reading 4 bytes of data from addresses 0x0 to 0x3. Users must read this data before any application overwrites the respective addresses to retrieve the chip identification code.

The following table describes the data to be read:

	0xC0003	0xC0002	0xC0001	0xC0000
FT81x	0x00	0x01	0x10 for FT810 0x11 for FT811 0x12 for FT812 0x13 for FT813	0x08
	0x00003	0x00002	0x00001	0x00000
BT88x	0x00	0x01	0x80 for BT880 0x81 for BT881 0x82 for BT882 0x83 for BT883	0x08

Table 3 – Read Chip Identification Code

2.4 Initialization Sequence

• Initialization Sequence during the boot up

During boot up, the following steps are required:

1. Send host command "**CLKEXT**" if the PLL input is from external crystal oscillator or external clock.
2. Send host command "**CLKSEL**" to select system clock frequency if the non-default system clock is to be used. By default, the system clock is set to 60MHz.
3. Send host command "**RST_PULSE**" to reset the core of **EVE**.
4. Send host command "**ACTIVE**" to enable the clock. The self-diagnosis process might take a maximum of 300 milliseconds to finish. Alternatively, you can perform sequential reads of **REG_ID** until a value of 0x7C is obtained, followed by reading **REG_CPURESET** until a value of zero is obtained.
5. Configure display control timing registers, except **REG_PCLK**
6. Write first display list to **RAM_DL**.
7. Write **REG_DLSWAP** to start graphics engine rendering process with first display list
8. Enable backlight control for display panel
9. Write **REG_PCLK** to configure the PCLK frequency of display panel, which leads to the output of the first display list

```

host_command(CLKEXT); //send command "CLKEXT" to use the external clock source
host_command(CLKSEL); //select the system clock frequency
host_command(RST_PULSE); //send host command "RST_PULSE" to reset
host_command(ACTIVE); //send host command "ACTIVE" to wake up

while (0x7C != rd8(REG_ID)); //Wait till clock is on
while (0x0 != rd16(REG_CPURESET)); //Check if EVE is in working status.

/* Configure display registers - demonstration for WQVGA 480x272 resolution */
wr16(REG_HCYCLE, 548);
wr16(REG_HOFFSET, 43);
wr16(REG_HSYNC0, 0);
wr16(REG_HSYNC1, 41);
wr16(REG_VCYCLE, 2923);
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 to display list memory RAM_DL*/
wr32(RAM_DL+0,CLEAR_COLOR_RGB(0,0,0));
wr32(RAM_DL+4,CLEAR(1,1,1));
int offset = 8;
for (int i=0; i < 16; i++)
{
    wr32(RAM_DL+offset,BITMAP_HANDLE(i));
    offset += 4;
    wr32(RAM_DL+offset,BITMAP_LAYOUT_H(0));
    offset += 4;
    wr32(RAM_DL+offset,BITMAP_SIZE_H(0));
    offset += 4;
}
wr32(RAM_DL+offset,DISPLAY());

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

/* Enable backlight of display panel */
wr8(REG_GPIO_DIR,0xff);
wr8(REG_GPIO,0xff);

wr8(REG_PCLK,5); //Configure the PCLK divisor to 5, i.e., PCLK = System CLK/5

```

Code Snippet 1 – Initialization Sequence

Note:

1. During initialization, the SPI clock frequency must not exceed 11MHz, but it can be increased up to 30MHz after the initialization process. However, once the initialization is complete, this frequency can be raised to a maximum of 30MHz if the **EVE** operates in single SPI mode. When the **EVE** is configured in Quad SPI mode, the highest allowable SPI frequency becomes 25MHz, provided it does not exceed *half* of the system clock frequency.
2. Upon bootup, the bitmap handle setup parameters "**bitmap_layout_h/bitmap_size_h**" may have a non-zero value, which can result in the corruption of the rendered bitmap, particularly if the bitmap size is less than 512 pixels. To avoid this issue, it is advisable to set these parameters to zero for bitmap handle 0 to 15 in the initial display list by sending display list as above.

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

1. Drive the **PD_N** pin high
2. Wait for at least 20ms
3. Execute "Initialization Sequence during the Boot UP"

- **Initialization Sequence from Sleep Mode:**

1. Send the Host command "**ACTIVE**" to enable the clock
2. Wait for at least 20ms
3. Execute "Initialization Sequence during the Boot UP" from step 5 to 9

- **Initialization Sequence from Standby Mode:**

1. Send the Host command "**ACTIVE**" to enable the clock
2. Wait for at least 20ms

3. Execute "Initialization Sequence during the Boot UP" from step 5 to 9

- **Initialization sequence from standby mode:**
Execute all the steps mentioned in "Initialization Sequence from Sleep Mode" except the waiting time.

2.5 PWM Control

The PWM signal is controlled by two registers: **REG_PWM_HZ** and **REG_PWM_DUTY**.

REG_PWM_HZ specifies the PWM output frequency.

REG_PWM_DUTY specifies the PWM output duty cycle.

2.6 RGB Color Signal

The RGB color signal is carried over 24 signals - 8 each for red, green and blue. Several registers affect the operation of these signals. The order of these operations in the display output system is as follows:

REG_DITHER enables color dither. To improve the image quality, **EVE** applies a 2×2 color dither matrix to output pixels. The dither option improves half-tone appearance on displays, even on 1-bit displays.

REG_OUTBITS gives the bit width of each color channel. The default is zero, meaning 8 bits each channel. Lower values mean that fewer bits are output for the color channel. This value also affects dither computation.

REG_SWIZZLE controls the arrangement of the output color pins, to help PCB routing with different LCD panel arrangements. Bit 0 of the register causes the order of bits in each color channel to be reversed. Bits 1-3 control the RGB order. Bit 1 set causes R and B channels to be swapped. Bit 3 is rotate enable. If bit 3 is set, then (R, G, B) is rotated right if bit 2 is one, or left if bit 2 is zero. Please refer to the datasheet for more details.

2.7 Touch Screen

The raw touch screen coordinate (x, y) values are available in register **REG_TOUCH_RAW_XY**. The range of these values is 0-1023. If the touch screen is not being pressed, both registers read 0xFFFF.

These touch values are transformed into screen coordinates using the matrix in registers **REG_TOUCH_TRANSFORM_A-F**. The post-transform coordinates are available in register **REG_TOUCH_SCREEN_XY**. If the touch screen is not being pressed, both registers read 0x8000 (-32768). The values for **REG_TOUCH_TRANSFORM A-F** may be computed using an on-screen calibration process. If the screen is being touched, the screen coordinates are looked up in the screen's tag buffer, delivering a final 8-bit tag value, in **REG_TOUCH_TAG**. Because the tag lookup takes a full frame, and touch coordinates change continuously, the original (x, y) used for the tag lookup is also available in **REG_TOUCH_TAG_XY**.

Screen touch pressure is available in **REG_TOUCH_RZ**. This register gives the resistance of the touch screen press, so lower values indicate more pressure. The register's range is 0 (maximum pressure) to 32767 (no touch). Because the values depend on the particular screen, and the instrument used for pressing (stylus, fingertip, gloved finger, etc.) a calibration or setup step shall be used to discover the useful range of resistance values.

REG_TOUCH_MODE controls the frequency of touch sampling. **TOUCHMODE_CONTINUOUS** is continuous sampling. Writing **TOUCHMODE_ONESHOT** causes a single sample to occur.

TOUCHMODE_FRAME causes a sample at the start of each frame. **TOUCHMODE_OFF** stops all sampling.

REG_TOUCH_ADC_MODE selects single-ended (**ADC_SINGLE_ENDED**) or differential (**ADC_DIFFERENTIAL**) ADC operation. Single-ended consumes less power, differential gives more accurate positioning. **REG_TOUCH_CHARGE** specifies how long to drive the touchscreen voltage before sampling the pen detect input.

REG_TOUCH_RZTHRESH specifies a threshold for touchscreen resistance. If the measured touchscreen resistance is greater than this threshold, then no touch is reported. The default value is 65535, so all touches are reported.

REG_TOUCH_SETTLE specifies how long to drive the touchscreen voltage before sampling the position. For screens with a large capacitance, this value should be increased. For low capacitance screens this value can be decreased to reduce "on" time and save power.

REG_TOUCH_OVERSAMPLE controls the oversampling factor used by the touchscreen system. Increase this value to improve noise rejection if necessary. For systems with low noise, this value can be lowered to reduce "on" time and save power.

Touch screen 32-bit register updates are atomic: all 32 bits are updated in a single cycle. So, when reading an XY register, for example, both (x, y) values are guaranteed to be from the same sensing cycle. When the sensing cycle is complete, and the registers have been updated, the **INT_CONV_COMPLETE** interrupt is triggered.

As well as the above high-level samples, the direct 10-bit **ADC** values are available in two registers, **REG_TOUCH_DIRECT_XY** and **REG_TOUCH_DIRECT_Z1Z2**. These registers are laid out as follows:

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

The S field is 0 if a touch is being sensed, in which case all fields hold their sensed values. If S is 1, then no touch is sensed and all fields should be ignored.

2.8 Audio Routines

The audio engine has two functionalities: synthesize built-in sound effects with selected pitches and play back the audio data in **RAM_G**.

2.8.1 Sound Effect

The audio engine has various sound data built-in to work as a sound synthesizer. Sample code to play C8 on the xylophone:

```

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

Code Snippet 2 – Play C8 on the Xylophone

Sample code to stop sound play:

```
wr16(REG_SOUND,0x0); //configure "silence" 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 3 – Stop Playing Sound

In order to prevent a popping sound during reset or changes in power state, or to minimize electromagnetic interference (**EMI**) noise from the audio output pin, initiate a "mute" sound and allow it to finish before proceeding.

```
wr16(REG_SOUND,0x60); //configure "mute" 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 – Avoid Pop Sound

Note: Refer to datasheet for more information on the sound synthesizer and audio playback.

2.8.2 Audio Playback

The audio engine supports an audio playback feature. For the audio data in the **RAM_G** to play back, it requires the start address in **REG_PLAYBACK_START** to be 8 bytes aligned. In addition, the length of audio data specified by **REG_PLAYBACK_LENGTH** is required to be 8 Bytes aligned.

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

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

2.9 Graphics Routines

This section describes graphics features and captures a few examples. Please note that the code in this section is for the purpose of illustrating the operation of Display Lists. Application will normally send the commands via **command FIFO (RAM_CMD)** instead of writing directly to **RAM_DL**.

2.9.1 Getting Started

The following example creates a screen with the text "TEXT" on it, with a red dot.

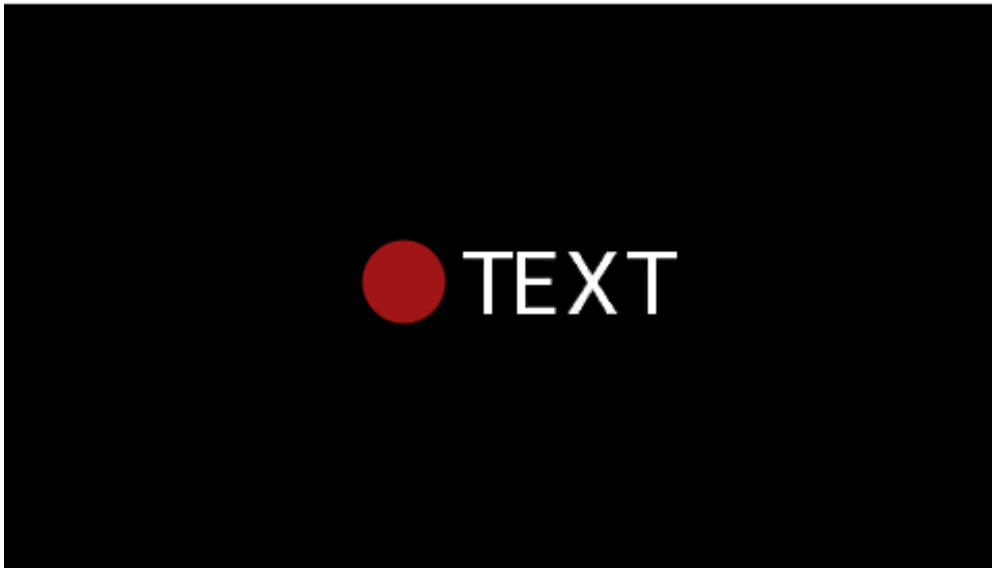


Figure 2 – Getting Started Example

The code to draw the screen is:

```

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

Code Snippet 8 – Getting Started

Upon loading the above drawing commands into **RAM_DL**, register **REG_DLSWAP** is required to be set to **0x02** in order to make the new display list active on the next frame refresh.

Note:

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

2.9.2 Coordinate Range and Pixel Precision

Apart from the single pixel precision, **EVE** supports a series of fractional pixel precision, which result in a different coordinate range. Users may trade the coordinate range against pixel precision. See **VERTEX_FORMAT** for more details.

Please note that the maximum screen resolution which **EVE** can render is up to 2048 by 2048 in pixels only, regardless of which pixel precision is specified.

VERTEX2F and **VERTEX_FORMAT** are the commands that enable the drawing operation to reach the full coordinate plane. The **VERTEX2II** command only allows positive screen coordinates. The **VERTEX2F** command allows negative coordinates. If the bitmap is partially off-screen, for example during a screen scroll, then it is necessary to specify negative screen coordinates.

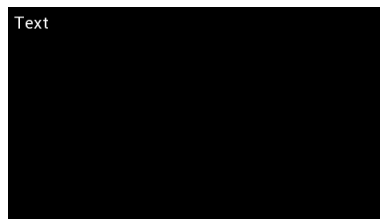
2.9.3 Screen Rotation

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

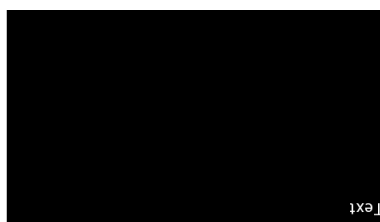
Note: The touch transformation matrix is not affected by setting **REG_ROTATE**.

To adjust the touch screen accordingly, users are recommended to use [CMD_SETROTATE](#) as opposed to setting **REG_ROTATE**.

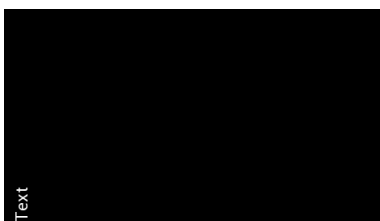
REG_ROTATE = 0 is the default landscape orientation:



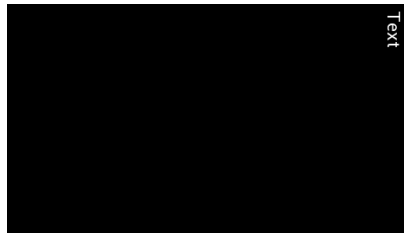
REG_ROTATE = 1 is inverted landscape:



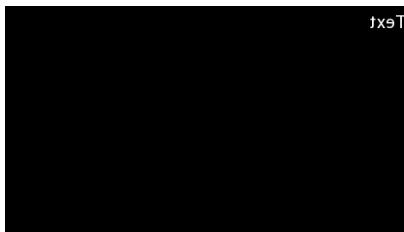
REG_ROTATE = 2 is portrait:



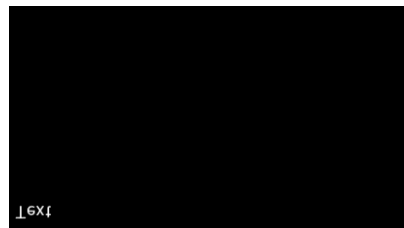
REG_ROTATE = 3 is inverted portrait:



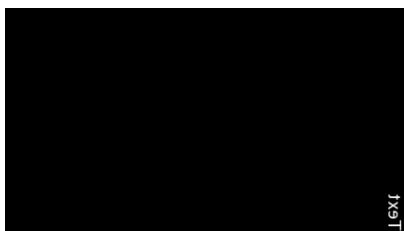
REG_ROTATE = 4 is mirrored landscape:



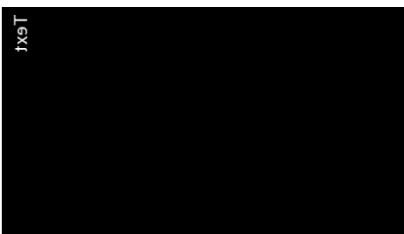
REG_ROTATE = 5 is mirrored inverted landscape:



REG_ROTATE = 6 is mirrored portrait:



REG_ROTATE = 7 is mirrored inverted portrait:



2.9.4 Drawing Pattern

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

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

Examples

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



```
//The VERTEX2F command gives the location of the
circle center
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) );
```

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



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

Draw rectangles with sizes of 5x25, 10x38 and 15x50 dimensions:

(Line width size is used for corner curvature, **LINE_WIDTH** pixels are added in both directions in addition to the rectangle dimension):



