DLP LightCommander API

Application Report

Literature Number: DLPA024C
June 2010–Revised May 2013

Contents

DLP LightCommander™ API Reference Manual

1 Overview

The DLP LightCommander API (DLPLC API) provides various software (SW) functions for controlling the DLP Light Commander Kit and chipset. The DLPC200 controller is included in the chipset and performs various image processing and control functions.

The DLPLC API allows clients to control different DLPC200 features like downloading image data, reading status, and configuring registers and Look-Up Tables (LUTs). The DLPLC API also supports USB 2.0 communication using the Cypress CY7C68013 USB peripheral controller chip.

The real-time loading of LUTs and registers is accomplished by executing the commands in a configuration batchfile, which is accomplished by executing a SW application that implements the RunBatch- File API. This API has an internal mapping between batchfile commands versus API functions that allows it to call an API function whenever its corresponding command is encountered. Please see Batchfile Language Specification V1.0 for further details.

See the links below for online resources and associated data sheets for additional details:

- API Functions – See DLP LightCommander API Functions
- API Data Types – See DLP LightCommander API Data Types, Variables, and Constants
- Batchfile Syntax – See Batchfile Language Specification V1.0
- Version Number – API Version Number and Release Notes
- Online resources:
  - http://www.ti.com/DLPLightCommander
  - http://www.logicpd.com/products/development-kits/dlp-lightcommander-development-kit

2 Module Index

3 File Index

Here is a list of all documented files:

- PortabilityLayer.h

4 Module Documentation

4.1 DLP LightCommander API Functions

Modules

- Display APIs
- Flash Compile APIs
- Flash Program APIs
- Image Transfer APIs
- LED APIs
- MISC APIs
- Register and LUT Read/Write APIs
- Data Source APIs
- Video Test Pattern Generator APIs
- Sync APIs
- Status APIs
- Batchfile APIs

4.2 Display APIs

Functions

- UINT8 DLP_Display_DisplayPatternManualStep()
- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)
- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)

The Display APIs are used control the display and appearance of image data. When displaying structured light (SL) patterns from external frame memory, some of the Display APIs can be used to start/stop/step through each SL image.

4.2.1 UINT8 DLP_Display_DisplayPatternManualStep ()

Repeatedly displays the next structured light image (as defined in Image Order LUT).

Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns as defined in the ImageOrder LUT.

If multiple image patterns are available for display, then each pattern can be displayed upon request by calling this function (repeatedly). When using an internally-programmed VSYNC, a single pattern will be displayed continuously, which allows for (human) visual inspection.

Returns

0 for success

4.2.2 UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()

Repeatedly displays the first SL image (as defined in Image Order LUT).

Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the 1st pattern as defined in the ImageOrder LUT.

Use this function in conjunction with DLP_Display_DisplayPatternManualStep to start at the beginning of the ImageOrder LUT and index through it to see of its corresponding images.

Returns

0 for success

4.2.3 UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()

Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns.

If multiple image patterns are available for display, then the series of patterns can be displayed by calling this function (once). If there are 10 patterns available for display, calling this function will display all 10 patterns, then disable the display.

Returns

0 for success

NOTE: This API is supported from MCU Version #2.1.5 onwards

4.2.4 UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()

Continuously displays all structured light images (as defined in Image Order LUT), one image per frame

Once bit plane patterns have been loaded into external memory, and the DLP Control Chip is fully configured, this function can be called to command the DMD and DLP Controller Chip to display the patterns.

If multiple image patterns are available for display, then the series of patterns can repeatedly be displayed by calling this function (once). If there are 10 patterns available for display, calling this function will display all 10 patterns then repeat.

Returns

0 for success

4.2.5 UINT8 DLP_Display_DisplayStop ()

This function will disable the display of bit plane patterns.

Returns

0 for success

4.2.6 UINT8 DLP_Display_ParkDMD ()

Issue a park DMD command.

Returns

0 for success

4.2.7 UINT8 DLP_Display_UnparkDMD ()

Issue an unpark DMD command

Returns

0 for success

4.2.8 UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)

Enables/disables degamma function.

The degamma function is typically associated with video data and is used to linearize the light output (by removing the applied Gamma transfer function).

When enabling degamma, a corresponding LUT must have already been programmed with non-zero values. The degamma LUT is an output of the LOGIC application SW.

Parameters

enableBit set to non-zero to enable, set to 0 to disable degamma

Returns

0 for success

4.2.9 UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)

Enable/disable horizontal flip of the display.

Parameters

enableBit set to non-zero to enable, set to 0 to disable

Returns

0 for success

4.2.10 UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)

Enable/disable vertical flip of the display.

Parameters

enableBit set to non-zero to enable, set to 0 to disable

Returns

0 for success

4.3 Flash Compile APIs

Functions

- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()

The Flash Compile APIs are used when converting a batchfile (produced by the LOGIC application SW) into binary data. This binary data can then be stored in the parallel (="user") flash for power-up configuration of the DLP Controller Chip. It should be noted, that normally a configuration batchfile sends commands to the DLP Controller Chip across USB or SPI, but in this context, these comm packets can be 'compiled' into binary data, then stored in parallel flash and used to configure the Controller on power-up.

These APIs are intended to be used with the Run Batchfile API. See Section 4.18.

4.3.1 void DLP_FlashCompile_SetCommPacketCallback (void(*)(UINT8 *packetBuffer, UINT16 nBufBytes) pf)

Initializes flash compile callback function.

Initialization function to setup callback function that will be used to copy the comm packet buffer into a buffer used for storing the flash output.

Example: void CopyCommPacketBufferToFlashBuffer(UINT8* packetBuffer, UINT16 nBufBytes { ... }

4.3.2 void DLP_FlashCompile_FlushCommPacketBuffer ()

Forces a flush of the communication data back to the calling client.

Flushes the contents of a partially filled comm packet buffer, which was populated by one or more calls to DLP_FlashCompile_AppendWriteRegisterFieldCmdWithMask.

4.3.3 void DLP_FlashCompile_SetCompileMode (UINT8 enableBit)

Enables/Disables "FlashCompile" mode.

If "Flash Compile" mode is enabled, USB/SPI communication becomes disabled, and any API calls will trigger calls to the callback function defined by 'DLP_FlashCompile_SetCommPacketCallback', which allows for the creation of binary communication packets of data. This binary data can then be used as an input for building flash configuration files.

Parameters

enableBit set to non-zero to enable, set to 0 to disable

4.3.4 UINT8 DLP_FlashCompile_GetCompileMode ()

Get "FlashCompile" mode flag.

Returns

=1 if enabled; =0 otherwise

4.4 Flash Program APIs

Functions

- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash*, UINT8 *pBuf, UINT32 *nBytesInBuf*, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 **out*CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash*, UINT32 *nBytesTo- Erase*)
- UINT8 DLP_FlashProgram_ReadParallelFlashToFile (const char *filename, UINT32 flash_byte_- offset, UINT32 tot_bytes)

The Flash Program APIs are used for programming the serial or parallel flash devices in our system. The serial flash contains the DLP Digital Controller firmware, while the parallel flash contains the configuration settings (e.g. LUTs, registers, SL image data, etc.) for the DLP Digital Controller that are applied upon power up or when requested.

4.4.1 UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash*, UINT8 **pBuf,* UINT32

nBytesInBuf, **UINT8 (*)(double completionPerCent)pf, UINT8** *verifyFlag*, **UINT16 *** *outCRC)*

Programs serial flash with new DLP Controller firmware.

Performs flash download operation by copying the contents of the supplied buffer to the SERIAL flash part. The starting write address can be defined by specifying the number bytes to skip.

Parameters

*nSkipBytesInFlas*h set equal to zero - future releases might require a non-zero value to update a portion of the DLP Controller firmware

pBuf ptr to byte buffer

nBytesInBuf number bytes in buffer = number bytes to write into flash pf function ptr for sending progress updates back to caller. If this callback returns nonzero, the download is cancelled.

verifyFlag Set this value to 1 for a CRC16 calculation to be done, set to 0 for no calculation to be performed.

outCRC calculated CRC16 for the supplied data

Returns 0 for success

4.4.2 UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 *nSkipBytesInFlash*, UINT8 **pBuf*, UINT32 *nBytesInBuf*, UINT8(*)(double completionPerCent)pf, UINT8 verifyFlag, UINT16 **outCRC*)

Programs parallel flash with configuration settings for DLP Controller.

Performs flash download operation by copying the contents of the supplied buffer to the PARALLEL flash part. The starting write address can be defined by specifying the number bytes to skip.

