Unigraf USB Capture Devices API

Reference Manual

14.December.2016

UNIGRAF

Copyright

Notice

The information in this manual has been verified on the date of issue. The authors reserve rights to make any changes to this product and revise the information without obligation to notify any person about such revisions or changes.

Edition

Title	UUCD Reference Manual
Document ID	**Error! Unknown document property name.**
Issue date	14.December.2016

Company information

Unigraf Oy

Piispantilankuja 4
FI-02240 ESPOO
Finland

Phone. +358 9 589 550

e-mail: info@unigraf.fi
web: http://www.unigraf.fi

Trademarks

Unigraf, UCD-1, UCD-2 and UUCD API are trademarks of Unigraf Oy

Table of Contents

1. GENERAL

1.1. About this document

This document describes Unigraf USB Capture Device API (UUCD API). UUCD API allows the user to configure and capture video data using Unigraf UCD-1 and UCD-2 capture devices.

This document applies to UUCD release 1.6 (UUCD.dll 1.6.31)

1.2. Acronyms and abbreviations

API	Application Programming Interface.
DLL	Dynamic Link Library.
GUI	Graphical User Interface.
UUCD	Unigraf USB Capture Devices API.
IDE	Integrated Development Environment
OS	Operating System

2. COMPONENTS AND FEATURES

UCD Capture devices are used to capture video from video sources, like LVDS, MiniLVDS and V-by-One (Vx1). The primary use of UCD class devices is video source verification on production lines and development laboratories.

2.1. UUCD SDK

The UUCD Software Development Kit (SDK) contains the required files to access UCD devices; source code to create new applications that use UCD devices and basic examples that help end users to use UCD devices in their testing environments.

Example applications are released in public domain.

2.1.1. Using the custom API

API function calling convention

On Windows, all functions use the __stdcall calling convention, while in Linux the standard C calling convention is used. Due to this, the "__stdcall" part is removed from the function prototypes in this manual.

In Linux

Please include files "UUCD.h" and "UUCD_Types.h" to your project.

In Windows

When using C or C++ include files "UUCD.h", "UUCD.c" and "UUCD_Types.h" to your project.

With C# include "UUCD.cs" file. Add *using Unigraf;* statement.

Thread safety

All API functions are protected against harmful concurrent access.

Handle validity

Handles can be transferred between threads, but not between processes. Each process must obtain it's own handles.

Firmware Versions

Required firmware files are built in to the API.

API Versions

UCD API is available for 32-bit application development on Windows and Linux.

Function return values

For functions that return a handle (Such as UUCD_HANDLE): Zero (or NULL) is used as an indication of error. This type of function has an optional pointer parameter which can be utilized by client applications to find out what went wrong.

Functions that return UUCD_RESULT: Negative values are always errors. Zero indicates generic success. Positive values also indicate success, but also convey some other meaning defined per function in this manual.

Known issues

UUCD_ERROR_USB_IO requires device close / re-open for recovery. Increasing the capture application's priority may reduce the frequency of the problem.

Performance

The UCD Performance is limited by USB 2.0 transfer speed. For full HD video the expected frame rate is approximately 3 FPS. Lower resolutions will achieve higher frame rates.

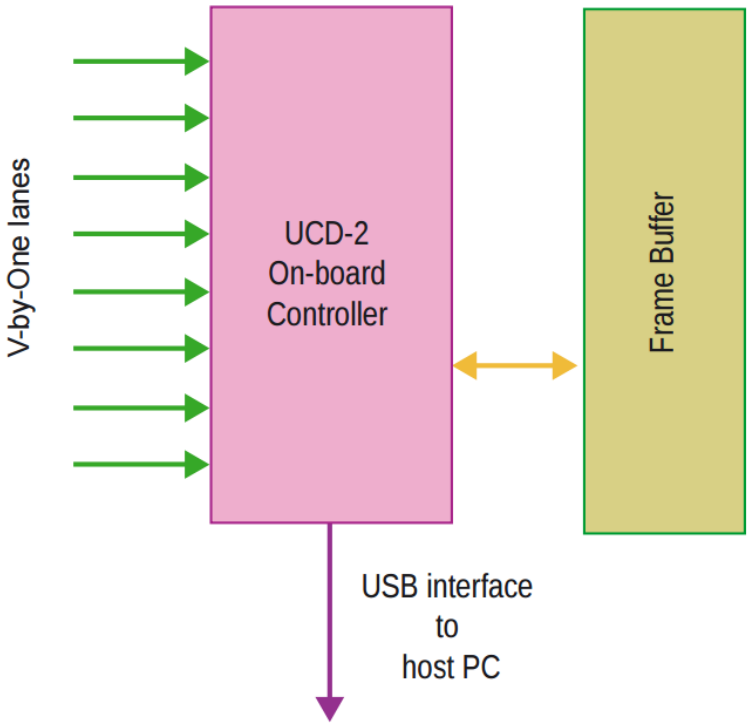
Features

The custom API can be used to perform the following tasks.

- Capture video
- Display video on screen *(Windows only)*
- Capture video with 6, 8, 10 and 12 bits per color channel.
- Re-arrange Vx1 lanes, and support Vx1 sections.
- Save video frames on disk with user selectable format.
- Capture a sequence of frames up to 256 frames long with no dropped frames. Please notice that the sequence length is limited by available memory.
- Memory management functions for simplicity.
- Software emulated “dummy device” for software development without real hardware.

2.2. Data flow

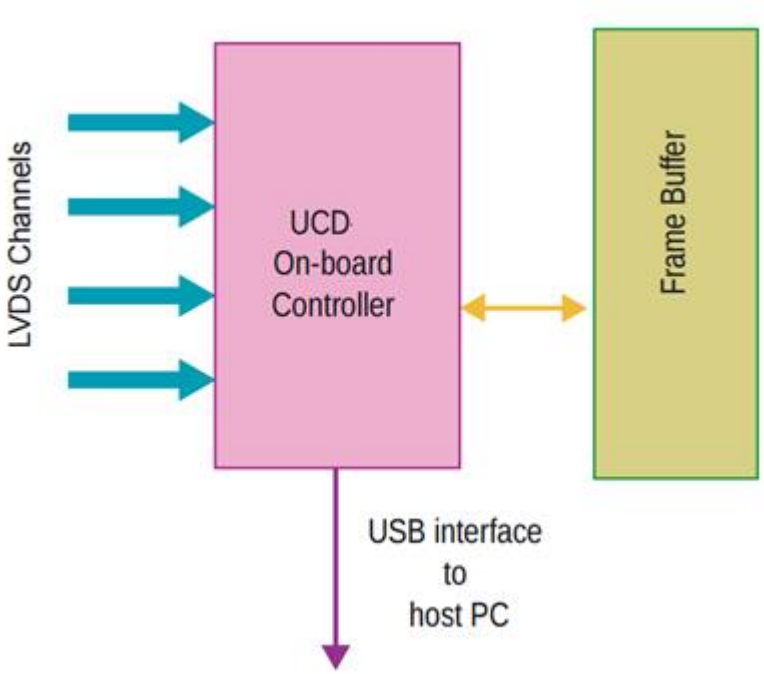
2.2.1. UCD-2 Vx1 device



The image on the left shows the data flow inside the UCD-2 Vx1 device.

The data is first captured from up to 8 V-by-One lanes and placed to the on-board frame buffer. Once at least one full frame is present in the frame buffer, the application may start to download frames over the USB-2 interface.

2.2.2. UCD-1 LV Device



The image on the left shows the data flow inside the UCD-1 LV device.

The data is first captured from up to 4 LVDS channels and placed to the on-board frame buffer. Once at least one full frame is present in the frame buffer, the application may start to download frames over the USB-2 interface.

2.3. SDK Examples

2.3.1. Generic capture procedure with UUCD API

In order to successfully capture a frame-sequence with UUCD API should follow the guidelines below:

1. Call UUCD_StartCapture function. The device will start to fill it's on-board frame buffer with video data.
2. Optionally call UUCD_GetFrameInfo function to check the dimensions and format of the next frame available frame.
3. Allocate a transfer buffer using UUCD_AllocateFrameBuffers function.
4. Transfer the next frame using UUCD_GetFrame function.
5. Check the result. If the result is “Frame-size mismatch”, go back to step 3. If the result is “Buffer Empty” go to step 7. If the result is OK, proceed to next step.
6. Process the frame. In order to capture next frame loop back to step 4; To stop capture proceed to next step.
7. Call UUCD_StopCapture function. To start a new capture loop back to step 1, or proceed to exit the application.

2.3.2. Example 1

This example opens up a console window for information and error message output purposes, and another window in which a video preview is displayed.

The example then starts a second thread which captures video frames and displays the frames in the preview window using UUCD API Helper functions. Each frame is also saved to a file using UUCD API Helper function.

3. REFERENCE

This section describes the functions designed in the API, and their intended operation.

3.1. External functions

3.1.1. UUCD_LoadApi

This function loads and resolves the UUCD functions and provides access to them.

```
UUCD_RESULT UUCD_LoadApi
(
char *DLL_Name
);
```

Synopsis

Linux: UUCD_LoadApi function forwards calls to UUCD_Initialize function. This function is provided to enable source code compatibility between different OS'es. The call to this function can be omitted.

Windows: Load and resolve functions contained in the UUCD API. This function also calls the UUCD_Initialize to allow immediate access to the API functions. For C or C++ is located in "UUCD.C" file, and with C# in "UUCD.cs" file.