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

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) );
```

2.9.5 Bitmap Transformation Matrix

To achieve the bitmap transformation, the bitmap transform matrix below is specified and denoted as m :

$$m = \begin{bmatrix} \text{BITMAP_TRANSFORM_A} & \text{BITMAP_TRANSFORM_B} & \text{BITMAP_TRANSFORM_C} \\ \text{BITMAP_TRANSFORM_D} & \text{BITMAP_TRANSFORM_E} & \text{BITMAP_TRANSFORM_F} \end{bmatrix}$$

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

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

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

i.e.:

$$\begin{aligned} x' &= x * A + y * B + C \\ y' &= x * D + y * E + F \end{aligned}$$

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

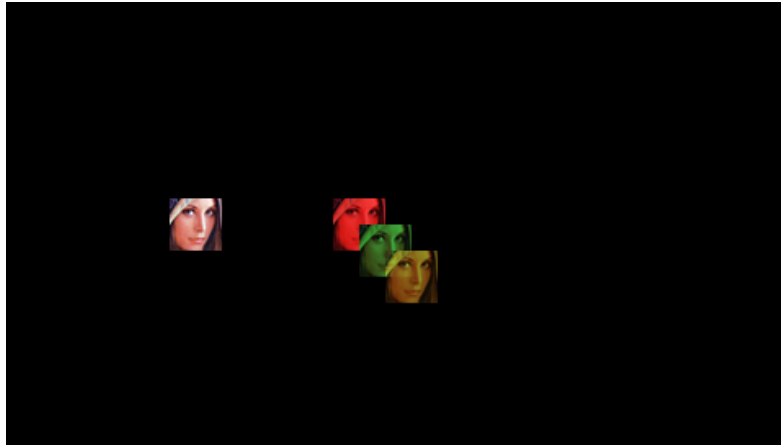
2.9.6 Color and Transparency

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

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

Code Snippet 9 – Color and Transparency

The **COLOR_RGB** command changes the current drawing color, which colors the bitmap. If it is omitted, the default color RGB (255,255,255) will be used to render the bitmap in its original colors. 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.



2.9.7 Bitmap Rendering Performance

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

Some performance limits:

- The display list length must be less than 2048 instructions, because the graphics engine fetches display list commands at a rate of one per clock.
- The usual performance of rendering pixels is 16 pixels per clock when the filter mode is in **NEAREST** mode, except for the following formats:
 - **TEXT8X8,**
 - **TEXTVGA**
 - **PALETTED4444/565**
 which renders 8 pixels per clock.
- If the filter mode is set to **BILINEAR**, the rendering pixel performance will be reduced to 25% of the performance achieved in **NEAREST** mode.

To summarize:

Filter Modes	Bitmap Formats	Drawing Rate
NEAREST	TEXT8X8, TEXTVGA, PALETTED4444/565	8 pixels per clock
	The rest formats	16 pixels per clock
BILINEAR	TEXT8X8, TEXTVGA, PALETTED4444/565	2 pixels per clock
	The rest formats	4 pixels per clock

Table 4 – Bitmap Rendering Performance

3 Register Description

The registers are classified into the following groups according to their functionality:

- Graphics Engine Registers,
- Audio Engine Registers,
- Touch Engine Registers,
- Coprocessor Engine Registers,
- Special Registers,
- Miscellaneous Registers.

The detailed definition for each register is listed here. Most of registers are **32** bit wide and the special cases are marked separately. Reading from or writing to the reserved bits shall be always **zero**.

The bit fields marked **r/o** are read-only.

The bit fields marked **w/o** are write only.

The bit fields marked **r/w** are read-write.

The offset of registers is based on the address **RAM_REG**.

3.1 Graphics Engine Registers

REG_TAG Definition			
31	8	7	0
Reserved			r/o
Offset: 0x7C			Reset Value: 0x0
Bit 31 – 8: Reserved bits			
Bit 7 – 0: These bits are updated with the tag value. The tag value here is corresponding to the touching point coordinator given in REG_TAG_X and REG_TAG_Y.			
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.			
REG_TOUCH_TAG is updated based on the current touching point captured from touch screen.			

Register Definition 1 – REG_TAG Definition

REG_TAG_Y Definition			
31	11	10	0
Reserved			r/w
Offset: 0x78			Reset Value: 0x0
Bit 31 – 11: Reserved Bits			
Bit 10 – 0: These bits are set by the host as the Y coordinate of the 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 where the host has already captured the touching point's coordinate; this register can be updated to query the tag value of respective touching point.			

Register Definition 2 – REG_TAG_Y Definition

REG_TAG_X Definition			
31	11	10	0
Reserved			r/w
Offset: 0x74			Reset Value: 0x0
Bit 31 – 11: Reserved Bits			
Bit 10 – 0: These bits are set by the host as the X coordinate of the touching point, which will enable the host to query the tag value. This register shall be used together with REG_TAG_Y and REG_TAG. Normally, in the case where the host has already captured the touching point's coordinate; this register can be updated to query the tag value of the respective touching point.			

Register Definition 3 – REG_TAG_X Definition

REG_PCLK Definition		
31	8 7	0
Reserved		r/w
Offset: 0x70		Reset Value: 0x0
Bit 31 – 8: Reserved bits		
Bit 7 – 0: These bits are set to divide the main clock for PCLK . If the main clock is 60Mhz and the value of these bits are set to 5, the PCLK will be set to 12 MHz If these bits are set to zero, it means there is no PCLK output.		

Register Definition 4 – REG_PCLK Definition

REG_PCLK_POL Definition		
31	1	0
reserved		r/w
Offset: 0x6C		Reset Value: 0x0
Bit 31 – 1: Reserved bits		
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		

Register Definition 5 – REG_PCLK_POL Definition

REG_CSPREAD Definition		
31	1	0
reserved		r/w
Offset: 0x68		Reset Value: 0x1
Bit 31 – 1: Reserved bits		
Bit 0: This bit controls the transition of RGB signals with PCLK active clock edge, which helps reduce the system noise. When it is zero, all the color signals are updated at the same time. When it is one, all the color signal timings are adjusted slightly so that fewer signals change simultaneously.		

Register Definition 6 – REG_CSPREAD Definition

REG_SWIZZLE Definition		
31	4 3	0
Reserved		r/w
Offset: 0x64		Reset Value: 0x0
Bit 31 – 4: Reserved bits		
Bit 3 – 0: These bits are set to control the arrangement of output RGB pins, which help support different LCD panels. Please see the datasheet for the exact definitions.		

Register Definition 7 – REG_SWIZZLE Definition

REG_DITHER Definition		
31	1	0
reserved		r/w
Offset: 0x60		Reset Value: 0x1
Bit 31 – 1: Reserved bits		
Bit 0: Set to 1 to enable dithering feature on RGB signals output. Set to 0 to disable dithering feature. Reading 1 from this bit means dithering feature is enabled. Reading 0 from this bit means dithering feature is disabled.		

Register Definition 8 – REG_DITHER Definition

REG_OUTBITS Definition			
31		9 8	0
Reserved		r/w	
Offset: 0x5C		Reset Value: 0x0	
Bit 31 – 9: Reserved Bits			

Bit 8 – 0: These 9 bits are split into 3 groups for Red, Green and Blue color output signals:
 Bit 8 – 6: Red Color signal lines number. Value zero means 8 output signals. 0b'111 means 7 output signals for red color.
 Bit 5 – 3: Green Color signal lines number. Value zero means 8 output signals. 0b'111 means 7 output signals for green color.
 Bit 2 – 0: Blue color signal lines number. Value zero means 8 output signals. 0b'111 means 7 output signals for blue color.
 Host can write these bits to control the number of output signals for each color:
 0b'000 : 8 output signals 0b'001 : 1 output signal
 0b'010 : 2 output signals 0b'011 : 3 output signals
 0b'100 : 4 output signals 0b'101 : 5 output signals
 0b'110 : 6 output signals 0b'111 : 7 output signals

Register Definition 9 – REG_OUTBITS Definition

REG_ROTATE Definition			
31	3	2	0
Reserved		r/w	
Offset: 0x58		Reset Value: 0x0	
Bit 31 – 3: Reserved bits			
Bit 2 – 0: screen rotation control bits.			
0b'000: Default landscape orientation			
0b'001: Inverted landscape orientation			
0b'010: Portrait orientation			
0b'011: Inverted portrait orientation			
0b'100: Mirrored landscape orientation			
0b'101: Mirrored invert landscape orientation			
0b'110: Mirrored portrait orientation			
0b'111: Mirrored inverted portrait orientation			
Note: Setting this register will NOT affect touch transform matrix.			

Register Definition 10 – REG_ROTATE Definition

REG_DLSWAP Definition			
31		21	0
Reserved			r/w
Offset: 0x54			Reset Value: 0x0
Bit 31 – 2: Reserved bits			
Bit 1 – 0: These bits can be set by the host to validate the display list buffer. The graphics engine will determine when to render the screen, depending on how these bits are set:			
0b'01: Graphics engine will render the screen immediately after current line is scanned out. It may cause tearing effect.			
0b'10: Graphics engine will render the screen immediately after current frame is scanned out.			
0b'00: Do not write this value into this register.			
0b'11: Do not write this value into this register.			
These bits can be also be read by the host to check the availability of the display list buffer. If the value is read as zero, the display list buffer is safe and ready to write. Otherwise, the host needs to wait till it becomes zero.			

Register Definition 11 – REG_DLSWAP Definition

REG_VSYNC1 Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x50		Reset Value: 0xA	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many lines of signal VSYNC1 takes at the start of a new frame			

Register Definition 12 – REG_VSYNC1 Definition

REG_VSYNC0 Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x4C		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many lines of signal VSYNC0 takes at the start of a new frame			

Register Definition 13 – REG_VSYNC0 Definition

REG_VSIZE Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x48		Reset Value: 0x110	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many lines of pixels in one frame. The valid range is from 0 to 2047.			

Register Definition 14 – REG_VSIZE Definition

REG_VOFFSET Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x44		Reset Value: 0xC	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many lines taken after the start of a new frame.			

Register Definition 15 – REG_VOFFSET Definition

REG_VCYCLE Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x40		Reset Value: 0x124	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many lines in one frame.			

Register Definition 16 – REG_VCYCLE Definition

REG_HSYNC1 Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x3C		Reset Value: 0x29	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many PCLK cycles for HSYNC1 during start of line.			

Register Definition 17 – REG_HSYNC1 Definition

REG_HSYNC0 Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x38		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: The value of these bits specifies how many PCLK cycles for HSYNC0 during start of line.			

Register Definition 18 – REG_HSYNC0 Definition

REG_HSIZE Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x34		Reset Value: 0x1E0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are used to specify the number of PCLK cycles per horizontal line.			

Register Definition 19 – REG_HSIZE Definition

REG_HOFFSET Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x30		Reset Value: 0x2B	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are used to specify the number of PCLK cycles before pixels are scanned out.			

Register Definition 20 – REG_HOFFSET Definition

REG_HCYCLE Definition			
31	12	11	0
Reserved		r/w	
Offset: 0x2C		Reset Value: 0x224	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are the number of total PCLK cycles per horizontal line scan.			

Register Definition 21 – REG_HCYCLE Definition

3.2 Audio Engine Registers

REG_PLAY Definition			
31		1	0
reserved			r/w
Offset: 0x8C			Reset Value: 0x0
Bit 31 – 1: Reserved bits			
Bit 0: A write to this bit triggers the play of the synthesized sound effect specified in REG_SOUND . Reading value 1 in this bit means the sound effect is playing. To stop the sound effect, the host needs to select the silence sound effect by setting up REG_SOUND and set this register to play.			

Register Definition 22 – REG_PLAY Definition

REG_SOUND Definition			
31		16	15
Reserved		r/w	
Offset: 0x88		Reset Value: 0x0	
Bit 31 – 16: Reserved bits			
Bit 15 – 0: These bits are used to select the synthesized sound effect. They are split into two groups: Bit 15 – 8 and Bit 7 – 0.			
Bit 15 – 8: The MIDI note for the sound effect defined in Bits 0 – 7.			
Bit 7 – 0: These bits define the sound effect. Some of them are pitch adjustable and the pitch is defined in Bits 8 – 15. Some of them are not pitch adjustable and the Bits 8 – 15 will be ignored.			
Note: Please refer to the section "Sound Synthesizer" in datasheet for details of this register.			

Register Definition 23 – REG_SOUND Definition

REG_VOL_SOUND Definition		
31	87	0
Reserved		r/w
Offset: 0x84		Reset Value: 0xFF
Bit 31 – 8: Reserved bits		
Bit 7 – 0: These bits control the volume of the synthesizer sound. The default value 0xFF is highest volume. The value zero means mute.		

Register Definition 24 – REG_VOL_SOUND Definition

REG_VOL_PB Definition		
31	87	0
Reserved		r/w
Offset: 0x84		Reset Value: 0xFF
Bit 31 – 8: Reserved bits		
Bit 7 – 0: These bits control the volume of the audio file playback. The default value 0xFF is highest volume. The value zero means mute.		

Register Definition 25 – REG_VOL_PB Definition

REG_PLAYBACK_PLAY Definition		
31	1	0
Reserved		r/w
Offset: 0xCC		Reset Value: 0x0
Bit 31 – 1: Reserved bits		
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 on-going, and 0 when playback completes.		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 26 – REG_PLAYBACK_PLAY Definition

REG_PLAYBACK_LOOP Definition		
31	1	0
Reserved		r/w
Offset: 0xC8		Reset Value: 0x0
Bit 31 – 1: Reserved bits		
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 section "Audio Playback" in datasheet for details of this register.		

Register Definition 27 – REG_PLAYBACK_LOOP Definition

REG_PLAYBACK_FORMAT Definition		
31	21	0
Reserved		r/w
Offset: 0xC4		Reset Value: 0x0
Bit 31 – 2: Reserved bits		
Bit 1 – 0: These bits define the format of the audio data in RAM_G .		
0b'00: Linear Sample format		
0b'01: uLaw Sample format		
0b'10: 4-bit IMA ADPCM Sample format		
0b'11: Undefined.		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 28 – REG_PLAYBACK_FORMAT Definition

REG_PLAYBACK_FREQ Definition		
31	16 15	0
Reserved		r/w
Offset: 0xC0		Reset Value: 0x1F40
Bit 31 – 16: Reserved bits		
Bit 15 – 0: These bits specify the sampling frequency of audio playback data. Unit is in Hz.		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 29 – REG_PLAYBACK_FREQ Definition

REG_PLAYBACK_READPTR Definition		
31	20 19	0
reserved		r/w
Offset: 0xBC		Reset Value: 0x0
Bit 31 – 20: Reserved bits		
Bit 19 – 0: These bits are updated by the 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. It should not exceed the size of RAM_G .		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 30 – REG_PLAYBACK_READPTR Definition

REG_PLAYBACK_LENGTH Definition		
31	20 19	0
reserved		r/w
Offset: 0xB8		Reset Value: 0x0
Bit 31 – 20: Reserved bits		
Bit 19 – 0: These bits specify the length of audio data in RAM_G to playback, starting from the address specified in REG_PLAYBACK_START register. It should not exceed the size of RAM_G .		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 31 – REG_PLAYBACK_LENGTH Definition

REG_PLAYBACK_START Definition		
31	20 19	0
reserved		r/w
Offset: 0xB4		Reset Value: 0x0
Bit 31 – 20: Reserved bits		
Bit 19 – 0: These bits specify the start address of audio data in RAM_G to playback. It should not exceed the size of RAM_G .		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 32 – REG_PLAYBACK_START Definition

REG_PLAYBACK_PAUSE Definition		
7	1 0	
reserved		r/w
Offset: 0x5EC		Reset Value: 0x0
Bit 7 – 1: Reserved bits		
Bit 0: Audio playback control bit. Writing 1 to pause the playback, writing 0 to start the playback.		
Note: Please refer to the section "Audio Playback" in datasheet for details of this register.		

Register Definition 33 – REG_PLAYBACK_PAUSE Definition

3.3 Touch Engine Registers

3.3.1 Overview

EVE supports screen touch functionality by either **Resistive Touch Screen Engine (RTSE)** or **Capacitive Touch Screen Engine (CTSE)**, depending on the package.

3.3.2 Common Registers

This chapter describes the common registers which are effective to both **RTSE** and **CTSE**.

Offset	Register Name	Description
0x150 – 0x164	REG_TOUCH_TRANSFORM_A~F	Transform coefficient matrix coefficient
0x168	REG_TOUCH_CONFIG	Configuration register

Table 5 – Common Registers Summary

REG_TOUCH_CONFIG Definition												
31	16	15	14	13	12	11	10	4	3	2	1	0
reserved			r/o	r/w	rsvd	r/w	r/w	r/w		r/w	r/w	r/w
Offset: 0x168						Reset Value: 0x8381 (RTSE) or 0x381(CTSE)						
Bit 31 – 16: Reserved bits												
Bit 15: Working mode of touch engine. 0: capacitive 1: resistive												
Bit 14: 1: Enable the host mode. 0: Normal mode												
Bit 13: Reserved bit												
Bit 12: Ignore short-circuit protection. For resistive touch screen only.												
Bit 11: Enable low-power mode(for FocalTech only)												
Bit 10 – 4: I2C address of capacitive touch screen module: 0b'0111000 for FocalTech/Hycontek 0b'1011101 for Goodix												
Bit 3: Reserved.												
Bit 2: Suppress 300ms startup (for FocalTech only)												
Bit 1 – 0: Sampling clocks(for resistive touch screen only)												

Register Definition 34 – REG_TOUCH_CONFIG Definition

REG_TOUCH_TRANSFORM_F Definition				
31	30	16	15	0
r/w	r/w		r/w	
Offset: 0x164			Reset Value: 0x0	
Bit 31 : The sign bit for fixed point number				
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.				
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.				
Note: This register represents a fixed-point number and the default value is +0.0 after reset.				

Register Definition 35 – REG_TOUCH_TRANSFORM_F Definition

REG_TOUCH_TRANSFORM_E Definition					
31	30		16	15	0
r/w	r/w		r/w		
Offset: 0x160			Reset Value: 0x10000		
Bit 31 : The sign bit for fixed point number					
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.					
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.					
Note: This register represents a fixed-point number and the default value is +1.0 after reset.					

Register Definition 36 – REG_TOUCH_TRANSFORM_E Definition

REG_TOUCH_TRANSFORM_D Definition			
31	30	16	15
			0
r/w	r/w		r/w
Offset: 0x15C		Reset Value: 0x0	
Bit 31: The sign bit for fixed point number			
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.			
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.			
Note: This register represents a fixed-point number and the default value is +0.0 after reset.			

Register Definition 37 – REG_TOUCH_TRANSFORM_D Definition

REG_TOUCH_TRANSFORM_C Definition			
31	30	16	15
			0
r/w	r/w		r/w
Offset: 0x158		Reset Value: 0x0	
Bit 31 : The sign bit for fixed point number			
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.			
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.			
Note: This register represents fixed point number and the default value is +0.0 after reset.			

Register Definition 38 – REG_TOUCH_TRANSFORM_C Definition

REG_TOUCH_TRANSFORM_B Definition			
31	30	16	15
			0
r/w	r/w		r/w
Offset: 0x154		Reset Value: 0x0	
Bit 31: The sign bit for fixed point number			
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.			
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.			
Note: This register represents a fixed-point number and the default value is +0.0 after reset.			

Register Definition 39 – REG_TOUCH_TRANSFORM_B Definition

REG_TOUCH_TRANSFORM_A Definition			
31	30	16	150
r/w	r/w	r/w	
Offset: 0x150		Reset Value: 0x10000	
Bit 31 : The sign bit for fixed point number			
Bit 30 – 16: The value of these bits represents the integer part of the fixed-point number.			
Bit 15 – 0: The value of these bits represents the fractional part of the fixed-point number.			
Note: This register represents a fixed-point number and the default value is +1.0 after reset.			

Register Definition 40 – REG_TOUCH_TRANSFORM_A Definition

3.3.3 Resistive Touch Screen Engine (RTSE)

When **EVE** has the built-in **RTSE**, the registers below are defined:

Offset	Register Name	Description
0x104	REG_TOUCH_MODE	Touch screen sampling Mode
0x108	REG_TOUCH_ADC_MODE	Select ADC working mode
0x10C	REG_TOUCH_CHARGE	Touch screen charge time, unit of 6 clocks
0x110	REG_TOUCH_SETTLE	Touch screen settle time, unit of 6 clocks
0x114	REG_TOUCH_OVERSAMPLE	Touch screen oversample factor
0x118	REG_TOUCH_RZTHRESH	Touch screen resistance threshold
0x11C	REG_TOUCH_RAW_XY	Touch screen raw x, y(16,16)
0x120	REG_TOUCH_RZ	Touch screen resistance
0x124	REG_TOUCH_SCREEN_XY	Touch screen x, y(16,16)
0x128	REG_TOUCH_TAG_XY	coordinate used to calculate the tag of touch point
0x12C	REG_TOUCH_TAG	Touch screen Tag result

Table 6 – RTSE Registers Summary

REG_TOUCH_TAG Definition			
31		8	70
Reserved			r/o
Offset: 0x12C			Reset Value: 0x0
Bit 31 – 8: Reserved Bits			
Bit v7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. These bits are updated once when all the lines of the current frame are scanned out to the screen.			
Note: The valid tag value is from 1 to 255 and the default value of this register is zero, meaning there is no touch by default.			

Register Definition 41 – REG_TOUCH_TAG Definition

REG_TOUCH_TAG_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x128		Reset Value: 0x0	
Bit 31 – 16: The value of these bits are X coordinates of the touch screen to look up the tag result.			
Bit 15 – 0: The value of these bits are the Y coordinates of the touch screen to look up the tag result.			
Note: Host can read this register to check the coordinates used by the touch engine to update the tag register REG_TOUCH_TAG .			

Register Definition 42 – REG_TOUCH_TAG_XY Definition

REG_TOUCH_SCREEN_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x124		Reset Value: 0x80008000	
Bit 31 – 16: The value of these bits is 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.			
Bit 15 – 0: The value of these bits is the Y 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.			

Register Definition 43 – REG_TOUCH_SCREEN_XY Definition

REG_TOUCH_DIRECT_Z1Z2 Definition															
31		26		25		16		15		10		9		0	
reserved				r/o				reserved				r/o			
Offset: 0x190												Reset Value: NA			
Bit 31 – 26: Reserved Bits															
Bit 25 – 16: 10-bit ADC value for touch screen resistance Z1															
Bit 15 – 10: Reserved Bits															
Bit 9 – 0: 10-bit ADC value for touch screen resistance Z2															
Note: To know it is touched or not, please check the 31 st bit of REG_TOUCH_DIRECT_XY . Touch engine will do the post-processing for these Z1 and Z2 values and update the result in REG_TOUCH_RZ .															

Register Definition 44 – REG_TOUCH_DIRECT_Z1Z2 Definition

REG_TOUCH_DIRECT_XY Definition								
31	30	26	25	16	15	10	9	0
r/o	reserved		r/o		reserved		r/o	
Offset: 0x18C						Reset Value: 0x0		
Bit 31: If this bit is zero, it means a touch is being sensed and the two fields above contain the sensed data. If this bit is one, it means no touch is being sensed and the data in the two fields above shall be ignored.								
Bit 30 – 26: Reserved Bits								
Bit 25 – 16: 10-bit ADC value for touch screen resistance Z1								
Bit 15 – 10: Reserved Bits								
Bit 9 – 0: 10-bit ADC value for touch screen resistance Z2								

Register Definition 45 – REG_TOUCH_DIRECT_XY

REG_TOUCH_RZ Definition			
31	16	15	0
Reserved		r/o	
Offset: 0x120			Reset Value: 0x7FFF
Bit 31 – 16: Reserved Bits			
Bit 15 – 0: These bits measure the touching pressure on the touch screen. The valid value is from 0 to 0x7FFF. The highest value(0x7FFF) means no touch and the lowest value (0) means the maximum touching pressure.			

Register Definition 46 – REG_TOUCH_RZ Definition

REG_TOUCH_RAW_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x11C		Reset Value: 0xFFFFFFFF	
Bit 31 – 16: These bits are the raw X coordinates before going through calibration process. The valid range is from 0 to 1023. If there is no touch on screen, the value shall be 0xFFFF.			
Bit 15 – 0: These bits are the raw Y coordinates of the touch screen before going through calibration process. The valid range is from 0 to 1023. If there is no touch on screen, the value shall be 0xFFFF.			
Note: The coordinates in this register have not mapped into the screen coordinates. To get the screen coordinates, please refer to REG_TOUCH_SCREEN_XY .			

Register Definition 47 – REG_TOUCH_RAW_XY Definition

REG_TOUCH_RZTHRESH Definition			
31	16	15	0
Reserved		r/w	
Offset: 0x118		Reset Value: 0xFFFF	
Bit 31 – 16: Reserved Bits.			
Bit 15 - 0: These bits control the touch screen resistance threshold. The host can adjust the touch screen touching sensitivity by setting this register. The default value after reset is 0xFFFF and it means the lightest touch will be accepted by the RTSE . The host can set this register by doing experiments. The typical value is 1200.			

Register Definition 48 – REG_TOUCH_RZTHRESH Definition

REG_TOUCH_OVERSAMPLE Definition			
31	4	3	0
reserved			r/w
Offset: 0x114			Reset Value: 0x7
Bit 31 – 4: Reserved Bits.			
Bit 3 – 0: These bits control the touch screen oversample factor. The higher value of this register causes more accuracy with more power consumption, but may not be necessary. The valid range is from 1 to 15.			

Register Definition 49 – REG_TOUCH_OVERSAMPLE Definition

REG_TOUCH_SETTLE Definition				
31		4	3	0
reserved			r/w	
Offset: 0x110			Reset Value: 0x3	
Bit 31 – 4: Reserved Bits.				
Bit 3 – 0: These bits control the touch screen settle time, in the unit of 6 clocks. The default value is 3, meaning the settle time is 18 (3*6) system clock cycles.				

Register Definition 50 – REG_TOUCH_SETTLE Definition

REG_TOUCH_CHARGE Definition			
31	16	15	0
Reserved		r/w	
Offset: 0x10C		Reset Value: 0x1770	
Bit 31 – 16: Reserved Bits.			
Bit 15 – 0: These bits control the touch screen charge time, in the unit of 6 clocks. The default value is 6000, meaning the charge time is (6000*6) system clock cycles.			

Register Definition 51 – REG_TOUCH_CHARGE Definition

REG_TOUCH_ADC_MODE Definition			
31		2	10
reserved			r/w
Offset: 0x108			Reset Value: 0x1
Bit 31 – 1: Reserved bits			
Bit 0: The host can set this bit to control the ADC sampling mode, 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.			

Register Definition 52 – REG_TOUCH_ADC_MODE Definition

REG_TOUCH_MODE Definition			
31		2	1 0
reserved			r/w
Offset: 0x104			Reset Value: 0x3
Bit 31 – 2: Reserved bits			
Bit 1 – 0: The host can set these two bits to control the touch screen sampling mode of touch engine, as per:			
0b'00: Off mode. No sampling happens. RTSE stops working.			
0b'01: Single mode. Cause one single sample to occur.			
0b'10: Frame mode. Cause a sample at the start of each frame.			
0b'11: Continuous mode. Up to 1000 times per seconds. Default mode after reset.			

Register Definition 53 – REG_TOUCH_MODE Definition

3.3.4 Capacitive Touch Screen Engine (CTSE)

Capacitive Touch Screen Engine (**CTSE**) has the following features:

- I²C interface to **Capacitive Touch Panel Module (CTPM)**
- Detects up to 5 touch points at the same time
- Supports **CTPM** with FocalTech and Goodix touch controller.
- Supports touch host mode. Please refer to the datasheet for details.
- Compatibility mode for single touching point and extended mode for multi-touching points.

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

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

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

Offset	Register Name	Description
0x104	REG_CTOUCH_MODE	Touch screen sampling Mode
0x108	REG_CTOUCH_EXTENDED	Select ADC working mode
0x11C	REG_CTOUCH_TOUCH1_XY	Coordinate of second touch point
0x120	REG_CTOUCH_TOUCH4_Y	Y coordinate of fifth touch point
0x124	REG_CTOUCH_TOUCH_XY	Coordinate of first touch point
0x128	REG_CTOUCH_TAG_XY	coordinate used to calculate the tag of first touch point
0x12C	REG_CTOUCH_TAG	Touch screen Tag result of first touch point
0x130	REG_CTOUCH_TAG1_XY	XY used to tag of second touch point
0x134	REG_CTOUCH_TAG1	Tag result of second touch point
0x138	REG_CTOUCH_TAG2_XY	XY used to tag of third touch point
0x13C	REG_CTOUCH_TAG2	Tag result of third touch point
0x140	REG_CTOUCH_TAG3_XY	XY used to tag of fourth touch point
0x144	REG_CTOUCH_TAG3	Tag result of fourth touch point
0x148	REG_CTOUCH_TAG4_XY	XY used to tag of fifth touch point
0x14C	REG_CTOUCH_TAG4	Tag result of fifth touch point
0x16C	REG_CTOUCH_TOUCH4_X	X coordinate of fifth touch point
0x18C	REG_CTOUCH_TOUCH2_XY	Third touch point coordinate
0x190	REG_CTOUCH_TOUCH3_XY	Fourth touch point coordinate

Table 7 – CTSE Registers Summary

The following tables define the registers provided by **CTSE**:

REG_CTOUCH_MODE Definition			
31		2	1 0
Reserved			r/w
Offset: 0x104			Reset Value: 0x3
Bit 31 – 2: Reserved bits			
Bit 1 – 0: The host can set these two bits to control the touch screen sampling mode of the touch engine, as per:			
0b'00: Off mode. No sampling happens. CTSE stops working.			
0b'01: Not defined.			
0b'10: Not defined.			
0b'11: On mode.			

Register Definition 54 – REG_CTOUCH_MODE Definition

REG_CTOUCH_EXTEND Definition		
31	1	0
reserved	r/w	
Offset: 0x108		Reset Value: 0x1
Bit 31 – 1: Reserved bits		
Bit 0: This bit controls the detection mode of the touch engine, as per:		
0: Extended mode, multi-touch detection mode		
1: Compatibility mode, single touch detection mode		

Register Definition 55 – REG_CTOUCH_EXTENDED Definition

REG_CTOUCH_TOUCH_XY Definition			
31		16	150
r/o		r/o	
Offset: 0x124		Reset Value: 0x80008000	
Bit 31 – 16: The value of these bits is X coordinate of the first touch point			
Bit 15 - 0: The value of these bits is Y coordinate of the first touch point.			
Note: This register is applicable for extended mode and compatibility mode. For compatibility mode, it reflects the position of the only touch point			

Register Definition 56 – REG_CTOUCH_TOUCH_XY Definition

REG_CTOUCH_TOUCH1_XY Definition			
31		16	15
			0
r/o		r/o	
Offset: 0x11C		Reset Value: 0x80008000	
Bit 31 – 16: The value of these bits is X coordinate of the second touch point			
Bit 15 – 0: The value of these bits is Y coordinate of the second touch point.			
Note: This register is only applicable in the extended mode			

Register Definition 57 – REG_CTOUCH_TOUCH1_XY Definition

REG_CTOUCH_TOUCH2_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x18C		Reset Value: 0x80008000	
Bit 31 – 16: The value of these bits is X coordinates of the third touch point			
Bit 15 - 0: The value of these bits is Y coordinates of the third touch point.			
Note: This register is only applicable in the extended mode			

Register Definition 58 – REG_CTOUCH_TOUCH2_XY Definition

REG_CTOUCH_TOUCH3_XY Definition		
31	16	15 0
r/o		r/o
Offset: 0x190		Reset Value: 0x80008000
Bit 31 – 16: The value of these bits is X coordinate of the fourth touch point		
Bit 15 – 0: The value of these bits is Y coordinate of the fourth touch point.		
Note: This register is only applicable in the extended mode		

Register Definition 59 – REG_CTOUCH_TOUCH3_XY Definition

REG_CTOUCH_TOUCH4_X Definition		
15		0
r/o		
Offset: 0x16C		Reset Value: 0x8000
Bit 15 – 0: The value of these bits is X coordinate of the fifth touch point.		
Note: This register is only applicable in the extended mode. It is a 16-bit register.		

Register Definition 60 – REG_CTOUCH_TOUCH4_X Definition

REG_CTOUCH_TOUCH4_Y Definition		
15		0
r/o		
Offset: 0x120		Reset Value: 0x8000
Bit 15 – 0: The value of these bits is Y coordinate of the fifth touch point.		
Note: This register is only applicable in the extended mode. It is a 16-bit register.		

Register Definition 61 – REG_CTOUCH_TOUCH4_Y Definition

REG_CTOUCH_RAW_XY Definition		
31	16	15 0
r/o		r/o
Offset: 0x11C		Reset Value: 0xFFFFFFFF
Bit 31 – 16: The value of these bits is the X coordinate of a touch point before going through calibration process		
Bit 15 – 0: The value of these bits is the Y coordinate of a touch point before going through calibration process		
Note: This register is only applicable in the compatibility mode		

Register Definition 62 – REG_CTOUCH_RAW_XY Definition

REG_CTOUCH_TAG Definition		
31	8	7 0
Reserved		r/o
Offset: 0x12C		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. These bits are updated once when all the lines of the current frame are scanned out to the screen. It works in both extended mode and compatibility mode. In extended mode, it is the tag of the first touch point, i.e., the tag value mapping to the coordinate in		
REG_CTOUCH_TAG_XY		
Note: The valid tag value range is from 1 to 255, therefore the default value of this register is zero, meaning there is no touch by default. In extended mode, it refers to the first touch point		

Register Definition 63 – REG_CTOUCH_TAG Definition

REG_CTOUCH_TAG1 Definition		
31	87	0
Reserved		r/o
Offset: 0x134		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the second touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.		
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. This register is only applicable in the extended mode.		

Register Definition 64 – REG_CTOUCH_TAG1 Definition

REG_CTOUCH_TAG2 Definition		
31	87	0
Reserved		r/o
Offset: 0x13C		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the third touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.		
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. This register is only applicable in the extended mode.		

Register Definition 65 – REG_CTOUCH_TAG2 Definition

REG_CTOUCH_TAG3 Definition		
31	87	0
Reserved		r/o
Offset: 0x144		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the fourth touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.		
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. This register is only applicable in the extended mode.		

Register Definition 66 – REG_CTOUCH_TAG3 Definition

REG_CTOUCH_TAG4 Definition		
31	87	0
Reserved		r/o
Offset: 0x14C		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are set as the tag value of the specific graphics object on the screen which is being touched. It is the fifth touch point in extended mode. These bits are updated once when all the lines of the current frame are scanned out to the screen.		
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. This register is only applicable in the extended mode.		

Register Definition 67 – REG_CTOUCH_TAG4 Definition

REG_CTOUCH_TAG_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x128		Reset Value: 0x0	
Bit 31 – 16: The value of these bits is X coordinate of the touch screen, used by the touch engine to look up the tag result.			
Bit 15 - 0: The value of these bits is Y coordinate of the touch screen, used by the touch engine to look up the tag result.			
Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG.			

Register Definition 68 – REG_CTOUCH_TAG_XY Definition

REG_CTOUCH_TAG1_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x130		Reset Value: 0x0	
Bit 31 – 16: The value of these bits is X coordinate of the touch screen to look up the tag result.			
Bit 15 – 0: The value of these bits is Y coordinate of the touch screen to look up the tag result.			
Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG1.			

Register Definition 69 – REG_CTOUCH_TAG1_XY Definition

REG_CTOUCH_TAG2_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x138		Reset Value: 0x0	
Bit 31 – 16: The value of these bits is X coordinate of the touch screen to look up the tag result.			
Bit 15 - 0: The value of these bits is Y coordinate of the touch screen to look up the tag result.			
Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG2.			

Register Definition 70 – REG_CTOUCH_TAG2_XY Definition

REG_CTOUCH_TAG3_XY Definition			
31		16	15
r/o		r/o	
Offset: 0x140		Reset Value: 0x0	
Bit 31 – 16: The value of these bits is X coordinate of the touch screen to look up the tag result.			
Bit 15 – 0: The value of these bits is Y coordinate of the touch screen to look up the tag result.			
Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG3.			

Register Definition 71 – REG_CTOUCH_TAG3_XY Definition

REG_CTOUCH_TAG4_XY Definition			
31	16	15	0
r/o		r/o	
Offset: 0x148		Reset Value: 0x0	
Bit 31 – 16: The value of these bits is X coordinate of the touch screen to look up the tag result.			
Bit 15 – 0: The value of these bits is Y coordinate of the touch screen to look up the tag result.			
Note: The Host can read this register to check the coordinates used by the touch engine to update the tag register REG_CTOUCH_TAG4.			

Register Definition 72 – REG_CTOUCH_TAG4_XY Definition

3.3.5 Calibration

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

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

3.4 Coprocessor Engine Registers

REG_CMD_DL Definition			
31	13	12	0
Reserved		r/w	
Offset: 0x100		Reset Value: 0x0	
Bit 31 – 13: Reserved Bits			
Bit 12 – 0: These bits indicate the offset from RAM_DL of the display list commands generated by the coprocessor engine. The coprocessor engine depends on these bits to determine the address in the display list buffer of generated display list commands. It will update this register as long as the display list commands are generated into the display list buffer. By setting this register properly, the host can specify the starting address in the display list buffer for the coprocessor engine to generate display commands. The valid value range is from 0 to 8191 (sizeof(RAM_DL)-1).			

Register Definition 73 – REG_CMD_DL Definition

REG_CMD_WRITE Definition			
31	12	11	0
Reserved		r/w	
Offset: 0xFC		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are updated by the host 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: The FIFO size of the command buffer is 4096 bytes and each coprocessor instruction is of 4 bytes in size. The value to be written into this register must be 4 bytes aligned.			

Register Definition 74 – REG_CMD_WRITE Definition

REG_CMD_READ Definition			
31	12	11	0
Reserved		r/o	
Offset: 0xF8		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are updated by the coprocessor engine as long as the coprocessor engine fetched the command from its FIFO. The host can read this register to determine the FIFO fullness of the coprocessor engine. The valid value range is from 0 to 4095. In the case of an error, the coprocessor engine writes 0xFFFF to this register.			
Note: The host shall not write into this register unless in an error recovery case. The default value is zero after the coprocessor engine is reset.			

Register Definition 75 – REG_CMD_READ Definition

REG_CMDB_SPACE Definition			
31	12	11	0
Reserved		r/o	
Offset: 0x574		Reset Value: 0xFFC	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are updated by the coprocessor engine to indicate the free space in RAM_CMD. The host can read this register to determine how many bytes are available to be written into RAM_CMD before writing to RAM_CMD .			
Note: The host shall not write into this register unless in an error recovery case. The default value is zero after the coprocessor engine is reset.			

Register Definition 76 – REG_CMDB_SPACE Definition

REG_CMDB_WRITE Definition	
31	0
w/o	
Offset: 0x578	Reset Value: 0x0
Bit 31 – 0: The data or command to be written into RAM_CMD . The Host can issue one write transfer with this register address to transfer data less than or equal to the amount of REG_CMDB_SPACE to make bulky data transfer possible.	
Note: This register helps programmers write to the coprocessor FIFO(RAM_CMD) . It was introduced from the FT810 series chip. Always write this register with 4 bytes aligned data.	

Register Definition 77 – REG_CMDB_WRITE Definition

3.5 Miscellaneous Registers

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

REG_CPURESET Definition				
31	3	2	1	0
reserved			r/w	
Offset: 0x20			Reset Value: 0x0	
Bit 31 – 3: Reserved Bits				
Bit 2: Control the reset of audio engine.				
Bit 1: Control the reset of touch engine.				
Bit 0: Control the reset of coprocessor engine.				
Note: Write 1 to reset the corresponding engine. Write 0 to go back to normal working status. Reading 1 means the engine is in reset status, and reading zero means the engine is in working status.				

Register Definition 78 – REG_CPURESET Definition

REG_MACRO_1 Definition	
31	0
r/w	
Offset: 0xDC	Reset Value: 0x0
Bit 31 – 0: Display list command macro-1. The value of this register will be copied over to RAM_DL to replace the display list command MACRO if its parameter is 1.	

Register Definition 79 – REG_MACRO_1 Definition

REG_MACRO_0 Definition	
31	0
r/w	
Offset: 0xD8	Reset Value: 0x0
Bit 31 – 0: Display list command macro-0. The value of this register will be copied over to RAM_DL to replace the display list command MACRO if its parameter is 0.	

Register Definition 80 – REG_MACRO_0 Definition

REG_PWM_DUTY Definition		
31	8 7	0
Reserved		r/w
Offset: 0xD4		Reset Value: 0x80
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits define the backlight PWM output duty cycle. The valid range is from 0 to 128. 0 means backlight completely off, 128 means backlight in max brightness.		

Register Definition 81 – REG_PWM_DUTY Definition

REG_PWM_HZ Definition		
31	14 13	0
Reserved		r/w
Offset: 0xD0		Reset Value: 0xFA
Bit 31 – 14: Reserved Bits		
Bit 13 – 0: These bits define the backlight PWM output frequency in HZ . The default is 250 Hz after reset. The valid frequency is from 250Hz to 10000Hz.		

Register Definition 82 – REG_PWM_HZ Definition

REG_INT_MASK Definition		
31	8 7	0
Reserved		r/w
Offset: 0xB0		Reset Value: 0xFF
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are used to mask the corresponding interrupt. 1 means to enable the corresponding interrupt source, 0 means to disable the corresponding interrupt source. After reset, all the interrupt source are eligible to trigger an interrupt by default.		
Note: Refer to the section "Interrupts" in datasheet for more details.		

Register Definition 83 – REG_INT_MASK Definition

REG_INT_EN Definition		
31	1	0
Reserved		r/w
Offset: 0xAC		Reset Value: 0x0
Bit 31 – 1: Reserved bits		
Bit 0: The host can set this bit to 1 to enable the global interrupt. To disable the global interrupt, the host can set this bit to 0.		
Note: Refer to the section "Interrupts" in datasheet for more details.		

Register Definition 84 – REG_INT_EN Definition

REG_INT_FLAGS Definition		
31	8 7	0
Reserved		r/w
Offset: 0xA8		Reset Value: 0x0
Bit 31 – 8: Reserved Bits		
Bit 7 – 0: These bits are interrupt flags. The host can read these bits to determine which interrupt takes place. These bits are cleared automatically by reading. The host shall not write to this register.		
Note: Refer to the section "Interrupts" in datasheet for more details.		

Register Definition 85 – REG_INT_FLAGS Definition

REG_GPIO_DIR Definition									
31	8	7	6	2	1	0			
Reserved		r/w	reserved		r/w				
Offset: 0x90				Reset Value: 0x0					
Bit 31 – 8: Reserved Bits									
Bit 7: The direction of pin DISP : 1 for output, 0 for input									
Bit 6 – 2: Reserved Bits									
Bit 1: The direction of pin GPIO1 : 1 for output, 0 for input									
Bit 0: The direction of pin GPIO0 : 1 for output, 0 for input									

Register Definition 86 – REG_GPIO_DIR Definition

REG_GPIO Definition									
31	8	7	6	5	4	3	2	1	0
Reserved				r/w					
Offset: 0x94				Reset Value: 0x0					
Bit 31 – 8: Reserved Bits									
Bit 7: It controls the high or low level of pin DISP .									
Bit 6-5: Drive strength settings for pins GPIO0,GPIO1, CTP_RST_N : 0b'00:5mA – default, 0b'01:10mA, 0b'10:15mA, 0b'11:20mA									
Bit 4: Drive strength settings for pins PCLK, DISP,VSYNC,HSYNC,DE, R,G,B, BACKLIGHT : 0b'0: 1.2mA – default, 0b'1: 2.4mA									
Bit 3-2: Drive Strength Setting for pins MISO, MOSI, INT_N : 0b'00:5mA – default, 0b'01:10mA, 0b'10:15mA, 0b'11:20mA									
Bit 1: It controls the high or low level of pin GPIO1 .									
Bit 0: It controls the high or low level of pin GPIO0 .									

Register Definition 87 – REG_GPIO Definition

REG_GPIOX_DIR Definition											
31				16	15	14			4	3	0
Reserved					r/w		reserved			r/w	
Offset: 0x98							Reset Value: 0x8000				
Bit 31 – 16: Reserved Bits											
Bit 15: It controls the direction of pin DISP . The default value is 1, meaning output.											
Bit 14 – 4: Reserved Bits											
Bit 3: It controls the direction of GPIO3 .											
Bit 2: It controls the direction of GPIO2 .											
Bit 1: It controls the direction of GPIO1 .											
Bit 0: It controls the direction of GPIO0 .											
Note: 1 is for output,0 is for input direction											

Register Definition 88 – REG_GPIOX_DIR Definition

REG_GPIOX Definition												
31	16	15	14	13	12	11	10	9	8	4	3	0
Reserved				r/w					reserved		r/w	
Offset: 0x9C										Reset Value: 0x8000		
Bit 31 – 16: Reserved Bits												
Bit 15: The high or low level of pin DISP . 1 for high level (default) and 0 for low level.												
Bit 14-13: Drive strength settings for pins GPIO0,GPIO1,GPIO2,GPIO3, CTP_RST_N : 0b'00:5mA – default, 0b'01:10mA, 0b'10:15mA, 0b'11:20mA												
Bit 12: Drive strength settings for pins PCLK, DISP,VSYNC,HSYNC,DE, R,G,B, BACKLIGHT : 0b'0: 5mA – default, 0b'1: 10mA												
Bit 11-10: Drive Strength Setting for pins MISO, MOSI, INT_N,IO2, IO3 : 0b'00:5mA – default, 0b'01:10mA, 0b'10:15mA, 0b'11:20mA												
Bit 9: It controls the type of pin INT_N . 0b'0: Open Drain – default, 0b'1: Push-pull												
Bit 8 – 4: Reserved Bits												
Bit 3: The high or low level of pin GPIO3 .												
Bit 2: The high or low level of pin GPIO2 .												
Bit 1: The high or low level of pin GPIO1 .												
Bit 0: The high or low level of pin GPIO0 .												
Note: Refer to datasheet for more details.												

Register Definition 89 – REG_GPIOX Definition

REG_FREQUENCY Definition	
31	0
r/w	
Offset: 0xC	Reset Value: 0x3938700
Bit 31 – 0: The main clock frequency is 60MHz by default. The value is in Hz. If the host selects the alternative frequency, this register must be updated accordingly.	

Register Definition 90 – REG_FREQUENCY Definition

REG_CLOCK Definition	
31	0
r/o	
Offset: 0x8	Reset Value: 0x0
Bit 31 – 0: These bits are set to zero after reset. The register counts the number of main clock cycles since reset. If the main clock's frequency is 60Mhz, it will wrap around after about 71 seconds.	

Register Definition 91 – REG_CLOCK Definition

REG_FRAMES Definition	
31	0
r/o	
Offset: 0x4	Reset Value: 0x0
Bit 31 – 0: These bits are set to zero after reset. The register counts the number of screen frames. If the refresh rate is 60Hz, it will wrap up till about 828 days after reset.	

Register Definition 92 – REG_FRAMES Definition

REG_ID Definition			
31	8	7	0
Reserved		r/o	
Offset: 0x0		Reset Value: 0x7C	
Bit 31 – 8: Reserved Bits			
Bit 7 – 0: These bits are the built-in ID of the chip. The value shall always be 0x7C . The host can read this to determine if the chip belongs to the EVE series and is in working mode after booting up.			
Register Definition 93 – REG_ID Definition			

REG_SPI_WIDTH Definition			
31	3	2	1 0
Reserved			r/w
Offset: 0x188			Reset Value: 0x0
Bit 31 – 3: Reserved Bits			
Bit 2: Extra dummy on SPI read transfer. Writing 1 enables one extra dummy byte on SPI read transfer.			
Bit 1 – 0: SPI data bus width: 0b'00: 1 bit – (Single-SPI), default 0b'01: 2 bit (Dual-SPI) 0b'10: 4 bit (Quad-SPI) 0b'11: undefined			
Note: Refer to datasheet for more details.			
Register Definition 94 – REG_SPI_WIDTH Definition			

3.6 Special Registers

The registers listed here are not located in **RAM_REG**. They are located in special addresses.

REG_TRACKER Definition			
31	16	15	870
r/o		reserved	r/o
Offset: 0x7000		Reset Value: 0x0	
Bit 31 – 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.			
Bit 15 – 8: Reserved Bits			
Bit 7 - 0: These bits are set to indicate the tag value of a graphics object which is being touched.			
Register Definition 95 – REG_TRACKER Definition			

REG_TRACKER_1 Definition			
31	16	15	870
r/o		reserved	r/o
Offset: 0x7004		Reset Value: 0x0	
Bit 31 – 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.			
Bit 15 – 8: Reserved Bits			
Bit 7 – 0: These bits are set to indicate the tag value of a graphics object which is being touched as the second point.			
Note: It is only applicable for extended mode of CTSE.			
Register Definition 96 – REG_TRACKER_1 Definition			

REG_TRACKER_2 Definition			
31	16	15	870
r/o	reserved		r/o
Offset: 0x7008			Reset Value: 0x0
Bit 31 – 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.			
Bit 15 – 8: Reserved Bits			
Bit 7 – 0: These bits are set to indicate the tag value of a graphics object which is being touched as the third point.			
Note: It is only applicable for extended mode of CTSE.			

Register Definition 97 – REG_TRACKER_2 Definition

REG_TRACKER_3 Definition			
31	16	15	870
r/o	reserved		r/o
Offset: 0x700C			Reset Value: 0x0
Bit 31 – 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.			
Bit 15 – 8: Reserved Bits			
Bit 7 – 0: These bits are set to indicate the tag value of a graphics object which is being touched as the fourth point.			
Note: It is only applicable for extended mode of CTSE.			

Register Definition 98 – REG_TRACKER_3 Definition

REG_TRACKER_4 Definition		
31	16	15
	8	7
	0	
r/o	reserved	r/o
Offset: 0x7010		Reset Value: 0x0
Bit 31 – 16: These bits are set to indicate the tracking value for the tracked graphics objects. The coprocessor calculates the tracking value that the touching point takes within the predefined range. Please check the CMD_TRACK for more details.		
Bit 15 – 8: Reserved Bits		
Bit 7 – 0: These bits are set to indicate the tag value of a graphics object which is being touched as the fifth point.		
Note: It is only applicable for extended mode of CTSE.		

Register Definition 99 – REG_TRACKER_4 Definition

REG_MEDIAFIFO_READ Definition	
31	0
r/o	
Offset: 0x7014	Reset Value: 0x0
Bit 31 – 0: The value specifies the read pointer pointing to the address in RAM_G as the media FIFO .	

Register Definition 100 – REG_MEDIAFIFO_READ Definition

REG_MEDIAFIFO_WRITE Definition	
31	0
w/o	
Offset: 0x7018	Reset Value: 0x0
Bit 31 – 0: The value specifies the write pointer pointing to the address in RAM_G as the media FIFO .	

Register Definition 101 – REG_MEDIAFIFO_WRITE Definition

4 Display List Commands

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

4.1 Graphics State

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

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

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

Parameters	Default values	Commands
func & ref	ALWAYS, 0	ALPHA_FUNC
func & ref	ALWAYS, 0	STENCIL_FUNC
Src & dst	SRC_ALPHA, ONE_MINUS_SRC_ALPHA	BLEND_FUNC
Cell value	0	CELL
Alpha value	0	COLOR_A
Red, Blue, Green colors	(255,255,255)	COLOR_RGB
Line width in 1/16 pixels	16	LINE_WIDTH
Point size in 1/16 pixels	16	POINT_SIZE
Width & height of scissor	HSIZE,2048	SCISSOR_SIZE
Starting coordinates of scissor	(x, y) = (0,0)	SCISSOR_XY
Current bitmap handle	0	BITMAP_HANDLE
Bitmap transform coefficients	+1.0,0,0,0,+1.0,0	BITMAP_TRANSFORM_A-F
Stencil clear value	0	CLEAR_STENCIL
Tag clear value	0	CLEAR_TAG
Mask value of stencil	255	STENCIL_MASK
spass and sfail	KEEP,KEEP	STENCIL_OP
Tag buffer value	255	TAG
Tag mask value	1	TAG_MASK
Alpha clear value	0	CLEAR_COLOR_A
RGB clear color	(0,0,0)	CLEAR_COLOR_RGB
Palette source address	RAM_G	PALETTE_SOURCE
Units of pixel precision	1/16 pixel	VERTEX_FORMAT, VERTEX2F

Table 8 – Graphics Context

4.2 Command Encoding

Each display list command has a 32-bit encoding. The most significant 8 bits of the code determine the command. Command parameters (if any) are present in the least significant bits. Any bits marked as “reserved” must be zero.

4.3 Command Groups

4.3.1 Setting Graphics State

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

4.3.2 Drawing Actions

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

4.3.3 Execution Control

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

4.4 ALPHA_FUNC

Specify the alpha test function

Encoding

31	24	23	11	10	8	7	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

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	0
0x1F		reserved			prim

Parameters

prim

The graphics primitive to be executed. The valid values are defined as below:

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

Table 9 – Graphics Primitive Definition

Description

All primitives supported 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:



```
d1( BEGIN(POINTS) );
d1( VERTEX2II(50, 5, 0, 0) );
d1( VERTEX2II(110, 15, 0, 0) );
d1( BEGIN(LINES) );
d1( VERTEX2II(50, 45, 0, 0) );
d1( VERTEX2II(110, 55, 0, 0) );
d1( BEGIN(BITMAPS) );
d1( VERTEX2II(50, 65, 31, 0x45) );
d1( 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	0
0x05			reserved		handle

Parameters

handle

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

Description

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

Graphics context

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

See also

[BITMAP_LAYOUT](#), [BITMAP_SIZE](#)

4.7 BITMAP_LAYOUT

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

Encoding

31	24	23	19	18	9	8	0
0x07		format		linstride		height	

Parameters

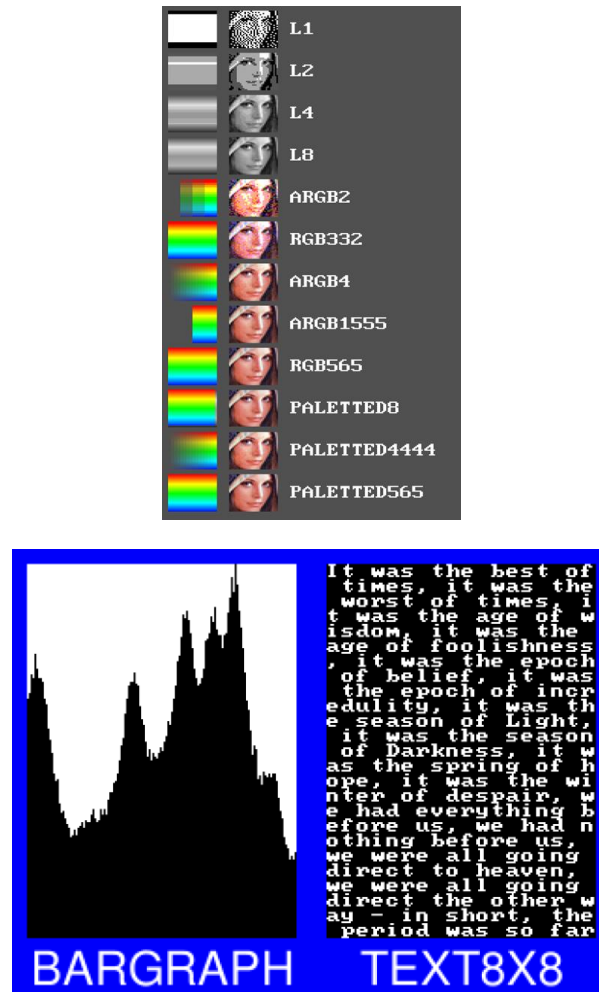
format

Bitmap pixel format. The valid range is from 0 to 17 and defined as per the table below.

Name	Value	Bits/pixel	Alpha bits	Red bits	Green bits	Blue bits
ARGB1555	0	16	1	5	5	5
L1	1	1	1	0	0	0
L4	2	4	4	0	0	0
L8	3	8	8	0	0	0
RGB332	4	8	0	3	3	2
ARGB2	5	8	2	2	2	2
ARGB4	6	16	4	4	4	4
RGB565	7	16	0	5	6	5
TEXT8X8	9	-	-	-	-	-
TEXTVGA	10	-	-	-	-	-
BARGRAPH	11	-	-	-	-	-
PALETTE565	14	8	0	5	6	5
PALETTE4444	15	8	4	4	4	4
PALETTE8	16	8	8	8	8	8
L2	17	2	2	0	0	0

Table 10 – BITMAP_LAYOUT Format List

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



BARGRAPH – render data as a bar graph. Looks up the x coordinate in a byte array, then gives an opaque pixel if the byte value is less than y, otherwise a transparent pixel. The result is a bar graph of the bitmap data. A maximum of screen widthx256 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*¹ font (built-in bitmap handles 16 & 17 are used for drawing TEXT8X8 format). The result is that the bitmap acts like a character grid. A single bitmap can be drawn which covers all or part of the display; each byte in the bitmap data corresponds to one 8x8 pixel character cell.