Parameters

nSkipBytesInFlash starting write address

pBuf ptr to byte buffer

nBytesInBuf number bytes in buffer = number bytes to write into flash

pf function ptr for sending progress updates back to caller. If this callback returns nonzero, the download is cancelled.

verifyFlag Set this value to 1 for a CRC16 calculation to be done, set to 0 for no calculation to be performed.

outCRC calculated CRC16 for the supplied data

Returns

0 for success

4.4.3 UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash*, UINT32 *nBytesToErase*)

Debug routine to erase the parallel flash.

Parameters

nSkipBytesInFlash number of bytes to skip before doing erase; ideally, needs to align with sector boundary

nBytesToErase number of bytes to erase

4.4.4 UINT8 DLP_FlashProgram_ReadParallelFlashToFile (const char **filename*, UINT32 *flash_byte_offset*, UINT32 *tot_bytes*)

Debug routine to read back any portion of the parallel flash.

Parameters

filename name of file to write the flash data to; if file already exists this API will overwrite it

flash_byte_offset offset, in bytes, to start reading from flash; used for flash read addr

tot_bytes total number of bytes to read

Returns

0 for success

4.5 Image Transfer APIs

Functions

- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- Status WriteExternalImage (String name, Byte imageIndex)
- IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)
- Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

The Image Transfer APIs are used to transfer grayscale image data to the DLP Controller's external frame memory, which can then be used for display in SL mode.

The downloaded image patterns are displayed using the display order defined by the Image Order LUT API (DLP_RegIO_WriteImageOrderLut).

4.5.1 UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)

Downloads single bit plane image using USB/SPI to external memory (DDR).

For this API to be used, the input image data must first be formatted properly, which involves 2 steps:

1. Split out each bit plane individually
2. Pack 8 pixels into each byte (e.g. pixel#0 maps to LSB, pixel#1 maps to LSB+1, etc.)

This API can be used along with a TBD LOGIC supplied image formatting function.

Parameters

pBuf ptr to packed bit plane buffer

tot_bytes total number bytes for bit plane data

bpNum0b 0-based bit plane number; used to compute DDR write addr

Returns

0 for success

4.5.2 UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)

Downloads single bit plane image from Parallel Flash to external memory. Uses the image data stored in flash to be copied to the static image buffer memory. For flash offsets, please check the flash layout offsets that were used for building the binary flash file. When grayscale images are being transferred, this API will need to be called once for each bit of bit depth (e.g. called 8 times when BPP = 8).

Parameters

flash_byte_offset offset, in bytes, to start of bit plane data; used for flash read addr

tot_bytes total number bytes for bit plane data

bpNum0b 0-based bit plane number; used for ext mem write addr

Returns

0 for success

4.5.3 Status WriteExternalImage (String name, Byte imageIndex)

Downloads a DBI image file to image buffer memory.

Downloads a DBI image file to the DLP Controller Chip's external image buffer memory. The DBI file is created during the "image import" process using LOGIC's application SW, and consists of a file header + formatted image data. Please see the SW users manual for more information on the DBI file specification.

Parameters

name DBI image filename

imageIndex write address (=index) for storing in image buffer memory.

Returns

STAT_OK for success

4.5.4 IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)

Performs memory allocation, then reads DBI image into allocated buffer.

Using the supplied DBI image filename, performs memory allocation to hold bit plane data. This API may be useful if storing DBI image data in parallel flash.

Parameters

name DBI image filename; DBI files contain a header followed by formatted image data

bpp bits per pixel for image

bitPlaneSize output num bytes for each bitplane image; total size of image data is bpp*bitPlaneSize

Returns

ptr (2-D) to allocated image buffer; to free, call destroyBitPlanes

4.5.5 Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

Performs memory de-allocation of bitplane memory.

This API is intended to be used with GetAllBitPlanes

Parameters

bitPlanes ptr (2D) to allocated image buffer bpp bits per pixel for image

Returns

STAT_OK for success

4.6 LED APIs

Functions

- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)
- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)

The LED APIs are used for controlling the LED illuminators contained inside the light engine.

4.6.1 UINT8 DLP_LED_SetLEDintensity (LED_tLED, double intensity_perCent)

Sets the LED intensity.

Sends a request to the LED driver to supply current to the LED using the specified value. For 100% intensity, the LED will be driven using the max allowed LED current.

For example, if the max LED current is 10A, then calling this API with a value of 50 will set the LED current to 5A (=10*50/100).

Parameters

LED target LED illuminator (see enum)

intensity_perCent desired LED intensity level, given as a percent: 0.0 < range ≤ 100.0

Returns

0 for success

4.6.2 UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)

Gets the LED intensity.

Parameters

LED target LED illuminator (see enum)
intensityPerCent intensity, given as a percent: 0.0 < range ≤ 100.0

Returns

0 for success

4.6.3 void DLP_LED_LEDdriverEnable (UINT8 enable)

Debug API for LED driver enable.

Send 0 to disable LED driver; send > 0 to enable.

In the past this API has been used to re-enable the LED driver for cases where it has "timed out" and become disabled.

4.6.4 void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)

Debug API to check if the LED driver has timed out.

Returns 1 when LED driver is timed out.

NOTE: With the latest OSRAM LED driver, the driver should never timeout.

Parameters

timedOut output variable to hold timeout state

4.6.5 UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)

Debug API to enable or disable an individual LED.

If the PWM Seq has been built with the specified LED, then this API can be used to disable (then re-enable) the LED.

Parameters

LED target LED illuminator (see enum) enableBit set to non-zero to enable, set to 0 to disable LED

Returns

0 for success

4.7 MISC APIs

Functions

- const char * Section 4.7.1 ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)
- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)
- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])
- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- Status ChangeLogLevel (Byte logLevel)
- Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)
- UINT8 DLP_Misc_GetTotalNumberOfUSBDevicesConnected(UINT8 *oNumDev)
- UINT8 DLP_Misc_SetUSBDeviceNumber (UINT8 devNum0b)

- UINT8 DLP_Misc_GetUSBDeviceNumber (UINT8 *oCurDevNum0b)

The Misc APIs are used for miscellaneous functions that don't naturally fit into any of the other categories.

4.7.1 const char * DLP_Misc_GetVersionString ()

4.7.2 void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))

Initialization function to setup API logging.

Parameters

loglvl logging visibility level; range = { 0, 1, 2}; Sending values > 0 will enable extra debug logging. The higher the value, the more debug logging will be available.

pf function ptr for function taking const char* arg with void return type.

Example 1: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(0, LogIt);

In this example, no API debug logging will be sent to the LogIt function since a logging visibility level of 0 was supplied.

Example 2: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(1, LogIt);

In this example, 'level 1' API debug logging will be sent to the LogIt function since a logging visibility level of 1 was supplied.

Example 3: void LogIt(const char* it) { ... }

DLP_Misc_SetLoggingCallback(2, LogIt);

In this example, 'level 1 and 2' API debug logging will be sent to the LogIt function since a logging visibility level of 2 was supplied.

4.7.3 void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)

Sets the logging visibility level (default = 0 for no debug logging).

Parameters

loglvl logging visibility level; range = { 0, 1, 2} Sending values > 0 will enable extra debug logging. The higher the value, the more debug logging will be available.

4.7.4 UINT8 DLP_Misc_InitAPI ()

Initializes API.

This function should be called before making API calls.

Returns

0 for success

4.7.5 UINT8 DLP_Misc_DisableCommunication (const char *ofileName)

Debug API to disable real-time communication and store communication payload in the supplied output filename.

This function would typically only be used for debug or regression testing.

4.7.6 void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()

Debug API to force a flush of communication packet data to disk.

This API is only applicable when USB or SPI communication is disabled.

4.7.7 UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])

Debug API to execute passthrough command.

A configuration batchfile can have "pass through" commands that allow a subset of these APIs to be called.

Parameters

args[ ] array of argument strings; max num elements = 16; supply a NULL ptr after last real element

Returns

0 for success

4.7.8 UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)

Program the EDID with data provided in a binary file. Note: Based on the EDID 1.3 data structure.

Provides for both partial and full updates.

Parameters

DMD DMD type (see enum)
offset offset of where in the PROM to begin updating with new data (valid range 0-127)
numBytes number of bytes in the supplied file to write to the PROM starting at above offset
fileName binary file containing the data to write to the PROM

Returns

0 for success

4.7.9 Status ChangeLogLevel (Byte logLevel)

Sets the PortAbilityLayer logging level.

