Vivante programming: DPU API reference guide

Note: *The current release supports DPU API version 3.0.*

About this document

Scope and purpose

This document describes various Display Processor Unit (DPU) APIs available for the user to make use of Display Controller.

Intended audience

This document is intended for anyone who wants to make use of the DPU API for performing Display Controller operations.

Table of contents

About this document **1**
Table of contents **2**
1 Overview **4**
1.1 Compatibility 4
1.2 API files 4
2 Data structures **5**
2.1 Value types 5
2.2 Pointer types 6
2.3 Enumerations 7
2.3.1 vivSTATUS 7
2.3.2 viv_input_format_type 7
2.3.3 viv_tiling_type 10
2.3.4 viv_display_type 11
2.3.5 viv_dbi_type 11
2.3.6 viv_display_format_type 11
2.3.7 viv_filter_tap_type 13
2.3.8 viv_cursor_size_type 13
2.3.9 viv_alpha_mode 13
2.3.10 viv_cache_mode 13
2.3.11 viv_global_alpha_mode 14
2.3.12 viv_porter_duff_mode 14
2.3.13 viv_pool_type 15
2.3.14 viv_display_size_type 15
2.3.15 viv_display 16
2.3.16 viv_dc_features 16
2.3.17 viv_dc_layer_cap 17
2.4 Structures 17
2.4.1 viv_dc_buffer 17
2.4.2 3.5.2 viv_set_cursor()viv_tilestatus_buffer 18
2.4.3 viv_cursor 19
2.4.4 viv_layer_alpha_mode 20
2.4.5 viv_dc_degamma 20
2.4.6 viv_dc_gamma 20
2.4.7 viv_output 20
2.4.8 viv_dc_rect 21
2.4.9 viv_dc_color 21
3 DPU APIs **22**
3.1 Initialization 22
3.1.1 viv_dc_init() 22
3.1.2 viv_dc_deinit() 23
3.1.3 viv_dc_reset() 23
3.2 Buffer allocation 24
3.2.1 viv_alloc_buffer() 24
3.2.2 viv_free_buffer() 25
3.3 Capability query 26
3.3.1 viv_query_chipinfo() 26
3.3.2 viv_dc_query_feature() 27

Table of contents

3.3.3 viv_layer_query_capability() 28
3.4 Overlay and video/graphic layers 28
3.4.1 viv_dc_select_layer() 28
3.4.2 viv_layer_enable() 29
3.4.3 viv_layer_set() 30
3.4.4 viv_layer_zorder() 31
3.4.5 viv_layer_set_position() 32
3.4.6 viv_layer_colorkey() 33
3.4.7 viv_layer_clear() 34
3.4.8 viv_set_alpha() 35
3.4.9 viv_layer_poterduff_blend() 35
3.4.10 viv_layer_set_watermark() 36
3.4.11 viv_dc_set_qos() 36
3.4.12 viv_layer_set_display() 37
3.5 Background and cursor layers 37
3.5.1 viv_layer_set_background() 37
3.5.2 viv_set_cursor() 38
3.5.3 viv_cursor_security() 39
3.5.4 viv_cursor_offset() 40
3.5.5 viv_cursor_move() 41
3.6 Post-processing for display 42
3.6.1 viv_gamma_enable() 42
3.6.2 viv_gamma_init() 43
3.6.3 viv_set_gamma() 44
3.6.4 viv_set_dither() 45
3.7 Display output 46
3.7.1 viv_set_display_size() 46
3.7.2 viv_set_custom_display_size() 47
3.7.3 viv_set_output() 48
3.7.4 viv_reset_dbi() 49
3.7.5 viv_set_output_dbi() 49
3.8 Display controller access and debugging 50
3.8.1 viv_set_commit() 50
3.8.2 viv_get_vblank_count() 51
4 Programming with DPU APIs 52
4.1 Example 1: Displaying through DPI 53
4.2 Example 2: Displaying through DBI Type B 55
5 List of Unsupported DPU APIs 57
Revision history 58
Disclaimer 59

1 Overview

The DPU API set serves to develop user applications to control Vivante display controllers of the newest architecture. For the supported display controllers, see Section 1.1, Compatibility.

This document describes the APIs, and the data structures used in the DPU API set. It also outlines the procedure of programming with this API set.

- Chapter 2, Data structures
- Chapter 3, DPU APIs
- Chapter 4, Programming with DPU APIs

1.1 Compatibility

This API set is compatible with the DC8000Nano Vivante display controller.

1.2 API files

The definition files for data structures and APIs are:

- Types and enumerations:

viv_dc_type.h

- APIs:

viv_dc_setting.h

2 Data structures

2.1 Value types

The following table lists the value types defined in the DPU API set.

Table 1 Value types

Name	Definition	Description
gctBOOL	int	A Boolean value • 0: SET_DISABLE, SET_NEGATIVE, vivFALSE, or gcvFALSE • 1: SET_ENABLE, SET_POSITIVE, vivTRUE, or gcvTRUE
gctCHAR	char	An 8-bit character value
gctFLOAT	float	A single-precision floating-point number
gctINT	int	A signed integer
gctINT8	signed char	A signed 8-bit integer
gctINT16	signed short	A signed 16-bit integer
gctINT32	signed int	A signed 32-bit integer
gctINT64	signed long long	A signed 64-bit integer
gctSIZE_T	unsigned long	An unsigned 64-bit integer
gctUINT	unsigned int	An unsigned integer
gctUINT8	unsigned char	An unsigned 8-bit integer
gctUINT16	unsigned short	An unsigned 16-bit integer
gctUINT32	unsigned int	An unsigned 32-bit integer
gctUINT64	unsigned long long	An unsigned 64-bit integer
gctVOID	void	Void
gctDOUBLE	double	A double-precision floating-point number

2.2 Pointer types

The following table lists the pointer types defined in the DPU API set. For the referent data types, see section 2.1, Value types.

Table 2 Pointer types

Pointer type	Name	Description
gctBOOL *	gctBOOL_PTR	A pointer to a Boolean value
void *	gctFILE	A pointer to a file
float *	gctFLOAT_PTR	A pointer to a single-precision floating-point number
void *	gctHANDLE	A handle of the operating system
gctINT *	gctINT_PTR	A pointer to a signed integer
gctINT8 *	gctINT8_PTR	A pointer to a signed 8-bit integer
gctINT16 *	gctINT16_PTR	A pointer to a signed 16-bit integer
gctINT32 *	gctINT32_PTR	A pointer to a signed 32-bit integer
gctINT64 *	gctINT64_PTR	A pointer to a signed 64-bit integer
void *	gctPHYS_ADDR	A pointer to a physical address
void *	gctPOINTER	A generic pointer
gctSIZE_T *	gctSIZE_T_PTR	A pointer to an unsigned 64-bit integer
void *	gctSTRING	A pointer to a string
gctUINT *	gctUINT_PTR	A pointer to an unsigned integer
gctUINT8 *	gctUINT8_PTR	A pointer to an unsigned 8-bit integer
gctUINT16 *	gctUINT16_PTR	A pointer to an unsigned 16-bit integer
gctUINT32 *	gctUINT32_PTR	A pointer to an unsigned 32-bit integer
gctUINT64 *	gctUINT64_PTR	A pointer to an unsigned 64-bit integer

2.3 Enumerations

2.3.1 vivSTATUS

Specifies the return code of a DPU API.

Enumeration value	Numeric value	Description
vivSTATUS_HEAP_CORRUPTED	-7	Heap corrupted
vivSTATUS_OUT_OF_RESOURCES	-6	Resource access out of bounds
vivSTATUS_TIMEOUT	-5	Timeout
vivSTATUS_NOT_SUPPORT	-4	Unsupported features
vivSTATUS_OOM	-3	Out of memory
vivSTATUS_FAILED	-2	File operation failed
vivSTATUS_INVALID_ARGUMENTS	-1	Invalid input parameters
vivSTATUS_OK	0	Function successful

2.3.2 viv_input_format_type

Specifies the color format of the input.