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

¹ https://en.wikipedia.org/wiki/Code_page_437

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

For **L1, L2, L4** format, the necessary data has to be padded to make it byte aligned. Normally, it can be calculated with the following formula:

$$\text{linestride} = \text{width} * \text{byte/pixel}$$

For example, if one bitmap is 64x32 pixels in L4 format, the line stride shall be
 $(64 * 1/2 = 32)$

height - Bitmap height, in lines

Description

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

L1 Format							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pixel 0	Pixel 1	Pixel 2	Pixel 3	Pixel 4	Pixel 5	Pixel 6	Pixel 7

L2 Format							
7	6	5	4	3	2	1	0
Pixel 0		Pixel 1		Pixel 2		Pixel 3	

L4 Format							
7	6	5	4	3	2	1	0
Pixel 0				Pixel 1			

L8 Format							
7	6	5	4	3	2	1	0
Pixel 0							

Table 11 – L1/L2/L4/L8 Pixel Format

ARGB2 Format							
7	6	5	4	3	2	1	0
Alpha Channel		Red Channel		Green Channel		Blue Channel	

RGB332 Format							
7	6	5	4	3	2	1	0
Red Channel		Green Channel		Blue Channel			

Table 12 – ARGB2/RGB332 Pixel Format

RGB565/PALETTED565 Format							
15	14	13	12	11	10	9	8
Red Channel				Green Channel		Blue Channel	

Table 13 – RGB565/PALETTED565 Pixel Format

ARGB1555 Format							
15	14	13	12	11	10	9	8
Alpha Channel		Red Channel		Green Channel		Blue Channel	

ARGB4/PALETTED4444 Format							
15	12	11	8	7	4	3	0
Alpha Channel		Red Channel		Green Channel		Blue Channel	

Table 14 – ARGB1555/ARGB4/PALETTED4444 Pixel Format

PALETTED8 Format							
31	24	23	16	15	8	7	0
Alpha Channel		Red Channel		Green Channel		Blue Channel	

Table 15 – PALETTED8 Pixel Format

Note: PALETTED8 is 8 bits per pixel as each pixel is represented by an 8-bit index value in the look-up table. It has a color depth of 24-bits and 8-bit alpha.

Graphics Context

None

Note: PALETTED8 format is supported indirectly and it is different from PALETTED format in FT80x. To render Alpha, Red, Green and Blue channels, multi-pass drawing action is required.

The following display list snippet shows:

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

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

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

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

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

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

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

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

Code Snippet 10 – PALETTED8 Drawing Example

See also

[BITMAP_HANDLE](#), [BITMAP_SIZE](#), [BITMAP_SOURCE](#), [PALETTE_SOURCE](#)

4.8 BITMAP_LAYOUT_H

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

Encoding

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

Parameters

linestride

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

height

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

Description

This command is the extension command of **BITMAP_LAYOUT** for bitmap larger than 511 by 511 pixels.

Examples

NA

See also

[BITMAP_LAYOUT](#)

4.9 BITMAP_SIZE

Specify the screen drawing of bitmaps for the current handle

Encoding

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

Parameters

filter

Bitmap filtering mode, **NEAREST** or **BILINEAR**.
 The value of **NEAREST** is 0 and the value of **BILINEAR** is 1.

wrapx

Bitmap x wrap mode, **REPEAT** or **BORDER**.
 The value of **BORDER** is 0 and the value of **REPEAT** is 1.

wrappy

Bitmap y wrap mode, **REPEAT** or **BORDER**

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

width

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

height

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

Description

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

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

4.10 BITMAP_SIZE_H

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

Encoding

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

Parameters

width

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

Height

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

Description

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

Graphics context

None

See also

[BITMAP_HANDLE](#), [BITMAP_LAYOUT](#), [BITMAP_SOURCE](#), [BITMAP_SIZE](#)

4.11 BITMAP_SOURCE

Specify the source address of bitmap data in **RAM_G**.

Encoding

31	24	23	22	21	0
0x01		reserved			addr

Parameters

addr

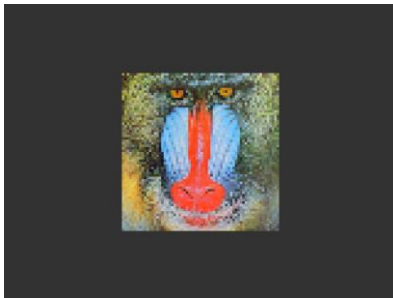
Bitmap address in **RAM_G**, aligned with respect to the bitmap format.
 For example, if the bitmap format is **RGB565/ARGB4/ARGB1555**, the bitmap source shall be aligned to 2 bytes.

Description

The bitmap source address specifies the address of the bitmap graphic data.

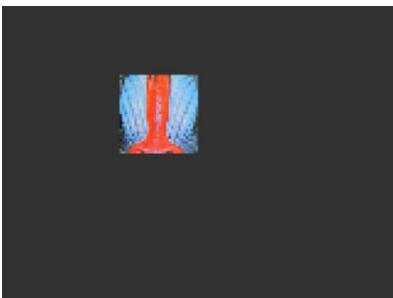
Examples

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



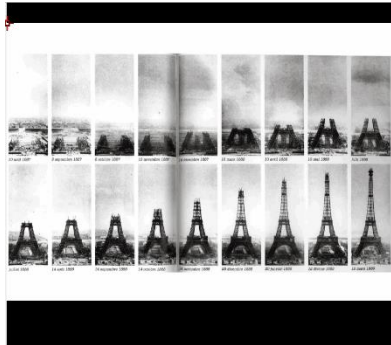
```
dl( BITMAP_SOURCE(0) );
dl( BITMAP_LAYOUT( RGB565, 128, 64 ) );
dl( BITMAP_SIZE( NEAREST, BORDER, BORDER, 64, 64 ) );
dl( BEGIN( BITMAPS ) );
dl( VERTEX2II( 48, 28, 0, 0 ) );
```

Using the same graphics data, but with source and size changed to show only a 32 x 32 detail:



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

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



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

Graphics context

None

See also

[BITMAP_LAYOUT](#), [BITMAP_SIZE](#)

4.12 BITMAP_TRANSFORM_A

Specify the *A* coefficient of the bitmap transform matrix.

Encoding

31	24	23	17	16	0
0x15		reserved			a

Parameters

a

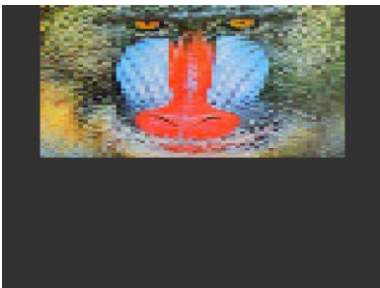
The *A* coefficient of the bitmap transform matrix, in signed 8.8 fixed point form.
 The initial value is 256.

Description

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

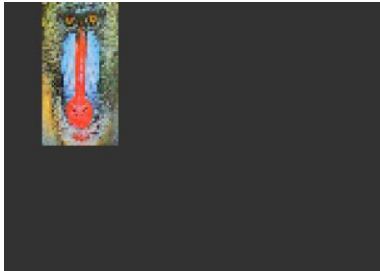
Examples

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



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

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



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

Graphics Context

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

See also

None

4.13 BITMAP_TRANSFORM_B

Specify the *B* coefficient of the bitmap transform matrix

Encoding

31	24	23	17	16	0
0x16			reserved	b	

Parameters

b

The *B* coefficient of the bitmap transform matrix, in signed 8.8 fixed point form.
 The initial value is 0.

Description

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

Graphics context

The value of *b* is part of the graphics context, as described in section 4.1.

See also

None

4.14 BITMAP_TRANSFORM_C

Specify the *C* coefficient of the bitmap transform matrix

Encoding

31	24	23	0
0x17			c

Parameters

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

Description

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

Graphics context

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

See also

None

4.15 BITMAP_TRANSFORM_D

Specify the *D* coefficient of the bitmap transform matrix

Encoding

31	24	23	17	16	0
0x18		reserved		d	

Parameters

d
 The *D* coefficient of the bitmap transform matrix, in signed 8.8 fixed point form. The initial value is 0.

Description

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

Graphics context

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

See also

None

4.16 BITMAP_TRANSFORM_E

Specify the *E* coefficient of the bitmap transform matrix.

Encoding

31	24	23	17	16	0
0x19		reserved		e	

Parameters

e

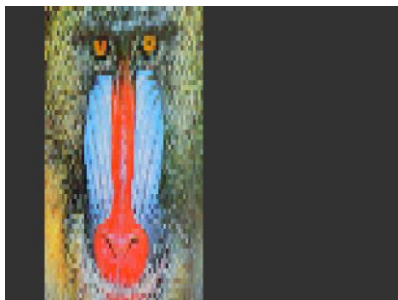
The *E* coefficient of the bitmap transform matrix, in signed 8.8 fixed point form.
 The initial value is 256.

Description

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

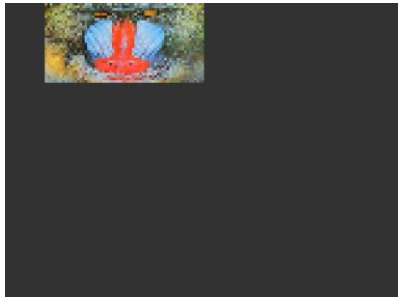
Examples

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



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

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



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

Graphics context

The value of *e* is part of the graphics context, as described in section [4.1](#)

See also

None

4.17 BITMAP_TRANSFORM_F

Specify the *f* coefficient of the bitmap transform matrix

Encoding

31	24	23	0
0x1A		f	

Parameters

f

The *F* component 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** transformation functionality.

Graphics context

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

See also

None

4.18 BLEND_FUNC

Specify pixel arithmetic

Encoding

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

Parameters

src

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

dst

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

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

Table 16 – BLEND_FUNC Constant Value 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:



```
d1( BEGIN(BITMAPS) );
d1( VERTEX2II(50, 30, 31, 0x47) );
d1( COLOR_A( 128 ) );
d1( VERTEX2II(60, 40, 31, 0x47) );
```

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



```
d1( BEGIN(BITMAPS) );
d1( BLEND_FUNC(SRC_ALPHA, ZERO) );
d1( VERTEX2II(50, 30, 31, 0x47) );
d1( COLOR_A( 128 ) );
d1( VERTEX2II(60, 40, 31, 0x47) );
```

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



```
d1( BEGIN(BITMAPS) );
d1( BLEND_FUNC(ZERO, SRC_ALPHA) );
d1( VERTEX2II(50, 30, 31, 0x47) );
```

Graphics context

The values of src and dst are part of the graphics context, as described in section [4.1](#).

See also

[COLOR_A](#)

4.19 CALL

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

Encoding

31	24	23	16	15	0
0x1D		reserved			dest

Parameters

dest

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

The valid range is from 0 to 2047 or (sizeof(**RAM_DL**)/4-1).

Description

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

Graphics context

None

See also

[JUMP](#), [RETURN](#)

4.20 CELL

Specify the bitmap cell number for the **VERTEX2F** command.

Encoding

31	24	23	7	6	0
0x06		reserved			cell

Parameters

cell

bitmap cell number. The initial value is 0

Graphics context

The value of cell is part of the graphics context, as described in section [4.1](#).

See also

None

4.21 CLEAR

Clear buffers to preset values

Encoding

31	24	23	3	2	1	0	
0x26		reserved			c	s	t

Parameters

c

Clear color buffer. Setting this bit to 1 will clear the color buffer to the preset value. Setting this bit to 0 will maintain the color buffer 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 to the preset value. Setting this bit to 0 will maintain the stencil buffer 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 to the preset value. Setting this bit to 0 will maintain the tag buffer 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 stencil do not affect the clear.

Examples

To clear the screen to bright blue:



```
d1( CLEAR_COLOR_RGB(0, 0, 255) );  
d1( CLEAR(1, 0, 0) );
```

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



```
d1( CLEAR_COLOR_RGB(100, 100, 100) );  
d1( CLEAR(1, 1, 1) );  
d1( CLEAR_COLOR_RGB(0, 0, 255) );  
d1( SCISSOR_SIZE(30, 120) );  
d1( CLEAR(1, 1, 1) );
```

Graphics context

None

See also

[CLEAR_COLOR_A](#), [CLEAR_STENCIL](#), [CLEAR_TAG](#), [CLEAR_COLOR_RGB](#)

4.22 CLEAR_COLOR_A

Specify clear value for the alpha channel

Encoding

31	24	23	8	7	0
0x0F		reserved			alpha

Parameters

alpha

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

Graphics context

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

See also

[CLEAR_COLOR_RGB](#), [CLEAR](#)

4.23 CLEAR_COLOR_RGB

Specify clear values for red, green and blue channels

Encoding

31	24	23	16	15	8	7	0
0x02		red		blue		green	

Parameters

red

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

green

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

blue

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

Description

Sets the color values used by a following **CLEAR**.

Examples

To clear the screen to bright blue:



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

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



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

Graphics context

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

See also

[CLEAR_COLOR_A](#), [CLEAR](#)

4.24 CLEAR_STENCIL

Specify clear value for the stencil buffer

Encoding

31	24	23	8	7	0
0x11	reserved			s	

Parameters

s

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

Graphics context

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

See also

[CLEAR](#)

4.25 CLEAR_TAG

Specify clear value for the tag buffer

Encoding

31	24	23	8	7	0
0x12		reserved			t

Parameters

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

Graphics context

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

See also

[TAG](#), [TAG_MASK](#), [CLEAR](#)

4.26 COLOR_A

Set the current color alpha

Encoding

31	24	23	8	7	0
0x10		reserved			alpha

Parameters

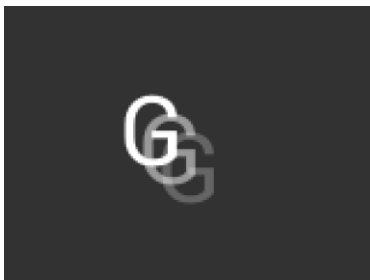
alpha
 Alpha for the current color. The initial value is 255

Description

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

Examples

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



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

Graphics context

The value of alpha is part of the graphics context, as described in section [4.1](#).

See also

[COLOR_RGB](#), [BLEND_FUNC](#)

4.27 COLOR_MASK

Enable or disable writing of color components

Encoding

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

Parameters

r

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

g

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

b

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

a

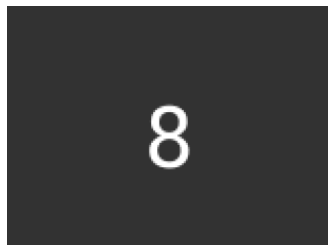
Enable or disable the alpha channel update of the color buffer. The initial value is 1 and means enable.

Description

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

Examples

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



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

Graphics context

The values of r, g, b and a are part of the graphics context, as described in section [4.1](#).

See also

[TAG_MASK](#)

4.28 COLOR_RGB

Set the current color red, green and blue.

Encoding

31	24	23	16	15	8	7	0
0x04		red		blue		green	

Parameters

red

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

green

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

blue

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

Description

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

Examples

Drawing three characters with different colors:



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

Graphics context

The values of red, green and blue are part of the graphics context, as described in section [4.1](#).

See also

[COLOR_A](#)

4.29 DISPLAY

End the display list. All the commands following this command will be ignored.

Encoding

31	24	23	0
0x0		reserved	

Parameters

None

Graphics context

None

See also

None

4.30 END

End drawing a graphics primitive.

Encoding

31	24	23	0
0x21		reserved	

Parameters

None

Description

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

Graphics context

None

See also

[BEGIN](#)

4.31 JUMP

Execute commands at another location in the display list

Encoding

31	24	23	16	15	0
0x1E		reserved			dest

Parameters

dest

Display list offset to be jumped. The valid range is from 0 to 2047(sizeof(**RAM_DL**)/4-1).

Graphics context

None

See also

[CALL](#)

4.32 LINE_WIDTH

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

Encoding

31	24	23	12	11	0
0x0E		reserved			width

Parameters

width

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

Description

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

Please note it will affect the **LINES**, **LINE_STRIP**, **RECTS** primitives.

Note: The lines are drawn with the requested width, but below around 6 the pixels get very dark and hard to see. Half pixel lines (width 8) are totally usable.

Examples

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



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

Graphics context

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

See also

None

4.33 MACRO

Execute a single command from a macro register.

Encoding

31	24	23	10
0x25		reserved	
		m	

Parameters

m
 Macro registers to read. Value 0 means the content in **REG_MACRO_0** is to be fetched and inserted in place. Value 1 means **REG_MACRO_1** is to be fetched and inserted in place. The content of **REG_MACRO_0** or **REG_MACRO_1** shall be a valid display list command, otherwise the behavior is undefined.

Graphics context

None

See also

None

4.34 NOP

No operation.

Encoding

31	24	23	0
0x2D		reserved	

Parameters

None

Description

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

Graphics context

None

See also

None

4.35 PALETTE_SOURCE

Specify the base address of the palette.

Encoding

31	24	23	22	21	0
0x2A		reserved		addr	

Parameters

addr

Address of palette in **RAM_G**, 2-byte alignment is required if pixel format is **PALETTED4444** or **PALETTED565**. The initial value is **RAM_G**.

Description

Specify the base address in **RAM_G** for palette

Graphics context

The value of addr is part of the graphics context

See also

None

4.36 POINT_SIZE

Specify the radius of points

Encoding

31	24	23	13	12	0
0x0D		reserved		size	

Parameters

size

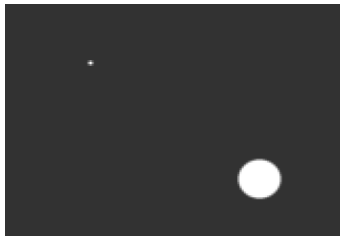
Point radius in 1/16-pixel precision. The initial value is 16. The valid range is from zero to 8191, i.e., from 0 to 511 pixels.

Description

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

Examples

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



```
d1 ( BEGIN(POINTS) );
d1 ( VERTEX2II (40, 30, 0, 0) );
d1 ( POINT_SIZE(160) );
d1 ( VERTEX2II (120, 90, 0, 0) );
```

Graphics context

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

See also

None

4.37 RESTORE_CONTEXT

Restore the current graphics context from the context stack.

Encoding

31	24	23	0
0x23	reserved		

Parameters

None

Description

Restores the current graphics context, as described in section 4.1. Four levels of **SAVE** and **RESTORE** stacks are available. 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:



```
d1 ( BEGIN(BITMAPS) );
d1 ( COLOR_RGB( 255, 0, 0 ) );
d1 ( SAVE_CONTEXT() );
d1 ( COLOR_RGB( 50, 100, 255 ) );
d1 ( VERTEX2II (80, 38, 31, 0x47) );
d1 ( RESTORE_CONTEXT() );
d1 ( VERTEX2II (110, 38, 31, 0x47) );
```

Graphics context

None

See also

[SAVE_CONTEXT](#)

4.38 RETURN

Return from a previous **CALL** command.

Encoding

31	24	23	0
0x24		reserved	

Parameters

None

Description

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

Graphics context

None

See also

[CALL](#)

4.39 SAVE_CONTEXT

Push the current graphics context on the context stack

Encoding

31	24	23	0
0x22		reserved	

Parameters

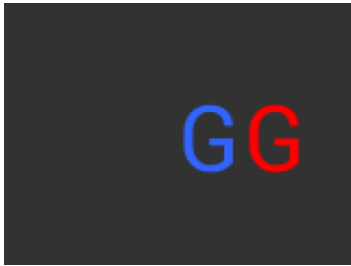
None

Description

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

Examples

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



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

Graphics context

None

See also

[RESTORE_CONTEXT](#)

4.40 SCISSOR_SIZE

Specify the size of the scissor clip rectangle.

Encoding

31	24	23	12	11	0
0x1C		width		height	

Parameters

width

The width of the scissor clip rectangle, in pixels. The initial value is 2048.
 The value of zero will cause zero output on screen.
 The valid range is from zero to 2048.

height

The height of the scissor clip rectangle, in pixels. The initial value is 2048.
 The value of zero will cause zero output on screen.
 The valid range is from zero to 2048.

Description

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

Examples

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



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


Graphics context

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

See also

None

4.41 SCISSOR_XY

Specify the top left corner of the scissor clip rectangle.

Encoding

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

Parameters

x

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

y

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

Description

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

Examples

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



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

Graphics context

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

See also

None

4.42 STENCIL_FUNC

Set function and reference value for stencil testing.

Encoding

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

Parameters

func

Specifies the test function, one of **NEVER**, **LESS**, **LEQUAL**, **GREATER**, **GEQUAL**, **EQUAL**, **NOTEQUAL**, or **ALWAYS**. The initial value is **ALWAYS**.
 About the value of these constants, refer to **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

Refer to [STENCIL_OP](#).

Graphics context

The values of func, ref and mask are part of the graphics context, as described in section [4.1](#).

See also

[STENCIL_OP](#), [STENCIL_MASK](#)

4.43 STENCIL_MASK

Control the writing of individual bits in the stencil planes

Encoding

31	24	23	8	7	0
0x13		reserved			mask

Parameters

mask

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

Graphics context

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

See also

[STENCIL_FUNC](#), [STENCIL_OP](#), [TAG_MASK](#)

4.44 STENCIL_OP

Set stencil test actions.

Encoding

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

Parameters

sfail

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

spass

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

Name	Value	Description
ZERO	0	check OpenGL definition
KEEP	1	check OpenGL definition
REPLACE	2	check OpenGL definition
INCR	3	check OpenGL definition
DECR	4	check OpenGL definition
INVERT	5	check OpenGL definition

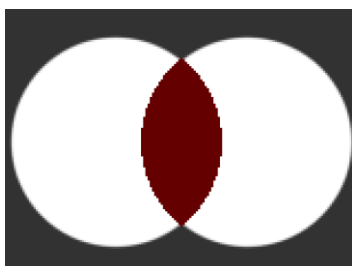
Table 17 – STENCIL_OP Constants Definition

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:



```
d1( STENCIL_OP(INCR, INCR) );
d1( POINT_SIZE(760) );
d1( BEGIN(POINTS) );
d1( VERTEX2II(50, 60, 0, 0) );
d1( VERTEX2II(110, 60, 0, 0) );
d1( STENCIL_FUNC(EQUAL, 2, 255) );
d1( COLOR_RGB(100, 0, 0) );
d1( VERTEX2II(80, 60, 0, 0) );
```

Graphics context

The values of sfail and spass are part of the graphics context, as described in section [4.1](#).

See also

[STENCIL_FUNC](#), [STENCIL_MASK](#)

4.45 TAG

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

Encoding

31	24	23	8	7	0
0x03		reserved			s

Parameters

s

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

Description

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

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

Graphics context

The value of s is part of the graphics context, as described in section [4.1](#).

See also

[CLEAR_TAG](#), [TAG_MASK](#)

4.46 TAG_MASK

Control the writing of the tag buffer

Encoding

31	24	23	1	0
0x14		reserved		mask

Parameters

mask

Allow updates to the tag buffer. The initial value is one and it means the tag buffer 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 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 tag buffer. The tag buffer can be updated by the **TAG** command.

The default value of the 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 tag buffer takes the value from the default value of the tag buffer or the **TAG** command of the display list.

Graphics context

The value of mask is part of the graphics context, as described in section [4.1](#).

See also

[TAG](#), [CLEAR_TAG](#), [STENCIL_MASK](#), [COLOR_MASK](#)

4.47 VERTEX2F

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

Encoding

31	30	29	15	14	0
0x1			x		y

Parameters

x

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

y

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

Description

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

Graphics context

None

See also

[VERTEX_FORMAT](#)

4.48 VERTEX2II

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

Encoding

31	30	29	21	20	12	11	7	6	0
0x2		x			y			handle	cell

Parameters

x

X-coordinate in pixels, unsigned integer ranging from 0 to 511.

y

Y-coordinate in pixels, unsigned integer ranging from 0 to 511.

handle

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

cell

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

Note: The handle and cell parameters are ignored unless the graphics primitive is specified as bitmap by command **BEGIN(BITMAPS)**, prior to this command.

Description

To draw the graphics primitives beyond the coordinate range [(0,0), (511, 511)], use VERTEX2F instead.

Graphics context

None

See also

[BITMAP_HANDLE](#), [CELL](#), [VERTEX2F](#)

4.49 VERTEX_FORMAT

Set the precision of **VERTEX2F** coordinates.

Encoding

31		24	23				32		0
0x27				reserved				frac	

Parameters

frac

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

Description

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

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

frac value	Unit of pixel precision	VERTEX2F range
0	1 pixel	-16384 to 16383
1	$\frac{1}{2}$ pixel	-8192 to 8191
2	$\frac{1}{4}$ pixel	-4096 to 4095
3	$\frac{1}{8}$ pixel	-2048 to 2047
4	$\frac{1}{16}$ pixel	-1024 to 1023

Table 18 – VERTEX_FORMAT and Pixel Precision

Graphics context

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

See also

[VERTEX2F](#), [VERTEX_TRANSLATE_X](#), [VERTEX_TRANSLATE_Y](#)

4.50 VERTEX_TRANSLATE_X

Specify the vertex transformations X translation component.