Sets the PortAbilityLayer (SW Layer = 2.5) logging level.

To set the logging level for the low-level (SW Layer = 2.0) functions, see Section 4.7.3.

Parameters

logLevel Set to values > 1 to increase the amount of debug logging.

Returns

STAT_OK for success

4.7.10 Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)

Initialization API that must be called prior to calling any API.

Example:

```
void Logit(String it) { printf("%s\n", it);}

...

Status rc = InitPortabilityLayer(loglvl+1, loglvl, Logit);
```

Parameters

logLevel SW Layer2.5 logging level; Set to values > 1 to increase the amount of debug logging.
detail SW Layer2.0 logging level; Set to values > 0 to increase the amount of debug logging.
callback logging callback function

Returns

STAT_OK for success

4.7.11 UINT8 DLP_Misc_GetTotalNumberOfUSBDevicesConnected (UINT8 *oNumDev)

Gets the total number of USB devices currently connected to the PC USB driver.

Parameters

oNumbDev output value for the number of USB devices connected

Returns

0 for success

4.7.12 UINT8 DLP_Misc_SetUSBDeviceNumber (UINT8 devNum0b)

Sets which USB device subsequent API calls will be sent to.

The range of the input argument is {0, 1, ..., –1} where N is the total number of USB devices connected to the PC USB driver.

For example, if 4 devices are connected to the USB driver, calls to this API would have to be made to switch the communication between devices using an argument of {0, 1, 2, 3}.

The API uses device 0 as the default device number.

The mapping of device numbers to USB ports on the PC is machine dependent and will require some experimentation on the part of the user to determine this mapping for their own PC.

Parameters

devNum0b 0-based USB device number; device number defaults to 0

Returns

0 for success

4.7.13 UINT8 DLP_Misc_GetUSBDeviceNumber (UINT8 *oCurDevNum0b)

Gets the current USB device number (0-based) that API calls are being sent to.

Parameters

oCurDevNum0b output value for current USB device number, 0-based value

Returns

0 for success

4.8 Register and LUT Read/Write APIs

Modules

LUT APIs

Functions

- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)
- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)

The Register/LUT APIs are used for configuring the DLP Digital control chip.

For the most part, the low-level Write-Register APIs are not intended to be called directory. More typically, they may be called while in the process of running a configuration batchfile. Configuration batchfiles are created by the LOGIC SW application.

As of API release V1.2.0, no plans have been made to publish a spec on the DLP Digital Control chip's low-level register map.

4.8.1 UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)

Initialize from parallel flash offset.

Loads a new solution from parallel flash. Address offset must match the offset of a solution listed in the parallel flash header

Parameters

offset Address offset to starting location in parallel flash memory reset 1 to perform system reset before initialization, 0 to skip reset

Returns

0 for success

4.8.2 UINT8 DLP_RegIO_ReadLUT (const char * LUTname)

Call this function to read the contents of LUT and print its contents to the log.

Parameters LUTname see LUT Names

Returns

0 for success

4.8.3 UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)

Debug API to read a DLP Controller register.

Read a value from a specific location within a 32-bit register

Parameters

addr HW register addr

full_reg_LSB_bitnum0b 0-based LSB bit location within 32-bit register; range = 0-31

full_reg_MSB_bitnum0b 0-based MSB bit location within 32-bit register; range = 0-31

ovalue output value from register field (range corresponds to num bits = MSBbitnum-LSBbitnum+1)

Returns

0 for success

Example: register 0x1100 holds a 2-bit register field in bits 9:8 UINT32 oval; UINT8 rc = DLP_RegIO_ReadRegisterField(0x1100, 8,9, oval); (result: oval = 0x2)

Parameters

addr input full_reg_LSB_bitnum0b input full_reg_MSB_bitnum0b input ovalue output

4.8.4 UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)

Set the Image Order LUT.

Allows users to define the display order for image data stored in ext memory which will be used in structured light mode. This allows the display order of the images to be changed without having to re-load the image data to external memory.

Example: Three 8-bit image patterns are stored in ext mem as {image 0, image 1, image2}, but user wants display order = {2, 1, 0}

UINT16 arrStartImgNum0b[3]={ 2,1,0 };

UINT8 rc = DLP_RegIO_WriteImageOrderLut(8, arrStartImgNum0b, 3);

Parameters

bpp bits per pixel (for each corresponding image file)

arrStartImgNum array containing starting 0-based IMAGE number

nArrElements number of array elements Returns 0 for success

Returns

0 for success

4.8.5 UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)

Debug API to write a DLP Controller register.

Write a value to a specific location within a 32-bit register. This API will typically be called during the processing of a configuration batchfile. The DLP Controller Chip's register map is currently undocumented.

Parameters

addr HW register addr

value raw value for register field

full_reg_LSB_bitnum0b 0-based LSB bit location within 32-bit register; range = 0-31

full_reg_MSB_bitnum0b 0-based MSB bit location within 32-bit register; range = 0-31 Example: register 0x1100 holds a 2-bit register field in bits 9:8 UINT8 rc = DLP_RegIO_- WriteRegisterField(0x1100, 0x3, 8,9); To do a full register write, set LSB bit num = 0 and MSB bit num = 31.

Returns

0 for success

4.9 LUT APIs

Functions

- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)

4.9.1 UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)

Configuration batchfile function used to define the start of LUT data.

When processing a configuration batchfile, call this function before initiating any LUT data via register write calls.

Parameters

LUTname see LUT Names

Returns

0 for success

4.9.2 UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)

Configuration batchfile function used to define the end of LUT data. Call this function after performing the last LUT data call (via register write).

Parameters

LUTname see LUT Names

Returns

0 for success

4.10 Data Source APIs

Functions

- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Source_GenerateSoftwareVsync () ()
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)

The Data Source APIs are used to configure the input data source (e.g. pixel data).

The Trigger APIs are used to define the characteristics of the incoming VSYNC trigger (from the perspective of the DLP Digital Controller).

4.10.1 UINT8 DLP_Source_SetDataSource (DATA_t source)

Sets the input data source.

Parameters

source the specified data source

Returns

0 for success

4.10.2 UINT8 DLP_Source_GenerateSoftwareVsync()

Generates one VSYNC inside DLP control chip. This API performs the same functionality as the trigger pulse in External hardware trigger mode. Sending this command once invokes the same functionality as a single trigger pulse input in the external hardware trigger input mode.

Parameters

Returns

0 for success

Note: This command packet is not supported by DLPC200 firmware versions earlier than 2.1.6.

4.10.3 UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)

Programs the trigger edge.

Parameters

edge set to non-zero for rising edge, set to 0 for falling edge

Returns

0 for success

4.11 Video Test Pattern Generator (TPG) APIs

Functions

- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)

The TPG APIs are used for defining internal test patterns for video mode. The spatial frequency of each pattern is controllable via an API input parameter.

4.11.1 UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_ttestPattern, TPG_Col_t color, UINT16 patternFreq)

When input source is Test Pattern Generator, programs which test pattern to display.

Parameters

DMD DMD Type (enum)

testPattern the desired test pattern to display (see enum)

color test pattern color selection (see enum)

patternFreq number of times to spatially repeat the pattern. Valid values for XGA are 1,2,4,8,16,32,64,128,256,512 Horizontal patterns will repeat according to this parameter. Due to the non power of 2 vertical DMD resolution, vertical patterns are only loosely based on this frequency.

Returns

0 for success

4.12 Sync APIs

Functions

- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)

- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)

The Sync APIs are used for conditioning the output sync signals. Output syncs can be used to synchronize an external object like a camera (e.g. the camera can be "slaved" to the LightCommander Projector).

4.12.1 UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)

Configures the specified output sync.

Parameters

syncNumber the output sync number to configure (1-3)

polarity set to non-zero for positive, set to 0 for negative polarity

delay sets the output sync delay in microseconds

width sets the output sync pulse width in microseconds

4.12.2 UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)

Enables/disables the specified output sync.

Parameters

syncNumber the output sync number to enable/disable (1-3)

enableBit set to non-zero to enable, set to 0 to disable

Returns

0 for success

4.13 PWM Port Configuration APIs

Functions

- UINT8 DLP_PWM_SetPeriod (UINT16 period)
- UINT8 DLP_PWM_GetPeriod (UINT16* period_out)
- UINT8 DLP_PWM_SetDutyCycle (UINT8 port, UINT16 dutyCycle)
- UINT8 DLP_PWM_GetDutyCycle (UINT8 port, UINT16* dutyCycle_out)

There are four programmable PWM ports in the DLP Controller, refer to DLPC200 datasheet for PWM0-PWM4 I/O pin details. The PWM port(s) enable control of low cost LED driver systems that accept PWM type input signals for LED current strength adjustment. Apart from LED driver control the PWM port(s) can also be used as general purpose PWM ports.

