



CRONO

Single Photon Timing Camera

Version 1.01

Software Development Kit Manual

May, 2026

Contents

1 CRONO Software Development Kit (CRONO-SDK)	1
2 Module Index	2
2.1 Modules	2
3 File Index	3
3.1 File List	3
4 Module Documentation	4
4.1 Crono SDK defines	4
4.1.1 Detailed Description	4
4.1.2 Macro Definition Documentation	4
4.1.2.1 MAX_DATETIME_LENGTH	4
4.1.2.2 MAX_PATTERN_LENGTH	4
4.1.2.3 MAX_SERIAL_LENGTH	5
4.1.2.4 MIN_GATE_OFF	5
4.1.2.5 MIN_GATE_ON	5
4.2 Crono SDK custom Types	5
4.2.1 Detailed Description	6
4.2.2 Data Structure Documentation	6
4.2.2.1 struct CRONO_DeviceInfo	6
4.2.2.2 struct CRONO_AcquisitionInfo	6
4.2.2.3 struct CRONO_AcquisitionStatus	7
4.2.2.4 struct CRONO_CameraSettings	7
4.2.3 Typedef Documentation	8
4.2.3.1 CRONO_H	9
4.2.3.2 cronoBool	9
4.2.4 Enumeration Type Documentation	9
4.2.4.1 CRONO_AuxInputMode	9
4.2.4.2 CRONO_PixelMode	9
4.2.4.3 CRONO_Return	10
4.2.4.4 CRONO_SensorType	11
4.3 Constructor, destructor	11
4.3.1 Detailed Description	11
4.3.2 Function Documentation	11
4.3.2.1 CRONO_Constr()	11
4.3.2.2 CRONO_Destr()	12
4.4 Crono device access functions.	12
4.4.1 Detailed Description	12
4.4.2 Function Documentation	12
4.4.2.1 CRONO_GetDeviceCount()	13
4.4.2.2 CRONO_GetDeviceInfo()	13
4.4.2.3 CRONO_GetDeviceSerial()	14
4.4.2.4 CRONO_OpenDeviceAsDataConverter()	14
4.4.2.5 CRONO_OpenDeviceBySerial()	15
4.5 Crono device configuration functions.	15

4.5.1 Detailed Description	15
4.5.2 Function Documentation	16
4.5.2.1 CRONO_ConfigAcquisition()	16
4.5.2.2 CRONO_ConfigSensor()	17
4.5.2.3 CRONO_GetActivePixels()	17
4.5.2.4 CRONO_GetCameraSettings()	18
4.5.2.5 CRONO_GetSensorType()	18
4.5.2.6 CRONO_LoadCameraSettings()	19
4.5.2.7 CRONO_LoadSettingsFromFile()	19
4.5.2.8 CRONO_SaveSettingsToFile()	19
4.5.2.9 CRONO_SetSpadDeadTime()	20
4.6 Crono device data retrieval functions.	20
4.6.1 Detailed Description	21
4.6.2 Function Documentation	21
4.6.2.1 CRONO_GetAcquisitionInfo()	21
4.6.2.2 CRONO_GetAcquisitionStatus()	22
4.6.2.3 CRONO_GetFramesData()	22
4.6.2.4 CRONO_GetHistogramData()	23
4.6.2.5 CRONO_GetLiveData()	23
4.6.2.6 CRONO_GetTimeTagsData()	24
4.6.2.7 CRONO_StartAcquisition()	24
4.6.2.8 CRONO_StopAcquisition()	25
4.7 Crono device control signal management functions.	26
4.7.1 Detailed Description	26
4.7.2 Function Documentation	26
4.7.2.1 CRONO_ConfigAuxiliaryInputs()	26
4.7.2.2 CRONO_ConfigFrameStartExternal()	27
4.7.2.3 CRONO_ConfigFrameStartInternal()	28
4.7.2.4 CRONO_ConfigGateExternal()	28
4.7.2.5 CRONO_ConfigGateInternal()	29
4.7.2.6 CRONO_ConfigLaserSync()	29
4.7.2.7 CRONO_ConfigOutputs()	30
4.7.2.8 CRONO_ConfigTriggerExternal()	30
4.7.2.9 CRONO_ConfigTriggerInternal()	31
4.7.2.10 CRONO_GetInputFrequencies()	31
5 File Documentation	33
5.1 CRONO_SDK.h File Reference	33
5.1.1 Detailed Description	35
5.2 CRONO_SDK.h	35
6 Example Documentation	39
6.1 CRONO_SDK_Example.c	39
Index	48

Chapter 1

CRONO Software Development Kit (CRONO-SDK)

The CRONO is a single photon counting and timing camera based on a high-performance imaging array of independent smart pixels. Each pixel comprises a single-photon avalanche diode (SPAD) detector, a dedicated Active Quenching Circuit (AQC), an 8-bit counter, a Time to Digital Converter (TDC) with 312.5 ps bin width and column/row readout electronics. This on-chip integrated device provides single-photon sensitivity, high electronic noise immunity, and rapid readout speed. Currently two imaging arrays are available: a 2-D imager of 64 pixels arranged in 8 rows by 8 columns, and a linear imager of 128 pixels.

IMPORTANT In order to execute a program which links to the SDK libraries, a set of DLL should be placed in the same directory as the executable. The list of the required files is:

CRONO_SDK.dll	Software development kit interface
okFrontPanel.dll	Low-level interface

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Crono SDK defines	4
Crono SDK custom Types	5
Constructor, destructor	11
Crono device access functions.	12
Crono device configuration functions.	15
Crono device data retrieval functions.	20
Crono device control signal management functions.	26

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

CRONO_SDK.h	
CRONO software development kit	33

Chapter 4

Module Documentation

4.1 Crono SDK defines

Macros

- #define `MAX_PATTERN_LENGTH` 8192
- #define `MIN_GATE_OFF` 12
- #define `MIN_GATE_ON` 5
- #define `MAX_SERIAL_LENGTH` 20
- #define `MAX_DATETIME_LENGTH` 100

4.1.1 Detailed Description

Constant defines used by the SDK.

4.1.2 Macro Definition Documentation

4.1.2.1 `MAX_DATETIME_LENGTH`

```
#define MAX_DATETIME_LENGTH 100
```

Maximum date and time string length.

4.1.2.2 `MAX_PATTERN_LENGTH`

```
#define MAX_PATTERN_LENGTH 8192
```

Maximum pattern length in ns.

4.1.2.3 MAX_SERIAL_LENGTH

```
#define MAX_SERIAL_LENGTH 20
```

Maximum serial number string length.

Examples

[CRONO_SDK_Example.c](#).

4.1.2.4 MIN_GATE_OFF

```
#define MIN_GATE_OFF 12
```

Minimum gate off period in a pattern in ns.

4.1.2.5 MIN_GATE_ON

```
#define MIN_GATE_ON 5
```

Minimum gate on period in a pattern in ns.

4.2 Crono SDK custom Types

Data Structures

- struct [CRONO_DeviceInfo](#)
- struct [CRONO_AcquisitionInfo](#)
- struct [CRONO_AcquisitionStatus](#)
- struct [CRONO_CameraSettings](#)

Typedefs

- typedef uint8_t [cronoBool](#)
- typedef struct Crono * [CRONO_H](#)

Enumerations

- enum `CRONO_Return` {
`NO_ERR` = 0 , `DEVICE_NOT_FOUND` = 1 , `LOGIC_VERSION_FAIL` = 2 , `DEVICE_NOT_OPEN` = 3 ,
`DEVICE_ALREADY_OPEN` = 4 , `DEVICE_OPEN_AS_CONVERTER` = 5 , `INVALID_DATA_FILE` = 6 ,
`SENSOR_FAILED` = 7 ,
`POWER_FAILED` = 8 , `INVALID_PIXEL_NUMBER` = 9 , `INVALID_DEAD_TIME` = 10 , `INVALID_HALVES_CONFIG`
= 11 ,
`INVALID_START_RATE` = 12 , `INVALID_DIVISION_FACTOR` = 13 , `INVALID_FRAME_BURST_COUNT` =
14 , `INVALID_GATE_LENGTH` = 15 ,
`INVALID_LASER_SYNC_LENGTH` = 16 , `INVALID_CONFIG` = 17 , `INVALID_THRESHOLD` = 18 ,
`NO_SUCH_FILE_OR_DIRECTORY` = 19 ,
`ACQ_ONGOING` = 20 , `ACQ_EXT_TRIG_TIMEOUT` = 21 , `RAM_FULL` = 22 , `NULL_POINTER` = 23 ,
`PATTERN_TOO_LONG` = 24 , `INVALID_CHANNEL` = 25 , `INVALID_ENUM` = 26 , `FPGA_FAILED` = -1 ,
`FPGA_TIMEOUT` = -2 }
- enum `CRONO_SensorType` { `CRONO_8X8` = 0 , `CRONO_128X1` = 1 }
- enum `CRONO_PixelMode` { `TIMING` = 0 , `COUNTING` = 1 }
- enum `CRONO_AuxInputMode` { `OFF` = 0 , `RISING_EDGE` = 1 , `FALLING_EDGE` = 2 , `BOTH_EDGES` = 3 }

4.2.1 Detailed Description

Custom types used by the SDK.

4.2.2 Data Structure Documentation

4.2.2.1 struct `CRONO_DeviceInfo`

Struct containing information about the version and connection speed grade of a connected Crono device.

Examples

[CRONO_SDK_Example.c](#).

Data Fields

int	FwMajorVersion	Firmware major version.
int	FwMinorVersion	Firmware minor version.
int	FwPatchVersion	Firmware patch version.
int	SdkMajorVersion	SDK major version.
int	SdkMinorVersion	SDK minor version.
int	SdkPatchVersion	SDK patch version.
CRONO_SensorType	SensorType	Sensor type, 128x1 or 8x8.
char	Serial[MAX_SERIAL_LENGTH]	Serial number of the CRONO camera.
int	UsbSpeed	Speed grade of the USB connection.

4.2.2.2 struct `CRONO_AcquisitionInfo`

Struct containing information about the acquisition.

Examples

[CRONO_SDK_Example.c.](#)

Data Fields

uint64_t	acquiredFramesCount	Number of acquired frames.
char	DateTime[MAX_DATETIME_LENGTH]	Date and time of acquisition.

4.2.2.3 struct CRONO_AcquisitionStatus

Struct containing the status of an ongoing Crono acquisition.

Examples

[CRONO_SDK_Example.c.](#)

Data Fields

double	acquisitionTime	Elapsed time in seconds since acquisition start.
int64_t	auxBitACount	Count of bit A samples>.
int64_t	auxBitBCount	Count of bit B samples>.
int64_t	availableFramesCount	Number of currently buffered frames.
int64_t	availableHistogramCount	Number of currently buffered histograms.
int64_t	availableRawCount	Number of currently buffered raw data values.
int64_t	availableTimeTagsCount	Number of currently buffered time tags values.
int64_t	capturedFrameCount	Number of frames captured by the camera since acquisition start.
int64_t	decodedFrameCount	Number of frames decoded by SDK since acquisition start.
cronoBool	deviceRamFull	Boolean value that indicates whether the Crono device RAM is full.
int64_t	deviceRamUsageBytes	Current usage of Crono device RAM expressed in bytes.
int64_t	downloadedBytes	Number of downloaded bytes on the host computer since acquisition start.
int64_t	hostAllocatedBytes	Current ram usage for storing acquired and decoded data on the host computer.
cronoBool	ongoing	Boolean value that indicates whether the acquisition is ongoing.

4.2.2.4 struct CRONO_CameraSettings

Struct containing camera settings.