Encoding

31	24	23	17	16	0
0x2B		reserved		x	

Parameters

x

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

Description

Specifies the offset added to vertex X coordinates. This command allows drawing to be shifted on the screen. It applies to both **VERTEX2F** and **VERTEX2II** commands.

Graphics context

The value of x is part of the graphics context

See also

NONE

4.51 VERTEX_TRANSLATE_Y

Specify the vertex transformation's Y translation component.

Encoding

31	24	23	17	16	0
0x2C		reserved		y	

Parameters

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

Description

Specifies the offset added to vertex Y coordinates. This command allows drawing to be shifted on the screen. It applies to both **VERTEX2F** and **VERTEX2II** commands.

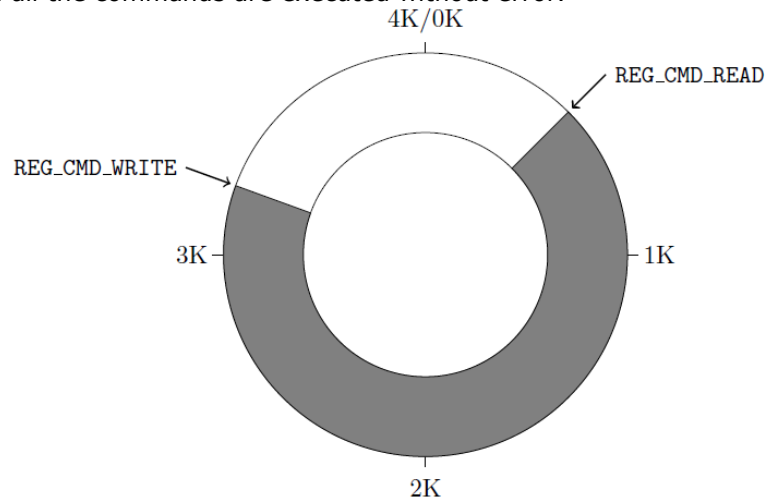
Graphics context

The value of y is part of the graphics context

5 Coprocessor Engine

5.1 Command FIFO

The coprocessor engine is fed via a 4K byte **FIFO** called **RAM_CMD**. The host writes coprocessor commands or display list commands into the **FIFO**, and the coprocessor engine reads and executes the commands. The host updates the register **REG_CMD_WRITE** to indicate that there are new commands in the **FIFO**, and the coprocessor engine updates **REG_CMD_READ** after the commands have been executed. Therefore, when **REG_CMD_WRITE** is equal to **REG_CMD_READ**, it indicates the **FIFO** is empty and all the commands are executed without error.



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

$$\begin{aligned} \text{fullness} &= (\text{REG_CMD_WRITE} - \text{REG_CMD_READ}) \bmod 4096 \\ \text{free space} &= (4096 - 4) - \text{fullness}; \end{aligned}$$

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

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

FIFO entries are always 4 bytes wide – it is an error for either **REG_CMD_READ** or **REG_CMD_WRITE** to have a value that is not a multiple of 4 bytes. Each command issued to the coprocessor 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 coprocessor engine ignores the extra 1, 2 or 3 bytes and continues reading the next command at the following 4-byte boundary.

To offload work from the host for checking the free space in the circular buffer, **EVE** offers a pair of registers **REG_CMDB_SPACE** and **REG_CMDB_WRITE**. It enables the host to write commands and data to the coprocessor in a bulk transfer, without computing the free space in the circular buffer and increasing the address. As long as the amount of data to be transferred is less than the value in the register **REG_CMDB_SPACE**, the host is able to safely write all the data to **REG_CMDB_WRITE** in one write transfer. All writes to **REG_CMDB_WRITE** are appended to the command **FIFO** and may be of any length that is a multiple of 4 bytes. To determine the free space of **FIFO**, reading **REG_CMDB_SPACE** and checking if it is equal to 4092 is easier and faster than comparing **REG_CMD_WRITE** and **REG_CMD_READ**.

5.2 Widgets

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

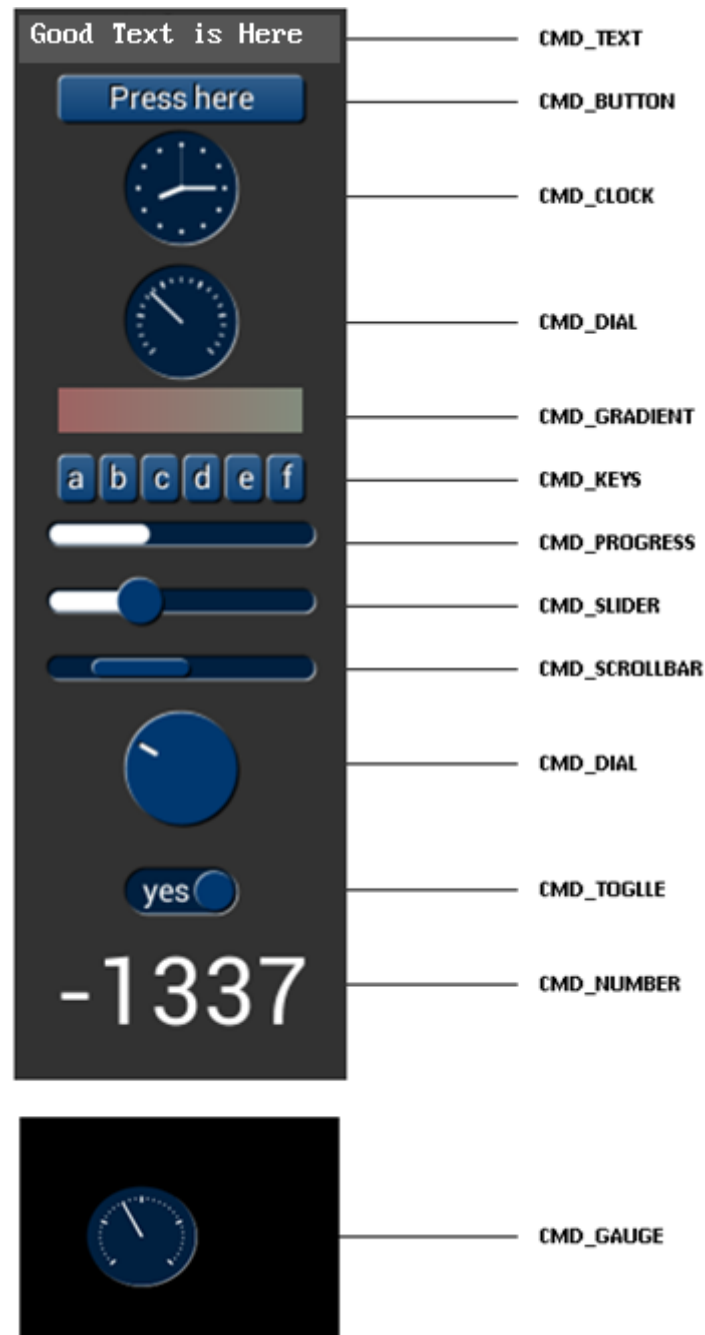


Figure 3 – Widget List

5.2.1 Common Physical Dimensions

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

- All rounded corners have a radius that is computed from the font used for the widget (curvature of lowercase 'o' character).

$$\text{Radius} = \text{font height} * 3 / 16$$

- All 3D shadows are drawn with:
 - (1) Highlight offsets 0.5 pixels above and left of the object
 - (2) Shadow offsets 1.0 pixel below and right of the object.
- For widgets such as progress bar, scrollbar and slider, the output will be a vertical widget in the case where width and height parameters are of same value.

5.2.2 Color Settings

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

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

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

Table 19 – Widgets Color Setup Table

5.2.3 Caveat

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

5.3 Interaction with RAM_DL

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

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

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

Of course, this display list could have been written directly to **RAM_DL**. The advantage of this technique is that you can mix low-level operations and high-level coprocessor 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!"); // Label of button
cmd(DISPLAY()); // Mark the end of display list
```

5.3.1 Synchronization between host & Coprocessor Engine

At some points, it is necessary to wait until the coprocessor engine has processed all outstanding commands. When the coprocessor engine completes the last outstanding command in the command buffer, it raises the **INT_CMDEEMPTY** interrupt. Another approach to detecting synchronization is that the host 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 host needs to do direct writes into the display list. In this situation the host should wait until the coprocessor engine is idle before reading **REG_CMD_DL**.

5.4 ROM and RAM Fonts

Fonts in **EVE** are treated as a set of bitmap-graphics with metrics block indexed by handles from 0 to 31. The following commands are using fonts:

- **CMD_BUTTON**
- **CMD_KEYS**
- **CMD_TOGGLE**
- **CMD_TEXT**
- **CMD_NUMBER**

5.4.1 Legacy Font Metrics Block

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

The format of the 148-bytes font metrics block is as below: (p is the base address of metrics block)

Address	Size	Value	Description
p + 0	128	width	width of each font character, in pixels
p + 128	4	format	bitmap format as defined in BITMAP_FORMAT , except TEXTVGA , TEXT8X8 , BARGRAPH and PALETTED formats.

P + 132	4	line stride	font bitmap line stride, in bytes
p + 136	4	pixel width	font screen width, in pixels
p + 140	4	pixel height	font screen height, in pixels
p + 144	4	pixel data pointer	pointer to glyph data in memory

Table 20 – Legacy Font Metrics Block

For built-in fonts, these blocks are located in built-in **ROM**, in an array of length 19 (font 16 to 34). The address of this array is held in ROM location **ROM_FONTROOT**.

For custom fonts, these blocks shall be located in **RAM_G**.

5.4.2 Example to find the width of character

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

read 32-bit pointer p from **ROM_FONTROOT**

$\text{widths} = p + (148 * (34 - 16))$ (table starts at font 16)

read byte from memory at $\text{widths}[0x67]$

5.4.3 ROM Fonts (Built-in Fonts)

In total, there are 19 **ROM** fonts numbered from 16 to 34.

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

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

For **ROM** fonts 16 to 34 (except 17 and 19), each font includes 95 printable **ASCII** characters from 0x20 to 0x7E inclusive. All these characters are indexed by its corresponding **ASCII** value. For ROM fonts 17 and 19, each font includes 127 printable **ASCII** characters from 0x80 to 0xFF, inclusive. All these characters are indexed using value from 0x0 to 0x7F, i.e., code 0 maps to **ASCII** character 0x80 and code 0x7F maps to **ASCII** character 0xFF. Users are required to handle this mapping manually.

The picture below shows the **ROM** font effects:

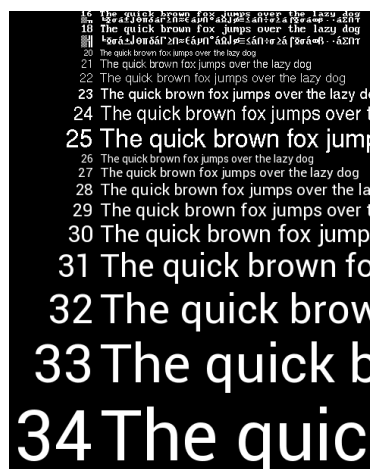


Figure 4 – ROM Font List

5.4.4 Using Custom Fonts

Users can define custom fonts by following the steps below:

- Select a bitmap handle 0-31
- Load the font bitmap(glyph) into **RAM_G**
- Create or load a font metrics block in **RAM_G**
- Set up bitmap parameters by using display list command:
 - **BITMAP_SOURCE,**
 - **BITMAP_LAYOUT/BITMAP_LAYOUT_H**
 - **BITMAP_SIZE/BITMAP_SIZE_H**
 or using the coprocessor command **CMD_SETBITMAP**.
- Use command **CMD_SETFONT** or **CMD_SETFONT2** to register the new font with the handle 0-31

After this setup, the bitmap handle 0-31 is redefined and can be used as a font parameter of coprocessor commands for custom font.

5.5 Coprocessor Faults

Some commands can cause coprocessor faults. These faults arise because the coprocessor 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 coprocessor:

1. sets **REG_CMD_READ** to 0xfff (an illegal value because all command buffer data is 32-bit aligned),
2. raises the **INT_CMDEEMPTY** interrupt
3. stops accepting new commands

In the event of a fault condition, the host can perform the following recovery process:

1. Set **REG_CPURESET** to 1, to hold the coprocessor engine in the reset condition
2. Set **REG_CMD_READ**, **REG_CMD_WRITE**, **REG_CMD_DL** to zero
3. Set **REG_CPURESET** to 0, to restart the coprocessor engine

5.6 Coprocessor Graphics State

The graphics drawing in the coprocessor engine is supported by a limited set of internal states, which are initialized to their default values during coprocessor engine reset or by using **CMD_COLDSTART**. These state values remain unaffected by **CMD_DLSTART** or **CMD_SWAP** commands, which means that the application only needs to set them once during the startup phase.

State	Default	Commands
background color	<i>dark blue (0x002040)</i>	CMD_BGCOLOR
foreground color	<i>light blue (0x003870)</i>	CMD_FGCOLOR
gradient color	<i>white (0xFFFFFFFF)</i>	CMD_GRADCOLOR
spinner style	<i>None</i>	CMD_SPINNER
object trackers	<i>all disabled</i>	CMD_TRACK
interrupt timer	<i>None</i>	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_SCALE, CMD_ROTATE
scratch bitmap handle	15	CMD_SETSCRATCH
font pointers 0-15	Undefined	CMD_SETFONT, CMD_SETFONT2
font pointer 16-31	ROM fonts 16-31	CMD_SETFONT, CMD_SETFONT2, CMD_ROMFONT
base of number	10	CMD_SETBASE
media FIFO	Address is zero and length is zero	CMD_MIDEAFIFO,

Table 21 – Coprocessor Engine Graphics State

5.7 Parameter OPTION

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

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

Name	Value	Description	Commands
OPT_RIGHTX	2048	Right justified style	CMD_KEYS, CMD_TEXT, CMD_NUMBER
OPT_NOBACK	4096	No background drawn	CMD_CLOCK, CMD_GAUGE
OPT_NOTICKS	8192	No Ticks	CMD_CLOCK, CMD_GAUGE
OPT_NOHM	16384	No hour and minute hands	CMD_CLOCK
OPT_NOPOINTER	16384	No pointer	CMD_GAUGE
OPT_NOSECS	32768	No second hands	CMD_CLOCK
OPT_NOHANDS	49152	No hands	CMD_CLOCK
OPT_NOTEAR	4	Synchronize video updates to the display blanking interval, avoiding horizontal "tearing" artefacts.	CMD_PLAYVIDEO
OPT_FULLSCREEN	8	Zoom the video so that it fills as much of the screen as possible.	CMD_PLAYVIDEO
OPT_MEDIAFIFO	16	source video/image/compressed(zlib) data from the defined media FIFO	CMD_PLAYVIDEO CMD_VIDEOFRAME CMD_LOADIMAGE
OPT_SOUND	32	Decode the audio data of MJPEG stream	CMD_PLAYVIDEO

Table 22 – Parameter OPTION Definition

5.8 Resources Utilization

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

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

- Bitmap handle 15 is used by the 3D-effect buttons, keys and gradient, unless it is set to another bitmap handle using **CMD_SETSCRATCH**.
When drawing graphics objects with **CMD_***, a single graphics context is utilized, and the **SAVE_CONTEXT/RESTORE_CONTEXT** commands have an effective stack depth of 3 levels.

5.9 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 a text string
- **CMD_BUTTON** – draw a button with a label.
- **CMD_CLOCK** – draw an analog clock
- **CMD_BGCOLOR** – set the background color
- **CMD_FGCOLOR** – set the foreground color
- **CMD_GRADCOLOR** – set up the highlight color used in 3D effects for **CMD_BUTTON** and **CMD_KEYS**
- **CMD_GAUGE** – draw a gauge
- **CMD_GRADIENT** – draw a smooth color gradient
- **CMD_KEYS** – draw a row of keys
- **CMD_PROGRESS** – draw a progress bar
- **CMD_SCROLLBAR** – draw a scroll bar

- **CMD_SLIDER** – draw a slider
- **CMD_DIAL** – draw a rotary dial control
- **CMD_TOGGLE** – draw a toggle switch with label
- **CMD_NUMBER** – draw a decimal number
- **CMD_SETBASE** – set the base for number output

Commands to operate on **RAM_G**:

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

Commands for loading data into **RAM_G**:

- **CMD_INFLATE** – decompress data into **RAM_G**
- **CMD_LOADIMAGE** – load a **JPEG/PNG** image into **RAM_G**
- **CMD_MEDIAFIFO** – set up a streaming media FIFO in **RAM_G**
- **CMD_VIDEOFRAME** – load video frame from **RAM_G**.

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
- **CMD_BITMAP_TRANSFORM** – compute a bitmap transform matrix

Commands for video playback:

- **CMD_VIDEOSTART** – Initialize the video frame decoder
- **CMD_VIDEOFRAME** – Load video frame data
- **CMD_PLAYVIDEO** – play back motion-**JPEG** encoded AVI video

Other commands:

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

5.10 CMD_DLSTART

This command starts a new display list. When the coprocessor engine executes this command, it waits until the current display list is ready for writing, and then sets **REG_CMD_DL** to zero.

C prototype

```
void cmd_dlstart( );
```

Command layout

+0	CMD_DLSTART (0xFFFF FF00)
----	----------------------------------

Examples

NA

5.11 CMD_INTERRUPT

This command is used to trigger Interrupt **CMDFLAG**. When the coprocessor engine executes this command, it triggers interrupt, which will set the bit field **CMDFLAG** of **REG_INT_FLAGS**, unless the corresponding bit in **REG_INT_MASK** is zero.

C prototype

```
void cmd_interrupt( uint32_t ms );
```

Parameters

ms

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

Command layout

+0	CMD_INTERRUPT(0xFFFF FF02)
+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);
//...
```

5.12 CMD_COLDSTART

This command sets the coprocessor engine to default reset states.

C prototype

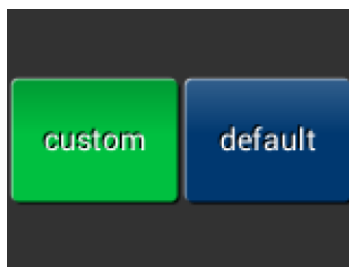
```
void cmd_coldstart( );
```

Command layout

+0	CMD_COLDSTART(0xFFFF FF32)
----	----------------------------

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");
```

5.13 CMD_SWAP

This command is used to swap the current display list. When the coprocessor engine executes this command, it requests a display list swap immediately after the current display list is scanned out. Internally, the coprocessor engine implements this command by writing to **REG_DLSWAP** with **0x02**.

This coprocessor engine command will not generate any display list command into display list memory **RAM_DL**. It is expected to be used with **CMD_DLSTART** in pair.

C prototype

```
void cmd_swap( );
```

Command layout

+0	CMD_SWAP(0xFFFF FF01)
----	-----------------------

Examples

NA

5.14 CMD_APPEND

This command adds additional commands from **RAM_G** to the existing display list memory address, with the offset specified in **REG_CMD_DL**.

C prototype

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

Parameters

ptr

Starting address of source commands in **RAM_G**

num

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

Command layout

+0	CMD_APPEND(0xFFFF FF1E)
+4	ptr
+8	num

Description

After appending is done, the coprocessor 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();
```

5.15 CMD_REGREAD

This command is used to read a register value.

C prototype

```
void cmd_regread( uint32_t ptr,
                  uint32_t result );
```

Parameters

ptr

Address of the register to be read

result

The register value which has been read from the ptr address. **OUTPUT** parameter.

Write a dummy 32-bit value 0x00000000 for this parameter and the Co-Processor will replace this value with the result after the command has been executed.

After execution, the host should then read the address of this parameter in **RAM_CMD** to get the result value.

Command layout

+0	CMD_REGREAD(0xFFFF FF19)
+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) % 4096));
```

5.16 CMD_MEMWRITE

This command writes the following bytes into the 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**.

Note: If using this command improperly, it may corrupt the memory.

Command layout

+0	CMD_MEMWRITE(0xFFFF FF1A)
+4	ptr
+8	num
+12 ...n	byte ₀ ... byte _n

Examples

```
//To change the backlight brightness to 0x64 (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);
```

5.17 CMD_INFLATE

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

C prototype

```
void cmd_inflate( uint32_t ptr );
```

Parameters

ptr

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

Description

In case the number of bytes is not a multiple of 4, it's necessary to append 1, 2, or 3 bytes to align the next command on a 4-byte boundary. The padding bytes can contain any value.

Command layout

+0	CMD_INFLATE(0xFFFF FF22)
+4	ptr
+8 ...n	byte ₀ ... byte _n

Examples

To load graphics data to main memory address 0x8000:

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

5.18 CMD_LOADIMAGE

This command is used to load a **JPEG** or **PNG** image. Decompress the following **JPEG** or **PNG** image data into an **EVE** specific bitmap, in **RAM_G**. The image data should be in the following formats:

- Regular baseline **JPEG** (**JFIF**)
- **PNG** with bit-depth 8 only and no interlace

C prototype

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

Parameters

ptr

Destination address, within **RAM_G**

options

For **JPEG** images, the bitmap is loaded as either a **RGB565** or **L8** format bitmap. If **OPT_MONO** is given, **L8** is used. Otherwise, **RGB565** is used. **OPT_RGB565** and **OPT_MONO** is specific to **JPEG** images only.

For **PNG** images, the **PNG** standard specifies various color formats. Each of these formats is loaded into a bitmap in the following manner:

Color type	Format	Loaded by CMD_LOADIMAGE
0	Grayscale	L8
2	Truecolor	RGB565
3	Indexed	PALETTED565 or PALETTED4444

4	Grayscale and alpha	not supported
6	Truecolor and alpha	ARGB4

Option **OPT_FULLSCREEN** causes the bitmap to be scaled so that it fills as much of the screen as possible.

If option **OPT_MEDIAFIFO** is given, the media FIFO is used for the image data source.

If option **OPT_MEDIAFIFO** is *not* given, then the byte data shall immediately follow in the command **FIFO**.

To minimize the programming effort to render the loaded image, there are a set of display list commands generated and appended to the current display list, unless **OPT_NODL** is given.

Description

The data byte should immediately follow in the command FIFO if **OPT_MEDIAFIFO** is *NOT* set. If the number of bytes is not a multiple of 4, then 1, 2 or 3 bytes should be appended to ensure 4-byte alignment of the next command. These padding bytes can have any value. The application on the host processor has to parse the **JPEG/PNG** header to get the properties of the **JPEG/PNG** image and decide to decode. Behavior is unpredictable in cases of non-baseline JPEG images or the output data generated is more than the **RAM_G** size.

Note: If the loading image is in **PNG** format, the top 42K bytes of **RAM_G** will be used as temporary data buffer for decoding process.

Command layout

+0	CMD_LOADIMAGE(0xFFFF FF24)
+4	ptr
+8	options
+12	byte 0
+13	byte 1
...	...
+n	byte 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)
```

5.19 CMD_MEDIAFIFO

This command is to set up a streaming media FIFO. Allocate the specified area of **RAM_G** and set it up as streaming media **FIFO**, which is used by:

- **MJPEG** video play-back: **CMD_PLAYVIDEO/CMD_VIDEOFRAME**
- **JPEG/PNG** image decoding: **CMD_LOADIMAGE**

if the option **OPT_MEDIAFIFO** is selected.

C prototype

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

Parameters

ptr

starting address of media **FIFO**

size

number of bytes of media **FIFO**

Command layout

+0	CMD_MEDIAFIFO (0xFFFF FF39)
+4	ptr
+8	size

Examples

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

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

It prints:

R=0x000F000 W=0x00F000

5.20 CMD_PLAYVIDEO

This command plays back **MJPEG**-encoded AVI video.

Playback starts immediately, and the command completes when playback ends.

During the command execution, the **RGB565** bitmap will be created at starting address of **RAM_G**, and is $2 \times W \times H$ bytes in size, where W and H are the width and height of the video. If **OPT_SOUND** is given then a 32 Kbyte audio buffer follows the bitmap. It means that area of RAM_G will be overwritten by **CMD_PLAYVIDEO**.

C prototype

```
void cmd_playvideo (uint32_t opts);
```

Parameters

opts: The options of playing video

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

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

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

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

OPT_OVERLAY: Append the video bitmap to an existing display list, instead of starting a new display list.

OPT_NODL: Will not change the current display list. There should already be a display list rendering the video bitmap.

data

The video data to be played unless **opts** is assigned with **OPT_MEDIAFIFO**. Data following parameter "opts" shall be padded to 4 bytes aligned with zero.

Command layout

+0	CMD_PLAYVIDEO (0xFFFF FF3A)
+4	opts
+8~ +n	byte ₁ ... byte _n

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

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

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

Examples

To play back an AVI video, full-screen:

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

5.21 CMD_VIDEOSTART

This command is used to initialize video frame decoder. The video data should be supplied using the media FIFO. This command processes the video header information from the media FIFO, and completes when it has consumed it.