RGB formats

Note: *The display controller does not support 24 bpp RGB formats for DC8000Nano.*

Enumeration value	Description
vivARGB4444	16-bit ARGB format with the alpha channel in bits 15:12, the red channel in bits 11:8, the green channel in bits 7:4, and the blue channel in bits 3:0
vivABGR4444	16-bit ABGR format with the alpha channel in bits 15:12, the blue channel in bits 11:8, the green channel in bits 7:4, and the red channel in bits 3:0
vivRGBA4444	16-bit RGBA format with the red channel in bits 15:12, the green channel in bits 11:8, the blue channel in bits 7:4, and the alpha channel in bits 3:0
vivBGRA4444	16-bit BGRA format with the blue channel in bits 15:12, the green channel in bits 11:8, the red channel in bits 7:4, and the alpha channel in bits 3:0
vivXRGB4444	16-bit XRGB format with the X channel in bits 15:12, the red channel in bits 11:8, the green channel in bits 7:4, and the blue channel in bits 3:0
vivXBGR4444	16-bit XBGR format with the X channel in bits 15:12, the blue channel in bits 11:8, the green channel in bits 7:4, and the red channel in bits 3:0
vivRGBX4444	16-bit RGBX format with the red channel in bits 15:12, the green channel in bits 11:8, the blue channel in bits 7:4, and the X channel in bits 3:0
vivBGRX4444	16-bit BGRX format with the blue channel in bits 15:12, the green channel in bits 11:8, the red channel in bits 7:4, and the X channel in bits 3:0
vivARGB1555	16-bit ARGB format with the alpha channel in bit 15, the red channel in bits 14:10, the green channel in bits 9:5, and the blue channel in bits 4:0
vivABGR1555	16-bit ABGR format with the alpha channel in bit 15, the blue channel in bits 14:10, the green channel in bits 9:5, and the red channel in bits 4:0

2 Data structures

Enumeration value	Description
vivRGBA1555	16-bit RGBA format with the red channel in bits 15:11 , the green channel in bits 10:6, the blue channel in bits 5:1, and the alpha channel in bit 0
vivBGRA1555	16-bit BGRA format with the blue channel in bits 15:11, the green channel in bits 10:6, the red channel in bits 5:1, and the alpha channel in bit 0
vivXRGB1555	16-bit XRGB format with the X channel in bit 15, the red channel in bits 14:10, the green channel in bits 9:5, and the blue channel in bits 4:0
vivXBGR1555	16-bit XBGR format with the X channel in bit 15, the blue channel in bits 14:10, the green channel in bits 9:5, and the red channel in bits 4:0
vivRGBX1555	16-bit RGBX format with the red channel in bits 15:11, the green channel in bits 10:6, the blue channel in bits 5:1, and the X channel in bit 0
vivBGRX1555	16-bit BGRX format with the blue channel in bits 15:11, the green channel in bits 10:6, the red channel in bits 5:1, and the X channel in bit 0
vivRGB565	16-bit RGB format with the red channel in bits 15:11, the green channel in bits 10:5, and the blue channel in bits 4:0
vivBGR565	16-bit BGR format with the blue channel in bits 15:11, the green channel in bits 10:5, and the red channel in bits 4:0
vivARGB8888	32-bit ARGB format with the alpha channel in bits 31:24, the red channel in bits 23:16, the green channel in bits 15:8, and the blue channel in bits 7:0
vivABGR8888	32-bit ABGR format with the alpha channel in bits 31:24, the blue channel in bits 23:16, the green channel in bits 15:8, and the red channel in bits 7:0
vivRGBA8888	32-bit RGBA format with the red channel in bits 31:24, the green channel in bits 23:16, the blue channel in bits 15:8, and the alpha channel in bits 7:0
vivBGRA8888	32-bit BGRA format with the blue channel in bits 31:24, the green channel in bits 23:16, the red channel in bits 15:8, and the alpha channel in bits 7:0
vivXRGB8888	32-bit XRGB format with the X channel in bits 31:24, the red channel in bits 23:16, the green channel in bits 15:8, and the blue channel in bits 7:0
vivXBGR8888	32-bit XBGR format with the X channel in bits 31:24, the blue channel in bits 23:16, the green channel in bits 15:8, and the red channel in bits 7:0
vivRGBX8888	32-bit RGBX format with the red channel in bits 31:24, the green channel in bits 23:16, the blue channel in bits 15:8, and the X channel in bits 7:0
vivBGRX8888	32-bit BGRX format with the blue channel in bits 31:24, the green channel in bits 23:16, the red channel in bits 15:8, and the X channel in bits 7:0
vivARGB2101010	32-bit ARGB2101010 format with the alpha channel in bits 31:30, the red channel in bits 29:20, the green channel in bits 19:10, and the blue channel in bits 9:0
vivCURSOR_ARGB	The ARGB8888 format, which is supported for the cursor. For details about this format, see the description of vivARGB8888.

YUV formats

Enumeration value	Description
vivYUY2	Packed YUV422 format, 32 bits for 2 pixels, with Y0 in bits 31:24, U0 in bits 23:16, Y1 in bits 15:8, and V0 in bits 7:0.
vivUYVY	Packed YUV422 format, 32 bits for 2 pixels, with U0 in bits 31:24, Y0 in bits 23:16, V0 in bits 15:8, and Y1 in bits 7:0.

2 Data structures

Enumeration value	Description
vivNV16	YUV422 semi-planar format with 8 bits occupied by the Y plane and 16 bits occupied by the UV plane per pixel.
vivNV12	YUV420 semi-planar format, also named NV12, with an 8-bit Y plane in one pixel placed before each 16-bit array of packed U (Cb) and V (Cr) planes. The stride of the V and U planes is the same as that of the Y plane, but a V or U plane contains half of the lines in a Y plane.
vivYV12	YUV420 planar format with 8 bits occupied by the Y plane, 8 bits occupied by the U plane, and 8 bits occupied by the V plane per pixel.
vivP010	YUV420 semi-planar format. The Y plane occupies 16 bits per pixel but uses only 10 bits. The UV plane occupies 32 bits per pixel but uses only 20 bits.
vivNV12_10BIT	YUV420 semi-planar format, also named NV12, with a 10-bit Y plane in one pixel placed before each 20-bit array of packed U (Cb) and V (Cr) planes. Four Y-plane pixels occupy 5 bytes. The stride of the V and U planes is the same as that of the Y plane, but a V or U plane contains half of the lines in a Y plane. Only special DC8200 IPs support this format.
vivYUV444	YUV444 planar format with 8 bits occupied by the Y plane, 8 bits occupied by the U plane, and 8 bits occupied by the V plane per pixel. Only special DC8200 IPs support this format.
vivYUV444_10BIT	YUV444 planar format with 10 bits occupied by the Y plane, 10 bits occupied by the U plane, and 10 bits occupied by the V plane per pixel. Four Y-plane pixels occupy 5 bytes. Only special DC8200 IPs support this format.

2.3.3 viv_tiling_type

Specifies the tiling type.

Enumeration value	Description
vivLINEAR	Linear
vivTILED4X4 [1]	Tile 4x4
vivTILED8X8 [1]	Tile 8x8
vivSUPERTILEDX [1]	SupertileX 8x8 or 8x4
vivSUPERTILEDY [1]	SupertileY 4x8

1. Valid only for layers that support tiling. DC8000Nano supports tiling feature,to checkviv_layer_query_capability() can be called with vivLAYER_CAP_TILED as an argument.

The following table lists the tiling types supported for each input color format.

Note: *Each RGB color format listed includes their varients with the same channel bitwidth and different channel orders.*

Color format	Linear	Tile 4x4	Tile 8x8	SupertileX	SupertileY
vivARGB8888	Supported			8x4 Supported	Supported
vivXRGB8888	Supported			8x4 Supported	Supported
vivARGB2101010	Supported			8x4 Supported	Supported
vivRGB565	Supported			8x8 Supported	
vivARGB1555	Supported			8x8 Supported	
vivXRGB1555	Supported			8x8 Supported	
vivARGB4444	Supported			8x8 Supported	
vivXRGB4444	Supported			8x8 Supported	
vivYUY2	Supported		Supported		
vivUYVY	Supported		Supported		
vivNV12	Supported	Supported	Supported		
vivP010	Supported		Supported		
vivNV16	Supported				
vivYV12	Supported				
vivNV12_10BIT		Supported			
vivYUV444		Supported			
vivYUV444_10BIT		Supported			

2.3.4 viv_display_type

Specifies the output interface type.

To query whether an output interface is supported, call viv_dc_query_feature().

Enumeration value	Description	Supported (Yes/No/Not Applicable)
vivDPI	The Display Pixel Interface (DPI)	Yes
vivDP	The DisplayPort (DP)	No
vivEDP	Reserved	Not Applicable
vivDBI	The Display Bus Interface (DBI)	Yes

2.3.5 viv_dbi_type

Specifies the type of the Display Bus Interface (DBI).

This enumeration is valid only for display controllers that support DBI output. To query whether DBI output is supported, call the viv_dc_query_feature() API with vivFEATURE_DBI.

Enumeration value	Description
vivDBI_AFIXED	Type A Fixed E mode
vivDBI_ACLOCK	Type A Clocked E mode
vivDBI_B	Type B
vivDBI_C	Type C

2.3.6 viv_display_format_type

Specifies the color format of the display controller output.