NOTE: These APIs not supported by DLPC200 MCU firmware version earlier than 2.1.6.

4.13.1 UINT8 DLP_PWM_SetPeriod(UINT16 period)

Configures PWM ports period value. The DLP Controller chip sets the same base period for all the four ports.

Parameters

Period Period parameter valid range (0 to 2047) which generates a period of 0 (0 * 40ns) to 81.88µs (2047 * 40ns) respectively.

Returns

0 for success

4.13.2 UINT8 DLP_PWM_GetPeriod(UINT16* period_out)

Returns the PWM period configuration value.

Parameters

period_out Contains the PWM period value in range (0 to 2047). The value to be interpreted as <value> * 40ns.

Returns

0 for success

4.13.3 DLP_PWM_SetDutyCycle(UINT8 port, UINT16 dutyCycle)

This API sets the PWM port(s) duty cycle. It requires the PWM port number and a two byte value. Upon setting the PWM period the value sent in this packet defines the output duty cycle for the PWM port.

Parameters

port PWM port number (0 to 3). port = 4 applies same PWM duty cycle to all four ports.

dutycycle ON time setting based on the PWM period setting (0 to 2047).

Returns

0 for success

Note 1: Setting ANY value >= PWM period will generate a 100% Duty Cycle.

Note 2: Setting 0 as the PWM Duty Cycle will generate a 0% Duty Cycle.

4.13.4 UINT8 DLP_PWM_GetDutyCycle(UINT8 port, UINT16* dutyCycle_out)

Returns the Duty Cycle set for the selected PWM Port in the API.

Parameters

port PWM port number (0 to 3)

**dutycycle_out* ON time setting based on the PWM Period Setting (0 to 2047).

Returns

0 for success

Example

Setting the PWM Period to 256 will cause a 10.24µs (256 * 40ns) period signal.
Setting a value of 64 in the PWM duty cycle API will generate 25% duty cycle.
Setting a value of 128 in the PWM duty cycle API will generate 50% duty cycle.
Setting a value of 192 in the PWM duty cycle API will generate 75% duty cycle.
Setting a value 256 in the PWM duty cycle API will generate 100% duty cycle.

4.14 Status APIs

Functions

- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)

- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)
- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 DLP_Status_GetSeqDataExposure (double * exp_out)
- UINT8 DLP_Status_GetSeqDataFrameRate (double * fr_out)
- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPatterns (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

The Status APIs are used for retrieving system status information.

4.14.1 UINT8 DLP_Status_CommunicationStatus ()

Detects USB/SPI communication status.

Returns

0 if communication ok, 1 if communication can not be made over USB/SPI Note: as of v1.2, only USB communication is supported.

4.14.2 UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)

Get External Memory BIST done state.

API used to determine if the Built-In-Self-Test (BIST) operation has completed. A BIST operation is performed on the external frame memory every power up.

Parameters

state_out state flag (output): 1 if BIST done, otherwise 0

Returns

0 for success

4.14.3 UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)

Get BIST fail state.

Every power up, a Built-In-Self-Test (BIST) operation is run on the external memory, and its pass/fail state is saved off. This API returns the pass/fail state of this test.

Parameters

state_out state flag (output): 1 if BIST failed, otherwise 0

Returns

0 for success

4.14.4 UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)

Get DLPA200/DAD SPI communication status.

Parameters

status_out status flag (output): 0 if ok, 1 if DAD SPI failure

Returns

0 for success

4.14.5 UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)

Check for DLPA200 (aka DAD = Analog reset driver) fault.

Parameters

fault_out fault flag (output): will be set to 1 if DAD fault detected, otherwise 0

Returns

0 for success

4.14.6 UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)

Get DLP Control Chip version number (major.minor.patch format).

Parameters

ver_out version string (output): will be pointed to a static buffer containing version string

Returns

0 for success

4.14.7 UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)

Get DMD SPI communication status.

Parameters status_out status flag (output): 0 if ok, 1 if DMD SPI failure

Returns

0 for success

4.14.8 UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)

Determine if DMD is parked by hardware switch.

Parameters state_out state flag (output): will be set to 1 if HW switch is activated (parked), otherwise 0

Returns

0 for success

4.14.9 UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)

Gets DMD Park State.

Parameters

state_out state flag (output): 0 if unparked, 1 if parked

Returns

0 for success

4.14.10 UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)

Determine if DMD park was requested by software.

Parameters

state_out state flag (output): will be set to 1 if park requested by SW, otherwise 0

Returns

0 for success

4.14.11 UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)

Check for EEPROM fault.

Parameters

fault_out fault flag (output): will be set to 1 if EEPROM fault detected, otherwise 0

Returns

0 for success

4.14.12 UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)

Get flash programming mode.

Parameters

mode_out mode (output): 0 if normal mode, 1 if flash programming mode

Returns

0 for success

4.14.13 UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)

PWM Sequence Compiler DLL version number.

Gets version number for the sequence compiler DLL used to build PWM seq. Format is major.minor.patch.

Parameters

ver_out version string (output): will be pointed to a static buffer containing version string

Returns

0 for success

4.14.14 UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)

Get failure state for initialization from parallel flash.

There are 3 states for parallel flash initialization. If flash initialization was completed successfully, the API will return 0. If flash initialization was attempted and failed, the API will return 1. If no flash initialization has been attempted, the API will return 2.

Parameters

state_out state flag (output): 0 if flash init successful, 1 if flash init failed, 2 if no flash init attempted

Returns

0 for success

4.14.15 UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)

Get LED Driver SPI communication status.

Parameters

status_out status flag (output): 0 if ok, 1 if LED Driver SPI failure

Returns

0 for success

4.14.16 UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)

Check for LED Driver fault.

Parameters

fault_out fault flag (output): will be set to 1 if LED driver fault detected, otherwise 0

Returns

0 for success

4.14.17 UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)

Get LED driver lit state by LED type.

Parameters

LED LED illuminator (see enum) state_out state flag (output): 0 if driver off, 1 if driver on

Returns

0 for success

4.14.18 UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)

Get LED driver temp timeout by LED type.

Parameters

LED LED illuminator (see enum) state_out timeout flag (output): 1 if driver timed out due to temperature, otherwise 0

Returns

0 for success

4.14.19 UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)

Internal MicroController SW version number.

Get MCU software version (major.minor.patch format)

Parameters

ver_out version string (output): will be pointed to a static buffer containing version string

Returns

0 for success

4.14.20 UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)

Get overall LED driver temperature timeout state.

Parameters s

tate_out state flag (output): 1 if LED driver timeout, otherwise

Returns

0 for success

4.14.21 UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)

Get overall LED/Lamp lit state.

Checks all channels on the LED driver to determine if all LEDs are LIT (e.g. ready for operation).

Parameters

state_out state flag (output): 0 if not lit, 1 if lit

Returns

0 for success

4.14.22 UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)

Get sequence data - number of bits per pixel.

Meta data describing the BPP used for building currently loaded PWM seq. BPP is only applicable in structured light mode.

Parameters

bpp_out number of bits per pixel (output)

Returns

0 for success

4.14.23 UINT8 DLP_Status_GetSeqDataExposure (double * exp_out)

Get sequence data - structured light exposure percentage.

Meta data describing the exposure time (in µsec) used for the building the currently load PWM seq. BPP is only applicable in structured light mode.

Parameters

exp_out exposure (output)

Returns

0 for success

4.14.24 UINT8 DLP_Status_GetSeqDataFrameRate (double * fr_out)

Get sequence data - framerate.

Meta data describing the frame rate (in Hz) used for building the currently loaded PWM seq.

Parameters

fr_out sequence framerate (output)

Returns

0 for success

4.14.25 UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t* mode_out)

Get sequence data mode.

Meta data describing the currently loaded PWM seq

Parameters

mode_out current sequence data mode

Returns

0 for success

4.14.26 UINT8 DLP_Status_GetSeqDataNumPatterns (UINT16 * numPatterns_out)

Get sequence data - number of patterns/frame.

Meta data describing the num patterns/frame used for building currently loaded PWM seq

Parameters

numPatterns_out number of patterns (output)

Returns

0 for success

4.14.27 UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)

Get PWM sequence running state. Parameters state_out state flag (output): 0 if stopped, 1 if running normally

Returns

0 for success

4.14.28 UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

Check for UART fault.