C prototype

```
void cmd_videostart( );
```

Parameters

None

Command layout

+0	CMD_VIDEOSTART (0xFFFF FF40)
----	-------------------------------------

Examples

To load frames of video at address 4:

```
cmd_videostart();
cmd_videoframe(4, 0);
```

5.22 CMD_VIDEOFRAME

This command is used to load the next frame of a video. The video data should be supplied in the media fifo. This command extracts the next frame of video, and completes when it has consumed it.

C prototype

```
void cmd_videoframe( uint32_t dst,
                    uint32_t ptr );
```

Parameters

dst

Memory location to load the frame data, this will be located in **RAM_G**.

ptr

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

Command layout

+0	CMD_VIDEOFRAME (0xFFFF FF41)
+4	dst
+8	ptr

Examples

To load frames of video at address 4:

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

5.23 CMD_MEMCRC

This command computes a CRC-32 for a block of **RAM_G** 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.

Command layout

+0	CMD_MEMCRC(0xFFFF FF18)
+4	ptr
+8	num
+12	result

Examples

To compute the CRC-32 of the first 1K byte of 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);

//wait till the command is complete
printf("CRC result is %08x\n", rd32(RAM_CMD + (x + 12) % 4096));
```

5.24 CMD_MEMZERO

This command is used to 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

Command layout

+0	CMD_MEMZERO(0xFFFF FF1C)
+4	ptr
+8	num

Examples

```
//To erase the first 1K of main memory:
cmd_memzero(0, 1024);
```

5.25 CMD_MEMSET

This command is used to 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

Command layout

+0	CMD_MEMSET(0xFFFF FF1B)
+4	ptr
+8	value
+12	num

Examples

```
//To write 0xff the first 1K of main memory:
cmd_memset(0, 0xff, 1024);
```

5.26 CMD_MEMCPY

This command is used to 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

Command layout

+0	CMD_MEMCPY(0xFFFF FF1D)
+4	dst
+8	src
+12	num

Examples

```
//To copy 1K byte of memory from 0 to 0x8000:
cmd_memcpy(0x8000, 0, 1024);
```

5.27 CMD_BUTTON

This command is used to draw a button with a label.

C prototype

```
void cmd_button( int16_t x,
                 int16_t y,
                 uint16_t w,
                 uint16_t h,
                 uint16_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

w

width of button, in pixels

h

height of button, in pixels

font

bitmap handle to specify the font used in the button label. [See ROM and RAM Fonts.](#)

options

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

s

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

Description

Refer to [Coprocessor engine widgets physical dimensions](#) for more information.

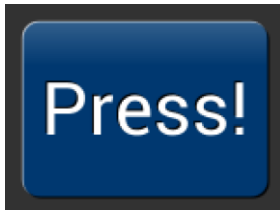
Command layout

+0	CMD_BUTTON(0xFFFF FF0D)
+4	x
+6	y
+8	w
+10	h
+12	font
+14	options

+16	s
+17	...
...	...
+n	0

Examples

A 140x100 pixel button with large text:



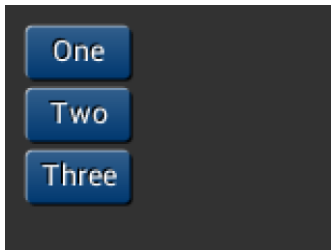
```
cmd_button(10, 10, 140, 100, 31, 0, "Press!");
```

Without the 3D look:



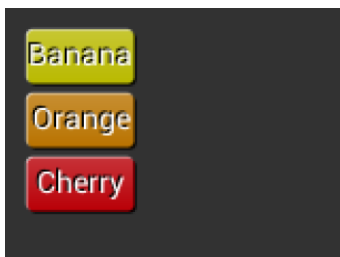
```
cmd_button(10, 10, 140, 100, 31, OPT_FLAT, "Press!");
```

Several smaller buttons:



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

Changing button color



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

5.28 CMD_CLOCK

This command is used to draw an analog clock.

C prototype

```
void cmd_clock(  int16_t x,  
                 int16_t y,  
                 uint16_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

r

the radius of clock, in pixels

options

By default, the clock dial is drawn with a 3D effect and the name of this option is **OPT_3D**. Option **OPT_FLAT** removes the 3D effect.

With option **OPT_NOBACK**, the background is not drawn.

With option **OPT_NOTICKS**, the twelve-hour ticks are not drawn.

With option **OPT_NOSECS**, the seconds hand is not drawn.

With option **OPT_NOHANDS**, no hands are drawn.

With option **OPT_NOHM**, no hour and minutes hands are drawn.

h

hours

m

minutes

s

seconds

ms

milliseconds

Description

The details of the physical dimensions are:

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

Refer to [Coprocessor engine widgets physical dimensions](#) for more information.

Command layout

+0	CMD_CLOCK(0xFFFF FF14)
+4	x
+6	y
+8	r
+10	options
+12	h

+14	m
+16	s
+18	ms

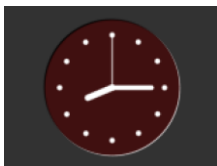
Examples

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:



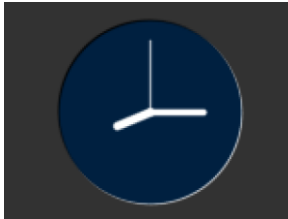
```
cmd_clock(80, 60, 50, OPT_NOSECS, 8, 15, 0, 0);
```


No Background:



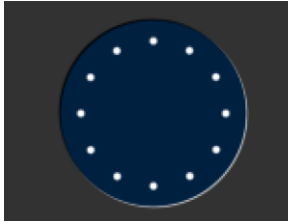
```
cmd_clock(80, 60, 50, OPT_NOBACK, 8, 15, 0, 0);
```

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);
```

5.29 CMD_FGCOLOR

This command is used to set the foreground color.

C prototype

```
void cmd_fgcolor( uint32_t c );
```

Parameters

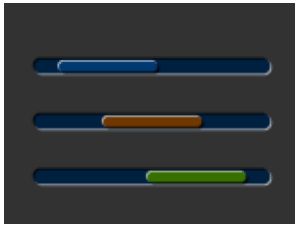
c
 New foreground color, as a 24-bit RGB number.
 Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red.
 Foreground color is applicable for things that the user can move such as handles and buttons.

Command layout

+0	CMD_FGCOLOR(0xFFFF FF0A)
+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);
```

5.30 CMD_BGCOLOR

This command is used to set the background color

C prototype

```
void cmd_bgcolor( uint32_t c );
```

Parameters

c
 New background color, as a 24-bit RGB number.
 Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red.

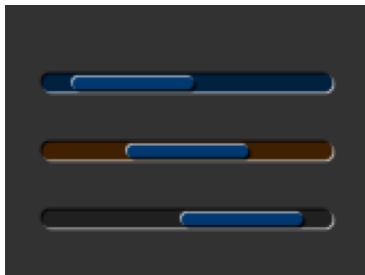
Background color is applicable for things that the user cannot move E.g. behind gauges and sliders etc.

Command layout

+0	CMD_BGCOLOR(0xFFFF FF09)
+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);
```

5.31 CMD_GRADCOLOR

This command is used to 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.
 White is the default value, i.e., 0xFFFFFFFF.

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(0xFFFF FF34)
+4	C

Examples

Changing the gradient color: white, 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");
```

5.32 CMD_GAUGE

This command is used to draw a Gauge.

C prototype

```
void cmd_gauge( int16_t x,
                int16_t y,
                uint16_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

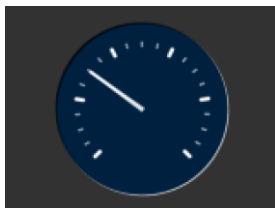
Refer to [Coproprocessor engine widgets physical dimensions](#) for more information.

Command layout

+0	CMD_GAUGE(0xFFFF FF13)
+4	x
+6	y
+8	r
+10	options
+12	major
+14	minor
+16	value
+18	range

Examples

A gauge with radius 50 pixels, five divisions of four ticks each, indicates 30%:



```
cmd_gauge(80, 60, 50, 0, 5, 4, 30, 100);
```

Without the 3D look:



```
cmd_gauge(80, 60, 50, OPT_FLAT, 5, 4, 30, 100);
```

Ten major divisions with two minor divisions each:



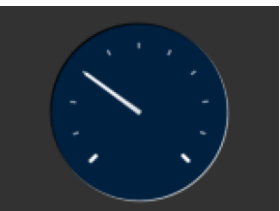
```
cmd_gauge(80, 60, 50, 0, 10, 2, 30, 100);
```

Setting the minor divisions to 1 makes them disappear:



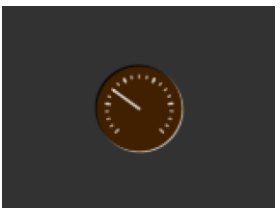
```
cmd_gauge(80, 60, 50, 0, 10, 1, 30, 100);
```

Setting the major divisions to 1 gives minor division only:



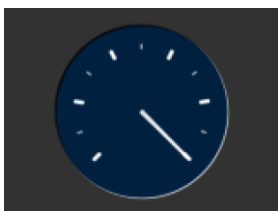
```
cmd_gauge(80, 60, 50, 0, 1, 10, 30, 100);
```

A smaller gauge with a brown background:



```
cmd_bgcolor(0x402000);  
cmd_gauge(80, 60, 25, 0, 5, 4, 30, 100);
```

Scale 0-1000, indicating 1000:



```
cmd_gauge(80, 60, 50, 0, 5, 2, 1000, 1000);
```

Scaled 0-65535, indicating 49152:



```
cmd_gauge(80, 60, 50, 0, 4, 4, 49152, 65535);
```

No background:



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

No tick marks:



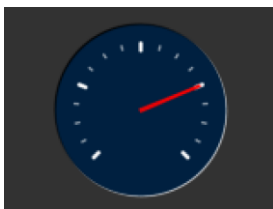
```
cmd_gauge(80, 60, 50, OPT_NOTICKS, 4, 4, 49152, 65535);
```

No pointer:



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

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



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

Add a custom graphic to the gauge by drawing its background, a bitmap, and then its foreground:



```
GAUGE_0 = OPT_NOPOINTER | OPT_NOTICKS;  
GAUGE_1 = OPT_NOBACK;  
cmd_gauge(80, 60, 50, GAUGE_0, 4, 4, 49152, 65535);  
cmd(COLOR_RGB(130, 130, 130));  
cmd(BEGIN(BITMAPS));  
cmd(VERTEX2II(80 - 32, 60 - 32, 0, 0));  
cmd(COLOR_RGB(255, 255, 255));  
cmd_gauge(80, 60, 50, GAUGE_1, 4, 4, 49152, 65535);
```

5.33 CMD_GRADIENT

This command is used to 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. Red is the most significant 8 bits, Blue 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, same definition as **rgb0**.

Description

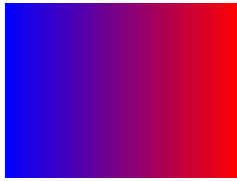
All the color step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is $R0 + t * (R1 - R0)$, where it is interpolated between 0 and 1. Gradient must be used with Scissor function to get the intended gradient display.

Command layout

+0	CMD_GRADIENT(0xFFFF FF0B)
+4	x0
+6	y0
+8	rgb0
+12	x1
+14	y1
+16	rgb1

Examples

A horizontal gradient from blue to red



A vertical gradient

```
cmd_gradient(0, 0, 0x0000ff, 160, 0, 0xff0000);
```



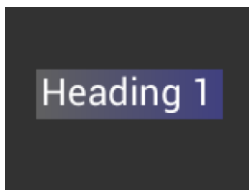
The same colors in a diagonal gradient

```
cmd_gradient(0, 0, 0x808080, 0, 120, 0x80ff40);
```



```
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");
```

5.34 CMD_GRADIENTA

This command is used to draw a smooth color gradient with transparency. The two points have RGB color values, and alpha values which specify their opacity in the range 0x00 to 0xff.

C prototype

```
void cmd_gradienta( int16_t x0,  
                   int16_t y0,  
                   uint32_t argb0,  
                   int16_t x1,  
                   int16_t y1,  
                   uint32_t argb1 );
```

Parameters

x0
x-coordinate of point 0, in pixels

y0
y-coordinate of point 0, in pixels

argb0

color of point 0, as a 32-bit ARGB number. A is the most significant 8 bits, B is the least. So 0x80ff0000 is 50% transparent bright red, and 0xff0000ff is solid blue.

x1

x-coordinate of point 1, in pixels

y1

y-coordinate of point 1, in pixels

argb1

color of point 1

Description

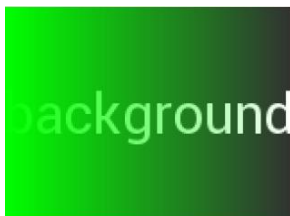
All the color step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is $R0 + t * (R1 - R0)$, where it is interpolated between 0 and 1. Gradient must be used with scissor function to get the intended gradient display.

Command layout

+0	CMD_GRADIENTA(0xFFFF FF57)
+4	x0
+6	y0
+8	argb0
+12	x1
+14	y1
+16	argb1

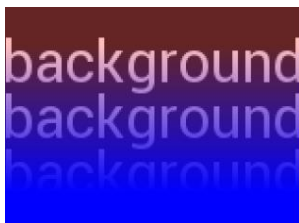
Examples

A solid green gradient, transparent on the right:



```
cmd_text(80, 60, 30, OPT_CENTER, "background");
cmd_gradienta(0,0,0xff00ff00,160,0,0x0000ff00);
```

A vertical gradient from transparent red to solid blue:



```
cmd_text(80, 30, 30, OPT_CENTER, "background");
cmd_text(80, 60, 30, OPT_CENTER, "background");
cmd_text(80, 90, 30, OPT_CENTER, "background");
cmd_gradienta(0,20,0x40ff0000,0,100,0xff0000ff);
```

5.35 CMD_KEYS

This command is used to draw a row of keys.

C prototype

```
void cmd_keys( int16_t x,
               int16_t y,
               uint16_t w,
               uint16_t h,
               uint16_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 [Coproprocessor engine widgets physical dimensions](#) for more information.

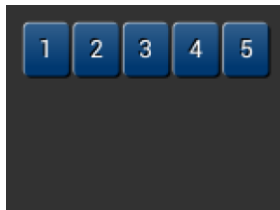
Command layout

+0	CMD_KEYS(0xFFFF FF0E)
+4	x
+6	y
+8	w
+10	h

+12	font
+14	options
+16	s
...	...
+n	0

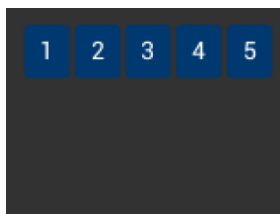
Examples

A row of keys:



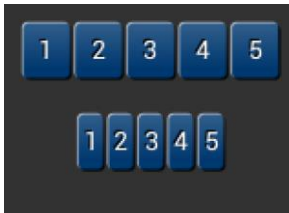
```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");
```

Without the 3D look:



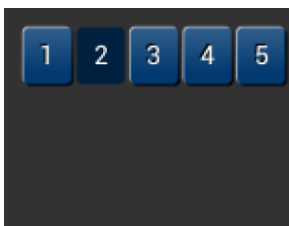
```
cmd_keys(10, 10, 140, 30, 26, OPT_FLAT, "12345");
```

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"" key pressed "" is ASCII code 0x32):



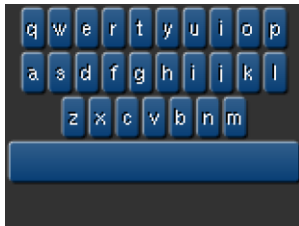
```
cmd_keys(10, 10, 140, 30, 26, 0x32, "12345");
```

A calculator-style keyboard using font 29:



```
cmd_keys(22, 1, 116, 28, 29, 0, "789");  
cmd_keys(22, 31, 116, 28, 29, 0, "456");  
cmd_keys(22, 61, 116, 28, 29, 0, "123");  
cmd_keys(22, 91, 116, 28, 29, 0, "0.");
```

A compact keyboard drawn in font 20:



```
cmd_keys(2, 2, 156, 21, 20, OPT_CENTER, "qwertyuiop");
cmd_keys(2, 26, 156, 21, 20, OPT_CENTER, "asdfghijkl");
cmd_keys(2, 50, 156, 21, 20, OPT_CENTER, "zxcvbnm");
cmd_button(2, 74, 156, 21, 20, 0, "");
```

Showing the f (ASCII 0x66) key pressed:



```
k = 0x66;
cmd_keys(2, 2, 156, 21, 20, k | OPT_CENTER, "qwertyuiop");
cmd_keys(2, 26, 156, 21, 20, k | OPT_CENTER,
"asdfghijkl");
cmd_keys(2, 50, 156, 21, 20, k | OPT_CENTER, "zxcvbnm");
Cmd_button(2, 74, 156, 21, 20, 0, "");
```

5.36 CMD_PROGRESS

This command is used to draw a progress bar.

C prototype

```
void cmd_progress(    int16_t x,
                     int16_t y,
                     uint16_t w,
                     uint16_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. Option **OPT_FLAT** remove the 3D effect and its value is 256

val
Displayed value of progress bar, between 0 and range inclusive

range
 Maximum value

Description

The details of physical dimensions are:

- x,y,w,h give outer dimensions of progress bar. Radius of progress bar is $\min(w,h)/2$
- Radius of inner progress line is $r*(7/8)$

Refer to [Coprocesor engine widgets physical dimensions](#) for more information.

Command layout

+0	CMD_PROGRESS(0xFFFF FF0F)
+4	X
+6	Y
+8	W
+10	h
+12	options
+14	val
+16	range

Examples

A progress bar showing 50% completion:



```
cmd_progress(20, 50, 120, 12, 0, 50, 100);
```

Without the 3D look:



```
cmd_progress(20, 50, 120, 12, OPT_FLAT, 50, 100);
```

A 4-pixel high bar, range 0-65535, with a brown background:



```
cmd_bgcolor(0x402000);  
cmd_progress(20, 50, 120, 4, 0, 9000, 65535);
```

5.37 CMD_SCROLLBAR

This command is used to draw a scroll bar.

C prototype

```
void cmd_scrollbar( int16_t x,
                   int16_t y,
                   uint16_t w,
                   uint16_t h,
                   uint16_t options,
                   uint16_t val,
                   uint16_t size,
                   uint16_t range );
```

Parameters

x

x-coordinate of scroll bar top-left, in pixels

y

y-coordinate of scroll bar top-left, in pixels

w

Width of scroll bar, in pixels. If width is greater than height, the scroll bar is drawn horizontally

h

Height of scroll bar, in pixels. If height is greater than width, the scroll bar is drawn vertically

options

By default, the scroll bar is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

val

Displayed value of scroll bar, between 0 and range inclusive

range

Maximum value

Description

Refer to CMD_PROGRESS for more information on physical dimension.

Command layout

+0	CMD_SCROLLBAR(0xFFFF FF11)
+4	x
+6	y
+8	w
+10	h
+12	options
+14	val
+16	size
+18	range

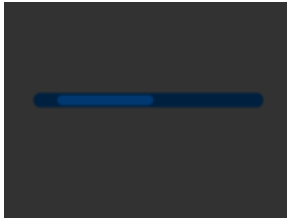
Examples

A scroll bar indicating 10-50%:



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

Without the 3D look:



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

A brown-themed vertical scroll bar:



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

5.38 CMD_SLIDER

This command is to draw a slider.

C prototype

```
void cmd_slider( int16_t x,  
                int16_t y,  
                uint16_t w,  
                uint16_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

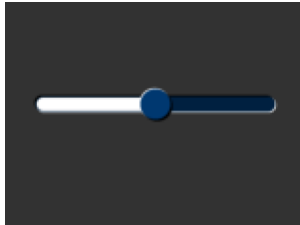
Refer to CMD_PROGRESS for more information on physical dimension.

Command layout

+0	CMD_SLIDER(0xFFFF FF10)
+4	x
+6	y
+8	w
+10	h
+12	options
+14	val
+16	range

Examples

A slider set to 50%:



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

Without the 3D look:



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

A brown-themed vertical slider with range 0-65535:



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


5.39 CMD_DIAL

This command is used to draw a rotary dial control.

C prototype

```
void cmd_dial( int16_t x,
               int16_t y,
               uint16_t r,
               uint16_t options,
               uint16_t val );
```

Parameters

x

x-coordinate of dial center, in pixels

y

y-coordinate of dial center, in pixels

r

radius of dial, in pixels.

options

By default, the dial is drawn with a 3D effect and the value of options is zero. Options OPT_FLAT remove the 3D effect and its value is 256

val

Specify the position of dial points by setting value between 0 and 65535 inclusive. 0 means that the dial points straight down, 0x4000 left, 0x8000 up, and 0xc000 right.

Description

The details of physical dimension are

- The marker is a line of width $r \cdot (12/256)$, drawn at a distance $r \cdot (140/256)$ to $r \cdot (210/256)$ from the center

Refer to [Coproprocessor engine widgets physical dimensions](#) for more information.

Command layout

+0	CMD_DIAL(0xFFFF FF2D)
+4	x
+6	y
+8	r
+10	options
+12	val

Examples

A dial set to 50%:



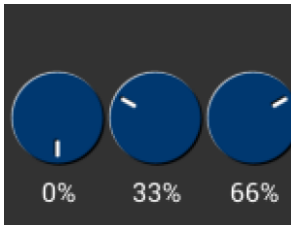
```
cmd_dial(80, 60, 55, 0, 0x8000);
```

Without the 3D look:



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

Dials set to 0%, 33% and 66%:



```
cmd_dial(28, 60, 24, 0, 0x0000);
cmd_text(28, 100, 26, OPT_CENTER, "0%");
cmd_dial(80, 60, 24, 0, 0x5555);
cmd_text(80, 100, 26, OPT_CENTER, "33%");
cmd_dial(132, 60, 24, 0, 0xaaaa);
cmd_text(132, 100, 26, OPT_CENTER, "66%");
```

5.40 CMD_TOGGLE

This command is used to draw a toggle switch with labels.

C prototype

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

Parameters

x
x-coordinate of top-left of toggle, in pixel

y
y-coordinate of top-left of toggle, in pixels

w
width of toggle, in pixels

font
Font to use for text, 0-31. See [ROM and RAM Fonts](#)

options
By default, the toggle is drawn with a 3D effect and the value of options is zero. Options **OPT_FLAT** remove the 3D effect and its value is 256

state

State of the toggle: 0 is off, 65535 is on.

s

string labels for toggle. A character value of 255 (in C it can be written as '\xff') separates the label strings.

Description

The details of physical dimension are:

- Widget height (h) is font height * (20/16) pixel.
- Outer bar radius (r) is font height * (10/16) pixel.
- Knob radius is (r-1.5) pixel, where r is the outer bar radius above.
- The center of outer bar's left round head is at (x, y + r/2) coordinate.

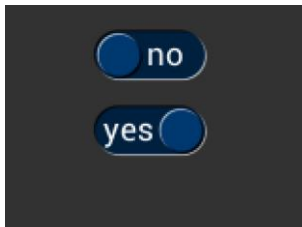
Refer to [Coprocessor engine widgets physical dimensions](#) for more information.

Command layout

+0	CMD_TOGGLE(0xFFFF FF12)
+4	x
+6	y
+8	w
+10	font
+12	options
+14	state
+16	s

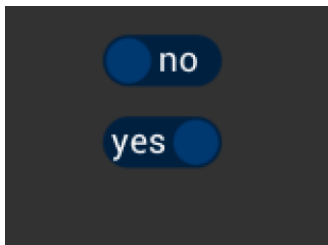
Examples

Using a medium font, in the two states:



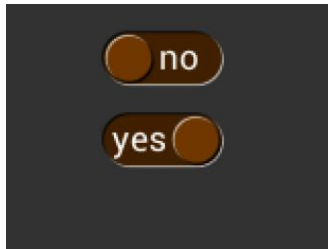
```
cmd_toggle(60, 20, 33, 27, 0, 0, "no" "\xff" "yes");
cmd_toggle(60, 60, 33, 27, 0, 65535, "no" "\xff" "yes");
```

Without the 3D look:



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

With different background and foreground colors:



```
cmd_bgcolor(0x402000);
cmd_fgcolor(0x703800);
cmd_toggle(60, 20, 33, 27, 0, 0, "no" "\xff" "yes");
cmd_toggle(60, 60, 33, 27, 0, 65535, "no" "\xff" "yes");
```

5.41 CMD_TEXT

This command is used to draw a text string.

C prototype

```
void cmd_text(    int16_t x,
                  int16_t y,
                  uint16_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.

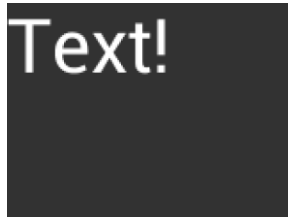
s
 Text string, the text string itself which should be terminated by a null character (ASCII code 0).