Examples

[CRONO_SDK_Example.c.](#)

Data Fields

uint64_t	acqMaxFrames	Total number of frames to be acquired. A value of 0 means no limit.
----------	--------------	--

Data Fields

cronoBool	acquireOnExtTrigger	True if acquisition is controller by external trigger.
int	activePixels	Number of active pixels.
CRONO_AuxInputMode	AuxInputASamplingMode	Aux A (Gate input) sampling mode.
CRONO_AuxInputMode	AuxInputBSamplingMode	Aux B (Trigger input) sampling mode.
int	chan1Selection	Signal selection for SMA output 1. See CRONO_ConfigOutputs() for selection table.
int	chan2Selection	Signal selection for SMA output 2. See CRONO_ConfigOutputs() for selection table.
int	chan3Selection	Signal selection for SMA output 3. See CRONO_ConfigOutputs() for selection table.
CRONO_PixelMode	countingMode	Camera operation mode.
int	deadTimeNs	SPAD deadtime in ns.
cronoBool	externalGateInvert	True if external start is inverted.
double	externalGateThreshold	Threshold for external gate signal.
int	externalStartDivisionFactor	Division factor for external start.
cronoBool	externalStartPosedge	True if external start is active on rising edge.
double	externalStartThreshold	Threshold for external start signal.
int	externalTriggerBurstCount	Number of frames to be acquired for each trigger pulse.
double	frameRate	Start frequency (actual if internal, maximum expected if external).
cronoBool	gateIsExternal	True if gate signal is externally generated.
cronoBool	half1Enable	True if 1st half of imager is enabled. Meaningfull only for 128x1 version.
cronoBool	half2Enable	True if 2nd half of imager is enabled. Meaningfull only for 128x1 version.
int	integPeriodFrames	Number of frames to be integrated for histogram or frame accumulation.
uint16_t	internalGatePattern	Pattern of internal gate, as a sequence of OFF and ON periods in ns.
int	internalGatePatternLenght	Number of elements in the gate pattern.
uint16_t	laserSyncPattern	Pattern of laser sync signal, as a sequence of OFF and ON periods in ns.
int	laserSyncPatternLenght	Number of elements in the gate sync pattern.
cronoBool	startIsExternal	True if start signal is externally generated.

4.2.3 Typedef Documentation

4.2.3.1 CRONO_H

```
typedef struct Crono* CRONO_H
```

Type CRONO_H.

Handle to the CRONO structure, to be passed to all SDK functions.

4.2.3.2 cronoBool

```
typedef uint8_t cronoBool
```

Type cronoBool.

Boolean type (8 bit)

4.2.4 Enumeration Type Documentation

4.2.4.1 CRONO_AuxInputMode

```
enum CRONO_AuxInputMode
```

Crono auxiliary inputs sampling mode enumeration.

Auxiliary inputs sampling mode: none, rising edge, falling edge, both edges.

Enumerator

OFF	Do not sample the auxiliary input.
RISING_EDGE	Sample the auxiliary input on rising edge.
FALLING_EDGE	Sample the auxiliary input on falling edge.
BOTH_EDGES	Sample the auxiliary input on both rising and falling edges.

4.2.4.2 CRONO_PixelMode

```
enum CRONO_PixelMode
```

Crono pixel mode enumeration.

CRONO operation mode, photon timing or photon counting.

Enumerator

TIMING	Event timing mode. In this mode the arrival time of each photon with respect to the start signal is measured.
COUNTING	Event counting mode. In this mode the number of detected photons in each frames is accumulated, like in a conventional camera.

4.2.4.3 CRONO_Return

enum [CRONO_Return](#)

Error table.

Error codes returned by the Crono SDK functions.

Enumerator

NO_ERR	The function returned successfully.
DEVICE_NOT_FOUND	No Crono device was found.
LOGIC_VERSION_FAIL	The version of the camera firmware is not compatible.
DEVICE_NOT_OPEN	There is no open device.
DEVICE_ALREADY_OPEN	There is already an open device.
DEVICE_OPEN_AS_CONVERTER	The device is open as data converter.
INVALID_DATA_FILE	Data file format is not valid.
SENSOR_FAILED	Hardware error. Contact MPD.
POWER_FAILED	Hardware error. Contact MPD.
INVALID_PIXEL_NUMBER	Pixel number out of bound.
INVALID_DEAD_TIME	Deadtime out of bound.
INVALID_HALVES_CONFIG	Halves configuration is not valid. At least one half must be enabled.
INVALID_START_RATE	Start rate out of bound.
INVALID_DIVISION_FACTOR	Division factor out of bound.
INVALID_FRAME_BURST_COUNT	Frame burst value out of bound.
INVALID_GATE_LENGTH	Gate length out of bound.
INVALID_LASER_SYNC_LENGTH	Laser sync length out of bound.
INVALID_CONFIG	Configuration is not valid.
INVALID_THRESHOLD	Threshold out of bound.
NO_SUCH_FILE_OR_DIRECTORY	File not found.
ACQ_ONGOING	Acquisition is ongoing.
ACQ_EXT_TRIG_TIMEOUT	Timeout waiting for external trigger.
RAM_FULL	Camera RAM is full.
NULL_POINTER	A NULL pointer was passed to the function.
PATTERN_TOO_LONG	Pattern too long.
INVALID_CHANNEL	Channel out of bound.
INVALID_ENUM	Enum out of bound.
FPGA_FAILED	Camera not responding. Please check camera power supply and data connections. If error persists, Contact MPD.
FPGA_TIMEOUT	Communication timeout. Please check camera power supply and data connections. If using external FrameStart, do not remove the signal for more than 10s during an acquisition.

4.2.4.4 CRONO_SensorType

enum [CRONO_SensorType](#)

Crono sensor types enumeration.

Enumerator

CRONO_8X8	8x8 pixels sensor
CRONO_128X1	128x1 pixels sensor

4.3 Constructor, destructor

Functions

- [CRONO_Return CRONO_Constr](#) ([CRONO_H](#) *crono)
- [CRONO_Return CRONO_Destr](#) ([CRONO_H](#) crono)

4.3.1 Detailed Description

Functions to construct and destruct Crono objects.

THE SDK mimic an object-oriented paradigm. A [CRONO_H](#) handle is first passed by reference to a constructor, which dynamically allocate all necessary resources, and then by value to all other functions operating on the object. A destructor function frees all resources when not needed any more.

4.3.2 Function Documentation

4.3.2.1 CRONO_Constr()

```
CRONO\_Return CRONO\_Constr (
    CRONO\_H * crono )
```

Constructs a new Crono object.

Parameters

<i>crono</i>	Pointer to a CRONO_H to contain a new Crono object.
--------------	---

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.3.2.2 CRONO_Destr()

```
CRONO_Return CRONO_Destr (
    CRONO_H crono )
```

Frees all resources owned by a Crono handle.

Parameters

<i>crono</i>	Handle to an existing Crono object.
--------------	-------------------------------------

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.4 Crono device access functions.**Functions**

- [CRONO_Return CRONO_GetDeviceCount](#) ([CRONO_H](#) crono, int *count)
- [CRONO_Return CRONO_GetDeviceSerial](#) ([CRONO_H](#) crono, int deviceIndex, char *buf)
- [CRONO_Return CRONO_OpenDeviceBySerial](#) ([CRONO_H](#) crono, char *serial)
- [CRONO_Return CRONO_GetDeviceInfo](#) ([CRONO_H](#) crono, char *serial, [CRONO_DeviceInfo](#) *deviceInfo)
- [CRONO_Return CRONO_OpenDeviceAsDataConverter](#) ([CRONO_H](#) crono, char *rawFilePath)

4.4.1 Detailed Description

Functions to find and access connected Crono devices.

4.4.2 Function Documentation

4.4.2.1 CRONO_GetDeviceCount()

```
CRONO_Return CRONO_GetDeviceCount (
    CRONO_H crono,
    int * count )
```

Gets the count of all active and connected Crono devices.

Devices already open and busy are not regarded as available and therefore not counted.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>count</i>	Pointer to an int to contain the count value

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.4.2.2 CRONO_GetDeviceInfo()

```
CRONO_Return CRONO_GetDeviceInfo (
    CRONO_H crono,
    char * serial,
    CRONO_DeviceInfo * deviceInfo )
```

Gets information about a device with a given serial number, or of the already open device.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>serial</i>	Serial number of the Crono device. Can be an empty string if the device is already opened.
<i>deviceInfo</i>	Struct containing information about the requested device, passed by reference.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.4.2.3 CRONO_GetDeviceSerial()

```
CRONO_Return CRONO_GetDeviceSerial (
    CRONO_H crono,
    int deviceIndex,
    char * buf )
```

Get the serial number of a connected Crono device given its numeric index.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>deviceIndex</i>	Zero-based index of Crono device. The valid range is 0 to (number of devices) - 1
<i>buf</i>	Pointer to a char buffer to contain the serial number. The buffer must be at least MAX_SERIAL_LENGTH characters long. The written serial number string is null-terminated.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)

4.4.2.4 CRONO_OpenDeviceAsDataConverter()

```
CRONO_Return CRONO_OpenDeviceAsDataConverter (
    CRONO_H crono,
    char * rawFilePath )
```

Open saved Crono raw file as a virtual device for data conversion.

A virtual device can be controlled as a real one, except for any device setting (all incompatible functions will return an error when a virtual device is open). In order to perform a conversion, first call the [CRONO_GetAcquisitionInfo](#) and [CRONO_GetCameraSettings](#) to retrieve the actual acquisition info and settings. Then the desired formats must be chosen using [CRONO_ConfigAcquisition\(\)](#) function, finally a virtual acquisition is started using [CRONO_StartAcquisition\(\)](#) and data is retrieved using [CRONO_GetHistogramData\(\)](#) or [CRONO_GetFramesData\(\)](#) or [CRONO_GetTimeTagsData\(\)](#). It is suggest to use the function [CRONO_GetAcquisitionStatus\(\)](#) to monitor the status of the internal buffers. When conversion is done, function must be called [CRONO_StopAcquisition\(\)](#). See the [SDK_Example](#) for more details.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>rawFilePath</i>	Char array with the full path to an existing Crono raw data file.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.4.2.5 CRONO_OpenDeviceBySerial()

```
CRONO_Return CRONO_OpenDeviceBySerial (
    CRONO_H crono,
    char * serial )
```

Opens a device with a given serial number.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>serial</i>	Serial number of the Crono device to be opened.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.5 Crono device configuration functions.

Functions

- [CRONO_Return CRONO_SetSpadDeadTime](#) ([CRONO_H](#) crono, int deadTimeNs)
- [CRONO_Return CRONO_GetSensorType](#) ([CRONO_H](#) crono, [CRONO_SensorType](#) *sensorType)
- [CRONO_Return CRONO_ConfigSensor](#) ([CRONO_H](#) crono, int activePixels, [cronoBool](#) half1Enable, [cronoBool](#) half2Enable, double *maxFrameStartFrequency)
- [CRONO_Return CRONO_GetActivePixels](#) ([CRONO_H](#) crono, int *activePixels)
- [CRONO_Return CRONO_LoadCameraSettings](#) ([CRONO_H](#) crono, [CRONO_CameraSettings](#) settings)
- [CRONO_Return CRONO_GetCameraSettings](#) ([CRONO_H](#) crono, [CRONO_CameraSettings](#) *settings)
- [CRONO_Return CRONO_LoadSettingsFromFile](#) ([CRONO_H](#) crono, char *path)
- [CRONO_Return CRONO_SaveSettingsToFile](#) ([CRONO_H](#) crono, char *path)
- [CRONO_Return CRONO_ConfigAcquisition](#) ([CRONO_H](#) crono, [CRONO_PixelMode](#) pixelMode, [cronoBool](#) framesEnable, [cronoBool](#) histogramsEnable, [cronoBool](#) timeTagsEnable, [cronoBool](#) liveEnable, [cronoBool](#) unused, const char *rawDataOutPath)

4.5.1 Detailed Description

Functions to configure various features of the Crono camera.

4.5.2 Function Documentation

4.5.2.1 CRONO_ConfigAcquisition()

```
CRONO_Return CRONO_ConfigAcquisition (
    CRONO_H crono,
    CRONO_PixelMode pixelMode,
    cronoBool framesEnable,
    cronoBool histogramsEnable,
    cronoBool timeTagsEnable,
    cronoBool liveEnable,
    cronoBool unused,
    const char * rawDataOutPath )
```

configures the acquisition modes and selects the output data formats.

Data will be generated only for active formats. Please refer to the User Manual for more details about data formats. Please refer also to the sample program

See also

CRONO_SDK_Example.c for use cases.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>pixelMode</i>	Desired pixel mode, can be either counting or timing.
<i>framesEnable</i>	If true enables the generation of timing or counting frames. Use CRONO_GetFramesData() function to retrieve data.
<i>histogramsEnable</i>	If true enables the generation of histograms. Use CRONO_GetHistogramData() function to retrieve data.
<i>timeTagsEnable</i>	If true enables the generation of time tags. Use CRONO_GetTimeTagsData() function to retrieve data.
<i>liveEnable</i>	If true enables the generation of live data. If no other format is selected, the PureLive mode is enabled, which allows changing some of the settings during acquisition. Use CRONO_GetLiveData() function to retrieve data.
<i>unused</i>	For future use. Value is ignored.
<i>rawDataOutPath</i>	Raw data file path. Set this parameter to a non empty C string to enable automatic saving of raw data from the SDK in the specified file.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.5.2.2 CRONO_ConfigSensor()

```
CRONO_Return CRONO_ConfigSensor (
    CRONO_H crono,
    int activePixels,
    cronoBool half1Enable,
    cronoBool half2Enable,
    double * maxFrameStartFrequency )
```

Configures the Crono spad array sensor region of interest, i.e.

configures the number of active pixels and enables/disables each sensor half on 128x1 sensors.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>activePixels</i>	Number of pixels to be activated. Valid values range from 1 to the number of pixels in the sensor (including limits). The maximum number of pixels is 64 for 8x8 arrays and 128 for 128x1 sensors.
<i>half1Enable</i>	Enables the first half of the Crono sensor on 128x1 sensors. Refer to the Crono datasheet for further details.
<i>half2Enable</i>	Enables the second half of the Crono sensor on 128x1 sensors. Refer to the Crono datasheet for further details.
<i>maxFrameStartFrequency</i>	Pointer to a double that will contain the calculated maximum allowable frame start frequency with the given configuration. This value gives an upper limit to the frame rate configurable via CRONO_ConfigFrameStartInternal() or CRONO_ConfigFrameStartExternal() .

