PicoScale

Programmer's Guide

www.smaract.com

SmarAct GmbH, Schuette-Lanz-Strasse 9, D-26135 Oldenburg
Phone: +49 (0) 441 - 800879-0, Telefax: +49 (0) 441 - 800879-21
Internet: www.smaract.com, E-Mail: info@smaract.com

Document Version: 2.1.0

TABLE OF CONTENTS

1 *INTRODUCTION*

This document describes the application programming interface (API) of the SmarAct PicoScale Laserinterferometer. It may be used to control one or more PicoScale devices by software. Its purpose is the transmission of sensor data from a device to a host.

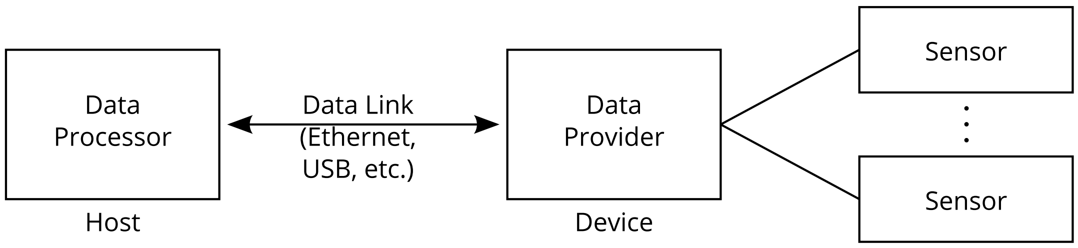


Figure 1.1: General Setup

Besides the description of the actual API usage in chapter 4, the function reference in chapter 5, the property reference in chapter 6 and the event reference in chapter 7, chapters 2 and 3 describe some basics that will help to understand how to configure and use the SmarAct PicoScale Laserinterferometer.

1.1 Terminologies

This section defines general terminologies that are used throughout this document. This section only gives a brief summary and the terminologies are explained in more detail later in this document.

Master The unit that requests sensor data so that it can be further processed is referred to as master (e.g. the host computer).

Slave The unit that produces sensor data and provides it to the master using the API specified in this document is referred to as slave (the PicoScale Laserinterferometer).

Data Type A data type refers to a format in which data values are transmitted from the slave to the master (e.g. 16-bit signed, double etc.).

Data Source A data source is a component of the slave that produces data which may be transmitted to the master. Typical data sources are sampled analog data, but also processed analog data (e.g. sin/cos data pairs to form position data).

Data Source Type The data source type specifies what kind of data a data source produces (e.g. position, temperature, etc.).

Channel A channel is a group of data sources that are related to each other and together form a unit. A slave may offer several channels.

Data Frame A data frame holds a complete set of data from data sources and represents one time slot.

Data Packet A data packet contains one or more data frames. Continuously sent data packets form the output data stream.

Frame Rate The frame rate is the frequency at which the slave transmits data frames to the master. The frame rate is given in Hz. The maximum frame rate depends on several factors (e.g. bit rate, data frame size etc.).

2 *BASIC CONCEPTS*

This chapter describes some basic concepts of the PicoScale Laserinterferometer, its configuration and communication.

2.1 Properties

Properties are configuration values that define the behavior of the device. Each property has a data type (see table A.3) and an access mode. Some properties may be read and written, while others are read only or (in rare cases) write only.

Generally, there are three categories of properties:

- **Basic properties** mainly cover device meta information and stream configuration (see section 2.4.1). These properties are shared among all SmarAct Sensor devices. They are further divided into
  - *device* properties,
  - *channel* properties and
  - *data source* properties (see section 2.2 about data sources).
- **Device specific properties** control the behavior of functionalities that are specific to a given SmarAct Sensor device (such as calibration routines etc.).
- **Library properties** configure the behavior of the library (API) in various details.

See chapter 6 for a list of available properties and their descriptions.

2.1.1 Property Keys

A property is identified by a *property key*. A property key is a 32-bit value that has the following structure:

	Property Key		
Bits	31 - 16	15 - 8	7 - 0
Meaning	Property Code	IndexH	IndexL

The *property code* actually identifies the property while the *index* values further specify the key. Some properties do not require index values, some only use one index value and others use both index values. When an index value is unused it should be set to `0x00` by default.

When accessing properties (e.g. with `SA_SI_GetProperty_i32`) you must provide a property *key* (not a property *code*) to the function. The easiest way to do this is to use the `SA_SI_EPK` helper function (see the example on page 77).

2.2 Data Sources

Data sources are the basic elements of a device that provides sensor data (the slave). The slave offers data sources that the master may tap. A data source provides data of any digital form over time.

Data sources that relate to each other are bundled together to form a channel. The number of channels a slave provides and the data sources that each channel has is defined by the architecture of the slave and cannot be changed. The structure may be queried by reading properties of the slave. Table C.1 in the appendix contains a list of all data sources the PicoScale currently offers.

Figure 2.1 shows an example of a slave with five data sources that are grouped into two channels.

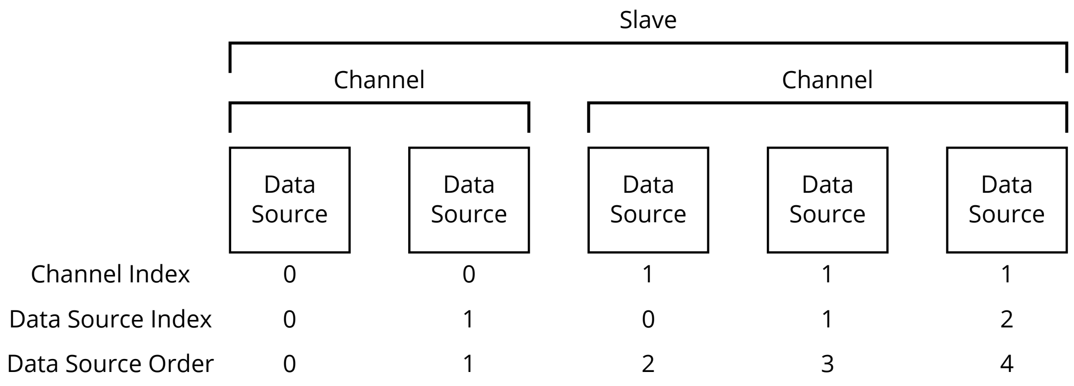


Figure 2.1: Data Sources

Data sources are addressed by supplying a channel index and a data source index. As indicated in the figure the data sources are also given a fixed order (data source order). This order is relevant when assembling data frames (see section 2.3).

Each data source has a set of properties. Some are constant read-only properties that describe the capabilities of the data source. Others may be read and written to configure the data source. Please refer to chapter 6 for a detailed description of all properties.

2.2.1 Retrieving Data Source Values

The values of data sources may be retrieved from the device in two different ways:

- **Polling** This is the easiest way to get data from the device: a simple function call will return the data source value (see functions `SA_SI_GetValue_i64` and `SA_SI_GetValue_f64`). The drawback of this method is that the values returned neither have a defined timing nor are they synchronized with other values. The device simply returns the value the data source currently has.
- **Streaming** This method enables the user to select a sub set of all available data sources and let the device stream the values of these data sources to the host over a period of time. While the data stream is active all selected data sources are synchronized and sampled in constant time intervals.

  See section 2.4 for more information.

Related Properties

Data Source Type, *Data Type*, *Available Compression Modes*, *Compression Mode*, *Streaming Enabled*, *Base Unit*, *Base Resolution*, *Resolution Shift*, *Data Source Name*, *Is Streamable*, *Component ID*, *Component Index*

2.3 Data Frames

Data frames are used when streaming data source values to the master. A data frame is a block of bytes that contains data from one or more data sources. Each frame represents one time slot and the data that the frame contains was captured during this time slot. Contents, structure and size of a data frame vary depending on how the data sources are configured. When the slave assembles a data frame it primarily iterates over its channels and secondarily over the data sources of each channel (see data source order in figure 2.1). The data from a data source is only added to the frame if the data source is enabled for streaming (see property *Streaming Enabled* on page 115). The number of bytes added depends on the data type of the data source. This implies that the master must know how the slave is configured in order to be able to interpret a received data frame correctly.

Example Frame Structure

The following table shows an example configuration of a slave that offers two channels. Channel 0 has two data sources, channel 1 has three data sources.

Property	Value				
Number of Channels	2				
Channel Index	0		1		
Data Source Index	0	1	0	1	2
Data Type	U16	S32	U16	U16	S48
Streaming Enabled	no	yes	yes	no	yes

Three data sources are enabled. In this case a data frame would have 12 bytes (4 bytes from data source [0,1], 2 bytes from data source [1,0] and 6 bytes from data source [1,2]).

2.3.1 Data Frame Rates

The rate at which the slave generates data frames may be configured by the master. Note that this rate is a parameter that applies to all data sources equally. It is not possible to have the slave send data from one data source at some rate and data from another data source at a different rate.

Note that a slave may not support all frame rates. When the master configures the desired frame rate it should read back the actual frame rate as the slave will choose a frame rate it is able to generate that is closest to the masters wish.

Reconstructing Time

Frames do not contain a time stamp. As mentioned above, each frame represents one time slot. The time stamp of a data frame may be reconstructed implicitly by the frame rate and order of the frames. For example, with a frame rate of 10 kHz the frame with the order number 5000 occurred at time t = 0.5 s relative to the start of the data stream.

2.3.2 Data Frame Aggregation

Data frame aggregation is a technique that may be used to increase the data throughput of the stream by reducing the protocol overhead. By default, each data frame is wrapped up in a stream packet and sent to the master. However, when data frame aggregation is enabled a packet may contain two or more data frames. Thus, only one packet header is generated to transmit several data frames.

The *aggregation value* determines how many data frames are merged to a single packet. The minimum (and default) aggregation value is 1 (no aggregation). Figure 2.2 shows an example stream for different aggregation values.

As with the structure of a single frame the master must know how many frames a packet contains in order to be able to interpret the packet correctly.

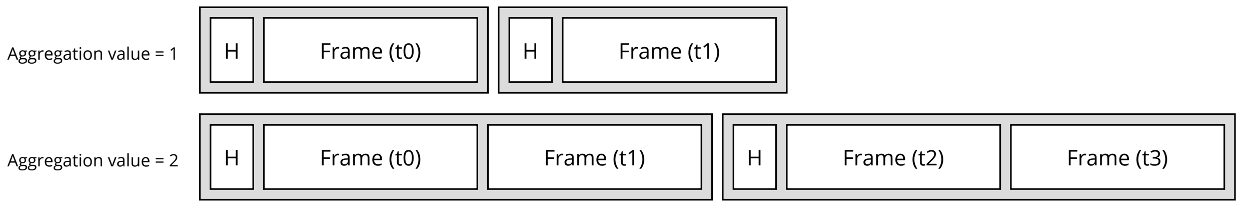


Figure 2.2: Frame aggregation

Although the API user does not need to bother with decoding of stream packets (this is handled by the library and data frames are passed to the user via data buffers, see section 4.5) it is still important to understand the concept of frame aggregation, since changing the aggregation value influences the maximum possible frame rate. See section 2.4.1 for more information.

2.4 Data Stream

Besides all the complex configuration and calibration options the PicoScale Laserinterferometer offers, the data stream is the actual "work horse" when it comes down to effectively getting position data out of the device. While it is possible to poll position data manually it is much more efficient to let the slave generate data at fixed intervals and transfer them to the master.

The data stream may be globally enabled or disabled by the master. When enabled (see property *Streaming Active*) the slave generates a continuous stream of stream packets until the stream is disabled again. Each packet contains one or more data frames and each frame contains the data from all data sources that are enabled for streaming.

2.4.1 Stream Configuration

The stream that the slave generates may be configured regarding which data is sent and with which timing. This is influenced by several parameters:

- The **Data Source configuration** determines which data sources are enabled for streaming (*Streaming Enabled* property) and which compression modes are used for transmission (*Compression Mode* property). This defines the frame size (number of bytes per frame).
- The **Frame Aggregation** determines how many frames are transmitted in each data packet (*Frame Aggregation* property). This defines the packet size (together with header and frame size).
- The **Frame Rate** determines how many frames per second are transmitted to the master (*Frame Rate* property).

Although these parameters may be freely set by the master there are also dependencies between them. Changing one parameter may change the valid range of the other parameters.

Therefore, it is recommended to configure the data sources first, then the frame aggregation and then the frame rate, since e.g. a change of the frame aggregation may implicitly change the frame rate.

2.4.2 Data Compression

To improve data throughput of the output data stream data sources may optionally offer a compression mode (see *Available Compression Modes* property). If enabled the data from the data source is compressed by the slave and must be decompressed by the master to regain the original data. There are different types of compression (see table A.5) which are described here.

NOTICE
Although the API user must activate the data compression via the corresponding property to improve performance the library takes care of the decompression automatically. Data frames received from the API via frame buffers are always uncompressed.

Lower Bits Absolute

These modes avoid transmitting the full bit width of the data type of the data source and instead only transmit the lower fraction of the value in each frame. The beginning of the data stream contains an uncompressed data frame and serves as a reference for all following (compressed) frames. The absolute data values may be reconstructed by comparing two consecutive fractions. These fractions must only differ by a certain amount. Otherwise the reconstruction fails.

As a result the data compression causes the data stream to use less bandwidth compared to no compression. Put it the other way around the compression allows to increase the frame rate.

Related Properties

Streaming Enabled, *Maximum Frame Aggregation*, *Frame Aggregation*, *Maximum Frame Rate*, *Frame Rate*, *Available Compression Modes*, *Compression Mode*

2.4.3 Stream Generation

When enabling the data stream (see *Streaming Active* property) the stream generator starts producing data frames. However, the PicoScale stream generator allows to configure when and with which timing data frames are produced. Figure 2.3 shows a structural overview of the stream generator.

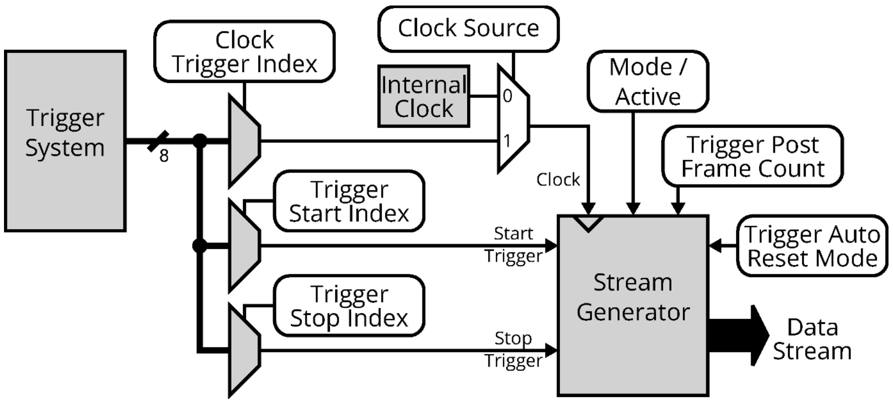


Figure 2.3: Stream Generator

The stream generator supports several modes of operation (see *Streaming Mode* property):

- `SA_SI_DIRECT_STREAMING` (`0x01`) In this mode a data frame will be generated on each clock pulse. The inputs *StartTrigger* and *StopTrigger* are ignored. The stream ends when disabling the stream again.
- `SA_SI_TRIGGERED_STREAMING` (`0x02`) Same as direct streaming, but clock pulses are ignored until a rising edge on the *StartTrigger* input is detected. After this, as soon as a rising edge on the *StopTrigger* input is detected the stream generator will produce a number of additional data frames (configurable via the *SG Trigger Post Frame Count* property) before

stopping the data stream. If start and stop trigger occur at the same time then the stream generator will generate *PostFrameCount* data frames and then stop.

Note that by default the stream generator will only react to one occurring start/stop trigger. After the stop trigger further start triggers are ignored and will *not* restart the stream. This behavior may be altered with the *SG Trigger Auto Reset Mode* property (see there).

The clock used by the stream generator may be selected via the *SG Clock Source* property. Note that the internal clock is configured via the *Frame Rate* property (see section 2.3.1). Selecting the trigger system as the clock source allows to feed external clocks to the stream generator. The trigger system is described in section 3.2.

Related Properties

Streaming Active, *Streaming Mode*, *SG Clock Source*, *SG Clock Trigger Index*, *SG Trigger Start Index*, *SG Trigger Stop Index*, *SG Trigger Post Frame Count*, *SG Trigger Auto Reset Mode*

2.5 Event Notification

In some situations events might occur that require further attention or reactions by the master. To avoid that the master has to poll the occurrence of such events the slave offers a notification system that may be configured by the master. Table A.2 shows a list of available event types. The master may "subscribe" to an event by setting the *Event Notification Enabled* property for the event to `SA_SI_ENABLED` (`0x01`). If an event occurs and the corresponding notification is enabled then the slave generates a notification event for the master informing about the situation.

The API user may receive events using the `SA_SI_WaitForEvent` function.

Each event is passed a 32-bit event parameter which typically gives a more detailed hint of what happened. The meaning of the parameter value depends on the event. These are described in chapter 7.

NOTICE
API events (such as `SA_SI_STREAMBUFFER_READY_EVENT`) cannot be subscribed to. They are always enabled. API events have a code of `0xf000` and higher.

Related Properties

Event Notification Enabled

2.6 Auto Functions

Auto functions are complex configuration processes that, once triggered, automatically perform a sequence of internal actions to optimize system performance.

This section describes the available auto functions.

2.6.1 Adjustment

The adjustment auto function performs everything necessary to achieve high accuracy position data and should be run before configuring and enabling the data stream. It has three states:

- **Disabled** By default the auto function is disabled.
- **Manual Adjust** In this state the user must adjust the mirrors that reflect the IR laser back into the device in order to get the highest possible signal quality. Once satisfied, the user must switch to the *AutomaticAdjust* state to complete the adjustment.
- **Automatic Adjust** In this state the device performs internal optimizations to improve the accuracy of the position data. Once done it will automatically switch to the *Disabled* state.

When starting the adjustment auto function the typical approach is to set the *Adjustment State* property to `SA_PS_ADJUSTMENT_STATE_MANUAL_ADJUST` (`0x01`). When done adjusting the mirrors set the property to `SA_PS_ADJUSTMENT_STATE_AUTO_ADJUST` (`0x02`) and wait until the device has switched to `SA_PS_ADJUSTMENT_STATE_DISABLED` (`0x00`). This may be done by either waiting for the event `SA_PS_AF_ADJUSTMENT_PROGRESS_EVENT` (`0x8a80`) or polling the *Adjustment State* property.

Note that a direct transition from the *Disabled* state to the *AutomaticAdjust* state is not allowed. To abort the adjustment function simply switch to the *Disabled* state.

Please refer to the PicoScale User Manual for more information on the adjustment function.

Related Properties

Adjustment State, *Adjustment Progress*

2.7 Device Features

In its basic form the SmarAct PicoScale offers a set of features that cover the most common use cases for measuring positions. However, some additional features may be purchased and unlocked by software to allow more sophisticated measuring setups.

Each device feature is in one of three states:

- **Disabled** Indicates that the feature is locked and may not be used. An error will be generated when trying to modify feature related properties (`SA_SI_PERMISSION_DENIED_ERROR` (`0x001f`)).

- **Evaluation** Indicates that the feature is *temporarily* unlocked and may be used for evaluation purposes.
- **Enabled** Indicates that the feature is unlocked permanently.

The table below lists the currently available device features:

Index	Feature Name	Page
0	Advanced Trigger System	27
1	Signal Generators	43
2	Calculation System	40

By default, each feature has a certain amount of evaluation periods available (issue a read on the *Feature Evaluate* property). If evaluation periods are left you may activate one by writing `SA_SI_ENABLED` (`0x01`) to the *Feature Evaluate* property. This will decrement the available evaluation periods and increase the evaluation time by a certain amount. Note that several periods may be activated at once (if available) to extend the evaluation time accordingly.

To purchase a device feature permanently, please contact SmarAct.

Related Properties

Feature Count, *Feature Name*, *Feature Time*, *Feature Evaluate*

2.8 Full Access vs. Limited Access Connections

The SmarAct PicoScale Laserinterferometer offers several connection interfaces (such as USB, Ethernet, etc.). While it is possible to connect to the device via more than one interface at the same time, modifying certain global configuration settings simultaneously would cause strange system behavior. Therefore, modifications to these settings are limited to one interface only, although the user(s) may choose dynamically which interface is used for these modifications.

An interface connection can be of type *full access* or *limited access*. A full access connection may modify any settings while limited access connections may only modify settings local to its interface and which do not interfere with other settings. Violating this policy will result in an `SA_PS_NO_FULL_ACCESS_ERROR` (`0x8002`).

By default a connection to the device is a limited access connection. In order to have full access to the device a full access connection must be acquired by setting the *Full Access Connection* property to `SA_SI_ENABLED` (`0x01`). (See section 2.1 about properties.) Before disconnecting from the device the full access connection should be released by setting the *Full Access Connection* property back to `SA_SI_DISABLED` (`0x00`).

Should another interface already be owner of the full access connection while trying to acquire it then the write operation to the property will fail. However, in this case it is possible to "steal" a full access connection by setting the *Full Access Connection* property to `SA_SI_DISABLED` (`0x00`) (thus forcing the other connection to become a limited access connection) before acquiring the full access connection normally. The interface that the full access connection is stolen from will generate a `SA_PS_FULL_ACCESS_CONNECTION_LOST_EVENT` (`0x8000`) event to notify the application software that it no longer is the owner of the full access connection.

2.9 Data Objects

In contrast to properties which are used to modify single configuration values of the device, data objects are data blocks that are stored on the device that may be used to manage complex data structures, e.g. look-up tables. Appendix B lists the available data objects and their meanings.

Data objects are identified and addressed via an *object name* which is a null terminated string with a maximum length of 63 characters plus the null termination. Some data objects may be read and written while others may only be read.

As an example, a custom signal shape for one of the signal generators may be uploaded to the device via a data object (see section 3.6).

The API functions `SA_SI_ReadDataObject` and `SA_SI_WriteDataObject` are used to access data objects (see their descriptions for more information).

2.10 Configuration Manager

While many measurement setups may be fairly simple to configure, some setups are more sophisticated and require a lot of property configuration each time the device is powered up. As a convenience feature the PicoScale offers two types to manage configurations:

Configuration Slots

Configuration slots are used to store the configuration settings to non-volatile memory. When a measurement is to be repeated in a later session the configuration may be loaded from a previously saved slot to avoid the complete reconfiguration of the PicoScale.

The following properties are used for the configuration slot interface:

- The *Configuration Count* property may be queried for the number of available configuration slots.
- The *Configuration Name* property may be used to give a configuration slot a generic name.
- The *Configuration Save* property writes the current PicoScale configuration to a memory slot.
- The *Configuration Load* property reads a previously stored configuration from a memory slot.

Upload/Download Configurations

The `SA_SI_WriteDataObject` function may be used to upload or download configurations to or from the device. Please refer to appendix B.4 for information on the format of this data object. After uploading a configuration, the containing property settings and thus the system state will be set automatically.

The following modules and interfaces will be saved by the configuration manager (except the enabled states of modules):

- **Modules:**
  - Advanced Trigger System
  - Calculation System
  - Clock Generator
  - Counter
  - Signal Generator
- **Interfaces:**
  - DAC
  - Digital IO
  - Digital Differential

2.11 Head Types

The PicoScale Laserinterferometer offers various head types that may be connected to the device. Depending on the application some head types may be more suitable than others. For proper operation you must configure each channel of the device with the head type that is connected to the channel. The head type is written on the housing of each head and in the corresponding data sheet.

Head types are divided into categories. Each category may have several head types. The list of categories and head types in each category that the device supports may be queried by the following properties:

- *Head Type Category Count* holds the number of head type categories.
- *Head Type Category Name* holds a descriptive name for the selected category.
- *Head Type Count* holds the number of head types that the selected category has.
- *Head Type Name* holds a descriptive name for the selected head type within the selected category.

The connected head type for a channel may be configured with the *Head Type* property. The value of this property has the following structure:

Bits 32 - 16	Bits 15 - 8	Bits 7 - 0
Unused	Category Index	Head Type Index

2.12 System Filters

The PicoScale data sources Position 1-3 and ADC 1-3 are subject to low pass filtering before they are provided to other internal components, including the stream generators. The system filter setting may configured with the *Filter Rate* property. This global setting affects the data streaming of all interfaces.

In contrast, the *Frame Rate* property is an interface specific setting and is independent from the filter rate. However, filter rate and frame rate should be matched to avoid effects like oversampling or aliasing.

Example: The global filter rate is set to 2.5 MHz and the Ethernet frame rate is also set to 2.5 MHz. Changing the filter rate via USB to match its frame rate of e.g. 5 MHz will then cause a mismatch between the filter rate and the Ethernet frame rate.

For these cases a special event may be generated that notifies an application if the global filter rate setting changes (see *Filter Setting Changed Event* event).

For more information please refer to the document *Customer Information CI00002*.

3 *COMPONENT DESCRIPTION*

This chapter descibes the sub components of the PicoScale Laserinterferometer and their configuration.

3.1 Environmental Sensor

The environmental sensor is an external module that may be attached to the PicoScale Laserinterferometer to measure environmental data (temperature, pressure and humidity) and apply it to the position calculation.

Figure 3.1 gives an overview for the software configuration of the module:

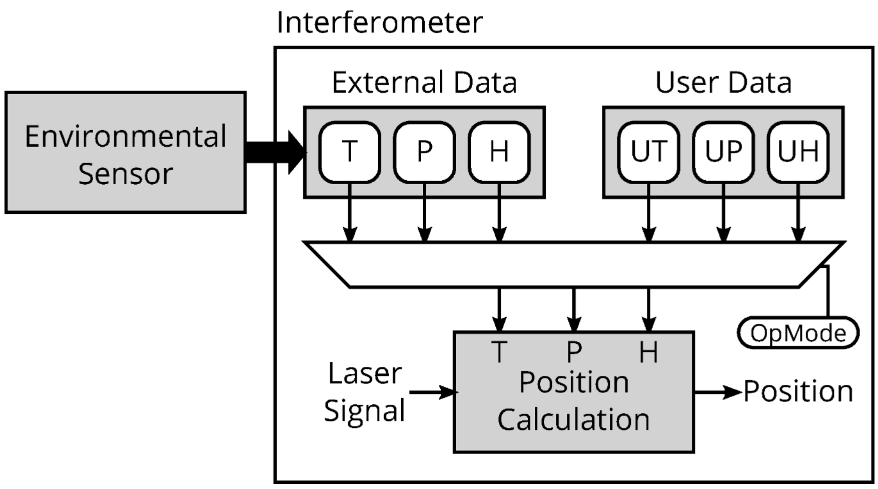


Figure 3.1: Environmental Data Processing

There are two modes of operation for the sensor module (*OpMode*):

- `SA_PS_ENV_OP_MODE_AUTOMATIC` (`0x01`, default): In this mode data from the environmental sensor is read continuously and used to correct the position data.
- `SA_PS_ENV_OP_MODE_MANUAL` (`0x00`): In this mode user defined environmental data is used to correct the position data.

Note: In order for the position data to account for the environmental data the *Dead Path Correction Enabled* property must be set to `SA_SI_ENABLED` (`0x01`).

Related Properties

Env Sensor Operation Mode, *Env Sensor State*, *Env Sensor Version*, *Env Sensor Temperature*, *Env Sensor Pressure*, *Env Sensor Humidity*, *Env User Temperature*, *Env User Pressure*, *Env User Humidity*, *Dead Path Correction Enabled*, *Dead Path*

3.2 Trigger System

The trigger system is a versatile sub system that may be used to control and synchronize internal and external processes. Figure 3.2 shows a block diagram of the trigger sub system.

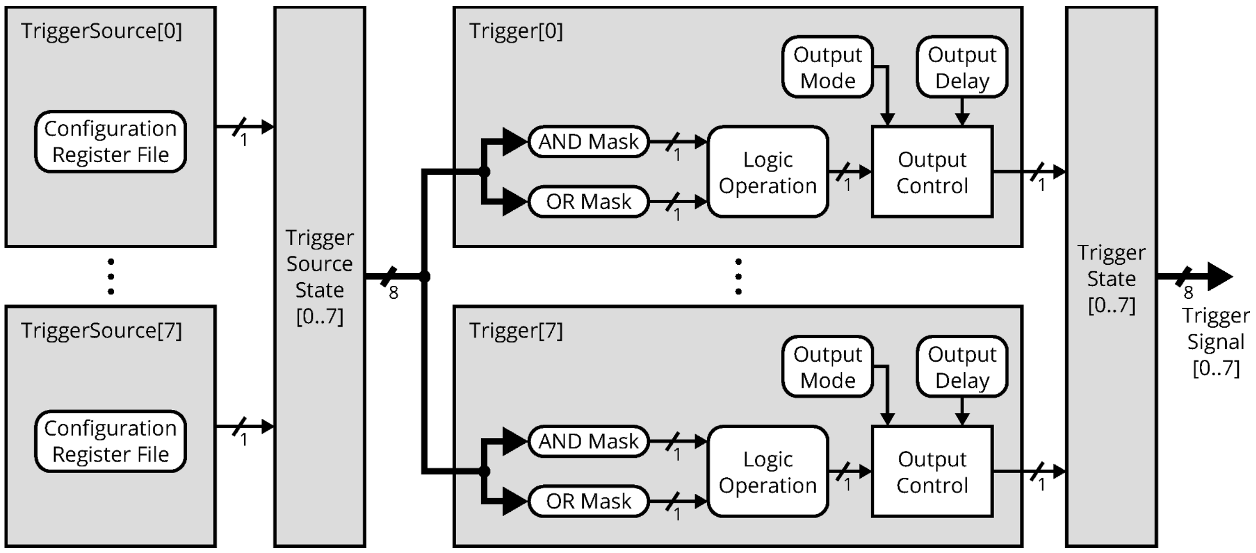


Figure 3.2: Trigger Sub System

The system offers 8 *trigger sources* and 8 *triggers* that may be configured and combined to generate *trigger signals* which may be used internally or externally.

The following components may make use of the trigger signals:

- **Stream Generator** Trigger signals may be used to
  - start the data stream
  - stop the data stream
  - define the rate at which data frames are generated

  For this the stream generator must be configured accordingly (see section 2.4.3).
- **Clock Generator** Trigger signals may be used to
  - start a clock generator
  - stop a clock generator

  For this the clock generators must be configured accordingly (see section 3.3).

- **Counters** Trigger signals may be used to
  - start a counter
  - stop a counter
  - increment a counter (let the counter count rising edges of the trigger signal)

  For this the counters must be configured accordingly (see section 3.4).
- **Digital GPIO** Trigger signals may be directly output to a digital GPIO pin in order to trigger external logic. For this the pin must be configured accordingly (see section 3.7.3).

The next sections describe the components of the trigger sub system in more detail.

NOTICE
The full functionality of the trigger system is only available if the device feature "Advanced Trigger System" is enabled (see also section 2.7 "Device Features"). Basic triggering only offers a very limited functionality. See section 3.2.5 for more information.

3.2.1 Trigger Sources

A trigger source may be configured to listen to events and how it should react to them. This is controlled by the trigger source configuration registers which are shown in the table below.

Register	Data Type
Trigger Source Reset	I32
Trigger Source Event	I32
Trigger Source Index 0	I32
Trigger Source Index 1	I32
Trigger Source Condition	I32
Trigger Source Value 0	I64
Trigger Source Value 1	I64

Each trigger source has a set of these registers which are accessed via according properties (see section 6.9). Depending on the content of the *Event* register the other registers are used differently. This is described in section 3.2.2.

A trigger source can be in one of two states: high and low. In the high state it will output a logic high level and in the low state it will output a logic low level. Depending on the configured event the trigger source may switch back and fourth between the high and low states. A write access to the *Reset* register always resets the trigger source to the low state.

3.2.2 Trigger Source Events

This section describes the available trigger source events (see also table A.13).

None (`0x00`, default)

When this event is configured then the trigger source will remain in the low state and continuously output a low level.

The other register fields are not used and have no meaning.

Software (`0x01`)

This event performs a state change of the trigger source when a software event is received (a write to the *Soft Trigger* property).

