APDLib

Application Programming Interface (API) for APDCAM

Version 2.3.2

17 February, 2015

Contents

1. Introduction 3
2. Programming language and target operating system 3
3. Overview of functions 4
4. Using APDLib 5
5. The APDLib library 7
5.1 Initiating and closing APDLib 7
5.2 Searching APDCAMs in an IP address range 7
5.3 Opening and closing APDCAM 8
5.4 Self test 9
5.5 Changing the IP address of the camera 9
5.6 Setting the ADC timing 9
5.7 Setting the signal offsets 11
5.8 Calibrating the offset 11
5.9 Setting up the digital filter 12
5.10 Calculating filter parameters 12
5.11 Allocating memory for measurements 13
5.12 Setting ring buffer size 15
5.13 Preparing for the measurement 15
5.14 Setting the trigger conditions 16
5.15 The measurement 17
5.16 Shutter control 19
5.17 Calibration light control 19
5.18 Setting and monitoring the detector bias voltage 20
5.19 Setting the overload protection 21
5.20 Using ADC test pattern 22
5.21 Reading the system status 23
6. Troubleshooting 24

1. Introduction

APDCAM is an Avalanche Photodiode (APD) detector camera developed for low light and high speed applications by Fusion Instruments Kft. This document describes the Application Programming Interface (API) of APDCAM. For a detailed description of APDCAM refer to its User's Guide.

Additionally to the set of C functions the APDCAM API also contains a scriptable test program which serves as an example but can also be used to perform routine measurement tasks.

2. Programming language and target operating system

APDLib is written in C++ for Windows XP, Windows 7 and for Linux operating systems. It was tested with Microsoft Visual Studio 2010 Express (10.0.402.19.1 SP1Rel, Microsoft .NET Framework 4.0.30319 SP1Rel) and Visual Studio 2005 Express but might also be usable for other compilers. On linux side it was tested in Qt 5.4 Framework with GCC 64bit compiler.

The Windows kit contains a static library APDLib.lib, header files and the source code of the APDTest test pogram. Solution and project files for MS Visual Studio are also supplied. The source code of APDLib.lib is not made public.

The linux kit contains also a static library APDLib.a, header files and the source code of the APDTest test program. The project files for QtCreator are also supplied. The source code of APDLib.a is not made public.

For memory allocation the Lock Pages in Memory option should be enabled in Windows Control Panel. This can be done by opening “Local Security Policy” in the XP Control Panel (in Administrative Tools under Windows 7). In the “Local Policies -> User Rights Assignments” panel click “Lock pages in memory” with the right mouse button and select “Properties” from the menu. Add the user to the list who is going to run the data acquisition program and reboot the computer.

3. Overview of functions

Function	Description	Page
APDCAM_Allocate()	Allocates memory for measurement.	12
APDCAM_ARM()	Prepares the measurement.	14
APDCAM_CalibLight()	Calibration light control.	18
APDCAM_Calibrate()	Calibrates the offset.	10
APDCAM_Close()	Closes APDCAM.	7
APDCAM_DataMode()	Sets ADC test patterns instead of measured data.	21
APDCAM_Filter()	Loads difital filter parameters to APDCAM.	11
APDCAM_Find()	Scans a range of IP addresses to find APDCAM devices.	6
APDCAM_Gain()	Sets the detector bias voltage.	19
APDCAM_GetADCOffsets()	Reads the signal offsets.	10
APDCAM_GetBuffers()	Reads the addresses of allocated channel buffers.	13
APDCAM_GetCalibLight()	Reads the state of calibration light.	19
APDCAM_GetHV()	Reads the state of detector bias voltage.	19
APDCAM_GetInfo()	System status.	21
APDCAM_GetOverload()	Reads the overload setting.	20
APDCAM_GetRingbufferSize()	Reads back the size of the ring buffer.	14
APDCAM_GetSampleInfo()	Reads the number of received samples and the starting addresses of logical buffers for the four streams.	13
APDCAM_Init(), APDCAM_Done()	Functions for library initialization and closing.	6
APDCAM_Open()	Opens an APDCAM.	7
APDCAM_Sampling()	Sets the sampling rate of APDCAM.	9
APDCAM_SelfTest()	Runs a test procedure.	8
APDCAM_SetADCOffsets()	Sets the signal offsets.	10
APDCAM_SetFilterParam()	Calculates the filter coefficients.	11
APDCAM_SetIP()	Sets the IP address of APDCAM.	8
APDCAM_SetOverload()	Sets the parameters of overload protection.	20
APDCAM_SetRingbufferSize()	Sets the size of the ring buffer.	14
APDCAM_SetTiming()	Sets the timings of APDCAM ADC.	8
APDCAM_Shutter()	Shutter control.	17
APDCAM_Start()	Starts the measurement.	16
APDCAM_Stop()	Stops the measurement.	17
APDCAM_Trigger()	Sets the trigger conditions.	16
APDCAM_Wait()	Checks or waits for the completion of measurement.	16
GetShutterMode()	Reads back the state of shutter control (internal or external).	18
Measure_NonCalibrated()	Starts a noncalibrated measurement.	17
SetShutterMode()	Sets the state of shutter control (internal or external).	18

4. Using APDLib

The functions of the APDLib library can be used with a C++ program. The library must be added to this program as a reference. See APDTest.c as an example.

