



Application Note

BRT_AN_086

BT82X Series Programming Guide

Version 1.0

Issue Date: 02-12-2024

This application note describes the process and practice required to program BT82X Series.

Table of Contents

1	Introduction.....	12
1.1	Scope.....	12
1.2	Intended Audience	12
1.3	Conventions	12
1.4	API Reference Definitions	12
1.5	What's new in BT82X Series?	13
2	Programming Model	14
2.1	Memory Maps.....	14
2.2	Data Flow Diagram	15
2.3	Host commands.....	16
2.4	Chip Identification Code (ID).....	19
2.5	Initialization Sequence during Boot Up	20
2.6	Power States.....	21
2.6.1	Entry Sequence to STANDBY state	21
2.6.2	Exit Sequence from STANDBY state.....	22
2.6.3	Entry Sequence to SLEEP state.....	22
2.6.4	Exit Sequence from SLEEP state	22
2.6.5	Entry Sequence to POWERDOWN state	22
2.6.6	Exit Sequence from POWERDOWN state	22
2.7	Backlight Control	22
2.8	Touch Screen.....	22
2.9	Flash Interface.....	23
2.10	Audio Engine	24
2.10.1	Sound Effect	24
2.10.2	Audio Playback	25
2.10.3	Audio PWM vs delta-sigma.....	25
2.11	Swap Chains	26
2.12	Relocatable asset loading	26

2.13	Graphics Routines	28
2.13.1	Getting Started	28
2.13.2	Coordinate Range and Pixel Precision	29
2.13.3	Screen Rotation	29
2.13.4	Drawing Pattern	31
2.14	Bitmap Transformation Matrix	33
2.15	Color and Transparency	33
2.16	Bitmap Formats	34
2.17	Bitmap ZOrder	35
2.18	REGION Instruction	36
2.19	System Limits	38
2.19.1	Render Engine	38
2.19.2	Video Playback	39
3	Register Description	40
3.1	Graphics Engine Registers	40
3.2	Audio Engine Registers	45
3.3	Flash Interface Registers	48
3.4	Touch Screen Engine Registers	49
3.4.1	Calibration	54
3.5	Swap Chain Registers	55
3.6	LVDSRX Registers	58
3.6.1	LVDSRX Core Registers	58
3.6.2	LVDSRX System Registers	60
3.7	LVDSTX Registers	62
3.7.1	LVDSTX System Registers	62
3.8	System Registers	65
3.9	I2S Registers	73
3.10	Coprocessor Engine Registers	75
3.11	Miscellaneous Registers	76
3.12	Auxiliary Registers	80

4	Display List Commands	83
4.1	Graphics State.....	83
4.2	Command Encoding	84
4.3	Command Groups	84
4.3.1	Setting Graphics State	84
4.3.2	Drawing Actions	85
4.3.3	Execution Control	85
4.4	ALPHA_FUNC	85
4.5	BEGIN	86
4.6	BITMAP_EXT_FORMAT	87
4.7	BITMAP_HANDLE	88
4.8	BITMAP_LAYOUT.....	89
4.9	BITMAP_LAYOUT_H.....	92
4.10	BITMAP_SIZE.....	93
4.11	BITMAP_SIZE_H	94
4.12	BITMAP_SOURCE	94
4.13	BITMAP_SOURCEH	95
4.14	BITMAP_SWIZZLE.....	96
4.15	BITMAP_TRANSFORM_A.....	98
4.16	BITMAP_TRANSFORM_B.....	99
4.17	BITMAP_TRANSFORM_C.....	99
4.18	BITMAP_TRANSFORM_D.....	100
4.19	BITMAP_TRANSFORM_E.....	100
4.20	BITMAP_TRANSFORM_F	101
4.21	BITMAP_ZORDER	102
4.22	BLEND_FUNC	102
4.23	CALL.....	104
4.24	CELL.....	104
4.25	CLEAR.....	105
4.26	CLEAR_COLOR A	106

4.27	CLEAR_COLOR_RGB	106
4.28	CLEAR_STENCIL	107
4.29	CLEAR_TAG	108
4.30	COLOR_A.....	108
4.31	COLOR_MASK	109
4.32	COLOR_RGB	110
4.33	DISPLAY	110
4.34	END	111
4.35	JUMP	111
4.36	LINE_WIDTH	112
4.37	MACRO.....	112
4.38	NOP	113
4.39	PALETTE_SOURCE	113
4.40	PALETTE_SOURCEH	114
4.41	POINT_SIZE	114
4.42	REGION	115
4.43	RESTORE_CONTEXT.....	116
4.44	RETURN.....	116
4.45	SAVE_CONTEXT	117
4.46	SCISSOR_SIZE	117
4.47	SCISSOR_XY	118
4.48	STENCIL_FUNC	119
4.49	STENCIL_MASK.....	120
4.50	STENCIL_OP	120
4.51	TAG	121
4.52	TAG_MASK.....	122
4.53	VERTEX2F.....	122
4.54	VERTEX2II	123
4.55	VERTEX_FORMAT.....	124

4.56	VERTEX_TRANSLATE_X	124
4.57	VERTEX_TRANSLATE_Y	125
5	Coprocessor Engine.....	126
5.1	Command FIFO	126
5.2	Widgets	127
5.2.1	Common Physical Dimensions.....	127
5.2.2	Color Settings.....	127
5.2.3	Caveat	128
5.3	Interaction with RAM_DL	128
5.4	Synchronization between MCU & Coprocessor Engine	129
5.5	ROM and RAM Fonts.....	129
5.5.1	Legacy Font Metrics Block.....	129
5.5.2	Extended Format 1 Font Metrics Block.....	130
5.5.3	Extended Format 2 Font Metrics Block.....	132
5.5.4	Fonts Flags.....	135
5.5.5	ROM Fonts (Built-in Fonts)	135
5.5.6	Using Custom Font	136
5.6	Animation support	136
5.7	String Formatting	139
5.7.1	The Flag Characters	139
5.7.2	The Field Width.....	140
5.7.3	The Precision	140
5.7.4	The Conversion Specifier	140
5.8	Coprocessor Faults	141
5.9	Coprocessor State	142
5.10	Parameter OPTION	143
5.11	Resources Utilization	145
5.12	Widget extents	145
5.13	Command list.....	145
5.14	Command Groups	145
5.15	CMD_DLSTART	148

5.16	CMD_INTERRUPT	148
5.17	CMD_COLDSTART	149
5.18	CMD_SWAP	149
5.19	CMD_APPEND	150
5.20	CMD_REGREAD	150
5.21	CMD_REGWRITE	151
5.22	CMD_MEMWRITE	152
5.23	CMD_INFLATE.....	152
5.24	CMD_LOADIMAGE	153
5.25	CMD_MEDIAFIFO	155
5.26	CMD_PLAYVIDEO	155
5.27	CMD_VIDEOSTART	157
5.28	CMD_VIDEOFRAME	157
5.29	CMD_PLAYWAV	158
5.30	CMD_LOADWAV	159
5.31	CMD_MEMCRC	160
5.32	CMD_MEMZERO	160
5.33	CMD_MEMSET	161
5.34	CMD_MEMCPY	161
5.35	CMD_BUTTON.....	162
5.36	CMD_CLOCK	164
5.37	CMD_FGCOLOR	166
5.38	CMD_BGCOLOR	167
5.39	CMD_GRADCOLOR	167
5.40	CMD_GAUGE	168
5.41	CMD_GRADIENT	172
5.42	CMD_GRADIENTA.....	173
5.43	CMD_CGRADIENT	174
5.44	CMD_GLOW.....	176

5.45	CMD_KEYS.....	176
5.46	CMD_PROGRESS.....	179
5.47	CMD_SCROLLBAR.....	180
5.48	CMD_SLIDER	182
5.49	CMD_DIAL.....	183
5.50	CMD_TOGGLE.....	185
5.51	CMD_FILLWIDTH	186
5.52	CMD_TEXT	187
5.53	CMD_TEXTDIM	189
5.54	CMD_ARC	190
5.55	CMD_SETBASE	191
5.56	CMD_NUMBER	192
5.57	CMD_LOADIDENTITY.....	193
5.58	CMD_SETMATRIX	194
5.59	CMD_GETMATRIX.....	194
5.60	CMD_GETPTR	195
5.61	CMD_GETPROPS.....	196
5.62	CMD_SCALE.....	196
5.63	CMD_ROTATE.....	197
5.64	CMD_ROTATEAROUND.....	198
5.65	CMD_TRANSLATE	199
5.66	CMD_CALIBRATE	200
5.67	CMD_CALIBRATESUB	201
5.68	CMD_SETROTATE	201
5.69	CMD_SPINNER.....	202
5.70	CMD_SCREENSAVER	204
5.71	CMD_SKETCH.....	204
5.72	CMD_STOP.....	206
5.73	CMD_SETFONT	206

5.74	CMD_SETSCRATCH	207
5.75	CMD_ROMFONT	207
5.76	CMD_RESETFONTS.....	208
5.77	CMD_TRACK.....	208
5.78	CMD_SETBITMAP	211
5.79	CMD_RENDERTARGET	212
5.80	CMD_LOGO	213
5.81	CMD_ENABLEREGION	214
5.82	CMD_FENCE.....	214
5.83	CMD_GRAPHICSFINISH	215
5.84	CMD_FLASHERASE	215
5.85	CMD_FLASHWRITE	215
5.86	CMD_FLASHPROGRAM	216
5.87	CMD_FLASHREAD	216
5.88	CMD_APPENDF	217
5.89	CMD_FLASHUPDATE.....	217
5.90	CMD_FLASHDETACH	218
5.91	CMD_FLASHATTACH.....	218
5.92	CMD_FLASHFAST.....	219
5.93	CMD_FLASHSPIDESEL	220
5.94	CMD_FLASHSPITX.....	220
5.95	CMD_FLASHSPIRX.....	221
5.96	CMD_FLASHSOURCE	221
5.97	CMD_SDATTACH	222
5.98	CMD_FSOPTIONS	222
5.99	CMD_FSREAD	223
5.100	CMD_FSSIZE.....	224
5.101	CMD_FSSOURCE	224
5.102	CMD_FSDIR	225

5.103	CMD_SDBLOCKREAD	226
5.104	CMD_DDRSHUTDOWN	227
5.105	CMD_DDRSTARTUP	227
5.106	CMD_ANIMSTART	227
5.107	CMD_RUNANIM	228
5.108	CMD_ANIMSTOP	230
5.109	CMD_ANIMXY	230
5.110	CMD_ANIMDRAW	230
5.111	CMD_ANIMFRAME	231
5.112	CMD_SYNC	232
5.113	CMD_BITMAP_TRANSFORM	232
5.114	CMD_TESTCARD	234
5.115	CMD_WAIT	234
5.116	CMD_WAITCHANGE	235
5.117	CMD_NEWLIST	235
5.118	CMD_ENDLIST	236
5.119	CMD_CALLLIST	237
5.120	CMD_COPYLIST	237
5.121	CMD_SAVECONTEXT	238
5.122	CMD_RESTORECONTEXT	238
5.123	CMD_SKIPCOND	239
5.124	CMD_WAITCOND	240
5.125	CMD_WATCHDOG	241
5.126	CMD_RESULT	241
5.127	CMD_RETURN	241
5.128	CMD_NOP	242
5.129	CMD_GETIMAGE	243
5.130	CMD_I2SSTARTUP	244
5.131	CMD_LOADASSET	244

5.132	CMD_SNAPSHOT	245
6	ASTC	247
6.1	ASTC Layout.....	247
7	YCbCr format.....	249
7.1	YcbCr Encoding.....	251
7.2	YcbCr Decoding.....	251
8	Contact Information	252
Appendix A – References		253
Document References		253
Acronyms and Abbreviations.....		253
Appendix B – List of Tables/ Figures/ Registers/ Code Snippets		255
List of Tables		255
List of Figures		256
List of Registers.....		256
List of Code Snippets.....		260
Appendix C – Revision History		261

1 Introduction

The **BT82X** Series chips are 5th generation **EVE** graphics controllers with add-on features such as audio playback, touch capabilities and video input. They consist of a rich set of graphics objects that can be used for displaying various menus and screenshots for a range of products including home appliances, toys, industrial machinery, home automation, elevators, and many more.

This document captures the programming details for the **BT82X** Series chips including graphics commands, widget commands and configurations to control **BT82X** Series chips for smooth and vibrant screen effects.

1.1 Scope

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

The descriptions in this document are applicable to **BT82X**, unless specified otherwise.

Within this document, the endianness of commands, register values, and data in **RAM_G** are in *little-endian* format.

Information on pin settings, hardware characteristics and hardware configurations can be found in the [BT82X datasheet](#).

1.2 Intended Audience

This document is intended for software programmers and system designers engaged in the development of graphical user interface (**GUI**) applications using the **BT82X** on any processor equipped with an SPI master interface that supports Single, Dual or Quad channel modes.

1.3 Conventions

All values are decimal by default.

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

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

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

Host commands refer to the **EVE** specific commands defined in [Table 3 – Host commands](#).

1.4 API Reference Definitions

The following table provides the functionality and nomenclature of the APIs used in this document.

wr32()	write 32 bits to intended address location
rd32()	read 32 bits from intended address location
cmd()	write 32 bits to command FIFO i.e. RAM_CMD
cmd_* ()	write 32 bits commands 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 in host command protocol

Table 1 – API Reference Definitions

1.5 What's new in BT82X Series?

In contrast to the preceding BT81X series, the **BT82X** Series boasts a completely revamped architecture, delivering superior performance and a range of interfaces. Significant improvements have been made to both hardware and firmware. It redefines the utilization of the general-purpose RAM (**RAM_G**) by moving it from internal **SRAM** to external **DDR DRAM** memory, thereby enhancing flexibility across various applications. In addition, **BT82X** includes the following major changes:

- Replaces RGB interface with LVDS interface to cater for the higher resolution LCD.
- Adds a DDR controller to support variable size of **RAM_G**.
- Supports new **YCbCr** format in the bitmap rendering and video decoding.
- Improves the audio output quality by supporting **stereo** audio on I2S protocol.
- Adds SD card and **NAND** flash controller to facilitate the access of the assets.
- Adds watchdog timer to facilitate automatic correction.
- Adds LVDS RX interface to source the video input.

Due to this entirely redesigned architecture, the **BT82X** is not backward compatible with the **BT81X** series. Additionally, ASTC-based bitmap rendering directly from flash, a feature present in the **BT81X**, is no longer available as the **BT82X** uses an external, larger **DDR DRAM**. This larger DRAM can accommodate a much greater number of assets simultaneously, allowing them to be displayed directly from DRAM for better performance. Assets can be easily copied into DRAM from the attached NOR/NAND Flash or SD card.

2 Programming Model

BT82X appears to the host as a memory-mapped **SPI** slave device. The host communicates with **BT82X** across the **SPI** bus, adhering to the serial data protocol detailed in the [BT82X datasheet](#).

2.1 Memory Maps

The entire memory and registers are mapped within a 2 gigabyte address space.

The address space definitions are provided in the following table:

Name	Start Address	End Address	Size (bytes)	Description
RAM_G	0x00000000	0x7FFFFFFF	2032 M	General purpose RAM . The actual size depends on the attached DDR RAM
RAM_CMD	0x7F000000	0x7F003FFF	16 K	Command FIFO
REG_CORE (Region 2)	0x7F004000	0x7F0047FF	2 K	Registers in region 2
RAM_ERR_REPORT	0x7F004800	0x7F00487F	128	Coprocessor fault report area
REG_CORE (Region 1)	0x7F006000	0x7F006FFF	4 K	Registers in region 1
RAM_DL	0x7F008000	0x7F00BFFF	16 K	Display list RAM
REG_LVDSTX	0x7F800300	0x7F8003FF	256	LVDS TX registers
REG_SYS	0x7F800400	0x7F8004FF	256	System registers
REG_LVDSRX	0x7F800500	0x7F8005FF	256	LVDS RX registers
REG_I2S	0x7F800800	0x7F8008FF	256	I2S registers

Table 2 – Memory Map

Note:

1. The addresses beyond this table are reserved and shall not be read or written unless otherwise specified.
2. Besides **RAM_G**, which is in the external **DDR RAM**, all the others are in the internal **RAM**.

The top **2.5** megabytes (2621440 bytes) of **DDR RAM** in the **RAM_G** address space is reserved. Avoid accessing this area; the rest of the memory is available for unrestricted use.

The diagram below shows the allocations for an application with attached 128 Mbyte **DDR RAM**:



2.2 Data Flow Diagram

Figure 1 illustrates the data flow between external components (MCU, DDR, Audio codec, SD card, LVDS, and Flash etc.) and the internal components of **BT82x**.

It's important to note that direct writing from the **MCU** to **RAM_DL** requires careful synchronization of the read/write pointers in the respective registers, as the coprocessor engine may also write generated display list commands into **RAM_DL**.

To avoid this complexity, a better approach is to write display list commands to **RAM_CMD** and let the coprocessor update **RAM_DL** accordingly. Direct writing into **RAM_DL** is not recommended.

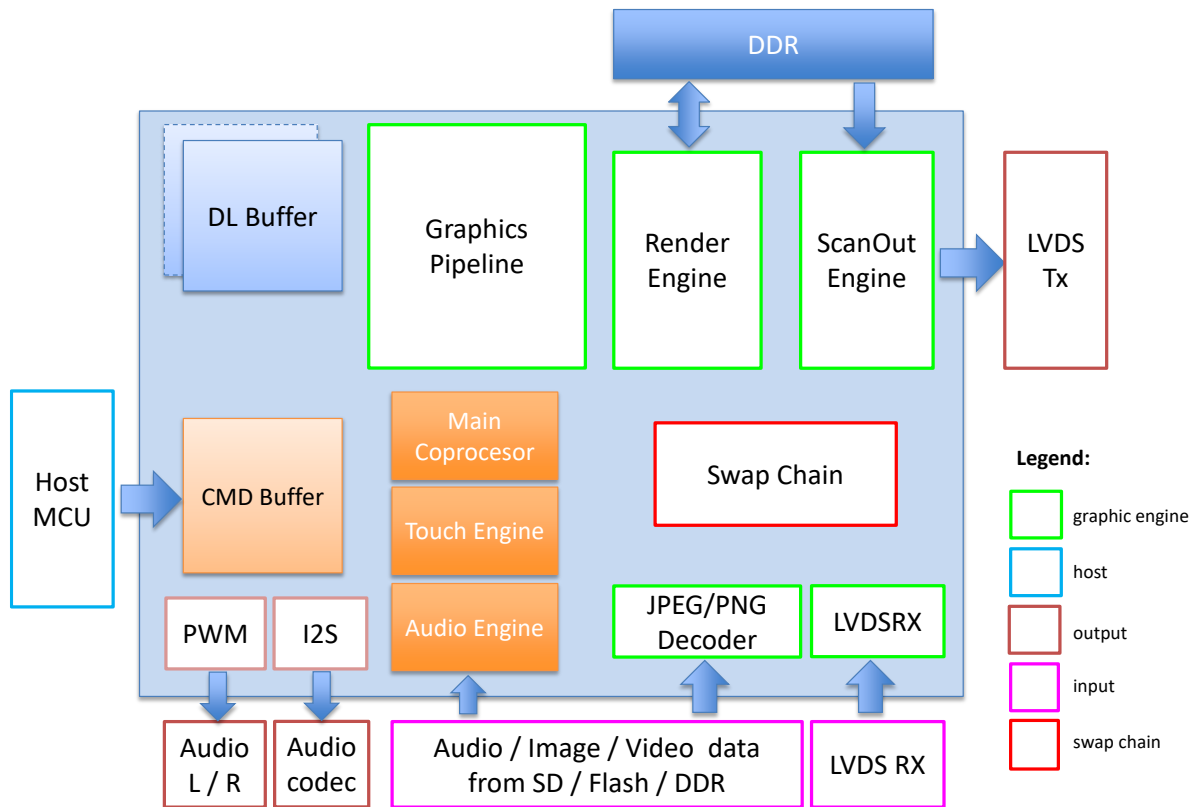


Figure 1 – BT82X data flow

The data here refers to the following items:

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

2.3 Host commands

Host commands are only operational in SPI Single channel mode and will be ignored in SPI Dual or Quad mode.

The configurations set by host commands remain effective across all power states and are only reset when a power-on reset event occurs or the **RST_N** pin is activated.

After the **RST_N** pin is released or power-up initialization completes, the BT82X will stay in the lowpower "SLEEP" state to conserve power.

About the power states, please refer to [BT82X datasheet](#) for more details.

Command	SPI Sequence	Description
ACTIVE	0x00, 0x00, 0x00, 0x00, 0x00	Wake up the system from low power state.
STANDBY	0xFF, 0xE0, 0x00, 0x00, 0x00	Place the BT82X into the low power "STANDBY" state: <ul style="list-style-type: none"> • Clocks will be halted
SLEEP	0xFF, 0xE1, 0x00, 0x00, 0x00	Place the BT82X into the low power "SLEEP" state: <ul style="list-style-type: none"> • Clocks will be halted

POWERDOWN	0xFF, 0xE2, 0x00, 0x00, 0x00	<ul style="list-style-type: none"> • PLLs and DDR will be disabled Place the BT82X into the low power "POWERDOWN" state: <ul style="list-style-type: none"> • Gated-power domain will be switched off except QSPI host interface • Clocks will be halted • PLLs and DDR will be disabled 										
SETPLLSP1	0xFF, 0xE4, 0x**, 0x00, 0x00	Byte3 configures the SYSPLL_NS bits in the REG_SYS_STAT register to change the system PLL frequency, according to the following formula: $\text{System PLL Frequency} = \text{oscillator} * \text{SYSPLL_NS}$ The default value of SYSPLL_NS is 15 and the system PLL frequency is 576MHz since oscillator produces 38.4 MHz frequency. This command will only be effective when the system is in a low power mode.										
SETSYSCLKDIV	0xFF, 0xE6, 0x**, 0x00, 0x00	Byte3 configures the SYSCLKDIV bits of REG_SYS_STAT register to change the system clock frequency, according to the following formula: $\text{System Clock Frequency} = \text{System PLL Frequency} / (\text{SYSCLKDIV} + 1)$ <table border="1" data-bbox="831 1099 1394 1279" style="margin-left: 20px;"> <tr> <td>Byte3[7:4]</td> <td>0b0001</td> </tr> <tr> <td>Byte3[3:0]</td> <td>SYSCLKDIV : System clock divider setting See Table 4 – System Clock Configuration</td> </tr> </table> The default value of SYSCLKDIV is 7 and the system clock frequency is 72 MHz if SYSPLL_NS in REG_SYS_STAT is in default value. This command will only be effective when BT82X is in a low power mode.	Byte3[7:4]	0b0001	Byte3[3:0]	SYSCLKDIV : System clock divider setting See Table 4 – System Clock Configuration						
Byte3[7:4]	0b0001											
Byte3[3:0]	SYSCLKDIV : System clock divider setting See Table 4 – System Clock Configuration											
SETBOOTCFG	0xFF, 0xE8, 0x**, 0x00, 0x00	Byte3 configures the REG_BOOT_CFG register to control the boot sequence. This command is effective only when the BT82X is in low power mode. <table border="1" data-bbox="831 1686 1394 1872" style="margin-left: 20px;"> <tr> <td>Byte3[7]</td> <td>Set 1 to enable DDR system</td> </tr> <tr> <td>Byte3[6]</td> <td>Set 1 to enable touch system</td> </tr> <tr> <td>Byte3[5]</td> <td>Set 1 to enable audio system</td> </tr> <tr> <td>Byte3[4]</td> <td>Set 1 to enable watchdog system</td> </tr> <tr> <td>Byte3[3:0]</td> <td>0b0000</td> </tr> </table> To use this command, the BOOTCFGEN command must first be sent with the BOOT_USER_SETTING and BOOTCFGEN_ALLOW bits set to 1. Otherwise, the default boot configuration will be applied	Byte3[7]	Set 1 to enable DDR system	Byte3[6]	Set 1 to enable touch system	Byte3[5]	Set 1 to enable audio system	Byte3[4]	Set 1 to enable watchdog system	Byte3[3:0]	0b0000
Byte3[7]	Set 1 to enable DDR system											
Byte3[6]	Set 1 to enable touch system											
Byte3[5]	Set 1 to enable audio system											
Byte3[4]	Set 1 to enable watchdog system											
Byte3[3:0]	0b0000											

		during boot, which only enables the DDR system.								
BOOTCFGEN	0xFF, 0xE9, 0x**, 0x00, 0x00	<p>Byte3 determines whether the settings specified in the SETBOOTCFG and SETDDRTYPE commands are used to update the respective registers and applied to the BT82X system configuration.</p> <p>This command is effective only when the BT82X is in low power mode.</p> <table border="1" style="width: 100%;"> <tr> <td>Byte3[7]</td> <td> BOOT_USER_SETTING: 1 – Allow the update of REG_BOOT_CFG register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1. </td> </tr> <tr> <td>Byte3[6]</td> <td> DDRTYPE_USER_SETTING 1 – Allow the update of REG_DDR_TYPE register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1. </td> </tr> <tr> <td>Byte3[5:1]</td> <td>Reserved</td> </tr> <tr> <td>Byte3[0]</td> <td> BOOTCFGEN_ALLOW This serves as a safety mechanism to prevent accidental overwriting of REG_BOOT_CFG and REG_DDR_TYPE. 1 – allow update 0 – ignore update </td> </tr> </table>	Byte3[7]	BOOT_USER_SETTING: 1 – Allow the update of REG_BOOT_CFG register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1.	Byte3[6]	DDRTYPE_USER_SETTING 1 – Allow the update of REG_DDR_TYPE register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1.	Byte3[5:1]	Reserved	Byte3[0]	BOOTCFGEN_ALLOW This serves as a safety mechanism to prevent accidental overwriting of REG_BOOT_CFG and REG_DDR_TYPE. 1 – allow update 0 – ignore update
Byte3[7]	BOOT_USER_SETTING: 1 – Allow the update of REG_BOOT_CFG register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1.									
Byte3[6]	DDRTYPE_USER_SETTING 1 – Allow the update of REG_DDR_TYPE register 0 – ignore update Update of this bit is only allowed when Byte3[0]=1.									
Byte3[5:1]	Reserved									
Byte3[0]	BOOTCFGEN_ALLOW This serves as a safety mechanism to prevent accidental overwriting of REG_BOOT_CFG and REG_DDR_TYPE. 1 – allow update 0 – ignore update									
SETDDRTYPE	0xFF, 0xEB, 0x**, 0x00, 0x00	<p>Byte3 programs the DDR DRAM type definition register, REG_DDR_TYPE. This command is effective only when the BT82X is in low power mode.</p> <p>See Table 5 – DDR DRAM Types Setting</p> <p>Default setting is 0x08 (DDR3L, 1333MT/s, 1G bits)</p> <p>To use this command, the BOOTCFGEN command must first be sent with the DDRTYPE_USER_SETTING and BOOTCFGEN_ALLOW bits set to 1.</p>								

Table 3 – Host commands

The table below describes the system clock related register settings. The default setting is highlighted in **BOLD**.

OSC (MHz)	SYSPLL_NS	Division Factor (SYSCLKDIV+1)	System Clock (MHz)
------------------	------------------	--------------------------------------	---------------------------

38.4	15 (default)	8 (default)	72.00
		9	64.00
		12	48.00
	13	16	36.00
		7	71.31
		16	31.20

Table 4 – System Clock Configuration

Below table shows the different DRAM types and their settings. The default setting is highlighted in **BOLD**. Any changes in the configuration will require the DDR to be initialized through host command.

DRAM Attribution			Byte3 of SETDDRTYPE		
TYPE	Size (Mbits)	Speed Grade (MT/s)	Bit [7:5] (SPEED GRADE)	Bit [4:3] (Type)	Bit[2:0] (SIZE)
DDR3L	8192	1333	0	1	3
DDR3L	4096	1333	0	1	2
DDR3L	2048	1333	0	1	1
DDR3L	1024	1333	0	1	0
DDR3	8192	1333	0	0	3
DDR3	4096	1333	0	0	2
DDR3	2048	1333	0	0	1
DDR3	1024	1333	0	0	0
DDR3L	8192	1066	1	1	3
DDR3L	4096	1066	1	1	2
DDR3L	2048	1066	1	1	1
DDR3L	1024	1066	1	1	0
DDR3	8192	1066	1	0	3
DDR3	4096	1066	1	0	2
DDR3	2048	1066	1	0	1
DDR3	1024	1066	1	0	0
LPDDR2	2048	1066	1	2	1
LPDDR2	1024	1066	1	2	0
LPDDR2	512	1066	1	2	4
DDR3L	8192	933	2	1	3
DDR3L	4096	933	2	1	2
DDR3L	2048	933	2	1	1
DDR3L	1024	933	2	1	0
DDR3	8192	933	2	0	3
DDR3	4096	933	2	0	2
DDR3	2048	933	2	0	1
DDR3	1024	933	2	0	0
LPDDR2	2048	933	2	2	1
LPDDR2	1024	933	2	2	0
LPDDR2	512	933	2	2	4
LPDDR2	2048	800	3	2	1
LPDDR2	1024	800	3	2	0
LPDDR2	512	800	3	2	4

Table 5 – DDR DRAM Types Setting

2.4 Chip Identification Code (ID)

After reset or reboot, the chip ID can be read from **REG_CHIP_ID**.

The following table describes Chip Identification Code:

Byte 3	Byte 2	Byte 1	Byte 0
0x08	0x20	0x01	0x00

Table 6 – Chip Identification Code

2.5 Initialization Sequence during Boot Up

The boot-up sequence is outlined as follows:

1. Host configures the system through host commands **if** the default settings are not matched with the application:
 - Send **BOOTCFGEN** to turn on the switch of user settings update.
 - Send **SETBOOTCFG** to select the systems of BT82X to be enabled.
 - Send **SETDDRTYPE** to select the DDR type to work with BT82X.
 - Send **BOOTCFGEN** to turn off the switch of user settings update.
 - Send **SETPLLSP1** and **SETSYSCLKDIV** to configure the system clock frequency
2. Host sends **ACTIVE** command.
3. Host reads **REG_BOOT_STATUS** till BT82X goes into the normal running status, i.e., **0x522E_2E2E** is returned.

Note: The default settings after reset are:

- DDR system is to be enabled, supporting DDR type of parameters “DDR3L, 1G bits, 1333MT/s”
- Audio system is to be disabled
- Touch system is to be disabled
- Watchdog system is to be disabled
- System PLL is running at 576 MHz and system clock at 72MHz

The code snippet below showcases the boot-up sequence which is different from the default settings:

```

host_command(BOOTCFGEN); // 0xFF, 0xE9, 0xC1, 0x00, 0x00 -- Turn on user setting switch
host_command(SETBOOTCFG); // 0xFF, 0xE8, 0xE0, 0x00, 0x00 -- Enable DDR, Touch and Audio
system
host_command(SETDDRTYPE); // 0xFF, 0xEB, 0x00, 0x00, 0x00 -- 1G bit DDR3 in 1333MT/s
host_command(BOOTCFGEN); // 0xFF, 0xE9, 0xC0, 0x00, 0x00 -- Turn off user setting switch
host_command(SYSCLKDIV); // 0xFF, 0xE6, 0x17, 0x00, 0x00 -- Set system clock to 72MHz
(same as default, optional)

host_command(ACTIVE); // 0x00, 0x00, 0x00, 0x00, 0x00 -- Switch power state to Active
msWait(40); // wait 40 milliseconds

while (0x522E2E2E != rd32(REG_BOOT_STATUS)); //Check if boot up is completed.
  
```

Code Snippet 1 – Initialization Sequence

Note: Byte3 in SETBOOTCFG with values 0xE0 will enable the DDR, touch and audio systems during boot-up while bytes3 in SETSYSCLKDIV with values 0x17 will configure system clock to run at 72Mhz.

The table below describes the different stages of the boot state. The status can be used for monitoring and diagnostics purpose.

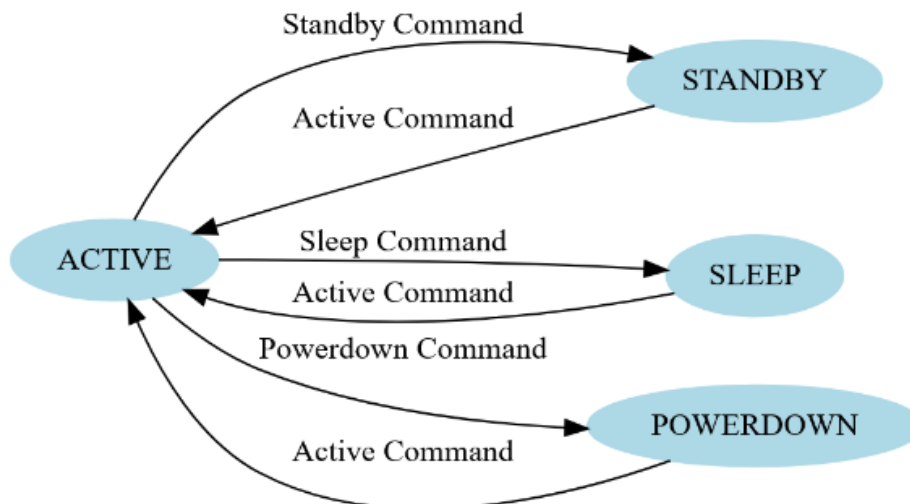
Code	ASCII	Description
492e2e2e	I...	coprocessor is running
4f2e2e2e	O...	OTP read started

442e2e2e	D...	DDR initialization started
444d3038	DM08	DDR initialization, waiting for DDR initialization done
44433035	DC05	DDR initialization, waiting for DDR out of reset
44553135	DU15	DDR initialization, 150 us delay
44553730	DU70	DDR initialization, 700 us delay
552e2e2e	U...	Decompressing rom main image to DDR
432e2e2e	C...	Copying into program memory
562e2e2e	V...	Decompressing rom asset image to DDR
4c2e2e2e	L...	Initializing local variables
542e2e2e	T...	Copying into touch program memory
462e2e2e	F...	Attempting to attach to flash
522e2e2e	R...	normal running
452e2e2e	E...	DDR shutdown started
454d3130	EM10	DDR shutdown, waiting for DDR enter self-refresh state
45433034	EC04	DDR shutdown, waiting for DDR enter reset
5a2e2e2e	Z...	DDR shutdown state
572e2e2e	W...	DDR warm start, started
57433035	WC05	DDR warm start, waiting for DDR out of reset
574d3038	WM08	DDR warm start, waiting for DDR initialization done
574d3130	WM10	DDR warm start, waiting for DDR enter self-refresh state
576d3130	Wm10	DDR warm start, waiting for DDR not in self-refresh state
57553135	WU15	DDR warm start, 150 us delay

Table 7 – EVE Boot Status

2.6 Power States

The BT82X supports various power states, including ACTIVE, STANDBY, SLEEP and POWERDOWN. The figure below illustrates the transitions between these power states as the EVE responds to corresponding host command.



2.6.1 Entry Sequence to STANDBY state

If the sequence below is followed, the BT82X will transition from **ACTIVE** state into **STANDBY** state:

1. Send host command **STANDBY**

2.6.2 Exit Sequence from STANDBY state

If the sequence below is followed, the BT82X will transition from **STANDBY** state into **ACTIVE** state:

1. Send host command **ACTIVE**

2.6.3 Entry Sequence to SLEEP state

If the sequence below is followed, the BT82X will transition from **ACTIVE** state into **SLEEP** state:

1. Send host command **CMD_DDRSHUTDOWN** to put **DDR RAM** into self-refresh mode
2. Wait till **CMD_DDRSHUTDOWN** is executed, i.e., **REG_CMD_READ** is equal to **REG_CMD_WRITE**
3. Send host command **SLEEP**

Clocks of BT82X will be disabled in the **SLEEP** state.

2.6.4 Exit Sequence from SLEEP state

If the sequence below is followed, the BT82X will transition from **SLEEP** state into **ACTIVE** state:

2. Send host command **ACTIVE**
3. Wait at least 10us
4. Send host command **CMD_DDRSTARTUP** to bring **DDR RAM** out of self-refresh mode.
5. Wait till **CMD_DDRSTARTUP** is executed, i.e., **REG_CMD_READ** is equal to **REG_CMD_WRITE**

Now BT82X is resumed to **ACTIVE** state.

2.6.5 Entry Sequence to POWERDOWN state

If the sequence below is followed, the BT82X will transition from **ACTIVE** state into **POWERDOWN** state:

1. Send host command **POWERDOWN**

Clocks of BT82X will be disabled in the **POWERDOWN** state.

2.6.6 Exit Sequence from POWERDOWN state

Follow section [2.5 Initialization Sequence during Boot Up](#).

2.7 Backlight Control

When **REG_DISP** is set to 1, the PWM signal can be used for backlight control.

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

REG_PWM_HZ specifies the PWM output frequency. Range is 250-10000 Hz.

REG_PWM_DUTY specifies the PWM output duty cycle. Range is 0-128. A value of 0 means that the PWM is completely OFF. A value of 128 means that the PWM signal is completely ON.

2.8 Touch Screen

The touch engine drives the touch system. Multiple touch devices are supported using multiple drivers. The raw touch screen (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 (65535).

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 24-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**.

Several registers control aspects of the touchscreen conversion process.

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.

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.

2.9 Flash Interface

To access an attached flash chip, **BT82x** provides the necessary registers to read/write flash with very high throughput. The graphics engine can fetch these assets directly, without going through the external host **MCU**, thus significantly off-loading the host **MCU** from feeding display contents.

The register **REG_FLASH_STATUS** indicates the state of the flash subsystem. During boot up, the flash state is **FLASH_STATE_INIT**. After detection has completed, flash is in the state **FLASH_STATE_DETACHED** or **FLASH_STATE_BASIC**, depending on whether an attached flash device was detected.

If no device is detected, then all the SPI output signals are driven low.

When the host MCU calls **CMD_FLASHFAST**, the flash system attempts to go to full-speed mode, setting the state to **FLASH_STATE_FULL**.

At any time, users can call **CMD_FLASHDETACH** to disable the flash communications. This tristates all flash signals, allowing a suitably connected MCU to drive the flash directly. Alternatively, in the detached state, commands such as **CMD_FLASHSPIDESEL**, **CMD_FLASHSPITX** and **CMD_FLASHSPIRX** can be used to control the SPI bus.

If detached, the host MCU can call **CMD_FLASHATTACH** to re-establish communication with the flash device.

Command	DETACHED	BASIC	FULL	Operation
CMD_FLASHERASE		✓	✓	Erase all of flash
CMD_FLASHWRITE		✓	✓	Write data from RAM_CMD to blank flash
CMD_FLASHUPDATE		✓	✓	Read the flash and update to flash if different (For NOR flash only)
CMD_FLASHPROGRAM		✓	✓	Write data from RAM_G to blank flash
CMD_FLASHREAD		✓	✓	Reads data from flash to main memory
CMD_FLASHDETACH		✓	✓	Detach from flash
CMD_FLASHATTACH	✓			Attach to flash

CMD_FLASHFAST		✓		Enter full-speed(fast) mode
CMD_FLASHSPIDESEL	✓			SPI bus: deselect device
CMD_FLASHSPITX	✓			SPI bus: write bytes
CMD_FLASHSPIRX	✓			SPI bus: read bytes

Table 8 – Flash Interface states and commands

To support different vendors of SPI NOR flash chips, the first block (4096 bytes) of the flash is reserved for the flash driver called **BLOB** file which is provided by **Bridgetek**. The **BLOB** file shall be programmed first so that flash state can enter full-speed (fast) mode. Please refer to [BT82X datasheet](#) for more details.

BT82x adds the support of NAND flash chips.

2.10 Audio Engine

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

It supports 2 output methods: stereo AUDIO pins or I2S interface. Note that concurrent usage of stereo AUDIO pins and the I2S interface is not supported. The **AUDIO_L** and **AUDIO_R** pins shall be tied low when I2S is enabled. This can be done by setting register bit **AUDIO_DISABLE** of **REG_I2S_CTL**.

2.10.1 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:

```
wr32(REG_VOL_SOUND,0xFF); //set the volume to maximum
wr32(REG_SOUND, (0x6C<< 8) | 0x41); // C8 MIDI note on xylophone
wr32(REG_PLAY, 1); // play the sound
```

Code Snippet 2 – Play C8 on the Xylophone

Sample code to stop sound play:

```
wr32(REG_SOUND,0x0); //configure "silence" sound to be played
wr32(REG_PLAY,1); //play sound
sound_status = rd32(REG_PLAY); //1-play is going on, 0-play has finished
```

Code snippet 3 – Stop Playing Sound

To avoid a pop sound on reset or power state change, trigger a "mute" sound, and wait for it to complete (i.e. **REG_PLAY** contains the value of 0). This sets the audio output pin to 0 levels. On reboot, the audio engine plays back the "unmute" sound.

```
wr32(REG_SOUND,0x60); //configure "mute" sound to be played
wr32(REG_PLAY,1); //play sound
sound_status = rd32(REG_PLAY); //1-play is going on, 0-play has finished
```

Code snippet 4 – Avoid Pop Sound

Note: Refer to [BT82X datasheet](#) for more information on the sound synthesizer and audio playback.

2.10.2 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 also required to be 8 bytes aligned.

The registers controlling audio playback are:

- REG_PLAYBACK_START**: the start address of the audio data
- REG_PLAYBACK_LENGTH**: the length of the audio data, in bytes
- REG_PLAYBACK_FREQ**: the playback frequency, in Hz. Range is 1-65535.
- REG_PLAYBACK_FORMAT**: the playback format, one of **LINEAR_SAMPLES**, **ULAW_SAMPLES**, **ADPCM_SAMPLES**, **S16_SAMPLES**, or **S16S_SAMPLES**.
- REG_PLAYBACK_LOOP**: if zero, sample is played once. If one, sample is repeated indefinitely
- REG_PLAYBACK_PLAY**: a write to this location triggers the start of audio playback
- REG_PLAYBACK_PAUSE**: pause/resume the audio playback
- REG_VOL_L_PB**: playback volume L, 0-255
- REG_VOL_R_PB**: playback volume R, 0-255

For **LINEAR_SAMPLES** and **ULAW_SAMPLES**, each sample is a signed byte. For **ADPCM_SAMPLES**, each sample is 4 bits, so two samples per byte. The first sample is in bits 0-3, the second in bits 4-7. For **S16_SAMPLES**, each sample is a signed little-endian 16-bit value. For **S16S_SAMPLES**, each sample is a pair of signed little-endian 16-bit values. The left channel sample is first, followed by the right channel sample.

The current audio playback read pointer can be queried by reading **REG_PLAYBACK_READPTR**. Using a large sample buffer, looping, and this read pointer, the host MCU can supply a continuous stream of audio.