Register field usage:

- **Index0** This value is compared with the trigger ID given with the software event. On a match the trigger source switches to the state given by the property value (high when the value is `SA_SI_ENABLED` (`0x01`) and low when the value is `SA_SI_DISABLED` (`0x00`)).

The other register fields are not used and have no meaning.

Data Source Value (`0x02`)

This event performs a state change of the trigger source when the value of a data source passes a certain threshold or enters a certain range.

NOTICE
Not all data sources may be used to trigger this event. Only position, ADC and counter data sources may be used.

Register field usage:

- **Index0** This field is used as the channel index of the target data source.
- **Index1** This field is used as the data source index within the selected channel.
- **Condition** This field defines how the selected data source value and the configured threshold(s) are compared against each other. The following conditions are available:
  - `SA_PS_TRIG_CONDITION_RISING` (`0x00`) The trigger source switches to the high state when the data source value passes *Value0* from below. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further rising edges (single shot).

  - `SA_PS_TRIG_CONDITION_FALLING` (`0x01`) The trigger source switches to the high state when the data source value passes *Value0* from above. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further falling edges (single shot).
  - `SA_PS_TRIG_CONDITION_EITHER` (`0x02`) The trigger source switches to the high state when the data source value passes *Value0* either from below or from above. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further edges (single shot).
  - `SA_PS_TRIG_CONDITION_POSITIVE_LEVEL` (`0x03`) The trigger source is in the high state when the data source value is above *Value0* and in the low state when the data source value is below *Value0*.
  - `SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL` (`0x04`) The trigger source is in the high state when the data source value is below *Value0* and in the low state when the data source value is above *Value0*.
  - `SA_PS_TRIG_CONDITION_POSITIVE_RANGE` (`0x05`) The trigger source is in the high state when the data source value is above *Value0* and below *Value1*. Otherwise it is in the low state.
  - `SA_PS_TRIG_CONDITION_NEGATIVE_RANGE` (`0x06`) The trigger source is in the high state when the data source value is below *Value0* or above *Value1*. Otherwise it is in the low state.

- **Value0** This field defines the threshold of the data source value for the conditions *Rising*, *Falling*, *Either*, *Positive Level*, *Negative Level*, *Positive Range* and *Negative Range*.
- **Value1** This field defines the threshold of the data source value for the condition *Positive Range* and *Negative Range*.

Data Source Increment (`0x03`)

This event generates positive pulses (a state change from low to high and back again) on the trigger output whenever a certain increment of a data source value is detected.

NOTICE
Not all data sources may be used to trigger this event. Only position, ADC and counter data sources may be used.

Register field usage:

- **Index0** This field is used as the channel index of the target data source.
- **Index1** This field is used as the data source index within the selected channel.
- **Condition** This field defines how the selected data source value and the configured threshold(s) are compared against each other. The following conditions are available:

- `SA_PS_TRIG_CONDITION_RISING` (`0x00`) A pulse is generated when the selected data source value passes *Value0* from below. When this happens *Value0* is incremented by *Value1*.
- `SA_PS_TRIG_CONDITION_FALLING` (`0x01`) A pulse is generated when the selected data source value passes *Value0* from above. When this happens *Value0* is decremented by *Value1*.

- **Value0** This field defines the threshold of the data source value. Note that this value is implicitly updated by the trigger system when the described events occur.
- **Value1** This field defines the increment by which *Value0* is updated when the described events occur.

NOTICE
Since the *Value0* register is updated by the device in the way described above, reading the *Value0* register will return the current threshold value rather than the initial start threshold that was written to the register.
Furthermore, when resetting the trigger source, not only the trigger source state is reset to the low state. Furthermore, the *Value0* register is reset to its initial start threshold.

GPIO Trigger (`0x04`)

This event reacts to level changes of an external signal that is fed into the device via a GPIO trigger input pin.

Register field usage:

- **Index0** This field selects the GPIO trigger input pin. 0 selects pin 0, 1 selects pin 1 and so on. The valid range is 0 . . . 8.
- **Condition** This field defines how the trigger source reacts on the selected input pin. The following conditions are available:
  - `SA_PS_TRIG_CONDITION_RISING` (`0x00`) The trigger source switches to the high state when a rising edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further rising edges (single shot).
  - `SA_PS_TRIG_CONDITION_FALLING` (`0x01`) The trigger source switches to the high state when a falling edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further falling edges (single shot).
  - `SA_PS_TRIG_CONDITION_EITHER` (`0x02`) The trigger source switches to the high state when a rising or a falling edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further edges (single shot).

  - `SA_PS_TRIG_CONDITION_POSITIVE_LEVEL` (`0x03`) The level of the selected input pin is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the input pin.
  - `SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL` (`0x04`) The inverted level of the selected input pin is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the input pin.

The other register fields are not used and have no meaning.

External Trigger (`0x05`)

This event is essentially the same as the GPIO Trigger Event with the difference that the input signal is taken from another port. GPIO Trigger inputs are fed into the DSUB plug on the back of the housing while External Trigger inputs are fed into the BNC plugs on the front of the housing (currently only one).

Register field usage:

- **Index0** This field selects the BNC plug. Currently, the only valid index is 0.
- **Condition** This field defines how the trigger source reacts on the selected input pin. The following conditions are available:
  - `SA_PS_TRIG_CONDITION_RISING` (`0x00`) The trigger source switches to the high state when a rising edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further rising edges (single shot).
  - `SA_PS_TRIG_CONDITION_FALLING` (`0x01`) The trigger source switches to the high state when a falling edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further falling edges (single shot).
  - `SA_PS_TRIG_CONDITION_EITHER` (`0x02`) The trigger source switches to the high state when a rising or a falling edge is detected on the selected input. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further edges (single shot).
  - `SA_PS_TRIG_CONDITION_POSITIVE_LEVEL` (`0x03`) The level of the selected input pin is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the input pin.
  - `SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL` (`0x04`) The inverted level of the selected input pin is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the input pin.

The other register fields are not used and have no meaning.

Internal Event (`0x06`)

This event reacts to signals that are generated when internal events occur.

Register field usage:

- **Index0** Selects the event signal. See table below for a list of internal events and their index values.
- **Index1** Sub selector that selects the channel associated with the event.
- **Condition** This field defines how the trigger source reacts on the selected event signal. The following conditions are available:
  - `SA_PS_TRIG_CONDITION_RISING` (`0x00`) The trigger source switches to the high state when a rising edge is detected on the selected event signal. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further rising edges (single shot).
  - `SA_PS_TRIG_CONDITION_FALLING` (`0x01`) The trigger source switches to the high state when a falling edge is detected on the selected event signal. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further falling edges (single shot).
  - `SA_PS_TRIG_CONDITION_EITHER` (`0x02`) The trigger source switches to the high state when a rising or a falling edge is detected on the selected event signal. The trigger source stays in the high state until reset. In particular, in the high state the trigger source does not react to further edges (single shot).
  - `SA_PS_TRIG_CONDITION_POSITIVE_LEVEL` (`0x03`) The level of the selected event signal is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the event signal.
  - `SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL` (`0x04`) The inverted level of the selected event signal is forwarded to the trigger source output. In this case the state of the trigger source depends on the level of the event signal.

The other register fields are not used and have no meaning. Currently, the following event signals are available (used for the *index0* selector):

Event Signal	Value
`SA_PS_INTERNAL_EVENT_BEAM_INTERRUPT`	`0x00`
`SA_PS_INTERNAL_EVENT_OVER_RANGE`	`0x01`

3.2.3 Triggers

Triggers logically combine the outputs of all trigger sources to generate a resulting trigger signal. These may in turn be used as input signals for other system components (or external components) to trigger certain functionalities.

The logical operations that are performed to generate the output signal is controlled by the trigger configuration registers which are shown in the table below.

Register	Data Type
AND Mask	I32
OR Mask	I32
Logic Operation	I32
Output Delay	I32
Output Mode	I32

Each trigger has a set of these registers which are accessed via according properties.

- **AND Mask** This register defines which trigger sources should be combined via a logic AND. Bit 0 of the mask represents trigger source 0, bit 1 trigger source 1 and so on. If all bits of the mask are 0 the logic operation will yield a logic 0.
- **OR Mask** This register defines which trigger sources should be combined via a logic OR. Bit 0 of the mask represents trigger source 0, bit 1 trigger source 1 and so on. If all bits of the mask are 0 the logic operation will yield a logic 0.
- **Logic Operation** This register defines which logic operation should be used after the mask operations (see below).
- **Output Delay** Some trigger events may cause high frequency jitter on the trigger signal. In order to get clean edges the triggers offer an optional debouncing or pulse widening of the resulting signal (see *output mode* below). The value in this register represents a multiple of 10 ns delay. The default value is 0. This causes the signal to be forwarded without a delay in which case the output mode has no effect.
- **Output Mode** In case an output delay is configured, this setting defines the way the resulting trigger signal is output. See section 3.2.4 for more information. If the output delay is 0 then the output mode has no effect.

In a first step a logical AND of all trigger sources that are selected by the AND mask is performed as well as a logical OR of all trigger sources that are selected by the OR mask. After this the results of both operations are combined by a selectable logic operation to form the final (and optionally debounced) trigger output signal (compare figure 3.2).

The following logic operations are available:

- `SA_PS_LOGIC_OP_NONE` (`0x00`) (disabled)
- `SA_PS_LOGIC_OP_OR` (`0x01`)

- `SA_PS_LOGIC_OP_NOR` (`0x02`)
- `SA_PS_LOGIC_OP_AND` (`0x03`)
- `SA_PS_LOGIC_OP_NAND` (`0x04`)
- `SA_PS_LOGIC_OP_XOR` (`0x05`)
- `SA_PS_LOGIC_OP_NXOR` (`0x06`)

The functionality of the trigger logic can be described by the following algorithm:

```
switch (LogicOperation) {
  case AND:
    state = ((ANDMask != 0) && ((TriggerStatus & ANDMask) == ANDMask))
         && (TriggerStatus & ORMask);
    break;
  case OR:
    state = ((ANDMask != 0) && ((TriggerStatus & ANDMask) == ANDMask))
         || (TriggerStatus & ORMask);
    break;
  // ...
}
```

3.2.4 Trigger Output Modes

The trigger signal that results from the logic operation may be modified by the output control component (see figure 3.2). It offers four different output modes:

- **Immediate No Reset** In this mode the output control logic forwards a change of the trigger state immediately and the state is kept stable for the configured delay.
- **Immediate Reset** In this mode the output control logic forwards a change of the trigger state immediately and the state is kept stable for the configured delay. A state change during this delay resets the delay counter.
- **Delayed No Reset** In this mode the output control logic starts a delay counter when the trigger state changes. State changes during this delay are ignored. At the end of the delay the current state is forwarded.
- **Delayed Reset** In this mode the output control logic starts a delay counter when the trigger state changes. State changes during this delay are ignored, but cause the delay counter to be reset. At the end of the delay the current state is forwarded.

Figure 3.3 illustrates the behavior of the ouput control component for the four different modes. The input trace is the result of the configured logic operation of the trigger and the four traces below show the resulting trigger signal for the four output modes.

Note that the output mode only takes effect when a non-zero output delay is configured. When the output delay is set to zero (default) then the resulting trigger signal is identical to the input signal. In figure 3.3 the output delay is set to 10 as an example.

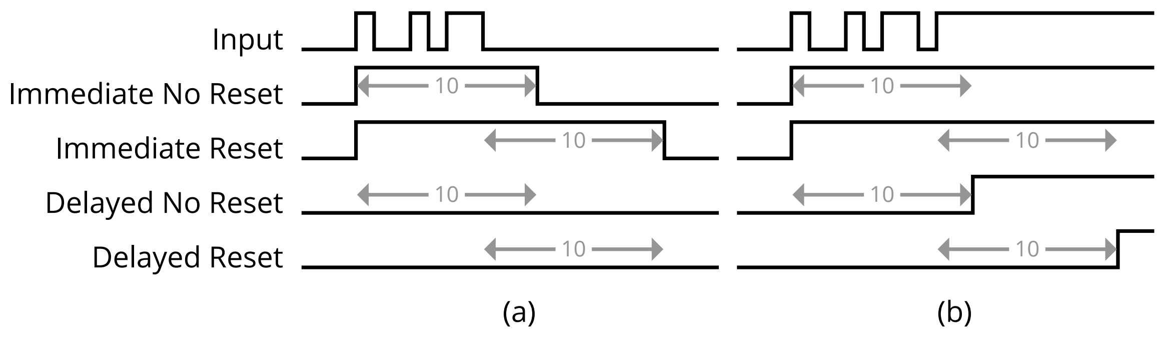


Figure 3.3: Trigger Output Control Behavior (output delay = 10)

The first part of the trace (a) shows some short pulses (e.g. a steady low level with some noise) while the second part of the trace (b) shows a noisy transition of the input signal from low to high (similar to a trace of a mechanical button when pressed).

3.2.5 Basic Trigger vs. Advanced Trigger

The trigger system has two global operation modes: *Basic Trigger* and *Advanced Trigger*. The active mode depends on the device feature "Advanced Trigger System" (see also section 2.7):

Feature State	Operation Mode
Disabled	Basic Trigger
Evaluation or Enabled	Advanced Trigger

In *Basic Trigger* mode the configuration of the trigger system is very limited. There is only one trigger source (index 0) and one trigger (index 0) available. Their configuration properties are set to fixed values and cannot be changed with the exception of the trigger source condition which may be chosen to be rising or falling. The following table summarizes the configuration of the trigger system in basic trigger mode. All other (not listed) properties are set to default values, thus trigger sources and triggers 1 . . . 7 are disabled.

Property	Valid Range
Trigger Source 0 Event	External Trigger
Trigger Source 0 Index 0	0
Trigger Source 0 Condition	Rising, Falling
Trigger 0 AND Mask	0 (0b00000000)
Trigger 0 OR Mask	1 (0b00000001)
Trigger 0 Logic Operation	OR
Trigger 0 Output Delay	0

In *Advanced Trigger* mode the full functionality of the trigger system is available.

Related Properties

Trigger Source Reset, *Trigger Source Event*, *Trigger Source Index 0*, *Trigger Source Index 1*, *Trigger Source Condition*, *Trigger Source Value 0*, *Trigger Source Value 1*, *Trigger Count*, *Trigger AND Mask*, *Trigger OR Mask*, *Trigger Logic Operation*, *Trigger Output Delay*, *Trigger Output Mode*, *Soft Trigger*

3.3 Clock Generators

The PicoScale offers 2 clock generators that may be used to feed external components that are connected via GPIO pins. They may be configured with independent frequencies and phase shifts and may be synchronized with each other. See section 3.7.3 on how to configure a GPIO pin to output clock signals.

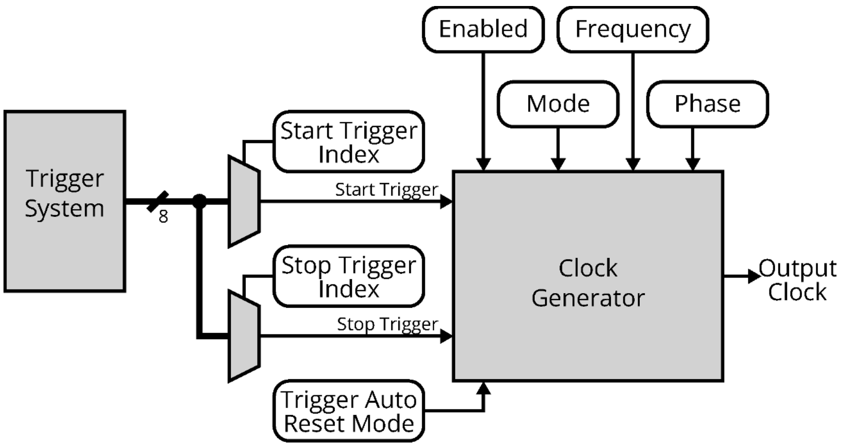


Figure 3.4: Clock Generator Configuration

Figure 3.4 shows the configuration options of a clock generator. In the simplest case it is configured with a frequency and a phase shift and is then enabled via software. However, more sophisticated setups allow the trigger system to control the start and stop behavior of the clock generator.

The behavior of a clock generator may be summarized as follows:

- A clock generator that is *disabled* will output a low level continuously. To avoid glitches it is recommended to configure the clock generator in this state.
- When a clock generator is *enabled* then the *mode* defines its behavior. There are two operation modes: *direct* and *triggered*:
  - In *direct* mode the clock generator will generate the clock signal with the configured frequency and phase as soon as it is enabled. Start and stop triggers are ignored. The clock generator stops when it is disabled again.
  - In *triggered* mode, as soon as the clock generator is enabled, it will listen for the configured start trigger (rising edge) before generating the clock signal with the configured frequency and phase. When the configured stop trigger occurs (rising edge) the clock generator will stop (outputting a low level again).

If the Trigger Auto Reset Mode is enabled then the clock generator will listen for another start trigger after receiving a stop trigger. Otherwise the clock generator can only be restarted by disabling and enabling it again.

Figure 3.5 shows an example trace on how the clock generator works.

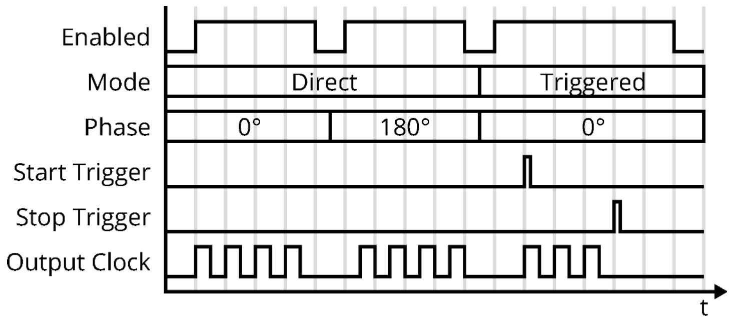


Figure 3.5: Clock Generator Example

Related Properties

Clock Generator Enable Sync, *Clock Generator Disable Sync*, *Clock Generator Enabled*, *Clock Generator Mode*, *Clock Generator Frequency*, *Clock Generator Phase*, *Clock Generator Start Trigger Index*, *Clock Generator Stop Trigger Index*, *Clock Generator Trigger Auto Reset Mode*

3.4 Counters

The PicoScale Laserinterferometer offers 2 counters which may be used as a timer (counting clock cycles) or as an event counter (counting trigger system events). Each counter has the same setup which is depicted in figure 3.6.

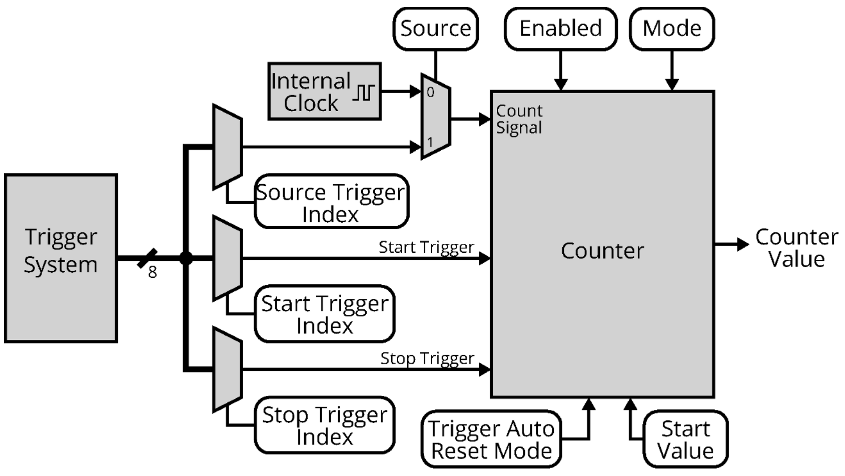


Figure 3.6: Counter

The counter values are available as a data source and may be streamed along with other data sources. Furthermore the counter value may be fed into the trigger system to generate trigger signals. For this the trigger source event must be set to *data source value* or *data source increment* (see section 3.2.2 for more information).

A counter can be disabled or enabled (*Counter Enabled* property) and can be in one of the following three states (*Counter State* property):

- **Stopped** If the counter is disabled it will always be in this state in which it does not count incoming signals. It will also not listen to incoming start or stop triggers.
- **Armed** If the counter is enabled in triggered mode (see below) it will switch to this state in which it does not count incoming signals, but will listen to a start trigger. As soon as the start trigger is received it will switch to the running state.
- **Running** If the counter is in this state then it is either enabled in direct mode or a start trigger was received in triggered mode. If in triggered mode the counter will listen to a stop trigger and then either switch to the stopped state or the armed state, depending on the *Counter Trigger Auto Reset Mode* property (see below).

The following set of properties may be used to configure the counter:

- The *Counter Enabled* property enables or disables the counter.
- The *Counter Mode* property selects whether the counter starts to run immediately after being enabled (*direct mode*) or if it is controlled by a start and a stop trigger (*triggered mode*).
- The *Counter Source* property selects which input signal the counter should count (internal clock cycles or trigger events).
- The *Counter Source Trigger Index* property selects a trigger output from the trigger system. If the *Counter Source* property is set to `SA_PS_COUNTER_SOURCE_TRIGGER` (`0x01`) then the counter counts the rising edges of the selected trigger output.
- The *Counter Start Trigger Index* property selects a trigger output from the trigger system. If the *Counter Mode* is set to `SA_PS_COUNTER_MODE_TRIGGERED` (`0x01`) then the selected trigger is used as a start trigger for the counter (rising edge).
- The *Counter Stop Trigger Index* property selects a trigger output from the trigger system. If the *Counter Mode* is set to `SA_PS_COUNTER_MODE_TRIGGERED` (`0x01`) then the selected trigger is used as a stop trigger for the counter (rising edge).
- The *Counter Trigger Auto Reset Mode* property defines the behavior of the counter in triggered mode after a stop trigger has been received. There are three modes:
  - `SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`) In this mode the counter switches to the stopped state and disables itself after receiving the stop trigger. Therefore, it will not listen to further start triggers.
  - `SA_PS_TRIGGER_AUTO_RESET_ENABLED` (`0x01`) In this mode the counter switches to the armed state after receiving the stop trigger. If another start trigger is received it will continue to count.

  - `SA_PS_TRIGGER_AUTO_RESET_VALUE` (`0x02`) This mode behaves the same as the `SA_PS_TRIGGER_AUTO_RESET_ENABLED` mode. Additionally, the counter value is reset to its start value when a start trigger is received while the counter is in the armed state.

- The *Counter State* property always holds the current state of the counter.
- The *Counter Start Value* property defines the start value of the counter. The current value of the counter is set to the start value if the start value is modified or a start trigger is received in the armed state while the *Counter Trigger Auto Reset Mode* property is set to `SA_PS_TRIGGER_AUTO_RESET_VALUE` (`0x02`).

Related Properties

Counter Enable Sync, *Counter Disable Sync*, *Counter Enabled*, *Counter Mode*, *Counter Source*, *Counter Source Trigger Index*, *Counter Start Trigger Index*, *Counter Stop Trigger Index*, *Counter Trigger Auto Reset Mode*, *Counter State*, *Counter Start Value*

3.5 Calculation System

The PicoScale Laserinterferometer offers 8 calculation systems which may be used to perform various calculations with different values of the device. Each calculation system has the same setup which is depicted in figure 3.7.

The output of a calculation system may be used to:

- Stream the result of the calculation as a data source to synchronize it with other data sources
- Output the calculation result to a GPIO DAC (see section 3.7.2). In this case you must ensure that the result of the calculation lies with a range of [-1;1]. Values greater than 1 will output the maximum DAC value and values smaller than -1 will output the minimum DAC value.

Each calculation system has two operation stages and one shape stage which are described in more detail in the next sections.

Stage 0

In this stage a total of eight operands may be combined by two operators (four operands per operator). Each operand may be configured independently and you may choose which value the operand should have.

The following set of properties may be used to configure the value of the operands and operators:

- The *CS Stage 0 Operand Select* property is the primary selector for the value of the operand. Depending on its setting a secondary selector must also be configured (see below).
- The *CS Stage 0 Const* property defines the direct value of the operand if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_CONST` (`0x00`).

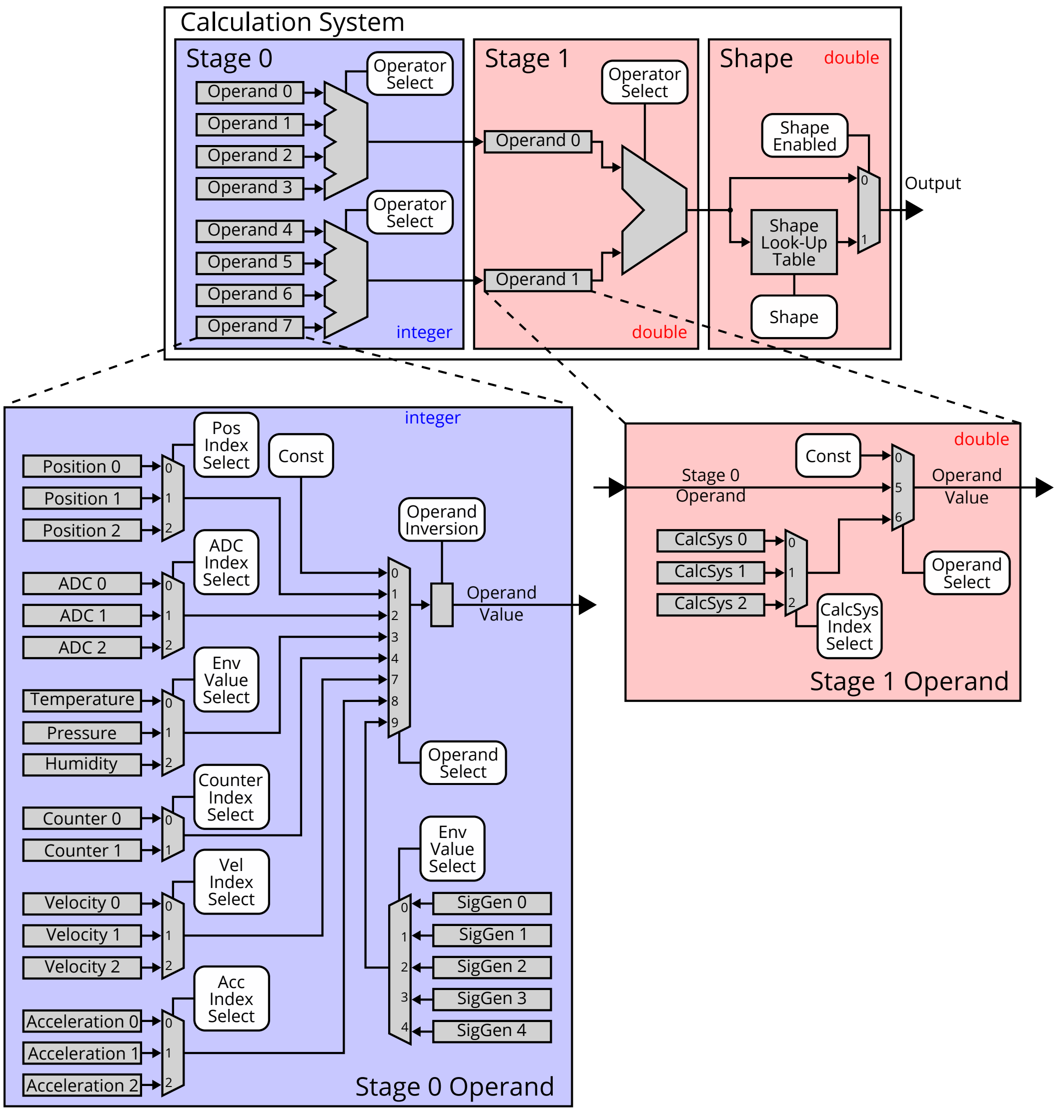


Figure 3.7: Calculation System

- The *CS Stage 0 Position Index Select* property selects a channel if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_POSITION` (`0x01`). The calculated position of the selected channel is taken as the operand value.
- The *CS Stage 0 Velocity Index Select* property selects a channel if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_VELOCITY` (`0x07`). The calculated velocity of the selected channel is taken as the operand value.
- The *CS Stage 0 Acceleration Index Select* property selects a channel if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_ACCELERATION` (`0x08`). The calculated acceleration of the selected channel is taken as the operand value.
- The *CS Stage 0 ADC Index Select* property selects an ADC if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_ADC` (`0x02`). The sample value of this ADC is taken as the operand value.

- The *CS Stage 0 Signal Generator Index Select* property selects a Signal Generator if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_SIG_GEN` (`0x09`). The output value of this Signal Generator is taken as the operand value.
- The *CS Stage 0 Environmental Value Select* property selects an environmental value if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_ENV_VALUE` (`0x03`).
- The *CS Stage 0 Counter Index Select* property selects a counter if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_COUNTER` (`0x04`).
- The *CS Stage 0 Operand Inversion* property may be enabled or disabled to invert the value of the selected operand.
- The *CS Stage 0 Operator Select* property selects which operator is used to combine four operands each. The operator with index 0 combines operands 0 . . . 3 and the operator with index 1 combines operands 4 . . . 7.

Stage 1

In this stage two operands are combined by one operator. The results from the stage 0 operations may be chosen as operands for this stage (see below).

The following set of properties may be used to configure the value of the operands and the operator:

- The *CS Stage 1 Operand Select* property is the primary selector for the value of the operand. Depending on its setting a secondary selector must also be configured (see below).
- The *CS Stage 1 Const* property defines the direct value of the operand if the *CS Stage 1 Operand Select* property is set to `SA_PS_OPERAND_CONST` (`0x00`).
- The *CS Stage 1 Calculation System Index Select* property selects a calculation system if the *CS Stage 0 Operand Select* property is set to `SA_PS_OPERAND_CALC_SYS` (`0x06`). The result of this calculation system is taken as the operand value.
- The *CS Stage 1 Operator Select* property selects which operator is used to combine both operands.

Shape

The shape stage is an optional stage that may be used to modify the result of stage 1 via a look-up table. By default the shape stage is disabled. In this case the final output of the calculation system equals the result of stage 1.

If the shape stage is enabled then the result of stage 1 is interpreted as an index and used to select an entry from the look-up table. The look-up table has a fixed size of 4096 values (index range 0 . . . 4095). Fractional index values are used to interpolate between the look-up table values of the neighboring integer index values. If the calculated index value exceeds the valid range then it is capped. This is illustrated in figure 3.8. It shows a non-linear look-up table. The shape is capped when the stage 1 result continues beyond the index range.

The following set of properties may be used to configure the shape stage of the calculation system:

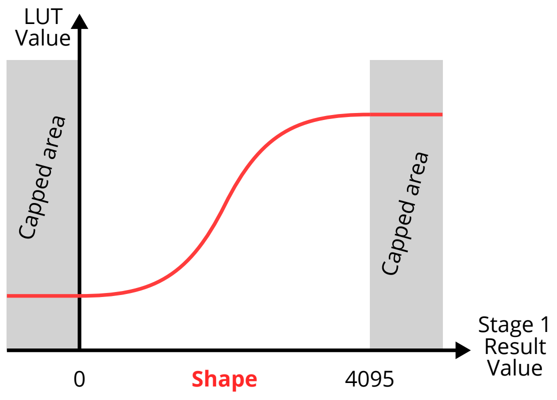


Figure 3.8: Look-up Table Mechanism of the Shape Stage

- The *CS Shape Enabled* property enables or disables the shape stage.
- The *CS Shape* property selects one of several custom shapes as content for the look-up table.

Custom shapes may be uploaded to the device using the `SA_SI_WriteDataObject` function. Please refer to appendix B.3 for information on the format of these data objects. After uploading the custom shape it must be selected by setting the *CS Shape* property (see there).

NOTICE
The calculation system is only available if the device feature "Calculation System" is enabled (see also section 2.7 "Device Features").

Related Properties

CS Stage 0 Operand Select, *CS Stage 0 Const*, *CS Stage 0 Position Index Select*, *CS Stage 0 ADC Index Select*, *CS Stage 0 Environmental Value Select*, *CS Stage 0 Counter Index Select*, *CS Stage 0 Operand Inversion*, *CS Stage 0 Operator Select*, *CS Stage 1 Operand Select*, *CS Stage 1 Const*, *CS Stage 1 Calculation System Index Select*, *CS Stage 1 Operator Select*, *CS Shape Enabled*, *CS Shape*

3.6 Signal Generators

The PicoScale offers 5 Signal Generators that may be used to generate analog signals on one or more of the DACs of the device (see section 3.7.2 on how to configure a DAC to output the signal of a signal generator).