Parameters

fault_out fault flag (output): will be set to 1 if UART port fault detected, otherwise 0

Returns

0 for success

4.15 Data Types, Variables, and Constants

Modules

- LUT Names

Defines

- #define DMD_t_CUR_VERNUM 1
- #define DATA_t_CUR_VERNUM 1
- #define TPG_t_CUR_VERNUM 1
- #define TPG_Col_t_CUR_VERNUM 1
- #define LED_t_CUR_VERNUM 1
- #define SEQDATA_t_CUR_VERNUM 1

Typedefs

- typedef char Char
- typedef double Double
- typedef signed char Int8
- typedef signed short Int16
- typedef signed int Int32
- typedef unsigned char UInt8
- typedef unsigned short UInt16
- typedef unsigned int UInt32
- typedef const char * String
- typedef UInt8 Byte
- typedef void Void
- typedef int Boolean
- typedef Byte ** IntPtr
- typedef char * StringBuilder

Enumerations

- enum DMD_t { DMD_XGA = 0 }
- enum DATA_t {
  DVI = 0, EXP,
  TPG,
  SL_AUTO,
  SL_EXT3P3,
  SL_EXT1P8,
  SL_SW }
- enum TPG_t {
  SOLID = 0,
  HORIZ_RAMP,
  VERT_RAMP,
  HORIZ_LINES,
  DIAG_LINES,
  VERT_LINES,
  HORIZ_STRIPES,
  VERT_STRIPES,
  GRID,
  CHECKERBOARD }
- enum TPG_Col_t {
  TPG_BLACK = 0,

TPG_RED,
TPG_GREEN,
TPG_BLUE,
TPG_YELLOW,
TPG_CYAN,
TPG_MAGENTA,
TPG_WHITE }

- enum LED_t {
  LED_R = 0,
  LED_G = 1,
  LED_B = 2,
  LED_IR = 3 }
- enum SEQDATA_t {
  SDM_SL = 0,
  SDM_SL_RT = 1,
  SDM_VIDEO = 2,
  SDM_MIXED = 3,
  SDM_OBJ = 4 }
- enum Status {
  STAT_OK,
  STAT_ERROR }

Variables

- static const char arTPGnames [NUM_PATTERNS][32]
- static const char arTPGcolorNames [8][16]
- static char LED_Color [NUM_LEDS][3]

4.15.1 Define Documentation

4.15.1.1 #define LED_t_CUR_VERNUM 1

Possible LED illuminators.

NOTE: It is possible that an LED could be replaced with a different type of LED, however, this enum is really just defined so that we can create the control signals (aka real-time LED enable).

4.15.2 Enumeration Type Documentation

4.15.2.1 enum DATA_t

Data sources.

Enumerator:

DVI DVI port. EXP Expansion port.
TPG Test pattern generator.
SL_AUTO Static frame memory, Auto generated VSYNC.
SL_EXT3P3 Static frame memory, 3.3V external VSYNC.
SL_EXT1P8 Static frame memory, 1.8V external VSYNC.
SL_SW Static frame memory, Software VSYNC.

4.15.3 enum DMD_t

Supported DMDs.

Enumerator

DMD_XGA XGA DMD (1024x768).

4.15.4 enum LED_t

Types of LEDs.

Enumerator

LED_R Red LED
LED_G Green LED
LED_B Blue LED
LED_IR Infrared LED

4.15.5 enum SEQDATA_t

Major operating modes.

Enumerator

SDM_SL Structured light, non real-time input.
SDM_SL_RT Structured light, real-time input.
SDM_VIDEO Video mode.
SDM_MIXED Video plus Structured Light.
SDM_OBJ Object mode.

4.15.6 enum Status

Status.

Enumerator

STAT_OK OK.
STAT_ERROR ERROR.

4.15.7 enum TPG_Col_t

TPG Colors.

See Video Test Pattern Generator APIs

Enumerator

TPG_BLACK TPG Black.
TPG_RED TPG Red.
TPG_GREEN TPG Green.
TPG_BLUE TPG Blue.
TPG_YELLOW TPG Yellow.
TPG_CYAN TPG Cyan.
TPG_MAGENTA TPG Magenta.
TPG_WHITE TPG White.

4.15.8 enum TPG_t

TPG Patterns.

See Video Test Pattern Generator APIs

Enumerator

SOLID TPG Solid Field.
HORIZ_RAMP TPG Horizontal Ramp.
VERT_RAMP TPG Vertical Ramp.
HORIZ_LINES TPG Horizontal Lines.
DIAG_LINES TPG Diagonal Lines.
VERT_LINES TPG Vertical Lines.

HORIZ_STRIPES TPG Horizontal Stripes.
VERT_STRIPES TPG Vertical Stripes.
GRID TPG Grid.
CHECKERBOARD TPG Checkerboard.

4.16 LUT Names

Defines

#define REGIO_SEQ_LUT "SEQ_LUT"
#define REGIO_CMT_LUT "CMT_LUT"
#define REGIO_RWC_LUT "RWC_LUT"
#define REGIO_UMCTDM_LUT "UMCTDM_LUT"

4.16.1 #define REGIO_SEQ_LUT "SEQ_LUT"

PMW Seq LUT Name.

Use for API calls to LUT APIs

4.16.2 #define REGIO_CMT_LUT "CMT_LUT"

CMT (=Degamma) LUT Name.

Use for API calls to LUT APIs

4.16.3 #define REGIO_RWC_LUT "RWC_LUT"

RWC LUT Name (rarely/never used).

Use for API calls to LUT APIs

4.16.4 #define REGIO_UMCTDM_LUT "UMCTDM_LUT"

UMCTDM LUT Name (rarely/never used).

Use for API calls to LUT APIs

4.17 API Version Number and Release Notes

Defines

#define DLP_API_VERSION_NUM "1.8.0"

- **1.8.0** 9/30/10 - v1.1 release
- **1.6.1** 05/21/10 - initial release for production
- **1.6.0** 05/11/10 - beta candidate

4.18 Batchfile APIs

Functions

- Status RunBatchCommand (String command)
- Status RunBatchFile (String name, Boolean stopOnError)

The Batchfile APIs are used to process the commands found within a configuration batchfile, which are currently generated by LOGIC's application SW that is included with the DLP LightCommander Kit.

4.18.1 Status RunBatchCommand (String command)

Processes a configuration batchfile command.

This API acts like a translator between batchfile commands and API functions.

See Batchfile Language Specification V1.0 for the currently supported listing of commands.

Parameters

command batchfile command

Returns

STAT_OK for success;

4.18.2 Status RunBatchFile (String name, Boolean stopOnError)

Processes a configuration batchfile.

Configuration batchfile are created by the LOGIC SW application and are used for configuring the DLP Controller Chip with Look-Up-Tables(LUTs), register settings, and image data. A configuration batchfile can contain multiple BF commands, which occur on separate lines. See Batchfile Language Specification V1.0 for the currently supported listing of commands.

Parameters

name filename

stopOnError boolean flag to control behavior after encountering errors set =1 to halt batchfile processing after hitting error; If =0, then return code will always be STAT_OK

Returns

STAT_OK for success;

4.19 Batchfile Language Specification V1.0

Modules

- Batchfile Status Commands
- Batchfile Display Commands
- Batchfile LED Commands
- Batchfile Image Commands
- Batchfile Misc Commands
- Batchfile LUT/Register Commands
- Batchfile Source/VSYNC Trigger Commands
- Batchfile Internal Test Pattern Generator Commands
- Batchfile Sync Commands

Background

Configuration batchfiles are plain text files that contain commands that control/configure the DLP Controller Chip. Currently, application SW written by LOGIC and TI can be used to generate configuration batchfiles, which can then be executed through that same SW, or via a separate SW application that implements the RunBatchFile API.

