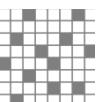


# ALP-4

Controller Suite

Application  
Programming  
Interface



ViALUX Messtechnik + Bildverarbeitung GmbH

Am Erlenwald 10

09128 Chemnitz

Germany

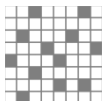
Phone: +49 (0) 371 33 42 47 0

Fax: +49 (0) 371 33 42 47 10

Web: [www.vialux.de](http://www.vialux.de)

E-Mail: [dlp@vialux.de](mailto:dlp@vialux.de)

© 2004-2020 ViALUX GmbH. All rights reserved.



## Table of Contents

1	General introduction.....	1
1.1	ALP Revision Information.....	1
1.2	Supported Products .....	1
1.3	ALP operation .....	1
1.4	ALP API files.....	2
1.5	Device handling .....	3
1.6	Return values.....	3
2	Basic ALP Functions .....	4
2.1	AlpDevAlloc.....	4
2.2	AlpDevControl.....	5
2.3	AlpDevInquire .....	7
2.4	AlpDevControlEx.....	9
2.5	AlpDevHalt.....	10
2.6	AlpDevFree .....	10
2.7	AlpSeqAlloc.....	11
2.8	AlpSeqControl.....	12
2.9	AlpSeqTiming.....	14
2.10	AlpSeqInquire .....	17
2.11	AlpSeqPut.....	19
2.12	AlpSeqPutEx.....	22
2.13	AlpSeqFree .....	27
2.14	AlpProjControl.....	27
2.15	AlpProjInquire .....	29
2.16	AlpProjControlEx.....	29
2.17	AlpProjInquireEx .....	30
2.18	AlpProjStart.....	31
2.19	AlpProjStartCont .....	32
2.20	AlpProjHalt.....	32
2.21	AlpProjWait .....	33
3	Extended ALP functions.....	34
3.1	Scrolling Extension .....	34
3.2	Frame Look up Table (FrameLUT) .....	35

3.3	Sequence Queue Mode .....	38
3.4	Gated Frame Synchronization Output.....	43
3.5	PWM Output .....	47
3.6	Externally Triggered Frame Transition.....	48
3.7	Flexible PWM Mode .....	49
4	LED Control.....	50
4.1	Introduction to the ALP LED API.....	50
4.2	AlpLedAlloc .....	51
4.3	AlpLedFree .....	53
4.4	AlpLedControl .....	53
4.5	AlpLedInquire.....	55
4.6	AlpLedControlEx.....	55
4.7	AlpLedInquireEx.....	56
5	Data types, Functions, Constants .....	56
5.1	Data types.....	57
5.2	List of Functions.....	57
5.3	Constant values .....	60

# ALP-4 Controller Suite: Application Programming Interface

---

## 1 General introduction

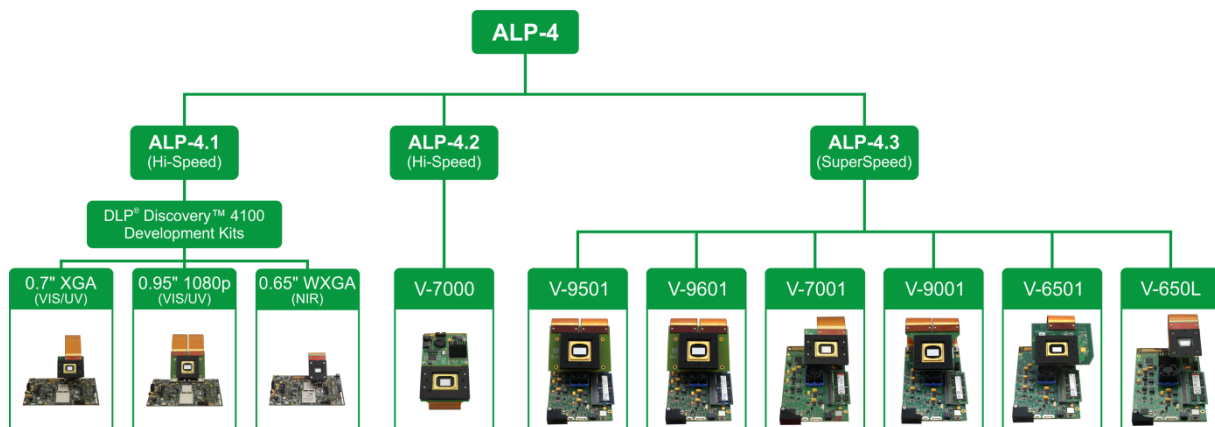
### 1.1 ALP Revision Information

This document summarizes release 19 of the ALP-4 application programming interface. It applies to the following file versions:

API DLL file: alpD41.dll, alpV42.dll 1.0.19.x

API header file (alp.h): 24

### 1.2 Supported Products



### 1.3 ALP operation

The ALP Controller Suite comes with an application programming interface (API) implemented as a dynamic-link library (DLL). It provides all functions required for the use of ALP hardware components. The following notes describe general software organization rules applicable for the whole library.

- Any ALP is identified by its serial number. Multiple ALP devices can be simultaneously controlled by a single PC.
- The API software provides an ALP device identifier (type `ALP_ID`) for each unit.
- The patterns to be displayed are organized in sequences of 1...8-bit pictures. Any sequence is addressed via a sequence handle (type `ALP_ID`).
- The sequences are loaded into ALP RAM using an API function (*AlpSeqPut*). This RAM is not directly accessible by the user.
- Multi-threading is supported. The library functions serialize critical operations internally.
- *AlpDevHalt* stops ALP operation instantly.

- All functions provide a return value (type long). The parameter list may point to other output data. It is strongly recommended to verify the return value always.
- Various programming samples are available in source code and distributed with the ALP Installation. They provide a quick insight into ALP API programming, and may serve as a template for building a custom application.
- Compatibility is maintained for all ALP-4 API versions running on different controller boards and with different DMD formats; in particular all DLP® V-Modules as well as the DLPLCRC410 Evaluation Modules and DLP® Discovery™ 4100 Developer Kits, respectively, can be controlled with the same software. See Release Notes for version-specific comments.

**Please consult the Release Notes for current implementation comments.**

Section 2 of this document contains a comprehensive reference of the ALP API functions. Section 3 describes extended options for advanced users, LED driver control is embedded in the ALP-4 API and the corresponding LED API reference is given in Section 4. Finally Section 5 of this document provides the specification of the interface in case users cannot take advantage of automatic processing of the C header file alp.h.

#### 1.4 ALP API files

The API of ALP-4 consists of a DLL file, an according import library LIB file, and the header file alp.h.

Include alp.h and link the LIB file to your C/C++ application for to use the ALP-4 API. The header declares functions and constant values<sup>1</sup>, and the LIB file cares for loading the DLL and imports the API functions when starting your application.

An additional DLL exports the same functions using “standard call” calling convention. It just translates calling conventions, so it depends on the “main” ALP API DLL. This library might be required for development environments that do not support the C calling convention<sup>2</sup>.

	Header file	Import library	Main DLL (C call)	StdCall DLL	Hardware
<b>ALP-4.1</b>	alp.h	alpD41.lib	alpD41.dll	alpD41S.dll	DLPLCRC410 Evaluation Modules DLP® Discovery™ 4100 Developer Kits
<b>ALP-4.2</b>	alp.h	alpV42.lib	alpV42.dll	alpV42S.dll	V-7000

<sup>1</sup> When using other programming languages, please read alp.h in a text editor, or refer to chapter 5.

<sup>2</sup> For more details please search msdn.microsoft.com for “calling conventions”.

## 1.5 Device handling

The ALP-4.1 and ALP-4.2 controllers are implemented with an encrypted FPGA code. The corresponding Virtex-5 FPGA key is factory installed and supported by a long life battery. The ALP-4.3 FPGA key does not use a battery.

*Important note (for ALP-4.1 only):* Do not overwrite the Virtex-5 FPGA key or remove the battery. The ALP-4.1 FPGA logic on the DLPLCRC410 Evaluation Modules or DLP® Discovery Developer Kit can be overwritten by the user at any time using the configuration PROM. ALP-4.1 can be restored after a power cycle. The SetUsbld.exe program, included in the ALP-4.1 installation, is to be used for switching between both FPGA configurations.

## 1.6 Return values

Some return values are commonly used by many of the API functions. This is an explanation of their general meaning.

### **ALP\_OK**

The function succeeded.

### **ALP\_PARM\_INVALID**

One of the parameters is invalid, or a *ControlType* is not supported.

Valid ranges of *ControlValues* may depend on other settings or the device state, so ensure to set up all parameters consistently.

### **ALP\_ADDR\_INVALID**

Functions that access user data through a pointer parameter (e.g. long *\*UserVarPtr*) return ALP\_ADDR\_INVALID when memory access fails. The most probable cause is that this pointer contains an invalid memory address.

### **ALP\_NOT\_READY**

This return value has different meanings depending on the called function.

*AlpDevAlloc*: The specified ALP is un-available because it is already allocated.

All functions can return it in multi-threading applications to denote that the ALP is currently in use by another thread.

### **ALP\_NOT\_IDLE**

To execute the function, the ALP must not display any sequence. Currently the projection loop of an *arbitrary* sequence is running. A concurrent *AlpSeqPut* may also inhibit execution of this function.

**ALP\_SEQ\_IN\_USE**

There are operations that are mutually exclusive using *the same* sequence. For example, a running projection loop may inhibit writing image data (*AlpSeqPut*) to the same sequence and vice versa.

**ALP\_NOT\_AVAILABLE**

All functions having an input parameter *DeviceId* can return this value. The specified *DeviceId* is invalid. Create one using *AlpDevAlloc*.

**ALP\_ERROR\_COMM, ALP\_DEVICE\_REMOVED**

Most of the ALP API functions communicate with the ALP device over the USB. These functions provide the following additional return values when a USB error occurs:

ALP_ERROR_COMM	a communication error occurred during the operation
ALP_DEVICE_REMOVED	the device has been disconnected

USB connection to the device can be checked using *AlpDevInquire*(ALP\_USB\_CONNECTION). *AlpDevControl*(ALP\_USB\_CONNECTION) can be used to re-connect to the device after a transient USB interruption.

**ALP\_ERROR\_POWER\_DOWN**

The DMD has failed to “wake up” from ALP\_DMD\_POWER\_FLOAT mode.

**ALP\_LOADER\_VERSION, ALP\_DRIVER\_VERSION**

A feature is missing in the installed ALP drivers. Update drivers and power-cycle device.

The latest ALP drivers are usually contained in the ALP installation file, available as download from [vialux.de](http://vialux.de).

**ALP\_SDRAM\_INIT**

SDRAM Initialization failed. Switch off the device and check on-board SO-DIMM.

**2 Basic ALP Functions****2.1 AlpDevAlloc****Format**

long AlpDevAlloc (long *DeviceNum*, long *InitFlag*, ALP\_ID \**DeviceIdPtr*)



## Description

This function allocates an ALP hardware system (board set) and returns an ALP handle so that it can be used by subsequent API functions.

An error is reported if the requested device is not available or not ready.

When you no longer need a particular ALP system, free it using *AlpDevFree*.

When terminating the ALP system, use *AlpDevFree* *before* disconnecting it from the USB to avoid problems after USB re-connection.

## Parameters

*DeviceNum* specifies the device to be used. Set this parameter to one of the following values:

ALP_DEFAULT	the next available system is allocated
ALP serial number	the system with the specified serial number is allocated

*InitFlag* specifies the type of initialization to perform on the selected system. This parameter can be set to one of the following:

ALP_DEFAULT	default initialization
-------------	------------------------

*DeviceIdPtr* specifies the address of the variable in which to write the ALP device identifier.

## Return values

ALP_OK	no errors
ALP_ADDR_INVALID	user data access not valid
ALP_NOT_ONLINE	specified ALP not found
ALP_NOT_READY	specified ALP already allocated
ALP_ERROR_INIT	initialization error
ALP_LOADER_VERSION	(ALP-4.1 only) This DLL requires the driver file VlxUsbLd.sys of at least version 0.1.0.22. Please update it and restart the ALP device.

## 2.2 AlpDevControl

### Format

long AlpDevControl (ALP\_ID *DeviceId*, long *ControlType*, long *ControlValue*)

### Description

This function is used to change the display properties of the ALP. The default values are assigned during device allocation by *AlpDevAlloc*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>ControlType</i>	control parameter that is to be modified
<i>ControlValue</i>	value of the parameter

The following settings are available:

ControlType	ControlValue	Description
ALP_SYNCH_POLARITY	ALP_LEVEL_HIGH or ALP_DEFAULT	active high frame synch output signal polarity
	ALP_LEVEL_LOW	active low frame synch output signal polarity
ALP_TRIGGER_EDGE	ALP_EDGE_FALLING or ALP_DEFAULT	high to low trigger input signal transition
	ALP_EDGE_RISING	low to high trigger input signal transition
ALP_DEV_DMDTYPE	See “ALP_DEV_DMDTYPE: List of supported DLP chips” below	ALP-4 is available with different DLP chip (DMD types). It detects automatically which DLP chip is connected, so this feature should not be necessary in most cases. For testing purposes, this <i>ControlType</i> can be used to force the API to behave as if another specified DLP chip is connected. See also <i>AlpSeqPut</i> .  <i>Note:</i> DMD type selection is accepted only before the first call of <i>AlpSeqAlloc</i> after <i>AlpDevAlloc</i> .
ALP_DEV_DMD_MODE	ALP_DMD_POWER_FLOAT	Set the whole DMD to an inactive state. All micro-mirrors are released from deflected to an almost flat (floating) state.  Sequence display is not available in this state, but ALP settings and memory are preserved.
	ALP_DMD_RESUME or ALP_DEFAULT	Wake up DMD from inactive state.
ALP_USB_CONNECTION	ALP_DEFAULT	Trigger a re-connect to the device after a temporary USB disconnect.
ALP_PWM_LEVEL	0 to 100 (Percentage)	duty cycle of the PWM pin, see PWM Output below

**ALP\_DEV\_DMDTYPE: List of supported DLP chips**

ControlValue	DLP chip	Note
ALP_DMDTYPE_XGA		For compatibility to ALP-3; maps to ALP_DMDTYPE_XGA_07A
ALP_DMDTYPE_XGA_07A	DLP7000	1024x768 pixels
ALP_DMDTYPE_XGA_055X		obsolete
ALP_DMDTYPE_1080P_095A	DLP9500	1920x1080 pixels
ALP_DMDTYPE_WUXGA_096A	DLP9600	1920x1200 pixels
ALP_DMDTYPE_WXGA_S450	DLP650LNIR	1280x800 pixels

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_NOT_IDLE	the specified ALP is not in idle state
ALP_PARM_INVALID	one of the parameters is invalid
ALP_NOT_ONLINE	the specified ALP was not found (valid return code when <i>ControlType</i> =ALP_USB_CONNECTION)
ALP_NOT_CONFIGURED	The specified ALP might have lost configuration data as a result of a power blackout. The device must be reconfigured using <i>AlpDevFree</i> and <i>AlpDevAlloc</i> .
ALP_ERROR_POWER_DOWN	The DMD has failed to “wake up” from ALP_DMD_POWER_FLOAT mode. This can be caused by a voltage drop during initialization of the DMD.  The return code is only valid if <i>ControlType</i> =ALP_DEV_DMD_MODE and <i>ControlValue</i> =ALP_DMD_RESUME.

**2.3 AlpDevInquire****Format**

long AlpDevInquire (ALP\_ID *DeviceId*, long *InquireType*, long *\*UserVarPtr*)

**Description**

This function inquires a parameter setting of the specified ALP device.

**Parameter**

<i>DeviceId</i>	ALP device identifier for which the information is requested
<i>InquireType</i>	specifies the ALP device parameter setting to inquire; See the table below.

*UserVarPtr* specifies the address of the variable in which the requested information is to be written. The variable must be of type long.