DPI output

The following table lists the color formats supported for DPI output.

Enumeration value	Description
vivD24	RGB888
vivD30	RGB101010
vivD16CFG1	RGB565, config 1
vivD16CFG2	RGB565, config 2
vivD16CFG3	RGB565, config 3
vivD18CFG1	RGB666, config 1
vivD18CFG2	RGB666, config 2

DP RGB output

The following table lists the color formats supported for DP RGB output.

Enumeration value	Description
vivDPRGB565	RGB565

vivDPRGB666	RGB666
vivDPRGB888	RGB888
vivDPRGB101010	RGB101010

DP YUV output

The following table lists the color formats supported for DP YUV output.

Enumeration value	Description
vivDPYUV420B8CFG1	8-bit YUV420, config 1
vivDPYUV420B8CFG2	8-bit YUV420, config 2
vivDPYUV420B8CFG3	8-bit YUV420, config 3
vivDPYUV422B8CFG1	8-bit YUV422, config 1
vivDPYUV422B8CFG2	8-bit YUV422, config 2
vivDPYUV444B8CFG1	8-bit YUV444, config 1
vivDPYUV444B8CFG2	8-bit YUV444, config 2
vivDPYUV444B8CFG3	8-bit YUV444, config 3
vivDPYUV420B10CFG1	10-bit YUV420 , config 1
vivDPYUV420B10CFG2	10-bit YUV420, config 2
vivDPYUV420B10CFG3	10-bit YUV420, config 3
vivDPYUV422B10CFG1	10-bit YUV422, config 1
vivDPYUV422B10CFG2	10-bit YUV422, config 2
vivDPYUV444B10CFG1	10-bit YUV444, config 1
vivDPYUV444B10CFG2	10-bit YUV444, config 2
vivDPYUV444B10CFG3	10-bit YUV444, config 3

DBI output

The following table lists the color formats supported for DBI output. To query whether DBI output is supported, call viv_dc_query_feature() with vivFEATURE_DBI.

Enumeration value	Description
vivD8R3G3B2	D8R3G3B2 for DBI type A and type B
vivD8R4G4B4	D8R4G4B4 for DBI type A and type B
vivD8R5G6B5	D8R5G6B5 for DBI type A and type B
vivD8R6G6B6	D8R6G6B6 for DBI type A and type B
vivD8R8G8B8	D8R8G8B8 for DBI type A and type B
vivD9R6G6B6	D9R6G6B6 for DBI type A and type B
vivD16R3G3B2	D16R3G3B2 for DBI type A and type B
vivD16R4G4B4	D16R4G4B4 for DBI type A and type B
vivD16R5G6B5	D16R5G6B5 for DBI type A and type B
vivD16R6G6B6OP1	D16R6G6B6 option 1 for DBI type A and type B
vivD16R6G6B6OP2	D16R6G6B6 option 2 for DBI type A and type B

vivD16R8G8B8OP1	D16R8G8B8 option 1 for DBI type A and type B
vivD16R8G8B8OP2	D16R8G8B8 option 2 for DBI type A and type B
vivD1R5G6B5OP1	D1R5G6B5 option 1 for DBI type C
vivD1R5G6B5OP2	D1R5G6B5 option 2 for DBI type C
vivD1R5G6B5OP3	D1R5G6B5 option 3 for DBI type C
vivD1R8G8B8OP1	D1R8G8B8 option 1 for DBI type C
vivD1R8G8B8OP2	D1R8G8B8 option 2 for DBI type C
vivD1R8G8B8OP3	D1R8G8B8 option 3 for DBI type C

2.3.7 viv_filter_tap_type

Specifies the filter tap type.

In the following table, filter tap patterns are described in [horizontal tap count] x [vertical tap count] format.

Enumeration value	Description
vivFILTER_H3_V3	The filter with 3 x 3 taps
vivFILTER_H5_V3	The filter with 5 x 3 taps

2.3.8 viv_cursor_size_type

Specifies the cursor size.

Enumeration value	Description
vivCURSOR_32x32	32 x 32 pixels.
vivCURSOR_64x64	64 x 64 pixels. This value is valid only if the cursor version is 1. To query the cursor version, call viv_dc_query_feature() with vivFEATURE_CURSOR_VERSION.

2.3.9 viv_alpha_mode

Specifies whether to reverse the alpha value.

Enumeration value	Description
vivALPHA_NORMAL	Does not reverse the alpha value.
vivALPHA_INVERSED	Reverses the alpha value.

2.3.10 viv_cache_mode

Specifies the cache mode.

Enumeration value	Description
vivCACHE_NONE	No cache configured
vivCACHE_128	128 bytes
vivCACHE_256	256 bytes

2.3.11 viv_global_alpha_mode

Specifies the type of alpha value.

Enumeration value	Description
vivGALPHA_NORMAL	Pixel alpha values are used.
vivGALPHA_GLOBAL	The global alpha value is used.
vivGALPHA_SCALED	The used alpha value is scaled from pixel alpha values and the global alpha value.

2.3.12 viv_porter_duff_mode

Specifies the Porter-Duff blend mode.

In the following table, Sa indicates the source alpha value, Da indicates the destination alpha value, Sc indicates the source color value, and Dc indicates the destination color value.

Enumeration value	Description
vivPD_CLEAR	Clears destination pixels covered by the source to 0. • $\text{Result alpha value} = 0$ • $\text{Result color value} = 0$
vivPD_SRC	Replaces the destination pixels with the source pixels. • $\text{Result alpha value} = \text{Sa}$ • $\text{Result color value} = \text{Sc}$
vivPD_DST	Discards the source pixels and leaves the destination unchanged. • $\text{Result alpha value} = \text{Da}$ • $\text{Result color value} = \text{Dc}$
vivPD_SRC_OVER	Draws the source pixels over the destination pixels. • $\text{Result alpha value} = \text{Sa} + (1 - \text{Sa}) \times \text{Da}$ • $\text{Result color value} = \text{Sc} + (1 - \text{Sa}) \times \text{Dc}$
vivPD_DST_OVER	Draws the source pixels behind the destination pixels. • $\text{Result alpha value} = \text{Da} + (1 - \text{Da}) \times \text{Sa}$ • $\text{Result color value} = \text{Dc} + (1 - \text{Da}) \times \text{Sc}$
vivPD_SRC_IN	Keeps the source pixels that cover the destination pixels and discards the remaining source and destination pixels. • $\text{Result alpha value} = \text{Sa} \times \text{Da}$ • $\text{Result color value} = \text{Sc} \times \text{Da}$
vivPD_DST_IN	Keeps the destination pixels that cover source pixels and discards the remaining source and destination pixels. • $\text{Result alpha value} = \text{Sa} \times \text{Da}$ • $\text{Result color value} = \text{Sa} \times \text{Dc}$
vivPD_SRC_OUT	Keeps the source pixels that do not cover destination pixels and discards the remaining source and destination pixels. • $\text{Result alpha value} = \text{Sa} \times (1 - \text{Da})$ • $\text{Result color value} = \text{Sc} \times (1 - \text{Da})$

Enumeration value	Description
vivPD_DST_OUT	Keeps the destination pixels that are not covered by source pixels and discards the remaining source and destination pixels. • Result alpha value = $Da \times (1 - Sa)$ • Result color value = $Dc \times (1 - Sa)$
vivPD_SRC_ATOP	Discards the source pixels that do not cover destination pixels and draws the remaining source pixels over destination pixels. • Result alpha value = Da • Result color value = $Sc \times Da + (1 - Sa) \times Dc$
vivPD_DST_ATOP	Discards the destination pixels that are not covered by source pixels and draws the remaining destination pixels over source pixels. • Result alpha value = Sa • Result color value = $Sa \times Dc + Sc \times (1 - Da)$
vivPD_XOR	Discards the source and destination pixels where source pixels cover destination pixels and draws the remaining source pixels. • Result alpha value = $Sa + Da - 2 \times Sa \times Da$ • Result color value = $Sc \times (1 - Da) + (1 - Sa) \times Dc$

2.3.13 viv_pool_type

Specifies the memory allocation strategy.

Enumeration Value	Description
gcvPOOL_CONTIGUOUS	Allocates contiguous memory. If no contiguous memory is available, the allocation fails.
gcvPOOL_DEFAULT	Tries to allocate contiguous memory. If no contiguous memory is available, the system allocates memory with discrete pages.
gcvPOOL_USER	Uses user-reserved memory.

2.3.14 viv_display_size_type

Specifies the display resolution and refresh rate.

