Przedsiębiorstwo Badawczo-Produkcyjne
OPTEL Sp. z o.o.
ul. Morelowskiego 30
PL-52-429 Wrocław
tel.: +48 (071) 329 68 54
fax.: +48 (071) 329 68 52
e-mail: optel@optel.pl
http://www.optel.pl
Wrocław

OPCARD ver. 2.2
PCI Ultrasonic Testing Card

Functional Description
and
API Functions Description

Rev.: 2.2
Date: 2016-08-08

1. General features of OPCARD 2.2

Analog

- Very low noise preamplifier (0.74 nV/√(Hz))
- High gain - up to +89 dB
- Wide bandwidth: from 0.5 MHz up to 25 MHz
- Switchable analog filters – 16 combinations (4x Low - Pass and 4x High - Pass)
- Support fast TGC (Time Gain Compensation)
- 2 analog inputs: PE and TT (Pulse Echo, Through Transmission)
- Input attenuator (-20dB)

Pulser

- Integrated step-pulser with fluently regulated amplitude up to 360V
- Very short fall time ≤ 20ns

Digital

- Up to 100 MHz sampling frequency, 8-bit resolution (internal resolution: 10-bit)
- Up to 512k x 16-bit samples memory
- 256k x 8-bit memory for TGC
- 8-bit, 100MSPS DAC for TGC
- 12 I/O line (6 Inputs and 6 Outputs), inputs for Encoder counters, clock outputs
- Expansion Connector for extra functions daughter-board (interfaces (RS232, RS485, SPI, I2C), analog I/O, digital I/O, encoders inputs etc.)

Data processing

- Acquisition depth settings from 4 up to 262088 (256k) samples in 4 sample step
- Delay setting from 0 up to 65535 sample periods in 1 sample period step
- Wide sampling frequency settings
- Fluently regulated gain with step up to 0.5 dB
- Hardware data processing: absolute, analog envelope (*)
- Hardware averaging (*)
- Hardware digital filter (*)
- Triggering (single shot, internal timer or from external inputs)
- PRF up to 20000 (*), up to 100000 with multiplexer (*)
- Data transfer speed - up to 132MB/s in burst using DMA

Size and external connectors

- Standard PCI short card (174.63mm x 106.68mm)
- BNC or Lemo analog connectors
- DB15 connector for I/O and power lines

(*) - details for request

2. Detailed parameters

Analog parameters:

- Input Amplifier Gain: from -31 dB to 65 dB (step 0.5 dB, error +/- 0.3dB)
- Input Post-Amplifier: off or +24dB
- Input Attenuator: off or -20dB
- Input Range: +/- 275mV (+/- 2.5V with Attenuator active)
- Full Bandwidth: 0.5 MHz - 25 MHz (-3dB)
- Input Impedance 50 Ohm, 10pF
- Switchable Filters (-3dB) 0.5 - 6MHz, 0.5 - 10MHz, 0.5 - 15MHz, 0.5 - 25MHz,
  1 – 6MHz, 1 - 10MHz, 1 - 15MHz, 1 - 25MHz,
  2 - 6MHz, 2 – 10MHz, 2 – 15MHz, 2 - 25MHz,
  4 - 6MHz, 4 - 10MHz, 4 - 15MHz, 4 – 25MHz

Pulser:

- Pulse Voltage: off (0V) to 360V (Positive pulse, Short circuit step pulser)
- Charging Time: regulated from 0 to 6.3 us in 0.1us steps
- Fall Time: ≤ 20 ns
- Pulse Duration: Short circuit, bandwidth up to ~50MHz
- Output impedance: < 1 Ohm

A/D data acquisition:

- Resolution: 8 bit (10 bit internally)
- Sampling Freq: 100M, 50M, 33.3M, 25M, 20M, 16.7M, 14.3M, 12.5M, 11.1M, 10M, 9.1M, 8.3M, 7.7M, 7.14M and 6.67MHz
- Data Buffer: from 4 to 262088 (256k) samples in step of 1
- Delay Time: Post trigger from 0 to 65535 sample periods in step of 1

Hardware data processing(*):

- Hardware Positive, Negative, Absolute data processing
- Hardware averaging
- Hardware digital filtering

DAC (TGC) with arbitrary waveform generator:

- Frequency: 100MHz
- Resolution: 8 bit
- Max. gain charging: 48 dB per sample

Trigger:

- Application trigger rate Software - controlled
- On-board trigger timer (*)
- External trigger 2x inputs, TTL Signal
- Trigger output signal TTL Signal

Counters / Input for incremental encoder:

- Counters for Incremental Encoders 2 channel, 32-bit

Connectors:

Analog inputs: 2 x BNC or Lemo connectors

DB15 connector:

Inputs (shared functions):

- 6 general purpose inputs (TTL levels)
- inputs for incremental encoders (2-channel, 3 inputs per channel)

- 2 external trigger inputs
- 2 inputs are shared with -12V and +12V power outputs (onboard jumpers)

Outputs (shared functions):

- 6 general purpose output (TTL levels)
- Trigger output signal (Sync Out)

Others:

- +12V, -12V fused supply for powering external devices
- +5V or Vreg (0-9V) output for control/supply external devices
- -12V and +12V outputs are shared with general purpose inputs GPI4 and GPI5

PCI Connector:

- PCI rev2.2, 32-bit, 33MHz, 132MB/s max transfer bandwidth
- Universal connector for 3.3V or 5V PCI sockets

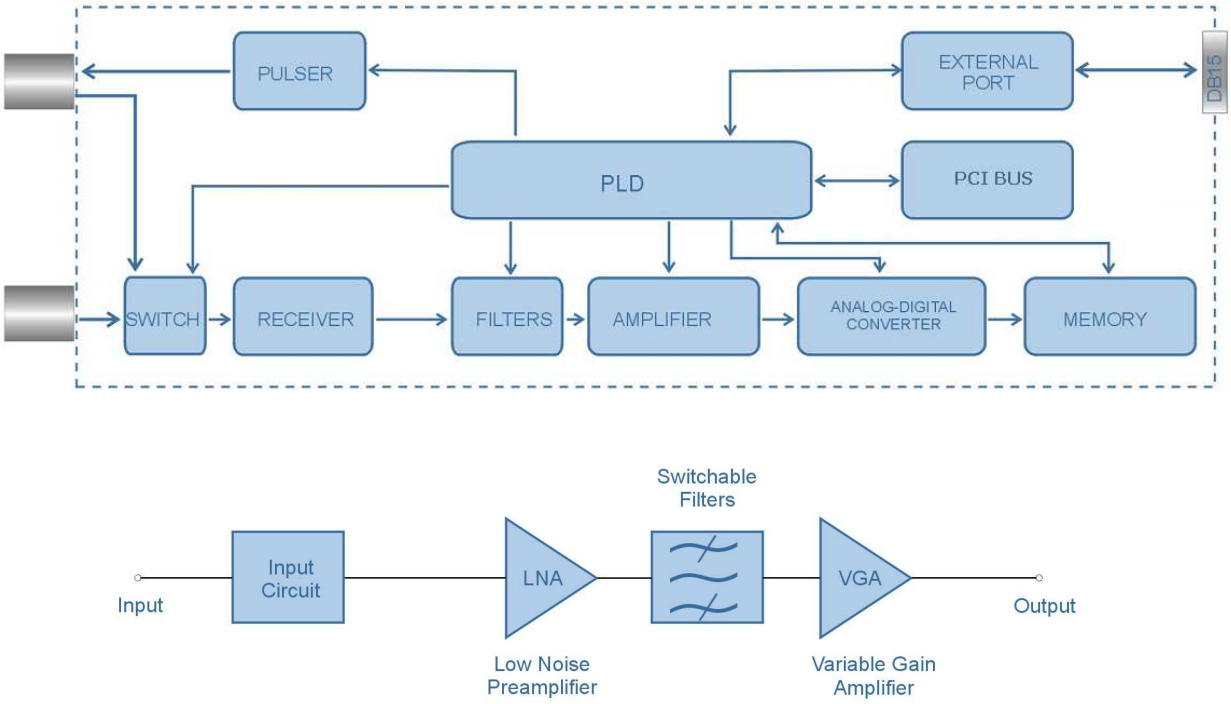
Expansion connector (*):

- 2x10pin 2.54mm connector for special purpose daughter-boards
- +12V, -12V, +5V, +3.3V power pins
- I/O pins serial interfaces (3.3V signal levels): RS232, SPI, I2C and others
- Inputs for general purpose counters
- Interrupt input
- Analog input for A/D converter

Special order

We can develop other hardware functions for special needs of our clients.

Block Diagram



Simplified Block Diagram of analog circuit of the Receiver.

3. Introduction

System of data measurement developed for OPCARD 2.2 was designed to reach real-time data acquisition with speed up to 20 000 measurements per second. It allows buffering of many acquisitions in internal memory and sharing packages of acquisitions with application to enable reading. As a result, high throughput of reading data from the device for different size of depth of measurement is achievable. It is possible when user minimizes communication time (reading the results and communicating with device on control level), e.g. checking flags, power, modules, etc.

Key feature of OPCARD 2.2 is implemented sequence mode. It enables user to store some of the parameters in up to 1024 different sets. During acquisition, subsequent sets are used as parameters of acquisition. This mode allows to develop easier and faster control systems for types of measurement, where periodical changes of settings are necessary.

3.1. Basic definitions

This section contains crucial definitions necessary to explain measurement process. It presents theory and main assumptions used throughout this document.

3.1.1. Acquisition

It is a single measurement sequence consisting consecutively of: sequence of transmission (pulse), countering of delay time, appropriate time of data acquisition (measurement window) and sequence of HEADER writing (registers and results of acquisition). Acquisition settings include inter alia: Sampling Frequency, delay time after pulse transmission, size of measurement window, choice of analogue input, attenuation of the input, analogue filter, sequence index etc.

3.1.2. Measurement window

Measurement Window specifies the number of samples (measurement data) of ana-

logue signal from ADC stored to memory in time of single Acquisition. The number of samples is defined by parameter DEPTH. DEPTH may range from 4 up to 256kB (262088) of samples. This parameter can be specified by function **Opcard_SetBufferDepth**. *NOTE*: **DEPTH set in the device has to be multiple of 4!**

3.1.3. Acquisition frame

Acquisition Frame (in short Frame) – it is the sequence of data consists of samples from one acquisition. Frame is made after each triggering of acquisition and consists consecutively of Header and Measurement Data. The number of bytes in Frame is a sum of Header size and number of Measurement Data - FRAME_SIZE [B] = HEADER_SIZE [B] + DEPTH [samples].

3.1.4. Header

Header – it is the first 52 bytes (HEADER_SIZE) of acquisition frame, which contain information about acquisition. Header consists of inter alia: sequence index and information captured during triggering: positions of Encoders, state of GPI, state of hardware counter of time TIMER and results of hardware measurement gates (Peak-Detectors). Header format is presented in Table 1.

Table 1 – HEADER DESCRIPTION

Byte No	Name	Register Address	Used bits	Description
1	**Start Frame**	- - -	'@' (0x40)	Marker of starting header
2		- - -	(0x78)	
3	SQ index	0xA4	[7:0]	Index sequence
4			[9:8]	
5	Time Stamp	0x1C	[7:0]	Counter of timer register captured on trigger
6			[15:8]	
7		(0x1E)	[23:16]	
8			(0x00)	- reserved
9	GPI Captured		[3:0]	General Purpose Inputs (GPI[5:0]) –

				captured on trigger
10	TrgOvrSource Captured		[2:0]	Trigger Overrun source flags – captured on trigger
11	Trigger Overrun	0x12	[7:0]	Trigger Overrun register – number of lost triggers from last acquisition
12			[15:8]	
13	Encoder 1 Position	0x7C	[7:0]	Encoder 1 Position – captured on trigger
14			[15:8]	
15		(0x7E)	[23:16]	
16			[31:24]	
17	Encoder 1 Position	0x8C	[7:0]	Encoder 2 Position – captured on trigger
18			[15:8]	
19		(0x8E)	[23:16]	
20			[31:24]	
21	Peak Detectors Status	0x44	[7:0]	Peak Detectors Status register
22			[15:8]	
23	PDA Ref	0x50	[7:0]	Gate PDA Reference Position (position for which occurred specified event)
24			[15:8]	
25		(0x52)	[17:16]	
26			[31:24]	Gate PDA Reference Value
27	PDA Max	0x54	[7:0]	Gate PDA Max Position
28			[15:8]	
29		(0x56)	[17:16]	
30			[31:24]	Gate PDA Max Value
31	PDB Ref	0x60	[7:0]	Gate PDB Reference Position (position for which occurred specified event)
32			[15:8]	
33		(0x62)	[17:16]	
34			[31:24]	Gate PDB Reference Value
35	PDB Max	0x64	[7:0]	Gate PDB Max Position
36			[15:8]	
37		(0x66)	[17:16]	
38			[31:24]	Gate PDB Max Value
39	PDC Ref	0x70	[7:0]	Gate PDB Reference Position (position for which occurred specified event)
40			[15:8]	
41		(0x72)	[17:16]	
42			[31:24]	Gate PDB Reference Value
43	PDC Max	0x74	[7:0]	Gate PDB Max Position
44			[15:8]	
45		(0x76)	[17:16]	

46			[31:24]	Gate PDB Max Value
47	Depth	0x38	[7:0]	No of samples in frame (Acquisition Depth)
48			[15:8]	
49		(0x3A)	[17:16]	
50			(0x00)	
51	End of Header	- - -	'/' (0x2F)	Marker of end of frame
52		- - -	'/' (0x2F)	
53	Sample 1		[7:0]	First sample
54	Sample 2		[7:0]	Second sample
...	[...]			
52+ (depth-1)	Sample [depth-1]		[7:0]	
52+depth	Sample [depth]		[7:0]	Last sample

4. Acquisition in real time

Term "Acquisition in real time" generally should be understood as implementing continuous series of Acquisitions without changing any device settings. Acquisitions are usually triggered by external signal, which is being modified in real time, out of application control (triggering by TIMER, Encoders position, etc.). Opcard enables to set several different settings / modes of acquisition. The first subsection (4.1) briefly presents functions used for any possible data acquisition mode. Following subsections (4.2. - 4.2.) contains information about acquisition modes in OPCARD 2.2. Subsection 4.5. provides detailed information about process of preparing Opcard to acquisition and data measurement control. Subsection 4.6. provides information and important notes about FIFO-based data acquisition.

4.1. General notes – basic functions

To enable data acquisition, you need to select source of trigger signal using function **Opcard_SetTriggerSource**. To enable triggering using selected trigger source, use function **Opcard_SetTriggerEnable**.