The other way to use the library is using the APDTest test application. The functions of APDLib can be called with a script file.

It is important to call the functions in the right order.

The following code outline demonstrates the right order of function calls:

```
/*1*/  APDCAM_Init();          //API initialization
/*2*/  APDCAM_Find(unsigned long from_ip_h, unsigned long to_ip_h, unsigned
       long *ip_table, int table_size, int *no_of_elements, char *filter_str =
       NULL, int timeout = 100); //Find device
/*3*/  APDCAM_Open(unsigned long ip_h); //Open device
/*4*/ APDCAM_SetTiming(ADT_HANDLE handle, int adcMult, int adcDiv, int strMult,
       int strDiv, int clkSorce, int clkMult, int clkDiv); //Set the timings
/*5*/  APDCAM_Sampling(ADT_HANDLE  handle,  int  sampleDiv,  int  sampleSrc);
       //Sampling
/*6*/  APDCAM_Allocate(ADT_HANDLE handle, LONGLONG sampleCount, int bits, int
       channelMask_1, int channelMask_2, int channelMask_3, int channelMask_4,
       int primary_buffer_size = 10); //Allocate memory for measurement
/*7*/    APDCAM_ARM(ADT_HANDLE  handle,  ADT_MEASUREMENT_MODE  mode,  LONGLONG
       sampleCount,  ADT_CALIB_MODE  calibMode,  int  signalFrequency  =  100);
       //Preapre the measurement
/*8*/  APDCAM_Start(ADT_HANDLE handle);  //Start the measurement
/*9*/  APDCAM_Wait(ADT_HANDLE handle, int timeout); //Wait for measurement
/*10*/ APDCAM_Close(ADT_HANDLE handle); //Close device
/*11*/ APDCAM_Done() //Close API
```

This is an example of a simple, untriggered measurement.

The script can be written in a simple text file. To run APDTest the file name of the script should be given APDTest.exe as a command line argument.

The structure of a scriptfile is the following:

```
Message *************** APD Camera demo *****************
Message  This  script  demonstrates  the  simple,  untriggered  operation  of  ADC
board.
Open 10.123.13.101
Rem SetTiming: p1=adc mult, p2=adc div, p3=stream mult, p4=stream div
SetTiming 20 40 30 10
Rem Sampling: p1=sampleDiv, p2=sampleSrc 0:internal, 1:external
Sampling 20 1
Rem  Allocate:  p1=sample  num,  p2=bit  num,  p3=channel_1  mask,  p4=channel_2
mask, p5=channel_3 mask, p6=channel_4 mask,
Allocate 1000000 14 255 255 255 255
Rem   Arm:   p1=measurement   mode   (0:one   shot   1:cyclic)   p2=sample   num,
p3=calibration mode (0:calibrated 1:non-calibrated)
Arm 0 1000000 0
Start
Wait 5000
Save
Pause
Close
```

The parameters of the commands can be given after their names, separated with spaces. Each command should start in a new line.

As can be seen there are options to write comments and messages in the script.

The following table contains the commands of APDTest for each function.