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.5.2.3 CRONO_GetActivePixels()

```
CRONO_Return CRONO_GetActivePixels (
    CRONO_H crono,
    int * activePixels )
```

Gets the number of active pixels currently configured.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>activePixels</i>	Pointer to an int that will contain the number of active pixels.

Returns

CRONO_Return

4.5.2.4 CRONO_GetCameraSettings()

```
CRONO_Return CRONO_GetCameraSettings (
    CRONO_H crono,
    CRONO_CameraSettings * settings )
```

Gets actual Crono settings and saves them in a user provided [CRONO_CameraSettings](#) struct passed by reference.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>settings</i>	Pointer to user-provided CRONO_CameraSettings struct to accomodate Crono settings.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.5.2.5 CRONO_GetSensorType()

```
CRONO_Return CRONO_GetSensorType (
    CRONO_H crono,
    CRONO_SensorType * sensorType )
```

Retrieves the sensor type (128x1 or 8x8)

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>sensorType</i>	Pointer to the retrieved Crono sensor type/format.

Returns

CRONO_Return

4.5.2.6 CRONO_LoadCameraSettings()

```
CRONO_Return CRONO_LoadCameraSettings (
    CRONO_H crono,
    CRONO_CameraSettings settings )
```

Loads Crono settings from a [CRONO_CameraSettings](#) struct.

Overrides any configuration previously set via other functions.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>settings</i>	User-provided CRONO_CameraSettings struct.

Returns

CRONO_Return

4.5.2.7 CRONO_LoadSettingsFromFile()

```
CRONO_Return CRONO_LoadSettingsFromFile (
    CRONO_H crono,
    char * path )
```

Loads Crono settings from a file.

Overrides any configuration previously set via other functions.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>path</i>	Full path to the settings file to be read.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.5.2.8 CRONO_SaveSettingsToFile()

```
CRONO_Return CRONO_SaveSettingsToFile (
    CRONO_H crono,
    char * path )
```

Saves current settings to file.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>path</i>	Full path to the settings file to be written.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)

4.5.2.9 CRONO_SetSpadDeadTime()

```
CRONO_Return CRONO_SetSpadDeadTime (
    CRONO_H crono,
    int deadTimeNs )
```

Sets the SPADs dead time in ns.

Value applies to all pixels.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>deadTimeNs</i>	Dead time value to be set expressed in nanoseconds. Valid values range from 40 nanoseconds to 150 nanoseconds (including limits).

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)

4.6 Crono device data retrieval functions.

Functions

- [CRONO_Return CRONO_StartAcquisition](#) ([CRONO_H](#) *crono*, [uint64_t](#) *acqMaxFrames*, [int](#) *integPeriod*↵
Frames)
- [CRONO_Return CRONO_StopAcquisition](#) ([CRONO_H](#) *crono*, [cronoBool](#) *force*)
- [CRONO_Return CRONO_GetAcquisitionStatus](#) ([CRONO_H](#) *crono*, [CRONO_AcquisitionStatus](#) **status*)
- [CRONO_Return CRONO_GetAcquisitionInfo](#) ([CRONO_H](#) *crono*, [CRONO_AcquisitionInfo](#) **acqInfo*)

- [CRONO_Return CRONO_GetFramesData](#) ([CRONO_H](#) crono, uint32_t *buf, uint64_t capacityNumFrames, uint64_t *copiedNumFrames)
- [CRONO_Return CRONO_GetTimeTagsData](#) ([CRONO_H](#) crono, uint64_t *dataBuf, uint64_t capacity, uint64_t *copiedLen)
- [CRONO_Return CRONO_GetHistogramData](#) ([CRONO_H](#) crono, uint32_t *buf, uint64_t capacityNumHists, uint64_t *copiedNumHists)
- [CRONO_Return CRONO_GetLiveData](#) ([CRONO_H](#) crono, uint32_t *countsBuf, uint32_t *histogramBuf, [cronoBool](#) *dataIsNew)

4.6.1 Detailed Description

Functions used to start and stop acquisition and to retrieve data during and after a data acquisition.

Different functions provide access to the acquired data in different data formats. When the acquisition is started several processes are created, to download, decode and optionally saves data. Depending on the acquisition settings, several buffers are also created, which are then filled by the SDK. Data from these buffers need to be read by the user application at a sufficiently high rate. If data is not read by the user fast enough, SDK buffers will grow up to a fixed level, upon which download will stall and internal camera 1GB buffer in turn starts to fill. When also camera buffer gets full the acquisition fails. Current status of buffer and ongoing acquisition can be retrieved using [CRONO_GetAcquisitionStatus\(\)](#) function.

4.6.2 Function Documentation

4.6.2.1 CRONO_GetAcquisitionInfo()

```
CRONO_Return CRONO_GetAcquisitionInfo (
    CRONO_H crono,
    CRONO_AcquisitionInfo * acqInfo )
```

Gets informations about the last acquisition.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>acqInfo</i>	Dedicated struct containing information about the last acquisition.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.6.2.2 CRONO_GetAcquisitionStatus()

```
CRONO_Return CRONO_GetAcquisitionStatus (
    CRONO_H crono,
    CRONO_AcquisitionStatus * status )
```

Gets the current acquisition status.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>status</i>	Status of the current acquisition. Refer to the CRONO_AcquisitionStatus definition for details about the various struct fields.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.6.2.3 CRONO_GetFramesData()

```
CRONO_Return CRONO_GetFramesData (
    CRONO_H crono,
    uint32_t * buf,
    uint64_t capacityNumFrames,
    uint64_t * copiedNumFrames )
```

Gets an acquisition data chunk in frames format.

In counting mode, frames contains the counts per pixel accumulated over `integPeriodFrames` as set in [CRONO_StartAcquisition\(\)](#). In timing mode, always timing data from individual frames are reported. Data will be retrieved from the internal SDK buffer and copied up to the capacity indicated by the user with the `capacityNumFrames` parameter, or until the internal buffer is empty.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>buf</i>	Pointer to user-allocated contiguous uint32 data buffer.
<i>capacityNumFrames</i>	Capacity of user-allocated buffer expressed as number of frames. The expected buffer length is <code>capacityNumFrames</code> times the number of active pixels.
<i>copiedNumFrames</i>	Pointer to a <code>uint64_t</code> that will contain the actual number of frames copied into <code>buf</code> . The value pointed by <code>copiedNumFrames</code> is always less than or equal to <code>capacityNumFrames</code> .

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).**4.6.2.4 CRONO_GetHistogramData()**

```
CRONO_Return CRONO_GetHistogramData (
    CRONO_H crono,
    uint32_t * buf,
    uint64_t capacityNumHists,
    uint64_t * copiedNumHists )
```

Gets an acquisition data chunk in histogram format.

Histograms are accumulated over `integPeriodFrames` as set in [CRONO_StartAcquisition\(\)](#). Data will be retrieved from the internal SDK buffer and copied up to the capacity indicated by the user with the `capacityNumHists` parameter, or until the internal buffer is empty.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>buf</i>	Pointer to user-allocated contiguous uint32 data buffer.
<i>capacityNumHists</i>	Capacity of user-allocated buffer expressed as a number of histograms. The expected buffer length is <code>capacityNumHists</code> times the number of active pixels times 4096.
<i>copiedNumHists</i>	Pointer to a <code>uint64_t</code> that will contain the actual number of histograms copied into <code>buf</code> . The returned value is always less than or equal to <code>capacityNumHists</code> .

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).**4.6.2.5 CRONO_GetLiveData()**

```
CRONO_Return CRONO_GetLiveData (
    CRONO_H crono,
    uint32_t * countsBuf,
    uint32_t * histogramBuf,
    cronoBool * dataIsNew )
```

Gets latest live data whenever available.

Histogram is meaningful only in timing mode. Counts and histogram data are accumulated over `integPeriodFrames` as set in [CRONO_StartAcquisition\(\)](#).

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>countsBuf</i>	Pointer to user-allocated contiguous uint32 buffer for live counts. The number of allocated elements must be equal or greater than the number of active pixels.
<i>histogramBuf</i>	Pointer to user-allocated contiguous uint32 buffer for live tdc histogram. The number of allocated elements must be equal or greater than the number of active pixels multiplied by 4096.
<i>dataIsNew</i>	Pointer to boolean value indicating whether newly available live data has been written to <i>countsBuf</i> and <i>histogramBuf</i>

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.6.2.6 CRONO_GetTimeTagsData()

```
CRONO_Return CRONO_GetTimeTagsData (
    CRONO_H crono,
    uint64_t * dataBuf,
    uint64_t capacity,
    uint64_t * copiedLen )
```

Gets an acquisition data chunk in time tags format.

Data will be retrieved from the internal SDK buffer and copied up to the capacity indicated by the user with the *capacity* parameter, or until the internal buffer is empty.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>dataBuf</i>	Pointer to user-allocated contiguous uint64 data buffer.
<i>capacity</i>	Capacity of user-allocated buffer expressed as number of uint64_t values.
<i>copiedLen</i>	Pointer to a uint64_t that will contain the actual number of data copied into <i>buf</i> . The value pointed by <i>copiedLen</i> is always less than or equal to <i>capacity</i> .

Returns

CRONO_Return

4.6.2.7 CRONO_StartAcquisition()

```
CRONO_Return CRONO_StartAcquisition (
    CRONO_H crono,
```

```
uint64_t acqMaxFrames,
int integPeriodFrames )
```

Starts a new acquisition.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>acqMaxFrames</i>	Desired number of frames in the acquisition. Set this parameter to zero to start an indefinitely long acquisition.
<i>integPeriodFrames</i>	Histograms, live counts and counting mode data are accumulated over <i>integPeriodFrames</i> frames. This parameter also affects the live data refresh rate. Any positive value larger than 0 is accepted, however, when not in PureLive mode, a value > 100 may be necessary to allow continuous acquisition depending on PC performance and camera configuration.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.6.2.8 CRONO_StopAcquisition()

```
CRONO_Return CRONO_StopAcquisition (
    CRONO_H crono,
    cronoBool force )
```

Stops an ongoing acquisition.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>force</i>	If false, this function will block until the desired number of frames is acquired. If true, forcibly stops the ongoing acquisition.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.7 Crono device control signal management functions.

Functions

- [CRONO_Return CRONO_ConfigFrameStartInternal](#) ([CRONO_H](#) crono, double frameRate, double *actualBin)
- [CRONO_Return CRONO_ConfigFrameStartExternal](#) ([CRONO_H](#) crono, double frameRate, double threshold, [cronoBool](#) externalStartPosedge, int divFactor)
- [CRONO_Return CRONO_ConfigGateExternal](#) ([CRONO_H](#) crono, double threshold, [cronoBool](#) externalGateInvert)
- [CRONO_Return CRONO_ConfigGateInternal](#) ([CRONO_H](#) crono, int *pattern, int patternLen)
- [CRONO_Return CRONO_ConfigLaserSync](#) ([CRONO_H](#) crono, int *pattern, int patternLen)
- [CRONO_Return CRONO_ConfigOutputs](#) ([CRONO_H](#) crono, int chan1Selection, int chan2Selection, int chan3Selection)
- [CRONO_Return CRONO_ConfigAuxiliaryInputs](#) ([CRONO_H](#) crono, [CRONO_AuxInputMode](#) bitModeA, [CRONO_AuxInputMode](#) bitModeB, double bitAThreshold)
- [CRONO_Return CRONO_ConfigTriggerExternal](#) ([CRONO_H](#) crono, int framesBurstCount)
- [CRONO_Return CRONO_ConfigTriggerInternal](#) ([CRONO_H](#) crono)
- [CRONO_Return CRONO_GetInputFrequencies](#) ([CRONO_H](#) crono, double *extStartFrequency, double *extGateFrequency, double *extTriggerFrequency)

4.7.1 Detailed Description

Functions to configure and manage internal and external control signals.