The *InquireType* supports the following values:

InquireType	Description
ALP_DEVICE_NUMBER	Serial number of the ALP device
ALP_VERSION	Version number of the ALP device, e.g. 0x0401 for ALP-4.1
ALP_AVAIL_MEMORY	ALP on-board sequence memory available for further sequence allocation ( <i>AlpSeqAlloc</i> ) – number of binary pictures; The returned number of binary pictures represents the largest free memory area available for sequence allocation. If ALP memory is fragmented because of repeated calls of <i>AlpSeqAlloc</i> and <i>AlpSeqFree</i> then this value may differ from the total non-allocated memory.
ALP_SYNCH_POLARITY	Frame synch output signal polarity: ALP_LEVEL_HIGH or ALP_LEVEL_LOW
ALP_TRIGGER_EDGE	trigger input signal slope: ALP_EDGE_FALLING or ALP_EDGE_RISING
ALP_DEV_DMDTYPE	write the DMD type to <i>*UserVarPtr</i> : See “ALP_DEV_DMDTYPE: List of supported DLP chips” above, or ALP_DMDTYPE_DISCONNECT If no DMD is detected (ALP_DMDTYPE_DISCONNECT), then the API emulates a 1080p DMD.
ALP_DEV_DMD_MODE	Write ALP_DMD_POWER_FLOAT or ALP_DMD_RESUME to <i>*UserVarPtr</i> . ALP_DMD_POWER_FLOAT can be entered either by <i>AlpDevControl</i> or by a voltage drop in the primary power supply.
ALP_DEV_DISPLAY_HEIGHT	number of mirror rows on the DMD, according to ALP_DEV_DMDTYPE
ALP_DEV_DISPLAY_WIDTH	number of mirror columns on the DMD, according to ALP_DEV_DMDTYPE
ALP_USB_CONNECTION	Check, if the USB connection is ok ( <i>*UserVarPtr</i> is set to ALP_DEFAULT) or the device is disconnected ( <i>*UserVarPtr</i> becomes ALP_DEVICE_REMOVED)
ALP_DDC_FPGA_TEMPERATURE, ALP_APPS_FPGA_TEMPERATURE	ALP-4.2 only: measure the temperature of the DDC FPGA (IC4) or Applications FPGA (IC3); (see below) The value is written as 1/256 °C to <i>*UserVarPtr</i> . It ranges from -128 °C to +127.96875 °C.
ALP_PCB_TEMPERATURE	ALP-4.2 only: measure the internal temperature of the temperature sensor IC (IC22); (see below) The value is written as 1/256 °C to <i>*UserVarPtr</i> . It ranges from -128 °C to +127.75 °C. Accuracy: +/- 3 °C
ALP_PWM_LEVEL	Percentage: duty cycle of the PWM pin, see PWM Output below

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ADDR_INVALID	user data access not valid
ALP_DEVICE_REMOVED	The device has been disconnected.

**ALP Temperature Measurement**

The VIALUX DLP controller boards (except VX4100) have a temperature sensor IC installed and connected to both FPGAs. The API of ALP-4.2 allows reading out the ICs local temperature (ALP\_PCB\_TEMPERATURE) as well as both FPGA temperatures (ALP\_DDC\_FPGA\_TEMPERATURE, ALP\_APPS\_FPGA\_TEMPERATURE).

Maximum FPGA Temperature: 80 °C.

The API returns a number in a fixed-precision format with 1 LSB=1/256 °C. Just divide \*UserVarPtr by 256 for converting it to a °C scale.

The return value ALP\_ERROR\_COMM can be caused by disturbance of the on-board I2C bus. A constant value of -128 °C points out conversion problems.

**2.4 AlpDevControlEx****Format**

long AlpDevControlEx (ALP\_ID *DeviceId*, long *ControlType*, void \**UserStructPtr*)

**Description**

Data objects that do not fit into a simple 32-bit number can be written using this function. Meaning and layout of the data depend on the *ControlType*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>ControlType</i>	name of the control parameter
<i>UserStructPtr</i>	pointer to a data structure whose values shall be send to the device

The following *ControlTypes* are supported:

ControlType	Description
ALP_DEV_DYN_SYNCH_OUT1_GATE, ALP_DEV_DYN_SYNCH_OUT2_GATE, ALP_DEV_DYN_SYNCH_OUT3_GATE	Set up synchronization pins to conditionally output frame synch pulses. See also Gated Frame Synchronization Output.  <i>UserStructPtr</i> points to a structure of type <i>tAlpDynSynchOutGate</i> .

**Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid

**2.5 AlpDevHalt****Format**

long AlpDevHalt (ALP\_ID *DeviceId*)

**Description**

This function puts the ALP in an idle wait state. Current sequence display is canceled (ALP\_PROJ\_IDLE) and the loading of sequences is aborted (*AlpSeqPut*).

**Parameter**

*DeviceId*                      ALP device identifier

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid

**2.6 AlpDevFree****Format**

long AlpDevFree (ALP\_ID *DeviceId*)

**Description**

This function de-allocates a previously allocated ALP device. The memory reserved by calling *AlpSeqAlloc* is also released.

The ALP has to be in idle wait state, see also *AlpDevHalt*.

**Parameter**

*DeviceId*                      ALP identifier of the device to be freed

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_NOT_IDLE	the ALP is not in idle state

**2.7 AlpSeqAlloc****Format**

long AlpSeqAlloc (ALP\_ID *DeviceId*, long *BitPlanes*, long *PicNum*, ALP\_ID \**SequenceIdPtr*)

**Description**

This function provides ALP memory for a sequence of pictures. All pictures of a sequence have the same bit depth. The function allocates memory from the ALP board RAM. The user has no direct read/write access. ALP functions provide data transfer using the sequence memory identifier (*SequenceId*) of type ALP\_ID.

Pictures can be loaded into the ALP RAM using the *AlpSeqPut* function.

The availability of ALP memory can be tested using the *AlpDevInquire* function.

When a sequence is no longer required, release it using *AlpSeqFree*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>BitPlanes</i>	bit depth of the patterns to be displayed; the following values are supported: 1, 2, 3, 4, 5, 6, 7, 8
<i>PicNum</i>	number of pictures belonging to the sequence; possible values depend upon the available memory (ALP_AVAIL_MEMORY) and the bit depth (BitPlanes)
<i>SequenceIdPtr</i>	specifies the address of the variable in which the ALP sequence identifier is to be written.

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ADDR_INVALID	user data access invalid
ALP_MEMORY_FULL	the memory requested is not available

**2.8 AlpSeqControl****Format**

long AlpSeqControl (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*, long *ControlType*, long *ControlValue*)

**Description**

This function is used to change the display properties of a sequence. The default values are assigned during sequence allocation by *AlpSeqAlloc*.

It is allowed to change settings of sequences that are currently in use. However the new settings become effective after restart using *AlpProjStart* or *AlpProjStartCont*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>SequenceId</i>	ALP sequence identifier
<i>ControlType</i>	control parameter that is to be modified
<i>ControlValue</i>	value of the parameter

The following settings are available:

ControlType	ControlValue	Description
ALP_SEQ_REPEAT	In the non-continuous mode ( <i>AlpProjStart</i> ), the projection loop can be configured to execute the sequence projection a certain number of iterations.	
	ALP_DEFAULT	single display of the sequence
	<number> 1 ... 1048576	the sequence is displayed this number of times
ALP_FIRSTFRAME ALP_LASTFRAME	<picture number> 0 ... <i>PicNum</i> – 1	a sequence can be displayed partially, the number of the first picture must be equal or lower than the number of the last picture



ControlType	ControlValue	Description
ALP_BITNUM	<bit number> 1 ... <i>BitPlanes</i>	a sequence can be displayed with reduced bit depth for faster speed; <i>Note:</i> A subsequent call of <i>AlpSeqTiming</i> is necessary, to adjust the sequence timing and to make the new bit depth effective. Until then the timing will not change.
ALP_BIN_MODE	In the binary mode (ALP_BITPLANES = 1 or ALP_BITNUM = 1) this control type can be used to adjust sequence display to have no dark phase between successive pictures. <i>Note:</i> This function has an impact on other timing settings. A subsequent call of <i>AlpSeqTiming</i> is necessary to maintain consistent sequence timing. The new mode becomes effective after this <i>AlpSeqTiming</i> call.	
	ALP_BIN_NORMAL or ALP_DEFAULT	Normal operation with programmable dark phase
	ALP_BIN_UNINTERRUPTED	Operation without dark phase (ALP_OFF_TIME=0)
ALP_DATA_FORMAT	Different formats are available for the image data. See also <i>AlpSeqPut</i> for more information.	
	ALP_DATA_MSB_ALIGN or ALP_DEFAULT	The gray value of a pixel is stored in one byte. The most significant bit plane is stored in bit 7 of each byte.
	ALP_DATA_LSB_ALIGN	The gray value of a pixel is stored in one byte. The least significant bit plane is stored in bit 0 of each byte.
	ALP_DATA_BINARY_TOPDOWN	Each bit plane is stored in its contiguous memory area, row 0 first.
	ALP_DATA_BINARY_BOTTOMUP	Each bit plane is stored in its contiguous memory area, row 0 last.
ALP_SEQ_PUT_LOCK	ALP_DEFAULT	Usually image data of a sequence are protected against writing ( <i>AlpSeqPut</i> ) during display.
	Not ALP_DEFAULT	Unlock sequence and allow starting of <i>AlpSeqPut</i> concurrently with sequence display. <i>Note:</i> This will likely cause transient image distortion.
ALP_SCROLL_FROM_ROW, ALP_SCROLL_TO_ROW, ALP_FIRSTLINE, ALP_LASTLINE, ALP_LINE_INC	See Scrolling Extension	
ALP_FLUT_MODE, ALP_FLUT_ENTRIES9, ALP_FLUT_OFFSET9	See Frame Look up Table (FrameLUT)	
ALP_PWM_MODE	See Flexible PWM Mode	

**Return values**

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid

**2.9 AlpSeqTiming****Format**

long AlpSeqTiming (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*, long *IlluminateTime*, long *PictureTime*, long *SynchDelay*, long *SynchPulseWidth*, long *TriggerInDelay*)

**Description**

This function controls the timing properties of the sequence display. Default values are assigned during sequence allocation (*AlpSeqAlloc*).

All timing parameters as well as some of their limits can be inquired using the *AlpSeqInquire* function.

*Note:* It is allowed to change settings of sequences that are currently in use. However, the new settings become effective after restart using *AlpProjStart* or *AlpProjStartCont*.

**Parameters**

*DeviceId* ALP device identifier

*SequenceId* ALP sequence identifier

*IlluminateTime* duration of the display of one picture in the sequence

ALP_DEFAULT	The sequence is displayed with the highest possible contrast available for the specified <i>PictureTime</i> .
<microseconds>	Time during that a single picture of the sequence is displayed; if the value is too small then ALP_DEFAULT is effective.

*PictureTime* time between the start of two consecutive pictures (i.e. this parameter defines the image display rate)

ALP_DEFAULT	If <i>IlluminateTime</i> is also ALP_DEFAULT then 33334 $\mu$ s are used according to a frame rate of 30 Hz. Otherwise <i>PictureTime</i> is set to minimize the dark time according to the specified <i>IlluminateTime</i> .
<microseconds>, up to $10^7 \mu$ s=10 s	If the value is too small then <i>AlpSeqTiming</i> returns ALP_PARM_INVALID.

**SynchDelay** specifies the time between start of the frame synch output pulse and the start of the display (master mode).

ALP_DEFAULT	0
<microseconds> 0...130,000	delay of the display start with respect to the trigger output (master mode)

**SynchPulseWidth** specifies the duration of the frame synch output pulse.

ALP_DEFAULT	= <i>SynchDelay</i> + <i>IlluminateTime</i> in normal mode, that is the pulse finishes at the same time as Illumination = $\frac{1}{2} * \text{ALP\_ILLUMINATE\_TIME}$ (if sequence is set to binary uninterrupted mode)
<microseconds> 1...max	length of the trigger signal, the maximum value is ALP_PICTURE_TIME

**TriggerInDelay** specifies the time between the incoming trigger edge and the start of the display (slave mode).

ALP_DEFAULT	0
<microseconds> 0 ... 130,000	delay of the start of the display with respect to the trigger input signal (slave mode)

Additional constraints for timing parameters (details are below):

$$\text{PictureTime} - \text{IlluminateTime} \geq \Delta t_1$$

$$\text{SynchDelay} \leq \text{PictureTime} - \text{IlluminateTime} - 2\mu\text{s}$$

$$\text{SynchPulseWidth} \leq \text{PictureTime} - \text{TriggerInDelay} - 1\mu\text{s}$$

DMD format	DMD part number	Minimum Dark Phase $\Delta t_1$
XGA types	DLP7000BFLP DLP7000UVFLP	44 $\mu\text{s}$
1080p	DLP9500BFLN DLP9500UVFLN	93 $\mu\text{s}$ (ALP-4.1)
WUXGA	1912N1237	103 $\mu\text{s}$ (ALP-4.1)
WXGA	DLP650LNIRFYL	93 $\mu\text{s}$ (ALP-4.1)

### Notes and Limitations: *IlluminateTime* and *PictureTime*

The *PictureTime* is the most important parameter for controlling frame rate. It defines an interval that contains the actual display of one frame as well as all related trigger and synch processing.



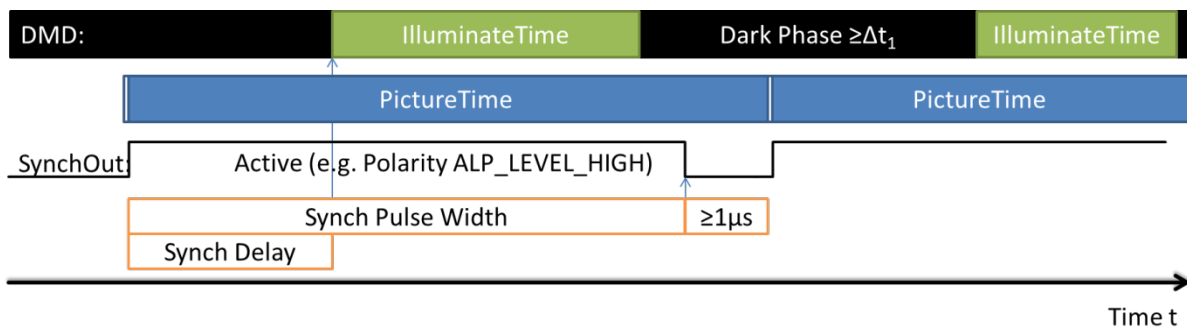
The frame display processing is done during a part of *PictureTime* called *IlluminateTime*. Afterwards it takes some time to initialize the next frame. During this time the DMD is cleared. This so-called dark phase determines the minimum difference  $\Delta t_1$  between *IlluminateTime* and *PictureTime*; that is,  $PictureTime - IlluminateTime \geq \Delta t_1$  (see the table above). The default value (ALP\_DEFAULT) can be used for *IlluminateTime* or *PictureTime*. If *IlluminateTime* is invalid then it is handled like ALP\_DEFAULT.

In the binary mode without dark phase (ALP\_BIN\_UNINTERRUPTED) the processing of a frame completes when it appears on the DMD. So in this mode the *IlluminateTime* is ignored.

Minimum values for *IlluminateTime* and *PictureTime* depend on the DMD type, ALP\_BITNUM, and ALP\_BIN\_MODE. They can be inquired using *AlpSeqInquire*.

### Notes and Limitations: Synch timing in ALP\_MASTER mode

As mentioned above, the complete synchronization output processing of a frame has to happen within its *PictureTime* interval. In master mode, the ALP displays frames and produces synch pulses solely based on internal timing. The *TriggerInDelay* setting is ignored.



The synch pulse starts at the beginning of *PictureTime* and stops after *SynchPulseWidth*. If the pulse width is not specified (ALP\_DEFAULT) then the pulse finishes at the end of illumination. When using ALP\_BINARY\_UNINTERRUPTED mode, the default *SynchPulseWidth* is half of *PictureTime*.