Figure 3.9 shows the configuration options of a signal generator.

A signal generator may be configured to output one of the available standard signal shapes or a custom signal shape (see section 3.6.2).

The following properties may be used to configure the signal generators:

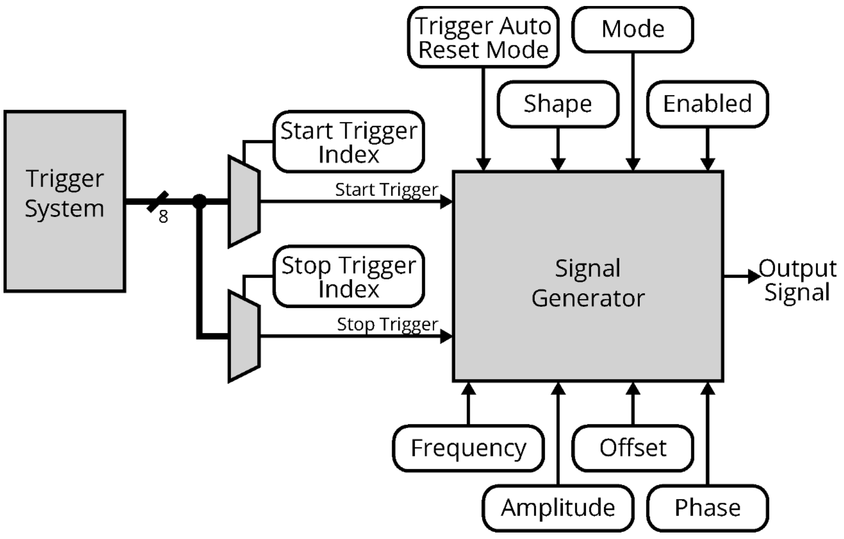
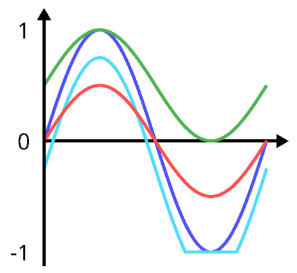


Figure 3.9: Signal Generator Configuration

- The *SigGen Enabled* property enables or disables the signal generator. A disabled signal generator will output a zero value continuously.
- The *SigGen Shape* property selects which shape the signal generator should output (either one of the standard shapes or a custom shape).
- The *SigGen Frequency* property defines the frequency with which the selected signal should be output.
- The *SigGen Amplitude* property defines the amplitude with which the selected signal should be output.
- The *SigGen Offset* property defines the offset with which the selected signal should be output.
- The *SigGen Phase* property defines the phase with which the selected signal should be output. This can be useful when synchronizing several signal generators (see *SigGen Enable Sync* property).
- The *SigGen Mode* property selects whether the signal generator starts to run immediately after being enabled (*direct mode*) or if it is controlled by a start and a stop trigger (*triggered mode*).
- The *SigGen Start Trigger Index* property selects a trigger output from the trigger system. If the *SigGen Mode* is set to `SA_PS_SIG_GEN_MODE_TRIGGERED` (`0x01`) then the selected trigger is used as a start trigger for the signal generator (rising edge).
- The *SigGen Stop Trigger Index* property selects a trigger output from the trigger system. If the *SigGen Mode* is set to `SA_PS_SIG_GEN_MODE_TRIGGERED` (`0x01`) then the selected trigger is used as a stop trigger for the signal generator (rising edge).
- The *SigGen Trigger Auto Reset Mode* property defines the behavior of the signal generator in triggered mode after a stop trigger has been received. There are two modes:

  - `SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`) In this mode the signal generator switches to the stopped state and disables itself after receiving the stop trigger. Therefore, it will not listen to further start triggers.
  - `SA_PS_TRIGGER_AUTO_RESET_ENABLED` (`0x01`) In this mode the signal generator switches to the armed state after receiving the stop trigger. If another start trigger is received the signal generation will continue.
- The *SigGen State* property always holds the current state of the signal generator.

The amplitude and offset parameters are given as 64-bit floating point values in the range of $[0 \ldots 1]$ (amplitude) and $[-1 \ldots 1]$ (offset) respectively. Figure 3.10 gives some examples on how these parameters affect the signal shape. Note that signals are capped rather than wrapped around as indicated by the cyan signal in the figure.



	Amplitude	Offset
Blue	1,0	0,0
Cyan	1,0	-0,25
Red	0,5	0,0
Green	0,5	0,5

Figure 3.10: Shape Parameters: Three sine shaped signals and their amplitude and offset parameters.

The resolution with which the signal is output depends on the DAC that is used to generate the signal. Therefore, the resolution may only be configured implicitly.

NOTICE
The signal generators are only available if the device feature "Signal Generators" is enabled (see also section 2.7 "Device Features").

3.6.1 Standard Shapes

Figure 3.11 illustrates the standard signal shapes that are available. To select one of these shapes the *SigGen Shape* property must be set to one of `SA_PS_SIG_GEN_SHAPE_SQUARE` (`0x01`), `SA_PS_SIG_GEN_SHAPE_SIN` (`0x02`), `SA_PS_SIG_GEN_SHAPE_SAWTOOTH_P` (`0x03`) or `SA_PS_SIG_GEN_SHAPE_SAWTOOTH_N` (`0x04`).

3.6.2 Custom Shapes

Custom signal shapes may be uploaded to the device using the `SA_SI_WriteDataObject` function. Please refer to appendix B.2 for information on the format of these data objects. After uploading the custom shape it must be selected by setting the *SigGen Shape* property to one of

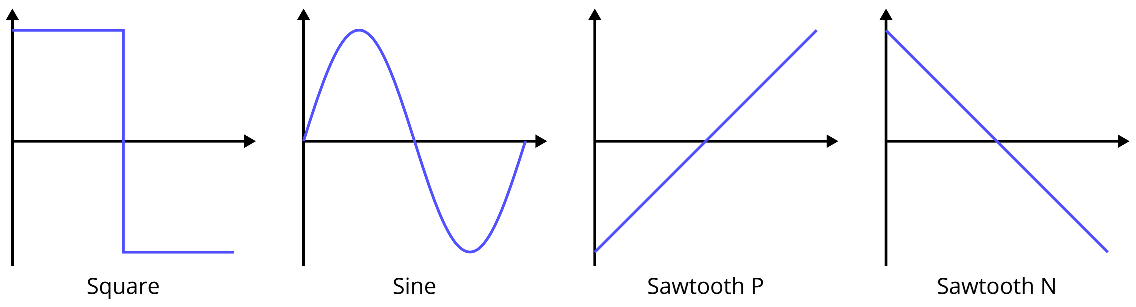


Figure 3.11: Standard Signal Generator Shapes

`SA_PS_SIG_GEN_SHAPE_CUSTOM0` (`0x80`), `SA_PS_SIG_GEN_SHAPE_CUSTOM1` (`0x81`), `SA_PS_SIG_GEN_SHAPE_CUSTOM2` (`0x82`), `SA_PS_SIG_GEN_SHAPE_CUSTOM3` (`0x83`) or `SA_PS_SIG_GEN_SHAPE_CUSTOM4` (`0x84`).

Related Properties

SigGen Enable Sync, *SigGen Disable Sync*, *SigGen Enabled*, *SigGen Frequency*, *SigGen Amplitude*, *SigGen Offset*, *SigGen Phase*, *SigGen Shape*, *SigGen Mode*, *SigGen Start Trigger Index*, *SigGen Stop Trigger Index*, *SigGen Trigger Auto Reset Mode*, *SigGen State*

3.7 General Purpose Input/Output

The general purpose input/output (GPIO) pins may be used to communicate and synchronize with other external devices.

3.7.1 Analog Inputs

The PicoScale offers 3 Analog Digital Converters (ADCs). See the data sheet for a detailed specification of these components.

The ADCs may be streamed as data sources (see also table C.1) or they may be fed into the Calculation System (see section 3.5).

3.7.2 Analog Outputs

The PicoScale offers 5 Digital Analog Converters (DACs). See the data sheet for a detailed specification of these components.

Figure 3.12 shows the setup of one DAC (the setup is the same for all DACs).

The following properties may be used to configure what is output on the DAC:

- The *GPIO DAC Source* property defines where the output value for the DAC is taken from.

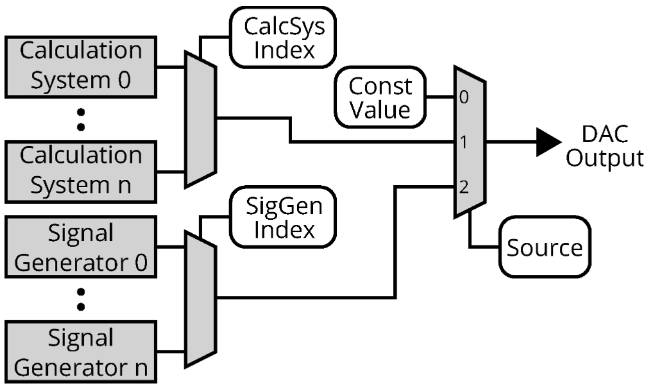


Figure 3.12: GPIO DAC Configuration

- The *GPIO DAC Const Value* property lets you define a constant output value for the DAC if the *GPIO DAC Source* property is set to `SA_PS_GPIO_DAC_SOURCE_CONST` (`0x00`).
- The *GPIO DAC Calculation System Index* property selects which output of the calculation system is forwarded to the DAC output if the *GPIO DAC Source* property is configured with the value `SA_PS_GPIO_DAC_SOURCE_CALCULATION_SYSTEM` (`0x01`). See section 3.5 for a description of the calculation system.

  Note: Output values of the calculation system that are greater than 1.0 are capped to 1.0 and values smaller than -1.0 are capped to -1.0 (no wrapping).
- The *GPIO DAC Signal Generator Index* property selects a signal generator if the *GPIO DAC Source* property is set to `SA_PS_GPIO_DAC_SOURCE_SIGNAL_GENERATOR` (`0x02`). The output of the selected signal generator is forwarded to the DAC. See section 3.6 for a description of the signal generators.

Related Properties

GPIO DAC Count, *GPIO DAC Bit Width*, *GPIO DAC Sample Rate*, *GPIO DAC Source*, *GPIO DAC Const Value*, *GPIO DAC Calculation System Index*, *GPIO DAC Signal Generator Index*

3.7.3 Digital Input/Output

The PicoScale offers 9 digital I/O pins that may be configured independently from each other. See the data sheet for a detailed specification of the I/O pins. Figure 3.13 shows the setup of one digital I/O port (the setup is the same for all I/O ports).

The following set of properties may be used to configure the I/O pins (the high index of the property key selects the I/O pin that is to be configured):

- The *GPIO Digital Direction* property defines whether the pin is used as an input or an output pin.
- The *GPIO Digital Source* property defines where the output value is taken from if the pin is configured as an output pin.

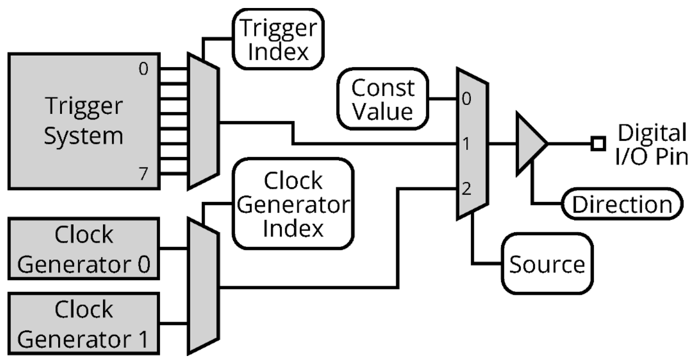


Figure 3.13: Digital I/O Configuration

- The *GPIO Digital Const Value* property lets you define a constant level of the output pin if the *GPIO Digital Source* property is set to `SA_PS_GPIO_DIG_SOURCE_CONST` (`0x00`).
- The *GPIO Digital Trigger Index* property selects the output of the trigger system that is used to define the output of the I/O pin if the *GPIO Digital Source* property is configured with the value `SA_PS_GPIO_DIG_SOURCE_TRIGGER` (`0x01`). See section 3.2 for more information on the trigger system.
- The *GPIO Digital Clock Generator Index* property selects a clock generator. If the *GPIO Digital Source* property is set to `SA_PS_GPIO_DIG_SOURCE_CLOCK_GENERATOR` (`0x02`) then the output of the selected clock generator is used to define the output of the I/O pin. See section 3.3 for more information on the clock generators.

Related Properties

GPIO Digital Direction, *GPIO Digital Source*, *GPIO Digital Const Value*, *GPIO Digital Trigger Index*, *GPIO Digital Clock Generator Index*

3.8 Digital Differential Interface

The Digital Differential Interface (DDI) is an output interface that outputs data via two output lines (each differential, four pins in total). Please refer to the user manual for information on the electrical specifications and setup.

The output lines may be used to feed external systems to supply them e.g. with the currently measured position. The format in which the data is output is configurable.

The following set of properties may be used to configure the interface:

- The *DDI Enabled* property enables or disables the digital differential interface.
- The *DDI Mode* property defines the format with which the data is output (quadrature or serial data).
- The *DDI Source* property selects the data source that should be output. Currently, only a position or the output of a calculation system may be selected.

The different modes of the DDI are described in the next sections.

3.8.1 Quadrature Output Mode

In this mode data may be output as standard quadrature signals (A and B).

The following properties may be used to configure the quadrature outputs:

- The *DDI Quad Step Size* property defines how many LSBs a data value must change before a quadrature pulse is output.
- The *DDI Quad Frequency* property defines the maximum frequency at which quadrature output signals are produced.

Note that the *DDI Quad Available Frequencies* property may be queried to get a list of valid values for the *DDI Quad Frequency* property.

Figure 3.14 shows an example of a position trace and the resulting quadrature output signals.

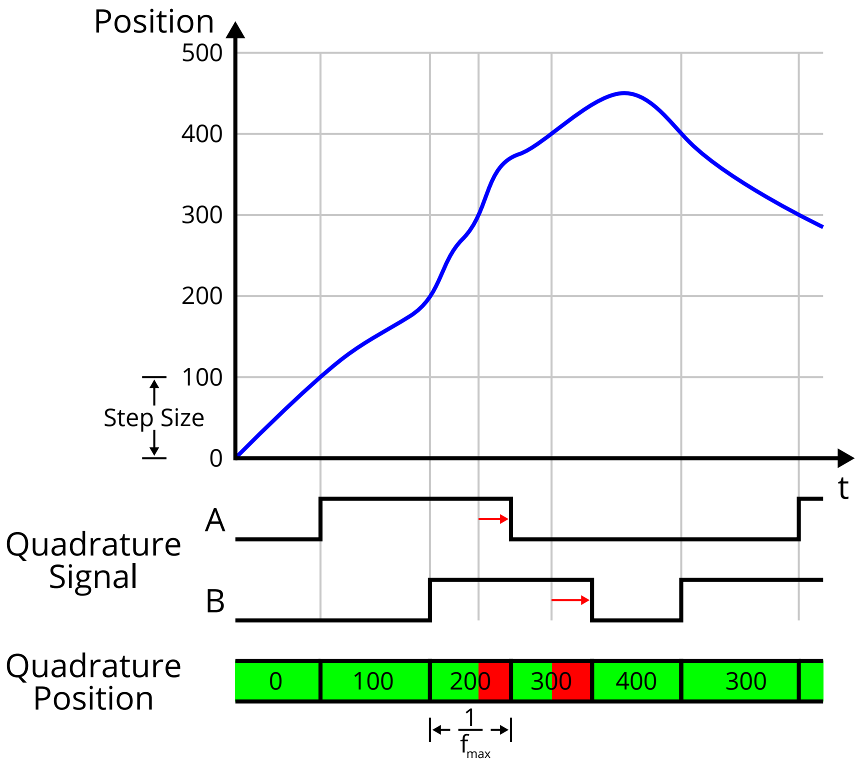


Figure 3.14: Quadrature Output Example

The red areas of the quadrature position mark temporary invalid position data. This happens if the quadrature step size and frequency (f_{max}) configuration is inappropriate for the occurring velocities. In this case the falling edges of A and B come too late. This may be avoided by increasing the step size or the frequency (or both).

3.8.2 Serial Data Output Mode

In this mode data is output in form of serial data (clock and data). The interface supports four different transmit modes, adjustable in clock polarity and clock phase. The relationship between the clock and data signals in the different modes can be seen in figure 3.15. You may insert an adjustable amount of pause cycles between data words in order to recognize a complete data cycle. It is possible to limit the desired range of values with the bit width and resolution shift. You may also set the polarity of the data line when it is idle. The interface sends all data with the **MSB first**.

The following properties may be used to configure the serial output:

- The *DDI Serial Clock Mode* property sets the clock polarity and phase (see figure 3.15).
- The *DDI Serial Clock Frequency* property defines the clock frequency in Hz.
- The *DDI Serial Clock Pause Delay* property defines the number of idle clock cycles before transmitting the next data word.
- The *DDI Serial Data Bit Width* property sets the desired word width of the data.
- The *DDI Serial Data Resolution Shift* property determines the least significant bit of the data word to shift the resolution.
- The *DDI Serial Data Idle Polarity* property sets the idle polarity on the serial data line.

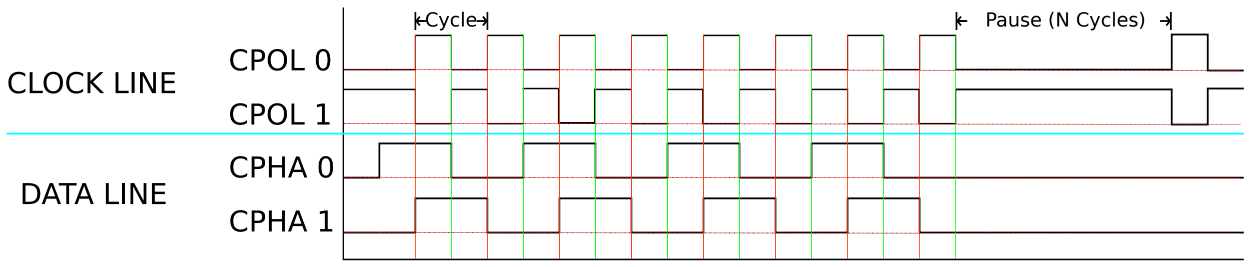


Figure 3.15: Serial Data transfer with 3x pause delay and 0xAA data

Releated Properties

DDI Enabled, *DDI Mode*, *DDI Source*, *DDI Quad Step Size*, *DDI Quad Frequency*, *DDI Quad Available Frequencies*, *DDI Serial Clock Mode*, *DDI Serial Clock Frequency*, *DDI Serial Clock Pause Delay*, *DDI Serial Data Bit Width*, *DDI Serial Data Resolution Shift*, *DDI Serial Data Idle Polarity*

4 *USING THE API*

This chapter gives an overview of how to use the API in order to communicate with the device and retrieve sensor data.

Generally, there are two ways of getting sensor data from the device: *polling* and *streaming*. While polling is very simple to use (but comparably slow), streaming is much more efficient, allowing data rates of up to 10MHz (although it requires more programming overhead). Furthermore, streaming is the only method to get synchronized samples from several data sources.

The typical approach when **polling** is:

1. Connect to the device.
2. Poll data source value(s).
3. Disconnect from the device.

The typical approach when **streaming** is:

1. Connect to the device.
2. Configure the data stream.
3. Start the data stream.
4. Receive sensor data.
5. Stop the data stream.
6. Disconnect from the device.

These steps are explained in more detail in the next sections.

4.1 Connecting and Disconnecting

Before being able to communicate with a device it must be initialized with a call to `SA_SI_Open`. This function connects to the device specified in the locator parameter and returns a handle to the device, if the call was successful. The returned device handle must be saved within the application and passed as a parameter to the other API functions. Once the connection is established you can use the other functions to interact with the connected device.

A device that has been acquired by an application cannot be acquired by a second application at the same time. You must close the connection to the device by calling `SA_SI_Close` before it is free to be used by other applications. Not closing a device will cause a resource leak.

If you have threads blocking on `SA_SI_WaitForEvent` you may unblock them for a clean shutdown by calling `SA_SI_Cancel`. The `SA_SI_WaitForEvent` function will then return with the error code `SA_SI_CANCELED_ERROR` (`0xf010`).

4.1.1 Locators for Device Identification

Devices are identified with *locator* strings, similar to URLs used to locate web pages. Typical locators are:

```
usb:ix:0
network:192.168.1.100:55555
```

The first locator identifies a device with index 0 connected over USB. The second one identifies a device that is connected to the network.

You may use the `SA_SI_EnumerateDevices` function to scan the network and USB ports for devices and return a list with the locator strings.

USB Device Locator Syntax

Devices with USB interface can be addressed with the following locator syntax:

```
usb:sn:<serial>
```

where `<serial>` is the device serial which is printed on the housing of the device.

Devices with USB interface may alternatively be addressed with the following locator syntax:

```
usb:ix:<n>
```

where the number <`n`> selects the *nth* device in the list of all currently attached devices with a USB interface. The drawback of identifying a device with this method is that the number and the order of connected devices may change between sessions, so the index `n` may not always refer to the same device. It is only safe to do this if you have exactly one device connected to the PC. It is recommended to use the `usb:sn:...` format for USB devices.

Network Device Locator Syntax

Devices with a network interface are addressed with the following locator syntax:

```
network:<ip>:<port>
```

where <ip> is an IPv4 address which consists of four integer numbers between 0 and 255 separated by a dot. <port> is an integer number.

For example, the locator `network:192.168.1.200:55555` addresses a device with the IP address 192.168.1.200 and TCP port 55555.

NOTICE
Data transmission bandwidth and latencies over networks can vary much more than over e.g. USB. A program should not rely on low transmission latencies.

4.2 Configuring the Device

To modify properties (see section 2.1) you may use the Get/SetProperty functions which come in different flavors, one for each basic data type. Since each property has a defined data type you must call the correct function in order to access the property. For example, to read the `SA_SI_DEVICE_ID_PROP` property you must use the `SA_SI_GetProperty_s` function. Using the `SA_SI_GetProperty_i32` function on this property will yield an error.

To generate the property key that is required by all of the get/set functions you may use the `SA_SI_EPK` helper function, which transforms a property code and two index values into a property key.

The properties that configure the data stream may be reset to default values with a simple call to `SA_SI_ResetStreamingConfiguration`.

4.3 Polling Data Source Values

The `SA_SI_GetValue_i64` and `SA_SI_GetValue_f64` functions may be used to retrieve data source values directly.

Note that polling data source values is rather slow compared to data streaming. Additionally, the values returned by the polling method are not synchronized. The device simply returns the value that the data source has at the time of the call.

4.4 Starting and Stopping the Data Stream

Once the data stream is configured via the various properties (see section section 2.4.1) you may activate the data stream. You do this by selecting the desired streaming mode with the *Streaming Mode* property and then setting the *Streaming Active* property to `SA_SI_ENABLED` (`0x01`). When data recording is finished set the *Streaming Active* property back to `SA_SI_DISABLED` (`0x00`) to disable the data stream.

The different streaming modes are explained in section 2.4.3.

4.5 Receiving Stream Data

The library uses *data buffers* to pass data received from the device to the user application. As soon as the data stream is enabled the slave starts sending data frames. The library receives them, possibly decompresses them and places them into the next free data buffer. Once the current buffer is full an event is generated notifying that the buffer may be read out.

The typical approach to receive data from the device is:

1. Call `SA_SI_WaitForEvent` and wait for the `SA_SI_STREAMBUFFER_READY_EVENT`.
2. Call `SA_SI_AcquireBuffer` to acquire the data buffer with the buffer ID that came with the event. When a buffer is acquired it is guaranteed that the content of the buffer is not changed.
3. Read the data from the data buffer.
4. Call `SA_SI_ReleaseBuffer` to release the buffer. After this the buffer will be re-used by the library to fill it with more incoming data.
5. Repeat from step 1 until the stream end flag of the buffer is set (see section 4.5.1).

4.5.1 Data Buffers

A data buffer is a memory structure that resides in the library and holds the data of one or more data frames. It has the following structure:

```
typedef struct
{
    uint32_t bufferId;
    uint32_t frameIndex;
    uint32_t numberOfSources;
    uint32_t numberOfFrames;
    uint32_t flags;
} SA_SI_DataBufferInfo;

typedef struct
{
    SA_SI_DataBufferInfo info;
    uint8_t **data;
} SA_SI_DataBuffer;
```

- `SA_SI_DataBuffer.data` is a pointer to an array of pointers that point to the received data. The array size is either equal to the number of data sources that are enabled for streaming or 1, depending on the interleaving mode (see section 4.5.3).
- `SA_SI_DataBuffer.info.bufferID` holds the ID of the buffer.
- `SA_SI_DataBuffer.info.frameIndex` holds the frame index of the first frame in the buffer.

- `SA_SI_DataBuffer.info.numberOfSources` holds the number of data sources that the buffer contains which is equal to the array size of the data pointer in non-interleaved mode (see section 4.5.3).
- `SA_SI_DataBuffer.info.numberOfFrames` holds the number of data frames that the buffer contains. In most cases this value is equal to the stream buffer aggregation value (see section 4.5.2), but may be smaller when the data stream ends.
- `SA_SI_DataBuffer.info.flags` is a bit mask that holds several binary meta information about the data buffer (see table 4.1).

Table 4.1: Stream Buffer Flags

Bit	Meaning
0	**Stream Begin** Is set in the first data buffer that is received after the data stream has been activated.
1	**Stream End** Is set if the device has terminated the data stream and no further data is generated.
2	**Stream Suspend** If the stream was suspended by a stop trigger then this flag indicates the temporary end of the data stream.
3	reserved
4	**Interleaved** When set indicates that the contents of data buffer is in interleaved format.
5	**Stream Incomplete** When set indicates packet loss during data transmission.
6	**Reference Lost** When set indicates that data from some sources is possibly wrong.
7 - 31	reserved

4.5.2 Buffer Aggregation

The `SA_SI_STREAMBUFFER_AGGREGATION_PROP` property defines the number of frames that a data buffer holds. This is similar to the data frame aggregation described in section 2.3.2 with the difference that the `SA_SI_FRAME_AGGREGATION_PROP` property affects the communication between the device and the library while the `SA_SI_STREAMBUFFER_AGGREGATION_PROP` property affects the communication between the library and the user application.

4.5.3 Buffer Interleaving

When receiving data from the device the API may pass the data to the user application in one of two ways:

- **Non-Interleaved** In this mode each data-source's data is stored in a dedicated memory and the array of pointers has a size equal to the number of data sources that are enabled for streaming.

- **Interleaved** In this mode the frames are stored in one consecutive memory block.

Whether to use the interleaved or non-interleaved mode depends on how you further wish to process the data. For example, when simply storing the data to disc for later processing it might be more efficient to use the interleaved mode. When having graphical UI elements, each displaying the data of one data source it might be more convenient to use the non-interleaved mode.

The *Stream Buffers Interleaved* property may be used to configure the mode. Figure 4.1 shows an example of how the data is organized in the data buffer with both the interleaved and non-interleaved mode.

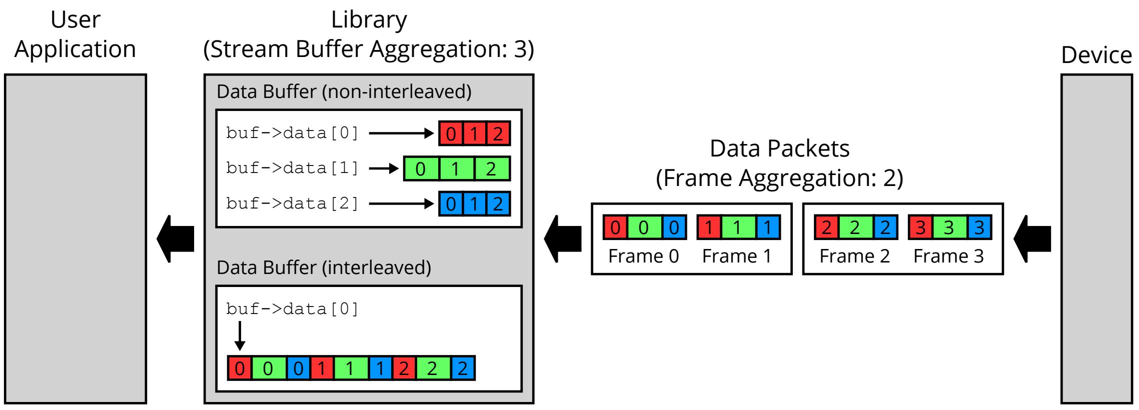


Figure 4.1: Interleaved vs. Non-Interleaved Data Buffers

There are three data sources enabled for streaming (indicated as red, green and blue data chunks).

5 *FUNCTION REFERENCE*

5.1 Summary

Table 5.1 – Function Summary

Function Name	Short Description	Page
`SA_SI_EnumerateDevices`	Returns a list of locator strings of connected devices.	59
`SA_SI_EnumerateInterfaces`	Returns a list of communication interfaces.	60
`SA_SI_Open`	Opens a connection to a device.	61
`SA_SI_Close`	Closes a connection to a device.	62
`SA_SI_SetProperty_i32`	Sets a 32 bit integer property.	63
`SA_SI_GetProperty_i32`	Gets a 32 bit integer property.	64
`SA_SI_SetProperty_i64`	Sets a 64 bit integer property.	66
`SA_SI_GetProperty_i64`	Gets a 64 bit integer property.	67
`SA_SI_SetProperty_f32`	Sets a float property.	68
`SA_SI_GetProperty_f32`	Gets a float property.	69
`SA_SI_SetProperty_f64`	Sets a double property.	70
`SA_SI_GetProperty_f64`	Gets a double property.	71
`SA_SI_SetProperty_s`	Sets a string property.	72
`SA_SI_GetProperty_s`	Gets a string property.	73
`SA_SI_GetDataSource`	Returns the channel and data source index for component selectors, component ID, component index and data source type.	75
`SA_SI_GetFullVersionString`	Returns the library version.	76
`SA_SI_EPK`	Encodes a property key.	77
`SA_SI_Cancel`	Aborts waiting functions.	78
`SA_SI_GetValue_i64`	Returns a data source value as integer.	79
`SA_SI_GetValue_f64`	Returns a data source value as double.	80

Continued on next page

Table 5.1 – *Continued from previous page*

Function Name	Short Description	Page
`SA_SI_ResetStreamingConfiguration`	Sets all stream parameters to default values.	81
`SA_SI_SetStreamingMode` (deprecated)	Enables or disables the data stream.	82
`SA_SI_WaitForEvent`	Blocks until an event occurs.	83
`SA_SI_AcquireBuffer`	Acquires a stream buffer for reading data.	85
`SA_SI_ReleaseBuffer`	Releases an acquired buffer.	86
`SA_SI_GetBufferInfo`	Returns an info struct for a buffer.	87
`SA_SI_CopyBuffer`	Copies buffer contents.	88
`SA_SI_GetFrameElementIndex`	Returns info about the frame structure.	89
`SA_SI_GetFrameElementDataSource`	Returns info about the frame structure.	91
`SA_SI_ReadDataObject`	Reads a data object from the device.	93
`SA_SI_WriteDataObject`	Writes a data object to the device.	95

5.2 Detailed Function Description

5.2.1 SA_SI_EnumerateDevices

Interface:

```
unsigned int SA_CTL_EnumerateDevices(
        const char *options,
        char *deviceList,
        size_t *deviceListLen
);
```

Description:

This function writes a list of locator strings of devices that are connected to the PC into *deviceList*. It may be used to list USB devices as well as network devices. *options* contains a list of configuration options for the find procedure. The caller must pass a pointer to a `char` buffer in *deviceList* and set *deviceListLen* to the size of the buffer. After the call the function has written a list of device locators into *deviceList* and the number of characters written into *deviceListLen*. If the supplied buffer is too small to contain the generated list, then the function will return with `SA_SI_QUERYBUFFER_SIZE_ERROR`. In this case the buffer will contain no valid content but *deviceListLen* contains the required buffer size (in characters).