Command layout

+0	CMD_TEXT(0xFFFF FF0C)
+4	x
+6	y
+8	font
+10	options
+12	s
..	..
..	0 (null character to terminate string)

Examples

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



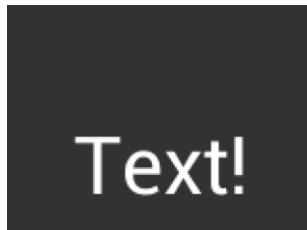
```
cmd_text(0, 0, 31, 0, "Text!");
```

Using a smaller font:



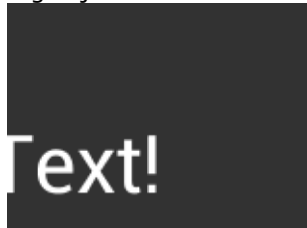
```
cmd_text(0, 0, 26, 0, "Text!");
```

Centered horizontally:



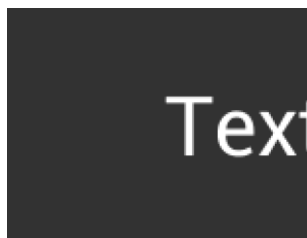
```
cmd_text(80, 60, 31, OPT_CENTERX, "Text!");
```

Right-justified:



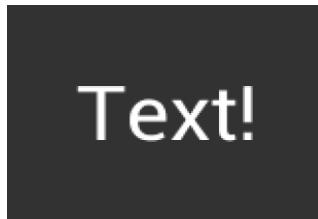
```
cmd_text(80, 60, 31, OPT_RIGHTX, "Text!");
```

Centered vertically:



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

Centered both horizontally and vertically:



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

Text with explicit newline:



```
cmd_text(80, 60, 29, OPT_CENTER, "one two\nthree four");
```

5.42 CMD_SETBASE

This command is used to set the base for number output.

C prototype

```
void cmd_setbase( uint32_t b );
```

Parameters

b
 Numeric base, valid values are from 2 to 36:
 2 for binary,
 8 for octal,
 10 for decimal,
 16 for hexadecimal

Description

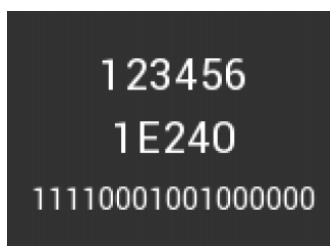
Set up numeric base for **CMD_NUMBER**

Command layout

+0	CMD_SETBASE(0xFFFF FF38)
+4	b

Examples

The number 123456 displayed in decimal, hexadecimal and binary:



```
cmd_number(80, 30, 28, OPT_CENTER, 123456);
cmd_setbase(16);
cmd_number(80, 60, 28, OPT_CENTER, 123456);
cmd_setbase(2);
cmd_number(80, 90, 26, OPT_CENTER, 123456);
```

5.43 CMD_NUMBER

This command is used to draw a number.

C prototype

```
void cmd_number(int16_t x,
               int16_t y,
               uint16_t font,
               uint16_t options,
               int32_t n );
```

Parameters

x

x-coordinate of text base, in pixels

y

y-coordinate of text base, in pixels

font

font to use for text, 0-31. See ROM and RAM Fonts

options

By default (x,y) is the top-left pixel of the text.

OPT_CENTERX centers the text horizontally,

OPT_CENTERY centers it vertically.

OPT_CENTER centers the text in both directions.

OPT_RIGHTX right-justifies the text, so that the x is the rightmost pixel.

By default, the number is displayed with no leading zeroes, but if a width 1-9 is specified in the options, then the number is padded if necessary, with leading zeroes so that it has the given width. If **OPT_SIGNED** is given, the number is treated as signed, and prefixed by a minus sign if negative.

n

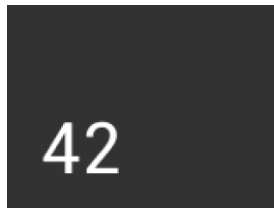
The number to display, is either unsigned or signed 32-bit, in the base specified in the preceding [CMD_SETBASE](#). If no **CMD_SETBASE** appears before **CMD_NUMBER**, it will be in decimal base.

Command layout

+0	CMD_NUMBER(0xFFFF FF2E)
+4	x
+6	y
+8	font
+10	options
+12	n

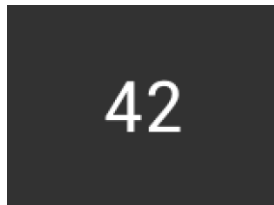
Examples

A number:



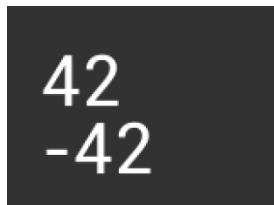
```
cmd_number(20, 60, 31, 0, 42);
```

Centered:



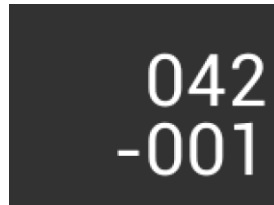
```
cmd_number(80, 60, 31, OPT_CENTER, 42);
```

Signed output of positive and negative numbers:



```
cmd_number(20, 20, 31, OPT_SIGNED, 42);  
cmd_number(20, 60, 31, OPT_SIGNED, -42);
```

Forcing width to 3 digits, right-justified



```
cmd_number(150, 20, 31, OPT_RIGHTX | 3, 42);  
cmd_number(150, 60, 31, OPT_SIGNED | OPT_RIGHTX | 3, -1);
```

5.44 CMD_LOADIDENTITY

This command instructs the coprocessor engine to set the current matrix to the identity matrix, so that the coprocessor engine is able to form the new matrix as requested by **CMD_SCALE**, **CMD_ROTATE**, **CMD_TRANSLATE** command.

For more information on the identity matrix, refer to the [Bitmap Transformation Matrix](#) section.

C prototype

```
void cmd_loadidentity( );
```

Command layout

+0	CMD_LOADIDENTITY(0xFFFF FF26)
----	--------------------------------------

5.45 CMD_SETMATRIX

The coprocessor 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(0xFFFF FF2A)
----	----------------------------

5.46 CMD_GETMATRIX

This command retrieves the current matrix within the context of the coprocessor engine. Note the matrix within the context of the coprocessor engine will not apply to the bitmap transformation until it is passed to the graphics engine through **CMD_SETMATRIX**.

C prototype

```
void cmd_getmatrix( int32_t a,
                   int32_t b,
                   int32_t c,
                   int32_t d,
                   int32_t e,
                   int32_t f );
```

Parameters

- a**
output parameter; written with matrix coefficient a. See the parameter of the command **BITMAP_TRANSFORM_A** for formatting.
- b**
output parameter; written with matrix coefficient b. See the parameter b of the command **BITMAP_TRANSFORM_B** for formatting.
- c**
output parameter; written with matrix coefficient c. See the parameter c of the command **BITMAP_TRANSFORM_C** for formatting.
- d**
output parameter; written with matrix coefficient d. See the parameter d of the command **BITMAP_TRANSFORM_D** for formatting.
- e**
output parameter; written with matrix coefficient e. See the parameter e of the command **BITMAP_TRANSFORM_E** for formatting.
- f**
output parameter; written with matrix coefficient f. See the parameter f of the command **BITMAP_TRANSFORM_F** for formatting.

Command layout

+0	CMD_GETMATRIX(0xFFFF FF33)
+4	a
+8	b
+12	c
+16	d
+20	e
+24	f

5.47 CMD_GETPTR

This command returns the end memory address of data inflated by **CMD_INFLATE**.

C prototype

```
void cmd_getptr( uint32_t result );
```

Parameters

result

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

Command layout

+0	CMD_GETPTR (0xFFFF FF23)
+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) % 4096);
```

5.48 CMD_GETPROPS

This command returns the source address and size of the bitmap loaded by the previous **CMD_LOADIMAGE**.

C prototype

```
void cmd_getprops( uint32_t ptr, uint32_t width, uint32_t height);
```

Parameters

ptr

The address of the image in RAM_G which was decompressed by the last **CMD_LOADIMAGE** before this command.

It is an output parameter.

width

The width of the image which was decoded by the last **CMD_LOADIMAGE** before this command.

It is an output parameter.

height

The height of the image which was decoded by the last **CMD_LOADIMAGE** before this command.

It is an output parameter

Command layout

+0	CMD_GETPROPS (0xFFFF FF25)
+4	wtr
+8	width
+12	height

Description

This command is used to retrieve the properties of the image which is decoded by **CMD_LOADIMAGE**. Respective image properties are updated by the coprocessor after this command is executed successfully.

Examples

Please refer to the [CMD_GETPTR](#).

5.49 CMD_SCALE

This command is used to 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

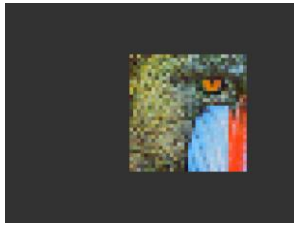
y scale factor, in signed 16. 16-bit fixed-point form.

Command layout

+0	CMD_SCALE(0xFFFF FF28)
+4	sx
+8	sy

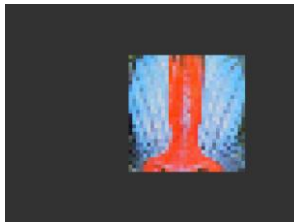
Examples

To zoom a bitmap 2X:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_scale(2 * 65536, 2 * 65536);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To zoom a bitmap 2X around its center:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(65536 * 32, 65536 * 32);
cmd_scale(2 * 65536, 2 * 65536);
cmd_translate(65536 * -32, 65536 * -32);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

5.50 CMD_ROTATE

This command is used to apply a rotation to the current matrix.

C prototype

```
void cmd_rotate( int32_t a );
```

Parameters

a

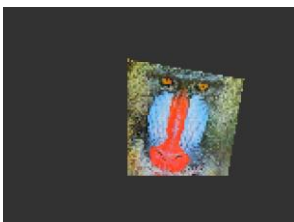
Clockwise rotation angle, in units of 1/65536 of a circle

Command layout

+0	CMD_ROTATE(0xFFFF FF29)
+4	a

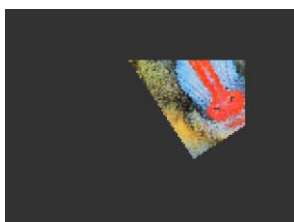
Examples

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 around the top left of the bitmap:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotate(-33 * 65536 / 360);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

Rotating a 64 x 64 bitmap around its center:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(65536 * 32, 65536 * 32);
cmd_rotate(90 * 65536 / 360);
cmd_translate(65536 * -32, 65536 * -32);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

5.51 CMD_TRANSLATE

This command is used to 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.

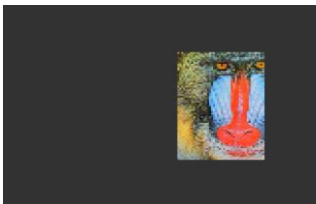
ty
 y translate factor, in signed 16.16-bit fixed-point form.

Command layout

+0	CMD_TRANSLATE(0xFFFF FF27)
+4	tx
+8	ty

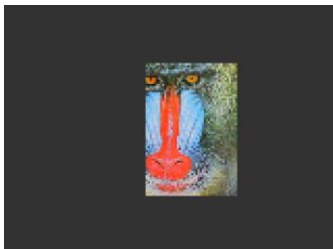
Examples

To translate the bitmap 20 pixels to the right:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(20 * 65536, 0);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To translate the bitmap 20 pixels to the left:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_translate(-20 * 65536, 0);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

5.52 CMD_CALIBRATE

This command is used to execute the touch screen calibration routine. The calibration procedure collects three touches from the touch screen, then computes and loads an appropriate matrix into **REG_TOUCH_TRANSFORM_A-F**. To use the function, create a display list and include **CMD_CALIBRATE**. The coprocessor engine overlays the touch targets on the current display list, gathers the calibration input and updates **REG_TOUCH_TRANSFORM_A-F**. There is no need to add the **DISPLAY** command and swap the frame by software because coprocessor engine will do it once this command is received.

Please note that this command only applies to **RTSE** or the compatibility mode of CTSE.

C prototype

```
void cmd_calibrate( uint32_t result );
```

Parameters

result

output parameter; written with 0 on failure of calibration.

Description

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

Command layout

+0	CMD_CALIBRATE(0xFFFF FF15)
+4	result

Examples

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

5.53 CMD_SETROTATE

This command is used to rotate the screen.

C prototype

```
void cmd_setrotate( uint32_t r );
```

Parameters

r

The value from 0 to 7. The same definition as the value in **REG_ROTATE**. Refer to the section [Rotation](#) for more details.

Description

CMD_SETROTATE sets **REG_ROTATE** to the given value *r*, causing the screen to rotate. It also appropriately adjusts the touch transform matrix so that coordinates of touch points are adjusted to rotated coordinate system.

Command layout

+0	CMD_SETROTATE (0xFFFF FF36)
+4	r

Examples

```
cmd_setrotate(2); //Put the display in portrait mode
```

5.54 CMD_SPINNER

This command is used to start an animated spinner. The spinner is an animated overlay that shows the user that some tasks is continuing. To trigger the spinner, create a display list and then use **CMD_SPINNER**. The coprocessor 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** register is utilized to perform the animation kind of effect. The frequency of point's movement is with respect to the display frame rate configured.

Typically for 480x272 display panels the display rate is ~60fps. For style 0 and 60fps, the point repeats the sequence within 2 seconds. For style 1 and 60fps, the point repeats the sequence within 1.25 seconds. For style 2 and 60fps, the clock hand repeats the sequence within 2 seconds. For style 3 and 60fps, the moving dots repeat the sequence within 1 second. Note that only one of **CMD_SKETCH**, **CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_spinner( int16_t x,
                 int16_t y,
                 uint16_t style,
                 uint16_t scale );
```

Command layout

+0	CMD_SPINNER(0xFFFF FF16)
+4	x
+6	y
+8	style
+10	scale

Parameters

x
The X coordinate of top left of spinner

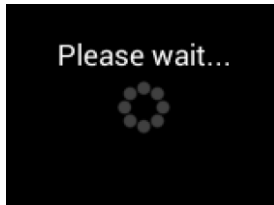
y
The Y coordinate of top left of spinner

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

scale
The scaling coefficient of spinner. 0 means no scaling.

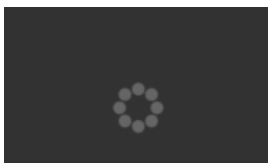
Examples

Create a display list, then start the spinner:

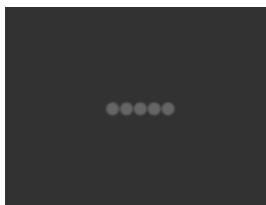


```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80,30,27, OPT_CENTER, "Please wait...");
cmd_spinner(80, 60, 0, 0);
```

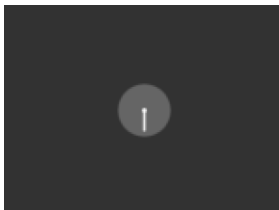
Spinner style 0, a circle of dots:



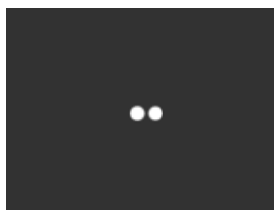
Style 1, a line of dots:



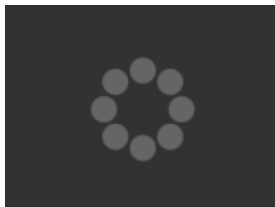
Style 2, a rotating clock hand:



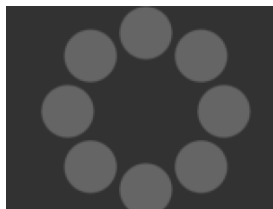
Style 3, two orbiting dots:



Half screen, scale:



Full screen, scale 2:



5.55 CMD_SCREENSAVER

This command is used to start an animated screensaver. After the screensaver command, the coprocessor engine continuously updates **REG_MACRO_0** with **VERTEX2F** with varying (x,y) coordinates. By using a suitable display list, a bitmap can be moved across the screen without involving the host. Command **CMD_STOP** stops the update process.

Note that only one of **CMD_SKETCH**, **CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_screensaver( );
```


Description

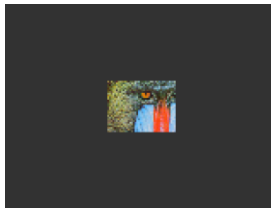
REG_MACRO_0 is updated with respect to frame rate (depending on the display registers configuration). Typically for a 480x272 display the frame rate is around 60 frames per second.

Command layout

+0	CMD_SCREENSAVER(0xFFFF FF2F)
----	-------------------------------------

Examples

To start the screensaver, create a display list using a MACRO instruction – the coprocessor engine will update it continuously:



```
cmd_screensaver();
cmd(BITMAP_SOURCE(0));
cmd(BITMAP_LAYOUT(0, 128, 64));
cmd(BITMAP_SIZE(NEAREST, BORDER, BORDER, 40, 30));
cmd(BEGIN(BITMAPS));
cmd(MACRO(0));
cmd(DISPLAY());
```

5.56 CMD_SKETCH

This command is used to start a continuous sketch update. The Coprocessor engine continuously samples the touch inputs and paints pixels into a bitmap, according to the given touch (x, y). This implies that the user's touch inputs can be incorporated into the bitmap without requiring any processing by the host. **CMD_STOP** is to be sent to stop the sketch process.

Note that only one of **CMD_SKETCH**, **CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_sketch( int16_t x,
                 int16_t y,
                 uint16_t w,
                 uint16_t h,
                 uint32_t ptr,
                 uint16_t format );
```

Parameters

- x**
x-coordinate of sketch area top-left, in pixels
- y**
y-coordinate of sketch area top-left, in pixels

w

Width of sketch area, in pixels

h

Height of sketch area, in pixels

ptr

Base address of sketch bitmap

format

Format of sketch bitmap, either L1 or L8

Note: The update frequency of bitmap data located at ptr depends on the sampling frequency of the built-in ADC circuit, which is up to 1000 Hz.

Command layout

+0	CMD_SKETCH(0xFFFF FF30)
+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();
```

5.57 CMD_STOP

This command is to inform the coprocessor engine to stop the periodic operation, which is triggered by **CMD_SKETCH**, **CMD_SPINNER** or **CMD_SCREENSAVER**.

C prototype

```
void cmd_stop( );
```

Command layout

+0	CMD_STOP(0xFFFF FF17)
-----------	------------------------------

Description

For **CMD_SPINNER** and **CMD_SCREENSAVER**, **REG_MACRO_0** and **REG_MACRO_1** updating will be stopped.

For **CMD_SKETCH**, the bitmap data in **RAM_G** updating will be stopped.

Examples

See [CMD_SKETCH](#), [CMD_SPINNER](#), [CMD_SCREENSAVER](#).

5.58 CMD_SETFONT

CMD_SETFONT is used to register one custom defined bitmap font into the coprocessor engine. After registration, the coprocessor engine is able to use the bitmap font with corresponding commands.

Note that **CMD_SETFONT** does not set up the font's bitmap parameters. The host shall do this before using the font. For further details about how to set up a custom font, refer to ROM and RAM Fonts.

C prototype

```
void cmd_setfont( uint32_t font,
                  uint32_t ptr );
```

Command layout

+0	CMD_SETFONT(0xFFFF FF2B)
+4	font
+8	ptr

Parameters

font

The bitmap handle from 0 to 31

ptr

The metrics block address in **RAM_G**. 4 bytes aligned is required.

Examples

With a suitable font metrics block loaded in **RAM_G** at address 1000, to set it up for use with objects as font 7:

```
cmd(BITMAP_LAYOUT(L8,16,10));
cmd(BITMAP_SIZE(NEAREST,BORDER,BORDER,16,10));
cmd(BITMAP_SOURCE(1000));
cmd_setfont(7,1000);
cmd_button(20,20,120,40,7,0,"custom font!");
// x,y
// width,height in pixels
// font 7, just loaded
// default options,3D style
```

5.59 CMD_SETFONT2

This command is used to set up a custom font. To use a custom font with the coprocessor objects, create the font definition data in **RAM_G** and issue **CMD_SETFONT2**, as described in ROM and RAM Fonts.

Note that **CMD_SETFONT2** sets up the font's bitmap parameters by appending commands **BITMAP_SOURCE**, **BITMAP_LAYOUT** and **BITMAP_SIZE** to the current display list.

For details about how to set up a custom font, refer to [ROM and RAM Fonts](#).

C prototype

```
void cmd_setfont2 ( uint32_t font,
                   uint32_t ptr,
                   uint32_t firstchar );
```

Command layout

+0	CMD_SETFONT2(0xFFFF FF3B)
+4	font
+8	ptr
+12	firstchar

Parameters

font

The bitmap handle from 0 to 31

ptr

32-bit aligned memory address in **RAM_G** of font metrics block

firstchar

The **ASCII** value of first character in the font. **For an extended font block, this should be zero.**

Examples

With a suitable font metrics block loaded in RAM_G at address 100000, first character's ASCII value 32, to use it for font 20:



```
cmd_setfont2(20, 100000, 32);
cmd_button(15, 30, 130, 20, 18, 0, "This is font 18");
cmd_button(15, 60, 130, 20, 20, 0, "This is font 20");
```

5.60 CMD_SETSCRATCH

This command is employed to designate the scratch bitmap for widget utilization. Graphic elements rely on a bitmap handle for rendering, typically using handle 15 as a default. However, this command grants the ability to assign a different bitmap handle, thereby ensuring the secure utilization of bitmap handle 15 by the user.

C prototype

```
void cmd_setscratch( uint32_t handle);
```

Parameters

handle

bitmap handle number, 0~31

Command layout

+0	CMD_SETSCRATCH (0xFFFF FF3C)
+4	handle

Examples

With the **cmd_setscratch** command, set the handle 31 as scratch bitmap handle so that bitmap handle 15 is available for application use as a custom font handle:



```
cmd_setscratch(31);
cmd_setfont2(15, 100000, 32);
cmd_button(15, 30, 130, 20, 15, 0, "This is font 15");

//Restore bitmap handle 31 to ROM Font number 31.
cmd_romfont(31, 31);
```

5.61 CMD_ROMFONT

This command is to load a ROM font into bitmap handle. By default, ROM fonts 16-31 are loaded into bitmap handles 16-31. This command allows any ROM font 16-34 to be loaded into any bitmap handle.

C prototype

```
void cmd_romfont (uint32_t font,
                  uint32_t romslot );
```

Parameters

font

bitmap handle number , 0~31

romslot

ROM font number, 16~34

Command layout

+0	CMD_ROMFONT (0xFFFF FF3F)
+4	font
+8	romslot

Examples

Loading hardware fonts 31-34 into bitmap handle 1:



```
cmd_romfont(1, 31);  
cmd_text( 0, 0, 1, 0, "31");  
cmd_romfont(1, 32);  
cmd_text( 0, 60, 1, 0, "32");  
cmd_romfont(1, 33);  
cmd_text(80, -14, 1, 0, "33");  
cmd_romfont(1, 34);  
cmd_text(60, 32, 1, 0, "34");
```

5.62 CMD_TRACK

This command is used to track touches for a graphics object. **EVE** can assist the host in tracking touches on graphical objects. For example, touches on dial objects can be reported as angles, saving the computation power of the host. To do this, the host draws the object using a chosen tag value, and registers a track area for that tag. From then on, any touch on that object is reported in **REG_TRACKER**, and multiple touches (if supported by the touch panel) in **REG_TRACKER_1**, **REG_TRACKER_2**, **REG_TRACKER_3**, **REG_TRACKER_4**.

The host can detect any touch on the object by reading the 32-bit value in the five registers above. The low 8 bits give the current tag, or zero if there is no touch. The high sixteen bits give the tracked value. For a rotary tracker - used for clocks, gauges and dials - this value is the angle of the touch point relative to the object center, in units of 1/65536 of a circle. 0 means that the angle is straight down, 0x4000 left, 0x8000 up, and 0xc000 right.

For a linear tracker - used for sliders and scrollbars - this value is the distance along the tracked object, from 0 to 65535.

Note: Multiple touch points are only available with capacitive touch screen connected.

C prototype