Parameters

DLL_Name

Windows: Pointer to a NULL terminated string containing the fully qualified name and path of the UUCD DLL. If a DLL name is provided, and the library is not found in given location, the function will fail.
This parameter can be NULL: If the parameter is NULL, then the loader will default to standard installation path and OS standard search paths.

Linux: Pointer to a NULL terminated string. This parameter is ignored, and provided only to enable compatibility with the Windows version.

(Continued...)

(...Continued)

Result

If the function succeeds, the return value is a positive value indicating the version of the loaded API library.

If the function fails, the return value is a negative error code.

Bits	Description
31	Error indication. If set the return value indicates error code, not version data.
30:16	Major version, 15 bits.
15:0	Minor version, 16 bits.

See Also

UUCD_UnloadApi, UUCD_Initialize

3.1.2. UUCD_UnloadApi

This function unloads the API DLL and releases any system resources used.

```
UUCD_RESULT UUCD_UnloadApi();
```

Synopsis

Linux: UUCD_UnloadAPi forwards calls to UUCD_CleanUp. This function is provided to enable source code compatibility between different OS'es. The call to this function can be omitted.

Windows: Unload the previously loaded UUCD DLL and releases any resources used by the UUCD loader. This function will call the UUCD_CleanUp() function. For C and C++ the function is defined in “UUCD.C” file, For C# use “UUCD.cs” instead.

Result

The return value is zero.

See Also

UUCD_LoadApi, UUCD_CleanUp

3.2. API initialization

3.2.1. UUCD_Initialize

This function initializes the API for subsequent calls.

```
UUCD_RESULT UUCD_Initialize();
```

Synopsis

Initialize the UUCD API for use. This function must be called at least once before calling any other functions. Calls to UUCD_Initialize are reference counted; The UUCD_CleanUp function must be called equal number of times in order to release library resources.

Windows: This function is called from the UUCD_LoadApi. It is not needed to call it explicitly.

Parameters

This function takes no parameters

Result

If the function fails, the return code is a negative error code. The API Function will not be available for use.

If the function succeeds, the return values a positive value indicating reference count after the Initialize call. The API Functions are ready for use.

See Also

UUCD_CleanUp

3.2.2. UUCD_CleanUp

Release API resources; Close all devices and stop capture processes.

```
UUCD_RESULT UUCD_CleanUp();
```

Synopsis

Call UUCD_CleanUp when the application is done using UUCD API. Calls to this function are reference counted: If the reference count reaches zero, all capture processes are stopped, devices are closed and API memory allocations are released.

Windows: This function is called from the UUCD_UnloadApi. It is not needed to call it explicitly.

Parameters

This function takes no parameters

Result

If the function fails, the return value is a negative error code:

Important: Receiving an error code from this function indicates a problem in the client application, which should be fixed prior to release. Test builds should check the return value from this function, while release versions should ignore error codes from this function. Due to the nature of this function, it is unlikely that the error could be recovered from.

If the function succeeds, the return values is zero.

See Also

UUCD_Initialize

3.3. Device information and control functions

3.3.1. UUCD_EnumDevices

Enumerate all devices compatible with this API and return information about the devices.

```
UUCD_RESULT UUCD_EnumDevices
(
UUCD_DEVICE_DESCRIPTOR *DeviceInfoArray,
int ArraySize
);
```

Synopsis

This functions searches the local system for any devices that are compatible with this API. Information about each device is placed into the given device descriptor structures, and the function returns the number of devices found.

Parameters

DeviceInfoArray

Pointer to an array of UUCD_DEVICE_DESCRIPTOR structure. This function will then fill these structures until all available structures are filled, or no more devices are found. If the array contains more descriptors than required, the remaining descriptors will remain unchanged.

This parameter can be NULL: If this parameter is NULL, then the function will only calculate the number of devices present in the system.

ArraySize

Indicates the number of UUCD_DEVICE_DESCRIPTORS in the descriptor array. If the DeviceInfoArray pointer is NON-NULL, then this value must be non-zero.

Result

If the function fails, the return value is a negative error code. The contents of DeviceInfoArray will be undefined.

If the function succeeds, the return value is a positive value indicating the number of devices found from the system regardless of ArraySize parameter.

If multiple devices are connected via synchronization cables only the master device is added to the result array. API controls slave devices transparently via the master device.

Important: If the return value is zero, it means that the function succeeded, but there are no compatible devices present.

See Also

UUCD_OpenDevice

3.3.2. UUCD_EnumOperationModes

Enumerates the operation modes available for the given device.

```
UUCD_RESULT UUCD_EnumOperationModes
(
UUCD_DEVICE_DESCRIPTOR Device,
UUCD_OPERATION_MODE_DESCRIPTOR *OpModeArray,
int OpModeArraySize
);
```

Synopsis

Enumerates the operation modes available for a given device. ID and name of each mode is copied into the given descriptors, and the total number of available operation modes is returned.

Parameters

Device
: Identifies the device for which to retrieve the operation modes.

OpModeArray
: Array of UUCD_OPERATION_MODE_DESCRIPTOR structures. This function will fill this array until no more descriptors are available, or all available operation modes are copied.
Unused UUCD_OPERATION_MODE_DESCRIPTOR structure contents will be left to an undefined state.
This parameter can be NULL; If this parameter is NULL, then the OpModeArraySize parameter must be zero. In this case the function only returns the number of operation modes available for the indicated device.

OpModeArraySize
: Number of UUCD_OPERATION_MODE_DESCRIPTOR structures in the array. This parameter can be zero only if the OpModeArray pointer is NULL.

Result

If the function fails, the return value is a negative error code, and the contents of the operation mode array is undefined.

If the function succeeds, the return value is a positive number (or zero) indicating the number of available operation modes for the device regardless of the OpModeArraySize parameter.

See Also

UUCD_EnumDevices, UUCD_OpenDevice

3.3.3. UUCD_OpenDevice

Open a device for operations

```
UUCD_HANDLE UUCD_OpenDevice
(
UUCD_DEVICE_DESCRIPTOR *DeviceToOpen,
UUCD_RESULT *ResultCode
UUCD_MODE OperationModeID
);
```

Synopsis

Open a device for operations. The device to open is identified through a device descriptor previously received from UUCD_EnumDevices function. If the device open succeeds, the device

is set to it's default operating mode. The device firmware is loaded while the device is opened and it may take a few seconds for this process to complete. In order to change the operation mode, it is necessary to close and re-open the device.

Parameters

DeviceToOpen
A Device Descriptor containing information about the device to be opened. These descriptors are received by calling UUCD_EnumDevices function.

ResultCode
Pointer to an UUCD_RESULT variable, which receives the result code from this function.

This parameter can be NULL: If the parameter is NULL, then no result code is available to calling application.

OperationModeID
Indicates the desired operation mode of the device. Use Zero to start in the device's "default" mode, or use UUCD_EnumOperationModes function to retrieve available operation modes.

Result

If the function fails, the return value is NULL.

If the function succeeds, the return values is NON-NULL.

If ResultCode parameter is NON-NULL, the return code is placed to the referenced variable regardless of success. (i.e. if the function succeeds, the referenced variable will be set to zero).

Important: Check the returned handle value to detect errors. While the result value is optionally passed to the calling application, it should be considered as additional detail as to why the function failed in case of a failure.

See Also

UUCD_EnumDevices, UUCD_EnumOperationModes, UUCD_CloseDevice

3.3.4. UUCD_CloseDevice

Stop capture process on the device and close the device.

```
UUCD_RESULT UUCD_CloseDevice
(
UUCD_HANDLE Device
);
```

Synopsis

Stop any capture processes on the device, and close the device. The function will also release any resources allocated for accessing the device.

Parameters

Device
Indicates the device to close. Once the device is closed, the handle is invalid and can't be used to access the device.

Result

If the function fails, the return value is a negative error code.

Important: Receiving an error code from this function indicates a problem in the client application, which should be fixed prior to release. Test builds should check the return value from this function, while release versions should ignore error codes from this function. Due to the nature of this function, it is unlikely that the error could be recovered from.

If the function succeeds, the return value is zero.

See Also

UUCD_OpenDevice

3.3.5. UUCD_EnumerateConnectors

Enumerates input/output connectors of the given device.

```
UUCD_RESULT UUCD_EnumConnectors
(
UUCD_HANDLE Device,
UUCD_CONNECTOR_TYPE *Connectors,
int ArraySize
);
```

Synopsis

Enumerates input/output connectors of a given device. Connector types are copied into the given array, and the number of available connectors is returned. Possible connector type values are listed in UUCD_CONNECTOR_TYPE.

Parameters

Device

Identifies the device for which to retrieve the connector list.

Connectors

Array of UUCD_CONNECTOR_TYPE elements which are defined as WORD. This function will fill this array until no more connectors are available, or all available connectors are copied.
Unused array elements will be left to an undefined state.
This parameter can be NULL. In this case the function only returns the number of the device connectors.

ArraySize

Number of UUCD_CONNECTOR_TYPE elements in the array. This parameter can be zero if the Connectors pointer is NULL.

Result

If the function fails, the return value is a negative error code, and the contents of the Connectors array is undefined.

If the function succeeds, the return value is a positive number (or zero) indicating the number of device connectors regardless of the ArraySize parameter.

See Also

UUCD_SelectConnector, UUCD_CONNECTOR_TYPE

3.3.6. UUCD_SelectConnector

Selects input/output connector for following operations.

```
UUCD_RESULT UUCD_SelectConnector
(
UUCD_HANDLE Device,
int Connector
);
```

Synopsis

Selects input/output connector of a given device for operations like StartCapture() etc.