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

```

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

Code Snippet 5 – Audio Playback

```

audioPlay_Status = rd32(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
wr32(REG_PLAYBACK_PLAY,1); //start audio playback
  
```

Code Snippet 7 – Stop the Audio Playback

2.10.3 Audio PWM vs delta-sigma

The mixed audio output appears on pins **AUDIO_L** and **AUDIO_R**. If **REG_AUD_PWM bit 2** is 0, then the output is in delta-sigma format. If **REG_AUD_PWM bit 2** is 1, then the output is in PWM format.

2.11 Swap Chains

BT82X uses a hardware object called a swap chain (**SC**) to coordinate buffer writers and buffer readers. The **SC** holds multiple pointers to buffers, plus the state about which buffers are full and which are empty. As the producer finishes writing to each buffer, it is added to the full chain. Similarly, when the consumer finishes using a buffer, it is added to the empty chain. Each **SC** may be configured with 2, 3 or 4 buffers. Double-buffering is a popular configuration, but triple-buffering and quad-buffering may be useful for video playback.

SWAPCHAIN_0 (0xFFFF00FF) is written by the render engine if **REG_RE_DEST** is set to **SWAPCHAIN_0**. It is read by scanout system if **REG_SO_SOURCE** is set to **SWAPCHAIN_0**.

SWAPCHAIN_1 (0xFFFF01FF) is written by the video decompressor. The consumer may be the scanout system (in the case of **CMD_PLAYVIDEO** with **OPT_DIRECT**), or frames may be consumed as bitmaps and used as input to the render engine.

SWAPCHAIN_2 (0xFFFF02FF) is written by the LVDSRX unit. The consumer may be the scanout system, or frames may be consumed as bitmaps and used as input to the render engine.

The swap chains have a common register interface for setup:

REG_SCx_RESET write to reset the swapchain state, marking all buffers free
REG_SCx_SIZE number of buffers in the chain, valid value is 2-4
REG_SCx_PTR0 buffer 0 start address
REG_SCx_PTR1 buffer 1 start address
REG_SCx_PTR2 buffer 2 start address
REG_SCx_PTR3 buffer 3 start address

(x stands for 0,1,2)

By default, each swap chain is configured with two buffers pointing to address 0, because **REG_SCx_SIZE** is set to 2 and **REG_SCx_PTR0/1** are set to 0. Users are expected to adjust these pointers -- using **REG_SCx_PTR0**, **REG_SCx_PTR1**, **REG_SCx_PTR2** and **REG_SCx_PTR3** -- to suit their own memory management schemes.

Example:

MCU does a clean start in EVE using **SWAPCHAIN_0**.

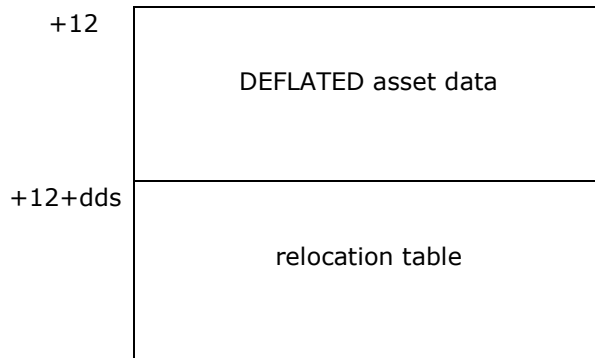
1. Configure video timing registers
2. Set the pointer registers (**REG_SC0_PTRx**) of **SWAPCHAIN_0**
3. Set the render engine destination (**REG_RE_DEST**) to **SWAPCHAIN_0**
4. Set the scanout source (**REG_SO_SOURCE**) to **SWAPCHAIN_0**
5. Write first display list and call **CMD_SWAP**
6. Write 1 to **REG_SO_EN** to enable scanout

2.12 Relocatable asset loading

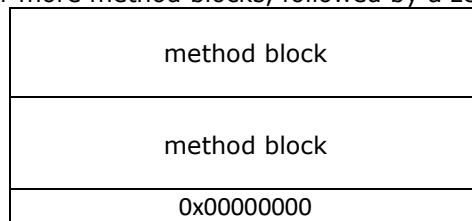
CMD_LOADASSET allows assets such as fonts and animations to be loaded at arbitrary addresses. It does this by using a special relocation table that specifies every address in the asset that must be adjusted.

The asset format is:

+0	signature, 0x0100aa44
+4	asset size
+8	deflated data size (dds)



The relocation table is zero or more method blocks, followed by a zero word:



Each method block specifies a method, a count, and one or more offsets:

+0	method	relocation method, see below
+4	count	number of offsets
+8	offset ₀	offset within asset of relocation start
+12	offset ₁	
	.	
	.	
	offset _n	

Each method specifies a relocation action, depending on the object:

P: a simple pointer offset

H: a pair of bitmap instructions, **BITMAP_SOURCEH**, **BITMAP_SOURCE** or **PALETTE_SOURCEH**, **PALETTE_SOURCE**

Example: Here is an example of a relocatable legacy font, "Roboto-Regular_36_L1.reloc". Please note that the values are shown in little-endian format.

+0	44 aa 00 01	signature
+4	fc 38 00 00	14588 = asset size
+8	d0 16 00 00	3052 = deflated data size
+12	DEFLATED legacy font	3052 deflated bytes
+12+3052	50 00 00 00	P method block
	01 00 00 00	1 relocated offset
	90 00 00 00	offset to relocate
	00 00 00 00	end of relocation table
		relocation table

Note: The asset size (14588) indicates that after the `cmd_loadasset()` completes execution, the relocatable asset will be inflated into `RAM_G` with a size exactly equal to 14588. There is only one offset that needs to be relocated at `0x90`, which is a pointer to the font graphic data. Since it is a simple pointer, we use the P method.

2.13 Graphics Routines

This section outlines graphics features and includes several examples.

2.13.1 Getting Started

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

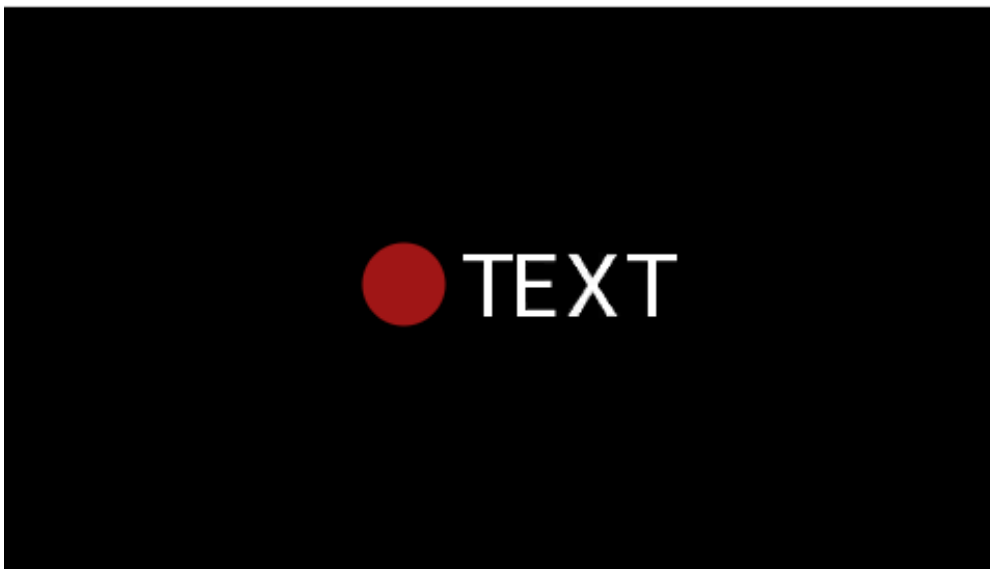


Figure 2 – Getting Started Example

The code to draw the screen is:

```
cmd(CLEAR(1, 1, 1)); // clear screen
cmd_text(220, 110, 31, 0, "TEXT"); // Draw "TEXT" at (220, 110)
cmd(COLOR_RGB(160, 22, 22)); // change color to red
cmd(POINT_SIZE(320)); // set point size
cmd(BEGIN(POINTS)); // start drawing points
cmd(VERTEX2II(192, 133, 0, 0)); // red point
cmd(DISPLAY()); // end of display list
cmd_swap(); // display the image
```

Code Snippet 8 – Getting Started

Note:

- This example writes the display list to **RAM_CMD** and lets coprocessor update the display list to **RAM_DL**
- 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**. **CMD_SWAP** will swap the display list and display the image on the screen.

2.13.2 Coordinate Range and Pixel Precision

Apart from the single pixel precision, **BT82x** supports a series of fractional pixel precision, which results 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 **BT82x** 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.

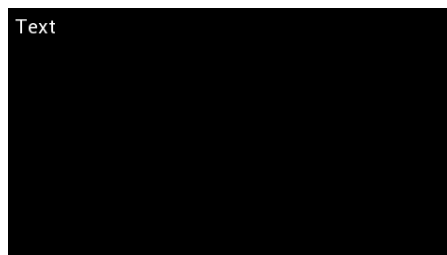
2.13.3 Screen Rotation

REG_RE_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.

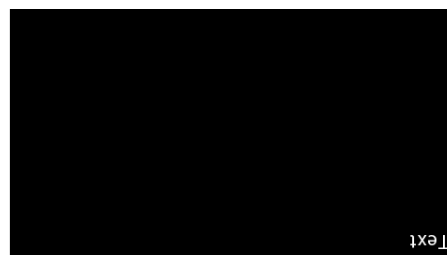
Note: The touch transformation matrix is not affected by setting **REG_RE_ROTATE**.

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

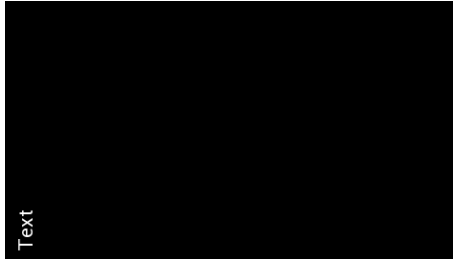
REG_RE_ROTATE = 0 is the default landscape orientation:



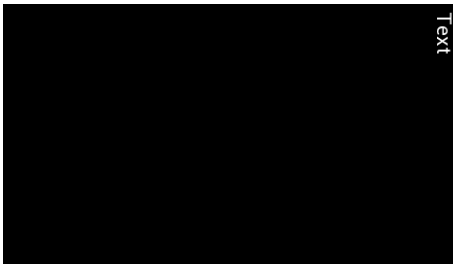
REG_RE_ROTATE = 1 is inverted landscape:



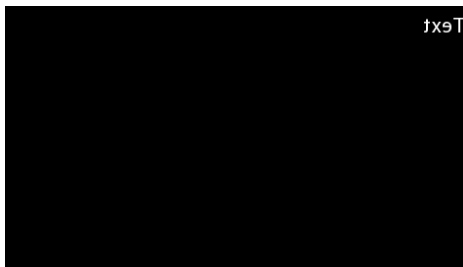
REG_RE_ROTATE = 2 is portrait:



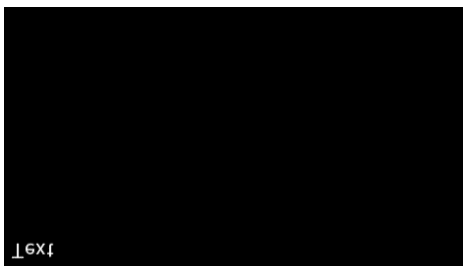
REG_RE_ROTATE = 3 is inverted portrait:



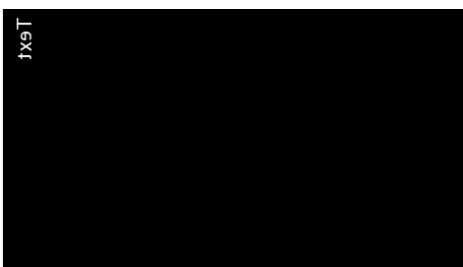
REG_RE_ROTATE = 4 is mirrored landscape:



REG_RE_ROTATE = 5 is mirrored inverted landscape:



REG_RE_ROTATE = 6 is mirrored portrait:



REG_RE_ROTATE = 7 is mirrored inverted portrait:



2.13.4 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, which is optional.

Examples

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



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



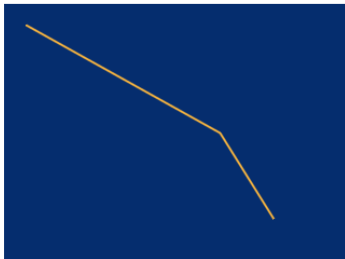
```
cmd(COLOR_RGB(128,0,0));
cmd(LINE_WIDTH(2*16));
cmd(BEGIN(LINES) );
cmd(VERTEX2F(30*16,38*16));
cmd(VERTEX2F(30*16,63*16));
cmd(COLOR_RGB(0,128,0));
cmd(LINE_WIDTH(4*16));
cmd(VERTEX2F(60*16,25*16));
cmd(VERTEX2F(60*16,63*16));
cmd(COLOR_RGB(128,128,0));
cmd(LINE_WIDTH(6*16));
cmd(VERTEX2F(90*16,13*16));
cmd(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.
cmd(COLOR_RGB(128, 0, 0));
cmd(LINE_WIDTH(1 * 16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(28 * 16, 38 * 16));
cmd(VERTEX2F(33 * 16, 63 * 16));
cmd(COLOR_RGB(0, 128, 0));
cmd(LINE_WIDTH(5 * 16));
cmd(VERTEX2F(50 * 16, 25 * 16));
cmd(VERTEX2F(60 * 16, 63 * 16));
cmd(COLOR_RGB(128, 128, 0));
cmd(LINE_WIDTH(10 * 16));
cmd(VERTEX2F(83 * 16, 13 * 16));
cmd(VERTEX2F(98 * 16, 63 * 16));
```

Draw line strips for sets of coordinates:



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

Draw Edge strips for above:



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

Draw Edge strips for below:



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


Draw Edge strips for right:



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

Draw Edge strips for left:



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

2.14 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.

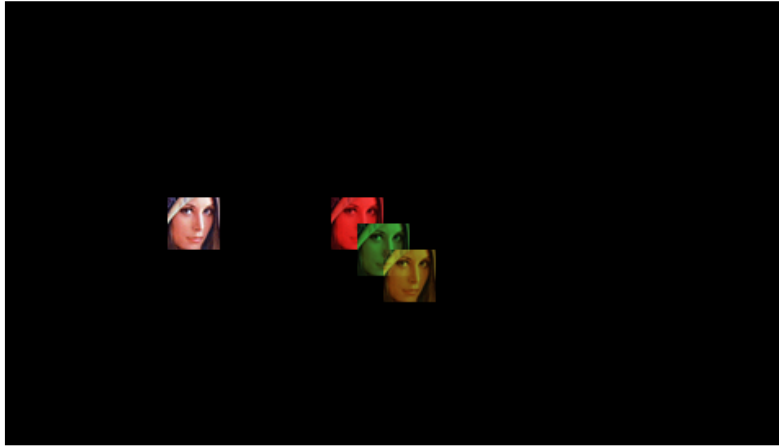
2.15 Color and Transparency

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

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



2.16 Bitmap Formats

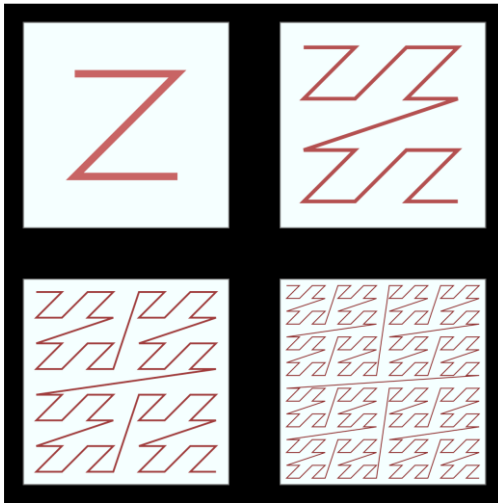
Supported bitmap formats are indicated in the below table. The table column descriptions are as follows:

- **bpp**: bits per pixel
- **bitmap**: the format can be used as a bitmap source
- **JPEG**: the hardware JPEG decompressor can write the format
- **PNG**: the hardware PNG decompressor can write the format
- **LVDSRX**: the hardware LVDS receiver can write the format
- **RT**: the format can be used as a render target by render engine
- **scanout**: the format can be used as a scanout source

format	bpp	note	bitmap	JPEG	PNG	LVDSRX	RT	scanout
L1	1		•					
L2	2		•					
L4	4		•					
L8	8	best quality	•	•	•		•	
LA1	2		•					
LA2	4		•					
LA4	8		•					
LA8	16	best quality	•		•		•	
RGB332	8		•					
RGB565	16		•	•	•	•	•	
RGB6	18		•				•	•
RGB8	24	best quality	•		•	•	•	•
ARGB2	8		•					
ARGB4	16		•		•		•	
ARGB1555	16		•				•	
ARGB6	24		•				•	
ARGB8	32	best quality	•	•	•		•	
PALETTERDARGB8	8	best quality	•		•			
BARGRAPH			•					
ASTC			•					
YCBCR	8		•	•			•	•

Table 9 – Supported bitmap format

2.17 Bitmap ZOrder



Morton ordering, also known as Z-ordering, is a method of transforming multidimensional data into one dimension while preserving the locality of the data points. For bitmaps, which are essentially two-dimensional grids of pixels, Morton ordering interleaves the bits of the x and y coordinates. This interleaving process results in a sequence of addresses that navigate through the bitmap in a "Z" shape. One of the primary advantages of Morton ordering for bitmaps is the preservation of spatial locality. That is, pixels that are close together in the two-dimensional space also tend to be close in the one-dimensional Morton order. By accessing bitmap data in Morton order, there's a higher likelihood of cache hits in memory, leading to improved performance for rotated bitmaps.

BITMAP_ZORDER species a byte that permutes the low-order bits of the (x; y) bitmap coordinates as follows (python function):

```
def zorder(i, j, z):
    # i is row
    # j is column
    # returns (row, column) permuted by z, the zorder

    r = 0
    s = 0
    for k in range(8):
        if ((z >> k) & 1) == 0:
            v = (j & 1)
            j >>= 1
        else:
            v = (i & 1)
            i >>= 1
            s += 1
            r |= (v << k)
            r |= (j << 8)

    return (i << s, r)
```

Each 0 bit in z means that the output x coordinate bit is sourced from the input x coordinate.

Each 1 bit in z means that the output x coordinate bit is sourced from the input y coordinate.

By shooing bits in this way, various Z-order patterns result.

Consider a simple 8x8 bitmap, which displays on the screen using the following address offsets:

0 1 2 3 4 5 6 7

8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63

Setting zorder to 0b000010 gives a 2x2 tiling:

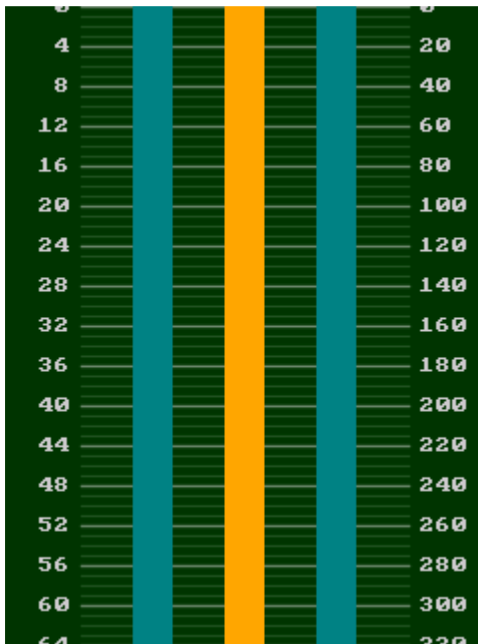
0 1 4 5 8 9 12 13
2 3 6 7 10 11 14 15
16 17 20 21 24 25 28 29
18 19 22 23 26 27 30 31
32 33 36 37 40 41 44 45
34 35 38 39 42 43 46 47
48 49 52 53 56 57 60 61
50 51 54 55 58 59 62 63

With a zorder of 0b101010 the tiling is extended to a 2x2, 4x4 and 8x8 pattern:

0 1 4 5 16 17 20 21
2 3 6 7 18 19 22 23
8 9 12 13 24 25 28 29
10 11 14 15 26 27 30 31
32 33 36 37 48 49 52 53
34 35 38 39 50 51 54 55
40 41 44 45 56 57 60 61
42 43 46 47 58 59 62 63

2.18 REGION Instruction

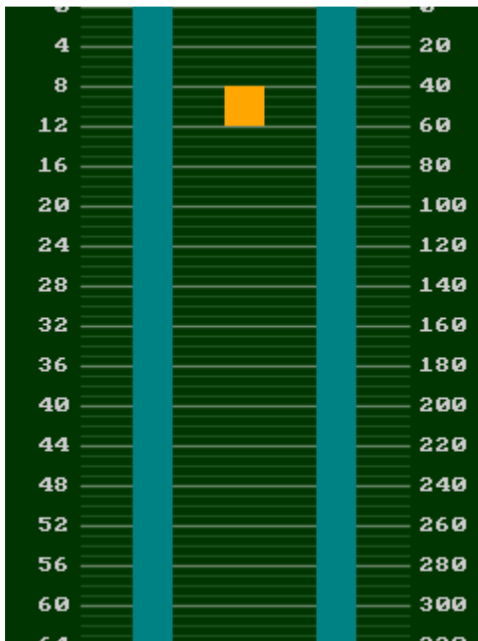
The following display list commands draw three rectangles using **CLEAR** with scissor. Note the center rectangle is wrapped in **SAVE_CONTEXT/RESTORE_CONTEXT**.



```

512 dl(SCISSOR_SIZE(20, 320))
513 dl(CLEAR_COLOR_RGB(0, 128, 128))
514 dl(SCISSOR_XY(64, 0))
515 dl(CLEAR(1, 1, 1))
516 dl(SAVE_CONTEXT())
517 dl(CLEAR_COLOR_RGB(255, 165, 0))
518 dl(SCISSOR_XY(110, 0))
519 dl(CLEAR(1, 1, 1))
520 dl(RESTORE_CONTEXT())
521 d(SCISSOR_XY(156, 0))
522 dl(CLEAR(1, 1, 1))
  
```

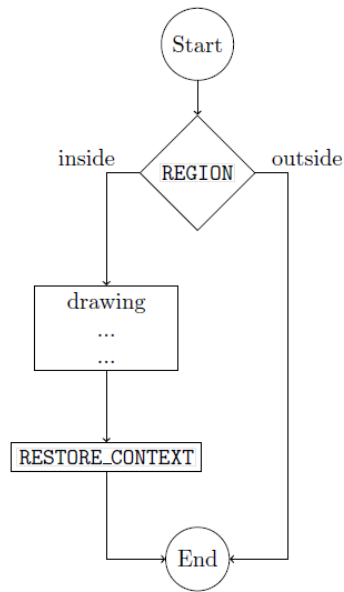
By replacing the **SAVE_CONTEXT** with **REGION**, the block of code from 517-520 is only executed when $(40 \leq y < 60)$. **REGION** is a conditional branch based on the scan-line's y coordinate.



```

512 dl(SCISSOR_SIZE(20, 320))
513 dl(CLEAR_COLOR_RGB(0, 128, 128))
514 dl(SCISSOR_XY(64, 0))
515 dl(CLEAR(1, 1, 1))
516 dl(REGION(8, 4, 521))
517 dl(CLEAR_COLOR_RGB(255, 165, 0))
518 dl(SCISSOR_XY(110, 0))
519 dl(CLEAR(1, 1, 1))
520 dl(RESTORE_CONTEXT())
521 d(SCISSOR_XY(156, 0))
522 dl(CLEAR(1, 1, 1))
  
```

REGION operates as a conditional branch. For lines that are inside the region, it does not take the branch, and executes as a **SAVE_CONTEXT**. For lines that are outside the region, it branches to the specified instruction.



The effect is that the enclosed code is conditionally executed, but because of the implicit **SAVE_CONTEXT** and explicit **RESTORE_CONTEXT**, at End the graphics state is the same whether the branch was taken or not.

Coordinate Calculation

REGION uses a pair of 6-bit fields to express the active region, whatever the screen size. For a graphical element starting at y_0 and ending just above y_1 , with **REG_RE_H** as H , the y and h values can be computed with:

```

y0s = (y0 * 64) / H;           // y0 scaled to 0-63, rounded down
y1s = (y1 * 64 + (H - 1)) / H; // y1 scaled to 0-63, rounded up
y = y0s;                       // start position
h = y1s - y0s;                 // height
  
```

Notes

When rendering in portrait mode (**REG_RE_ROTATE** non-zero), **REGION** uses the X-coordinates of the graphics elements, because of the coordinate swap.

REGION does not take **VERTEX_TRANSLATE_Y** into account.

Because bitmap parameters are not affected by **SAVE_CONTEXT** / **RESTORE_CONTEXT** it is recommended to avoid changing bitmap parameters inside a **REGION** block.

2.19 System Limits

This section outlines the limitations for **BT82X**.

2.19.1 Render Engine

There are some limitations that the render engine can handle a bitmap.

- The maximum supported bitmap size is 2048x2048
- The maximum supported bitmap stride is 8192 bytes

2.19.2 Video Playback

The video playback pipeline can only guarantee playback up to certain video stream bandwidth. The following table shows the maximum bandwidth video stream that can be played back without frame dropping, in Mbit/s.

Format	OPT_SOUND	Audio Channels	Bandwidth
YCBCR	No	2	48
YCBCR	Yes	2	44
YCBCR	No	1	52
YCBCR	Yes	1	48
RGB565	No	2	36
RGB565	Yes	2	32
RGB565	No	1	40
RGB565	Yes	1	36

Table 10 – Video Playback Maximum Bandwidth

3 Register Description

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

- Graphics Engine Registers,
- Audio Engine Registers,
- Flash Interface Registers,
- Touch Engine Registers,
- Swap Chain Registers,
- LVDSRX Registers,
- LVDSTX Registers,
- System Registers,
- I2S Registers,
- Coprocessor Engine Registers,
- Miscellaneous Registers,
- Auxiliary Registers.

The detailed definition for each register is listed here. All the registers in **BT82X** are **32** bits wide. 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.

3.1 Graphics Engine Registers

The base address is **0x7F00_6000**.

REG_TAG Definition			
31		24	23
		0	
Reserved		r/o	
Offset: 0xC4		Reset Value: 0x0	
Bit 31 – 24: Reserved bits			
Bit 23 – 0: These bits are updated with the tag value. The tag value here corresponds 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: 0xC0		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: 0xBC		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_POL Definition			
31		1	0
Reserved		r/w	
Offset: 0xB8		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 4 – REG_PCLK_POL Definition

REG_DLSWAP Definition			
31		2	1 0
Reserved		r/w	
Offset: 0xB4		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 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 5 – REG_DLSWAP Definition

REG_VSYNC1 Definition			
31		12	11 0
Reserved		r/w	
Offset: 0xB0		Reset Value: 0x3	
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 6 – REG_VSYNC1 Definition

REG_VSYNC0 Definition			
31		12	11 0
Reserved		r/w	
Offset: 0xAC		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many lines of signal VSYNC0 takes at the start of a new frame.			

Register Definition 7 – REG_VSYNC0 Definition

REG_VSIZE Definition			
31		12	11
	Reserved		r/w
Offset: 0xA8		Reset Value: 0x320	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many lines of pixels in one frame. The valid range is from 0 to 2047.			

Register Definition 8 – REG_VSIZE Definition

REG_VOFFSET Definition			
31		12	11
	Reserved		r/w
Offset: 0xA4		Reset Value: 0x6	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many lines taken after the start of a new frame.			

Register Definition 9 – REG_VOFFSET Definition

REG_VCYCLE Definition			
31		12	11
	Reserved		r/w
Offset: 0xA0		Reset Value: 0x337	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many lines in one frame.			

Register Definition 10 – REG_VCYCLE Definition

REG_HSYNC1 Definition			
31		12	11
	Reserved		r/w
Offset: 0x9C		Reset Value: 0x32	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many PCLK cycles for HSYNC1 during start of line.			

Register Definition 11 – REG_HSYNC1 Definition

REG_HSYNC0 Definition			
31		12	11
	Reserved		r/w
Offset: 0x98		Reset Value: 0x0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many PCLK cycles for HSYNC0 during start of line.			

Register Definition 12 – REG_HSYNC0 Definition

REG_HSIZE Definition			
31		12	11
	Reserved		r/w
Offset: 0x94		Reset Value: 0x500	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many PCLK cycles per horizontal line.			
Note: REG_HSIZE must be a multiple of 16 and total number of displayed pixels (REG_HSIZE x REG_VSIZE) must be a multiple of 128			

Register Definition 13 – REG_HSIZE Definition

REG_HOFFSET Definition			
31		12	11
	Reserved		r/w
Offset: 0x90		Reset Value: 0x64	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify how many PCLK cycles before pixels are scanned out.			

Register Definition 14 – REG_HOFFSET Definition

REG_HCYCLE Definition			
31		12	11
	Reserved		r/w
Offset: 0x8C		Reset Value: 0x5A0	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits specify total PCLK cycles per horizontal line scan.			

Register Definition 15 – REG_HCYCLE Definition

REG_RE_DEST Definition			
31			0
		r/w	
Offset: 0x10		Reset Value: 0x00000000	
Bit 31 – 0: The destination frame buffer address where the render engine sends its output data. To use the buffers managed by swapchain 0, set it to the value (0xFFFF00FF)			

Register Definition 16 – REG_RE_DEST Definition

REG_RE_FORMAT Definition			
31		16	15
	Reserved		r/w
Offset: 0x14		Reset Value: 0x13	
Bit 31 – 16: Reserved Bits			
Bit 15 – 0: These bits are RE format, as bitmap format.			

Register Definition 17 – REG_RE_FORMAT Definition

REG_RE_ROTATE Definition			
31		3	2
	Reserved		r/w
Offset: 0x18		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 18 – REG_RE_ROTATE Definition

REG_RE_W Definition			
31		12	11
	Reserved		r/w
Offset: 0x1C		Reset Value: 0x500	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are RE target width, in pixels.			

Register Definition 19 – REG_RE_W Definition

REG_RE_H Definition			
31		12	11
	Reserved		r/w
Offset: 0x20		Reset Value: 0x320	
Bit 31 – 12: Reserved Bits			
Bit 11 – 0: These bits are RE target height, in pixels.			

Register Definition 20 – REG_RE_H Definition

REG_RE_DITHER Definition			
31		1	0
	Reserved		r/w
Offset: 0x24		Reset Value: 0x0	
Bit 31 – 1: Reserved bits			
Bit 0: This bit controls RE target dither.			
0b'0: RE target dither is disabled			
0b'1: RE target dither is enabled			

Register Definition 21 – REG_RE_DITHER Definition

REG_RE_ACTIVE Definition			
31		1	0
	Reserved		r/o
Offset: 0x28		Reset Value: -	
Bit 31 – 1: Reserved bits			
Bit 0: This bit is to confirm RE write path.			
0b'0: RE write path is inactive			
0b'1: RE write path is active			

Register Definition 22 – REG_RE_ACTIVE Definition

REG_RE_RENDERERS Definition			
31			0
	r/o		
Offset: 0x2C		Reset Value: -	
Bit 31 – 0: RE render counter.			

Register Definition 23 – REG_RE_RENDERERS Definition

REG_SO_MODE Definition			
31		3	2
	Reserved		r/w
Offset: 0x5F4		Reset Value: 0x0	
Bit 31 – 3: Reserved bits			
Bit 2 – 0: Scanout pixel mode.			
0b'000: 1-pixel per clock for single LVDS channel mode			
0b'001: 2-pixel per clock for single LVDS channel mode			
0b'010: 2-pixel per clock for dual LVDS channel mode			
0b'011: 4-pixel per clock for dual LVDS channel mode			
Note: When REG_SO_MODE is 1 or 2, REG_HCYCLE, REG_HOFFSET, REG_HSYNC0 and REG_HSYNC1 must be even. When REG_SO_MODE is 3, they must all be a multiple of 4			

Register Definition 24 – REG_SO_MODE Definition

REG_SO_SOURCE Definition			
31			0
	r/w		
Offset: 0x5F8		Reset Value: 0	
Bit 31 – 0: The memory address from which the scanout engine retrieves its data. To use the buffer managed by swapchain 0, set it to the value swapchain_0 (0xFFFF00FF)			

Register Definition 25 – REG_SO_SOURCE Definition

REG_SO_FORMAT Definition		
31	16	15
Reserved		r/w
Offset: 0x5FC		Reset Value: 0x13
Bit 31 – 16: Reserved Bits		
Bit 15 – 0: These bits are Scanout format. Refer to Table 9 – Supported bitmap format for the supported Scanout format		

Register Definition 26 – REG_SO_FORMAT Definition

REG_SO_EN Definition		
31		1 0
Reserved		r/w
Offset: 0x600		Reset Value: 0
Bit 31 – 1: Reserved bits		
Bit 0: This bit is to enable/disable Scanout. 0b'0: Disable 0b'1: Enable		

Register Definition 27 – REG_SO_EN Definition

3.2 Audio Engine Registers

The base address is **0x7F00_6000**.

REG_PLAY Definition		
31		1 0
Reserved		r/w
Offset: 0xD8		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 28 – REG_PLAY Definition

REG_SOUND Definition		
31	16	15
Reserved		r/w
Offset: 0xD4		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 BT82X datasheet for details of this register.		

Register Definition 29 – REG_SOUND Definition

REG_VOL_SOUND Definition		
31		8 7
Reserved		r/w
Offset: 0xD0		Reset Value: 0xFF
Bit 31 – 8: Reserved bits		
Bit 7 – 0: These bits control the volume of the synthesizer sound. The default value is 0xFF which is the highest volume. The value zero means mute.		

Register Definition 30 – REG_VOL_SOUND Definition

REG_VOL_L_PB Definition		
31	8 7	0
Reserved		r/w
Offset: 0xC8		Reset Value: 0xFF
Bit 31 – 8: Reserved bits		
Bit 7 – 0: These bits control the volume of the audio file playback left. The default value is 0xFF which is the highest volume. The value zero means mute.		

Register Definition 31 – REG_VOL_L_PB Definition

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

Register Definition 32 – REG_VOL_R_PB Definition

REG_PLAYBACK_PLAY Definition		
31		1 0
Reserved		r/w
Offset: 0x124		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 section 2.10.2 for details of this register.		

Register Definition 33 – REG_PLAYBACK_PLAY Definition

REG_PLAYBACK_LOOP Definition		
31		1 0
Reserved		r/w
Offset: 0x120		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. 0b'0: LOOP is disabled 0b'1: LOOP is enabled		
Note: Please refer to section 2.10.2 for details of this register.		

Register Definition 34 – REG_PLAYBACK_LOOP Definition

REG_PLAYBACK_FORMAT Definition		
31		3 2 0
Reserved		r/w
Offset: 0x11C		Reset Value: 0x0
Bit 31 – 3: Reserved bits		
Bit 2 – 0: These bits define the format of the audio data in RAM_G . 0b'000: Linear Sample format 0b'001: uLaw Sample format 0b'010: 4 bit IMA ADPCM Sample format 0b'011: S16 MONO. 0b'100: S16S STEREO. Others: Undefined.		
Note: Please refer to section 2.10.2 for details of this register.		

Register Definition 35 – REG_PLAYBACK_FORMAT Definition

REG_PLAYBACK_FREQ Definition			
31	16	15	0
Reserved		r/w	
Offset: 0x118		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 section 2.10.2 for details of this register.			

Register Definition 36 – REG_PLAYBACK_FREQ Definition

REG_PLAYBACK_READPTR Definition			
31	30	0	
Reserved	r/o		
Offset: 0x114		Reset Value: 0x0	
Bit 31: Reserved bit			
Bit 30 – 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.			
Note: Please refer to section 2.10.2 for details of this register.			

Register Definition 37 – REG_PLAYBACK_READPTR Definition

REG_PLAYBACK_LENGTH Definition			
31	30	0	
Reserved	r/w		
Offset: 0x110		Reset Value: 0x0	
Bit 31: Reserved bit			
Bit 30 – 0: These bits specify the length of audio data in RAM_G to playback, starting from the address specified in REG_PLAYBACK_START register.			
Note: Please refer to section 2.10.2 for details of this register.			

Register Definition 38 – REG_PLAYBACK_LENGTH Definition

REG_PLAYBACK_START Definition			
31	30	0	
Reserved	r/w		
Offset: 0x10C		Reset Value: 0x0	
Bit 31: Reserved bits			
Bit 30 – 0: These bits specify the start address of audio data in RAM_G to playback.			
Note: Please refer to section 2.10.2 for details of this register.			

Register Definition 39 – REG_PLAYBACK_START Definition

REG_PLAYBACK_PAUSE Definition			
31			1 0
Reserved			r/w
Offset: 0x5D0		Reset Value: 0x0	
Bit 31 – 1: Reserved bits			
Bit 0: Audio playback control bit.			
0b'0: Start playback			
0b'1: Pause playback			
Note: This register can only be used to pause or resume audio loaded in RAM_G using cmd_loadwav . Writing to this register while audio is played with cmd_playwav may result in undefined behavior. Please refer to section 2.10.2 for details of this register.			

Register Definition 40 – REG_PLAYBACK_PAUSE Definition

REG_AUD_PWM Definition				
31				3 2 1 0
Reserved				r/w Res r/w
Offset: 0x13C			Reset Value: 0x0	

Bit 31 – 3: Reserved bits	
Bit 2:	Mixed audio output format appears on AUDIO_L and AUDIO_R pins 0b'0: Delta-sigma format 0b'1: PWM format
Bit 1: Reserved	
Bit 0:	Audio filter bank 0b'0: Enable 0b'1: Disable

Register Definition 41 – REG_AUD_PWM Definition

REG_I2S_EN Definition	
31	1 0
Reserved	
Reset Value: 0x0	
Offset: 0x714	
Bit 31 – 1: Reserved bits	
Bit 0:	I2S interface control bit. 0b'0: Disable I2S interface 0b'1: Enable I2S interface

Register Definition 42 – REG_I2S_EN Definition

REG_I2S_FREQ Definition	
31	21 20 0
Reserved	
Reset Value: 0xAC44	
Offset: 0x718	
Bit 31 – 21: Reserved bits	
Bit 20 – 0: I2S sampling frequency in Hz. If I2S sampling frequency is 44.1KHz, write 44100 (0xAC44) to this register.	

Register Definition 43 – REG_I2S_FREQ Definition

3.3 Flash Interface Registers

The base address is **0x7F00_6000**.

REG_FLASH_STATUS Definition	
31	2 1 0
Reserved	
Reset Value: 0x0	
Offset: 0x5D4	
Bit 31 – 2: Reserved bits	
Bit 1 – 0:	These bits reflect the status of attached flash. 0b'00: FLASH_STATUS_INIT 0b'01: FLASH_STATUS_DETACHED 0b'10: FLASH_STATUS_BASIC 0b'11: FLASH_STATUS_FULL

Register Definition 44 – REG_FLASH_STATUS Definition

The base address is **0x7F00_0000**.

REG_FLASH_SIZE Definition	
31	0
r/o	
Reset Value: 0x0	
Offset: 0x4024	
Bit 31 – 0: The value indicates the capacity of attached flash in Mbytes.	

Register Definition 45 – REG_FLASH_SIZE Definition

3.4 Touch Screen Engine Registers

The base address is **0x7F00 6000**.

Offset	Register Name	Description
0x158	REG_TOUCH_MODE	Touch screen sampling Mode
0x164	REG_TOUCH_RAW_XY	Touch screen raw x,y(16,16)
0x160	REG_TOUCH_SCREEN_XY	Touch screen x,y(16,16)
0x174	REG_TOUCH_TAG_XY	coordinate used to calculate the tag of touch point
0x178	REG_TOUCH_TAG	Touch screen Tag result 0
0x17C	REG_TOUCH_TAG1_XY	coordinate used to calculate the tag of touch point
0x180	REG_TOUCH_TAG1	Touch screen Tag result 1
0x184	REG_TOUCH_TAG2_XY	coordinate used to calculate the tag of touch point
0x188	REG_TOUCH_TAG2	Touch screen Tag result 2
0x18C	REG_TOUCH_TAG3_XY	coordinate used to calculate the tag of touch point
0x190	REG_TOUCH_TAG3	Touch screen Tag result 3
0x194	REG_TOUCH_TAG4_XY	coordinate used to calculate the tag of touch point
0x198	REG_TOUCH_TAG4	Touch screen Tag result 4
0x19C - 0x1B0	REG_TOUCH_TRANSFORM_A-F	Transform coefficient matrix coefficient
0x1B4	REG_TOUCH_CONFIG	Configuration register

Table 11 – Touch Registers Summary

REG_TOUCH_CONFIG Definition							
31		12	11	10	4	3	0
Reserved			r/w		r/w		r/w
Offset: 0x1B4					Reset Value: 0x0		
Bit 31 – 12: Reserved Bits							
Bit 11 : I2C_SPEED configuration 0b'0: 400 KHz 0b'1: 100 KHz							
Bit 10 – 4 : I2C_ADDRESS of touch screen controller 0x38: FocalTech device address (e.g. FT5206) 0x5D: Goodix device address (e.g. GT911) 0x4D: Microchip device address (e.g. AR1021) 0x48: TI device address (e.g. TSC2007) 0x55: Sitronix device address (e.g. ST1633i)							
Bit 3 – 0 : DEVICE_SELECT 0b'0001: FocalTech 0b'0010: Goodix 0b'0011: Microchip 0b'0100: Reserved 0b'0101: TI 0b'0110: Sitronix 0b'0111 to 0b'1111: Reserved							
Note: I2C_ADDRESS must match DEVICE_SELECT. IF both I2C_ADDRESS and DEVICE_SELECT are zero, auto discovery is triggered. In this mode, EVE probes for attached I2C touch controllers in ascending order of DEVICE_SELECT. Among the six supported touch controllers, Microchip's AR1021 and TI's TSC2007 are resistive touch controllers and do not support extended mode .							

Register Definition 46 – REG_TOUCH_CONFIG Definition

REG_TOUCH_TRANSFORM_F Definition						
31	30		16	15		0
r/w		r/w		r/w		
Offset: 0x1B0					Reset Value: 0x0	
Bit 31 : The sign bit for fixed-point number.						

Bit 30 – 16: These bits represent the integer part of the fixed-point number.
 Bit 15 – 0 : These bits represent 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 47 – REG_TOUCH_TRANSFORM_F Definition

REG_TOUCH_TRANSFORM_E Definition			
31	30	16	15
r/w			
Offset: 0x1AC		Reset Value: 0x10000	
Bit 31 : The sign bit for fixed-point number.			
Bit 30 – 16: These bits represent the integer part of the fixed-point number.			
Bit 15 – 0 : These bits represent 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 48 – REG_TOUCH_TRANSFORM_E Definition

REG_TOUCH_TRANSFORM_D Definition			
31	30	16	15
r/w			
Offset: 0x1A8		Reset Value: 0x0	
Bit 31 : The sign bit for fixed-point number.			
Bit 30 – 16: These bits represent the integer part of the fixed-point number.			
Bit 15 – 0 : These bits represent 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 49 – REG_TOUCH_TRANSFORM_D Definition

REG_TOUCH_TRANSFORM_C Definition			
31	30	16	15
r/w			
Offset: 0x1A4		Reset Value: 0x0	
Bit 31 : The sign bit for fixed-point number.			
Bit 30 – 16 : These bits represent the integer part of the fixed-point number.			
Bit 15 – 0 : These bits represent 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 50 – REG_TOUCH_TRANSFORM_C Definition

REG_TOUCH_TRANSFORM_B Definition			
31	30	16	15
r/w			
Offset: 0x1A0		Reset Value: 0x0	
Bit 31 : The sign bit for fixed-point number.			
Bit 30 – 16: These bits represent the integer part of the fixed-point number.			
Bit 15 – 0 : These bits represent 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 51 – REG_TOUCH_TRANSFORM_B Definition

REG_TOUCH_TRANSFORM_A Definition			
31	30	16	15
r/w			
Offset: 0x19C		Reset Value: 0x10000	
Bit 31 : The sign bit for fixed-point number.			
Bit 30 – 16: These bits represent the integer part of the fixed-point number.			
Bit 15 – 0 : These bits represent 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 52 – REG_TOUCH_TRANSFORM_A Definition

REG_TOUCH_TAG Definition	
31	24 23 0
Reserved	r/o
Offset: 0x178 Reset Value: 0x0	
Bit 31 – 24: Reserved Bits	
Bit 23 – 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 16,777,215 and the default value of this register is zero, meaning there is no touch by default.	

Register Definition 53 – REG_TOUCH_TAG Definition

REG_TOUCH_TAG_XY Definition	
31	16 15 0
r/o	r/o
Offset: 0x174 Reset Value: 0x0	
Bit 31 – 16: These bits are the X coordinates of the touch screen to look up the tag result.	
Bit 15 – 0 : 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_TAG1 .	

Register Definition 54 – REG_TOUCH_TAG_XY Definition

REG_TOUCH_TAG1 Definition	
31	24 23 0
Reserved	r/o
Offset: 0x180 Reset Value: 0x0	
Bit 31 – 24: Reserved Bits	
Bit 23 – 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 16,777,215 and the default value of this register is zero, meaning there is no touch by default.	

Register Definition 55 – REG_TOUCH_TAG1 Definition

REG_TOUCH_TAG1_XY Definition	
31	16 15 0
r/o	r/o
Offset: 0x17C Reset Value: 0x0	
Bit 31 – 16: These bits are the X coordinates of the touch screen to look up the tag result.	
Bit 15 – 0: 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_TAG1 .	

Register Definition 56 – REG_TOUCH_TAG1_XY Definition

REG_TOUCH_TAG2 Definition	
31	24 23 0
Reserved	r/o
Offset: 0x188 Reset Value: 0x0	
Bit 31 – 24: Reserved Bits	
Bit 23 – 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 16,777,215 and the default value of this register is zero, meaning there is no touch by default.	

Register Definition 57 – REG_TOUCH_TAG2 Definition

REG_TOUCH_TAG2_XY Definition		
31	16 15	0
r/o		r/o
Offset: 0x184		Reset Value: 0x0
Bit 31 – 16: These bits are the X coordinates of the touch screen to look up the tag result.		
Bit 15 – 0: 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_TAG2 .		

Register Definition 58 – REG_TOUCH_TAG2_XY Definition

REG_TOUCH_TAG3 Definition		
31	24 23	0
Reserved		r/o
Offset: 0x190		Reset Value: 0x0
Bit 31 – 24: Reserved Bits		
Bit 23 – 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 16,777,215 and the default value of this register is zero, meaning there is no touch by default.		

Register Definition 59 – REG_TOUCH_TAG3 Definition

REG_TOUCH_TAG3_XY Definition		
31	16 15	0
r/o		r/o
Offset: 0x18C		Reset Value: 0x0
Bit 31 – 16: These bits are the X coordinates of the touch screen to look up the tag result.		
Bit 15 – 0: 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_TAG3 .		

Register Definition 60 – REG_TOUCH_TAG3_XY Definition

REG_TOUCH_TAG4 Definition		
31	24 23	0
Reserved		r/o
Offset: 0x198		Reset Value: 0x0
Bit 31 – 24: Reserved Bits		
Bit 23 – 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 16,777,215 and the default value of this register is zero, meaning there is no touch by default.		

Register Definition 61 – REG_TOUCH_TAG Definition

REG_TOUCH_TAG4_XY Definition		
31	16 15	0
r/o		r/o
Offset: 0x194		Reset Value: 0x0
Bit 31 – 16: These bits are the X coordinates of the touch screen to look up the tag result.		
Bit 15 – 0: 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_TAG4 .		

Register Definition 62 – REG_TOUCH_TAG_XY Definition

REG_TOUCH_SCREEN_XY Definition		
31	16 15	0
r/o		r/o
Offset: 0x160		Reset Value: 0x80008000
Bit 31 – 16: These bits are the X coordinates of the touch screen. After doing the 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: These bits are the Y coordinates of the touch screen. After doing the calibration, it shall be within the height of the screen size. If the touch screen is not being touched, it shall be 0x8000.		

Register Definition 63 – REG_TOUCH_SCREEN_XY Definition

REG_TOUCH_RAW_XY Definition		
31	16 15	0
r/o		r/o
Offset: 0x164		Reset Value: 0xFFFFFFFF
Bit 31 – 16: These bits are the raw X 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.		
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 64 – REG_TOUCH_RAW_XY Definition

REG_TOUCH_MODE Definition		
31	2 1	0
	Reserved	r/w
Offset: 0x158		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.		
<ul style="list-style-type: none"> 0b'00: TOUCHMODE_OFF. No sampling happens. Touch engine stops working. 0b'01: TOUCHMODE_ONESHOT. Cause one single sample occur. 0b'10: TOUCHMODE_FRAME. Cause a sample at the start of each frame. 0b'11: TOUCHMODE_CONTINUOUS. Up to 1000 times per second. Default mode after reset. 		

Register Definition 65 – REG_TOUCH_MODE Definition

The base address is **0x7F00_6000**.

Offset	Register Name	Description
0x15C	REG_CTOUCH_EXTENDED	Select ADC working mode
0x160	REG_CTOUCH_TOUCH0_XY	Touchscreen touch 0
0x164	REG_CTOUCH_TOUCHA_XY	Touchscreen touch 1
0x168	REG_CTOUCH_TOUCHB_XY	Touchscreen touch 2
0x16C	REG_CTOUCH_TOUCHC_XY	Touchscreen touch 3
0x170	REG_CTOUCH_TOUCH4_XY	Touchscreen touch 4

Table 12 – CTSE Registers Summary

REG_CTOUCH_EXTEND Definition		
31	1	0
	Reserved	r/w
Offset: 0x15C		Reset Value: 0x1
Bit 31 – 1: Reserved bits		
Bit 0 : This bit controls the detection mode of the touch engine.		
<ul style="list-style-type: none"> 0b'0: Extended mode, multi-touch detection mode 0b'1: Compatibility mode, single touch detection mode 		

Register Definition 66 – REG_CTOUCH_EXTENDED Definition

REG_CTOUCH_TOUCH0_XY Definition		
31	16	15 0
	r/o	r/o
Offset: 0x160		Reset Value: 0x80008000
Bit 31 - 16: These bits represent the X coordinate of touch point0.		
Bit 15 - 0: These bits represent the Y coordinate of touch point0.		
Note: This register is only applicable in the extended mode		

Register Definition 67 – REG_CTOUCH_TOUCH0_XY Definition

REG_CTOUCH_TOUCHA_XY Definition		
31	16	15 0
	r/o	r/o
Offset: 0x164		Reset Value: 0x80008000
Bit 31 - 16: These bits represent the X coordinates of touch point1.		
Bit 15 - 0: These bits represent the Y coordinates of touch point1.		
Note: This register is only applicable in the extended mode		

Register Definition 68 – REG_CTOUCH_TOUCHA_XY Definition

REG_CTOUCH_TOUCHB_XY Definition		
31	16	15 0
	r/o	r/o
Offset: 0x168		Reset Value: 0x80008000
Bit 31 - 16: These bits represent the X coordinate of touch point2.		
Bit 15 - 0: These bits represent the Y coordinate of touch point2.		
Note: This register is only applicable in the extended mode		

Register Definition 69 – REG_CTOUCH_TOUCHB_XY Definition

REG_CTOUCH_TOUCHC_XY Definition		
31	16	15 0
	r/o	r/o
Offset: 0x16C		Reset Value: 0x80008000
Bit 31 - 16: These bits represent the X coordinates of touch point3.		
Bit 15 - 0: These bits represent the Y coordinates of touch point3.		
Note: This register is only applicable in the extended mode		

Register Definition 70 – REG_CTOUCH_TOUCHC_XY Definition

REG_CTOUCH_TOUCH4_XY Definition		
31	16	15 0
	r/o	r/o
Offset: 0x170		Reset Value: 0x80008000
Bit 31 - 16: These bits represent the X coordinate of touch point4.		
Bit 15 - 0: These bits represent the Y coordinate of touch point4.		
Note: This register is only applicable in the extended mode		

Register Definition 71 – REG_CTOUCH_TOUCH4_XY Definition

3.4.1 Calibration

The calibration process is initiated by **CMD_CALIBRATE**. 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 extended mode.

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

3.5 Swap Chain Registers

The base address is **0x7F00_6000**.

Offset	Name	Description
0x034	REG_SC0_RESET	Swapchain 0, write to reset
0x038	REG_SC0_SIZE	Swapchain 0, ring size 1-4
0x03C	REG_SC0_PTR0	Swapchain 0, pointer 0
0x040	REG_SC0_PTR1	Swapchain 0, pointer 0
0x044	REG_SC0_PTR2	Swapchain 0, pointer 0
0x048	REG_SC0_PTR3	Swapchain 0, pointer 0
0x04C	REG_SC1_RESET	Swapchain 1, write to reset
0x050	REG_SC1_SIZE	Swapchain 1, ring size 1-4
0x054	REG_SC1_PTR0	Swapchain 1, pointer 0
0x058	REG_SC1_PTR1	Swapchain 1, pointer 0
0x05C	REG_SC1_PTR2	Swapchain 1, pointer 0
0x060	REG_SC1_PTR3	Swapchain 1, pointer 0
0x064	REG_SC2_RESET	Swapchain 2, write to reset
0x068	REG_SC2_SIZE	Swapchain 2, ring size 1-4
0x06C	REG_SC2_PTR0	Swapchain 2, pointer 0
0x070	REG_SC2_PTR1	Swapchain 2, pointer 0
0x074	REG_SC2_PTR2	Swapchain 2, pointer 0
0x078	REG_SC2_PTR3	Swapchain 2, pointer 0
0x780	REG_SC2_STATUS	Swapchain 2 status, write to reset
0x784	REG_SC2_ADDR	Swapchain 2 output buffer

Table 13 – Swap Chain Registers Summary

REG_SC0_RESET Definition	
31	1 0
Reserved	r/o
Offset: 0x034 Reset Value: -	
Bit 31 – 1: Reserved bits	
Bit 0: Write to reset Swapchain 0.	

Register Definition 72 – REG_SC0_RESET Definition

REG_SC0_SIZE Definition	
31	3 2 0
Reserved	r/w
Offset: 0x038 Reset Value: 2	
Bit 31 – 3: Reserved bits	
Bit 2 – 0: The number of buffers managed by Swapchain 0. The valid values are: 2, 3, 4	
Note: Refer to section 2.11 for more details	

Register Definition 73 – REG_SC0_SIZE Definition

REG_SC0_PTR0 Definition	
31	0
r/w	
Offset: 0x03C Reset Value: 0	
Bit 31 – 0: pointer 0 value for Swapchain 0, indicating the address of the first buffer managed by Swapchain 0	

Register Definition 74 – REG_SC0_PTR0 Definition

REG_SC0_PTR1 Definition	
31	0
r/w	
Offset: 0x040	Reset Value: 0
Bit 31 – 0: pointer 1 value for Swapchain 0, indicating the address of the second buffer managed by Swapchain 0	

Register Definition 75 – REG_SC0_PTR1 Definition

REG_SC0_PTR2 Definition	
31	0
r/w	
Offset: 0x044	Reset Value: 0
Bit 31 – 0: pointer 2 value for Swapchain 0, indicating the address of the third buffer managed by Swapchain 0	

Register Definition 76 – REG_SC0_PTR2 Definition

REG_SC0_PTR3 Definition	
31	0
r/w	
Offset: 0x048	Reset Value: 0
Bit 31 – 0: pointer 3 value for Swapchain 0, indicating the address of the fourth buffer managed by Swapchain 0	

Register Definition 77 – REG_SC0_PTR3 Definition

REG_SC1_RESET Definition	
31	1 0
Reserved	
r/o	
Offset: 0x04C	Reset Value: -
Bit 31 – 1: Reserved bits	
Bit 0: Write to reset Swapchain 1.	

Register Definition 78 – REG_SC1_RESET Definition

REG_SC1_SIZE Definition	
31	3 2 0
Reserved	
r/w	
Offset: 0x050	Reset Value: 2
Bit 31 – 3: Reserved bits	
Bit 2 – 0: The number of buffers managed by Swapchain 1. The valid values are: 2, 3, 4	
Note: Refer to section 2.11 for more details	

Register Definition 79 – REG_SC1_SIZE Definition

REG_SC1_PTR0 Definition	
31	0
r/w	
Offset: 0x054	Reset Value: 0
Bit 31 – 0: pointer 0 value for Swapchain 1, indicating the address of the first buffer managed by Swapchain 1	

Register Definition 80 – REG_SC1_PTR0 Definition

REG_SC1_PTR1 Definition	
31	0
r/w	
Offset: 0x058	Reset Value: 0
Bit 31 – 0: pointer 1 value for Swapchain 1, indicating the address of the second buffer managed by Swapchain 1	

Register Definition 81 – REG_SC1_PTR1 Definition

REG_SC1_PTR2 Definition	
31	0
r/w	
Offset: 0x05C	Reset Value: 0
Bit 31 – 0: pointer 2 value for Swapchain 1, indicating the address of the third buffer managed by Swapchain 1	

Register Definition 82 – REG_SC1_PTR2 Definition

REG_SC1_PTR3 Definition	
31	0
r/w	
Offset: 0x060	Reset Value: 0
Bit 31 – 0: pointer 3 value for Swapchain 1, indicating the address of the fourth buffer managed by Swapchain 1	

Register Definition 83 – REG_SC1_PTR3 Definition

REG_SC2_RESET Definition	
31	1 0
Reserved	
r/o	
Offset: 0x064	Reset Value: -
Bit 31 – 1: Reserved bits	
Bit 0: Write to reset Swapchain 2.	

Register Definition 84 – REG_SC2_RESET Definition

REG_SC2_SIZE Definition	
31	3 2 0
Reserved	
r/w	
Offset: 0x068	Reset Value: 2
Bit 31 – 3 : Reserved bits	
Bit 2 – 0: The number of buffers managed by Swapchain 2. The valid values are: 2, 3, 4	
Note: Refer to section 2.11 for more details	

Register Definition 85 – REG_SC2_SIZE Definition

REG_SC2_PTR0 Definition	
31	0
r/w	
Offset: 0x06C	Reset Value: 0
Bit 31 – 0: pointer 0 value for Swapchain 2, indicating the address of the first buffer managed by Swapchain 2	

Register Definition 86 – REG_SC2_PTR0 Definition

REG_SC2_PTR1 Definition	
31	0
r/w	
Offset: 0x070	Reset Value: 0
Bit 31 – 0: pointer 1 value for Swapchain 2, indicating the address of the second buffer managed by Swapchain 2	

Register Definition 87 – REG_SC2_PTR1 Definition

REG_SC2_PTR2 Definition	
31	0
r/w	
Offset: 0x074	Reset Value: 0
Bit 31 – 0: pointer 2 value for Swapchain 2, indicating the address of the third buffer managed by Swapchain 2	

Register Definition 88 – REG_SC2_PTR2 Definition

REG_SC2_PTR3 Definition	
31	0
r/w	
Offset: 0x078	Reset Value: 0
Bit 31 – 0: pointer 3 value for Swapchain 2, indicating the address of the fourth buffer managed by Swapchain 2	

Register Definition 89 – REG_SC2_PTR3 Definition

REG_SC2_STATUS Definition	
31	0
Reserved	
Offset: 0x780	Reset Value: -
Bit 31 – 2: Reserved bits	
Bit 1 – 0: Swapchain 2 status, write to reset	
Note: This register should be read using CMD_REGREAD, CMD_MEMCPY or CMD_WAITCOND	

Register Definition 90 – REG_SC2_STATUS Definition

REG_SC2_ADDR Definition	
31	0
r/o	
Offset: 0x784	Reset Value: 0
Bit 31 – 0: Swapchain 2 input/receive buffer	
The value will only be valid when REG_SC2_STATUS[1:0] is not 0	
Note: This register should be read using CMD_REGREAD, CMD_MEMCPY or CMD_WAITCOND	

Register Definition 91 – REG_SC2_ADDR Definition

3.6 LVDSRX Registers

The LVDS RX interface receives LVDS signals from other devices, such as a video encoder or camera, and captures the pixel data frame into **RAM_G** (DDR memory). By default, Swap Chain 2 is assigned to manage the pixel data buffers written by the LVDS RX interface.

3.6.1 LVDSRX Core Registers

The base address is **0x7F00_6000**.

Offset	Name	Description
0x670	REG_LVDSRX_CORE_ENABLE	LVDSRX enable register
0x674	REG_LVDSRX_CORE_CAPTURE	LVDSRX enable capture register
0x678	REG_LVDSRX_CORE_SETUP	LVDSRX pixel setup control register
0x67C	REG_LVDSRX_CORE_DEST	LVDSRX destination frame address register
0x680	REG_LVDSRX_CORE_FORMAT	LVDSRX output pixel format register
0x684	REG_LVDSRX_CORE_DITHER	LVDSRX enable dither register

Table 14 – LVDSRX registers Summary

REG_LVDSRX_CORE_ENABLE Definition	
31	0
Reserved	
Offset: 0x0670	Reset Value: 0
Bit 31 – 1 : Reserved bits	
Bit 0 : LVDSRX enable	
0b'0: disable LVDSRX	
0b'1: enable LVDSRX	

Register Definition 92 – REG_LVDSRX_CORE_ENABLE Definition

REG_LVDSRX_CORE_CAPTURE Definition			
31		1	0
Reserved			r/w
Offset: 0x0674		Reset Value: 0	
Bit 31 – 1 : Reserved bits			
Bit 0	: LVDSRX enable capture		
	0b'0: disable capture		
	0b'1: enable capture		

Register Definition 93 – REG_LVDSRX_CORE_CAPTURE Definition

REG_LVDSRX_CORE_SETUP Definition			
31		2	1
Reserved		r/w	r/w
Offset: 0x0678		Reset Value: 0	
Bit 31 – 2 : Reserved bits			
Bit 1	: CHCNT (number of channel to receive LVDS signal)		
	Channel count enabled		
	0b'0: one channel		
	0b'1: two channels		
Bit 0	: PPC (number of pixel per clock to receive LVDS signal)		
	Pixels per clock		
	0b'0: 1 pixel per clock		
	0b'1: 2 pixels per clock		

Register Definition 94 – REG_LVDSRX_CORE_SETUP Definition

REG_LVDSRX_CORE_DEST Definition	
31	0
r/w	
Offset: 0x067C	
Reset Value: 0	
Bit 31 – 0: destination frame address	

Register Definition 95 – REG_LVDSRX_CORE_DEST Definition

REG_LVDSRX_CORE_FORMAT Definition			
31		16	15
Reserved		r/w	
Offset: 0x0680		Reset Value: 0x13	
Bit 31 – 16 : Reserved bits			
Bit 15 – 0	: LVDSRX output pixel		
	7 : RGB565		
	19 : RGB8		
	Others: invalid settings		

Register Definition 96 – REG_LVDSRX_CORE_FORMAT Definition

REG_LVDSRX_CORE_DITHER Definition			
31		1	0
Reserved			r/w
Offset: 0x0684		Reset Value: 0	
Bit 31 – 1: Reserved bits			
Bit 0	: LVDSRX enable dither (RGB565 only)		
	0b'0: disable dither		
	0b'1: enable dither		

Register Definition 97 – REG_LVDSRX_CORE_DITHER Definition

3.6.2 LVDSRX System Registers

The base address is **0x7F80_0500**.

Offset	Name	Description
0x00	REG_LVDSRX_SETUP	LVDSRX system set-up
0x04	REG_LVDSRX_CTRL	LVDSRX analog block configuration
0x08	REG_LVDSRX_STAT	LVDSRX status register

Table 15 – LVDSRX registers Summary

REG_LVDSRX_SETUP Definition							
31		5	4	3	2	1	0
Reserved			r/w	r/w	r/w	r/w	r/w
0			0b'10	0	0	0	0
Offset: 0x00							
Bit 31 – 5 : Reserved bits							
Bit 4 – 3: LVDS_MODE[1:0] Selects the LVDS bit mapping schemes. 0b'00: JEIDA_18 0b'01: JEIDA_24 0b'10: VESA_24 0b'11: illegal setting							
Bit 2 : VS_POL VSYNC and HSYNC polarity 0b'0: syncs are active low 0b'1: syncs are active high							
Bit 1 : CHCNT (number of channel to receive LVDS signal from external) Channel count enabled 0b'0: one channel 0b'1: two channels							
Bit 0 : PPC (number of pixel per clock to receive LVDS signal from external) Pixels per clock 0b'0: 1 pixel per clock 0b'1: 2 pixels per clock							

Register Definition 98 – REG_LVDSRX_SETUP Definition

REG_LVDSRX_CTRL Definition														
31		16	15	12	11	10	9	8	7	4	3	2	1	0
Reserved			r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0			0b'1000	0	0b'00	0	0b'1000	0	0b'00	0	0b'00	0	0	0
Offset: 0x04														
Bit 31 - 16: Reserved bits														
Bit 15 - 12: CH1_DS[3:0] Channel 1 de-skew control 55ps (Typ) per step														
Bit 11 : CH1_CLKSEL Channel 1 output clock edge selection 0b'0: data edge aligns with rising edge of RCKO 0b'1: data edge aligns with falling edge of RXCKO														
Bit 10 – 9 : CH1_FRANGE[1:0] Channel 1 input clock frequency range 0b'00: 10 to 30 MHz 0b'01: 30 to 60 MHz 0b'10: 60 to 100 MHz 0b'11: 100 to 180 MHz														
Bit 8 : CH1_PWDN_B Channel 1 power down														

	0b'0: Power down mode (Reset) 0b'1: Normal operation
Bit 7 - 4	: CH0_DS[3:0] Channel 0 de-skew control
Bit 3	: CH0_CLKSEL Channel 0 output clock edge selection 0b'0: data edge aligns with rising edge of RCKO 0b'1: data edge aligns with falling edge of RXCKO
Bit 2 - 1	: CH0_FRANGE[1:0] Channel 0 input clock frequency range 0b'00: 10 to 30 MHz 0b'01: 30 to 60 MHz 0b'10: 60 to 100 MHz 0b'11: 100 to 180 MHz
Bit 0	: CH0_PWDN_B Channel 0 power down 0b'0: Power down mode (Reset) 0b'1: Normal operation

Register Definition 99 – REG_LVDSRX_CTRL Definition

REG_LVDSRX_STAT Definition													
31	28	27	26	25	24	23	16	15	8	7	2	1	0
r/o	r/o	r/o	r/o	r/o	r/w	r/w	r/o	r/o	Reserved	Reserved	r/w	r/w	r/w
0b'1111				0	0	0	0	0	0	0	0	0	0
Offset: 0x08													
Bit 31 - 28: SIG_VALUE[3:0] Real-time values of sync signals. Bit[0] = HSYNC0 Bit[1] = VSYNC0 Bit[2] = HSYNC1 Bit[3] = VSYNC1													
Bit 27 - 26: Reserved bits													
Bit 25 : CH1_DLL_LOCKED Channel 1 LVDSRX DLL status 0b'0: not locked 0b'1: locked Note: If DDL locked status still indicate locked after LVDSRX video input is disconnected, set CH1_PWDN_B to 0 to clear the DDL locked status													
Bit 24 : CH0_DLL_LOCKED Channel 0 LVDSRX DLL status 0b'0: not locked 0b'1: locked Note: If DDL locked status still indicate locked after LVDSRX video input is disconnected, set CH0_PWDN_B to 0 to clear the DDL locked status													
Bit 23 - 16: OVERFLOW_CNT1[7:0] Channel 1 overflow count													
Bit 15 - 8 : OVERFLOW_CNT0[7:0] Channel 0 overflow count													
Bit 7 - 2 : Reserved bits													
Bit 1 : CH1_OVERFLOW Channel 1 overflow flag. Write 0 to clear setting.													
Bit 0 : CH0_OVERFLOW Channel 0 overflow flag. Write 0 to clear setting.													

Register Definition 100 – REG_LVDSRX_STAT Definition

3.7 LV DSTX Registers

To support 2K@60 video streams, **BT82X** implements 2 LVDS channels of 4 data lanes each to transmit parallel 24/18-bit RGB video signals.

BT82X supports two channels, each comprising of 4 LVDS lanes to carry existing pixel video signals of 18 or 24-bit RGB format. The video format selection is controlled by the register REG_LVDSTX_CTRL_CH*. CH*_LVDS_MODE[1:0]. Four modes are implemented as listed in Table 16. VESA/Format 2 mapping for 24-bit RGB is implemented as the default selection for both LVDS channels.

LVDS_MODE[1:0]	Description
11	VESA/Format 2 Mapping for 18-bit, Single Pixel per Clock
10 (default)	VESA/Format 2 Mapping for 24-bit, Single Pixel per Clock
01	JEIDA/Format 1 Mapping for 24-bit, Single Pixel per Clock
00	JEIDA/Format 1 Mapping for 18-bit, Single Pixel per Clock

Table 16 – LVDS mode assignment for VESA and JEIDA formats

3.7.1 LV DSTX System Registers

The base address is **0x7F80_0300**.

Offset	Name	Description
0x00	REG_LVDSTX_EN	LVDS enables
0x04	REG_LVDSTX_PLLCFG	LVDS PLL and Clock configuration
0x14	REG_LVDSTX_CTRL_CH0	LVDS channel 0 control
0x18	REG_LVDSTX_CTRL_CH1	LVDS channel 1 control
0x1C	REG_LVDSTX_STAT	LVDS status
0x20	REG_LVDSTX_ERR_STAT	LVDS error status

Table 17 – LV DSTX registers Summary

REG_LVDSTX_EN Definition				
31		3	2	1 0
	Reserved		r/w	r/w r/o
	0		0	0 0
Offset: 0x00				
Bit 31 – 3 : Reserved bits				
Bit 2	: LVDS_CH1_EN	Enable LVDS Channel 1 output.		
Bit 1	: LVDS_CH0_EN	Enable LVDS Channel 0 output.		
Bit 0	: Reserved bit			

Register Definition 101 – REG_LVDSTX_EN Definition

REG_LVDSTX_PLLCFG Definition						
31	28	27	25	24	13	12 11 10 4 3 0
Reserved	r/w		r/w		r/w	r/w r/w
0	0b'000		384		0b'01	7 7
Offset: 0x04						
Bit 31 – 28 : Reserved bits						
Bit 27 – 25 : LVDSPLL_CPS[2:0]						
LVDS PLL: Loop Bandwidth setting						

Bit 24 – 13 : LVDSPLL_CKS[1:0] LVDS PLL: Lock period in number of osc clock cycles. Note that the internal counter starts from value 0, so actual period is ((LOCK+1)*osclk_period)
Bit 12 – 11 : LVDSPLL_CKS[1:0] LVDS PLL: Output clock range selection. 0b'00: illegal settings 0b'01: CTLCLK = 200MHz ~ 400MHz 0b'10: CTLCLK = 400MHz ~ 800MHz 0b'11: illegal settings
Bit 10 – 4 : LVDSPLL_NS[6:0] LVDS PLL: Programmable loop divider. LVDS PLL freq = NS * PCLK freq
Bit 3 – 0 : TXCLKDIV[3:0] Division factor to generate scanclk_2x from system PLL. Note that the internal counter starts from value 0, so actual division factor is TXCLKDIV+1. ie. txClk freq = SysPLL ckout freq / (TXCLKDIV+1)

Register Definition 102 – REG_LVDSTX_PLLCFG Definition

OSC (MHz)	SYSPLL.NS	Division Factor (TXCLKDIV+1)	LVDSTX clock (MHz)
38.4	15 (default)	4	72.00
		5	57.60
		6	48.00
		7	41.14
		8 (default)	36.00
	13	3	83.20
		4	62.40
		5	49.92
		6	41.60
		7	35.66

Table 18 – LVDSTX Clock Configuration

REG_LVDSTX_CTRL_CH0 Definition																				
31	30	29	23	22	20	19	17	16	14	13	11	10	8	7	5	4	2	1	0	
r/w	r/w		r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	0b'10
0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0b'10
Offset: 0x14																				
Bit 31 : RGB0_RGB1_SWAP Swaps RGB0 and RGB1 This feature is only valid for ExtSync mode 010.																				
Bit 30 : RGB18_REMAP When using 18-bit RGB mode, set 1 to map RGB[7:2] to LVDS data bits[5:0]																				
Bit 29 - 23: Reserved for future use																				
Bit 22 - 20: CH0_CLK_DLY_SEL[2:0] Select one of 7 delay buffers for clock lane																				
Bit 19 - 17: CH0_D3_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 3																				
Bit 16 - 14: CH0_D2_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 2																				
Bit 13 - 11: CH0_D1_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 1																				
Bit 10 - 8 : CH0_D0_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 0																				
Bit 7- 5 : Reserved for future use																				
Bit 4- 2 : CH0_LVDS_LATENCY[2:0] Latency with respect to PCLK. Default value of 0 corresponds to a latency of 2. Valid values of 0~6 to adjust the offset of bit[0] with respect to PCLK.																				
Bit 1 - 0 : CH0_LVDS_MODE[1:0]																				

Lane and colour depth settings.

0b'11: VESA/Format 2 Mapping for 18-bit, Single Pixel per Clock

0b'10: VESA/Format 2 Mapping for 24-bit, Single Pixel per Clock

0b'01: JEIDA/Format 1 Mapping for 24-bit, Single Pixel per Clock

0b'00: JEIDA/Format 1 Mapping for 18-bit, Single Pixel per Clock

Register Definition 103 – REG_LVDSTX_CTRL_CH0 Definition

REG_LVDSTX_CTRL_CH1 Definition																	
31	23	22	20	19	17	16	14	13	11	10	8	7	5	4	2	1	0
Reserved			r/w	r/w		r/w		r/w		r/w		Reserved		r/w		r/w	
0			0	0		0		0		0		0		0		0b'10	
Offset: 0x18																	
Bit 31 - 23: Reserved bits																	
Bit 22 - 20: CH1_CLK_DLY_SEL[2:0] Select one of 7 delay buffers for clock lane																	
Bit 19 - 17: CH1_D3_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 3																	
Bit 16 - 14: CH1_D2_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 2																	
Bit 13 - 11: CH1_D1_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 1																	
Bit 10 - 8 : CH1_D0_DLY_SEL[2:0] Select one of 7 delay buffers for data lane 0																	
Bit 7- 5 : Reserved bits																	
Bit 4- 2 : CH1_LVDS_LATENCY[2:0] Latency with respect to PCLK. Default value of 0 corresponds to a latency of 2. Valid values of 0~6 to adjust the offset of the 0 th bits with respect to PCLK.																	
Bit 1 - 0 : CH1_LVDS_MODE[1:0] Lane and colour depth settings. 0b'11: VESA/Format 2 Mapping for 18-bit, Single Pixel per Clock 0b'10: VESA/Format 2 Mapping for 24-bit, Single Pixel per Clock 0b'01: JEIDA/Format 1 Mapping for 24-bit, Single Pixel per Clock 0b'00: JEIDA/Format 1 Mapping for 18-bit, Single Pixel per Clock																	

Register Definition 104 – REG_LVDSTX_CTRL_CH1 Definition

REG_LVDSTX_STAT Definition																	
31	25	24	21	20	17	16	15	13	12	9	8	5	4	3	1	0	
Reserved		r/o		r/o		r/o		Reserved		r/o		r/o		Reserved		r/o	
0		0		0		0		0		0		0		0		0	
Offset: 0x1C																	
Bit 31 - 25: Reserved bits																	
Bit 24 - 21: LVDS_CH1_OVFCNT[3:0] LVDS CH1 overflow count. Count is reset when LVDS channel is disabled.																	
Bit 20 - 17: LVDS_CH1_UNDFCNT[3:0] LVDS CH1 underflow count. Count is reset when LVDS channel is disabled.																	
Bit 16 : LVDS_CH1_ACTIVE LVDS CH1 active status.																	
Bit 15 - 13: Reserved bits																	
Bit 12 - 9 : LVDS_CH0_OVFCNT[3:0] LVDS CH0 overflow count. Count is reset when LVDS channel is disabled.																	
Bit 8 - 5 : LVDS_CH0_UNDFCNT[3:0] LVDS CH0 underflow count. Count is reset when LVDS channel is disabled.																	
Bit 4 : LVDS_CH0_ACTIVE LVDS CH0 active status.																	
Bit 3 - 1 : Reserved bits																	
Bit 0 : LVDSPLL_LOCK																	

LVDS PLL lock status.
 0b'1: PLL locked
 0b'0: PLL not locked

Register Definition 105 – REG_LVDSTX_STAT Definition

REG_LVDSTX_ERR_STAT Definition						
31		4	3	2	1	0
Reserved			r/o	r/o	r/o	r/o
0			0	0	0	0
Offset: 0x20						
Bit 31 – 4 : Reserved bits						
Bit 3	: OVERFLOW_CH1	LVDS CH1 FIFO overflow error flag. Error is flagged when FIFO overflow condition is detected. Error flag is cleared when LVDS channel is disabled.				
Bit 2	: UNDERFLOW_CH1	LVDS CH1 FIFO underflow error flag. Error is flagged when FIFO underflow condition is detected. Error flag is cleared when LVDS channel is disabled.				
Bit 1	: OVERFLOW_CH0	LVDS CH0 FIFO overflow error flag. Error is flagged when FIFO overflow condition is detected. Error flag is cleared when LVDS channel is disabled.				
Bit 0	: UNDERFLOW_CH0	LVDS CH0 FIFO underflow error flag. Error is flagged when FIFO underflow condition is detected. Error flag is cleared when LVDS channel is disabled.				

Register Definition 106 – REG_LVDSTX_ERR_STAT Definition

3.8 System Registers

System registers retain their settings during system **POWERDOWN** mode. They are only reset when the power-on-reset event is triggered or **RST_N** pin is activated. They can only be updated during system ACTIVE mode when the system clock is available.

The base address is **0x7F80_0400**.

Offset	Name	Description
		Pin drive strength setting. At 3.3V: 0b'00 – 12.5mA 0b'01 – 15mA 0b'10 – 17.5mA 0b'11 – 20mA At 2.5V: 0b'00 – 10mA 0b'01 – 12mA 0b'10 – 14mA 0b'11 – 16mA At 1.8V: 0b'00 – 6mA 0b'01 – 7.2mA 0b'10 – 8.4mA 0b'11 – 9.6mA
0x08	REG_PIN_DRV_0	
0x0C	REG_PIN_DRV_1	Refer to REG_PIN_DRV_0 above.
		Pin output slew rate setting. 0b'0 – fast 0b'1 – slow
0x10	REG_PIN_SLEW_0	
		Pin type setting. 0b'00 – float
0x14	REG_PIN_TYPE_0	

		0b'01 – pull down 0b'10 – pull up 0b'11 – keep last value
0x18	REG_PIN_TYPE_1	Refer to REG_PIN_TYPE_0 above.
0x20	REG_SYS_CFG	Miscellaneous system configuration
0x24	REG_SYS_STAT	System status
0x48	REG_CHIP_ID	CHIP_ID info
0x4C	REG_BOOT_STATUS	EVE boot status
0x54	REG_DDR_TYPE	DDR DRAM type setting This is the actual value after considering any software override via host command. This register is read by EVE to determine which DRAM type is attached.
0x64	REG_PIN_DRV_2	Refer to REG_PIN_DRV_0 above.
0x68	REG_PIN_SLEW_1	Refer to REG_PIN_SLEW_0 above.
0x6C	REG_PIN_TYPE_2	Refer to REG_PIN_TYPE_0 above.

Table 19 – System Register Memory Map Summary

REG_PIN_DRV_0 Definition																															
31	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		
r/o		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w		r/w	
0		0b'00		0b'00		0b'00		0b'00		0b'00		0b'11		0b'00		0b'00		0b'00		0b'00		0b'00		0b'00		0b'00		0b'00		0b'00	
Offset: 0x08																															
Bit 31 – 28 : Reserved bits																															
Bit 27 - 26: SPIS Drive strength setting for SPI pins: SCK, SS_N, MISO, MOSI, IO2, IO3. Note: GPIO0 and GPIO1 shares the same pin as SPIS IO2 and IO3. When SPIS is set in Quad mode, IO2 and IO3 settings will be used, otherwise GPIO0 and GPIO1 settings are used.																															
Bit 25 - 24: GPIO8 Drive strength setting for GPIO8 pin																															
Bit 23 - 22: GPIO7 Drive strength setting for GPIO7 pin																															
Bit 21 - 20: GPIO6 Drive strength setting for GPIO6 pin																															
Bit 19 - 18: GPIO5 Drive strength setting for GPIO5 pin																															
Bit 17 - 16: AUDIO Drive strength setting for AUDIO_L and AUDIO_R pins																															
Bit 15 - 14: BACKLIGHT Drive strength setting for BACKLIGHT pin																															
Bit 13 - 12: DISP Drive strength setting for DISP pin																															
Bit 11 - 10: INT_N Drive strength setting for INT_N pin																															
Bit 9 – 8 : GPIO4 Drive strength setting for GPIO4 pin																															
Bit 7 – 6 : GPIO3 Drive strength setting for GPIO3 pin																															
Bit 5 – 4 : GPIO2 Drive strength setting for GPIO2 pin																															
Bit 3 – 2 : GPIO1 Drive strength setting for GPIO1 pin. Note: GPIO1 shares the same pin as SPI IO3. When SPI is set in Quad mode, IO3 setting will be used, otherwise GPIO1 setting is used.																															
Bit 1 – 0 : GPIO0 Drive strength setting for GPIO0 pin. Note: GPIO0 shares the same pin as SPI IO2. When SPI is set in Quad mode, IO2 setting will be used, otherwise GPIO0 setting is used.																															

Register Definition 107 – REG_PIN_DRV_0 Definition

REG_PIN_DRV_1 Definition																											
31	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		
Reserved		r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w		
0		0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00		
Offset: 0x0C																											
Bit 31 - 24 : Reserved bits																											
Bit 23 - 22: I2S Drive strength setting for I2S pins; I2S_SDAO Not applicable to input only pins; I2S_BCLK, I2S_LRCLK																											
Bit 21 - 20: SDWP Drive strength setting for Drive strength setting for SD card SDWP pin																											
Bit 19 - 18: SDCD Drive strength setting for SD card SDCD pin																											
Bit 17 - 16: SDD Drive strength setting for SD card SDD0, SDD1, SDD2, SDD3 pins																											
Bit 15 - 14: SDCMD Drive strength setting for SD card SDCMD pin																											
Bit 13 - 12: SDCLK Drive strength setting for SD card SDCLK pin																											
Bit 11 - 10: SPIM_IO3 Drive strength setting for SPIM_IO3 pin																											
Bit 9 - 8 : SPIM_IO2 Drive strength setting for SPIM_IO2 pin																											
Bit 7 - 6 : SPIM_MOSI Drive strength setting for SPIM_MOSI pin																											
Bit 5 - 4 : SPIM_MISO Drive strength setting for SPIM_MISO pin																											
Bit 3 - 2 : SPIM_SS_N Drive strength setting for SPIM_SS_N pin																											
Bit 1 - 0 : SPIM_SCLK Drive strength setting for SPIM_SCLK pin																											

Register Definition 108 - REG_PIN_DRV_1 Definition

REG_PIN_SLEW_0 Definition																											
31	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
r/o	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Offset: 0x10																											
Bit 31 - 26 : Reserved bits																											
Bit 25: I2S Slew rate setting for I2S pins; I2S_BCLK, I2S_LRCLK, I2S_SDAO																											
Bit 24: SDWP Slew rate setting for SD card SDWP pin																											
Bit 23: SDCD Slew rate setting for SD card SDCD pin																											
Bit 22: SDD Slew rate setting for SD card SDD0, SDD1, SDD2, SDD3 pin																											
Bit 21: SDCMD Slew rate setting for SD card SDCMD pin																											
Bit 20: SDCLK Slew rate setting for SD card SDCLK pin																											
Bit 19: SPIM_IO3 Slew rate setting for SPIM_IO3 pin																											
Bit 18: SPIM_IO2 Slew rate setting for SPIM_IO2 pin																											
Bit 17: SPIM_MOSI Slew rate setting for SPIM_MOSI pin																											
Bit 16: SPIM_MISO Slew rate setting for SPIM_MISO pin																											

Bit 15: SPIM_SS_N Slew rate setting for SPIM_SS_N pin
Bit 14: SPIM_SCLK Slew rate setting for SPIM_SCLK pin
Bit 13: SPIS Slew rate setting for SPI pins; SCK, SS_N, MISO, MOSI, IO2, IO3. Note: GPIO0 and GPIO1 shares the same pin as SPIS IO2 and IO3. When SPIS is set in Quad mode, IO2 and IO3 settings will be used, otherwise GPIO0 and GPIO1 settings are used.
Bit 12: GPIO8 Slew rate setting for GPIO8 pin
Bit 11: GPIO7 Slew rate setting for GPIO7 pin
Bit 10 - 20: GPIO6 Slew rate setting for GPIO6 pin
Bit 9 : GPIO5 Slew rate setting for GPIO5 pin
Bit 8 : AUDIO Slew rate setting for AUDIO_L and AUDIO_R pins
Bit 7 : BACKLIGHT Slew rate setting for BACKLIGHT pin
Bit 6 : DISP Slew rate setting for DISP pin
Bit 5 : INT_N Slew rate setting for INT_N pin
Bit 4 : GPIO4 Slew rate setting for GPIO4 pin
Bit 3 : GPIO3 Slew rate setting for GPIO3 pin
Bit 2 : GPIO2 Slew rate setting for GPIO2 pin
Bit 1 : GPIO1 Slew rate setting for GPIO1 pin. Note: GPIO1 shares the same pin as SPI IO3. When SPI is set in Quad mode, IO3 setting will be used, otherwise GPIO1 setting is used.
Bit 0 : GPIO0 Slew rate setting for GPIO0 pin. Note: GPIO0 shares the same pin as SPI IO2. When SPI is set in Quad mode, IO2 setting will be used, otherwise GPIO0 setting is used.

Register Definition 109 – REG_PIN_SLEW_0 Definition

REG_PIN_TYPE_0 Definition																													
31	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
r/o	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0	0b'00	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'00	0b'00	0b'00	0b'00	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	
Offset: 0x14																													
Bit 31 - 28 : Reserved bit																													
Bit 27 - 26: SPIS Pin type setting for SPI pins; SCK, SS_N, MISO, MOSI, IO2, IO3. Note: GPIO0 and GPIO1 shares the same pin as SPIS, IO2 and IO3. When SPIS is set in Quad mode, IO2 and IO3 settings will be used, otherwise GPIO0 and GPIO1 settings are used.																													
Bit 25 - 24: GPIO8 Pin type setting for GPIO8 pin																													
Bit 23 - 22: GPIO7 Pin type setting for GPIO7 pin																													
Bit 21 - 20: GPIO6 Pin type setting for GPIO6 pin																													
Bit 19 - 18: GPIO5 Pin type setting for GPIO5 pin																													
Bit 17 - 16: AUDIO																													

Pin type setting for AUDIO_L and AUDIO_R pins	
Bit 15 - 14:	BACKLIGHT Pin type setting for BACKLIGHT pin
Bit 13 - 12:	DISP Pin type setting for DISP pin
Bit 11 - 10:	INT_N Pin type setting for INT_N pin
Bit 9 - 8 :	GPIO4 Pin type setting for GPIO4 pin
Bit 7 - 6 :	GPIO3 Pin type setting for GPIO3 pin
Bit 5 - 4 :	GPIO2 Pin type setting for GPIO2 pin
Bit 3 - 2 :	GPIO1 Pin type setting for GPIO1 pin. Note: GPIO1 shares the same pin as SPI IO3. When SPI is set in Quad mode, IO3 setting will be used, otherwise GPIO1 setting is used. However, the application software might require the SPIS interface to switch between different channel modes, including QUAD mode. In this case, external pull-downs will be required on IO3 pins, even though SPIS is not consistently in QUAD mode. The GPIO1 PIN_TYPE settings should then be set to match the SPIS setting (i.e. PIN_TYPE =0b'00 (float)). Leaving it in its default pull-up setting will cause unnecessary power leakage.
Bit 1 - 0 :	GPIO0 Pin type setting for GPIO0 pin. Note: GPIO0 shares the same pin as SPI IO2. When SPI is set in Quad mode, IO2 setting will be used, otherwise GPIO0 setting is used. However, the application software might require the SPIS interface to switch between different channel modes, including QUAD mode. In this case, external pull-downs will be required on IO2 pins, even though SPIS is not consistently in QUAD mode. The GPIO0 PIN_TYPE settings should then be set to match the SPIS setting (ie. PIN_TYPE =0b'00 (float)). Leaving it in its default pull-up setting will cause unnecessary power leakage.

Register Definition 110 – REG_PIN_TYPE_0 Definition

REG_PIN_TYPE_1 Definition																											
31	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		
Reserved		r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	
0		0b'00	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'00	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10
Offset: 0x18																											
Bit 31 - 24: Reserved bits																											
Bit 23 - 22: I2S																											
Pin type setting for I2S pins; I2S_SDAO. Not applicable to input only pins; I2S_BCLK, I2S_LRCLK																											
Bit 21 - 20: SDWP Pin type setting for SD card SDWP pin																											
Bit 19 - 18: SD CD Pin type setting for SD card SD CD pin																											
Bit 17 - 16: SDD Pin type setting for SD card SDD0, SDD1, SDD2, SDD3 pins																											
Bit 15 - 14: SDCMD Pin type setting for SD card SDCMD pin																											
Bit 13 - 12: SDCLK Pin type setting for SD card SDCLK pin																											
Bit 11 - 10: SPIM_IO3 Pin type setting for SPIM_IO3 pin																											
Bit 9 - 8 : SPIM_IO2 Pin type setting for SPIM_IO2 pin																											
Bit 7 - 6 : SPIM_MOSI Pin type setting for SPIM_MOSI pin																											
Bit 5 - 4 : SPIM_MISO Pin type setting for SPIM_MISO pin																											
Bit 3 - 2 : SPIM_SS_N																											

	Pin type setting for SPIM_SS_N pin
Bit 1 – 0	: SPIM_SCLK Pin type setting for SPIM_SCLK pin

Register Definition 111 – REG_PIN_TYPE_1 Definition

REG_SYS_CFG Definition											
31		12	11	10	9	8	7		2	1	0
Reserved			r/w	r/w	r/w	r/o			r/w	r/w	
0			0	0	0	0			0	0	
Offset: 0x20											
Bit 31 – 12: Reserved bits											
Bit 11	: SPIM_POL_CTL Determine sampling edge of SPIM_SCLK for flash memory data. This bit should not be updated when flash interface is active. 0b'0: sample on falling edge 0b'1: sample on rising edge										
Bit 10	: SPIS_RBURST_DISABLE Disable SPIS read burst transaction. 0b'0: SPIS read burst enabled 0b'1: SPIS read burst disabled										
Bit 9 – 8	: SPI_WIDTH[1:0] Configure SPIS data width 0 – 1bit 1 – 2bits 2 – 4bits 3 – unused, defaults to 1bit if set										
Note: Please refer to BT82X datasheet for more details.											
Bit 7 – 2 : Reserved for future use											
Bit 1	: SD_EN Enable SD clock generation 0b'0: disable SD clock 0b'1: enable SD clock										
Bit 0	: INT_N_PIN_TYPE Enable push-pull mode for interrupt (INT_N) pad 0b'0: INT_N in open-drain mode 0b'1: INT_N in push-pull mode										

Register Definition 112 – REG_SYS_CFG Definition

REG_SYS_STAT Definition																
31	30	29	28		17	16	15	14	8	7		4	3	2	1	0
Reserved		r/o	r/o			r/o	r/o			r/o	r/o	r/o	r/o	r/o	r/o	r/o
0		0	384			0b'10	15			7	0	0	0	0	0	0
Offset: 0x24																
Bit 31 – 30: Reserved bits																
Bit 29	: DDRTYPE_USER_SETTING Select DDR type settings from default or user setting via host command. 0b'0: Use default REG_DDR_TYPE settings 0b'1: Use user REG_DDR_TYPE value via host command															
Bit 28 – 17: Hard-coded value: 384																
Bit 16 – 15	: SYSPLL_CKS[1:0] SYS PLL: Output clock range selection. 0b'10: CTLCLK = 400MHz ~ 800MHz															
Bit 14 – 8	: SYSPLL_NS[6:0] SYS PLL: Programmable loop divider. SYS PLL frequency = NS * Oscillator frequency Value programmed via host command															
Bit 7 – 4	: SYSCLKDIV[3:0] Division factor to generate system clock from system PLL. Value programmed via host command															

Note: The internal counter starts from value 0, so actual division factor is SYSCLKDIV +1.

Bit 3	: MREPAIR_FAIL SRAM memory repair status. 0b'0: repair passed 0b'1: repair failed
Bit 2	: MREPAIR_DONE SRAM memory repair status. 0b'0: repair in progress 0b'1: repair done
Bit 1	: BOOT_USER_SETTING Select REG_BOOT_CFG settings from default or user setting via host command. 0b'0: Use default REG_BOOT_CFG settings 0b'1: Use REG_BOOT_CFG user value via host command
Bit 0	: BOOTCFGEN_ALLOW Enable update of REG_BOOT_CFG settings 0b'0: ignore any update 0b'1: enable update Value programmed via host command

Register Definition 113 – REG_SYS_STAT Definition

REG_CHIP_ID Definition	
31	0
r/o	
0x08200100	
Offset: 0x48	
Bit 31 – 0: CHIP_ID 0x08200100	

Register Definition 114 – REG_CHIP_ID Definition

REG_BOOT_STATUS Definition	
31	0
r/o	
0	
Offset: 0x4C	
Bit 31 – 0: BOOT_STATUS EVE will write to this register to indicate its boot status. Host MCU can then read this register to understand EVE's current boot status. Host should not write to this register as it will corrupt the record of EVE's boot status. Register content is cleared when EVE reset is triggered.	
Note: Refer to Table 7 – EVE Boot Status for BOOT_STATUS details	

Register Definition 115 – REG_BOOT_STATUS Definition

REG_DDR_TYPE Definition				
31		87	54	32 0
Reserved		r/o	r/o	r/o
0		0	1	0
Offset: 0x54				
Bit 31 – 8: Reserved bits				
Bit 7 - 5 : DDR_SPEEDGRADE DDR DRAM speed grade 0 – 1333 MT/s 1 – 1066 MT/s 2 - 933 MT/s 3 - 800 MT/s 4 to 7 - reserved				
Bit 4 - 3 : DDR_TYPE DDR DRAM type				

0 - DDR3 1 - DDR3L (Default) 2 - LPDDR2 3 - reserved
Bit 2 - 0 : DDR_SIZE DDR DRAM size 0 - 1Gbits 1 - 2Gbits 2 - 4Gbits 3 - 8Gbits 4 - 512Mbits 5 to 7 - reserved

Register Definition 116 – REG_DDR_TYPE Definition

REG_PIN_DRV_2 Definition															
31	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0				0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00	0b'00
Offset: 0x64															
Bit 31 – 14: Reserved bits															
Bit 13 – 12: GPIO15 Drive strength setting for GPIO15 pin															
Bit 11 – 10: GPIO14 Drive strength setting for GPIO14 pin															
Bit 9 – 8 : GPIO13 Drive strength setting for GPIO13 pin															
Bit 7 – 6 : GPIO12 Drive strength setting for GPIO12 pin															
Bit 5 – 4 : GPIO11 Drive strength setting for GPIO11 pin															
Bit 3 – 2 : GPIO10 Drive strength setting for GPIO10 pin															
Bit 1 – 0 : GPIO9 Drive strength setting for GPIO9 pin															

Register Definition 117 – REG_PIN_DRV_2 Definition

REG_PIN_SLEW_1 Definition														
31	7	6	5	4	3	2	1	0	0					
Reserved							r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0							0	0	0	0	0	0	0	0
Offset: 0x68														
Bit 31 – 7: Reserved bits														
Bit 6 : GPIO15 Slew rate setting for GPIO15 pin														
Bit 5 : GPIO14 Slew rate setting for GPIO14 pin														
Bit 4 : GPIO13 Slew rate setting for GPIO13 pin														
Bit 3 : GPIO12 Slew rate setting for GPIO12 pin														
Bit 2 : GPIO11 Slew rate setting for GPIO11 pin														
Bit 1 : GPIO10 Slew rate setting for GPIO10 pin														
Bit 0 : GPIO9 Slew rate setting for GPIO9 pin														

Register Definition 118 – REG_PIN_SLEW_1 Definition

REG_PIN_TYPE_2 Definition															
31	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0		0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10	0b'10
Offset: 0x6C															
Bit 31 - 14: Reserved bits															
Bit 13 - 12: GPIO15 Pin type setting for GPIO15 pin															
Bit 11 - 10: GPIO14 Pin type setting for GPIO14 pin															
Bit 9 - 8 : GPIO13 Pin type setting for GPIO13 pins															
Bit 7 - 6 : GPIO12 Pin type setting for GPIO12 pin															
Bit 5 - 4 : GPIO11 Pin type setting for GPIO11 pin															
Bit 3 - 2 : GPIO10 Pin type setting for GPIO10 pin															
Bit 1 - 0 : GPIO9 Pin type setting for GPIO9 pin															

Register Definition 119 – REG_PIN_TYPE_2 Definition

3.9 I2S Registers

The following is a summary of the main I2S features supported by **BT82X**:

- Only I2S transmit functionality is available. Receive functionality is disabled.
- Only I2S slave mode is supported. Master mode support is disabled.
- Supported audio data lengths per channel: 16, 20, 24 or 32 bits.
Audio formats supported: Left-Justified, Right-Justified and I2S mode.
- Max I2S_BCLK frequency supported is 12.288MHz.
- An interrupt output is available to flag I2S data buffer status.

Note: EVE only supports 16bits audio data lengths.

The base address is **0x7F80_0800**.

Offset	Name	Description
0x00	REG_I2S_CFG	I2S configuration registers
0x04	REG_I2S_CTL	I2S control registers
0x10	REG_I2S_STAT	I2S status
0x14	REG_I2S_PAD_CFG	I2S padding configuration

Table 20 – I2S Memory Map Summary

REG_I2S_CFG Definition											
31	12	11	10	9	8	7	5	4	3	2	0
Reserved		r/w	r/w	Reserved	r/w	r/w	Reserved	r/w	r/w	Reserved	Reserved
0		0	0	0	0	0	0	0	0	0	0
Offset: 0x00											
Bit 31 - 12: Reserved bits											
Bit 11 - 10: FORMAT Audio data format 0 - I2S Format 1 - Left Justified Format 2 - Right Justified Format Others - defaults to I2S Format											
Bit 9 - 8 : LENGTH Audio data bit length											

	0 – 16-bit Format
	1 – 20-bit Format
	2 – 24-bit Format
	3 – 32-bit Format
Bit 7 – 5	: Reserved bits
Bit 4	: I2S_BCLK_POL I2S_BCLK polarity 0b'0: No clock inversion 0b'1: Invert the polarity of I2S_BCLK
Bit 3	: I2S_LRCLK polarity for transmit 0b'0: No signal inversion 0b'1: Invert the polarity of I2S_LRCLK
Bit 2 – 0	: Reserved bits

Register Definition 120 – REG_I2S_CFG Definition

REG_I2S_CTL Definition						
31		4	3	2	1	0
	Reserved		r/w	Reserved	r/w	r/w
	0		0	0	0	0
Offset: 0x04						
Bit 31 – 12: Reserved bits						
Bit 3	: AUDIO_DISABLE 0b'0: Enable AUDIO_L/R pin output 0b'1: Disable AUDIO_L/R pin output					
Bit 2	: Reserved bit					
Bit 1	: TX_EN 0b'0: Disable transmit channel 0b'1: Enable transmit channel					
Bit 0	: SRST Set to trigger software reset. Reset is self-clearing. Read will return current software reset state. Software reset will not reset I2S configuration registers but will clear FIFOs, interrupts and reset state-machines.					

Register Definition 121 – REG_I2S_CTL Definition

REG_I2S_STAT Definition						
31		16	15			0
	Reserved					r/o
	0					0
Offset: 0x10						
Bit 31 – 16: Reserved bits						
Bit 15 – 0	: TXFIFO_CNT Indicates the number of data words in the transmit FIFO.					

Register Definition 122 – REG_I2S_STAT Definition

REG_I2S_PAD_CFG Definition		
31	10 9	0
	Reserved	r/w
	0	0
Offset: 0x14		
Bit 31 – 10: Reserved bits		
Bit 9 – 0 : PADDING		
Number of dummy bits to insert before shifting out audio data in Right-Justified format. This setting is ignored in other audio formats. 0x000: no padding required (bit length = number of I2S_BCLK cycles) 0x001: insert 1 extra bit ... 0x3FF: insert 1023 extra bits		

Register Definition 123 – REG_I2S_PAD_CFG Definition

3.10 Coprocessor Engine Registers

The base address is **0x7F00_6000**.

REG_CMD_DL Definition		
31	14 13	0
	Reserved	r/w
Offset: 0x154		Reset Value: 0x0
Bit 31 – 14: Reserved Bits		
Bit 13 – 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 when 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 16383 (sizeof(RAM_DL)-1).		

Register Definition 124 – REG_CMD_DL Definition

REG_CMD_WRITE Definition		
31	14 13	0
	Reserved	r/w
Offset: 0x150		Reset Value: 0x0
Bit 31 – 14: Reserved Bits		
Bit 13 – 0 : These bits are updated by the MCU to inform the coprocessor engine of the ending address of valid data feeding into its FIFO . Typically, the host will update this register after it has downloaded the coprocessor commands into its FIFO. The valid range is from 0 to 16383, i.e. within the size of the FIFO .		
Note: The FIFO size of the command buffer is 16384 bytes and each coprocessor instruction is 4 bytes in size. The value to be written into this register must be 4 bytes aligned.		

Register Definition 125 – REG_CMD_WRITE Definition

REG_CMD_READ Definition		
31	14 13	0
	Reserved	r/o
Offset: 0x14C		Reset Value: 0x0
Bit 31 – 14: Reserved Bits		
Bit 13 – 0 : These bits are updated by the coprocessor engine when 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 16383. In the case of an error, the coprocessor engine writes 0x3FFF 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 126 – REG_CMD_READ Definition

REG_CMDB_SPACE Definition			
31		14	13
Reserved		r/o	
Offset: 0x594		Reset Value: 0x3FFC	
Bit 31 – 14: Reserved Bits			
Bit 13 – 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 127 – REG_CMDB_SPACE Definition

REG_CMDB_WRITE Definition	
31	0
w/o	
Addr: 0x7F010000	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 bulk data transfer possible.	
Note: Programmers can use this register to write command or data to the coprocessor FIFO(RAM_CMD) . This register was introduced since FT810 series chip. Ensure that all writes to this register are 4 bytes aligned.	

Register Definition 128 – REG_CMDB_WRITE Definition

3.11 Miscellaneous Registers

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

The base address is **0x7F00_6000**.

REG_BOOT_CFG Definition													
31							8	7	6	5	4	3	0
r/o													
Offset: 0x628						Reset Value: 0x80							
Bit 31 – 8: Reserved Bits													
Bit 3 – 0: Reserved Bits													
Bit 7 : Enable DDR system initialization while booting up.													
Bit 6 : Enable Touch system initialization while booting up													
Bit 5 : Enable Audio system initialization while booting up.													
Bit 4 : Enable Watchdog system initialization while booting up.													
Note: This register is read-only and can only be updated through host commands by following a predefined process.													

Register Definition 129 – REG_BOOT_CFG Definition

REG_CPURESET Definition									
31						3	2	1	0
Reserved								r/w	
Offset: 0x88					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 0 means the engine is in working status.									

Register Definition 130 – REG_CPURESET Definition

REG_MACRO_1 Definition	
31	0
r/w	
Offset: 0x134 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 131 – REG_MACRO_1 Definition

REG_MACRO_0 Definition	
31	0
r/w	
Offset: 0x130 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 132 – REG_MACRO_0 Definition

REG_PWM_DUTY Definition			
31	0		
	8	7	
Reserved		r/w	
Offset: 0x12C 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 133 – REG_PWM_DUTY Definition

REG_PWM_HZ Definition			
31	0		
	14	13	
Reserved		r/w	
Offset: 0x128 Reset Value: 0xFA			
Bit 31 – 14: Reserved Bits			
Bit 13 – 0 : These bits define the backlight PWM output frequency in Hz . The default is 250Hz after reset. The valid frequency is from 250Hz to 10000Hz.			

Register Definition 134 – REG_PWM_HZ Definition

REG_INT_MASK Definition			
31	0		
	8	7	
Reserved		r/w	
Offset: 0x108 Reset Value: 0xFF			
Bit 31 – 8: Reserved Bits			
Bit 7	: INT_CONV_COMPLETE interrupt mask bit	Set 1 to enable touchscreen conversion complete interrupt	
Bit 6	: INT_CMDFLAG interrupt mask bit	Set 1 to enable command FIFO flag interrupt. Use with CMD_INTERRUPT	
Bit 5	: INT_CMDEEMPTY interrupt mask bit	Set 1 to enable command FIFO empty interrupt	
Bit 4	: INT_PLAYBACK interrupt mask bit	Set 1 to enable audio playback ended interrupt	
Bit 3	: INT_SOUND interrupt mask bit	Set 1 to enable sound effect ended interrupt	
Bit 2	: INT_TAG interrupt mask bit	Set 1 to enable touchscreen tag value REG_TOUCH_TAG change interrupt	
Bit 1	: INT_TOUCH interrupt mask bit	Set 1 to enable touchscreen touch detected interrupt	
Bit 0	: INT_SWAP interrupt mask bit	Set 1 to enable display list swap occurred interrupt	
Note: After reset, all the interrupt sources are eligible to trigger an interrupt by default.			

Register Definition 135 – REG_INT_MASK Definition

REG_INT_EN Definition	
31	1 0
Reserved	
Offset: 0x104	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.	

Register Definition 136 – REG_INT_EN Definition

REG_INT_FLAGS Definition	
31	8 7
Reserved	
r/o	
Offset: 0x100	Reset Value: 0x0
Bit 31 – 8: Reserved Bits	
Bit 7 : INT_CONV_COMPLETE interrupt flag 1 means touchscreen conversion complete	
Bit 6 : INT_CMDFLAG interrupt flag 1 means command FIFO flag	
Bit 5 : INT_CMDEEMPTY interrupt flag 1 means command FIFO empty	
Bit 4 : INT_PLAYBACK interrupt flag 1 means audio playback ended	
Bit 3 : INT_SOUND interrupt flag 1 means sound effect ended	
Bit 2 : INT_TAG interrupt flag 1 means touchscreen tag value REG_TOUCH_TAG change	
Bit 1 : INT_TOUCH interrupt flag 1 means touchscreen touch detected	
Bit 0 : INT_SWAP interrupt flag 1 means display list swap occurred	
Note: These bits are cleared automatically by reading. The host shall not write to this register.	

Register Definition 137 – REG_INT_FLAGS Definition

REG_GPIO_DIR Definition	
31	16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved	
Unused	
r/w	
Offset: 0xDC	Reset Value: 0x0
Bit 31 – 16: Reserved Bits	
Bit 15 – 9 : Unused	
Bit 8 : It controls the direction of GPIO8 .	
Bit 7 : It controls the direction of GPIO7 .	
Bit 6 : It controls the direction of GPIO6 .	
Bit 5 : It controls the direction of GPIO5 .	
Bit 4 : It controls the direction of GPIO4 .	
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. This register is a legacy register for backward compatibility only	

Register Definition 138 – REG_GPIO_DIR Definition

REG_GPIO Definition																	
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Unused				r/w					
Offset: 0xE0 Reset Value: 0x0																	
Bit 31 – 16: Reserved Bits																	
Bit 15 – 9 : Unused																	
Bit 8 : It controls the high or low level of pin GPIO8 .																	
Bit 7 : It controls the high or low level of pin GPIO7 .																	
Bit 6 : It controls the high or low level of pin GPIO6 .																	
Bit 5 : It controls the high or low level of pin GPIO5 .																	
Bit 4 : It controls the high or low level of pin GPIO4 .																	
Bit 3 : It controls the high or low level of pin GPIO3 .																	
Bit 2 : It controls the high or low level of pin GPIO2 . For audio amp enable/disable																	
Bit 1 : It controls the high or low level of pin GPIO1 . Pin mux with SPIS IO3																	
Bit 0 : It controls the high or low level of pin GPIO0 . Pin mux with SPIS IO2																	

Register Definition 139 – REG_GPIO Definition

REG_DISP Definition		
31	1	0
Reserved		r/w
Offset: 0xE4 Reset Value: 0x0		
Bit 31 – 1: Reserved bits		
Bit 0 : Set to 1 for DISP output.		

Register Definition 140 – REG_DISP Definition

REG_FREQUENCY Definition		
31	28	27
reserved		0
Offset: 0xC Reset Value: 0x44AA200		
Bit 31 – 28: Reserved.		
Bit 27 – 0 : The main clock frequency is 72MHz by default. The value is in Hz. If the host selects the alternative frequency, this register must be updated accordingly.		

Register Definition 141 – 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 72Mhz, it will wrap around after about 59 seconds.	

Register Definition 142 – 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 143 – 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 144 – REG_ID Definition

3.12 Auxiliary Registers

The registers listed here are supporting registers that add flexibility and functionality to those registers mentioned in section 3.1 to 3.11. These registers are located in **REG_CORE** (region 2).

The base address is **0x7F00_4000**.

REG_TRACKER Definition			
31	16	15	0
r/o		Reserved	
Offset: 0x00		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 145 – REG_TRACKER Definition

REG_TRACKER_1 Definition			
31	16	15	0
r/o		Reserved	
Offset: 0x04		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 146 – REG_TRACKER_1 Definition

REG_TRACKER_2 Definition			
31	16	15	0
r/o		Reserved	
Offset: 0x08		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 147 – REG_TRACKER_2 Definition

REG_TRACKER_3 Definition			
31		16 15	8 7
	r/o	Reserved	r/o
Offset: 0x0C		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 148 – REG_TRACKER_3 Definition

REG_TRACKER_4 Definition			
31		16 15	8 7
	r/o	Reserved	r/o
Offset: 0x10		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 149 – REG_TRACKER_4 Definition

REG_MEDIAFIFO_READ Definition	
31	0
r/o	
Offset: 0x14	
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 150 – REG_MEDIAFIFO_READ Definition

REG_MEDIAFIFO_WRITE Definition	
31	0
w/o	
Offset: 0x18	
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 151 – REG_MEDIAFIFO_WRITE Definition

REG_ANIM_ACTIVE Definition	
31	0
r/o	
Offset: 0x2C	
Reset Value: 0x0	
Bit 31 – 0: 32-bit mask of currently playing animations. Each bit indicates the active state of an animation channel. 0 means animation ends 1 means animation runs	
Note: Only applicable for the animation channel is played with ANIM_ONCE flag.	

Register Definition 152– REG_ANIM_ACTIVE Definition

REG_OBJECT_COMPLETE Definition	
31	0
r/w	
Offset: 0x38	Reset Value: 0x0
Bit 31 – 0: Valid when OPT_COMPLETEREG is set to 1. The application should set the register REG_OBJECT_COMPLETE to 1 at the end of the video data. This prevents a hang with truncated video data in the media FIFO, enabling the EVE to detect this condition and raise an exception.	

Register Definition 153 – REG_OBJECT_COMPLETE Definition

REG_EXTENT_X0 Definition	
31	0
r/o	
Offset: 0x3C	Reset Value: -
Bit 31 – 0: Store the minimum x-coordinate of the current screen extents of the rendered widget in pixels	

Register Definition 154 – REG_EXTENTS_X0 Definition

REG_EXTENT_Y0 Definition	
31	0
r/o	
Offset: 0x40	Reset Value: -
Bit 31 – 0: Store the minimum y-coordinate of the current screen extents of the rendered widget in pixels	

Register Definition 155 – REG_EXTENTS_Y0 Definition

REG_EXTENT_X1 Definition	
31	0
r/o	
Offset: 0x44	Reset Value: -
Bit 31 – 0: Store the maximum x-coordinate of the current screen extents of the rendered widget in pixels	

Register Definition 156 – REG_EXTENTS_X1 Definition

REG_EXTENT_Y1 Definition	
31	0
r/o	
Offset: 0x48	Reset Value: -
Bit 31 – 0: Store the maximum y-coordinate of the current screen extents of the rendered widget in pixels	

Register Definition 157 – REG_EXTENTS_Y1 Definition

REG_PLAY_CONTROL Definition	
31	0
r/w	
Offset: 0x50	Reset Value: 0x1
Bit 31 – 0: Video playback control. The following values are defined: -1: exit playback 0: pause playback 1: play normally	

Register Definition 158 – REG_PLAY_CONTROL Definition

4 Display List Commands

The display list **RAM(RAM_DL)** is filled with display list commands, starting at offset zero. Each command sets graphics state, performs a drawing action, or controls execution flow within the display list. When register **REG_DLSWAP** is set to validate the display list, the render engine will fetch and execute the content to output the pixel data in the **DDR** memory specified by register **REG_RE_DEST**.

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 are changed.

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

Parameters	Default values	Commands
func & ref	ALWAYS, 0	ALPHA_FUNC
func, ref & mask	ALWAYS, 0, 255	STENCIL_FUNC
src & dst	SRC_ALPHA, ONE_MINUS_SRC_ALPHA	BLEND_FUNC
cell	0	CELL
alpha	255	COLOR_A
red, green & blue	(255,255,255)	COLOR_RGB
width:Line width in 1/16 pixels	16	LINE_WIDTH
size:Point size in 1/16 pixels	16	POINT_SIZE
width & height	HSIZE,2048	SCISSOR_SIZE
x & y:Starting coordinates of scissor	(x, y) = (0,0)	SCISSOR_XY
handle:Current bitmap handle	0	BITMAP_HANDLE
Bitmap transform coefficients	0,0,0,0,0,0	BITMAP_TRANSFORM_A-F
s:Stencil clear value	0	CLEAR_STENCIL
s:Tag clear value	0	CLEAR_TAG
mask:Mask value of stencil	255	STENCIL_MASK
spass & sfail	KEEP,KEEP	STENCIL_OP
s:Tag buffer value	255	TAG
mask:Tag mask value	1	TAG_MASK
alpha:Alpha clear value	0	CLEAR_COLOR_A
red, green & blue	(0,0,0)	CLEAR_COLOR_RGB
addr:Palette source address	RAM_G	PALETTE_SOURCE, PALETTE_SOURCEH
frac:Units of pixel precision	4:1/16 pixel	VERTEX_FORMAT, VERTEX2F

Table 21 – Graphics Context

4.2 Command Encoding

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

4.3 Command Groups

4.3.1 Setting Graphics State

ALPHA_FUNC	set the alpha test function
BEGIN	start drawing a graphics primitive
BITMAP_EXT_FORMAT	specify the extended format of the bitmap
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 BITMAP_SOURCEH	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_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
END	finish drawing a graphics primitive
LINE_WIDTH	set the line width
POINT_SIZE	set point size
REGION	Branch if Y is outside region
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/ PALETTE_SOURCEH	specify the base address of the palette
BITMAP_ZORDER	set the z-order address pattern for a bitmap

4.3.2 Drawing Actions

CLEAR	clear buffers to preset values
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
REGION	branch if Y is outside region

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 22 – 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:



```

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

Graphics context

None

See also

[END](#)

4.6 BITMAP_EXT_FORMAT

Specify the extended format of the bitmap.

Encoding

31	24	23	16	15	0
0x2E	reserved	reserved	reserved	format	format

Parameters

format

Bitmap pixel format.

Description

If **BITMAP_LAYOUT** specifies a format for **GLFORMAT**, then the format is taken from **BITMAP_EXT_FORMAT** instead.

Valid values for the field format are:

Format Name	Value	Bits per Pixel
ARGB1555	0	16
L1	1	1
L4	2	4
L8	3	8
RGB332	4	8
ARGB2	5	8
ARGB4	6	16
RGB565	7	16
BARGRAPH	11	8
L2	17	2
RGB8	19	24
ARGB8	20	32
PALETEDARGB8	21	8
RGB6	22	18
ARGB6	23	24
LA1	24	2
LA2	25	4
LA4	26	8
LA8	27	16
YCBCR	28	8
COMPRESSED_RGBA_ASTC_4x4_KHR	37808	8.00
COMPRESSED_RGBA_ASTC_5x4_KHR	37809	6.40
COMPRESSED_RGBA_ASTC_5x5_KHR	37810	5.12
COMPRESSED_RGBA_ASTC_6x5_KHR	37811	4.27
COMPRESSED_RGBA_ASTC_6x6_KHR	37812	3.56
COMPRESSED_RGBA_ASTC_8x5_KHR	37813	3.20

COMPRESSED_RGBA_ASTC_8x6_KHR	37814	2.67
COMPRESSED_RGBA_ASTC_8x8_KHR	37815	2.00
COMPRESSED_RGBA_ASTC_10x5_KHR	37816	2.56
COMPRESSED_RGBA_ASTC_10x6_KHR	37817	2.13
COMPRESSED_RGBA_ASTC_10x8_KHR	37818	1.60
COMPRESSED_RGBA_ASTC_10x10_KHR	37819	1.28
COMPRESSED_RGBA_ASTC_12x10_KHR	37820	1.07
COMPRESSED_RGBA_ASTC_12x12_KHR	37821	0.89

Table 23 – Bitmap formats and bits per pixel

Graphics context

None

See also

[BITMAP_LAYOUT](#)

4.7 BITMAP_HANDLE

Specify the bitmap handle.

Encoding

31	24	23	65	0
0x05		reserved		handle

Parameters

handle

Bitmap handle. The initial value is 0. The valid value range is from 0 to 63.

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](#)

4.8 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 27 and defined as per the table below.

Format	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
BARGRAPH	11	-	-	-	-	-
L2	17	2	2	0	0	0
RGB8	19	24	0	8	8	8
ARGB8	20	32	8	8	8	8
PALETTERGB8	21	8	8	8	8	8
RGB6	22	18	0	6	6	6
ARGB6	23	24	6	6	6	6
LA1	24	2	1		1	
LA2	25	4	2		2	
LA4	26	8	4		4	
LA8	27	16	8		8	
YCBCR	28	8	-	-	-	-
GLFORMAT	31	Check BITMAP_EXT_FORMAT				

Table 24 – BITMAP_LAYOUT Format List

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 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.

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

Normally, it can be calculated with the following formula:

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

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

The table below lists the bitmap formats that may require padding to align pixels with memory for each line of bitmap pixels.

Format	Linstride	Image Width (pixel)
ARGB1555	2	-
L1	1	8 pixel aligned

L4	1	2 pixel aligned
L8	1	-
RGB332	1	-
ARGB2	1	-
ARGB4	2	-
RGB565	2	-
BARGRAPH	-	-
L2	1	4 pixel aligned
RGB8	3	2 pixel aligned
ARGB8	4	-
PALETTERDARGB8	1	-
RGB6	3	8 pixel aligned
ARGB6	3	2 pixel aligned
LA1	1	4 pixel aligned
LA2	1	2 pixel aligned
LA4	1	-
LA8	2	-

Table 25 – BITMAP_LAYOUT Format in pixel aligned

The bitmap formats listed on the table above are in width pixel aligned. The last two bitmap formats, **YCBCR** and **ASTC**, are trickier as they are arranged in blocks.

YCBCR pixels are encoded in 2 x 2 blocks, each quad taking 32 bits of memory. So, **YCBCR** width must be 2 pixel aligned and height must also be 2 pixel aligned. Refer to section 7: [YCbCr format](#) for more information.

ASTC blocks are represented between 4 x 4 to 12 x 12 blocks, each block taking 128 bits (16 bytes) of memory regardless of the resolution of the block (e.g. 4 x 4 or 6 x 6 or 12 x 12). So, **ASTC** must be 16-byte aligned. Refer to section 6: [ASTC](#) for more information.

Padding can sometimes cause visible artifacts or distortions in the rendered bitmap. Here is an example to ensure that padded pixels are not rendered as part of the display for a 181x185 YCBCR bitmap.

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd(BITMAP_HANDLE(0));
cmd_setbitmap(0, YCBCR, 182, 186);
cmd(BITMAP_SIZE(NEAREST, BORDER, BORDER, 181, 185)); // display the
// original size
// only

cmd(BITMAP_SIZE_H(0, 0));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2F(3232, 2720));
```

height - Bitmap height, in lines

Description

For more details about memory layout according to pixel format, refer to the tables 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							
Bit 7	6	Bit 5	4	Bit 3	2	Bit 1	0
Pixel 0		Pixel 1		Pixel 2		Pixel 3	

L4 Format			
7	4	3	0
Pixel 0		Pixel 1	

L8 Format	
7	0
Pixel 0	

Table 26 – L1/L2/L4/L8 Pixel Format

ARGB2 Format			
7	6	5	0
Alpha Channel	Red Channel	Green Channel	Blue Channel

RGB332 Format		
7	5	0
Red Channel	Green Channel	Blue Channel

Table 27 – ARGB2/RGB332 Pixel Format

RGB565 Format		
15	11	0
Red Channel	Green Channel	Blue Channel

Table 28 – RGB565 Pixel Format

ARGB1555 Format			
15	14	10	0
Alpha Channel	Red Channel	Green Channel	Blue Channel

ARGB4 Format			
15	12	11	0
Alpha Channel	Red Channel	Green Channel	Blue Channel

Table 29 – ARGB1555/ARGB4 Pixel Format

RGB6 Format		
17	12	0
Red Channel	Green Channel	Blue Channel

Table 30 – RGB6 Pixel Format

ARGB6 Format			
23	18	17	0
Alpha Channel	Red Channel	Green Channel	Blue Channel

Table 31 – ARGB6 Pixel Format

RGB8 Format		
23	16	0
Red Channel	Green Channel	Blue Channel

Table 32 – RGB8 Pixel Format

ARGB8 Format			
31	24	23	0
Alpha Channel	Red Channel	Green Channel	Blue Channel

Table 33 – ARGB8 Pixel Format

LA1 Format							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Alpha Channel	Pixel 0	Alpha Channel	Pixel 1	Alpha Channel	Pixel 2	Alpha Channel	Pixel 3

LA2 Format							
Bit 7	6	Bit 5	4	Bit 3	2	Bit 1	0
Alpha Channel		Pixel 0		Alpha Channel		Pixel 1	

LA4 Format				
7	4		3	0
Alpha Channel			Pixel 0	

LA8 Format				
15	8		7	0
Alpha Channel			Pixel 0	

Table 34 – LA1/LA2/LA4/LA8 Pixel Format

PALETTERDARGB8 Format							
31	24	23	16	15	8	7	0
Alpha Channel		Red Channel		Green Channel		Blue Channel	

Table 35 – PALETTERDARGB8 Pixel Format

Note: PALETTERDARGB8 are 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

See also

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

4.9 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](#)

4.10 BITMAP_SIZE

Specify the screen drawing of bitmaps for the current handle.

Encoding

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

Parameters

filter

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

wrapx

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

wrapy

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

width

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

height

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

Description

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

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

4.11 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](#)

4.12 BITMAP_SOURCE

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

Encoding

31	24	23	0
0x01		addr	

Parameters

addr

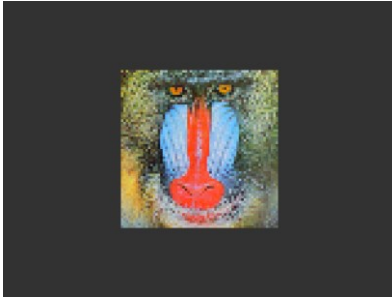
Bitmap address in **RAM_G**, pixel-aligned with respect to the bitmap format.
 For example, if the bitmap format is **RGB565/ARGB4/ARGB1555**, the bitmap source shall be aligned to 2 bytes.

Description

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

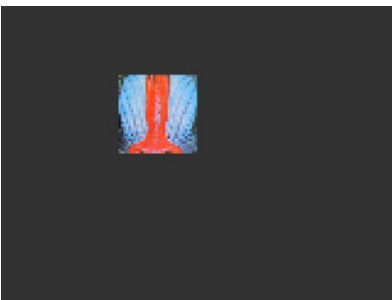
Examples

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



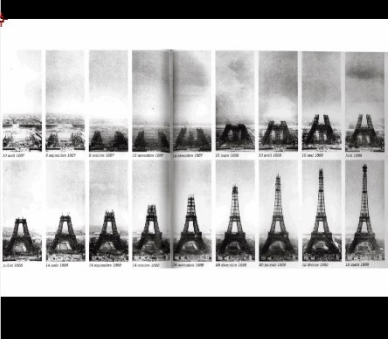
```
d1( BITMAP_SOURCE(0) );
d1( BITMAP_LAYOUT( RGB565, 128, 64 ) );
d1( BITMAP_SIZE( NEAREST, BORDER, BORDER,
64, 64 ) );
d1( BEGIN( BITMAPS ) );
d1( VERTEX2II( 48, 28, 0, 0 ) );
```

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



```
d1( BITMAP_SOURCE( 128 * 16 + 32 ) );
d1( BITMAP_LAYOUT( RGB565, 128, 64 ) );
d1( BITMAP_SIZE( NEAREST, BORDER, BORDER, 32,
32 ) );
d1( BEGIN( BITMAPS ) );
d1( VERTEX2II( 48, 28, 0, 0 ) );
```

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



```
d1( BITMAP_HANDLE( 0 ) );
d1( BITMAP_SOURCE( 0 ) );
d1( BITMAP_SIZE_H( 1, 0 ) );
d1( BITMAP_SIZE( NEAREST, BORDER, BORDER, 288,
480 ) );
d1( BITMAP_LAYOUT_H( 1, 0 ) );
d1( BITMAP_LAYOUT( ARGB1555, 576, 480 ) );
d1( BEGIN( BITMAPS ) );
d1( VERTEX2II( 76, 25, 0, 0 ) );
d1( END() );
```

Graphics context

None

See also

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

4.13 BITMAP_SOURCEH

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

Encoding

31	24 23	8 7	0
0x31	reserved	addr	

Parameters

addr
 Bitmap start address high part.

Description

The bitmap source address specifies the address of the bitmap graphic data. The address consists of 8 (high part) + 24 (low part) = 32 bits.

Graphics context

None

See also

None

4.14 BITMAP_SWIZZLE

Set the source for the red, green, blue and alpha channels of a bitmap.

Encoding

31	24	23	12	11	9	8	6	5	3	2	0
0x2f	reserved					r	g	b	a		

Parameters

r
 red component source channel

g
 green component source channel

b
 blue component source channel

a
 alpha component source channel

Description

Bitmap swizzle allows the channels of the bitmap to be exchanged or copied into the final color channels. Each final color component can be sourced from any of the bitmap color components or can be set to zero or one. Valid values for each source are:

Name	Value	Description
ZERO	0	Set the source channel to zero
ONE	1	Set the source channel to 1
RED	2	Specify RED component as source channel
GREEN	3	Specify GREEN component as source channel
BLUE	4	Specify BLUE component as source channel
ALPHA	5	Specify ALPHA component as source channel

Bitmap swizzle is only applied when the format parameter of **BITMAP_LAYOUT** is specified as **GLFORMAT**. Otherwise, the four components are in their default order. The default swizzle is (RED, GREEN, BLUE, ALPHA)

Note: Please refer to OpenGL API specification for more details

Examples

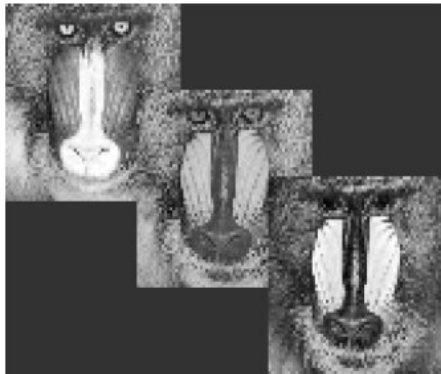
Bitmap drawn with default swizzle, and with red/blue exchanged:



```

d1 (BITMAP_SOURCE (0));
d1 (BITMAP_LAYOUT (GLFORMAT, 128, 64));
d1 (BITMAP_EXT_FORMAT (RGB565));
d1 (BITMAP_SIZE (NEAREST, BORDER, BORDER, 64, 64));
d1 (BEGIN (BITMAPS));
d1 (BITMAP_SWIZZLE (RED, GREEN, BLUE, ALPHA));
d1 (VERTEX2II (8, 28, 0, 0));
d1 (BITMAP_SWIZZLE (BLUE, GREEN, RED, ALPHA));
d1 (VERTEX2II (88, 28, 0, 0));
  
```

Red, green, and blue channels extracted to create three grayscale images:



```

d1 (BITMAP_LAYOUT (GLFORMAT, 128, 64));
d1 (BITMAP_EXT_FORMAT (RGB565));
d1 (BEGIN (BITMAPS));
d1 (BITMAP_SWIZZLE (RED, RED, RED, ALPHA));
d1 (VERTEX2II (0, 0, 0, 0));
d1 (BITMAP_SWIZZLE (GREEN, GREEN, GREEN, ALPHA));
d1 (VERTEX2II (48, 28, 0, 0));
d1 (BITMAP_SWIZZLE (BLUE, BLUE, BLUE, ALPHA));
d1 (VERTEX2II (96, 56, 0, 0));
  
```

Graphics Context

None

See also

None

4.15 BITMAP_TRANSFORM_A

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

Encoding

31		24	23		18	17	16		0
0x15			reserved			p	v		

Parameters

p

Precision control: 0 is 8.8, 1 is 1.15. The initial value is 0.

v

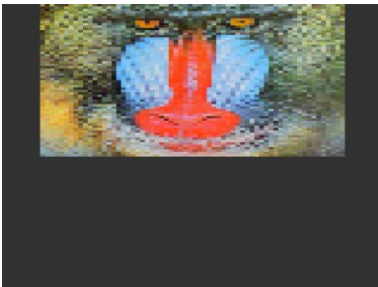
A component of the bitmap transform matrix, in signed 8.8 or 1.15 fixed point form. The initial value is 256.

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

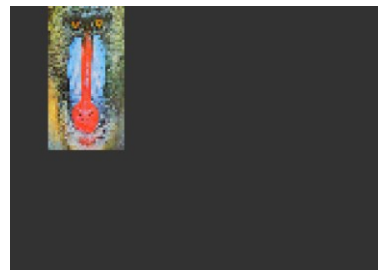
A value of 0.5 (128) causes the bitmap to appear double width:



```

d1 ( BITMAP_SOURCE(0) );
d1 ( BITMAP_LAYOUT (RGB565, 128, 64) );
d1 ( BITMAP_TRANSFORM_A(128) );
d1 (BITMAP_SIZE (NEAREST, BORDER, BORDER, 128,
128) );
d1 ( BEGIN (BITMAPS) );
d1 ( VERTEX2II (16, 0, 0, 0) );
  
```

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



```

d1 ( BITMAP_SOURCE(0) );
d1 ( BITMAP_LAYOUT (RGB565, 128, 64) );
d1 ( BITMAP_TRANSFORM_A(512) );
d1 (BITMAP_SIZE (NEAREST, BORDER, BORDER, 128,
128) );
d1 ( BEGIN (BITMAPS) );
d1 ( VERTEX2II (16, 0, 0, 0) );
  
```

Graphics Context

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

See also

None

4.16 BITMAP_TRANSFORM_B

Specify the *b* coefficient of the bitmap transform matrix

Encoding

31	24	23	18	17	16	0
0x16	reserved	p	v			

Parameters

p

Precision control: 0 is 8.8, 1 is 1.15. The initial value is 0.

v

The component of the bitmap transform matrix, in signed 8.8 or 1.15 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 *p*, *v* is part of the graphics context, as described in section 4.1.

See also

None

4.17 BITMAP_TRANSFORM_C

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

Encoding

31	24	23	0
0x17	c		

Parameters

c

The *c* 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 transform functionality.

Graphics context

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

See also

None

4.18 BITMAP_TRANSFORM_D

Specify the *d* coefficient of the bitmap transform matrix

Encoding

31		24	23		18	17	16		0
0x18			reserved			p	v		

Parameters

p

Precision control: 0 is 8.8, 1 is 1.15. The initial value is 0.

v

The *d* component of the bitmap transform matrix, in signed 8.8 or 1.15 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 *p*, *v* of the graphics context, as described in section 4.1.

See also

None

4.19 BITMAP_TRANSFORM_E

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

Encoding

31		24	23		18	17	16		0
0x19			reserved			p	v		

Parameters

p

Precision control: 0 is 8.8, 1 is 1.15. The initial value is 0.

v

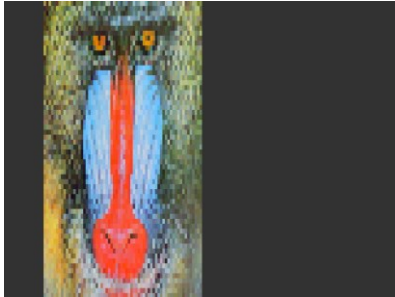
The *e* component of the bitmap transform matrix, in signed 8.8 or 1.15 fixed point form. The initial value is 256.

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.

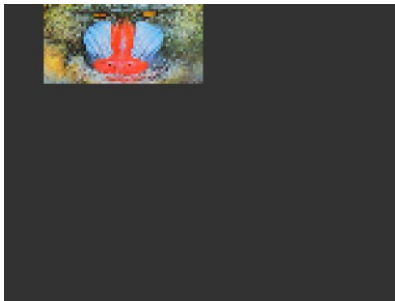
Examples

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 p and v of the graphics context, as described in section 4.1.

See also

None

4.20 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

4.21 BITMAP_ZORDER

Set the z-order address pattern for a bitmap

Encoding

31	24	23	8	7	0
0x33	reserved			o	

Parameters

o
z-order code.

Description

BITMAP_ZORDER set the z-order address pattern for a bitmap.

Graphics context

None

See also

None

4.22 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 36 – BLEND_FUNC Constant Value Definition

Description

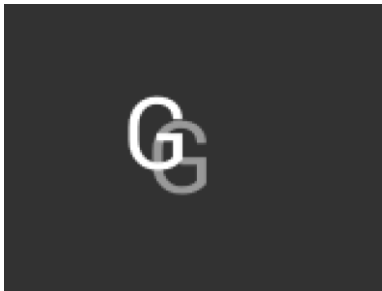
The blend function controls how new color values are combined with the values already in the color buffer. Given a pixel value source and a previous value in the color buffer destination, the computed color is:

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

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:



```

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

```

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

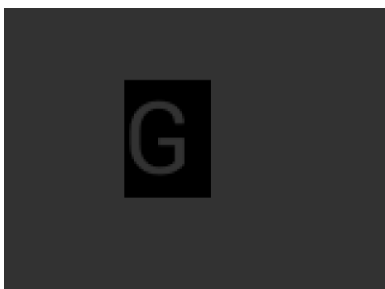


```

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

```

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



```

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

```

Graphics context

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

See also

[COLOR_A](#)

4.23 CALL

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

Encoding

31	24	23	16	15	0
0x1D	reserved	reserved	reserved	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 4095(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](#)

4.24 CELL

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

Encoding

31	24	23	7	6	0
0x06	reserved	reserved	reserved	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

4.25 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 the 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 the command CLEAR_TAG.

Description

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

Examples

To clear the screen to bright blue:



```
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](#)

4.26 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](#)

4.27 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

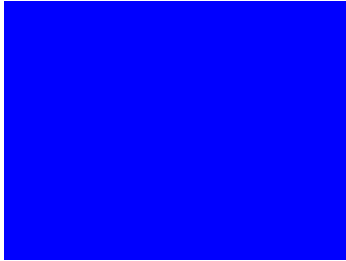
Blue value used when the color buffer is cleared. The initial value is 0.