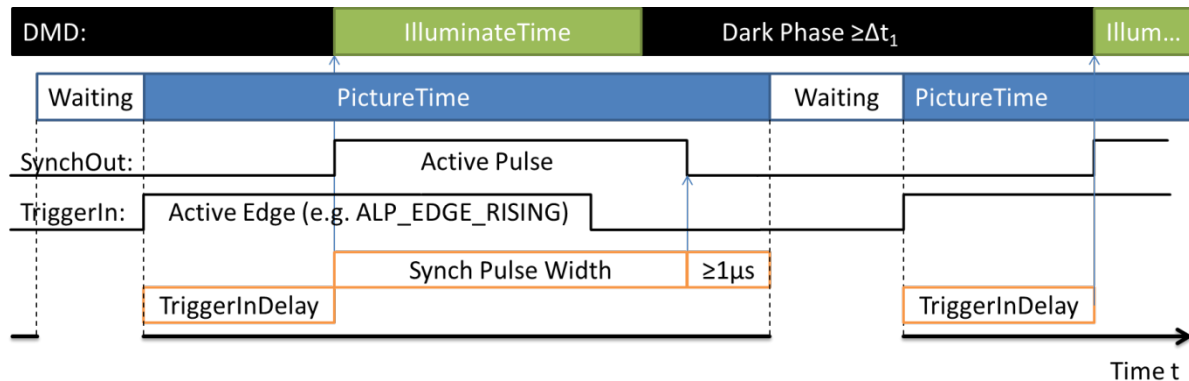
Illumination is delayed from the beginning of the *PictureTime* interval by *SynchDelay*, which is 0  $\mu s$  by default. Frame display must be completed until 2  $\mu s$  before *PictureTime* elapses. So there is a limit:  $SynchDelay \leq PictureTime - IlluminateTime - 2\mu s$ . This limit can be inquired

after *AlpSeqTiming* (with ALP\_DEFAULT as Synch and Trigger Delay) using *AlpSeqInquire*(ALP\_MAX\_SYNCH\_DELAY).

### Notes and Limitations: Synch timing in ALP\_SLAVE mode

In slave mode the timer is paused at the beginning of the *PictureTime* interval. The ALP waits until it detects the selected edge (ALP\_TRIGGER\_EDGE) at the trigger input.

Then the timer continues and, after *TriggerInDelay* elapses, it starts the illumination. The frame synch output pulse is started simultaneously and the *SynchDelay* setting is ignored in slave mode.



*SynchPulseWidth* is evaluated even in slave mode, but it can also be ALP\_DEFAULT in order to finish the frame synchronization pulse at the end of illumination.

The ALP API allows changing ALP\_PROJ\_MODE between *AlpSeqTiming* and *AlpProjStart*. That's why it has to enforce consistent parameters for ALP\_MASTER and ALP\_SLAVE mode. This leads to another constraint:  $SynchPulseWidth \leq PictureTime - TriggerInDelay - 1\mu s$ .

The ALP device obviously becomes ready for the next trigger input edge after *PictureTime* elapses. The trigger period must not fall below *PictureTime*. Otherwise a premature trigger slope will be unnoticed.

### Return values

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_SEQ_IN_USE	the specified sequence is currently in use

## 2.10 AlpSeqInquire

### Format

long AlpSeqInquire (ALP\_ID DeviceId, ALP\_ID SequenceId, long InquireType, long \*UserVarPtr)

## Description

This function provides information about the settings of the specified picture sequence. The settings are controlled either during allocation (*AlpSeqAlloc*) or using the *AlpSeqControl* and *AlpSeqTiming* functions, respectively.

Note that updated values can be queried for ALP\_MIN\_PICTURE\_TIME and ALP\_MIN\_ILLUMINATION\_TIME immediately after modification of ALP\_BITNUM and ALP\_BIN\_MODE (*AlpSeqControl*). The other timing settings are updated by *AlpSeqTiming*.

## Parameters

<i>DeviceId</i>	ALP device identifier
<i>SequenceId</i>	ALP sequence identifier
<i>InquireType</i>	specifies the sequence parameter setting to inquire.
<i>UserVarPtr</i>	specifies the address of the variable in which the requested information is to be written.

The *InquireType* parameter can be set to one of the following values:

InquireType	Description
ALP_BITPLANES	bit depth of the pictures in the sequence
ALP_BITNUM	bit depth for display
ALP_BIN_MODE	sequence display in binary mode
ALP_PICNUM	number of pictures in the sequence
ALP_FIRSTFRAME	number of the first picture in the sequence selected for display
ALP_LASTFRAME	number of the last picture in the sequence selected for display
ALP_FIRSTLINE, ALP_LASTLINE, ALP_SCROLL_FROM_ROW, ALP_SCROLL_TO_ROW, ALP_LINE_INC	Scrolling parameters, see section Scrolling Extension
ALP_SEQ_REPEAT	number of automatically repeated displays of the sequence
ALP_PICTURE_TIME	time between the start of consecutive pictures in the sequence in $\mu\text{s}$ , the corresponding picture rate [fps] = $1,000,000 / \text{ALP\_PICTURE\_TIME} [\mu\text{s}]$
ALP_MIN_PICTURE_TIME	minimum time between the start of consecutive pictures
ALP_MAX_PICTURE_TIME	maximum time between the start of consecutive pictures
ALP_ILLUMINATE_TIME	duration of the display of one picture in $\mu\text{s}$
ALP_MIN_ILLUMINATE_TIME	minimum duration of the display of one picture in $\mu\text{s}$
ALP_ON_TIME	total active projection time
ALP_OFF_TIME	total inactive projection time

InquireType	Description
ALP_SYNCH_DELAY	delay of the start of picture display with respect to the trigger output (master mode) in $\mu$ s
ALP_MAX_SYNCH_DELAY	maximal duration of trigger delay in $\mu$ s
ALP_SYNCH_PULSEWIDTH	duration of the output trigger in $\mu$ s
ALP_TRIGGER_IN_DELAY	delay of the start of picture display with respect to the trigger input in $\mu$ s
ALP_MAX_TRIGGER_IN_DELAY	maximal duration of trigger delay in $\mu$ s
ALP_DATA_FORMAT	currently selected image data format (see also <i>AlpSeqControl</i> , <i>AlpSeqPut</i> )
ALP_SEQ_PUT_LOCK	Protect sequence data against writing ( <i>AlpSeqPut</i> ) during display.
ALP_FLUT_MODE, ALP_FLUT_ENTRIES9, ALP_FLUT_OFFSET9	See section Frame Look up Table (FrameLUT)
ALP_PWM_MODE	See section Flexible PWM Mode

### Return values

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_PARM_INVALID	one of the parameters is invalid

## 2.11 AlpSeqPut

### Format

long AlpSeqPut (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*, long *PicOffset*, long *PicLoad*, void *\*UserArrayPtr*)

### Description

This function allows loading user supplied data via the USB connection into the ALP memory of a previously allocated sequence (*AlpSeqAlloc*) or a part of such a sequence. The loading operation can run concurrently to the display of *other* sequences. Data cannot be loaded into sequences that are currently started for display. *Note*: This protection can be disabled by ALP\_SEQ\_PUT\_LOCK.

The function loads *PicNum* pictures into the ALP memory reserved for the specified sequence starting at picture *PicOffset*. The calling program is suspended until the loading operation is completed.

The ALP API compresses image data before sending it over USB. This results in a virtual improvement of data transfer speed. Compression ratio is expected to vary depending on image data. Incompressible data do not cause overhead delays.

**Parameters**

*DeviceId* ALP device identifier

*SequenceId* ALP sequence identifier

*PicOffset* Picture number in the sequence (starting at 0) where the data upload is started; the meaning depends upon ALP\_DATA\_FORMAT. The following values are allowed:

ALP_DEFAULT	0
<picture number>	0...ALP_PICNUM – 1 (ALP_DATA_MSB_ALIGN or ALP_DATA_LSB_ALIGN data format)
<bit plane number>	0...ALP_PICNUM*ALP_BITPLANES – 1 (ALP_DATA_BINARY_TOPDOWN, ALP_DATA_BINARY_BOTTOMUP)

*PicLoad* number of pictures that are to be loaded into the sequence memory;  
Depending on ALP\_DATA\_FORMAT the following values are allowed:

ALP_DEFAULT	load a complete sequence
<number of pictures>	1 ... ALP_PICNUM – <i>PicOffset</i> (ALP_DATA_MSB_ALIGN or ALP_DATA_LSB_ALIGN data format)
<number of bit planes>	1 ... ALP_PICNUM*ALP_BITPLANES – <i>PicOffset</i> (ALP_DATA_BINARY_TOPDOWN, ALP_DATA_BINARY_BOTTOMUP)

*UserArrayPtr* pointer to the user data to be loaded

**Data format**

Depending on the ALP\_DATA\_FORMAT setting (*AlpSeqControl*) the input data *UserArrayPtr* as well as *PicOffset* and *PicLoad* parameters are interpreted differently.

By default, user supplied image data consist of a number of gray level images. The gray level of a pixel is coded in one byte. The API extracts the bit planes of each image by means of optimized code that is much faster than the USB transfer.

Alternatively, the user supplied image data can consist of a number of binary frames. This allows more flexibility in bit plane addressing and compact data storing.

The DMD type (*AlpDevControl*, ALP\_DEV\_DMDTYPE) affects the data format by the means of three parameters: Columns, Rows, Stride.



Parameter	Description	XGA	1080p	WUXGA	WXGA
<i>Columns</i>	Number of active mirror columns on the DMD	1024	1920	1920	1280
<i>Rows</i>	Number of active mirror rows on the DMD	768	1080	1200	800
<i>Stride</i>	Number of bytes per row in the binary data format	128	240	240	160

The following sections summarize the available data formats.

#### ALP\_DATA\_MSB\_ALIGN (default)

PicOffset	PicLoad	type of UserArrayPtr
0... <i>PicNum</i> -1	1... <i>PicNum</i> - <i>PicOffset</i>	char unsigned [ <i>PicLoad</i> * <i>Rows</i> * <i>Columns</i> ]

The gray value of the pixel at position  $x$  ( $0 \dots \text{Columns}-1$ ; 0 = left-most),  $y$  ( $0 \dots \text{Rows}-1$ ; 0 = top-most) of image  $\text{PicOffset}+n$  ( $n = 0 \dots \text{PicLoad}-1$ ) is represented by byte  $\text{UserArrayPtr}[n * \text{Columns} * \text{Rows} + y * \text{Columns} + x]$ .

Data are aligned at the highest bit position. The alignment of the least significant bit depends upon *BitPlanes* of this sequence.

Example: *BitPlanes* = 6

5	4	3	2	1	0	X	X
---	---	---	---	---	---	---	---

#### ALP\_DATA\_LSB\_ALIGN

The Pixel/Byte relation is equal to ALP\_DATA\_MSB\_ALIGN.

Data are aligned at the lowest bit position. The alignment of the most significant bit depends upon *BitPlanes* of this sequence.

Example: *BitPlanes* = 6

X	X	5	4	3	2	1	0
---	---	---	---	---	---	---	---

#### ALP\_DATA\_BINARY\_TOPDOWN

PicOffset	PicLoad	type of UserArrayPtr
0... <i>PicNum</i> * <i>BitPlanes</i> -1	1... <i>PicNum</i> * <i>BitPlanes</i> - <i>PicOffset</i>	char unsigned [ <i>PicLoad</i> * <i>Rows</i> * <i>Stride</i> ]

Data are organized in bit planes. One gray-scale image consists of binary weighted bit-planes. They are stored in descending order (MSB first). One bit plane consists of  $\text{Rows} * \text{Stride}$  bytes, that is 96 KiB (XGA) or about 253 KiB (1080p).

Example: *PicNum* = 2, *BitPlanes* = 5 (see *AlpSeqAlloc*), *PicOffset* = 5, *PicLoad* = 4

Image/Bit Plane	0/4	0/3	0/2	0/1	0/0	1/4	1/3	1/2	1/1	1/0
-----------------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Bit planes consist of strides; each stride contains the data of one row in this bit plane. Each image data byte contains 8 adjacent pixel values; bit 7 is the left-most.

Example: the first two strides of *UserArrayPtr* of an XGA bit plane

DMD mirror column	0	1	...	7	8	...	1023
Row 0: Byte.Bit	0.7	0.6	...	0.0	1.7	...	127.0
Row 1: Byte.Bit	128.7	128.6	...	128.0	129.7	...	255.0

Example: the first two strides of *UserArrayPtr* of a 1080p bit plane

Column	0	1	...	7	8	...	1919
Row 0	0.7	0.6	...	0.0	1.7	...	239.0
Row 1	240.7	240.6	...	240.0	241.7	...	479.0

## ALP\_DATA\_BINARY\_BOTTOMUP

This data format differs from ALP\_DATA\_BINARY\_TOPDOWN only in the row alignment. The first image data stride in memory represents the bottom row of DMD mirrors.

### Return values

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ERROR_COMM	the ALP has been disconnected during the loading operation; loading is incomplete
ALP_SEQ_IN_USE	a display is started of the sequence to be loaded
ALP_HALTED	<i>AlpDevHalt</i> has interrupted the execution of <i>AlpSeqPut</i>
ALP_ADDR_INVALID	user data access invalid

## 2.12 AlpSeqPutEx

### Format

long AlpSeqPutEx (ALP\_ID DeviceId, ALP\_ID SequenceId, void \*UserStructPtr,  
void \*UserArrayPtr)

## Description

Image data transfer using *AlpSeqPut* is based on whole DMD frames. Applications that only update small regions inside a frame suffer from overhead of this default behavior. An extended ALP API function is available to reduce this overhead.

The *AlpSeqPutEx* function offers the same functionality as the standard function (*AlpSeqPut*), but in addition, it is possible to select a section within a sequence frame using the *LineOffset* and *LineLoad* parameters of the *tAlpLinePut* data-structure (see below) and update only this section of the SDRAM-memory associated with the sequence for a range of sequence-pictures (selected via the *PicOffset* and *PicLoad* parameters of *tAlpLinePut* in similarity to *AlpSeqPut*).

This results in accelerated transfer-time of small image data updates (due to the fact that the amount of transferred data is reduced).

Therefore, the user only passes the lines of the pictures he wants to update via the *UserArrayPtr* (that would be *PicLoad\*LineLoad* lines in total).

## Parameters

<i>DeviceId</i>	ALP device identifier
<i>SequenceId</i>	ALP sequence identifier
<i>UserStructPtr</i>	Pointer to the <i>tAlpLinePut</i> -data structure (see below), which contains information about the transferred data
<i>UserArrayPtr</i>	Pointer to the user data to be loaded (has to contain only the data of all lines to be updated one after the other)

Type (size in bytes), Function	Member data type (size)	Member (byte offset)
tAlpLinePut (20), AlpDevControlEx	long (4)	TransferMode (0)
	long (4)	PicOffset (4)
	long (4)	PicLoad (8)
	long (4)	LineOffset (12)
	long (4)	LineLoad (16)

<i>TransferMode</i>	Must be set to ALP_PUT_LINES otherwise ALP_PARAM_INVALID is returned
---------------------	--

<i>PicOffset</i>	This parameter is interpreted like the <i>PicOffset</i> parameter used in <i>AlpSeqPut</i> . The value of the parameter must be greater or equal to zero otherwise ALP_PARAM_INVALID is returned.
<i>PicLoad</i>	This parameter is interpreted like the <i>PicLoad</i> parameter used in <i>AlpSeqPut</i> . If the value of this parameter is smaller than zero or if <i>PicOffset+PicLoad</i> is exceeding the boundary of the sequence-memory, ALP_PARAM_INVALID is returned. If the value is zero, the parameter is automatically adjusted to include all pictures of the sequence starting at frame <i>PicOffset</i> .
<i>LineOffset</i>	Defines the offset of the frame-section. The frame-data of this section is transferred for each of the frames selected with <i>PicOffset</i> and <i>PicLoad</i> . The value of this parameter must be greater or equal to zero, otherwise ALP_PARAM_INVALID is returned.
<i>LineLoad</i>	Defines the size of the frame-section. If the value of the parameter is less than zero or if <i>LineOffset+LineLoad</i> exceeds the number of lines per sequence-frame, ALP_PARAM_INVALID is returned. If <i>LineLoad</i> is zero, this value is adjusted to include all lines of the frame, starting at line <i>LineOffset</i> .