Parameters

Device
Identifies the device for which the connector needs to be selected.

Connector
An index of a connector in the array of UUCD_CONNECTOR_TYPE elements return by UUCD_EnumConnectors() function.

Result

If the function fails, the return value is a negative error code, and the contents of the Connectors array is undefined.

If the function succeeds, the return value is UUCD_SUCCESS.

See Also

UUCD_EnumConnectors

3.4. Video Capture Functions

3.4.1. UUCD_SetOption

Set a capture option.

```
UUCD_RESULT UUCD_SetOption
(
UUCD HANDLE Device,
UUCD_OPTION OptionID,
int OptionValue
);
```

Synopsis

Change capture options of a device. All options reset to default when the device is first opened, or when the operation mode is changed. All capture options should be set-up before starting capture. Options can't be changed during capture operations, unless explicitly defined possible during capture.

Parameters

Device
Indicates the device for which to change options.

OptionID
Indicates the option to change. See "Supported Options" for table of available option ID's.

OptionValue
New setting for the option. The exact purpose of a value is defined per option.

Only positive values can be set for options, negative values are considered invalid and will not be applied; However, using a negative value does not cause an error: Instead the current value of the option is returned, and the setting is left unchanged.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value is the previous setting of the option.

See Also

UUCD_StartCapture, UUCD_StopCapture

3.4.2. UUCD_StartCapture

Start a capture operation on a device.

```
UUCD_RESULT UUCD_StartCapture
(
UUCD_HANDLE Device
);
```

Synopsis

Start a capture process on the device. Once the capture process is successfully started, the captured data can be recovered using the UUCD_GetFrame function.

Parameters

Device
Indicates the device on which to start the capture.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value is zero.

See Also

UUCD_StopCapture, UUCD_GetFrame, UUCD_SetDeviceMode

3.4.3. UUCD_StopCapture

Stop a running capture process on a device.

```
UUCD_RESULT UUCD_StopCapture
(
UUCD_HANDLE Device
);
```

Synopsis

Stop a capture process previously started with UUCD_StartCapture.

Parameters

Device

Indicates the device on which to stop the capture process.

Result

If the function fails, the return value is a negative error code.

Important: Receiving an error from this function implies an issue in the client application capture control mechanism, which should be fixed prior to release.

If the function succeeds, the return value is zero.

See Also

UUCD_StartCapture

3.4.4. UUCD_GetFrame

Retrieve a single frame of capture data from a device.

```
UUCD_RESULT UUCD_GetFrame
(
UUCD_HANDLE Device,
UUCD_FRAME_DESCRIPTOR *Frame
);
```

Synopsis

Retrieves a single frame of capture data. The capture process must be running on the indicated device. On output, the frame contains video.

Parameters

Device

Indicates the device from which to retrieve the frame data.

Frame

Pointer to a frame descriptor structure. The frame-descriptor structure describes the video buffer to which the frame data is transferred. Please use

UUCD_AllocateFrameBuffers function to allocate memory for frame transfers and frame descriptor initialization.

Result

If the function succeeds, the return value is zero.

If the function fails, the return value is a negative error code.

If frames are captured in buffered mode (multiple UUCD_GetFrame() calls after a single UUCD_StartCapture() that fills up the device internal memory) then the UUCD_ERROR_BUFFER_EMPTY error means that all buffered frames have been transferred. Call UUCD_StartCapture() again to refill the device buffer.

Important: This function may fail due to changes in video input status. In this case a new video frame buffer needs to be allocated. Once a suitable buffer is allocated, the transfer can be re-tried. For other errors the normal response is to start a new capture sequence.

See Also

UUCD_StartCapture, UUCD_AllocateFrameBuffers, UUCD_StopCapture

3.5. Helper functions

3.5.1. UUCD_GetFrameInfo

Retrieves the video frame information of the next frame in transfer queue on the device.

```
UUCD_RESULT UUCD_GetFrameInfo
(
UUCD_HANDLE Device,
UUCD_FRAME_INFO *FrameInfo
);
```

Synopsis

Retrieves the dimensions and format information of the next frame available for transfer from the device. If there are no frames ready to be transferred on the device, the function will fail.

Parameters

Device

Indicates the device from which the frame information is retrieved from.

FrameInfo

Pointer to frame information structure, which will receive the frame information data.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value is zero and the frame information is placed to the provided structure.

See Also

UUCD_FRAME_INFO

3.5.2. UUCD_AllocateFrameBuffers

Allocate memory buffers for video capture and initialize frame descriptors

```
UUCD_RESULT UUCD_AllocateFrameBuffers
(
UUCD_HANDLE Device,
UUCD_FRAME_DESCRIPTOR *FrameArray,
int FrameArraySize
);
```

Synopsis

Allocate memory buffers for video capture and initialize the frame descriptors. The function reads the current video format from the device and allocates memory buffers of necessary size. If there are no frames ready for transfer on the device, the function will fail.

Parameters

Device
Indicates the device from which to read the video format information.

FrameArray
Pointer to an array of UUCD_FRAME_DESCRIPTOR structures. The function will fill out the structures. If the function fails, the content of the descriptors will be undefined.

FrameArraySize
Number of frame descriptors in the frame descriptor array.

Result

If the function fails, the return value is a negative error code. The FrameArray contents will be undefined.

If the function succeeds, the return value is zero and the frame array is filled with valid frame-descriptor data. The buffers must be released with UUCD_FreeFrameBuffers.

See Also

UUCD_FreeFrameBuffers

UUCD_FreeFrameBuffers

3.5.3. UUCD_FreeFrameBuffers

Release frame buffer allocation previously allocated with UUCD_AllocateFrameBuffers.

```
UUCD_RESULT UUCD_FreeFrameBuffers
(
UUCD_DEVICE Device,
UUCD_FRAME_DESCRIPTOR *FrameArray,
int FrameArraySize
);
```

Synopsis

Release a previously allocated array of frame buffers. Each call to FreeFrameBuffers must match a previous allocation call: Allocations can't be freed partially nor can two allocations be freed with a single call to UUCD_FreeFrameBuffers. The frame array contents will be cleared by this function.

Parameters

Device
Handle to the device which was used to allocate the frame.

FrameArray
Pointer to an array of UUCD_FRAME_DESCRIPTOR previously filled by a call to UUCD_AllocateFrameBuffers.

FrameArraySize
Number of frame descriptors in the frame array.

Result

If the function fails, the return value is a negative error code, no memory is released and the frame descriptor array is unchanged.

Important: Receiving an error from this function implies a memory management issue in the client application, which should be fixed before release.

If the function succeeds, the return value is zero and the frame descriptor array is cleared.

See Also

UUCD_AllocateFrameBuffers

3.5.4. UUCD_FrameCompare

Compares two frames in specified way and stores results to a third frame buffer.

```
UUCD_RESULT UUCD_FrameCompare
(
UUCD_FRAME_DESCRIPTOR *Reference,
UUCD_FRAME_DESCRIPTOR *CompareWith,
UUCD_FRAME_DESCRIPTOR *Result,
int CompareMode
);
```

Synopsis

Compare the frame data in *CompareWith* to *Reference* and save the comparison results to *Result* frame. All frame buffers must have identical dimensions and color-space. At the moment there are binary (XOR) comparison mode, and arithmetic (SUB) comparison mode. In XOR mode, the Reference data and Compare data are XOR'ed together; In SUB mode the difference between Reference and Compare data are calculated by subtracting the lower value from the higher one. The function works in all supported color-spaces.

Parameters

Reference

Reference frame for the comparison.

CompareWith

A frame to compare against the reference frame.

Result

A frame to contain the comparison results.

CompareMode

Identifies which comparison mode to use.

Define	Value	Description
UUCD_COMPARE_XOR	1	Reference and CompareWith frames XOR'ed, and the result of the operations is placed into Result.
UUCD_COMPARE_SUB	2	The absolute difference between Reference and CompareWith is calculated and placed into Result.

Results

If the function succeeds, the return value is zero.

If the function fails, the return value is a negative error code and the contents of the Result frame are undefined.

See Also

UUCD_AnalyseFrame

3.5.5. UUCD_AnalyzeFrame

Analyze a results frame generated by UUCD_FrameCompare function.

```
UUCD_RESULT UUCD_AnalyseFrame
(
UUCD_FRAME_DESCRIPTOR *Frame,
int RESERVED1,
UUCD_TAG *ResultTagArray,
int ArraySize
);
```

Synopsis

Run an analysis function on a frame. Analysis results are placed to a tag-list. The tag-list is filled until there are no more result values available, or until there is no more space in the tag-list.

Currently the function is limited to working on result frames received from UUCD_FrameCompare. This function may be extended to provide additional analysis methods in the future

Important: *Don't call the function twice to find the correct tag-list size for each frame. Doing so will destroy performance. Instead remember the previous tag-list size returned and use this value to allocate tag lists. If the function subsequently returns a larger value, release the current list and re-allocate using new value.*

Important: *Do not assume that the order of the result tags will always remain constant.*

Parameters

Frame
: Pointer to a frame to be analyzed.

RESERVED1
: Reserved; Must be zero.

ResultTagArray
: Pointer to an array of UUCD_TAG structures. The function will fill the tag-list with analysis results.

ArraySize
: Number of UUCD_TAG structures provided in ResultTagArray.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value is a positive value indicating the number of result tags regardless of ArraySize parameter.