Description

Sets the color values to be used follow by **CLEAR**.

Examples

To clear the screen to bright blue:



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

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

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](#)

4.28 CLEAR_STENCIL

Specify clear value for the stencil buffer

Encoding

31	24	23	8	7	0
0x11	reserved	reserved	reserved	s	0

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](#)

4.29 CLEAR_TAG

Specify clear value for the tag buffer

Encoding

31	24	23	0
0x12	s		

Parameters

s
 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](#)

4.30 COLOR_A

Set the current color alpha

Encoding

31	24	23	87	0
0x10	reserved			alpha

Parameters

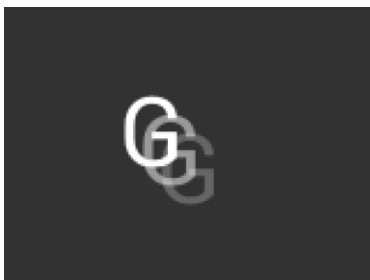
alpha
 Alpha for the current color. The initial value is 255

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:



```
d1 ( BEGIN (BITMAPS) );
d1 ( VERTEX2II (50, 30, 31, 0x47) );
d1 ( COLOR_A ( 128 ) );
d1 ( VERTEX2II (58, 38, 31, 0x47) );
d1 ( COLOR_A ( 64 ) );
d1 ( 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](#)

4.31 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.

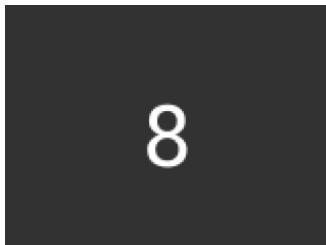
- a**
Enable or disable the alpha channel update of the color buffer. The initial value is 1 and means enable.

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:



```
d1( BEGIN(BITMAPS) );
d1( VERTEX2II(68, 40, 31, 0x38) );
d1( POINT_SIZE(40 * 16) );
d1( COLOR_MASK(0, 0, 0, 0) );
d1( BEGIN(POINTS) );
d1( TAG( 0x38 ) );
d1( 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](#)

4.32 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:



```
d1( BEGIN(BITMAPS) );
d1( VERTEX2II(50, 38, 31, 0x47) );
d1( COLOR_RGB( 255, 100, 50 ) );
d1( VERTEX2II(80, 38, 31, 0x47) );
d1( COLOR_RGB( 50, 100, 255 ) );
d1( 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](#)

4.33 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

4.34 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** to save space for extra graphics instructions in **RAM_DL**.

Graphics context

None

See also

[BEGIN](#)

4.35 JUMP

Execute commands at another location in the display list.

Encoding

31	24	23	16	15	0
0x1E	reserved			dest	

Parameters

dest

Display list address (offset from RAM_DL) to be jumped. The valid range is from 0 to 4095(sizeof(**RAM_DL**)/4-1).

Graphics context

None

See also

[CALL](#)

4.36 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 the LINE_WIDTH command will affect the LINES, LINE_STRIP, RECTS, EDGE_STRIP_A/B/R/L 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 44.1.

See also

None

4.37 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

4.38 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

4.39 PALETTE_SOURCE

Specify the base address of the palette.

Encoding

31	24	23	0
0x2A		addr	

Parameters

addr

Address of palette in **RAM_G**, 2-byte alignment is required. The initial value is **0**.

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

4.40 PALETTE_SOURCEH

Specify high part of the base address of the palette.

Encoding

31	24	23	8	7	0
0x32		reserved		addr	

Parameters

addr

Palette address high part. The initial value is **0**.

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

4.41 POINT_SIZE

Specify the radius of points

Encoding

31	24	23	13	12	0
0x0D		reserved		size	

Parameters

size

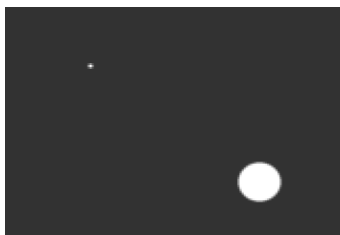
Point radius in 1/16 pixel precision. The initial value is 16. The valid range is from zero to 8191, i.e. from 0 to 511 pixels.

Description

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

Examples

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



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

Graphics context

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

See also

None

4.42 REGION

Branch if Y is outside region

Encoding

31	24 23	18	17	12	11	0
0x34	y		h			dest

Parameters

- y**
Region Y start
- h**
Region height
- dest**
Display list address

Description

For the currently drawing line, checks if it is inside or outside the region as described below. If the line is inside the region, acts like SAVE_CONTEXT. Otherwise, the line is outside the region, and action is the same as JUMP, transferring control to dest.

Region evaluation first computes y0 and y1 as follows:

$$y0 = \lfloor y \times \text{REG_RE_H} / 64 \rfloor$$

$$y1 = \lfloor (y+h) \times \text{REG_RE_H} / 64 \rfloor$$

and then considers lines that are $y0 \leq y < y1$ to be inside the region.

Note that if h is 0 then the behavior is an unconditional jump, like JUMP.

Note also that it is not possible to specify a region that covers the entire screen. In this case **SAVE_CONTEXT** should be used instead.

Examples

Refer to [REGION Instruction](#)

Graphics context

None

See also

[JUMP](#)

4.43 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:



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

Graphics context

None

See also

[SAVE_CONTEXT](#)

4.44 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](#)

4.45 SAVE_CONTEXT

Push the current graphics context on the context stack.

Encoding

31	24	23	0
0x22	reserved		

Parameters

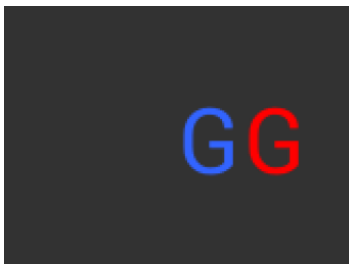
None

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:



```

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

[RESTORE_CONTEXT](#)

4.46 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 in, as described in section 4.1.

See also

None

4.47 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, as described in section 4.1.

See also

None

4.48 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](#)

4.49 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](#)

4.50 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

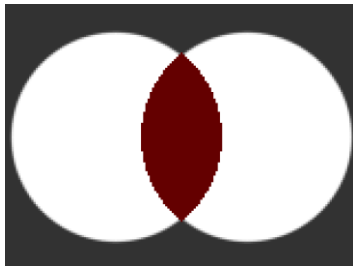
Table 37 – STENCIL_OP Constants Definition

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](#)

4.51 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	0
0x03			s

Parameters

s

Tag value. Valid value range is from 1 to 16,777,215. The initial value is 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.

Note: The valid tag value range for graphics objects is 1-16,777,215 but only tag value 1-255 can be used by **CMD_TRACK**.

Graphics context

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

See also

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

4.52 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](#)

4.53 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](#)

4.54 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](#)

4.55 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	<i>1 pixel</i>	-16384 to 16383
1	<i>½ pixel</i>	-8192 to 8191
2	<i>¼ pixel</i>	-4096 to 4095
3	<i>1/8 pixel</i>	-2048 to 2047
4	<i>1/16 pixel</i>	-1024 to 1023

Table 38 – 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](#)

4.56 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

4.57 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.

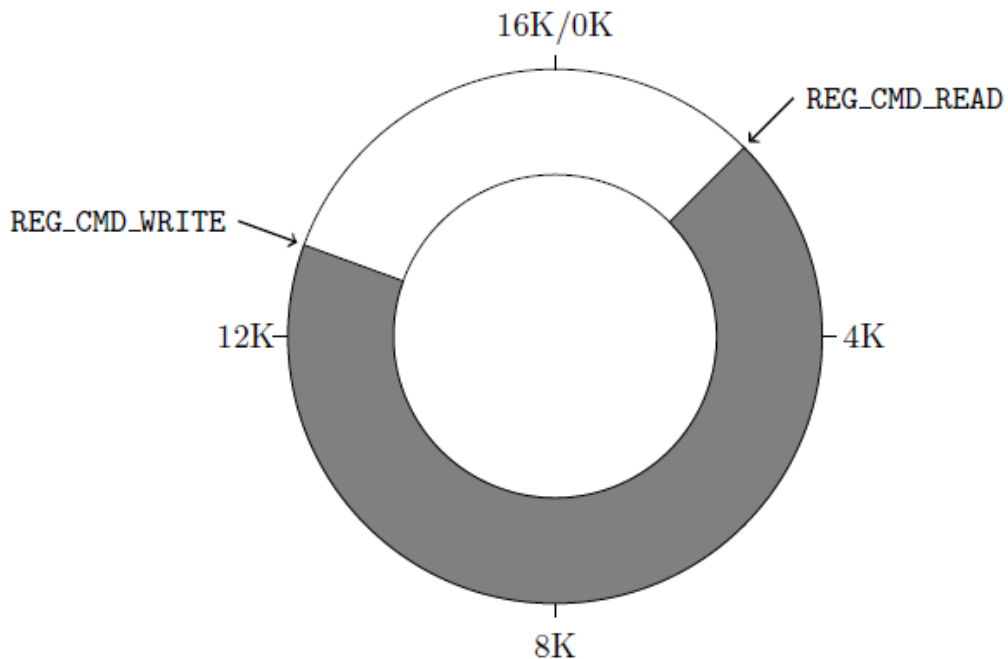
Graphics context

The value of y is part of the graphics context.

5 Coprocessor Engine

5.1 Command FIFO

The coprocessor engine is fed via a 16K byte **FIFO** called **RAM_CMD**. The MCU writes coprocessor commands or display list commands into the **FIFO**, and the coprocessor engine reads and executes the commands. The MCU 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 MCU can apply the following formula:

$$\begin{aligned} \text{fullness} &= (\text{REG_CMD_WRITE} - \text{REG_CMD_READ}) \text{ mod } 16384 \\ \text{free space} &= (16384 - 4) - \text{fullness}; \end{aligned}$$

This calculation does not report 16384 bytes of free space, to prevent completely wrapping the circular buffer and making it appear empty.

If enough space is available in the FIFO, the MCU writes the commands at the appropriate location in the **FIFO**, and then updates **REG_CMD_WRITE**. To simplify the MCU code, **EVE** automatically wraps continuous writes from the top address (**RAM_CMD** + 16383) 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 must be a multiple of 4 bytes: the length depends on the command itself, and any appended data. Some commands are followed by variable-length data, so the total length may not be a multiple of 4 bytes. Padding 0 to the variable-length data is required to make total length a multiple of 4 bytes. This is to prevent coprocessor engine from ignoring the extra 1, 2 or 3 bytes and continue reading the next command at the following 4 bytes boundary.

To offload work from the MCU 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 MCU 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 transferred is less than the value in the register **REG_CMDB_SPACE**, the MCU is able to safely write all the data to **REG_CMDB_WRITE** in one write transfer. 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 16380 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.

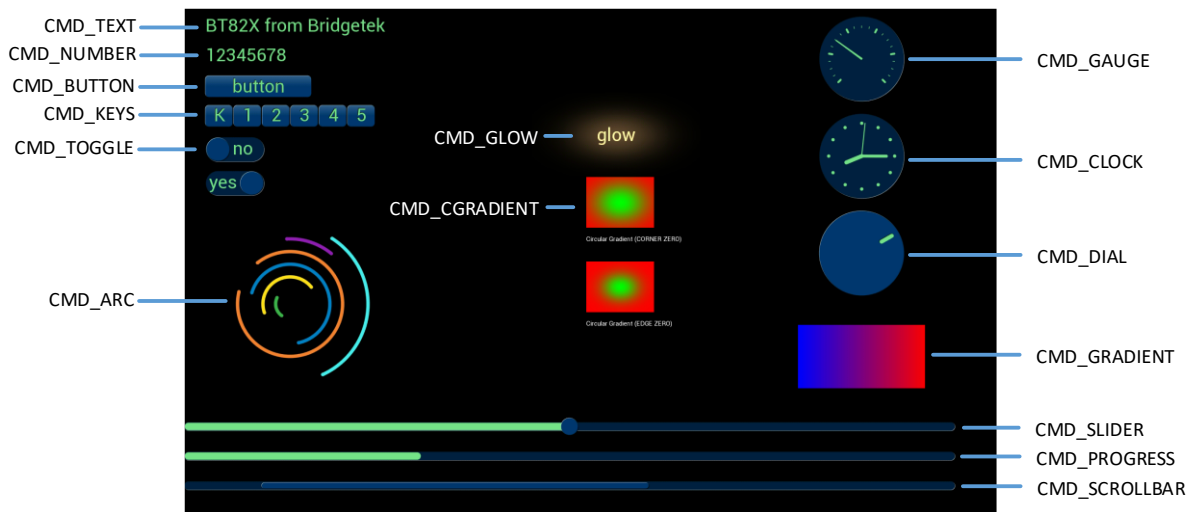


Figure 3 – Widget List

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).

$$Radius = font\ height * 3 / 16$$

- All 3D shadows are drawn with:
 - Highlight offsets 0.5 pixels above and left of the object
 - 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 39 – 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
```


5.4 Synchronization between MCU & 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 MCU can poll **REG_CMD_READ** until it is equal to **REG_CMD_WRITE**.

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

5.5 ROM and RAM Fonts

Fonts in **EVE** are treated as a set of bitmap-graphics with metrics block indexed by handles from 0 to 63. The following commands are using fonts:

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

The BT82X series defines three types of metrics blocks: legacy, extended format 1 and extended format 2. Extended format 2 is a new addition introduced in the fifth-generation **EVE**, the BT82X. This format supports the **UTF-8** encoding scheme for **Unicode** characters and includes support for **Kerning** features.

5.5.1 Legacy Font Metrics Block

This legacy font metrics block, introduced in any **EVE** series ICs **prior to** BT81X Series, continues to be used in BT82X series. There are a total of 10 legacy fonts that range from font 16 to font 25.

In each legacy font, there is one 148-bytes font metrics block associated with it and support up to 128 characters.

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

Address	Size	Value	Description
p + 0	128	width	width of each font character, in pixels
p + 128	2	format	bitmap format as defined in BITMAP_EXT_FORMAT , except BARGRAPH and PALETTED formats.
p + 130	2	font flags	BIN (bit 0), CJK (bit 1)
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	gptr	pointer to glyph data in memory

Table 40 – Legacy Font Metrics Block

In BT82X series, there are 64 bitmap handles and they are stored at the top of the **DDR RAM** in **RAM_G**. Legacy font 16 to 25 are attached to bitmap handles 16 to 25. Extended format 2 font 26 to 34 are attached to bitmap handles 26 to 34. For a 128 MByte **DDR RAM**, font 16 will be attached at 0x7FFF_FF40.

Example: Find the width of character for legacy font

To find the width of character 'g' (ASCII 0x67) in ROM font 24:

handle = DDR RAM size - ((64 - 24) * 4)

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

widths = *p*

read byte from memory at widths[0x67]

5.5.2 Extended Format 1 Font Metrics Block

From BT81X series onwards, extended format 1 font metrics block is introduced. This font metrics block supports a full range of **Unicode** characters with **UTF-8** coding points (note: CMD_KEYS command does not support **Unicode** characters).

This font metrics block is variable-sized, depending on the number of characters.

Address	Size	Value	Description
p + 0	4	signature	Must be 0x0100AAFF
p + 4	4	size	Total size of the font block, in bytes
p + 8	2	format	Bitmap format, as defined in BITMAP_EXT_FORMAT , except BarGraph and Paletted formats.
p + 10	2	flags	BIN (bit 0), CJK (bit 1)
P + 12	4	swizzle	Bitmap swizzle value, see BITMAP_SWIZZLE
p + 16	4	layout_width	Font bitmap line stride, in bytes
p + 20	4	layout_height	Font bitmap height, in pixels
p + 24	4	pixel_width	Font screen width, in pixels
p + 28	4	pixel_height	Font screen height, in pixels
p + 32	4	start_of_Graphic_data	Pointer to font graphic data in memory, including flash.
P + 36	4	number_of_characters	Total number of characters in font: <i>N</i> (multiple of 128)
p + 40	4 x [N / 128]	gptr	Offsets to glyph data
p + 40 + 4 x [N / 128]	4 x [N / 128]	wptr	Offsets to width data
p + 40 + 8 x [N / 128]	<i>N</i>	width_data	Width data, one byte per character

Table 41 – Extended Font Metrics Block

The table gptr contains offsets to graphic data. There is one offset for every 128 code points. The offsets are all relative to the start_of_graphic_data. The start_of_graphic_data may be an address in **RAM_G**. Similarly, the table wptr contains offsets to width data, but the offsets are relative to p, the start of the font block itself. So, to find the bitmap address and width of a code point *cp*, please refer to the pseudo-code below:

```
struct xfont {
    uint32_t signature,
    uint32_t size,
    uint32_t format,
    uint32_t swizzle,
    uint32_t layout_width,
    uint32_t layout_height,
    uint32_t pixel_width,
    uint32_t pixel_height,
    uint32_t start_of_graphic_data;
    uint32_t number_of_characters;
    uint32_t gptr[N/128];
    uint32_t wptr[N/128];
    uint8_t width_data[N];
};

uint32_t cp_address(xfont *xf, uint32_t cp)
{
    uint32_t bytes_per_glyph;
    bytes_per_glyph = xf->layout_width * xf->layout_height;

    // the graphic data is in RAM_G
    return (xf->start_of_graphic_data +
        (xf->gptr[cp / 128] + bytes_per_glyph * (cp % 128)));
}

uint8_t cp_width(xfont *xf, uint32_t cp)
{
    return *(
        (uint8_t*)xf +
        xf->wptr[cp / 128] + (cp % 128));
}
```

Note that the structure above is shown to illustrate the fields of the xfont block clearly. A code implementation of the above structure could use the following defines. The defines help to ensure that the structure can be compiled without errors due to the variable sizes of the last three entries in the structure.

```

#define XF_GPTR(xf) ( (unsigned int*)&(((int*)xf)[10]) )
#define XF_WPTR(xf) ( (unsigned int*)&(((char*)xf)[40 + 4 * \
    (xf->number_of_characters / 128)]) )
#define XF_WIDTH(xf) ( (unsigned char*)&(((char*)xf)[0]))

typedef struct
{
    uint32_t signature; // Must be 0x0100AAFF
    uint32_t size;      // Total size of the font block, in bytes
    uint32_t format;    // Bitmap format
    uint32_t swizzle;   // Bitmap swizzle value
    uint32_t layout_width; // Font bitmap line stride, in bytes
    uint32_t layout_height; // Font bitmap height, in pixels
    uint32_t pixel_width; // Font screen width, in pixels
    uint32_t pixel_height; // Font screen height, in pixels
    uint32_t start_of_graphic_data; // Pointer to font graphic data
    uint32_t number_of_characters; // Total number of characters
    // in font: N (multiple of 128)
} XFONT_EXTENDED;

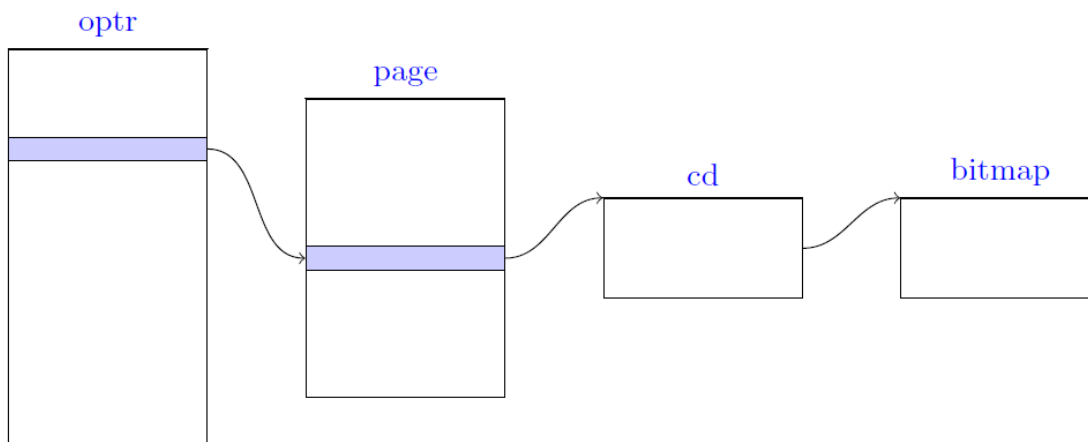
uint32_t cp_address(const XFONT_EXTENDED * xf, uint32_t cp)
{
    uint32_t bytes_per_glyph;
    bytes_per_glyph = xf->layout_width * xf->layout_height;

    // the graphic data is in RAM_G
    return (xf->start_of_graphic_data +
        (XF_GPTR(xf)[cp / 128] + bytes_per_glyph * (cp % 128)));
}

uint8_t cp_width(const XFONT_EXTENDED * xf, uint32_t cp)
{
    uint32_t offset = XF_WPTR(xf)[cp / 128] + (cp % 128);
    return XF_WIDTH(xf)[offset];
}

```

5.5.3 Extended Format 2 Font Metrics Block



The font block is variable-sized, depending on the number of characters. The optr block holds pointers to each 128-entry page. Each 128-entry page holds pointers to 128 cds (character descriptor).

Each cd holds the character's width, a pointer to the bitmap (or "glyph"), and a list of kern pairs. Kern pairs adjust the current character's x coordinate according to the previous character. The kern pairs are in ascending order of character, for efficient linear search. (As soon as a character greater than the previous one is found, search can exit.) Hence the character code 0x7fffffff effectively ends the kern pair list, since it is greater than any legal character code.

+0	signature	Must be 0x0200aaff
+4	size	Total size of the font block, in bytes
+8	format	offset within asset of relocation start
+10	flags	Font flags (see below)
+12	swizzle	Bitmap swizzle, see BITMAP_SWIZZLE
+16	layout_width	Font bit map line stride, in bytes
+20	layout_height	Font bitmap height, in bytes
+24	pixel_width	Font screen width, in pixels
+28	pixel_height	Font screen width, in pixels
+32	xPadding	See diagram
+34	leading	See diagram
+36	no_of_chars	Total number of glyphs in font (N)
+40	midline	Font midline height, in pixels
+42	baseline	Font baseline height, in pixels
+44	optr [0]	Pointer to page 0
+48	optr [1]	Pointer to page 1
	.	
	.	
	.	
+n	optr [N]	Pointer to page [N/128]

Each cd defines one character, and has the following layout, specifying width, glyph address and zero or more kern pairs.

+0	glyph	Address of the character's glyph
+4	width	Character width in pixels
+8	code [0]	Kerning previous character code
+12	distance [0]	Kerning distance, in pixels
+16	code [1]	
+20	distance [1]	
	.	
	.	
	.	
	0x7fffffff	Kern list end marker

So, to find the bitmap address and width of a code point cp:

```
structure of font2:
typedef struct {
uint32_t signature;
uint32_t size;
uint16_t format;
uint16_t flags;
uint32_t swizzle;
uint32_t layout_width;
uint32_t layout_height;
uint32_t pixel_width;
uint32_t pixel_height;
uint16_t xPadding;
uint16_t leading;
uint32_t number_of_characters;
uint16_t midline;
uint16_t baseline;
uint32_t dummy;
uint32_t ** optr[N/128];
} xfont;
typedef struct {
uint32_t code;
uint32_t distance;
} kern;
typedef struct {
uint32_t glyph;
uint32_t width;
kern * kerns;
} chblk;
chblk * cp_address(xfont * xf, uint32_t cp)
{
chblk ** page = &((chblk **) ((uint8_t *)xf + 48))[cp / 128];
return page[cp % 128];
}
int cp_width(xfont * xf, uint32_t cp)
{
return cp_address(xf, cp)->width;
}
int cp_glyph(xfont * xf, uint32_t cp)
{
return cp_address(xf, cp)->glyph;
}
```

5.5.4 Fonts Flags

All font formats have a 16-bit flags field that modifies font rendering behavior.

15		2	1	0
	reserved		CJK	BIN

If BIN (bit 0) is set, the font is binary, disabling newline interpretation of the font string.
 If CJK (bit 1) is set, then OPT_FILL breaks lines on any character, instead of at word breaks.

5.5.5 ROM Fonts (Built-in Fonts)

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

During startup, **ROM** fonts 16 to 34 are attached to bitmap handles 16 to 34. These fonts can be attached to arbitrary bitmap handles with **CMD_ROMFONT**. Refer to [CMD_ROMFONT](#) for more details.

On the other hand, bitmap handle 16 to 34 can also be overwritten with a custom font if desired. To reset the bitmap handles to the default **ROM** font, use [CMD_RESETFONTS](#).

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:

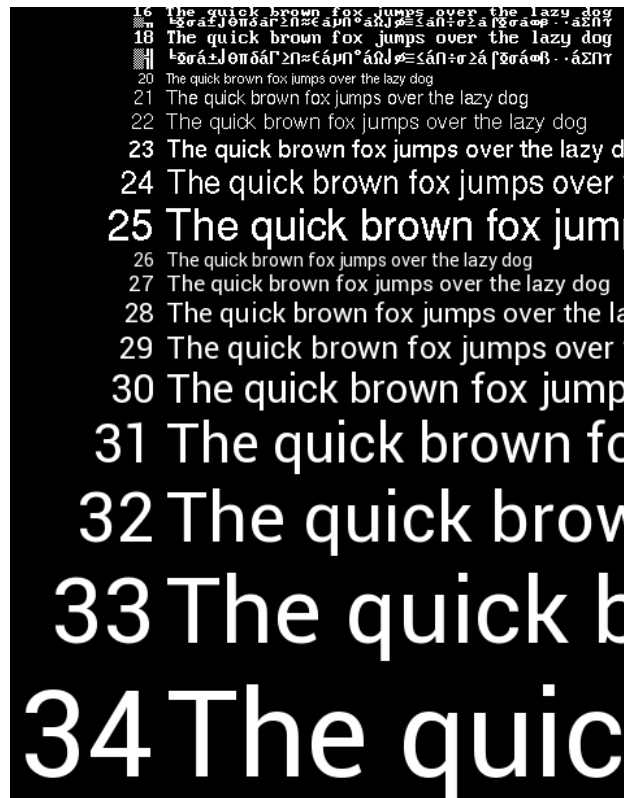


Figure 4 – ROM Font List

5.5.6 Using Custom Font

Method 1

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

- load the font data into RAM
- Use command **CMD_SETFONT** to register the new font with the handle 0-63

After this setup, the font's handle 0-63 can be used as a font argument of coprocessor commands.

Method 2

To use a custom font using **CMD_LOADASSET** (preferred way):

- Open and load the font data
- Get the first unallocated memory location using command **CMD_GETPTR**
- Set the font data destination address using command **CMD_LOADASSET**
- Use **RAM_CMD** to send the font data
- Use command **CMD_SETFONT** to register the new font with the handle 0-63

After this setup, the font's handle 0-63 can be used as a font argument of coprocessor commands.

```
FILE * fp = fopen("custom_font.reloc", "rb");
fseek(fp, 0 SEEK_END);
uint32_t file_size = ftell(fp);
rewind(fp);
fread(buffer, 1, file_size, fp);

uint32_t x = rd32(REG_CMD_WRITE);
uint32_t newfont_address = 0;
cmd_getptr(0); // get first unallocated location
newfont_address = rd32(RAM_CMD + (x + 4) % 16384);
cmd_loadasset(0xffffffff, 0); // current allocation pointer is used

x = 0;
while (x < file_size) {
    cmd(buffer[x] | (buffer[x+1] << 8) |
        (buffer[x+2] << 16) | (buffer[x+3] << 24)); // 32-bits
    x = x + 4;
}

uint32_t HANDLE = 8; // assign a new handle
Cmd_setfont(HANDLE, newfont_address, 0);
```

5.6 Animation support

Based on **ASTC** format of bitmap data, **BT82X** can play back the animation efficiently with minimum **MCU** effort and memory usage. To achieve that, the animation data and object are defined. The utility has been provided to generate these animation assets.

The animation data consists of a sequence of display list fragments. Each fragment must be 64-byte aligned, and has a length that is a multiple of 4. The animation object is also 64-byte aligned, and contains:

- a signature
- a frame count
- an array of references to the display list fragments.


```
// A fragment is: a pointer to display list data, and a size
struct fragment {
    uint32_t nbytes; // must be 4-byte aligned
    uint32_t ptr;    // must be 64-byte aligned
};

struct animation_header {
    uint32_t signature; // always ANIM_SIGNATURE (0xAAAA0100)
    uint32_t num_frames;
    struct fragment table[num_frames];
};
```

Note that a fragment can appear multiple times in a table, for example for animation that is slower than the frame rate. Fragments contain regular display list commands. The fragment code is appended to the display list as follows in order that the fragment can:

1. change graphics state,
2. load and use any bitmaps using the current bitmap handle.

Typically, the bitmap data for a fragment also resides in flash and a typical display list to show the fragment is as below:

```
SAVE_CONTEXT
BITMAP_HANDLE(scratch_handle)
<fragment>
RESTORE_CONTEXT
```

Animations run in channels. A channel keeps track of the animation state. There are 32 animation channels. Each channel can handle one animation. The animation commands are:

- **CMD_ANIMFRAME** - render one frame of an animation
- **CMD_ANIMSTART** - start an animation
- **CMD_ANIMSTOP** - stop animation
- **CMD_ANIMXY** - set the (x; y) coordinates of an animation
- **CMD_ANIMDRAW** - draw active animations

All animation functions accept a channel number within 0-31. Many of the functions also accept an argument of -1, with a special meaning. For example, **CMD_ANIMSTOP** with argument 0-31 stops an animation in that channel, or with argument -1 stops animation in all channels. The 32-bit register **REG_ANIM_ACTIVE** is a bitmask. Bit N is 1 if animation channel N is running. By polling **REG_ANIM_ACTIVE** the host can check for completion of an animation.

In addition, another command **CMD_RUNANIM** is also introduced to simplify the playing back animation.

Example 1:

```
/**
play back an animation once in flash
***/

//set up channel 1
cmd_animstart(1,4096, ANIM_ONCE);
cmd_animxy(400, 240); //The center of animation

//draw each frame in the animation object in a while loop.
while (0 == rd32 (REG_DLSWAP)) {
    cmd_dlstart();
    cmd_animdraw();

    cmd_swap();

    if (0 == rd32 (REG_ANIM_ACTIVE))
        break;
}
cmd_animstop(1);
```

Example 2:

```
/**
play back the animation from frame to frame using cmd_animframe.
FRAME_COUNT is the number of frames to be rendered.
***/
for (int i = 0; i < FRAME_COUNT; i++)
{
    cmd_dlstart();
    cmd(CLEAR(1,1,1));
    cmd_animframe(400,240, 4096, i); //draw the ith frame.
    cmd(DISPLAY());
    cmd_swap();
}
```

5.7 String Formatting

Some coprocessor commands, such as **CMD_TEXT**, **CMD_BUTTON**, **CMD_TOGGLE**, accept a zero-terminated string argument. This string may contain UTF-8 characters, if the selected font contains the appropriate code points.

If the **OPT_FORMAT** option is given in the command, then the string is interpreted as a printf-style format string. The supported formatting is a subset of standard C99. The output string may be up to 4096 bytes in length. Arguments to the format string follow the string and its padding. They are always 32-bit and aligned to 32-bit boundaries. For example, the command:

```
cmd_text(0, 0, 26, OPT_FORMAT, "%d", 237);
```

Should be serialized as:

Offset	Size (in bytes)	Value	Remarks
0	4	0xFFFFFFFF0C	CMD_TEXT
4	2	0	X coordinate
6	2	0	Y coordinate
8	2	26	Font handle
10	2	OPT_FORMAT	Options
12	1	'%'	Format specifier
13	1	'd'	Conversion specifier
14	1	0	Padding bytes for 32 bits alignment
15	1	0	
16	4	237	Integer

The format string is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments from the input stream. Each conversion specification is introduced by the character specifier. In between there may be (in this order) zero or more flags, an optional minimum field width and an optional precision.

5.7.1 The Flag Characters

The character % is followed by zero or more of the following flags:

Flag	Description
0	The value should be zero padded. For d , i , u , o , x , and X conversions, the converted value is padded on the left with zeros rather than blanks. If the 0 and - flags both appear, the 0 flag is ignored. For other conversions, the behavior is undefined.
-	The converted value is to be left adjusted on the field boundary. (The default is right justification.) The converted value is padded on the right with blanks, rather than on the left with blanks or zeros
' ' (a space)	A blank should be left before a positive number (or empty string) produced by a signed conversion
+	A sign (+ or -) should always be placed before a number produced by a signed conversion. By default, a sign is used only for negative numbers.

5.7.2 The Field Width

An optional decimal digit string (with nonzero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given). Instead of a decimal digit string one may write '*' to specify that the field width is given in the next argument. A negative field width is taken as a '-' flag followed by a positive field width. In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

5.7.3 The Precision

An optional precision, in the form of a period ('.') followed by an optional decimal digit string. Instead of a decimal digit string one may write '*' to specify that the field width is given in the next argument. If the precision is given as just '.', the precision is taken to be zero. This gives the minimum number of digits to appear for d, i, u, o, x, and X conversions, the number of digits to appear after the radix character for a, A, e, E, f, and F conversions, the maximum number of significant digits for g and G conversions, or the maximum number of characters to be printed from a string for s and S conversions.

5.7.4 The Conversion Specifier

A character that specifies the type of conversion to be applied. The conversion specifiers and their meanings are:

Specifiers	Meaning
d,i	The integer argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros
u, o, x, X	The unsigned integer argument is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal (x and X) notation. The letters abcdef are used for x conversions; the letters ABCDEF are used for X conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.
c (lower case)	The integer argument is treated as a Unicode code point, and encoded as UTF-8 .
s (lower case)	The argument is expected to be an address of RAM_G storing an array of characters. Characters from the array are written up to (but not including) a terminating null byte; if a precision is specified, no more than the number specified are written. If a precision is given, no null byte need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating null byte.
%	A '%' is written. No argument is converted. The complete conversion specification is '%%'.

Table 42 – String Format Specifier

Examples:

Format string	Output	Assumption
"%3d%% complete", c	51% complete	int c = 51
"base address %06x", a	base address 12a000	int a = 0x12a000
"%+5.3umV", mv	+1947 mV	unsigned int mv = 1947
"Temp %d%.1d degree", t / 10, t % 10	Temp 68.0 degrees	int c = 680
"%s %d times", RAM_G + 4, nTimes	Hello 5 times	" RAM_G +4" is the starting address of the string

		int nTimes = 5
--	--	----------------

5.8 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 size of display list **RAM_DL**.

In the fault condition, the coprocessor:

1. writes a 128-byte diagnostic string to memory starting at **RAM_ERR_REPORT** (0x7f004800).
2. sets **REG_CMD_READ** to 0x3fff (an illegal value because all command buffer data is 32-bit aligned),
3. raises the **INT_CMDEEMPTY** interrupt
4. stops accepting new commands

The diagnostic string gives details of the problem, and the command that triggered it. The string is up to 128 bytes long, including the terminating 0x00. It always starts with the text "ERROR" For example, after a fault the memory buffer might contain:

```

45 52 52 4f 52 3a 20 69 6c 6c 65 67 61 6c 20 6f |ERROR: illegal o|
70 74 69 6f 6e 20 69 6e 20 63 6d 64 5f 69 6e 66 |ption in cmd_inf|
6c 61 74 65 32 28 29 00 00 00 00 00 00 00 00 00 |late2().....|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

```

The possible errors are:

Error string	Remarks
display list overflow	The RAM_DL has exceeded its limit with over 4096 instructions
illegal font or bitmap handle	valid handles are 0-63
out of channel	the animation channels are used up
uninitialized font	font should be set up with CMD_ROMFONT or CMD_SETFONT
illegal alignment	flash commands only support certain alignments
illegal flash address	access 32-bit address location while CMD_FLASHFAST has not been called and REG_FLASH_STATUS does not indicate FULL
illegal option	a command's option(parameter) was not recognized
invalid animation	the animation object or frame is not valid
invalid animation channel	animation channel number is not valid
invalid base	a number base was given outside the range 2-36
unsupported JPEG	the JPEG image is not supported (e.g. progressive)
invalid size	a radius, width, or height was negative or zero
corrupted JPEG	the JPEG image data is corrupted
unsupported PNG	the PNG image is not supported
corrupted PNG	the PNG image data is corrupted
image type not recognized	the image is not a PNG or JPG

unknown bitmap format	CMD_SETBITMAP was called with a unknown bitmap format
corrupted AVI	the AVI data is corrupted
invalid format character	an invalid character appeared in a format
invalid format string	the format conversion specifier was not found
format buffer overflow	the format output buffer used more than 256 bytes

Table 43 – Coprocessor Faults Strings

When the host MCU encounters the fault condition, it can recover as follows:

- Set **REG_CMD_READ** to zero
- Wait for **REG_CMD_WRITE** to be zero

After this sequence the read and write pointers are both zero, so the command FIFO is empty. The coprocessor is ready to accept a command.

5.9 Coprocessor State

The coprocessor maintains a small amount of internal state for graphics drawing. This state is set to the default at coprocessor reset, and by **CMD_COLDSTART**. The state values are not affected by **CMD_DLSTART** or **CMD_SWAP**, so an application needs only to set them once at startup. Graphics state (marked with • below) can be saved and preserved with **CMD_SAVECONTEXT** and **CMD_RESTORECONTEXT**.

State	Graphics	Default	Commands
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, CMD_ROTATEAROUND
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
numeric base	•	10	CMD_SETBASE
text fill width	•	256	CMD_FILLWIDTH, CMD_TEXT
spinner		<i>None</i>	CMD_SPINNER
object trackers		<i>all disabled</i>	CMD_TRACK
interrupt timer		<i>None</i>	CMD_INTERRUPT
Font pointers 0-15		<i>Undefined</i>	CMD_SETFONT
Font pointers 16-34		<i>ROM fonts 16-31</i>	CMD_SETFONT, CMD_ROMFONT
Font pointers 35-63		<i>Undefined</i>	CMD_SETFONT
Scratch bitmap handle		15	CMD_SETSCRATCH
allocation pointer		0	CMD_LOADIMAGE, CMD_INFLATE, etc

Table 44 – Coprocessor Engine State

5.10 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 CMD_PLAYVIDEO
OPT_TRUECOLOR	0x0200	Use ARGB8 format bitmaps for video frames or image.	CMD_LOADIMAGE CMD_PLAYVIDEO
OPT_YCBCR	0x0400	Use YCBCR format bitmaps for video frames or image.	CMD_LOADIMAGE CMD_PLAYVIDEO
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 CMD_PLAYVIDEO
OPT_FLAT	256	No 3D effect	CMD_BUTTON CMD_CLOCK CMD_KEYS CMD_GAUGE CMD_SLIDER CMD_DIAL CMD_TOGGLE CMD_PROGRESS CMD_SCROLLBAR
OPT_SIGNED	256	The number is treated as a 32 bit signed integer	CMD_NUMBER
OPT_CENTERX	512	Horizontally-centred style	CMD_KEYS CMD_TEXT CMD_NUMBER
OPT_CENTERY	1024	Vertically centred style	CMD_KEYS CMD_TEXT CMD_NUMBER
OPT_CENTER	1536	horizontally and vertically centred style	CMD_KEYS CMD_TEXT CMD_NUMBER
OPT_RIGHTX	2048	Right justified style	CMD_KEYS CMD_TEXT CMD_NUMBER
OPT_NOBACK	4096	No background drawn	CMD_CLOCK CMD_GAUGE
OPT_FILL	8192	Breaks the text at spaces into multiple lines, with maximum width set by CMD_FILLWIDTH .	CMD_BUTTON CMD_TEXT
OPT_FLASH	64	Fetch the data from flash memory	CMD_INFLATE CMD_LOADIMAGE CMD_PLAYVIDEO CMD_PLAYWAV CMD_VIDEOSTART CMD_LOADASSET
OPT_FORMAT	4096	Flag of string formatting	CMD_TEXT CMD_BUTTON

Name	Value	Description	Commands
			CMD_TOGGLE
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_FULLSCREEN	8	Zoom the video so that it fills as much of the screen as possible.	CMD_PLAYVIDEO CMD_LOADIMAGE
OPT_MEDIAFIFO	16	Source video/image/compressed(zlib) data from the defined media FIFO	CMD_PLAYVIDEO CMD_VIDEOSTART CMD_LOADIMAGE CMD_INFLATE CMD_PLAYWAV CMD_LOADASSET
OPT_OVERLAY	128	Append the video bitmap to an existing display list	CMD_PLAYVIDEO
OPT_SOUND	32	Decode the audio data	CMD_PLAYVIDEO
OPT_DITHER	256	Enable dithering feature in decoding PNG process	CMD_LOADIMAGE
OPT_DIRECT	0x800	Play back the video directly to the scanout system	CMD_PLAYVIDEO
OPT_COMPLETEREG	0x1000	The application should set the REG_OBJECT_COMPLETE flag at the end of the video data. This prevents a hang situation with truncated video data, instead the EVE raises an exception	CMD_PLAYVIDEO
OPT_FS	0x2000	The filesystem is the source	CMD_LOADASSET CMD_LOADIMAGE CMD_PLAYVIDEO CMD_PLAYWAV CMD_VIDEOSTART CMD_INFLATE
OPT_4BIT	0x0002	The SD interface to be initialized in 4-bit mode	CMD_SDATTACH
OPT_HALFSPEED	0x0004	The SD card to run at half-speed (one quarter of the system clock)	CMD_SDATTACH
OPT_QUARTERSPEED	0x0008	The SD card to run at quarter-speed (one eighth of the system clock)	CMD_SDATTACH
OPT_IS_SD	32	Assume the attached storage is an SD card	CMD_SDATTACH
OPT_IS_MMC	16	Assume the attached storage is eMMC	CMD_SDATTACH
OPT_SFNLOWER	0x0001	FAT32 short filenames are mapped to lower case, in accordance with the Microsoft de-facto standard	CMD_FSOPTIONS
OPT_CASESENSITIVE	0x0002	Filename comparison is case sensitive	CMD_FSOPTIONS
OPT_DIRSEP_WIN	0x0004	Treat backslash as a directory path separator	CMD_FSOPTIONS
OPT_DIRSEP_UNIX	0x0008	Treat forward slash as a directory path separator	CMD_FSOPTIONS

Table 45 – Parameter OPTION Definition

5.11 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**.
- One graphics context is used by objects, and the effective stack depth for **SAVE_CONTEXT** and **RESTORE_CONTEXT** commands is 3 levels.

5.12 Widget extents

After rendering a widget, the screen extents of the widget are written to registers:

REG_EXTENT_X0, **REG_EXTENT_Y0**, **REG_EXTENT_X1** and **REG_EXTENT_Y1**.

The corners of the rectangle are

(**REG_EXTENT_X0**, **REG_EXTENT_Y0**) and (**REG_EXTENT_X1**, **REG_EXTENT_Y1**), shown as below:

(REG_EXTENT_X0, REG_EXTENT_Y0)



(REG_EXTENT_X1, REG_EXTENT_Y1)

5.13 Command list

Command list enables users to construct a series of coprocessor command or display list at **RAM_G**. These are the new commands to facilitate:

- **CMD_NEWLIST**
- **CMD_CALLLIST**
- **CMD_RETURN**
- **CMD_ENDLIST**
- **CMD_COPYLIST**
- **CMD_SAVECONTEXT**
- **CMD_RESTORECONTEXT**

The examples can be found in the sections of the commands below.

5.14 Command Groups

The tables below capture the new and updated commands in **BT82X**:

Coprocessor Commands	BT82X	Remarks
<i>CMD_SDATTACH</i> <i>CMD_FSSOURCE</i> <i>CMD_FSREAD</i> <i>CMD_FSSIZE</i> <i>CMD_FSOPTIONS</i> <i>CMD_SDBLOCKREAD</i>	New	SD card feature related coprocessor commands
<i>CMD_NEWLIST</i> <i>CMD_ENDLIST</i> <i>CMD_CALLLIST</i>	New	Start the compilation of a command list into RAM_G. End the compilation of a command list into RAM_G Call a command list / display list
<i>CMD_ARC</i>	New	Draw a circular arc
<i>CMD_CGRADIENT</i>	New	Draw a circular gradient

<i>CMD_COPYLIST</i>	New	Copy the display list into RAM_G
<i>CMD_DDRSHUTDOWN</i> <i>CMD_DDRSTARTUP</i>	New	DDR shutdown DDR system start up
<i>CMD_ENABLEREGION</i>	New	Enable / disable REGION optimization
<i>CMD_FENCE</i>	New	Wait for outstanding writes
<i>CMD_FLASHPROGRAM</i>	New	Write data to flash
<i>CMD_GLOW</i>	New	Draw an additive glow
<i>CMD_GRAPHICSFINISH</i>	New	Wait for the render engine to complete
<i>CMD_INFLATE</i>	Updated	Add options (OPT_Flash, OPT_FS, OPT_MEDIAFIFO)
<i>CMD_LOADIMAGE</i>	Updated	Add more options (OPT_FS)
<i>CMD_LOADASSET</i>	New	Load an asset into RAM_G
<i>CMD_LOADWAV</i>	New	Load a WAV file for background playback
<i>CMD_PLAYVIDEO</i>	Updated	Add options (OPT_YCBCR, OPT_FS, OPT_COMPLETEREG)
<i>CMD_PLAYWAV</i>	New	Playing back an audio sample
<i>CMD_REGWRITE</i>	New	Write a register
<i>CMD_RENDERTARGET</i>	New	Set the render target
<i>CMD_RESTORECONTEXT</i>	New	Restore graphics state
<i>CMD_RESULT</i>	New	Write previous result to memory
<i>CMD_SAVECONTEXT</i>	New	Save graphics state
<i>CMD_SETBITMAP</i>	Updated	Source address for bitmap in RAM_G
<i>CMD_SETFONT</i>	Updated	Replace CMD_SETFONT2 in older chips
<i>CMD_SKIPCOND</i>	New	Skip commands if condition is true
<i>CMD_TEXTDIM</i>	New	Compute the size of a UTF-8 text string
<i>CMD_WAITCHANGE</i> <i>CMD_WAITCOND</i>	New	Wait for a register to change Wait for a condition
<i>CMD_WATCHDOG</i>	New	Set the watchdog timeout
<i>CMD_VIDEOSTART</i>	Updated	Add options (OPT_FS)

Table 46 – Updated Coprocessor Commands in BT82X

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 **UTF-8** text string
- **CMD_BUTTON**-- draw a button with a **UTF-8** label.
- **CMD_APPEND**-- append memory to display list
- **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 **UTF-8** labels
- **CMD_NUMBER**-- draw a decimal number
- **CMD_SETBASE**-- set the base for number output
- **CMD_FILLWIDTH**-- set the text fill width
- **CMD_ARC**-- draw a circular arc
- **CMD_GLOW**-- draw an additive glow
- **CMD_CGRADIENT**-- draw a circular gradient

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**

Commands for loading data into **RAM_G**:

- **CMD_INFLATE**-- decompress data into **RAM_G** with more options
- **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** or flash memory.
- **CMD_PLAYVIDEO**-- video playback
- **CMD_PLAYWAV**-- wav audio playback
- **CMD_LOADWAV**-- load a WAV file for background playback.
- **CMD_FSREAD**-- read file from filesystem to **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_ROTATEAROUND**-- apply a rotation and scale around the specified pixel
- **CMD_SETMATRIX**-- write the current matrix as a bitmap transform
- **CMD_GETMATRIX**-- retrieves the current matrix coefficients

Commands for flash operation:

- **CMD_FLASHERASE**-- Erase all of flash
- **CMD_FLASHWRITE**-- Write data from **RAM_CMD** to blank flash
- **CMD_FLASHUPDATE**-- write data to flash, erasing if necessary
- **CMD_FLASHPROGRAM**-- write data from **RAM_G** to blank flash
- **CMD_FLASHDETACH**-- detach from flash
- **CMD_FLASHATTACH**-- attach to flash
- **CMD_FLASHFAST**-- enter full-speed mode
- **CMD_FLASHSPIDESEL**--SPI bus: deselect device
- **CMD_FLASHSPITX**-- SPI bus: write bytes
- **CMD_FLASHSPIRX**-- SPI bus: read bytes
- **CMD_FLASHSOURCE**-- specify the flash source address for the following coprocessor commands
- **CMD_APPENDF**-- Read data from flash to **RAM_DL**

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

Commands for animation:

- **CMD_ANIMFRAME**-- render one frame of an animation
- **CMD_ANIMSTART**-- start an animation
- **CMD_ANIMSTOP**-- stop animation
- **CMD_ANIMXY**-- set the (x,y) coordinates of an animation
- **CMD_ANIMDRAW**-- draw active animation
- **CMD_RUNANIM**-- run an animation until complete

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_CALIBRATESUB**-- execute the touch screen calibration routine for a subwindow
- **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_TESTCARD**-- display a panel testcard
- **CMD_LOGO**-- play device logo animation

5.15 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

5.16 CMD_INTERRUPT

This command is used to trigger Interrupt **INT_CMDFLAG**. When the coprocessor engine executes this command, it triggers interrupt, which will set the bit field **INT_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 **ms** parameter specifies the delay before the interrupt is triggered, measured in milliseconds. If set to 0, the interrupt fires immediately.

Note: The maximum allowable value for ms is 65536.

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);
//...
  
```

5.17 CMD_COLDSTART

This command sets the coprocessor engine to default reset states.

C prototype

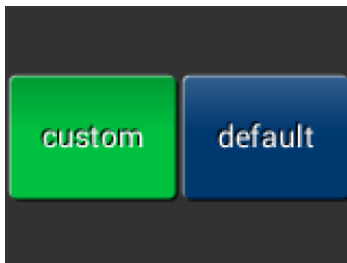
```
void cmd_coldstart( );
```

Command layout

+0	CMD_COLDSTART(0xFFFF FF2E)
----	-----------------------------------

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");
  
```

5.18 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

5.19 CMD_APPEND

This command appends more commands resident in **RAM_G** to the current display list memory address where the offset is specified in **REG_CMD_DL**.

C prototype

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

Parameters

ptr

Starting address of source commands in RAM_G

num

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

Command layout

+0	CMD_APPEND(0xFFFF FF1C)
+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.

Note that any commands appended still count towards the maximum total of 16Kbytes for **RAM_DL**.

Examples

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

5.20 CMD_REGREAD

This command is used to read a register value.

Note: This command only supports graphics engine related registers.

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 FF17)
+4	ptr
+8	result

Examples

```
//To capture the exact time when a command completes:
uint32_t x = rd32(REG_CMD_WRITE);
cmd_regread(REG_CLOCK, 0);
//...
printf("%08x\n", rd32(RAM_CMD + (x + 8) % 16384));
```

5.21 CMD_REGWRITE

The cmd_regwrite command writes a value to a core register. While it is possible to write a value directly to a register, the advantage of using this command is that the write happens in order with other commands.

Note: This command only supports graphics engine related registers.

C prototype

```
void cmd_regwrite( uint32_t dst,
                  uint32_t value );
```

Parameters

ptr
Register address to write, 32-bit aligned

value
32-bit value to write

Command layout

+0	CMD_REGWRITE(0xFFFF FF86)
+4	dst
+8	value

Examples

To write 1200 to REG_HSIZE:

```
cmd_regwrite(REG_HSIZE, 1200);
```

5.22 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 FF18)
+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);
```

5.23 CMD_INFLATE

This command is used to decompress the following compressed data into **RAM_G**. The data may be supplied in the command buffer, the media **FIFO**, or from flash memory. 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,
                 uint32_t options );
```


Parameters

ptr

Destination address. If -1 is given as the address, then the current allocation pointer is used instead.

options

By default, the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

OPT_FLASH : flash memory is the source, see CMD_FLASHSOURCE.

OPT_FS : the filesystem is the source, see CMD_FSSOURCE.

OPT_MEDIAFIFO: the media FIFO is used for the source, see CMD_MEDIAFIFO.

Description

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

Command layout

+0	CMD_INFLATE(0xFFFF FF4A)
+4	ptr
+8	options
+12....+n	byte ₁ ...byte _n

5.24 CMD_LOADIMAGE

This command is used to load a **JPEG** or **PNG** image, decompress it into an EVE-specific bitmap in RAM_G, and generate the display list to set up the bitmap handle.

The input **JPEG** image shall be:

- A regular baseline **JPEG (JFIF)** with chroma subsampling 4:2:0 or 4:2:2 or 4:4:4

The input **PNG** image shall be:

- A non-interlaced **PNG** with bit-depth 8 only

C prototype

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

Parameters

ptr

Destination address, where the decompressed bitmap stores.

options

For **JPEG** images, the decompressed bitmap is loaded as **RGB565** format by default, unless:

OPT_MONO is specified, in which case it is loaded in **L8** format.

OPT_YCBCR is specified, in which case it is loaded in **YCBCR** format.

OPT_TRUECOLOR is specified, in which case it is loaded in **ARGB8** format.

When **OPT_YCBCR** is specified, the input **JPEG** image shall be:

- A regular baseline **JPEG(JFIF)** with chroma subsampling pattern **4:2:0** only,
- Width and height in multiple of 2 pixels

For **PNG** images, the **PNG** standard defines several color formats. Each format is loaded as a bitmap as follows:

Color type	Format	Default bitmap format	OPT_TRUECOLOR bitmap format	Remarks
0	Grayscale	L8	L8	
2	Truecolor	RGB565	RGB8	
3	Indexed	PALETTERDARGB8	PALETTERDARGB8	PALETTE_SOURCE/PALETTE_SOURCEH is generated if OPT_NODL is not given
4	Grayscale and alpha	LA8	LA8	
6	Truecolor and alpha	ARGB4	ARGB8	

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

By default, the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

- **OPT_FLASH**: flash memory is the source, see CMD_FLASHSOURCE
- **OPT_FS**: the filesystem is the source, see CMD_FSSOURCE
- **OPT_MEDIAFIFO**: the media FIFO is used for the source, see CMD_MEDIAFIFO

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:

- BITMAP_SOURCE/BITMAP_SOURCEH
- PALETTE_SOURCE/PALETTE_SOURCEH
- BITMAP_LAYOUT/BITMAP_LAYOUTH
- BITMAP_SIZE/BITMAP_SIZEH

Description

The data byte should immediately follow in the command FIFO if **OPT_MEDIAFIFO** or **OPT_FLASH** or **OPT_FS** 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 must parse the JPEG/PNG header to get the properties of the JPEG/PNG image and decide to decode. Behavior is unpredictable in cases of non-baseline JPEG images, or the output data generated is more than the **RAM_G** size.

Command layout

+0	CMD_LOADIMAGE(0xFFFF FF21)
+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)
```

5.25 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**
- Compressed data by zlib: **CMD_INFLATE**

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**, 4-byte aligned

size
number of bytes of media **FIFO**, 4-byte aligned

Command layout

+0	CMD_MEDIAFIFO (0xFFFF FF34)
+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=0x000F000

5.26 CMD_PLAYVIDEO

Plays back MJPEG-encoded AVI video.

Playback starts immediately, and the command completes when playback ends. The playback may be paused or terminated by writing to 32-bit register **REG_PLAY_CONTROL**.

The register's value controls playback as follows:

- -1 exit playback
- 0 pause playback
- 1 play normally

The register's reset value is 1.

The video playback uses intermediate bitmap buffers. The default format of these buffers is RGB565, but may be overridden with **OPT_YCBCR** or **OPT_TRUECOLOR**. **OPT_YCBCR** gives the best performance. Before starting video playback, the user must configure swap chain 1 with suitable locations for the video frames. (See section 2.11: [Swap Chains](#).) Each frame contains a YCBCR bitmap and is W xH bytes in size, where W and H are the width and height of the video.

C prototype

```
void cmd_playvideo (uint32_t options);
```

Parameters

options: The options of playing video

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

OPT_SOUND: play back the video's sound through the EVE audio output.

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

OPT_DIRECT: play back the video directly to the scanout system. (**REG_SO_FMT** must be YCBCR and scanout screen size must match the video size.)

OPT_YCBCR uses YCBCR format bitmaps for video frames. The default is RGB565.

OPT_TRUECOLOR uses ARGB8 format bitmaps for video frames. The default is RGB565.

OPT_COMPLETEREG: the application should set the register REG_OBJECT_COMPLETE to 1 at the end of the video data. This prevents a hang with truncated video data in the media FIFO, enabling the EVE to detect this condition and raise an exception.

By default, the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

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

OPT_FLASH: Source video data from flash. When flash is the source, call CMD_FLASHSOURCE before this command to specify the address. See CMD_FLASHSOURCE.

OPT_FS: the filesystem is the source, see CMD_FSSOURCE.

Command layout

+0	CMD_PLAYVIDEO (0xFFFF FF35)
+4	options
+8~ +n	byte ₁ ... byte _n

Data following parameter "options" shall be padded to 4 bytes aligned with zero.

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

4 Bit IMA ADPCM, 8 Bit unsigned PCM, 8 Bit u-Law, 16 bit PCM (S16LE)

Examples

To play back an AVI video, full-screen with sound:

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

5.27 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. Video frames can then be decoded with CMD_VIDEOFRAME.

The default format of the bitmap is RGB565 but may be overridden with **OPT_YCBCR** (best performance) or **OPT_TRUECOLOR** before starting video playback.

C prototype

```
void cmd_videostart( uint32_t options );
```

Parameters

options

One of the following options must be given:

OPT_FLASH: flash memory is the source, see CMD_FLASHSOURCE.

OPT_FS: the filesystem is the source, see CMD_FSSOURCE.

OPT_MEDIAFIFO: the media FIFO is used for the source, see CMD_MEDIAFIFO.

Command layout

+0	CMD_VIDEOSTART (0xFFFF FF3A)
+4	options

Examples

To load frames of video at address 4:

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

5.28 CMD_VIDEOFRAME

This command is used to load the next frame of a video. The video data should be supplied using the method specified in CMD_VIDEOSTART. This command extracts the next frame of video from the selected source 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 must be a physical address not a swapchain destination.

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 FF3B)
+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);
```

5.29 CMD_PLAYWAV

Plays back an audio sample.

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

Note that for **cmd_playwav**, there's no option to enable looping playback. As **REG_PLAYBACK_LOOP** has no effect with the **cmd_playwav** command. Therefore, looping playback cannot be enabled using **cmd_playwav**.

C prototype

```
Void cmd_playwav (uint32_t options);
```

Parameters

Options

By default, the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

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

OPT_FLASH: Source video data from flash. When flash is the source, call **CMD_FLASHSOURCE** before this command to specify the address. See **CMD_FLASHSOURCE**.

OPT_FS: the filesystem is the source, see **CMD_FSSOURCE**.

data

The audio data to be played in **RAM_G** unless **options** is assigned with **OPT_MEDIAFIFO** or **OPT_FLASH** or **OPT_FS**.

Command layout

+0	CMD_PLAYWAV (0xFFFF FF79)
+4	options
+8~ +n	byte ₁ ... byte _n

Data following parameter "options" shall be padded to 4 bytes aligned with zero.

Examples

To play back a sample:

```
cmd_playwav(0);
//... append WAV data ...
```

5.30 CMD_LOADWAV

The **cmd_loadwav** command loads a WAV file into memory so that it can be played or looped asynchronously. As part of the WAV file loading operation, the following audio playback registers are set to the correct values for the incoming samples, according to the header of WAV file and parameter **dst**:

1. REG_PLAYBACK_START
2. REG_PLAYBACK_FREQ
3. REG_PLAYBACK_LENGTH
4. REG_PLAYBACK_FORMAT

For **cmd_loadwav**, the loop flag is turned on after it is loaded. So, if users set **REG_PLAYBACK_PLAY** after executing **cmd_loadwav**, the audio will play in a loop.

C prototype

```
void cmd_loadwav ( uint32_t dst,
                  uint32_t options);
```

Parameters

dst

Destination address. If -1 is given as the address, then the current allocation pointer is used instead.

options

By default, the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

OPT_FLASH: flash memory is the source, see CMD_FLASHSOURCE.

OPT_FS: the filesystem is the source, see CMD_FSSOURCE.

OPT_MEDIAFIFO: the media FIFO is used for the source, see CMD_MEDIAFIFO.

Command layout

+0	CMD_LOADWAV (0xFFFF FF85)
+4	dst
+8	options
+12~ +n	byte ₁ ... byte _n

Examples

To play back a sample:

```
cmd_sdattach(0)
cmd_fssource("background.wav", 0)
cmd_loadwav(addr, OPT_FS);
cmd_regwrite(REG_PLAYBACK_PLAY, 1)
```

5.31 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 FF16)
+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:

```
uint32_t x = rd32(REG_CMD_WRITE);
cmd_memcrc(0, 1024, 0);

//wait till the command is complete
printf("CRC result is %08x\n", rd32(RAM_CMD + (x + 12) % 16384));
```

5.32 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 FF1A)
+4	ptr
+8	num

Examples

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

5.33 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 FF19)
+4	ptr
+8	value
+12	num

Examples

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

5.34 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 FF1B)
+4	dst
+8	src
+12	num

Examples

```

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

```

5.35 CMD_BUTTON

This command is used to draw a button with a UTF-8 label.

C prototype

```

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

```

Parameters

x
 X-coordinate of button top-left, in pixels

y
 Y-coordinate of button top-left, in pixels

w
 width of button, in pixels

h
 height of button, in pixels

font
 font to use for text, 0-63. See [ROM and RAM Fonts](#).

Options

By default, the button is drawn with a 3D effect. **OPT_FLAT** removes the 3D effect. **OPT_FORMAT** processes the text as a format string, see [5.7 String Formatting](#). **OPT_FILL** breaks the text at spaces into multiple lines, with maximum width set by **CMD_FILLWIDTH**. 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. UTF-8 encoded. If **OPT_FILL** is not given then the string may contain newline (\n) characters, indicating line breaks. See [5.7 String Formatting](#).

Description

Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_BUTTON(0xFFFF FFOB)
+4	x
+6	y
+8	w
+10	h
+12	font
+14	options
+16	s
+17	...
...	...
+n	0

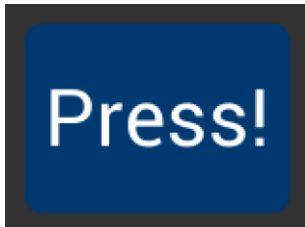
Examples

A 140x100 pixel button with large text:



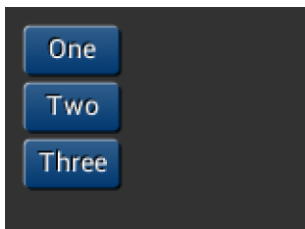
```
cmd_button(10, 10, 140, 100, 31, 0, "Press!");
```

Without the 3D look:



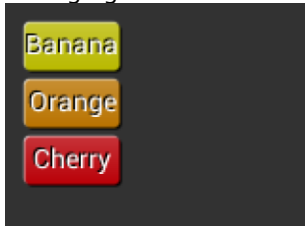
```
cmd_button(10, 10, 140, 100, 31, OPT_FLAT, "Press!");
```

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");
```

5.36 CMD_CLOCK

This command is used to draw an analog clock.

C prototype

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

Parameters

x
x-coordinate of clock center, in pixels

y
y-coordinate of clock center, in pixels

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.

h
hours

m
minutes

s
seconds

ms
milliseconds

Description

The details of the physical dimensions are:

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

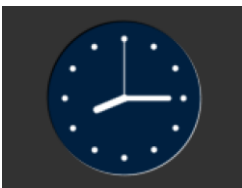
Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_CLOCK(0xFFFF FF12)
+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);
```

5.37 CMD_FGCOLOR

This command is used to set the foreground color.

C prototype

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

Parameters

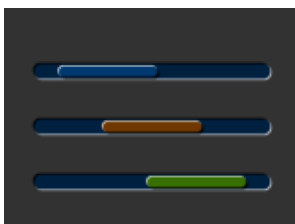
c
 New foreground color, as a 24-bit RGB number.
 Red is the most significant 8 bits, blue is the least. So 0xff0000 is bright red.
 Foreground color is applicable for things that the user can move such as handles and buttons.

Command layout

+0	CMD_FGCOLOR(0xFFFF FF08)
+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);
```

5.38 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.

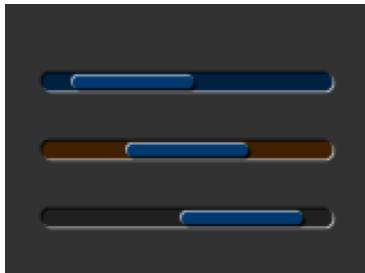
Background color is applicable for things that the user cannot move E.g. behind gauges and sliders etc.

Command layout

+0	CMD_BGCOLOR(0xFFFF FF07)
+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);
```

5.39 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 FF30)
+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");
```

5.40 CMD_GAUGE

This command is used to draw a Gauge.

C prototype

```
void cmd_gauge( int16_t x,
               int16_t y,
               int16_t r,
               uint16_t options,
               uint16_t major,
               uint16_t minor,
               uint16_t val,
               uint16_t range );
```

Parameters

x
X-coordinate of gauge center, in pixels

y
Y-coordinate of gauge center, in pixels

r
Radius of the gauge, in pixels

options

By default, the gauge dial is drawn with a 3D effect and the value of options is zero. OPT_FLAT removes the 3D effect.

With option OPT_NOBACK, the background is not drawn.

With option OPT_NOTICKS, the tick marks are not drawn.

With option OPT_NOPOINTER, the pointer is not drawn.

major

Number of major subdivisions on the dial, 1-10

minor

Number of minor subdivisions on the dial, 1-10

val

Gauge indicated value, between 0 and range, inclusive

range

Maximum value

Description

The details of physical dimension are:

- The tick marks are placed on a 270 degree arc, clockwise starting at south-west position
- Minor ticks are lines of width $r*(2/256)$, major $r*(6/256)$
- Ticks are drawn at a distance of $r*(190/256)$ to $r*(200/256)$
- The pointer is drawn with lines of width $r*(4/256)$, to a point $r*(190/256)$ from the center
- The other ends of the lines are each positioned 90 degrees perpendicular to the pointer direction, at a distance $r*(3/256)$ from the center

Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_GAUGE(0xFFFF FF11)
+4	x
+6	y
+8	r
+10	options
+12	major
+14	minor
+16	value
+18	range

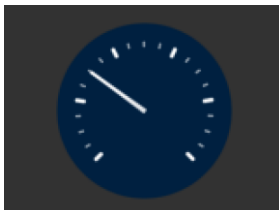
Examples

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:



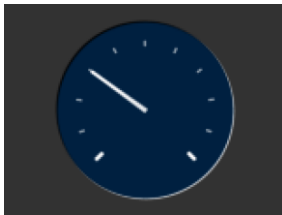
```
cmd_gauge(80, 60, 50, 0, 10, 2, 30, 100);
```

Setting the minor divisions to 1 makes them disappear:



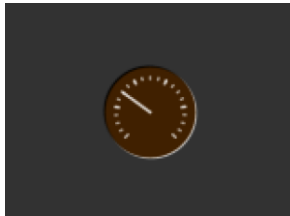
```
cmd_gauge(80, 60, 50, 0, 10, 1, 30, 100);
```

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:



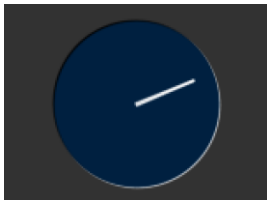
```
cmd_gauge(80, 60, 50, 0, 4, 4, 49152, 65535);
```

No background:



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

No tick marks:



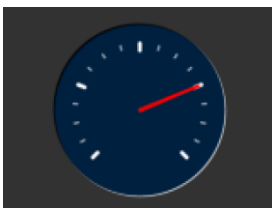
```
cmd_gauge(80, 60, 50, OPT_NOTICKS, 4, 4, 49152, 65535);
```

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);
```

5.41 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

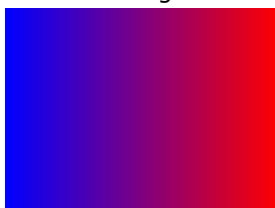
All the color step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is $R0 + t * (R1 - R0)$, where it is interpolated between 0 and 1. Gradient must be used with Scissor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

Command layout

+0	CMD_GRADIENT(0xFFFF FF09)
+4	x0
+6	y0
+8	rgb0
+12	x1
+14	y1
+16	rgb1

Examples

A horizontal gradient from blue to red



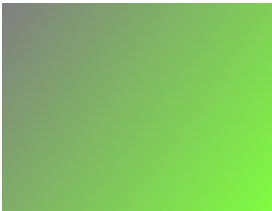
```
cmd_gradient(0, 0, 0x0000ff, 160, 0, 0xff0000);
```

A vertical gradient



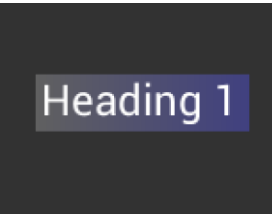
```
cmd_gradient(0, 0, 0x808080, 0, 120, 0x80ff40);
```

The same colors in a diagonal gradient



```
cmd_gradient(0, 0, 0x808080, 160, 120, 0x80ff40);
```

Using a scissor rectangle to draw a gradient stripe as a background for a title:



```
cmd(SCISSOR_XY(20, 40));  
cmd(SCISSOR_SIZE(120, 32));  
cmd_gradient(20, 0, 0x606060, 140, 0, 0x404080);  
cmd_text(23, 40, 29, 0, "Heading 1");
```

5.42 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

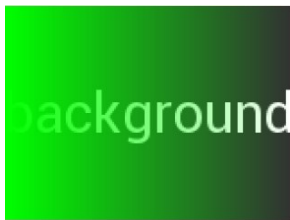
All the color step values are calculated based on smooth curve interpolated from the RGB0 to RGB1 parameter. The smooth curve equation is independently calculated for all three colors and the equation used is $R0 + t * (R1 - R0)$, where it is interpolated between 0 and 1. Gradient must be used with scissor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

Command layout

+0	CMD_GRADIENTA(0xFFFF FF50)
+4	x0
+6	y0
+8	argb0
+12	x1
+14	y1
+16	argb1

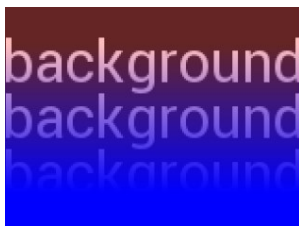
Examples

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);
```

5.43 CMD_CGRADIENT

This command is used to draw a rectangle containing a circular gradient, smoothly blending between one color in the center and another at the corner or edge. The blend effect uses one of two high-quality Gaussian filters.

C prototype

```
void cmd_cgradient( uint32_t shape,
                   int16_t x,
                   int16_t y,
                   int16_t w,
                   int16_t h,
                   uint32_t rgb0,
                   uint32_t rgb1 );
```

Parameters

Shape

Gradient shape. CORNER_ZERO has a gradual falloff that reaches the outer color at the corner. EDGE_ZERO has a sharper falloff that reaches the outer color at the edge.

Name	Value	Description
CORNER_ZERO	0	a gradual falloff that reaches the outer color at the corner.
EDGE_ZERO	1	a sharper falloff that reaches the outer color at the edge

x

x-coordinate of top-left of rectangle, in pixels

y

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

w

width of rectangle in pixels

h

height of rectangle in pixels

rgb0

Outer color

rgb1

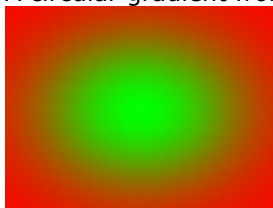
Inner color

Command layout

+0	CMD_CGRADIENT(0xFFFF FF8A)
+4	shape
+8	X
+10	Y
+12	W
+14	H
+16	rgb0
+20	rgb1

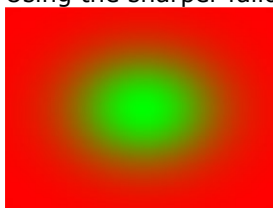
Examples

A circular gradient from red (outer) to green (inner):



```
cmd_cgradient(CORNER_ZERO, 0, 0, 160, 120,
0xff0000, 0x00ff00);
```

Using the sharper falloff of EDGE ZERO:



```
cmd_cgradient(EDGE_ZERO, 0, 0, 160, 120,
0xff0000, 0x00ff00);
```

5.44 CMD_GLOW

The glow command draws an additive glow effect centered in a rectangle, using the current color. The glow effect uses a high-quality Gaussian filter.

C prototype

```
void cmd_glow( int16_t x,
              int16_t y,
              int16_t w,
              int16_t h );
```

Parameters

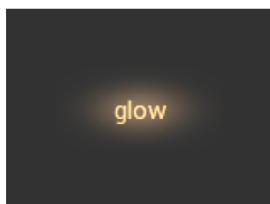
- x**
x-coordinate of top-left of glow rectangle, in pixels
- y**
y-coordinate of top-left of glow rectangle, in pixels
- w**
width of rectangle in pixels
- h**
height of rectangle in pixels

Command layout

+0	CMD_GLOW(0xFFFF FF8B)
+4	x
+6	y
+8	w
+10	h

Examples

A glow covering a piece of text:



```
cmd(COLOR_RGB(0xff, 0xc0, 0x80));
cmd_text(80, 60, 26, OPT_CENTER, "glow");
cmd(COLOR_RGB(0x60, 0x48, 0x30));
cmd_glow(80 - 50, 60 - 25, 2 * 50, 2 * 25);
```

5.45 CMD_KEYS

This command is used to draw a row of keys.

C prototype

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


Parameters

x
 x-coordinate of keys top-left, in pixels.

y
 y-coordinate of keys top-left, in pixels.

font
 Bitmap handle to specify the font used in key label. The valid range is from 0 to 63

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

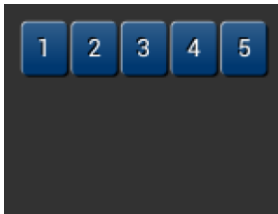
Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_KEYS(0xFFFF FF0C)
+4	x
+6	y
+8	w
+10	h
+12	font
+14	options
+16	s
...	...
+n	0

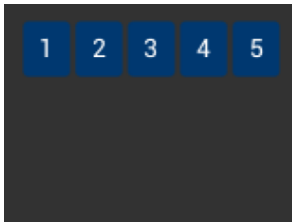
Examples

A row of keys:



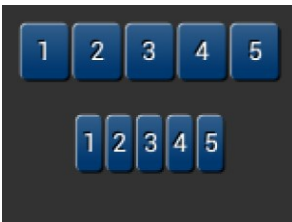
```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");
```

Without the 3D look:



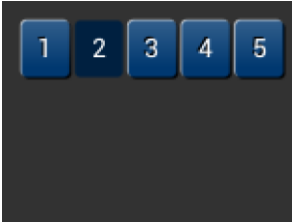
```
cmd_keys(10, 10, 140, 30, 26, OPT_FLAT, "12345");
```

Default vs. centered:



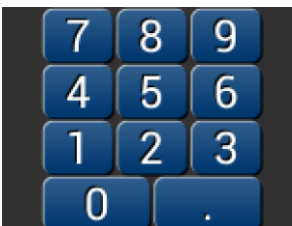
```
cmd_keys(10, 10, 140, 30, 26, 0, "12345");  
cmd_keys(10, 60, 140, 30, 26, OPT_CENTER, "12345");
```

Setting the options to show '2' key pressed ('2' is ASCII code 0x32):



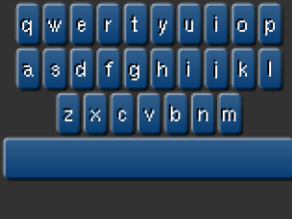
```
cmd_keys(10, 10, 140, 30, 26, 0x32, "12345");
```

A calculator-style keyboard using font 29:



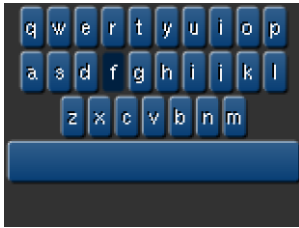
```
cmd_keys(22, 1, 116, 28, 29, 0, "789");  
cmd_keys(22, 31, 116, 28, 29, 0, "456");  
cmd_keys(22, 61, 116, 28, 29, 0, "123");  
cmd_keys(22, 91, 116, 28, 29, 0, "0.");
```

A compact keyboard drawn in font 20:



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

Showing the f (ASCII 0x66) key pressed:



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

5.46 CMD_PROGRESS

This command is used to draw a progress bar.

C prototype

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

Parameters

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

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

w
width of progress bar, in pixels

h
height of progress bar, in pixels

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

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

range
Maximum value

Description

The details of physical dimensions are--

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

Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_PROGRESS(0xFFFF FF0D)
+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);
```

5.47 CMD_SCROLLBAR

This command is used to draw a scroll bar.

C prototype

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

Parameters

x

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

y

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

w

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

h

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

options

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

val

Displayed value of scroll bar, between 0 and range inclusive

range

Maximum value

Description

Refer to [CMD_PROGRESS](#) for more information on physical dimension.

Command layout

+0	CMD_SCROLLBAR(0xFFFF FFOF)
+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);
```

5.48 CMD_SLIDER

This command is to draw a slider.

C prototype

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

Parameters

x

x-coordinate of slider top-left, in pixels

y

y-coordinate of slider top-left, in pixels

w

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

h

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

options

By default, the slider is drawn with a 3D effect. **OPT_FLAT** removes the 3D effect

val

Displayed value of slider, between 0 and **range** inclusive

range

Maximum value

Description

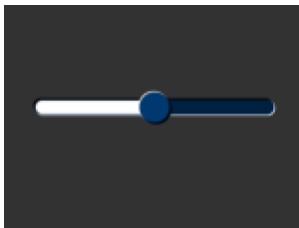
Refer to [CMD_PROGRESS](#) for more information on physical dimension.

Command layout

+0	CMD_SLIDER(0xFFFF FF0E)
+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);
```

5.49 CMD_DIAL

This command is used to draw a rotary dial control.

C prototype

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

Parameters

x
x-coordinate of dial center, in pixels

y
y-coordinate of dial center, in pixels

r
radius of dial, in pixels.

options
By default, the dial is drawn with a 3D effect and the value of options is zero. Option **OPT_FLAT** remove the 3D effect and its value is 256. Option **OPT_NOBACK**, the background is not drawn.

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 $r*(12/256)$, drawn at a distance $r*(140/256)$ to $r*(210/256)$ from the center

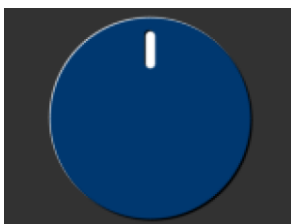
Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_DIAL(0xFFFF FF29)
+4	x
+6	y
+8	r
+10	options
+12	val

Examples

A dial set to 50%:



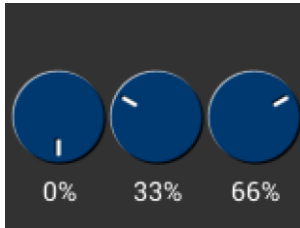
```
cmd_dial(80, 60, 55, 0, 0x8000);
```

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%");
```

5.50 CMD_TOGGLE

This command is used to draw a toggle switch with UTF-8 labels.

C prototype

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

Parameters

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

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

w
width of toggle, in pixels

font
Font to use for text, 0-63. 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. **OPT_FORMAT** processes the text as a format string, see [5.7 String Formatting](#).

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

s
string labels for toggle, UTF-8 encoding. A character value of 255 (in C it can be written as '\xff') separates the label strings. See [5.7 String Formatting](#).

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.

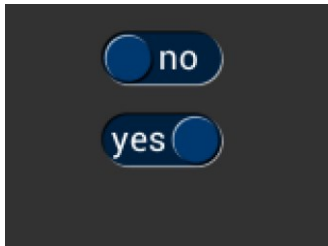
Refer to [5.2.1 Common Physical Dimensions](#) for more information.

Command layout

+0	CMD_TOGGLE(0xFFFF FF10)
+4	x
+6	y
+8	w
+10	font
+12	options
+14	state
+16	s

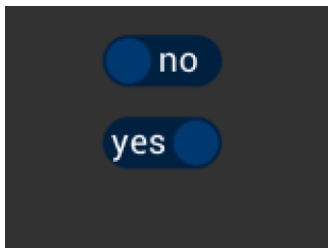
Examples

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");
```

5.51 CMD_FILLWIDTH

This command sets the pixel fill width for **CMD_TEXT** and **CMD_BUTTON** with the **OPT_FILL** option.

C prototype

```
void cmd_fillwidth( uint32_t s );
```

Parameters

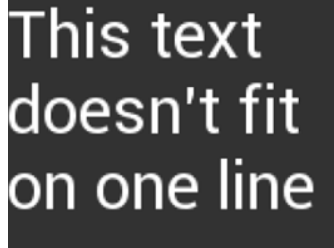
s
 line fill width, in pixels

Command layout

+0	CMD_FILLWIDTH(0xFFFF FF51)
+4	s

Examples

Long text split into lines of no more than 160 pixels:



```
cmd_fillwidth(160);
cmd_text(0, 0, 30, OPT_FILL, "This text doesn't
fit on one line");
```

5.52 CMD_TEXT

This command is used to draw a UTF-8 Text string.

C prototype

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

Parameters

x
x-coordinate of text base, in pixels

y
y-coordinate of text base, in pixels

font
Font to use for text, 0-63. 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.
OPT_FORMAT processes the text as a format string, see [5.7 String Formatting](#).
OPT_FILL breaks the text at spaces into multiple lines, with maximum width set by **CMD_FILLWIDTH**.

s
Text string, UTF-8 encoding. If **OPT_FILL** is not given then the string may contain newline (\n) characters, indicating line breaks. See [5.7 String Formatting](#).

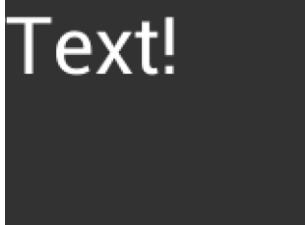
Command layout

+0	CMD_TEXT(0xFFFF FF0A)
+4	x
+6	y
+8	font

+10	options
+12	s
..	..
..	0 (null character to terminate string)

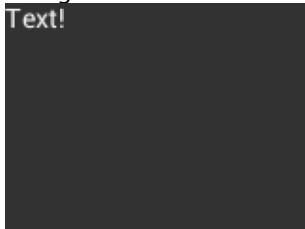
Examples

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



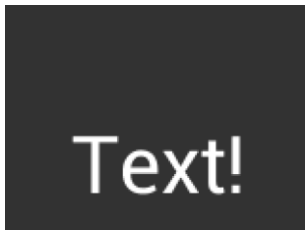
```
cmd_text(0, 0, 31, 0, "Text!");
```

Using a smaller font:
Text!



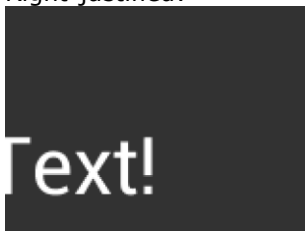
```
cmd_text(0, 0, 26, 0, "Text!");
```

Centered horizontally:



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

Right-justified:



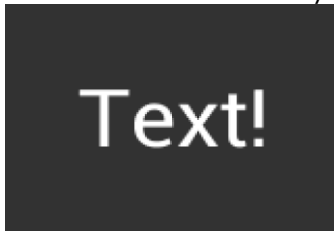
```
cmd_text(80, 60, 31, OPT_RIGHTX, "Text!");
```

Centered vertically:



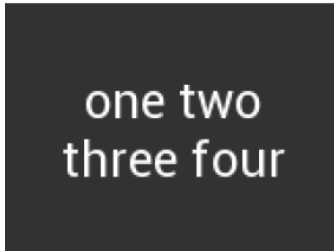
```
cmd_text(80, 60, 31, OPT_CENTERY, "Text!");
```

Centered both horizontally and vertically:



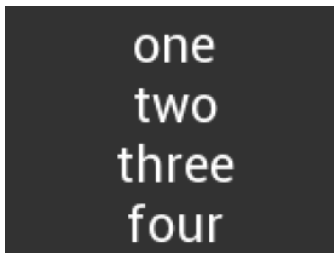
```
cmd_text(80, 60, 31, OPT_CENTER, "Text!");
```

Text with explicit newline:



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

Text split into lines and centered:



```
cmd_fillwidth(80);
cmd_text(80, 60, 29, OPT_FILL | OPT_CENTER, "one two three four");
```

5.53 CMD_TEXTDIM

Compute the size of a UTF-8 text.

C prototype

```
void cmd_text(    uint32_t dimensions,
                 int16_t font,
                 uint16_t options,
                 const char* s );
```

Parameters

dimensions

Pointer to an 8-byte result area. Must be 4-byte aligned. The dimensions are written to this area as a pair of 32-bit unsigned numbers: width then height, in pixels.

font

Font to use for text, 0-63. See [ROM and RAM Fonts](#).

options

OPT_FORMAT processes the text as a format string, see [5.7 String Formatting](#).

OPT_FILL breaks the text at spaces into multiple lines, with maximum width set by **CMD_FILLWIDTH**.

s

Text string, UTF-8 encoding. If **OPT_FILL** is not given then the string may contain newline (\n) characters, indicating line breaks. See [5.7 String Formatting](#).

Command layout

+0	CMD_TEXTDIM(0xFFFF FF84)
+4	dimensions
+8	font
+10	options
+12	s
..	..
..	0

0: (null character to terminate string)

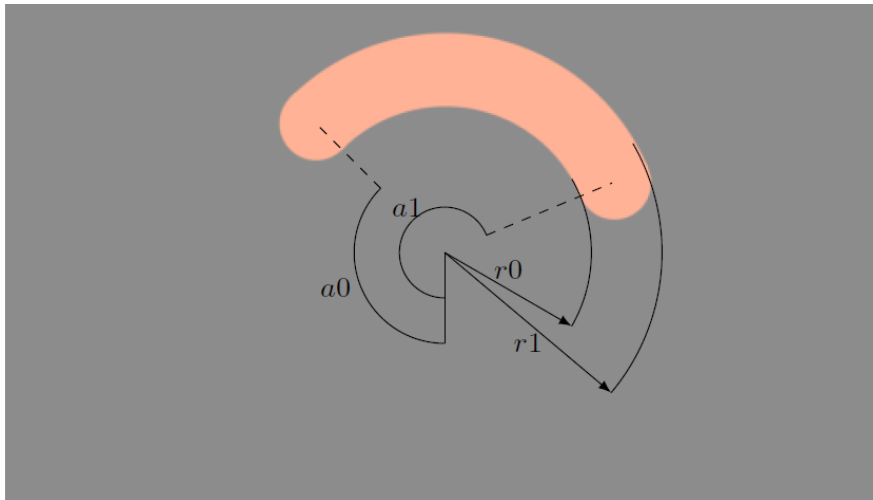
Examples

To find the dimensions of some text in font 26:

```
cmd_textdim(0x100, 26, 0, "hello world");
printf("width = %d\n", rd32(0x100));
printf("height = %d\n", rd32(0x100+4));
```

5.54 CMD_ARC

The arc command draws a circular arc. The end caps of the arc are rounded. If the angles specify a complete circle, a disc is drawn.



C prototype

```
void cmd_arc( int16_t x,
             int16_t y,
             uint16_t r0,
             uint16_t r1,
             uint16_t a0,
             uint16_t a1 );
```

Parameters

x
x-coordinate of arc center, in pixels

y
y-coordinate of arc center, in pixels

r0
inner radius, in pixels, 1-511

r1
outer radius, in pixels, 1-511

a0
arc starting angle, in furmans, 0x0000-0xffff

a1
arc ending angle, in furmans, 0x0000-0xffff.

Command layout

+0	CMD_ARC(0xFFFF FF87)
+4	x
+6	y
+8	r0
+10	r1
+12	a0
+14	a1

Examples

```
cmd_arc();
```

5.55 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

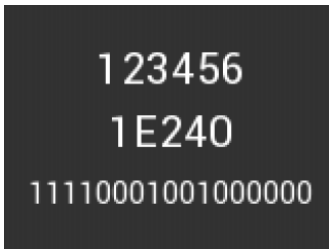
Set up numeric base for **CMD_NUMBER**.

Command layout

+0	CMD_SETBASE(0xFFFF FF33)
+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);
```

5.56 CMD_NUMBER

This command is used to draw a number.