### Return values

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ERROR_COMM	the ALP has been disconnected during the loading operation; loading is incomplete
ALP_SEQ_IN_USE	a display is started of the sequence to be loaded
ALP_HALTED	<i>AlpDevHalt</i> has interrupted the execution of <i>AlpSeqPutEx</i>
ALP_ADDR_INVALID	user data access invalid

### 2.12.1 Code-Sample

```
/* This sample-code transfers
5 frames with a section of 100 lines height and an offset
of 50 lines */

ALP_ID DeviceId, SequenceId;
tAlpLinePut AlpLinePut;

// This pointer should point to the image-data
UCHAR* SeqData;

// Allocate device
AlpDevAlloc( ALP_DEFAULT, ALP_DEFAULT, &DeviceId);

// Allocate a sequence with 5 frames
AlpSeqAlloc(DeviceId, 1, 5, &SequenceId);

// Setting up the Parameters for AlpSeqPutEx
AlpLinePut.TransferMode      = ALP_PUT_LINES;
AlpLinePut.PicOffset        = 0;
AlpLinePut.PicLoad           = 5;
AlpLinePut.LineOffset       = 50;
AlpLinePut.LineLoad         = 100;

// Transfer image-data
AlpSeqPutEx(DeviceId, SequenceId, &AlpLinePut, SeqData);
```

Memory layout of the code-example:

F0 ... F4 referring to the frames of the sequence

R000 ... R767 referring to the 768 lines per frame (e.g. XGA)

Sequence-data passed to  
AlpSeqPutEx

F0.R050
...
F0.R149
F1.R050
...
F1.R149
F2.R050
...
F2.R149
F3.R050
...
F3.R149
F4.R050
...
F4.R149

Sequence-data stored  
in the SDRAM

F0.R000
...
F0.R050
...
F0.R149
...
F0.R767
F1.R000
...
F1.R050
...
F1.R149
...
F1.R767
F2.R000
...
F2.R050
...
F2.R149
...
F2.R767
F3.R000
...
F3.R050
...
F3.R149
...
F3.R767
F4.R000
...
F4.R050
...
F4.R149
...
F4.R767

## 2.13 AlpSeqFree

### Format

long AlpSeqFree (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*)

### Description

This function frees a previously allocated sequence. The ALP memory reserved for the specified sequence in the device *DeviceId* is released.

### Parameters

*DeviceId*                      ALP device identifier

*SequenceId*                 ALP sequence identifier

### Return values

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_NOT_IDLE	the ALP is not in the idle wait state
ALP_SEQ_IN_USE	the sequence specified is currently in use
ALP_PARM_INVALID	one of the parameters is invalid

## 2.14 AlpProjControl

### Format

long AlpProjControl (ALP\_ID *DeviceId*, long *ControlType*, long *ControlValue*)

### Description

This function controls the system parameters that are in effect for all sequences. These parameters are maintained until they are modified again or until the ALP is freed. Default values are in effect after ALP allocation. All parameters can be read out using the *AlpProjInquire* function.

This function is only allowed if the ALP is in idle wait state (ALP\_PROJ\_IDLE), which can be enforced by the *AlpProjHalt* function.

### Parameters

*DeviceId*                      ALP device identifier

*ControlType*                 name of the control parameter

*ControlValue*                value of the control parameter

The following settings are available:

ControlType	ControlValue	Description
ALP_PROJ_MODE	The ALP operation is controlled by triggers. The projection loop waits for a trigger event before the next picture of the sequence is displayed. This <i>ControlType</i> is used to select from an internal or external trigger source. The trigger output is always effective, sending out a pulse for every displayed frame. The ALP must be idle, i.e. not executing a projection loop, for this <i>ControlType</i> .	
	ALP_MASTER or ALP_DEFAULT	internal timing triggers ALP operation
	ALP_SLAVE	the transition of an input port triggers frame display
ALP_PROJ_INVERSION	The gray values of the image pixels can be inverted for projection. The ALP is not required to be idle for this <i>ControlType</i> . But because the effect is asynchronous it cannot be expected to take effect immediately upon change.	
	ALP_DEFAULT	normal operation
	not ALP_DEFAULT	reverse dark into bright
ALP_PROJ_UPSIDE_DOWN	The image can be flipped for projection. The ALP is not required to be idle for this <i>ControlType</i> . But because the effect is asynchronous it cannot be expected to take effect immediately upon change.	
	ALP_DEFAULT	normal operation
	not ALP_DEFAULT	flip the pictures upside down
ALP_PROJ_WAIT_UNTIL	The <i>AlpProjWait</i> function blocks program execution until the ALP becomes idle. If sequence timing is adjusted to have a substantially longer <i>PictureTime</i> than <i>IlluminateTime</i> , then there is a remarkable difference between end of illumination and completion of <i>PictureTime</i> of the last frame.	
	ALP_PROJ_WAIT_PIC_TIME (default)	Adjust <i>AlpProjWait</i> to return control after <i>PictureTime</i> is completed.
	ALP_PROJ_WAIT_ILLU_TIME	Adjust <i>AlpProjWait</i> to return after <i>Illumination</i> has finished.
ALP_PROJ_STEP	ALP_DEFAULT ALP_LEVEL_HIGH   LOW ALP_EDGE_RISING   FALLING	See Externally Triggered Frame Transition below

### Return values

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_NOT_IDLE	the ALP is not in the idle wait state



## 2.15 AlpProjInquire

### Format

long AlpProjInquire (ALP\_ID *DeviceId*, long *InquireType*, long *\*UserVarPtr*)

### Description

This function provides information about general ALP settings for the sequence display.

### Parameters

*DeviceId* ALP device identifier for that the information is inquired

*InquireType* property for that the parameter provided. The following values are allowed:

InquireType	Value	Description
ALP_PROJ_MODE	ALP_MASTER or ALP_SLAVE	
ALP_PROJ_STATE	ALP_PROJ_ACTIVE	ALP projection active
	ALP_PROJ_IDLE	no projection active
ALP_PROJ_INVERSION	reverse dark into bright	
ALP_PROJ_UPSIDE_DOWN	flip the pictures upside down	
ALP_PROJ_WAIT_UNTIL	ALP_PROJ_WAIT_PIC_TIME or ALP_PROJ_WAIT_ILLU_TIME: <i>AlpProjWait</i> behavior, see also <i>AlpProjControl</i>	
ALP_FLUT_MAX_ENTRIES9	FrameLUT Size, see also Frame Look up Table (FrameLUT)	
ALP_PROJ_STEP	See Externally Triggered Frame Transition below	

*UserVarPtr* specifies the address of the variable in which the requested information is to be written

### Return values

ALP_OK	no errors
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ADDR_INVALID	user data access invalid

## 2.16 AlpProjControlEx

### Format

long AlpProjControlEx (ALP\_ID *DeviceId*, long *ControlType*, void *\*UserStructPtr*)

**Description**

Data objects that do not fit into a simple 32-bit number can be written using this function. These objects are unique to the ALP device, so they may affect display of all sequences.

Meaning and layout of the data depend on the *ControlType*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>ControlType</i>	name of the control parameter
<i>UserStructPtr</i>	pointer to a data structure whose values shall be send to the device

The following *ControlTypes* are supported:

ControlType	Description
ALP_FLUT_WRITE_9BIT	The FrameLUT entries are sent to the ALP. The required data structure is <i>tFlutWrite</i> , and its members are evaluated according to 9-bit FrameLUT mode. See also Writing the FrameLUT. Note that it is allowed to write the FrameLUT even it is currently in use for sequence display. The application program must guard write and read access as required.
ALP_FLUT_WRITE_18BIT	Same as ALP_FLUT_WRITE_9BIT, but data is evaluated for 18-bit FrameLUT mode.

**2.16.1 Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid

**2.17 AlpProjInquireEx****Format**

long AlpProjInquireEx (ALP\_ID *DeviceId*, long *InquireType*, void \**UserStructPtr*)

**Description**

Data objects that do not fit into a simple 32-bit number can be inquired using this function. Meaning and layout of the data depend on the *InquireType*.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>InquireType</i>	select which information is to be inquired, and select the data structure of <i>UserStructPtr</i>
<i>UserStructPtr</i>	pointer to a data structure which shall be filled out by <i>AlpSeqInquireEx</i>

The following *InquireTypes* are supported:

InquireType	Description
ALP_PROJ_PROGRESS	Retrieve progress information about active sequences and the sequence queue. The required data structure is <i>tAlpProjProgress</i> . See also Inquire Progress of Active Sequences.

**2.17.1 Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid

**2.18 AlpProjStart****Format**

long AlpProjStart (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*)

**Description**

A call to this function causes the display of the specified sequence that was previously loaded by the *AlpSeqPut* function. The sequence is displayed with the number of repetitions controlled by ALP\_SEQ\_REPEAT (once by default). This can be interrupted prematurely using the *AlpProjHalt* function.

The calling program gets control back immediately. Use *AlpProjWait* to synchronize your application if required.

The sequence usage flag (ALP\_SEQ\_IN\_USE) is active for a sequence that is currently selected for display. Data cannot be loaded into this sequence (*AlpSeqPut*) and it cannot be freed. Timing adjustments are active after restart of a sequence.

A transition to the next sequence can take place without any gaps. See also the description of *AlpProjStartCont* for details.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>SequenceId</i>	ALP sequence identifier of the sequence to be displayed

**Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_SEQ_IN_USE	the sequence data are currently loaded ( <i>AlpSeqPut</i> )
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ERROR_POWER_DOWN	The DMD is powered down. Call <i>AlpDevControl</i> (ALP_DEV_DMD_MODE, ALP_DMD_RESUME) first.

**2.19 AlpProjStartCont****Format**

long AlpProjStartCont (ALP\_ID *DeviceId*, ALP\_ID *SequenceId*)

**Description**

This function displays the specified sequence in an infinite loop.

The sequence display can be stopped using *AlpProjHalt* or *AlpDevHalt*.

A transition to the next sequence can take place without any gaps, if a sequence display is currently active. Depending on the start mode of the current sequence, the switch happens after the completion of the *last* repetition (controlled by ALP\_SEQ\_REPEAT, *AlpProjStart*), or after the completion of the *current* repetition (*AlpProjStartCont*). Only one sequence start request can be queued. Further requests are replacing the currently waiting request.

**Parameters**

*DeviceId*                      ALP device identifier

*SequenceId*                  ALP sequence identifier of the sequence to be displayed

**Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	one of the parameters is invalid
ALP_ERROR_POWER_DOWN	The DMD is powered down. Call <i>AlpDevControl</i> (ALP_DEV_DMD_MODE, ALP_DMD_RESUME) first.

**2.20 AlpProjHalt****Format**

long AlpProjHalt (ALP\_ID *DeviceId*)

**Description**

This function can be used to stop a running sequence display and to set the ALP in idle wait state ALP\_PROJ\_IDLE. The running sequence loop is displayed until completion of the current iteration.

This function returns immediately. Use *AlpProjWait* to recognize when the projection is finished.

**Parameters**

*DeviceId*                      ALP device identifier

**Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid

**2.21 AlpProjWait****Format**

long AlpProjWait (ALP\_ID *DeviceId*)

**Description**

This function is used to wait for the completion of the running sequence display.

Using this function during the display of an infinite loop (*AlpProjStartCont*) causes the ALP\_PARM\_INVALID error return value. (This applies to ALP\_PROJ\_LEGACY mode only. See also Inquire Progress of Active Sequences and Legacy Mode Behavior.)

*AlpProjControl* can adjust the timing with *ControlType* ALP\_PROJ\_WAIT\_UNTIL.

**Parameters**

*DeviceId*                      ALP device identifier

**Return values**

ALP_OK	no error
ALP_NOT_AVAILABLE	the specified ALP identifier is not valid
ALP_NOT_READY	the specified ALP is in use by another function
ALP_PARM_INVALID	an infinite projection loop is active

### 3 Extended ALP functions

#### 3.1 Scrolling Extension

By default, an ALP sequence is considered as a number *PicNum* of pictures, each having the same extent as the DMD. The ALP shows them as DMD frames in the order according to their location in ALP memory. When instead thinking of the ALP sequence as of one very high picture (*PicNum*\**DmdHeight* rows), the scrolling extension allows linearly stepping through it by an arbitrary number of rows. In fact, this interpretation transforms the default behavior to a special case having step size=*DmdHeight*.

*AlpSeqControl* has additional *ControlType* parameters to enable the vertically scrolling display: ALP\_SCROLL\_FROM\_ROW, ALP\_SCROLL\_TO\_ROW, ALP\_FIRSTLINE, ALP\_LASTLINE, and ALP\_LINE\_INC.

ControlType and ControlValue	Description
ALP_SCROLL_FROM_ROW, ALP_SCROLL_TO_ROW <line number> 0 ... <i>DmdHeight</i> *( <i>PicNum</i> -1)	Consider the whole ALP sequence as one big picture of <i>PicNum</i> * <i>DmdHeight</i> rows, and select the scroll range. ALP_SCROLL_FROM and TO_ROW select the frames displayed on the DMD: The top-most DMD row is always in this range. Note that complete DMD frames are displayed starting at these rows. This is the reason for the valid range of these parameters being limited to one <i>DmdHeight</i> above the bottom edge of the picture.
ALP_FIRSTLINE, ALP_LASTLINE <line number> 0... <i>DmdHeight</i> -1	Define the first and last top-line of the scroll range, related to ALP_FIRSTFRAME and LASTFRAME. This is just another method to specify this range. (Historically the first method, compatible down to ALP-1). <i>Note:</i> The ALP_FIRSTLINE must be 0 when the ALP_FIRSTFRAME is the last frame of the sequence (i.e. ALP_PICNUM-1). The ALP_LASTLINE must be 0 when the ALP_LASTFRAME is the last frame of the sequence.
ALP_LINE_INC <line increment> can be positive or negative	The top row of subsequent DMD frames differs by this ALP_LINE_INC number of lines. A value of 0 is interpreted as full DMD height, e.g. 768 (XGA). Negative values make scrolling start at ALP_SCROLL_TO_ROW.

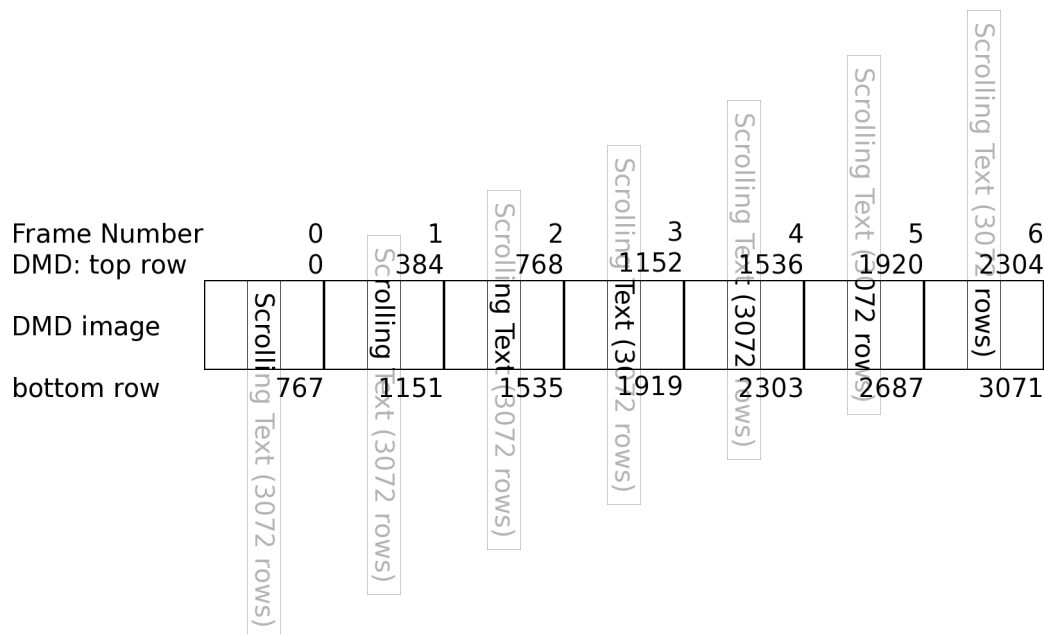
In positive scrolling mode ( $ALP\_LINE\_INC \geq 0$ ) the top-most row of the first displayed frame is  $ALP\_SCROLL\_FROM\_ROW = ALP\_FIRSTLINE + DmdHeight * ALP\_FIRSTFRAME$ . Each subsequent frame starts  $ALP\_LINE\_INC$  rows lower. Scrolling finishes when the DMDs top-most row is  $ALP\_SCROLL\_TO\_ROW = ALP\_LASTLINE + DmdHeight * ALP\_LASTFRAME$ . If the range is not an integer multiple of the step width then  $ALP\_SCROLL\_TO\_ROW$  is never exceeded.

Negative scrolling ( $ALP\_LINE\_INC < 0$ ) behaves similar. Its first frame starts at  $ALP\_SCROLL\_TO\_ROW$ . The display scrolls up until  $ALP\_SCROLL\_FROM\_ROW$  is reached and not exceeded.