Function	APDTest call	Parameters						
Comment	Rem							
Message to the output	Message							
Wait for pressing enter	Pause							
Saving measurement data	Save							
APDCAM_Allocate()	Allocate	LONGLONG sampleCount	int bits (8, 12, 14)	int channelMask_1	int channelMask_2	int channelMask_3	int channelMask_4	int primaryBufferSize (in megabyte)
APDCAM_ARM()	Arm	int measurementMode (MM_ONE_SHOT)	int sampleCount	int calibrationMode (CM_NONCALIBRATED)	int signalFrequency			
APDCAM_CalibLight()	Caliblight	int value (0..4096)						
APDCAM_Calibrate()	Calibrate							
APDCAM_Close()	Close							
APDCAM_DataMode()	Datamode	int mode (0: data, 1-7: test patterns)						
APDCAM_Filter()	Filter	int *coeffs						
APDCAM_Gain()	Gain	double hV1	double hV2	int state (0: off, 1: on)				
APDCAM_GetADCOffsets()	Getoffsets	short values[32]						
APDCAM_GetBuffers()	Getbuffers	short **buffers						
APDCAM_GetCalibLight()	Getcaliblight	int &value						
APDCAM_GetHV()	Gethv	double &hv1	double &hv2	int &state				
APDCAM_GetInfo()	Stat	ADT_SYSTEM_STATUS *systemStatus						
APDCAM_GetOverload()	Getoverload	ADT_OVERLOADINFO overloadInfo	unsigned short overloadTime	unsigned char status				
APDCAM_GetSampleInfo()	Sampleinfo	ULONGLONG *sampleCounts	ULONGLONG *sampleIndices					
APDCAM_Open()	Open							
APDCAM_Sampling()	Sampling	int sampleDiv (1..0xFFFF)	int sampleSrc (0: int, 1: ext)					
APDCAM_SelfTest()	Selftest							
APDCAM_SetADCOffsets()	Setoffsets	short offset						
APDCAM_SetFilterParam()	Filterparam	FILTER_COEFFICIENTS fc	double f_fir (1.0..5.0)	double f_rec (0.1..50.0)				
APDCAM_SetHWTriggerDelay()	Hwtriggerdelay	int delay						
APDCAM_SetIP()	Setip	TCHAR *param0						
APDCAM_SetRingbufferSize()	Setringbuffer	unsigned short bufferSize (0..1023)						
APDCAM_SetTiming()	Settiming	int adcMult (20..50)	int adcDiv (8..100)	int strMult (20..50)	int strDiv (8..100)	int clkSrc (0: int, 1: ext)	int clkMult (2..33)	int clkDiv (1..32)
APDCAM_Shutter()	Shutter	int open (1: open, 0: close)						
APDCAM_Start()	Start							
APDCAM_Stop()	Stop							
APDCAM_Trigger()	Trigger	int triggerSource (0: TR_SOFTWARE, 1: TR_HARDWARE)	int triggerMode (0: TRM_EXTERNAL, 1: TRM_INTERNAL)	int triggerEdge (0: TRE_RISING, 1: TRE_FALLING)	int delay	ADT_TRIGGERINFO *trigger		
APDCAM_Wait()	Wait							
GetShutterMode()	Getshutter	int &state						
Measure_NonCalibrated()	M_noncalib	LONGLONG sampleCount	int signalFrequency					
SetShutterMode()	Setshutter	int mode (0: int, 1: ext)						

5. The APDLib library

5.1 Initiating and closing APDLib

The library must be initialized before use, and preferably it has to be closed for proper shut down before exiting the application. There are two functions for doing that:

Opening APDLib:
```
void APDCAM_Init();
```

Closing APDLib:
```
void APDCAM_Done();
```

5.2 Searching for APDCAMs in an IP address range

The APD Camera must be opened before use. For this its IP address has to be known. To find APDCAMs in an address range, use the APD_Find() function. This function scans a certain address range and returns the addresses of cameras in that range.

```
void APDCAM_Find(unsigned long from_ip_h, unsigned long to_ip_h, unsigned
long *ip_table, int table_size, int *no_of_elements, char *filter_str =
NULL, int timeout = 100);
```

Parameters:

unsigned long from_ip_h	The starting address of the range.
unsigned long to_ip_h	The end address of the range.
unsigned long *ip_table	The address of the resulting table. (Supplied by the caller.)
int table_size	The size of the IP_table table.
int *no_of_elements	The number of boards, found. Set by the function.
char *filter_str	Filter to find an IP in the IP table, in case of one IP address or to scan all IP addresses in the range use "*" or NULL
int timeout	Time limit in milliseconds, until the function waits for the answer from a board..

The function requires the IP addresses in an unsigned long, so-called „host” format. This format is „little-endian” on the Intel based computers. (That means, the least significant byte is stored in the lowest address.) The TCP/UDP protocol uses „big-endian” format. That is the so called „network” format. The socket functions require addresses in that format. E.g. the
```
unsigned long ip_n = inet_addr("10.123.13.101");
```
returns the address in „network” format. To convert it into „host” format use the
```
unsigned long ip_h = ntohl(ip_n);
```
function.

For TCP/UDP ports there are two formats too: „host” and „network” formats. The APDLib function parameters always use „host” format. We indicate this with the postfix _h in the parameter names. (For network format: _n)

5.3 Opening and closing APDCAM

APDCAM must be opened before and closed after using it. For that, use the

```
ADT_HANDLE APDCAM_Open(unsigned long ip_h);
ADT_RESULT APDCAM _Close(ADT_HANDLE handle);
```

functions.

Parameters:

unsigned long ip_h	The IP address of the camera.

Return value:	0 or handle.

On error, APDCAM_Open() returns 0, else it returns a handle value, which identifies the camera in the further operations.