See Also

UUCD_FrameCompare

3.5.6. UUCD_CreateDataMapping

Generate mapping data for UUCD_Remap function.

```
UUCD_RESULT UUCD_CreateDataMapping
(
UUCD_REMAP_CONFIG *Config,
UUCD_DATA_MAPPING *Mapping
);
```

Synopsis

Allocates required space for the mapping information and generates the data remapping table required by the UUCD_Remap function.

Parameters

Config

Pointer to UUCD_REMAP_CONFIG Structure which contains required information to build the mapping table.

Mapping

Pointer to Mapping data structure. The structure must be initialized to all zero when calling this function.

Result

If the function fails, the return value is a negative error code and the Mapping structure contents remain unmodified.

If the function succeeds, the return value is zero.

See Also

UUCD_DeleteDataMapping, UUCD_Remap

3.5.7. UUCD_DeleteDataMapping

Release resource allocation of the given data mapping.

```
UUCD_RESULT UUCD_DeleteDataMapping
(
UUCD_DATA_MAPPING *Mapping
);
```

Synopsis

Release memory allocated for the Mapping table.

Parameters

Mapping

Pointer to a mapping structure. The associated memory is released and the mapping structure is cleared for re-use.

Results

If the function fails, the return value is a negative error code and the Mapping contents remain unmodified.

If the function succeeds, the return value is zero. The Mapping structure can be re-used after a successful call.

See Also

UUCD_CreateDataMapping, UUCD_Remap

3.5.8. UUCD_Remap

Re-arrange pixels in a captured frame.

```
UUCD_RESULT UUCD_Remap
(
UUCD_FRAME_DESCRIPTOR *Source,
UUCD_FRAME_DESCRIPTOR *Target,
UUCD_DATA_MAPPING *Mapping
);
```

Synopsis

Copy and re-arrange data from Source to Target as described in the Mapping. Currently this function is used to deliver Vx1 sections support, and Vx1 channel re-arrange features. Source and Target frames must be equal in dimensions and color-space.

Parameters

Source
Pointer to a frame received from UUCD_GetFrame.

Target
Pointer to a frame that will contain a re-organized copy of the source frame.

Mapping
Pointer to a mapping data table created with UUCD_CreateDataMapping function.

Results

If the function fails, the return value is a negative error code and the target frames contents are undefined.

If the function succeeds, the return value is zero.

3.5.9. VP_CreateVideoView

Create a video preview component inside an existing window.

```
VIDEO_HANDLE VP_CreateVideoView
(
HWND ContainerWindow,
UUCD_FRAME_DESCRIPTOR *FirstFrame,
UUCD_RESULT *ResultCode
);
```

Synopsis

Create a video graphics display component inside the given window. The created view will completely fill the container window's client area. The graphics content from the FirstFrame will be placed into the window. The FirstFrame descriptor also defines the source video dimensions. All subsequent frames must have identical dimensions compared to the FirstFrame.

The underlying technology is either Direct 2D, or Direct Draw depending on availability on the local system. The function prefers Direct 2D.

Important: This function is available in Windows version only!

Parameters

ContainerWindow
Handle to a window which will contain the graphics display.

FirstFrame
Pointer to a frame descriptor, which is used to determined graphics dimension, and to initialize the video display contents.

ResultCode
Pointer to a result code variable.

This parameter can be NULL: If the parameter is NULL, then no result code will be available to the calling application.

Result

If the function fails, the return value is NULL.

If the function succeeds, the return value is a NON-NULL handle value used to identify the created graphics display component.

If the ResultCode pointer is NON-NULL, then a result code is placed to the indicated location regardless of function result.

Important: Check function success from the resulting handle value. The optional result code should be considered as additional information as to why the function failed in case of failure.

See Also

VP_DisplayFrmae, VP_DestroyVideoView

3.5.10. VP_DisplayFrame

Update the contents of a video view component created with VP_CreateVideoView.

```
UUCD_RESULT VP_DisplayFrame
(
VIDEO_HANDLE View,
UUCD_FRAME_DESCRIPTOR *Frame
);
```

Synopsis

Updates graphics content of a video display component previously created with VP_CreateVideoView. The given video frame dimensions must match the dimensions of the

graphics display. Please note that the preview display only supports 8 bits per color channel modes: Attempting to display images with deeper color depth will result to an error.

Important: This function is available in Windows version only!

Parameters

View
Handle to a video view component.

Frame
Pointer to a frame descriptor containing graphics data to be placed in the view component.

Result

If the function fails, the return value is a negative error code.

Important: This function most likely fails due to frame dimension(s) change, or color space change. The common response to these errors is to destroy the video component and then create a new component with updated parameters.

If the function succeeds, the return value is zero.

See Also

VP_CreateVideoView, VP_DestroyVideoView

3.5.11. VP_DestroyVideoView

Destroy a video view component.

```
UUCD_RESULT VP_DestroyVideoView
(
VIDEO_HANDLE View
);
```

Synopsis

Destroy a video view component.

Important: This function is available in Windows version only!

Parameters

View
Handle to the video view component to destroy.

Result

If the function fails, the return value is a negative error code.

Important: Receiving an error from this function implies an issue in the client application which should be fixed prior to release. Due to the nature of the function, there likely is no way to recover from the error.

If the function succeeds, the return value is zero.

See Also

VP_CreateVideoView

3.5.12. UUCD_SaveFrameToFile

Save given frame to a file

```
UUCD_RESULT UUCD_SaveFrameToFile
(
char *FileName,
UUCD_FRAME_DESCRIPTOR *Frame,
UUCD_FILE_FORMAT_ID FileFormat
);
```

Synopsis

Save the indicated frame to an on-disk file. The function assumes that any existing file may be overwritten: The function will delete any existing file and create a new file.

For PPM files, frame that contain 8 bits per color channel or less are up-sampled to 8 bits per color, and frames that contain more than 8 bits per color are up-sampled to 16 bits per color. The up-sampling will add new bits as least significant bits. Any inserted bits are zeroed.

For BMP and JPEG files, frames are always converted to 8 bits. Inserted bits are zeroed, discarded bits are ignored, no dithering applied.

Parameters

FileName

Pointer to a NULL terminated string containing full path and file name of the target file. The file must be reachable with standard file access functions.

Frame

Pointer to the frame descriptor to be saved on disk.

FileFormat

ID of the file-format.

ID	Name	Description
0	FFID_PPM	8 or 16 bits per color channel PPM file. The color-depth is automatically selected.
1	FFID_BMP	8 bits per channel BMP file.
2	FFID_JPEG	8 bits per channel JPEG file.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value a positive value indicating the resulting size of the target file.

See Also

-

3.5.13. UUCD_GetVideoTimingData

Retrieves timing measurement data of the given channel (Vx1 lane).

```
UUCD_RESULT UUCD_GetVideoTimingData
(
UUCD_HANDLE Device,
int Channel,
UUCD_TAG *ResultTagArray,
int TagArraySize
);
```

Synopsis

Retrieves the timing measurement results as an array of tagged values.

Parameters

Device
Indicates the device from which the timing information is retrieved.

Channel
The V-By-One lane number.

ResultTagArray
Pointer to an array of UUCD_TAG elements to store measurement results to. The pointer can be NULL to determine the amount of available measurement data.

TagArraySize
The dimension of the results array. Ignored if the array pointer is NULL.

Result

If the function fails, the return value is a negative error code.

If the function succeeds, the return value is the number of measurement data available. It might differ from the actual amount copied to the result array if its dimension specified in the *TagArraySize* is not large enough.

See Also

UUCD_TAG structure.

3.6. Plugins

3.6.1. Overview

An UUCD plugin is a DLL that applies additional transformation to each frame of captured video. The DLL should be placed in the dedicated folder located next to the UUCD.dll which is typically installed under *Unigraf\shared*, so the plugins location would be:

C:\Program Files (x86)\Common Files\Unigraf\shared\UUCD_Plugins

All plugins are loaded when UUCD_Initialize (see 5.2.1) is called. Plugins designed to work with a particular UCD device appear as additional operation modes in the list an application obtains with UUCD_EnumOperationModes (see 5.3.2).

In order to associate a plugin with a capture device and thus apply the plugin's processing functionality to the captured video, the plugin's mode should be given as a parameter to UUCD_OpenDevice call (see 5.3.3).

It is possible to have multiple plugins for the same device but no more than one plugin can be taken into use at a time. Plugins cannot be chained.

A plugin DLL should supply the functions each of them is described in the following chapters:

- UUCD_InitPlugin
- UUCD_GetAdditionalMode
- UUCD_GetFirmware
- UUCD_ProcessFrame
- UUCD_SetOption

The last function is optional.

Note: plugins are available only in Windows.

3.6.2. UUCD_InitPlugin

This function initializes a plugin.

```
__declspec(dllexport) char *__stdcall
UUCD_InitPlugin
(
);
```

Synopsis

All plugins are initialized when UUCD_Initialize (see 5.2.1) is called.

Parameters

None

Result

The function returns the name of the capture device a plugin is designed to be associated with. The name should match the name of the USB device as shown in the Windows Device Manager, for example, "UCD-1 MLC A".

Implementation Example

```
__declspec(dllexport) char *__stdcall UUCD_InitPlugin()
{
return "UCD-1 MLC A";
}
```

See Also

UUCD_Initialize

3.6.3. UUCD_GetAdditionalMode

This function gets plugin's operation mode.

```
__declspec(dllexport) UUCD_OPERATION_MODE_DESCRIPTOR *
__stdcall UUCD_GetAdditionalMode
(
);
```

Synopsis

Each plugin provides one operation mode which is added to the native list of the associated device's operation modes.