Scrolling is disabled by setting  $ALP\_LINE\_INC$ ,  $ALP\_FIRSTLINE$ , and  $ALP\_LASTLINE$  to 0.

Hint: Scroll range parameters must always be consistent, that means  $0 \leq ALP\_SCROLL\_FROM\_ROW \leq ALP\_SCROLL\_TO\_ROW \leq DmdHeight * (PicNum - 1)$ . If both numbers are to be adjusted, then this condition can generally be achieved by the following sequence: First reset  $FROM\_ROW = 0$ , then set  $TO\_ROW$  to the required value, finally set  $FROM\_ROW$ .

Example: Scroll through a picture showing “Scrolling Text (3072 rows)”, having  $4 * DmdHeight$  rows. Step width is set to  $ALP\_LINE\_INC = 384$  (positive, half of XGA DMD), and scroll range is from row 0 to row 2304 ( $= 3072 - 768$ ). This results in the 7 DMD frames presented in the picture below:



Determine the number of DMD frames by division of scroll range extent + 1 by absolute scroll step, and round up.

Example: XGA DMD, i.e.  $DmdHeight = 768$  rows,  $ALP\_SCROLL\_FROM\_ROW = 80$ ,  $ALP\_SCROLL\_TO\_ROW = 808$ ,  $ALP\_LINE\_INC = 8$ . The number of frames shown by this scrolling sequence is:  $\lceil (808 - 80 + 1) / 8 \rceil = 92$ .

The same applies if  $ALP\_LINE\_INC$  is negative with same step size (-8).

### 3.2 Frame Look up Table (FrameLUT)

Besides the linear display of a sequence, the ALP supports a random access order via Look up Table. *AlpSeqControl* supports *ControlTypes* to enable FrameLUT mode, and to select the number of frames to be displayed using the look up table. These settings can be adjusted

at any time, but they affect display only from the next time, the sequence is started (*AlpProjStart*). There is exactly one FrameLUT in the ALP device, so values are written using *AlpProjControlEx* rather than ALP sequence functions.

FrameLUT is available in two modes: 9-bit mode (ALP\_FLUT\_9BIT) supports up to ALP\_FLUT\_MAX\_ENTRIES9 values in the range of 0 to 511. Mode ALP\_FLUT\_18BIT allows values from 0 to  $2^{18}-1=262143$ , but has only half the size: ALP\_FLUT\_MAX\_ENTRIES9 / 2.

The table below shows related ALP API functions together with their *ControlTypes* and *Values*.

Function (Control/InquireType)	Description
<i>AlpProjInquire</i> (ALP_FLUT_MAX_ENTRIES9)	Inquire the size of the FrameLUT. This function writes the available number of 9-bit values to <i>*UserVarPtr</i> .
<i>AlpSeqControl/Inquire</i> (ALP_FLUT_MODE)	Select whether and how this sequence uses the FrameLUT: ALP_FLUT_NONE (default, linear sequence display), ALP_FLUT_9BIT, or ALP_FLUT_18BIT.
<i>AlpSeqControl/Inquire</i> (ALP_FLUT_ENTRIES9)	1 ... ALP_FLUT_MAX_ENTRIES9 (ALP_DEFAULT: 1) Adjust the number of frames to be displayed in FrameLUT 9-bit mode. There is no according 18-bit parameter: ALP_FLUT_18BIT mode displays ALP_FLUT_ENTRIES9 / 2 frames.
<i>AlpSeqControl/Inquire</i> (ALP_FLUT_OFFSET9)	0 ... ALP_FLUT_MAX_ENTRIES9-1 (ALP_DEFAULT: 0), only integer multiples of 256 are supported Select the part of the FrameLUT used by this sequence.
<i>AlpProjControlEx</i> (ALP_FLUT_WRITE_9BIT), <i>AlpProjControlEx</i> (ALP_FLUT_WRITE_18BIT)	Write entry values to the LUT. In both cases <i>*UserStructPtr</i> points to a structure of type <i>tFlutWrite</i> . But the member values of this structure are interpreted according to the <i>ControlType</i> .



## FrameLUT Memory Partitioning

Both FrameLUT modes access the data entries from the same memory. The indexes are mapped as shown in the table below:

Index to FLUT18 (18-bit entries)	Index to FLUT9 (9-bit entries)
0	0
	1
1	2
	3
...	...
2047	4094
	4095

Several sequences can easily share their use of the FrameLUT, as long as the sum of individual `ALP_FLUT_ENTRIES9` does not exceed the available size (`ALP_FLUT_MAX_ENTRIES9`). The `ALP_FLUT_OFFSET9` setting allows selecting which part of the LUT is used by each sequence. It is up to the application program to manage partitions, for example locking parts for exclusive use.

## Writing the FrameLUT

*AlpProjControlEx* supports two *ControlTypes* for that purpose: `ALP_FLUT_WRITE_9BIT` and `ALP_FLUT_WRITE_18BIT`.

Create a variable of type *tFlutWrite* and initialize its contents: *nSize* is the number of FrameLUT entries to be written, *nOffset* selects the destination inside the FrameLUT, and the *FrameNumbers* array contains the actual values of FrameLUT entries.

*AlpProjControlEx* first checks the valid range ( $nSize + nOffset \leq ALP\_FLUT\_MAX\_ENTRIES9$  for `ALP_FLUT_WRITE_9BIT` or  $ALP\_FLUT\_MAX\_ENTRIES9 / 2$  for `ALP_FLUT_WRITE_18BIT`; on error: `ALP_PARM_INVALID`).

Then it transfers the least significant 9 or 18 bits of each *FrameNumber* to the FrameLUT: for all *i* in 0 to *nSize*-1: copy *FrameNumbers[i]* to *FrameLUT[i+nOffset]*.

**Note:** The range of each *FrameNumber* is not validated by ALP. The application program shall limit the values to 9 or 18 bit accordingly. Moreover the ALP API cannot validate that all values address the allocated sequence memory. This must also be ensured by the application.

All parts of the FrameLUT may be written at any time, even when the ALP displays sequences.

## Interaction with Scrolling Parameters

Scrolling parameters and FrameLUT are combined. By default `ALP_LINE_INC=DmdHeight` resulting in true DMD frames to be addressed by the FrameLUT. Internally the FrameLUT

entries are multiplied by `ALP_LINE_INC`. This provides look-up resolution of up to 1 DMD row.

`ALP_FIRSTFRAME` (or more general: the scroll range, `ALP_SCROLL_FROM_ROW`) affects display in FrameLUT modes. FrameLUT entry value 0 shows picture `ALP_FIRSTFRAME`. This allows for example for 9-bit FrameLUT display of the higher part of a sequence of 1024 pictures. It also allows moving through a sequence with always re-using the same FrameLUT access pattern.

DMD frames ( $i=0\dots\text{Entries}-1$ ) are displayed with their top-most row starting from the Sequence Pictures row  $\text{ALP\_SCROLL\_FROM\_ROW} + \text{FrameLUT}[i] * \text{ALP\_LINE\_INC}$ .

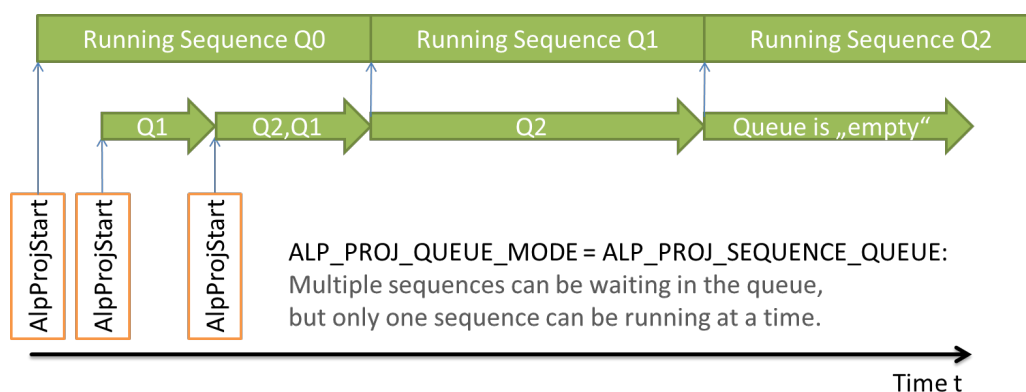
For negative `ALP_LINE_INC` the FrameLUT access is based on `ALP_SCROLL_TO_ROW`.

### 3.3 Sequence Queue Mode

Subsequent ALP sequences can be run without any break in between. This requires that a sequence is already waiting for execution while another one is still running. The last frame of the currently running sequence completes its *PictureTime*, and then the waiting sequence immediately starts its first frame.

This means that two sequences having the same timing setup will display flawlessly. But even when changing timing, the synch, trigger, and illumination timing is well defined, see below.

The ALP API can be switched to `ALP_PROJ_SEQUENCE_QUEUE` mode. In this mode it allows having multiple active sequences. Like in a waiting line each sequence waits until the previous one has finished. Then it starts running automatically. At any given time there can 0 or 1 sequence be *running* in the ALP while 0 to “n” sequences are *waiting*. Both are called *active* sequences.



In `ALP_PROJ_LEGACY` mode (default) the ALP API behaves compatible to previous versions. It emulates 1 waiting position with some special behavior, see below.

#### Related API Controls

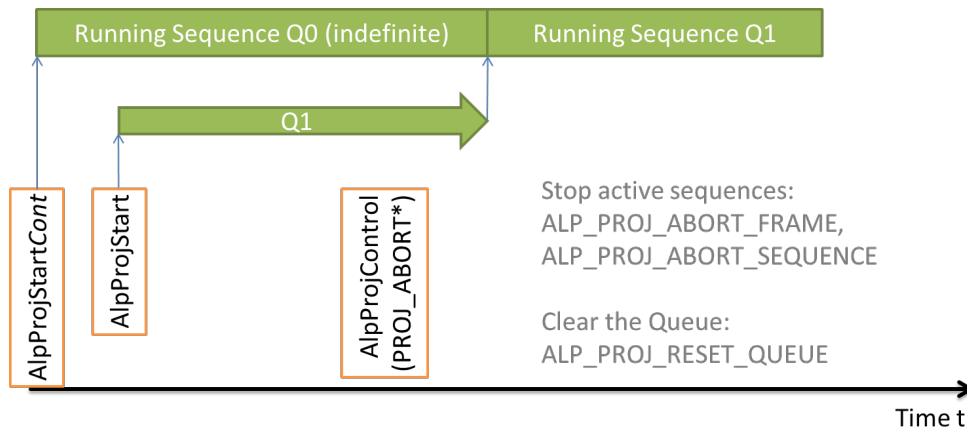
The table below shows Sequence Queue related ALP API functions together with their *ControlTypes* and *Values*.

Function (Control/InquireType)	Description
<i>AlpProjControl</i> (ALP_PROJ_QUEUE_MODE), <i>AlpProjInquire</i> (ALP_PROJ_QUEUE_MODE)	Switch between ALP_PROJ_LEGACY (default) and ALP_PROJ_SEQUENCE_QUEUE mode. This is only allowed when the ALP is idle (no sequence active).
<i>AlpProjInquire</i> (ALP_PROJ_QUEUE_AVAIL)	Inquire how much space is free in the queue.
<i>AlpProjInquire</i> (ALP_PROJ_QUEUE_MAX_AVAIL)	Inquire the whole size of the Sequence Queue.

<i>AlpProjStart</i> (called in Queue Mode)	Activate a sequence for finite iterations. This sequence will run until its regular end according to its ALP_SEQ_REPEAT, or until aborted. If no sequence is currently running, then the new sequence starts immediately. Else it is enqueued and waits. This function can return ALP_MEMORY_FULL if there is no space available in the queue.
<i>AlpProjStartCont</i>	Similar to <i>AlpProjStart</i> , but activate the sequence for indefinite iterations. Once started, it will run until aborted.
<i>AlpProjInquire</i> (ALP_PROJ_QUEUE_ID)	Provide the <i>QueueID</i> (ALP_ID) of the most recently enqueued sequence (or ALP_INVALID_ID, if <i>AlpProjStart[Cont]</i> has not yet been called since <i>AlpDevAlloc</i> )
<i>AlpProjControl</i> (ALP_PROJ_ABORT_SEQUENCE)	Abort an active sequence at the end of a repetition (after ALP_LASTFRAME). <i>ControlValue</i> can be ALP_DEFAULT or a <i>QueueID</i> . See below.
<i>AlpProjControl</i> (ALP_PROJ_ABORT_FRAME)	Abort an active sequence after the next frame (not necessarily ALP_LASTFRAME). See below.
<i>AlpProjControl</i> (ALP_PROJ_RESET_QUEUE)	Remove all waiting sequences from the queue. The currently running sequence is not affected. <i>ControlValue</i> must be ALP_DEFAULT.
<i>AlpProjHalt</i>	Same as ALP_PROJ_RESET_QUEUE followed by ALP_PROJ_ABORT_SEQUENCE.
<i>AlpProjWait</i>	Wait until device is idle. Note that in Sequence Queue Mode it is allowed to concurrently activate or abort sequences. See below for details.
<i>AlpProjInquireEx</i> (ALP_PROJ_PROGRESS)	Inquire detailed progress information of the running sequence and the queue. See below.

### Abort and Reset: Influence Active Sequences

There are situations when a premature abort of an active sequence is desirable. For example, in Sequence Queue Mode, a continuous sequence can be followed by other sequences. These sequences would wait forever, because the first one has no regular end.



Two different abort requests are available: `ALP_PROJ_ABORT_SEQUENCE` finishes the current iteration of a sequence and stops. The other one, `ALP_PROJ_ABORT_FRAME`, stops after the next frame.

In both cases, the sequence stops synchronously. A complete frame is displayed and if there is a waiting sequence then it is started according to its timing setup after completion of *PictureTime*. In contrast to that, *AlpDevHalt* would abort everything asynchronously. Frame display is interrupted and the DMD is cleared immediately, and all waiting sequences are removed.

There exists a race condition when the sequence to be aborted is not running continuously. It could happen that the sequence has already regularly finished, i.e. after `ALP_SEQ_REPEAT` iterations, before the abort request actually arrives in the ALP device. In order to not accidentally abort the next sequence, the ALP API allows specifying the *QueueID* as *ControlValue*. If it is `ALP_DEFAULT`, then the currently running sequence is stopped, whichever it exactly is. If *ControlValue* is a valid *QueueID* then the abort request stays pending until the addressed active sequence runs.

This obviously means that not only the *running* sequence, but also a *waiting* sequence can be addressed to be aborted. Because there can be only one abort request pending at any given time, the *AlpProjControl* function returns `ALP_NOT_IDLE` if another sequence shall be aborted in the meantime.

Consequently it is not allowed to abort a sequence that is waiting after any continuous sequence, because this would prohibit aborting it and make the continuous sequence run infinitely. If this malicious behavior is attempted then *AlpProjControl* returns `ALP_PARM_INVALID` to avoid deadlocks.

The same race condition as mentioned above could invalidate a previously valid *QueueID*. The sequence could just have been regularly finished. The ALP API cannot determine this condition in all cases, so *AlpProjControl* returns `ALP_OK` when *ControlValue* is not a valid *QueueID*.

Besides stopping a running sequence, it could also be required to clear the queue. The *AlpProjControl ControlType* `ALP_PROJ_RESET_QUEUE` removes all waiting sequences from the queue without affecting the running sequence.

## Inquire Progress of Active Sequences

Once started ALP executes all sequences in the queue without any USB interaction. This concurrency of execution is a natural impact of the real-time requirements of ALP. But sometimes it is required to synchronize the program running in the computer with the ALP.

The most basic form of synchronization is waiting for a certain condition. The ALP API function *AlpProjWait* allows blocking while the ALP device is busy executing active sequences. In Sequence Queue Mode the queue can be operated concurrently by means of activating or aborting sequences, for example.

Warning: if waiting for a continuous sequence, without having another thread to abort it, then the thread dead-locks (waits endless). Because of that the ALP\_PROJ\_LEGACY mode forbids any calls to *AlpProjWait* while running a continuous sequence.