You can read data using two functions: **Opcard_ReadAllData** or **Opcard_ReadData**. **Opcard_ReadAllData** reads all available data from memory and returns information about data size. Function **Opcard_ReadData** reads only amount of data specified by user.

One frame (data from one acquisition) consists of header and measurement curve. **It is very important that depth has to be multiple of 4! In header, you can find useful information about measurement and various acquisition settings.** Header bytes are documented in Table 1. **Range of returned values of data samples varies from -0.5 to 0.5 V and corresponds to -128 to 127 given in char.**

4.2. 'FIFO' mode vs 'ACK' mode

For ACK mode only one acquisition is stored in memory. Then, Opcard blocks every incoming trigger until you confirm reading data. It is necessary to confirm receiving acquisition data by sending information to the device. It can be done using function **Opcard_SetUploadAck**. You can check for new data in memory using function **Opcard_GetNewDataReady**.

For FIFO mode, reading and writing data follows rules of FIFO communication (*First in - First out*). All acquisitions are stored in memory unless it is full. In this mode, user does not need to confirm data receiving.

ACK mode is slower than FIFO mode due to necessity of confirmation for every data acquisition. FIFO mode allows to read data from many acquisitions in one time. However, in ACK mode, it is easier to control the acquisition process.

To switch between these two modes, use function **Opcard_SetAckMode**.

4.3. 'Normal' mode vs 'Store Disabled' mode

'Normal' mode is a measurement, where device enables user to store headers and data for each acquisition. User can obtain useful information from header and present collected data or process it to achieve additional information.

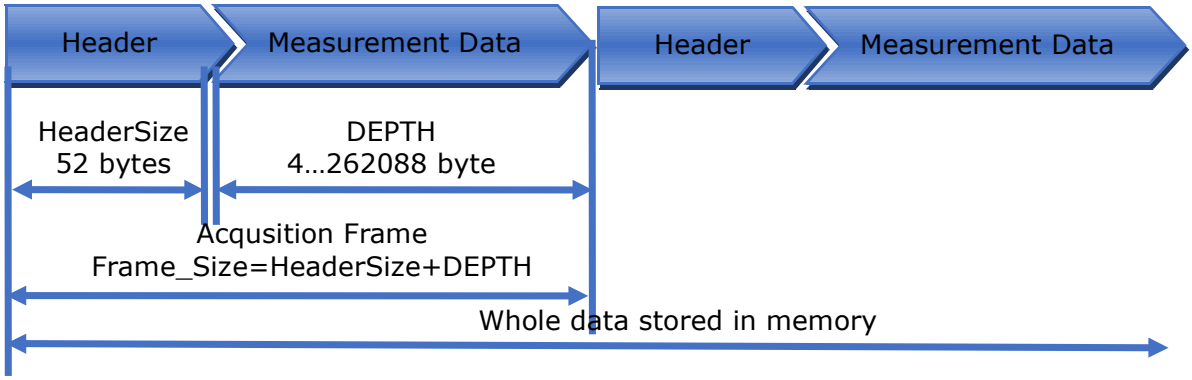


Fig. 1. Organization of data in memory for 'Normal' mode.

During acquisition in 'Store Disabled' mode data from ADC is not stored - only headers are stored in memory. This mode is useful when user wants to make acquisition faster and does not need raw measurement data (e.g. only results of preconfigured Peak Detectors matter).

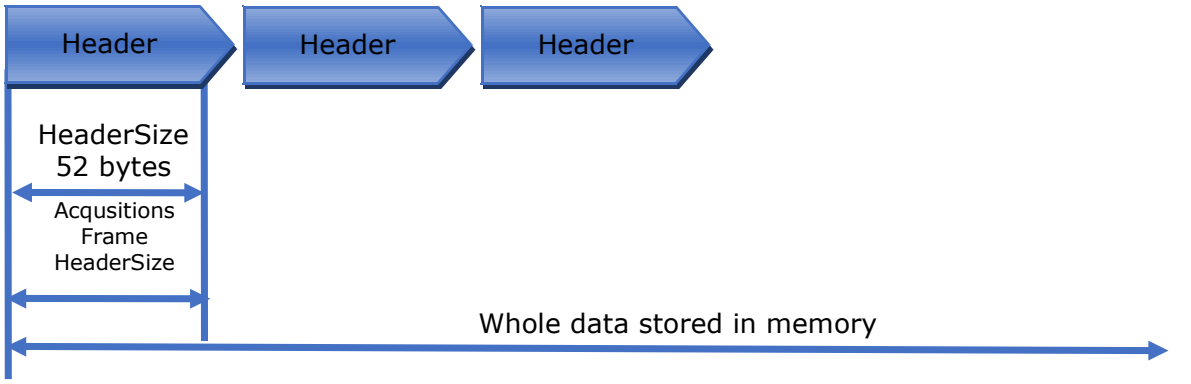


Fig. 2. Organization of data in memory for 'Store Disabled' mode.

'Normal' / 'Store Disable' mode can be switched using bit [9] 'storeDisable' of MEASURE register, which can be controlled using function **Opcard_SetStoreDisabled.**

4.4. 'Single' mode vs 'Sequence' mode

Single mode should be understood as typical acquisition, where for each subsequent acquisition during measurement session is done using single set of parameters. Data acquisition is similar to 'Normal' mode presented in 4.1. (see Fig. 2).
When device works in 'Sequence' mode, it is possible to work with up to 1024 different settings, which are stored in device memory. In 'Sequence' mode, a single data frame is collected for all sequences (each with different, modifiable settings), one after another. After each sequence, **seqIndex** is incremented. When **seqIndex** reaches **seqLength** value, **seqIndex** is set to 0 and acquisitions for all sequences (one after another) are triggered again. Settings specific for each sequence:

- Delay,
- Depth,
- Sampling frequency,
- Gain mode,
- Constant gain value (but not TGC curve – see: Section 5.),
- Data processing mode (raw / absolute),
- 'Store Disable' mode, (only header / header + data stored),
- FIFO / Single data mode,
- All settings for peak detectors (control settings and gates' positions).

During Acquisition in real time, all parameters listed above are being switched after every acquisition to subsequent sequences (see Fig. 3).

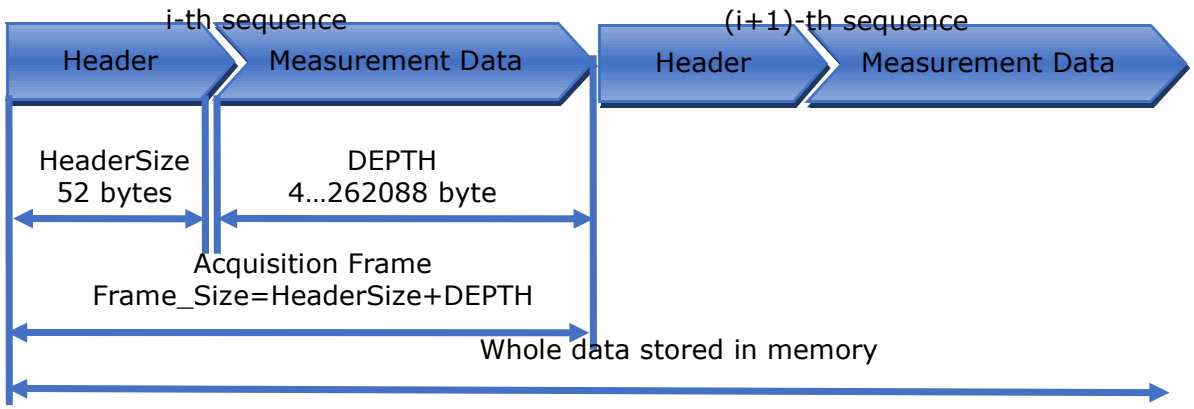


Fig. 3. Org o data in memory for 'Sequence' mode. When index o sequence reaches maximum value (defined by SeqLength), next index is set to value 0. Number o measured data samples c be different for each sequence.

4.5. Configuration, control and data management in real-time acquisition

Preparation and control of Acquisition in real time should be done as follows:

1. If it is the first measurement during session with Opcard, it is recommended to reset Opcard to default parameters using **Opcard_Reset** function.

2. Stop current acquisition by blocking global trigger (bit 'TriggerEnable' in register TRIGGER) – it can be set by function **Opcard_SetTriggerEnable**;

3. Set the Acquisition parameters, which remains the same for all sequences in 'Sequence' mode (global parameters):

   - Transmitter configuration – time and level of transmission (registers PULSE_TIME and PULSE_AMPLITUDE) – it can be set respectively using functions **Opcard_SetPulseVoltage** and **Opcard_SetTimePulse**,
   - Configuration of the analog circuit – selecting input source, input attenuator, post amplifier, filters: low- and high-pass (register ARM_SPI) – it can be set by functions: **Opcard_SetAnalogInput**, **Opcard_SetAttenHilo** and **Opcard_SetAnalogFilters** or **Opcard_SetAll** (which sets all mentioned parameters with a single command),
   - Configuration of Encoders modules (several functions),
   - Configuration of trigger source (TRIGGER) – select source (function **Opcard_SetTriggerSource**) and set parameter(s) specific for selected trigger source (TIMER – **Opcard_SetTimerValues**; XY_DIVIDER – **Opcard_SetCounterXY_TopValue**, **Opcard_SetCounterEnable** and **Opcard_SetCounterReset** for Encoders),
   - Set number of sequences and choose between 'Sequence' and 'Single' mode – it can be done using functions **Opcard_SetSeqLength** and **Opcard_SetSeqEnable**.

4. Select sequence (switch to certain sequence – specified by sequence index) for which you would like to change parameter using function **Opcard_SetSeqIndex**. NOTE: ***In 'Single' mode, Opcard uses parameters set at SeqIndex set by user. It means that during switch between 'Sequence' mode and 'Single' mode device will make acquisit***

5. Set acquisition parameters specific for each sequence (sequence parameters):

- Configuration of parameters of signal acquisition:
    - Delay of acquisition (register DELAY) – **Opcard_SetDelay**,
    - Sampling Frequency (MEASURE) – **Opcard_SetSamplingFreq**,
    - Size of Measurement Window (DEPTH) – **Opcard_SetBufferDepth,**
    - Data hardware processing mode (register DATA_MODE) – **Opcard_SetDataProcMode**,
- Configuration of FIFO / ACK mode (specific for each sequence) using **Opcard_SetAckMode**,
- Configuration of gain – choosing gain mode (constant or with TGC) and gain constant value – it can be set by functions **Opcard_SetGainMode** and **Opcard_SetConstGain**,

NOTE: ***Working with TGC fo***

- Configuration of Peak Detectors – reference level, gate positions and type of crossing level event.

6. If you would like to set parameters for another sequence, repeat point 4 and 5.

7. If you would like to clean FIFO memory, use function **Opcard_SetResetFifo.**

8. Prepare new TGC and send to Opcard using Opcard_WriteData. *NOTE:* ***See Sect***

9. Start acquisition – unlock global trigger using function **Opcard_SetTriggerEnable**.

10. **Receive data from FIFO memory**

The easiest way to receive data is to read all data available to read – you can do it using function **Opcard_ReadAllData**. To reduce communication with device (and increase acquisition speed) you can check number of available data samples / frames using functions **Opcard_GetNewDataReady** (for ACK mode) and **Opcard_GetFifoCnt** (for FIFO). In ACK mode, you can get the information about single frame from one acquisition ready to collect. In FIFO mode, you can check number of bytes ready to read - you can wait to get the specified number of bytes before collecting it.

Additionally, you do not have to read all possible data at one time. The work in FIFO mode (First In - First Out) allows receive only specified number of bytes using function **Opcard_ReadData**. NOTE: ***reading a full data from one acuisition change of parameters of acquisition (example DEPTH) do n***

If there is data remaining in memory before parameter modification, the easiest way to deal with the problem is to reset FIFO memory (**Opcard_SetResetFifo**) after switching parameter or before enabling the triggering. The other option is to ensure reading all data before switching parameter.

Moreover, HEADER can also provide information useful in reaching better control of received data. The most important parameters stored in HEADER are: the sequence index (only in 'Sequence' mode – provides information about sequence index, for which measurements stored in frame were done), information about lost triggers (Trigger Overrun) and DEPTH.

Range of returned values of data samples varies from -0.5 to 0.5 V and corresponds to -128 to 127 given in char.

4.6. FIFO buffer – important notes

Key features of the acquisition in FIFO mode:

1) Allows to execute acquisitions in real time until buffer is not full without necessity of reading the data ('Store Disabled' mode – see: Section 4.3). In this way, high processing speed (PRF) is possible to achieve.

2) Big device FIFO buffer (512 kB) provides stability of PRF – effect of software processing fluctuations and possible decrease of data-reading frequency is reduced.

3) Acquisition has higher priority than read / write operations (communication with Opcard).

4) Each acquisition forms data frame, where first 52 bytes are HEADER and the rest of frame is filled with data samples forming analog signal from ADC.

5) HEADER consists of useful information of given acquisition including: sequence index, position of Encoders during triggering, hardware results of peak detectors (PD) and crossing signal level.

6) Settings provided for acquisition allows for adapting buffering of acquisition to actual requirements of application and hardware capabilities of computer. It is helpful in achieving required processing speed of measurement in real time.

7) Information from HEADER about sequence index allows to match the results to current sequence setting and keep the order of consecutive acquisitions.

5. Gain curve (TGC)

This section briefly describes configuration of Opcard to work using **TGC Gain Curves** during acquisition in FIFO mode. Acquisition in FIFO mode requires preparation of array of TGC gain curves in a special way. Size of TGC buffer must be equal to half of FIFO buffer size (512 / 2 kB).

Each consecutive curve corresponds to consecutive acquisitions, but **one point on TGC memory provides gain for two data samples in measurement data**. Single TGC curve should be prepared as follows:

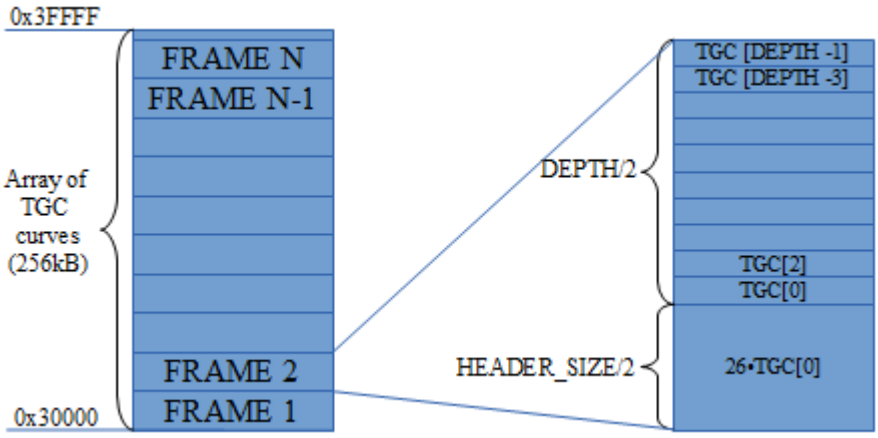
- The first HEADER_SIZE / 2 (= 26) bytes should be written by first value of gain from TGC curve,
- Next bytes should contain TGC curves, and each point from it corresponds to two point of measurement data from one acquisition,
- Formula to get the value of gain from [dB] coded in byte is presented below:

  $Gain[A.U.]=2\times Gain[dB]+70$, where *Gain [A.U.]* ranges from 8 up to 200.