Parameters:

- *options* (const char *), **input**: Options for the find procedure. **Currently unused.**
- *deviceList* (char *), **output**: Pointer to a buffer which holds the device locators after the function has returned. The locator strings are separated by a newline character.
- *deviceListLen* (size_t *), **inout**: Specifies the size (in bytes) of *deviceList* before the function call. After the function call it holds the number of characters written to *deviceList*.

Example:

```
char buffer[4096];
size_t bufferSize = sizeof(buffer);
SASIResult result = SA_SI_EnumerateDevices("",buffer,&bufferSize);
if (result == SA_SI_OK) {
  // deviceList holds the locator strings, separated by '\n'
  // bufferSize holds the number of characters written to deviceList
}
```

5.2.2 SA_SI_EnumerateInterfaces

Interface:

```
unsigned int SA_CTL_EnumerateInterfaces(
       const char *options,
       char *outBuffer,
       size_t *ioBufferSize
);
```

Description:

Writes a list of communication interfaces to *outBuffer* in form of a null-terminated string.

Parameters:

- *options* (const char *), **input**: Currently unused.
- *outBuffer* (char *), **output**: Buffer to write the list to.
- *ioBufferSize* (size_t *), **inout**: Size of the target buffer (in bytes). After the call it will hold the size of the written string (excluding the null-terminator).

Example:

```
char buffer[4096];
size_t bufferSize = sizeof(buffer);
SASIResult result = SA_SI_EnumerateInterfaces("",buffer,&bufferSize);
if (result == SA_SI_OK) {
  // deviceList holds the locator strings, separated by '\n'
  // bufferSize holds the number of characters written to deviceList
}
```

5.2.3 SA_SI_Open

Interface:

```
unsigned int SA_SI_Open(
        SA_SA_Handle *handle,
        const char *locator,
        const char *config
);
```

Description:

Establishes a connection to a device for communication.

Parameters:

- *handle* (SA_SI_Handle *), **output**: Handle to the device. Used to be passed to following function calls.
- *locator* (const char *), **input**: Specifies the device to be opened (see section 4.1.1).
- *config* (const char *), **input**: Currently unused.

Example:

```
SA_SI_Handle handle;
unsigned int result = SA_SI_Open(&handle,"usb:sn:PSC-00000016","");
if (result == SA_SI_OK) {
  // success
}
```

See also:

SA_SI_Close

5.2.4 SA_SI_Close

Interface:

```
unsigned int SA_SI_Close(
        SA_SA_Handle handle
);
```

Description:

Closes a previously established connection to a device.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.

Example:

```
SA_SI_Handle handle;
unsigned int result = SA_SI_Open(&handle,"usb:sn:PSC-00000016","");
if (result == SA_SI_OK) {
  // success
  result = SA_SI_Close(handle);
}
```

See also:

SA_SI_Open

5.2.5 SA_SI_SetProperty_i32

Interface:

```
unsigned int SA_SI_SetProperty_i32(
      SA_SA_Handle handle,
      SA_SI_PropertyKey key,
      int32_t value
);
```

Description:

This function writes a 32-bit integer property value to the device.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (int32_t), **input**: Value that should be written.

Example:

```
// set frame rate to 2.5 MHz
unsigned int result;
result = SA_SI_SetProperty_i32(handle,
  SA_SI_EPK(SA_SI_FRAME_RATE_PROP,0,0),
  2500000
);
if (result == SA_SI_OK) {
  // success
}
```

See also:

`SA_SI_GetProperty_i32`, `SA_SI_SetProperty_i64`, `SA_SI_SetProperty_f32`, `SA_SI_SetProperty_f64`, `SA_SI_SetProperty_s`

5.2.6 SA_SI_GetProperty_i32

Interface:

```
unsigned int SA_SI_GetProperty_i32(
       SA_SI_Handle handle,
       SA_SI_PropertyKey key,
       int32_t *value,
       unsigned int *ioArraySize
);
```

Description:

This function retrieves a 32-bit integer property value (array) from the device. The caller must supply a pointer to a buffer where the result should be written to as well as a size information which indicates how many values may be written into the buffer. The function then writes the resulting value(s) into the buffer and sets the size information to the number of values written.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (int32_t *), **output**: Pointer to a buffer where the result should be written to.
- *ioArraySize* (unsigned int *), **inout**: Pointer to a size value that must contain the size of the value buffer when the function is called (in number of values, not bytes). On function return it contains the number of values written to the buffer. A `nullptr` is allowed which implicitly indicates an array size of 1.

Example:

```
// get single value (number of channels)
int32_t numChannels;
unsigned int result;
result = SA_SI_GetProperty_i32(
  handle,
  0x00110000,
  &numChannels,
  nullptr
);
if (result == SA_SI_OK) {
  // numChannels holds the number of channels
}
// get value array (available compression modes of channel 0,
// data source 0)
```

```
int32_t modes[16];
unsigned int numModes = 16;
result = SA_SI_GetProperty_i32(handle, 0x20030000, modes, &numModes);
if (result == SA_SI_OK) {
  // numModes holds the number of compression modes
  // modes holds the compression modes
}
```

See also:

SA_SI_SetProperty_i32, SA_SI_GetProperty_i64,
SA_SI_GetProperty_f32, SA_SI_GetProperty_f64, SA_SI_GetProperty_s

5.2.7 SA_SI_SetProperty_i64

Interface:

```
unsigned int SA_SI_SetProperty_i64(
      SA_SA_Handle handle,
      SA_SI_PropertyKey key,
      int64_t value
);
```

Description:

This function writes a 64-bit integer property value to the device.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (int64_t), **input**: Value that should be written.

Example:

See example on page 63.

See also:

`SA_SI_GetProperty_i64`, `SA_SI_SetProperty_i32`, `SA_SI_SetProperty_f32`, `SA_SI_SetProperty_f64`, `SA_SI_SetProperty_s`

5.2.8 SA_SI_GetProperty_i64

Interface:

```
unsigned int SA_SI_GetProperty_i64(
      SA_SI_Handle handle,
      SA_SI_PropertyKey key,
      int64_t *value,
      unsigned int *ioArraySize
);
```

Description:

This function retrieves a 64-bit integer property value (array) from the device. The caller must supply a pointer to a buffer where the result should be written to as well as a size information which indicates how many values may be written into the buffer. The function then writes the resulting value(s) into the buffer and sets the size information to the number of values written.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (int64_t *), **output**: Pointer to a buffer where the result should be written to.
- *ioArraySize* (unsigned int *), **inout**: Pointer to a size value that must contain the size of the value buffer when the function is called (in number of values, not bytes). On function return it contains the number of values written to the buffer. A `nullptr` is allowed which implicitly indicates an array size of 1.

Example:

See example on page 64.

See also:

`SA_SI_SetProperty_i64`, `SA_SI_GetProperty_i32`, `SA_SI_GetProperty_f32`, `SA_SI_GetProperty_f64`, `SA_SI_GetProperty_s`

5.2.9 SA_SI_SetProperty_f32

Interface:

```
unsigned int SA_SI_SetProperty_f32(
      SA_SA_Handle handle,
      SA_SI_PropertyKey key,
      float value
);
```

Description:

This function writes a `float` property value to the device.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (float), **input**: Value that should be written.

Example:

See example on page 63.

See also:

`SA_SI_GetProperty_f32`, `SA_SI_SetProperty_i32`, `SA_SI_SetProperty_i64`, `SA_SI_SetProperty_f64`, `SA_SI_SetProperty_s`

5.2.10 SA_SI_GetProperty_f32

Interface:

```
unsigned int SA_SI_GetProperty_f32(
      SA_SI_Handle handle,
      SA_SI_PropertyKey key,
      float *value,
      unsigned int *ioArraySize
);
```

Description:

This function retrieves a `float` property value (array) from the device. The caller must supply a pointer to a buffer where the result should be written to as well as a size information which indicates how many values may be written into the buffer. The function then writes the resulting value(s) into the buffer and sets the size information to the number of values written.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (float *), **output**: Pointer to a buffer where the result should be written to.
- *ioArraySize* (unsigned int *), **inout**: Pointer to a size value that must contain the size of the value buffer when the function is called (in number of values, not bytes). On function return it contains the number of values written to the buffer. A `nullptr` is allowed which implicitly indicates an array size of 1.

Example:

See example on page 64.

See also:

`SA_SI_SetProperty_f32`, `SA_SI_GetProperty_i32`,
`SA_SI_GetProperty_i64`, `SA_SI_GetProperty_f64`,
`SA_SI_GetProperty_s`

5.2.11 SA_SI_SetProperty_f64

Interface:

```
unsigned int SA_SI_SetProperty_f64(
      SA_SA_Handle handle,
      SA_SI_PropertyKey key,
      double value
);
```

Description:

This function writes a `double` property value to the device.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (double), **input**: Value that should be written.

Example:

See example on page 63.

See also:

`SA_SI_GetProperty_f64`, `SA_SI_SetProperty_i32`, `SA_SI_SetProperty_i64`, `SA_SI_SetProperty_f32`, `SA_SI_SetProperty_s`

5.2.12 SA_SI_GetProperty_f64

Interface:

```
unsigned int SA_SI_GetProperty_f64(
        SA_SI_Handle handle,
        SA_SI_PropertyKey key,
        double *value,
        unsigned int *ioArraySize
);
```

Description:

This function retrieves a `double` property value (array) from the device. The caller must supply a pointer to a buffer where the result should be written to as well as a size information which indicates how many values may be written into the buffer. The function then writes the resulting value(s) into the buffer and sets the size information to the number of values written.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (double *), **output**: Pointer to a buffer where the result should be written to.
- *ioArraySize* (unsigned int *), **inout**: Pointer to a size value that must contain the size of the value buffer when the function is called (in number of values, not bytes). On function return it contains the number of values written to the buffer. A `nullptr` is allowed which implicitly indicates an array size of 1.

Example:

See example on page 64.

See also:

`SA_SI_SetProperty_f64`, `SA_SI_GetProperty_i32`,
`SA_SI_GetProperty_i64`, `SA_SI_GetProperty_f32`,
`SA_SI_GetProperty_s`

5.2.13 SA_SI_SetProperty_s

Interface:

```
unsigned int SA_SI_SetProperty_s(
      SA_SA_Handle handle,
      SA_SI_PropertyKey key,
      const char *value
);
```

Description:

This function writes a string property value to the device. Note that the length of strings may never exceed `63` bytes plus a null terminator.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (const char *), **input**: String that should be written (C-string).

Example:

```
unsigned int result;
// set device name to "Foo"
result = SA_SI_SetProperty_s(
  handle,
  SA_SI_EPK(SA_SI_DEVICE_NAME_PROP,0,0),
  "Foo"
);
if (result == SA_SI_OK) {
  // success
}
```

See also:

`SA_SI_GetProperty_s`, `SA_SI_SetProperty_i32`, `SA_SI_SetProperty_i64`, `SA_SI_SetProperty_f32`, `SA_SI_SetProperty_f64`

5.2.14 SA_SI_GetProperty_s

Interface:

```
unsigned int SA_SI_GetProperty_s(
      SA_SI_Handle handle,
      SA_SI_PropertyKey key,
      char *value,
      unsigned int *ioArraySize
);
```

Description:

This function retrieves a string property value (array) from the device. The caller must supply a pointer to a buffer where the result should be written to as well as a size information which indicates how many bytes may be written into the buffer. The function then writes the resulting string(s) into the buffer and sets the size information to the number of characters written.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *key* (SA_SI_PropertyKey), **input**: The key that identifies the property (see section 2.1.1).
- *value* (char *), **output**: Pointer to a buffer where the result should be written to.
- *ioArraySize* (unsigned int *), **inout**: Pointer to a size value that must contain the size of the value buffer (in bytes) when the function is called. On function return it contains the number of characters written to the buffer.

Example:

```
char deviceName[128];
unsigned int len = sizeof(deviceName);
unsigned int result;
// read device name
result = SA_SI_GetProperty_s(
  handle,
  SA_SI_EPK(SA_SI_DEVICE_NAME_PROP,0,0),
  deviceName,
  &len
);
if (result == SA_SI_OK) {
  // deviceName holds the name of the device
  // len holds the length of the name
}
```

See also:

`SA_SI_SetProperty_s`, `SA_SI_GetProperty_i32`, `SA_SI_GetProperty_i64`, `SA_SI_GetProperty_f32`, `SA_SI_GetProperty_f64`

5.2.15 SA_SI_GetDataSource

Interface:

```
unsigned int SA_SI_GetDataSource(
      unsigned int componentId,
      unsigned int componentIndex,
      unsigned int dsourceType,
      unsigned int *channel,
      unsigned int *dsourceIndex
);
```

Description:

Translates a data source "description" into a tuple of channel index and data source index.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *componentId* (unsigned int), **input**: Component Id of the selected data source.
- *componentIndex* (unsigned int), **input**: Component index of the selected data source.
- *dsourceType* (unsigned int), **input**: Data Type of the selected data source.
- *channel* (unsigned int *), **output**: Channel index of the found data source.
- *dsourceIndex* (unsigned int *), **output**: Data source index of the found data source.

Example:

```
unsigned int result, chIdx, dsIdx;

// query channel and data source index of the counter1 data source
result = SA_SI_GetDataSource(
  handle,
  SA_PS_COUNTER_CID,
  1,
  SA_SI_COUNTER_DSOURCE
  &chIdx,
  &dsIdx
);
if (result == SA_SI_OK) {
  // chIdx holds the channel index
  // dsIdx holds the data source index
}
```

5.2.16 SA_SI_GetFullVersionString

Interface:

```
const char* SA_SI_GetFullVersionString();
```

Description:

This function returns the version of the library as a null terminated string.

Parameters:

None

Example:

```
cout << "version is: " << SA_SI_GetFullVersionString() << endl;
```

5.2.17 SA_SI_EPK

Interface:

```
SA_SI_PropertyKey SA_SI_EPK(
       uint16_t propertyCode,
       uint8_t index0,
       uint8_t index1
);
```

Description:

This function (Encode Property Key) is used in conjunction with various functions to get or set property values. The property which is to be read or written is selected via the *key* parameter. The `SA_SI_EPK` helper function may be used to encode a property key consisting of a property code and two index parameter. (See section 2.1.1.) Valid property codes are listed in `SmarActSIConstants.h` and `SmarActSIConstants_PS.h`. Not all property codes require index values to be specified.

Parameters:

- *propertyCode* (uint16_t), **input**: The 16-bit property code.
- *index0* (uint8_t), **input**: An 8-bit index value. Its meaning depends on the property code. An unused index should be set to `0x00`.
- *index1* (uint8_t), **input**: An 8-bit index value. Its meaning depends on the property code. An unused index should be set to `0x00`.

Example:

```
unsigned int result;
int32_t value;
unsigned int size = 1;
result = SA_SI_GetProperty_i32(
  handle,
  SA_SI_EPK(SA_SI_NUMBER_OF_CHANNELS_PROP,0x00,0x00),
  &value,
  &size
);
if (result == SA_SI_OK) {
  // size has value 1
  // value holds the number of channels the device has
}
```

5.2.18 SA_SI_Cancel

Interface:

```
unsigned int SA_SI_Cancel(
      SA_SI_Handle handle
);
```

Description:

A call to this function unblocks waiting functions, such as `SA_SI_WaitForEvent`.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.

Example:

See example on page 83.

See also:

`SA_SI_WaitForEvent`

5.2.19 SA_SI_GetValue_i64

Interface:

```
unsigned int SA_SI_GetValue_i64(
       SA_SI_Handle handle,
       unsigned int channelIndex,
       unsigned int dataSourceIndex,
       int64_t *value
);
```

Description:

This function retrieves the current value of a data source as a 64-bit integer value.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *channelIndex* (unsigned int), **input**: Index of the addressed channel.
- *dataSourceIndex* (unsigned int), **input**: Index of the addressed data source of the selected channel.
- *value* (int64_t *), **output**: Current value of the addressed data source.

Example:

```
int64_t value;
unsigned int result;
result = SA_SI_GetValue_i64(handle,0,0,&value);
if (result == SA_SI_OK) {
  // value holds the current value of data source 0 of channel 0
}
```

See also:

SA_SI_GetValue_f64

5.2.20 SA_SI_GetValue_f64

Interface:

```
unsigned int SA_SI_GetValue_f64(
      SA_SI_Handle handle,
      unsigned int channel,
      unsigned int dataSourceIndex,
      double *value
);
```

Description:

This function retrieves the current value of a data source as a double value.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *channel* (unsigned int), **input**: Index of the addressed channel.
- *dataSourceIndex* (unsigned int), **input**: Index of the addressed data source of the selected channel.
- *value* (double *), **output**: Current value of the addressed data source.

Example:

```
double value;
unsigned int result;
result = SA_SI_GetValue_f64(handle,0,0,&value);
if (result == SA_SI_OK) {
  // value holds the current value of data source 0 of channel 0
}
```

See also:

SA_SI_GetValue_i64

5.2.21 SA_SI_ResetStreamingConfiguration

Interface:

```
unsigned int SA_SI_ResetStreamingConfiguration(
        SA_SI_Handle handle
);
```

Description:

Turns off the data stream and resets all stream parameters to default values. These include:

- All data sources are disabled for streaming.
- All compression modes are set to their default compression mode (usually no compression).
- All resolution shifts are set to their default values.
- The frame rate is set to 1.
- The frame aggregation is set to 1.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.

Example:

```
unsigned int result;
result = SA_SI_ResetStreamingConfiguration(handle);
if (result == SA_SI_OK) {
  // the stream configuration was successfully reset
}
```

5.2.22 SA_SI_SetStreamingMode

Interface:

```
unsigned int SA_SI_SetStreamingMode(
       SA_SI_Handle handle,
       unsigned int mode
);
```

Description:

This function enables or disables the data stream.

Note that this function is deprecated. Use properties *Streaming Mode* and *Streaming Active* instead.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *mode* (unsigned int), **input**: The new streaming mode. See table A.7 for a list of valid modes and also section 2.4.3.

Example:

```
unsigned int result;
result = SA_SI_SetStreamingMode(handle,SA_SI_NO_STREAMING);
if (result == SA_SI_OK) {
  // the data stream was disabled
}
```

See also:

`SA_SI_ResetStreamingConfiguration`

5.2.23 SA_SI_WaitForEvent

Interface:

```
unsigned int SA_SI_WaitForEvent(
      SA_SI_Handle handle,
      SA_SI_Event *event,
      unsigned int timeout
);
```

Description:

This function blocks until the device reports an event. The function returns when:

- An event has occurred within the given timeout. In this case the return value of the function will be `SA_SI_OK` (`0x0000`) and the output parameter *event* will hold the event that occurred.
- No event occurred within the given timeout. In this case the return value of the function will be `SA_SI_TIMEOUT_ERROR` (`0x0004`) and the *event* parameter is undefined.
- The call is canceled with `SA_SI_Cancel`. In this case the return value of the function will be `SA_SI_CANCELED_ERROR` (`0xf010`) and the *event* parameter is undefined.

Note that you must subscribe to events in order to be able to receive them. See section 2.5 for more information.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *event* (SA_SI_Event *), **output**: The event that occurred. Only valid if the return value of the function call is `SA_SI_OK`.
- *timeout* (unsigned int), **input**: Maximum time to wait for an event to occur. The timeout is given in milli seconds. The special value `SA_SI_TIMEOUT_INFINITE` (`0xffffffff`) is also valid.

Example:

```
// thread 1:
SA_SI_Event event;
unsigned int result;
result = SA_SI_WaitForEvent(handle,&event,SA_SI_TIMEOUT_INFINITE);
if (result == SA_SI_CANCELED_ERROR) {
  // SA_SI_WaitForEvent was canceled before an event occurred
}
```

```
// thread 2:
// wake up waiting thread 1
unsigned int result = SA_SI_Cancel(handle);
```

See also:

`SA_SI_Cancel`

5.2.24 SA_SI_AcquireBuffer

Interface:

```
unsigned int SA_SI_AcquireBuffer(
       SA_SI_Handle handle,
       unsigned int bufferID,
       const SA_SI_DataBuffer **buffer
);
```

Description:

Acquires a stream buffer for reading data. As long as the buffer is acquired, the caller may read data from the buffer. The buffer must be ready before it can be acquired, which will be signaled by a `SA_SI_STREAMBUFFER_READY_EVENT` (`0xf000`). See section 4.5.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *bufferID* (unsigned int), **input**: ID of the buffer to be acquired.
- *buffer* (const SA_SI_DataBuffer **), **output**: Pointer to a data buffer pointer.

See also:

`SA_SI_ReleaseBuffer`

5.2.25 SA_SI_ReleaseBuffer

Interface:

```
unsigned int SA_SI_ReleaseBuffer(
        SA_SI_Handle handle,
        unsigned int bufferID,
);
```

Description:

Releases an acquired buffer. Accessing the buffer data after release will cause an undefined behavior.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *bufferID* (unsigned int), **input**: ID of the buffer to be released.

See also:

`SA_SI_AcquireBuffer`

5.2.26 SA_SI_GetBufferInfo

Interface:

```
unsigned int SA_SI_GetBufferInfo(
      SA_SI_Handle handle,
      unsigned int bufferID,
      uint32_t *seqCounter,
      uint32_t *numOfSources,
      uint32_t *numOfFrames,
      uint32_t *flags
);
```

Description:

Returns information about the buffer referred to by *bufferId*. The buffer must have been acquired by the caller with the `SA_SI_AcquireBuffer` function.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *bufferID* (unsigned int), **input**: ID of the buffer to get the info from.
- *seqCounter* (uint32_t *), **output**: Frame index of the first frame in the buffer.
- *numOfSources* (uint32_t *), **output**: The number of data sources that the frames contain.
- *numOfFrames* (uint32_t *), **output**: The number of frames that the buffer contains.
- *flags* (uint32_t *), **output**: A bit mask that holds several binary meta information about the buffer (see table 4.1).

5.2.27 SA_SI_CopyBuffer

Interface:

```
unsigned int SA_SI_CopyBuffer(
        SA_SI_Handle handle,
        unsigned int bufferID,
        unsigned int elementIndex,
        uint8_t *dest,
        unsigned int destSize
);
```

Description:

Copies the content of the buffer referred to by *bufferID* to a destination array. The buffer must have been acquired by the caller with the `SA_SI_AcquireBuffer` function. The *elementIndex* parameter selects which data to copy:

- In *non-interleaved* mode the function copies the value of one data source. In this case *elementIndex* specifies the element in the frames to be copied.
- In *interleaved* mode the function copies the whole buffer content. In this case *elementIndex* must be set to `SA_SI_ALL_STREAMBUFFER_ELEMENTS` to copy all data sources.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *bufferID* (unsigned int), **input**: ID of the buffer to copy.
- *elementIndex* (unsigned int), **input**: Sepcifies the element(s) to copy.
- *dest* (uin8_t *), **output**: Buffer to copy the data to.
- *destSize* (unsigned int), **input**: Size of the destination buffer in bytes. If *destSize* is smaller than the data to copy, then the function returns with `SA_SI_QUERYBUFFER_SIZE_ERROR` and no data is copied to *dest*.

5.2.28 SA_SI_GetFrameElementIndex

Interface:

```
unsigned int SA_SI_GetFrameElementIndex(
      SA_SI_Handle handle,
      unsigned int channelIndex,
      unsigned int dsourceIndex,
      unsigned int *elementIndex
);
```

Description:

When receiving stream frames this function gives information about the structure of a stream data buffer. It returns the index of the data source specified by *channelIndex* and *dsourceIndex* within the buffer. If the data source is not enabled for streaming then the function returns an error. See section 2.2 for more information on the order of data sources in a stream configuration.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *channelIndex* (unsigned int), **input**: Index of the addressed channel.
- *dsourceIndex* (unsigned int), **input**: Index of the addressed data source of the selected channel.
- *elementIndex* (unsigned int *), **output**: Index of the specified data source within the stream buffer. Note that this value is undefined if the function returns with an error.

Example:

```
// disable all data sources for streaming
SA_SI_ResetStreamingConfiguration(handle);
// enable data sources (2,0), (1,0) and (1,1) for streaming
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,2,0),SA_SI_ENABLED)
);
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,1,0),SA_SI_ENABLED)
);
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,1,1),SA_SI_ENABLED)
);
// check which index channel 1, data source 1 has within the
// stream buffer
unsigned int index;
```

```
SA_SI_GetFrameElementIndex(handle,1,1,&index);
// since the data source order is (1,0), (1,1), (2,0) index holds a 1.
```

See also:

`SA_SI_GetFrameElementDataSource`

5.2.29 SA_SI_GetFrameElementDataSource

Interface:

```
unsigned int SA_SI_GetFrameElementDataSource(
      SA_SI_Handle handle,
      unsigned int elementIndex,
      unsigned int *channel,
      unsigned int *dsourceIndex
);
```

Description:

When receiving stream frames this function gives information about the structure of a stream data buffer. It returns the channel index and data source index of the nth element within the buffer. See section 2.2 for more information on the order of data sources in a stream configuration.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *elementIndex* (unsigned int), **input**: Index of the element in the data buffer. The index is zero-based. For example, if four data sources are enabled for streaming then the valid range is 0 . . . 3.
- *channel* (unsigned int *), **output**: Channel index of the data source.
- *dsourceIndex* (unsigned int *), **output**: Index of the data source in the channel.

Example:

```
// disable all data sources for streaming
SA_SI_ResetStreamingConfiguration(handle);
// enable data sources (2,0), (1,0) and (1,1) for streaming
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,2,0),SA_SI_ENABLED)
);
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,1,0),SA_SI_ENABLED)
);
SA_SI_SetProperty_i32(
  handle,SA_SI_EPK(SA_SI_STREAMING_ENABLED,1,1),SA_SI_ENABLED)
);
// check which data source the third element of the
// stream buffer represents
unsigned int channel, dataSource;
SA_SI_GetFrameElementDataSource(handle,2,&channel,&dataSource);
```

```
// since the data source order is (1,0), (1,1), (2,0) channel holds
// a 2 and dataSource holds a 0.
```

See also:

`SA_SI_GetFrameElementIndex`

5.2.30 SA_SI_ReadDataObject

Interface:

```
unsigned int SA_SI_ReadDataObject(
       SA_SI_Handle handle,
       const char *objectName,
       char *buffer,
       size_t *ioBufferSize
);
```

Description:

This function reads the contents of a data object from the device. The data object to read is identified by its name. The caller must supply a pointer to a buffer and the maximum size of this buffer. On success the function will write the data object that was read from the decice to the buffer and update the size information.

If the buffer should be too small for the data object then the function will return with an error (`SA_SI_QUERYBUFFER_SIZE_ERROR`, `0xf006`). In this case the *ioBufferSize* parameter will hold the minimum required size for the buffer.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *objectName* (const char *), **input**: Name of the object as a null-terminated string. The maximum allowed string length is 63 characters plus a terminating null.
- *buffer* (char *), **output**: Pointer to a buffer where the content of the data object should be written to after reading it from the device.
- *ioBufferSize* (size_t *), **inout**: Pointer to a size value that must contain the size of the buffer (in number of bytes) when the function is called. On function return it contains the number of bytes that were written to the buffer.

Example:

```
unsigned int result;
char buffer[32768];
size_t bufferSize = 32768;
result = SA_SI_ReadDataObject(
  handle,"CalcSysCustomShape0",buffer,&bufferSize
);
if (result == SA_SI_OK) {
  // bufferSize holds the data object size (in bytes) and the
  // buffer contains the object data
}
```

See also:

`SA_SI_WriteDataObject`

5.2.31 SA_SI_WriteDataObject

Interface:

```
unsigned int SA_SI_WriteDataObject(
       SA_SI_Handle handle,
       const char *objectName,
       const char *data,
       size_t dataSize
);
```

Description:

This function writes a data object to the device. The data object to write is identified by its name. The caller must supply a pointer to a buffer that holds the object data and the size of the data.