A more sophisticated synchronization method is supervision of sequence progress. The ALP API function *AlpProjInquireEx* with *InquireType* ALP\_PROJ\_PROGRESS returns detailed information to the user. Note that due to the nature of the inquiry over USB, the result is already “out-of-date” by maybe a few milliseconds. The following details are filled into a structure of type *tAlpProjProgress*:

**QueueID and SequenceID** of the running sequence – note that multiple active sequences (*AlpProjStart[Cont]*, ALP\_PROJ\_QUEUE\_ID) can be started from the same *SequenceID* (created by *AlpSeqAlloc*)

**nWaitingSequences** – fill level of the queue, same as ALP\_PROJ\_QUEUE\_MAX\_AVAIL-ALP\_PROJ\_QUEUE\_AVAIL

**nSequenceCounter** – Number of iterations left after the current one, according to ALP\_SEQ\_REPEAT. This counter starts at ALP\_SEQ\_REPEAT-1 and counts down to 0.

**nFrameCounter** – Number of frames left in the current sequence iteration. This counter starts at *nFramesPerSubSequence*-1 and counts down to 0 in each sequence repetition.

**nSequenceCounterUnderflow** – useful for continuous sequences. Sequences started by *AlpProjStartCont* have an initial Sequence Counter of  $2^{20}-1=1048575$ . It counts down, and restarts after *reaching* value 0. At this time the Underflow flag is set to denote that the Frame Counter is not reliable any more due to one or more underflows. The *nSequenceCounterUnderflow* starts at value 0 and then steps to the number of completed iterations before underflow ( $2^{20}$ ).

**PictureTime and nFramesPerSubSequence** – These values are returned for convenience. They shall simplify the estimation of the time it still takes for the sequence to complete. Note that the sequence parameters may have already been changed (*AlpSeqControl*, *AlpSeqTiming*) since *AlpProjStart*. Hence the active sequence has settings which differ from values returned by *AlpSeqInquire*. Furthermore inquiry of *nFramesPerSubSequence* is much easier than determining it from the different settings it depends on (Scrolling settings, FrameLUT).

**Flags** summarize other values and reveal additional details. They are bit-wise combined, so a binary “and” must be used to determine which flags are set. The following four flags are available:

- ALP\_FLAG\_QUEUE\_IDLE: there are currently no active sequences
- ALP\_FLAG\_SEQUENCE\_INDEFINITE: the running sequence is started continuously
- ALP\_FLAG\_SEQUENCE\_ABORTING: the running sequence is about to be aborted
- ALP\_FLAG\_FRAME\_FINISHED: the last frame of a sequence has completed illumination, but *PictureTime* has not yet elapsed. This flag can only appear if there is no waiting sequence.

The ALP updates its counters after *IlluminateTime* of a frame. This could become noticeable if *PictureTime* is much longer than *IlluminateTime* (see also *AlpSeqTiming*).

### Timing Details

The section *AlpSeqTiming* defines some of the events within on *PictureTime* interval as well as relation between subsequent frames. This applies within one sequence, but it is also useful to understand the timing of the first frame of a sequence related to the very last frame of the previous one.

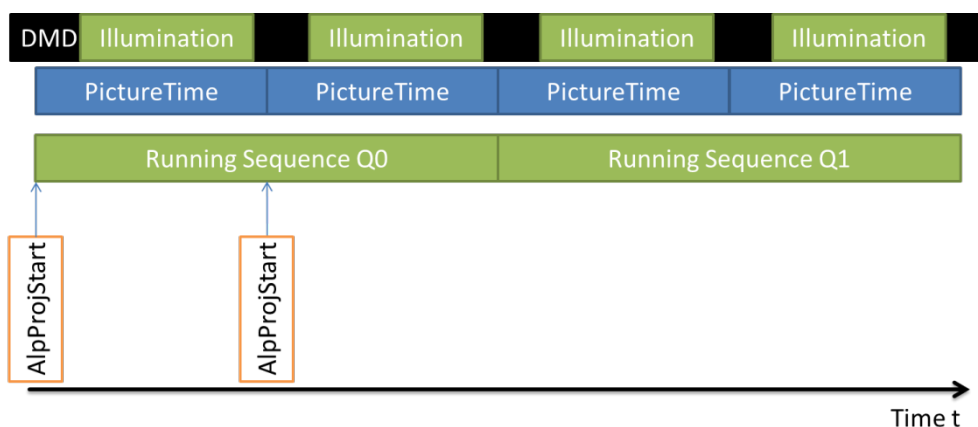
If both sequences have the same timing settings, and of course the next one is already waiting when the first one finishes, then no glitches should happen.

If timing changes, then two constraints apply:

The “first” *PictureTime* expires completely before the “next” *PictureTime* interval starts.

A break of at least  $\Delta t_1$  is required between both illuminations.

Especially if *SynchDelay* (ALP\_MASTER mode) or *TriggerInDelay* (ALP\_SLAVE mode) is reduced then the second condition could cause an extra break. Impacts of different settings can easily be understood using the pictures in the *AlpSeqTiming* section. For that purpose just join different *PictureTime* intervals together.



## Legacy Mode Behavior

ALP\_PROJ\_LEGACY mode takes care that there is never more than one waiting sequence. Each time a sequence is activated, the previously waiting sequence is removed. Of course this only happens when one sequence is running, making another one wait.

Additionally this mode makes the active sequence start running within a finite time. If the current sequence runs an infinite number of iterations (started with *AlpProjStartCont*), then the API aborts it synchronously after the last frame of the current iteration.

Formally, *AlpProjStart* or *AlpProjStartCont* execute these commands in legacy mode:

1. ALP\_PROJ\_RESET\_QUEUE
2. Enqueue the sequence
3. Query queue state, and if the running sequence is continuous (*AlpProjStartCont*):  
ALP\_PROJ\_ABORT\_SEQUENCE

In addition to these changes, the API also adjusts the *AlpProjWait* behavior. If the running or waiting sequence is continuous (*AlpProjStartCont*), then *AlpProjWait* rejects the call and returns ALP\_PARM\_INVALID. This avoids application dead-locks by waiting infinitely long.

## 3.4 Gated Frame Synchronization Output

This ALP API extension handles additional features regarding management of multi-purpose pins. It also targets towards multi-color operation by the means of multiple synchronization outputs.

This ALP API adds 3 new logical signals: SYNCH\_OUT1, SYNCH\_OUT2, and SYNCH\_OUT3. Each one of them can be configured to selectively output certain pulses of the SYNCH signal. This means that they share the same timing as the frame synchronization output (*SynchDelay*, *SynchPulseWidth*, see *AlpSeqTiming*), but stay inactive during frames for which they are deselected.

The function

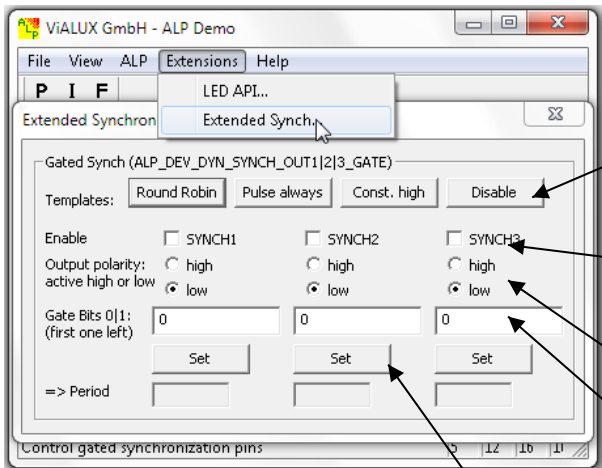
*AlpDevControlEx* has an input structure of type *tAlpDynSynchOutGate* and it supports control codes ALP\_DEV\_DYN\_SYNCH\_OUT1\_GATE to ALP\_DEV\_DYN\_SYNCH\_OUT3\_GATE. It configures the output based on circulating frame counters. Each of the output ports has one counter. The counters start counting with 0 at the first frame of a sequence after *AlpProjStart[Cont]*. After a counter reaches its maximum value (*Period-1*), it restarts at 0.

The indexes of the array *Gate* control the output at each state of the counter. Valid values are 0 for “stay inactive” and 1 meaning “output pulse”.

Default for all 3 synch outputs is *Period*=0. On ALP-4.1 this means to tristate the pin, on ALP-4.2: drive the inactive value (=“not *Polarity*”).



## Interactive Demo



Set up certain common scenarios (only dialogue elements!) Push the “Set” buttons afterwards!

Enable not checked:  
tAlpDynSynchOutGate::Period=0.

Enable checked:  
tAlpDynSynchOutGate::Period

tAlpDynSynchOutGate::Polarity

tAlpDynSynchOutGate::Gate[0..n]

Set: Call AlpDevControlEx now.

(new settings apply only to new started sequences; if the ALP is idle then pin polarity changes immediately)

The ALP Demo serves as a good starting-point for understanding the behavior of this API extension. But it is highly recommended to develop a custom application using the API.

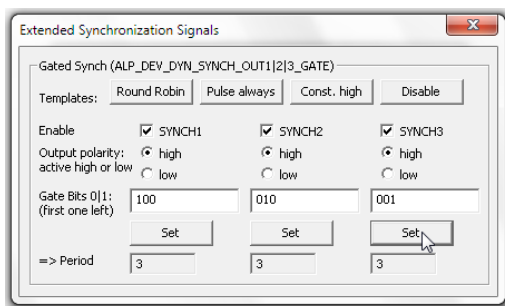


**Example 1: Round Robin**

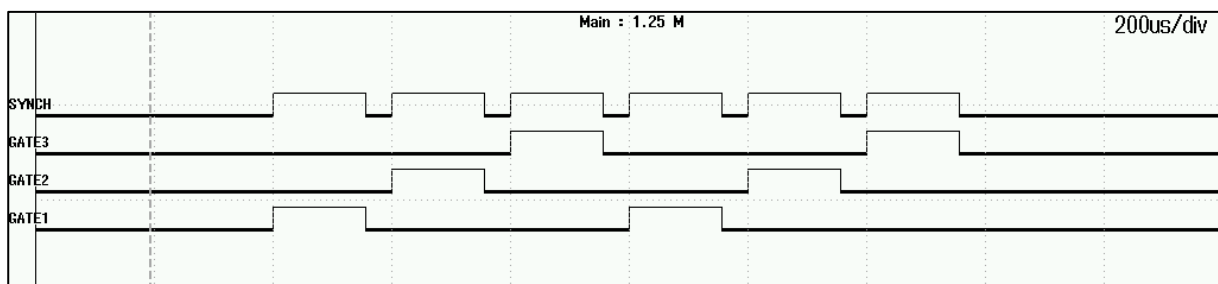
A typical use case for dynamic gates is periodically enabling different light sources. This example pulses all three pins one after another and then restarts:

```
Pre-Condition: allocate device, allocate sequence, and download image
data
// Set up the Gate for SYNCH_OUT1|2|3
tAlpDynSynchOutGate Gate;
ZeroMemory( &Gate, 18 ); // 18 = sizeof(tAlpDynSynchOutGate)
Gate.Period = 3;
Gate.Polarity = 1;
Gate.Gate[0] = 1; // the other "Gates" have already been initialized
to 0
AlpDevControlEx ( DeviceId, ALP_DEV_DYN_SYNCH_OUT1_GATE, &Gate );
// Period and Polarity stays the same, update Gate setting for second
port:
Gate.Gate[0] = 0; Gate.Gate[1] = 1;
AlpDevControlEx ( DeviceId, ALP_DEV_DYN_SYNCH_OUT2_GATE, &Gate );
// Update Gate setting for third port:
Gate.Gate[1] = 0; Gate.Gate[2] = 1;
AlpDevControlEx ( DeviceId, ALP_DEV_DYN_SYNCH_OUT3_GATE, &Gate );
// Start
AlpProjStart( DeviceId, SequenceId );
```

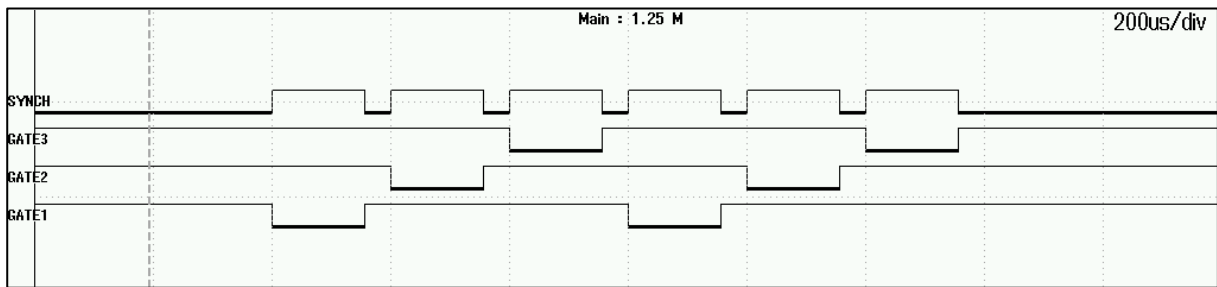
This source code complies with the following ALP Demo set up:



Given a sequence with *PictureTime*=200  $\mu$ s and 6 Frames to be displayed, the synchronization ports show the pulses as in the picture below. The frame synch output polarity is active-high.

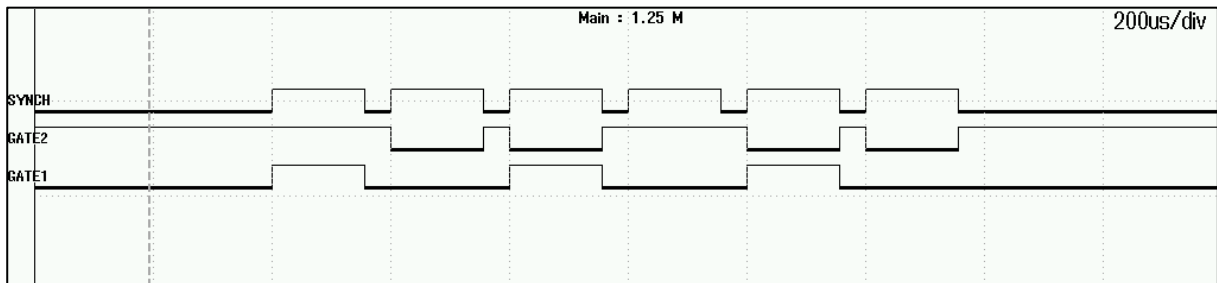
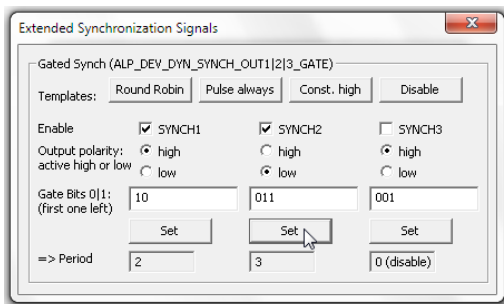


If the light-sources are enabled by an active-low signal, then simply change the source code above to `Gate.Polarity=0`. This results in signals shown in the next picture:



## Example 2

All settings can be applied individually for all three synch outputs. This example uses Period=2, Gates=(1,0), Polarity=high for SYNCH\_OUT1 and Period=3, Gates=(0,1,1), Polarity=low for SYNCH\_OUT2:



## Pin assignment

For now, Multi-Purpose IO connector pins are allocated to logical signals according to the fixed table below:

	ALP-4.1	ALP-4.2
SYNCH_OUT1	Pin 2 (can be tri-stated, has a weak pull-up <sup>3</sup> ) Initially high-Z	Pin 2 (no tri-state <sup>4</sup> ) Initially constant low
SYNCH_OUT2	Pin 3 (can be tri-stated, has a weak pull-up) Initially high-Z	Pin 3 (no tri-state) Initially constant low
SYNCH_OUT3	Pin 4 (can be tri-stated, has a weak pull-up) Initially high-Z	Pin 4 (no tri-state) Initially constant low

## 3.5 PWM Output

There are use cases when an analog signal is required, for example for controlling a light-source. Even though all GPIO pins are digital, the ALP supports this by means of pulse-width modulation.

Use *AlpDevControl* with *ControlType* ALP\_PWM\_LEVEL in order to set up the duty-cycle (in percent).

The PWM signal is connected to the Multi-Purpose IO connector. Please refer to the hardware specific documents for electrical parameters (I/O Standard).

	ALP-4.1	ALP-4.2
PWM Pin Number	Pin 9	Pin 9
Value after reset by <i>AlpDevAlloc</i>	0 % (constant low)	
Value after <i>AlpDevFree</i>	0 % (constant low) (Former software versions: last ALP_PWM_LEVEL value.)	
Pulse Period	64 $\mu$ s	
Accuracy	$\pm 2$ % absolute	

**Note:** Other device states are unspecified. Be aware that especially after power-up or when running ALP *basic*, the PWM pin can also be tri-stated (high-impedance).