4.7.2 Function Documentation

4.7.2.1 CRONO_ConfigAuxiliaryInputs()

```
CRONO_Return CRONO_ConfigAuxiliaryInputs (
    CRONO_H crono,
    CRONO_AuxInputMode bitModeA,
    CRONO_AuxInputMode bitModeB,
    double bitAThreshold )
```

Configure aux inputs sampling.

When sampling is active, time tagging for each aux event can be retrieved using [CRONO_GetTimeTagsData\(\)](#), even if `timeTagsEnable` is set to false in [CRONO_ConfigAcquisition\(\)](#), provided that `histogramsEnable` or `framesEnable` are set to true, i.e. acquisition is not raw only.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>bitModeA</i>	Mode of sampling for auxiliary input A (external gate). Refer to the definition of CRONO_AuxInputMode for details about all modes.
<i>bitModeB</i>	Mode of sampling for auxiliary input B (external trigger). Refer to the definition of CRONO_AuxInputMode for details about all modes.
<i>bitAThreshold</i>	Threshold for input A (this is the same as external gate threshold).

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)**4.7.2.2 CRONO_ConfigFrameStartExternal()**

```
CRONO_Return CRONO_ConfigFrameStartExternal (
    CRONO_H crono,
    double frameRate,
    double threshold,
    cronoBool externalStartPosedge,
    int divFactor )
```

Sets the camera in external frame start mode and configure the associated parameters.

Even when start is external the expected maximum frame rate must be provided to the SDK, in order to properly configure the internal camera signal generator for laser sync. When in external start mode, internal gate is fixed to always-on. External gate can be arbitrary.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>frameRate</i>	Expected maximum start rate after division of externally provided frame start signal, expressed in Hz. When employing aperiodic signals this value should be greater than or equal to the reciprocal of the minimum inter-time between triggering edges. Valid values range from 20e3 (20kHz) to <code>maxFrameStartFrequency</code> (including limits).
<i>threshold</i>	Triggering voltage level i.e. threshold of external frame start signal. The value is expressed in V (Volts).
<i>externalStartPosedge</i>	Selects the triggering edge of external frame start signal. When true, Crono frames start at rising edges. When false Crono frames start at falling edges.
<i>divFactor</i>	The external start signal triggers a frame every <code>divFactor</code> rising or falling edges of the external start signal. When equal to 1, Crono will trigger on all incoming rising or falling edges. Valid values range from 1 to 1000 (including limits).

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)

4.7.2.3 CRONO_ConfigFrameStartInternal()

```
CRONO_Return CRONO_ConfigFrameStartInternal (
    CRONO_H crono,
    double frameRate,
    double * actualBin )
```

Sets the camera in internal frame start mode and sets the frame start rate.

The actual time bin (nominally 1ns) used for gate and laser sync generation depends on the internal start frequency. The calculated correction factor is returned by the referenced `actualBin` parameter.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>frameRate</i>	Desired frame rate. Please make sure that this value never exceeds <code>maxFrameStartFrequency</code> provided by CRONO_ConfigSensor() . Neglect of such requirement will potentially yield to corrupted even counts and /or tdc conversions. The minimum allowed frame rate is 20e3.
<i>actualBin</i>	Pointer to the calculated bin value in ns, to be used for internal gate and laser sync signals.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.7.2.4 CRONO_ConfigGateExternal()

```
CRONO_Return CRONO_ConfigGateExternal (
    CRONO_H crono,
    double threshold,
    cronoBool externalGateInvert )
```

Sets the camera in external gate mode and configure the associated parameters.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>threshold</i>	Threshold level of external gate signal.
<i>externalGateInvert</i>	When true inverts the external external gate signal.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.7.2.5 CRONO_ConfigGateInternal()

```
CRONO_Return CRONO_ConfigGateInternal (
    CRONO_H crono,
    int * pattern,
    int patternLen )
```

Set the camera in internal gate mode and configures the pattern of internally generated gate.

The pattern is an array on integer, expressing in bins (of nominal duration of 1ns) alternatively the OFF and ON sections of the gate signal. The actual bin size depends on the applied Frame-Start frequency and is returned by the function [CRONO_ConfigFrameStartInternal\(\)](#). Note that external gate signals, if applied to the camera, will be ignored for gate generation but can still be sampled as auxiliary input A.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>pattern</i>	Pointer to user-defined pattern array. The sum of all pattern value must never exceed 65536, and the number of entries must not exceed 8192. Each OFF section, except the very first, must be at least MIN_GATE_OFF bins. Each ON section must be at least MIN_GATE_ON bins. Please refer to the Crono user manual for further details.
<i>patternLen</i>	Length to user-defined pattern array.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c](#).

4.7.2.6 CRONO_ConfigLaserSync()

```
CRONO_Return CRONO_ConfigLaserSync (
    CRONO_H crono,
    int * pattern,
    int patternLen )
```

Configures the pattern of internally generated laser synchronization signal.

The pattern is an array on integer, expressing in bins alternatively the OFF and ON sections of the sync signal. When Frame-Start is external the bin size is 1ns. When Frame-Start is internal the nominal bin size is 1ns, but the actual bin size depends on the applied Frame-Start frequency and is returned by the function [CRONO_ConfigFrameStartInternal\(\)](#).

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>pattern</i>	Pointer to user-defined pattern array. The sum of all pattern value must never exceed 65536, and the number of entries must not exceed 8192. Please refer to the Crono user manual for further details.
<i>patternLen</i>	Length to user-defined pattern array.

Returns

CRONO_Return

Examples[CRONO_SDK_Example.c.](#)**4.7.2.7 CRONO_ConfigOutputs()**

```
CRONO_Return CRONO_ConfigOutputs (
    CRONO_H crono,
    int chan1Selection,
    int chan2Selection,
    int chan3Selection )
```

Configures which Crono signals are forwarded to each of the three output ports.

Note that not each port has a specific selection of allowed signal. Please refer to the Crono user manual for further details on available signals.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>chan1Selection</i>	Selection for channel 1. Valid numbers range from 0 to 7. 0 - LASER SYNC OUT, 1 - 10 MHz REF, 2 - BUSY, 3 - EXTERNAL START, 4 - TRIGGER IN, 5 - AUX IN A SAMPLE, 6 - AUX IN B SAMPLE, 7 - OFF.
<i>chan2Selection</i>	Selection for channel 1. Valid numbers range from 0 to 7. 0 - FRAME START, 1 - 10 MHz REF, 2 - BUSY, 3 - EXTERNAL START, 4 - TRIGGER IN, 5 - AUX IN A SAMPLE, 6 - AUX IN B SAMPLE, 7 - OFF.
<i>chan3Selection</i>	Selection for channel 1. Valid numbers range from 0 to 7. 0 - GATE, 1 - 10 MHz REF, 2 - BUSY, 3 - EXTERNAL START, 4 - TRIGGER IN, 5 - AUX IN A SAMPLE, 6 - AUX IN B SAMPLE, 7 - OFF.

Returns

CRONO_Return

Examples[CRONO_SDK_Example.c.](#)**4.7.2.8 CRONO_ConfigTriggerExternal()**

```
CRONO_Return CRONO_ConfigTriggerExternal (
    CRONO_H crono,
    int framesBurstCount )
```

Sets the camera in external trigger mode.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>framesBurstCount</i>	Number of frames in a triggered frame burst. When equal to zero the triggered burst is indefinitely long. Valid values range from 0 (infinite burst mode) to 65536 (including limits).

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)**4.7.2.9 CRONO_ConfigTriggerInternal()**

```
CRONO_Return CRONO_ConfigTriggerInternal (
    CRONO_H crono )
```

Sets the camera in internal trigger mode.

External trigger signals will be ignored but external trigger port can still be sampled as auxiliary input B.

Parameters

<i>crono</i>	Handle to an existing Crono object.
--------------	-------------------------------------

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)**4.7.2.10 CRONO_GetInputFrequencies()**

```
CRONO_Return CRONO_GetInputFrequencies (
    CRONO_H crono,
    double * extStartFrequency,
    double * extGateFrequency,
    double * extTriggerFrequency )
```

Gets the current measured input signal frequencies at the SMA inputs.

Parameters

<i>crono</i>	Handle to an existing Crono object.
<i>extStartFrequency</i>	Frequency at the external start input.
<i>extGateFrequency</i>	Frequency at the external gate input.
<i>extTriggerFrequency</i>	Frequency at the trigger input.

Returns

CRONO_Return

Examples

[CRONO_SDK_Example.c.](#)

Chapter 5

File Documentation

5.1 CRONO_SDK.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
```

Data Structures

- struct [CRONO_DeviceInfo](#)
- struct [CRONO_AcquisitionInfo](#)
- struct [CRONO_AcquisitionStatus](#)
- struct [CRONO_CameraSettings](#)

Macros

- #define [MAX_PATTERN_LENGTH](#) 8192
- #define [MIN_GATE_OFF](#) 12
- #define [MIN_GATE_ON](#) 5
- #define [MAX_SERIAL_LENGTH](#) 20
- #define [MAX_DATETIME_LENGTH](#) 100

Typedefs

- typedef uint8_t [cronoBool](#)
- typedef struct Crono * [CRONO_H](#)

Enumerations

- enum `CRONO_Return` {
`NO_ERR` = 0 , `DEVICE_NOT_FOUND` = 1 , `LOGIC_VERSION_FAIL` = 2 , `DEVICE_NOT_OPEN` = 3 ,
`DEVICE_ALREADY_OPEN` = 4 , `DEVICE_OPEN_AS_CONVERTER` = 5 , `INVALID_DATA_FILE` = 6 ,
`SENSOR_FAILED` = 7 ,
`POWER_FAILED` = 8 , `INVALID_PIXEL_NUMBER` = 9 , `INVALID_DEAD_TIME` = 10 , `INVALID_HALVES_CONFIG`
= 11 ,
`INVALID_START_RATE` = 12 , `INVALID_DIVISION_FACTOR` = 13 , `INVALID_FRAME_BURST_COUNT` =
14 , `INVALID_GATE_LENGTH` = 15 ,
`INVALID_LASER_SYNC_LENGTH` = 16 , `INVALID_CONFIG` = 17 , `INVALID_THRESHOLD` = 18 ,
`NO_SUCH_FILE_OR_DIRECTORY` = 19 ,
`ACQ_ONGOING` = 20 , `ACQ_EXT_TRIG_TIMEOUT` = 21 , `RAM_FULL` = 22 , `NULL_POINTER` = 23 ,
`PATTERN_TOO_LONG` = 24 , `INVALID_CHANNEL` = 25 , `INVALID_ENUM` = 26 , `FPGA_FAILED` = -1 ,
`FPGA_TIMEOUT` = -2 }
- enum `CRONO_SensorType` { `CRONO_8X8` = 0 , `CRONO_128X1` = 1 }
- enum `CRONO_PixelMode` { `TIMING` = 0 , `COUNTING` = 1 }
- enum `CRONO_AuxInputMode` { `OFF` = 0 , `RISING_EDGE` = 1 , `FALLING_EDGE` = 2 , `BOTH_EDGES` = 3 }

Functions

- `CRONO_Return CRONO_Constr (CRONO_H *crono)`
- `CRONO_Return CRONO_Destr (CRONO_H crono)`
- `CRONO_Return CRONO_GetDeviceCount (CRONO_H crono, int *count)`
- `CRONO_Return CRONO_GetDeviceSerial (CRONO_H crono, int deviceIndex, char *buf)`
- `CRONO_Return CRONO_OpenDeviceBySerial (CRONO_H crono, char *serial)`
- `CRONO_Return CRONO_GetDeviceInfo (CRONO_H crono, char *serial, CRONO_DeviceInfo *deviceInfo)`
- `CRONO_Return CRONO_OpenDeviceAsDataConverter (CRONO_H crono, char *rawFilePath)`
- `CRONO_Return CRONO_SetSpadDeadTime (CRONO_H crono, int deadTimeNs)`
- `CRONO_Return CRONO_GetSensorType (CRONO_H crono, CRONO_SensorType *sensorType)`
- `CRONO_Return CRONO_ConfigSensor (CRONO_H crono, int activePixels, cronoBool half1Enable,
cronoBool half2Enable, double *maxFrameStartFrequency)`
- `CRONO_Return CRONO_GetActivePixels (CRONO_H crono, int *activePixels)`
- `CRONO_Return CRONO_LoadCameraSettings (CRONO_H crono, CRONO_CameraSettings settings)`
- `CRONO_Return CRONO_GetCameraSettings (CRONO_H crono, CRONO_CameraSettings *settings)`
- `CRONO_Return CRONO_LoadSettingsFromFile (CRONO_H crono, char *path)`
- `CRONO_Return CRONO_SaveSettingsToFile (CRONO_H crono, char *path)`
- `CRONO_Return CRONO_ConfigAcquisition (CRONO_H crono, CRONO_PixelMode pixelMode, cronoBool
framesEnable, cronoBool histogramsEnable, cronoBool timeTagsEnable, cronoBool liveEnable, cronoBool
unused, const char *rawDataOutPath)`
- `CRONO_Return CRONO_StartAcquisition (CRONO_H crono, uint64_t acqMaxFrames, int integPeriod↵
Frames)`
- `CRONO_Return CRONO_StopAcquisition (CRONO_H crono, cronoBool force)`
- `CRONO_Return CRONO_GetAcquisitionStatus (CRONO_H crono, CRONO_AcquisitionStatus *status)`
- `CRONO_Return CRONO_GetAcquisitionInfo (CRONO_H crono, CRONO_AcquisitionInfo *acqInfo)`
- `CRONO_Return CRONO_GetFramesData (CRONO_H crono, uint32_t *buf, uint64_t capacityNumFrames,
uint64_t *copiedNumFrames)`
- `CRONO_Return CRONO_GetTimeTagsData (CRONO_H crono, uint64_t *dataBuf, uint64_t capacity,
uint64_t *copiedLen)`
- `CRONO_Return CRONO_GetHistogramData (CRONO_H crono, uint32_t *buf, uint64_t capacityNumHists,
uint64_t *copiedNumHists)`
- `CRONO_Return CRONO_GetLiveData (CRONO_H crono, uint32_t *countsBuf, uint32_t *histogramBuf,
cronoBool *dataIsNew)`
- `CRONO_Return CRONO_ConfigFrameStartInterval (CRONO_H crono, double frameRate, double *actual↵
Bin)`

- [CRONO_Return CRONO_ConfigFrameStartExternal](#) (CRONO_H crono, double frameRate, double threshold, [cronoBool](#) externalStartPosedge, int divFactor)
- [CRONO_Return CRONO_ConfigGateExternal](#) (CRONO_H crono, double threshold, [cronoBool](#) externalGateInvert)
- [CRONO_Return CRONO_ConfigGateInternal](#) (CRONO_H crono, int *pattern, int patternLen)
- [CRONO_Return CRONO_ConfigLaserSync](#) (CRONO_H crono, int *pattern, int patternLen)
- [CRONO_Return CRONO_ConfigOutputs](#) (CRONO_H crono, int chan1Selection, int chan2Selection, int chan3Selection)
- [CRONO_Return CRONO_ConfigAuxiliaryInputs](#) (CRONO_H crono, [CRONO_AuxInputMode](#) bitModeA, [CRONO_AuxInputMode](#) bitModeB, double bitAThreshold)
- [CRONO_Return CRONO_ConfigTriggerExternal](#) (CRONO_H crono, int framesBurstCount)
- [CRONO_Return CRONO_ConfigTriggerInternal](#) (CRONO_H crono)
- [CRONO_Return CRONO_GetInputFrequencies](#) (CRONO_H crono, double *extStartFrequency, double *extGateFrequency, double *extTriggerFrequency)

5.1.1 Detailed Description

CRONO software development kit.

This C header contains all the functions to operate the CRONO camera in user defined applications.

5.2 CRONO_SDK.h

[Go to the documentation of this file.](#)