The following code extract demonstrates the use of APDCAM_Find and APDCAM_Open.

```
int _tmain(int argc, _TCHAR* argv[])
{
      APDCAM _Init();

      unsigned long ip_n = inet_addr("10.123.13.101");
      if (ip_n == INADDR_NONE) return 0;

      unsigned long ip_h = ntohl(ip_n);

      unsigned long ip_table[32];
      int no_of_elements;
      APDCAM_Find(ip_h, ip_h+10, ip_table, 32, &no_of_elements, "*");

      if (no_of_elements > 0)
      {
            printf("%d board found,\n", no_of_elements);
            ADT_HANDLE handle = APDCAM_Open(ip_table[0]);
            if (handle != 0)
            {
                  APDCAM_Close(handle);
            }
      }

      APDCAM _Done();

      return 0;
}
```

5.4 Self test

APDCAM can run a test procedure to verify major functionality. After checking a few internal registers this function performs a short measurement with an ADC test pattern. The function uses the first test pattern of APDCAM_DataMode().

```
ADT_RESULT APDCAM_SelfTest(ADT_HANDLE handle, int mode)
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.

Return value:

ADT_OK	Self test OK.

5.5 Changing the IP address of the camera

The IP address of the camera can be changed with the

```
ADT_RESULT APDCAM_SetIP(ADT_HANDLE handle, unsigned long ip_h)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
unsigned long ip_h	IP address in host format.

Return value:

ADT_OK	IP address is successfully changed.
ADT_ERROR	Error in changing IP address.

Note: If the IP address of the camera is changed note it on the backplate label. If the camera IP address is lost it can be reset to the factory default (10.123.13.101) by switching the camera on while the depressed reset button is pressed with a pointed device. On the next switch-on the IP address and all other camera parameters are reset to factory default. After this the camera needs to be initialized to the APDinit program and the camera configuration file.

5.6 Setting the ADC timing

To set the timings of APD Camera, use the

```
ADT_RESULT APDCAM_SetTiming(ADT_HANDLE handle, int adcMult, int adcDiv, int
strMult, int strDiv, int clkSorce, int clkMult, int clkDiv);
```

function.

Parameters:

ADT_HANDLE handle
The handle value returned by the APDCAM_Open() function.

int adcMult, int adcDiv
The multiplier and divider of ADC PLL respectively. In the case of negative value, the previous settings are used.
Values: $20 \le adcMult \le 50$
$8 \le adcDiv \le 100$

int strMult, int strDiv
The multiplier and divider of Stream PLL respectively. In the case of negative value, the previous settings are used. Use strMult=30, strDiv=10 as default, change only in special circumstances.
Values: $20 \le strMult \le 50$
$8 \le strDiv \le 100$

int clkSource
Clock source. 0: internal, 1: external. In the case of negative value, the previous settings are used.

int clkMult, int clkDiv
The multiplier and divider of Clock PLL respectively. In the case of negative value, the previous settings are used.
Values: $2 \le clkMult \le 33$
$1 \le clkDiv \le 32$

Return value:
ADT_OK
Operation completed successfully.
ADT_INVALID_HANDLE_ERROR
Invalid handle value supplied.
ADT_ERROR
The PLLs can not be synthesize the requested frequencies.

For a detailed description of the APDCAM timing scheme see the APDCAM User's Guide. After the call the ADC sampling frequency will be f_{ADC}=20/adcDiv*adcMult [MHz] if internal clock source is used. For external clock 20 MHz is replaced by f_{ext}*clkMult/clkDiv.

The final sampling rate of APDCAM can be set by the

```
ADT_RESULT APDCAM_Sampling(ADT_HANDLE handle, int sampleDiv, int sampleSrc);
```

function.

Parameters:
ADT_HANDLE handle
The handle value returned by the APDCAM_Open() function.

int sampleDiv
The value of Sample Clock Divider. Only each sampleDiv/th sample will be transmitted from APDCAM.
$1 \le sampleDiv \le$ 0xFFFF

int sampleSrc	The source of sample clock. 0: internal, 1: external.

Return value:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.

5.7 Setting the signal offsets

APDCAM has a sensitive amplifier and high-gain APD detector. Such a system is prone to offset drifts as a function of temperature and bias voltage. This can be compensated for digitally by shifting the signals through a DAC. For a standard 1 MHz APDCAM offsets around 800 are reasonable values, for special versions different values should be used. The DAC value is unrelated to the measured values. The function for setting the offset is:

```
ADT_RESULT APDCAM_SetADCOffsets(ADT_HANDLE handle, short *values)
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
short *values (0...4096)	Array with 32 elements which contains the offset values.

Return value:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in setting values.

The offsets can be read back with the

```
ADT_RESULT APDCAM_GetADCOffsets(ADT_HANDLE handle, short *values)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
short *values	The offset values will be loaded to this array.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in loading values to the array.