- To set desired Gain[dB] values in TGC curve in the device, use modification of latter formula:

  $$Gain[dB]=\frac{Gain[A.U.]}{2}-35$$

The idea of single TGC curve preparation is presented on the following figure.



Writing the array of TGC curves to device memory should always involve transfer of full array data (256 KB). Partial write to TGC memory may result in invalid work of gain for some acquisitions. Invalid preparation of this array causes invalid work of acquisition, because each acquisition can be done with wrong (e.g. shifted in time domain) gain curve. If only one TGC curve is defined (the rest is undefined), it is recommended to upload TGC buffer containing as many complete copies of this curve as buffer can contain.

In TGC mode, device requires as many TGC curves stored in TGC buffer as buffer can contain (256 KB). When device works in ACK mode or 'Store Disabled' mode, user has to prepare only one TGC **Gain Curve** - for these modes, only the first curve stored in TGC buffer is used by device. However, when user wants to switch to other mode, whole TGC buffer must be filled with TGC curves like described latter before acquisition starts.

To send prepared TGC array, use function **Opcard_WriteData:**

Prototype:

```
DLLReturnType
Opcard_WriteData  (
                        int deviceNr,
                        unsigned int* data,
                        int sizeData
                    );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

data – buffer containing TGC curve points. It is recommended to create array of type *unsigned char;* **data** must be pointer of type *unsigned int* to this array.

sizeData – length of used TGC curve **in data samples**, **for which TGC buffer is created**

Description:

Example: User wants to make measurement using 'Store Disabled' mode containing **DATA_SIZE** = 500 samples. You must:

1. Initialize 256 KB array as **data**.
2. Fill **at least** $\frac{HeaderSize+DataSize}{2}=226$ first elements of this array with Gain [A.U.] values (in Disabled mode device uses the first TGC curve, 1 TGC point provided for 2 FIFO data points including header). It is recommended to create array of *unsigned char;* **data** must be defined as pointer of type *unsigned int* to this array.
3. **sizeData** defines size of filled elements of TGC buffer **including header** – in this example, **sizeData = (HEADER_SIZE + DATA_SIZE)/2** = 226.
4. Call **Opcard_WriteData** (**deviceNr**, **data**, **sizeData**);

NOTE: ***If you work with TGC c***

6. OpCard and OpMux configuration

This section briefly describes configuration of OpCard and OpMux in order to create system designed for complex scans using array of transducers.

1. OpMux must be configured separately and independently from OpCard. OpMux is configured via the serial interface (RS-232 standard) – see manual for OpMux for details of configuration.

2. Trigger for OpCard in sequence mode generates sequence incrementation in OpMux. Before measurement, in the simplest configuration (without TGC), to obtain synchronisation you must:
   a) Set the same value of sequence length on OpMux and OpCard (or sequence length of OpCard is multiple of sequence length of OpMux).
   b) Stop triggering on OpCard (if it has been started).
   c) Reset FIFO (using Opcard_SetResetFifo (deviceNo);).
   d) Using sequence mode, before triggering for the first time, you should reset

sequence index on both OpMux (sending CT 1 – turning on the trigger automatically resets sequence index to 0) and OpCard (Opcard_SetSeqIndex(deviceNo, 0);).

e) Send ACK confirmation to OpCard using Opcard_SetUploadAck (deviceNo)

f) Enable trigger on Opcard (Opcard_SetTriggerEnable (deviceNo, 1);).

g) Any valid trigger should automatically increment sequence index on OpMux and OpCard – there is no need of additional control of sequence index from now on.

Note: Every reset of FiFO requires reconfiguration – you should repeat points b – e.

3. TGC using OpCard and OpMux should be prepared as follows:

a) The first HEADER_SIZE / 2 (= 26) bytes should be written by first value of gain from TGC curve,

b) Next bytes should contain TGC curves, and each point from it corresponds to two point of measurement data from one acquisition,

c) Formula to get the value of gain from [dB] coded in byte is presented below:

$$Gain[A.U.] = 2 \times Gain[dB] + 70$$, where *Gain [A.U.]* ranges from 8 up to 200.

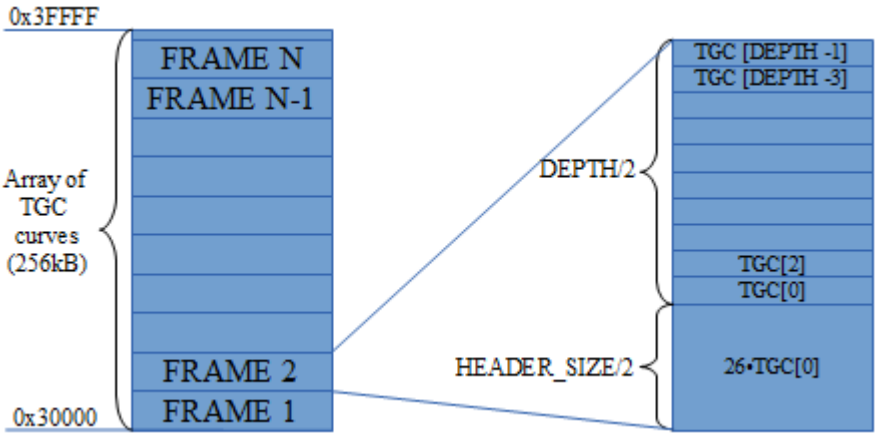
d) To set desired Gain[dB] values in TGC curve in the device, use modification of latter formula:

$$Gain[dB] = \frac{Gain[A.U.]}{2} - 35$$

The idea of single TGC curve preparation is presented on the following figure. Total size of array of TGC curves:

TGC_LENGTH = N * HEADER_SIZE / 2 + (DEPTH_1 + DEPTH_2 + … + DEPTH_N) / 2,

where: DEPTH_1, DEPTH_2, ... , DEPTH_N – size of data for each sequence index



Using TGC in OpCard + OpMux configuration requires meeting additional conditions:

a) Total memory used for storage of all sequence acquisitions on OpCard must be lower than FIFO memory and TGC memory.

b) Using TGC, it is not possible to work in Store Disabled mode.

c) The first sequence (seq index = 0) should be configured to work in ACK mode.

d) Next sequences (seq index = 1 … (SeqLength – 1)) should be configured to work in FIFO mode.

e) In this configuration of sequence, you should get and send confirmation after reception of acquisition for the last sequence index – after every last sequence, flag NewDataReady (which can be checked using Opcard_GetNewDataReady) is set to 1.

f) It is very important to confirm reception of data **after every last sequence** in the following way:

- Check if all data has been collected – if not, read it.
- Reset FIFO (Opcard_SetResetFifo),
- Confirm reception of data (Opcard_SetUploadAck).

Note: If TGC is the same for all sequences, it is possible to configure it in the same way as described at Chapter 5 of the OpCard manual. In this case, requirements presented at point 2. does not have to be met.

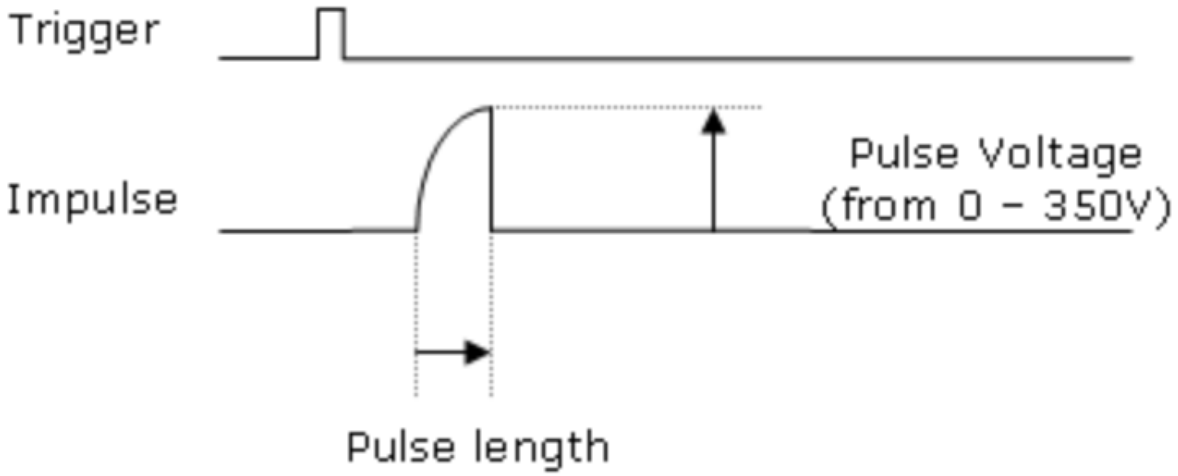
Example:

User wants to configure sequence of 3 acquisitions of size DEPTH_1, DEPTH_2, DEPTH_3.

1) Stop triggering using Opcard_SetTriggerEnable (deviceNo, 0);
2) Configure MUX sequence: ST 1 1 2 2 3 3 – transducers are configured as transmitters and receivers for the same channel (see: manual for OpMux).
3) Set OpCard sequence: Opcard_SetSeqLength (deviceNo, 3).
4) Set mode for each sequence: ACK mode for seqIndex = 0, FIFO mode for other (seqIndex = 1, seqIndex = 2) – Opcard_SetAckMode (deviceNo, ackMode) and other parameters: depth, delay and other parameters for each sequence index on OpCard (0, 1, 2).
5) Reset FIFO – Opcard_SetResetFifo (deviceNo)
6) Reset sequence index for OpMux and OpCard
7) Prepare and send TGC: TGC length = (DEPTH_1 + DEPTH_2 + DEPTH_3) / 2 + 3 * HEADER_SIZE / 2
8) Send ACK confirmation to OpCard using Opcard_SetUploadAck (deviceNo)
9) Enable triggering (Opcard_SetTriggerEnable (deviceNo, 1))
10) This configuration works as follows:
    1) System will do three acquisitions, which will be sent to FIFO.
    2) Now, we must wait for ACK flag – check it using Opcard_NewDataReady (deviceNo).
    3) **Now, measurements will not be made until user confirms reception of data.**
    4) Receive data using Opcard_ReadAllData or Opcard_ReadData
    5) Reset FIFO using Opcard_SetResetFifo (deviceNo)
    6) Send confirmation using Opcard_SetUploadAck (deviceNo).
    7) Points 1 – 6 should be repeated to continue measurement.

7. List of parameters and control functions

Pulse Parameters



Functions and parameters:

Pulser length:

- **Opcard_GetTimePulse** *(int deviceNr, double* timeP)*; - get pulse length (timeP) in µs
- **Opcard_SetTimePulse** *(int deviceNr, double timeP)*; - set pulse length

Pulser Amplitude (range = 0 – about 360 V, scaled in 0...255):

- **Opcard_GetPulseVoltage** *(int deviceNr, int *pulseVoltage)*; - get pulse voltage
- **Opcard_SetPulseVoltage** *(int deviceNr, int pulseVoltage)*; - set pulse voltage

Pulser Enable:

- **Opcard_GetDriverEnable** *(int deviceNr, bool* drvEnable)*; - get mode - enable (0) or disable (1)

- **Opcard_SetDriverEnable** *(int deviceNr, bool drvEnable)*; - set mode - enable (0) or disable (1)

Source (mode of work: Pulse Echo or Through Transmission):

There are two possible modes of work – user can switch analog signal source between PE and TT BNC connector.

Source: 3 – PE connector is analog signal source (Pulse Echo)
1 – TT connector is analog signal source (Through Transmission)

Functions for source:

- **Opcard_SetAnalogInput** *(int deviceNr, int analogInput)***;**
- **Opcard_GetAnalogInput** *(int deviceNr, int* analogInput)***;**

Analog parameters:

Input Amplifier Gain: -31dB to 65 dB (step 0.5 dB, error +/- 0.3dB)
Input Post-Amplifier: off or +24 dB
Input Attenuator: off or -20 dB
Full Bandwidth: 0.5 MHz - 25 MHz (-3dB)
Switchable filters (-3dB): 0.5 – 6 MHz,
0.5 – 10 MHz,
0.5 – 15 MHz,
0.5 – 25 MHz,
1 – 6 MHz,
1 – 10 MHz,
1 – 15 MHz,
1 – 25 MHz,
2 – 6 MHz,
2 – 10 MHz,
2 – 15 MHz,
2 – 25 MHz,
4 – 6 MHz,
4 – 10 MHz,
4 – 15 MHz,
4 – 25 MHz.

Functions for analog parameters:

Filters:

- **Opcard_SetAnalogFilters** *(int deviceNr, int analogFilters);* set analog filter specified latter
- **Opcard_GetAnalogFilters** *(int deviceNr, int* analogFilters)*; get analog filter specified latter

Input Attenuator:

- **Opcard_SetAttenHilo** *(int deviceNr, int attenHilo)*; - set hardware amplification, 0 – off (0 dB), 1 – amplifier (+ 24dB), 2 – attenuator (- 20 dB), 3 – amplifier and attenuator (+ 4 dB)
- **Opcard_GetAttenHilo** *(int deviceNr, int* attenHilo)*; - get hardware amplification, 0 – off (0 dB), 1 – amplifier (+ 24dB), 2 – attenuator (- 20 dB), 3 – amplifier and attenuator (+ 4 dB)

Gain mode – constant value / TGC:

- **Opcard_SetGainMode** *(int deviceNr, int gainMode)*; - set gain mode, 0 – constant gain, 1 – TGC curve
- **Opcard_GetGainMode** *(int deviceNr, int* gainMode)*; - get gain mode.

TGC buffer:

- **Opcard_WriteData** (*int deviceNr, unsigned int* data, int sizeData*); - set TGC curve buffer; *data* is a pointer to TGC curve.

Constant gain:

- **Opcard_GetConstGain** *(int deviceNr, double* constantGain)*; - get gain in dB (-31 … 65, step: 0.5)
- **Opcard_SetConstGain** *(int deviceNr, double constantGain)*; - set gain in dB (-31 … 65, step: 0.5)

A/D data acquisition:

Sampling Frequency (MHz): 100; 50; 33.3; 25; 20; 16.7; 14.3; 12.5;

	11.1; 10; 9.1; 8.3; 7.7; 7.14; 6.67 MHz.
Data Buffer:	from 4 to 262088 (256k) samples in step of 1
Delay Time (Post trigger):	from 0 to 65535 sample periods in step of 1

Functions for A/D data acquisition:

Sampling frequency:

- **Opcard_SetSamplingFreq** *(int deviceNr, int samplingFreq);* *samplingFreq* = 100 / **Sampling frequency in MHz**
- **Opcard_GetSamplingFreq** *(int deviceNr, int* samplingFreq)*;* **Sampling frequency in MHz** = 100 / *samplingFreq*;

Post - trigger (Delay):

- **Opcard_SetDelay** *(int deviceNr, int delayVal)*; set delay in Ts
- **Opcard_GetDelay** *(int deviceNr, int* delay*)*; get delay in Ts

Measurement window (depth of measurement):

- **Opcard_GetBufferDepth** *(int deviceNr, int* depth*)*; get depth in Ts
- **Opcard_SetBufferDepth** *(int deviceNr, int depth);* set depth in Ts

Hardware data processing:

Hardware Positive, Negative, Absolute data processing;

Data acquisition mode: Store data buffer, Store only header (Store Disabled)

3 x measure gate – Peak Detectors (PD) measurement mode:

- Maximum – always when peak is enabled,
- Level (0)– first sample above the level,
- Rising (1)- rising edge,
- Falling (2)– falling edge,
- Transition (3) – rising or falling edge.

Functions for hardware data processing:

Absolute / raw mode:

- **Opcard_SetDataProcMode** *(int deviceNr, bool dataProcMode)*; set data

processing mode: 0 – raw, 1 – absolute

- **Opcard_GetDataProcMode** *(int deviceNr, bool* dataProcMode)*; get data processing mode

Store mode:

- **Opcard_SetStoreDisabled** *(int deviceNr, bool storeDisabled)*; set store data mode: 0 – store acquisition data, 1 – store only header
- **Opcard_GetStoreDisabled** *(int deviceNr, bool* storeDisabled)*; get store data mode

Peak detectors:

- **Opcard_GetPdxMode** *(int deviceNr, int pdNr, int* pdMode);* get PD mode, *pdNr* – peak number, *pdMode* – peak mode (level = 0, rising = 1, falling = 2, transition = 3)
- **Opcard_SetPdxMode** *(int deviceNr, int pdNr, int pdMode); s*et PD mode, *pdNr* – peak number, *pdMode* – peak mode (level, rising, falling, transition)
- **Opcard_GetPdxEnable** *(int deviceNr, int pdNr, bool* pdEnable)*; get status of PD (0 = disabled, 1 = enabled)
- **Opcard_SetPdxEnable** *(int deviceNr, int pdNr, bool pdEnable)*; set status of PD (disabled/enabled)
- **Opcard_GetPdxResult** *(int deviceNr, int pdNr, bool* pdResult)*; get result of PD comparator level (1 = found, 0 = not found)
- **Opcard_GetPdxStart** *(int deviceNr, int pdNr, int* pdStart)*; get start gate position for the PD comparator (in samples)
- **Opcard_SetPdxStart** *(int deviceNr, int pdNr, int pdStart)*; set start gate position for the PD comparator (in samples)
- **Opcard_GetPdxStop** *(int deviceNr, int pdNr, int* pdStop)*; get stop gate position for the PD comparator (in samples)
- **Opcard_SetPdxStop** *(int deviceNr, int pdNr, int pdStop)*; set stop gate position for the PD comparator (in samples)
- **Opcard_GetPdxLevelPosition** *(int deviceNr, int pdNr, int* pdLevelPos);* get position of level crossing for PD
- **Opcard_GetPdxLevelValue** *(int deviceNr, int pdNr, char* pdLevelVal);*

get level value of PD comparator

- **Opcard_SetPdxLevelValue** *(int deviceNr, int pdNr, char pdLevelVal);* set level value of PD comparator
- **Opcard_SetPdxLevelPosition** *(int deviceNr, int pdNr, int start, int stop, int level);* set start and stop gates, and crossing level for PD
- **Opcard_GetPdxMaxPosition** *(int deviceNr, int pdNr, int* pdMaxPos)*; get position of maximum for PD
- **Opcard_GetPdxMaxValue** *(int deviceNr, int pdNr, char* pdMaxVal)*; get maximum value for PD

Trigger:

On-board trigger timer;
Application trigger rate Software controlled;
External trigger 2x inputs, TTL Signal.

Functions:

- **Opcard_GetTriggerSource** *(int deviceNr, int* trgSource)*; get trigger source
- **Opcard_SetTriggerSource** *(int deviceNr, int trgSource)*; set trigger source
- **Opcard_GetTriggerEnable** *(int deviceNr, bool* trgEnable)*; get global trigger enable
- **Opcard_SetTriggerEnable** *(int deviceNr, bool trgEnable)*;set global trigger enable
- **Opcard_GetCounterXY_Value** *(int deviceNr, int* counterXY)*; get counter value
- **Opcard_SetCounterXY_TopValue** *(int deviceNr, int counterTopValue)*; set counter top value
- **Opcard_GetCounterEnable** *(int deviceNr, bool* counterCountingEnable)*; get counting enable
- **Opcard_SetCounterEnable** *(int deviceNr, bool*

counterCountingEnable); set counting enable

- **Opcard_GetTimerValues** *(int deviceNr, bool* timerEnable, double* timerPeriod)*; get timer period in kHz
- **Opcard_SetTimerValues** *(int deviceNr, bool timerEnable, double timerPeriod)*; set timer period in kHz

Serial Peripheral Interface (SPI):

Functions:

- **Opcard_Set_extSPI_CTRL** *(int deviceNr, int FrameL, bool F_MS, bool pol, bool line, bool Transition)*; set SPI controls: frame length, MISO / MOSI line, polarization of serial clock (SCK), line for communication (0 = P13, 1 = P15), transition (1 = start, 0 = no effect)