Real-time operations that can be performed by executing batchfiles include:

- Writing LUTs and (limited) registers
- Reading system status
- Controlling the LED driver (e.g. enable/disable LED, increase/decrease LED brightness)
- Downloading DBI image files
- Programming EDID
- Controlling the DLP display (e.g. start, stop, next image, park, unpark, H-flip, V-flip)
- Selecting input data ports, Internal Test Patterns, and VSYNC triggers
- Controlling the output sync signals (= camera input trigger)

The use of batchfiles is driven by the following:

- Some LUTs can not be generated via simple API calls and thus require the use of other SW for calculating; Once calculated, the LUTs can easily be passed to the API functions for configuring the system

- Since batchfiles are text files, they can easily be modified to add additional configuration or even created from scratch
- Executing different batchfiles allows users to easily switch between different configurations in realtime
- Configuration of the system can often be accomplished without having to write a custom SW application

In most cases, there is a one-to-one relationship between a batchfile command and a corresponding API function, and whenever possible, the same name is used in both places. For example, a batchfile containing the command "$L2.5 DLP_Display_ParkDMD" will end up calling DLP_Display_ParkDMD

For advanced users, a configuration batchfile can be 'compiled' into binary format, then stored in parallel flash to allow auto-configuration on power up. The specification for the parallel flash layout is currently not available, so this capability is really intended for future use. See Flash Compile APIs.

Batchfile Syntax

- Batchfiles should have a single command on each line
- Use the '#' character for adding comments, preferably at the start of a line
- Command syntax: $L2.5 command [arg1 arg2 ...] [# optional comment]
- The "$L2.5" directive (which stands for SW layer2.5) and is used for test purposes and future capability
- In most cases, "command" matches the API name exactly.
- Line tokens can be separated by either whitespace or the "," character
- Example: "$L2.5 DLP_RegIO_WriteImageOrderLut 8, 0, 1, 2"
- Any commands that perform a "read" operation will log the results to std out.

Organization of Batchfile Commands

4.20 Batchfile Status Commands

- DLP_Status_GetMCUversionString
  - Description: See API
  - API: DLP_Status_GetMCUversionString
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetDlpControllerVersionString
  - Description: See API
  - API: DLP_Status_GetDlpControllerVersionString
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetFlashProgrammingMode
  - Description: See API
  - API: DLP_Status_GetFlashProgrammingMode
  - Arguments: None
  - Output (see logging): See API for output

- DLP_Status_GetDADcommStatus
  - Description: See API
  - API: DLP_Status_GetDADcommStatus
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetDMDcommStatus
  - Description: See API
  - API: DLP_Status_GetDMDcommStatus
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetLEDcommStatus
  - Description: See API
  - API: DLP_Status_GetLEDcommStatus
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetBISTdone
  - Description: See API
  - API: DLP_Status_GetBISTdone
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetBISTfail
  - Description: See API
  - API: DLP_Status_GetBISTfail
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetInitFromParallelFlashFail
  - Description: See API
  - API: DLP_Status_GetInitFromParallelFlashFail
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetOverallLEDlampLitState
  - Description: See API
  - API: DLP_Status_GetOverallLEDlampLitState
  - Arguments: None –
  - Output (see logging): See API for output
- DLP_Status_GetOverallLEDdriverTempTimeoutState
  - Description: See API
  - API: DLP_Status_GetOverallLEDdriverTempTimeoutState
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetSeqDataFrameRate
  - Description: See API
  - API: DLP_Status_GetSeqDataFrameRate
  - Arguments: None
  - Output (see logging): See API for output

- DLP_Status_GetSeqDataExposure
  - Description: See API
  - API: DLP_Status_GetSeqDataExposure
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetSeqDataNumPatterns
  - Description: See API
  - API: DLP_Status_GetSeqDataNumPattern
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetSeqDataBPP
  - Description: See API
  - API: DLP_Status_GetSeqDataBPP
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_CommunicationStatus
  - Description: See API
  - API: DLP_Status_CommunicationStatus
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetLEDdriverTempTimeoutState
  - Description: See API
  - API: DLP_Status_GetLEDdriverTempTimeoutState (LED_t
  - Arguments: 1. LEDnum see LED_t
  - Output (see logging): See API for out put
- DLP_Status_GetSeqDataMode
  - Description: See API
  - API: DLP_Status_GetSeqDataMode
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetLEDdriverLitState
  - Description: See API
  - API: DLP_Status_GetLEDdriverLitState
  - Arguments: 1. LEDnum: see LED_t
  - Output (see logging): See API for output
- DLP_Status_GetFlashSeqCompilerVersionString
  - Description: See API
  - API: DLP_Status_GetFlashSeqCompilerVersionString
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetDMDparkState
  - Description: See API
  - API: DLP_Status_GetDMDparkState
  - Arguments: None
  - Output (see logging): See API for output

- DLP_Status_GetDMDhardwareParkState
  - Description: See API
  - API: DLP_Status_GetDMDhardwareParkState
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetDMDsoftwareParkState
  - Description: See API
  - API: DLP_Status_GetDMDsoftwareParkState
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetSeqRunState
  - Description: See API
  - API: DLP_Status_GetSeqRunState
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetEEPROMfault
  - Description: See API
  - API: DLP_Status_GetEEPROMfault
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetDADfault
  - Description: See API
  - API: DLP_Status_GetDADfault
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetLEDdriverFault
  - Description: See API
  - API: DLP_Status_GetLEDdriverFault
  - Arguments: None
  - Output (see logging): See API for output
- DLP_Status_GetUARTfault –
  - Description: See API
  - API: DLP_Status_GetUARTfault
  - Arguments: None
  - Output (see logging): See API for output

4.21 Batchfile Display Commands

- DLP_Display_DisplayStop
  - Description: See API
  - API: DLP_Display_DisplayStop
  - Arguments: None
- DLP_Display_ParkDMD
  - Description: See API
  - API: DLP_Display_ParkDMD
  - Arguments: None

- DLP_Display_UnparkDMD
  - Description: See API
  - API: DLP_Display_UnparkDMD
  - Arguments: None
- DLP_Display_SetDegammaEnable
  - Description: See API
  - API: DLP_Display_SetDegammaEnable
  - Arguments: 1. enableBit: range = { 0,1 }
- DLP_Display_HorizontalFlip
  - Description: See API
  - API: DLP_Display_HorizontalFlip
  - Arguments: 1. enableBit: range = { 0,1 }
- DLP_Display_VerticalFlip
  - Description: See API
  - API: DLP_Display_VerticalFlip
  - Arguments: 1. enableBit: range = { 0,1 }
  - Example: $L2.5 DLP_Display_VerticalFlip 1
- DLP_Display_DisplayPatternManualStep
  - Description: See API
  - API: DLP_Display_DisplayPatternManualStep
  - Arguments: None
- DLP_Display_DisplayPatternManualForceFirstPattern
  - Description: See API
  - API: DLP_Display_DisplayPatternManualForceFirstPattern
  - Arguments: None
- DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses
  - Description: See API
  - API: DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses
  - Arguments: None

4.22 Batchfile LED Commands

- DLP_LED_LEDdriverEnable
  - Description: See API
  - API: DLP_LED_LEDdriverEnable
  - Arguments: 1. enableBit: range = { 0,1 }
- DLP_LED_SetLEDintensity
  - Description: See API
  - API: DLP_LED_SetLEDintensity
  - Arguments: 1. LEDnum: see LED_t 2. intensityPerCent: range = { 0 - 100.0 }
- DLP_LED_GetLEDintensity
  - Description: See API
  - API: DLP_LED_GetLEDintensity
  - Arguments: 1. LEDnum: see LED_t – Output (see logging): intensityPerCent: range = { 0 - 100.0 }
- DLP_LED_SetLEDEnable
  - Description: See API

- API: DLP_LED_SetLEDEnable
- Arguments: 1. LEDnum: see LED_t 2. enableBit: range = { 0,1 }

4.23 Batchfile Image Commands

- WriteExternalImage
  - Description: Downloads a DBI image to the static image buffer using the specified write location.
  - API: None
  - Arguments: 1. filename: filename of DBI file 2. imgIndex: 0-based image index for writing into image buffer
  - Example: $L2.5 WriteExternalImage bob.dbi 0 Writes the specified DBI image to image slot number 0 in the static image buffer memory. If this is an 8 BPP image, then the first 8 locations (e.g. bitplane indices 0-7) of the image buffer will contain data for this image.
- DLP_Img_DownloadBitplanePatternFromFlashToExtMem
  - Description: See API
  - API: DLP_Img_DownloadBitplanePatternFromFlashToExtMem
  - Arguments: 1. flashOffsetBytes: location in flash of single image bit plane 2. totalBytes: size of single image bit plane; size for XGA = 1024*768/8 = 98304 3. bitPlaneIndex: 0-based index to use for storing in static image buffer memory – Ex