Parameters:

- *handle* (SA_SI_Handle), **input**: The handle of the addressed device.
- *objectName* (const char *), **input**:
- *data* (const char *), **intput**:
- *dataSize* (size_t), **input**:

Example:

```
unsigned int result;
char buffer[65536]
// ... fill buffer with data ...
// a custom shape for the calculation system has a size
// of 16 kB
size_t bufferSize = 16384;
result = SA_SI_WriteDataObject(
  handle,"CalcSysCustomShape0",buffer,bufferSize
);
```

See also:

`SA_SI_ReadDataObject`

6 *PROPERTY REFERENCE*

6.1 Property Summary

Table 6.1 – Property Summary

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Basic Device Properties					
Device Type	`0x0002`	`0x00`	`0x00`	I32	R
Device Serial Number	`0x0003`	`0x00`	`0x00`	String	R
Device Name	`0x0004`	`0x00`	`0x00`	String	RW
Number of Firmware Versions	`0x0005`	`0x00`	`0x00`	I32	R
Firmware Version	`0x0006`	Index	`0x00`	I32 [4]	R
Firmware Version String	`0x0007`	Index	`0x00`	String	R
Number of Channels	`0x0011`	`0x00`	`0x00`	I32	R
Maximum Frame Rate	`0x0020`	`0x00`	`0x00`	I32	R
Frame Rate	`0x0021`	`0x00`	`0x00`	I32	RW
Maximum Frame Aggregation	`0x0022`	`0x00`	`0x00`	I32	R
Frame Aggregation	`0x0023`	`0x00`	`0x00`	I32	RW
Precise Frame Rate	`0x0025`	`0x00`	`0x00`	F64	R
Event Notification Enabled	`0x0030`	Index		I32	RW
Streaming Active	`0x0040`	`0x00`	`0x00`	I32	RW
Streaming Mode	`0x0041`	`0x00`	`0x00`	I32	RW
Basic Channel Properties					
Number of Data Sources	`0x1001`	Index	`0x00`	I32	R
Channel Name	`0x1002`	Index	`0x00`	String	RW
Basic Data Source Properties					
Data Source Type	`0x2001`	Index	Index	I32	R
Data Type	`0x2002`	Index	Index	I32	R
Available Compression Modes	`0x2003`	Index	Index	I32 [ ]	R
Compression Mode	`0x2004`	Index	Index	I32	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Streaming Enabled	`0x2005`	Index	Index	I32	RW
Base Unit	`0x2006`	Index	Index	I32	R
Base Resolution	`0x2007`	Index	Index	I32	R
Resolution Shift	`0x2008`	Index	Index	I32	RW
Data Source Name	`0x2009`	Index	Index	String	R
Is Streamable	`0x200a`	Index	Index	I32	R
Component ID	`0x200b`	Index	Index	I32	R
Component Index	`0x200c`	Index	Index	I32	R
PicoScale Device Properties					
Full Access Connection	`0x8000`	`0x00`	`0x00`	I32	RW
LVDS LS Connected	`0x8012`	`0x00`	`0x00`	I32	RW
Pilot Laser Active	`0x8020`	`0x00`	`0x00`	I32	RW
System Stable	`0x8030`	`0x00`	`0x00`	I32	R
Working Distance Min	`0x8040`	`0x00`	`0x00`	F64	RW
Working Distance Max	`0x8041`	`0x00`	`0x00`	F64	RW
Working Distance Activate	`0x8042`	`0x00`	`0x00`	I32	W
Working Distance Shrink Mode	`0x8043`	`0x00`	`0x00`	I32	RW
Network Current IP	`0x8052`	`0x00`	`0x00`	String	R
Network Config Activate	`0x8060`	`0x00`	`0x00`	I32	W
Network Config DHCP	`0x8061`	`0x00`	`0x00`	I32	RW
Network Config IP	`0x8062`	`0x00`	`0x00`	String	RW
Network Config Gateway	`0x8063`	`0x00`	`0x00`	String	RW
Network Config Netmask	`0x8064`	`0x00`	`0x00`	String	RW
Network Config Nameserver	`0x8065`	`0x00`	`0x00`	String	RW
Network Config Domain Name	`0x8066`	`0x00`	`0x00`	String	RW
Network MAC	`0x8070`	`0x00`	`0x00`	String	R
Head Type Category Count	`0x8081`	`0x00`	`0x00`	I32	R
Head Type Count	`0x8082`	Index	`0x00`	I32	R
Head Type Category Name	`0x8083`	Index	`0x00`	String	R
Head Type Name	`0x8084`	Index	Index	String	R
Fiber Length Head	`0x8090`	`0x00`	`0x00`	F64	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Fiber Length Extension	0x8091	0x00	0x00	F64	RW
Position all Channels	0x80a0	0x00	0x00	I64	RW
Configuration Save	0x80c0	Index	0x00	I32	W
Configuration Load	0x80c1	Index	0x00	I32	W
Configuration Name	0x80c2	Index	0x00	String	RW
Configuration Count	0x80c3	0x00	0x00	I32	R
Filter CutOff Frequency	0x80de	0x00	0x00	F64	RW
Filter Rate	0x80df	0x00	0x00	F64	RW
Bootloader Version	0x80e0	0x00	0x00	I32 [4]	R
Bootloader Version String	0x80e1	0x00	0x00	String	R
Hardware Version	0x80e2	0x00	0x00	I32 [5]	R
Hardware Version String	0x80e3	0x00	0x00	String	R
Product Version	0x80e4	0x00	0x00	I32 [4]	R
Product Version String	0x80e5	0x00	0x00	String	R
Feature Count	0x80f0	0x00	0x00	I32	R
Feature Name	0x80f1	Index	0x00	String	R
Feature Time	0x80f2	Index	0x00	I32	R
Feature Evaluate	0x80f3	Index	0x00	I32	RW
PicoScale Channel Properties					
Channel Enabled	0x8100	Index	0x00	I32	RW
Is Valid	0x8101	Index	0x00	I32	R
Position	0x8102	Index	0x00	I64	W
Scale Inversion	0x8103	Index	0x00	I32	RW
Dead Path Correction Enabled	0x8110	Index	0x00	I32	RW
Dead Path	0x8111	Index	0x00	I64	RW
Head Type	0x8112	Index	0x00	I32	RW
Beam Interrupt Tolerance	0x8113	Index	0x00	I32	RW
Environmental Sensor Properties					
Env Sensor Operation Mode	0x8200	0x00	0x00	I32	RW
Env Sensor State	0x8201	0x00	0x00	I32	R
Env Sensor Version	0x820e	0x00	0x00	I32 [4]	R
Env Sensor Version String	0x820f	0x00	0x00	String	R

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Env Sensor Temperature	`0x8210`	`0x00`	`0x00`	I32	R
Env Sensor Pressure	`0x8211`	`0x00`	`0x00`	I32	R
Env Sensor Humidity	`0x8212`	`0x00`	`0x00`	I32	R
Env User Temperature	`0x8220`	`0x00`	`0x00`	I32	RW
Env User Pressure	`0x8221`	`0x00`	`0x00`	I32	RW
Env User Humidity	`0x8222`	`0x00`	`0x00`	I32	RW
GPIO Properties					
GPIO DAC Count	`0x8310`	`0x00`	`0x00`	I32	R
GPIO DAC Bit Width	`0x8311`	Index	`0x00`	I32	R
GPIO DAC Sample Rate	`0x8312`	Index	`0x00`	I32	R
GPIO DAC Const Value	`0x8313`	Index	`0x00`	F64	RW
GPIO DAC Source	`0x8318`	Index	`0x00`	I32	RW
GPIO DAC Calculation System Index	`0x8319`	Index	`0x00`	I32	RW
GPIO DAC Signal Generator Index	`0x831a`	Index	`0x00`	I32	RW
GPIO Digital Const Value	`0x8320`	Index	`0x00`	I32	RW
GPIO Digital Direction	`0x8321`	Index	`0x00`	I32	RW
GPIO Digital Source	`0x8322`	Index	`0x00`	I32	RW
GPIO Digital Trigger Index	`0x8323`	Index	`0x00`	I32	RW
GPIO Digital Clock Generator Index	`0x8324`	Index	`0x00`	I32	RW
Trigger System Properties					
Trigger Source Count	`0x8400`	`0x00`	`0x00`	I32	R
Trigger Source Reset	`0x8401`	Index	`0x00`	I32	W
Trigger Source Event	`0x8402`	Index	`0x00`	I32	RW
Trigger Source Index 0	`0x8403`	Index	`0x00`	I32	RW
Trigger Source Index 1	`0x8404`	Index	`0x00`	I32	RW
Trigger Source Condition	`0x8405`	Index	`0x00`	I32	RW
Trigger Source Value 0	`0x8406`	Index	`0x00`	I64	RW
Trigger Source Value 1	`0x8407`	Index	`0x00`	I64	RW
Trigger Count	`0x8410`	`0x00`	`0x00`	I32	R
Trigger AND Mask	`0x8411`	Index	`0x00`	I32	RW
Trigger OR Mask	`0x8412`	Index	`0x00`	I32	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Trigger Logic Operation	0x8413	Index	0x00	I32	RW
Trigger Output Delay	0x8414	Index	0x00	I32	RW
Trigger Output Mode	0x8415	Index	0x00	I32	RW
Soft Trigger	0x8420	Index	0x00	I32	W
Trigger Source State	0x8430	0x00	0x00	I32	R
Trigger State	0x8431	0x00	0x00	I32	R
Digital Differential Interface Properties					
DDI Enabled	0x8500	Index	0x00	I32	RW
DDI Mode	0x8501	Index	0x00	I32	RW
DDI Source	0x8502	Index	0x00	I32	RW
DDI Quad Step Size	0x8510	Index	0x00	I32	RW
DDI Quad Frequency	0x8511	Index	0x00	I32	RW
DDI Quad Available Frequencies	0x8512	0x00	0x00	I32 [ ]	R
DDI Serial Clock Mode	0x8520	Index	0x00	I32	RW
DDI Serial Clock Frequency	0x8521	Index	0x00	I32	RW
DDI Serial Clock Pause Delay	0x8522	Index	0x00	I32	RW
DDI Serial Data Bit Width	0x8523	Index	0x00	I32	RW
DDI Serial Data Resolution Shift	0x8524	Index	0x00	I32	RW
DDI Serial Data Idle Polarity	0x8525	Index	0x00	I32	RW
Clock Generator Properties					
Clock Generator Enable Sync	0x8600	0x00	0x00	I32	W
Clock Generator Disable Sync	0x8601	0x00	0x00	I32	W
Clock Generator Enabled	0x8610	Index	0x00	I32	RW
Clock Generator Mode	0x8611	Index	0x00	I32	RW
Clock Generator Frequency	0x8612	Index	0x00	I32	RW
Clock Generator Phase	0x8613	Index	0x00	I32	RW
Clock Generator Start Trigger Index	0x8614	Index	0x00	I32	RW
Clock Generator Stop Trigger Index	0x8615	Index	0x00	I32	RW
Clock Generator Trigger Auto Reset Mode	0x8616	Index	0x00	I32	RW
Clock Generator State	0x8617	Index	0x00	I32	R
Stream Generator Properties					
SG Clock Source	0x8700	0x00	0x00	I32	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
SG Clock Trigger Index	`0x8701`	`0x00`	`0x00`	I32	RW
SG Trigger Start Index	`0x8710`	`0x00`	`0x00`	I32	RW
SG Trigger Stop Index	`0x8711`	`0x00`	`0x00`	I32	RW
SG Trigger Post Frame Count	`0x8712`	`0x00`	`0x00`	I32	RW
SG Trigger Auto Reset Mode	`0x8713`	`0x00`	`0x00`	I32	RW
Counter Properties					
Counter Enable Sync	`0x8900`	`0x00`	`0x00`	I32	W
Counter Disable Sync	`0x8901`	`0x00`	`0x00`	I32	W
Counter Enabled	`0x8910`	Index	`0x00`	I32	RW
Counter Mode	`0x8911`	Index	`0x00`	I32	RW
Counter Source	`0x8912`	Index	`0x00`	I32	RW
Counter Source Trigger Index	`0x8913`	Index	`0x00`	I32	RW
Counter Start Trigger Index	`0x8914`	Index	`0x00`	I32	RW
Counter Stop Trigger Index	`0x8915`	Index	`0x00`	I32	RW
Counter Trigger Auto Reset Mode	`0x8916`	Index	`0x00`	I32	RW
Counter State	`0x8917`	Index	`0x00`	I32	R
Counter Start Value	`0x8920`	Index	`0x00`	I64	RW
Calculation System Properties					
CS Stage 0 Operand Select	`0x8a00`	Index	Index	I32	RW
CS Stage 0 Const	`0x8a01`	Index	Index	I64	RW
CS Stage 0 Position Index Select	`0x8a02`	Index	Index	I32	RW
CS Stage 0 ADC Index Select	`0x8a03`	Index	Index	I32	RW
CS Stage 0 Environmental Value Select	`0x8a04`	Index	Index	I32	RW
CS Stage 0 Counter Index Select	`0x8a05`	Index	Index	I32	RW
CS Stage 0 Operand Inversion	`0x8a06`	Index	Index	I32	RW
CS Stage 0 Velocity Index Select	`0x8a07`	Index	Index	I32	RW
CS Stage 0 Acceleration Index Select	`0x8a08`	Index	Index	I32	RW
CS Stage 0 Operator Select	`0x8a10`	Index	Index	I32	RW
CS Stage 0 Signal Generator Index Select	`0x8a11`	Index	Index	I32	RW
CS Stage 1 Operand Select	`0x8a20`	Index	Index	I32	RW
CS Stage 1 Const	`0x8a21`	Index	Index	F64	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
CS Stage 1 Calculation System Index Select	`0x8a22`	Index	Index	I32	RW
CS Stage 1 Operator Select	`0x8a30`	Index	Index	I32	RW
CS Shape Enabled	`0x8a40`	Index	`0x00`	I32	RW
CS Shape	`0x8a41`	Index	`0x00`	I32	RW
CS Output Label	`0x8a42`	Index	`0x00`	String	RW
Signal Generator Properties					
SigGen Enable Sync	`0x8b00`	Index	`0x00`	I32	W
SigGen Disable Sync	`0x8b01`	Index	`0x00`	I32	W
SigGen Enabled	`0x8b02`	Index	`0x00`	I32	RW
SigGen Frequency	`0x8b03`	Index	`0x00`	I32	RW
SigGen Amplitude	`0x8b04`	Index	`0x00`	F64	RW
SigGen Offset	`0x8b05`	Index	`0x00`	F64	RW
SigGen Phase	`0x8b06`	Index	`0x00`	I32	RW
SigGen Shape	`0x8b08`	Index	`0x00`	I32	RW
SigGen Mode	`0x8b09`	Index	`0x00`	I32	RW
SigGen Start Trigger Index	`0x8b0a`	Index	`0x00`	I32	RW
SigGen Stop Trigger Index	`0x8b0b`	Index	`0x00`	I32	RW
SigGen Trigger Auto Reset Mode	`0x8b0c`	Index	`0x00`	I32	RW
SigGen State	`0x8b0d`	Index	`0x00`	I32	R
Break Out Box Properties					
BoB Device Serial Number	`0x8c00`	`0x00`	`0x00`	String	R
BoB Hardware Version	`0x8c02`	`0x00`	`0x00`	I32 [5]	R
BoB Hardware Version String	`0x8c03`	`0x00`	`0x00`	String	R
BoB Firmware Version	`0x8c04`	`0x00`	`0x00`	I32 [4]	R
BoB Firmware Version String	`0x8c05`	`0x00`	`0x00`	String	R
Auto Function Properties					
Adjustment State	`0x9010`	`0x00`	`0x00`	I32	RW
Adjustment Progress	`0x9011`	`0x00`	`0x00`	I32	R
Adjustment Signal Control Active	`0x9012`	`0x00`	`0x00`	I32	RW
Adjustment Autostart Autoadjust Active	`0x9013`	`0x00`	`0x00`	I32	RW

Continued on next page

Table 6.1 – *Continued from previous page*

Name	Key 31-16	Key 15-8	Key 7-0	Type	R/W
Library Properties					
Stream Buffer Data Type	`0xf000`	Index	Index	I32	RW
Number of Stream Buffers	`0xf001`	`0x00`	`0x00`	I32	RW
Stream Buffers Interleaved	`0xf002`	`0x00`	`0x00`	I32	RW
Stream Buffer Aggregation	`0xf003`	`0x00`	`0x00`	I32	RW

6.2 Basic Device Properties

6.2.1 Device Type

Definition

C Definition	Code	Type	Access
`SA_SI_DEVICE_TYPE_PROP`	`0x0002`	I32	R

Description

The device type is a numerical value that identifies the type of device. This value will be the same for all devices of a given type.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.2 Device Serial Number

Definition

C Definition	Code	Type	Access
`SA_SI_DEVICE_SERIAL_NUMBER_PROP`	`0x0003`	String	R

Description

This property holds a unique string that identifies an individual device. It is useful when operating several devices of the same type.

The device serial number is a null terminated string with a maximum length of 64 characters (including the null termination).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.3 Device Name

Definition

C Definition	Code	Type	Access
`SA_SI_DEVICE_NAME_PROP`	`0x0004`	String	RW

Description

This property holds an arbitrary name. It may also be used to identify an individual device. The difference to the device serial number is that the device name may be defined by the user. When set it is stored in non-volatile memory and may be recalled in later sessions.

By default the device name is identical to the device serial number.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

The device name is a null terminated string with a maximum length of 64 characters (including the null termination).

6.2.4 Number of Firmware Versions

Definition

C Definition	Code	Type	Access
`SA_SI_NUMBER_OF_FIRMWARE_VERSIONS_PROP`	`0x0005`	I32	R

Description

Each device has one or more software operated components. Each component maintains its own firmware version which may be read out via software. This property defines the valid range for the index value of the *Firmware Version* property (see there).

Note that the index is zero-based. If there are n firmware versions then the valid index range is $0 \ldots n-1$.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.5 Firmware Version

Definition

C Definition	Code	Type	Access
`SA_SI_FIRMWARE_VERSION_PROP`	`0x0006`	I32 [4]	R

Description

This property holds the firmware version of the component with the given index. Read the *Number of Firmware Versions* property to know the valid range of the index.

The firmware version is an array of four 32-bit words with the following meaning:

Word	Meaning
0	Firmware Version Revision
1	Firmware Version Update
2	Firmware Version Minor
3	Firmware Version Major

Index Parameter

- **Index High**: *Component Index* - Selects the component for which the firmware version is to be accessed. The valid range is $0 \ldots NumberOfFirmwareVersions - 1$.
- **Index Low**: Not used (`0x00`).

6.2.6 Firmware Version String

Definition

C Definition	Code	Type	Access
`SA_SI_FIRMWARE_VERSION_STRING_PROP`	`0x0007`	String	R

Description

This property holds the firmware version of the component with the given index as a null terminated string. Read the *Number of Firmware Versions* property to know the valid range of the index.

Index Parameter

- **Index High**: *Component Index* - Selects the component for which the firmware version is to be accessed. The valid range is $0 \ldots NumberOfFirmwareVersions - 1$.
- **Index Low**: Not used (`0x00`).

6.2.7 Number of Channels

Definition

C Definition	Code	Type	Access
`SA_SI_NUMBER_OF_CHANNELS_PROP`	`0x0011`	I32	R

Description

This property holds the number of channels that the slave offers. The value of this property is a constant and depends on the architecture of the slave.

This value defines the valid range for index values of properties that require a channel index. Note that the index is zero-based. If there are *n* channels then the valid index range is $0 \ldots n - 1$.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.8 Maximum Frame Rate

Definition

C Definition	Code	Type	Access
`SA_SI_MAX_FRAME_RATE_PROP`	`0x0020`	I32	R

Description

This property holds the highest frame rate (in Hz) that may be achieved with the current stream configuration. Note that while this is a read-only property it may implicitly change when e.g. enabling or disabling data sources for streaming.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.9 Frame Rate

Definition

C Definition	Code	Type	Access
`SA_SI_FRAME_RATE_PROP`	`0x0021`	I32	RW

Description

This property holds the rate (in Hz) at which data frames for the enabled data sources are generated.

Note that a slave may not be able to generate every desired rate. Therefore, it is recommended to read out this property after writing it to get the actual frame rate that is generated. The slave will chose the closest possible frame rate.

See also: *Precise Frame Rate* property.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . *MaximumFrameRate*

6.2.10 Maximum Frame Aggregation

Definition

C Definition	Code	Type	Access
`SA_SI_MAX_FRAME_AGGREGATION_PROP`	`0x0022`	I32	R

Description

This property holds the highest aggregation value that may be configured with the current stream configuration (see section 2.3.2). Note that while this is a read-only property it may implicitly change when e.g. enabling or disabling data sources for streaming.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.11 Frame Aggregation

Definition

C Definition	Code	Type	Access
`SA_SI_FRAME_AGGREGATION_PROP`	`0x0023`	I32	RW

Description

This property holds the frame aggregation value which defines how many data frames are assembled in one (normal) stream packet (see section 2.3.2).

The default value is 1.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . *MaximumFrameAggregation*

6.2.12 Precise Frame Rate

Definition

C Definition	Code	Type	Access
`SA_SI_PRECISE_FRAME_RATE_PROP`	`0x0025`	F64	R

Description

For frame rates that cannot be represented by an integer value without loss of precision this property may be queried for the currently configured frame rate as a floating point value. It may be useful when reconstructing the time stamps of the stream frames.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.2.13 Event Notification Enabled

Definition

C Definition	Code	Type	Access
`SA_SI_EVENT_NOTIFICATION_ENABLED_PROP`	`0x0030`	I32	RW

Description

This property controls which events the master will be notified about when they occur. To subscribe to an event simply set this property with the corresponding event index to `SA_SI_ENABLED` (`0x01`).

By default the notification for all events are disabled.

Index Parameter

- **Index High/Low**: *Event Index* - 16-bit event index for which the notification should be accessed. See table A.2 for a list of valid event indexes.

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.2.14 Streaming Active

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMING_ACTIVE_PROP`	`0x0040`	I32	RW

Description

This property enables or disables the stream generator of the slave. Before it is enabled the streaming mode should be configured via the *Streaming Mode* property. Once enabled the slave will start generating data frames according to the stream configuration (see section 2.4.1).

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.2.15 Streaming Mode

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMING_MODE_PROP`	`0x0041`	I32	RW

Description

This property defines the behavior of the stream generation once it is enabled via the *Streaming Active* property. The different streaming modes are explained in section 2.4.3.

The default value is `SA_SI_DIRECT_STREAMING` (`0x01`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DIRECT_STREAMING` (`0x01`),
`SA_SI_TRIGGERED_STREAMING` (`0x02`)

6.3 Basic Channel Properties

6.3.1 Number of Data Sources

Definition

C Definition	Code	Type	Access
`SA_SI_NUMBER_OF_DATA_SOURCES_PROP`	`0x1001`	I32	R

Description

This property holds the number of data sources a channel offers. The value of this property is a constant and depends on the architecture of the slave.

This value defines the valid range for index values of properties that require a data source index. Note that the index is zero-based. If there are n data sources then the valid index range is $0 \dots n-1$.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \dots \mathit{NumberOfChannels} - 1$.
- **Index Low**: Not used (`0x00`).

6.3.2 Channel Name

Definition

C Definition	Code	Type	Access
`SA_SI_CHANNEL_NAME_PROP`	`0x1002`	String	RW

Description

This property holds a descriptive name for the channel and has otherwise no further functionality. When set it is stored in non-volatile memory and may be recalled in later sessions.

By default the channel name is "channel x" where x is the channel index.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

The channel name is a null terminated string with a maximum length of 64 characters (including the null termination).

6.4 Basic Data Source Properties

6.4.1 Data Source Type

Definition

C Definition	Code	Type	Access
`SA_SI_DATA_SOURCE_TYPE_PROP`	`0x2001`	I32	R

Description

This property holds the data type that a data source has. See table A.4 for a list of data source types. It gives a basic hint of what kind of data the data source produces (e.g. position, temperature, etc.).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfDataSources* − 1.

6.4.2 Data Type

Definition

C Definition	Code	Type	Access
SA_SI_DATA_TYPE_PROP	0x2002	I32	R

Description

This property holds the data type of the data source. See table A.3 for a list of data types. The data type influences the number of bytes that are added to a data stream packet if the corresponding data source is enabled for streaming.

Note: If data compression is configured for a data source then the number of bytes added to the data stream depends on the compression mode. However, the data type property will not change, since it reflects the basic data type of the data source.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfDataSources* − 1.

6.4.3 Available Compression Modes

Definition

C Definition	Code	Type	Access
SA_SI_AVAILABLE_COMPRESSION_MODES_PROP	0x2003	I32 [ ]	R

Description

This property holds a list of the compression modes that a data source offers. The compression mode `SA_SI_NO_COMPRESSION_MODE` (`0x0000`) is supported by all data sources, so the returned array size is 1 or greater. This property defines the valid range for the *Compression Mode* property.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

6.4.4 Compression Mode

Definition

C Definition	Code	Type	Access
`SA_SI_COMPRESSION_MODE_PROP`	`0x2004`	I32	RW

Description

This property holds the currently configured compression mode. See section 2.4.2 for more information.

The default compression mode is `SA_SI_NO_COMPRESSION_MODE` (`0x0000`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

Valid Range

[*AvailableCompressionModes*]

6.4.5 Streaming Enabled

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMING_ENABLED_PROP`	`0x2005`	I32	RW

Description

When a data source is enabled for streaming then its data is added to the data frames that are generated on each frame clock.

By default all data sources are disabled for streaming.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.4.6 Base Unit

Definition

C Definition	Code	Type	Access
`SA_SI_BASE_UNIT_PROP`	`0x2006`	I32	R

Description

This property holds the basic unit of the values that the data source produces (e.g. seconds, Hertz, percent). See table A.6 for a list of base units.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots \mathit{NumberOfChannels} - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots \mathit{NumberOfDataSources} - 1$.

6.4.7 Base Resolution

Definition

C Definition	Code	Type	Access
`SA_SI_BASE_RESOLUTION_PROP`	`0x2007`	I32	R

Description

This property holds the basic resolution of the values that the data source produces in powers of 10 and is needed to interpret the produced data correctly. For example, a data source with a base unit of Kelvin and a base resolution of -3 would produce 10^{-3} Kelvin (mK) as values. So a value of 273 150 would correspond to 273.15 K.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots \mathit{NumberOfChannels} - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots \mathit{NumberOfDataSources} - 1$.

6.4.8 Resolution Shift

Definition

C Definition	Code	Type	Access
`SA_SI_RESOLUTION_SHIFT_PROP`	`0x2008`	I32	RW

Description

This property is only relevant when using data compression. It may be used to modify the data produced by the data sources by right-shifting their values by a power of two. This is useful for position data sources to configure the trade-off between high position resolution and high movement speeds. With data compression the maximum detectable move speed is limited. By adjusting the resolution shift this limit may be extended while reducing the position resolution on the other hand. When receiving data frames the position data must be shifted back to get the correct position information.

The default resolution shift is 0.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

Valid Range

For data sources of type `SA_SI_POSITION_DSOURCE` (`0x0008`) the valid range is $0 \ldots 4$. For all other data sources only 0 is allowed.

6.4.9 Data Source Name

Definition

C Definition	Code	Type	Access
`SA_SI_DATA_SOURCE_NAME_PROP`	`0x2009`	String	R

Description

This property holds a descriptive string for the data source. Note that this is a read-only property.

The data source name is a null terminated string with a maximum length of 64 characters (including the null termination).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

6.4.10 Is Streamable

Definition

C Definition	Code	Type	Access
`SA_SI_IS_STREAMABLE_PROP`	`0x200a`	I32	R

Description

This property indicates whether the data source may be enabled for streaming or not. Not all data sources are streamable. Some may only be polled by retrieving their data source value manually.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

6.4.11 Component ID

Definition

C Definition	Code	Type	Access
`SA_SI_COMPONENT_ID_PROP`	`0x200b`	I32	R

Description

This property indicates to which component of the device the data source is associated to. This is useful when there are several data sources of the same data source type and serves as an identification mechanism. See also table C.1 in the appendix.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

6.4.12 Component Index

Definition

C Definition	Code	Type	Access
`SA_SI_COMPONENT_INDEX_PROP`	`0x200c`	I32	R

Description

Similar to the component ID, this property further specifies the identity of the data source. See also table C.1 in the appendix.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

6.5 PicoScale Device Properties

6.5.1 Full Access Connection

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FULL_ACCESS_CONNECTION_PROP`	`0x8000`	I32	RW

Description

Writing a `SA_SI_ENABLED` (`0x01`) to this property acquires Full Access Connection. To release a Full Access Connection write a `SA_SI_DISABLED` (`0x00`). See section 2.8 for more information.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

SA_SI_ENABLED (0x01),
SA_SI_DISABLED (0x00)

6.5.2 LVDS LS Connected

Definition

C Definition	Code	Type	Access
SA_PS_SYS_LVDS_LS_CONNECTED_PROP	0x8012	I32	R

Description

This property will hold the value 0x01 if a device is connected via the LVDS interface. Otherwise it will read 0x00.

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

6.5.3 Pilot Laser Active

Definition

C Definition	Code	Type	Access
SA_PS_SYS_PILOT_LASER_ACTIVE_PROP	0x8020	I32	RW

Description

This property may be used to enable or disable the pilot laser.

The default value is SA_SI_DISABLED (0x00).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.5.4 System Stable

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_IS_STABLE_PROP`	`0x8030`	I32	R

Description

This property may be queried to check whether the measuring system is stable and ready to produce accurate position values. The value returned may be 0 (not stable) or 1 (stable).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.5 Working Distance Min

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_WORKING_DISTANCE_MIN_PROP`	`0x8040`	F64	RW

Description

This property holds the minimum working distance. After writing it you should perform a write access to `SA_PS_SYS_WORKING_DISTANCE_ACTIVATE_PROP` (`0x8042`) to make the changes take effect. The minimum working distance is given in mm.

The default value is 0, which is an invalid minimum working distance that indicates that the working distance has not been configured yet.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . 500

6.5.6 Working Distance Max

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_WORKING_DISTANCE_MAX_PROP`	`0x8041`	F64	RW

Description

This property holds the maximum working distance. After writing it you should perform a write access to `SA_PS_SYS_WORKING_DISTANCE_ACTIVATE_PROP` (`0x8042`) to make the changes take effect. The maximum working distance is given in mm.

The default value is 0, which is an invalid maximum working distance that indicates that the working distance has not been configured yet.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . 500

6.5.7 Working Distance Activate

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_WORKING_DISTANCE_ACTIVATE_PROP`	`0x8042`	I32	W

Description

When this property is written (with any value) then the previously written minimum and maximum working distance properties take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.5.8 Working Distance Shrink Mode

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_WORKING_DISTANCE_SHRINK_MODE_PROP`	`0x8043`	I32	RW

Description

When activating the working distance with the *Working Distance Activate* property the device internally adjusts the working distance for an optimal configuration. The shrink mode influences this adjustment. With mode "left" the maximum working distance is fix and only the minimum working distance is adjusted. With mode "right" the minimum working distance is fix and only the maximum working distance is adjusted. With mode "left-right" both minimum and maximum are adjusted.

The default value is `SA_PS_WORKING_DISTANCE_SHRINK_MODE_LEFT_RIGHT` (`0x00`).

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

SA_PS_WORKING_DISTANCE_SHRINK_MODE_LEFT_RIGHT (0x00),
SA_PS_WORKING_DISTANCE_SHRINK_MODE_LEFT (0x01),
SA_PS_WORKING_DISTANCE_SHRINK_MODE_RIGHT (0x02)

6.5.9 Network Current IP

Definition

C Definition	Code	Type	Access
SA_PS_SYS_NETWORK_CURRENT_IP_PROP	0x8052	String	R

Description

This property holds the IP address that is currently configured for the device.

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

6.5.10 Network Config Activate

Definition

C Definition	Code	Type	Access
SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP	0x8060	I32	W

Description

When writing SA_SI_ENABLED (0x01) to this property, the network configuration parameters that were previously written to the corresponding properties are activated. The affected properties are: *Network Config DHCP*, *Network Config IP*, *Network Config Gateway*, *Network Config Netmask*, *Network Config Nameserver*

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`)

6.5.11 Network Config DHCP

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_DHCP_PROP`	`0x8061`	I32	RW

Description

This property enables or disables the usage of a DHCP server for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

The default value is `SA_SI_ENABLED` (`0x01`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.5.12 Network Config IP

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_IP_PROP`	`0x8062`	String	RW

Description

This property configures the IP address that should be used for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any valid IP address as a null terminated string.

6.5.13 Network Config Gateway

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_GATEWAY_PROP`	`0x8063`	String	RW

Description

This property configures the IP address of the gateway that should be used for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any valid IP address as a null terminated string.

6.5.14 Network Config Netmask

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_NETMASK_PROP`	`0x8064`	String	RW

Description

This property configures the network mask that should be used for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any valid network mask as a null terminated string.

6.5.15 Network Config Nameserver

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_NAMESERVER_PROP`	`0x8065`	String	RW

Description

This property configures the IP address of the name server that should be used for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any valid IP address as a null terminated string.

6.5.16 Network Config Domain Name

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_CONFIG_DOMAINNAME_PROP`	`0x8066`	String	RW

Description

This property configures the domain name that is used for the network connection. Note that a write access to `SA_PS_SYS_NETWORK_CONFIG_ACTIVATE_PROP` (`0x8060`) is required to make the changes take effect.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

The device name is a null terminated string with a maximum length of 64 characters (including the null termination).

6.5.17 Network MAC

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_NETWORK_MAC_PROP`	`0x8070`	String	R

Description

This property holds the network MAC address of the device as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.18 Head Type Category Count

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HEAD_TYPE_CATEGORY_COUNT_PROP`	`0x8081`	I32	R

Description

This property holds the number of head type categories that the device supports. Each category may contain several head types that may be selected with the *Head Type* property.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.19 Head Type Count

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HEAD_TYPE_COUNT_PROP`	`0x8082`	I32	R

Description

This property holds the number of head types within a head type category.

Index Parameter

- **Index High**: Category index.
- **Index Low**: Not used (`0x00`).

6.5.20 Head Type Category Name

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HEAD_TYPE_CATEGORY_NAME_PROP`	`0x8083`	String	R

Description

This property holds a descriptive name of a head type category.

Index Parameter

- **Index High**: Category index.
- **Index Low**: Not used (`0x00`).

6.5.21 Head Type Name

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HEAD_TYPE_NAME_PROP`	`0x8084`	String	R

Description

This property holds a descriptive name of a head type selected from a head type category.

Index Parameter

- **Index High**: Category index.
- **Index Low**: Head type index.

6.5.22 Fiber Length Head

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FIBERLENGTH_HEAD_PROP`	`0x8090`	F64	RW

Description

This property tells the PicoScale how long the fiber from the device connector to the measuring head is. This value is used internally for optimization purposes. It is given in mm.

Note that this value is a global value that applies to all channels equally. If written it is stored to non-volatile memory. Therefore, there is no need to configure it on each start-up.

The default value is 1600.0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive

6.5.23 Fiber Length Extension

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FIBERLENGTH_EXTENSION_PROP`	`0x8091`	F64	RW

Description

In case the fiber of the measuring heads of the channels are extended the length of the extension may be written to this property. This value is used internally for optimization purposes. It is given in mm.

Note that this value is a global value that applies to all channels equally. If written it is stored to non-volatile memory. Therefore, there is no need to configure it on each start-up.

The default value is 0.0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive

6.5.24 Position all Channels

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_POSITION_ALL_CH_PROP`	`0x80a0`	I64	W

Description

Sets the position values of all channels synchronously.

Valid Range

Any positive

6.5.25 Configuration Save

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_CONFIGURATION_SAVE_PROP`	`0x80c0`	I32	W

Description

When writing this property the PicoScale configuration is saved to a non-volatile memory slot. The slot is selected via the index high parameter. Useful when performing the same measurement repeatedly across several sessions. See section 2.10 for more information.

Index Parameter

- **Index High**: *Configuration Slot Index* - Selects the configuration slot.
- **Index Low**: Not used (`0x00`).

Valid Range

1

6.5.26 Configuration Load

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_CONFIGURATION_LOAD_PROP`	`0x80c1`	I32	W

Description

When writing this property a previously saved PicoScale configuration is restored. See section 2.10 for more information.

Index Parameter

- **Index High**: *Configuration Slot Index* - Selects the configuration slot.
- **Index Low**: Not used (`0x00`).

Valid Range

1

6.5.27 Configuration Name

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_CONFIGURATION_NAME_PROP`	`0x80c2`	String	RW

Description

This property holds the name of the selected configuration slot. The name may be chosen freely. The slot is selected via the index high parameter. See section 2.10 for more information.