5.8 Calibrating the offset

The APDCAM_Calibrate function performs an automatic calibration process. It closes the shutter and with successive approximation sets the ADC offsets so as the measured values are close to the lower measurement limit.

```
ADT_RESULT APDCAM_Calibrate(ADT_HANDLE handle);
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.

5.9 Setting up the digital filter

There is a 5 stage digital filter on the ADC board inside the camera, which connects directly to the output of AD converter. (The a detailed description of the digital filter see the APDCAM User's Guide.) To setup the filter parameters, use the

```
ADT_RESULT APDCAM_Filter(ADT_HANDLE handle, FILTER_COEFFICIENTS
filterCoefficients)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
filterCoefficients	Filter parameters (structure), see below.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in setting filter parameters.

The filterCoefficients structure has the following format:

```
typedef struct FILTER_COEFFICIENTS
{
      short FIR[5];
      short RecursiveFilter;
      short Reserved;
      short FilterDevideFactor;
};
```

5.10 Calculating filter parameters

The following function calculates the filter coefficients:

```
ADT_RESULT APDCAM_SetFilterParam(ADT_HANDLE handle, double f_fir, double
f_rec, FILTER_COEFFICIENTS &filtCoeffs)
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
double f_fir	Frequency of the FIR filter[MHz]. $1.0 \leq f\_fir \leq 5.0$
double f_rec	Frequency of the recursive filter[MHz]. $0.1 \leq f\_rec \leq 50.0$

&filtCoeffs	The function loads the calculated values to this structure.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.

The returned structure can be used to setup the filter with the APDCAM_Filter function.

It is important to use the previous two functions *after* setting the timings, because adcDiv and adcMult are necessary to calculate the parameters of the filters.

This function can handle only the most common digital filter configurations. A finite number of FIR filter frequencies are used (0.1, 0.2, 0.3, 0.5, 1.0), the input filter frequency will be rounded to the closest value in the table. The user is free to calculate his own filter coefficients and load them using the APDCAM_Filter function.

5.11 Allocating memory for measurements

Before starting measurement, you have to allocate memory for storing data. First you have to be familiar with the working of measurement.

The APD Camera sends the required quantity of data when the measurement starts, or the trigger event occurs.

To Allocate memory use the

```
ADT_RESULT APDCAM _Allocate(ADT_HANDLE handle, LONGLONG sampleCount, int
bits, int channelMask_1, int channelMask_2, int channelMask_3, int
channelMask_4, int primary_buffer_size = 10);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
LONGLONG sampleCount	The size of requested memory per channels, measured in samples. In the case of negative value, the system uses the value, previously set. (1)
int bits	Resolution. Possible values 8,12,14 bit. In the case of negative value, the system uses the value, previously set. (2)
int channelMask_1	Enable bits of 1st stream (1-8 channels). Channel 1 is the bit 0, etc. In the case of negative value, the system uses the value, previously set. (2)
int channelMask_2	Enable bits of 2nd stream (9-16 channels). Channel 9 is the bit 0, etc. In the case of negative value, the system uses the value, previously set. (2)
int channelMask_3	Enable bits of 3rd stream (17-24 channels). Channel 17 is the bit 0, etc. In the case of negative value, the system uses the value, previously set. (2)
int channelMask_4	Enable bits of 4th stream (25-32 channels). Channel 25 is the bit 0, etc. In the case of negative value, the system uses the value, previously set. (2)
int primary_buffer_size	The size of primary buffer, in MB.

Notes:
(1) The values are preserved on the board, until the power supply is on. Switching of the power, the values are lost.
(2) The values are preserved, independently of the power supply switch on/off.

During the period of data acquisition, the data are stored in a primary buffer. A thread evaluates the data to user-friendly format, and stores them in the channel buffers. The last parameter determines the size of the primary buffer, in Megabyte. The default value is 10 MB. The primary buffer size can be increased if data loss occurs.

Return values:

ADT_OK	The memory allocation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	Parameter error.
ADT_ERROR	The memory can not be allocated for any reason.