Enumeration value	Description
vivDISPLAY_320_480_60	320 x 480 resolution at a refresh rate of 60 Hz
vivDISPLAY_480_800_60	480 x 800 resolution at a refresh rate of 60 Hz
vivDISPLAY_480_864_60	480 x 864 resolution at a refresh rate of 60 Hz
vivDISPLAY_640_480_60	640 x 480 resolution at a refresh rate of 60 Hz
vivDISPLAY_720_480_60	720 x 480 resolution at a refresh rate of 60 Hz
vivDISPLAY_800_480_60	800 x 480 resolution at a refresh rate of 60 Hz
vivDISPLAY_1024_600_60	1024 x 600 resolution at a refresh rate of 60 Hz
vivDISPLAY_1024_768_60	1024 x 768 resolution at a refresh rate of 60 Hz
vivDISPLAY_1280_720_60	1280 x 720 resolution at a refresh rate of 60 Hz
vivDISPLAY_1920_1080_60	1920 x 1080 resolution at a refresh rate of 60 Hz

Enumeration value	Description
vivDISPLAY_3840_2160_30	3840 x 2160 resolution at a refresh rate of 30 Hz
vivDISPLAY_3840_2160_60	3840 x 2160 resolution at a refresh rate of 60 Hz
vivDISPLAY_4096_2160_60	4096 x 2160 resolution at a refresh rate of 60 Hz
vivDISPLAY_5760_756_60	5760 x 756 resolution at a refresh rate of 60 Hz
vivDISPLAY_CUSTOMIZED	Custom resolution

2.3.15 viv_display

Specifies the display panel.

Enumeration value	Description
vivDISPLAY_0	The display0 panel
vivDISPLAY_1	The display1 panel

2.3.16 viv_dc_features

Specifies the feature for viv_dc_query_feature() to query.

Enumeration value	Description	Support (Yes/No/Value)
vivFEATURE_DISPLAY_COUNT	The number of supported panels	1
vivFEATURE_LAYER_COUNT	The total number of supported overlay layers and video/graphic layers	3
vivFEATURE_CURSOR_COUNT	The number of supported cursor layers	1
vivFEATURE_GAMMA_BIT_OUT	The number of output bits of the display gamma module	8
vivFEATURE_SECURITY	The secure mode, Trusted Execution Environment (TEE)	No
vivFEATURE_MMU	The memory management unit (MMU)	No
vivFEATURE_CURSOR_VERSION	The cursor version • The value 0 indicates that only the cursor size 32 x 32 in pixels is supported. • Value 1 indicates that the cursor sizes 32 x 32 and 64 x 64 in pixels are supported.	1
vivFEATURE_CSC_MOUDLE	The programmable color space conversion (CSC) matrix	No
vivFEATURE_3D_LUT	The 3D lookup table (LUT) feature	No
vivFEATURE_DE_GAMMA	The degamma feature	No
vivFEATURE_DP	Display Port feature	No
vivFEATURE_DP_YUV	Display Port with YUV feature	No
vivFEATURE_DBI	The Display Bus Interface (DBI) output	Yes
vivFEATURE_DPI	The Display Parallel Interface (DPI) output	Yes
vivFEATURE_NEW_GAMMA	The new gamma feature	No
vivFEATURE_COLOR_BAR	The color bar	No
vivFEATURE_CRC	The cyclic redundancy check (CRC) feature	No

Enumeration value	Description	Support (Yes/No/Value)
vivFEATURE_40BIT_ADDRESS	The 40-bit address space	No
vivFEATURE_WRITEBACK	The write-back feature	No
vivFEATURE_PROGRAM_WB	Program Write Back buffer	No
vivFEATURE_CUSTOMER_TILE4X4	Custom tile alignment	Yes
vivFEATURE_DUAL_OS	The dual-OS feature.	No

2.3.17 viv_dc_layer_cap

Specifies the feature for viv_layer_query_capability() to query. Only vivLAYER_CAP_TILED is supported.

Enumeration Value	Description
vivLAYER_CAP_DEC400_DECOMPRESSION	The DEC400 decompression feature
vivLAYER_CAP_SCALE	The filtering feature for scaling and sharpness
vivLAYER_CAP_TILED	The tiling feature
vivLAYER_CAP_ROTATION	The full rotation feature
vivLAYER_CAP_ROI	The region of interest (ROI) feature

2.4 Structures

2.4.1 viv_dc_buffer

The structure to configure the buffer for a layer to access.

Member	Type	Description
handle[3]	gctPOINTER	(Optional) The handle of the buffer memory, obtained from the viv_alloc_buffer() API.
logical[3]	gctPOINTER	(Optional) A pointer to the CPU logical address of the buffer, obtained from the viv_alloc_buffer() API.
phyAddress[3]	gctUINT64	The physical addresses of the buffer.
gpuAddress[3]	gctUINT64	(Optional) The DPU virtual addresses of the buffer, obtained from the viv_alloc_buffer() API. This member is valid only for display controllers with MMU. To query the support for MMU, call viv_dc_query_feature() with vivFEATURE_MMU.
format	viv_input_format_type	The color format of the layer input. This parameter is valid only for overlay layers and video/graphic layers. For cursor layers, use the format member in the viv_cursor structure instead.

Member	Type	Description
security	gctBOOL	(Optional) Specifies whether to enable the secure mode for the buffer. The available values include: • vivTRUE: Enables the secure mode. • vivFALSE: Disables the secure mode. Set this member to the same value as the security parameter in the viv_alloc_buffer() API. This member is valid only if the display controller is configured with MMU and supports the secure mode. To query the support for the secure mode and MMU, call viv_dc_query_feature() with vivFEATURE_SECURITY and vivFEATURE_MMU separately.
pool	viv_pool_type	(Optional) The memory allocation strategy. Set this member to the same value as the Pool parameter in the viv_alloc_buffer() API.
tiling	viv_tiling_type	The tiling type of the layer input.
width	gctUINT32	The width of the layer input.
height	gctUINT32	The height of the layer input.
stride[3]	gctUINT32	The stride of each plane.

See also

3.2.1 viv_alloc_buffer()

3.3.2 viv_dc_query_feature()

3.4.3 viv_layer_set()

2.4.2 3.5.2 viv_set_cursor()viv_tilestatus_buffer

The structure to configure the tile status buffer for a layer.

Member	Type	Description
tileStatusHandle[3]	gctPOINTER	(Optional) The handle of the buffer memory, obtained from the viv_alloc_buffer() API.
tileStatusLogical[3]	gctPOINTER	(Optional) A pointer to the CPU logical address of the buffer, obtained from the viv_alloc_buffer() API.
tileStatusHWAddress[3]	gctUINT64	The physical addresses of the buffer.
tileStatusGPUAddress[3]	gctUINT64	(Optional) This member is valid only for display controllers with MMU. Not Supported
format	viv_input_format_type	(Optional) The color format of the layer input.
security	gctBOOL	(Optional) This member is valid only if the display controller is configured with MMU and supports the secure mode. To query the support for the secure mode and MMU. Not Supported
pool	viv_pool_type	(Optional) The memory allocation strategy.

Member	Type	Description
		Set this member to the same value as the Pool parameter in the viv_alloc_buffer() API.
tiling	viv_tiling_type	(Optional) The tiling type of the layer input.
width	gctUINT32	(Optional) The width of the layer input.
height	gctUINT32	(Optional) The height of the layer input.

See also

3.2.1 viv_alloc_buffer()

2.4.3 viv_cursor

The structure of a cursor.

Member	Type	Description
hsx	gctUINT32	The X offset, in pixels, of the top-left point to the hotspot.
hsy	gctUINT32	The Y offset, in pixels, of the top-left point to the hotspot.
x	gctUINT32	The X coordinate, in pixels, of the hotspot.
y	gctUINT32	The Y coordinate, in pixels, of the hotspot.
size	Not supported (viv_write_back_type)	The size of the cursor.
bg_color	gctUINT	The background color in the specified format.
fg_color	gctUINT	The foreground color in the specified format.
format	viv_input_format_type	The input color format of the cursor. Set this parameter to vivCURSOR_ARGB or leave it unspecified.

2.4.4 viv_layer_alpha_mode

The structure of the alpha value configurations.

Member	Type	Description
srcGlobalAlphaMode	viv_global_alpha_mode	The type of alpha value to use for the source.
srcGlobalAlphaValue	gctUINT32	The global alpha value for the source.
srcAlphaMode	viv_alpha_mode	Specifies whether to reverse the source alpha value.
srcAlphaValue	gctUINT32	Reserved.
dstGlobalAlphaMode	viv_global_alpha_mode	The type of the alpha value to use for the destination.
dstGlobalAlphaValue	gctUINT32	The global alpha value for the destination.
dstAlphaMode	viv_alpha_mode	Specifies whether to reverse the destination alpha value.
dstAlphaValue	gctUINT32	Reserved.