```

1 /*
2 #####
3
4 Copyright 2024 Micro-Photon-Devices s.r.l.
5
6 SOFTWARE PRODUCT: CRONO_SDK 1.0.1
7
8 Micro-Photon-Devices (MPD) expressly disclaims any warranty for the SOFTWARE
9 PRODUCT. The SOFTWARE PRODUCT is provided 'As Is' without any express or
10 implied warranty of any kind, including but not limited to any warranties of
11 merchantability, noninfringement, or fitness of a particular purpose. MPD does
12 not warrant or assume responsibility for the accuracy or completeness of any
13 information, text, graphics, links or other items contained within the SOFTWARE
14 PRODUCT. MPD further expressly disclaims any warranty or representation to
15 Authorized Users or to any third party. In no event shall MPD be liable for
16 any damages (including, without limitation, lost profits, business interruption
17 or lost information) rising out of 'Authorized Users' use of or inability to
18 use the SOFTWARE PRODUCT, even if MPD has been advised of the possibility of
19 such damages. In no event will MPD be liable for loss of data or for indirect,
20 special, incidental, consequential (including lost profit), or other damages
21 based in contract, tort or otherwise. MPD shall have no liability with respect
22 to the content of the SOFTWARE PRODUCT or any part thereof, including but not
23 limited to errors or omissions contained therein, libel, infringements of
24 rights of publicity, privacy, trademark rights, business interruption, personal
25 injury, loss of privacy, moral rights or the disclosure of confidential
26 information.
27 #####
28 */
29
30 #pragma once
31
32 #ifndef __CRONO_SDK_h__
33 #define __CRONO_SDK_h__
34
35 #ifdef __cplusplus
36 extern "C"
37 {
38 #endif
39
40 #include <stdlib.h>
41 #include <stdint.h>
42
43 #ifdef _WIN32

```

```

66 #ifdef CRONO_SDK_EXPORTS
67 #define CRONO_SDK_API __declspec(dllexport)
68 #else
69 #define CRONO_SDK_API __declspec(dllimport)
70 #endif
71 #else
72 #define CRONO_SDK_API
73 #endif
74
81 #define MAX_PATTERN_LENGTH 8192
83 #define MIN_GATE_OFF 12
85 #define MIN_GATE_ON 5
87 #define MAX_SERIAL_LENGTH 20
89 #define MAX_DATETIME_LENGTH 100
99 typedef uint8_t cronoBool;
102 typedef struct Crono *CRONO_H;
107 typedef enum
108 {
109     NO_ERR = 0,
110     DEVICE_NOT_FOUND = 1,
111     LOGIC_VERSION_FAIL = 2,
112     DEVICE_NOT_OPEN = 3,
113     DEVICE_ALREADY_OPEN = 4,
114     DEVICE_OPEN_AS_CONVERTER = 5,
115     INVALID_DATA_FILE = 6,
116     SENSOR_FAILED = 7,
117     POWER_FAILED = 8,
118     INVALID_PIXEL_NUMBER = 9,
119     INVALID_DEAD_TIME = 10,
120     INVALID_HALVES_CONFIG = 11,
121     INVALID_START_RATE = 12,
122     INVALID_DIVISION_FACTOR = 13,
123     INVALID_FRAME_BURST_COUNT = 14,
124     INVALID_GATE_LENGTH = 15,
125     INVALID_LASER_SYNC_LENGTH = 16,
126     INVALID_CONFIG = 17,
127     INVALID_THRESHOLD = 18,
128     NO_SUCH_FILE_OR_DIRECTORY = 19,
129     ACQ_ONGOING = 20,
130     ACQ_EXT_TRIG_TIMEOUT = 21,
131     RAM_FULL = 22,
132     NULL_POINTER = 23,
133     PATTERN_TOO_LONG = 24,
134     INVALID_CHANNEL = 25,
135     INVALID_ENUM = 26,
136     FPGA_FAILED = -1,
137     FPGA_TIMEOUT = -2,
138 } CRONO_Return;
139
141 typedef enum
142 {
143     CRONO_8X8 = 0,
144     CRONO_128X1 = 1,
145 } CRONO_SensorType;
146
149 typedef struct
150 {
151     CRONO_SensorType SensorType;
152     char Serial[MAX_SERIAL_LENGTH];
153     int FwMajorVersion;
154     int FwMinorVersion;
155     int FwPatchVersion;
156     int SdkMajorVersion;
157     int SdkMinorVersion;
158     int SdkPatchVersion;
159     int UsbSpeed;
160 } CRONO_DeviceInfo;
161
164 typedef struct
165 {
166     char DateTime[MAX_DATETIME_LENGTH];
167     uint64_t acquiredFramesCount;
168 } CRONO_AcquisitionInfo;
169
173 typedef enum
174 {
175     TIMING = 0,
176     COUNTING = 1
177 } CRONO_PixelMode;
178
182 typedef enum
183 {
184     OFF = 0,
185     RISING_EDGE = 1,
186     FALLING_EDGE = 2,
187     BOTH_EDGES = 3
188 } CRONO_AuxInputMode;

```

```

189
192 typedef struct
193 {
194     double acquisitionTime;
195     int64_t capturedFrameCount;
196     int64_t decodedFrameCount;
197     int64_t deviceRamUsageBytes;
198     int64_t downloadedBytes;
199     int64_t hostAllocatedBytes;
200     int64_t availableRawCount;
201     int64_t availableFramesCount;
202     int64_t availableTimeTagsCount;
203     int64_t availableHistogramCount;
204     int64_t auxBitACount;
205     int64_t auxBitBCount;
206     cronoBool ongoing;
207     cronoBool deviceRamFull;
208 } CRONO_AcquisitionStatus;
209
212 typedef struct
213 {
214     int deadTimeNs;
215     // sensor roi
216     int activePixels;
217     cronoBool half1Enable;
218     cronoBool half2Enable;
219     // counting/timing mode
220     CRONO_PixelMode countingMode;
221     // acquisition
222     int integPeriodFrames;
223     uint64_t acqMaxFrames;
224     // start
225     cronoBool startIsExternal;
226     double frameRate;
227     double externalStartThreshold;
228     cronoBool externalStartPosedge;
229     int externalStartDivisionFactor;
230     // gate
231     cronoBool gateIsExternal;
232     double externalGateThreshold;
233     cronoBool externalGateInvert;
234     uint16_t internalGatePattern[MAX_PATTERN_LENGTH];
235     int internalGatePatternLenght;
236     // laser sync
237     uint16_t laserSyncPattern[MAX_PATTERN_LENGTH];
238     int laserSyncPatternLenght;
239     // sma output channels selection
240     int chan1Selection;
241     int chan2Selection;
242     int chan3Selection;
243     // ext trigger
244     cronoBool acquireOnExtTrigger;
245     int externalTriggerBurstCount;
246     // aux signals sampling modes
247     CRONO_AuxInputMode AuxInputASamplingMode;
248     CRONO_AuxInputMode AuxInputBSamplingMode;
249 } CRONO_CameraSettings;
250
267 CRONO_SDK_API CRONO_Return CRONO_Constr(CRONO_H *crono);
268
275 CRONO_SDK_API CRONO_Return CRONO_Destr(CRONO_H crono);
276
291 CRONO_SDK_API CRONO_Return CRONO_GetDeviceCount(CRONO_H crono, int *count);
292
301 CRONO_SDK_API CRONO_Return CRONO_GetDeviceSerial(CRONO_H crono, int deviceIndex, char *buf);
302
310 CRONO_SDK_API CRONO_Return CRONO_OpenDeviceBySerial(CRONO_H crono, char *serial);
311
320 CRONO_SDK_API CRONO_Return CRONO_GetDeviceInfo(CRONO_H crono, char *serial, CRONO_DeviceInfo
    *deviceInfo);
321
332 CRONO_SDK_API CRONO_Return CRONO_OpenDeviceAsDataConverter(CRONO_H crono, char *rawFilePath);
333
349 CRONO_SDK_API CRONO_Return CRONO_SetSpadDeadTime(CRONO_H crono, int deadTimeNs);
350
358 CRONO_SDK_API CRONO_Return CRONO_GetSensorType(CRONO_H crono, CRONO_SensorType *sensorType);
359
371 CRONO_SDK_API CRONO_Return CRONO_ConfigSensor(CRONO_H crono, int activePixels, cronoBool half1Enable,
    cronoBool half2Enable, double *maxFrameStartFrequency);
372
380 CRONO_SDK_API CRONO_Return CRONO_GetActivePixels(CRONO_H crono, int *activePixels);
381
389 CRONO_SDK_API CRONO_Return CRONO_LoadCameraSettings(CRONO_H crono, CRONO_CameraSettings settings);
390
398 CRONO_SDK_API CRONO_Return CRONO_GetCameraSettings(CRONO_H crono, CRONO_CameraSettings *settings);
399
407 CRONO_SDK_API CRONO_Return CRONO_LoadSettingsFromFile(CRONO_H crono, char *path);

```