Parameters

None

Result

The function returns a pointer to the mode descriptor. The mode name can be shown in GUI and the mode purpose is a hint for software about how to use this mode.

Implementation Example

```
static UUCD_OPERATION_MODE_DESCRIPTOR Mode = {
0, // UUCD_MODE assigned dynamically
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // GUID if needed
"Through Plugin", // the mode name
1,11, // firmware version
1,5,  // software version
UUCD_MODE_MLC_CAPTURE // mode purpose
};

__declspec(dllexport) UUCD_OPERATION_MODE_DESCRIPTOR *
__stdcall UUCD_GetAdditionalMode()
{
return &Mode;
}
```

See Also

UUCD_OPERATION_MODE_DESCRIPTOR structure.

3.6.4. UUCD_GetFirmware

This function gets plugin's firmware or a reference to device's firmware to use.

```
__declspec(dllexport) FW_ITEM * __stdcall UUCD_GetFirmware
(
);
```

Synopsis

Each plugin either contains special firmware to support its operation mode or provides an ID of the associated device's firmware that should be used.

Parameters

None

Result

The function returns a pointer to the firmware descriptor.

Implementation Example

```
static FW_ITEM Firmware = {
// no special firmware is included in this plugin
0x31304756 /*VG01*/, 0, 0, 0, 0
};

__declspec(dllexport) FW_ITEM * __stdcall UUCD_GetFirmware()
{
return &Firmware;
}
```

See Also

FW_ITEM structure.

3.6.5. UUCD_SetOption

This function copies options set for the device to the associated plugin.

```
UUCD_RESULT __declspec(dllexport) __stdcall
UUCD_SetOption
(
UUCD_OPTION OptionID,
int OptionValue
);
```

Synopsis

This plugin's function is invoked from UUCD_SetOption (see 5.4.1) call if a device is associated with a plugin.

This allows passing additional transformation parameters to the plugin if required.

This function is optional as plugins may or may not need additional information for image processing.

Parameters

OptionID

Indicates the option to change. See "Supported Options" for table of available option ID's.

OptionValue

New setting for the option. The exact purpose of a value is defined per option.

Result

If the function fails, the return value is a negative error code that would be then returned by UUCD_GetFrame.

If the function succeeds, it should return UUCD_SUCCESS.

Implementation Example

```
static int nChannels = 0;

UUCD_RESULT __declspec(dllexport) __stdcall
UUCD_SetOption(UUCD_OPTION OptionID, int OptionValue)
{
if (OptionID == UUCD_OPTION_CAPTURE_CHANNELS)
{
      // the value might be later used for processing
      nChannels =  OptionValue;
}
return UUCD_SUCCESS;
}
```

See Also

UUCD_GetFrame

3.6.6. UUCD_ProcessFrame

Applies desired additional transformation to a captured video frame.

```
UUCD_RESULT __declspec(dllexport) __stdcall
UUCD_ProcessFrame
(
 UUCD_FRAME_DESCRIPTOR *Frame
);
```

Synopsis

The function is invoked from UUCD_GetFrame (see 5.4.4) if a device is associated with a plugin.

Transformation should be applied in place, that is result overwrites the input data.

Parameters

Frame

Pointer to the frame descriptor that comprises frame dimensions and a pointer to the buffer that contains a frame image.

Result

If the function fails, the return value is a negative error code that would be then returned by UUCD_GetFrame.

If the function succeeds, it should return UUCD_SUCCESS.

Implementation Example

```
__declspec(dllexport) UUCD_RESULT __stdcall
UUCD ProcessFrame(UUCD FRAME DESCRIPTOR *Frame)
{
unsigned char *buff = (unsigned char *)Frame->Buffer;
int num = Frame->FrameHeight * Frame->FrameWidth;
int elem =  Frame->BytesPerElement; // element size
for (int j = 0; j < num; j++, buff += elem)
{
      // swaps two bytes of each image element
      unsigned char tmp = *buff;
      *buff = buff[1];
      buff[1] = tmp;
}
return UUCD_SUCCESS;
}
```

See Also

UUCD_GetFrame

3.7. Supported Options

Table of Option ID's supported

Name	ID (Hex)	Supported Devices	Description
UUCD_OPTION_ALL	7fffffff	all	Refers to all options supported per device.
UUCD_OPTION_LVDS_MODE	1	LV, MLC	Defines number of LVDS channels for LV devices to capture
UUCD_OPTION_LVDS_MAP	2	LV	Defines which LVDS mapping method to use
UUCD_OPTION_COLOR_DEPTH	3	LV, Vx1	Select color-depth used for video capture.
UUCD_OPTION_COMBINE_MODE	4	LV, Vx1	Determines how the video frame is constructed from LVDS lanes
UUCD_OPTION_FRAMEINFO_TIMEOUT	5	All	Determines how long to wait for a frame to arrive for transfer.
UUCD_OPTION_CAPTURE_CHANNELS	6	Vx1	Number of V-By-One lanes used to input video.
UUCD_OPTION_INPUT_COLOR_DEPTH	7	LV, Vx1	Color-depth of input video. This option must be set after setting UUCD_OPTION_COLOR_DEPTH.
UUCD_OPTION_DUMMY_FRAMESIZE	8	Vx1	Defines frame resolution for output from the dummy-device. Bits [31:16] define horizontal resolution, and bits [15:0] define vertical resolution.
UUCD_OPTION_SYNC_MODE	9	LV, Vx1	Use Horizotanl/Vertical sync signals or Data Enable to capture video frames.
UUCD_OPTION_HTPDN_CONTROL	10	Vx1	Defines Vx1 HTPDn signal behaviour
UUCD_OPTION_LOCKN_CONTROL	11	Vx1	Defines Vx1 LOCKn signal behaviour
UUCD_OPTION_LOCN_DELAY	12	Vx1	Defines Vx1 LOCKn signal delay
UUCD_OPTION_VIDEO_VALID_DELAY	13	Vx1	Defines when Vx1 video signal is valid.

UUCD_OPTION_EXT_TRIGGER	14	Vx1	Enables external trigger
UUCD_OPTION_MLC_CAPTURE	15	MLC	Selects line or frame capturing mode
UUCD_OPTION_MLC_INVERT	16	MLC	Selects which pairs need to be inverted
UUCD_OPTION_MLC_PAIRS_NUM	17	MLC	Number of pairs to capture
UUCD_OPTION_MLC_LINE_BYTES	18	MLC	Number of bytes per line
UUCD_OPTION_MLC_FRAME_LINES	19	MLC	Number of lines per frame
UUCD_OPTION_MLC_LINE_OFFSET	20	MLC	Number of lines from frame synchronization marker to frame start
UUCD_OPTION_MLC_NO_FSYNC	21	MLC	FW tries to synchronize with vertical blanking area
UUCD_OPTION_MLC_FSYNC_DELAY	22	MLC	Use frame synchronization delay
UUCD_OPTION_MLC_FSYNC_FALLING_EDGE	23	MLC	Frame synchronization on falling or rising edge
UUCD_OPTION_MLC_LSYNC_NO_FORCE_LEND	24	MLC	Line synchronization marker does not force line end
UUCD_OPTION_MLC_B_USE_LSYNC_A	25	MLC	B channel takes line sync. from A
UUCD_OPTION_MLC_NO_LSYNC	26	MLC	Presence of line synchronization marker
UUCD_OPTION_MLC_LSYNC_FALLING_EDGE	27	MLC	Line synchronization on falling or rising edge
UUCD_OPTION_MLC_RESET_PULSE_PAIR	28	MLC	Sets differential pair which holds the mini-LVDS reset pulse (line start marker)

In this table, LV and MLC are modes of UCD-1 device; Vx1 refers to UBCD-2 device.

3.7.1. UUCD_OPTION_ALL

Special option, which refers to all options supported in the device. This option is used to reset all options to defaults. Unlike most other options, this option has no state to read. When setting, the return value is always zero. This option is supported on all device types.

Value Range	Description
0	Reset all supported options to their default values.
Negative values	No effect; Return zero.
Non-zero positive values	No effect.

3.7.2. UUCD_OPTION_LVDS_MODE

This option selects LVDS channel count to input video.

Value Range	Description
1, 2 or 4	Number of LVDS channels used to input video
Other values	No Effect.

3.7.3. UUCD_OPTION_LVDS_MAP

This option selects the “pin-mapping” mode used for LVDS.

Name	Value	Description
VESA_MAPPING	0	Use VESA mapping.

JEIDA_MAPPING	1	Use JEIDA mapping.

3.7.4. UUCD_OPTION_CAPTURE_CHANNELS

Number of V-By-One lanes used to input video.

Value Range	Description
1, 2, 4 or 8	Number of V-By-One lanes used to input video.
Other values	No Effect.

3.7.5. UUCD_OPTION_COLOR_DEPTH

This option selects color depth for both, video capture and video input. To override video input color-depth use UUCD_OPTION_INPUT_COLOR_DEPTH.

Important: *When a different input color-depth is preferred set the UUCD_OPTION_INPUT_COLOR_DEPTH option after setting this option. Otherwise, the input color-depth setting will have no effect.*

Name	Value	Description
COLOR_DEPTH_6BIT	0	6 bits per color channel
COLOR_DEPTH_8BIT	1	8 bits per color channel (DEFAULT)
COLOR_DEPTH_10BIT	2	10 bits per color channel
COLOR_DEPTH_12BIT	3	12 bits per color channel

3.7.6. UUCD_OPTION_INPUT_COLOR_DEPTH

This option selects physical input signals' color-depth.