4.24 Batchfile Misc Commands

- DLP_Misc_GetVersionString
  - Description: Returns the version number for the API in 'major.minor.patch' format
  - API: DLP_Misc_GetVersionString
  - Arguments: None
- RunBatchFile
  - Description: Use this command to call one batchfile from within another batchfile.
  - API: None
  - Arguments: 1. filename: filename of batchfile to execute 2. stopOnError: set =1 to halt execution whenever errors are encountered; set = 0 otherwise
  - Example: $L2.5 RunBatchFile img_order_lut.bf 0 Executes the specified batchfile, which allows for a batchfile to be called within a batchfile.
- RunBatchCommand
  - Description: Executes a batchfile command. This command probably only makes sense if an application is executing BF commands stored in memory (instead of having a physical file on disk).
  - API: None
  - Arguments: 1. bfCmd: batchfile command
  - Example: $L2.5 RunBatchCommand WriteExternalImage bob.dbi 0
- ExecutePassthroughDLPAPI
  - Description: An alternate method of calling an API function; mainly used during early SW development and for calling non-exported DLL functions
  - API: None
  - Arguments: 1. APIname: API function name 2. 0 or more additional arguments (varies for each API)
- ChangeLogLevel
  - Description: Changes the output logging level. Larger values increase debug logging.
  - API: None
  - Arguments: 1. logLevel: 1-based logging level; default = 1; increase for extra debug logging
- DLP_Misc_EnableCommunication
  - Description: See API

  - API: DLP_Misc_EnableCommunication
  - Arguments: None
- DLP_Misc_DisableCommunication
  - Description: See API
  - API: DLP_Misc_DisableCommunication
  - Arguments: 1. output filename
- DLP_Misc_DisableCommunication_FlushOutputFileToDisk
  - Description: See API
  - API: DLP_Misc_DisableCommunication_FlushOutputFileToDisk
  - Arguments: None
- DLP_Misc_ProgramEDID
  - Description: See API
  - API: DLP_Misc_ProgramEDID
  - Arguments: 1. DMDnum: see DMD_t 2. byteOffset: skip bytes 3. numBytes: number bytes to program 4. filename: input filename
- DLP_Misc_GetTotalNumberOfUSBDevicesConnected
  - Description: Returns the total number of USB devices currently connected to the PC USB driver
  - API: DLP_Misc_GetTotalNumberOfUSBDevicesConnected
  - Arguments: None
- DLP_Misc_SetUSBDeviceNumber
  - Description: See API
  - API: DLP_Misc_SetUSBDeviceNumber
  - Arguments:
    - devNum0b: 0-based USB device number; device number defaults
- DLP_Misc_GetUSBDeviceNumber
  - Description: Returns the current USB device number (0-based) that API calls are being sent to
  - API: DLP_Misc_GetUSBDeviceNumber
  - Arguments: None

4.25 Batchfile LUT/Register Commands

- ReadReg
  - Description: Low-level debug API for reading back full 32-bit register values. The DLP Controller Chip's register specification is not a published document.
  - API: DLP_RegIO_ReadRegisterField
  - Arguments: 1. addr: hex or decimal register address; register addresses are currently private entities
  - Output (see logging): See API for output
- WriteReg
  - Description: Low-level debug API for writing 32-bit register values. The DLP Controller Chip's register specification is not a published document.
  - API: DLP_RegIO_WriteRegisterField
  - Arguments: 1. addr: hex or decimal register address; register addresses are currently private entities 2. val32: 32-bit hex or decimal register value
- DLP_RegIO_WriteImageOrderLut
  - Description: See API
  - API: DLP_RegIO_WriteImageOrderLut
  - Arguments: 1. BPP: num bits per pixel for downloaded images 2. arrImgNumbers: list of one or

more index numbers corresponding to the playback order of the downloaded image files stored in the image buffer
  - Example: $L2.5 DLP_RegIO_WriteImageOrderLut 8, 2, 0, 1 Changes Changes the display order of three downloaded images to 2, 0, 1. Images are 8BPP.
- DLP_RegIO_BeginLUTdata
  - Description: See API
  - API: DLP_RegIO_BeginLUTdata
  - Arguments: 1. LUTname: see LUT Names
- DLP_RegIO_EndLUTdata
  - Description: See API
  - API: DLP_RegIO_EndLUTdata
  - Arguments: 1. LUTname: see LUT Names
- DLP_RegIO_InitFromParallelFlashOffset
  - Description: See API
  - API: DLP_RegIO_InitFromParallelFlashOffset
  - Arguments: 1. offset: byte offset = read start addr; should correspond to the starting offset of a solution 2. resetFlag: reset flag; set = 1 to request system reset (mimics power-up initialization)

4.26 Batchfile Source/VSYNC Trigger Commands

- DLP_Source_SetDataSource
  - Description: See API
  - API: DLP_Source_SetDataSource
  - Arguments: 1. source: see DATA_t
- DLP_Trigger_SetExternalTriggerEdge
  - Description: See API
  - API: DLP_Trigger_SetExternalTriggerEdge
  - Arguments: 1. edge: see API

4.27 Batchfile Internal Test Pattern Generator Commands

- DLP_TPG_SetTestPattern
  - Description: See API
  - API: DLP_TPG_SetTestPattern
  - Arguments: 1. DMDnum: see DMD_t 2. patnum: test pattern number; see TPG_t } 3. clrnum: test pattern color; see TPG_Col_t } 4. patFreq: frequency of spatial pattern; see API for valid values

4.28 Batchfile Sync Commands

- DLP_Sync_SetEnable
  - Description: See API
  - API: DLP_Sync_SetEnable
  - Arguments: 1. syncNum: 1-based sync number; range = { 1,2,3 } 2. enableBit: range = { 0,1 }
- DLP_Sync_Configure
  - Description: See API
  - API: DLP_Sync_Configure
  - Arguments: 1. syncnum: 1-based sync number; range = { 1,2,3 } 2. polarity: see API 3. delay: see API 4. width: see API

5 File Documentation

5.1 dlp_types.h File Reference

Defines

- #define DMD_t_CUR_VERNUM 1
- #define DATA_t_CUR_VERNUM 1
- #define TPG_t_CUR_VERNUM 1
- #define TPG_Col_t_CUR_VERNUM 1
- #define LED_t_CUR_VERNUM 1
- #define SEQDATA_t_CUR_VERNUM 1
- #define REGIO_SEQ_LUT "SEQ_LUT"
- #define REGIO_CMT_LUT "CMT_LUT"
- #define REGIO_RWC_LUT "RWC_LUT"
- #define REGIO_UMCTDM_LUT "UMCTDM_LUT"

Enumerations

- enum DMD_t {
  DMD_XGA = 0 }
- enum DATA_t {
  DVI = 0,
  EXP,
  TPG,
  SL_AUTO,
  SL_EXT3P3,
  SL_EXT1P8,
  SL_SW }
- enum TPG_t {
  SOLID = 0,
  HORIZ_RAMP,
  VERT_RAMP,
  HORIZ_LINES,
  DIAG_LINES,
  VERT_LINES,
  HORIZ_STRIPES,
  VERT_STRIPES,
  GRID,
  CHECKERBOARD }
- enum TPG_Col_t {
  TPG_BLACK = 0,
  TPG_RED,
  TPG_GREEN,
  TPG_BLUE,
  TPG_YELLOW,
  TPG_CYAN,
  TPG_MAGENTA,
  TPG_WHITE }
- enum LED_t {
  LED_R = 0,
  LED_G = 1,
  LED_B = 2,
  LED_IR = 3 }
- enum SEQDATA_t {
  SDM_SL = 0,
  SDM_SL_RT = 1,

SDM_VIDEO = 2,
SDM_MIXED = 3,
SDM_OBJ = 4 }

Variables

- static const char arTPGnames [NUM_PATTERNS][32]
- static const char arTPGcolorNames [8][16]
- static char LED_Color [NUM_LEDS][3]

5.2 DlpApi.c File Reference

#include "DlpAPI.h"

#include <stdio.h>

#include "DLP_types.h"

#include "stdarg.h"

#include "string.h"

#include "stdlib.h"

#include "..\DlpAPIlib\DLP_types.h"

Defines

- #define DLP_API_VERSION_NUM "1.6.1"

Functions

- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternManualStep()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)
- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)
- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()
- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash*, UINT8 *pBuf, UINT32 *nBytesInBuf*, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 **out*CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash*, UINT32 *nBytesTo- Erase*)
- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)

- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)
- const char * DLP_Misc_GetVersionString ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)
- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)
- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])
- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)
- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)
- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)
- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)
- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)
- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)
- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)
- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)

- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 (double * exp_out)
- UINT8 DLP_Status_GetSeqDataExposure DLP_Status_GetSeqDataFrameRate (double * fr_out)
- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPatterns (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