The addresses of the allocated channel buffers can be obtained with the

```
ADT_RESULT APDCAM_GetBuffers(ADT_HANDLE handle, short **buffers);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
short **buffers	Address of a pointer array. The function fills it with the pointers to the channel buffers. Take care of the size of array, because the function does not check it.

Return values:

ADT_OK	Operation completed successfully.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.

If you need the index of oldest data in the buffer (this is the beginning of logical buffer.), use the

```
ADT_RESULT APDCAM_GetSampleInfo(ADT_HANDLE handle, ULONGLONG *sampleCounts,
ULONGLONG *sampleIndices);
```

function, which returns the whole number of received data, and the starting address of logical buffers for the four stream. In normal case the four values are identical.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ULONGLONG *sampleCount	Four element ULONGLONG array, to return the whole number of received data for each stream.
ULONGLONG *sampleIndices	Four element ULONGLONG array, to return the indices of oldest data in the channel buffer for each stream.

Return values:

ADT_OK	Operation succeeded.

ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	Parameter error.

5.12 Setting ring buffer size

To set the size of the ring buffer use the

```
ADT_RESULT APDCAM_SetRingbufferSize(ADT_HANDLE handle, unsigned short
bufferSize)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
unsigned short bufferSize	Value of the buffer size (0..1023).

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	Parameter error.
ADT_ERROR	Setting ring buffer size error.

To read back the size of the ring buffer use the

```
ADT_RESULT APDCAM_GetRingbufferSize(ADT_HANDLE handle, unsigned short
*bufferSize)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
unsigned short *bufferSize	Unsigned short for the value of the ring buffer size.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Reading back ring buffer size failed.

5.13 Preparing for the measurement

To prepare for the measurement, use the

```
ADT_RESULT   APDCAM_ARM(ADT_HANDLE   handle,   ADT_MEASUREMENT_MODE   mode,
LONGLONG sampleCount, ADT_CALIB_MODE calibMode, int signalFrequency = 100);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ADT_MEASUREMENT_MODE mode	Measure certain number of samples or continuous measurement. Its values: MM_ONE_SHOT or MM_CYCLIC respectively. (1)
LONGLONG sampleCount	The number of required samples.
ADT_CALIB_MODE calibMode	Does nothing in the ADC board. Always use CM_NONCALIBRATED
int signalFrequency	The frequency data block processing. default is 100. (2)

Notes:

(1) At present “MM_CYCLIC” mode is not implemented, only “MM_ONE_SHOT” is available.

(2) As earlier mentioned, there is a thread, which evaluates primary data and converts them to user friendly form. The thread processes data in blocks. The signal frequency tells the evaluations thread the number of blocks, processed at a time. It default value is 100.

Return values:

ADT_OK	Operation completed successfully.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	The data receiver could not be started.

The function deletes all trigger conditions. If you want to use any trigger mechanism, you have to set it with the APDCAM_Trigger function.

5.14 Setting the trigger conditions

The APDCAM ADC is running all the time, trigger events start the data transmission to the computer. There are two functions to set the trigger behavior. With the first you can set the trigger conditions:

```
ADT_RESULT APDCAM_Trigger(ADT_HANDLE handle, ADT_TRIGGER trigger,
ADT_TRIGGER_MODE mode, ADT_TRIGGER_EDGE edge, int triggerDelay,
ADT_TRIGGERINFO* triggerInfo);
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ADT_TRIGGER trigger	The trigger can be TR_SOFTWARE: software, or TR_HARDWARE hardware.
ADT_TRIGGER_MODE mode	In the case of HW trigger: external or internal. TRM_EXTERNAL: external, TRM_INTERNAL: internal.
ADT_TRIGGER_EDGE edge	In the case of external HW trigger the acquisition can be started by the rising or falling edge of trigger signal. Possible values: TRE_RISING, TRE_FALLING
int triggerDelay	The data transmission starts after the trigger event with the amount of delay time expressed in units of the base clock period. For internal clock the base clock

	frequency is 20 MHz. For external clock the external clock PLL settings and the clock period determine the base clock: f_{ext}/ext_div*ext_mul
ADT_TRIGGERINFO *triggerInfo	In the case of internal HW or SW trigger, the trigger conditions for each channel. (1)

Return value:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_SETUP_ERROR	The function is called before APDCAM_ARM(), or error in trigger setup.

Notes:

(1) triggerinfo is a 32 elements array of type ADT_TRIGGERINFO, one element for each channel. An element sets the trigger condition for the corresponding channel. There are three members in the ADT_TRIGGERINFO structure:

TriggerLevel	Trigger level. Its value range depends on the resolution. It can be: 0..255 (for 8 bit), 0..4095 (for 12 bit), 0..16383 (for 14 bit)
Sensitivity	Sensitivity. 0: the trigger event occurs when the measured value is below the trigger level. 1: the trigger event is when the measured value is above the trigger level. This trigger is not edge but level controlled.
Enable	Enable trigger event in the corresponding channel.

APDCAM data transmission starts when any of the trigger events occur.

5.15 The measurement

When the system is ready to start the measurement (conditions are set, memory is allocated, APDCAM_ARM() returned successfully and the trigger conditions are set – if necessary), use the