- **Opcard_Get_extSPI_status** *(int deviceNr, bool* busy, bool* finished, bool* errorOUT)*; get SPI control parameters: busy (0 = idle, 1 = busy), communication finished (0 = finished, 1 = not finished), error code (1 = error ocurred)
- **Opcard_Set_extSPI_speed** *(int deviceNr, double speed)*; set speed of SPI communication
- **Opcard_Get_extSPI_speed** *(int deviceNr, double* speed)*; get speed of SPI communication
- **Opcard_Set_extSPI_Data** *(int deviceNr, int FrameL, UINT16* data)*; store data received through SPI communication in *data* buffer
- **Opcard_Send_extSPI_ST** *(int deviceNr, int FrameL)*; send data in standard way
- **Opcard_Send_extSPI_Detal** *(int deviceNr, int FrameL, bool F_MS, bool pol, bool line)*; send all SPI parameters
- **Opcard_SetAMRIOAll** *(int deviceNr, int IOAll)*; set direction on all I/O lines on EXP_CON
- **Opcard_GetAMRIOAll** *(int deviceNr, int* IOAll)*; get direction of all I/O lines on EXP_CON
- **Opcard_SetAMRIOSAll** *(int deviceNr, int IOAll)*; set states on all I/O

lines on EXP_CON

- **Opcard_GetAMRIOSAll** *(int deviceNr, int* IOAll)*; get states of all I/O lines on EXP_CON
- **Opcard_ResetArmSettings** *(int deviceNr)*; reset all ARM settings (filters, inputs, attenuators, voltage) to default values

I/O:

Functions:

- **Opcard_GetGPI** *(int deviceNr, bool* gpi0, bool* gpi1, bool* gpi2, bool* gpi3)*; get state of GPI line
- **Opcard_GetGpoSettings** *(int deviceNr, bool* gpo0, bool* gpo1, bool* gpo2, bool* gpo3)*; get state of GPO line
- **Opcard_SetGpoSettings** *(int deviceNr, bool gpo0, bool gpo1, bool gpo2, bool gpo3)*; set state of GPO line
- **Opcard_GetOutputEnable** *(int deviceNr, bool* gpo0, bool* gpo1, bool* gpo2, bool* gpo3)*; get control mode of GPO line (0 = GPO controlled by register bits, 1 = GPO controlled internally)
- **Opcard_SetOutputEnable** *(int deviceNr, bool gpo0, bool gpo1, bool gpo2, bool gpo3)*; set control mode of GPO line
- **Opcard_GetFastGpo** *(int deviceNr, int lineNr, bool* gpo)*; get state of Fast GPO line (0 = low state, 1 = high state: LVTTL 3.3V)
- **Opcard_SetFastGpo** *(int deviceNr, int lineNr, bool gpo)*; set state of Fast GPO line (0 = low state, 1 = high state: LVTTL 3.3V)
- **Opcard_GetGpiCaptured** *(int deviceNr, bool* gpi0, bool* gpi1, bool* gpi2, bool* gpi3)*;

Functions for sequence control:

- **Opcard_SetSeqEnable** *(int deviceNr, int segEnable)*; set sequence mode (1 – sequence enable (multi mode), 0 – sequence disable (single mode));
- **Opcard_GetSeqEnable** *(int deviceNr, int* segEnable)*; get sequence mode (1 – sequence enable (multi mode), 0 – sequence disable (single mode));
- **Opcard_SetSeqIndex** *(int deviceNr, int segIndex)*; set current sequence index: 0...1023
- **Opcard_GetSeqIndex** *(int deviceNr, int* segIndex)*; get current

sequence index: 0...1023

- **Opcard_SetSeqLength** *(int deviceNr, int segLength)*; set number of sequences: 1...1024
- **Opcard_GetSeqLength** *(int deviceNr, int* segLength)*; get number of sequences: 1...1024

7. List of API functions:

1) **Opcard_Reset** (int **deviceNr**);
2) **Opcard_GetInfo** (int **deviceNr**, char* **About**);
3) **Opcard_ReadData** (int **deviceNr**, unsigned int ***data**, int **sizeData**);
4) **Opcard_ReadAllData** (int **deviceNr**, unsigned int ***data**,int ***sizeData**);
5) **Opcard_GetTimePulse** (int **deviceNr**, double* **timeP**);
6) **Opcard_SetTimePulse** (int **deviceNr**, double **timeP**);
7) **Opcard_GetDriverEnable** (int **deviceNr**, bool* **drvEnable**);
8) **Opcard_SetDriverEnable** (int **deviceNr**, bool **drvEnable**);
9) **Opcard_GetGPI** (int **deviceNr**, bool* **gpi0**, bool* **gpi1**, bool* **gpi2**, bool* **gpi3**);
10) **Opcard_GetGpoSettings** (int **deviceNr**, bool* **gpi0**, bool* **gpi1**, bool* **gpi2**, bool* **gpi3**, bool* **gpi4**, bool* **gpi5**);
11) **Opcard_SetGpoSettings** (int **deviceNr**, bool **gpi0**, bool **gpi1**, bool **gpi2**, bool **gpi3**, bool **gpi4**, bool **gpi5**);
12) **Opcard_GetOutputEnable (**int **deviceNr,** bool* **gpi0,** bool* **gpi1,** bool* **gpi2,** bool* **gpi3**, bool* **gpi4**, bool* **gpi5**);
13) **Opcard_SetOutputEnable** (int **deviceNr,** bool **gpi0,** bool **gpi1,** bool **gpi2,** bool **gpi3**, bool **gpi4**, bool **gpi5**);
14) **Opcard_GetGpiCaptured** (int **deviceNr,** bool* **gpi0,** bool* **gpi1,** bool* **gpi2,** bool* **gpi3**, bool* **gpi4**, bool* **gpi5**);
15) **Opcard_GetTriggerSource** (int **deviceNr**, int* **trgSource**);
16) **Opcard_SetTriggerSource** (int **deviceNr**, int **trgSource**);
17) **Opcard_GetTriggerEnable** (int **deviceNr**, bool* **trgEnable**);
18) **Opcard_SetTriggerEnable** (int **deviceNr**, bool **trgEnable**);
19) **Opcard_SetTriggerApl** (int **deviceNr**);
20) **Opcard_SetUploadAck** (int **deviceNr**);
21) **Opcard_GetTriggerStatus** (int **deviceNr**, bool* **status**);

22) **Opcard_SetNewDataReady** (int **deviceNr,** bool **newData**);

23) **Opcard_GetTriggerOverrun** (int **deviceNr**, bool* **trgOverrun**);

24) **Opcard_GetTrgOvrScr** (int **deviceNr**, int **flagNr**, bool* **trgOvrScr**);

25) **Opcard_GetTrgOverrunCounter** (int **deviceNr**, int* **trgOverCounter**);

26) **Opcard_GetCounterXY_Value** (int **deviceNr**, int* **counterXY**);

27) **Opcard_SetCounterXY_TopValue** (int **deviceNr**, int **counterTopValue**);

28) **Opcard_GetCounterEnable** (int **deviceNr**, bool* **counterCountingEnable**);

29) **Opcard_SetCounterEnable** (int **deviceNr**, bool **counterCountingEnable**);

30) **Opcard_GetCounterReset** (int **deviceNr**, bool* **counterReset**);

31) **Opcard_SetCounterReset** (int **deviceNr**, bool **counterReset**);

32) **Opcard_GetTimerValues** (int **deviceNr**, bool* **timerEnable**, double* **timerPeriod**);

33) **Opcard_SetTimerValues** (int **deviceNr**, bool **timerEnable**, double **timerPeriod**);

34) **Opcard_GetTimerCaptured** (int **deviceNr**, double* **timerCaptured**);

35) **Opcard_SetAnalogFilters** (int **deviceNr**, int **analogFilters**);

36) **Opcard_GetAnalogFilters** (int **deviceNr**, int* **analogFilters**);

37) **Opcard_SetAnalogInput** (int **deviceNr**, int **analogInput**);

38) **Opcard_GetAnalogInput** (int **deviceNr**, int* **analogInput**);

39) **Opcard_SetAttenHilo** (int **deviceNr**, int **attenHilo**);

40) **Opcard_GetAttenHilo** (int **deviceNr**, int* **attenHilo**);

41) **Opcard_SetAll** (int **deviceNr**, int **analogF**, int **analogI**, int **attenH**);

42) **Opcard_GetAll** (int **deviceNr**, int* **analogF**, int* **analogI**, int* **attenH**);

43) **Opcard_SetPulseVoltage** (int **deviceNr**, int **pulseVoltage**);

44) **Opcard_GetPulseVoltage** (int **deviceNr**, int **pulseVoltage**);

45) **Opcard_ResetArmSettings** (int **deviceNr**);

46) **Opcard_Set_extSPI_CTRL** (int **deviceNr**, int **FrameL**, bool **F_MS**, bool **pol**, bool **line**, bool **Transition**);

47) **Opcard_Get_extSPI_status** (int **deviceNr**, bool* **busy**, bool* **finished**, bool* **errorOUT**);

48) **Opcard_Set_extSPI_speed** (int **deviceNr**, double **speed**);

49) **Opcard_Set_extSPI_speed** (int **deviceNr**, double* **speed**);

50) **Opcard_Get_extSPI_Data** (int **deviceNr**, int **FrameL**, UINT16* **data**);

51) **Opcard_Send_extSPI_ST** (int **deviceNr**, int **FrameL**);

52) **Opcard_Send_extSPI_Detal** (int **deviceNr**, int **FrameL**, bool **F_MS**, bool **pol**, bool **line**);

53) **Opcard_SetAMRIOAll** (int **deviceNr**, int **IOAll**);

54)**Opcard_GetAMRIOAll** (int **deviceNr**, int* **IOAll**);

55) **Opcard_SetAMRIOSAll** (int **deviceNr**, int **IOAll**);

56)**Opcard_GetAMRIOSAll** (int **deviceNr**, int* **IOAll**);

57)**Opcard_GetSpiReset** (int **deviceNr**, bool* **spiReset**);

58)**Opcard_SetSpiReset** (int **deviceNr**, bool **spiReset**);

59)**Opcard_GetSpiBusy** (int **deviceNr**, bool **spiBusy**);

60)**Opcard_SetResetFifo** (int **deviceNr**);

61)**Opcard_SetResetIdx** (int **deviceNr**);

62)**Opcard_GetFifoFull** (int **deviceNr**, bool* **fifoFull**);

63)**Opcard_GetFrameIdx** (int **deviceNr**, int* **frameIdx**);

64)**Opcard_GetFifoCnt** (int **deviceNr**, UINT* **fifoCount**);

65)**Opcard_GetSamplingFreq** (int **deviceNr**, int* **samplingFreq**);

66)**Opcard_SetSamplingFreq** (int **deviceNr**, int **samplingFreq**);

67)**Opcard_GetGainMode** (int **deviceNr**, int* **gainMode**);

68)**Opcard_SetGainMode** (int **deviceNr**, int **gainMode**);

69)**Opcard_GetDataProcMode** (int **deviceNr**, bool* **dataProcMode**);

70)**Opcard_SetDataProcMode** (int **deviceNr**, bool **dataProcMode**);

71)**Opcard_GetStoreDisabled** (int **deviceNr**, bool* **storeDisabled**);

72)**Opcard_SetStoreDisabled** (int **deviceNr**, bool **storeDisabled**);

73)**Opcard_GetAckMode** (int **deviceNr**, bool* **ackMode**);

74)**Opcard_SetAckMode** (int **deviceNr**, bool **ackMode**);

75) **Opcard_GetDelay** (int **deviceNr**, int * **delayVal**);

76) **Opcard_SetDelay** (int **deviceNr**, int **delayVal**);

77) **Opcard_GetBufferDepth** (int **deviceNr**, int* **bufferDepth**);

78) **Opcard_SetBufferDepth** (int **deviceNr**, int **bufferDepth**);

79) **Opcard_GetConstGain** (int **deviceNr**, double* **constantGain**);

80) **Opcard_SetConstGain** (int **deviceNr**, double **constantGain**);

81) **Opcard_WriteData** (int **deviceNr**, unsigned int* **data,** int **sizeData**);

82) **Opcard_GetPdxMode** (int **deviceNr**, int **pdNr**, int* **pdMode**);

83) **Opcard_SetPdxMode** (int **deviceNr**, int **pdNr**, int **pdMode**);

84) **Opcard_GetPdxEnable** (int **deviceNr**, int **pdNr**, bool* **pdEnable**);

85) **Opcard_SetPdxEnable** (int **deviceNr**, int **pdNr**, bool **pdEnable**);

86) **Opcard_GetPdxResult** (int **deviceNr**, int **pdNr**, bool* **pdResult**);

87) **Opcard_GetPdxStart** (int **deviceNr**, int **pdNr**, int* **pdStart**);

88) **Opcard_SetPdxStart** (int **deviceNr**, int **pdNr**, int **pdStart**);

89) **Opcard_GetPdxStop** (int **deviceNr**, int **pdNr**, int* **pdStop**);

90) **Opcard_SetPdxStop** (int **deviceNr**, int **pdNr**, int **pdStop**);

91) **Opcard_GetPdxLevelPosition** (int **deviceNr**, int **pdNr**, int* **pdLevelPos**);

92) **Opcard_GetPdxLevelValue** (int **deviceNr**, int **pdNr**, char* **pdLevelVal**);

93) **Opcard_SetPdxLevelValue** (int **deviceNr**, int **pdNr**, char **pdLevelVal**);

94) **Opcard_SetPdxLSSP** (int **deviceNr**, int **pdNr**, int **start**, int **stop**, int **level**);

95) **Opcard_GetPdxMaxPosition** (int **deviceNr**, int **pdNr**, int* **pdMaxPos**);

96) **Opcard_GetPdxMaxValue** (int **deviceNr**, int **pdNr**, char* **pdMaxVal**);

97) **Opcard_GetEncxEnable** (int **deviceN**r, int **encNr**, bool* **enable**);

98) **Opcard_SetEncxEnable** (int **deviceNr**, int **encNr,** bool **enable**);

99) **Opcard_SetEncxReset** (int **deviceNr**, int **encNr**, bool **reset**);

100) **Opcard_GetEncxDirectionInvert** (int **deviceNr**, int **encNr**, bool* **dirInvert**);

101) **Opcard_SetEncxDirectionInvert** (int **deviceNr**, int **encNr**, bool **dirInvert**);

102) **Opcard_GetEncxIndexEnable** (int **deviceNr**, int **encNr**, bool* **idxEnable**);

103) **Opcard_SetEncxIndexEnable** (int **deviceNr**, int **encNr**, bool **idxEnable**);

104) **Opcard_GetEncxDecodeMode** (int **deviceNr**, int **encNr**, int* **decMode**);

105) **Opcard_SetEncxDecodeMode** (int **deviceNr**, int **encNr**, int **decMode**);

106) **Opcard_GetEncxFilterEnable** (int **deviceNr**, int **encNr**, bool* **filterEnable**);

107) **Opcard_SetEncxFilterEnable** (int **deviceNr**, int **encNr**, bool **filterEnable**);

108) **Opcard_GetEncxCompareEnable** (int **deviceNr**, int **encNr**, bool* **compEnable**);

109) **Opcard_SetEncxCompareEnable** (int **deviceNr**, int **encNr**, bool **compEnable**);

110) **Opcard_GetEncxCompareStep** (int **deviceNr**, int **encNr,** int* **compStep**);

111) **Opcard_SetEncxCompareStep** (int **deviceNr**, int **encNr**, int **compStep**);

112) **Opcard_GetEncxPosition** (int **deviceNr**, int **encNr**, int* **position**);

113) **Opcard_GetEncxCapture** (int **deviceNr**, int **encNr**, int* **capture**);

114) **Opcard_GetEncxFilter** (int **deviceNr**, int **encNr**, int* **filterLength**);

115) **Opcard_SetEncxFilter** (int **deviceNr**, int **encNr**, int **filterLength**);

116) **Opcard_SetSeqEnable** (int **deviceNr**, int **segEnable**);

117) **Opcard_GetSeqEnable** (int **deviceNr**, int* **segEnable**);

118) **Opcard_SetSeqIndex** (int **deviceNr**, int **segIndex**);

119) **Opcard_GetSeqIndex** (int **deviceNr**, int* **segIndex**);

120) **Opcard_SetSeqLength** (int **deviceNr**, int **segLenth**);

121) **Opcard_GetSeqLength** (int **deviceNr**, int* **segLenth**);

8. API Functions Detailed Description

1. Opcard_Reset

Prototype:

DLLReturnType

Opcard_Reset (

int **deviceNr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function makes a reset of used device (card). All settings are reset to default values. It is recommended to call this function at the start of session with Opcard.

2. Opcard_GetInfo

Prototype:

DLLReturnType

Opcard_GetInfo (

int **deviceNr ,**

char* **info**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

info is a pointer to array, where function returns information about Opcard. Array size should not exceed range 18...32.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function provides information about detected card. Information returned in ***Inf***

„SN YY.SN rev. RRR", where:

YY – year of production,

SN – serial number,

RRR – firmware revision.

3. Opcard_ReadData

Prototype:

DLLReturnType

Opcard_ReadData (

int **deviceNr ,**

unsigned int ***data,**

int **sizeData**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

data is a pointer to prepared data buffer for samples

sizeData specifies number of data samples to read **(must be a multiple of 4!)**

Returns:

Returns 0 (zero) when success. Returns -1 when user sends incorrect parameters.

Description:

This function reads data from on-board acquisition data memory. Data is not converted. The **sizeData** parameter is in range of **4 to 262088 (256k)** samples.

4. Opcard_ReadAllData

Prototype:

DLLReturnType

Opcard_ReadDataAll (

int **deviceNr ,**

unsigned int ***data,**

int ***sizeData**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

data is a pointer to prepared data buffer for samples

sizeData returns the data count read from Opcard's buffer

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function reads all available data from on-board acquisition data memory. Data is not converted.

5. Opcard_GetTimePulse

Prototype:

DLLReturnType

Opcard_GetTimePulse (

int **deviceNr ,**

double* **timeP**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

timeP is a pointer to Pulse Time value in [us].

Returns:

Returns 0 (zero) when success. Otherwise returns an error code.

Description:

This function returns current value of a Pulse Time parameter.

6. Opcard_SetTimePulse

Prototype:

DLLReturnType

Opcard_SetTimePulse (

int **deviceNr ,**

double **timeP**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

timeP is a value of Pulse Time ranged from 0 to 6.3 [us]

Returns:

Returns 0 (zero) when success. Otherwise returns an error code.

Description:

This function determines Pulse Time of on-board Pulser.

7. Opcard_GetDriverEnable

Prototype:

```
DLLReturnType
Opcard_GetDriverEnable (
                        int deviceNr ,
                        bool *drvEnable
                       );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

drvEnable returns information about pulse driver: 1 = disabled, 0 = enabled

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns information about enabled (**drvEnable** = 0) or disabled (**drvEnable** = 1) pulser driver.

8. Opcard_SetDriverEnable

Prototype:

```
DLLReturnType
Opcard_SetDriverEnable (
                        int deviceNr ,
                        bool drvEnable
                       );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

drvEnable disable (1) / enable (0) pulse driver

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function disables (**drvEnable** = 1) / enables (**drvEnable** = 0) pulse driver.

9. Opcard_GetGPI

Prototype:

DLLReturnType

Opcard_GetGPI (

int **deviceNr ,**

bool * **gpi0,**

bool * **gpi1,**

bool * **gpi2,**

bool * **gpi3,**

bool * **gpi4,**

bool * **gpi5,**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpi0,1,2,3,4,5 return states on PINs respectively 11, 4, 12, 5, 15, 8 for DB15 connector.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function reads input states for DB15 connector through PIN (11, 4, 12, 5, 15, 8); returns 0 when corresponding TTL input is at low level and 1 when corresponding TTL input is at high level.

10. Opcard_GetGpoSettings

Prototype:

DLLReturnType

Opcard_GetGpoSettings (

int **deviceNr ,**
bool * **gpo0,**
bool * **gpo1,**
bool * **gpo2,**
bool * **gpo3,**
bool * **gpo4,**
bool * **gpo5,**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpo0,1,2,3 returns states on PINs, respectively: 1, 9, 2, 10, 13, 6 for DB15 connector.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function reads outputs states from DB15 connector and returns PINs (1, 9, 2, 10, 13, 6) states. Returned value = 0 corresponds with output TTL low level, value = 1 corresponds with output TTL high level.

11. Opcard_SetGpoSettings

Prototype:

DLLReturnType

Opcard_SetGpoSettings (
int **deviceNr ,**
bool **gpo0,**
bool **gpo1,**
bool **gpo2,**
bool **gpo3,**
bool **gpo4,**
bool **gpo5,**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpo0,1,2,3,4,5 sets PINs states, respectively: 1, 9, 2, 10, 13, 6

for DB15 connector.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets outputs states from DB15 connector and sets PINs (1, 9, 2, 10, 13, 6) in a certain state. Set value = 0 corresponds with output TTL low level, value = 1 corresponds with output TTL high level.

12. Opcard_GetOutputEnable

Prototype:

DLLReturnType

Opcard_GetOutputEnable (

int **deviceNr ,**

bool * **gpo0,**

bool * **gpo1,**

bool * **gpo2,**

bool * **gpo3,**

bool * **gpo4,**

bool * **gpo5,**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpo0,1,2,3,4,5 return states of PIN respectively: 1, 9, 2, 10, 13, 6 for DB15 connector:

0 = GPO output is controlled using corresponding bits [7:4] [13] [14] of register,

1 = GPO is controlled internally.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function checks control mode of GPO PINs: 0 = GPO output is controlled using corresponding bits of register, 1 = GPO is controlled internally.

13. Opcard_SetOutputEnable

Prototype:

DLLReturnType

Opcard_SetOutputEnable (

int **deviceNr ,**

bool **gpo0,**

bool **gpo1,**

bool **gpo2,**

bool **gpo3,**

bool **gpo4,**

bool **gpo5,**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpo0,1,2,3,4,5 return states of PIN respectively: 1, 9, 2, 10, 13, 6 for DB15 connector:

0 = GPO output is controlled using corresponding bits [7:4] [13] [14] of register,

1 = GPO is controlled internally.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets control mode of GPO PINs: 0 = GPO output is controlled using corresponding bits [7:4] [13] [14] of register, 1 = GPO is controlled internally.

14. Opcard_GetGpiCaptured

Prototype:

DLLReturnType

Opcard_GetGpiCaptured (

int **deviceNr ,**

bool * **gpi0,**

bool * **gpi1,**

bool * **gpi2,**

bool * **gpi3,**

bool * **gpi4,**
bool * **gpi5,**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gpi0,1,2,3,4,5 return states on PINs respectively 11, 4, 12, 5, 15, 8 for DB15 connector captured during last acquisition: 0 = TTL low level, 1 = TTL high level.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns values of GPI pins for DB15 connector captured during last acquisition. Returned 0 value corresponds with input TTL low level and returned 1 value corresponds with input TTL high level.

15. Opcard_GetTriggerSource

Prototype:

DLLReturnType

Opcard_GetTriggerSource (

int **deviceNr ,**

int * **trgSource**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgSource returns trigger source currently used in acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns trigger source for acquisition. Trigger sources are listed below.

TriggerSource	Name	Description
0	TriggerSW	Trigger from application
1	Ext X	Trigger from external source (DB15 pin 11)
2	Ext Y	Trigger from external source (DB15 pin 4)
3	PRF Timer	Trigger from internal PRF Timer module
4	Enc 1	Trigger from Encoder 1 module

5	Enc 2	Trigger from Encoder 2 module

16. Opcard_SetTriggerSource

Prototype:

DLLReturnType

Opcard_SetTriggerSource (

int **deviceNr ,**

int **trgSource**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgSource sets trigger source used in acquisition.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets trigger source for acquisition. Trigger sources are listed below.

trgSource	Name	Description
0	TriggerSW	Trigger from application
1	Ext X	Trigger from external source (DB15 pin 11)
2	Ext Y	Trigger from external source (DB15 pin 4)
3	PRF Timer	Trigger from internal PRF Timer module
4	Enc 1	Trigger from Encoder 1 module
5	Enc 2	Trigger from Encoder 2 module

17. Opcard_GetTriggerEnable

Prototype:

DLLReturnType

Opcard_GetTriggerEnable (

int **deviceNr ,**

int * **trgEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgEnable returns information about trigger: disabled (0) / enabled (1)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that global trigger is disabled (**trgEnable** = 0) or enabled (**trgEnable** = 1)

18. Opcard_SetTriggerEnable

Prototype:

```
DLLReturnType
Opcard_SetTriggerEnable  (
                              int deviceNr ,
                              int trgEnable
                            );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgEnable disables (0) or enables (1) trigger

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function disables (**trgEnable** = 0) or enables (**trgEnable** = 1) global trigger

19. Opcard_SetTriggerApl

Prototype:

```
DLLReturnType
Opcard_SetTriggerApl  (
                              int deviceNr
                          );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function triggers the measurement (software trigger).

20. Opcard_SetUploadAck

Prototype:

DLLReturnType

Opcard_SetUploadAck (

int **deviceNr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function confirms receipt of data in ACK mode.

21. Opcard_GetTriggerStatus

Prototype:

DLLReturnType

Opcard_GetTriggerStatus (

int **deviceNr,**

bool * **status**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

status returns status of trigger: 1 = acquisition continues, 0 = waiting for trigger

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the status of trigger. If acquisition goes on status = 1, if status = 0 device waits for a trigger.

22. Opcard_GetNewDataReady

Prototype:

DLLReturnType

Opcard_GetNewDataReady (

int **deviceNr,**

int * **newData**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

newData returns 1 if acquisition is finished and new data is ready to download, otherwise returns 0.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks if new data is available (**newData** = 1).

23. Opcard_GetTriggerOverrun

Prototype:

DLLReturnType

Opcard_GetTriggerOverrun (

int **deviceNr,**

bool * **trgOverrun**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgOverrun returns 1 if at least one trigger was lost since last acquisition, 0 if not

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks if any triggers were lost since last acquisition (in this case **trgOverrun** = 1 regardless of number of lost triggers).

24. Opcard_GetTrgOvrScr

Prototype:

DLLReturnType

Opcard_GetTrgOvrScr (

int **deviceNr,**

int **flagNr,**

bool* **trgOvrScr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

flagNr is a flag number to check.

trgOvrScr returns 1 if any trigger were lost due to reason selected to check using *flagNr*, 0 if not

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks if at least one trigger was lost (**trgOvrScr** = 1 if yes) due to selected reasons. Flag of available reasons for losing trigger are listed below.

flagNr	Name	Description (if **trgOvrScr** = 1)
0	A	Triggers were lost during acquisition
1	H	Triggers were lost during HoldOff
2	F	Triggers were lost due to full FIFO

25. Opcard_GetTrgOverrunCounter

Prototype:

DLLReturnType

Opcard_GetTrgOverrunCounter (

int **deviceNr,**

int * **trgOverCounter**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

trgOverCounter returns number of lost triggers since last correct acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks how many triggers were lost since last correct acquisition (maximum 65535). If **trgOverCounter** = 65535 it might mean that number of lost triggers is higher.

26. Opcard_GetCounterXY_Value

Prototype:

DLLReturnType

Opcard_GetCounterXY_Value (

int **deviceNr,**

int * **counterXY**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterXY returns current value of counterXY

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current value of **counterXY**. Returned value (**counterXY**) is between 0 and TopValue set by function **Opcard_SetCounterXY_TopValue**.

27. Opcard_SetCounterXY_TopValue

Prototype:

DLLReturnType

Opcard_SetCounterXY_TopValue (

int **deviceNr,**

int **counterTopValue**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterTopValue sets top value of CounterXY

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the top value of **CounterXY**. The function changes the range of counter. **counterTopValue** has to be between 0 and 65535. If it is different, it is coerced to appropriate value inside specified range.

28. Opcard_GetCounterEnable

Prototype:

```
DLLReturnType
Opcard_GetCounterEnable (
                                    int deviceNr,
                                    bool * counterCountingEnable
                                );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterCountingEnable returns information about external counter: enabled (1) or disabled (0)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the state of external counter. If **counterCountingEnable** = 0 the counter is disabled, if **counterCountingEnable** = 1 that it is enabled.

29. Opcard_SetCounterEnable

Prototype:

```
DLLReturnType
Opcard_SetCounterEnable (
                                    int deviceNr,
                                    bool counterCountingEnable
                                );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterCountingEnable enables (1) or disables (0) external counter.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function enables (**counterCountingEnable** = 1) or disables (**counterCountingEnable** = 0) external counter.

30. Opcard_GetCounterReset

Prototype:

DLLReturnType

Opcard_GetCounterReset(

int **deviceNr,**

bool * **counterReset**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterReset returns the state of reset mode of external counter.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the state of reset mode of external counter. If (**counterReset** = 0) the counter is switched off and value of the counter is 0, if (**counterReset** = 1) the counter is switched on (it can count).

31. Opcard_SetCounterReset

Prototype:

DLLReturnType

Opcard_GetCounterReset(

int **deviceNr,**

bool **counterReset**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

counterReset sets the state of reset mode of external counter.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the state of reset mode of external counter. If (**counterReset** = 0) the function does reset and counting is disabled, if (**counterReset** = 1) counting is enabled.

32. Opcard_GetTimerValues

Prototype:

DLLReturnType

Opcard_GetTimerValues(

int **deviceNr**,

bool * **timerEnable**,

double * **timerPeriod**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

timerEnable returns the information that internal timer is enabled/disabled

timerPeriod returns timer frequency in kHz.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks if internal timer is enabled/disabled (**timerEnable** = 1/0) and the timer frequency value. Range of timer frequency: 0.06 Hz to 100 kHz.

33. Opcard_SetTimerValues

Prototype:

DLLReturnType

Opcard_SetTimerValues(

int **deviceNr**,

bool **timerEnable**,

double **timerPeriod**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

timerEnable enables (0) / disables (1) the internal timer

timerPeriod sets the timer frequency in kHz.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function enables/disables the internal timer (**timerEnable** = 0/1) and sets the timer frequency. Available range of timer frequency: 0.06 Hz to 100 kHz. If value is out of range, it is coerced to appropriate value.

34. Opcard_GetTimerCaptured

Prototype:

DLLReturnType

Opcard_GetTimerCaptured(

int **deviceNr**,

int * **timerCaptured**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

timerCaptured returns counter value of internal timer captured during last acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the counter value of internal timer captured during last acquisition. Range of **timerCaptured** is between 0 and internal timer time period in us.

35. Opcard_SetAnalogFilters

Prototype:

DLLReturnType

Opcard_SetAnalogFilters (

int **deviceNr**,

int **analogFilters**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogFilters is a parameter for setting on-board analog filters

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets the analog filters like specified below:

analogFilters	Settings
0	0.5 – 6 MHz,
1	1 – 6 MHz,
2	2 – 6 MHz,
3	4 – 6 MHz,
4	0.5 – 10 MHz,
5	1 – 10 MHz,
6	2 – 10 MHz,
7	4 – 10 MHz,
8	0.5 – 15 MHz,
9	1 – 15 MHz,
10	2 – 15 MHz,
11	4 – 15 MHz,
12	0.5 – 25 MHz,
13	1 – 25 MHz,
14	2 – 25 MHz,
15	4 – 25 MHz.

36. Opcard_GetAnalogFilters

Prototype:

DLLReturnType

Opcard_GetAnalogFilters (

int **deviceNr**,

int* **analogFilters**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogFilters returns parameter for setting on-board analog filters

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns analog filters settings like specified below:

analogFilters	Settings
0	0.5 – 6 MHz,
1	1 – 6 MHz,
2	2 – 6 MHz,
3	4 – 6 MHz,
4	0.5 – 10 MHz,
5	1 – 10 MHz,
6	2 – 10 MHz,
7	4 – 10 MHz,
8	0.5 – 15 MHz,
9	1 – 15 MHz,
10	2 – 15 MHz,
11	4 – 15 MHz,
12	0.5 – 25 MHz,
13	1 – 25 MHz,
14	2 – 25 MHz,
15	4 – 25 MHz.

37. Opcard_SetAnalogInput

Prototype:

DLLReturnType

Opcard_SetAnalogInput (

int **deviceNr**,

int **analogInput**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogInput is a parameter for setting the analog source: PE (3) or TT (1)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets the analog source: PE (**analogInput** = 3) or TT (**analogInput** = 1).

38. Opcard_GetAnalogInput

Prototype:

DLLReturnType

Opcard_GetAnalogInput (

int **deviceNr**,

int* **analogInput**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogInput returns analog source: PE (3) or TT (1)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function checks and returns analog source set in Opcard: PE (**analogInput** = 3) or TT (**analogInput** = 1).

39. Opcard_SetAttenHilo

Prototype:

DLLReturnType

Opcard_SetAttenHilo (

int **deviceNr**,

int **attenHilo**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

attenHilo sets on-board analog input: Attenuator and PostAmplifier

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function set the input Attenuator (-20dB) and/or PostAmplifier (+24dB):

attenHilo = 0 – 0dB – Attenuator and PostAmplifier are turned off

attenHilo = 1 – +24dB – PostAmplifier turned on

attenHilo = 2 – - 20dB – Attenuator turned on

attenHilo = 3 - +4dB - Attenuator and PostAmplifier are turned on (not recommended)

40. Opcard_GetAttenHilo

Prototype:

DLLReturnType

Opcard_GetAttenHilo (

int **deviceNr**,

int* **attenHilo**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

attenHilo is setting for the on-board analog input: Attenuator and PostAmplifier

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns input Attenuator (-20dB) and PostAmplifier (+24dB) settings:

attenHilo = 0 – 0dB – Attenuator and PostAmplifier are turned off

attenHilo = 1 – +24dB – PostAmplifier turned on

attenHilo = 2 – - 20dB – Attenuator turned on

attenHilo = 3 - +4dB - Attenuator and PostAmplifier are turned on (not recommended)

41. Opcard_SetAll

Prototype:

DLLReturnType

Opcard_SetAll (

int **deviceNr**,

int **analogFilters,**

int **analogInput ,**

int **attenHilo**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogInput is a parameter for setting the analog source PE or TT

attenHilo is setting for the on-board analog input Attenuator and PostAmplifier

analogFilters is a parameter for setting on-board analog filters

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets all analog parameters, which could be set using functions: **Opcard_SetAnalogInput**, **Opcard_SetAttenHilo, Opcard_SetAnalogFilters** one after another.

42. Opcard_GetAll

Prototype:

DLLReturnType

Opcard_GetAll (

int **deviceNr,**
int* **analogFilters,**
int* **analogInput,**
int* **attenHilo**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

analogInput returns parameter for setting the analog source PE or TT

attenHilo returns setting for the on-board analog input Attenuator and PostAmplifier

analogFilters returns parameter for setting on-board analog filters

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns all analog parameters, which could be check separately using following functions: **Opcard_GetAnalogInput**, **Opcard_GetAttenHilo, Opcard_GetAnalogFilters** one after another.

43. Opcard_SetPulseVoltage

Prototype:

DLLReturnType

Opcard_SetPulseVoltage (

int **deviceNr**,

int **pulseVoltage**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pulseVoltage is Pulser Voltage value in range from 0 (pulser off) to 255 (max amplitude). Range of pulse voltage corresponds to range 0 up to 10 V (0…360 V without load).

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function sets the pulser amplitude.

44. Opcard_GetPulseVoltage

Prototype:

DLLReturnType

Opcard_SetPulseVoltage (

int **deviceNr**,

int* **pulseVoltage**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pulseVoltage returns pulser voltage value in range from 0 (pulser off) up to 255 (max amplitude). Range of pulse voltage corresponds to range from 0 up to 10 V (0...360 V without load).

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function returns pulser amplitude.

45. Opcard_ResetArmSettings

Prototype:

DLLReturnType

Opcard_ResetArmSettings (

int **deviceNr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

This function does a reset of parameters: analog filter, source input (PE/TT), setting of the on-board Attenuator and PostAmplifier and pulse amplitude to their default settings. The following parameters are modifed:

- Analog filter - 0.5 – 6 MHz
- Source - PE,

- Attenuator/PostAmplifier - both off,
- Pulse Amplitude - 0 V.

46. Opcard_Set_extSPI_CTRL

Prototype:

DLLReturnType

Opcard_Set_extSPI_CTRL (

int **deviceNr**,

int **FrameL**,

bool **F_MS**,

bool **pol**,

bool **line**,

bool **Transition**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

FrameL – frame length: 8...16 bits

F_MS – SPI CPHA parameter: 0 = sampling on leading edge, 1 = sampling on trailing edge

pol – SPI CPOL parameter: 0 = idle state low, first edge is rising, 1 = idle state high, first edge is falling

line – slave select line: 0 = EXP_P13, 1 = EXP_P15

Transition – frame transmit triggering: 0 = no effect, 1 = start

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the following SPI parameters: frame length, CPHA, CPOL and slave select line. When Transition = 1 function initializes SPI transmission.

47. Opcard_Get_extSPI_status

Prototype:

DLLReturnType

Opcard_Get_extSPI_status (

int **deviceNr**,

bool* **busy**,

bool* **finished**,

bool* **errorOUT**,

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

busy – status of SPI interface: 0 = idle, 1 = busy

finished – status of SPI communication: 0 = finished, 1 = not finished

errorCODE – error code from SPI interface: 0 = no error, 1 = error occurred

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns current SPI communication status.

48. Opcard_Set_extSPI_speed

Prototype:

DLLReturnType

Opcard_Set_extSPI_speed (

int **deviceNr**,

double **speed**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

speed – speed of SPI transmission in [MHz], ranged from 0.236 MHz up to 7.5 MHz

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets speed of SPI transmission.

49. Opcard_Get_extSPI_speed

Prototype:

DLLReturnType

Opcard_Get_extSPI_speed (

int **deviceNr**,

double* **speed**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

speed – returns speed of SPI transmission in [MHz], which may range from 0.236 MHz up to 7.5 MHz

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns speed of SPI transmission.

50. Opcard_Set_extSPI_Data

Prototype:

DLLReturnType

Opcard_Set_extSPI_Data (

int **deviceNr**,

int **FrameL,**

UINT16* data

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

FrameL – frame length of collected data: 8...16 bits

data – data received through SPI

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function stores data received through SPI communication in *data* buffer.

51. Opcard_Send_extSPI_ST

Prototype:

DLLReturnType

Opcard_Send_extSPI_ST (

int **deviceNr**,

int **FrameL**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

FrameL – frame length of collected data: 8...16 bits

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Standard procedure for sending data via SPI.

52. Opcard_Send_extSPI_Detal

Prototype:

DLLReturnType

Opcard_Send_extSPI_Detal (

int **deviceNr**,

int **FrameL,**

bool **F_MS**,

bool **pol**,

bool **line**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

FrameL – frame length: 8...16 bits

F_MS – SPI CPHA parameter: 0 = sampling on leading edge, 1 = sampling on trailing edge

pol – SPI CPOL parameter: 0 = idle state low, first edge is rising, 1 = idle state high, first edge is falling

line – slave select line: 0 = EXP_P13, 1 = EXP_P15

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Advanced procedure for sending data via SPI with additional settings.

53. Opcard_SetAMRIOAll

Prototype:

DLLReturnType

Opcard_SetAMRIOAll (

int **deviceNr**,

int **IOAll**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

IOAll – communication outputs for all lines: 0…63

bit 0 - EXP_P8 output: 0 = IN, 1 = OUT

bit 1 - EXP_P10 output: 0 = IN, 1 = OUT

bit 2 - EXP_P12 output: 0 = IN, 1 = OUT

bit 3 - EXP_P14 output: 0 = IN, 1 = OUT

bit 4 - EXP_P16 output: 0 = IN, 1 = OUT

bit 5 - EXP_P18 output: 0 = IN, 1 = OUT

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets output for I/O lines on EXP_CON.

54. Opcard_GetAMRIOAll

Prototype:

DLLReturnType

Opcard_GetAMRIOAll (

int **deviceNr**,

int* **IOAll**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

IOAll – communication outputs for all lines: 0…63

bit 0 - EXP_P8 output: 0 = IN, 1 = OUT

bit 1 - EXP_P10 output: 0 = IN, 1 = OUT

bit 2 - EXP_P12 output: 0 = IN, 1 = OUT

bit 3 - EXP_P14 output: 0 = IN, 1 = OUT

bit 4 - EXP_P16 output: 0 = IN, 1 = OUT

bit 5 - EXP_P18 output: 0 = IN, 1 = OUT

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns output from all I/O lines on EXP_CON

55. Opcard_SetAMRIOSAll

Prototype:

DLLReturnType

Opcard_SetAMRIOAll (

int **deviceNr**,

int **IOAll**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

IOAll – state for all I/O lines: 0…63

bit 0 - EXP_P8 direction: 0 = LOW, 1 = HIGH

bit 1 - EXP_P10 direction: 0 = LOW, 1 = HIGH

bit 2 - EXP_P12 direction: 0 = LOW, 1 = HIGH

bit 3 - EXP_P14 direction: 0 = LOW, 1 = HIGH

bit 4 - EXP_P16 direction: 0 = LOW, 1 = HIGH

bit 5 - EXP_P18 direction: 0 = LOW, 1 = HIGH

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets state for all I/O lines on EXP_CON

56. Opcard_GetAMRIOSAll

Prototype:

DLLReturnType

Opcard_GetAMRIOAll (

int **deviceNr**,

int* **IOAll**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

IOAll – state for all I/O lines: 0…63

bit 0 - EXP_P8 direction: 0 = LOW, 1 = HIGH

bit 1 - EXP_P10 direction: 0 = LOW, 1 = HIGH

bit 2 - EXP_P12 direction: 0 = LOW, 1 = HIGH

bit 3 - EXP_P14 direction: 0 = LOW, 1 = HIGH

bit 4 - EXP_P16 direction: 0 = LOW, 1 = HIGH

bit 5 - EXP_P18 direction: 0 = LOW, 1 = HIGH

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns state for all I/O lines on EXP_CON

57. Opcard_GetSpiReset

Prototype:

DLLReturnType

Opcard_GetSpiReset (

int **deviceNr**,

bool * **spiReset**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

spiReset returns the state of SPI engine

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns whether SPI is in reset state (**spiReset** = 0) or not (**spiReset** = 1)

58. Opcard_SetSpiReset

Prototype:

DLLReturnType

Opcard_SetSpiReset (

int **deviceNr**,

bool **spiReset**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

spiReset controls the SPI engine state

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function does a reset of SPI (**spiReset** = 0) or unlocks SPI (**spiReset** = 1).

59. Opcard_GetSpiBusy

Prototype:

DLLReturnType

Opcard_GetSpiBusy (

int **deviceNr**,

bool * **spiBusy**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

spiBusy returns information that SPI is busy

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that SPI is busy (**spiBusy** = 1) – it means that transmission of data goes on.

60. Opcard_SetResetFifo

Prototype:

DLLReturnType

Opcard_SetResetFifo (

int **deviceNr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function resets FIFO controller and discard acquisition data stored in FIFO memory.

61. Opcard_SetResetIdx

Prototype:

DLLReturnType

Opcard_SetResetIdx (

int **deviceNr**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function resets frame index counter.

62. Opcard_GetFifoFull

Prototype:

DLLReturnType

Opcard_GetFifoFull (

int **deviceNr**,

bool * **fifoFull**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

fifoFull returns information about FIFO memory: 1 = full memory, 0 = FIFO has free space available

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks if FIFO memory is full (**fifoFull** = 1) or has free space available (**fifoFull** = 0).

63. Opcard_GetFrameIdx

Prototype:

DLLReturnType

Opcard_GetFrameIdx (

int **deviceNr**,

int * **frameIdx**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

frameIdx returns frame index

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns frame index which is incremented after each trigger. If **frameIdx** reaches maximum value (65535), it is reset (set to 0).

64. Opcard_GetFifoCnt

Prototype:

DLLReturnType

Opcard_GetFifoCnt (

int **deviceNr**,

UINT * **fifoCount**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

fifoCount returns number of data bytes available to read from FIFO buffer

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks how many bytes are available to read from FIFO memory.

65. Opcard_GetSamplingFreq

Prototype:

DLLReturnType

Opcard_GetSamplingFreq (

int **deviceNr**,

int * **samplingFreq**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

samplingFreq returns information about sampling frequency

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the sampling frequency.

SamplingFreq	Frequency
0	100 MHz
1	100 MHz
2	50 MHz
3	33.3 MHz
4	25 MHz
5	20 MHz
6	16.7 MHz
7	14.3 MHz
8	12.5 MHz
9	11.1 MHz
10	10 MHz
11	9.1 MHz
12	8.3 MHz
13	7.7 MHz
14	7.1 MHz
15	6.7 MHz

66. Opcard_SetSamplingFreq

Prototype:

DLLReturnType

Opcard_SetSamplingFreq (

int **deviceNr**,

int **samplingFreq**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

samplingFreq sets sampling frequency

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the sampling frequency.

SamplingFreq	Frequency
0	100 MHz
1	100 MHz
2	50 MHz
3	33.3 MHz
4	25 MHz
5	20 MHz
6	16.7 MHz
7	14.3 MHz
8	12.5 MHz
9	11.1 MHz
10	10 MHz
11	9.1 MHz
12	8.3 MHz
13	7.7 MHz
14	7.1 MHz
15	6.7 MHz

67. Opcard_GetGainMode

Prototype:

```
DLLReturnType
Opcard_GetGainMode  (
                        int deviceNr,
                        int * gainMode
                    );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gainMode returns gain mode for current acquisition.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the gain mode.

GainMode = 0 **-** constant gain during acquisition

GainMode = 1 **-** time gain compensation (TGC) mode of gain during acquisition

68. Opcard_SetGainMode

Prototype:

DLLReturnType

Opcard_SetGainMode (

int **deviceNr**,

int **gainMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

gainMode setting for the gain mode.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the gain mode.

gainMode = 0 **-** constant gain during acquisition

gainMode = 1 **-** time gain compensation (TGC) mode of gain during acquisition

69. Opcard_GetDataProcMode

Prototype:

DLLReturnType

Opcard_GetDataProcMode (

int **deviceNr**,

bool * **dataProcMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

dataProcMode returns hardware signal processing mode for current acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns a hardware signal processing mode.

dataProcMode = 0 – no processing (raw data)

dataProcMode = 1 – hardware absolute processing

70. Opcard_SetDataProcMode

Prototype:

DLLReturnType

Opcard_SetDataProcMode (

int **deviceNr**,

bool **dataProcMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

dataProcMode defines a hardware signal processing mode

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets a hardware signal processing mode.

dataProcMode = 0 – no processing (raw data)

dataProcMode = 1 – hardware absolute processing

71. Opcard_GetStoreDisabled

Prototype:

DLLReturnType

Opcard_GetStoreDisabled (

int **deviceNr**,
bool * **storeDisabled**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

storeDisabled returns the information about work mode: full data (0) or only headers (1) are stored to memory

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks which option of storing data are chosen.

storeDisabled = 0 – all data is stored to memory

storeDisabled = 1 – only headers is stored to memory

72. Opcard_SetStoreDisabled

Prototype:

DLLReturnType

Opcard_SetStoreDisabled (
int **deviceNr**,
bool * **storeDisabled**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

storeDisabled sets one of two options of storing the data: full data (0) or only headers (1) are stored to memory

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets option of storing data.

storeDisabled = 0 – all data are stored to memory

storeDisabled = 1 – only headers are stored to memory

73. Opcard_GetAckMode

Prototype:

DLLReturnType

Opcard_GetAckMode (

int **deviceNr**,

bool * **ackMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

ackMode returns current setting of data transmission mode: FIFO (1) or ACK (0)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns current setting of kind of transmission data.

ackMode = 0 – ACK mode, required confirmation after every acquisition

ackMode = 1 – FIFO mode, confirmation after every acquisition is not needed, next acquisitions can be triggered without confirmation of receipt

74. Opcard_SetAckMode

Prototype:

DLLReturnType

Opcard_SetAckMode (

int **deviceNr**,

bool **ackMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

ackMode sets current setting of data transmission mode: FIFO (1) or ACK (0)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets current setting of data transmission mode:

ackMode = 0 – single mode, required confirmation after every acquisition,

ackMode = 1 – FIFO mode, no - required confirmation after every acquisition, next acquisitions can be triggered without confirmation of receipt.

75. Opcard_GetDelay

Prototype:

DLLReturnType

Opcard_GetDelay (

int **deviceNr**,

int * **delayVal**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

delayVal returns delay parameter

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current delay. Returned value is in Ts units.

76. Opcard_SetDelay

Prototype:

DLLReturnType

Opcard_SetDelay (

int **deviceNr**,

int **delayVal**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

delayVal is a delay parameter in Ts unit

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets delay.

77. Opcard_GetBufferDepth

Prototype:

DLLReturnType

Opcard_GetBufferDepth (

int **deviceNr**,

int * **bufferDepth**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

bufferDepth returns depth parameter

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current depth. Returned value is in Ts unit.

78. Opcard_SetBufferDepth

Prototype:

DLLReturnType

Opcard_SetBufferDepth (

int **deviceNr**,

int **bufferDepth**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

bufferDepth is depth parameter in Ts unit

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets depth.

79. Opcard_GetConstGain

Prototype:

DLLReturnType

Opcard_GetConstGain (

int **deviceNr**,

double * **constantGain**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

constantGain returns constant gain (-31 … 65 dB, step: 0.5 dB)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current constant gain. Returned value is in dB unit.

80. Opcard_SetConstGain

Prototype:

DLLReturnType

Opcard_SetConstGain (

int **deviceNr**,

double **constantGain**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

constantGain constant gain in dB (-31 … 65 dB, step: 0.5 dB)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets gain parameter for constant mode.

81. Opcard_WriteData

Prototype:

DLLReturnType

Opcard_WriteData (

int **deviceNr**,

unsigned int* **data,**

int **sizeData**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

data – buffer containing TGC curve points

sizeData – length of used TGC curve in samples

Description:

Function sets TGC curve buffer for acquisition.

Example: User wants to make measurement using 'Store Disabled' mode containing **DATA_SIZE** = 500 samples. You must:

1. Initialize 256 KB array as **data**.
2. Fill **at least** $\frac{HeaderSize+DataSize}{2}=226$ first elements of this array with Gain [A.U.] values (in Disabled mode device uses the first TGC curve, 1 TGC point provided for 2 FIFO data points including header). It is recommended to create array of *unsigned char;* **data** must be defined as pointer of type *unsigned int* to this array.
3. **sizeData** defines size of filled elements of TGC buffer **including header** – in this example, **sizeData = (HEADER_SIZE + DATA_SIZE)/2** = 226.
4. Call **Opcard_WriteData** (**deviceNr**, **data**, **sizeData**);

82. Opcard_GetPdxMode

Prototype:

DLLReturnType

Opcard_GetPdxMode (

int **deviceNr**,

int **pdNr**,

int * **pdMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdMode returns information of peak detector mode

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current mode of specified peak detector. The parameters correspond with options as follows:

pdNr	PD name	
0	A	
1	B	
2	C	
pdMode	Name	Description
0	Level	first sample above the level
1	Rising	rising edge (sample below and next above the level)
2	Falling	falling edge (sample above and next below the level)
3	Transition	rising edge or falling edge

83. Opcard_SetPdxMode

Prototype:

DLLReturnType

Opcard_SetPdxMode (

int **deviceNr**,
int **pdNr**,
int **pdMode**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdMode is peak detector mode

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets mode of specified peak detector. The parameters correspond with options as follows:

pdNr	PD name	
0	A	
1	B	
2	C	
pdMode	Name	description
0	Level	first sample above the level
1	Rising	rising edge (sample below and next above the level)
2	Falling	falling edge (sample above and next below the level)
3	transition	rising edge or falling edge

84. Opcard_GetPdxEnable

Prototype:

DLLReturnType
Opcard_GetPdxEnable (
int **deviceNr**,
int **pdNr**,
bool * **pdEnable**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdEnable returns the information if PD is enabled (1) / disabled (0)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that specified PD is enabled/disabled (**pdEnable** = 1/0).

pdNr	PD name
0	A
1	B
2	C

85. Opcard_SetPdxEnable

Prototype:

DLLReturnType

Opcard_SetPdxEnable (

int **deviceNr**,

int **pdNr**,

bool **pdEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is peak detector number (one of three available detectors)

pdEnable enables (1) or disables (0) the PD

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function enables or disables the specified PD.

pdNr	PD name
0	A
1	B
2	C

86. Opcard_GetPdxResult

Prototype:

DLLReturnType

Opcard_GetPdxResult(

int **deviceNr**,

int **pdNr**,

bool * **pdResult**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdResult returns the information that PD found (1) or not found (0) event during last acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that specified PD found (**pdResult** = 1) event.

pdNr	PD name
0	A
1	B
2	C

87. Opcard_GetPdxStart

Prototype:

DLLReturnType

Opcard_GetPdxStart (

int **deviceNr**,

int **pdNr**,

int * **pdStart**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdStart returns the number of sample for which PD measurement window is

started. This parameter must not exceed range **0**...**pdStop** and should be less than DEPTH!