2.4.5 viv_dc_degamma

The structure of a degamma table.

Member	Type	Description
degammaTable[260][3]	gctUINT16	The degamma table with a size of 260 rows and 3 columns • degammaTable[*index*][0]: The value for the red channel • degammaTable[*index*][1]: The value for the green channel • degammaTable[*index*][2]: The value for the blue channel where, *index* is a value in the range of [0, 259].

2.4.6 viv_dc_gamma

The structure of a gamma table.

Member	Type	Description
gammaTable[260][3]	gctUINT16	The gamma table with a size of 260 rows and 3 columns • gammaTable[*index*][0]: The value for the red channel • gammaTable[*index*][1]: The value for the green channel • gammaTable[*index*][2]: The value for the blue channel where *index* is a value in the range of [0, 259].

2.4.7 viv_output

Specifies the output interface type, color format and DPI-related attributes.

Member	Type	Description
type	viv_display_type	The output interface type
format	viv_display_format_type	The output color format

2.4.8 viv_dc_rect

The structure of a rectangle.

Member	Type	Description
x	gctUINT32	The X coordinate, in pixels, of the rectangle top-left point
y	gctUINT32	The Y coordinate, in pixels, of the rectangle top-left point
w	gctUINT32	The width, in pixels, of the rectangle
h	gctUINT32	The height, in pixels, of the rectangle

2.4.9 viv_dc_color

The structure of an RGB color.

Member	Type	Description
a	gctUINT8	The value of the alpha channel
r	gctUINT8	The value of the red channel
g	gctUINT8	The value of the green channel
b	gctUINT8	The value of the blue channel

3 DPU APIs

3.1 Initialization

This section describes the APIs for display controller initialization, de-initialization, and reset.

3.1.1 viv_dc_init()

Description

Starts the display controller and initializes platform-related functions.

Call this API before using the display controller hardware. Do not repeatedly call this API before viv_dc_deinit() is called.

Syntax

```
vivSTATUS viv_dc_init(
        gctVOID
);
```

Parameters

None.

Returns

vivSTATUS

See also

3.1.2 viv_dc_deinit()

3.1.2 viv_dc_deinit()

Description

Terminates the platform-related functions and stops the display controller.

Call this API after the application is completed. After this API is executed, call viv_dc_init() before any other DPU API.

Syntax

```
vivSTATUS viv_dc_deinit(
        gctVOID
);
```

Parameters

None.

Returns

vivSTATUS

See also

3.1.1 viv_dc_init()

3.1.3 viv_dc_reset()

Description

Resets the display controller.

Syntax

```
vivSTATUS viv_dc_reset(
        gctVOID
);
```

Parameters

None.

Returns

vivSTATUS

3.2 Buffer allocation

This section describes the APIs for buffer management.

3.2.1 viv_alloc_buffer()

Description

Allocates system memory to a buffer.

Syntax

```
vivSTATUS viv_alloc_buffer(
        gctUINT32 Size,
        gctPOINTER *Handle,
        gctUINT32 *HardwareAddress,
        gctPOINTER *Logical,
        gctBOOL security,
        viv_pool_type Pool
);
```

Parameters

Parameter	Data type	Description
Size	gctUINT32	The memory size, in bytes, to allocate. For viv_dc_buffer: • If the input data is in RGB format, the required memory size is: $\text{height} \times \text{width} \times \text{Number of bytes per pixel (Bpp)}$ • If the input data is in YUV format, the required memory size is the sum of the result of the above formula for each plane. For viv_tilestatus_buffer, the recommended memory size is: $\text{Memory size of viv\_dc\_buffer} / 128$
Handle	gctPOINTER *	An output parameter that serves as the handle of the allocated memory.
HardwareAddress	gctUINT32 *	An output parameter that indicates the physical address of the memory buffer.
Logical	gctPOINTER *	An output parameter that indicates the current CPU logical address of the memory buffer.
security	gctBOOL	Specifies whether to enable the secure mode for the memory buffer. Set this parameter to one of the following values: • vivTRUE: Enables the secure mode. If the secure mode is enabled for the buffer, enable the secure mode for the layer with viv_cursor_security() or viv_layer_security() to ensure successful access from the layer to the buffer. • vivFALSE: Disables the secure mode. This parameter is valid only if the display controller is configured with MMU and supports the secure mode. To query the support for the

Parameter	Data type	Description
		secure mode and MMU, call viv_dc_query_feature() with vivFEATURE_SECURITY and vivFEATURE_MMU separately.
Pool	viv_pool_type	The memory allocation strategy. Set this parameter to gcvPOOL_CONTIGUOUS or gcvPOOL_DEFAULT. The value must be the same as that you set in the viv_dc_buffer or viv_tilestatus_buffer object.

Returns

vivSTATUS

See also

2.4.1 viv_dc_buffer

2.4.2 viv_tilestatus_buffer

3.2.2 viv_free_buffer()

3.3.2 viv_dc_query_feature()

3.5.3 viv_cursor_security()

3.2.2 viv_free_buffer()

Description

Frees the buffer memory allocated with viv_alloc_buffer().

Syntax

```
vivSTATUS viv_free_buffer(
        gctPOINTER handle
);
```

Parameters

Parameter	Data type	Description
handle	gctPOINTER	The handle of the buffer memory to free.

Returns

vivSTATUS

See also

3.2.1 viv_alloc_buffer()

3.3 Capability query

This section describes the APIs you can use to query the feature support of the display controller and each layer.

3.3.1 viv_query_chipinfo()

Description

Queries the list of features that the display controller supports. The order of the listed features follows that in the viv_dc_features structure, where the first feature is vivFEATURE_DISPLAY_COUNT.

Syntax

```
vivSTATUS viv_query_chipinfo (
       gctBOOL *Features
);
```

Parameters

Parameter	Data type	Description
Features	gctBOOL *	A pointer to the feature list.

Returns

vivSTATUS

See also

2.3.16 viv_dc_features

3.3.2 viv_dc_query_feature()

3.3.2 viv_dc_query_feature()

Description

Queries whether the display controller supports a specified feature. Before you call this API, use the viv_query_chipinfo() API to obtain the list of supported features.

To query the write-back support, use this API with vivFEATURE_WRITEBACK and the viv_display_query_capability() API with vivDISPLAY_CAP_PROGRAM_WB separately.

Syntax

```
vivSTATUS viv_dc_query_feature(
        viv_dc_features feature
        gctUINT* value
);
```

Parameters

Parameter	Data type	Description
feature	viv_dc_features	The feature to query.
value	gctUINT *	The return value. • If the queried feature is vivFEATURE_DISPLAY_COUNT, vivFEATURE_LAYER_COUNT, or vivFEATURE_CURSOR_COUNT, the return value indicates the number of panels or layers. • If the queried feature is vivFEATURE_CURSOR_VERSION: – The value 1 indicates that the cursor sizes 32 x 32 and 64 x 64 in pixels are supported. – The value 0 indicates that only the cursor size 32 x 32 in pixels is supported. • For other features: – The value 1 indicates that the queried feature is supported. – The value 0 indicates that the queried feature is not supported.

Returns

vivSTATUS

See also

3.3.1 viv_query_chipinfo()

3.3.3 viv_layer_query_capability()

Description

Queries whether a layer supports a specified feature.

Syntax

```
vivSTATUS viv_layer_query_capability(
        gctUINT layer_id,
        viv_dc_layer_cap cap
        gctUINT* value
);
```

Parameters

Parameter	Data type	Description
layer_id	gctUINT	The ID of the layer to query. For the ID of each layer, see Section 3.4.1, viv_dc_select_layer().
Cap	viv_dc_layer_cap	The feature to query.
value	gctUINT *	The return value. The possible values include: • vivTRUE: The queried feature is supported. • vivFALSE: The queried feature is not supported.

Returns

vivSTATUS

3.4 Overlay and video/graphic layers

This section describes the APIs you can use to configure the overlay and video/graphic layers. For the support of overlay and video/graphic layers by each display controller, see Section 3.4.1, viv_dc_select_layer().

3.4.1 viv_dc_select_layer()

Description

Selects an overlay or video/graphic layer by ID. The ID of each layer for different display controller hardware revisions is listed in the following table.

Hardware revision	Software-hardware layer mapping
DC8000Nano 5_5_4_rc3d	Software: {0, 1, 2} Hardware: {video0, overlay0, overlay1}

Before configuring an overlay or video/graphic layer, call this API to switch to the layer.

Syntax

```
vivSTATUS viv_dc_select_layer (
        gctUINT  layerId
);
```

Parameters

Parameter	Data type	Description
layerId	gctUINT	The ID of the layer to select.