```

408
416 CRONO_SDK_API CRONO_Return CRONO_SaveSettingsToFile(CRONO_H crono, char *path);
417
432 CRONO_SDK_API CRONO_Return CRONO_ConfigAcquisition(CRONO_H crono, CRONO_PixelMode pixelMode, cronoBool
    framesEnable, cronoBool histogramsEnable, cronoBool timeTagsEnable, cronoBool liveEnable, cronoBool
    unused, const char *rawDataOutPath);
433
455 CRONO_SDK_API CRONO_Return CRONO_StartAcquisition(CRONO_H crono, uint64_t acqMaxFrames, int
    integPeriodFrames);
456
464 CRONO_SDK_API CRONO_Return CRONO_StopAcquisition(CRONO_H crono, cronoBool force);
465
473 CRONO_SDK_API CRONO_Return CRONO_GetAcquisitionStatus(CRONO_H crono, CRONO_AcquisitionStatus *status);
474
482 CRONO_SDK_API CRONO_Return CRONO_GetAcquisitionInfo(CRONO_H crono, CRONO_AcquisitionInfo *acqInfo);
483
494 CRONO_SDK_API CRONO_Return CRONO_GetFramesData(CRONO_H crono, uint32_t *buf, uint64_t capacityNumFrames,
    uint64_t *copiedNumFrames);
495
505 CRONO_SDK_API CRONO_Return CRONO_GetTimeTagsData(CRONO_H crono, uint64_t *dataBuf, uint64_t capacity,
    uint64_t *copiedLen);
506
516 CRONO_SDK_API CRONO_Return CRONO_GetHistogramData(CRONO_H crono, uint32_t *buf, uint64_t
    capacityNumHists, uint64_t *copiedNumHists);
517
527 CRONO_SDK_API CRONO_Return CRONO_GetLiveData(CRONO_H crono, uint32_t *countsBuf, uint32_t *histogramBuf,
    cronoBool *dataIsNew);
528
546 CRONO_SDK_API CRONO_Return CRONO_ConfigFrameStartInternal(CRONO_H crono, double frameRate, double
    *actualBin);
547
559 CRONO_SDK_API CRONO_Return CRONO_ConfigFrameStartExternal(CRONO_H crono, double frameRate, double
    threshold, cronoBool externalStartPosedge, int divFactor);
560
568 CRONO_SDK_API CRONO_Return CRONO_ConfigGateExternal(CRONO_H crono, double threshold, cronoBool
    externalGateInvert);
569
580 CRONO_SDK_API CRONO_Return CRONO_ConfigGateInternal(CRONO_H crono, int *pattern, int patternLen);
581
591 CRONO_SDK_API CRONO_Return CRONO_ConfigLaserSync(CRONO_H crono, int *pattern, int patternLen);
592
606 CRONO_SDK_API CRONO_Return CRONO_ConfigOutputs(CRONO_H crono, int chan1Selection, int chan2Selection,
    int chan3Selection);
607
617 CRONO_SDK_API CRONO_Return CRONO_ConfigAuxiliaryInputs(CRONO_H crono, CRONO_AuxInputMode bitModeA,
    CRONO_AuxInputMode bitModeB, double bitAThreshold);
618
626 CRONO_SDK_API CRONO_Return CRONO_ConfigTriggerExternal(CRONO_H crono, int framesBurstCount);
627
634 CRONO_SDK_API CRONO_Return CRONO_ConfigTriggerInternal(CRONO_H crono);
635
645 CRONO_SDK_API CRONO_Return CRONO_GetInputFrequencies(CRONO_H crono, double *extStartFrequency, double
    *extGateFrequency, double *extTriggerFrequency);
646
649 #ifdef _MPD_DEV_
650 #include "MPD_Utils.h"
651 #endif
652
655 #ifdef __cplusplus
656 } // extern "C"
657 #endif
658
659 #endif // ifndef __CRONO_SDK_h__

```

Chapter 6

Example Documentation

6.1 CRONO_SDK_Example.c

```
/*
#####

Copyright 2024 Micro-Photon-Devices s.r.l.

SOFTWARE PRODUCT: CRONO_SDK

Micro-Photon-Devices (MPD) expressly disclaims any warranty for the SOFTWARE PRODUCT.
The SOFTWARE PRODUCT is provided 'As Is' without any express or implied warranty of any kind,
including but not limited to any warranties of merchantability, noninfringement, or
fitness of a particular purpose. MPD does not warrant or assume responsibility for the
accuracy or completeness of any information, text, graphics, links or other items contained
within the SOFTWARE PRODUCT. MPD further expressly disclaims any warranty or representation
to Authorized Users or to any third party.
In no event shall MPD be liable for any damages (including, without limitation, lost profits,
business interruption, or lost information) rising out of 'Authorized Users' use of or inability
to use the SOFTWARE PRODUCT, even if MPD has been advised of the possibility of such damages.
In no event will MPD be liable for loss of data or for indirect, special, incidental,
consequential (including lost profit), or other damages based in contract, tort
or otherwise. MPD shall have no liability with respect to the content of the
SOFTWARE PRODUCT or any part thereof, including but not limited to errors or omissions contained
therein, libel, infringements of rights of publicity, privacy, trademark rights, business
interruption, personal injury, loss of privacy, moral rights or the disclosure of confidential
information.

#####
*/
#include "CRONO_SDK.h"
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#ifdef __linux__
#define SLEEP usleep
#define MILLIS 1000
#elif defined(__APPLE__)
#define SLEEP usleep
#define MILLIS 1000
#include <unistd.h>
#elif defined(_WIN32)
#include <Windows.h>
#define SLEEP Sleep
#define MILLIS 1
#endif
//*****//
//
//          Support functions          //
//
//*****//
//clear screen
void clear_screen()
{
#ifdef _WIN32
system("cls");
#else
// Assume POSIX
system("clear");
```

```

#endif
}
// Calculate the mean value of a uint16_t image
double meanImage(uint32_t * Img, uint16_t NPixel)
{
    int i=0;
    double res =0.0;
    for(i=0;i<NPixel;i++)
        res+= (double)Img[i];
    return res/(double)NPixel;
}
// Create an accumulated histogram over the imager
void AccHist(uint32_t* HistIn, uint32_t* HistOut, int Npixel)
{
    for (int i = 0; i < 4096; i++)
    {
        for (int j = 0; j < Npixel; j++)
        {
            HistOut[i] += *(HistIn + j * 4096 + i);
        }
    }
}
// Print some status info
void print_status(CRONO_AcquisitionStatus s)
{
    double acquisitionTime = (s.acquisitionTime);
    int64_t capturedFrameCount = (s.capturedFrameCount);
    int64_t decodedFrameCount = (s.decodedFrameCount);
    int64_t deviceRamUsageBytes = (s.deviceRamUsageBytes);
    int64_t downloadedBytes = (s.downloadedBytes);
    int64_t hostAllocatedBytes = (s.hostAllocatedBytes);
    printf(" acquisitionTime = %.1f s\n", acquisitionTime);
    printf(" capturedFrameCount = %lld\n", capturedFrameCount);
    printf(" decodedFrameCount = %lld\n", decodedFrameCount);
    printf("deviceRamUsageBytes = %.1f MB\n", deviceRamUsageBytes * 1e-6);
    printf(" downloadedBytes = %.1f MB\n", downloadedBytes * 1e-6);
    printf(" dl rate = %.1f MB/s\n", 1e-6 * (downloadedBytes / acquisitionTime));
    printf(" hostAllocatedBytes = %.1f MB\n", hostAllocatedBytes * 1e-6);
}
//*****//
//                               //
// Main code                               //
//                               //
//*****//
int main(void)
{
    // variables definition //
    CRONO_H crono= NULL;
    CRONO_AcquisitionStatus status;
    CRONO_AcquisitionInfo acqInfo;
    CRONO_CameraSettings settings;
    CRONO_Return err;
    CRONO_DeviceInfo devInfo;
    cronoBool dataIsNew;
    uint32_t* CntsBuf;
    uint32_t* HistBuf;
    uint32_t* AccHistBuf;

    char serial[MAX_SERIAL_LENGTH];
    int pattern[10];
    int device_count = 0, dev_num, nPixel, loop = 1;
    uint64_t num;
    uint32_t total;
    char c = 0;
    double maxFrameStartFrequency, extStartFreq, extGateFreq, extTrigFreq, average;
    char *menu_strings[] = {
        "Live counting mode: write on stdout an image", //a
        "Live timing mode: write on stdout an histogram", //b
        "Acquisition timing mode: get time tagging, frames and histogram", //c
        "Holdoff: mean number of photons at different holdoff values", //d
        "Save settings to file", //e
        "Load settings from file", //f
        "Configure outputs", //g
        "Acquire with external start", //h
        "Acquire with external gate and aux-input timetagging", //i
        "Use external trigger", //j
        "Load raw data from file" //k
    };
    printf("\n*****\n");
    printf("                               CRONO SDK Example\n");
    printf("*****\n");

    CRONO_Constr(&crono);
    CRONO_GetDeviceCount(crono, &device_count);
    printf("Found %d CRONO devices: \n", device_count);
    for (int d = 0; d < device_count; d++) {
        CRONO_GetDeviceSerial(crono, d, serial);
    }
}

```

```

        printf("%d: %s\n", d+1, serial);
    }
    printf("\nChoose a device:\n");
    scanf("%d", &dev_num);
    getchar();
    CRONO_GetDeviceSerial(crono, dev_num-1, serial);
    err = CRONO_OpenDeviceBySerial(crono, serial);
    if (!err) {
        printf("\nSUCCESSFUL!\nCamera with serial number %s Opened.\n", serial);
    }
    else {
        printf("Error %i !\n", err);
        return -1;
    }
    CRONO_GetDeviceInfo(crono, "", &devInfo);
    switch (devInfo.SensorType)
    {
        case CRONO_8X8:
            printf("Sensor Type: Crono 8x8\n");
            nPixel = 64;
            break;
        case CRONO_128X1:
            printf("Sensor Type: Crono 128x1\n");
            nPixel = 128;
            break;
        default:
            printf("Error, invalid sensor type\n");
            return -1;
    }
    printf("FW version: %d.%d.%d\n", devInfo.FwMajorVersion, devInfo.FwMinorVersion,
devInfo.FwPatchVersion);
    printf("SDK version: %d.%d.%d\n", devInfo.SdkMajorVersion, devInfo.SdkMinorVersion,
devInfo.SdkPatchVersion);
    CntsBuf = (uint32_t*)calloc(nPixel, sizeof(uint32_t));
    HistBuf = (uint32_t*)calloc(4096 * nPixel, sizeof(uint32_t));
    AccHistBuf = (uint32_t*)calloc(4096, sizeof(uint32_t));
    //initial configuration
    //config sensor
    err += CRONO_ConfigSensor(crono, nPixel, 1, 1, &maxFrameStartFrequency);
    //set deadtime
    err += CRONO_SetSpadDeadTime(crono, 50);
    //config acquisition
    err += CRONO_ConfigAcquisition(crono, COUNTING, 0, 0, 0, 1, 0, "");
    //config internal start
    err += CRONO_ConfigFrameStartInternal(crono, maxFrameStartFrequency, NULL);
    //config internal gate
    pattern[0] = 0;
    pattern[1] = 700;
    err += CRONO_ConfigGateInternal(crono, pattern, 2);
    if (err != 0)
    {
        loop = 0;
        printf("Configuration error!");
    }
    while (loop) {
        // ----- MENU -----

    printf("\n*****\n");
    printf(" Choose an option\n");
    printf("*****\n");
    for (int i = 0; i < sizeof(menu_strings) / sizeof(char*); i++) {
        printf("\t%c) %s\n", i + 'a', menu_strings[i]);
    }
    printf("\tq) Quit\n> ");
    c = getchar();
    getchar();
    if (c >= 'a' && c <= 'a' + sizeof(menu_strings) / sizeof(char*) - 1) {

    printf("*****\n");
        printf("%s\n", menu_strings[c - 'a']);

    printf("*****\n");
        }
        switch (c)
        {
            case 'a': //Test live counting mode
                CRONO_ConfigAcquisition(crono, COUNTING, 0, 0, 0, 1, 0, "");
                CRONO_StartAcquisition(crono, 0, 100);
                dataIsNew = 0;
                while (!dataIsNew)
                {
                    SLEEP(10);
                    CRONO_GetLiveData(crono, CntsBuf, NULL, &dataIsNew);
                }
                CRONO_StopAcquisition(crono, 1);
                printf("Acquired counts per pixel:\n");
                for (int i = 0; i < nPixel; i++)

```