Important: *The UUCD_OPTION_COLOR_DEPTH option should be set before setting this option. Otherwise, this option will have no effect.*

Name	Value	Description
COLOR_DEPTH_6BIT	0	6 bits per color channel
COLOR_DEPTH_8BIT	1	8 bits per color channel (DEFAULT)
COLOR_DEPTH_10BIT	2	10 bits per color channel
COLOR_DEPTH_12BIT	3	12 bits per color channel

3.7.7. UUCD_OPTION_COMBINE_MODE

This options selects how the frame is constructed from the data received from the active LVDS lanes

Name	Value	Description
-	0	One pixel from each active channel in sequence.

3.7.8. UUCD_OPTION_FRAMEINFO_TIMEOUT

This option determines the time which the API waits for a frame to arrive for transfer when calling GetFrame.

Value Range	Description
Positive values	Time to wait for a frame to arrive, in milliseconds.

3.7.9. UUCD_OPTION_SYNC_MODE

This option defines which synchronization method the device uses.

Name	Value	Description
SYNC_DE_ONLY	0	Use Data Enable for syncronization only.
SYNC_HV	1	Use Horizontal and Vertical sync markers.

3.7.10.UUCD_OPTION_HTPDN_CONTROL

Defines how the HTPDn signal behaves.

Name	Value	Description
HTPDN_NORMAL	0	Normal operation (Default)
HTPDN_FORCE_LOW	1	The HTPDn signal is forced LOW at all times.
HTPDN_FORCE_HIGH	2	The HTPDn signal is forced HIGH at all times.

3.7.11.UUCD_OPTION_LOCKN_CONTROL

Defines how the LOCKn signal behaves.

Name	Value	Description
LOCKN_NORMAL	0	Normal Operation (Default). The LOCKn signal is delayed. Use UUCD_OPTION_LOCKN_DELAY to alter the delay time.
LOCKN_FORCE_LOW	1	The LOCKn signal is forced LOW at all times.
LOCKN_FORCE_HIGHT	2	The LOCKn signal is forced HIGHT at all times.
LOCKN_DELAY_LOW_AFTER_HTPDN	3	The LOCKn signal is forced low after the following conditions are met: HTPDn signal is detected, and the delay time has elapsed. Use UUCD_OPTION_LOCKN_DELAY to configure the delay time.

3.7.12.UUCD_OPTION_LOCKN_DELAY

Defines a delay time for LOCKn signal assertion.

Value Range	Description
0 - 671088	Number of 1¬µs intervals to delay LOCKn. Default is ~41.9 ms.

3.7.13.UUCD_OPTION_VIDEO_VALID_DELAY

Defines a delay time to wait before starting video capture after LOCKn has been pulled low.

Value Range	Description
0 - 671088	Number of 1µs intervals to delay video capture. Default is ~41.9ms

3.7.14.UUCD_OPTION_EXT_TRIGGER

Enables external trigger input of the device.

Name	Value	Description

EXT_TRIGGER_NONE	0	External trigger disabled, free running capture mode.
EXT_TRIGGER_FALLING	1	External trigger enabled, triggered by falling edge
EXT_TRIGGER_RISING	3	External trigger enabled, triggered by rising edge

3.7.15.UUCD_OPTION_MLC_CAPTURE

One new line or the whole frame is given each NEXT command.

Name	Value	Description
MLC_CAPTURE_LINES	0	External trigger disabled, free running capture mode.
MLC_CAPTURE_FRAMES	1	External trigger enabled, triggered by falling edge

3.7.16.UUCD_OPTION_MLC_INVERT

This setting affects both A and B channels. Setting of '1' will invert data. Use this if differential pair '+' and '-' are wrong.

Value Range	Description
0 - 127	Bit mask that specifies which LVDS pairs should be inverted. Bit 0 = pair 1, … bit 5 = pair 6, bit 6 = invert clock pair.

3.7.17.UUCD_OPTION_MLC_PAIRS_NUM

This setting affects both A and B channels. Higher pair number is dropped e.g. setting 4 pairs will not capture data in channel A pairs 5-6 or B pairs 5-6.

Value Range	Description
1 - 6	The number of LVDS pairs to capture.

3.7.18.UUCD_OPTION_MLC_LINE_BYTES

Number of bytes per line to capture.

Value Range	Description
0 - 32767	The number of LVDS pairs to capture.

3.7.19.UUCD_OPTION_MLC_FRAME_LINES

Number of lines per frame.

Value Range	Description
0 - 32767	The number of lines per frame.

3.7.20.UUCD_OPTION_MLC_LINE_OFFSET

Number of lines from frame synchronization marker to frame start. This is a SIGNED (two's complement) number.

Value Range	Description
-16384 - 16383	The frame offset from the synchronization marker. Signed value.

3.7.21.UUCD_OPTION_MLC_NO_FSYNC

Without frame synchronization marker, the FW tries to synchronize with vertical blanking area. Frame sync. is set if line start has not been found within approximately 8200 clocks.

Name	Value	Description
MLC_SYNC_PRESENT	0	Require frame synchronization signal.
MLC_SYNC_NOT_PRESENT	1	There is no frame synchronization marker.

3.7.22.UUCD_OPTION_MLC_FSYNC_DELAY

Use delay if frame synchronization is close to line end

Name	Value	Description
MLC_DONT_USE	0	No delay
MLC_USE	1	Add constant delay to frame synchronization

3.7.23.UUCD_OPTION_MLC_FSYNC_FALLING_EDGE

Frame synchronization on either falling or rising edge.

Name	Value	Description
MLC_SYNC_RISING_EDGE	0	Use frame synchronization rising edge.
MLC_SYNC_FALLING_EDGE	1	Use frame synchronization falling edge.

3.7.24.UUCD_OPTION_MLC_LSYNC_NO_FORCE_LEND

Specifies how line ends are treated.

Name	Value	Description
MLC_LEND_MARKER	0	Line sync. marker cuts line to search new reset pulse
MLC_LEND_BYTES	1	Line synchronization marker does not force line end (line ends after byte count)

3.7.25.UUCD_OPTION_MLC_B_USE_LSYNC_A

Specifies whether channel B takes line synchronization from channel A.

Name	Value	Description
MLC_DONT_USE	0	B channel uses its own line synchronization
MLC_USE	1	B channel takes line synchronization from channel A

3.7.26.UUCD_OPTION_MLC_NO_LSYNC

Similar to UUCD_OPTION_MLC_NO_FSYNC.

Name	Value	Description
MLC_SYNC_YES	0	Require frame synchronization signal.
MLC_SYNC_NO	1	There is no frame synchronization marker.

3.7.27. UUCD_OPTION_MLC_LSYNC_FALLING_EDGE

Line synchronization on either falling or rising edge.

Name	Value	Description
MLC_SYNC_RISING_EDGE	0	Use line synchronization rising edge.
MLC_SYNC_FALLING_EDGE	1	Use line synchronization falling edge.

3.7.28. UUCD_OPTION_MLC_RESET_PULSE_PAIR

Sets differential pair which holds the mini-LVDS reset pulse (line start marker).

Value Range	Description
1 - 6	0 = Pair 1, 1 = Pair 2,..., 5 = Pair 6

3.8. Types and structures.

This section describes structures used in the UUCD API.

3.8.1. UUCD_FRAME_DESCRIPTOR

```
Typedef struct
{
int FrameWidth;
int FrameHeight;
int FormatID;
void *Buffer;
char BytesPerElement;
int TimeStamps[8];
unsigned int Flags;
} UUCD_FRAME_DESCRIPTOR;
```

Fields

FrameWidth

Width of the frame, in pixels.

FrameHeight

Height of frame, in lines.

FormatID

Identifies the pixel format.

Name	Value	Description
CFID_ARGB18	0	4 bytes per pixel, 6 bits per color channel. Color channels are aligned to most significant bits per byte.

CFID_ARGB24	1	4 bytes per pixel, 8 bits per color channel.
CFID_ARGB30	2	8 bytes per pixel, 10 bits per color channel. Color channels are aligned to most significant bits per short (16 bits).
CFID_ARGB36	3	8 bytes per pixel, 12 bits per color channel. Color channels are aligned to most significant bits per short (16 bits)

Buffer

Pointer to the buffer that holds the frame data. The size of this buffer can be calculated as follows: *FrameWidth * FrameHeight * BytesPerElement*

BytesPerElement

Indicates the size of pixel in bytes.

TimeStamps

A table of up-to eight timestamps to convey time-stamping information from each receiving FPGA chip. The time stamps values can be used to verify that each sub-frame belongs to the same final output frame.

(continued...)

(...continued)

Flags

Misc. info about the frame.

Bit	Description
0	3D Frame indicator. 0 = 2D frame format. 1 = 3D frame format.
1	3D view info. 0 = Right eye. 1 = Left eye.

Fields

nFrameWidth

Width of the next frame to be transferred, in pixels.

nFrameHeight

Height of the next frame to be transferred, in lines.

3.8.2. UUCD_FRAME_INFO

```
typedef struct
{
int nFrameWidth;
int nFrameHeight;
}UUCD_FRAME_INFO;
```

Fields

nFrameWidth

Width of the next frame to be transferred, in pixels.

nFrameHeight

Height of the next frame to be transferred, in lines.

3.8.3. UUCD_DATA_MAPPING

```
typedef struct
{
int   TableSize;
unsigned int *Offsets;
} UUCD_DATA_MAPPING;
```

Fields

TableSize

Size of the Offsets table, in 32-bit quantities.

Offsets

Frame remapping offsets table.