Returns

vivSTATUS

3.4.2 viv_layer_enable()

Description

Enables or disables the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_layer_enable (
        gctBOOL enable
);
```

Parameters

Parameter	Data type	Description
Enable	gctBOOL	Specifies whether to enable the layer. Set this parameter to one of the following values: • vivTRUE: Enables the layer. • vivFALSE: Disables the layer.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.3 viv_layer_set()

Description

Assigns a buffer to the selected overlay or video/graphic layer.

Before you call this API, make sure that the buffer memory is available. To use system memory, call `viv_alloc_buffer()` for memory allocation.

Syntax

```
vivSTATUS viv_layer_set(
        viv_dc_buffer *buffer
);
```

Parameters

Parameter	Data type	Description
buffer	viv_dc_buffer *	A pointer to the buffer.

Returns

vivSTATUS

See also

3.2.1 viv_alloc_buffer()

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.4 viv_layer_zorder()

Description

Configures the z-order for the selected overly or video/graphic layer in a blending or color keying operation. For the blending and color keying description, see the Vivante hardware features document specific to the DPU hardware version.

Syntax

```
vivSTATUS viv_layer_zorder(
        gctUINT8 zorder
);
```

Parameters

Parameter	Data type	Description
zorder	gctUINT8	The z-order of the layer. The value 0 indicates the bottom layer sitting above the background layer. Other numbers index the ordering of the front layers. The default background color is RGB(0, 0, 0). To change the background color, use the viv_layer_set_background() API.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.5.1 viv_layer_set_background()

3.4.5 viv_layer_set_position()

Description

Sets the start coordinates for the selected overlay or video/graphic layer to display on a panel.

Syntax

```
vivSTATUS viv_layer_set_position(
        gctUINT   x,
        gctUINT   y
);
```

Parameters

Parameter	Data type	Description
x	gctUINT	The X coordinate of the start point for the layer.
y	gctUINT	The Y coordinate of the start point for the layer.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.6 viv_layer_colorkey()

Description

Configures the color keying feature for the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_layer_colorkey(
        viv_dc_color *colorkey,
        viv_dc_color *colorkeyHigh,
        gctBOOL transparency
);
```

Parameters

Parameter	Data type	Description
colorkey	viv_dc_color *	A pointer to the low color value of the range to key out.
colorkeyHigh	viv_dc_color *	A pointer to the high color value of the range to key out.
transparency	gctBOOL	Specifies whether to enable transparency of the matching range at the layer. Set this parameter to one of the following values: • vivTRUE: Enables transparency • vivFALSE: Disables transparency

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.7 viv_layer_clear()

Description

Configures the clear feature for the selected overlay or video/graphic layer.

If the clear feature is enabled for the layer, the system takes the clear color as the source data of the layer and does not read data from the buffer of the layer.

If the layer clear feature is disabled and the DEC400 fast clear feature is enabled, the clear color is used as the DEC400 fast clear color. The system determines whether DEC400 fast clear is enabled based on the compression information in the buffer specified with the viv_layer_decompress() API.

Syntax

```
vivSTATUS viv_layer_clear(
        viv_dc_color *clearColor,
        gctBOOL enable
);
```

Parameters

Parameter	Data type	Description
clearColor	viv_dc_color *	A pointer to the clear color.
enable	gctBOOL	Specifies whether to enable the clear feature for the layer. Set this parameter to one of the following values: • vivTRUE: Enables the clear feature. • vivFALSE: Disables the clear feature.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.8 viv_set_alpha()

Description

Configures the source and destination alpha values for the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_set_alpha(
        viv_layer_alpha_mode *Alpha
);
```

Parameters

Parameter	Data type	Description
Alpha	viv_layer_alpha_mode *	A pointer to the alpha value configurations of the layer.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.9 viv_layer_poterduff_blend()

3.4.9 viv_layer_poterduff_blend()

Description

Enables or disables alpha blending for the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_layer_poterduff_blend(
        gctBOOL   enable,
        viv_porter_duff_mode Mode
);
```

Parameters

Parameter	Data type	Description
Enable	gctBOOL	Specifies whether to enable alpha blending. Set this parameter to one of the following values: • vivTRUE: Enables alpha blending. • vivFALSE: Disables alpha blending.
Mode	viv_porter_duff_mode	The Porter-Duff blend mode.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.8 viv_set_alpha()

3.4.10 viv_layer_set_watermark()

Description

Sets the watermark value for the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_layer_set_watermark(
        gctUINT32     watermark
);
```

Parameters

Parameter	Data type	Description
watermark	gctUINT32	The watermark value.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.4.11 viv_dc_set_qos()

3.4.11 viv_dc_set_qos()

Description

Sets the DPU QoS values for all overlay layers and video/graphic layers.

Syntax

```
vivSTATUS viv_dc_set_qos(
        gctUINT32     low,
        gctUINT32     high
);
```

Parameters

Parameter	Data type	Description
low	gctUINT32	The QoS low value.
high	gctUINT32	The QoS high value.

Returns

vivSTATUS

See also

3.4.10 viv_layer_set_watermark()

3.4.12 viv_layer_set_display()

Description

Selects the display panel for the selected overlay or video/graphic layer.

Syntax

```
vivSTATUS viv_layer_set_display(
        viv_display     display
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.

Returns

vivSTATUS

See also

3.4.1 viv_dc_select_layer()

3.4.2 viv_layer_enable()

3.5 Background and cursor layers

This section describes the APIs you can use to configure background layers and cursor layers.

DC8000Nano supports only one background layer and one cursor layer.

3.5.1 viv_layer_set_background()

Description

Sets the background color for a display panel. The panel shows the background color in a region only if the color bar is disabled for the region.

Syntax

```
vivSTATUS viv_layer_set_background(
        viv_display    display,
        viv_dc_color     *bgColor
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
bgColor	viv_dc_color *	A pointer to the background color, which is default to RGB(0,0,0).

Returns

vivSTATUS

3.5.2 viv_set_cursor()

Description

Enables or disables the cursor layer for a display panel.

Before you call this API, make sure that a viv_dc_buffer object is configured for the layer. If you want the buffer to use system memory, call viv_alloc_buffer() for memory allocation.

Syntax

```
vivSTATUS viv_set_cursor(
        viv_display  display,
        viv_dc_buffer  *buffer,
        viv_cursor  *cursor,
        gctBOOL  enable
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel whose cursor layer you want to configure. *Note: The cursor of the display0 panel is cursor0. The cursor of the display1 panel is cursor1.*
buffer	viv_dc_buffer *	A pointer to the buffer of the cursor layer.
cursor	viv_cursor *	A pointer to the cursor.
enable	gctBOOL	Specifies whether to enable the cursor layer. Set this parameter to one of the following values: • vivTRUE: Enables the cursor layer. • vivFALSE: Disables the cursor layer.

Returns

vivSTATUS

See also

3.2.1 viv_alloc_buffer()

3.5.3 viv_cursor_security()

3.5.4 viv_cursor_offset()

3.5.5 viv_cursor_move()

3.5.3 viv_cursor_security()

Description

Enables or disables the cursor layer in secure mode for a display panel.

This API is valid only for DC8200 that supports the secure mode. Before using this API, call viv_dc_query_feature() with vivFEATURE_SECURITY to check whether the secure mode is supported.

Note: Calling this API overwrites the setting of the enable parameter in viv_set_cursor().

Syntax

```
vivSTATUS  viv_cursor_security(
       viv_display display,
       gctBOOL enable
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel whose cursor layer you want to configure. *Note: The cursor of the display0 panel is cursor0. The cursor of the display1 panel is cursor1.*
enable	gctBOOL	Specifies whether to enable the cursor layer. Set this parameter to one of the following values: • vivTRUE: Enables the cursor layer. • vivFALSE: Disables the cursor layer.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.5.2 viv_set_cursor()

3.5.4 viv_cursor_offset()

Description

Sets the offset of the cursor top-left point to the hotspot. The top-left point refers to the upper-left corner of the cursor bounding box.

Make sure that the hotspot is located within the cursor image. To query the supported cursor sizes, call viv_dc_query_feature() with vivFEATURE_CURSOR_VERSION.

Syntax

```
vivSTATUS viv_cursor_hotspot(
        viv_display display,
        gctUINT32 hsx,
        gctUINT32 hsy
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel whose cursor layer you want to configure. *Note: The cursor of the display0 panel is cursor0. The cursor of the display1 panel is cursor1.*
hsx	gctUINT32	The X offset, in pixels, of the top-left point to the hotspot.
hsy	gctUINT32	The Y offset, in pixels, of the top-left point to the hotspot.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.5.2 viv_set_cursor()

3.5.5 viv_cursor_move()

Description

Sets a new position for the cursor hotspot on the same display panel.