5.3 DlpApi.h File Reference

#include <stdio.h>

#include "DLP_types.h"

Functions

- UINT8 DLP_Display_DisplayPatternManualForceFirstPattern ()
- UINT8 DLP_Display_DisplayPatternManualStep()
- UINT8 DLP_Display_DisplayPatternAutoStepForSinglePass ()
- UINT8 DLP_Display_DisplayPatternAutoStepRepeatForMultiplePasses ()
- UINT8 DLP_Display_DisplayStop ()
- UINT8 DLP_Display_ParkDMD ()
- UINT8 DLP_Display_UnparkDMD ()
- UINT8 DLP_Display_SetDegammaEnable (UINT8 enableBit)
- UINT8 DLP_Display_HorizontalFlip (UINT8 enableBit)
- UINT8 DLP_Display_VerticalFlip (UINT8 enableBit)
- void DLP_FlashCompile_SetCommPacketCallback (void(*pf)(UINT8 *packetBuffer, UINT16 nBufBytes))
- void DLP_FlashCompile_FlushCommPacketBuffer ()
- void DLP_FlashCompile_SetCompileMode (UINT8 *enableBit*)
- UINT8 DLP_FlashCompile_GetCompileMode ()
- UINT8 DLP_FlashProgram_ProgramSerialFlash (UINT32 *nSkipBytesInFlash*, UINT8 *pBuf, UINT32 *nBytesInBuf*, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 **out*CRC)
- UINT8 DLP_FlashProgram_ProgramParallelFlash (UINT32 nSkipBytesInFlash, UINT8 *pBuf, UINT32 nBytesInBuf, UINT8(*pf)(double completionPerCent), UINT8 verifyFlag, UINT16 *outCRC)
- UINT8 DLP_FlashProgram_EraseParallelFlash (UINT32 *nSkipBytesInFlash*, UINT32 *nBytesTo- Erase*)
- UINT8 DLP_Img_DownloadBitplanePatternToExtMem (const UINT8 *pBuf, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_Img_DownloadBitplanePatternFromFlashToExtMem (UINT32 flash_byte_offset, UINT32 tot_bytes, UINT16 bpnum0b)
- UINT8 DLP_LED_SetLEDintensity (LED_t LED, double intensity_perCent)
- UINT8 DLP_LED_GetLEDintensity (LED_t LED, double *intensityPerCent)
- void DLP_LED_LEDdriverEnable (UINT8 enable)
- void DLP_LED_GetLEDdriverTimeout (UINT8 *timedOut)
- UINT8 DLP_LED_SetLEDEnable (LED_t LED, UINT8 enableBit)
- const char * DLP_Misc_GetVersionString ()
- void DLP_Misc_SetLoggingCallback (UINT8 logVisibilityLevel, void(*pf)(const char *))
- void DLP_Misc_SetLoggingVisibilityLevel (UINT8 logVisibilityLevel)

- UINT8 DLP_Misc_InitAPI ()
- UINT8 DLP_Misc_EnableCommunication ()
- UINT8 DLP_Misc_DisableCommunication (const char *ofileName)
- void DLP_Misc_DisableCommunication_FlushOutputFileToDisk ()
- UINT8 DLP_Misc_PassThroughExecute (const char *args[ ])
- UINT8 DLP_Misc_ProgramEDID (DMD_t DMD, UINT8 offset, UINT8 numBytes, const char *fileName)
- UINT8 DLP_RegIO_InitFromParallelFlashOffset (UINT32 offset, UINT8 reset)
- UINT8 DLP_RegIO_ReadLUT (const char * LUTname)
- UINT8 DLP_RegIO_ReadRegisterField (UINT16 addr, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b, UINT32 * ovalue)
- UINT8 DLP_RegIO_WriteImageOrderLut (UINT8 bpp, const UINT16 8 arrStartImgNum, UINT16 nArrElements)
- UINT8 DLP_RegIO_WriteRegisterField (UINT16 addr, UINT32 value, UINT8 full_reg_LSB_bitnum0b, UINT8 full_reg_MSB_bitnum0b)
- UINT8 DLP_RegIO_BeginLUTdata (const char * LUTname)
- UINT8 DLP_RegIO_EndLUTdata (const char * LUTname)
- UINT8 DLP_Source_SetDataSource (DATA_t source)
- UINT8 DLP_Trigger_SetExternalTriggerEdge (UINT8 edge)
- UINT8 DLP_TPG_SetTestPattern (DMD_t DMD, TPG_Col_t testPattern, TPG_Col_t color, UINT16 patternFreq)
- UINT8 DLP_Sync_Configure (UINT8 syncNumber, UINT8 polarity, UINT32 delay, UINT32 width)
- UINT8 DLP_Sync_SetEnable (UINT8 syncNumber, UINT8 enableBit)
- UINT8 DLP_Status_CommunicationStatus ()
- UINT8 DLP_Status_GetBISTdone (UINT8 * state_out)
- UINT8 DLP_Status_GetBISTfail (UINT8 * state_out)
- UINT8 DLP_Status_GetDADcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDADfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetDlpControllerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetDMDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetDMDhardwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDparkState (UINT8 * state_out)
- UINT8 DLP_Status_GetDMDsoftwareParkState (UINT8 * state_out)
- UINT8 DLP_Status_GetEEPROMfault (UINT8 * fault_out)
- UINT8 DLP_Status_GetFlashProgrammingMode (UINT8 * mode_out)
- UINT8 DLP_Status_GetFlashSeqCompilerVersionString (const char ** ver_out)
- UINT8 DLP_Status_GetInitFromParallelFlashFail (UINT8 * state_out)
- UINT8 DLP_Status_GetLEDcommStatus (UINT8 * status_out)
- UINT8 DLP_Status_GetLEDdriverFault (UINT8 * fault_out)
- UINT8 DLP_Status_GetLEDdriverLitState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetLEDdriverTempTimeoutState (LED_t LED, UINT8 * state_out)
- UINT8 DLP_Status_GetMCUversionString (const char ** ver_out)
- UINT8 DLP_Status_GetOverallLEDdriverTempTimeoutState (UINT8 * state_out)
- UINT8 DLP_Status_GetOverallLEDlampLitState (UINT8 * state_out)
- UINT8 DLP_Status_GetSeqDataBPP (UINT8 * bpp_out)
- UINT8 (double * exp_out)
- UINT8 DLP_Status_GetSeqDataExposure DLP_Status_GetSeqDataFrameRate (double * fr_out)

- UINT8 DLP_Status_GetSeqDataMode (SEQDATA_t * mode_out)
- UINT8 DLP_Status_GetSeqDataNumPatterns (UINT16 * numPatterns_out)
- UINT8 DLP_Status_GetSeqRunState (UINT8 * state_out)
- UINT8 DLP_Status_GetUARTfault (UINT8 * fault_out)

5.4 dummy_portabilityLayer.h File Reference

Typedefs

- typedef char Char
- typedef double Double
- typedef signed char Int8
- typedef signed short Int16
- typedef signed int Int32
- typedef unsigned char UInt8
- typedef unsigned short UInt16
- typedef unsigned int UInt32
- typedef const char * String
- typedef UInt8 Byte
- typedef void Void
- typedef int Boolean
- typedef Byte ** IntPtr
- typedef char * StringBuilder

Enumerations

- enum Status {
  STAT_OK,
  STAT_ERROR }

Functions

- Status RunBatchFile (String name, Boolean stopOnError)
- Status RunBatchCommand (String command)
- Status ChangeLogLevel (Byte logLevel)
- Status InitPortabilityLayer (Byte logLevel, Byte detail, OutputCallback callback)
- Status WriteExternalImage (String name, Byte imageIndex)
- IntPtr GetAllBitPlanes (String name, UInt32 bpp, UInt32 *bitPlaneSize)
- Status destroyBitPlanes (IntPtr bitPlanes, UInt32 bpp)

Revision History

Changes from A Revision (September 2010) to B Revision	Page
• Changed the text in the UNIT8 DLP_DisplayPatternAutoStepForSinglePass section	5
• Added a NOTE to the UNIT8 DLP_DisplayPatternAutoStepForSinglePass() section	5

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Revision History

Changes from B Revision (March 2012) to C Revision	Page
• Added section 4.13 PWM Port Configuration APIs	2
• Added PWM Port Configuration APIs to Module Index	3
• Added UINT8 DLP_Source_GenerateSoftwareVsync() API	17
• Added section 4.13 PWM Port Configuration APIs	18

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265