The default value is an empty string.

Index Parameter

- **Index High**: *Configuration Slot Index* - Selects the configuration slot.
- **Index Low**: Not used (`0x00`).

Valid Range

The configuration slot name is a null terminated string with a maximum length of 64 characters (including the null termination).

6.5.28 Configuration Count

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_CONFIGURATION_COUNT_PROP`	`0x80c3`	I32	R

Description

This property holds the number of available configuration slots. It defines the valid range of the index high parameter for the other configuration slot properties. See section 2.10 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.29 Filter CutOff Frequency

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FILTER_CUTOFF_FREQUENCY_PROP`	`0x80de`	F64	RW

Description

Sets the global device filter configuration to the desired -3dB cutoff frequency by selecting the next smallest filter setting. Alternatively, the filter may be configured via the *Filter Rate* property. The value is given in Hz.

The default value is 2.9 MHz.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 2 900 000

6.5.30 Filter Rate

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FILTER_RATE_PROP`	`0x80df`	F64	RW

Description

Sets the global device filter configuration to the desired frequency in Hz.

The default value is 10 MHz.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 10 000 000

6.5.31 Bootloader Version

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_BOOTLOADER_VERSION_PROP`	`0x80e0`	I32 [4]	R

Description

This property holds the version of the bootloader. The version is an array of four 32-bit words with the following meaning:

Word	Meaning
0	Version Revision
1	Version Update
2	Version Minor
3	Version Major

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.32 Bootloader Version String

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_BOOTLOADER_VERSION_STRING_PROP`	`0x80e1`	String	R

Description

This property holds the bootloader version as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.33 Hardware Version

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HARDWARE_VERSION_PROP`	`0x80e2`	I32 [5]	R

Description

This property holds the hardware version of the device. The version is an array of five 32-bit words.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.34 Hardware Version String

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_HARDWARE_VERSION_STRING_PROP`	`0x80e3`	String	R

Description

This property holds the hardware version of the device as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.35 Product Version

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_PRODUCT_VERSION_PROP`	`0x80e4`	I32 [4]	R

Description

This property holds the product version of the device. The version is an array of four 32-bit words with the following meaning:

Word	Meaning
0	Version Revision
1	Version Update
2	Version Minor
3	Version Major

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.36 Product Version String

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_PRODUCT_VERSION_STRING_PROP`	`0x80e5`	String	R

Description

This property holds the product version of the device as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.37 Feature Count

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FEATURE_COUNT_PROP`	`0x80f0`	I32	R

Description

This property returns the number of features that the device offers. It may be used to iterate over the other device feature properties. See section 2.7 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.5.38 Feature Name

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FEATURE_NAME_PROP`	`0x80f1`	String	R

Description

This property holds the name of the feature with the given index. The feature name is a null-terminated string with a maximum length of 64 bytes (including the null termination). See section 2.7 for more information.

Index Parameter

- **Index High**: *Feature Index* - Selects the feature for which the property is to be accessed. The valid range is $0 \ldots FeatureCount - 1$.
- **Index Low**: Not used (`0x00`).

6.5.39 Feature Time

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FEATURE_TIME_PROP`	`0x80f2`	I32	R

Description

This property holds the time (in seconds) that the feature with the given index may still be used. A value of 0 means the feature is disabled. The feature may be evaluated by activating an evaluation period (if available, see property *Feature Evaluate*). This will set the feature time to a non-zero value. The special value 2 147 483 647 means the feature has been purchased and is available infinitely. See section 2.7 for more information.

Index Parameter

- **Index High**: *Feature Index* - Selects the feature for which the property is to be accessed. The valid range is $0 \ldots FeatureCount - 1$.
- **Index Low**: Not used (`0x00`).

6.5.40 Feature Evaluate

Definition

C Definition	Code	Type	Access
`SA_PS_SYS_FEATURE_EVALUATE_PROP`	`0x80f3`	I32	RW

Description

This property controls the feature evaluation. Reading it returns the number of evaluation periods the user may start for the feature with the given index.

Writing a `SA_SI_ENABLED` (`0x01`) to this property starts an evaluation period. The evaluation counter is decremented and the feature time is incremented by a fixed amount of time. If the counter is already zero at the time of write, the write will fail. See section 2.7 for more information.

The default value is 10.

Index Parameter

- **Index High**: *Feature Index* - Selects the feature for which the property is to be accessed. The valid range is $0 \ldots FeatureCount - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`)

6.6 PicoScale Channel Properties

6.6.1 Channel Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_CH_ENABLED_PROP`	`0x8100`	I32	RW

Description

This property enables or disables the channel with the given index.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.6.2 Is Valid

Definition

C Definition	Code	Type	Access
`SA_PS_CH_IS_VALID_PROP`	`0x8101`	I32	R

Description

This property indicates whether a channel is ready to produce accurate position data. The value returned will be 0 (not valid) or 1 (valid).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

6.6.3 Position

Definition

C Definition	Code	Type	Access
`SA_PS_CH_POSITION_PROP`	`0x8102`	I64	W

Description

This write-only property may be used to configure the current position of a channel to an arbitrary value.

Note that this property cannot be used to read the current position of a channel. This is achieved by calling the function to get a data source value instead.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.6.4 Scale Inversion

Definition

C Definition	Code	Type	Access
`SA_PS_CH_SCALE_INVERSION_PROP`	`0x8103`	I32	RW

Description

This property may be used to invert the position scale of a channel.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.6.5 Dead Path Correction Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_CH_DEAD_PATH_CORRECTION_ENABLED_PROP`	`0x8110`	I32	RW

Description

This property enables or disables the dead path correction of a channel. See also section 3.1.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.6.6 Dead Path

Definition

C Definition	Code	Type	Access
`SA_PS_CH_DEAD_PATH_PROP`	`0x8111`	I64	RW

Description

This property holds the dead path value used to correct the calculated position values. Note that the dead path is only applied if the dead path correction is enabled. See also section 3.1.

The default value is 0.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive

6.6.7 Head Type

Definition

C Definition	Code	Type	Access
`SA_PS_CH_HEAD_TYPE_PROP`	`0x8112`	I32	RW

Description

This property configures the head type that is connected to a channel. The head type is a 16-bit value where the upper eight bits code the head type category and the lower eight bits code the head type within the category. You may query the supported categories and head types with the properties *Head Type Category Count*, *Head Type Category Name*, *Head Type Count* and *Head Type Name*.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

Bits 15 . . . 8: a valid head type category index
Bits 7 . . . 0: a valid head type index

6.6.8 Beam Interrupt Tolerance

Definition

C Definition	Code	Type	Access
`SA_PS_CH_BEAM_INTERRUPT_TOLERANCE_PROP`	`0x8113`	I32	RW

Description

Determines the tolerance for the generation of beam interrupt events. The value is given in percent. Higher values result in a higher tolerance.

The default value is 33.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 100

6.7 Environmental Sensor Properties

6.7.1 Env Sensor Operation Mode

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_OPERATION_MODE_PROP`	`0x8200`	I32	RW

Description

This property defines the operation mode of the environmental sensor (if one is attached). See section 3.1 for more information.

The default value is `SA_PS_ENV_OP_MODE_AUTOMATIC` (`0x01`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_ENV_OP_MODE_MANUAL` (`0x00`),
`SA_PS_ENV_OP_MODE_AUTOMATIC` (`0x01`)

6.7.2 Env Sensor State

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_STATE_PROP`	`0x8201`	I32	R

Description

This property holds a bit mask that indicates the state of the environmental sensor.

Bit	Meaning
0	Temperature Error - If this bit is set then the temperature is invalid.
1	Humidity Error - If this bit is set then the humidity is invalid.
2	Pressure Error - If this bit is set then the pressure is invalid.
3	Sensor Offline - If this bit is set then the environmental sensor is offline.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.3 Env Sensor Version

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_VERSION_PROP`	`0x820e`	I32 [4]	R

Description

This property may be read to get the firmware version of the connected environmental sensor.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.4 Env Sensor Version String

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_VERSION_STRING_PROP`	`0x820f`	String	R

Description

This property may be read to get the firmware version of the connected environmental sensor as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.5 Env Sensor Temperature

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_TEMPERATURE_PROP`	`0x8210`	I32	R

Description

This property holds the temperature that is detected by the environmental sensor. The temperature is given in m°C. See section 3.1 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.6 Env Sensor Pressure

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_PRESSURE_PROP`	`0x8211`	I32	R

Description

This property holds the pressure that is detected by the environmental sensor. The pressure is given in mbar. See section 3.1 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.7 Env Sensor Humidity

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_SENSOR_HUMIDITY_PROP`	`0x8212`	I32	R

Description

This property holds the humidity that is detected by the environmental sensor. The humidity is given in m%. See section 3.1 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.7.8 Env User Temperature

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_USER_TEMPERATURE_PROP`	`0x8220`	I32	RW

Description

This property holds the temperature value that is used when the environmental sensor is operated in the `SA_PS_ENV_OP_MODE_MANUAL` (`0x00`). The temperature is given in m°C. See section 3.1 for more information.

The default value is 20°C (20 000).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

$-50000 \ldots 150000$

6.7.9 Env User Pressure

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_USER_PRESSURE_PROP`	`0x8221`	I32	RW

Description

This property holds the pressure value that is used when the environmental sensor is operated in the `SA_PS_ENV_OP_MODE_MANUAL` (`0x00`). The pressure is given in mbar. See section 3.1 for more information.

The default value is 1013mbar (1013).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots 2000$

6.7.10 Env User Humidity

Definition

C Definition	Code	Type	Access
`SA_PS_ENV_USER_HUMIDITY_PROP`	`0x8222`	I32	RW

Description

This property holds the humidity value that is used when the environmental sensor is operated in the `SA_PS_ENV_OP_MODE_MANUAL` (`0x00`). The humidity is given in m%. See section 3.1 for more information.

The default value is 50% (50 000).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 100000

6.8 GPIO Properties

6.8.1 GPIO DAC Count

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_COUNT_PROP`	`0x8310`	I32	R

Description

This property holds the number of DACs the device offers. It defines the valid range of the index values of the other DAC related properties.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.8.2 GPIO DAC Bit Width

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_BIT_WIDTH_PROP`	`0x8311`	I32	R

Description

This property holds the bit width of the selected DAC. The DACs of a device may have different resolutions (typically 12 or 16 bit).

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC ($0 \ldots DACCount - 1$).
- **Index Low**: Not used (`0x00`).

6.8.3 GPIO DAC Sample Rate

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_SAMPLE_RATE_PROP`	`0x8312`	I32	R

Description

This property holds the sample rate of the selected DAC. The DACs of a device may have different sample rates (e.g. 200 kHz). The value returned is given in Hz.

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC ($0 \ldots DACCount - 1$).
- **Index Low**: Not used (`0x00`).

6.8.4 GPIO DAC Const Value

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_CONST_VALUE_PROP`	`0x8313`	F64	RW

Description

This property holds the value that is output to the DAC if the `SA_PS_GPIO_DAC_SOURCE_PROP` property is set to `SA_PS_GPIO_DAC_SOURCE_CONST` (`0x00`). Note that a value of 0.0 will let the DAC output its minimum voltage and a value of 1.0 will let the DAC output its maximum voltage. The absolute voltage depends on the DAC that is used (see User Manual for detailed specifications). See also section 3.7.2 "Analog Outputs" for more information.

The default value is 0.0.

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC ($0 \ldots DACCount - 1$).
- **Index Low**: Not used (`0x00`).

Valid Range

$-1.0 \ldots 1.0$

6.8.5 GPIO DAC Source

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_SOURCE_PROP`	`0x8318`	I32	RW

Description

This property selects what value should be output to the DAC. There are three options:

- **Static** In this case the output value of the DAC is constant and defined by the value written to the *GPIO DAC Const Value* property.
- **Calculation System** Outputs the value of a calculation system on the DAC. The calculation system is selected by the *GPIO DAC Calculation System Index* property.
- **Signal Generator** Outputs the value of a signal generator on the DAC. The signal generator is selected by the *GPIO DAC Signal Generator Index* property.

See section 3.7.2 for more information.

The default value is `SA_PS_GPIO_DAC_SOURCE_CONST` (`0x00`).

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC (0 . . . *DACCount* − 1).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_GPIO_DAC_SOURCE_CONST` (`0x00`),
`SA_PS_GPIO_DAC_SOURCE_CALCULATION_SYSTEM` (`0x01`),
`SA_PS_GPIO_DAC_SOURCE_SIGNAL_GENERATOR` (`0x02`)

6.8.6 GPIO DAC Calculation System Index

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_CALC_SYS_INDEX_PROP`	`0x8319`	I32	RW

Description

This property selects the calculation system of which the result is output on the DAC. Note that this setting only takes effect if the *GPIO DAC Source* property is set to `SA_PS_GPIO_DAC_SOURCE_CALCULATION_SYSTEM` (`0x01`). See section 3.7.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC (0 . . . *DACCount* − 1).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 2

6.8.7 GPIO DAC Signal Generator Index

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DAC_SIG_GEN_INDEX_PROP`	`0x831a`	I32	RW

Description

This property selects the signal generator of which the result is output on the DAC. Note that this setting only takes effect if the *GPIO DAC Source* property is set to `SA_PS_GPIO_DAC_SOURCE_SIGNAL_GENERATOR` (`0x02`). See section 3.7.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *DAC Index* - Selects the DAC ($0 \ldots DACCount - 1$).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots 4$

6.8.8 GPIO Digital Const Value

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DIGITAL_CONST_VALUE_PROP`	`0x8320`	I32	RW

Description

This property controls the level of an I/O pin, but only if the pin direction is configured as output and the *GPIO Digital Source* property is set to `SA_PS_GPIO_DIG_SOURCE_CONST` (`0x00`). See section 3.7.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Pin index. Selects the I/O pin (0 . . . 8).
- **Index Low**: Not used (`0x00`).

Valid Range

0 (low level), 1 (high level)

6.8.9 GPIO Digital Direction

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DIGITAL_DIRECTION_PROP`	`0x8321`	I32	RW

Description

This property defines the direction of the digital I/O pin. If set to 0 the pin is used as an input pin. If set to 1 the pin is used as an output pin. See section 3.7.3 for more information.

The default value is 0 (input).

Index Parameter

- **Index High**: Pin index. Selects the I/O pin (0 . . . 8).
- **Index Low**: Not used (`0x00`).

Valid Range

0 (input), 1 (output)

6.8.10 GPIO Digital Source

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DIGITAL_SOURCE_PROP`	`0x8322`	I32	RW

Description

This property selects where the value to be output on the I/O pin is taken from if the pin is used as an output pin. If set to `SA_PS_GPIO_DIG_SOURCE_CONST` (`0x00`) then the output level is defined by the *GPIO Digital Const Value* property. If set to `SA_PS_GPIO_DIG_SOURCE_TRIGGER` (`0x01`) then the output level is defined by the selected trigger output (selected by the *GPIO Digital Trigger Index* property). If set to `SA_PS_GPIO_DIG_SOURCE_CLOCK_GENERATOR` (`0x02`) then the output is defined by the selected clock generator (selected by the *GPIO Digital Clock Generator Index* property). See section 3.7.3 for more information.

The default value is `SA_PS_GPIO_DIG_SOURCE_CONST` (`0x00`).

Index Parameter

- **Index High**: Pin index. Selects the I/O pin (0 . . . 8).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_GPIO_DIG_SOURCE_CONST` (`0x00`),
`SA_PS_GPIO_DIG_SOURCE_TRIGGER` (`0x01`),
`SA_PS_GPIO_DIG_SOURCE_CLOCK_GENERATOR` (`0x02`)

6.8.11 GPIO Digital Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DIGITAL_TRIGGER_INDEX_PROP`	`0x8323`	I32	RW

Description

If the I/O pin selected by the Index High is used as an output pin and the *GPIO Digital Source* property is set to `SA_PS_GPIO_DIG_SOURCE_TRIGGER` (`0x01`) then this property selects which trigger output is used to define the level of the I/O pin. See section 3.7.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Pin index. Selects the I/O pin (0 . . . 8).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.8.12 GPIO Digital Clock Generator Index

Definition

C Definition	Code	Type	Access
`SA_PS_GPIO_DIGITAL_CLK_GEN_INDEX_PROP`	`0x8324`	I32	RW

Description

If the I/O pin selected by the Index High is used as an output pin and the *GPIO Digital Source* property is set to `SA_PS_GPIO_DIG_SOURCE_CLOCK_GENERATOR` (`0x02`) then this property selects which clock generator is used to define the output of the I/O pin. See section 3.7.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Pin index. Selects the I/O pin ($0 \ldots 8$).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots 1$

6.9 Trigger System Properties

6.9.1 Trigger Source Count

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_COUNT_PROP`	`0x8400`	I32	R

Description

This property holds the number of trigger sources that the device offers. It defines the valid range of the index high parameter for the other trigger source configuration properties. See section 3.2 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.9.2 Trigger Source Reset

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_RESET_PROP`	`0x8401`	I32	W

Description

Writing this property (with any value) will reset the selected trigger source to the inactive state. See section 3.2 for more information.

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.9.3 Trigger Source Event

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_EVENT_PROP`	`0x8402`	I32	RW

Description

This property defines to which event the selected trigger source should react to. See section 3.2 for more information.

The default value is `SA_PS_TRIG_EVENT_NONE` (`0x00`).

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIG_EVENT_NONE` (`0x00`),
`SA_PS_TRIG_EVENT_SOFTWARE` (`0x01`),
`SA_PS_TRIG_EVENT_DATA_SOURCE_VALUE` (`0x02`),
`SA_PS_TRIG_EVENT_DATA_SOURCE_INCREMENT` (`0x03`),
`SA_PS_TRIG_EVENT_GPIO_TRIGGER` (`0x04`),
`SA_PS_TRIG_EVENT_EXTERNAL_TRIGGER` (`0x05`),
`SA_PS_TRIG_EVENT_INTERNAL` (`0x06`)

6.9.4 Trigger Source Index 0

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_INDEX0_PROP`	`0x8403`	I32	RW

Description

This property defines the first index value that the trigger source uses to react to events. Its meaning depends on the configured event. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

Depends on selected event.

6.9.5 Trigger Source Index 1

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_INDEX1_PROP`	`0x8404`	I32	RW

Description

This property defines the second index value that the trigger source uses to react to events. Its meaning depends on the configured event. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

Depends on selected event.

6.9.6 Trigger Source Condition

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_CONDITION_PROP`	`0x8405`	I32	RW

Description

This property defines the condition under which the trigger source reacts to events. Its meaning depends on the configured event. See section 3.2 for more information.

The default value is `SA_PS_TRIG_CONDITION_RISING` (`0x00`).

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIG_CONDITION_RISING` (`0x00`),
`SA_PS_TRIG_CONDITION_FALLING` (`0x01`),
`SA_PS_TRIG_CONDITION_EITHER` (`0x02`),
`SA_PS_TRIG_CONDITION_POSITIVE_LEVEL` (`0x03`),
`SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL` (`0x04`),
`SA_PS_TRIG_CONDITION_POSITIVE_RANGE` (`0x05`),
`SA_PS_TRIG_CONDITION_NEGATIVE_RANGE` (`0x06`)

6.9.7 Trigger Source Value 0

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_VALUE0_PROP`	`0x8406`	I64	RW

Description

This property defines the first threshold value that the trigger source uses to react to events. Its meaning depends on the configured event. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.9.8 Trigger Source Value 1

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_SOURCE_VALUE1_PROP`	`0x8407`	I64	RW

Description

This property defines the second threshold value that the trigger source uses to react to events. Its meaning depends on the configured event. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Source Index* - Selects the trigger source.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.9.9 Trigger Count

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_COUNT_PROP`	`0x8410`	I32	R

Description

This property holds the number of triggers that the device offers. It defines the valid range of the index high parameter for the other trigger configuration properties. See section 3.2 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.9.10 Trigger AND Mask

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_AND_MASK_PROP`	`0x8411`	I32	RW

Description

This property holds the AND mask that is used for the logic operations described in section 3.2. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Index* - Selects the trigger.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.9.11 Trigger OR Mask

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_OR_MASK_PROP`	`0x8412`	I32	RW

Description

This property holds the OR mask that is used for the logic operations described in section 3.2. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Index* - Selects the trigger.
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.9.12 Trigger Logic Operation

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_LOGIC_OPERATION_PROP`	`0x8413`	I32	RW

Description

This property holds the logic operation that is used for the logic operations described in section 3.2. See section 3.2 for more information.

The default value is `SA_PS_LOGIC_OP_NONE` (`0x00`).

Index Parameter

- **Index High**: *Trigger Index* - Selects the trigger.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_LOGIC_OP_NONE` (`0x00`),
`SA_PS_LOGIC_OP_OR` (`0x01`),
`SA_PS_LOGIC_OP_NOR` (`0x02`),
`SA_PS_LOGIC_OP_AND` (`0x03`),
`SA_PS_LOGIC_OP_NAND` (`0x04`),
`SA_PS_LOGIC_OP_XOR` (`0x05`),
`SA_PS_LOGIC_OP_NXOR` (`0x06`)

6.9.13 Trigger Output Delay

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_OUTPUT_DELAY_PROP`	`0x8414`	I32	RW

Description

This property defines the delay (in multiples of 10ns) that is used to debounce the final trigger output signals. See section 3.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Trigger Index* - Selects the trigger.
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive

6.9.14 Trigger Output Mode

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_OUTPUT_MODE_PROP`	`0x8415`	I32	RW

Description

This property defines the output mode of the selected trigger. See section 3.2 for more information.

The default value is `SA_PS_TRIGGER_OUTPUT_MODE_IMMEDIATE_NO_RESET` (`0x00`).

Index Parameter

- **Index High**: *Trigger Index* - Selects the trigger.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIGGER_OUTPUT_MODE_IMMEDIATE_NO_RESET` (`0x00`),
`SA_PS_TRIGGER_OUTPUT_MODE_IMMEDIATE_RESET` (`0x01`),
`SA_PS_TRIGGER_OUTPUT_MODE_DELAYED_NO_RESET` (`0x02`),
`SA_PS_TRIGGER_OUTPUT_MODE_DELAYED_RESET` (`0x03`)

6.9.15 Soft Trigger

Definition

C Definition	Code	Type	Access
SA_PS_TRIGGER_SOFT_TRIGGER_PROP	0x8420	I32	W

Description

Writing this property will execute a software trigger. Depending on the property value that is written the targeted trigger source(s) will switch to the active state (when writing SA_SI_ENABLED (0x01)) or inactive state (when writing SA_SI_DISABLED (0x00)). See section 3.2 for more information.

Note that since several trigger sources may be configured with the same trigger ID it is possible to trigger groups of trigger sources at the same time.

Example:

```
// Set all trigger sources that are configured for software
// event with trigger ID 3 to the activated state.
SA_SI_SetProperty_i32(
    SA_SI_EPK(SA_PS_TRIGGER_SOFT_TRIGGER_PROP,0x03,0x00),
    SA_SI_ENABLED
);
```

Index Parameter

- **Index High**: *Trigger ID* - Triggers all trigger sources that are configured with this ID.
- **Index Low**: Not used (0x00).

Valid Range

SA_SI_ENABLED (0x01),
SA_SI_DISABLED (0x00)

6.9.16 Trigger Source State

Definition

C Definition	Code	Type	Access
SA_PS_TRIGGER_SOURCE_STATE_PROP	0x8430	I32	R

Description

Holds the current state of all trigger sources as a bit mask. See section 3.2 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.9.17 Trigger State

Definition

C Definition	Code	Type	Access
`SA_PS_TRIGGER_STATE_PROP`	`0x8431`	I32	R

Description

Holds the current state of all triggers as a bit mask. See section 3.2 for more information.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.10 Digital Differential Interface Properties

6.10.1 DDI Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_ENABLED_PROP`	`0x8500`	I32	RW

Description

This property enables or disables the Digital Differential Interface of a channel.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.10.2 DDI Mode

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_MODE_PROP`	`0x8501`	I32	RW

Description

This property selects the operation mode for the Digital Differential Interface. Note that the mode may only be changed while the DDI is disabled.

The default value is `SA_PS_DDI_MODE_QUADRATURE` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is 0 . . . *NumberOfChannels* − 1.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_DDI_MODE_QUADRATURE` (`0x00`),
`SA_PS_DDI_MODE_SERIAL_DATA` (`0x01`)

6.10.3 DDI Source

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SOURCE_PROP`	`0x8502`	I32	RW

Description

This property selects the data source that is output on the DDI.

The default value is `SA_PS_DDI_SOURCE_POSITION` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots \mathit{NumberOfChannels} - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_DDI_SOURCE_POSITION` (`0x00`),
`SA_PS_DDI_SOURCE_CALC_SYS` (`0x01`)

6.10.4 DDI Quad Step Size

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_QUAD_STEP_SIZE_PROP`	`0x8510`	I32	RW

Description

This property defines how many LSBs a data value must change before a quadrature pulse is output. For example, in case of position data this corresponds to pico meters.

The default value is 1.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

$1 \ldots 1\,000\,000\,000$

6.10.5 DDI Quad Frequency

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_QUAD_FREQUENCY_PROP`	`0x8511`	I32	RW

Description

This property defines the maximum frequency (in Hz) at which quadrature output signals are produced.

The default value is 5 000 000.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

See *DDI Quad Available Frequencies* property.

6.10.6 DDI Quad Available Frequencies

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_QUAD_AVAILABLE_FREQUENCIES_PROP`	`0x8512`	I32 [ ]	R

Description

This property may be read to get a list of valid frequencies that may be written to the *DDI Quad Frequency* property.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.10.7 DDI Serial Clock Mode

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_CLOCK_MODE_PROP`	`0x8520`	I32	RW

Description

Sets the transfer mode if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL0_CPHA0` (`0x00`),
`SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL0_CPHA1` (`0x01`),
`SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL1_CPHA0` (`0x02`),
`SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL1_CPHA1` (`0x03`)

6.10.8 DDI Serial Clock Frequency

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_CLOCK_FREQUENCY_PROP`	`0x8521`	I32	RW

Description

Sets the frequency (in Hz) of the clock signal if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

The default value is 1 250 000 Hz.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . 10 000 000

6.10.9 DDI Serial Clock Pause Delay

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_CLOCK_PAUSE_DELAY_PROP`	`0x8522`	I32	RW

Description

Configures the number of delay cycles between data word transfers if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

The default value is 3.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots 2\,147\,483\,647$

6.10.10 DDI Serial Data Bit Width

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_DATA_BITWIDTH_PROP`	`0x8523`	I32	RW

Description

Configures the width of the transmitted data words (in bits) if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

The default value is 48.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

$8 \ldots 64$

6.10.11 DDI Serial Data Resolution Shift

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_DATA_RESOLUTION_SHIFT_PROP`	`0x8524`	I32	RW

Description

Defines the resolution of the transmitted data words if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

The default value is 0.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots 55$

6.10.12 DDI Serial Data Idle Polarity

Definition

C Definition	Code	Type	Access
`SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_PROP`	`0x8525`	I32	RW

Description

Defines the idle polarity of the data signal if the DDI is operated in serial data output mode. See section 3.8.2 for more information.

The default value is `SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_LOW` (`0x00`).

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_LOW` (`0x00`),
`SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_HIGH` (`0x01`)

6.11 Clock Generator Properties

6.11.1 Clock Generator Enable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_ENABLE_SYNC_PROP`	`0x8600`	I32	W

Description

A bit mask may be written to this property to synchronously enable several clock generators at once. If bit 0 is set then clock generator 0 is enabled, if bit 1 is set then clock generator 1 is enabled and so on.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.11.2 Clock Generator Disable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_DISABLE_SYNC_PROP`	`0x8601`	I32	W

Description

A bit mask may be written to this property to synchronously disable several clock generators at once. If bit 0 is set then clock generator 0 is disabled, if bit 1 is set then clock generator 1 is disabled and so on.

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

Any

6.11.3 Clock Generator Enabled

Definition

C Definition	Code	Type	Access
SA_PS_CLOCK_GENERATOR_ENABLED_PROP	0x8610	I32	RW

Description

This property holds the enabled state of the selected clock generator. It may be written to enable or disable the clock generator. Note that to enable or disable several clock generators at once you must use the *Clock Generator Enable Sync* and *Clock Generator Disable Sync* properties instead.

The default value is SA_SI_DISABLED (0x00).

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (0x00).

Valid Range

SA_SI_DISABLED (0x00),
SA_SI_ENABLED (0x01)

6.11.4 Clock Generator Mode

Definition

C Definition	Code	Type	Access
SA_PS_CLOCK_GENERATOR_MODE_PROP	0x8611	I32	RW

Description

Selects the operation mode of the clock generator. In *direct* mode the clock generator will start to run as soon as it is enabled and stop when it is disabled. In *triggered* mode, after being enabled, the clock generator is controlled by the trigger system. See section 3.3 for more information.

The default value is `SA_PS_CLK_GEN_MODE_DIRECT` (`0x00`).

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_CLK_GEN_MODE_DIRECT` (`0x00`),
`SA_PS_CLK_GEN_MODE_TRIGGERED` (`0x01`)

6.11.5 Clock Generator Frequency

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_FREQUENCY_PROP`	`0x8612`	I32	RW

Description

This property holds the frequency (in mHz) of the clock that is generated. Note that not all frequencies may be generated precisely. When writing this property the system will chose a frequency as close as possible to the desired frequency. Reading this property will yield the actual frequency that is generated (which is not necessarily equal to the frequency that was written to the property).

The default value is 10 000 000 000.

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . 10 000 000 000

6.11.6 Clock Generator Phase

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_PHASE_PROP`	`0x8613`	I32	RW

Description

This property may be used to generate a phase shift of the clock signal. Useful when starting several clock generators synchronously (see *Clock Generator Enable Sync* property). The phase shift is given in degrees. E.g. when having two 1 kHz clocks with one of them having a phase shift of 540° then one clock starts at t = 0 ms and the other one at t = 1.5 ms.

The default value is 0.

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive (including 0).

6.11.7 Clock Generator Start Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_START_TRIGGER_INDEX_PROP`	`0x8614`	I32	RW

Description

This property selects which trigger from the trigger system is used to start the generation of the clock signal. Only relevant in triggered mode (see *Clock Generator Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.11.8 Clock Generator Stop Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_STOP_TRIGGER_INDEX_PROP`	`0x8615`	I32	RW

Description

This property selects which trigger from the trigger system is used to stop the generation of the clock signal. Only relevant in triggered mode (see *Clock Generator Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.11.9 Clock Generator Trigger Auto Reset Mode

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_TRIGGER_AUTO_RESET_MODE_PROP`	`0x8616`	I32	RW

Description

If the clock generator is operated in triggered mode then this property defines the behavior of the clock generator after a stop trigger has been received. If the auto reset mode is enabled then further start triggers will resume the generation of the clock signal. Otherwise the clock generator must be explicitly disabled and enabled again before it listens to a start trigger.

The default value is `SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`),
`SA_PS_TRIGGER_AUTO_RESET_ENABLED` (`0x01`)

6.11.10 Clock Generator State

Definition

C Definition	Code	Type	Access
`SA_PS_CLOCK_GENERATOR_STATE_PROP`	`0x8617`	I32	R

Description

This property holds the current state of the selected clock generator. May be one of the following:

- `SA_PS_CLK_GEN_STATE_STOPPED` (`0x00`)
- `SA_PS_CLK_GEN_STATE_ARMED` (`0x01`)
- `SA_PS_CLK_GEN_STATE_RUNNING` (`0x02`)

Index Parameter

- **Index High**: Clock generator index (zero based).
- **Index Low**: Not used (`0x00`).

6.12 Stream Generator Properties

6.12.1 SG Clock Source

Definition

C Definition	Code	Type	Access
`SA_PS_SG_CLOCK_SOURCE_PROP`	`0x8700`	I32	RW

Description

This property defines which clock the stream generator uses to generate data frames. By default the internal clock is used. Alternatively, the trigger system may be selected which (if configured properly) allows e.g. to feed an external clock. See section 2.4.3 for more information.