C prototype

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

Parameters

x
x-coordinate of text base, in pixels

y
y-coordinate of text base, in pixels

font
font to use for text, 0-63. 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.

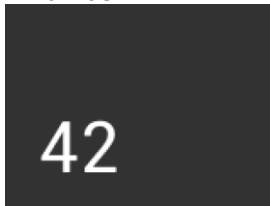
n
The number to display, is either unsigned or signed 32-bit, in the base specified in the preceding **CMD_SETBASE**. If no **CMD_SETBASE** appears before **CMD_NUMBER**, it will be in decimal base.

Command layout

+0	CMD_NUMBER(0xFFFF FF2A)
+4	x
+6	y
+8	font
+10	options
+12	n

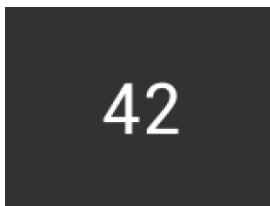
Examples

A number:



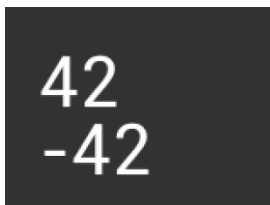
```
cmd_number(20, 60, 31, 0, 42);
```

Centered:



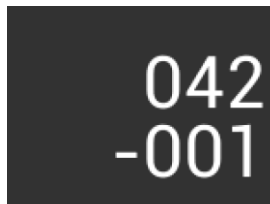
```
cmd_number(80, 60, 31, OPT_CENTER, 42);
```

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);
```

5.57 CMD_LOADIDENTITY

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

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

C prototype

```
void cmd_loadidentity( );
```

Command layout

+0	CMD_LOADIDENTITY(0xFFFF FF23)
----	--------------------------------------

5.58 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 FF27)
----	----------------------------

5.59 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 FF2F)
+4	a
+8	b
+12	c
+16	d
+20	e
+24	f

5.60 CMD_GETPTR

This command returns the first unallocated memory location.

The allocation pointer is advanced by the following commands:

- cmd_inflate
- cmd_loadasset
- cmd_loadimage
- cmd_loadwav
- cmd_playvideo
- cmd_playwav
- cmd_videoframe
- cmd_endlist

C prototype

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

Parameters

result

The first unallocated memory location.

Command layout

+0	CMD_GETPTR (0xFFFF FF20)
+4	result

Examples

```
cmd_inflate(1000, 0); //Decompress the data into RAM_G + 1000
..... //Following the zlib compressed data
While(rd32(REG_CMD_WRITE) != rd32(REG_CMD_READ)); //Wait till the
decompression was done

uint32_t x = rd32(REG_CMD_WRITE);
uint32_t ending_address = 0;
cmd_getptr(0);
ending_address = rd32(RAM_CMD + (x + 4) % 16384);
```

5.61 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

Source address of bitmap.

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 FF22)
+4	ptr
+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](#).

5.62 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

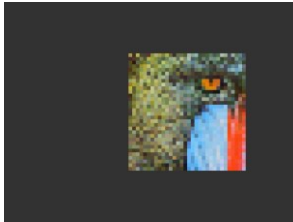
y scale factor, in signed 16.16 bit fixed-point form.

Command layout

+0	CMD_SCALE(0xFFFF FF25)
+4	sx
+8	sy

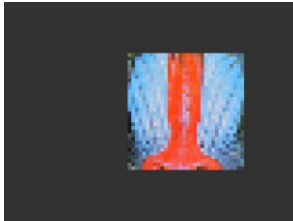
Examples

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));
```

5.63 CMD_ROTATE

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

C prototype

```
void cmd_rotate( int32_t a );
```

Parameters

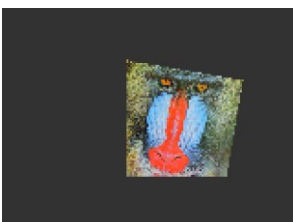
a
 Clockwise rotation angle, in units of 1/65536 of a circle

Command layout

+0	CMD_ROTATE(0xFFFF FF26)
+4	a

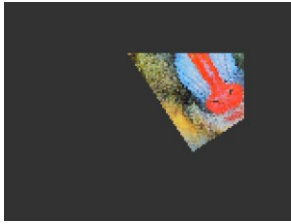
Examples

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



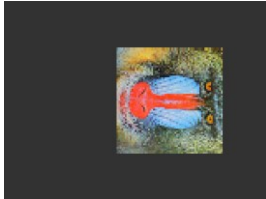
```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotate(10 * 65536 / 360);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To rotate the bitmap counterclockwise 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));
```

5.64 CMD_ROTATEAROUND

This command is used to apply a rotation and scale around a specified coordinate.

C prototype

```
void cmd_rotatearound( int32_t x,
                      int32_t y,
                      int32_t a,
                      int32_t s );
```

Parameters

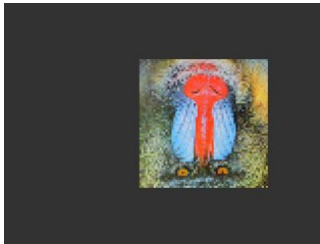
- x**
center of rotation/scaling, x-coordinate
- y**
center of rotation/scaling, y-coordinate
- a**
clockwise rotation angle, in units of 1/65536 of a circle
- s**
scale factor, in signed 16.16 bit fixed-point form

Command layout

+0	CMD_ROTATEAROUND(0xFFFF FF4B)
+4	x
+8	y
+12	a
+16	s

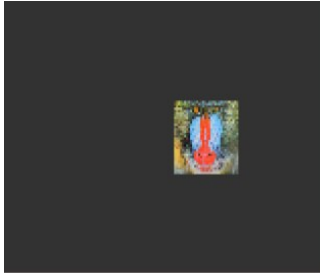
Examples

Rotating a 64 x 64 bitmap around its center:



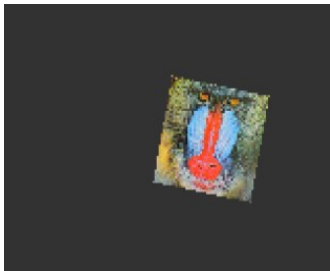
```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotatearound(32,32, 180 * 65536 /360,
65536 *1);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

To halve the bitmap size, again around the center:



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotatearound(32, 32, 0, 0.5 * 65536);
cmd_setmatrix();
cmd(VERTEX2II(68, 28, 0, 0));
```

A combined 11 degree rotation and shrink by 0.75



```
cmd(BEGIN(BITMAPS));
cmd_loadidentity();
cmd_rotatearound(32, 32,11*65536/360,0.75
* 65536);
cmd_setmatrix();
cmd(VERTEX2II(68,28,0,0));
```

5.65 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.

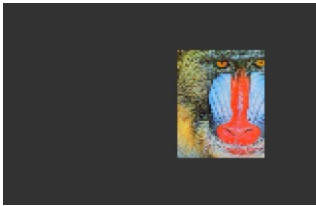
ty
y translate factor, in signed 16.16 bit fixed-point form.

Command layout

+0	CMD_TRANSLATE(0xFFFF FF24)
+4	tx
+8	ty

Examples

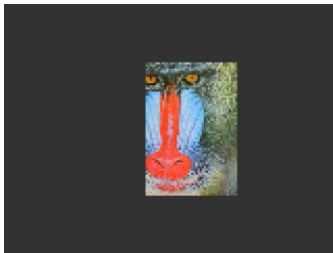
To translate the bitmap 20 pixels to the right:



```
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));
```

5.66 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 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 FF13)
+4	result

Examples

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


5.67 CMD_CALIBRATESUB

This command is used to execute the touch screen calibration routine for a sub-window. Like **CMD_CALIBRATE**, except that instead of using the whole screen area, uses a smaller sub-window specified for the command. This is intended for panels which do not use the entire defined surface.

Please note that this command only applies to compatibility mode of CTSE.

C prototype

```
void cmd_calibratesub( uint16_t x,
                      uint16_t y,
                      uint16_t w,
                      uint16_t h,
                      uint32_t result );
```

Parameters

x
x-coordinate of top-left of subwindow, in pixels.

y
y-coordinate of top-left of subwindow, in pixels.

w
width of subwindow, in pixels.

h
height of subwindow, in pixels.

result
output parameter; written with 0 on failure.

Command layout

+0	CMD_CALIBRATESUB(0xFFFF FF56)
+4	x
+6	y
+8	w
+10	h
+12	result

Examples

```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(600, 140, 31, OPT_CENTER, "Please tap on the dot");
//Calibrate a touch screen for 1200x280 screen
cmd_calibratesub(0,0, 1200,280,0);
```

5.68 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_RE_ROTATE**.

Description

CMD_SETROTATE sets **REG_RE_ROTATE** to the given value *r*, causing the screen to rotate. It also appropriately adjusts the touch transform matrix so that the coordinates of touch points are adjusted to rotated coordinate system.

Command layout

+0	CMD_SETROTATE (0xFFFF FF31)
+4	<i>r</i>

Examples

```
cmd_setrotate(2); //Put the display in portrait mode
```

5.69 CMD_SPINNER

This command is used to start an animated spinner. The spinner is an animated overlay that shows the user that some task is continuing. To trigger the spinner, create a display list and then use **CMD_SPINNER**. The 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** registers are utilized to perform the animation kind of effect. The frequency of point's movement is with respect to the display frame rate configured.

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 FF14)
+4	<i>x</i>
+6	<i>y</i>
+8	<i>style</i>
+10	<i>scale</i>

Parameters

x
 The X coordinate of top left of spinner, in pixels

y
 The Y coordinate of top left of spinner, in pixels

style

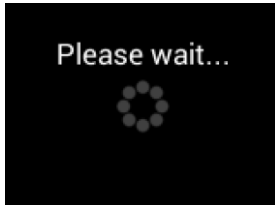
The style of spinner. Valid range is from 0 to 3.

scale

Spinner size: 0 (small), 1 (half-screen), 2 (full screen)

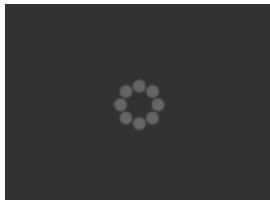
Examples

Create a display list, then start the spinner:



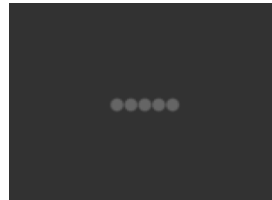
```
cmd_dlstart();
cmd(CLEAR(1,1,1));
cmd_text(80, 30, 27, OPT_CENTER, "Please
wait...");
cmd_spinner(80, 60, 0, 0);
```

Spinner style 0, a circle of dots:



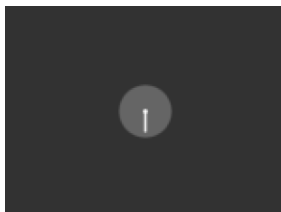
cmd_spinner(80, 60, 0, 0);

Style 1, a line of dots:



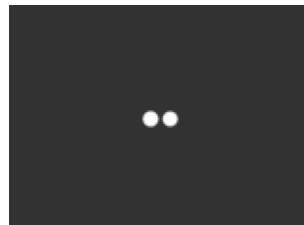
cmd_spinner(80, 60, 1, 0);

Style 2, a rotating clock hand:



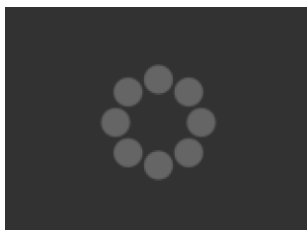
cmd_spinner(80, 60, 2, 0);

Style 3, two orbiting dots:



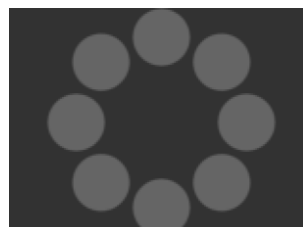
cmd_spinner(80, 60, 3, 0);

Half screen, scale:



cmd_spinner(80, 60, 0, 1);

Full screen, scale 2:



cmd_spinner(80, 60, 0, 2);

5.70 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. With an appropriate display list, this causes a bitmap to move around the screen without any MCU work. Command **CMD_STOP** stops the update process. Note that only one of **CMD_SKETCH**, **CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

See [VERTEX_FORMAT](#) and [VERTEX2F](#) for varying (x, y) coordinates and the pixel precision setting.

C prototype

```
void cmd_screensaver( );
```

Description

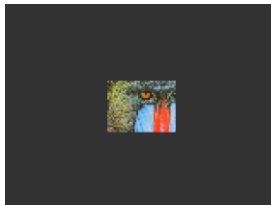
REG_MACRO_0 is updated with respect to frame rate (depending on the display registers configuration).

Command layout

+0	CMD_SCREENSAVER(0xFFFF FF2B)
----	-------------------------------------

Examples

To start the screensaver, create a display list using a MACRO instruction – the coprocessor engine will update it continuously:



```
cmd_screensaver();
cmd(BITMAP_HANDLE(0));
cmd_setbitmap(0, RGB565, 40, 30);
cmd(BEGIN(BITMAPS));
cmd(MACRO(0));
cmd(DISPLAY());
```

5.71 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 means that the user touch inputs are drawn into the bitmap without any need for MCU work. **CMD_STOP** is to be sent to stop the sketch process.

Note that only one of **CMD_SKETCH**, **CMD_SCREENSAVER**, or **CMD_SPINNER** can be active at one time.

C prototype

```
void cmd_sketch( int16_t x,
                int16_t y,
                uint16_t w,
                uint16_t h,
                uint32_t ptr,
```

uint16_t format);

Parameters

x
 x-coordinate of sketch area top-left, in pixels

y
 y-coordinate of sketch area top-left, in pixels

w
 Width of sketch area, in pixels

h
 Height of sketch area, in pixels

ptr
 Base address of sketch bitmap

format
 Format of sketch bitmap, either L1 or L8

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 FF2C)
+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_HANDLE(0));
cmd_setbitmap(0, L1, 480, 272);
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(0, 0, 0, 0));

//Finally, to stop sketch updates
cmd_stop();
```

5.72 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 FF15)
----	------------------------------

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](#).

5.73 CMD_SETFONT

This **CMD_SETFONT** is equivalent to **CMD_SETFONT2** in previous generations and 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_SETFONT**, as described in [ROM and RAM Fonts](#).

Note that **CMD_SETFONT** 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_setfont ( uint32_t font,
                  uint32_t ptr,
                  uint32_t firstchar );
```

Command layout

+0	CMD_SETFONT(0xFFFF FF36)
+4	font
+8	ptr
+12	firstchar

Parameters

font

The bitmap handle from 0 to 63

ptr

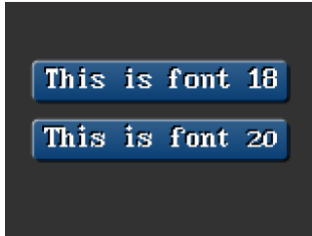
32 bits aligned memory address in **RAM_G** of font metrics block

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_setfont(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");
```

5.74 CMD_SETSCRATCH

This command is used to set the scratch bitmap for widget use. Graphical objects use a bitmap handle for rendering. By default, this is bitmap handle 15. This command allows it to be set to any bitmap handle. This command enables the user to utilize bitmap handle 15 safely.

C prototype

```
void cmd_setscratch( uint32_t handle);
```

Parameters

handle
 bitmap handle number, 0 to 63

Command layout

+0	CMD_SETSCRATCH (0xFFFF FF37)
+4	handle

Examples

With the setscratch command, set the handle 31, handle 15 is available for application use, for example as a font:



```
cmd_setscratch(31);
cmd_setfont(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);
```

5.75 CMD_ROMFONT

This command is to load a ROM font into bitmap handle. By default, ROM fonts 16-31 are loaded into bitmap handles 16-34. 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 to 63

romslot
 ROM font number, 16 to 34

Command layout

+0	CMD_ROMFONT (0xFFFF FF39)
+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");
```

5.76 CMD_RESETFONTS

This command loads bitmap handles 16-34 with their default fonts.

C prototype

```
void cmd_resetfonts();
```

Parameters

NA

Command layout

+0	CMD_RESETFONTS (0xFFFF FF4C)
-----------	-------------------------------------

Examples

NA

5.77 CMD_TRACK

This command is used to track touches for a graphics object. **EVE** can assist the **MCU** in tracking touches on graphical objects. For example, touches on dial objects can be reported as angles, saving MCU computation. To do this the MCU 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 MCU can detect any touch on the object by reading the 32-bit value in the five registers above. The lower 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.

C prototype

```
void cmd_track(  int16_t x,
                int16_t y,
                int16_t w,
                int16_t h,
                int16_t tag );
```

Parameters

x
 For linear tracker functionality, x-coordinate of track area top-left, in pixels.
 For rotary tracker functionality, x-coordinate of track area center, in pixels.

y
 For linear tracker functionality, y-coordinate of track area top-left, in pixels.
 For rotary tracker functionality, y-coordinate of track area center, in pixels.

w
 idth 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

Note: The valid tag value range for graphics objects is 1-16,777,215 but only tag value 1-255 can be used by **CMD_TRACK**.

Command layout

+0	CMD_TRACK(0xFFFF FF28)
+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:

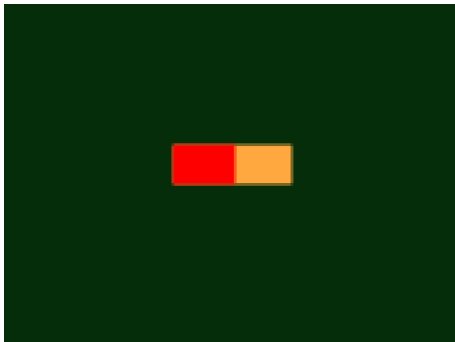
- rotary tracker mode – Track the angle between the touch point and the center of the graphics object specified by the tag value. The value is in units of 1/65536 of a circle. 0 means that the angle is straight down, 0x4000 left, 0x8000 up, and 0xC000 right from the center.
- Linear tracker mode – If 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.

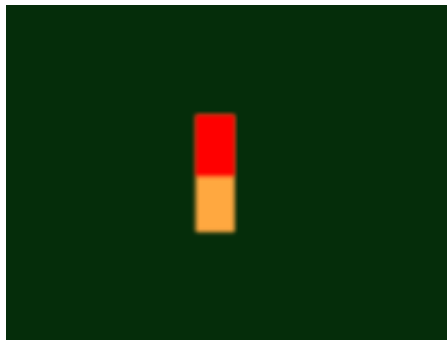
Examples

Horizontal track of rectangle dimension 40x12 pixels and the present touch is at 50%:



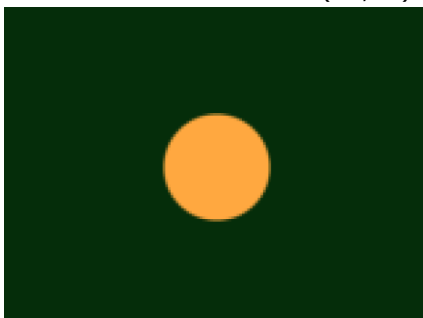
```
d1( CLEAR_COLOR_RGB(5, 45, 110) );
d1( COLOR_RGB(255, 168, 64) );
d1( CLEAR(1, 1, 1) );
d1( BEGIN(RECTS) );
d1( VERTEX2F(60 * 16, 50 * 16) );
d1( VERTEX2F(100 * 16, 62 * 16) );
d1( COLOR_RGB(255, 0, 0) );
d1( VERTEX2F(60 * 16, 50 * 16) );
d1( VERTEX2F(80 * 16, 62 * 16) );
d1( COLOR_MASK(0, 0, 0, 0) );
d1( TAG(1) );
d1( VERTEX2F(60 * 16, 50 * 16) );
d1( VERTEX2F(100 * 16, 62 * 16) );
cmd_track(60, 50, 40, 12, 1);
```

Vertical track of rectangle dimension 12x40 pixels and the present touch is at 50%:



```
d1( CLEAR_COLOR_RGB(5, 45, 110) );
d1( COLOR_RGB(255, 168, 64) );
d1( CLEAR(1, 1, 1) );
d1( BEGIN(RECTS) );
d1( VERTEX2F(70 * 16, 40 * 16) );
d1( VERTEX2F(82 * 16, 80 * 16) );
d1( COLOR_RGB(255, 0, 0) );
d1( VERTEX2F(70 * 16, 40 * 16) );
d1( VERTEX2F(82 * 16, 60 * 16) );
d1( COLOR_MASK(0, 0, 0, 0) );
d1( TAG(1) );
d1( VERTEX2F(70 * 16, 40 * 16) );
d1( VERTEX2F(82 * 16, 80 * 16) );
cmd_track(70, 40, 12, 40, 1);
```

Circular track centered at (80,60) display location:



```
d1( CLEAR_COLOR_RGB(5, 45, 110) );
d1( COLOR_RGB(255, 168, 64) );
d1( CLEAR(1, 1, 1) );
d1( TAG(1) );
d1( BEGIN(POINTS) );
d1( POINT_SIZE(20 * 16) );
d1( VERTEX2F(80 * 16, 60 * 16) );
cmd_track(80, 60, 1, 1, 1);
```

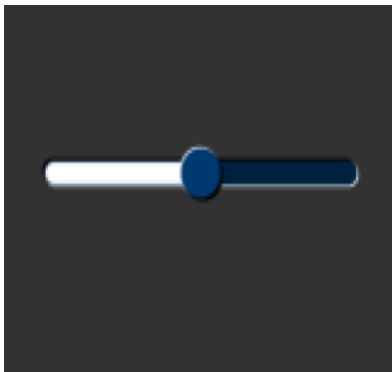
To draw a dial with tag 33 centered at (80, 60), adjustable by touch:



```
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;
}
```

5.78 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_SOURCEH**
- **BITMAP_LAYOUT/ BITMAP_LAYOUT_H**
- **BITMAP_SIZE/ BITMAP_SIZE_H**
- **BITMAP_EXT_FORMAT**

The parameters filter/wrapx/wrapy in **BITMAP_SIZE** are 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**, as a **BITMAP_SOURCE** parameter.

fmt

Bitmap format, see the definition in **BITMAP_EXT_FORMAT**.

width

bitmap width, in pixels. 2 bytes value.

height

bitmap height, in pixels. 2 bytes value.

Command layout

+0	CMD_SETBITMAP(0xFFFF FF3D)
+4	source
+8	fmt
+10	width
+12	height

Examples

Display an ASTC image with width 35 and height 35 pixels residing in **RAM_G** address 1024:

```
cmd_dlstart();
cmd_setbitmap(1024, COMPRESSED_RGBA_ASTC_5x5_KHR, 35, 35);
cmd(CLEAR(1, 1, 1));
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(10, 10, 0, 0));
```

Note:

Two bytes need to be appended after the last parameter for 4 bytes alignment. When format is **PALETEDARGB8**, due to no display list commands **PALETTE_SOURCE** and **PALETTE_SOURCEH** are generated, user need to write the **PALETTE_SOURCE** and **PALETTE_SOURCEH** commands manually.

5.79 CMD_RENDERTARGET

The rendertarget command sets the following registers:

- **REG_RE_DEST**
- **REG_RE_FORMAT**
- **REG_RE_W**
- **REG_RE_H**

It also raises an exception if the render target parameters are invalid. If width and height are zero, then the render target is set to a dummy area, so rendering has no effect. This is useful when using a display list that initializes bitmap parameters.

C prototype

```
void cmd_rendertarget( uint32_t dst,
                     uint16_t fmt,
                     int16_t w,
                     int16_t h );
```

Parameters

dst

Render target start address, either SWAPCHAIN 0 or a 128-byte aligned memory.

fmt

Render target format, one of **L8, LA8, RGB565, RGB6, RGB8, ARGB4, ARGB1555, ARGB6, ARGB8, YCBCR**.

w

Render target width in pixels and must be a multiple of 16.

h

Render target height in pixels. $w \times h$ must be a multiple of 128.

Command layout

+0	CMD_RENDERTARGET(0xFFFF FF8D)
+4	dst
+8	fmt
+10	w
+12	h

Examples

To set the render target for a 2K screen starting at 0x1000:

```
cmd_rendertarget(0x1000, RGB8, 1920, 1200);
```

To setup multiple bitmaps at application start:

```
cmd_rendertarget(0, 0, 0, 0);
cmd(BITMAP_HANDLE(0));
cmd_setbitmap(0, RGB565, 64, 64);
cmd(BITMAP_HANDLE(1));
cmd_setbitmap(0x1000, YCBCR, 512, 512);
...
cmd(DISPLAY());
cmd_swap();
cmd_dlstart();
```

5.80 CMD_LOGO



The logo command causes the coprocessor engine to play back a short animation of the Bridgetek logo. During logo playback the MCU 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 MCU shall write the next command to the starting address of **RAM_CMD**.

C prototype

```
void cmd_logo( );
```

Command layout

+0	CMD_LOGO(0xFFFF FF2D)
----	------------------------------

Examples

To play back the logo animation:

```
cmd_logo();
delay(3000); // Optional to wait
//Wait till both read & write pointer register are equal.
While(rd32(REG_CMD_WRITE) != rd32(REG_CMD_READ));
```

5.81 CMD_ENABLEREGION

The default all widgets use the **REGION** instruction to optimize rendering. The enableregion command disables and enables these optimizations.

C prototype

```
void cmd_enableregion( uint32_t en );
```

Parameters

en
enable flag for all **REGION** optimizations.

Command layout

+0	CMD_ENABLEREGION (0xFFFF FF7E)
+4	en

Examples

To disable optimizations:

```
cmd_enableregion(false);
```

5.82 CMD_FENCE

This command is used to wait for outstanding writes. The fence command waits until all preceding outstanding memory writes are completed. It guarantees that any subsequent memory read will use the written value.

C prototype

```
void cmd_fence( );
```

Command layout

+0	CMD_FENCE (0xFFFF FF68)
-----------	--------------------------------

Examples

```
cmd(CLEAR_COLOR_RGB(55, 55, 55));
cmd(CLEAR(1, 1, 1));
cmd_loadimage(0, 0);
... // JPEG file data follows
cmd_fence(); // This command make sure all outstanding
// writes have completed

cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(4, 4, 0, 0)); // draw bitmap at (4,4)
cmd(DISPLAY()); // end of display list
cmd_swap(); // display the image
```

5.83 CMD_GRAPHICSFINISH

The command waits until the render engine is idle.

C prototype

```
void cmd_graphicsfinish( );
```

Command layout

+0	CMD_GRAPHICSFINISH (0xFFFF FF6B)
----	---

Examples

To accurately measure the cycles taken to draw a frame:

```
cmd_memcpy(0, REG_CLOCK, 4);
cmd_swap();
cmd_graphicsfinish();
cmd_memcpy(4, REG_CLOCK, 4);
//The difference between the values at 0x0 and 0x4 is the time taken.
```

5.84 CMD_FLASHERASE

This command erases the attached flash storage.

C prototype

```
void cmd_flasherase( );
```

Command layout

+0	CMD_FLASHERASE(0xFFFF FF3E)
----	------------------------------------

Examples

NA

5.85 CMD_FLASHWRITE

This command writes the following inline data to flash storage. The storage should have been previously erased using **CMD_FLASHERASE**.

C prototype

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

Parameters

ptr

Destination address in flash memory. Must be 256-byte aligned. Start address of first block is from **zero**.

num

Number of bytes to write, must be multiple of 256

Command layout

+0	CMD_FLASHWRITE (0xFFFF FF3F)
+4	ptr
+8	num
+12...n	bytes ₁ ...byte _n

Examples

NA

5.86 CMD_FLASHPROGRAM

This command writes the data to a blank flash. It assumes that the flash is previously programmed to all-ones, which is the default state of flash chip by manufacturers.

C prototype

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

Parameters

dst

destination address in flash memory. Must be 4096-byte aligned. Start address of first block is from **zero**.

src

source data in main memory. Must be 4-byte aligned

num

number of bytes to write, must be multiple of 4096

Command layout

+0	CMD_FLASHPROGRAM (0xFFFF FF64)
+4	dst
+8	src
+12	num

Examples

NA

5.87 CMD_FLASHREAD

This command reads data from flash into main memory.

C prototype

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

Parameters

dest

Destination address in **RAM_G**. Must be 4-byte aligned. Start address of first block is from **zero**.

src

source address in flash memory. Must be 64-byte aligned.

num

number of bytes to write, must be multiple of 4

Command layout

+0	CMD_FLASHREAD(0xFFFF FF40)
+4	dest
+8	src
+12	num

Examples

```
// Read all main RAM (1M bytes) from flash:
cmd_flashread(0, 4096, 1048576);
```

5.88 CMD_APPENDF

This command appends data from flash to the next available location in display list memory **RAM_DL**, which was specified by **REG_CMD_WRITE**.

C prototype

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

Parameters

ptr

start of source commands in flash memory. Must be 64-byte aligned. Start address of first block is from **zero**.

num

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

Command layout

+0	CMD_APPENDF (0xFFFF FF52)
+4	ptr
+8	num

5.89 CMD_FLASHUPDATE

This command writes the given data to flash. If the data matches the existing contents of flash, nothing is done. Otherwise, the flash is erased in 4K units, and the data is written.

Note that this command is available for NOR flash only.

C prototype

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

Parameters

dest

Destination address in flash memory. Must be 4096-byte aligned. Start address of first block is from **zero**.

src

source address in main memory **RAM G**. Must be 4-byte aligned.

num
 number of bytes to write, must be multiple of 4096

Command layout

+0	CMD_FLASHUPDATE (0xFFFF FF41)
+4	dest
+8	src
+12	num

Example

```

// The pseudo code below shows how to program the blob file to first
// block of flash

// Assume the flash is in detach mode and now attach it
cmd_flashattach();

// Now check if the flash is in basic mode after attaching
while (FLASH_STATUS_BASIC != rd32(REG_FLASH_STATUS));

//Write the BLOB file into the first block of flash
//Assume the BLOB file is in RAM_G
cmd_flashupdate(0, RAM_G, 4096);

// To check if the blob is valid , try to switch to full mode
cmd_flashfast(0);
while (FLASH_STATUS_BASIC != rd32(REG_FLASH_FULL));

```

5.90 CMD_FLASHDETACH

This command puts the SPI signals of flash device into hi-Z state. The only valid flash operations when detached are the low-level SPI access commands as follows:

- **CMD_FLASHSPIDSEL**
- **CMD_FLASHSPITX**
- **CMD_FLASHSPIRX**
- **CMD_FLASHATTACH**

Refer to the section - **Flash interface** in [BT82X datasheet](#) for details.

C prototype

```
void cmd_flashdetach( );
```

Command layout

+0	CMD_FLASHDETACH (0xFFFF FF42)
-----------	--------------------------------------

5.91 CMD_FLASHATTACH

This command re-connects to the attached SPI flash storage. After the command, register **REG_FLASH_STATUS** should be **FLASH_STATUS_BASIC**.

Refer to the section - **Flash interface** in [BT82X datasheet](#) for details.

C prototype

```
void cmd_flashattach( );
```

Command layout

+0	CMD_FLASHATTACH (0xFFFF FF43)
----	--------------------------------------

5.92 CMD_FLASHFAST

This command drives the attached flash in full-speed mode, if possible. Refer to the section - **Flash interface** in [BT82X datasheet](#) for details.

C prototype

```
void cmd_flashfast ( uint32_t result );
```

Parameters

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise, an error code is set as follows:

Error Code	Meaning
0xE001	flash is not supported
0xE002	no header detected in sector 0 – is flash blank?
0xE003	sector 0 data failed integrity check
0xE004	device/blob mismatch – was correct blob loaded?
0xE005	failed full-speed test – check board wiring

Command layout

+0	CMD_FLASHFAST (0xFFFF FF44)
+4	result

Note: To access any data in flash, host needs to send this command at least once in order to drive flash in full-speed mode. In addition, the flash chip is assumed to have the correct blob file programmed in its first block (4096 bytes). Otherwise, it will cause the failure of this command.

To program the blob into the first block, flash shall be attached and enter into BASIC mode. The code is as below:

```

/*
  Assume the Blob file (4096 bytes) is at RAM_G address
  "blob_addr" and the flash is in detach mode
*/

//Attach the flash
cmd_flashattach();

// Now check if the flash is in basic mode after attaching
while (FLASH_STATUS_BASIC != rd32(REG_FLASH_STATUS));

//program the blob(4096 bytes) to the offset zero (first block of
flash)
cmd_flashupdate(0, blob_addr, 4096);

```

Example

NA

5.93 CMD_FLASHSPIDESEL

This command de-asserts the SPI CS signal. It is only valid when the flash has been detached, using **CMD_FLASHDETACH**.

C prototype

```
void cmd_flashspidesel ( );
```

Command layout

+0	CMD_FLASHSPIDESEL (0xFFFFF45)
-----------	--------------------------------------

Parameters

NA

5.94 CMD_FLASHSPITX

This command transmits the following bytes over the flash SPI interface. It is only valid when the flash has been detached, using **CMD_FLASHDETACH** and the outputs enabled using **CMD_FLASHSPIDESEL**.

C prototype

```
void cmd_flashspitx ( uint32_t num );
```

Parameters

num
 number of bytes to transmit

Command layout

+0	CMD_FLASHSPITX (0xFFFF FF46)
+4	num
byte1...byten	the data to transmit

Example

Transmit single-byte 06:

```
cmd_flashdetach();
cmd_flashspidesel();
cmd_flashspitx(1);
cmd(0x00000006);
```

5.95 CMD_FLASHSPIRX

This command receives bytes from the flash SPI interface and writes them to main memory. It is only valid when the flash has been detached, using **CMD_FLASHDETACH** and the outputs enabled using **CMD_FLASHSPIDESEL**.

C prototype

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

Parameters

ptr
destination address in **RAM_G**

num
number of bytes to receive

Command layout

+0	CMD_FLASHSPIRX (0xFFFF FF47)
+4	ptr
+4	num

Example

Read 3 bytes from SPI flash to main memory locations 100,101,102:

```
cmd_flashdetach();
cmd_flashspidesel();
cmd_flashspirx(100, 3);
```

5.96 CMD_FLASHSOURCE

This command specifies the source address for flash data loaded by the **CMD_LOADIMAGE**, **CMD_PLAYVIDEO**, **CMD_VIDEOSTART** and **CMD_INFLATE** commands with the **OPT_FLASH** option.

C prototype

```
void cmd_flashsource ( uint32_t ptr );
```

Parameters

ptr
flash address, must be 64-byte aligned. Start address of first block is from **zero**.

Command layout

+0	CMD_FLASHSOURCE (0xFFFF FF48)
+4	ptr

5.97 CMD_SDATTACH

This command causes the EVE to connect to the attached SD storage. After the command, result code can be queried to confirm the SD system status.

EVE supports both SDCard and eMMC storage. By default, **CMD_SDATTACH** probes for eMMC first, then probes for SDCard. This may be overridden with options **OPT_IS_SD** or **OPT_IS_MMC**.

Note that only one of **OPT_HALFSPEED** and **OPT_QUARTERSPEED** can be specified.

C prototype

```
void cmd_sdattach( uint32_t options
                  uint32_t result );
```

Parameters

options

A bitwise-or of the following flags:

OPT_4BIT causes the SD interface to be initialized in 4-bit mode. The default is 1-bit mode.

OPT_HALFSPEED causes the SD card to run at half-speed (one quarter of the system clock)

OPT_QUARTERSPEED causes the SD card to run at quarter-speed (one eighth of the system clock)

OPT_IS_SD assume the attached storage is an SDCard

OPT_IS_MMC assume the attached storage is eMMC

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise, an error code is set as follows:

0xd000: SD card is not detected. Check card detect line

0xd001: no response from card. Probable signal or card problem

0xd002: card timeout during initialization. Probable problem with card

0xd003: sector 0 (MBR) not found on card. Is the card blank?

0xd004: FAT filesystem not found. Is the card properly formatted with a FAT filesystem?

0xd005: Card failed to enter high-speed mode

Command layout

+0	CMD_SDATTACH (0xFFFF FF6E)
+4	options
+8	result

Examples

```
cmd_sdattach(OPT_4BIT | OPT_IS_SD, 0);
```

5.98 CMD_FSOPTIONS

This command configures options affecting the behaviour of the FAT subsystem.

C prototype

```
void cmd_fsoption( uint32_t options );
```

Parameters

options

options bitfield, a union of

- **OPT_SFNLLOWER** if enabled, FAT32 short filenames are mapped to lower case, in accordance with the Microsoft de-facto standard.
- **OPT_CASESENSITIVE** if enabled, filename comparison is case sensitive.
- **OPT_DIRSEP_WIN** treat backslash as a directory path separator.
- **OPT_DIRSEP_UNIX** treat forward slash as a directory path separator.

The default options are: **OPT_SFNLLOWER**, **OPT_DIRSEP_WIN** and **OPT_DIRSEP_UNIX**

Command layout

+0	CMD_FSOPTION (0xFFFF FF6D)
+4	options

Examples

NA

5.99 CMD_FSREAD

This command reads the named file directly into **RAM_G**. Errors are indicated by the result field and should be checked by the host.

C prototype

```
void cmd_fsread(uint32_t dst,
                const char* filename,
                uint32_t result );
```

Parameters

dst

destination address in RAM, 32 bytes aligned

filename

complete filename of the file, as a UTF-8 encoded NUL terminated string, padded to a 4-byte boundary

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise an error code is set as follows:

- 2 (ENOENT) file not found
- 5 (EIO) IO error

Command layout

+0	CMD_FSREAD (0xFFFF FF71)
+4	dst
+8	filename ... 0
+n	result

Examples

NA

5.100 CMD_FSSIZE

This command returns the size of the named file, in bytes. If the file is not found, the returned size is set to 0xffffffff.

C prototype

```
void cmd_fssize(const char* filename,
               uint32_t size );
```

Parameters

filename

complete filename of the file, as a UTF-8 encoded NUL terminated string, padded to a 4-byte boundary

size

the size of the named file

Command layout

+0	CMD_FSSIZE (0xFFFF FF80)
+4	filename ... 0
+n	size

Examples

To find the size of a JPEG file:

```
cmd_fssize("mandrill.jpg", 0);
```

5.101 CMD_FSSOURCE

Set source file for a future load. It sets the source file for use with a data command with the **OPT_FS** option.

C prototype

```
void cmd_fssource(const char* filename,
                 uint32_t result );
```

Parameters

filename

complete filename of the file, as a UTF-8 encoded NUL terminated string, padded to a 4-byte boundary

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise an error code is set as follows:

- 2 (ENOENT) file not found
- 5 (EIO) IO error

Command layout

+0	CMD_FSSOURCE (0xFFFF FF7F)
+4	filename ... 0
+n	result

Examples

```
To do something
cmd_fssource("video.avi", 77);
```

5.102 CMD_FSDIR

This command writes a list of the files in an SDcard directory to memory. The order of files in the list is undefined. If the list of files does not fit into the given memory buffer, **ENOMEM** is returned. The format of the directory listing is consecutive UTF-8 strings, each NUL-terminated. The end of the list is indicated by a zero-length string.

C prototype

```
void cmd_fsdir( uint32_t dst,
                uint32_t num,
                const char* path,
                uint32_t result );
```

Parameters

dst

Start of buffer

num

Size of buffer, in bytes

path

Path of directory to list. Must end with a directory separator character

result

Written with the result code. If the command succeeds, zero is written as a result. Otherwise an error code is set as follows:

- 2 (ENOENT) file not found
- 5 (EIO) IO error
- 12 (ENOMEM) buffer not large enough for data

Command layout

+0	CMD_FSDIR (0xFFFF FF8E)
+4	dst
+8	num
+12	path ... 0
+n	result

Examples

```

//To retrieve the files in the main directory, and write the
result to a 4Kbyte buffer starting at 0x0:

cmd_fsdir(0, 4096, "/", 1);

//And to print the strings in the return buffer:

{
    uint8_t buf[4096], pc;
    pc = buf;
    while (pc[0] != 0x0) {
        printf("--> %s\n", pc);
        pc += strlen(pc) + 1;
    }
}

```

5.103 CMD_SDBLOCKREAD

The sdblockread command reads multiple 512-byte blocks from SD into main memory.

C prototype

```

void cmd_sdblockread(    uint32_t dst,
                        uint32_t src,
                        uint32_t count,
                        uint32_t result );

```

Parameters

- dst**
Destination in main memory, must be 4-byte aligned

- src**
Source block address

- num**
Number of 512-byte blocks to read

- result**
Result code, 0 for success

Command layout

+0	CMD_SDBLOCKREAD (0xFFFF FF6F)
+4	dst
+8	src
+12	count
+16	result

Examples

```
To read a single block at address x into memory at 0x100:
cmd_sdblockread(0x100, x, 1, 0);
```

5.104 CMD_DDRSHUTDOWN

This command will perform essential maintenance duties and deactivate the DDR interface. It's a crucial procedure preceding the transition into the SLEEP state. The DDR memory will enter self-refresh mode, ensuring no data loss upon resumption.