---

<sup>3</sup> Weak Pull-Up: There is a weak internal pull-up resistor implemented for each pin. This avoids unknown levels when output pins are disabled (tri-state).

<sup>4</sup> No tri-state: The pin is not tri-stated when disabled. It drives “not polarity” in this case.

### 3.6 Externally Triggered Frame Transition

The ALP-4 API supports an additional trigger mode. It allows frame display with internal timing (i.e. master mode) with conditional frame transitions. ALP repeats the display of each frame of a sequence until the trigger event is detected, causing the transition to the next frame.

The number of frame transitions is not changed by this mode. Settings like ALP\_FIRSTFRAME, ALP\_LASTFRAME, and ALP\_SEQ\_REPEAT still apply.

The trigger input port is Pin 7 of the Multi-Purpose I/O (Synchronization) connector (see V4100 Technical Reference Manual). This is the same as frame trigger input in ALP\_SLAVE mode.

#### Related API Controls

The ALP\_PROJ\_STEP trigger mode is selected using *AlpProjControl* with *ControlType*=ALP\_PROJ\_STEP. The table below shows the meaning of different *ControlValues*.

<b>ControlValue of ALP_PROJ_STEP</b>	<b>Description</b>
ALP_DEFAULT	step forward after each displayed DMD frame
ALP_LEVEL_HIGH   LOW	step forward if and only if the trigger input is high / low
ALP_EDGE_RISING   FALLING	frame transition depends on a trigger edge

#### Interaction with other ALP settings

The ALP must be idle in order to switch ALP\_PROJ\_STEP trigger mode.

Use ALP\_PROJ\_STEP only with ALP\_PROJ\_MODE=ALP\_MASTER.

*Frame Lookup Table* (FrameLUT): ALP\_PROJ\_STEP applies, conditionally stepping forward through the Lookup Table.

*AlpProjHalt*: The currently running sequence runs until all frames are displayed. This requires trigger events according to the ALP\_PROJ\_STEP setting.

*Gated Frame Synchronization Outputs* are not affected by ALP\_PROJ\_STEP. Their index counter progresses for each frame that is displayed on the DMD, even if it has the same image data.

#### Timing (Latency)

Frame Transition happens between  $1 * PictureTime + SynchDelay$  and  $2 * PictureTime + SynchDelay$  after the trigger edge.

ALP\_LEVEL\_\* modes: hold the trigger constant at the start of SynchOut pulse  $\pm 1\mu s$

ALP\_EDGE\_\* modes: detected edges are stored internally; this memory is cleared each time when processing of the frame-transition starts, or when calling *AlpProjControl*(ALP\_PROJ\_STEP)

### 3.7 Flexible PWM Mode

The ALP-4 API supports an additional mode of displaying gray-scale sequences. In normal mode it assembles gray-scale values from bit planes using fixed weights. In ALP\_FLEX\_PWM mode, bit plane timing is controlled by a trigger input.

#### Related API Controls

A sequence must be allocated (*AlpSeqAlloc*) with the required number of *BitPlanes*. Then *AlpSeqControl* selects the ALP\_FLEX\_PWM mode. This implicitly switches the sequence to binary uninterrupted mode with fastest possible timing and zero *TriggerInDelay*.

Display order is: most-significant bit plane first.

*AlpProjControl* must be used to enable ALP\_SLAVE mode. This way active trigger edges control the bit plane timing.

The new *ControlType* of *AlpSeqControl* is ALP\_PWM\_MODE. It can be used with *ControlValues* ALP\_DEFAULT or ALP\_FLEX\_PWM.

ControlValue of ALP_PWM_MODE	Description
ALP_DEFAULT	Generate gray-scale display with internal timing using fixed bit-plane weights
ALP_FLEX_PWM	Process trigger edges to for bit-plane processing in order to implement custom weights

#### Interaction with other ALP settings

- Trigger edges and Synchronization pulses apply to bit planes rather than full gray-scale frames. A sequence of n Frames in normal mode displays n\**BitNum* binary frames in ALP\_FLEX\_PWM mode.
- Use ALP\_FLEX\_PWM only with ALP\_PROJ\_MODE=ALP\_SLAVE.
- ALP\_BIN\_MODE is not available when a sequence is in ALP\_FLEX\_PWM mode. It is fixed ALP\_BIN\_UNINTERRUPTED.
- ALP\_DEV\_DYN\_SYNCH\_OUTx\_GATE settings apply to bit-planes rather than gray-scale frames
- ALP\_BITNUM can be used to reduce the number of bit planes. No additional *AlpSeqTiming* call is required in ALP\_FLEX\_PWM mode.
- Scrolling and ALP\_FLUT\_MODE can be combined with ALP\_FLEX\_PWM mode
- *AlpSeqTiming* can be used for adjusting *TriggerInDelay* and others. It is recommended to use the minimum possible *PictureTime*. It can be inquired using *AlpSeqInquire*(ALP\_MIN\_PICTURE\_TIME)
- When leaving ALP\_FLEX\_PWM mode (set ALP\_PWM\_MODE=ALP\_DEFAULT), an additional *AlpSeqTiming* call is necessary to make the change effective.

## 4 LED Control

### 4.1 Introduction to the ALP LED API

The ALP-4 API contains features for controlling the ViALUX high-power LED driver (HLD). This extension consists of API functions exported by the DLLs (AlpLed...), as well as control or inquire types and data structures.

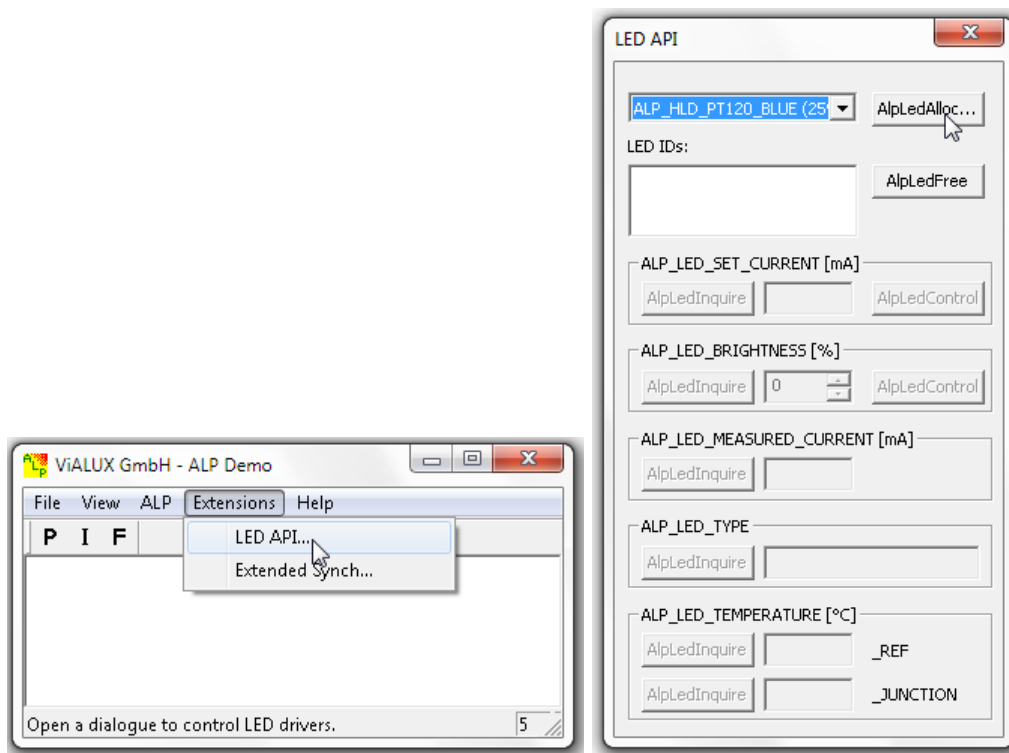
The HLD must be connected to the I2C bus of the ALP device. Please refer to the “HLD-Connections” and to the “ALP-4.1 Connections” document or “V4100 Technical Reference” (ALP-4.2).

Users can manage even multiple LED drivers connected to a single ALP, set up the light intensity by means of the electrical current, and supervise the LED temperature. Please also consider the Gated Frame Synchronization Outputs for switching different LEDs on a frame-by-frame basis.

For convenience purpose an approach is implemented using a *Nominal Current* value and a relative *Brightness* value. The nominal value is represented in milliamps (mA) and brightness in percent. The HLD drives the  $Nominal\ Current * Brightness / 100$ .

Many of the supported LEDs contain an on-chip temperature sensor. Even though this sensor is placed next to the thermal source (LED junction), there is a temperature difference. The API estimates the actual junction temperature based on a model of the LED type and the measured value.

The following sections explain the ALP LED API functions. The ALP Demo (AlpDemo.exe) allows to interactively use these functions. This might be helpful for getting to know how to use them.



*Note:* Even custom light sources could be software-controlled by the PWM output pin of ALP. Please see PWM Output above.

## 4.2 AlpLedAlloc

### Format

```
long AlpLedAlloc (ALP_ID DeviceId, long LedType, void *UserVarPtr, ALP_ID *LedIdPtr )
```

### Description

AlpLedAlloc initializes and allocates a LED driver of the given type. It is addressed by its identifier *LedId* in subsequent ALP LED API calls.

### Parameters

<i>DeviceId</i>	ALP device identifier
<i>LedType</i>	one of ALP_HLD_PT120_RED, ALP_HLD_PT120_GREEN, ALP_HLD_PT120_BLUE, ALP_HLD_PT120_390, or ALP_HLD_PT120_405 (see table below)
<i>UserVarPtr</i>	NULL or a pointer to structured data; the structure depends on <i>LedType</i> (see “Setting structures”).

*LedIdPtr* address of a variable in which the ALP API stores the LED identifier; this number identifies the LED in subsequent API calls; it will be deleted by *AlpLedFree*

### Compatible hardware

LedType	Compatible Hardware
ALP_HLD_PT120_RAX (2016 <sup>5</sup> )	PT-120-RAX-L15
ALP_HLD_PT120_RED	PT-120-R-C11-MPB, PT-121-R-C11-MPB (obsolete since 2016, replaced by RAX type)
ALP_HLD_PT120_GREEN	PT-120-G-C11-MPB, PT-121-G-C11-MPB
ALP_HLD_PT120TE_BLUE	PT-121-B-L11
ALP_HLD_PT120_BLUE	PT-120-B-C11-EPA, PT-121-B-C11-EPA (obsolete, devices shipped since 2013 have the thermally enhanced TE package)
ALP_HLD_CBT90_WHITE	CBT-90-W*-C11
ALP_HLD_CBT140_WHITE	CBT-140-W*
ALP_HLD_CBT90_UV	CBT-90-UV
ALP_HLD_CBM120_UV365 (2016)	CBM-120-UV with maximum continuous current of 12A (wave length range between 365nm and 375nm)
ALP_HLD_CBM120_UV (2016)	CBM-120-UV with maximum continuous current of 18A (wave lengths 380nm and above)
ALP_HLD_CBT120_UV	CBT-120-UV

### Setting structures

#### tAlpHldAllocParams

The *LedTypes* ALP\_HLD\_\* use the structure tAlpHldAllocParams for *UserVarPtr*:

```
struct tAlpHldAllocParams {
    long I2cDacAddr;
    long I2cAdcAddr;
};
```

For HLD, the structure contains only bus addresses (I2C bus) of the LED driver. The structure can be omitted by passing a NULL pointer for *UserVarPtr*. In this case the ALP software scans the bus for a certain set of known addresses. The bus addresses are hard-

<sup>5</sup> New in 2016; Old ALP API DLL do not yet support these LED types and return ALP\_PARM\_INVALID.



wired on the HLD, so the order of returned devices is stable even without selecting certain addresses. It only varies when changing the set of HLD's that are connected to the ALP.

Valid settings for (I2cDacAddr, I2cAdcAddr) are (24, 64), (26, 66), (28, 68), and (30, 70).

### Return values

The function can return one of the standard ALP API return codes. Please consider the additional hints below:

- ALP\_PARM\_INVALID is the result of an unsupported *LedType*
- ALP\_ERROR\_INIT: invalid *AllocParams*, or error initializing the LED driver
- ALP\_NOT\_ONLINE: when *UserStructPtr* = NULL and none of the known addresses works
- ALP\_NOT\_READY: one of the requested I2C addresses has already been allocated (e.g. a *LedId* for this device exists already)

## 4.3 AlpLedFree

### Format

long AlpLedFree(ALP\_ID DeviceId, ALP\_ID LedId)

### Description

The LED is switched off and the software object is released. *LedId* becomes unusable.

### Parameters

<i>DeviceId</i>	ALP device identifier
<i>LedId</i>	LED identifier

## 4.4 AlpLedControl

### Format

long AlpLedControl(ALP\_ID DeviceId, ALP\_ID LedId, long ControlType, long Value)

### Description

Adjust the LED.

AlpLedControl allows to individually modify the nominal current (ALP\_LED\_SET\_CURRENT) and the brightness (ALP\_LED\_BRIGHTNESS). This allows to easily use a percentage scale. Since the HLD's drive strength is adjusted immediately to the according product of both values, consider to dim the brightness before increasing the current.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>LedId</i>	LED identifier
<i>ControlType</i>	control parameter that is to be modified
<i>Value</i>	value of the parameter

The following settings are available:

ControlType	Value	Description
ALP_LED_SET_CURRENT	ALP_DEFAULT (0)	Because the symbol ALP_DEFAULT has value 0, the LED is switched off. The initial value is only restored after <i>AlpLedAlloc</i> . Note: In former versions of the ALP API, ALP_DEFAULT had restored the drive current as specified for continuous operation for the given LED type.
	> 0	The value is interpreted as milliamps. The LED driver drives a current of $\text{Value} * \text{ALP\_LED\_BRIGHTNESS} / 100\%$ .
ALP_LED_BRIGHTNESS	0	The LED is switched off. This is the default value after initialization.
	> 0	The value is interpreted as percentage. The LED driver drives a current of $\text{ALP\_LED\_SET\_CURRENT} * \text{Value} / 100\%$ . Value is allowed to become >100%.
ALP_LED_FORCE_OFF	There may be a small LED current remaining even after the HLD is set to 0 A. Use the enable-signal to guarantee that the LED stops emitting any light. Drawback of this approach is that re-starting the LED takes several milliseconds instead of microseconds. If speed is more crucial than residual light, then please set this parameter to ALP_LED_ON.	
	ALP_LED_AUTO_OFF	(ALP_DEFAULT): The ALP LED API disables the LED if $\text{ALP\_LED\_SET\_CURRENT} * \text{ALP\_LED\_BRIGHTNESS} / 100\% = 0$
	ALP_LED_OFF	The LED is a disabled, independent of the brightness and current setting
	ALP_LED_ON	The LED stays enabled. Some residual light could be emitted even for current or brightness set to 0.

**Return values**

ALP\_NOT\_AVAILABLE: invalid DeviceId

ALP\_PARM\_INVALID: Value out of range ( $\text{current} * \text{brightness} / 100\%$  exceeds the capabilities of the LED type), invalid ControlType, or invalid LedId

ALP\_ERROR\_COMM: USB communication error or I2C bus error

## 4.5 AlpLedInquire

### Format

long AlpLedInquire(ALP\_ID *DeviceId*, ALP\_ID *LedId*, long *InquireType*, long *\*UserVarPtr*)

### Description

This function measures a value or inquires a parameter setting.

The LED temperature (ALP\_LED\_TEMPERATURE\_JUNCTION) is calculated from the measured value (ALP\_LED\_TEMPERATURE\_REF) using a thermal model for the LED type. The thermal model depends on the currently driven LED current, so bear in mind that it might be inaccurate short after changing the drive current.

### Parameters

*DeviceId* ALP device identifier

*LedId* LED identifier

*InquireType* specifies the ALP device parameter setting to inquire; See the table below.

*UserVarPtr* specifies the address of the variable in which the requested information is to be written. The variable must be of type long.