The default value is `SA_PS_SG_SOURCE_INTERNAL` (`0x00`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_SG_SOURCE_INTERNAL` (`0x00`),
`SA_PS_SG_SOURCE_TRIGGER` (`0x01`)

6.12.2 SG Clock Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_SG_CLOCK_TRIGGER_INDEX_PROP`	`0x8701`	I32	RW

Description

If the stream generator clock is set to `SA_PS_SG_SOURCE_TRIGGER` (`0x01`) then this property selects the trigger that the stream generator uses as a clock signal. See section 2.4.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots \mathit{TriggerCount} - 1$

6.12.3 SG Trigger Start Index

Definition

C Definition	Code	Type	Access
`SA_PS_SG_TRIGGER_START_INDEX_PROP`	`0x8710`	I32	RW

Description

When using the `SA_SI_TRIGGERED_STREAMING` (`0x02`) mode this property selects which trigger is used as a start trigger. See section 2.4.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots \mathit{TriggerCount} - 1$

6.12.4 SG Trigger Stop Index

Definition

C Definition	Code	Type	Access
`SA_PS_SG_TRIGGER_STOP_INDEX_PROP`	`0x8711`	I32	RW

Description

When using the `SA_SI_TRIGGERED_STREAMING` (`0x02`) mode this property selects which trigger is used as a stop trigger. See section 2.4.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.12.5 SG Trigger Post Frame Count

Definition

C Definition	Code	Type	Access
`SA_PS_SG_TRIGGER_POST_FRAME_COUNT_PROP`	`0x8712`	I32	RW

Description

When using the `SA_SI_TRIGGERED_STREAMING` (`0x02`) mode this property defines how many data frames are additionally generated after the stop trigger has been received. See section 2.4.3 for more information.

The default value is 0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any positive

6.12.6 SG Trigger Auto Reset Mode

Definition

C Definition	Code	Type	Access
`SA_PS_SG_TRIGGER_AUTO_RESET_MODE_PROP`	`0x8713`	I32	RW

Description

When using the `SA_SI_TRIGGERED_STREAMING` (`0x02`) mode this property defines whether or not the stream generator reacts to a further start trigger after a stop trigger was received. By default, the streaming mode must be disabled and then set to triggered streaming again to react to further start triggers. However, when this property is enabled the stream generator will automatically produce further frames when the next start trigger is detected. See section 2.4.3 for more information.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.13 Counter Properties

6.13.1 Counter Enable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_ENABLE_SYNC_PROP`	`0x8900`	I32	W

Description

A bit mask may be written to this property to synchronously enable several counters at once. If bit 0 is set then counter 0 is enabled, if bit 1 is set then counter 1 is enabled and so on.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.13.2 Counter Disable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_DISABLE_SYNC_PROP`	`0x8901`	I32	W

Description

A bit mask may be written to this property to synchronously disable several counters at once. If bit 0 is set then counter 0 is disabled, if bit 1 is set then counter 1 is disabled and so on.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.13.3 Counter Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_ENABLED_PROP`	`0x8910`	I32	RW

Description

This property holds the enabled state of the selected counter. It may be written to enable or disable the counter. Note that to enable or disable several counters at once you must use the *Counter Enable Sync* and *Counter Disable Sync* properties instead.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.13.4 Counter Mode

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_MODE_PROP`	`0x8911`	I32	RW

Description

Selects the operation mode of the counter. In *direct* mode the counter will start to run as soon as it is enabled and stop when it is disabled. In *triggered* mode, after being enabled, the counter is controlled by the trigger system.

The default value is `SA_PS_COUNTER_MODE_DIRECT` (`0x00`).

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_COUNTER_MODE_DIRECT` (`0x00`),
`SA_PS_COUNTER_MODE_TRIGGERED` (`0x01`)

6.13.5 Counter Source

Definition

C Definition	Code	Type	Access
SA_PS_COUNTER_SOURCE_PROP	0x8912	I32	RW

Description

Selects what the counter should count. Either internal clock ticks are counted or trigger events. In the latter case the *Counter Source Trigger Index* property should also be configured.

The default value is SA_PS_COUNTER_SOURCE_INTERNAL (0x00).

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (0x00).

Valid Range

SA_PS_COUNTER_SOURCE_INTERNAL (0x00),
SA_PS_COUNTER_SOURCE_TRIGGER (0x01)

6.13.6 Counter Source Trigger Index

Definition

C Definition	Code	Type	Access
SA_PS_COUNTER_SOURCE_TRIGGER_INDEX_PROP	0x8913	I32	RW

Description

Only relevant if the counter counts trigger events (see *Counter Source* property). Selects the trigger from the trigger system that is used for counting.

The default value is 0.

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . *TriggerCount* − 1

6.13.7 Counter Start Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_START_TRIGGER_INDEX_PROP`	`0x8914`	I32	RW

Description

This property selects which trigger from the trigger system is used to start the counting of the selected source. Only relevant in triggered mode (see *Counter Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . *TriggerCount* − 1

6.13.8 Counter Stop Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_STOP_TRIGGER_INDEX_PROP`	`0x8915`	I32	RW

Description

This property selects which trigger from the trigger system is used to stop the counting of the selected source. Only relevant in triggered mode (see *Counter Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.13.9 Counter Trigger Auto Reset Mode

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_TRIGGER_AUTO_RESET_MODE_PROP`	`0x8916`	I32	RW

Description

If the counter is operated in triggered mode then this property defines the behavior of the counter after a stop trigger has been received. If the auto reset mode is *enabled* then further start triggers will resume the counting. If set to *value* then further start triggers will restart counting from the configured start value of the counter (see *Counter Start Value* property). Otherwise the counter must be explicitly disabled and enabled again before it listens to a start trigger.

The default value is `SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`),
`SA_PS_TRIGGER_AUTO_RESET_ENABLED` (`0x01`),
`SA_PS_TRIGGER_AUTO_RESET_VALUE` (`0x02`)

6.13.10 Counter State

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_STATE_PROP`	`0x8917`	I32	R

Description

This property holds the current state of the selected counter. See section 3.4 for more information.

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

6.13.11 Counter Start Value

Definition

C Definition	Code	Type	Access
`SA_PS_COUNTER_START_VALUE_PROP`	`0x8920`	I64	RW

Description

Holds the start value of the counter. The current value of the counter is set to the start value when a start trigger is received while the counter is operated in triggered mode (see *Counter Mode* property).

Note that writing this property will also set the current value of the counter to the given start value.

The default value is 0.

Index Parameter

- **Index High**: Counter index (0 . . . 1).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 281 474 976 710 655

6.14 Calculation System Properties

6.14.1 CS Stage 0 Operand Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_OPERAND_SELECT_PROP`	`0x8a00`	I32	RW

Description

This property selects which operand is to be used for a stage 0 operation. See section 3.5 for more information.

The default value is `SA_PS_OPERAND_CONST` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 9).

Valid Range

`SA_PS_OPERAND_CONST` (`0x00`),
`SA_PS_OPERAND_POSITION` (`0x01`),
`SA_PS_OPERAND_ADC` (`0x02`),
`SA_PS_OPERAND_ENV_VALUE` (`0x03`),
`SA_PS_OPERAND_COUNTER` (`0x04`),
`SA_PS_OPERAND_VELOCITY` (`0x07`),
`SA_PS_OPERAND_ACCELERATION` (`0x08`),
`SA_PS_OPERAND_SIG_GEN` (`0x09`).

6.14.2 CS Stage 0 Const

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_CONST_PROP`	`0x8a01`	I64	RW

Description

Defines a constant value for a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_CONST` (`0x00`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

Any

6.14.3 CS Stage 0 Position Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_POS_INDEX_SELECT_PROP`	`0x8a02`	I32	RW

Description

Selects which channels position value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_POSITION` (`0x01`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots \mathit{NumberOfChannels} - 1$

6.14.4 CS Stage 0 ADC Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_ADC_INDEX_SELECT_PROP`	`0x8a03`	I32	RW

Description

Selects which ADC value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_ADC` (`0x02`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots \mathit{ADCCount} - 1$

6.14.5 CS Stage 0 Environmental Value Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_ENV_VALUE_SELECT_PROP`	`0x8a04`	I32	RW

Description

Selects which environmental value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_ENV_VALUE` (`0x03`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is `SA_PS_ENV_VALUE_TEMPERATURE` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

`SA_PS_ENV_VALUE_TEMPERATURE` (`0x00`),
`SA_PS_ENV_VALUE_PRESSURE` (`0x01`),
`SA_PS_ENV_VALUE_HUMIDITY` (`0x02`)

6.14.6 CS Stage 0 Counter Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_COUNTER_INDEX_SELECT_PROP`	`0x8a05`	I32	RW

Description

Selects which counter value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_COUNTER` (`0x04`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots CounterCount - 1$

6.14.7 CS Stage 0 Operand Inversion

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_OPERAND_INVERSION_PROP`	`0x8a06`	I32	RW

Description

If this property is enabled then the stage 0 operand is inverted before it is processed by the selected stage 0 operation. This is useful, since stage 0 does not support a minus operator. See section 3.5 for more information.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.14.8 CS Stage 0 Velocity Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_VEL_INDEX_SELECT_PROP`	`0x8a07`	I32	RW

Description

Selects which channels velocity value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_VELOCITY` (`0x07`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots \mathit{NumberOfChannels} - 1$

6.14.9 CS Stage 0 Acceleration Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_ACC_INDEX_SELECT_PROP`	`0x8a08`	I32	RW

Description

Selects which channels acceleration value should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_ACCELERATION` (`0x08`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots \mathit{NumberOfChannels} - 1$

6.14.10 CS Stage 0 Operator Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_OPERATOR_SELECT_PROP`	`0x8a10`	I32	RW

Description

This property selects the stage 0 operation that should be performed on its operands. Note that a minus operation is not supported. Instead, use the operand inversion (see *CS Stage 0 Operand Inversion* property). See section 3.5 for more information.

The default value is `SA_PS_OPERATOR_PLUS` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operator index (0, 1).

Valid Range

`SA_PS_OPERATOR_PLUS` (`0x00`),
`SA_PS_OPERATOR_MAX` (`0x04`),
`SA_PS_OPERATOR_MIN` (`0x05`)

6.14.11 CS Stage 0 Signal Generator Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE0_SIG_GEN_INDEX_SELECT_PROP`	`0x8a11`	I32	RW

Description

Selects which Signal Generator should be used as a stage 0 operand. The *CS Stage 0 Operand Select* property must be set to `SA_PS_OPERAND_SIG_GEN` (`0x09`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0 . . . 7).

Valid Range

$0 \ldots \mathit{SigGenCount} - 1$

6.14.12 CS Stage 1 Operand Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE1_OPERAND_SELECT_PROP`	`0x8a20`	I32	RW

Description

This property selects which operand is to be used for a stage 1 operation. See section 3.5 for more information.

The default value is `SA_PS_OPERAND_CONST` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0, 1).

Valid Range

`SA_PS_OPERAND_CONST` (`0x00`),
`SA_PS_OPERAND_STAGE0` (`0x05`),
`SA_PS_OPERAND_CALC_SYS` (`0x06`)

6.14.13 CS Stage 1 Const

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE1_CONST_PROP`	`0x8a21`	F64	RW

Description

Defines a constant value for a stage 1 operand. The *CS Stage 1 Operand Select* property must be set to `SA_PS_OPERAND_CONST` (`0x00`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0, 1).

Valid Range

Any

6.14.14 CS Stage 1 Calculation System Index Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE1_CALC_SYS_INDEX_SELECT_PROP`	`0x8a22`	I32	RW

Description

Selects which calculation systems result value should be used as a stage 1 operand. The *CS Stage 1 Operand Select* property must be set to `SA_PS_OPERAND_CALC_SYS` (`0x06`) in order for this setting to have an effect. See section 3.5 for more information.

The default value is 0.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operand index (0, 1).

Valid Range

0 . . . 2

6.14.15 CS Stage 1 Operator Select

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_STAGE1_OPERATOR_SELECT_PROP`	`0x8a30`	I32	RW

Description

This property selects the stage 1 operation that should be performed on its operands. See section 3.5 for more information.

The default value is `SA_PS_OPERATOR_PLUS` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Operator index (must be 0).

Valid Range

`SA_PS_OPERATOR_PLUS` (`0x00`),
`SA_PS_OPERATOR_MINUS` (`0x01`),
`SA_PS_OPERATOR_MULT` (`0x02`),
`SA_PS_OPERATOR_DIV` (`0x03`),
`SA_PS_OPERATOR_MAX` (`0x04`),
`SA_PS_OPERATOR_MIN` (`0x05`)

6.14.16 CS Shape Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_SHAPE_ENABLED_PROP`	`0x8a40`	I32	RW

Description

This property enables or disables the shape stage of the calculation system. See section 3.5 for more information.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.14.17 CS Shape

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_SHAPE_PROP`	`0x8a41`	I32	RW

Description

This property selects the shape that should be used to modify the final result of the calculation system. See section 3.5 for more information.

The default value is `SA_PS_CALC_SYS_SHAPE_NONE` (`0x00`).

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_CALC_SYS_SHAPE_NONE` (`0x00`),
`SA_PS_CALC_SYS_SHAPE_CUSTOM0` (`0x80`),
`SA_PS_CALC_SYS_SHAPE_CUSTOM1` (`0x81`),
`SA_PS_CALC_SYS_SHAPE_CUSTOM2` (`0x82`),
`SA_PS_CALC_SYS_SHAPE_CUSTOM3` (`0x83`)

6.14.18 CS Output Label

Definition

C Definition	Code	Type	Access
`SA_PS_CALC_SYS_OUTPUT_LABEL_PROP`	`0x8a42`	String	RW

Description

This property may be used to give a calculation system a generic name that gives a hint of what is calculated. See section 3.5 for more information.

The default value is an empty string.

Index Parameter

- **Index High**: Calculation system index (0 . . . 2).
- **Index Low**: Not used (`0x00`).

Valid Range

The output label is a null terminated string with a maximum length of 64 characters (including the null termination).

6.15 Signal Generator Properties

6.15.1 SigGen Enable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_ENABLE_SYNC_PROP`	`0x8b00`	I32	W

Description

A bit mask may be written to this property to synchronously enable several signal generators at once. If bit 0 is set then signal generator 0 is enabled, if bit 1 is set then signal generator 1 is enabled and so on.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.15.2 SigGen Disable Sync

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_DISABLE_SYNC_PROP`	`0x8b01`	I32	W

Description

A bit mask may be written to this property to synchronously disable several signal generators at once. If bit 0 is set then signal generator 0 is disabled, if bit 1 is set then signal generator 1 is disabled and so on.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

Any

6.15.3 SigGen Enabled

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_ENABLED_PROP`	`0x8b02`	I32	RW

Description

This property holds the enabled state of the selected signal generator. It may be written to enable or disable the signal generator. Note that to enable or disable several signal generators at once you must use the *SigGen Enable Sync* and *SigGen Disable Sync* properties instead.

The default value is `SA_SI_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_DISABLED` (`0x00`),
`SA_SI_ENABLED` (`0x01`)

6.15.4 SigGen Frequency

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_FREQUENCY_PROP`	`0x8b03`	I32	RW

Description

Defines the frequency (in mHz) at which the signal generator should output the selected signal shape. The default value is 1000 (1 Hz).

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

1 . . . 1 000 000 000

6.15.5 SigGen Amplitude

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_AMPLITUDE_PROP`	`0x8b04`	F64	RW

Description

Defines the amplitude with which the selected signal shape should be output (compare figure 3.10).

The default value is 1.0 (full amplitude).

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

0.0 . . . 1.0

6.15.6 SigGen Offset

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_OFFSET_PROP`	`0x8b05`	F64	RW

Description

Defines the offset with which the selected signal shape should be output (compare figure 3.10). The default value is 0.0.

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

$-1.0 \ldots 1.0$

6.15.7 SigGen Phase

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_PHASE_PROP`	`0x8b06`	I32	RW

Description

Defines the phase with which the selected signal shape should be output. Useful when operating several signal generators at once (see also *SigGen Enable Sync* property).

The default value is 0.

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

0 . . . 359

6.15.8 SigGen Shape

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_SHAPE_PROP`	`0x8b08`	I32	RW

Description

Defines the signal shape that should be output on the selected signal generator.

The default value is `SA_PS_SIG_GEN_SHAPE_NONE` (`0x00`).

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_SIG_GEN_SHAPE_NONE` (`0x00`),
`SA_PS_SIG_GEN_SHAPE_SQUARE` (`0x01`),
`SA_PS_SIG_GEN_SHAPE_SIN` (`0x02`),
`SA_PS_SIG_GEN_SHAPE_SAWTOOTH_P` (`0x03`),
`SA_PS_SIG_GEN_SHAPE_SAWTOOTH_N` (`0x04`),
`SA_PS_SIG_GEN_SHAPE_CUSTOM0` (`0x80`),
`SA_PS_SIG_GEN_SHAPE_CUSTOM1` (`0x81`),
`SA_PS_SIG_GEN_SHAPE_CUSTOM2` (`0x82`),
`SA_PS_SIG_GEN_SHAPE_CUSTOM3` (`0x83`),
`SA_PS_SIG_GEN_SHAPE_CUSTOM4` (`0x84`)

6.15.9 SigGen Mode

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_MODE_PROP`	`0x8b09`	I32	RW

Description

Selects the operation mode of the signal generator. In *direct* mode the signal generator will start to run as soon as it is enabled and stop when it is disabled. In *triggered* mode, after being enabled, the signal generator is controlled by the trigger system.

The default value is `SA_PS_SIG_GEN_MODE_DIRECT` (`0x00`).

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_SIG_GEN_MODE_DIRECT` (`0x00`),
`SA_PS_SIG_GEN_MODE_TRIGGERED` (`0x01`)

6.15.10 SigGen Start Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_START_TRIGGER_INDEX_PROP`	`0x8b0a`	I32	RW

Description

This property selects which trigger from the trigger system is used to start the signal generator. Only relevant in triggered mode (see *SigGen Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Signal Generator index ($0 \ldots 4$).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots \mathit{TriggerCount} - 1$

6.15.11 SigGen Stop Trigger Index

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_STOP_TRIGGER_INDEX_PROP`	`0x8b0b`	I32	RW

Description

This property selects which trigger from the trigger system is used to stop the signal generator. Only relevant in triggered mode (see *SigGen Mode* property).

The default value is 0.

Index Parameter

- **Index High**: Signal Generator index ($0 \ldots 4$).
- **Index Low**: Not used (`0x00`).

Valid Range

$0 \ldots TriggerCount - 1$

6.15.12 SigGen Trigger Auto Reset Mode

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_TRIGGER_AUTO_RESET_MODE_PROP`	`0x8b0c`	I32	RW

Description

If the signal generator is operated in triggered mode then this property defines the behavior of the signal generator after a stop trigger has been received. If the auto reset mode is enabled then further start triggers will resume the signal generation. Otherwise the signal generator must be explicitly disabled and enabled again before it listens to a start trigger.

The default value is `SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`).

Index Parameter

- **Index High**: Signal Generator index ($0 \ldots 4$).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_PS_TRIGGER_AUTO_RESET_DISABLED` (`0x00`),
`SA_PS_TRIGGER_AUTO_RESET_ENABLED` (`0x01`)

6.15.13 SigGen State

Definition

C Definition	Code	Type	Access
`SA_PS_SIGNAL_GENERATOR_STATE_PROP`	`0x8b0d`	I32	R

Description

This property holds the current state of the selected signal generator. See section 3.6 for more information.

Index Parameter

- **Index High**: Signal Generator index (0 . . . 4).
- **Index Low**: Not used (`0x00`).

6.16 Break Out Box Properties

6.16.1 BoB Device Serial Number

Definition

C Definition	Code	Type	Access
`SA_PS_BOB_DEVICE_SERIAL_NUMBER_PROP`	`0x8c00`	String	R

Description

This property holds a unique string that identifies an individual Break Out Box device.

The serial number is a null terminated string with a maximum length of 64 characters (including the null termination).

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.16.2 BoB Hardware Version

Definition

C Definition	Code	Type	Access
`SA_PS_BOB_HARDWARE_VERSION_PROP`	`0x8c02`	I32 [5]	R

Description

This property holds the hardware version of the Break Out Box device. The version is an array of five 32-bit words.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.16.3 BoB Hardware Version String

Definition

C Definition	Code	Type	Access
`SA_PS_BOB_HARDWARE_VERSION_STRING_PROP`	`0x8c03`	String	R

Description

This property holds the hardware version of the Break Out Box device as a null terminated string.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

6.16.4 BoB Firmware Version

Definition

C Definition	Code	Type	Access
`SA_PS_BOB_FIRMWARE_VERSION_PROP`	`0x8c04`	I32 [4]	R

Description

This property holds the firmware version of the Break Out Box

The firmware version is an array of four 32-bit words with the following meaning:

Word	Meaning
0	Firmware Version Revision
1	Firmware Version Update
2	Firmware Version Minor
3	Firmware Version Major

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

6.16.5 BoB Firmware Version String

Definition

C Definition	Code	Type	Access
SA_PS_BOB_FIRMWARE_VERSION_STRING_PROP	0x8c05	String	R

Description

This property holds the firmware version of the Break Out Bix as a null terminated string.

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

6.17 Auto Function Properties

6.17.1 Adjustment State

Definition

C Definition	Code	Type	Access
SA_PS_AF_ADJUSTMENT_STATE_PROP	0x9010	I32	RW

Description

Writing this property will enable or disable the mirror adjustment auto function. Reading this property will return the active state of the auto function. See section 2.6.1 for more information.

The default value is SA_PS_ADJUSTMENT_STATE_DISABLED (0x00).

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

SA_PS_ADJUSTMENT_STATE_DISABLED (0x00),
SA_PS_ADJUSTMENT_STATE_MANUAL_ADJUST (0x01),
SA_PS_ADJUSTMENT_STATE_AUTO_ADJUST (0x02)

6.17.2 Adjustment Progress

Definition

C Definition	Code	Type	Access
SA_PS_AF_ADJUSTMENT_PROGRESS_PROP	0x9011	I32	R

Description

This property may be read to query the progress of the mirror adjustment auto function. See section 2.6.1 for more information.

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

6.17.3 Adjustment Signal Control Active

Definition

C Definition	Code	Type	Access
SA_PS_AF_ADJUSTMENT_SIGNAL_CONTROL_ACTIVE_PROP	0x9012	I32	RW

Description

This property enables or disables the signal control during the manual adjustment phase.

The default value is SA_SI_ENABLED (0x01).

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

SA_SI_ENABLED (0x01),
SA_SI_DISABLED (0x00)

6.17.4 Adjustment Autostart Autoadjust Active

Definition

C Definition	Code	Type	Access
SA_PS_AF_ADJUSTMENT_AUTOSTART_AUTOADJUST_ACTIVE_PROP	0x9013	I32	RW

Description

This property, if enabled, causes the manual adjustment to automatically switch to the auto adjust state as soon as laser signal is considered stable. This may be useful if you know that the mirror is already aligned perfectly and no further manual adjustment is necessary.

The default value is SA_SI_DISABLED (0x00).

Index Parameter

- **Index High**: Not used (0x00).
- **Index Low**: Not used (0x00).

Valid Range

SA_SI_ENABLED (0x01),
SA_SI_DISABLED (0x00)

6.18 Library Properties

6.18.1 Stream Buffer Data Type

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMBUFFER_DATA_TYPE_PROP`	`0xf000`	I32	RW

Description

Specifies the data type of the received and decompressed frame values in the stream buffers. This data type can be the same as the data source's data type if that type has 8,16,32,64 bits, or it can be set to a larger data type.

Index Parameter

- **Index High**: *Channel Index* - Selects the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfChannels - 1$.
- **Index Low**: *Data Source Index* - Selects the data source within the channel for which the property is to be accessed. The valid range is $0 \ldots NumberOfDataSources - 1$.

Valid Range

[*DataType*]

6.18.2 Number of Stream Buffers

Definition

C Definition	Code	Type	Access
`SA_SI_NUMBER_OF_STREAMBUFFERS_PROP`	`0xf001`	I32	RW

Description

Specifies the number of buffers used to store incoming stream data.

The default value is 2.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

2 . . . 256

6.18.3 Stream Buffers Interleaved

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMBUFFERS_INTERLEAVED_PROP`	`0xf002`	I32	RW

Description

If enabled, the received frames are stored consecutively in one memory block of a stream buffer. If disabled, the frames are split and the values of each data source are stored in different memory blocks. See section 4.5.3 for more information.

By default, interleaving is enabled.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

`SA_SI_ENABLED` (`0x01`),
`SA_SI_DISABLED` (`0x00`)

6.18.4 Stream Buffer Aggregation

Definition

C Definition	Code	Type	Access
`SA_SI_STREAMBUFFER_AGGREGATION_PROP`	`0xf003`	I32	RW

Description

This property holds the number of frames that are aggregated in a stream buffer. If set to 0, the stream buffer aggregation will be the same as the device's streaming packet aggregation. If it is not 0, it should be larger than the streaming packet aggregation.

The default value is 0.

Index Parameter

- **Index High**: Not used (`0x00`).
- **Index Low**: Not used (`0x00`).

Valid Range

0 or >= 32

7 *EVENT REFERENCE*

7.1 Summary

An event always carries a 32-bit parameter. The meaning of this parameter depends on the event. The last column in the following table indicates the usage of the parameter.

Table 7.1 – Event Summary

Event Name	Code	Parameter
Stream Aborted	`0x0001`	Error Code (reason)
Full Access Connection Lost	`0x8000`	N/A
Beam Interrupt	`0x8010`	Channel Index, State
Overrange	`0x8011`	Comp. Idx, Comp. ID
Overheat	`0x8012`	Comp. Idx, Comp. ID
CalcSys Data Interrupt	`0x8013`	Calc. Sys. Idx, State
Stable State Changed	`0x8100`	State
Channel Enabled State Changed	`0x8101`	Channel Index, State
Channel Valid State Changed	`0x8102`	Channel Index, State
Pilot Laser State Changed	`0x8103`	State
Env Sensor State Changed	`0x8104`	State
Counter State Changed	`0x8105`	Counter Idx
Clock Generator State Changed	`0x8106`	Clock Gen. Idx
Signal Generator State Changed	`0x8107`	State
BoB Connect State Changed	`0x8108`	State
LVDS LS Connect State Changed Event	`0x8109`	State
Auto Function Adjustment Progress	`0x8a80`	Progress, State
Filter Setting Changed Event	`0x8a82`	State
Auto Function Channel Validation Progress	`0x8a83`	Progress
Stream Buffer Ready	`0xf000`	Buffer ID
Stream Stopped	`0xf001`	Reason

7.2 Detailed Event Description

7.2.1 Stream Aborted

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_SI_STREAM_ABORTED_EVENT`	`0x0001`	Error Code (reason)			

Description

This event is generated when the data stream was turned off by the device irregularly due to an error that occured.

Parameter

- **Error Code** (32-bit): An error code that indicates why the stream was aborted (see table A.1).

7.2.2 Full Access Connection Lost

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_FULL_ACCESS_CONNECTION_LOST_EVENT`	`0x8000`	N/A			

Description

This event is generated if the acquired full access connection was terminated, because another interface "stole" it.

Parameter

Not used.

7.2.3 LVDS LS Connect State Changed Event

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_LVDS_LS_CONNECT_STATE_CHANGED_EVENT`	`0x8109`	State			

Description

This event occurs if a device was connected via the LVDS LS interface.

Parameter

- **0x00**: Device is disconnected.
- **0x01**: Device is connected.

7.2.4 Beam Interrupt

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_BEAM_INTERRUPT_EVENT`	`0x8010`	Channel Index		State	

Description

This event is generated when a beam interrupt state change is detected on a channel.

Parameter

- **State** (lower 16-bit): If this value is 1 then the beam is interrupted. Otherwise the beam is not interrupted.
- **Channel Index** (upper 16-bit): Indicates which channels beam was interrupted.

7.2.5 Overrange

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_OVERRANGE_EVENT`	`0x8011`	Comp. Idx		Comp. ID	

Description

This event is generated when an internal overrange in a component is detected. This usually makes the data that the device generates become invalid.

Parameter

- **Component ID** (lower 16-bit): Identifies the component that detected the overrange (see section A.8 for a list of component IDs).
- **Component Index** (upper 16-bit): Identifies the sub component.

7.2.6 Overheat

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_OVERHEAT_EVENT`	`0x8012`	Comp. Idx		Comp. ID	

Description

This event is generates when an internal temperature sensor has detected a critical temperature.

Parameter

- **Component ID** (lower 16-bit): Identifies the component that detected the overheat (see section A.8 for a list of component IDs).
- **Component Index** (upper 16-bit): Identifies the sub component.

7.2.7 CalcSys Data Interrupt

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_CALC_SYS_DATA_INTERRUPT_EVENT`	`0x8013`	Calc. Sys. Idx		State	

Description

This event is generated when at least one operand generates a state change event (i.e. position channel is configured as operand and generates a beam interrupt stage change event).

Parameter

- **Component ID** (lower 16-bit): Identifies the component that detected the overheat (see section A.8 for a list of component IDs).
- **CalcSys Index** (upper 16-bit): Identifies the affected Calculation System index.

7.2.8 Stable State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_STABLE_STATE_CHANGED_EVENT`	`0x8100`	State			

Description

This event is generated when the device's stable state changes from unstable to stable or vice versa. This is also reflected by the Stable LED on the front of the housing.

Parameter

- **State** (32-bit): The new state. A 0 means unstable while a 1 means stable.

7.2.9 Channel Enabled State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_CHANNEL_ENABLED_STATE_CHANGED_EVENT`	`0x8101`	ChIdx		State	

Description

This event is generated whenever a channel is enabled or disabled.

Parameter

- **State** (lower 16-bit): The new state. Will be either `SA_SI_ENABLED` (`0x01`) or `SA_SI_DISABLED` (`0x00`).
- **ChIdx** (upper 16-bit): The affected channel index.

7.2.10 Channel Valid State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_CHANNEL_VALID_STATE_CHANGED_EVENT`	`0x8102`	ChIdx		State	

Description

This event is generated when the channel's valid state changes from invalid to valid or vice versa. This is also reflected by the channel LEDs on the front of the housing.

Parameter

- **State** (lower 16-bit): The new state. A 0 means invalid while a 1 means valid.
- **ChIdx** (upper 16-bit): The affected channel index.

7.2.11 Pilot Laser State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_PILOT_LASER_STATE_CHANGED_EVENT`	`0x8103`	State			

Description

This event is generated when the pilot laser was switched on or off (either programatically or via the pilot laser button on the front of the housing).

Parameter

- **State** (32-bit): The new state. Will be either `SA_SI_ENABLED` (`0x01`) or `SA_SI_DISABLED` (`0x00`).

7.2.12 Env Sensor State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_ENV_SENSOR_STATE_CHANGED_EVENT`	`0x8104`	State			

Description

This event is generated whenever the state of the environmental sensor changes.

The state is a bit mask with the following meaning:

Bit	Meaning
0	Temperature error. If this bit is set then the temperature sensor failed to supply valid data.
1	Humidity error. If this bit is set then the humidity sensor failed to supply valid data.
2	Pressure error. If this bit is set then the pressure sensor failed to supply valid data.
3	Sensor offline. If this bit is set then the environmental sensor module is offline (detached).

Parameter

- **State** (32-bit): The new state (see above).

7.2.13 Counter State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_COUNTER_STATE_CHANGED_EVENT`	`0x8105`	Cnt. Idx		Reserved	

Description

This event is generated whenever the state of a counter changes. The *Counter State* property may be used to poll the new state.

Parameter

- **Cnt. Idx** (upper 16-bit): The index of the counter that made the state transition.

7.2.14 Clock Generator State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_CLOCK_GEN_STATE_CHANGED_EVENT`	`0x8106`	ClkGen Idx		Reserved	

Description

This event is generated whenever the state of a clock generator changes. The *Clock Generator State* property may be used to poll the new state.

Parameter

- **ClkGen Idx** (upper 16-bit): The index of the clock generator that made the state transition.

7.2.15 Signal Generator State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_SIG_GEN_STATE_CHANGED_EVENT`	`0x8107`	SigGen Idx		Reserved	

Description

This event is generated whenever the state of a signal generator changes. The *SigGen State* property may be used to poll the new state.

Parameter

- **SigGen Idx** (upper 16-bit): The index of the signal generator that made the state transition.

7.2.16 BoB Connect State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_BOB_CONNECT_STATE_CHANGED_EVENT`	`0x8108`	State			

Description

This event is generated whenever a Break-out-Box was connected to or disconnected from the device.