3.8.4. UUCD_OPERATION_MODE_DESCRIPTOR

```
typedef struct
{
UUCD_MODE         ModeID;
unsigned char     ModeGUID[16];
char              ModeDisplayName[65];
unsigned short    FW_MajorVer,
unsigned short    FW_MinorVer,
unsigned short    SW_MajorVer,
unsigned short    SW_MinorVer,
unsigned short    Purpose
} UUCD_OPERATION_MODE_DESCRIPTOR;
```

Fields

ModeID

Identifies an operation mode ID value. To select the named operation mode, open the device with this ID value as the third parameter of the open device function.

ModeGUID

Used to identify plugins from one another programmatically.

ModeDisplayName

Contains operation mode ID name. The intended use of this is to allow a user to pick an operation mode from a list of modes that have meaningful names.

FW_MajorVer, FW_MinorVer, SW_MajorVer, SW_MinorVer

These values identify which firmware version will be used, and which SW implementation version will be used.

Purpose

Specifies the purpose of the given mode. Possible options are:

- Video capture (`UUCD_MODE_CAPTURE`)
- Timing measurement (`UUCD_MODE_TIMING`)
- Video capture in miniLVDS mode (`UUCD_MODE_MLC_CAPTURE`)
- Electrical measurement (`UUCD_MODE_ELECTRICAL`)

3.8.5. UUCD_REMAP_CONFIG

```
typedef struct
{
int   FrameWidth;
int   TotalLanes;
int   Sections;
unsigned int ChannelMap[64];
} UUCD_REMAP_CONFIG;
```

Fields

FrameWidth

Width of the frame to be mapped in pixels. The width must be evenly divisible with TotalLanes value.

TotalLanes

Number of Vx1 lanes used to capture the frame to be mapped. The expected value is number of cascaded devices multiplied with the number of used lanes per device. The TotalLanes value must be evenly divisible with Sections value.

Sections

Number of Vx1 sections in the image. The value must be non-zero, and may not be higher than TotalLanes.

ChannelMap

Vx1 lane remapping. The lane-remap is performed before sections remapping. To disable channel remapping, initialize this array with values starting from 0 and stepping to 63.

3.8.6. UUCD_DEVICE_DESCRIPTOR

```
typedef struct _UUCD_DEVICE_DESCRIPTOR
{
unsigned int DeviceID;
char DeviceName[65];
char DeviceSN[17];
unsigned char revision;
int slave_count;
struct _UUCD_DEVICE_DESCRIPTOR *slave
} UUCD_DEVICE_DESCRIPTOR;
```

Notes

This structure should be considered as “read-only” for client applications: This applies especially to the slave device descriptors. Do not modify device descriptor contents.

Fields

DeviceID

Hardware device ID.

Name	Value (Hex)	Device Name
USB_MLC_DEVICE_ID	0x16A67100	UCD-1 MLC
USB_LVDS_DEVICE_ID	0x16A67110	UCD-1 LV
USB_VX1_DEVICE_ID	0x16A67200	UCD-2 Vx1
USB_DUMMY_DEVICE_ID	0x16A67300	UCD SW Emulated device.

DeviceName

Name of device in human readable form. The string is NULL terminated.

DeviceSN

Serial number of the device in human readable form. The string is NULL terminated.

Revision

Indicates device hardware revision.

slave_count

Number of additional chained devices.

**slave*

Pointer to next UUCD_DEVICE_DESCRIPTOR structure. NULL indicates end of list.

3.8.7. UUCD_TAG Structure

Contains a single TAG List item.

```
typedef struct
{
WORD  TagID;
DWORD TagData;
} UUCD_TAG;
```

Fields

TagID

Identifies the data contained in TagData field. The meaning depends on the function that uses the structure.
The following table shows tag IDs for UUCD_AnalyzeFrame():

ID	Name	Description
0x0001	TAG_SUBCH_1_ERRORS	Data is number of detected non-zero values in analyzed frame for sub-channel 1 (Red or Cr)
0x0002	TAG_SUBCH_2_ERRORS	Sub-channel 2 error count (See above) (Green or Y)
0x0003	TAG_SUBCH_3_ERRORS	Sub-channel 3 error count (See above) (Blue or Cb)
0x0004	TAG_PIXEL_ERRORS	Total number of pixels with one or more sub-channel errors in analyzed frame.
0x0005	TAG_SUBCH_1_BIT_ERRORS	Cumulative bit-pattern of detected errors on sub-channel 1 (Red or Cr)
0x0006	TAG_SUBCH_2_BIT_ERRORS	Cumulative bit-pattern of detected errors on sub-channel 2 (Green or Y)
0x0007	TAG_SUBCH_3_BIT_ERRORS	Cumulative bit-pattern of detected errors on sub-channel 3 (Blue or Cb)
0xfffe	TAG_END_LIST	Indicates end of TAG-LIST.
0xffff	TAG_INVALID	Undefined data/content.
0x*ffd	TAG_EXT*_RESERVED	Reserved for future TAG-LIST features.

The tag IDs for UUCD_GetVideoTimingData() are described below in the UUCD_GetVideoTimingData Tag IDs chapter.

TagData

32-bit Value.

3.9. UUCD_GetVideoTimingData Tag IDs

0x0000	TSTAMP	
BIT	**BIT NAME**	**FUNCTION**
31	CLK_ERROR	'1' = Clock was momentarily lost and measurement results may be garbled. Consider reset.
30	INPUT_ERROR	'1' = V-by-One link lock was lost or license error.
29	3DEN	'1' = 3D video is being transmitted '0' = 2D video is being transmitted
28	3DLR	'1'=Frame is left view '0'=Frame is right view
27:23		
22:0	PTSTAMP	Packet time stamp which can be used to check if there is packet drop.

Timestamp counts 10 us ticks and is synchronized with master and slave devices. 3D flags are read only during data enable area and taken from the lower numbered lane per FPGA. 3D flags sent during blanking interval are ignored.

0x0001	HTOTAL_SYSCLK	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID)
29:27		
26:0	HTOTAL_SYSC	Line length in system clocks (100MHz). Use to compute frame rate with HTOTAL.

0x0002	HTOTAL_SYSCLK_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HTOTAL_SYSC_MIN	Minimum value of HTOTAL_SYSC.

0x0003	HTOTAL_SYSCLK_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HTOTAL_SYSC_MAX	Maximum value of HTOTAL_SYSC.

0x0004	VIDCLK_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:16	VIDCLK_MIN	Number of video clocks in 10us (1000*system clock period 100MHz) minimum.
15:0	VIDCLK_MAX	Number of video clocks in 10us maximum. Can be used to approximately measure spread spectrum clocking.

0x0005	VS_HIGH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). V-Sync. is constantly '1'.
29:27		
26:0	VS_HIGH	Number of clocks Vertical synchronization signal is high. Unfiltered V-sync. is used.

0x0006	VS_HIGH_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	VS_HIGH_MIN	Minimum value of VS_HIGH .

0x0007	VS_HIGH_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	VS_HIGH_MAX	Maximum value of VS_HIGH .

0x0008	VS_LOW	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). V-Sync. is constantly '0'.
29:27		
26:0	VS_LOW	Number of clocks Vertical synchronization signal is low. Unfiltered V-sync. is used.

0x0009	VS_LOW_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	VS_LOW_MIN	Minimum value of VS_LOW .

0x000A	VS_LOW_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	VS_LOW_MAX	Maximum value of VS_LOW .

0x000B	HS_HIGH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). H-Sync. is constantly '1'.
29:27		
26:0	HS_HIGH	Number of clocks horizontal synchronization signal is high. Unfiltered H-sync. is used.

0x000C	HS_HIGH_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HS_HIGH_MIN	Minimum value of HS_HIGH .

0x000D	HS_HIGH_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HS_HIGH_MAX	Maximum value of HS_HIGH .

0x000E	HS_LOW	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). H-Sync. is constantly '0'.
29:27		
26:0	HS_LOW	Number of clocks horizontal synchronization signal is low. Unfiltered H-sync. is used.

0x000F	HS_LOW_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HS_LOW_MIN	Minimum value of HS_LOW .

0x0010	HS_LOW_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HS_LOW_MAX	Maximum value of HS_LOW .

0x0011	DE_HIGH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). DE is constantly '1'.
29:27		
26:0	DE_HIGH	Number of clocks data enable signal is high. Unfiltered DE is used.

0x0012	DE_HIGH_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_HIGH_MIN	Minimum value of DE_HIGH .

0x0013	DE_HIGH_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_HIGH_MAX	Maximum value of DE_HIGH .

0x0014	DE_LOW_HBLANK	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). DE is constantly '0'.
29:27		
26:0	DE_LOW_HB	Number of clocks data enable signal is low during horizontal blanking period. Unfiltered DE is used.

0x0015	DE_LOW_HBLANK_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_LOW_HB_MIN	Minimum value of DE_LOW_HB .

0x0016	DE_LOW_HBLANK_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_LOW_HB_MAX	Maximum value of DE_LOW_HB .

0x0017	DE_LOW_VBLANK	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID). DE is constantly '0'.
29:27		
26:0	DE_LOW_VB	Number of clocks data enable signal is low during vertical blanking period. Unfiltered DE is used.

0x0018	DE_LOW_VBLANK_MIN	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_LOW_VB_MIN	Minimum value of DE_LOW_VB .

0x0019	DE_LOW_VBLANK_MAX	
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	DE_LOW_VB_MAX	Maximum value of DE_LOW _VB.