```

        {
            printf("%d\n", *(CntsBuf + i));
        }
        printf("\n\tAcquisition status:\n");
        CRONO_GetAcquisitionStatus(crono, &status);
        print_status(status);
        break;
case 'b': //Test live timing mode
    CRONO_ConfigAcquisition(crono, TIMING, 0, 0, 0, 1, 0, "");
    CRONO_StartAcquisition(crono, 0, 1000);
    dataIsNew = 0;
    while (!dataIsNew)
    {
        SLEEP(10);
        CRONO_GetLiveData(crono, CntsBuf, HistBuf, &dataIsNew);
    }
    CRONO_StopAcquisition(crono, 1);
    total = 0;
    for (int j = 0; j < nPixel; j++)
        total += CntsBuf[j];

    printf("\nTotal events in frame: %d\n", total);
    total = 0;
    AccHist(HistBuf, AccHistBuf, nPixel);
    for (int j = 0; j < 4094; j++)
        total += AccHistBuf[j];
    printf("\nTotal events in histogram: %d\n", total);
    printf("Accumulated histogram for all pixels:");
    for (int i = 0; i < 4096; i++)
    {
        printf("%d\n", *(AccHistBuf + i));
    }
    printf("\n\tAcquisition status:\n");
    CRONO_GetAcquisitionStatus(crono, &status);
    print_status(status);
    break;
case 'c': //Test acquisition timing mode

    CRONO_ConfigAcquisition(crono, TIMING, 1, 1, 1, 0, 0, "");
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    SLEEP(1000);
    err = CRONO_GetAcquisitionStatus(crono, &status);
    printf("\nAvailable histograms: %lld\n", status.availableHistogramCount);
    printf("Available frames: %lld\n", status.availableFramesCount);
    printf("Available time-tags: %lld\n", status.availableTimeTagsCount);
    free(AccHistBuf);
    AccHistBuf = (uint32_t*)calloc(4096, sizeof(uint32_t));
    total = 0;
    for (int i = 0; i < status.availableHistogramCount; i++)
    {
        CRONO_GetHistogramData(crono, HistBuf, 1, &num);
        if (num != 1)
            break;
        AccHist(HistBuf, AccHistBuf, nPixel);
    }
    for (int j = 0; j < 4096; j++)
        total += AccHistBuf[j];
    printf("\nTotal events in histogram: %d\n", total);
    break;
case 'd': //Test holdoff effect

    for (int h = 50; h < 151; h = h + 50)
    {
        printf("\n ** Deadtime %d ns **\n", h);
        CRONO_SetSpadDeadTime(crono, h);
        CRONO_ConfigAcquisition(crono, COUNTING, 1, 0, 0, 0, 0, "");
        CRONO_StartAcquisition(crono, 1000, 1000);
        CRONO_StopAcquisition(crono, 0);
        while (status.ongoing)

```

```

        {
            SLEEP(100);
            CRONO_GetAcquisitionStatus(crono, &status);
        }
        print_status(status);
        printf("\n\nAvailable accumulated frames: %lld\n",
status.availableFramesCount);
        CRONO_GetFramesData(crono, CntsBuf, 1, &num);
        if (num != 1)
            break;
        average = meanImage(CntsBuf, nPixel);
        printf("\n\nAverage counts: %.2f at %d ns deadtime\n\n", average,h);
        printf("-----\n");
    }

    break;
case 'e': // Save settings to file

    err = CRONO_SaveSettingsToFile(crono, "crono_example.ccf");
    if (err)
    {
        printf("\n\nCannot save settings!\n");
        break;
    }
    printf("\n\nSettings saved to file crono_example.ccf\n");
    break;
case 'f': // Load settings from file
    CRONO_SetSpadDeadTime(crono, 42);
    CRONO_GetCameraSettings(crono, &settings);
    printf("\n\nActual deadtime: %d ns\n", settings.deadTimeNs);
    err = CRONO_LoadSettingsFromFile(crono, "crono_example.ccf");
    if (err)
    {
        printf("\n\nCannot load settings!\n");
        break;
    }
    printf("\n\nSettings loaded from file crono_example.ccf\n");
    CRONO_GetCameraSettings(crono, &settings);
    printf("\n\nLoaded deadtime: %d ns\n", settings.deadTimeNs);

    break;
case 'g': // Configure outputs

    //config outputs: laser, start, gate
    CRONO_ConfigOutputs(crono, 0, 0, 0);
    //config laser pattern: 50ns off, 10ns on
    pattern[0] = 50;
    pattern[1] = 10;
    CRONO_ConfigLaserSync(crono, pattern, 2);
    //config gate pattern
    pattern[0] = 0;
    pattern[1] = 200;
    CRONO_ConfigGateInternal(crono, pattern, 2);
    CRONO_ConfigAcquisition(crono, TIMING, 0, 0, 0, 1, 0, "");
    printf("Acquisition will now start and run for 10s.\n\n");
    printf("Output 1 is configured for laser sync: 50ns off, 10ns on.\n");
    printf("Output 2 is configured for start sync at %3f MHz.\n",
maxFrameStartFrequency*1e-6);
    printf("Output 3 is configured for gate: 0ns off, 200ns on.\n\n");
    CRONO_StartAcquisition(crono, 0, 1000);
    for (int i = 1; i <= 10; i++)
    {
        SLEEP(1000);
        printf("%ds elapsed...\n",i);
    }

    CRONO_StopAcquisition(crono, 1);
    printf("\n\nAcquisition ended.\n");
    break;
case 'h': // Acquire with external start

    printf("Apply an external start signal at max frequency of 1MHz, and amplitude of
3.3V, and press ENTER when ready.\n");
    printf("Note that acquisition will not progress if the signal is not applied or
paused.\n");

    getchar();
    CRONO_ConfigFrameStartExternal(crono, 1000000, 1.5, 1, 10);
    CRONO_ConfigAcquisition(crono, TIMING, 0, 1, 0, 0, 0, "");
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();

```

```

        printf(" *****\n");
        printf(" * Acquisition status *\n");
        printf(" *****\n");
        print_status(status);
        CRONO_GetInputFrequencies(crono, &extStartFreq, &extGateFreq,
&extTrigFreq);
        printf("External start frequency: %.3f MHz\n", extStartFreq*1e-6);
    }
    else
        printf("\nError %d", err);
}
CRONO_StopAcquisition(crono, 0);
//restore internal start
CRONO_ConfigFrameStartInternal(crono, maxFrameStartFrequency, NULL);
break;
case 'i': // Acquire with external gate and aux-input timetagging

    CRONO_ConfigFrameStartInternal(crono, 1000000, NULL);
    CRONO_ConfigGateExternal(crono, 1.5, 0);
    CRONO_ConfigAuxiliaryInputs(crono, OFF, RISING_EDGE, 0);
    printf("Camera is now configured for using external gate and time tagging pulses on
trigger input.\n");
    printf("Apply an external gate signal at max frequency of 100MHz, and amplitude of
3.3V, and press ENTER when ready.\nNote that there are timetags due to detected photons.");
    getchar();
    CRONO_ConfigAcquisition(crono, TIMING, 1, 1, 1, 0, 0, "");
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
            CRONO_GetInputFrequencies(crono, &extStartFreq, &extGateFreq,
&extTrigFreq);
            printf("External gate frequency: %.3f MHz\n", extGateFreq*1e-6);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    printf("\nAvailable histograms: %lld\n", status.availableHistogramCount);
    printf("Available frames: %lld\n", status.availableFramesCount);
    printf("Available time-tags: %lld\n", status.availableTimeTagsCount);
    printf("Now remove gate signal and press ENTER when ready. Note there are no more
timetags\n");
    getchar();
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
            CRONO_GetInputFrequencies(crono, &extStartFreq, &extGateFreq,
&extTrigFreq);
            printf("External gate frequency: %.3f MHz\n", extGateFreq*1e-6);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    printf("\nAvailable histograms: %lld\n", status.availableHistogramCount);
    printf("Available frames: %lld\n", status.availableFramesCount);
    printf("Available time-tags: %lld\n", status.availableTimeTagsCount);
    printf("Now apply a 3.3V signal to the trigger input and press ENTER when
ready.\nNote that, despite no gate is applied, there are now timetags due to pulses into trigger
input.\n");
    getchar();
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);

```

```

        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
            CRONO_GetInputFrequencies(crono, &extStartFreq, &extGateFreq,
&extTrigFreq);

            printf("External gate frequency: %.3f MHz\n", extGateFreq*1e-6);
            printf("Aux (trigger-in) frequency: %.3f MHz\n", extTrigFreq*1e-6);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    printf("\nAvailable histograms: %lld\n", status.availableHistogramCount);
    printf("Available frames: %lld\n", status.availableFramesCount);
    printf("Available time-tags: %lld\n", status.availableTimeTagsCount);
    printf("Aux B counts: %lld\n", status.auxBitBCount);
    //restore internal gate and no aux inputs
    CRONO_ConfigGateInternal(crono, pattern, 2);
    CRONO_ConfigAuxiliaryInputs(crono, OFF, OFF, 0);
    //restore internal start frequency
    CRONO_ConfigFrameStartInternal(crono, maxFrameStartFrequency, NULL);
    break;
case 'j': // Use external trigger
    printf("Camera is now in triggered mode, for each trigger pulse 50 frames will be
acquired up to 500 frames.\n");
    printf("Acquisition will now start.\n");
    printf("Apply single trigger pulses and note that captured frames number increases
accordingly, up to 500, and then they are downloaded and decoded.\n");
    printf("Press ENTER when ready.\n");
    getchar();
    CRONO_ConfigOutputs(crono, 0, 0, 0);
    CRONO_ConfigTriggerExternal(crono, 50);
    CRONO_ConfigAcquisition(crono, TIMING, 0, 1, 0, 1, 0, "");
    CRONO_StartAcquisition(crono, 500, 50);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    //restore internal trigger
    CRONO_ConfigTriggerInternal(crono);
    break;
case 'k': //Save raw data to file, reload it and count events in histograms
    //acquiring data
    printf("\nAcquiring data and saving to file acquisition.raw\n");
    printf("\nPress ENTER to start the acquisition and then reopen the file.");
    getchar();
    //configure to only save raw data to file
    CRONO_ConfigAcquisition(crono, TIMING, 0, 0, 0, 0, 0, "acquisition.raw.dat");
    CRONO_StartAcquisition(crono, 100000, 1000);
    CRONO_GetAcquisitionStatus(crono, &status);
    while (status.ongoing)
    {
        SLEEP(100);
        err = CRONO_GetAcquisitionStatus(crono, &status);
        if (!err)
        {
            clear_screen();
            printf(" *****\n");
            printf(" * Acquisition status *\n");
            printf(" *****\n");
            print_status(status);
        }
        else
            printf("\nError %d", err);
    }
    CRONO_StopAcquisition(crono, 0);
    //open raw data as data converter
    printf("\nRaw file acquisition.raw.dat created. Now closing physical device and
opening the file.\n");

```

```

CRONO_Destr(crono);
crono = NULL;
CRONO_Constr(&crono);
err = CRONO_OpenDeviceAsDataConverter(crono, "acquisition.raw.dat");
if (!err) {
    printf("\nSUCCESSFUL!\nRaw file acquisition.raw.dat opened.\n");
}
else {
    printf("Error %i !\n", err);
    loop = 0;
    break;
}
//get acquisition info and settings to properly configure playback
CRONO_GetAcquisitionInfo(crono, &acqInfo);
printf("Acquisition time: %s\n", acqInfo.DateTime);
printf("Number of frames: %lld\n", acqInfo.acquiredFramesCount);
CRONO_GetCameraSettings(crono, &settings);
printf("\nPress ENTER to start reading the raw file.");
getchar();
//configure and run playback
CRONO_ConfigAcquisition(crono, settings.countingMode, 1, 1, 1, 0, 0, "");
CRONO_StartAcquisition(crono, acqInfo.acquiredFramesCount, 1000);
CRONO_GetAcquisitionStatus(crono, &status);
while (status.ongoing)
{
    SLEEP(100);
    err = CRONO_GetAcquisitionStatus(crono, &status);
    if (!err)
    {
        clear_screen();
        printf(" *****\n");
        printf(" * Acquisition status *\n");
        printf(" *****\n");
        print_status(status);
    }
    else
        printf("\nError %d", err);
}
CRONO_StopAcquisition(crono, 0);
err = CRONO_GetAcquisitionStatus(crono, &status);
printf("\nAvailable histograms: %lld\n", status.availableHistogramCount);
printf("Available frames: %lld\n", status.availableFramesCount);
printf("Available time-tags: %lld\n", status.availableTimeTagsCount);
free(AccHistBuf);
AccHistBuf = (uint32_t*)calloc(4096, sizeof(uint32_t));
total = 0;
for (int i = 0; i < status.availableHistogramCount; i++)
{
    CRONO_GetHistogramData(crono, HistBuf, 1, &num);
    if (num != 1)
        break;
    AccHist(HistBuf, AccHistBuf, nPixel);
}
for (int j = 0; j < 4096; j++)
    total += AccHistBuf[j];
printf("\nTotal events in histogram: %d\n", total);
//reopen physical device
CRONO_Destr(crono);
crono = NULL;
CRONO_Constr(&crono);
err = CRONO_OpenDeviceBySerial(crono, serial);
if (!err) {
    printf("\nSUCCESSFUL!\nCamera with serial number %s opened.\n", serial);
}
else {
    printf("Error %i !\n", err);
    loop = 0;
    break;
}
//reconfigure physical device
err += CRONO_ConfigSensor(crono, nPixel, 1, 1, &maxFrameStartFrequency);
err += CRONO_SetSpadDeadTime(crono, 50);
err += CRONO_ConfigAcquisition(crono, COUNTING, 0, 0, 0, 1, 0, "");
err += CRONO_ConfigFrameStartInternal(crono, maxFrameStartFrequency, NULL);
pattern[0] = 0;
pattern[1] = 700;
err += CRONO_ConfigGateInternal(crono, pattern, 2);
if (err != 0)
{
    loop = 0;
    printf("Configuration error!");
}
break;
case 'q':
    loop = 0;
    break;
}

```