The *InquireType* supports the following values:

InquireType	Description
ALP_LED_TYPE	The <i>LedType</i> value used in AlpLedAlloc.
ALP_LED_SET_CURRENT	Milliamps: Nominal current.
ALP_LED_BRIGHTNESS	Percentage.
ALP_LED_FORCE_OFF	Additional method to disable residual LED current.
ALP_LED_MEASURED_CURRENT	Milliamps. The HLD measures the current it drives.
ALP_LED_TEMPERATURE_REF	1/256 °C: Measured temperature on the LED chip, near the LED junction.
ALP_LED_TEMPERATURE_JUNCTION	1/256 °C: Calculated temperature at the LED junction.

## 4.6 AlpLedControlEx

### Format

long AlpLedControlEx(ALP\_ID *DeviceId*, ALP\_ID *LedId*, long *ControlType*, void *\*UserStructPtr*)

**Description**

This function is similar to `AlpLedControl`. However, it changes settings that do not fit into a scalar (long) value.

We have not yet defined any *ControlTypes* for `AlpLedControlEx`. It is prepared for future use.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>LedId</i>	LED identifier
<i>ControlType</i>	control parameter that is to be modified
<i>UserStructPtr</i>	pointer to structured data; structure depends on <i>ControlType</i>

**4.7 AlpLedInquireEx****Format**

```
long AlpLedInquireEx(ALP_ID DeviceId, ALP_ID LedId, long InquireType, void
*UserStructPtr)
```

**Description**

This function is similar to `AlpLedInquire`. However, it returns data that do not fit into a scalar (long) value.

**Parameters**

<i>DeviceId</i>	ALP device identifier
<i>LedId</i>	LED identifier
<i>InquireType</i>	control parameter that is to be inquired
<i>UserStructPtr</i>	pointer to structured data; structure depends on <i>InquireType</i>

**InquireTypes and Data Structures****ALP\_LED\_ALLOC\_PARAMS**

`UserStructPtr` points to a data structure as in `AlpLedAlloc`, depending on `LedType`.

If the device has been automatically selected then this function can be used to inquire the actual set-up values.

**5 Data types, Functions, Constants**

The prototypes of all exported DLL functions are declared in the header file `alp.h`. This file also contains the values of symbolic constants like control types, return values etc.

Most C/C++ programmers will only require to #include “alp.h”. When using other programming languages, then please use the next sections as a quick reference.

## 5.1 Data types

The ALP API uses these data types, with 1 byte being 8 bits:

- Long (4-byte signed integer)
- ALP\_ID (4-byte unsigned integer)
- Char (1-byte integer)
- Void (wild card – the actual type is defined by another setting or parameter value)

Data structures additional to standard ALP API data types:

Type (size in bytes), Function	Member data type (size)	Member (byte offset)
tAlpHldPt120AllocParams (8), AlpLedAlloc, AlpLedInquireEx	long (4)	I2cDacAddr (0)
	long (4)	I2cAdcAddr (4)
tAlpDynSynchOutGate (18), AlpDevControlEx	unsigned char (1)	Period (0)
	unsigned char (1)	Polarity (1)
	unsigned char array (16)	Gate (2)
tFlutWrite (up to 8+ 4*ALP_FLUT_MAX_ENTRIES9), AlpProjControlEx	long (4)	nOffset (0)
	long (4)	nSize (4)
	long array (4*nSize)	FrameNumbers (8)
tAlpProjProgress (36), AlpProjInquireEx	ALP_ID (4)	CurrentQueueId (0)
	ALP_ID (4)	SequenceId (4)
	unsigned long (4)	nWaitingSequences (8)
	unsigned long (4)	nSequenceCounter (12)
	unsigned long (4)	nSequenceCounterUnderflow (16)
	unsigned long (4)	nFrameCounter (20)
	unsigned long (4)	nPictureTime (24)
	unsigned long (4)	nFramesPerSubSequence (28)
	unsigned long (4)	nFlags (32)

## 5.2 List of Functions

The ALP DLL is available in different versions. They differ in calling convention (\_cdecl, \_stdcall) and target CPU (32-bit, 64-bit).

The exported function names are not decorated in order to achieve portability.

All functions return a 4-byte integer value. Pointer sizes depend on the target CPU: 4-byte pointers for 32-bit DLLs and 8-byte pointers for the 64-bit DLL. The following functions are available:

Function	Parameters
AlpDevAlloc	DeviceNum: 4-byte integer, InitFlag: 4-byte integer, DeviceIdPtr: pointer to a writable 4-byte integer
AlpDevControl	DeviceId: 4-byte integer, ControlType: 4-byte integer, ControlValue: 4-byte integer
AlpDevInquire	DeviceId: 4-byte integer, InquireType: 4-byte integer, UserVarPtr: pointer to a writable 4-byte integer
AlpDevControlEx	DeviceId: 4-byte integer, ControlType: 4-byte integer, UserStructPtr: pointer to a read-able structure according to ControlType
AlpDevHalt	DeviceId: 4-byte integer
AlpDevFree	DeviceId: 4-byte integer
AlpSeqAlloc	DeviceId: 4-byte integer, BitPlanes: 4-byte integer, PicNum: 4-byte integer, SequencIdPtr: pointer to a writable 4-byte integer
AlpSeqControl	DeviceId: 4-byte integer, SequencId: 4-byte integer, ControlType: 4-byte integer, ControlValue: 4-byte integer
AlpSeqTiming	DeviceId: 4-byte integer, SequencId: 4-byte integer, IlluminateTime: 4-byte integer, PictureTime: 4-byte integer, SynchDelay: 4-byte integer, SynchPulseWidth: 4-byte integer, TriggerInDelay: 4-byte integer
AlpSeqInquire	DeviceId: 4-byte integer, SequencId: 4-byte integer, InquireType: 4-byte integer, UserVarPtr: pointer to a writable 4-byte integer

Function	Parameters
AlpSeqPut	DeviceId: 4-byte integer, SequenceId: 4-byte integer, PicOffset: 4-byte integer, PicLoad: 4-byte integer, UserArrayPtr: pointer to a readable image data buffer; see also AlpSeqPut
AlpSeqFree	DeviceId: 4-byte integer, SequenceId: 4-byte integer
AlpProjControl	DeviceId: 4-byte integer, ControlType: 4-byte integer, ControlValue: 4-byte integer
AlpProjInquire	DeviceId: 4-byte integer, InquireType: 4-byte integer, UserVarPtr: pointer to a writable 4-byte integer
AlpProjControlEx	DeviceId: 4-byte integer, ControlType: 4-byte integer, UserStructPtr: pointer to a read-able structure according to ControlType
AlpProjInquireEx	DeviceId: 4-byte integer, InquireType: 4-byte integer, UserStructPtr: pointer to a write-able structure according to InquireType
AlpProjStart	DeviceId: 4-byte integer, SequenceId: 4-byte integer
AlpProjStartCont	DeviceId: 4-byte integer, SequenceId: 4-byte integer
AlpProjHalt	DeviceId: 4-byte integer
AlpProjWait	DeviceId: 4-byte integer
AlpLedAlloc	DeviceId: 4-byte integer, LedType: 4-byte integer, UserStructPtr: pointer to a readable structure according to LedType LedIdPtr: pointer to a writable 4-byte integer
AlpLedFree	DeviceId: 4-byte integer, LedId: 4-byte integer,
AlpLedControl	DeviceId: 4-byte integer, LedId: 4-byte integer, ControlType: 4-byte integer, ControlValue: 4-byte integer
AlpLedInquire	DeviceId: 4-byte integer, LedId: 4-byte integer, InquireType: 4-byte integer, UserVarPtr: pointer to a writable 4-byte integer

Function	Parameters
AlpLedControlEx	DeviceId: 4-byte integer, LedId: 4-byte integer, ControlType: 4-byte integer, UserStructPtr: pointer to a readable structure according to ControlType
AlpLedInquireEx	DeviceId: 4-byte integer, LedId: 4-byte integer, InquireType: 4-byte integer, UserStructPtr: pointer to a writable structure according to InquireType

### 5.3 Constant values

#### Special values

ALP\_DEFAULT=0

ALP\_INVALID\_ID=ULONG\_MAX (=  $2^{32}-1$  = 4294967295)

#### Return values

- ALP\_OK = 0
- ALP\_NOT\_ONLINE = 1001
- ALP\_NOT\_IDLE = 1002
- ALP\_NOT\_AVAILABLE = 1003
- ALP\_NOT\_READY = 1004
- ALP\_PARM\_INVALID = 1005
- ALP\_ADDR\_INVALID = 1006
- ALP\_MEMORY\_FULL = 1007
- ALP\_SEQ\_IN\_USE = 1008
- ALP\_HALTED = 1009
- ALP\_ERROR\_INIT = 1010
- ALP\_ERROR\_COMM = 1011
- ALP\_DEVICE\_REMOVED = 1012
- ALP\_NOT\_CONFIGURED = 1013
- ALP\_LOADER\_VERSION = 1014
- ALP\_ERROR\_POWER\_DOWN = 1018
- ALP\_DRIVER\_VERSION = 1019
- ALP\_SDRAM\_INIT = 1020



**Device Inquire and Control Types (AlpDevControl, AlpDevInquire)**

- ALP\_DEVICE\_NUMBER = 2000
- ALP\_VERSION = 2001
- ALP\_AVAIL\_MEMORY = 2003
- ALP\_SYNCH\_POLARITY = 2004
- ALP\_LEVEL\_HIGH = 2006
- ALP\_LEVEL\_LOW = 2007
- ALP\_TRIGGER\_EDGE = 2005
- ALP\_EDGE\_FALLING = 2008
- ALP\_EDGE\_RISING = 2009
- ALP\_DEV\_DMDTYPE = 2021
- ALP\_DMDTYPE\_XGA = 1
- ALP\_DMDTYPE\_1080P\_095A = 3
- ALP\_DMDTYPE\_XGA\_07A = 4
- ALP\_DMDTYPE\_XGA\_055X = 6
- ALP\_DMDTYPE\_WUXGA\_096A = 7
- ALP\_DMDTYPE\_WXGA = 12
- ALP\_DMDTYPE\_DISCONNECT = 255
- ALP\_USB\_CONNECTION = 2016
- ALP\_DEV\_DYN\_SYNCH\_OUT1\_GATE = 2023
- ALP\_DEV\_DYN\_SYNCH\_OUT2\_GATE = 2024
- ALP\_DEV\_DYN\_SYNCH\_OUT3\_GATE = 2025
- ALP\_DDC\_FPGA\_TEMPERATURE = 2050
- ALP\_APPS\_FPGA\_TEMPERATURE = 2051
- ALP\_PCB\_TEMPERATURE = 2052
- ALP\_DEV\_DISPLAY\_HEIGHT = 2057
- ALP\_DEV\_DISPLAY\_WIDTH = 2058
- ALP\_PWM\_LEVEL = 2063
- ALP\_DEV\_DMD\_MODE = 2064
- ALP\_DMD\_RESUME = 0
- ALP\_DMD\_POWER\_FLOAT = 1

**Sequence Inquire and Control Types (AlpSeqControl, AlpSeqInquire)**

• ALP_BITPLANES	= 2200
• ALP_BITNUM	= 2103
• ALP_BIN_MODE	= 2104
• ALP_BIN_NORMAL	= 2105
• ALP_BIN_UNINTERRUPTED	= 2106
• ALP_PICNUM	= 2201
• ALP_FIRSTFRAME	= 2101
• ALP_LASTFRAME	= 2102
• ALP_FIRSTLINE	= 2111
• ALP_LASTLINE	= 2112
• ALP_LINE_INC	= 2113
• ALP_SCROLL_FROM_ROW	= 2123
• ALP_SCROLL_TO_ROW	= 2124
• ALP_SEQ_REPEAT	= 2100
• ALP_PICTURE_TIME	= 2203
• ALP_MIN_PICTURE_TIME	= 2211
• ALP_MAX_PICTURE_TIME	= 2213
• ALP_ILLUMINATE_TIME	= 2204
• ALP_MIN_ILLUMINATE_TIME	= 2212
• ALP_ON_TIME	= 2214
• ALP_OFF_TIME	= 2215
• ALP_SYNCH_DELAY	= 2205
• ALP_MAX_SYNCH_DELAY	= 2209
• ALP_SYNCH_PULSEWIDTH	= 2206
• ALP_TRIGGER_IN_DELAY	= 2207
• ALP_MAX_TRIGGER_IN_DELAY	= 2210
• ALP_DATA_FORMAT	= 2110
• ALP_DATA_MSB_ALIGN	= 0
• ALP_DATA_LSB_ALIGN	= 1
• ALP_DATA_BINARY_TOPDOWN	= 2
• ALP_DATA_BINARY_BOTTOMUP	= 3
• ALP_SEQ_PUT_LOCK	= 2119
• ALP_FLUT_MODE	= 2118

- ALP\_FLUT\_NONE = 0
- ALP\_FLUT\_9BIT = 1
- ALP\_FLUT\_18BIT = 2
- ALP\_FLUT\_ENTRIES9 = 2120
- ALP\_FLUT\_OFFSET9 = 2122
- ALP\_PWM\_MODE = 2107
- ALP\_FLEX\_PWM = 3

#### **Projection Inquire and Control Types (AlpProjControl[Ex], AlpProjInquire[Ex])**

- ALP\_PROJ\_MODE = 2300
- ALP\_MASTER = 2301
- ALP\_SLAVE = 2302
- ALP\_PROJ\_STEP = 2329
- ALP\_PROJ\_STATE = 2400
- ALP\_PROJ\_ACTIVE = 1200
- ALP\_PROJ\_IDLE = 1201
- ALP\_PROJ\_INVERSION = 2306
- ALP\_PROJ\_UPSIDE\_DOWN = 2307
- ALP\_PROJ\_QUEUE\_MODE = 2314
- ALP\_PROJ\_LEGACY = 0
- ALP\_PROJ\_SEQUENCE\_QUEUE = 1
- ALP\_PROJ\_QUEUE\_ID = 2315
- ALP\_PROJ\_QUEUE\_MAX\_AVAIL = 2316
- ALP\_PROJ\_QUEUE\_AVAIL = 2317
- ALP\_PROJ\_PROGRESS = 2318
- ALP\_FLAG\_QUEUE\_IDLE = 1
- ALP\_FLAG\_SEQUENCE\_ABORTING = 2
- ALP\_FLAG\_SEQUENCE\_INDEFINITE = 4
- ALP\_FLAG\_FRAME\_FINISHED = 8
- ALP\_PROJ\_RESET\_QUEUE = 2319
- ALP\_PROJ\_ABORT\_SEQUENCE = 2320
- ALP\_PROJ\_ABORT\_FRAME = 2321
- ALP\_PROJ\_WAIT\_UNTIL = 2323
- ALP\_PROJ\_WAIT\_PIC\_TIME = 0

- ALP\_PROJ\_WAIT\_ILLU\_TIME = 1
- ALP\_FLUT\_MAX\_ENTRIES9 = 2324
- ALP\_FLUT\_WRITE\_9BIT = 2325
- ALP\_FLUT\_WRITE\_18BIT = 2326

#### **LED Types**

- ALP\_HLD\_PT120\_RAX = 268
- ALP\_HLD\_PT120\_RED = 257
- ALP\_HLD\_PT120\_GREEN = 258
- ALP\_HLD\_PT120TE\_BLUE = 263
- ALP\_HLD\_PT120\_BLUE = 259
- ALP\_HLD\_CBT90\_UV = 265
- ALP\_HLD\_CBM120\_UV365 = 266
- ALP\_HLD\_CBM120\_UV = 267
- ALP\_HLD\_CBT120\_UV = 260
- ALP\_HLD\_CBT90\_WHITE = 262
- ALP\_HLD\_CBT140\_WHITE = 264

#### **LED Inquire and Control Types (AlpLedControl, AlpLedInquire)**

- ALP\_LED\_SET\_CURRENT = 1001
- ALP\_LED\_BRIGHTNESS = 1002
- ALP\_LED\_FORCE\_OFF = 1003
- ALP\_LED\_AUTO\_OFF = 0
- ALP\_LED\_OFF = 1
- ALP\_LED\_ON = 2
- ALP\_LED\_TYPE = 1101
- ALP\_LED\_MEASURED\_CURRENT = 1102
- ALP\_LED\_TEMPERATURE\_REF = 1103
- ALP\_LED\_TEMPERATURE\_JUNCTION = 1104

#### **Extended LED Inquire and Control Types (AlpLedControlEx, AlpLedInquireEx)**

- ALP\_LED\_ALLOC\_PARAMS = 2101