0x001A		HTOTAL
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:27		
26:0	HTOTAL	Number of clocks between data enable rising edges of the 1st and 2nd data lines. Filtered DE is used.

0x001B		HTOTAL_MIN
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HTOTAL_MIN	Minimum value of HTOTAL .

0x001C		HTOTAL_MAX
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	HTOTAL_MAX	Maximum value of HTOTAL.

0x001D		FRAME_TOTAL
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:27		
26:0	FRAME_TOTAL	Number of clocks per frame. Filtered DE is used.

0x001E		FRAME_TOTAL_MIN
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	FRAME_TOTAL_MIN	Minimum value of FRAME_TOTAL.

0x001F		FRAME_TOTAL_MAX
BIT	**BIT NAME**	**FUNCTION**
31:27		
26:0	FRAME_TOTAL _MAX	Maximum value of FRAME_TOTAL.

0x0020		HBPORCH
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:16		
15:0	HBPORCH	Number of clocks between H-sync. edge and data enable rising edges of the 1st data lines. Filtered DE and HS is used.

0x0021		HBPORCH_MINMAX
BIT	**BIT NAME**	**FUNCTION**
31:16	HBPORCH_MAX	Maximum value of HBPORCH.
15:0	HBPORCH_MIN	Minimum value of HBPORCH.

0x0022		HFPORCH
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:16		
15:0	HFPORCH	Number of clocks between data enable falling edges of the 1st H-sync. rising edge. Filtered DE and HS is used.

0x0023	HFPORCH_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:16	HFPORCH_MAX	Maximum value of HFPORCH.
15:0	HFPORCH_MIN	Minimum value of HFPORCH.

0x0024	VS_DE_BPORCH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:16		
15:0	VS_DE_BPORCH	Number of clocks between V-sync. edge and data enable rising edges of the 1st data lines. Filtered DE and VS is used.

0x0025	VS_DE_BPORCH_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:16	VS_DE_BPORCH_MAX	Maximum value of VS_DE_BPORCH.
15:0	VS_DE_BPORCH_MIN	Minimum value of VS_DE_BPORCH.

0x0026	VS_DE_FPORCH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30	FULL_COUNT	'1' = Counter reached full error (used with NOT_VALID).
29:16		
15:0	VS_DE_FPORCH	Number of clocks between data enable falling edges of the V-sync. rising edge. Filtered DE and VS is used.

0x0027	VS_DE_FPORCH_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:16	VS_DE_FPORCH_MAX	Maximum value of VS_DE_FPORCH.
15:0	VS_DE_FPORCH_MIN	Minimum value of VS_DE_FPORCH.

0x0028	VSYNC_VACT	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement Note: If this bit is set BIT 30 indicates FULL COUNT.
30:16	VSYNC	Number of H-syncs when VS is active. Filtered VS and HS are used.
15	NOT_VALID	'1' = Data is not valid or there is no new measurement Note: If this bit is set BIT 14 indicates FULL COUNT.
14:0	VACT	Number of data enable rising edges . Filtered DE is used.

0x0029	VFPORCH_VBPORCH	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement Note: If this bit is set BIT 30 indicates FULL COUNT.
30:16	VFPORCH	Number of H-syncs between last data line and VS active. Filtered VS, DE and HS are used.
15	NOT_VALID	'1' = Data is not valid or there is no new measurement Note: If this bit is set BIT 14 indicates FULL COUNT.
14:0	VBPORCH	Number of H-syncs after VS active and before the first data line. Filtered VS, DE and HS are used.

0x002A	CRC_RED	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_RED	CRC value for red component. 0x00000000 indicates a not valid value.

Polynomial is CRC32: x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1.

0x002B	CRC_GREEN	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_GREEN	CRC value for green component. 0x00000000 indicates a not valid value.

Polynomial is CRC32: x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1.

0x002C	CRC_BLUE	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_BLUE	CRC value for blue component. 0x00000000 indicates a not valid value.

Polynomial is CRC32: x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1.

0x002D	CRC_COUNT_RED	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_CNT_RED	Number of frames with erroneous red CRC.

0x002E	CRC_ COUNT_GREEN	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_CNT_GREEN	Number of frames with erroneous green CRC.

0x002F	CRC_ COUNT_BLUE	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_CNT_BLUE	Number of frames with erroneous blue CRC.

0x0030	CRC_ COUNT_TOTAL	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_CNT_TOTAL	Number of frames with erroneous CRC on any component.

0x0031	CRC_ COUNT_FRAMES	
BIT	**BIT NAME**	**FUNCTION**
31:0	CRC_CNT_FRAMES	Number of CRC checked frames.

0x0032	BIT_ACTIVITY_RG	
BIT	**BIT NAME**	**FUNCTION**
31:28		
27:16	BIT_ACT_GREEN	Bit is set to '1' when corresponding bit has an edge.
15:12		
11:0	BIT_ACT_RED	Bit is set to '1' when corresponding bit has an edge.

0x0033	BIT_ACTIVITY_B	
BIT	**BIT NAME**	**FUNCTION**
31:12		
11:0	BIT_ACT_BLUE	Bit is set to '1' when corresponding bit has an edge.

0x0034	GRABBED_RG	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30:28		
27:16	GRABBED_GREEN	Data value of green in grabbing position.
15:12		
11:0	GRABBED_RED	Data value of red in grabbing position.

0x0035	GRABBED_B	
BIT	**BIT NAME**	**FUNCTION**
31	NOT_VALID	'1' = Data is not valid or there is no new measurement
30:12		
11:0	GRABBED_BLUE	Data value of blue in grabbing position.

0x0036	RED_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:28		
27:16	RED_MAX	Maximum value of red data.
15:12		
11:0	RED_MIN	Minimum value of red data.

0x0037	GREEN_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:28		
27:16	GREEN_MAX	Maximum value of green data.
15:12		
11:0	GREEN_MIN	Minimum value of green data.

0x0038	BLUE_MINMAX	
BIT	**BIT NAME**	**FUNCTION**
31:28		
27:16	BLUE_MAX	Maximum value of blue data.
15:12		
11:0	BLUE_MIN	Minimum value of blue data.

3.10. Error code reference

For complete list of error codes, please refer to “UUCD_Types.h” file.

Name	Value	Description
UUCD_ERROR_UUCD_NOT_FOUND	-1	UUCD.dll was not found.
UUCD_ERROR_UUCD_FUNC_NOT_FOUND	-2	A required function was not found inside UUCD.dll
UUCD_ERROR_DLL_VERSION	-3	UUCD.dll version info is missing, or the DLL is of unacceptable version.
UUCD_ERROR_INIT	-4	UUCD Initialization failed.
UUCD_ERROR_NOT_INITIALIZED	-5	UUCD API is not initialized.
UUCD_ERROR_NOT_SUPPORTED	-6	Requested operation is not supported.
UUCD_ERROR_INVALID_DEVICE_TYPE	-7	Device type is not recognized.
UUCD_ERROR_INVALID_PARAMETER	-8	One, or more of the parameters given to a function were invalid.
UUCD_ERROR_D2D_FAILED	-9	Direct2D preview error.

UUCD_ERROR_DDRAW_FAILED	-10	DirectDraw preview error.
UUCD_ERROR_UNSUPPORTED_COLORSPACE	-11	The frame colors pace is not supported by the function.
UUCD_ERROR_INVALID_HANDLE	-12	The handle value is not valid.
UUCD_ERROR_NO_SUCH_ALLOCATION	-13	Attempting to free a buffer that is either not allocated, or is already freed.
UUCD_ERROR_FUNCTION_NOT_FOUND	-14	USB Device Driver API function entry point not found.
UUCD_ERROR_USB_DRIVER	-15	USB Device Driver API Missing.
UUCD_ERROR_DEVICE_OPEN_FAILED	-16	USB Device could not be opened.
UUCD_ERROR_DEVICE_SETUP_FAILED	-17	USB Device configuration has failed.
UUCD_ERROR_ENUMERATION_ERROR	-18	Could not enumerate USB devices.
UUCD_ERROR_OUT_OF_MEMORY	-19	Insufficient memory or other resource.
UUCD_ERROR_INVALID_OP_MODE	-20	The requested operation mode is not available for this device type.
UUCD_ERROR_FW_LOAD_FAILED	-21	Failed to upload firmware.
UUCD_ERROR_TIMEOUT	-22	Timeout waiting for device to perform an operation.
UUCD_ERROR_USB_IO	-23	USB I/O Has failed. If this error appears, it is necessary to close and re-open the device.
UUCD_ERROR_NO_VIDEO_SIGNAL	-24	No signal detected on input.
UUCD_ERROR_FAILED_TO_CREATE_FILE	-25	Error when trying to create file. If a file-name or path was provided, make sure that the file location exists and is writable.
UUCD_ERROR_INVALID_FORMAT_ID	-26	Unsupported format ID was given.
UUCD_ERROR_FILE_WRITE_ERROR	-27	Error while writing to a file.
UUCD_ERROR_FRAME_SIZE_MISMATCH	-28	Sub-frame sizes are not equal to one another.
UUCD_ERROR_INCOMPATIBLE_BUFFERS	-29	Video frames being compared have different dimensions.
UUCD_ERROR_SYNC_CABLE	-30	Either a loose cable is connected to a sync out socket or loop in cables is detected.
UUCD_ERROR_BUFFER_EMPTY	-31	No more frames in the device buffer. Call StartCapture() again.
UUCD_ERROR_EXT_TRIGGER_TIMEOUT	-32	Timeout due to waiting for external trigger
UUCD_ERROR_VALIDATION_FAILED	-33	License validation failed