```
ADT_RESULT APDCAM_Start(ADT_HANDLE handle);
```

function. The function returns immediately. This does not mean, that the acquisition is finished. The data acquisition runs in the background, and you can check its state with the ADC_Wait() function, described later.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.

Return values:

ADT_OK	The data acquisition started successfully.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.

The completion of measurement can be checked (or wait for) with the

```
ADT_RESULT APDCAM_Wait(ADT_HANDLE handle, int timeout);
```

function. If the measurement completed during the timeout interval, the function stops the measurement.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int timeout	Waiting time in milliseconds. –1 means infinite waiting time. 0 value can be used to check the state of measurement.

Return values:

ADT_OK	The measurement completed.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_TIMEOUT	The measurement has not been completed yet.
ADT_ERROR	The measurement has not been started, or other error occurred.

The measurement can be stopped with the

```
ADT_RESULT APDCAM_Stop(ADT_HANDLE handle);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	The measurement has not been started, or other error occurred.

The following function starts a noncalibrated measurement:

```
ADT_RESULT Measure_NonCalibrated(ADT_HANDLE handle, LONGLONG sampleCount, int signalFrequency)
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
LONGLONG sampleCount	Number of samples.
int signalFrequency	Signal frequency.

Return values:

ADT_OK	Operation succeeded.

The function uses the APDCAM_ARM(), APDCAM_Start() and APDCAM_Wait() functions, so the noncalibrated measurement can return with the return values of these.

5.16 Shutter control

The shutter can be opened and closed with the

```
ADT_RESULT APDCAM_Shutter(ADT_HANDLE handle, int open);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int open	1 opens, 0 closes the shutter.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	int open is not 0 or 1.
ADT_ERROR	External shutter control is set or error in open/close.

Shutter control can be external or internal, this option can be set with the

```
ADT_RESULT SetShutterMode(ADT_HANDLE handle, int mode)
```

function and can be read back with the

```
ADT_RESULT GetShutterMode(ADT_HANDLE handle, int *mode)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int mode	SetShutterMode(): 0: internal, 1: external. GetShutterMode(): Integer where the current setting will be loaded to.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	int mode is not 0 or 1 in case of SetShutterMode().
ADT_ERROR	Error in set or get shutter mode.

5.17 Calibration light control

There is a calibration LED in the APD camera, which can be used to check detector. The current of the LED is controlled with the

```
ADT_RESULT APDCAM_CalibLight(ADT_HANDLE handle, int value);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int value	Its value can be in the 0-4095 range. 0 switches off the LED.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	int value is not between 0 and 4095.
ADT_ERROR	Error in calibration light setting.

The value of the calibration light can be read back with the

```
ADT_RESULT APDCAM_GetCalibLight(ADT_HANDLE handle, int *value);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int *value	Integer where the current setting will be loaded to.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in reading settings.

5.18 Setting and monitoring the detector bias voltage

```
ADT_RESULT APDCAM_Gain(ADT_HANDLE handle, double highVoltage1, double
highVoltage2, int state);
```

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
double highVoltage1	1st detector bias voltage in volts. Should be set between 200 and 400 V.
double highVoltage2	2nd detector bias voltage in volts. (Not used in standard APDCAM).
int state	0 switches off, 1 switches on the detector bias voltage.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	One or more parameters are not correct.
ADT_FACTORY_SETUP_ERROR	Empty factory table.
ADT_ERROR	Error in setting voltages and/or state.

The output of high voltage generators are connected to internal ADCs, which are calibrated in the factory. So, the real voltages can be read back from the camera. To get that voltages, use the

```
ADT_RESULT APDCAM_GetHV(ADT_HANDLE handle, double &highVoltage1, double
&highVoltage2, int &state)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
double &highVoltage1, double &highVoltage2	These double variables will contain the voltages.
int &state	Integer for getting the state of bias voltage.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_FACTORY_SETUP_ERROR	Empty factory table.
ADT_ERROR	Error in reading voltages and/or state.

5.19 Setting the overload protection

The overload protection circuit switches off the detector bias voltage when the signal is above a certain level for a certain time. The parameters can be set with the

```
ADT_RESULT   APDCAM_SetOverload(ADT_HANDLE   handle,   ADT_OVERLOADINFO
overloadInfo, unsigned short overloadTime)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ADT_OVERLOADINFO overloadInfo	Structure for the overload parameters.
unsigned short overloadTime	Overload protection time.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in setting parameters and/or time.

overloadInfo:

```
typedef union ADT_OVERLOADINFO
{
      unsigned short OverloadInfo;
      struct
      {
            unsigned short level : 14;
            unsigned short polarity : 1;
            unsigned short enable : 1;
      };
};
```

The overload setting can be read back with the

```
ADT_RESULT   APDCAM_GetOverload(ADT_HANDLE   handle,   ADT_OVERLOADINFO
&overloadInfo, unsigned short &overloadTime, unsigned char &status)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ADT_OVERLOADINFO &overloadInfo	Structure for the overload parameters.
unsigned short &overloadTime	Unsigned short for overload time.
unsigned char &status	Unsigned char for status. 0: overload off, 1: overload on

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in reading overload protection settings.