Syntax

```
vivSTATUS viv_cursor_move(
        viv_display display,
        gctUINT32 x,
        gctUINT32 y
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel whose cursor layer you want to configure. *Note:* *The cursor of the display0 panel is cursor0. The cursor of the display1 panel is cursor1.*
x	gctUINT32	The new position X coordinate, in pixels, of the cursor hotspot.
y	gctUINT32	The new position Y coordinate, in pixels, of the cursor hotspot.

Returns

vivSTATUS

See also

3.5.2 viv_set_cursor()

3.6 Post-processing for display

This section describes the APIs for post-processing configurations.

3.6.1 viv_gamma_enable()

Description

Enables or disables the gamma feature for a display panel.

Syntax

```
vivSTATUS viv_gamma_enable(
       viv_display display,
       gctBOOL   enable
);
```

Parameters

Parameter	Data type	Description
Display	viv_display	The ID of the display panel.
enable	gctBOOL	Specifies whether to enable the gamma feature. Set this parameter to one of the following values: • vivTRUE: Enables the feature. • vivFALSE: Disables the feature.

Returns

vivSTATUS

See also

3.6.2 viv_gamma_init()

3.6.3 viv_set_gamma()

3.6.2 viv_gamma_init()

Description

Initializes the gamma table.

Syntax

```
vivSTATUS viv_gamma_init(
        viv_dc_gamma  *gamma,
        gctFLOAT gamma_value,
        viv_dc_curve_type curve_type
);
```

Parameters

Parameter	Data type	Description
gamma	viv_dc_gamma *	A pointer to the gamma table.
gamma_value	gctFLOAT	The index of the gamma function. This parameter is valid only if curve_type is set to VIV_DC_CURVE_GAMMA.

Returns

vivSTATUS

See also

3.6.1 viv_gamma_enable()

3.6.3 viv_set_gamma()

3.6.3 viv_set_gamma()

Description

Sets a row in a gamma table for a display panel.

Before you use this API, make sure that the table of the gamma feature is available. To initiate the table, use the viv_gamma_init() API.

Syntax

```
vivSTATUS viv_set_gamma(
        viv_display display,
        gctUINT32 index,
        gctUINT16 r,
        gctUINT16 g,
        gctUINT16 b
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
index	gctUINT32	The index of the row in the gamma table.
r	gctUINT16	The gamma correction value for the red channel.
g	gctUINT16	The gamma correction value for the green channel.
b	gctUINT16	The gamma correction value for the blue channel.

Returns

vivSTATUS

See also

3.6.1 viv_gamma_enable()

3.6.2 viv_gamma_init()

3.6.4 viv_set_dither()

Description

Enables or disables the dithering feature for a display panel.

Syntax

```
vivSTATUS viv_set_dither(
        viv_display display,
        gctBOOL enable
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
enable	gctBOOL	Specifies whether to enable the dithering feature. Set this parameter to one of the following values: • vivTRUE: Enables dithering. • vivFALSE: Disables dithering.

Returns

vivSTATUS

3.7 Display output

This section describes the APIs you can use to configure display output on panels.

Before configuring display output, call viv_dc_query_feature() with vivFEATURE_DISPLAY_COUNT to query the number of panels supported by the display controller. The current number of supported panels is 1.

3.7.1 viv_set_display_size()

Description

Sets the display resolution and refresh rate of a display panel.

Syntax

```
vivSTATUS viv_set_display_size(
        viv_display    display,
        viv_display_size_type type
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
type	viv_display_size_type	The display resolution and refresh rate of the display panel.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.7.2 viv_set_custom_display_size()

Description

Sets the custom display resolution and refresh rate of a display panel.

Syntax

```
vivSTATUS viv_set_custom_display_size(
        viv_display    display,
        gctUINT      hactive,
        gctUINT      hsync_start,
        gctUINT      hsync_end,
        gctUINT      htotal,
        gctUINT      vactive,
        gctUINT      vsync_start,
        gctUINT      vsync_end,
        gctUINT      vtotal
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
hactive	gctUINT	The active resolution in the horizontal direction.
hsync_start	gctUINT	The synchronization start of the horizontal direction, which is calculated as follows: $$\text{hsync\_start} = \text{hactive} + \text{h\_front\_porch}$$ Where h_front_porch indicates the front porch of horizontal synchronization signals.
hsync_end	gctUINT	The synchronization end of horizontal direction, which is calculated as follows: $$\text{hsync\_end} = \text{hsync\_start} + \text{h\_sync\_length}$$ Where h_sync_length indicates the width of horizontal synchronization signals.
htotal	gctUINT	The total resolution in the horizontal direction, which is calculated as follows: $$\text{htotal} = \text{hsync\_end} + \text{h\_back\_porch}$$ Where h_back_porch indicates the back porch of horizontal synchronization signals.
vactive	gctUINT	The active resolution in the vertical direction.
vsync_start	gctUINT	The synchronization start of vertical direction, which is calculated as follows: $$\text{vsync\_start} = \text{vactive} + \text{v\_front\_porch}$$ Where v_front_porch indicates the front porch of vertical synchronization signals.
vsync_end	gctUINT	The synchronization end of vertical direction, which is calculated as follows: $$\text{vsync\_end} = \text{vsync\_start} + \text{v\_sync\_length}$$ Where v_sync_length indicates the width of vertical synchronization signals.
vtotal	gctUINT	The total resolution of vertical direction, which is calculated as follows: $$\text{vtotal} = \text{vsync\_end} + \text{v\_back\_porch}$$

Parameter	Data type	Description
		Where v_back_porch indicates the back porch of vertical synchronization signals.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.7.3 viv_set_output()

Description

Enables or disables the output of a display panel and sets the output format.

If the Display Bus Interface (DBI) is selected as the output interface, call viv_reset_dbi() to reset it to the idle state.

Syntax

```
vivSTATUS viv_set_output(
        viv_display    display,
        viv_output    *output,
        gctBOOL        enable
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
output	viv_output *	A pointer to the output configurations.
enable	gctBOOL	Specifies whether to enable the output for the display panel. Set this parameter to one of the following values: • vivTRUE: Enables the output. Pixels are displayed on the display panel through the specified output interface. • vivFALSE: Disables the output. All pixels are black. This allows a display panel to have correct timing without pixel display.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.7.4 viv_reset_dbi()

3.7.4 viv_reset_dbi()

Description

Resets the Display Bus Interface (DBI) to the idle state. Call this API if the DBI is selected as the output interface by using viv_set_output().

Syntax

```
vivSTATUS viv_reset_dbi(
        gctVOID
);
```

Parameters

None.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.7.2 viv_set_output()

3.7.5 viv_set_output_dbi()

Description

Enables or disables DBI output and sets the DBI type for a display panel.

.

Syntax

```
vivSTATUS viv_set_output_dbi(
        viv_display    display,
        viv_dbi_type   type
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel.
type	viv_dbi_type	The DBI type.

Returns

vivSTATUS

See also

3.3.2 viv_dc_query_feature()

3.8 Display controller access and debugging

This section describes the APIs that you can use to commit configurations and for debugging.

3.8.1 viv_set_commit()

Description

Commits the configurations to shadow registers and triggers display panels.

Note: *The shadow registers pass the configurations to the counterpart registers at the next frame. To check whether a frame is finished, use the viv_get_vblank_count() API.*

Syntax

```
vivSTATUS viv_set_commit(
        gctUINT32     display_mask
);
```

Parameters

Parameter	Data type	Description
display_mask	gctUINT32	Specifies the display panels to trigger. Set this parameter to one of the following values: • 1: Triggers display0. • 2: Triggers display1. • 3: Triggers both display0 and display1.

Returns

vivSTATUS

See also

3.8.2 viv_get_vblank_count()

3.8.2 viv_get_vblank_count()

Description

Queries the number of interrupts caused. An interrupt is generated once a DC frame is finished.

You can use this API to check whether frames are finished. For details, see Section 4, Programming with DPU APIs.

Syntax

```
vivSTATUS viv_get_vblank_count(
        viv_display display,
        gctUINT32 *count
);
```

Parameters

Parameter	Data type	Description
display	viv_display	The ID of the display panel
count	gctUINT32 *	An output parameter that indicates the number of interrupts

Returns

vivSTATUS

4 Programming with DPU APIs

The general procedure of programming with DPU APIs is as follows:

1. Start the display controller device with the viv_dc_init() API.
2. Reset the display controller with the viv_dc_reset() API.
3. Query the list of features supported by the hardware with the viv_query_chipinfo() and viv_dc_query_feature() APIs.
4. (Optional) Allocate buffers for layers with the viv_alloc_buffer() API.

   Skip this step if user-reserved memory is used.

5. Program the display controller by using the APIs described in the following sections:
   - Section 3.4, Overlay and video/graphic layers
   - Section 3.5, Background and cursor layers
   - Section 3.7, Display output
   - Section 3.6, Post-processing for display
6. Configure the hardware registers and trigger panels with the viv_set_commit() API.

   The configurations take effect after the current frame.

7. Check whether the current frame finishes, based on the number of caused interrupts queried with the viv_get_vblank_count() API.
8. After the frame is finished, disable the layers and the write-back feature that are enabled in Steps 5 and 7.

   This ensures that the display controller no longer has read or write requests.

   For the APIs to use, see Section 3.4.2, viv_layer_enable(), Section 3.5.2, viv_set_cursor()

4.1 Example 1: Displaying through DPI

This section provides an example of using the DPU API for image display through the DPI output interface. This example shows Layer 0 from (0,0) to (640,480) on the display0 panel with the output format vivD24. Layer 0 is fed with ARGB8888-formatted linear data in the size of 640x480.

For detailed description of the programing settings, see Chapter 2, Data structures and Chapter 3, DPU APIs.