Parameter

- **State** (32-bit): The new state. Will be 1 on a connect and 0 on a disconnect.

7.2.17 LVDS LS Connect State Changed

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_LVDS_LS_CONNECT_STATE_CHANGED_EVENT`	`0x8109`	State			

Description

This event is generated whenever a device was connected to or disconnected from the LVDS LS interface.

Parameter

- **State** (32-bit): The new state. Will be 1 on a connect and 0 on a disconnect.

7.2.18 Auto Function Adjustment Progress

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_AF_ADJUSTMENT_PROGRESS_EVENT`	`0x8a80`	Progress		State	

Description

This event is generated when the progress of the adjustment auto function is updated.

Parameter

- **State** (lower 16-bit): Indicates the current state the auto function is in. Will be either `SA_PS_ADJUSTMENT_STATE_DISABLED` (`0x00`), `SA_PS_ADJUSTMENT_STATE_MANUAL_ADJUST` (`0x01`) or `SA_PS_ADJUSTMENT_STATE_AUTO_ADJUST` (`0x02`).
- **Progress** (upper 16-bit): Indicates the overall progress of the auto function as a value between 0 . . . 1000 (permille).

7.2.19 Filter Setting Changed Event

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_FILTER_SETTING_CHANGED_EVENT`	`0x8a82`	Filter Setting			

Description

This event is generated when the filter settings have changed.

Parameter

- **State**: Indicates the exponent of the new filter setting n.

$$\frac{(10MHz)}{2^n} \tag{7.1}$$

7.2.20 Auto Function Channel Validation Progress

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_PS_AF_CHANNEL_VALIDATION_PROGRESS_EVENT`	`0x8a83`	State			

Description

This event is generated when the channel validation auto function starts or stops.

Parameter

- **State** (lower 16-bit): Indicates the current state the auto function is in. Will be either 1 for started and 0 for stopped.

7.2.21 Stream Buffer Ready

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_SI_STREAMBUFFER_READY_EVENT`	`0xf000`	Buffer ID			

Description

This event is generated whenever a stream buffer is ready to be read out by the application.

Parameter

- **Buffer ID** (32-bit): The ID of the buffer that may be read.

7.2.22 Stream Stopped

Definition

C Definition	Code	31-24	23-16	15-8	7-0
`SA_SI_STREAM_STOPPED_EVENT`	`0xf001`	Reserved		Reason	

Description

This event indicates that the data stream was terminated.

Parameter

- **Reason** (16-bit): Reason for the stopped stream (see table A.36).

A CODE DEFINITION REFERENCE

A.1 Error Codes

Table A.1 – Error Codes

Code	C-Definition / Description
`0x0000`	`SA_SI_OK` No error occurred. Corresponds to an acknowledge.
`0x0001`	`SA_SI_UNKNOWN_COMMAND_ERROR` An unknown command opcode was received and the packet was dropped.
`0x0002`	`SA_SI_PACKET_SIZE_ERROR` Indicates that the size field of a packet does not match the packet structure.
`0x0004`	`SA_SI_TIMEOUT_ERROR` A timeout occurred while receiving a complete packet.
`0x0005`	`SA_SI_PROTOCOL_ERROR` A packet was received that does not comply to the protocol.
`0x000c`	`SA_SI_BUFFER_UNDERFLOW_ERROR` The targeted buffer is empty (e.g. error queue).
`0x000d`	`SA_SI_BUFFER_OVERFLOW_ERROR` The targeted Buffer is filled and has no more space for further data.
`0x0010`	`SA_SI_INVALID_PACKET_ERROR` A packet with an inconsistent structure was received.
`0x0011`	`SA_SI_INVALID_STREAM_PACKET_ERROR` A stream packet with an inconsistent structure was received.
`0x0012`	`SA_SI_INVALID_PROPERTY_ERROR` The given property name could not be resolved.
`0x0013`	`SA_SI_INVALID_PARAMETER_ERROR` The passed parameter is not in the valid range.
`0x0014`	`SA_SI_INVALID_CHANNEL_INDEX_ERROR` An invalid channel index has been passed.
`0x0015`	`SA_SI_INVALID_DSOURCE_INDEX_ERROR` An invalid data source index has been passed.
`0x0016`	`SA_SI_INVALID_DATA_TYPE_ERROR` Indicates that the data type of a parameter is invalid.

Continued on next page

Table A.1 – *Continued from previous page*

Code	C-Definition / Description
0x001f	SA_SI_PERMISSION_DENIED_ERROR The request cannot be processed due to an access violation.
0x0020	SA_SI_NO_DATA_SOURCES_ENABLED_ERROR Indicates that no data source is enabled. To activate a data stream at least one data source must be enabled.
0x0021	SA_SI_STREAMING_ACTIVE_ERROR This error indicates that the streaming is running due it should be reconfigured. The data stream can only be reconfigured when the device does currently not stream any data.
0x0022	SA_SI_DATA_SOURCE_NOT_STREAMABLE_ERROR The selected data source cannot be streamed and therefore cannot be enabled (or disabled) for streaming.
0x0030	SA_SI_UNKNOWN_DATA_OBJECT_ERROR A data object name was specified that is unknown to the device.
0x00ff	SA_SI_COMMAND_NOT_PROCESSABLE_ERROR The given command cannot be processed in the current device state.
0x7ffd	SA_SI_FEATURE_NOT_SUPPORTED_ERROR Indicates that a requested feature is not available on the given device.
0x7ffe	SA_SI_NOT_IMPLEMENTED_ERROR Indicates that a feature is not yet implemented. The device may have to be updated to a newer version.
0x7fff	SA_SI_OTHER_ERROR An error which can not be categorized.
0x8000	SA_PS_REQUEST_DENIED_ERROR A request was denied.
0x8001	SA_PS_INTERNAL_COMMUNICATION_ERROR An internal communication error.
0x8002	SA_PS_NO_FULL_ACCESS_ERROR A property that requires a full access connection was tried to be modified, but the current connection is a limited access connection.
0x8200	SA_PS_WORKING_DISTANCE_NOT_SET_ERROR An operation could not be executed, because it is required to set the working distance first.
0xf000	SA_SI_DEVICE_LIMIT_ERROR
0xf001	SA_SI_INVALID_LOCATOR_ERROR
0xf002	SA_SI_INITIALIZATION_ERROR

Continued on next page

Table A.1 – *Continued from previous page*

Code	C-Definition / Description
`0xf003`	`SA_SI_NOT_INITIALIZED_ERROR`
`0xf004`	`SA_SI_COMMUNICATION_ERROR`
`0xf006`	`SA_SI_QUERYBUFFER_SIZE_ERROR`
`0xf007`	`SA_SI_INVALID_HANDLE_ERROR`
`0xf008`	`SA_SI_DATA_SOURCE_ENABLED_ERROR`
`0xf009`	`SA_SI_INVALID_STREAMBUFFER_ID_ERROR`
`0xf00a`	`SA_SI_STREAM_SEQUENCE_ERROR`
`0xf00b`	`SA_SI_NO_DATABUFFER_AVAILABLE_ERROR`
`0xf00d`	`SA_SI_STREAMBUFFER_NOT_ACQUIRED_ERROR`
`0xf00f`	`SA_SI_UNEXPECTED_PACKET_RECEIVED_ERROR`
`0xf010`	`SA_SI_CANCELED_ERROR`
`0xf012`	`SA_SI_BUFFER_INTERLEAVING_ERROR`
`0xf013`	`SA_SI_DRIVER_ERROR`
`0xf014`	`SA_SI_DATA_OBJECT_BUSY_ERROR`

A.2 Event Types

Table A.2 – Event Types

Code	C-Definition
0x0001	SA_SI_STREAM_ABORTED_EVENT
0x8000	SA_PS_FULL_ACCESS_CONNECTION_LOST_EVENT
0x8010	SA_PS_BEAM_INTERRUPT_EVENT
0x8011	SA_PS_OVERRANGE_EVENT
0x8012	SA_PS_OVERHEAT_EVENT
0x8013	SA_PS_CALC_SYS_DATA_INTERRUPT_EVENT
0x8100	SA_PS_STABLE_STATE_CHANGED_EVENT
0x8101	SA_PS_CHANNEL_ENABLED_STATE_CHANGED_EVENT
0x8102	SA_PS_CHANNEL_VALID_STATE_CHANGED_EVENT
0x8103	SA_PS_PILOT_LASER_STATE_CHANGED_EVENT
0x8104	SA_PS_ENV_SENSOR_STATE_CHANGED_EVENT
0x8105	SA_PS_COUNTER_STATE_CHANGED_EVENT
0x8106	SA_PS_CLOCK_GEN_STATE_CHANGED_EVENT
0x8107	SA_PS_SIG_GEN_STATE_CHANGED_EVENT
0x8108	SA_PS_BOB_CONNECT_STATE_CHANGED_EVENT
0x8109	SA_PS_LVDS_LS_CONNECT_STATE_CHANGED_EVENT
0x8a80	SA_PS_AF_ADJUSTMENT_PROGRESS_EVENT
0x8a82	SA_PS_FILTER_SETTING_CHANGED_EVENT
0x8a83	SA_PS_AF_CHANNEL_VALIDATION_PROGRESS_EVENT
0xf000	SA_SI_STREAMBUFFER_READY_EVENT
0xf001	SA_SI_STREAM_STOPPED_EVENT

A.3 Data Types

Table A.3 – Data Types

Code	C-Definition
0x00	SA_SI_INT8_DTYPE
0x01	SA_SI_UINT8_DTYPE
0x02	SA_SI_INT16_DTYPE
0x03	SA_SI_UINT16_DTYPE
0x06	SA_SI_INT32_DTYPE

Continued on next page

Table A.3 – *Continued from previous page*

Code	C-Definition
0x07	SA_SI_UINT32_DTYPE
0x0a	SA_SI_INT48_DTYPE
0x0b	SA_SI_UINT48_DTYPE
0x0e	SA_SI_INT64_DTYPE
0x0f	SA_SI_UINT64_DTYPE
0x10	SA_SI_FLOAT32_DTYPE
0x11	SA_SI_FLOAT64_DTYPE
0x12	SA_SI_STRING_DTYPE

A.4 Data Source Types

Table A.4 – Data Source Types

Code	C-Definition
0x0000	SA_SI_ANALOG_RAW_DSOURCE
0x0001	SA_SI_SIN_RAW_DSOURCE
0x0002	SA_SI_COS_RAW_DSOURCE
0x0003	SA_SI_SIN_QUALITY_DSOURCE
0x0004	SA_SI_COS_QUALITY_DSOURCE
0x0005	SA_SI_SIN_CORRECTED_DSOURCE
0x0006	SA_SI_COS_CORRECTED_DSOURCE
0x0008	SA_SI_POSITION_DSOURCE
0x0009	SA_SI_STATUS_DSOURCE
0x000a	SA_SI_TEMPERATURE_DSOURCE
0x000b	SA_SI_HUMIDITY_DSOURCE
0x000c	SA_SI_PRESSURE_DSOURCE
0x000d	SA_SI_VELOCITY_DSOURCE
0x000e	SA_SI_ACCELERATION_DSOURCE
0x000f	SA_SI_COUNTER_DSOURCE
0x0010	SA_SI_GENERIC_DSOURCE

A.5 Data Source Compression Modes

Table A.5 – Data Source Compression Modes

Code	C-Definition
0x0000	SA_SI_NO_COMPRESSION_MODE
0x0002	SA_SI_LOWER16ABS_COMPRESSION_MODE

A.6 Base Unit Types

Table A.6 – Base Unit Types

Code	C-Definition
0x0000	SA_SI_NO_UNIT
0x0001	SA_SI_PERCENT_UNIT
0x0002	SA_SI_METRE_UNIT
0x0003	SA_SI_DEGREE_UNIT
0x0004	SA_SI_SECOND_UNIT
0x0005	SA_SI_HERTZ_UNIT
0x0006	SA_SI_KILOGRAM_UNIT
0x0007	SA_SI_NEWTON_UNIT
0x0008	SA_SI_WATT_UNIT
0x0009	SA_SI_JOULE_UNIT
0x000a	SA_SI_VOLT_UNIT
0x000b	SA_SI_AMPERE_UNIT
0x000c	SA_SI_OHM_UNIT
0x000d	SA_SI_PASCAL_UNIT
0x000e	SA_SI_KELVIN_UNIT
0x000f	SA_SI_DEGREE_CELSIUS_UNIT
0x0010	SA_SI_SQUARE_METRE_UNIT
0x0011	SA_SI_METRE_PER_SECOND_UNIT
0x0012	SA_SI_METRE_PER_SQUARE_SECOND_UNIT

A.7 Streaming Modes

Table A.7 – Streaming Modes

Code	C-Definition
0x00	SA_SI_NO_STREAMING

Continued on next page

Table A.7 – *Continued from previous page*

Code	C-Definition
0x01	SA_SI_DIRECT_STREAMING
0x02	SA_SI_TRIGGERED_STREAMING

A.8 Component IDs

Table A.8 – Component IDs

Code	C-Definition
0x8000	SA_PS_POWER_SUPPLY_CID
0x8001	SA_PS_CPU_CID
0x8002	SA_PS_FPGA_CID
0x8003	SA_PS_IR_LASER_CID
0x8100	SA_PS_GPIO_ADC_CID
0x8110	SA_PS_WLS_ADC_CID
0x8112	SA_PS_WLS_PREAMP_CID
0x8200	SA_PS_CHANNEL_CID
0x8201	SA_PS_CHANNEL_PREAMP_CID
0x8202	SA_PS_CHANNEL_SOMEGA_CID
0x8203	SA_PS_CHANNEL_S2OMEGA_CID
0x8210	SA_PS_CALC_SYS_CID
0x8310	SA_PS_ENV_SENSOR_CID
0x8400	SA_PS_COUNTER_CID

A.9 Environmental Module

A.9.1 Operation Mode Selectors

Table A.9 – Environmental Module: Operation Mode Selectors

Code	C-Definition
0x00	SA_PS_ENV_OP_MODE_MANUAL
0x01	SA_PS_ENV_OP_MODE_AUTOMATIC

A.9.2 Value Selectors

Table A.10 – Environmental Module: Value Selectors

Code	C-Definition
0x00	SA_PS_ENV_VALUE_TEMPERATURE
0x01	SA_PS_ENV_VALUE_PRESSURE
0x02	SA_PS_ENV_VALUE_HUMIDITY

A.10 General Purpose I/O

A.10.1 DAC Source Selectors

Table A.11 – GPIO DAC Source Selectors

Code	C-Definition
0x00	SA_PS_GPIO_DAC_SOURCE_CONST
0x01	SA_PS_GPIO_DAC_SOURCE_CALCULATION_SYSTEM
0x02	SA_PS_GPIO_DAC_SOURCE_SIGNAL_GENERATOR

A.10.2 Digital Source Selectors

Table A.12 – GPIO Digital Source Selectors

Code	C-Definition
0x00	SA_PS_GPIO_DIG_SOURCE_CONST
0x01	SA_PS_GPIO_DIG_SOURCE_TRIGGER
0x02	SA_PS_GPIO_DIG_SOURCE_CLOCK_GENERATOR

A.11 Trigger System

A.11.1 Trigger Source Event Selectors

Table A.13 – Trigger Source Event Selectors

Code	C-Definition
0x00	SA_PS_TRIG_EVENT_NONE
0x01	SA_PS_TRIG_EVENT_SOFTWARE
0x02	SA_PS_TRIG_EVENT_DATA_SOURCE_VALUE
0x03	SA_PS_TRIG_EVENT_DATA_SOURCE_INCREMENT
0x04	SA_PS_TRIG_EVENT_GPIO_TRIGGER

Continued on next page

Table A.13 – *Continued from previous page*

Code	C-Definition
0x05	SA_PS_TRIG_EVENT_EXTERNAL_TRIGGER
0x06	SA_PS_TRIG_EVENT_INTERNAL

A.11.2 Trigger Source Condition Selectors

Table A.14 – Trigger Source Condition Selectors

Code	C-Definition
0x00	SA_PS_TRIG_CONDITION_RISING
0x01	SA_PS_TRIG_CONDITION_FALLING
0x02	SA_PS_TRIG_CONDITION_EITHER
0x03	SA_PS_TRIG_CONDITION_POSITIVE_LEVEL
0x04	SA_PS_TRIG_CONDITION_NEGATIVE_LEVEL
0x05	SA_PS_TRIG_CONDITION_POSITIVE_RANGE
0x06	SA_PS_TRIG_CONDITION_NEGATIVE_RANGE

A.11.3 Logic Operation Selectors

Table A.15 – Logic Operation Selectors

Code	C-Definition
0x00	SA_PS_LOGIC_OP_NONE
0x01	SA_PS_LOGIC_OP_OR
0x02	SA_PS_LOGIC_OP_NOR
0x03	SA_PS_LOGIC_OP_AND
0x04	SA_PS_LOGIC_OP_NAND
0x05	SA_PS_LOGIC_OP_XOR
0x06	SA_PS_LOGIC_OP_NXOR

A.11.4 Trigger Output Mode Selectors

Table A.16 – Trigger Output Mode Selectors

Code	C-Definition
0x00	SA_PS_TRIGGER_OUTPUT_MODE_IMMEDIATE_NO_RESET
0x01	SA_PS_TRIGGER_OUTPUT_MODE_IMMEDIATE_RESET

Continued on next page

Table A.16 – *Continued from previous page*

Code	C-Definition
0x02	SA_PS_TRIGGER_OUTPUT_MODE_DELAYED_NO_RESET
0x03	SA_PS_TRIGGER_OUTPUT_MODE_DELAYED_RESET

A.11.5 Internal Event Selectors

Table A.17 – Internal Event Selectors

Code	C-Definition
0x00	SA_PS_INTERNAL_EVENT_BEAM_INTERRUPT
0x01	SA_PS_INTERNAL_EVENT_OVER_RANGE

A.12 Digital Differential Interface

A.12.1 Mode Selectors

Table A.18 – Mode Selectors

Code	C-Definition
0x00	SA_PS_DDI_MODE_QUADRATURE
0x01	SA_PS_DDI_MODE_SERIAL_DATA

A.12.2 Source Selectors

Table A.19 – Source Selectors

Code	C-Definition
0x00	SA_PS_DDI_SOURCE_POSITION
0x01	SA_PS_DDI_SOURCE_CALC_SYS

A.12.3 Serial Clock Mode Selectors

Table A.20 – Serial Clock Mode Selectors

Code	C-Definition
0x00	SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL0_CPHA0
0x01	SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL0_CPHA1
0x02	SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL1_CPHA0
0x03	SA_PS_DDI_SERIAL_CLOCK_MODE_CPOL1_CPHA1

A.12.4 Serial Data Polarity Selectors

Table A.21 – Serial Data Polarity Selectors

Code	C-Definition
0x00	SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_LOW
0x01	SA_PS_DDI_SERIAL_DATA_IDLE_POLARITY_HIGH

A.13 Clock Generator

A.13.1 Mode Selectors

Table A.22 – Mode Selectors

Code	C-Definition
0x00	SA_PS_CLK_GEN_MODE_DIRECT
0x01	SA_PS_CLK_GEN_MODE_TRIGGERED

A.13.2 States

Table A.23 – States

Code	C-Definition
0x00	SA_PS_CLK_GEN_STATE_STOPPED
0x01	SA_PS_CLK_GEN_STATE_ARMED
0x02	SA_PS_CLK_GEN_STATE_RUNNING

A.14 Stream Generator

A.14.1 Clock Sources

Table A.24 – Clock Sources

Code	C-Definition
0x00	SA_PS_SG_SOURCE_INTERNAL
0x01	SA_PS_SG_SOURCE_TRIGGER

A.15 Counter

A.15.1 Mode Selectors

Table A.25 – Mode Selectors

Code	C-Definition
0x00	SA_PS_COUNTER_MODE_DIRECT
0x01	SA_PS_COUNTER_MODE_TRIGGERED

A.15.2 Source Selectors

Table A.26 – Source Selectors

Code	C-Definition
0x00	SA_PS_COUNTER_SOURCE_INTERNAL
0x01	SA_PS_COUNTER_SOURCE_TRIGGER

A.15.3 States

Table A.27 – States

Code	C-Definition
0x00	SA_PS_COUNTER_STATE_STOPPED
0x01	SA_PS_COUNTER_STATE_ARMED
0x02	SA_PS_COUNTER_STATE_RUNNING

A.16 Calculation System

A.16.1 Operand Selectors

Table A.28 – Operand Selectors

Code	C-Definition
0x00	SA_PS_OPERAND_CONST
0x01	SA_PS_OPERAND_POSITION
0x02	SA_PS_OPERAND_ADC
0x03	SA_PS_OPERAND_ENV_VALUE
0x04	SA_PS_OPERAND_COUNTER
0x05	SA_PS_OPERAND_STAGE0
0x06	SA_PS_OPERAND_CALC_SYS
0x07	SA_PS_OPERAND_VELOCITY
0x08	SA_PS_OPERAND_ACCELERATION
0x09	SA_PS_OPERAND_SIG_GEN

A.16.2 Operator Selectors

Table A.29 – Operator Selectors

Code	C-Definition
0x00	SA_PS_OPERATOR_PLUS
0x01	SA_PS_OPERATOR_MINUS
0x02	SA_PS_OPERATOR_MULT
0x03	SA_PS_OPERATOR_DIV
0x04	SA_PS_OPERATOR_MAX
0x05	SA_PS_OPERATOR_MIN

A.16.3 Shape Selectors

Table A.30 – Shape Selectors

Code	C-Definition
0x00	SA_PS_CALC_SYS_SHAPE_NONE
0x80	SA_PS_CALC_SYS_SHAPE_CUSTOM0
0x81	SA_PS_CALC_SYS_SHAPE_CUSTOM1
0x82	SA_PS_CALC_SYS_SHAPE_CUSTOM2
0x83	SA_PS_CALC_SYS_SHAPE_CUSTOM3

A.17 Signal Generator

A.17.1 Mode Selectors

Table A.31 – Mode Selectors

Code	C-Definition
0x00	SA_PS_SIG_GEN_MODE_DIRECT
0x01	SA_PS_SIG_GEN_MODE_TRIGGERED

A.17.2 States

Table A.32 – States

Code	C-Definition
0x00	SA_PS_SIG_GEN_STATE_STOPPED

Continued on next page

Table A.32 – *Continued from previous page*

Code	C-Definition
0x01	SA_PS_SIG_GEN_STATE_ARMED
0x02	SA_PS_SIG_GEN_STATE_RUNNING

A.17.3 Shape Selectors

Table A.33 – Shape Selectors

Code	C-Definition
0x00	SA_PS_SIG_GEN_SHAPE_NONE
0x01	SA_PS_SIG_GEN_SHAPE_SQUARE
0x02	SA_PS_SIG_GEN_SHAPE_SIN
0x03	SA_PS_SIG_GEN_SHAPE_SAWTOOTH_P
0x04	SA_PS_SIG_GEN_SHAPE_SAWTOOTH_N
0x80	SA_PS_SIG_GEN_SHAPE_CUSTOM0
0x81	SA_PS_SIG_GEN_SHAPE_CUSTOM1
0x82	SA_PS_SIG_GEN_SHAPE_CUSTOM2
0x83	SA_PS_SIG_GEN_SHAPE_CUSTOM3
0x84	SA_PS_SIG_GEN_SHAPE_CUSTOM4

A.18 Auto Functions

A.18.1 Adjustment State Selectors

Table A.34 – Adjustment State Selectors

Code	C-Definition
0x00	SA_PS_ADJUSTMENT_STATE_DISABLED
0x01	SA_PS_ADJUSTMENT_STATE_MANUAL_ADJUST
0x02	SA_PS_ADJUSTMENT_STATE_AUTO_ADJUST

A.19 Auto Trigger Reset Mode Selectors

Continued on next page

Table A.35 – *Continued from previous page*

Table A.35 – Auto Trigger Reset Mode Selectors

Code	C-Definition
0x00	SA_PS_TRIGGER_AUTO_RESET_DISABLED
0x01	SA_PS_TRIGGER_AUTO_RESET_ENABLED
0x02	SA_PS_TRIGGER_AUTO_RESET_VALUE

A.20 Stream Stopped Reasons

Table A.36 – Stream Stopped Reasons

Code	C-Definition
0x0001	SA_SI_STOPPED_BY_USER
0x0002	SA_SI_STOPPED_BY_TRIGGER
0x00f1	SA_SI_STOPPED_BY_BUFFER_OVERFLOW

B *DATA OBJECTS*

The following table summarizes the available data objects that the PicoScale offers.

Table B.1 – Data Objects

Name	Format	Total Size (bytes)	Access
`SystemLog`	Text file	varies	R
`SigGenCustomShape0`	Array of 4096 double values	32768	RW
`SigGenCustomShape1`	Array of 4096 double values	32768	RW
`SigGenCustomShape2`	Array of 4096 double values	32768	RW
`SigGenCustomShape3`	Array of 4096 double values	32768	RW
`SigGenCustomShape4`	Array of 4096 double values	32768	RW
`CalcSysCustomShape0`	Array of 4096 double values	32768	RW
`CalcSysCustomShape1`	Array of 4096 double values	32768	RW
`CalcSysCustomShape2`	Array of 4096 double values	32768	RW
`CalcSysCustomShape3`	Array of 4096 double values	32768	RW
`SystemConfiguration`	Text file (XML)	varies	RW

B.1 System Log

Object Name	Access
`SystemLog`	R

The System Log is a text file that contains information about system events that have occurred. For example, firmware updates are logged and may contain useful information in case an update failed.

B.2 Signal Generator Custom Shapes

Object Name	Access
`SigGenCustomShape0` ... `SigGenCustomShape4`	RW

The custom shape of a signal generator is a look-up table with 4096 entries. Each entry is a 64-bit floating point value (`double`) in the range of $[-1 \ldots 1]$. Therefore, the total size of one of these objects is 32768 bytes.

Each 8 byte value is stored in LSB format in the 32 kB data block.

B.3 Calculation System Custom Shapes

Object Name	Access
`CalcSysCustomShape0` ... `CalcSysCustomShape3`	RW

The custom shape of a calculation system shape stage is a look-up table with 4096 entries. Each entry is a 64-bit floating point value (`double`). Therefore, the total size of one of these objects is 32768 bytes.

Each 8 byte value is stored in LSB format in the 32 kB data block.

B.4 System Configuration

Object Name	Access
`SystemConfiguration`	RW

The System Configuration is a XML text file that contains property settings to memorize a system state. The intended use is to save system configurations and exchange them between PicoScale devices.

C DATA SOURCES

The following table summarizes the available data sources that the PicoScale offers.

Table C.1 – Data Sources

Ch Idx	DSrc Idx	Data Source	Data Source Type	Comp ID	Comp Idx
0	0	Position	SA_SI_POSITION_DSOURCE	SA_PS_CHANNEL_CID	0
0	1	Velocity	SA_SI_VELOCITY_DSOURCE	SA_PS_CHANNEL_CID	0
0	2	Acceleration	SA_SI_ACCELERATION_DSOURCE	SA_PS_CHANNEL_CID	0
0	3	SωRaw	SA_SI_SIN_RAW_DSOURCE	SA_PS_CHANNEL_CID	0
0	4	S2ωRaw	SA_SI_COS_RAW_DSOURCE	SA_PS_CHANNEL_CID	0
0	5	Sω	SA_SI_SIN_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	0
0	6	S2ω	SA_SI_COS_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	0
0	7	SωQuality*	SA_SI_SIN_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	0
0	8	S2ωQuality*	SA_SI_COS_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	0
0	9	Counter 0	SA_SI_COUNTER_DSOURCE	SA_PS_COUNTER_CID	0
0	10	Counter 1	SA_SI_COUNTER_DSOURCE	SA_PS_COUNTER_CID	1
0	11	Env Temp	SA_SI_TEMPERATURE_DSOURCE	SA_PS_ENV_SENSOR_CID	0
0	12	Env Humidity	SA_SI_HUMIDITY_DSOURCE	SA_PS_ENV_SENSOR_CID	0
0	13	Env Pressure	SA_SI_PRESSURE_DSOURCE	SA_PS_ENV_SENSOR_CID	0
0	14	GPIO ADC 0	SA_SI_ANALOG_RAW_DSOURCE	SA_PS_GPIO_ADC_CID	0
0	15	GPIO ADC 1	SA_SI_ANALOG_RAW_DSOURCE	SA_PS_GPIO_ADC_CID	1
0	16	GPIO ADC 2	SA_SI_ANALOG_RAW_DSOURCE	SA_PS_GPIO_ADC_CID	2
0	17	Calc Sys 0	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	0
0	18	Calc Sys 1	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	1
0	19	Calc Sys 2	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	2
0	19	Calc Sys 3	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	3
0	19	Calc Sys 4	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	4
0	19	Calc Sys 5	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	5
0	19	Calc Sys 6	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	6

Continued on next page

Table C.1 – *Continued from previous page*

Ch Idx	DSrc Idx	Data Source	Data Source Type	Comp ID	Comp Idx
0	19	Calc Sys 7	SA_SI_GENERIC_DSOURCE	SA_PS_CALC_SYS_CID	7
1	0	Position	SA_SI_POSITION_DSOURCE	SA_PS_CHANNEL_CID	1
1	1	Velocity	SA_SI_VELOCITY_DSOURCE	SA_PS_CHANNEL_CID	1
1	2	Acceleration	SA_SI_ACCELERATION_DSOURCE	SA_PS_CHANNEL_CID	1
1	3	SωRaw	SA_SI_SIN_RAW_DSOURCE	SA_PS_CHANNEL_CID	1
1	4	S2ωRaw	SA_SI_COS_RAW_DSOURCE	SA_PS_CHANNEL_CID	1
1	5	Sω	SA_SI_SIN_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	1
1	6	S2ω	SA_SI_COS_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	1
1	7	SωQuality*	SA_SI_SIN_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	1
1	8	S2ωQuality*	SA_SI_COS_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	1
2	0	Position	SA_SI_POSITION_DSOURCE	SA_PS_CHANNEL_CID	2
2	1	Velocity	SA_SI_VELOCITY_DSOURCE	SA_PS_CHANNEL_CID	2
2	2	Acceleration	SA_SI_ACCELERATION_DSOURCE	SA_PS_CHANNEL_CID	2
2	3	SωRaw	SA_SI_SIN_RAW_DSOURCE	SA_PS_CHANNEL_CID	2
2	4	S2ωRaw	SA_SI_COS_RAW_DSOURCE	SA_PS_CHANNEL_CID	2
2	5	Sω	SA_SI_SIN_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	2
2	6	S2ω	SA_SI_COS_CORRECTED_DSOURCE	SA_PS_CHANNEL_CID	2
2	7	SωQuality*	SA_SI_SIN_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	2
2	8	S2ωQuality*	SA_SI_COS_QUALITY_DSOURCE	SA_PS_CHANNEL_CID	2

* These data sources cannot be streamed. Their values may only be polled.

Sales partner / Contacts

Headquarters

SmarAct GmbH

Schuette-Lanz-Strasse 9
26135 Oldenburg
Germany

T: +49 441 – 800 87 90
Email: info@smaract.com
www.smaract.com

France

SmarAct GmbH

Schuette-Lanz-Strasse 9
26135 Oldenburg
Germany

T: +49 441 – 80 08 79 956
Email: nicoul@smaract.com
www.smaract.com

Israel

Trico Israel Ltd.

P.O.Box 6172
46150 Herzeliya
Israel

T: +972 9 – 950 60 74

www.trico.co.il

Japan

Physix Technology Inc.

Ichikawa-Business-Plaza
4-2-5 Minami-yawata,
Ichikawa-shi
272-0023 Chiba
Japan
T/F: +81 47 – 370 86 00
Email: info@physix-tech.com
www.physix-tech.com

South Korea

SEUM Tronics

Room 502, 534 Seobusaet-gil
Geumcheon-Gu
08505 Seoul
Korea

T: +82 2 868 – 10 02
Email: hslee@seumtronics.com
www.seumtronics.com

USA

SmarAct Inc.

2140 Shattuck Ave., Suite 1103
94704 Berkeley, CA
United States of America

T: +1 415 – 766 90 06
Email: info@smaract.com
www.smaract.com