5.20 Using ADC test pattern

The APD Camera can transmit test sequences, instead of measured data. These can be used to check the data transmission. You can select between data and various test sequences with the

```
ADT_RESULT APDCAM_DataMode(ADT_HANDLE handle, int modeCode)
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
int modeCode	Data or test sequence identifier.

The possible values of modeCode:

0	Measurement data
1	10 0000 0000 0000
2	11 1111 1111 1111
3	00 0000 0000 0000
4	10 1010 1010 1010, 01 0101 0101 0101
5	Long pseudo-random. (l. ITU-T 0.150 (05/96) standard
6	Short pseudo-random. (l. ITU-T 0.150 (05/96) standard
7	11 1111 1111 1111, 00 0000 0000 0000

From the above 14 bit patterns APDCAM sends only the upper 8, 12 or 14 bits depending on the bit resolution selected. The values returned in the measurement are calculated from the above numbers (v) the following way:

- For 8 bit resolution: 256-v
- For 12 bit resolution: 4096-v
- For 14 bit resolution: 16384-v

It has to be noted that for the above values uncalibrated data should be measured. For the pseudo random patterns only the samples selected by sampleDiv are transmitted.

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_PARAMETER_ERROR	int modeCode is not between 0 and 7
ADT_ERROR	Error in setting test sequence.

5.21 Reading the system status

The various system status can be retrieved by the

```
ADT_RESULT      APDCAM_GetInfo(ADT_HANDLE     handle,     ADT_SYSTEM_STATUS
*systemStatus);
```

function.

Parameters:

ADT_HANDLE handle	The handle value returned by the APDCAM_Open() function.
ADT_SYSTEM_STATUS *systemStatus	Address of an ADT_SYSTEM_STATUS structure. (At this time the 16 temperatures are in the system status.)

Return values:

ADT_OK	Operation succeeded.
ADT_INVALID_HANDLE_ERROR	Invalid handle value supplied.
ADT_ERROR	Error in reading status.

The ADT_SYSTEM_STATUS structure:

```
typedef struct ADT_SYSTEM_STATUS
{
      unsigned short Firmaware;
      int   HVSate;
      double HighVoltages[4];
      double Temperatures[16];
      double PeltierOutputVoltage;
      unsigned char ErrorCode;
      unsigned char ShutterState;
      unsigned short CalibrationLigth;
};
```

Parameters:

unsigned short Firmaware	Firmware version.
int HVSate	Bias voltage state.
double HighVoltages[4]	Bias voltages.
double Temperatures[16]	Temperatures.
double PeltierOutputVoltage	Peltier cooler/heater voltage.
unsigned char ErrorCode	Error code.
unsigned char ShutterState	Shutter state.
unsigned short CalibrationLigth	Calibration light state.

6. Troubleshooting

This section contains some tips what to do when problems occur.

Continuity counter error
This error is indicated by APDTest when missing UDP packages are detected. There are various possible reasons for this:

- Slow or poor network connection. When using high data rates connect a APDCAM and the PC directly using a good quality UTP network cable.
- Too high data rate is set. The maximum sampling rate for 32 channels and 14 bits is 2 MHz. Exceeding this limit will result in data loss on the network connection. If some channels are disabled or the bit resolution is decreased the sampling rate can be proportionally increased.
- PC activity. If too many additional tasks are running on the computer it might result in data loss.
- Poor PC performance. Use modern PCs, quad-core processors are recommended although smaller versions might also work.
- Power save mode is set. Especially on laptops during battery operation the processor speed is limited and/or PCI bus is switched off after some inactivity. Set the minimum processor power at 100% and disable PCI bus switch off.

Missing data (0 values)
If data contains 0 from some point it indicates continuity counter error, see above

Oscillating data or data at minimum/maximum.
The detector HV is not set. The sensitive amplifiers of APDCAM require a minimum HV to be set. Depending on amplifier parameters the minimum HV is typically 150-200 V.

No data but camera communication OK
Check Firewall settings on the PC. Firewalls can detect camera register communication as originating from the PC and will allow it, but they often treat data transmission (which is sent to another port) as attack and throw away communication data without warning.

Program chashes at the allocate call.
Locking memory in physical space is not enabled for the user. See section 2 for setting this feature.