```
gctINT ret = 0;
gctUINT width = 640, height = 480, Bpp = 4, stride = 640*4, phyAddr = 0, vblank0 =
0,
vblank1 = 0;
void *logical = 0, *handle = 0;
viv_dc_buffer buffer = {0};
viv_dc_rect display_rect = {0};
viv_output display_output = {0};
/* open device */
ret = viv_dc_init();
if(ret)
     return ret;
/* reset DC */
viv_dc_reset();
/* select layer0 */
viv_dc_select_layer(0);
/* enable layer0 */
viv_layer_enable(vivTRUE);
/* alloc contiguous memory for the frame buffer of layer0 */
ret = viv_alloc_buffer(width*height*Bpp, &handle, &phyAddr, &logical, vivFALSE,
gcvPOOL_DEFAULT);
if(ret)
     return ret;
/* config the buffer's phyAddr/format/tilemode/bufferWidth/bufferHeight/stride to
kernel */
buffer.handle[0] = handle;
buffer.logical[0] = logical;
buffer.phyAddress[0] = phyAddr;
buffer.stride[0] = stride;
buffer.format = vivARGB8888;
buffer.tiling = vivLINEAR;
buffer.width = 640;
buffer.height = 480;
viv_layer_set(&buffer);
/* config display region on panel */
display_rect.x = 0;
display_rect.y = 0;
display_rect.w = 640;
display_rect.h = 480;
/* if display region is equal to layer0's bufferSize, don't do scale */
viv_layer_scale(&display_rect, vivFILTER_H3_V3);
/* config the start coordinates of layer0 display region */
```

```
viv_layer_set_position(0, 0);
/* config layer's zorder number */
viv_layer_zorder(0);
/* config layer0 show on panel0 */
viv_layer_set_display(vivDISPLAY_0);
/* config panel0's resolution and timing */
viv_set_display_size(vivDISPLAY_0, vivDISPLAY_640_480_60);
/* config panel0's output type and format */
display_output.type = vivDPI;
display_output.format = vivD24;
viv_set_output(vivDISPLAY_0, &display_output, vivTRUE);
/* fill data to layer0's buffer */
memset(logical, 100, width*height*Bpp);
/* config register and trig panel0 */
viv_set_commit(0x1);
/* when vblank1 > vblank0 means one frame finished */
/* commit completes after the current frame is processed. */
viv_get_vblank_count(vivDISPLAY_0, &vblank0);
do{
     usleep(10000);
     viv_get_vblank_count(vivDISPLAY_0, &vblank1);
}while(vblank0 == vblank1);
/* we need to disable layer or cursor we have enabled before */
viv_layer_enable(vivFALSE);
/* commit again to let layer0 disabled */
viv_set_commit(0x1);
/* commit completes after the current frame is processed. */
/* new config will take effect at next frame */
viv_get_vblank_count(vivDISPLAY_0, &vblank0);
do{
     usleep(10000);
     viv_get_vblank_count(vivDISPLAY_0, &vblank1);
}while(vblank0 == vblank1);
/* free memory allocated for layer0 */
viv_free_buffer(buffer.handle[0]);
```

4.2 Example 2: Displaying through DBI Type B

This section provides an example of using the DPU API for image display through the DBI Type B interface. This example shows Layer 0 from (0,0) to (320,480) on the display0 panel with the output format D8R8G8B8. Layer 0 is fed with ARGB8888-formatted linear data in the size of 320x480.

For detailed description of the programing settings, see Chapter 2, Data structures and Chapter 3, DPU APIs.

```
gctINT ret = 0;
gctUINT width = 320, height = 480, Bpp = 4, stride = 320*4, phyAddr = 0, vblank0 =
0,
vblank1 = 0;
void *logical = 0, *handle = 0;
viv_dc_buffer buffer = {0};
viv_dc_rect display_rect = {0};
viv_output display_output = {0};
viv_dbi_type dbi_type = vivDBI_B;
/* open device */
ret = viv_dc_init();
if(ret)
     return ret;
/* reset DC */
viv_dc_reset();
/* select layer0 */
viv_dc_select_layer(0);
/* enable layer0 */
viv_layer_enable(vivTRUE);
/* alloc contiguous memory for the frame buffer of layer0 */
ret = viv_alloc_buffer(width*height*Bpp, &handle, &phyAddr, &logical, vivFALSE,
gcvPOOL_DEFAULT);
if(ret)
     return ret;
/* config the buffer's phyAddr/format/tilemode/bufferWidth/bufferHeight/stride to
kernel */
buffer.handle[0] = handle;
buffer.logical[0] = logical;
buffer.phyAddress[0] = phyAddr;
buffer.stride[0] = stride;
buffer.format = vivARGB8888;
buffer.tiling = vivLINEAR;
buffer.width = width;
buffer.height = height;
viv_layer_set(&buffer);
/* config display region on panel */
display_rect.x = 0;
display_rect.y = 0;
display_rect.w = width;
display_rect.h = height;
/* if display region is equal to layer0's bufferSize, don't do scale */
viv_layer_scale(&display_rect, vivFILTER_H3_V3);
```

```
/* config the start coordinates of layer0 display region */
viv_layer_set_position(0, 0);
/* config layer's zorder number */
viv_layer_zorder(0);
/* config layer0 show on panel0 */
viv_layer_set_display(vivDISPLAY_0);
/* config panel0's resolution and timing */
viv_set_display_size(vivDISPLAY_0, vivDISPLAY_320_480_60);
/* config panel0's output type and format */
display_output.type = vivDBI;
display_output.format = vivD8R8G8B8;
viv_set_output(vivDISPLAY_0, &display_output, vivTRUE);
viv_reset_dbi();
viv_set_output_dbi(dbi_type);
/* fill data to layer0's buffer */
memset(logical, 100, width*height*Bpp);
/* config register and trig panel0 */
viv_set_commit(0x1);
/* when vblank1 > vblank0 means one frame finished */
/* commit completes after the current frame is processed. */
viv_get_vblank_count(vivDISPLAY_0, &vblank0);
do{
     usleep(10000);
     viv_get_vblank_count(vivDISPLAY_0, &vblank1);
}while(vblank0 == vblank1);
/* we need to disable layer or cursor we have enabled before */
viv_layer_enable(vivFALSE);
/* commit again to let layer0 disabled */
viv_set_commit(0x1);
/* commit completes after the current frame is processed. */
/* new config will take effect at next frame */
viv_get_vblank_count(vivDISPLAY_0, &vblank0);
do{
     usleep(10000);
     viv_get_vblank_count(vivDISPLAY_0, &vblank1);
}while(vblank0 == vblank1);
/* free memory allocated for layer0 */
viv_free_buffer(buffer.handle[0]);
```

5 List of Unsupported DPU APIs

This section lists unsupported functions corresponding to the list of unsupported features for DCNano8000 hardware. These functions and their structures are removed from this documentation. Their implementation is available within the code but is currently not supported by the current hardware.

- viv_map_buffer
- viv_unmap_buffer
- viv_layer_security
- viv_layer_rotation
- viv_layer_cache_mode
- viv_layer_roi_enable
- viv_layer_roi_rect
- viv_layer_set_y2r
- viv_layer_degamma_enable
- viv_layer_degamma_init
- viv_layer_set_degamma
- viv_layer_set_r2r
- viv_layer_get_status
- viv_dc_request
- viv_set_color_bar
- viv_set_3d_lut
- viv_set_3d_lut_enlarge
- viv_set_output_csc
- viv_set_dest
- viv_set_writeback_dither

Revision history

Document revision	Date	Description of changes
*A	2025-09-24	Release to web.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-09-24

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-40807 Rev. *A

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.