C prototype

```
void cmd_ddrshutdown( );
```

Command layout

+0	CMD_DDRSHUTDOWN (0xFFFF FF65)
-----------	--------------------------------------

Examples

```
cmd_ddrshutdown(); // shutdown DDR operations
... issue host command to sleep
...
... issue host command to wake up
cmd_ddrstartup(); // restart DDR operations
```

5.105 CMD_DDRSTARTUP

Startup DDR system. This command is issued to **EVE** to bring DDR out of DDR SLEEP state.

C prototype

```
void cmd_ddrstartup( );
```

Command layout

+0	CMD_DDRSTARTUP (0xFFFF FF66)
-----------	-------------------------------------

Examples

```
cmd_ddrshutdown(); // shutdown DDR operations
... issue shutdown commands
...
... issue wakeup commands
cmd_ddrstartup(); // restart DDR operations
```

5.106 CMD_ANIMSTART

This command is used to start an animation in **RAM_G**. If the channel was previously in use, the previous animation would be replaced. The animation object is in **RAM_G**.

C prototype

```
void cmd_animstartram( int32_t ch,
                      uint32_t aoptr,
                      uint32_t loop );
```

Parameters

ch

Animation channel, 0-31. If no channel is available, then an "out of channels" exception is raised.

aoptr

Pointer to the animation object in **RAM_G**. Must be 64-byte aligned.

loop

Loop flags.

Name	Value	Description
ANIM_ONCE	0	plays the animation once, then cancels it
ANIM_LOOP	1	plays the animation in a loop
ANIM_HOLD	2	plays the animation once, then displays the final frame

Command layout

+0	CMD_ANIMSTART(0xFFFF FF5F)
+4	ch
+8	aoptr
+12	loop

Example

NA

5.107 CMD_RUNANIM

This command is used to play/run animations until complete. Playback ends when either a specified animation completes, or when host **MCU** writes to a control byte. Note that only animations started with **ANIM_ONCE** complete. Pseudocode for **CMD_RUNANIM** is:

```
do {
  if ((play != -1) && (*play != 0))
    break;
  CMD_DLSTART();
  Clear(1,1,1);
  CMD_ANIMDRAW(-1);
  CMD_SWAP();
} while ((waitmask & REG_ANIM_ACTIVE) == 0);
```

C prototype

```
void cmd_runram( uint32_t waitmask,
                 uint32_t play );
```

Parameters

waitmask

32-bit mask specifying which animation channels to wait for. Animation ends when the

logical **AND** of this mask and **REG_ANIM_ACTIVE** is zero.

play

Address of play control byte. Animation stops when the byte at play is not zero. If this feature is not required, the special value of -1 (0xFFFF FFFF) means that there is no control byte.

Command layout

+0	CMD_RUNANIM(0xFFFF FF60)
+4	waitmask
+8	play

Example

```

/****
play back several animations simultaneously
assume the animation is in flash
****/

/*
set up a channel for first animation
*/
cmd_animstart(1,4096, ANIM_ONCE);
cmd_animxy(400, 240); //The center of animation

/*
set up another channel for second animation
*/
cmd_animstart(2,4096 + 10*1024, ANIM_ONCE);
cmd_animxy(400, 240); //The center of animation

/*
play back both animations and set up the control byte at 0xF0000 of
RAM_G
*/
wr32(0xF0000, 1);
cmd_runanim(-1, 0xF0000); //The animation will be shown on display.

//.....

/*
To stop the animation before it ends , write the contro byte to zero
*/
wr32(0xF0000, 0);

```

5.108 CMD_ANIMSTOP

This command stops one or more active animations.

C prototype

```
void cmd_animstop( int32_t ch );
```

Parameters

ch

Animation channel, 0-31. If ch is -1, then all animations are stopped.

Command layout

+0	CMD_ANIMSTOP (0xFFFF FF4D)
+4	ch

5.109 CMD_ANIMXY

This command sets the coordinates of an animation.

C prototype

```
void cmd_animxy ( int32_t ch,  
                 int16_t x,  
                 int16_t y );
```

Parameters

ch

Animation channel, 0-31.

x

x screen coordinate for the animation center, in pixels

y

y screen coordinate for the animation center, in pixels

Command layout

+0	CMD_ANIMXY (0xFFFF FF4E)
+4	ch
+8	x
+10	y

NOTE: If the pixel precision is not set to 1/16 in current graphics context, a **VERTEX_FOMART(4)** is mandatory to precede this command.

5.110 CMD_ANIMDRAW

This command draws one or more active animations

C prototype

```
void cmd_animdraw ( int32_t ch );
```

Parameters

ch

Animation channel, 0-31. If ch is -1, then it draws all undrawn animations in ascending order.

Command layout

+0	CMD_ANIMDRAW(0xFFFF FF4F)
+4	ch

5.111 CMD_ANIMFRAME

This command draws the specified frame of an animation

C prototype

```
void cmd_animframe ( int16_t x,
                    int16_t y,
                    uint32_t aoptr,
                    uint32_t frame );
```

Parameters

x

x screen coordinate for the animation center, in pixels.

y

y screen coordinate for the animation center, in pixels.

aoptr

The address of the animation object in **RAM_G**. Must be 64-byte aligned

frame

Frame number to draw, starting from zero.

Command layout

+0	CMD_ANIMFRAME (0xFFFF FF5E)
+4	x
+6	y
+8	aoptr
+12	frame

Note: If the pixel precision is not set to 1/16 in current graphics context, a **VERTEX_FORMAT(4)** is mandatory to precede this command.

Example

```
//Draw a frame located at 4096 (RAM_G).
cmd_animframe(0, 400, 4096, 65);
```

5.112 CMD_SYNC

This command waits for the end of the video scanout 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 FF3C)
-----------	------------------------------

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.
```

5.113 CMD_BITMAP_TRANSFORM

This command computes a bitmap transform and appends commands **BITMAP_TRANSFORM_A – BITMAP_TRANSFORM_F** to the display list. It computes the transform given three corresponding points in screen space and bitmap space. 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 FF1F)
+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

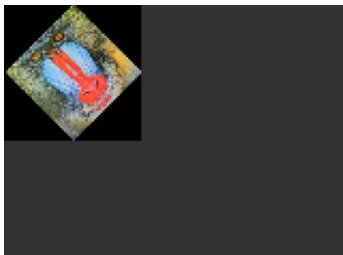
Point 2 bitmap coordinate, in pixels

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:

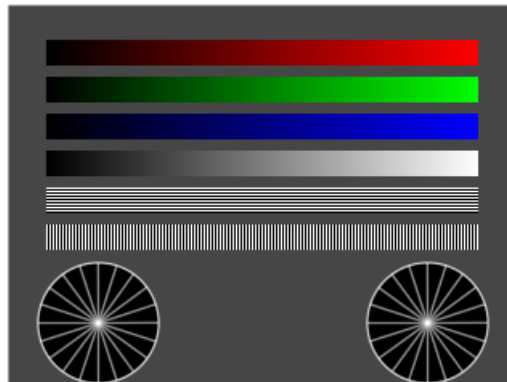


```
cmd(BLEND_FUNC(ONE, ZERO));
cmd_bitmap_transform(32,0, 64,32, 32,64,0,0,
0,64, 64,64, 0);
cmd(BEGIN(BITMAPS));
cmd(VERTEX2II(0, 0, 0, 0));
```

5.114 CMD_TESTCARD

The testcard command loads a display list with a testcard graphic and executes **CMD_SWAP** - swap the current display list to display it. The graphic is automatically scaled for the current display size, taking into account of **REG_HSIZE**, **REG_VSIZE** and **REG_RE_ROTATE**. Features of the testcard are:

- white border at the extents to confirm screen edges and clock stability
- red, green, blue and white gradients to confirm color bit depth
- horizontal and vertical checker patterns to confirm signal integrity
- circle graphics to confirm aspect ratio
- radial line pattern to confirm antialias performance



C prototype

```
void cmd_testcard ( );
```

Command layout

+0	CMD_TESTCARD(0xFFFF FF57)
----	----------------------------------

Parameters

NA

Examples

```
//To display a test card, call the command:  
cmd_testcard();
```

5.115 CMD_WAIT

This command waits for a specified number of microseconds. Delays of more than 1s (1000000 μ s) are not supported.

C prototype

```
void cmd_wait ( uint32_t us );
```

Command layout

+0	CMD_WAIT(0xFFFF FF59)
+4	us

Parameters

us
 Delay time, in microseconds

Examples

```
//To delay for 16.7 ms:
cmd_wait(16700);
```

5.116 CMD_WAITCHANGE

The waitchange command waits for the given register value to change.

C prototype

```
void cmd_waitchange ( uint32_t a );
```

Command layout

+0	CMD_WAITCHANGE(0xFFFF FF67)
+4	a

Parameters

a
 Address of RAM or register to wait on, 32-bit aligned

Examples

To wait until FRAMES changes, indicating the start of a new scanout frame:

```
cmd_waitchange (REG_FRAMES);
```

5.117 CMD_NEWLIST

This command starts the compilation of a command list into **RAM_G**. Instead of being executed, the following commands are appended to the list, until the following **CMD_ENDLIST**. The list can then be called with **CMD_CALLLIST**. The following commands are not supported in command lists. Their behavior is undefined:

- **CMD_FLASHSPITX**
- **CMD_FLASHWRITE**
- **CMD_INFLATE**
- **CMD_NEWLIST**

The following commands are supported only when using **OPT_MEDIAFIFO**, **OPT_FLASH** or **OPT_FS**:

- **CMD_INFLATE**
- **CMD_LOADIMAGE**
- **CMD_PLAYVIDEO**
- **CMD_PLAYWAV**
- **CMD_LOADWAV**

C prototype

```
void cmd_newlist ( uint32_t a );
```

Command layout

+0	CMD_NEWLIST(0xFFFF FF5C)
+4	a

Parameters

a
 memory address of start of command list. If -1 is given as the address, then the current allocation pointer is used instead.

Examples

```

/**
Create a command list at RAM_G address 0xF0000 by
sending the following commands to command buffer
***/
cmd_newlist(RAM_G + 0xF0000);
cmd(COLOR_RGB(255, 100, 0));
cmd_button(20, 20, 60, 60, 30, 0, "OK!");
cmd_endlist();

//.....

/**
Invoke the command list
***/
cmd_dlstart();
cmd(COLOR_RGB(255, 255, 255));
cmd(CLEAR(1,1,1));
cmd_calllist(RAM_G + 0xF0000);
cmd(DISPLAY());
cmd_swap();

```

5.118 CMD_ENDLIST

This command terminates the compilation of a command list into **RAM_G**. **CMD_GETPTR** can be used to find the first unused memory address following the command list.

C prototype

```
void cmd_endlist ( );
```

Command layout

+0	CMD_ENDLIST(0xFFFF FF5D)
----	---------------------------------

Examples

See [CMD_NEWLIST](#).

5.119 CMD_CALLLIST

This command calls a command list. After this command, all the commands compiled into the command list between **CMD_NEWLIST** and **CMD_ENDLIST** are executed, as if they were executed at the point of the **CMD_CALLLIST**.

The command list itself may contain **CMD_CALLLIST** commands, up to a depth of 4 levels.

C prototype

```
void cmd_calllist ( uint32_t a );
```

Command layout

+0	CMD_CALLLIST(0xFFFF FF5B)
+4	a

Parameters

a
memory address of the command list

Examples

See [CMD_NEWLIST](#).

5.120 CMD_COPYLIST

The **CMD_COPYLIST** command copies the current display list verbatim into **RAM_G**, so that it can be called with **CMD_CALLLIST** later.

The current display list is not affected by this command.

CMD_GETPTR can be used to find the first unused memory address following the command list.

C prototype

```
void cmd_copylist ( uint32_t dst )
```

Command layout

+0	CMD_COPYLIST(0xFFFF FF88)
+4	dst

Parameters

dst
Destination address for the display list, 4-byte aligned. If -1 is given as the address, then the current allocation pointer is used instead.

Examples

To copy the display list to 0x100 and call it later:

```

// Record the current display list before rendering it
cmd(CLEAR_COLOR_RGB(100, 0, 0));
cmd(CLEAR(1,1,1));
cmd_text(500, 500, 28, OPT_CENTER, "This is a cached
display list");
cmd_copylist(0x100); // Copy it to RAM_G at 0x100
cmd(DISPLAY());
cmd_swap();
...
// Now use the list
cmd_dlstart();
cmd_calllist(0x100);
cmd(DISPLAY());
cmd_swap();
  
```

5.121 CMD_SAVECONTEXT

This command adds a **SAVE_CONTEXT** to the display list and preserves the coprocessor graphics state on the state stack. See section 5.9 for a list of the affected coprocessor state.

C prototype

```
void cmd_savecontext ( )
```

Command layout

+0	CMD_SAVECONTEXT(0xFFFF FF7C)
----	-------------------------------------

Parameters

NA

Examples

NA

5.122 CMD_RESTORECONTEXT

This command adds a **RESTORE_CONTEXT** to the display list and restores the coprocessor graphics state from the stack. This stack is deep enough to hold 4 copies of the graphics state. See section 5.9 for a list of the affected coprocessor state.

This command sets all coprocessor state to its reset defaults, see section 5.9.

C prototype

```
void cmd_restorecontext ( )
```

Command layout

+0	CMD_RESTORECONTEXT(0xFFFF FF7D)
----	--

Parameters

NA

Examples

NA

5.123 CMD_SKIPCOND

Skip following command bytes if a given condition is true. The condition is expressed using a similar scheme as [STENCIL_FUNC](#). Pseudocode for the operation is:

```
if (((*a & mask) <func> (ref & mask)))
  skip subsequent num bytes;
```

Note that **ALWAYS** is an unconditional skip.

C prototype

```
void cmd_skipcond ( uint32_t a,
                   uint32_t func,
                   uint32_t ref,
                   uint32_t mask,
                   uint32_t num );
```

While this command can be used in the command buffer, it is more useful with a call list invoked through **CMD_CALLLIST**.

Command layout

+0	CMD_SKIPCOND(0xFFFF FF8C)
+4	a
+8	func
+12	ref
+16	mask
+20	num

Parameters

a

Address of **RAM_G** or register to check, 32-bit aligned

func

one of **NEVER, LESS, LEQUAL, GREATER, GEQUAL, EQUAL, NOTEQUAL** or **ALWAYS**

ref

32-bit reference value

mask

32-bit mask

Example

To set the current color to red if the bit 5 of **REG_FRAMES** is 1:

```
cmd_skipcond(REG_FRAMES, EQUAL, 0x00, 0x20, 4);
cmd(COLOR_RGB(0xff, 0x00, 0x00));
```

As a more complex example, alternate red and green on the same bit, using an if-then-else structure:

```
cmd_skipcond(REG_FRAMES, EQUAL, 0x00, 0x20, 4 + 20);
cmd(COLOR_RGB(0xff, 0x00, 0x00)); // executed if FRAMES[5] == 0
cmd_skipcond(0x0, ALWAYS, 0x0, 0x0, 4); // unconditional branch
cmd(COLOR_RGB(0x00, 0xff, 0x00)); // executed if FRAMES[5] == 1
```

5.124 CMD_WAITCOND

Wait until the given condition is true. The condition is expressed using a similar scheme as [STENCIL_FUNC](#). Pseudocode for the operation is:

```
while (not ((*a & mask) <func> (ref & mask)))
;
```

Note that **ALWAYS** never waits and **NEVER** waits indefinitely, so neither are useful.

C prototype

```
void cmd_waitcond ( uint32_t a,
                   uint32_t func,
                   uint32_t ref,
                   uint32_t mask );
```

Command layout

+0	CMD_WAITCOND(0xFFFF FF78)
+4	a
+8	func
+12	ref
+16	mask

Parameters

a
Address of **RAM_G** or register to check, 32-bit aligned

func
one of **NEVER**, **LESS**, **LEQUAL**, **GREATER**, **GEQUAL**, **EQUAL**, **NOTEQUAL**, or **ALWAYS**

ref
32-bit reference value

mask
32-bit mask

Examples

```
cmd_waitcond(REG_FRAMES, GREATER, 123, 0xffffffff); // Wait until
REG_FRAMES > 123
cmd_waitcond(REG_PLAY, EQUAL, 0, 0xffffffff); // Wait until REG_PLAY is
zero
```


5.125 CMD_WATCHDOG

The watchdog command enables the watchdog timer and sets the watchdog reset intervals in clocks. The watchdog flag in **REG_BOOT_CFG** register must be set.

C prototype

```
void cmd_watchdog ( uint32_t init_val );
```

Command layout

+0	CMD_WATCHDOG(0xFFFF FF83)
+4	init_val

Parameters

init_val

Watchdog timeout initialization value, in clocks. Must be a sensible value to prevent watchdog being triggered prematurely.

Examples

To set the watchdog timeout every 72000000 clocks, which is 1 second on a 72 MHz system:

```
cmd_watchdog(72000000);
```

5.126 CMD_RESULT

The result command copies the result field of the preceding command into memory. This can make accessing the result value more convenient than reading directly from the command FIFO.

C prototype

```
void cmd_result ( );
```

Command layout

+0	CMD_RESULT(0xFFFF FF89)
+4	dst

Parameters

dst

Memory address to write, 32-bit aligned

Examples

To copy the result of CMD_SDATTACH to memory at 0x100:

```
cmd_sdattach(0, 0x7777);  
cmd_result(0x100);
```

5.127 CMD_RETURN

This command ends a command list. Normally it is not needed by the user, because **CMD_ENDLIST** appends **CMD_RETURN** to the command list. However, it may be used in situations where the user is constructing command lists offline.

C prototype

```
void cmd_return ( )
```

Command layout

+0	CMD_RETURN(0xFFFF FF5A)
-----------	--------------------------------

Examples

```

/**
Construct a command list in RAM_G to show a button
***/
wr32(RAM_G + 0 * 4, SAVE_CONTEXT());
wr32(RAM_G + 1 * 4, COLOR_RGB(125, 125, 128));
wr32(RAM_G + 2 * 4, CMD_BUTTON);

wr32(RAM_G + 3 * 4, 160); //x coordinate of button
wr32(RAM_G + 3 * 4 + 2, 160); //y coordinate of button
wr32(RAM_G + 4 * 4, 123); //w
wr32(RAM_G + 4 * 4 + 2, 234); //h
wr32(RAM_G + 5 * 4, 31); //Font handle
wr32(RAM_G + 5 * 4 + 2, 0); //option parameter of cmd_button
wr32(RAM_G + 6 * 4, 'T');
wr32(RAM_G + 6 * 4 + 1, 'E');
wr32(RAM_G + 6 * 4 + 2, 'S');
wr32(RAM_G + 6 * 4 + 3, 'T');

wr32(RAM_G + 7 * 4, '\0'); //the null terminators for string "TEST"

//Append extra 3 bytes for alignment purpose
wr32(RAM_G + 7 * 4 + 1, '\0');
wr32(RAM_G + 7 * 4 + 2, '\0');
wr32(RAM_G + 7 * 4 + 3, '\0');

wr32(phost, RAM_G + 8 * 4, RESTORE_CONTEXT());
wr32(phost, RAM_G + 9 * 4, CMD_RETURN); //Indicate the end of command
list
//.....

/**
Invoke the command list in RAM_G to render the button
***/
cmd_dlstart();
cmd(COLOR_RGB(255, 255, 255));
cmd(CLEAR(1,1,1));
cmd_calllist(RAM_G);
cmd(DISPLAY());
cmd_swap();

```

5.128 CMD_NOP

The nop command does nothing.

C prototype

```
void cmd_nop ( );
```

Command layout

+0	CMD_NOP(0xFFFF FF53)
----	----------------------

Examples

To do nothing:

```
cmd_nop();
```

5.129 CMD_GETIMAGE

This command returns all the attributes of the bitmap made by the previous **CMD_LOADIMAGE**, **CMD_PLAYVIDEO** or **CMD_VIDEOSTART**.

C prototype

```
void cmd_getimage( uint32_t source,
                  uint32_t fmt,
                  uint32_t w,
                  uint32_t h,
                  uint32_t palette );
```

Command layout

+0	CMD_GETIMAGE (0xFFFF FF58)
+4	source
+8	fmt
+12	w
+16	h
+20	palette

Parameters

source

Output parameter; source address of bitmap.

fmt

Output parameter; format of the bitmap

w

Width of bitmap, in pixels

h

Height of bitmap, in pixels

palette

Palette data of the bitmap if fmt is **PALETTERGB8**. Otherwise, zero.

Examples

```
//To find the base address and dimensions of the previously loaded image
uint32_t source, fmt, w, h, palette;
uint32_t ram_fifo_offset = rd32(REG_CMD_WRITE);

cmd_getimage(src, fmt, w, h, palette);

src = rd32(RAM_CMD + (ram_fifo_offset + 4) % 16384);
fmt = rd32(RAM_CMD + (ram_fifo_offset + 8) % 16384);
w = rd32(RAM_CMD + (ram_fifo_offset + 12) % 16384);
h = rd32(RAM_CMD + (ram_fifo_offset + 16) % 16384);
palette = rd32(RAM_CMD + (ram_fifo_offset + 20) % 16384);

cmd_setbitmap(src, fmt, w, h);
if (palette != 0) PaletteSource(palette);
```

5.130 CMD_I2SSTARTUP

The i2sstartup command fills the I2S output **FIFO** to the halfway mark with zeroes, writes the given frequency to **REG_I2S_FREQ** and enables I2S by writing 1 to **REG_I2S_EN**.

C prototype

```
void cmd_i2sstartup( uint32_t freq );
```

Command layout

+0	CMD_I2SSTARTUP (0xFFFF FF69)
+4	freq

Parameters

freq
 I²S sample frequency, in Hz

Examples

```
// To prepare 44100 Hz I2S output
cmd_i2sstartup(44100);
```

5.131 CMD_LOADASSET

The loadasset command loads an asset in .reloc format at the given address. The .reloc format contains both the asset data and relocation information so the asset can be loaded and used at any address. A good and preferred way of loading fonts is through this loadasset command as fonts in .reloc format are much smaller and therefore faster to load.

C prototype

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

Command layout

+0	CMD_LOADASSET (0xFFFF FF81)
+4	ptr
+8	options
+12	byte ₁ ... byte _n

Parameters

ptr

Destination address. If -1 is given as the address, then the current allocation pointer is used instead.

options

By default the data is sourced from the command buffer, immediately following the command itself. Other sources can be selected using one of the following options:

OPT_FLASH: flash memory is the source, see CMD_FLASHSOURCE.

OPT_FS: the filesystem is the source, see CMD_FSSOURCE.

OPT_MEDIAFIFO: the media FIFO is used for the source, see CMD_MEDIAFIFO.

Examples

```
cmd_fssource("serif.reloc", 0);
cmd_loadasset(0, OPT_FS);
```

5.132 CMD_SNAPSHOT

The snapshot command causes the coprocessor to take a snapshot of the current screen and write it into graphics memory as a **ARGB4** bitmap. The size of the bitmap is the size of the screen, given by the **REG_RE_W** and **REG_RE_H** registers.

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

C prototype

```
void cmd_snapshot (uint32_t ptr);
```

Parameters

ptr

The address of the snapshot

Command layout

+0	CMD_SNAPSHOT (0xFFFF FF1D)
+4	ptr

Examples

To take a snapshot of the current 160×120 screen, then use it as a bitmap in the new display list:

```
cmd_snapshot(0);  
cmd_dlstart();  
cmd(CLEAR(1,1,1));  
cmd(BITMAP_HANDLE(0));  
cmd_setbitmap(0, ARGB4, 160, 120);  
cmd(BEGIN(BITMAPS));  
cmd(VERTEX2II(10, 10, 0, 0));
```

6 ASTC

ASTC stands for **A**daptive **S**calable **T**exture **C**ompression, an open standard developed by **ARM**® for use in mobile GPUs.

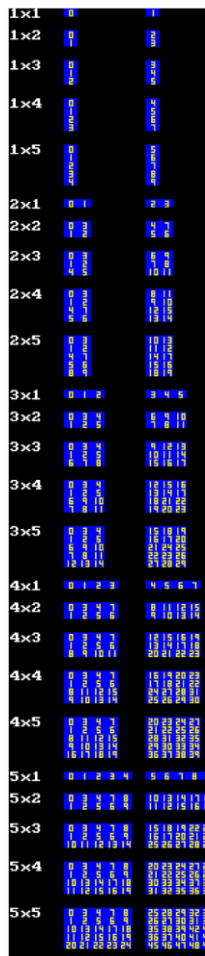
ASTC is a block-based lossy compression format. The compressed image is divided into several blocks of uniform size, which makes it possible to quickly determine which block a given texel resides in. Each block has a fixed memory footprint of 128 bits, but these bits can represent varying numbers of texels (the block footprint).

Block footprint sizes are not confined to powers-of-two and are also not confined to square. For 2D formats the block dimensions range from 4 to 12 texels.

Using ASTC for the large ROM fonts can save considerable space. Encoding the four largest fonts in ASTC format gives no noticeable loss in quality and reduces the **ROM** size from 1M Byte to about 640K Kbytes.

BT82X series empowers animation features and **Unicode** support based on **ASTC** format. With enough ASTC images in **RAM_G**, it is possible for the user to construct an image-rich **GUI** application.

6.1 ASTC Layout



ASTC blocks represent between 4x4 to 12x12 pixels. Each block is 16 bytes in memory. Please see the Table 24 – BITMAP_LAYOUT Format List for more details. In a nutshell, 4x4 stands for lowest compression rate but best quality while 12x12 means for highest compression rate but worst quality. Users may need to evaluate the image quality of various **ASTC blocks** on hardware to achieve the trade-off.

ASTC bitmaps in main memory must be 16-byte aligned. However, for a multi-cell bitmap to use the **CELL** command, the source address of each bitmap cell must start on a multiple of 4 blocks, i.e., 64-byte aligned.

The mapping from bitmap coordinates to memory locations is not always linear. Instead, blocks are grouped into 2x2 tiles. Within the tile, the order is:

0	3
1	2

When there is an odd number of blocks on a line, the final two blocks are packed into a 1x2. When there is an odd number of rows, then the final row of blocks is linear.

The above diagram shows the same piece of memory loaded with ASTC blocks drawn with ascending memory addresses. The first column shows the addresses used by cell 0, the second column cell 1.

7 YCbCr format

YCbCr is a lossy pixel format using 1 byte per pixel. Pixels are encoded in 2×2 blocks, each quad taking 32 bits of memory.

a	b
c	d

It is particularly suited to JPEG output but is also effective as a framebuffer format with very little loss in quality.

There are three encoding variants, depending on the distribution of Y pixels in the quad. The 5-bit chroma encoding is most frequently used, with 6-bit and 8-bit chroma used for quads with similar Y values. This improves color fidelity in image regions with little luminance detail.

5-bit chroma

31	28	27	23	22	16	15	8	7	0
C'_b		C_r		index		Y_1		Y_0	

Here Y_0 and Y_1 are the endpoints of the pixel Y values. index gives the Y values of each of the four pixels, using a codebook described below. C_r is the color component used for all four pixels, reduced to five bits. C_b is similarly expressed, using the four bits of C'_b plus an extra bit given by $(Y_0 > Y_1)$.

The index field specifies the Y values for the four pixels. Each is a 2-bit index specifying a position in the interval (Y_0, Y_1)

- 0 $Y = Y_0$
- 1 $Y = \frac{2 \cdot Y_0 + Y_1}{3}$
- 2 $Y = \frac{Y_0 + 2 \cdot Y_1}{3}$
- 3 $Y = Y_1$

Since codes 0 and 3 must occur somewhere in the quad, some combinations are unused. Hence of the $4^4 = 256$ combinations, only 110 are legal. The encoding uses a table to express this index in 7 bits rather than 8 bits. Codes 0-109 give the four pixel indices. Codes 110-127 are reserved, and currently unused.

6-bit chroma

For the case when $(Y_1 - Y_0) = 1$, only 1 bit is required to select Y_0 or Y_1 per pixel, so the index field uses 4 bits.

31	26	25	20	19	16	15	8	7	0
C_b		C_r		index		Y		Y	

7.1 YcbCr Encoding

Using the standard JPEG equations, transform the four RGB pixels to YCbCr:

$$\begin{aligned}
 Y' &= 0 + (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
 C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
 C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
 \end{aligned}$$

7.2 YcbCr Decoding

$$\begin{aligned}
 R'_D &= Y' + 1.402 \cdot (C_R - 128) \\
 G'_D &= Y' - 0.344136 \cdot (C_B - 128) - 0.714136 \cdot (C_R - 128) \\
 B'_D &= Y' + 1.772 \cdot (C_B - 128)
 \end{aligned}$$

8 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 any application 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 damages, claims, suits, or expenses 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

[BT82X datasheet](#)
[OpenGL 4.5 Reference Pages](#)

Acronyms and Abbreviations

Terms	Description
ADC	Analog-to-digital
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASTC	Adaptive Scalable Texture Compression
AVI	Audio Video Interactive
CTSE	Capacitive Touch Screen Engine
DDR RAM	Double Data Rate Random-Access Memory
EVE	Embedded Video Engine
FIFO	First In First Out buffer
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
I2S	Inter-IC Sound
JEIDA	Japan Electronics and Information Technology Industries Association
JPEG	Joint Photographic Experts Group
LCD	Liquid Crystal Display
LVDS	Low-Voltage Differential Signaling
MCU	Micro controller unit
MPU	Microprocessor Unit
PCM	Pulse-Code Modulation
PNG	Portable Network Graphics
PWM	Pulse Width Modulation
RAM	Random Access Memory
RTE	Resistive Touch Engine

ROM	Read Only Memory
SD	Secure Digital
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
VESA	Video Electronics Standards Association

Appendix B – List of Tables/ Figures/ Registers/ Code Snippets

List of Tables

Table 1 – API Reference Definitions	13
Table 2 – Memory Map	14
Table 3 – Host commands	18
Table 4 – System Clock Configuration	19
Table 5 – DDR DRAM Types Setting	19
Table 6 – Chip Identification Code	20
Table 7 – EVE Boot Status	21
Table 8 – Flash Interface states and commands	24
Table 9 – Supported bitmap format	34
Table 10 – Video Playback Maximum Bandwidth	39
Table 11 – Touch Registers Summary	49
Table 12 – CTSE Registers Summary	53
Table 13 – Swap Chain Registers Summary	55
Table 14 – LVDSRX registers Summary	58
Table 15 – LVDSRX registers Summary	60
Table 16 – LVDS mode assignment for VESA and JEIDA formats	62
Table 17 – LVDSTX registers Summary	62
Table 18 – LVDSTX Clock Configuration	63
Table 19 – System Register Memory Map Summary	66
Table 20 – I2S Memory Map Summary	73
Table 21 – Graphics Context	83
Table 22 – Graphics Primitive Definition	86
Table 23 – Bitmap formats and bits per pixel	88
Table 24 – BITMAP_LAYOUT Format List	89
Table 25 – BITMAP_LAYOUT Format in pixel aligned	90
Table 26 – L1/L2/L4/L8 Pixel Format	91
Table 27 – ARGB2/RGB332 Pixel Format	91
Table 28 – RGB565 Pixel Format	91
Table 29 – ARGB1555/ARGB4 Pixel Format	91
Table 30 – RGB6 Pixel Format	91
Table 31 – ARGB6 Pixel Format	91
Table 32 – RGB8 Pixel Format	91
Table 33 – ARGB8 Pixel Format	91
Table 34 – LA1/LA2/LA4/LA8 Pixel Format	92
Table 35 – PALETTEDARGB8 Pixel Format	92

Table 36 – BLEND_FUNC Constant Value Definition	103
Table 37 – STENCIL_OP Constants Definition	120
Table 38 – VERTEX_FORMAT and Pixel Precision.....	124
Table 39 – Widgets Color Setup Table	128
Table 40 – Legacy Font Metrics Block.....	129
Table 41 – Extended Font Metrics Block	130
Table 42 – String Format Specifier	140
Table 43 – Coprocessor Faults Strings	142
Table 44 – Coprocessor Engine State	142
Table 45 – Parameter OPTION Definition.....	144
Table 46 – Updated Coprocessor Commands in BT82X	146

List of Figures

Figure 1 – BT82X data flow.....	16
Figure 2 – Getting Started Example.....	28
Figure 3 – Widget List	127
Figure 4 – ROM Font List	135

List of Registers

Register Definition 1 – REG_TAG Definition.....	40
Register Definition 2 – REG_TAG_Y Definition	40
Register Definition 3 – REG_TAG_X Definition	41
Register Definition 4 – REG_PCLK_POL Definition	41
Register Definition 5 – REG_DLSWAP Definition.....	41
Register Definition 6 – REG_VSYNC1 Definition	41
Register Definition 7 – REG_VSYNC0 Definition	41
Register Definition 8 – REG_VSIZE Definition	42
Register Definition 9 – REG_VOFFSET Definition	42
Register Definition 10 – REG_VCYCLE Definition.....	42
Register Definition 11 – REG_HSYNC1 Definition	42
Register Definition 12 – REG_HSYNC0 Definition	42
Register Definition 13 – REG_HSIZE Definition.....	42
Register Definition 14 – REG_HOFFSET Definition.....	43
Register Definition 15 – REG_HCYCLE Definition	43
Register Definition 16 – REG_RE_DEST Definition	43
Register Definition 17 – REG_RE_FORMAT Definition.....	43
Register Definition 18 – REG_RE_ROTATE Definition	43
Register Definition 19 – REG_RE_W Definition.....	43
Register Definition 20 – REG_RE_H Definition	44

Register Definition 21 – REG_RE_DITHER Definition	44
Register Definition 22 – REG_RE_ACTIVE Definition	44
Register Definition 23 – REG_RE_RENDERS Definition.....	44
Register Definition 24 – REG_SO_MODE Definition	44
Register Definition 25 – REG_SO_SOURCE Definition	44
Register Definition 26 – REG_SO_FORMAT Definition	45
Register Definition 27 – REG_SO_EN Definition	45
Register Definition 28 – REG_PLAY Definition.....	45
Register Definition 29 – REG_SOUND Definition.....	45
Register Definition 30 – REG_VOL_SOUND Definition.....	45
Register Definition 31 – REG_VOL_L_PB Definition	46
Register Definition 32 – REG_VOL_R_PB Definition	46
Register Definition 33 – REG_PLAYBACK_PLAY Definition	46
Register Definition 34 – REG_PLAYBACK_LOOP Definition.....	46
Register Definition 35 – REG_PLAYBACK_FORMAT Definition.....	46
Register Definition 36 – REG_PLAYBACK_FREQ Definition	47
Register Definition 37 – REG_PLAYBACK_READPTR Definition	47
Register Definition 38 – REG_PLAYBACK_LENGTH Definition	47
Register Definition 39 – REG_PLAYBACK_START Definition	47
Register Definition 40 – REG_PLAYBACK_PAUSE Definition	47
Register Definition 41 – REG_AUD_PWM Definition.....	48
Register Definition 42 – REG_I2S_EN Definition	48
Register Definition 43 – REG_I2S_FREQ Definition	48
Register Definition 44 – REG_FLASH_STATUS Definition	48
Register Definition 45 – REG_FLASH_SIZE Definition	48
Register Definition 46 – REG_TOUCH_CONFIG Definition	49
Register Definition 47 – REG_TOUCH_TRANSFORM_F Definition	50
Register Definition 48 – REG_TOUCH_TRANSFORM_E Definition	50
Register Definition 49 – REG_TOUCH_TRANSFORM_D Definition	50
Register Definition 50 – REG_TOUCH_TRANSFORM_C Definition	50
Register Definition 51 – REG_TOUCH_TRANSFORM_B Definition	50
Register Definition 52 – REG_TOUCH_TRANSFORM_A Definition	50
Register Definition 53 – REG_TOUCH_TAG Definition.....	51
Register Definition 54 – REG_TOUCH_TAG_XY Definition	51
Register Definition 55 – REG_TOUCH_TAG1 Definition	51
Register Definition 56 – REG_TOUCH_TAG1_XY Definition.....	51
Register Definition 57 – REG_TOUCH_TAG2 Definition	51
Register Definition 58 – REG_TOUCH_TAG2_XY Definition.....	52
Register Definition 59 – REG_TOUCH_TAG3 Definition	52

Register Definition 60 – REG_TOUCH_TAG3_XY Definition	52
Register Definition 61 – REG_TOUCH_TAG Definition.....	52
Register Definition 62 – REG_TOUCH_TAG_XY Definition	52
Register Definition 63 – REG_TOUCH_SCREEN_XY Definition	53
Register Definition 64 – REG_TOUCH_RAW_XY Definition.....	53
Register Definition 65 – REG_TOUCH_MODE Definition	53
Register Definition 66 – REG_CTOUCH_EXTENDED Definition	53
Register Definition 67 – REG_CTOUCH_TOUCH0_XY Definition.....	54
Register Definition 68 – REG_CTOUCH_TOUCHA_XY Definition	54
Register Definition 69 – REG_CTOUCH_TOUCHB_XY Definition	54
Register Definition 70 – REG_CTOUCH_TOUCHC_XY Definition	54
Register Definition 71 – REG_CTOUCH_TOUCH4_XY Definition.....	54
Register Definition 72 – REG_SC0_RESET Definition.....	55
Register Definition 73 – REG_SC0_SIZE Definition.....	55
Register Definition 74 – REG_SC0_PTR0 Definition.....	55
Register Definition 75 – REG_SC0_PTR1 Definition.....	56
Register Definition 76 – REG_SC0_PTR2 Definition.....	56
Register Definition 77 – REG_SC0_PTR3 Definition.....	56
Register Definition 78 – REG_SC1_RESET Definition.....	56
Register Definition 79 – REG_SC1_SIZE Definition.....	56
Register Definition 80 – REG_SC1_PTR0 Definition.....	56
Register Definition 81 – REG_SC1_PTR1 Definition.....	56
Register Definition 82 – REG_SC1_PTR2 Definition.....	57
Register Definition 83 – REG_SC1_PTR3 Definition.....	57
Register Definition 84 – REG_SC2_RESET Definition.....	57
Register Definition 85 – REG_SC2_SIZE Definition.....	57
Register Definition 86 – REG_SC2_PTR0 Definition.....	57
Register Definition 87 – REG_SC2_PTR1 Definition.....	57
Register Definition 88 – REG_SC2_PTR2 Definition.....	57
Register Definition 89 – REG_SC2_PTR3 Definition.....	58
Register Definition 90 – REG_SC2_STATUS Definition	58
Register Definition 91 – REG_SC2_ADDR Definition	58
Register Definition 92 – REG_LVDSRX_CORE_ENABLE Definition.....	58
Register Definition 93 – REG_LVDSRX_CORE_CAPTURE Definition	59
Register Definition 94 – REG_LVDSRX_CORE_SETUP Definition	59
Register Definition 95 – REG_LVDSRX_CORE_DEST Definition	59
Register Definition 96 – REG_LVDSRX_CORE_FORMAT Definition.....	59
Register Definition 97 – REG_LVDSRX_CORE_DITHER Definition	59
Register Definition 98 – REG_LVDSRX_SETUP Definition.....	60

Register Definition 99 – REG_LVDSRX_CTRL Definition	61
Register Definition 100 – REG_LVDSRX_STAT Definition	61
Register Definition 101 – REG_LVDSTX_EN Definition	62
Register Definition 102 – REG_LVDSTX_PLLCFG Definition	63
Register Definition 103 – REG_LVDSTX_CTRL_CH0 Definition	64
Register Definition 104 – REG_LVDSTX_CTRL_CH1 Definition	64
Register Definition 105 – REG_LVDSTX_STAT Definition	65
Register Definition 106 – REG_LVDSTX_ERR_STAT Definition	65
Register Definition 107 – REG_PIN_DRV_0 Definition	66
Register Definition 108 – REG_PIN_DRV_1 Definition	67
Register Definition 109 – REG_PIN_SLEW_0 Definition	68
Register Definition 110 – REG_PIN_TYPE_0 Definition	69
Register Definition 111 – REG_PIN_TYPE_1 Definition	70
Register Definition 112 – REG_SYS_CFG Definition	70
Register Definition 113 – REG_SYS_STAT Definition	71
Register Definition 114 – REG_CHIP_ID Definition	71
Register Definition 115 – REG_BOOT_STATUS Definition	71
Register Definition 116 – REG_DDR_TYPE Definition	72
Register Definition 117 – REG_PIN_DRV_2 Definition	72
Register Definition 118 – REG_PIN_SLEW_1 Definition	72
Register Definition 119 – REG_PIN_TYPE_2 Definition	73
Register Definition 120 – REG_I2S_CFG Definition	74
Register Definition 121 – REG_I2S_CTL Definition	74
Register Definition 122 – REG_I2S_STAT Definition	74
Register Definition 123 – REG_I2S_PAD_CFG Definition	75
Register Definition 124 – REG_CMD_DL Definition	75
Register Definition 125 – REG_CMD_WRITE Definition	75
Register Definition 126 – REG_CMD_READ Definition	75
Register Definition 127 – REG_CMDB_SPACE Definition	76
Register Definition 128 – REG_CMDB_WRITE Definition	76
Register Definition 129 – REG_BOOT_CFG Definition	76
Register Definition 130 – REG_CPURESET Definition	76
Register Definition 131 – REG_MACRO_1 Definition	77
Register Definition 132 – REG_MACRO_0 Definition	77
Register Definition 133 – REG_PWM_DUTY Definition	77
Register Definition 134 – REG_PWM_HZ Definition	77
Register Definition 135 – REG_INT_MASK Definition	77
Register Definition 136 – REG_INT_EN Definition	78
Register Definition 137 – REG_INT_FLAGS Definition	78

Register Definition 138 – REG_GPIO_DIR Definition	78
Register Definition 139 – REG_GPIO Definition	79
Register Definition 140 – REG_DISP Definition.....	79
Register Definition 141 – REG_FREQUENCY Definition.....	79
Register Definition 142 – REG_CLOCK Definition.....	79
Register Definition 143 – REG_FRAMES Definition.....	79
Register Definition 144 – REG_ID Definition	80
Register Definition 145 – REG_TRACKER Definition	80
Register Definition 146 – REG_TRACKER_1 Definition.....	80
Register Definition 147 – REG_TRACKER_2 Definition.....	80
Register Definition 148 – REG_TRACKER_3 Definition.....	81
Register Definition 149 – REG_TRACKER_4 Definition.....	81
Register Definition 150 – REG_MEDIAFIFO_READ Definition	81
Register Definition 151 – REG_MEDIAFIFO_WRITE Definition.....	81
Register Definition 152– REG_ANIM_ACTIVE Definition	81
Register Definition 153 – REG_OBJECT_COMPLETE Definition	82
Register Definition 154 – REG_EXTENTS_X0 Definition	82
Register Definition 155 – REG_EXTENTS_Y0 Definition	82
Register Definition 156 – REG_EXTENTS_X1 Definition	82
Register Definition 157 – REG_EXTENTS_Y1 Definition	82
Register Definition 158 – REG_PLAY_CONTROL Definition	82

List of Code Snippets

Code Snippet 1 – Initialization Sequence	20
Code Snippet 2 – Play C8 on the Xylophone	24
Code snippet 3 – Stop Playing Sound	24
Code snippet 4 – Avoid Pop Sound	24
Code Snippet 5 – Audio Playback.....	25
Code Snippet 6 – Check the status of Audio Playback	25
Code Snippet 7 – Stop the Audio Playback.....	25
Code Snippet 8 – Getting Started	28
Code Snippet 9 – Color and Transparency	33

Appendix C – Revision History

Document Title: BRT_AN_086 BT82X Series Programming Guide

Document Reference No.: BRT_000409

Clearance No.: BRT#215

Product Page: <http://brtchip.com/product/>

Document Feedback: [Send Feedback](#)

Revision	Changes	Date (DD-MM-YYYY)
Version 1.0	Initial release	02-12-2024