```
void cmd_track(  int16_t x,  
                 int16_t y,  
                 uint16_t w,  
                 uint16_t h,  
                 uint16_t tag );
```

Parameters

x

For linear tracker functionality, x-coordinate of track area top-left, in pixels.
For rotary tracker functionality, x-coordinate of track area center, in pixels.

y

For linear tracker functionality, y-coordinate of track area top-left, in pixels.
For rotary tracker functionality, y-coordinate of track area center, in pixels.

w

Width of track area, in pixels.

h

Height of track area, in pixels.

Note: A w and h of (1,1) means that the tracker is rotary, and reports an angle value in **REG_TRACKER**. A w and h of (0,0) disables the track functionality of the coprocessor engine. Other values mean that the tracker is linear, and reports values along its length from 0 to 65535 in **REG_TRACKER**

tag

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

Command layout

+0	CMD_TRACK(0xFFFF FF2C)
+4	x
+6	y
+8	w
+10	h
+12	tag

Description

The Coprocessor engine tracks the graphics object in rotary tracker mode and linear tracker mode:

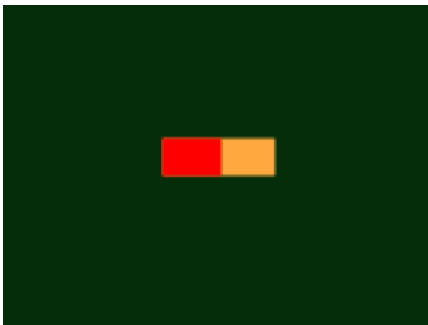
- rotary tracker mode – Track the angle between the touch point and the center of the graphics object specified by the tag value. The value is in units of 1/65536 of a circle. 0 means that the angle is straight down, 0x4000 left, 0x8000 up, and 0xC000 right from the center.

Linear tracker mode – If parameter w is greater than h, track the relative distance of the touch point to the width of the graphics object specified by the tag value. If parameter w is not greater than h, track the relative distance of touch points to the height of the graphics object specified by the tag value. The value is in units of 1/65536 of the width or height of the graphics object. The distance of the touch point refers to the distance from the top left pixel of graphics object to the coordinate of the touch point.

Please note that the behavior of **CMD_TRACK** is not defined if the center of the track object (in case of rotary track) or top left of the track object (in case of linear track) is outside the visible region in display panel.

Examples

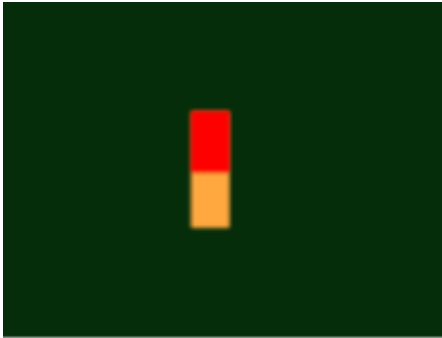
Horizontal track of rectangle dimension 40x12 pixels and the present touch is at 50%:



```

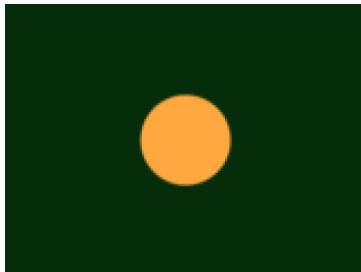
dl ( CLEAR_COLOR_RGB(5, 45, 110) );
dl ( COLOR_RGB(255, 168, 64) );
dl ( CLEAR(1, 1, 1) );
dl ( BEGIN(RECTS) );
dl ( VERTEX2F(60 * 16, 50 * 16) );
dl ( VERTEX2F(100 * 16, 62 * 16) );
dl ( COLOR_RGB(255, 0, 0) );
dl ( VERTEX2F(60 * 16, 50 * 16) );
dl ( VERTEX2F(80 * 16, 62 * 16) );
dl ( COLOR_MASK(0,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

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);
```

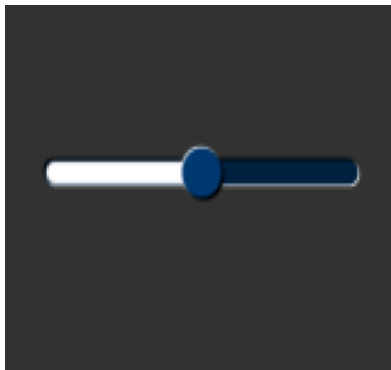
To draw a dial with tag 33 centered at (80, 60), adjustable by touch:



```
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, val, 65535);

  uint32_t tracker = rd32(REG_TRACKER);
  if ((tracker & 0xff) == 34)
    val = tracker >> 16;
}
```


5.63 CMD_SNAPSHOT

This command causes the coprocessor engine to take a snapshot of the current screen, and write the result into **RAM_G** as an **ARGB4** bitmap. The size of the bitmap is the size of the screen, given by the **REG_HSIZE** and **REG_VSIZE** registers.

During the snapshot process, the display should be disabled by setting **REG_PCLK** to 0 to avoid display glitch. Since the coprocessor engine needs to write the result into the destination address, the destination address must never be used or referenced by the graphics engine.

C prototype

```
void cmd_snapshot( uint32_t ptr );
```

Parameters

ptr
 Snapshot destination address, in **RAM_G**

Command layout

+0	CMD_SNAPSHOT(0xFFFF FF1F)
+4	ptr

Examples

To take a snapshot of the current 160 x 120 screens, then use it as a bitmap in the new display list:

```
wr(REG_PCLK,0); //Turn off the PCLK
wr16(REG_HSIZE,120);
wr16(REG_WSIZE,160);

cmd_snapshot(0); //Taking snapshot.

wr(REG_PCLK,5); //Turn on the PCLK
wr16(REG_HSIZE,272);
wr16(REG_WSIZE,480);

cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd(BITMAP_SOURCE(0));
cmd(BITMAP_LAYOUT(ARGB4, 2 * 160, 120));
cmd(BITMAP_SIZE(NEAREST, BORDER, BORDER, 160, 120));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 0, 0));
```

5.64 CMD_SNAPSHOT2

The snapshot command causes the coprocessor to take a snapshot of part of the current screen, and write it into graphics memory (**RAM_G**) as a bitmap. The size, position and format of the bitmap may be specified. During the snapshot process, the display output process is suspended. LCD displays can easily tolerate variation in display timing, so there is no noticeable flicker.

C prototype

```
void cmd_snapshot2( uint32_t fmt,
                   uint32_t ptr,
                   int16_t x,
```

```
int16_t y,
uint16_t w,
uint16_t h);
```

Parameters

fmt

Output bitmap format, one of **RGB565**, **ARGB4** or **0x20**. The value **0x20** produces an **ARGB8** format snapshot.

Refer to [BITMAP_LAYOUT](#) for format List.

Ptr

Snapshot destination address, in **RAM_G**

x

x-coordinate of snapshot area top-left, in pixels

y

y-coordinate of snapshot area top-left, in pixels

w

width of snapshot area, in pixels. Note when *fmt* is **0x20**, i.e., in **ARGB8** format, the value of width shall be doubled.

H

height of snapshot area, in pixels

Command layout

+0	CMD_SNAPSHOT2(0xFFFF FF37)
+4	fmt
+8	ptr
+12	x
+14	y
+16	w
+18	h

Examples

To take a 32x32 snapshot of the top-left of the screen, then use it as a bitmap in the new display list:

```
cmd_snapshot2(RGB565, 0, 0, 0, 32, 32);
cmd_dlstart();
cmd_setbitmap(0, RGB565, 32, 32);
cmd(CLEAR(1,1,1));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 0, 0));
```

Note: For ARGB8 format, pixel memory layout is as below:

ARGB8 Pixel Format							
31	24	23	16	15	8	7	0
Alpha Channel		Red Channel		Green Channel		Blue Channel	

5.65 CMD_SETBITMAP

This command will generate the corresponding display list commands for given bitmap information, sparing the effort of writing display list manually. The display list commands to be generated candidates are as below:

- **BITMAP_SOURCE**
- **BITMAP_LAYOUT/ BITMAP_LAYOUT_H**
- **BITMAP_SIZE/ BITMAP_SIZE_H**
- **BITMAP_FORMAT**

The parameters filter/wrapx/wrapy in **BITAMP_SIZE** is always set to **NEAREST/BORDER/BORDER** value in the generated display list commands.

C prototype

```
void cmd_setbitmap(  uint32_t  source,
                    uint16_t  fmt,
                    uint16_t  width,
                    uint16_t  height );
```

Parameters

source

Source address for bitmap, in **RAM_G** memory, as a **BITMAP_SOURCE** parameter.

Fmt

bitmap format, see the definition in 4.7.

width

bitmap width, in pixels. 2 bytes value.

Height

bitmap height, in pixels. 2 bytes value.

Command layout

+0	CMD_SETBITMAP(0xFFFF FF43)
+4	source
+8	fmt
+10	width
+12	height

Examples

Display a bitmap with width 35 and height 35 pixels and loaded at **RAM_G**:

```
cmd_dlstart();
cmd(BITMAPS(2)); //Use bitmap handle 2
cmd_setbitmap(RAM_G, RGB565, 35, 35); //Set up parameters for
bitmap handle 2
cmd(CLEAR(1,1,1));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 2, 0)); //draw image with bitmap handle 2
cmd_swap();
```

Note:

Two bytes needs to be appended after last parameter for 4 bytes alignment.

When format is **PALETTED4444/PALETTED8/PALETTED565**, due to no display list commands **PALETTE_SOURCE** is generated, user need write the **PALETTE_SOURCE** command manually.

```
cmd_dlstart();
cmd(BITMAPS(2)); //Use bitmap handle 2
cmd_setbitmap(RAM_G, PALETTED565, 35, 35); //Set up parameters for
bitmap handle 2
cmd(PALETTE_SOURCE(RAM_G + 35*35)); //Specify the address of
palette(color table), assuming it comes right after the index
cmd(CLEAR(1,1,1));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 2, 0)); //draw image with bitmap handle 2
cmd_swap();
```

5.66 CMD_LOGO



The logo command causes the coprocessor engine to play back a short animation of the **Bridgetek** logo for **BT88x** or **FTDI Chip** logo for **FT81x**. During logo playback the host shall not write or render any display list. After 2.5 seconds have elapsed, the coprocessor engine writes zero to **REG_CMD_READ** and **REG_CMD_WRITE**, and starts waiting for commands. After this command is complete, the host shall write the next command to the starting address of **RAM_CMD**.

C prototype

```
void cmd_logo( );
```

Command layout

+0	CMD_LOGO(0xFFFF FF31)
----	------------------------------

Examples

To play back the logo animation:

```
cmd_logo();
delay(3000); // Optional to wait
//Wait till both read & write pointer register are equal.
While(rd16(REG_CMD_WRITE) != rd16(REG_CMD_READ));
```

5.67 CMD_SYNC

This command waits for the end of the video scan out period, then it returns immediately. It may be used to synchronize screen updates that are not part of a display list, and to accurately measure the time taken to render a frame.

C prototype

```
void cmd_sync( );
```

Command layout

+0	CMD_SYNC(0xFFFF FF42)
----	-----------------------

Examples

```
//To synchronize with a frame:
cmd_sync();

//To update REG_MACRO_0 at the end of scan out, to avoid tearing:
cmd_sync();
cmd_memwrite(REG_MACRO_0, 4);
cmd(value);

//To measure frame duration
cmd_sync();
cmd_memcpy(0, REG_CLOCK, 4);
cmd_sync();
cmd_memcpy(4, REG_CLOCK, 4);
//the difference between locations 4 and 0 give the frame interval in clocks.
```

5.68 CMD_BITMAP_TRANSFORM

This command computes a bitmap transform and appends **BITMAP_TRANSFORM_A** – **BITMAP_TRANSFORM_F** to the display list. Given three corresponding points in screen space and bitmap space, it calculates the transform. Using these three points, the command computes a matrix that transforms the bitmap coordinates into screen space, and appends the display list commands to set the bitmap matrix.

C prototype

```
void cmd_bitmap_transform( int32_t x0,
                          int32_t y0,
                          int32_t x1,
                          int32_t y1,
                          int32_t x2,
                          int32_t y2,
                          int32_t tx0,
                          int32_t ty0,
                          int32_t tx1,
                          int32_t ty1,
                          int32_t tx2,
                          int32_t ty2,
                          uint16_t result )
```

Command layout

+0	CMD_BITMAP_TRANSFORM(0xFFFF FF21)
+4	X ₀
+8	Y ₀
+10	X ₁
+16	Y ₁
+20	X ₂
+24	Y ₂
+28	tx ₀
+32	ty ₀
+36	tx ₁
+40	ty ₁
+44	tx ₂
+48	ty ₂
+52	result

Parameters

X₀,Y₀

Point 0 screen coordinate, in pixels

X₁,Y₁

Point 1 screen coordinate, in pixels

X₂,Y₂

Point 2 screen coordinate, in pixels

tx₀,ty₀

Point 0 bitmap coordinate, in pixels

tx₁,ty₁

Point 1 bitmap coordinate, in pixels

tx₂,ty₂

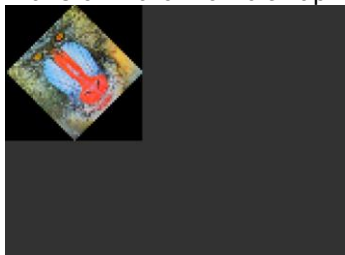
Point 2 bitmap coordinate, in pixels

result

result return. Set to -1 on success, or 0 if it is not possible to find the solution matrix.

Examples

Transform a 64x64 bitmap:



```
cmd(BLEND_FUNC(ONE, ZERO));
cmd_bitmap_transform(32,0, 64,32, 32,64,0,0,
0,64, 64,64, 0);
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(0, 0, 0, 0));
```

5.69 CMD_CSKETCH - Deprecated

This command is the legacy command from the FT801 chip. Users are recommended to use "CMD_SKETCH" for **FT81X/BT88x** since it works for both **RTSE** and **CTSE**.

C prototype

```
void cmd_csketch( int16_t x,
                  int16_t y,
                  uint16_t w,
                  uint16_t h,
                  uint32_t ptr,
                  uint16_t format,
                  uint16_t freq );
```

Command layout

+0	CMD_CSKETCH (0xFFFF FF35)
+4	x
+6	y
+8	w
+10	h
+12	ptr
+16	format
+18	freq

Parameters

x x-coordinate of sketch area, top-left, in pixels

y y-coordinate of sketch area, top-left, in pixels

w width of sketch area, in pixels

h height of sketch area, in pixels

ptr base address of sketch bitmap

format format of sketch bitmap, either **L1** or **L8**

freq Deprecated

6 Contact Information

Refer to <https://brtchip.com/contact-us/> for contact information

Distributor and Sales Representatives

Please visit the Sales Network page of the [Bridgetek Web site](#) for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Bridgetek Pte Ltd (BRTChip) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested Bridgetek devices and other materials) is provided for reference only. While Bridgetek has taken care to assure it is accurate, this information is subject to customer confirmation, and Bridgetek disclaims all liability for system designs and for any applications assistance provided by Bridgetek. Use of Bridgetek devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless Bridgetek from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Bridgetek Pte Ltd, 1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464. Singapore Registered Company Number: 201542387H.

Appendix A – References

Document References

[FT81x Datasheet](#)
[BT88x Datasheet](#)
[OpenGL 4.5 Reference Pages](#)

Acronyms and Abbreviations

Terms	Description
ADC	Analog-to-digital
API	Application Programming Interface
AVI	Audio Video Interactive
ASCII	American Standard Code for Information Interchange
CTPM	Capacitive Touch Panel Module
CTSE	Capacitive Touch Screen Engine
EVE	Embedded Video Engine
FIFO	First In First Out buffer
I ² C	Inter-Integrated Circuit
JPEG	Joint Photographic Experts Group
LCD	Liquid Crystal Display
MCU	Micro controller unit
MPU	Microprocessor Unit
PCB	Printed Circuit Board
PCM	Pulse-Code Modulation
PNG	Portable Network Graphics
PWM	Pulse Width Modulation
RAM	Random Access Memory
RTSE	Resistive Touch Screen Engine
CTSE	Capacitive Touch Screen Engine
ROM	Read Only Memory
SPI	Serial Peripheral Interface

Appendix B – List of Tables/ Figures/ Registers/ Code Snippets

List of Tables

Table 1 – API Reference Definitions	10
Table 2 – Memory Map	11
Table 3 – Read Chip Identification Code	13
Table 4 – Bitmap Rendering Performance.....	24
Table 5 – Common Registers Summary	32
Table 6 – RTSE Registers Summary.....	34
Table 7 – CTSE Registers Summary.....	37
Table 8 – Graphics Context	50
Table 9 – Graphics Primitive Definition.....	53
Table 10 – BITMAP_LAYOUT Format List	54
Table 11 – L1/L2/L4/L8 Pixel Format	56
Table 12 – ARGB2/RGB332 Pixel Format	56
Table 13 – RGB565/PALETTED565 Pixel Format	56
Table 14 – ARGB1555/ARGB4/PALETTED4444 Pixel Format.....	57
Table 15 – PALETTED8 Pixel Format	57
Table 16 – BLEND_FUNC Constant Value Definition	65
Table 17 – STENCIL_OP Constants Definition.....	83
Table 18 – VERTEX_FORMAT and Pixel Precision	87
Table 19 – Widgets Color Setup Table.....	91
Table 20 – Legacy Font Metrics Block	93
Table 21 – Coprocessor Engine Graphics State.....	95
Table 22 – Parameter OPTION Definition	96

List of Figures

Figure 1 – FT81x/BT88x data flow	12
Figure 2 – Getting Started Example.....	18
Figure 3 – Widget List.....	90
Figure 4 – ROM Font List.....	93

List of Registers

Register Definition 1 – REG_TAG Definition	25
Register Definition 2 – REG_TAG_Y Definition	25
Register Definition 3 – REG_TAG_X Definition.....	25
Register Definition 4 – REG_PCLK Definition	26
Register Definition 5 – REG_PCLK_POL Definition	26

Register Definition 6 – REG_CSPREAD Definition	26
Register Definition 7 – REG_SWIZZLE Definition.....	26
Register Definition 8 – REG_DITHER Definition	26
Register Definition 9 – REG_OUTBITS Definition.....	27
Register Definition 10 – REG_ROTATE Definition	27
Register Definition 11 – REG_DLSWAP Definition	27
Register Definition 12 – REG_VSYNC1 Definition	28
Register Definition 13 – REG_VSYNC0 Definition	28
Register Definition 14 – REG_VSIZE Definition.....	28
Register Definition 15 – REG_VOFFSET Definition.....	28
Register Definition 16 – REG_VCYCLE Definition.....	28
Register Definition 17 – REG_HSYNC1 Definition	28
Register Definition 18 – REG_HSYNC0 Definition	28
Register Definition 19 – REG_HSIZE Definition.....	29
Register Definition 20 – REG_HOFFSET Definition.....	29
Register Definition 21 – REG_HCYCLE Definition	29
Register Definition 22 – REG_PLAY Definition.....	29
Register Definition 23 – REG_SOUND Definition	29
Register Definition 24 – REG_VOL_SOUND Definition.....	30
Register Definition 25 – REG_VOL_PB Definition	30
Register Definition 26 – REG_PLAYBACK_PLAY Definition	30
Register Definition 27 – REG_PLAYBACK_LOOP Definition	30
Register Definition 28 – REG_PLAYBACK_FORMAT Definition	30
Register Definition 29 – REG_PLAYBACK_FREQ Definition.....	31
Register Definition 30 – REG_PLAYBACK_READPTR Definition	31
Register Definition 31 – REG_PLAYBACK_LENGTH Definition.....	31
Register Definition 32 – REG_PLAYBACK_START Definition	31
Register Definition 33 – REG_PLAYBACK_PAUSE Definition.....	31
Register Definition 34 – REG_TOUCH_CONFIG Definition	32
Register Definition 35 – REG_TOUCH_TRANSFORM_F Definition.....	32
Register Definition 36 – REG_TOUCH_TRANSFORM_E Definition.....	32
Register Definition 37 – REG_TOUCH_TRANSFORM_D Definition	33
Register Definition 38 – REG_TOUCH_TRANSFORM_C Definition	33
Register Definition 39 – REG_TOUCH_TRANSFORM_B Definition	33
Register Definition 40 – REG_TOUCH_TRANSFORM_A Definition	33
Register Definition 41 – REG_TOUCH_TAG Definition	34
Register Definition 42 – REG_TOUCH_TAG_XY Definition	34
Register Definition 43 – REG_TOUCH_SCREEN_XY Definition	34

Register Definition 44 – REG_TOUCH_DIRECT_Z1Z2 Definition	35
Register Definition 45 – REG_TOUCH_DIRECT_XY	35
Register Definition 46 – REG_TOUCH_RZ Definition	35
Register Definition 47 – REG_TOUCH_RAW_XY Definition	35
Register Definition 48 – REG_TOUCH_RZTHRESH Definition	36
Register Definition 49 – REG_TOUCH_OVERSAMPLE Definition	36
Register Definition 50 – REG_TOUCH_SETTLE Definition	36
Register Definition 51 – REG_TOUCH_CHARGE Definition	36
Register Definition 52 – REG_TOUCH_ADC_MODE Definition	36
Register Definition 53 – REG_TOUCH_MODE Definition	37
Register Definition 54 – REG_CTOUCH_MODE Definition	38
Register Definition 55 – REG_CTOUCH_EXTENDED Definition	38
Register Definition 56 – REG_CTOUCH_TOUCH_XY Definition	38
Register Definition 57 – REG_CTOUCH_TOUCH1_XY Definition	38
Register Definition 58 – REG_CTOUCH_TOUCH2_XY Definition	38
Register Definition 59 – REG_CTOUCH_TOUCH3_XY Definition	39
Register Definition 60 – REG_CTOUCH_TOUCH4_X Definition	39
Register Definition 61 – REG_CTOUCH_TOUCH4_Y Definition	39
Register Definition 62 – REG_CTOUCH_RAW_XY Definition	39
Register Definition 63 – REG_CTOUCH_TAG Definition	39
Register Definition 64 – REG_CTOUCH_TAG1 Definition	40
Register Definition 65 – REG_CTOUCH_TAG2 Definition	40
Register Definition 66 – REG_CTOUCH_TAG3 Definition	40
Register Definition 67 – REG_CTOUCH_TAG4 Definition	40
Register Definition 68 – REG_CTOUCH_TAG_XY Definition	41
Register Definition 69 – REG_CTOUCH_TAG1_XY Definition	41
Register Definition 70 – REG_CTOUCH_TAG2_XY Definition	41
Register Definition 71 – REG_CTOUCH_TAG3_XY Definition	41
Register Definition 72 – REG_CTOUCH_TAG4_XY Definition	41
Register Definition 73 – REG_CMD_DL Definition	42
Register Definition 74 – REG_CMD_WRITE Definition	42
Register Definition 75 – REG_CMD_READ Definition	42
Register Definition 76 – REG_CMDB_SPACE Definition	43
Register Definition 77 – REG_CMDB_WRITE Definition	43
Register Definition 78 – REG_CPURESET Definition	43
Register Definition 79 – REG_MACRO_1 Definition	43
Register Definition 80 – REG_MACRO_0 Definition	43
Register Definition 81 – REG_PWM_DUTY Definition	44

Register Definition 82 – REG_PWM_HZ Definition	44
Register Definition 83 – REG_INT_MASK Definition	44
Register Definition 84 – REG_INT_EN Definition	44
Register Definition 85 – REG_INT_FLAGS Definition	44
Register Definition 86 – REG_GPIO_DIR Definition	45
Register Definition 87 – REG_GPIO Definition	45
Register Definition 88 – REG_GPIOX_DIR Definition	45
Register Definition 89 – REG_GPIOX Definition	46
Register Definition 90 – REG_FREQUENCY Definition	46
Register Definition 91 – REG_CLOCK Definition	46
Register Definition 92 – REG_FRAMES Definition	46
Register Definition 93 – REG_ID Definition	47
Register Definition 94 – REG_SPI_WIDTH Definition	47
Register Definition 95 – REG_TRACKER Definition	47
Register Definition 96 – REG_TRACKER_1 Definition	47
Register Definition 97 – REG_TRACKER_2 Definition	48
Register Definition 98 – REG_TRACKER_3 Definition	48
Register Definition 99 – REG_TRACKER_4 Definition	48
Register Definition 100 – REG_MEDIAFIFO_READ Definition	48
Register Definition 101 – REG_MEDIAFIFO_WRITE Definition	49

List of Code Snippets

Code Snippet 1 – Initialization Sequence	14
Code Snippet 2 – Play C8 on the Xylophone	16
Code snippet 3 – Stop Playing Sound	17
Code snippet 4 – Avoid Pop Sound	17
Code Snippet 5 – Audio Playback	17
Code Snippet 6 – Check the status of Audio Playback	17
Code Snippet 7 – Stop the Audio Playback	17
Code Snippet 8 – Getting Started	18
Code Snippet 9 – Color and Transparency	23
Code Snippet 10 – PALETTED8 Drawing Example	57

Appendix C – Revision History

Document Title: BRT_AN_088 FT81x/BT88x Series Programming Guide
Document Reference No.: BRT_000422
Clearance No.: BRT#202
Product Page: <https://brtchip.com/product-category/products/>
Document Feedback: [Send Feedback](#)

Revision	Changes	Date (DD-MM-YYYY)
Version 1.0	Initial Release	13-09-2023