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the number of sample for which specified PD measurement window is started.

pdNr	PD name
0	A
1	B
2	C

88. Opcard_SetPdxStart

Prototype:

```
DLLReturnType
Opcard_SetPdxStart (
                        int deviceNr,
                        int pdNr,
                        int pdStart
                    );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdStart is a number of sample for which PD measurement window is started. This parameter must not exceed range **0**...**pdStop** and should be less than DEPTH!

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the number of sample for which specified PD measurement window is started.

pdNr	PD name
0	A
1	B
2	C

89. Opcard_GetPdxStop

Prototype:

DLLReturnType

Opcard_GetPdxStop (

int **deviceNr**,

int **pdNr**,

int * **pdStop**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdStop returns the number of sample for which PD measurement window is stopped. PdStop must not be lower than **pdStart** and must not exceed DEPTH value!

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the number of sample for which it is end of specified PD measurement window.

pdNr	PD name
0	A
1	B
2	C

90. Opcard_SetPdxStop

Prototype:

DLLReturnType

Opcard_GetPdxStop (

int **deviceNr**,

int **pdNr**,

int **pdStop**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdStop is the number of sample for which it is end of PD measurement window. PdStop must not be lower than **pdStart** and must not exceed DEPTH value!

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the number of sample for which it is end of specified PD measurement window.

pdNr	PD name
0	A
1	B
2	C

91. Opcard_GetPdxLevelPosition

Prototype:

DLLReturnType

Opcard_GetPdxLevelPosition (

int **deviceNr**,

int **pdNr**,

int * **pdLevelPos**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a number of peak detector (one of three available detectors)

pdLevelPos returns position value of level crossing

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the number of sample for which event of crossing the level occurred for specified PD.

pdNr PD name

0	A
1	B
2	C

92. Opcard_GetPdxLevelValue

Prototype:

DLLReturnType

Opcard_GetPdxLevelValue (

int **deviceNr**,

int **pdNr**,

char * **pdLevelVal**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdLevelVal returns reference level for PD – it ranges from -128 up to 127, which corresponds to -0.5 … +0.5 V.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks reference level for specified PD.

pdNr	PD name
0	A
1	B
2	C

93. Opcard_SetPdxLevelValue

Prototype:

DLLReturnType

Opcard_SetPdxLevelValue (

int **deviceNr**,

int **pdNr**,

char **pdLevelVal**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdLevelVal is reference level for PD – it ranges from -128 up to 127, which corresponds to -0.5 … +0.5 V.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets reference level for specified PD.

pdNr	PD name
0	A
1	B
2	C

94. Opcard_SetPdxLSSP

Prototype:

```
DLLReturnType
Opcard_SetPdxLSSP (
                        int deviceNr,
                        int pdNr,
                        int start,
                        int stop,
                        int level
                    );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

start is the number of sample for which PD measurement window is started

stop is the number of sample for which it is end of PD measurement window

level is reference level for PD

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets parameters of measurement window for specified PD.

pdNr	PD name
0	A
1	B
2	C

95. Opcard_GetPdxMaxPosition

Prototype:

DLLReturnType

Opcard_GetPdxMaxPosition (

int **deviceNr**,

int **pdNr**,

int * **pdMaxPos**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a number of peak detector (one from thee detectors)

pdMaxPos returns position of maximum value for PD window in Ts domain (range: 0...DEPTH)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the number of sample of maximum value for specified PD window.

pdNr	PD name
0	A
1	B
2	C

96. Opcard_GetPdxMaxValue

Prototype:

DLLReturnType

Opcard_GetPdxMaxValue (

int **deviceNr**,

int **pdNr**,

char * **pdMaxVal**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

pdNr is a peak detector number (one of three available detectors)

pdMaxVal returns the maximum value from measurement window for PD - it ranges from -128 up to 127, which corresponds to -0.5 … +0.5 V.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks maximum value from measurement window for specified PD.

pdNr	PD name
0	A
1	B
2	C

97. Opcard_GetEncxEnable

Prototype:

DLLReturnType

Opcard_GetEncxEnable (

int **deviceNr**,

int **encNr**,

bool * **enable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

enable returns information that encoder is disabled (0) or enabled (1)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that specified encoder is disabled (**enable** = 0) or enabled (**enable** = 1)

98. Opcard_SetEncxEnable

Prototype:

DLLReturnType

Opcard_SetEncxEnable (

int **deviceNr**,

int **encNr**,

bool **enable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number (0 = ENC A, 1 = ENC B)

enable disables (0) or enables (1) the encoder

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function disables (**enable** = 0) or enables (**enable** = 1) specified encoders.

99. Opcard_SetEncxReset

Prototype:

DLLReturnType

Opcard_SetEncxEnable (

int **deviceNr**,

int **encNr**,

bool **reset**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

reset value = 1 does a reset of encoder current position and position captured during last acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function does a reset of positions of specified encoder.

100. Opcard_GetEncxDirectionInvert

Prototype:

DLLReturnType

Opcard_GetEncxDirectionInvert (

int **deviceNr**,

int **encNr**,

bool * **dirInvert**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

dirInvert returns encoder mode of position counting direction

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks specified encoder mode of position counting direction.

dirInvert = 0 – normal mode, input of CHA is before input of CHB

dirInvert = 1 – inverted mode, input of CHB is before input of CHA

101. Opcard_SetEncxDirectionInvert

Prototype:

DLLReturnType

Opcard_SetEncxDirectionInvert (

int **deviceNr**,

int **encNr**,

bool **dirInvert**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

dirInvert is encoder mode of position counting direction

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets specified encoder mode of position counting direction.

dirInvert = 0 – normal mode, input of CHA is before input of CHB

dirInvert = 1 – inverted mode, input of CHB is before input of CHA

102. Opcard_GetEncxIndexEnable

Prototype:

DLLReturnType

Opcard_GetEncxIndexEnable (

int **deviceNr**,

int **encNr**,

bool * **idxEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

idxEnable returns setting of Index (IDX) input

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks the setting of Index (IDX) input for specified encoder

idxEnable = 0 – IDX input is ignored

idxEnable = 1 – IDX input is unlocked

103. Opcard_SetEncxIndexEnable

Prototype:

DLLReturnType

Opcard_SetEncxIndexEnable (

int **deviceNr**,
int **encNr**,
bool **idxEnable**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

idxEnable is setting of Index (IDX) input

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the setting of Index (IDX) input for specified encoder

idxEnable = 0 – IDX input is ignored

idxEnable = 1 – IDX input is unlocked

104. Opcard_GetEncxDecodeMode

Prototype:

DLLReturnType

Opcard_GetEncxDecodeMode (
int **deviceNr**,
int **encNr**,
bool * **decMode**
);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

decMode returns decoding mode of input pulses

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks mode of input pulses for specified encoder.

decMode	mode	description
0	1X	only rising edges on CHA are counted
1	2X	rising and falling edges on CHA are counted

2	4X	both edges for CHA and CHB are counted

For modes 1X and 2X the CHB specifies only direction of counting.

105. Opcard_SetEncxDecodeMode

Prototype:

DLLReturnType

Opcard_SetEncxDecodeMode (

int **deviceNr**,

int **encNr**,

bool * **decMode**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

decMode is decoding mode of input pulses

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets mode of input pulses for specified encoder.

decMode	mode	description
0	1X	only rising edges on CHA are counted
1	2X	rising and falling edges on CHA are counted
2	4X	both edges for CHA and CHB are counted

For modes 1X and 2X the CHB specifies only direction of counting.

106. Opcard_GetEncxFilterEnable

Prototype:

DLLReturnType

Opcard_GetEncxFilterEnable (

int **deviceNr**,

int **encNr**,

bool * **filterEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

filterEnable returns information that filter of encoder enabled or disabled

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks that filter specified encoder is enabled (**filterEnable** = 1) or disabled (**filterEnable** = 0).

107. Opcard_SetEncxFilterEnable

Prototype:

DLLReturnType

Opcard_SetEncxFilterEnable (

int **deviceNr**,

int **encNr**,

bool **filterEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

filterEnable enables or disables filter of encoder

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function enables (**filterEnable** = 1) or disables (**filterEnable** = 0) filter for specified encoder.

108. Opcard_GetEncxCompareEnable

Prototype:

DLLReturnType

Opcard_GetEncxCompareEnable (

int **deviceNr**,

int **encNr**,

bool * **compEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

compEnable returns information that comparator of position is locked.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks comparator of position for specified encoder. For **compEnable** = 0 comparator of position is locked and for **compEnable** = 1, comparator of position is unlocked and trigger generation is possible.

109. Opcard_SetEncxCompareEnable

Prototype:

DLLReturnType

Opcard_SetEncxCompareEnable (

int **deviceNr**,

int **encNr**,

bool **compEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

compEnable is state a comparator of position.

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets comparator of position for specified encoder. For **compEnable** = 0 comparator of position is locked and for **compEnable** = 1, comparator of position

is unlocked and trigger generation is possible.

110. Opcard_GetEncxCompareStep

Prototype:

DLLReturnType

Opcard_GetEncxCompareEnable (

int **deviceNr**,

int **encNr**,

int * **compStep**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

compStep returns the setting of step of comparator of position for triggering of acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks step of position comparator used for triggering acquisition for specified encoder. If **compStep** numbers of step has been passed by encoder, pulse is generated and acquisition is triggered. After that, encoder position register is modified by **compStep**. Maximum **compStep** is 255.

111. Opcard_SetEncxCompareStep

Prototype:

DLLReturnType

Opcard_SetEncxCompareEnable (

int **deviceNr**,

int **encNr**,

int **compStep**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

compStep is the setting of step of comparator of position for triggering of acquisition

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets step of position comparator used for triggering acquisition for specified encoder. If **compStep** numbers of step has been passed by encoder, pulse is generated and acquisition is triggered. After that, encoder position register is modified by **compStep**. Maximum **compStep** is 255.

112. Opcard_GetEncxPosition

Prototype:

DLLReturnType

Opcard_GetEncxPosition (

int **deviceNr**,

int **encNr**,

int * **position**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

position returns current position of encoder register (32-bit: 0...4294967295)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks current position of specified encoder (32-bit: 0...4294967295).

113. Opcard_GetEncxCapture

Prototype:

DLLReturnType

Opcard_GetEncxCapture (

int **deviceNr**,

int **encNr**,

int * **capture**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

capture returns captured position of encoder register during triggering last acquisition (32-bit: 0...4294967295)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks captured position of specified encoder register during triggering last acquisition (32-bit: 0...4294967295).

114. Opcard_GetEncxFilter

Prototype:

DLLReturnType

Opcard_GetEncxFilter (

int **deviceNr**,

int **encNr**,

int * **filterLength**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

filterLength returns length of filter: 0...255 x 10 MHz cycle period = 0...25.5 us

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function checks length of filter of specified encoder for inputs CHA, CHB and IDX. This is the number of 10 MHz cycle period, for which input state has to remain unchanged, to be recognized as correct.

115. Opcard_SetEncxFilter

Prototype:

DLLReturnType

Opcard_SetEncxFilter (

int **deviceNr**,

int **encNr**,

int **filterLength**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

encNr is encoder number

filterLength is a length of filter: 0...255 x 10 MHz cycle period = 0...25.5 us

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets a length of filter of specified encoder for inputs CHA, CHB and IDX. This is the number of 10 MHz cycle period, for which input state has to remain unchanged, to be recognized as correct.

116. Opcard_SetSeqEnable

Prototype:

DLLReturnType

Opcard_SetSeqEnable (

int **deviceNr**,

int **segEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

segEnable enables (1) or disables (0) the sequence mode

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets a sequence mode. Sequence is locked for **segEnable** = 0 or switching of sequences is enabled for **segEnable** = 1.

117. Opcard_GetSeqEnable

Prototype:

DLLReturnType

Opcard_GetSeqEnable (

int **deviceNr**,

int* **segEnable**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

segEnable returns enable (1) / disable (0) state of sequence mode

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets a sequence mode. Sequence mode is locked if **segEnable** = 0; when sequence auto-switching is enabled, **segEnable** = 1.

118. Opcard_SetSeqIndex

Prototype:

DLLReturnType

Opcard_SetSeqIndex (

int **deviceNr**,

int **seqIndex**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

seqIndex is current sequence index (**0**...**seqLength**)

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets the current sequence index. The next acquisition will be triggered for parameters specified for **seqIndex** index sequence. Switching **seqIndex** modifies acquisition parameters. **seqIndex** can not be higher than sequence length, which can be modified using function **Opcard_SetSeqLenth**.

119. Opcard_GetSeqIndex

Prototype:

DLLReturnType

Opcard_GetSeqIndex (

int **deviceNr**,

int* **seqIndex**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

seqIndex returns current sequence index

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns current sequence index. The next acquisition will be triggered for parameters specified for **seqIndex** index sequence. Switching **seqIndex** modifies sequence parameters. **seqIndex** can not be higher than sequence length, which can be modified using function **Opcard_SetSeqLenth**.

120. Opcard_SetSeqLength

Prototype:

DLLReturnType

Opcard_GetSeqLength (

int **deviceNr**,

int* **segLength**

);

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

segLength is number of available sequences for acquisition (1...1024).

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function sets number of available sequences for acquisition. If the index of sequence will be higher than **segLength**, the index will be changed to **0**. Maximum **segLength** available to set is 1024.

121. Opcard_GetSeqLength

Prototype:

```
DLLReturnType
Opcard_SetSeqLength  (
                        int deviceNr,
                        int segLength
                      );
```

Parameters:

deviceNr is ordinal number of Opcard in list of Opcards detected by system

segLength returns number of available sequences for acquisition (1...1024).

Returns:

Returns 0 (zero) when success. Otherwise return an error code.

Description:

Function returns number of available sequences for acquisition. If the index of sequence will be higher than **segLength**, the index will be changed to **0**. Maximum **segLength** is 1024.