```
    }  
// Destructors                                     //  
  
    if(crono)  
        CRONO_Destr(crono);  
    crono = NULL;  
    free(CntsBuf);  
    free(HistBuf);  
    free(AccHistBuf);  
    printf("Press ENTER to continue\n");  
    getchar();  
    return 0;  
}
```

Index

- ACQ_EXT_TRIG_TIMEOUT
 - Crono SDK custom Types, 10
- ACQ_ONGOING
 - Crono SDK custom Types, 10
- BOTH_EDGES
 - Crono SDK custom Types, 9
- Constructor, destructor, 11
 - CRONO_Constr, 11
 - CRONO_Destr, 12
- COUNTING
 - Crono SDK custom Types, 10
- Crono device access functions., 12
 - CRONO_GetDeviceCount, 12
 - CRONO_GetDeviceInfo, 13
 - CRONO_GetDeviceSerial, 13
 - CRONO_OpenDeviceAsDataConverter, 14
 - CRONO_OpenDeviceBySerial, 15
- Crono device configuration functions., 15
 - CRONO_ConfigAcquisition, 16
 - CRONO_ConfigSensor, 16
 - CRONO_GetActivePixels, 17
 - CRONO_GetCameraSettings, 18
 - CRONO_GetSensorType, 18
 - CRONO_LoadCameraSettings, 18
 - CRONO_LoadSettingsFromFile, 19
 - CRONO_SaveSettingsToFile, 19
 - CRONO_SetSpadDeadTime, 20
- Crono device control signal management functions., 26
 - CRONO_ConfigAuxiliaryInputs, 26
 - CRONO_ConfigFrameStartExternal, 27
 - CRONO_ConfigFrameStartInternal, 27
 - CRONO_ConfigGateExternal, 28
 - CRONO_ConfigGateInternal, 29
 - CRONO_ConfigLaserSync, 29
 - CRONO_ConfigOutputs, 30
 - CRONO_ConfigTriggerExternal, 30
 - CRONO_ConfigTriggerInternal, 31
 - CRONO_GetInputFrequencies, 31
- Crono device data retrieval functions., 20
 - CRONO_GetAcquisitionInfo, 21
 - CRONO_GetAcquisitionStatus, 21
 - CRONO_GetFramesData, 22
 - CRONO_GetHistogramData, 23
 - CRONO_GetLiveData, 23
 - CRONO_GetTimeTagsData, 24
 - CRONO_StartAcquisition, 24
 - CRONO_StopAcquisition, 25
- Crono SDK custom Types, 5
- ACQ_EXT_TRIG_TIMEOUT, 10
- ACQ_ONGOING, 10
- BOTH_EDGES, 9
- COUNTING, 10
- CRONO_128X1, 11
- CRONO_8X8, 11
- CRONO_AuxInputMode, 9
- CRONO_H, 8
- CRONO_PixelMode, 9
- CRONO_Return, 10
- CRONO_SensorType, 11
- cronoBool, 9
- DEVICE_ALREADY_OPEN, 10
- DEVICE_NOT_FOUND, 10
- DEVICE_NOT_OPEN, 10
- DEVICE_OPEN_AS_CONVERTER, 10
- FALLING_EDGE, 9
- FPGA_FAILED, 10
- FPGA_TIMEOUT, 10
- INVALID_CHANNEL, 10
- INVALID_CONFIG, 10
- INVALID_DATA_FILE, 10
- INVALID_DEAD_TIME, 10
- INVALID_DIVISION_FACTOR, 10
- INVALID_ENUM, 10
- INVALID_FRAME_BURST_COUNT, 10
- INVALID_GATE_LENGTH, 10
- INVALID_HALVES_CONFIG, 10
- INVALID_LASER_SYNC_LENGTH, 10
- INVALID_PIXEL_NUMBER, 10
- INVALID_START_RATE, 10
- INVALID_THRESHOLD, 10
- LOGIC_VERSION_FAIL, 10
- NO_ERR, 10
- NO_SUCH_FILE_OR_DIRECTORY, 10
- NULL_POINTER, 10
- OFF, 9
- PATTERN_TOO_LONG, 10
- POWER_FAILED, 10
- RAM_FULL, 10
- RISEING_EDGE, 9
- SENSOR_FAILED, 10
- TIMING, 10
- Crono SDK defines, 4
 - MAX_DATETIME_LENGTH, 4
 - MAX_PATTERN_LENGTH, 4
 - MAX_SERIAL_LENGTH, 4
 - MIN_GATE_OFF, 5
 - MIN_GATE_ON, 5

- CRONO_128X1
 - Crono SDK custom Types, [11](#)
- CRONO_8X8
 - Crono SDK custom Types, [11](#)
- CRONO_AcquisitionInfo, [6](#)
- CRONO_AcquisitionStatus, [7](#)
- CRONO_AuxInputMode
 - Crono SDK custom Types, [9](#)
- CRONO_CameraSettings, [7](#)
- CRONO_ConfigAcquisition
 - Crono device configuration functions., [16](#)
- CRONO_ConfigAuxiliaryInputs
 - Crono device control signal management functions., [26](#)
- CRONO_ConfigFrameStartExternal
 - Crono device control signal management functions., [27](#)
- CRONO_ConfigFrameStartInternal
 - Crono device control signal management functions., [27](#)
- CRONO_ConfigGateExternal
 - Crono device control signal management functions., [28](#)
- CRONO_ConfigGateInternal
 - Crono device control signal management functions., [29](#)
- CRONO_ConfigLaserSync
 - Crono device control signal management functions., [29](#)
- CRONO_ConfigOutputs
 - Crono device control signal management functions., [30](#)
- CRONO_ConfigSensor
 - Crono device configuration functions., [16](#)
- CRONO_ConfigTriggerExternal
 - Crono device control signal management functions., [30](#)
- CRONO_ConfigTriggerInternal
 - Crono device control signal management functions., [31](#)
- CRONO_Constr
 - Constructor, destructor, [11](#)
- CRONO_Destr
 - Constructor, destructor, [12](#)
- CRONO_DeviceInfo, [6](#)
- CRONO_GetAcquisitionInfo
 - Crono device data retrieval functions., [21](#)
- CRONO_GetAcquisitionStatus
 - Crono device data retrieval functions., [21](#)
- CRONO_GetActivePixels
 - Crono device configuration functions., [17](#)
- CRONO_GetCameraSettings
 - Crono device configuration functions., [18](#)
- CRONO_GetDeviceCount
 - Crono device access functions., [12](#)
- CRONO_GetDeviceInfo
 - Crono device access functions., [13](#)
- CRONO_GetDeviceSerial
 - Crono device access functions., [13](#)
- CRONO_GetFramesData
 - Crono device data retrieval functions., [22](#)
- CRONO_GetHistogramData
 - Crono device data retrieval functions., [23](#)
- CRONO_GetInputFrequencies
 - Crono device control signal management functions., [31](#)
- CRONO_GetLiveData
 - Crono device data retrieval functions., [23](#)
- CRONO_GetSensorType
 - Crono device configuration functions., [18](#)
- CRONO_GetTimeTagsData
 - Crono device data retrieval functions., [24](#)
- CRONO_H
 - Crono SDK custom Types, [8](#)
- CRONO_LoadCameraSettings
 - Crono device configuration functions., [18](#)
- CRONO_LoadSettingsFromFile
 - Crono device configuration functions., [19](#)
- CRONO_OpenDeviceAsDataConverter
 - Crono device access functions., [14](#)
- CRONO_OpenDeviceBySerial
 - Crono device access functions., [15](#)
- CRONO_PixelMode
 - Crono SDK custom Types, [9](#)
- CRONO_Return
 - Crono SDK custom Types, [10](#)
- CRONO_SaveSettingsToFile
 - Crono device configuration functions., [19](#)
- CRONO_SDK.h, [33](#)
- CRONO_SensorType
 - Crono SDK custom Types, [11](#)
- CRONO_SetSpadDeadTime
 - Crono device configuration functions., [20](#)
- CRONO_StartAcquisition
 - Crono device data retrieval functions., [24](#)
- CRONO_StopAcquisition
 - Crono device data retrieval functions., [25](#)
- cronoBool
 - Crono SDK custom Types, [9](#)
- DEVICE_ALREADY_OPEN
 - Crono SDK custom Types, [10](#)
- DEVICE_NOT_FOUND
 - Crono SDK custom Types, [10](#)
- DEVICE_NOT_OPEN
 - Crono SDK custom Types, [10](#)
- DEVICE_OPEN_AS_CONVERTER
 - Crono SDK custom Types, [10](#)
- FALLING_EDGE
 - Crono SDK custom Types, [9](#)
- FPGA_FAILED
 - Crono SDK custom Types, [10](#)
- FPGA_TIMEOUT
 - Crono SDK custom Types, [10](#)
- INVALID_CHANNEL

- Crono SDK custom Types, [10](#)
- INVALID_CONFIG
 - Crono SDK custom Types, [10](#)
- INVALID_DATA_FILE
 - Crono SDK custom Types, [10](#)
- INVALID_DEAD_TIME
 - Crono SDK custom Types, [10](#)
- INVALID_DIVISION_FACTOR
 - Crono SDK custom Types, [10](#)
- INVALID_ENUM
 - Crono SDK custom Types, [10](#)
- INVALID_FRAME_BURST_COUNT
 - Crono SDK custom Types, [10](#)
- INVALID_GATE_LENGTH
 - Crono SDK custom Types, [10](#)
- INVALID_HALVES_CONFIG
 - Crono SDK custom Types, [10](#)
- INVALID_LASER_SYNC_LENGTH
 - Crono SDK custom Types, [10](#)
- INVALID_PIXEL_NUMBER
 - Crono SDK custom Types, [10](#)
- INVALID_START_RATE
 - Crono SDK custom Types, [10](#)
- INVALID_THRESHOLD
 - Crono SDK custom Types, [10](#)
- LOGIC_VERSION_FAIL
 - Crono SDK custom Types, [10](#)
- MAX_DATETIME_LENGTH
 - Crono SDK defines, [4](#)
- MAX_PATTERN_LENGTH
 - Crono SDK defines, [4](#)
- MAX_SERIAL_LENGTH
 - Crono SDK defines, [4](#)
- MIN_GATE_OFF
 - Crono SDK defines, [5](#)
- MIN_GATE_ON
 - Crono SDK defines, [5](#)
- NO_ERR
 - Crono SDK custom Types, [10](#)
- NO_SUCH_FILE_OR_DIRECTORY
 - Crono SDK custom Types, [10](#)
- NULL_POINTER
 - Crono SDK custom Types, [10](#)
- OFF
 - Crono SDK custom Types, [9](#)
- PATTERN_TOO_LONG
 - Crono SDK custom Types, [10](#)
- POWER_FAILED
 - Crono SDK custom Types, [10](#)
- RAM_FULL
 - Crono SDK custom Types, [10](#)
- RISING_EDGE
 - Crono SDK custom Types, [9](#)
- SENSOR_FAILED
 - Crono SDK custom Types, [10](#)
- TIMING
 - Crono SDK custom Types, [10](#)